第二十四屆旺宏金矽獎

作品企劃書

髖關節外骨骼暨 AR 交互運動輔助裝置

Hip Exoskeleton and AR Interactive Motion Assistance Device

參賽組別：Robotics

參賽編號：A24-192

隊　　長：賴宏達

隊　　員：沈恩佑、徐翊紘、劉芸婷

1. *Project proposal*

1.1. *Introduction*

Prolonged physical activities not only lead to muscle fatigue but may also pose safety and health risks during exercises. This reality has prompted us to consider how technology can address this issue. Consequently, we propose the development of an exoskeleton system designed to enhance strength across all age groups[1]. This system aims to reduce the intensity of specific muscle usage and extend the duration of activities that consume physical strength, such as long-distance hiking and stair climbing. By alleviating muscle strain, we hope to enhance the safety of these activities, allowing individuals to confidently engage in various exercises, from everyday stair climbing to more rigorous physical challenges. For static activities, such as squats, the system can also monitor muscle usage to provide motion protection.

To achieve this goal, we plan to utilize AR glasses combined with a virtual coach to offer users an immersive exercise experience (**Figure 1**.) . Furthermore, we intend to integrate multiple EMG patches with the exoskeleton device, using a control chip to monitor joint angles and IMU information, and display the muscle force status in real-time on the AR interface. This design not only facilitates users in understanding their muscle condition but also activates joint restrictions and strength enhancement features of the exoskeleton in case of fatigue or incorrect movements, providing warnings. The exoskeleton is designed to be as portable as an electric scooter, integrating seamlessly into daily life. Therefore, we have focused on developing a hip joint exoskeleton, choosing the hip joint due to its significance in activities like upright walking. The so-called "hip hinge" movement, which starts from the hip joint and relies on the strength of the legs and lower back, supports the body and external weights. Our motor control strategy will employ a reinforcement learning model to adapt to the user's gait, aiming to alleviate the burden on specific muscles.

Figure 1. User exercising with AR glasses and wearing exoskeleton alongside a virtual coach.

1.2. *Specification*

1.2.1. *Exoskeleton system:*

1.2.1.1. *Hip Exoskeleton*

<u>Purpose:</u>

The exoskeleton device is designed to assist users in physical activities such as walking or climbing stairs by providing powered assistance to reduce muscle fatigue, increase activity duration,

and enhance safety. Therefore, it is designed with two degrees of freedom to accommodate the hip joint motor. Design diagrams and actual samples are presented in **Figure 2** and **Figure 3**.

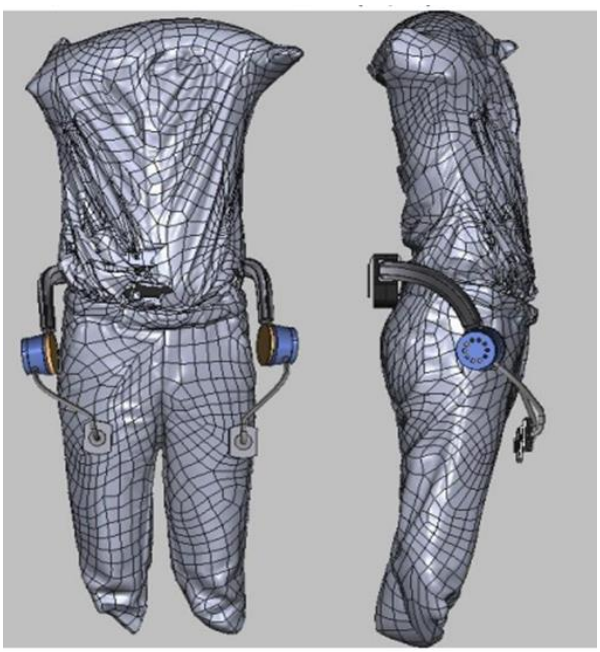


Figure 2. design schematic diagram

(a) (b) (c)

Figure 3. actual sample display (a) front view (b) side view, left side is front (c) rear view

Materials used:

- The main frame utilizes a lightweight yet high-strength aluminum alloy material to ensure device stability while minimizing the burden on the user.
- Skin-contact areas such as waist belts and thigh straps are crafted from breathable, elastic, and hypoallergenic medical-grade materials to ensure comfort during extended wear.
- The housing for the power components is constructed from engineering plastic to shield the internal motors from external influences.

Dimensions and Weight:

- The overall design of the device is compact, with dimensions suitable for most adult body types, and can be fine-tuned through adjustable straps.

- The diameter of the hip power unit is approximately 7 centimeters, with a thickness of about 4 centimeters.
- The total weight of the entire device is kept below 5.5 kilograms, ensuring that users can wear it for extended periods without excessive burden.

Fastening Method:

- The hip fixation utilizes wide, adjustable nylon belts equipped with Velcro or buckle designs, along with a rear-sliding track design, facilitating users to adjust the tightness according to their waist circumference.
- The power unit is connected to the waist belt via rigid yet flexible support arms, accommodating users of varying heights.
- The thigh area is secured using elastic straps with embedded anti-slip material to adapt to different thigh circumferences, ensuring the device remains in place during movement.

1.2.1.2. EMG Cygnus dongle and Electrode patches

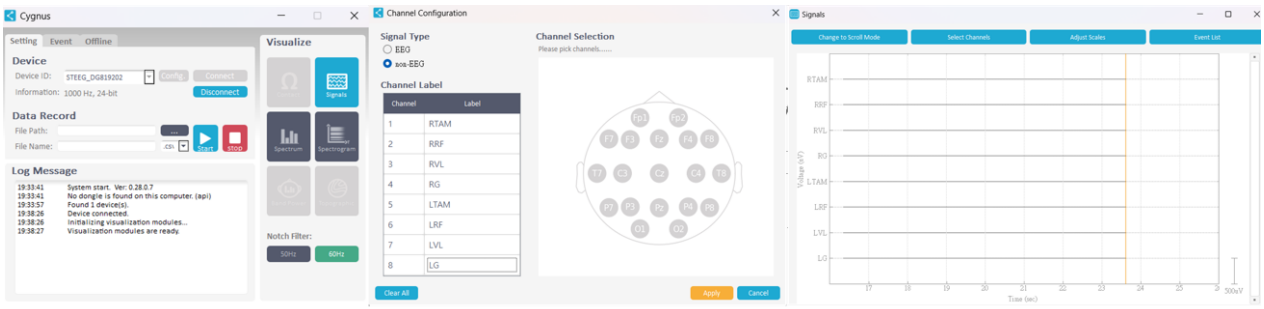
The Cygnus system is a pioneering integration of software and hardware designed for enhancing exoskeleton technology through the precise capture and analysis of electromyographic (EMG) signals. These signals, generated by muscle contractions, are accurately detected by high-sensitivity, noise-resistant sensors. The accompanying software processes these signals in real-time, translating human muscle activity into controlled exoskeleton movements for rehabilitative support or enhanced physical capabilities. This system enables a seamless connection between human intention and robotic precision, offering users improved autonomy and efficiency. The dimensions of the dongle connecting to the computer are 6.40 cm x 2.54 cm x 1.50 cm, highlighting its sleek and functional design for seamless integration with EMG systems. Conversely, the dongle that connects to the EMG wires measures 6 cm x 4.78 cm x 1.83 cm, demonstrating the system's compact and efficient component architecture. Additionally, the electrode patches, essential for capturing electromyographic signals, are square-shaped with dimensions of 3.53 cm x 3.53 cm, ensuring precise and comfortable placement on the skin.

(a) (b)

Figure 4. The actual attachment locations of EMG patches, (a) for right leg and (b) for left leg.

(a) (b)

Figure 5. (a) EMG Cygnus dongle and (b) Electrode patch collector.



(a) (b) (c)

Figure 6. (a) The Cygnus system's user interface, (b) the channel settings, and (c) the visualization of real-time electromyographic (EMG) signals.

(a) (b)

Figure 7. (a) EMG electrode patches and measurement wires

1.2.1.3. NodeMCU-32S

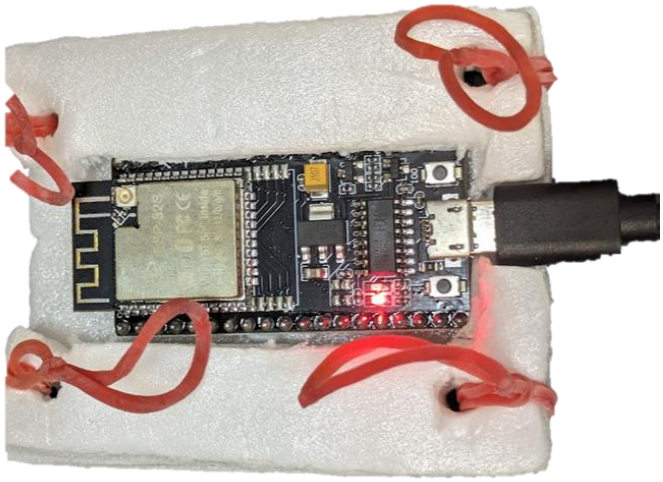


Figure 8. The ESP32 NodeMCU-32S development board, along with a temporary expanded polystyrene (EPS) protection platform, is currently used for integration with the exoskeleton.

Category	Description
Core	ESP-WROOM-32s A highly adaptable universal Wi-Fi+BT+BLE MCU module

Certification	FCC/CE-RED/IC/TELEC/KCC/SRRC/NCC/BQB/RoHS/REACH
SPI Flash	32Mbit (default)
Integrated Crystal	40MHz crystal oscillator
I/O Ports	38 in total
Antenna	On-board antenna
Wi-Fi Standard	802.11b/g/n (up to 150Mbps)
Wi-Fi Frequency Range	2.4GHz ~ 2.5GHz
Processor	Adjustable clock frequency range from 80 MHz to 240 MHz Supports Real-Time Operating System (RTOS)
Analog Converters	Built-in 2-channel 12-bit high-precision ADC, supports up to 18 channels
Interfaces	UART/GPIO/ADC/DAC/SDIO/SD card/PWM/I2C/I2S
Sleep Modes	Supports multiple sleep modes, with sleep current less than 5μA for ESP32 chip
Protocol Stack	Embedded LwIP protocol stack
Operation Modes	Supports STA/AP/STA+AP operation modes
Remote Firmware Upgrade	Supports Firmware Over-the-Air (FOTA) updates
Development	Supports easy use with general AT commands Supports secondary development, integrates Windows, Linux development environment
Power Supply	Voltage: 3.0V ~ 3.6V, typical 3.3V, current greater than 500mA
Operating Temperature	-40°C ~ 85°C
Storage Environment	-40°C ~ 120°C

1.2.1.4. STM32F415 microcontroller

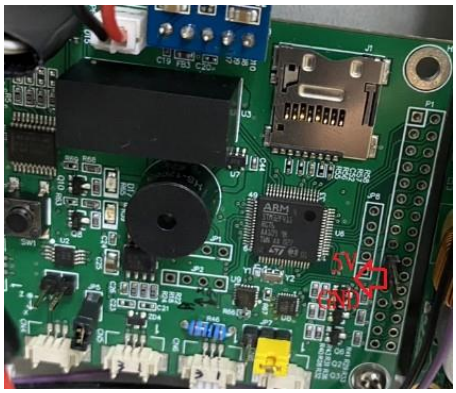


Figure 9. The STM32F415 microcontroller is housed within a pre-packaged plastic enclosure, adjacent to the power supply, and featuring external power ports for providing power externally and accommodating the ESP32.

