# Exoskeleton Research Document

August 1, 2024

## 1 Initial research

### 1.1 Control methods for upper-limb exoskeletons

Control methods for upper-limb exoskeletons can have biological and nonbiological input. The biological inputs can be EMG and RMG signals. These inputs help control the exoskeleton by measuring the strength of the signals coming straight from the muscle, and determining which muscle the signal is coming from.
Non-biological signals could be force or torque. Measurement of these inputs can be done with a torque sensor in the joint and with a force myography band measuring the force exerted on a band around the arm when the wearer of the exoskeleton tightens their muscle[JGL12].
Sometimes it can be beneficial to combine both types of inputs such that the nonbiological signals are prioritized when the signal strength of the biological signals is insufficient.

The desired output from both types of input can be tuned either manually or with machine learning. Manual tuning typically involves gradually increasing the output signal until it provides adequate support to the individual who wears the exoskeleton. Using machine learning it is both feasible to utilize supervised aswell as unsupervised learning methods. A supervised method could be SVM used for classification of different types of signals or different levels of signal strength. The same classification could be found with an unsupervised method such as a neural network. Using machine learning has the advantage that the researcher does not have to find the optimal output/input relationship themselves.

The control method itself can have various degrees of complexity. The simplest control method is an on-off controller which just provide a binary signal to the actuators depending on the signal strength. The on-off method has been widely used in the early stages of exoskeleton control. Using more complicated control methods such as PI, PID, admittance or impedance control can provide improved control and perceived user experience.

### 1.2 FMG for Desired Human Motion Intention (DMI) detection/quantification

To get a satisfactory human-machine interaction, determining the human motion intention is crucial. For that purpose, it is common to use bio signals collected using Electromyography EMG or Force myography FMG, for example [ZWZ+22].

FMG is a non-invasive technology that senses changes in the volume of the musculotendinous complex beneath it. The armband used in this project consists of 8 Force sensing resistors (FSR) which translate force to a change in resistance. Therefore, the band must be placed above the muscles involved in the motion of interest, in this case, it is the flexion and extension of the elbow where the biceps and the triceps are the principal responsible muscles. There are a couple of merits and drawbacks of FMG that are worth noting. The advantages over the common EMG are[ZWZ+22]:

- Less pre-processing needed.

- Higher signal-to-noise ratio: this is due to its robustness against the common interference sources of EMG like electromagnetic interference, changes of conductivity of the skin due to sweat, etc.
- More affordable: this could increase the accessibility of the technology and therefore improve the accessibility of health technology such as prostheses or assistive exoskeletons.
- More suitable for high-speed motion.

However, the technology also presents some challenges, mainly related to variations of the positioning, orientation, and tightness of the device against the human body[ZWZ+22]. Some of these issues can be mitigated by performing a standardization of the FMG signal during the start-up[]. A pre-processing algorithm called SOMI significantly reduced the position and orientation dependence. In this study, the payload weight was classified by a lightweight neural network with an accuracy of 98.8%[She23].

## 1.3 Admittance control for exoskeleton

Admittance control is a common type of controller used in robotics that allows a robot to follow an external force by using the equations of motion for a virtual object. The controller outputs a desired position/acceleration/velocity/torque that can be used as a set point in a lower level controller ,such as PID [TF22].

The block diagram of an Admittance control system can be seen in the figure.

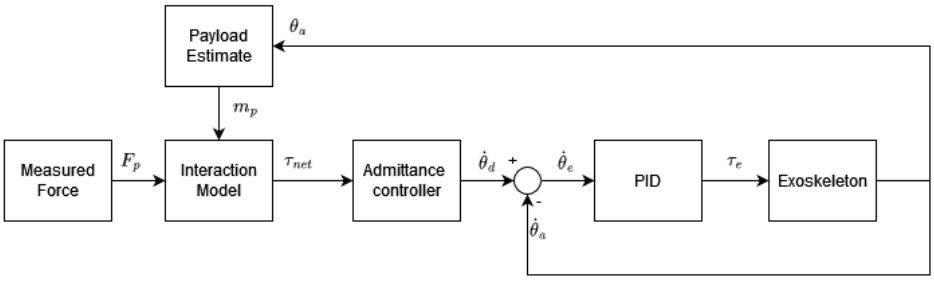


Figure 1: Block Diagram of Admittance control for a 1DOF exoskeleton

The typical equation for an admittance controller is as follows[MRUI19].

$$Y(s) = \frac{1}{B \cdot s + D + K \cdot s^{-}1} \tag{1}$$

- B is the desired inertia parameter.
- D is the desired dampening parameter.
- K is the desired stiffness parameter.

The reason these are the desired parameters comes from the fact that admittance control works by using a virtual object. Meaning the admittance controller will take the output from the interaction model and adjust it such that the output will behave with the desired dynamics of the virtual object[TF22].

### 1.3.1 The Interaction model of 1DOF exoskeleton

In order to apply admittance control to a 1DOF exoskeleton an interaction model is required. Assuming the desired output is the net torque exerted by the payload and the motor joint then it can be expressed as follows[MRUI19].

$$\tau_{net} = \tau_a + \tau_p \tag{2}$$

- $\tau_a$, the assisting torque provided by the motor joint.
- $\tau_p$ the torque exerted by the interaction between the human/exeskeleton and the payload

These torques can be further expressed as

$$\tau_a = k_a l_f m_p g sin(\theta_a) \tag{3}$$

$$\tau_p = F_p \cdot l_f \tag{4}$$

- $k_a$ is the assitance coefficient. It determines how much the joint should assist.
- $l_f$ is the distance between the joint and the center of the payload.
- $m_p$ is the mass of the payload.
- $g$ is the acceleration due to gravity.
- $\theta_a$ is the measured angle of the forearm joint w.r.t the upper arm which is assumed to be fixed, if this is not the case the angle of the upper arm w.r.t the shoulder can simply be added to this value.
- $F_p$ is the measured force from the interaction between the human/exoskeleton and the payload.

In order to obtain $m_p$ and $F_p$ there needs to be way to either directly measure them or estimate them somewhat accurately in real time. For the purpose of estimating the payload a strategy that can be used is the application of FMG sensors. The output from these sensors can be passed through a Deep Neural Network or State Vector Machine in order to classify the weight of the payload. This will result in some discrete payload weight values[She23].