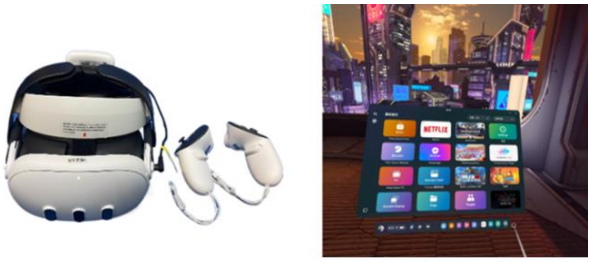
Category	Description
Core Processor	ARM® Cortex®-M4 32-bit RISC core with floating-point unit Operating frequency up to 168 MHz
Memory	1 MB Flash memory 192+4 KB SRAM
I/O Interfaces	Up to 51 General Purpose Input/Output (GPIO) pins Up to 3 12-bit Analog/Digital Converters (ADC) with 24 channels Digital/Analog Converter (DAC) Up to 17 timers, including one high-precision timer
Communication Interfaces	3 I2C bus interfaces 4 USART and 2 UART interfaces 3 SPI serial interfaces 2 CAN interfaces SDIO interface USB OTG full-speed/high-speed
Debugging and Programming Interfaces	SWD (Serial Wire Debug) and JTAG interfaces
Additional Features	RTC (Real-Time Clock) CRC (Cyclic Redundancy Check) calculation unit 96-bit unique device identity code
Power	VDD from 2.0V to 3.6V Multiple power management modes for power optimization
Form Factor	LQFP64 package, 10x10 mm package size

1.2.1.5. R14 Servo Motor:MCX-R14-50-68

Specification	**Description**
CAN Bus Baud Rates	1Mbps (default), 500K, 250K
RS485 Bus Baud Rates	115200bps, 500Kbps, 1Mbps, 1.5Mbps, 2.5Mbps
Message Format	Defines the format of data communication.
Identifier	Single Motor Command Sending: 0x140 + ID(1~32) Multiple Motor Command Sending: 0x280 Reply: 0x240 + ID(1~32)
Rated Torque (N*m)	3.8
Repeated Peak Torque (N*m)	12
Maximum Peak Torque (N*m)	25
Motor Type	BLDC
Input Voltage (V)	24

Rated Torque (N*m)	0.1
Peak Torque (N*m)	0.2
Rated Speed (RPM)	1600
Rated Current (A)	1.5
Communication Interface	RS485 / CAN BUS
Control Mode	Position/Speed/Torque
Encoder	17-bit Magnetic Type
Protection	IP65 / Dustproof, Waterproof
Outline Dimensions (mm)	68
Weight (g)	400

1.2.2. VR goggles



(a) (b)

Figure 10. The exterior (a) and interior (b) presentation of the Quest 3.

Info	Manufacturer	Meta
	Device Type	Standalone VR
Optics	Optics	Pancake lenses
	Ocularity	Binocular
	IPD Range	58-71 mm hardware adjustable (manual)
	Passthrough	Dual 18 PPD color passthrough cameras
Display	Display Type	2 x LCD binocular
	Subpixel Layout	RGB stripe 3 subpixels per pixel
	Resolution	2064x2208 per-eye
	Refresh Rate	120Hz
Image	Visible FoV	110° horizontal 96° vertical
	Peak Pixel Density	25 PPD
Device	Weight	515 g with headstrap
	Material	Plastic, foam facial interface
	Headstrap	Flexible fabric strap
Tracking	Tracking Type	6 DoF Inside-out via 4 integrated cameras

		Also includes depth sensor
	Hand Tracking	Yes
	Body Tracking	Yes
Controllers	Controllers	2 x Meta Quest Touch Plus Controllers 6 DoF
	Finger Tracking	Partial finger and thumb tracking via capacitive sensors
Connectivity	Ports	USB Type-C, charging contacts
	Wired Video	USB Type-C Oculus Link
	Wireless Video	WiFi streaming Virtual Desktop, AirLink
	WiFi	WiFi 6E
System	Operating System	Android
	Chipset	Qualcomm Snapdragon XR2 Gen 2
	CPU	Octa-core Kryo (1 x 3.19 GHz, 4 x 2.8 GHz, 3 x 2.0 GHz)
	GPU	Adreno 740
Battery	Battery Life	2.2 hours
	Charge Time	2.3 hours 18W power adapter

1.3. Block diagram

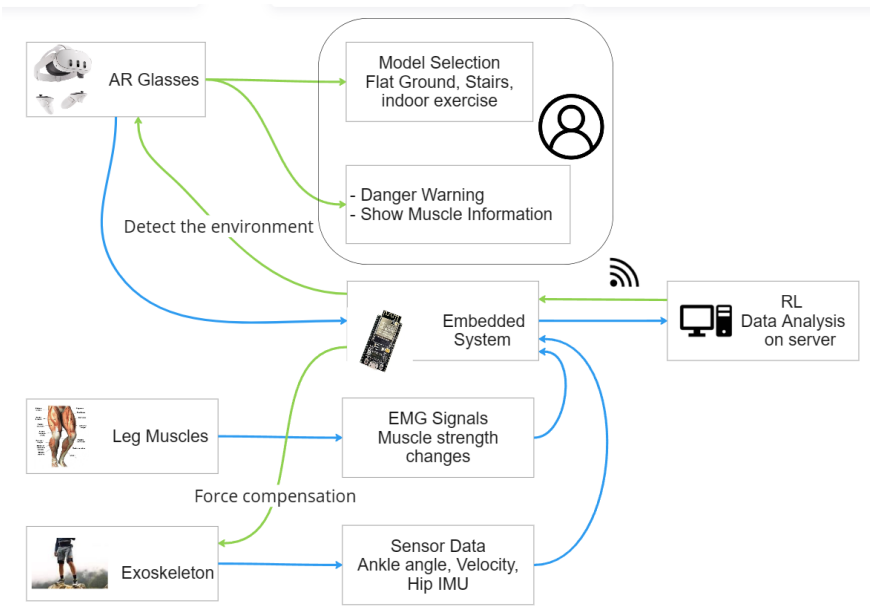


Figure 11. Assistive Exoskeleton Control System

1.4. Work theory

1.4.1. Algorithm

We utilize the Distributed Deep Deterministic Policy Gradient (D4PG) as our core strategy to first predict thigh gait[2] and enhance the power of hip joint rotation through dynamic assistance, aiming to create a product that reduces lower limb labor burden and strengthens sports protection for everyday use. In this process, we will integrate various data, including EMG signals from 8 muscle channels[3, 4], angles, velocities, and accelerations of 2 joints, along with data from 3 IMU sensors. The processing of EMG signals is especially critical for real-time control and marks our biggest difference from previous exoskeleton control systems[5-7]. Due to its personalized characteristics, we opt for a Deep Learning (DL) architecture, expecting it to outperform traditional dynamics models[8-11]. We also hope to address the technical challenges faced by traditional whole-body dynamics models in handling rapid motion transitions. This data-driven approach, which does not rely on complex mathematical modeling (Model-free)[12, 13], could potentially resolve the issues that traditional dynamics models face in adapting to real-time or rapidly changing scenarios. Learning directly from data allows for more flexible adaptation to rapid changes in motion and provides more accurate control[14], which is our core philosophy for this algorithm.

D4PG is a recently proposed reinforcement learning strategy that extends the basic concept of Deep Deterministic Policy Gradient (DDPG)[15] by introducing a distributed framework and a new method for estimating return distributions. Reinforcement learning (RL) has unique advantages over traditional supervised deep learning architectures. Supervised learning depends on a large dataset of labeled data for training, and the process of manual labeling can be limiting when robots learn complex, dynamically changing tasks. In contrast, reinforcement learning learns through interaction with the environment without the need for a pre-labeled dataset. It allows robots to autonomously explore and learn from successes and failures, finding the optimal strategy through a trial-and-error process.[16]

1.4.1.1. <u>**Brief review of traditional RL methods: Policy Gradient and Q-learning**</u>

The concepts of Actor and Environment mentioned above can actually be described through the following mathematics[17]. The Actor determines its behavior in the environment through a policy function π, which can be deterministic or stochastic. A deterministic policy provides a specific action a for each state s, acting like a function $a = \pi(s)$, whereas a stochastic policy provides a probability for each possible action, in the form $P(a \mid s) = \pi(a \mid s)$. In policy gradient methods, the probability of a trajectory (1):

$$P(\tau) = p(s_0) \prod_{t=0}^{T} \pi(a_t|s_t) p(s_{t+1}|s_t, a_t) \tag{1}$$

describing the probability of generating a complete trajectory τ under the premise that the actor follows policy π . A trajectory is a sequence of states and actions, including the environment's response to the actor's actions, i.e., the state transition probability $p(s_{t+1}|s_t, a_t)$. To maximize the expected return, there is a method called Proximal Policy Optimization (PPO) (2):

$$\nabla_\theta J(\theta) = E_{s\sim\rho^\pi, a\sim\pi}[Q^\pi(s,a) \nabla_\theta \log \pi(a|s;\theta)] \tag{2}$$

Where $Q^{\pi}(s,a)$ is the value function determined by the environment, evaluating the expected return of taking action a in state s. The process of updating weights is essentially the opposite of weight updating in traditional DL classification problems (adding a negative sign). The policy performance function $J(\theta)$ is the expected value of rewards obtained under policy π, represented by the formula (3):

$$\mathrm{J}(\theta) = \mathrm{E}_{\tau\sim\pi_\theta}[R(\tau)] \tag{3}$$

$R(\tau)$ represents the reward of the trajectory, and π_θ represents the policy defined by parameters θ. this estimated gradient is used to update the policy parameters θ through the gradient ascent method (4), where α is the learning rate:

$$\theta \leftarrow \theta + \alpha\nabla\theta J(\theta)$$
$$\nabla_\theta J(\theta) = E_{\tau\sim\pi_\theta}[R(\tau)\nabla_\theta log\pi_\theta(\tau)] \tag{4}$$

Since directly calculating the expected value is difficult, an approximation of the expected value is made by sampling (5):

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{n=1}^{N} R(\tau^n)\nabla_\theta log\pi_\theta(\tau^n) \tag{5}$$

$\nabla_\theta log\pi_\theta(\tau^n)$, n reality, this logarithmic gradient can be decomposed into the sum of the logarithmic gradients of each step's action, since the choice of policy at each step is independent. Therefore, we can rewrite this expression as the sum of the logarithmic gradients of action choices at each moment (6), i.e $\sum_{t=1}^{T^n}\nabla_\theta \log\pi_\theta\,(a_t^n|s_t^n)$.