## 1.4 Inverse dynamics model of exoskeleton

To get an interaction model that can be used to predict torque needed to compensate for the exoskeleton and payload an inverse dynamics model can be used. To achieve transparent control the applied torque from the motor $\tau_a$ needs to compensate for the torque of the payload and the exoskeleton $\tau_p$. Meaning

$$\tau_a = \tau_E + \tau_P \tag{5}$$

Where $\tau_E$ is the torque exerted by the exoskeleton and $\tau_P$ is the torque exerted by the payload. $\tau_E$ can be expressed as.

$$\tau_E = M(q)\ddot{q} + C(q, \dot{q}) + g(q) \tag{6}$$

$\tau_P$ can be expressed as.

$$\tau_P = J^T W_P \tag{7}$$

Where $M(q)$ is the inertial matrix, $C(q, \dot{q})$ is the coriolis effects and $g(q)$ are the gravity effects. $\ddot{q}, \dot{q}, q$ are the joint acceleration, velocity and position vectors. $J^T$ is the transpose of the Jacobian matrix at the end-effector. $W_P$ is the wrench applied by the payload[AOS23]. These torque expressions can be derived for the specific case of a 1-DOF exoskeleton arm by using the lagrangian.

$$L = K - P \tag{8}$$

K is kinetic energy and P is potential energy

$$K = \frac{1}{2} m_E L_E^2 \dot{\theta}^2 \tag{9}$$

$$P = m_E g L_E (1 - cos(\theta)) \tag{10}$$

$$L = \frac{1}{2} m_E L_E^2 \dot{\theta}^2 + m_E g L_E cos(\theta) - m_E g L_E \tag{11}$$

Where $m_E$ is the mass of the exoskeleton, $L_e$ is the distance from the joint to the COM of the exoskeleton arm, $\dot{\theta}$ is the joint angular velocity and $\theta$ is the joint angle. The exoskeleton torque can then be derived as

$$\tau_E = \frac{d}{dt}\frac{dL}{d\dot{\theta}} - \frac{dL}{d\theta} \tag{12}$$

$$\tau_E = m_E L_E^2 \ddot{\theta} + m_E g L_E sin(\theta) \tag{13}$$

The torque for the payload can be expressed similarly.

$$\tau_P = m_p L_p^2 \ddot{\theta} + m_p g L_p sin(\theta) \tag{14}$$

This would then give the final applied torque as.

$$\tau_a = m_E L_E^2 \ddot{\theta} + m_p L_p^2 \ddot{\theta} + m_E g L_E sin(\theta) + m_p g L_p sin(\theta) \tag{15}$$

Now this value does not take into account dynamic elements such as friction terms that compensate for static and viscous friction can be added as follows.

$$\tau_a = m_E L_E^2 \ddot{\theta} + m_p L_p^2 \ddot{\theta} + m_E g L_E sin(\theta) + m_p g L_p sin(\theta) + F_v \dot{\theta} + F_s sgn(\dot{\theta}) \tag{16}$$

When the exoskeleton is at rest the static friction is enough to keep the joint in place. This means that the gravity terms will only be relevant when the exoskeleton is in motion. Additionally, the contribution of friction from the joint will be counteracting the gravity when the joint is moving in the direction of gravity. Meaning in most instances the gravity will have a small effect on the torque. So in the interest of making a system that is simpler to implement the applied torque $\tau_a$ can be expressed as.

$$\tau_a = m_E L_E^2 \ddot{\theta} + m_p L_p^2 \ddot{\theta} = (J_E + J_p)\ddot{\theta} \tag{17}$$

## 1.5 Applications of upper limb exoskeletons

The applications of upper limb exoskeletons spans a fairly wide spectrum. Intuitively, when just thinking about the applications, you can consider just about any kind of work which requires physical hard work, where it would be ideal to have some type of assistance. This comes from that a high number of people dealing with physical work in their everyday life report that there muscles is overloaded [MRS22].

Many factors contribute to muscle deterioration, some of them are more natural then others, like ageing and the genetics of how a human being is build and so on. Some of the other reasons which aren't natural can include posture (The way we position ourselves during different work related tasks, or how we sit on a chair), accidents that leads to permanent damage to our muscles, or worn down muscles, caused by overload or performing repetitive work tasks [Mar07].

One of the most exposed muscle groups is placed in the shoulders. This is due to the fact that both active physical work, and work done in an office in a chair harms the muscle group in the shoulders in different ways. Therefore, the shoulders, with it's large range of motion is the muscle group which can be expected to take the most damage over time [TC00].

Since the shoulder area is the most common place for muscle overload, it makes sense to focus on creating an exoskeleton for the upper part of the body (upper limb). Therefore, this report (Scientific paper) aims to design for an upper limb controlled exoskeleton.

Upper limb controlled exoskeletons can be used in many different applications, and it does not really limit it all that much that the lower part is not considered. The range of the upper limb exoskeleton includes applications for military, medical, industrial, and sports [Bog18]. The type of support the exoskeleton delivers can also be classified as active or passive, and of course which part it supports (In our case upper limb) [Bog18].

A good example of an exoskeleton application combines the world of the military with the world of medicine. In therapy or rehabilitation, exoskeletons assist individuals in overcoming physical limitations [Bog15]. This is especially helpful for veterans or soldiers who have been injured in combat. Furthermore, with ongoing advancements in materials and control mechanisms, exoskeletons are finding applications in various other fields [Bog18].

## 2 Neural Network

For the project, there is some need of controlling the exoskeleton. It was assessed that more conventional control methods would not be sufficient, and therefore it was decided to try and implement some form of machine learning.

Since the architectural design of the control of the exoskeleton can quickly become fairly complicated, a good approach would be to try and implement some form of neural network, which is a type of machine learning method.

They consist of interconnected nodes, called neurons, organized in layers. Each neuron receives input signals, processes them using an activation function, and produces an output signal. The connections between neurons are weighted, allowing the network to learn patterns and make predictions based on input data [Bis06].

Let's denote:

- $x$ as the input vector to the neural network.
- $w$ as the weight matrix, representing the connections between neurons in adjacent layers.
- $b$ as the bias vector, representing the offset added to the weighted sum.
- $z$ as the weighted sum of inputs and biases.
- $a$ as the output of the neuron after applying an activation function.

The output of a neuron can be mathematically represented as:

$$z = w \cdot x + b$$

$$a = f(z)$$

Here, $f$ is the activation function, which introduces non-linearity into the model.

Activation functions introduce non-linearity into the neural network, allowing it to learn complex patterns in the data. Common activation functions include:

- **Sigmoid:** $\sigma(z) = \frac{1}{1+e^{-z}}$
- **ReLU (Rectified Linear Unit):** $\text{ReLU}(z) = \max(0, z)$
- **Tanh:** $\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$

During the training process, the neural network learns optimal weights and biases to minimize a predefined loss function, typically through gradient descent optimization algorithms like backpropagation. Backpropagation calculates the gradients of the loss function with respect to the network parameters, allowing for the iterative adjustment of weights and biases to improve the model's performance [Bis06].

There are many different neural network architectures, which each serves it's purpose, depending on what you are trying to do. In our case with the exoskeleton, we need something that can learn long-term dependencies from some data. After some research, it was discovered that LSTM would be a good application for this.

## 3 Long Short Term Memory (LSTM)

Regular RNNs experience the vanishing/exploding gradient which restrains the possibility of learning long-term dependencies of the data. A solution to this problem is to use the architecture LSTM neural network. This architecture has addressed this problem by creating a path where information can travel from step to step without being altered and with a system of gates that regulate the information flow.

This recurrent neural network architecture has some distinct elements that constitute them:

- Cell state: is the long-term memory of the system and it is managed and updated by the gates.

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{18}$$

- Forget Gate: Its role is to eliminate or forget information from the cell state that is no longer relevant for future predictions.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{19}$$

- Input Gate: regulates the amount of information from the current input and previous hidden state that would be stored in the cell state.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{20}$$

Since the gate has a sigmoid activation function, it outputs a number or a vector of numbers from 0 to 1. This behavior is analogous to the one a gate may have, for example, if the forget gate outputs a 0, then all the information contained in the cell state is deleted or forgotten; conversely, if the gate's output is 1 the information is kept.

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{21}$$

Equation 21 shows the candidate for the cell state, it is computed as a combination of the current input and the previous hidden state, transformed by a weight matrix and offset by a bias term. This combination is then passed through the hyperbolic tangent function, which scales the result to fall within the range of -1 to 1. This nonlinear transformation allows the candidate cell state to manage the magnitude of its values, aiding in the regulation of the network's information flow.

- Output Gate: decides how much from the cell state is carried out to the hidden state and output.

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{22}$$

$$h_t = o_t * \tanh(C_t) \tag{23}$$

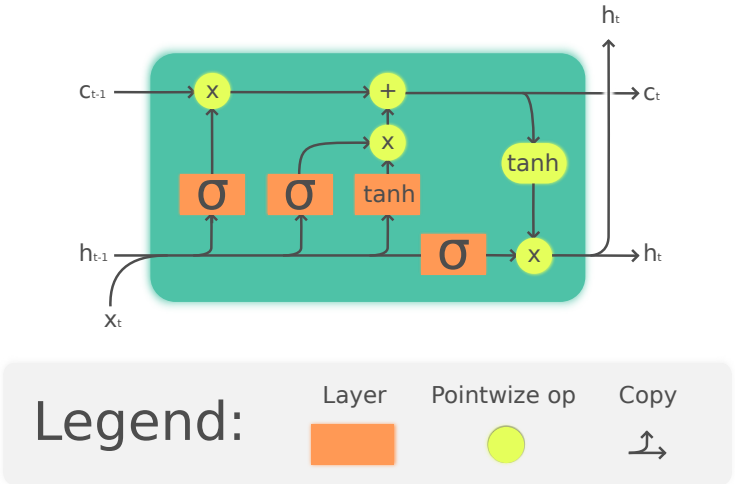


Figure 2: LSTM unit computational diagram.[Che24b]

With the components of the LSTM unit now identified, Figure 2 provides a summary and visual representation to enhance comprehension of how each component plays a role in the information processing workflow. First, the forget gate decides how much information is kept or remembered from the previous cell state (long-term memory), subsequently, the input gate section is in charge of updating the

cell state with new information provided by the candidate to the cell state. This process of the cell update can be mathematically shown in the equation18. Finally, the output gate decides how much of the cell state becomes the next hidden state and output.

Utilizing this architecture offers significant benefits over traditional RNNs, including overcoming the challenges of vanishing or exploding gradients, which enhances the retention of long-term dependencies within the input data. Additionally, its capacity to disregard irrelevant or noisy information contributes to its resilience against such data disturbances.

However, this method has its drawbacks. Its tendency to overfit and the requirement for substantial data volumes to achieve optimal performance.

# 4 Data Preprocessing for LSTM

The main objective of preprocessing is to facilitate model training and improve model performance. For training predictive models like ARIMA, the time series is often transformed into a stationary time series by removing its trend and seasonal components using techniques like differencing. These classical preprocessing techniques have a different effect on LSTM training; this has been studied in the following article [TVRRCOdJD19]. According to this study, there is no significant improvement in removing the trend. This is because the LSTM architecture, with its gates managing the flow of long-term information, effectively adapts to changes in data trends over time. It can handle non-stationary data better than statistical models like ARMA or ARIMA; which require the data to have stationary statistical properties to be effective.

Deciding the number of lagged data points to include in the model is crucial. This is typically determined by analyzing the partial autocorrelation of the data, which helps identify relevant lags that contribute additional information (include plot and add a discussion).

For neural network training, particularly with LSTMs that use the hyperbolic tangent activation function, it is advisable to scale the data to match the function's output range of -1 to 1. Otherwise, the risk of vanishing gradients will increase as the gradient of tanh quickly goes to 0 in the asymptotic zones ($y = -1$ and $y = 1$) of the activation function i.e., its extremities [Bae23].

After the prediction is made, all transformations must be undone such that the predictions are brought back to their original format for better interpretation.

# 5 LSTM Implementation with Keras

The LSTM was programmed in Python using the Keras deep learning API. There are a few aspects to consider regarding the data format expected as input and the hyperparameters.

## 5.1 Input format

The input format has to be in a 3D tensor format with the following structure: $(batch, time\_steps, features)$

- Batch Size: Each batch contains one or multiple samples, where each sample is one sequence.
- Time steps: This dimension specifies the number of time intervals considered in each sample. For instance, if you are using time series data for forecasting, having three time steps per batch would mean that the data from the three previous time intervals (e.g., days, minutes, seconds) are used to predict future values.
- Features: The third dimension represents the number of features or variables recorded at each time step in the data. In this project, there is only one feature in the dataset which is the torque to compensate at the joint.