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{n=1}^{N}\left(\sum_{t=1}^{T^n} R(t)\nabla_\theta \log\pi_\theta\,(a_t^n|s_t^n)\right) \tag{6}$$

If each state transition implicitly assumes the Markov property defined in the Markov Decision Process(MDP)(7):

$$G_t = \sum_{k=0}^{\infty}\gamma^k R_{t+k+1}\,, then$$
$$\nabla_\theta J(\theta) = E[\sum_{t=0}^{T}\nabla_\theta \log\pi_\theta\,(a_t|s_t)G_t] \tag{7}$$

PPO optimizes directly in the policy space, aiming to improve the policy itself directly. However, if one first learns a value function and then derives the optimal policy based on this value function, this method, which directly estimates $Q(s,a)$ values for learning, is called Q-learning. The relationship between the value function and the Q-function can be expressed as follows (8): The value of a state s under policy π is equal to the Q-value obtained by taking the action recommended by policy π in state s.

$$V^{\pi}(s) = Q^{\pi}\big(s, \pi(s)\big) \tag{8}$$

We can prove that when (9), the optimization of the value function can lead to the optimal strategy.

$$\pi'(s) = \arg\max_{a} Q^{\pi}(s,a), \text{ then } V^{\pi'}(s) \geq V^{\pi}(s) \text{ for all states}$$

$$V^{\pi}(s) \leq \arg\max_{a} Q^{\pi}(s,a) = Q^{\pi}\big(s, \pi'(s)\big), \text{ then}$$

$$\begin{aligned} V^{\pi}(s) &\leq E[r_{t+1} + V^{\pi}(s_{t+1}) | s_t = s, a_t = \pi'(s_t)] \\ &\leq E\big[r_{t+1} + Q^{\pi}\big(s_{t+1}, \pi'(s_{t+1})\big) \big| s_t = s, a_t = \pi'(s_t)\big] \\ &= E[r_{t+1} + r_{t+2} + \gamma V^{\pi}(s_{t+2}) | \dots] \leq E\big[r_{t+1} + r_{t+2} + \gamma Q^{\pi}\big(s_{t+2}, \pi'(s_{t+2})\big) \big| \dots\big] \dots \\ &\leq V^{\pi'}(s) \end{aligned} \tag{9}$$

By subtracting the current estimate of $Q(s,a)$ from the Temporal Difference Error ($TD\ Error$) (10), we are able to evaluate the discrepancy between the actual observed reward (including immediate rewards and estimated future returns) and the initially estimated reward.

$$TDError = \left(R(s,a) + \gamma \max_{a'} Q(s',a')\right) - Q(s,a) \tag{10}$$

And we get bellman equation (11) that can be used to iteratively update the Q-value, thereby continuously approaching the optimal policy.

$$Q^{new}(s,a) \leftarrow Q(s,a) + \alpha\left[R(s,a) + \gamma \max_{a'} Q(s',a') - Q(s,a)\right] \tag{11}$$

1.4.1.2. <u>**From Q-learning to Deep Deterministic Policy Gradient (DDPG)**</u>

In traditional reinforcement learning approaches, the Q-function is represented in a tabular form, which becomes impractical for problems involving large state spaces. Consequently, Deep Neural Networks (DNNs) have been introduced to facilitate the mapping from high-dimensional perceptual inputs to the action space, effectively bridging complex inputs to discrete action outputs. For more sophisticated mappings that cater to continuous action spaces, the Deep Deterministic Policy Gradient (DDPG) algorithm is relied upon[18].[18]. This advancement allows for the handling of complex and high-dimensional environments that were previously challenging for conventional Q-learning algorithms. The mathematical structure of learnable neurons in artificial neural networks (NN) will be detailed in the "Source Code 2.1.5" section of the document. Let's first introduce the difference between Deep Q-learning and the backpropagation algorithm in a typical neural network regarding the calculation of the loss function. Neural networks can be used to approximate the value function Q of state-action pairs. In TD(0) evaluation, the loss function for this value function is (12):

$$J(\theta) = (TDError)^2, \text{ that is}$$

$$J(\theta) = \left(r_t + \gamma \hat{Q}(s_{t+1}, a_{t+1}; \theta^-) - \hat{Q}(s_t, a_t; \theta)\right)^2 \tag{12}$$

DQN employs certain features, including experience replay and fixed target Q-networks, to address the instability and data sample correlation issues during the learning process. While these techniques enhance the stability and convergence of the algorithm, they also introduce additional complexity and latency, potentially impacting the real-time performance and applicability of the algorithm. By separating the actor to directly learn the policy and the critic to evaluate the value of actions, Actor-Critic can effectively handle problems with continuous action spaces. DDPG is an Actor-Critic method designed for continuous action spaces, making it applicable for tasks such as robotic control. To achieve this, we need to modify the formula stated above.

For the actor net, We instead utilize the gradient of the Q-function to indirectly achieve the same objective. Since the optimization goal of the Q-function is to make $Q(s, a|\theta^Q)$, approach the actual Q^π. By maximizing the Q-function, we indirectly optimize the policy to obtain the maximum G_t (13). We also adopt a deterministic policy NN $\mu(s|\theta^\mu)$, where for a given state s, the policy directly outputs a specific action a, rather than providing a probability distribution over actions as in traditional policy gradient methods. This approach enables the algorithm to select actions directly in continuous action spaces without the need for sampling to determine the action to be executed.

$$\nabla_{\theta^\mu} J \approx E_{s\sim\rho^\beta}\left[\nabla_a Q(s, a|\theta^Q)|_{a=\mu(s|\theta^\mu)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)\right] \tag{13}$$

In addition, in DDPG, due to the use of deterministic policies, the Ornstein-Uhlenbeck (OU) process (14) is commonly employed to generate temporally correlated noise. This noise not only facilitates exploration but also, due to its temporal correlation, can smooth out the exploration actions over time, which is particularly important for control problems, especially in exoskeleton motors where coherence between actions is required.

$$dx_t = \theta(\mu - x_t)dt + \sigma dW_t \tag{14}$$

- x_t represents the current noise value, indicating the noise level at time t.
- θ is the rate parameter, determining how quickly the noise returns to its long-term mean μ. A higher value results in a faster return of noise to its mean.
- μ is the long-term mean around which the Ornstein-Uhlenbeck (OU) process fluctuates. It serves as the center or equilibrium point of the noise, around which the noise values fluctuate
- σ is the scale parameter controlling the amplitude of the noise, i.e., the magnitude of noise fluctuations. A higher value leads to larger noise generation and a broader exploration range.

- dW_t is the standard Wiener process (i.e., Brownian motion), representing the stochastic component of random fluctuations. It is a random process used to simulate random fluctuations in noise.

As for the differences between the Critic and DQN, the action is derived from the actor network (15). Further distinctions related to iteration and sampling will be explained in the "Source Code 2.1.5".

$$L(\theta^c) = E_{(s,a,r,s')\sim D}\left[\left(r + \gamma Q_{\theta^{c-}}\left(s', \mu_{\theta^{\mu-}}(s')\right) - Q_{\theta^c}(s,a)\right)^2\right] \tag{15}$$

1.4.1.3. <u>Optimizing the Distribution of Expected Returns with D4PG: Enhancing Robustness and Effectiveness of DDPG in Continuous Action Spaces</u>

D4PG addresses potential issues arising from deterministic policies by introducing distributed Q functions.[19] Such issues include insufficient exploration due to overly deterministic action selection and oversimplified estimation of environmental rewards, failing to capture the dynamics and uncertainties of the environment adequately. [19] Such issues include insufficient exploration due to overly deterministic action selection and oversimplified estimation of environmental rewards, failing to capture the dynamics and uncertainties of the environment adequately. Below is the mathematical representation and solution approach of employing distributed value functions in D4PG. Distributed Q functions is a distribution concerning the returns. This distribution can be described by a series of probability masses on fixed support points (16), where each support point represents a possible return value, and the corresponding probability indicates the likelihood of that return value (17). The maximum and minimum values of the return will be predefined.

$$z_i = V_{\min} + i \cdot \Delta z \quad \text{for} \quad i = 0,1,\dots,N-1$$

$$\Delta z = \frac{V_{\max} - V_{\min}}{N-1} \tag{16}$$

$$tz_j = \min\left(V_{\max}, \max\left(V_{\min}, r + \gamma z_j\right)\right) \tag{17}$$

The interpolation weights for the target distribution's support points are calculated relative to a predefined set of support points (18), thereby identifying the two nearest support points for accurate mapping(19):

$$b_j = \frac{tz_j - V_{min}}{\Delta z}$$

$$l = \lfloor b_j \rfloor, u = \lceil b_j \rceil \tag{18}$$

$$L(\theta^c) = \text{Distance(ProjDistr,PredDistr)}$$

$$L(\theta^c) = E(s,a,r,s') \sim D[Distance\left(Proj[R(s,a) + \gamma Q\theta c - \left(s', \mu\theta\mu - (s')\right)], Q\theta c(s,a)\right)] \tag{19}$$

The aforementioned delineates the fundamental essence of the D4PG algorithm employed for model optimization. Within the framework of our model-free exoskeleton reinforcement system, we endeavor to directly discern the mapping relation from physiological signals to actions. This endeavor enables us to perpetually refine and ultimately synthesize precise individual gait patterns. Such an approach obviates the necessity for a priori model knowledge, thereby facilitating the provision of robust assistance. Concurrently, it facilitates the realization of personalized exoskeleton functionality. Moreover, this method adeptly circumvents the longstanding challenge encountered by conventional exoskeleton control systems—namely, their incapacity to swiftly adapt to dynamic movement transitions. The forthcoming section, 2.1.5 will undertake an in-depth exploration of the concrete realization of these conceptual tenets, encompassing both the hierarchical architecture of neural networks and the requisite data structures essential for iterative updates[20].The forthcoming section, 2.1.5 will undertake an in-depth exploration of the concrete realization of these conceptual tenets, encompassing both the hierarchical architecture of neural networks and the requisite data structures essential for iterative updates[20].