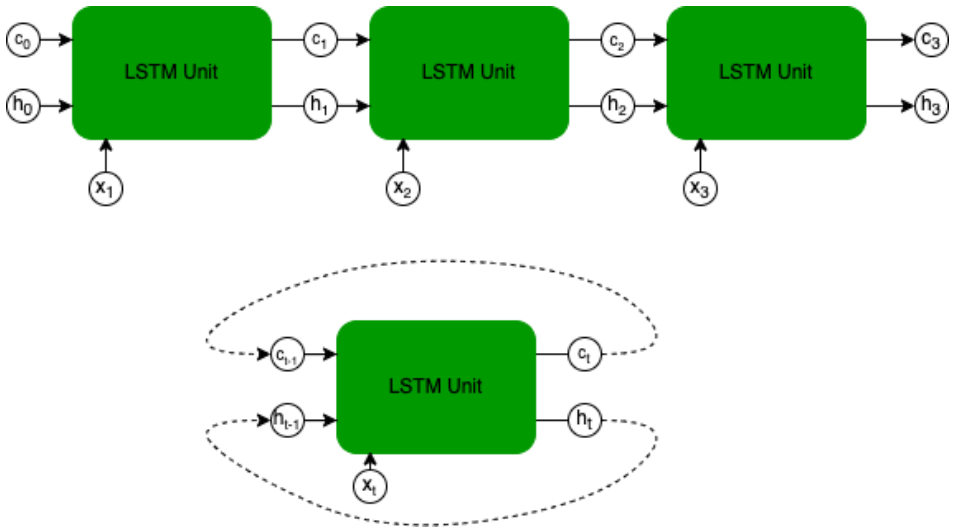


Figure 3: This diagram features a representation of an LSTM layer consisting of a single unit, depicted in two different forms to highlight that the same unit processes the input. The upper part of the diagram shows the unfolded graph, and the lower part shows the folded graph.

The size of each of the dimensions must be specified when creating an LSTM layer.

Example of how to prepare the input data for an LSTM that takes three values as input. If we have a 1D time series and we arrange it in the following array: (samples, time steps) where there are samples that have three time steps. Then if we specify the batch size to be 1 sample or sequence we get the following tensor with dimensions $(batch\_size = 1, time\_steps = 3, features = 1)$:

$$\text{Tensor} = \left\{ \begin{bmatrix} -0.1032 \\ 0.4106 \\ 0.1440 \end{bmatrix}, \begin{bmatrix} 0.4106 \\ 0.1440 \\ 0.1550 \end{bmatrix} \right\}$$

Here there are two batches. Each batch contains three-time steps, and at each time step, there is one feature value.

- Each major bracket represents a batch.
- Each column matrix within a bracket represents the time steps for that batch.
- Each element in the column matrix represents a feature value at that time step.

## 5.2 Hyperparameters

There are multiple choices one can make when designing an LSTM layer, this choice can be made by changing its hyperparameters.

Some of the most relevant hyperparameters are:

- Units: It refers to the number of LSTM units on the layer. Each one of them processes the input independently from each other and they do not share any parameters. An example of a unit layer that processes three inputs is illustrated in figure 3. When more units are added to an LSTM layer, the dimensionality of the hidden state increases relative to the dimensionality of the input, and therefore the representation encoded becomes richer in information. Allowing the LSTM to retain more complex patterns.
- Return sequences: If this is false only the last hidden state from each unit would be the output. If true, then the output becomes all the hidden states from all time steps from each unit. So if there are two units and three time steps, for example, there will be six outputs from the layer; which correspond to all hidden states from each time step from both units.

- Stateful: If it is set to be true, then the last state produced at sample i is fed to the next batch which starts at sample i. If there is only one sample per batch then the last state is passed to the next batch as the initial hidden state; maintaining the memory between batches.

The output can also be fed to a feed-forward neural network with the output dimensionality of choice, that way it is possible to specify how many steps into the future we want to predict for example.

Putting all together, this is an example of how to setup a LSTM model in Python with Keras:

```
...
n_batch=1
model = Sequential()
model.add(LSTM(n_neurons, batch_input_shape=(n_batch, X.shape[1], X.shape[2])))
model.add(Dense(y.shape[1]))
model.compile(loss='mean_squared_error', optimizer='adam')
```

- $X$ is the input dataset of size $(samples, time\_steps, features)$.
- $y$ is the label set. It has as many columns as samples to predict.

## 5.3 Model Quantization

Quantization is a technique used to reduce the size of machine learning models and inference time; which might be crucial for deployment in edge devices with limited storage and computational resources. This is achieved by reducing the bit resolution used to store numbers, which can lead to a decrease in accuracy[Che24a].

### 5.3.1 Quantization format

Given the original tensors of the model are Float32 there are a few possibilities to consider to reduce the bit size:

- Float16: if compatible hardware with operations in this format is available. reduces the memory required to store a number by half.
- Int8: consists of mapping Float32 numbers to the 8-bit integer range; which is the range of integers from -127 to 128. For that, the maximum and minimum numbers must be known. The memory required is four times smaller.

### 5.3.2 Quantization on a machine learning model

Quantization can be applied to different parts of the model achieving different effects:

- Weight-only quantization: This consists of only reducing the bit precision of the weights of the machine learning model. Reduces inference latency[Che24a].
- Dynamic range quantization: In addition to weight quantization, activations are also quantized. Unlike weights, the range for activations is estimated dynamically during inference. This method is useful when the range is not known beforehand, such as with input tensors[Che24a].
- Static quantization (Full-Integer quantization): Because the activation ranges need to be precisely known, a representative input dataset is used to determine the quantization parameters accurately. After a few iterations, these parameters are set, resulting in a fully integer model, including input and output tensors. This method offers the greatest reduction in model size and ensures all operations are integer operations, which significantly improves computational efficiency on compatible hardware.

### 5.3.3 Types of quantization methods

There are two ways to apply quantization to a machine learning model: during training or after training.

- Quantization-aware training: the model is trained having into account that the model will be quantized, that way optimization can be applied to reduce the effect of the quantization on the model performance. This method yields the best results at the expense of being more complex to apply.
- Post-training quantization: the model is quantized after it is fully trained. Even though accuracy can suffer from this process it's a simpler to implement approach[Che24a].

On this project, full integer quantization was used to ensure compatibility with the EloquentTensorflow32 library.

# 6 Testing

## 6.1 Data collection procedure with FMG-band

**Purpose of test**
Collect data from FMG band with varying weight loads.

**Requirements**

- FMG-band
- Computer with MatLab installed, able to connect to the FMG band via Bluetooth
- Test subject
- Data collector (can be the same person as test subject).
- A dumbbell of known weight.