In the D4PG algorithm, the environment defines the upper limit for the system to learn and optimize[21]. To align with our objectives and obtain reasonable rewards, we request participants to collect electromyographic (EMG) signals during natural walking for 5 seconds. To align with our objectives and obtain reasonable rewards[21], we request participants to collect electromyographic (EMG) signals during natural walking for 5 seconds. Within this timeframe, we record the maximum and minimum root mean square (RMS) values of eight muscles and divide the difference between these values into 10 equal parts, using one part as the unit for muscle force levels. We set the initial minimum muscle force level to 5 and the initial maximum muscle force level to -5. We design a reward function $R = \sum w_i * ch_i$, where is the w_i assigned to each muscle, and ch_i is the corresponding muscle force level. The muscles of the right and left legs, including the tibialis anterior, rectus femoris, biceps femoris, and gastrocnemius, are allocated reward weights of 10%, 20%, 12.5%, and 7.5%, respectively, for the right leg, and the same percentages for the left leg. Through this reward mechanism, we aim to optimize EMG signals during training to enable the exoskeleton to assist walking more effectively. These implementations are detailed in sections 2.1.3 and 2.1.4.[22, 23]

1.4.2. Flow-chart

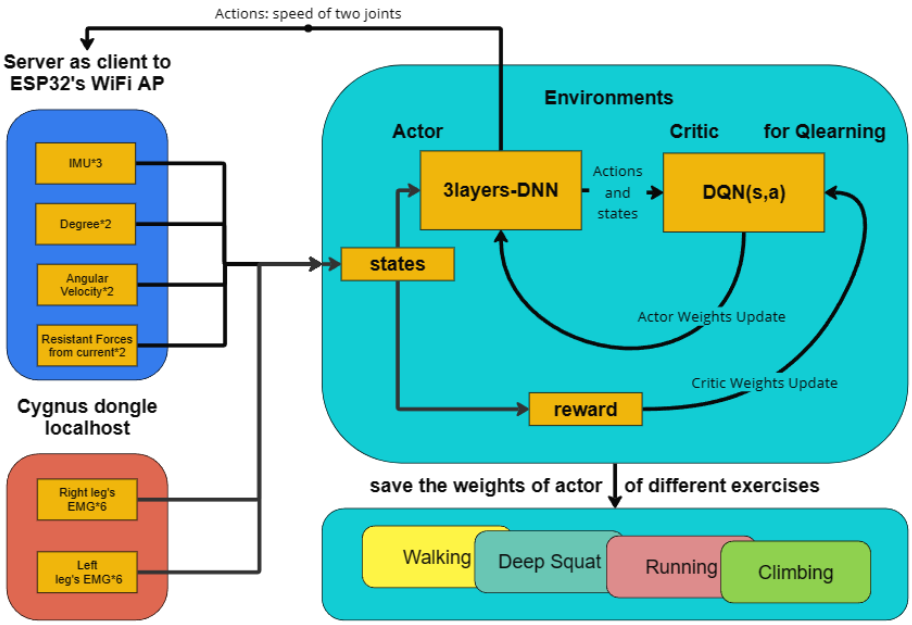


Figure 12. Flow-chart of model training Once this iterative training process is connected to the server, it can continuously optimize the control of the exoskeleton. If there is no connection, different weights can be loaded into the Actor, providing personalized assistance and protection solutions.

1.5. User manual

<u>Wearing exoskeletons and EMG patches</u>

For optimal deployment of the system, involving both the exoskeleton and the EMG patches, a collaborative approach is recommended, starting with the exoskeleton for the hips. This preliminary step facilitates the subsequent application of EMG patches. The exoskeleton should be worn with its sponge pads making direct contact with the user's back, ensuring that the dual motors align precisely with the hips. Upon alignment, fasten the waist belt and complete the fixation process. The device extends metal rods that must sit flush on the thighs, held firmly with elastic bands to mitigate any displacement during ambulation.

The attachment of EMG patches, the subsequent phase, necessitates meticulous placement. The patches must adhere to the identical muscle group, maintaining a set separation—commonly the span of two to three fingers. Patches are to be affixed to the anterior tibialis and gastrocnemius on both legs, plus the rectus femoris and vastus lateralis on the thighs, with a duo of patches per muscle, resulting in a total of sixteen patches. The anterior tibialis, located on the lateral aspect of the shin, and the gastrocnemius, characterized by two fan-like muscles apparent upon calf contraction, are relatively accessible for patch placement. The latter pair, situated on the thigh, require additional steps for identification; the rectus femoris is discernible as the middle muscle of the three visible when the leg is fully extended. For the vastus lateralis, the user should assume a prone position on a chair and attempt to raise the lower leg, wherein an assistant will provide counterforce to prevent elevation. This maneuver reveals the vastus lateralis on the posterior lateral aspect of the thigh.

Instructions for weaking VR glasses and using the interface

Users will don the Meta Quest 3 VR headset and activate the passthrough mode to achieve an augmented reality (AR) effect. Upon launching our Unity application, the user interface presents two exercise modes: walking on flat ground and standard squatting movements. Interaction with the interface is facilitated through the original controllers provided by the manufacturer. Users can select options by pointing the controller at buttons on the interface. For both movement modes, there are two buttons available for selection: one for action cue sounds and another for a virtual coach demonstration. The former plays correct movement cues, allowing users to pause momentarily by pressing the button, with a subsequent press resuming playback. The latter manifests a three-dimensional human model in the real world, demonstrating the correct movements. Users can observe how to perform the movements accurately by pressing the button, with another press pausing the virtual coach's demonstration. To ensure the interface does not obstruct the user's view during exercise, we have also developed a window scaling feature. By pressing the 'B' button on the controller, users can adjust the interface to their preference. This integration aims to enhance user experience by combining physical exercise with interactive, virtual guidance, offering a seamless blend of technology and fitness. Overall, you can refer to the user flowchart in Figure 13.

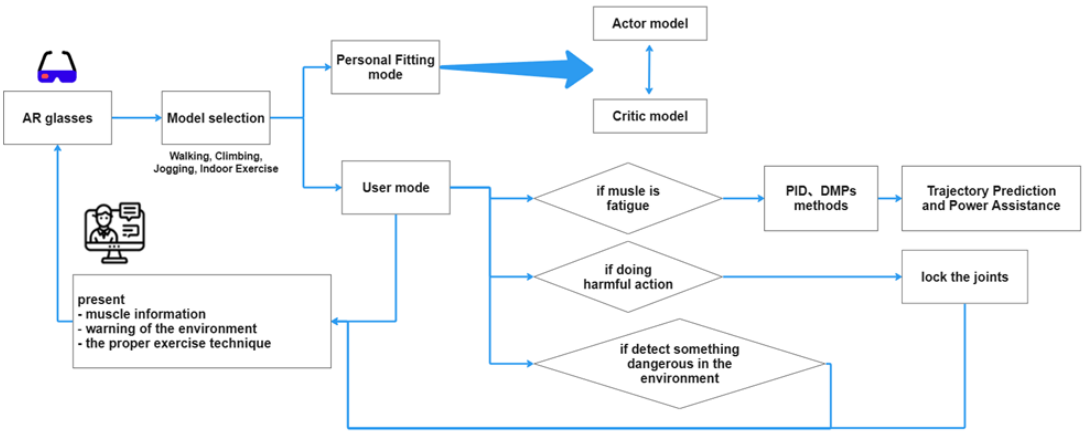


Figure 13. User workflow

2. *Detailed Design Data Files*

2.1. *Source code*

Due to space constraints, the complete source code spans a total of 25 pages and is provided as an attachment. Alternatively, it can be viewed on GitHub at the following link:
eddLai/ExoskeletonPowerAsistance (github.com)

2.1.1. *ESP32*

Using the Arduino framework, we've engineered a sophisticated program for an exoskeleton system. This program not only sets up Wi-Fi communication through the ESP32 S3 NodeMCU but also establishes a data exchange link with the STM32F415 microcontroller via UART(the STM32F415 controller's direct management of a crucial component of the exoskeleton—the R14 Motor Harmonic

Reducer Encoder Antriebsplatine, a large hollow servo integrated connection module). The pivotal aspects of this program involve launching a server that clients can connect to over Wi-Fi and managing two concurrent tasks: one for processing commands received from Wi-Fi clients and another for fetching data from the STM32F415.

The adoption of a dual-core approach is a strategic choice, motivated by extensive testing that revealed the necessity to handle signal transmission and reception at a frequency of 20Hz. This frequency requirement necessitates a high level of responsiveness and efficiency, which is achieved by leveraging the dual-core capabilities of the ESP32 S3 NodeMCU.

Wi-Fi Server Setup

- The Wi-Fi is initialized in AP (Access Point) mode with the SSID "MyESP32AP" and password "12345678".
- The IP address is configured to 192.168.4.1, with the corresponding gateway and subnet mask settings.
- A Wi-Fi server is launched to listen for connections on port 8080.

UART Communication

The program uses the ESP32's second serial port (a hardware serial communication port), connected to the STM32F415 via GPIO 16 and GPIO 17, with the baud rate set to 115200.

Task Creation and Management

Using FreeRTOS, tasks are created on both cores of the ESP32:

- handleCommandsTask on core 0 processes commands from Wi-Fi clients
- getAndSendHipInfoTask on core 1 receives data from the STM32F415 via UART and sends it to the Wi-Fi client.

Processing Wi-Fi Client Commands

When a command is received from a Wi-Fi client, the command function reads the data and sends the received command string to the STM32F415 via UART, enabling remote control.

Receiving and Sending Data

The getHip_INFO function is responsible for receiving data from the STM32F415 and transmitting this data to connected clients over Wi-Fi, providing real-time updates on the exoskeleton's status or sensor readings.

2.1.1.1. platform.ini

2.1.1.2. AP_server_version.cpp

2.1.2. client_order.py:

This script, running on the PC side, is a Python script for low-level communication, designed to control and monitor a wearable exoskeleton device. Moreover, it integrates a custom library to receive data from Cygnus EMG patch sensors. The script establishes a TCP connection with the exoskeleton device's open AP through network sockets, sends control commands, and processes data returned from the exoskeleton device. Additionally, by incorporating a specific library, the script can also read EMG

data in real-time. This data, collected from Cygnus EMG patch sensors worn by the user, is crucial for further analysis and improvement of the exoskeleton's control strategies.

Connection Establishment with the Exoskeleton Device:

Utilizing the connect_FREEX function, the script establishes a TCP connection with the exoskeleton device through a specified IP address and port number, enabling remote control and monitoring of the device's status.

Receiving and Parsing Data:

Through the read_line and analysis functions, the script receives and parses data from the exoskeleton device. This data includes information on the device's angle, speed, and current, which are vital for ensuring the safety and efficiency of the exoskeleton's operations.

Integration of Cygnus EMG Patch Data:

The script integrates the function to receive data from Cygnus EMG patch sensors through a custom library, emg_nonasync. In the get_INFO function, besides processing data from the exoskeleton, it also calls this library function to obtain EMG data in real-time. This data is invaluable for analyzing the wearer's muscle activity patterns and adjusting the exoskeleton's response accordingly.

Safety Checks:

Before sending any control commands, the if_not_safe function performs safety checks on the exoskeleton's current state based on predetermined safety limits, such as angle and current restrictions, to avoid potential dangers or discomfort.

Sending Action Commands:

The send_action_to_exoskeleton function decides on the appropriate action commands based on the current state of the exoskeleton and the data received from the EMG sensors. This includes adjusting the size and direction of the movements and stopping the action if necessary to ensure safety.

2.1.3. emg_nonasync.py

This code implements a process for real-time reception and processing of electromyography (EMG) signals from Cygnus EMG patch sensors via a WebSocket connection. It involves signal acquisition, filtering, rectification, and final analysis, aimed at providing real-time EMG feedback during muscle strength training or monitoring. Cygnus integrates data from 16 channels and connects to the PC program via Bluetooth, transmitting the data to the local network. Below are detailed explanations of the main components of the code:

WebSocket Data Reception

The read_specific_data_from_websocket function establishes a real-time WebSocket connection to a specified URI address, continuously receiving EMG data until a valid EMG array is successfully retrieved.

Data Processing

The received data is first processed by the process_data_from_websocket function. This function parses the JSON-formatted data, extracting the EMG signals and applying a series of signal processing steps to each batch of signals.

The processing includes bandpass filtering (to remove noise), notch filtering (to eliminate power line interference), full-wave rectification (to ensure the signal has only positive values), and low-pass filtering (to extract the signal envelope).

Signal Processing Functions

bandpass_filter, notch_filter, full_wave_rectification, and lowpass_filter are specific signal processing functions used to implement the aforementioned filtering and rectification operations.The process_emg_signal function integrates all the signal processing steps mentioned above, applying comprehensive preprocessing to each batch of EMG data.

Muscle Strength Level Calculation

The calculate_emg_level function calculates muscle strength levels based on the processed signals. It maps the root mean square (RMS) values of the signal to different muscle strength levels, thus providing instant muscle strength feedback. During training or monitoring, this level can be used to assess the user's muscle activity, aiding in adjusting training intensity or evaluating muscle fatigue.

Incorporating data from Cygnus, which combines 16 channels and connects via Bluetooth to transmit data to the local network, enhances the utility and application range of this script, making it a comprehensive solution for real-time EMG signal analysis in applications such as muscle strength training, rehabilitation therapy, and biofeedback.

2.1.4. Env.py

To facilitate modular development in Reinforcement Learning (RL) using the `ptan` library and to save time developing frameworks such as Experience (Exp)[24], this script wraps the functionality into an environment (ENV) compatible with the OpenAI Gym interface. It eliminates the need for asynchronous communication libraries like asyncio, proving that this level of network communication does not require such complexity. The script connects to an Exoskeleton device and Cygnus EMG patch sensors via Bluetooth, integrating the data into the local network for real-time processing. Here's how the code functions within this context:

Environment Setup

- Inherits from gym.Env, making it compatible with various RL algorithms and tools that utilize the Gym interface.
- Initializes communication with the Exoskeleton device through a specified IP address and WebSocket URI for real-time EMG data streaming.

Observation and Action Spaces

- Defines the observation space as a combination of device states (like angles, velocities, and currents from different joints, and IMU data) and processed EMG signals, allowing the RL model to make decisions based on a comprehensive view of the system's current state.

- The action space is defined to control the Exoskeleton device, with actions potentially representing desired movement velocities or positions for the exoskeleton limbs.

Step Function

- Handles the core logic for advancing the environment by one timestep based on an action provided by an RL agent.
- Sends control commands to the Exoskeleton device, processes incoming data (including EMG signals), calculates rewards based on the agent's actions, and checks whether the current episode has ended.

Reset Function

Resets the environment to a default state at the beginning of each episode, ensuring that learning starts from a consistent state. This includes resetting muscle power levels and re-establishing network connections if necessary.

Reward Calculation

Implements a method for calculating rewards based on the EMG signal processing, which could involve evaluating muscle activation levels, ensuring that movements are within safe and effective thresholds.

Rendering and Logging

Provides mechanisms for visualizing the state of the environment and logging detailed information about each timestep, which can be crucial for debugging and understanding the behavior of the RL agent.

Closing the Environment

Ensures that all connections are properly closed and that the device is returned to a safe state when the environment is closed, preventing potential harm to the user or damage to the equipment.

2.1.5. models.py

This code implements the core model architecture of the Deep Deterministic Policy Gradient (D4PG) reinforcement learning algorithm using PyTorch, with the goal of simulating the action of thigh gait through reinforcement learning techniques. The code consists of several parts, including the behavior model (Actor), evaluation model (Critic), agent (Agent), and batch data processing function. Here is a detailed explanation of each component:

DDPGActor

The DDPGActor model, acting as the behavior policy, generates actions a_t for observed states s_t. It is formalized as a neural network to fitting thigh gait pattern[5]:

- Input layer: Accepts the environmental state vectors$s_t \in R^{17}$, incorporating 8 muscle signals, 3 IMU readings, and 2 each for joint angles, velocities, and accelerations.
- Hidden layer: A linear transformation followed by a Tanh activation, $\mathrm{Tanh}(W_h s_t + b_h)$, where $W_h \in R^{17\times 20}$, and $b_h \in R^{20}$
- Output layer: Produces the action vector $a_t \in R^2$, corresponding to the velocities of left and right motors, through $W_o h + b_o$.

D4PGCritic

The D4PGCritic serves as the evaluation mechanism, assessing potential rewards for state-action pairs:

- Observation network: Transforms the state $s \in R^{obs_size}$ to a latent representation through $ReLU(W_{obs}s + b_{obs}\}$ with $W_{obs} \in R^{obs_size \times 400}$
- Merging layer: Concatenates the encoded state with the action vector, $[obs; a]$, for combined processing.
- Output network: Evaluates the action's value, outputting a distribution over n_atoms discrete values representing the action value distribution, $W_{out}[obs; a] + b_{out}$, where $\boldsymbol{W_{out}} \in R^{obs_size \times 400 \times n_atoms}$.

AgentD4PG

AgentD4PG determines actions based on the current state, incorporating exploration noise:

- It employs the DDPGActor to compute a_t from s_t, adding Gaussian noise scaled by ϵ to each action for exploration, $a'_t = a_t + \epsilon \cdot \mathcal{N}(0, I)$, and clips a'_t to $[-1, 1]$.

unpack_batch

The unpack_batch function is used for processing a batch of experiences, unpacking them into a format suitable for training. It extracts states, actions, rewards, end signals, and last states from the batch and converts them into PyTorch tensors.

2.1.6. d4pg_train_sync.py

Directly computing the expectation is often infeasible, as it requires integrating over all possible combinations of states and actions, which is computationally prohibitive in most cases. Therefore, these algorithms typically resort to sampling methods to estimate these expectations. In practice, data structures relying on experience replay buffers (20) are employed to store experiences of interactions between the agent and the environment. By sampling from this buffer, the algorithm can estimate the expected updates of value functions or policy functions. This sampling-based approach allows the algorithm to learn from an incomplete dataset, enabling it to operate efficiently even in complex environments.

$$\text{ReplayBuffer} = \{(s_t, a_t, r_t, s_{t+1}, d_t)_i\}_{i=1}^{N} \tag{20}$$

Each iteration process consists of the following key steps:

2.1.6.1. Exploration and Experience Collection:

Given a state s_t, according to the current policy π, an action a_t is selected. This action is then executed, resulting in a reward r_t, and observing a new state s_{t+1}. This step involves the execution of the policy and interaction with the environment, allowing the agent to gather valuable experiences.

- The actions are selected and executed in the environment within the `while True` loop
- The instance generator `exp_source` of the class `ptan.experience.ExperienceSourceFirstLast` calls `env.step(action)`

- `env.step(action)` within the `test_net` function also performs this action in the environment.

2.1.6.2. Storage and Sampling:

The collected experiences s_t, a_t, r_t, s_{t+1} are stored in a replay buffer, and a batch (typically a batch of experiences) is randomly sampled from it for subsequent learning. This step breaks the temporal correlation between experiences, contributing to the stability and efficiency of the learning process.

- Using the `ptan.experience.ExperienceReplayBuffer``buffer.populate(1)` command, which populates the buffer with experiences.
- Sampling from the buffer to create a batch for learning is done with the `buffer.sample(BATCH_SIZE)` command.

2.1.6.3. Target and Regression:

For each sampled experience, a target value y is computed, combining the immediate reward $r_t$with an estimate of discounted future rewards (utilizing a target Q-function $\hat{Q}$ to stabilize the learning process). The Q-function parameters are then updated by regression methods to bring $Q(s_i, a_i)$ closer to the target value y. This step is pivotal to the algorithm, directly influencing the learning of the value function.

- The target Q-values are computed in the `distr_projection` function.
- With the aid of Pytorch, the regression to update the Q-function parameters is implemented in the optimization steps with `crt_opt.step()` for the critic network.

2.1.6.4. Policy Update:

The parameters of the policy π are updated by maximizing $Q(s_i, \pi(s_i))$, ensuring the selection of actions with the maximum Q-value at each state. This step embodies the core idea of policy gradient, optimizing the policy directly through gradient ascent.

- Actor's loss is calculated and then backpropagated through the network with `actor_loss_v.backward()` followed by `act_opt.step()` by using the class from Pytorch.

2.1.6.5. Periodic Update of Target Networks:

To enhance learning stability, every C steps, the target Q-function $\hat{Q}$ and target policy $\hat{\pi}$ are reset to the current Q-function and policy. This technique, known as "fixed target networks," serves to mitigate the instability encountered in traditional Q-learning, where updates to the Q values are directly based on their own predictions. It circumvents the potential for feedback loops and instability that can occur when two neural networks—target and primary—are updated too frequently in tandem during the learning process.