**Test setup**
The test requires two people, one wearing the FMG band (test subject) and one capturing the data (data collector). The data collector needs a computer with the test software open in MatLab and be able to connect to the band via bluetooth. The test should be done twice for every person, one with and and one without holding a weight.

**Testprocedure**

- Subject attaches the FMG band to the right arm so that the sensors in the band are right on top of the place where the bicep muscle grows the most when the subject is flexing.
- The data collector connects the computer to the FMG band via bluetooth.
- The subject follows the setup and calibration of the band (this is always done without holding a weight).
- The subject then lifts their arm to be perpendicular to the body, while keeping the elbow flush with the body, holding it there for 10 seconds, before lowering it again (this can be done with or without a dumbbell).
- The subject rests, not flexing, for 10 seconds
- The subject repeats step 4 and 5 ten times (for a total og 10x10s rest and 10x10s flex).
- The test subject then waits for the test time to run out
- The data is automatically saved to a .xlsx file
- The test is done.

**Test results**
The test results are stored in an excel file format (.xlsx) with 8 columns of data, each containing data from one of the 8 load cells.

**Comments**
The standard weight is a bar of 1.23 kg with two plates of 1.13 kg, resulting in a combined weight of 3.49 kg.

## 6.2 SOMI algorithm

To make sure, that the data does not depend on how or on which arm you put the FMG-band, the paper [She23] came up with a "SOMI" algorithm. This algorithm is a 4-step process which normalizes the data in a way, such that the data is independent on the variety in placement of the FMG-band. The algorithm is described below:

1. Acquire sensory data from FMG-band
    - Sensory data is a matrix with 8 columns (one for each sensor) and n rows, where n is number of samples.
2. Column-wise normalization
    - By using the formula $z = \frac{(x-\mu_{tr})}{\sigma_{tr}}$, all 8 columns of data (one for each sensor) is normalized, by subtracting the mean and dividing by the variance.
3. Row-wise normalization
    - All the values of each sensor is normalized to a value between 0 and 1 by using the formula: $z = \frac{z}{max(abs(z))}$.
4. Sort sensory data in ascending order
    - The last step in the SOMI algorithm is ordering the columns of data in ascending order based on the mean value of each column.

After passing the data through this algorithm, the data is normalised and ready for the neural network.

## 6.3 Transparency Test

**Purpose of test**
To confirm the validity of out approach in terms of improving transparency using Inverse Dynamics and LSTM. The way this is achieved by the test is through measuring the interaction torque using the built in load cell placed at the wrist. By comparing the load cell measurements with and without the control algorithm enabled as well as with and without the payload. It can be determined whether or not our approach improves transparency
**Requirements**

- Exoskeleton arm
- Exoskeleton harness
- Test subject
- PC with Matlab and Arduino IDE installed
- 1.1kg Payload
- Matlab code for receiving serial monitor data. Found here
- Motor control code. Found here

**Test Setup**
The test requires two persons. One to wear the exoskeleton and one to run the data collection software. The code should be configured such that the correct data is collected. Overall there are four iterations of this test. They are as follows:

- Control disabled and 0kg payload
- Control disabled and 1.1kg payload

- Control enabled and 0kg payload
- Control enabled and 1.1kg payload

To account for this the following code parameters need to be changed between tests in order to get the correct data. For control enabled test the variable 'char control status' must be 'C'. This enables the control algorithm. This variable can be found in the motor control code here on line 31. Conversely for control disabled tests then 'char control status = 'D'. To change the payload mass the variable 'float payloadMass' can simply be changed to the desired value. Found here on line 34.

**Test Procedure**

1. Mount exoskeleton on test subject
2. Make necessary adjustments to motor control code
3. Compile and Run motor control code on exoskeleton
4. Run matlab data collection code
5. Have test subject pick up payload if needed
6. Have test subject perform 10 flexions and extensions of the arm.
7. Save Collected interaction torque, inverse dynamics and LSTM prediction data.
8. Repeat 2-8 for all test iterations.

**Results**

The test results can be stored in a .mat file where they can be processed. For the interaction torque the RMS can be calculated for the parts of the data where the exoskeleton is in motion. The inverse dynamics torque and the LSTM prediction torque can be used to validate the accuracy of the LSTM prediction.

# 7 Code Understanding

This section explains the detalied work regardings the algorithms we implemented in our work.

## 7.1 Admittance Control

The Admittance filer equation was discretized into a first-order IIR filter in Direct Form I. First, the admittance transfer function is transformed from the s-domain to the z-domain using the first-order approximation of s in terms of z, i.e. the bilinear transform.

$$\frac{\omega(s)}{\tau(s)} = \frac{1}{M \cdot s + D} \tag{24}$$

$$s \leftarrow \frac{2}{T}\frac{1 - z^{-1}}{1 + z^{-1}} \tag{25}$$

$$\frac{\omega(z)}{\tau(z)} = \frac{1}{M \cdot \frac{2}{T}\frac{1-z^{-1}}{1+z^{-1}} + D} \tag{26}$$

Rearrange terms and transform from a z-domain transfer function to a difference equation.

$$[2M(1 - z^{-1}) + DT(1 + z^{-1})]\omega(z) = T(1 + z^{-1})\tau(z) \tag{27}$$

$$2M\omega[n] - 2M\omega[n-1] + DT\omega[n] + DT\omega[n-1] = T\tau[n] + T\tau[n-1] \tag{28}$$

Then isolate for $y[n]$.

$$\omega[n] = -\frac{(-2M + DT)}{2M + DT}\omega[n-1] + \frac{T}{2M + DT}(\tau[n] + \tau[n-1]) \tag{29}$$

The resulting equation can now be implemented in an embedded computer.

This filter can be found in the code as a function: float admittance_filter_MR(float interaction_torque).

## 7.2 Payload classification algorithm

For the payload classification, a MATLAB script was first used to pull data out of a FMG band, and afterwards the data was preprocessed, before inserted into the actual payload classification algorithm.

In 7.2 an overview of the algorithm for the payload classification can be seen, in the form of a pseudo code.