- The synchronization of the target networks is facilitated by the `alpha_sync` function, a feature provided by the `ptan` library, as observed in the usage of `tgt_act_net.alpha_sync(alpha=1-1e-3)` and `tgt_crt_net.alpha_sync(alpha=1 - 1e-3)`.

2.2. Self-documented coding style

To ensure the readability and maintainability of the code, we prefer adopting a self-documenting coding style, combined with the following specific practices to address the specific challenges encountered in the project:

2.2.1. Clear Naming and Function Layering with Logical Decomposition:

We enhance code clarity and readability by using descriptive naming and breaking down complex logic into multiple simple boolean functions. Additionally, we distribute functions of different functionalities into multiple custom libraries, roughly categorized into communication protocols, abstraction and integration, and RL model integration.

2.2.2. Team Collaboration Using GitHub and Live Share:

We utilize GitHub for version control and collaboration, leveraging Live Share for real-time collaborative programming. We also employ shared hosts and native pip venv virtual environments to unify development environments, ensuring efficient collaboration among team members.

2.2.3. Stability and Error Handling:

When communicating with WiFi devices, we face challenges regarding network stability. To address this, we extensively use try-except structures with while loops in the code to handle possible exceptions and retry logic, ensuring stable communication with external devices even in unstable network conditions. For instance, when reading data from WiFi devices, we capture any exceptions and debug by printing error messages, while using loops to continuously attempt until successful data reception.

2.2.4. Complexity of EMG Signal Processing:

Considering the complexity of EMG signal processing, we particularly emphasize clear documentation and meticulous error handling in this part of the code. By adding detailed docstrings in signal processing functions, we explain the purpose and implementation of each step, while also employing appropriate exception handling mechanisms to address potential signal processing errors, ensuring the robustness and reliability of the signal processing workflow.

2.3. Application software source code

2.3.1. Scene setup, voice, motion safety expert system, and extensive use of existing VR APIs in Unity

We have developed an application for the Meta Quest 3 using Unity, incorporating a range of libraries such as XR Plugin Management, XR Core Utilities, AR Foundation, Oculus Integration, and the Meta XR All-in-One SDK to design a virtual coach application. Initially, spatial detection available on the Meta Quest 3 is employed to construct representations of the surrounding planes and environment. Subsequently, the AR Plane Manager from the AR Foundation is utilized to configure these planes adequately. Upon the completion of plane detection, athlete models and animations are imported and configured with rigid bodies and collisions to achieve the objective of standing on the detected planes. This setup allows for interactions between the model and the environment, as well as between the model and the user. Moreover, enhancements in animation logic and control have been implemented to ensure that the model's movements closely mimic real-life athletic motions. The next phase involves designing a control panel for importing voice introduction and model selections. Finally,

through the use of joystick raycasting and button inputs, users can select the desired actions to follow along with the virtual coach for an immersive exercising session. The state of the exoskeleton is also imported via Wi-Fi streaming, allowing us to implement an expert system for motion safety assessment and provide voice reminders.

2.3.2. SceneController.cpp

The `SceneController` script is for an augmented reality (AR) application made with Unity, utilizing AR Foundation for plane detection and the XR Interaction Toolkit for interactive features. It manages the AR session by handling the visibility of detected planes, responding to user inputs for activating objects or interactions, and facilitating the deletion of objects within the AR environment. It listens for specific actions from the user, like toggling the AR plane visibility, activating objects with a left or right controller, and deleting spawned characters. The script also monitors changes in AR planes and anchors, adding and removing them as needed. A notable feature is spawning a predefined prefab when the left controller detects a hit on a floor plane, indicating the user's interaction point. This controller interaction is crucial for placing objects in AR. The script is designed to be attached to a central game object in the scene that serves as the hub for AR management.

2.3.3. PlayerAnimationController.cpp

The `PlayerAnimationController` script in Unity is designed to manage a player character's walking animation using an input system. It references an input action for toggling walking and an animator to control the animation states. When the game starts, it resets the walking animation trigger. The script subscribes to the toggle walk input action when enabled, and upon receiving input that the action has been performed, it triggers the walking animation. Conversely, it unsubscribes from the input action when the script is disabled. This setup allows for a responsive and interactive character control, where a specific player input directly influences the character's animation state, giving the game a dynamic and immersive feel.

2.3.4. CharacterMovement_walk.cpp

The `CharacterMovement_walk ` script in Unity propels a character forward when the "Walk" animation is active. It relies on a Rigidbody component for physics-based movement and uses an Animator to check the animation state. Upon starting, it grabs the attached Rigidbody and then continuously checks in the physics update loop if the "Walk" state is active. If so, it moves the character at a specified speed in the direction they're facing, providing a consistent and realistic motion. This approach ensures the character's movement is synchronized with the walking animation, creating a cohesive and responsive gameplay experience.

3. Verification Results and Materials

3.1. Demonstration

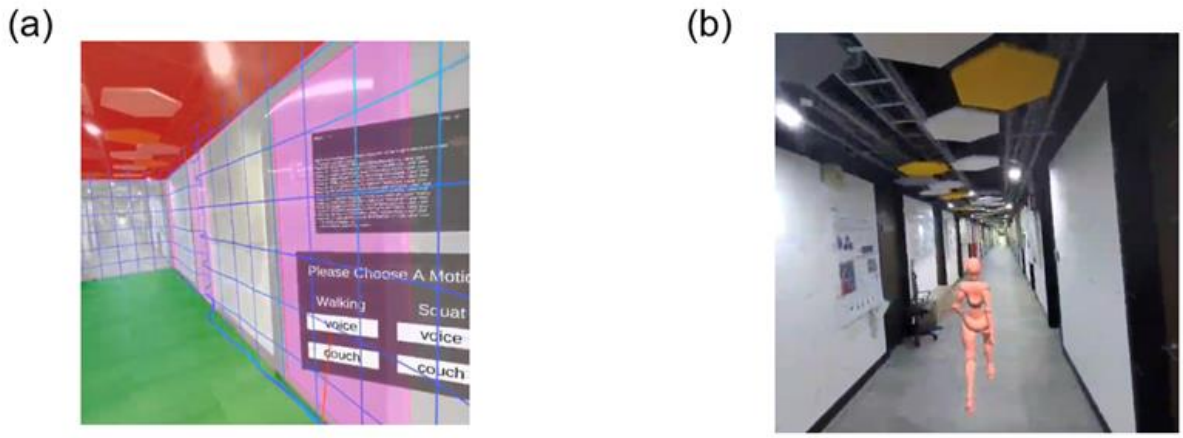


Figure 14. demonstration of the Unity application with Meta Quest 3. (a) plane detection and control panel (b) walking alongside the virtual coach

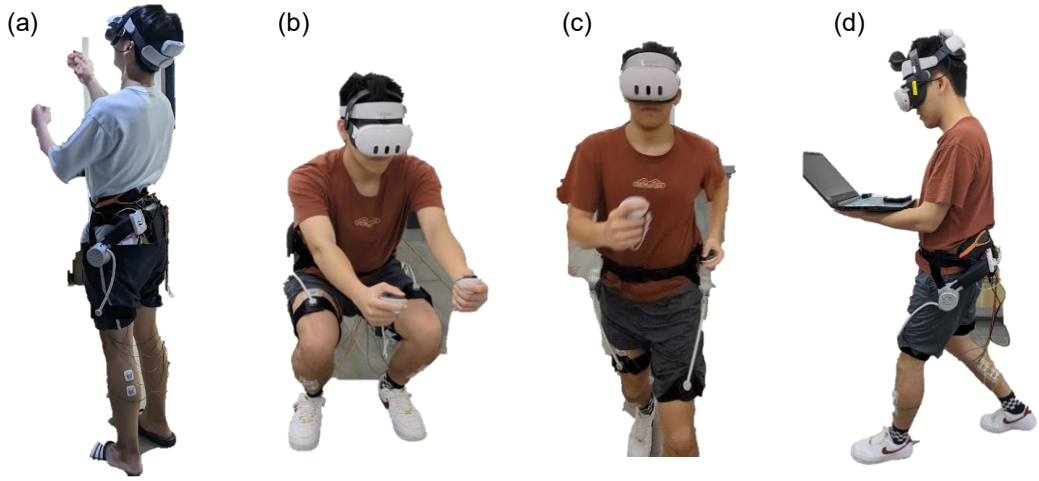


Figure 15. demonstration with exoskeleton and Meta Quest 3.(a) selecting models (b) Squat with Motion Monitoring (c) running with virtual coach (d) walking with desktop used as a server to train the model

3.2. System test result

Due to the large volume of data, we extract a segment to illustrate the data received by our system, The unit on the x-axis is "steps." Each step corresponds to receiving one state and issuing one command to the exoskeleton.

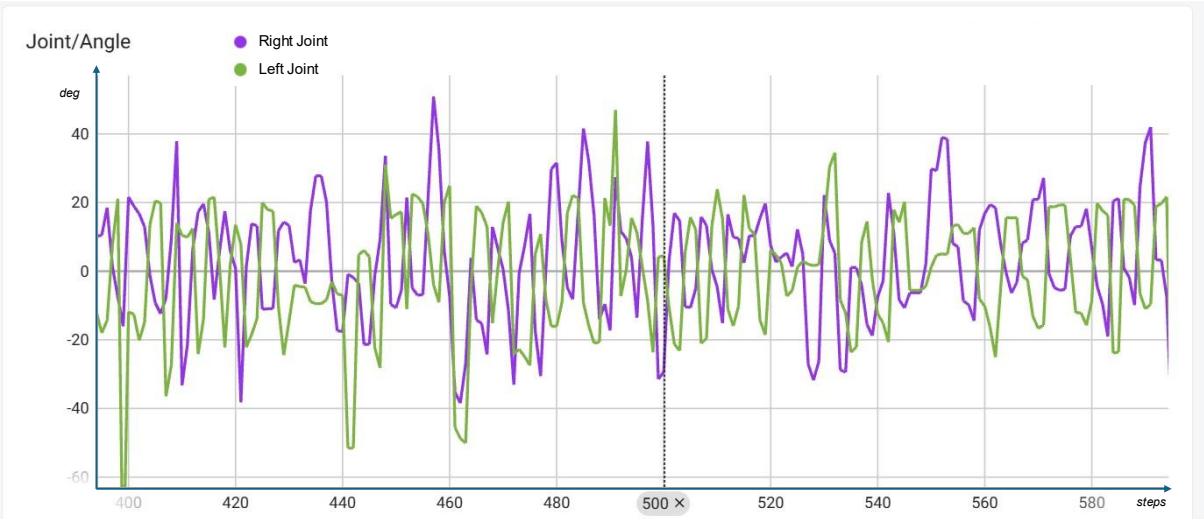


Figure 16. Joints' angle changes when walking Display the variation in walking angles returned by the exoskeleton, excerpting the results of model training initiated on March 30, 2024, at 23:30:36, for the command steps 400 to 600 interval.