**Algorithm 1** Pseudo code implementation of payload classification algorithm

**Load and Combine Data:**
Load datasets $D_1, D_2, \ldots, D_n$
Combine datasets: $D \leftarrow \text{concat}(D_1, D_2, \ldots, D_n)$
**Assign Labels:**
Assign labels $y$ to combined data $D$
**Preprocess Data:**
Split data: $X \leftarrow D_{\text{features}}, y \leftarrow D_{\text{labels}}$
Normalize features: $X' \leftarrow \frac{X-\mu}{\sigma}$ where $\mu$ and $\sigma$ are mean and standard deviation
Split into training and testing sets: $(X_{\text{train}}, X_{\text{test}}, y_{\text{train}}, y_{\text{test}}) \leftarrow \text{split}(X', y)$
Further normalize training and testing sets: $X'_{\text{train}} \leftarrow \frac{X_{\text{train}}-\mu_{\text{train}}}{\sigma_{\text{train}}}, X'_{\text{test}} \leftarrow \frac{X_{\text{test}}-\mu_{\text{test}}}{\sigma_{\text{test}}}$
**Convert Labels:**
One-hot encode labels: $y'_{\text{train}} \leftarrow \text{one\_hot}(y_{\text{train}}), y'_{\text{test}} \leftarrow \text{one\_hot}(y_{\text{test}})$
**Build and Compile Model:**
Define Sequential model $M$
Add layers: $M \leftarrow \text{Dense}(128, \text{activation} = 'relu'), \ldots, \text{Dense}(10, \text{activation} = 'softmax')$
Compile model: $M \leftarrow \text{compile}(M, \text{optimizer} = 'adam', \text{loss} = 'categorical\_crossentropy', \text{metrics} = ['accuracy'])$
**Setup Callbacks:**
TensorBoard callback: TensorBoard(...)
ModelCheckpoint callback: ModelCheckpoint(...)
**Train Model:**
Train model: $M_{\text{history}} \leftarrow M.\text{fit}(X'_{\text{train}}, y'_{\text{train}}, \text{validation\_split} = 0.2, \text{epochs} = 50, \text{callbacks} = [\ldots])$
**Evaluate Model:**
Evaluate on test data: test_results $\leftarrow M.\text{evaluate}(X'_{\text{test}}, y'_{\text{test}})$
Print test results: print(test_results)
**Predict and Display:**
Predict on test data: $y_{\text{pred}} \leftarrow M.\text{predict}(X'_{\text{test}})$
Display confusion matrix: conf_matrix $\leftarrow \text{confusion\_matrix}(y_{\text{test}}, y_{\text{pred}})$
Print confusion matrix: print(conf_matrix)
**Optional: K-Fold Cross Validation:**
**for** each fold $k \in \{1, 2, \ldots, K\}$ **do**
    Split data into $k$-th fold: $(X_{\text{train\_k}}, X_{\text{val\_k}}, y_{\text{train\_k}}, y_{\text{val\_k}}) \leftarrow \text{k\_fold\_split}(X', y, k)$
    Normalize $k$-th fold: $X'_{\text{train\_k}} \leftarrow \frac{X_{\text{train\_k}}-\mu_{\text{train\_k}}}{\sigma_{\text{train\_k}}}, X'_{\text{val\_k}} \leftarrow \frac{X_{\text{val\_k}}-\mu_{\text{val\_k}}}{\sigma_{\text{val\_k}}}$
    One-hot encode $k$-th fold labels: $y'_{\text{train\_k}} \leftarrow \text{one\_hot}(y_{\text{train\_k}}), y'_{\text{val\_k}} \leftarrow \text{one\_hot}(y_{\text{val\_k}})$
    Train model: $M_{\text{history\_k}} \leftarrow M.\text{fit}(X'_{\text{train\_k}}, y'_{\text{train\_k}}, \text{validation\_data} = (X'_{\text{val\_k}}, y'_{\text{val\_k}}), \text{epochs} = 50)$
    Evaluate model: val_results_k $\leftarrow M.\text{evaluate}(X'_{\text{val\_k}}, y'_{\text{val\_k}})$
    Print $k$-th fold results: print(val_results_k)
**end for**

In figure 4, a more simple overview of the code can be seen in the form of a flowchart, displaying the code workflow and the architecture.

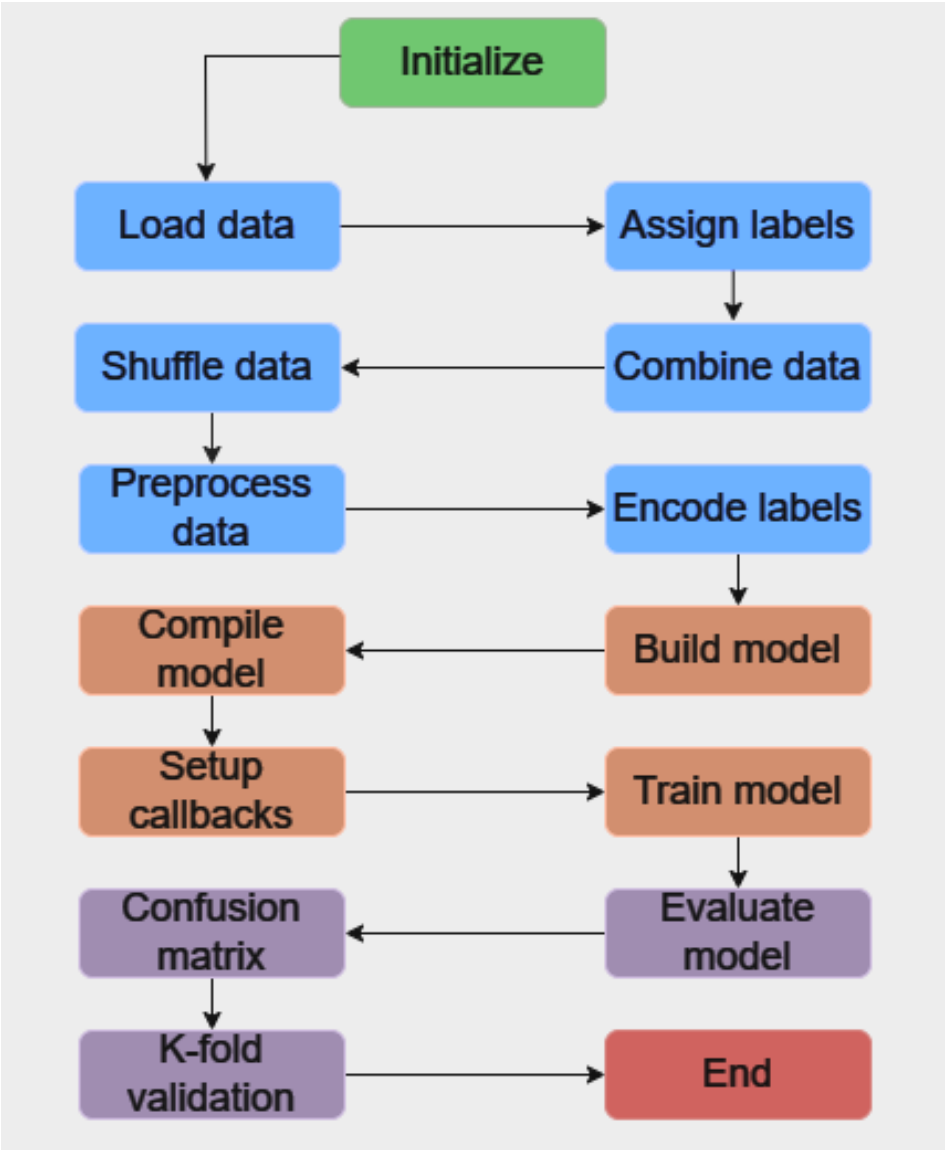


Figure 4: Flowchart of payload classification algorithm. Green color indicates initialization, blue color indicates the preprocessing of the data, orange indicates the working blocks of the model, purple indicates the evaluation of the model, and red indicates the end of algorithm.

The algorithm starts by acquiring the payload data, which is stored in raw data format from CSV files. Before training the model, so after the data is loaded, some preprocessing is happening. This preprocessing includes handling missing values, normalizing features to a standard scale, and separating features from the target labels, and using the SOMI algorithm, which was explained earlier.

Once the data is preprocessed, we split it into training and test sets. The training set is used to train the machine learning model, while the test set is reserved for evaluating its performance. This division helps us assess how well the trained model generalizes to unseen data.

Since machine learning algorithms typically require numerical inputs, we encode categorical labels (such as payload types or states) into a numerical format. One common technique for this task is one-hot encoding, where each category is represented by a binary vector.

With the data prepared, we define the architecture of the machine learning model. This involves specifying the type and configuration of the model's layers, such as dense layers in a neural network. The model's architecture is designed to effectively capture patterns and relationships present in the

payload data.

After defining the model architecture, we compile it by specifying the loss function, optimizer, and evaluation metrics. The choice of these components depends on the nature of the classification task and the desired model performance.

The compiled model is then trained using the training data. During training, the model learns to map input features to their corresponding labels by adjusting its internal parameters through an optimization process. This iterative training phase aims to minimize the defined loss function.

Once training is complete, we evaluate the model's performance using the test data. This evaluation assesses how well the model generalizes to unseen data and provides insights into its accuracy, precision, recall, and other relevant metrics. For the evaluations of the predictions, we employ confusion matrices, which provide a detailed breakdown of the model's classification performance across different payload categories, in order to get a visual overview of performance.

To ensure robustness and reliability in our payload classification process, we employ k-fold cross-validation. This technique involves dividing the dataset into k subsets or "folds" of approximately equal size. For each iteration of the cross-validation process, one of the k subsets is retained as the validation set, while the remaining k-1 subsets are used as the training set. This process is repeated k times, with each subset serving as the validation set exactly once.

By averaging the performance metrics obtained from the k iterations, we obtain a more accurate estimate of the model's performance compared to a single train-test split. This approach helps mitigate the variability in model performance that may arise from differences in the training and test data partitions.

Furthermore, k-fold cross-validation allows us to assess the generalization capabilities of the model across different subsets of the data, providing insights into its stability and robustness. This comprehensive evaluation helps to identify potential issues such as overfitting or underfitting and guides us in fine-tuning the model parameters for optimal performance.

### 7.3 Inverse Dynamics and LSTM Torque Prediction

In the code implementation the Inverse Dynamics is applied using a function based on equation 17. This function is as follows.

$$\tau_a = (0.21 + 0.09 \cdot m_p)\ddot{\theta} \tag{30}$$

The values $J_E$ and $l_p$ from equation 17 are known and are therefore set as constants. $m_p$ should come from the payload classification but in this case it is simply set as a constant before the code is compiled to avoid the need for implementation of online functionality with the payload classifier. $\ddot{\theta}$ is calculated from motor encoder readings and passed to the Inverse Dynamics torque calculation function on call.

- **Initialise Inverse Dynamics:** Defining inverse dynamics objects, constants and initialising variables.
- **Initialise LSTM:** Load pre-trained LSTM model, define number of operations and arena size.
- **Read Encoder:** Read angle and velocity data from motor encoder and pass it through a low-pass filter. Calculate acceleration from velocity.
- **Apply Inverse Dynamics:** Pass acceleration reading from encoder to torque calculation function. The torque calculation function applies the equation seen in 30.
- **Scaling Transformation:** Scales the calculated torque value to a value between 1 and -1.
- **Differencing:** Differences the scaled calculation
- **Predict Torque with LSTM:** Utilises the pre-trained model to predict the torque 5 time steps in the future using the model found here. The model uses three inputs namely the current torque and the two preceding torques.
- **Reverse Differencing:** Takes the prediction and reverses the differencing from earlier
- **Inverse Scaling Transformation:** Reverts the predicted torque back to the original scaling.

- **Admittance Filter:** The predicted torque is passed to the admittance filter which uses equation 29. Values $M = 0.1 + 0.09 \cdot m_p$, $D = 0.01$ and $T = 0.002$.

How the inverse dynamics along with the LSTM torque prediction fits into the remaining motor control code can be seen on figure 5.