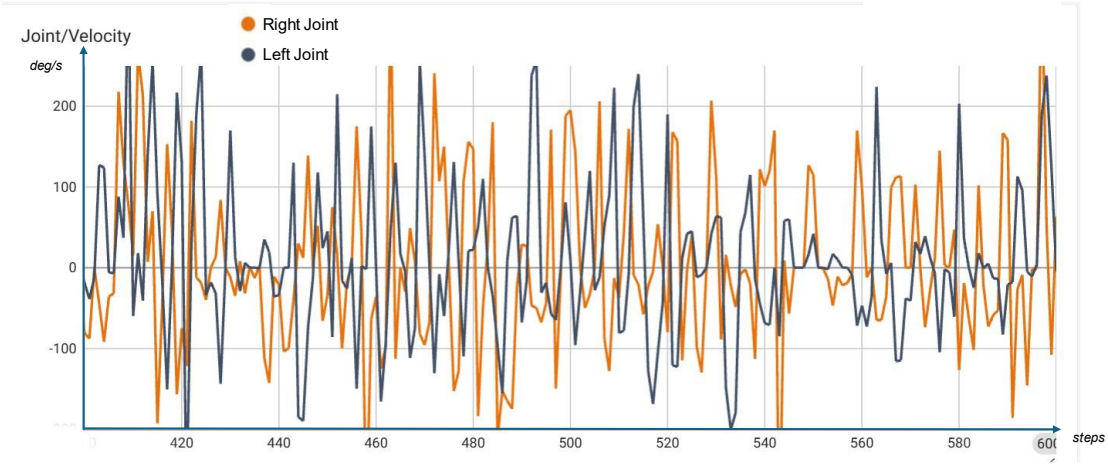


Figure 17. Joints' velocity changes when walking The data depicted in this figure is extracted from the same interval as the preceding one. The units are in deg/s.

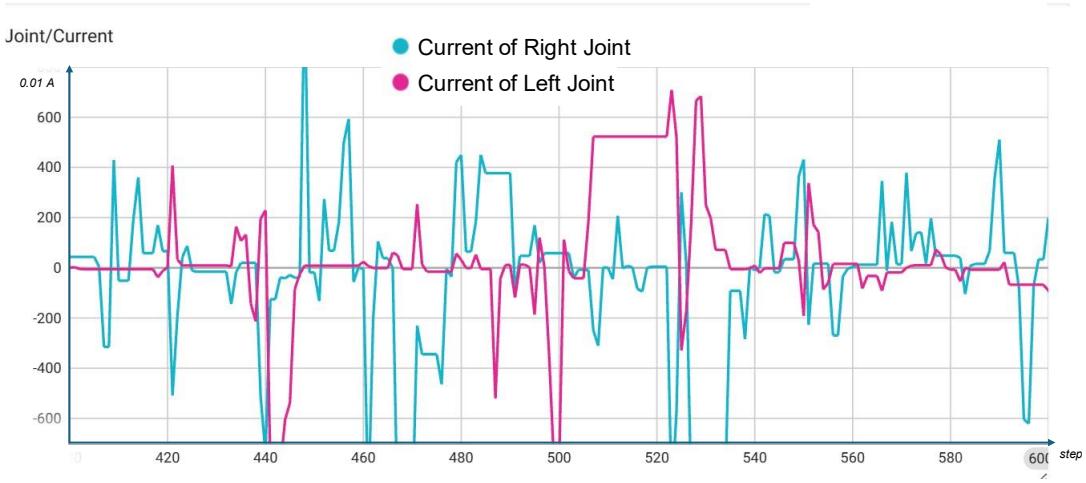


Figure 18. Joints' current changes when walking When a command is issued and resistance current occurs, the resulting values will approximately double in proportion, thereby allowing inference regarding the resistance experienced by the exoskeleton. The unit of y-axis: 0.01A.

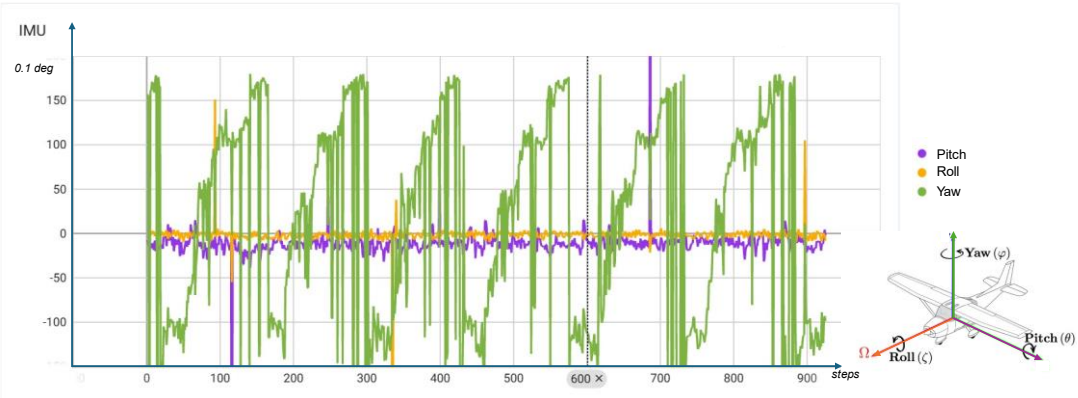


Figure 19. Joints' IMU changes when walking This is the training process starting at the same time, demonstrating changes in gait IMU, which can therefore be utilized for motion protection judgment. The units are in 0.1 degrees.

3.3. Performance benchmark

Due to our reliance on actual training sessions to obtain EMG signals that are challenging to capture in a virtual environment[25], we cannot conduct continuous model optimization for hours on end as traditional RL training permits within software. Our data reflects results aggregated across multiple distinct sessions. We also compare the rewards after EMG magnitude weighting to determine if there is an improvement (less force used corresponds to a lower cumulative weighted score).

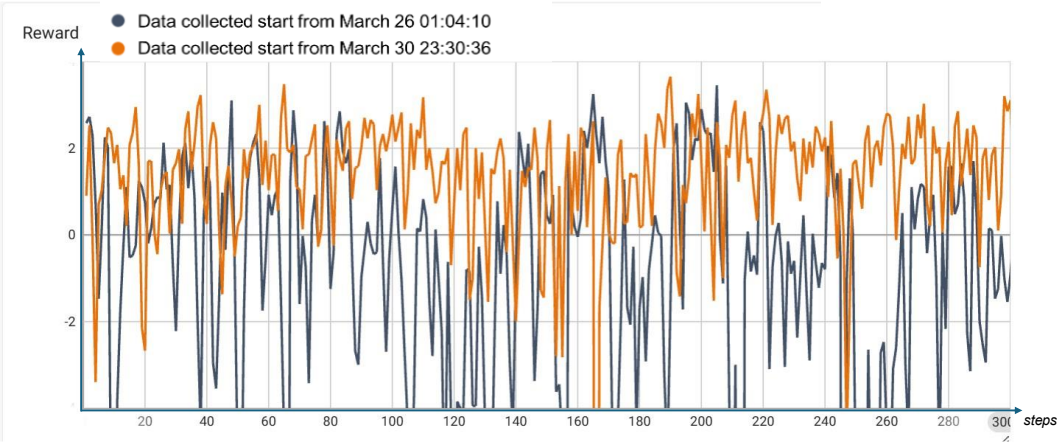


Figure 20. Differences of Rewards when walking after model optimization

Furthermore, we inspect whether there is a general downward trend in the filtered EMG signals, anticipating an outcome of reduced force usage, signified by an overall decrease in EMG activity. Since it is implausible for EMG levels to drop to zero during physical activities.

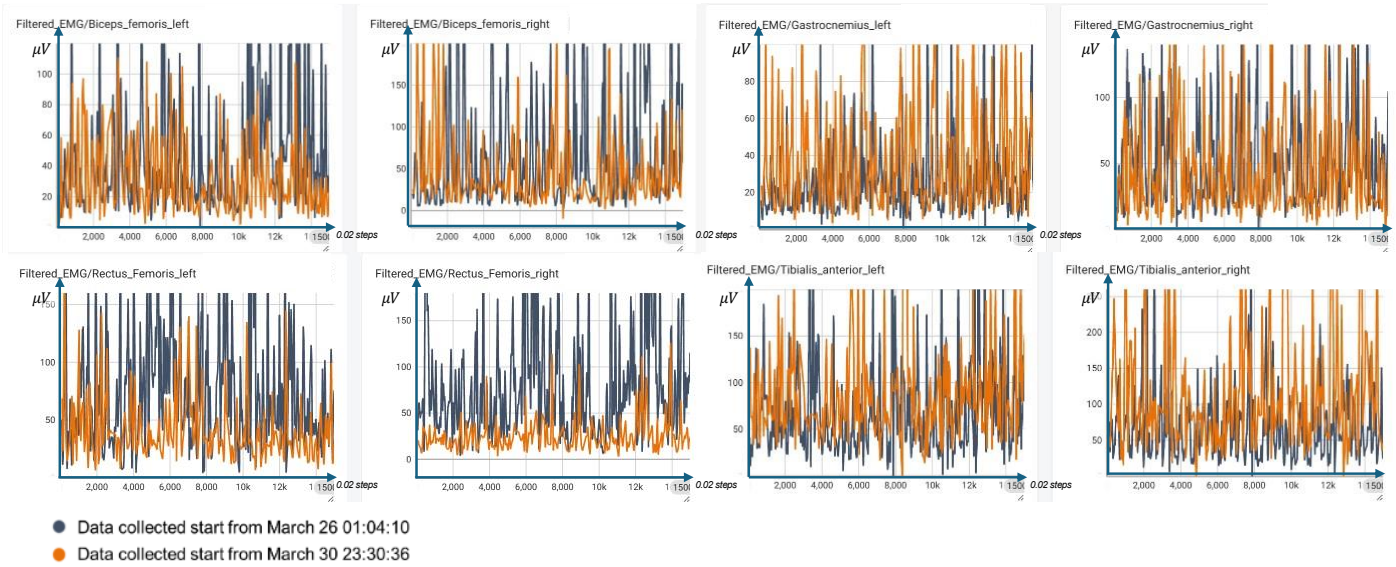


Figure 21. Differences of filtered-EMG signals of different channels when walking after model optimization

Because the exoskeleton's joint data does not accurately represent the actual human joint condition, we resort to these two methods for performance evaluation. After approximately four hours of training distributed over four days, the model has already reduced the EMG signal utilization.