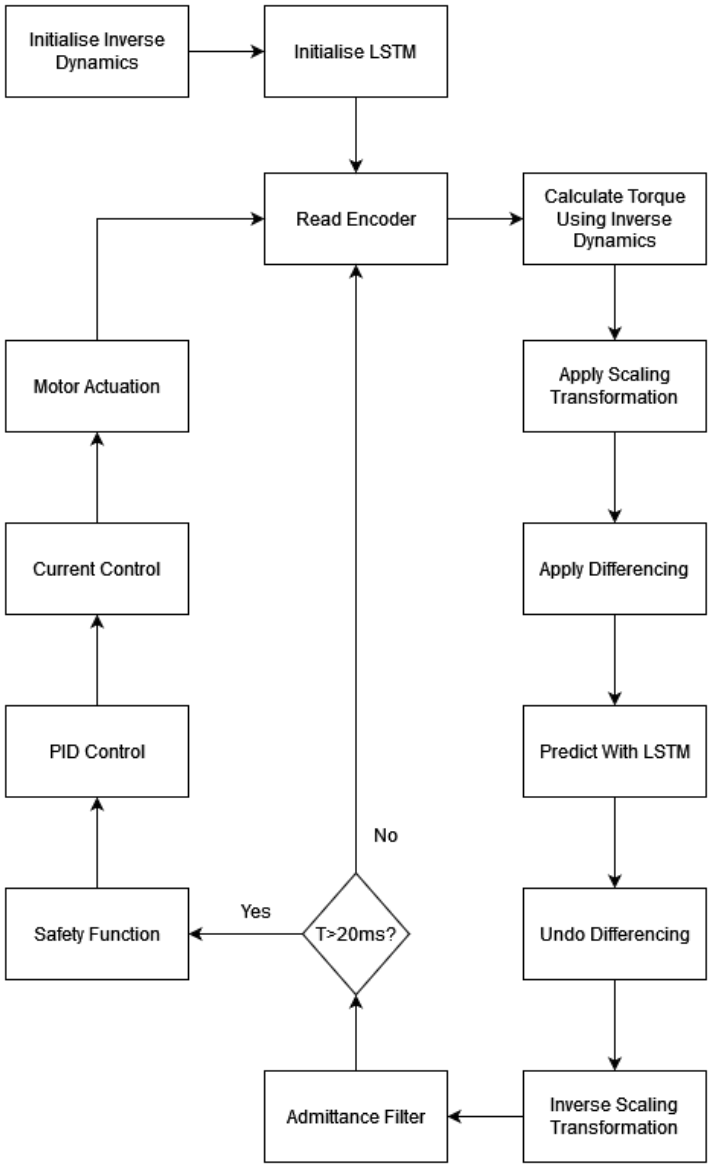


Figure 5: Overview of the exoskeleton motor control code with Inverse Dynamics and LSTM torque prediction applied

## 7.4 Preexisting functions

The remaining parts of the code loop namely the safety function, PID controller and Current Control was already implemented in the exoskeleton code that was provided for us and did not require any modification on our part. The same can be said for the admittance filter, the only thing we changed for the admittance filter was the input and virtual object parameters(M and D). Further description of these functions can be found in the next section

### 7.4.1 Safety Function

This function takes the desired velocity output from the admittance filter and checks if it is above the maximum allowed velocity of 4.7 rad/s. If this is the case it simply sets the desired velocity to the maximum value. Additionally, if the position of the joint is below 10 deg or above 110 deg the function will set the desired velocity to 0 if the velocity would cause the arm to move to the lower limit or the upper limit of the joint angle respectively.

### 7.4.2 PID Controller

The PID controller takes the error between the desired velocity and the encoder's measured velocity. The PID controller in our case though is a simple P controller with a gain of 1. Meaning the output becomes

$$Y_{PID} = k_P error = error \tag{31}$$

### 7.4.3 Current Control

This function translates the PID controller output to a PWM signal which is then written to the analog pins that connect to the motor drivers. The drivers can then translate this into a current that actuates the motors.

# References


[AOS23] François Charpillet Pauline Maurice Serena Ivaldi Alexandre Oliveira Souza, Jordane Grenier. Towards data-driven predictive control of active upper-body exoskeletons for load carrying. https://ieeexplore.ieee.org/document/10187548, June 2023.

[Bae23] Baeldung. Normalizing inputs of neural networks. https://www.baeldung.com/cs/normalizing-inputs-artificial-neural-network, 2023. Accessed: 2023-04-20.

[Bis06] Christopher M Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[Bog15] Robert Bogue. Robotic exoskeletons: a review of recent progress. https://www.emerald.com/insight/content/doi/10.1108/IR-08-2014-0379/full/html, Januar 2015.

[Bog18] Robert Bogue. Exoskeletons – a review of industrial applications. https://www.emerald.com/insight/content/doi/10.1108/IR-05-2018-0109/full/html, July 2018.

[Che24a] Wei Loon Cheng. Introducing post-training model quantization feature and mechanics explained, 2024. Accessed: 2024-05-24.

[Che24b] Guillaume Chevalier. Image. Wikimedia Commons, 2024. CC BY-SA 4.0.

[JGL12] T.S.S Jayawardane J.M.P Gunasekara, R.A.R.C Gopura and S.W.H.M.T.D Lalitharathne. Control methodologies for upper limb exoskeleton robots. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=arnumber=6427387, December 2012.

[Mar07] William S. Marras. The future of research in understanding and controlling work-related low back disorders. https://www.tandfonline.com/doi/full/10.1080/00140130400029175, February 2007.

[MRS22] Tobias Moeller, Janina Krell Roesch, and Thorsten Stein. Effects of upper-limb exoskeletons designed for use in the working environment—a literature review. https://www.frontiersin.org/articles/10.3389/frobt.2022.858893/full, 2022.

[MRUI19] Shaoping Bai Muhammad R. U. Islam. Payload estimation using forcemyography sensors for control of upper body exoskeleton in load carrying assistance. Modelling, Identification and Control, 2019.

[She23] Abdullah Tahir; Zeliang An; Shaoping Bai; Ming Shen. Robust payload recognition based on sensor-over-muscle-independence deep learning for the control of exoskeletons. https://ieeexplore.ieee.org/document/10102267, 2023.

[TC00] GC Terry and TM Chopp. Functional anatomy of the shoulder. https://pubmed.ncbi.nlm.nih.gov/16558636/, July 2000.

[TF22] Kenji Tahara Takuto Fukiji. Series admittance–impedance controller for more robust and stable extension of force control. https://robomechjournal.springeropen.com/articles/10.1186/s40648-022-00237-5, 2022.

[TVRRCOdJD19] Daniel Trujillo Viedma, Antonio Jesús Rivera Rivas, Francisco Charte Ojeda, and María José del Jesus Díaz. A first approximation to the effects of classical time series preprocessing methods on lstm accuracy. In *Advances in Computational Intelligence: 15th International Work-Conference on Artificial Neural Networks, IWANN 2019, Gran Canaria, Spain, June 12-14, 2019, Proceedings, Part I 15*, pages 270–280. Springer, 2019.

[ZWZ+22] Zhuo Zheng, Zinan Wu, Runkun Zhao, Yinghui Ni, Xutian Jing, and Shuo Gao. A review of emg-, fmg-, and eit-based biosensors and relevant human–machine interactivities and biomedical applications. *Biosensors*, 12(7):516, 2022.