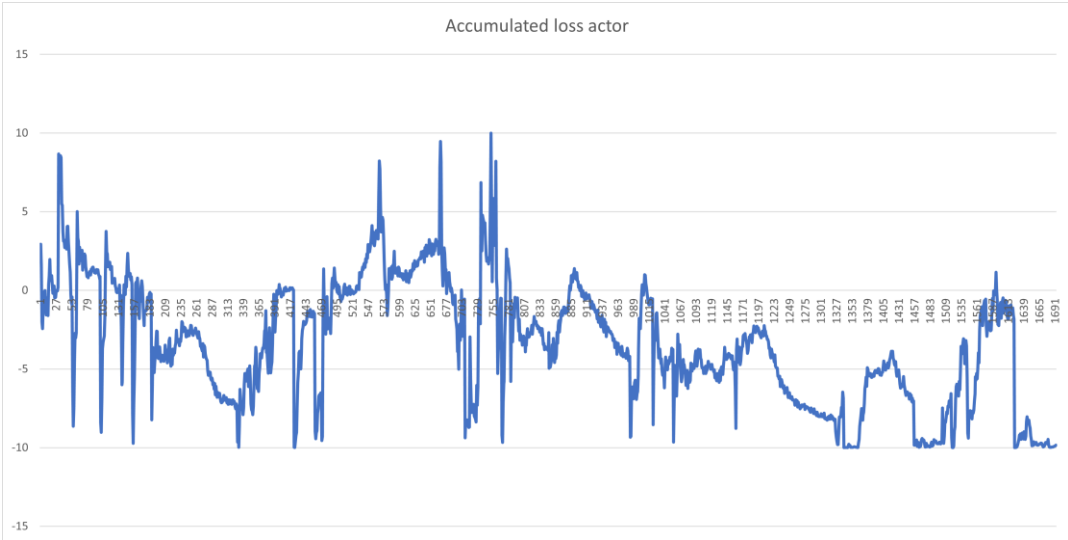


Figure 22. Loss of the actor NN

3.4. The artwork of the project

In the project, we addressed challenges related to managing EMG wires and ensuring that the exoskeleton's power source did not hinder mobility. We used a fanny pack for organized wire storage and data transmission, enabling freedom of movement. To prevent the device from running out of power, we added a power bank to the fanny pack in addition to the internal power supply of the exoskeleton. The ESP32S was securely housed in a custom-sized polystyrene foam and mounted onto the sliding track module of the exoskeleton, facilitating WiFi transmission while providing stability and protection. Although this project is still in its prototype phase and may not excel in aesthetics, it represents a significant advancement in functionality. We aim to enhance convenience and improve packaging integration in future iterations, demonstrating our commitment to innovative thinking and practical problem-solving.

Figure 23. Method for wearing hardware objects on the body

4. Summary Report

4.1. Summaries of the project

We have developed a preliminary prototype of an interactive motion assistance device that combines exoskeleton technology with augmented reality (AR). In contrast to medical applications of exoskeletons[26-28], our goal is to enhance the strength and endurance of general users during physical activities while reducing the risk of muscle fatigue. We hope that such a system can seamlessly integrate into daily life, similar to a skateboard. After all, the agility of our legs has evolved over many years, far surpassing what wheels can handle in terms of terrain. From the performance of the model, it is evident that a simple three-layer model, trained for a short period of time, has already shown

effectiveness. Although only the walking model has been trained due to time constraints, and further generalization of the model is required to fit different users' individual gaits and muscle information more efficiently, it still represents the potential of the algorithmic framework for fitting thigh gaits. As VR glasses emerge as the next-generation mobile device, they provide a highly immersive visual experience, unmatched by smartphones in terms of interactivity. The introduction of the concept of a virtual coach as a digital twin, serving as a sports partner and providing intuitive sports protection reminders, far exceeds the effectiveness of text warnings provided by phones.

4.2. Proposal for future plan as a result of the project

To improve user experience, we propose integrating wireless EMG patch technology into our system, allowing real-time muscle activity data collection without disrupting natural movements. Furthermore, our plan involves the incorporation of the Xilinx KR260 SoM, a high-performance FPGA, as the computational core of our system. This strategic decision is anticipated to reduce the workload on the server to tasks such as storage and recording, while also mitigating the constraints posed by WiFi instability. As a result, we expect to achieve more efficient model inference and overall system performance. In addition, we aim to explore the potential of utilizing a more sophisticated actor model[29, 30], which may offer greater capabilities for capturing EMG features and thereby improving gait prediction accuracy. This avenue of research holds promise for advancing the effectiveness of our system in assisting users during physical activity. Furthermore, we plan to integrate LLM's virtual coach dialogue system into our platform, leveraging solutions downloaded to the server, such as chatwithRTX. This integration will enhance the interactive capabilities of our virtual coach. We recognize the importance of outdoor hazard detection. By making our sports protection system responsive not only to user input but also to environmental cues, we aim to enhance the safety and effectiveness of our solution.

5. *References*

1. Chang, Y., et al., *Verification of industrial worker walking efficiency with wearable hip exoskeleton.* Applied Sciences, 2023. **13**(23): p. 12609.
2. Yuan, Y., et al., *DMP-based motion generation for a walking exoskeleton robot using reinforcement learning.* IEEE Transactions on Industrial Electronics, 2019. **67**(5): p. 3830-3839.
3. Oghogho, M., et al. *Deep reinforcement learning for emg-based control of assistance level in upper-limb exoskeletons*. in *2022 International Symposium on Medical Robotics (ISMR)*. 2022. IEEE.
4. Zheng, J., et al., *Designing deep reinforcement learning systems for musculoskeletal modeling and locomotion analysis using wearable sensor feedback.* IEEE Sensors Journal, 2020. **20**(16): p. 9274-9282.
5. Kang, I., P. Kunapuli, and A.J. Young, *Real-time neural network-based gait phase estimation using a robotic hip exoskeleton.* IEEE Transactions on Medical Robotics and Bionics, 2019. **2**(1): p. 28-37.
6. Yingying, W. and S. Yutian, *Reinforcement learning based adaptive walking assistance control of a lower limb exoskeleton.* 2020.
7. Rose, L., M.C. Bazzocchi, and G. Nejat, *A model-free deep reinforcement learning approach for control of exoskeleton gait patterns.* Robotica, 2022. **40**(7): p. 2189-2214.
8. Mahdi, S.M., et al. *Adaptive synergetic motion control for wearable knee-assistive system: a rehabilitation of disabled patients*. in *Actuators*. 2022. MDPI.
9. Huang, R., et al. *Hierarchical interactive learning for a human-powered augmentation lower exoskeleton*. in *2016 IEEE international conference on robotics and automation (ICRA)*. 2016. IEEE.
10. Lin, J., et al. *Optimal task-invariant energetic control for a knee-ankle exoskeleton*. in *2021 American Control Conference (ACC)*. 2021. IEEE.
11. Chen, S., et al., *Precision interaction force control of an underactuated hydraulic stance leg exoskeleton considering the constraint from the wearer.* Machines, 2021. **9**(5): p. 96.
12. Tu, X., et al. *A data-driven reinforcement learning solution framework for optimal and adaptive personalization of a hip exoskeleton*. in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. 2021. IEEE.
13. Zhang, X., et al., *Model-free based neural network control with time-delay estimation for lower extremity exoskeleton.* Neurocomputing, 2018. **272**: p. 178-188.
14. Rose, L., M.C. Bazzocchi, and G. Nejat. *End-to-end deep reinforcement learning for exoskeleton control*. in *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. 2020. IEEE.
15. Guo, Y., et al., *DDPG-based controlling algorithm for upper limb prosthetic shoulder joint.* International Journal of Control, 2023: p. 1-11.
16. Pong, V.H., et al., *Skew-fit: State-covering self-supervised reinforcement learning.* arXiv preprint arXiv:1903.03698, 2019.
17. Hu, M., *The Art of Reinforcement Learning: Fundamentals, Mathematics, and Implementations with Python*. 2023, Springer. p. 21-46.

18. Silver, D., et al. *Deterministic policy gradient algorithms*. in *International conference on machine learning*. 2014. Pmlr.
19. Barth-Maron, G., et al., *Distributed distributional deterministic policy gradients.* arXiv preprint arXiv:1804.08617, 2018.
20. Hessel, M., et al. *Rainbow: Combining improvements in deep reinforcement learning*. in *Proceedings of the AAAI conference on artificial intelligence*. 2018.
21. Pathak, D., et al. *Curiosity-driven exploration by self-supervised prediction*. in *International conference on machine learning*. 2017. PMLR.
22. Mokri, C., M. Bamdad, and V. Abolghasemi, *Muscle force estimation from lower limb EMG signals using novel optimised machine learning techniques.* Medical & biological engineering & computing, 2022. **60**(3): p. 683-699.
23. Carvalho, C.R., et al., *Review of electromyography onset detection methods for real-time control of robotic exoskeletons.* Journal of NeuroEngineering and Rehabilitation, 2023. **20**(1): p. 141.
24. Lapan, M., *Deep Reinforcement Learning Hands-On: Apply modern RL methods, with deep Q-networks, value iteration, policy gradients, TRPO, AlphaGo Zero and more*. 2018: Packt Publishing Ltd.
25. Luo, S., et al., *Robust walking control of a lower limb rehabilitation exoskeleton coupled with a musculoskeletal model via deep reinforcement learning.* Journal of neuroengineering and rehabilitation, 2023. **20**(1): p. 34.
26. MajidiRad, A., et al. *The effect of lower limb exoskeleton alignment on knee rehabilitation efficacy*. in *Healthcare*. 2022. MDPI.
27. Cao, W., et al., *Development and evaluation of a hip exoskeleton for lateral resistance walk exercise.* IEEE/ASME Transactions on Mechatronics, 2023.
28. Su, D., et al., *Review of adaptive control for stroke lower limb exoskeleton rehabilitation robot based on motion intention recognition.* Frontiers in Neurorobotics, 2023. **17**.
29. Ren, B., et al. *Motion trajectories prediction of lower limb exoskeleton based on long short-term memory (LSTM) networks*. in *Actuators*. 2022. MDPI.
30. Abbasi, M., et al. *AutoEncoder-based Safe Reinforcement Learning for Power Augmentation in a Lower-limb Exoskeleton*. in *2021* 9th RSI International Conference on Robotics and Mechatronics (ICRoM). 2021. IEEE.