# POLITECNICO DI TORINO

**Master's Degree in Mechatronic Engineering**


**Master's Degree Thesis**

# Transformer-Based Prediction of Human Motions and Contact Forces for Physical Human-Robot Interaction

**Supervisors**

**Prof. Alessandro RIZZO**

**Prof. Marco COGNETTI**

**Candidate**

**Alessia FUSCO**

**October 2023**

# Summary

As the field of robotics continues to evolve, there is a growing emphasis on achieving seamless collaboration between humans and robots. This thesis delves into the intricate dynamics of human-robot interaction. It lays the foundation for a more natural and fluid collaboration paradigm, inspired by human-to-human interactions. This approach replaces the traditional reactive model with a predictive one, where the robot does not merely react to human stimuli but anticipates human intentions and responds accordingly.

The purpose of this thesis is to harness the power of Transformer Neural Networks to forecast human forces and motions. In order to train our Neural Network we collected data from brief interactions between a human and a robotic arm, specifically a Panda arm developed by Franka Emika.

In our approach, we implemented an impedance controller on the robotic arm to establish a secure and harmonious collaboration. The manipulator not only ensures the safety of the interaction but also allows us to estimate the force exerted by the human operator on the end effector.

We captured kinematic information from the human using the Xsens motion capture suit. This tool enables the precise tracking of human movements, thereby furnishing our network with additional contextual information. This contextual data is pivotal in ensuring the accurate identification of both the direction and the intensity of the force signal.

The data collected was accurately filtered to remove external noise and given as input to a Transformer architecture. The chosen network is based on GPT-2 Transformer with appropriate modification to make it suitable for time series forecasting.

In conclusion, the approach proposed in this study can predict contact forces and human motions. These estimations can, in the future, be used as a foundational tool to allow robots to have deeper insights on human intentions and as a consequence improve the interaction between humans and robots.

# Acknowledgements

I would like to express my heartfelt gratitude to the following individuals for their unwavering support and encouragement throughout my thesis journey:

To my beloved pets, for providing comfort and companionship during the long hours of research.

To my family, for their unconditional love, understanding, and continuous encouragement.

To my boyfriend, for his patience, understanding, and unwavering belief in me.

To my friends, for their support, laughter, and for being there in both the good times and the challenging ones.

To my professors, for their guidance, mentorship, and invaluable insights that have shaped my academic journey.

# Table of Contents

# List of Tables

# List of Figures

# Acronyms

**AI**
Artificial Intelligence

**ML**
Machine Learning

**NN**
Neural Network

**HMM**
Hidden Markov Models

**SC**
Shared Control

**SA**
Shared Autonomy

**ReLU**
Rectified Linear Unit

**ELU**
Exponential Linear Unit

**RNN**
Recursive Neural Network

**CNN**
Convolutional Neural Network

**LSTM**

Long Short Term Memory

**ARIMA**

Autoregressive Integrated Moving Average

**mocap**

Motion-capture

**MSE**

Mean Square error

**RMSE**

Root Mean Square error

**MAE**

Mean Absolute error

# Chapter 1

# Introduction

## 1.1 Thesis Description

As robots increasingly become part of our daily lives, the domain of Human-Robot interaction takes on greater significance, particularly in the context of ensuring safe and secure interactions with humans.

Various strategies have been devised for this purpose. Initially, the focus was on *safe co-existence*, ensuring that robots could avoid or halt their actions in human presence to prevent any potential harm. However, with the continuous advancements in robotics, a paradigm shift occurred, leading to the emergence of the *safe cooperation* approach. In this paradigm, robots actively assist humans in tasks without requiring physical contact.

An unsolved challenge in this domain is the facilitation of direct physical interaction between agents, a concept known as "safe physical interaction" [1]. This would allow human operators to guide robots in tasks while the robots estimate human intentions to adapt assistance. To achieve collaborative dynamics resembling human-human interaction, robots must tackle the intricate task of predicting human movements and forecasting force dynamics. This capability would allow robots to have an insight into human intentions so that they could pre-emptively foresee and manage risks. Robots could also estimate the skill level or fatigue of the human performing the task ahead, this information can also allow the robot to regulate the level of assistance that it should have on the task based on the need required by the human. This is a new paradigm that is known as shared autonomy [1], where a robot can autonomously regulate the level of interaction based on the estimated human intentions and environmental conditions.

Moreover, force and motion predictions can be used to estimate the human's emotions, for example, if the human performs uneven motions, it could be an indicator of fear and could prompt the robot to adopt slower movement and reduce

close interaction, while a more confident person could work at a faster pace in a more efficient manner.

In the field of human motion forecasting, a multitude of studies address this complex challenge. For instance, Ackermann et al. [2] propose a model-based approach, employing a dynamical system to predict human behavior. Nevertheless, the predominant focus in research has leaned toward model-free techniques. For example, a study applies Hidden Markov Models (HMMs) to predict motion [3]. More recently, there has been a shift towards the utilization of Neural Networks (NNs) for this purpose, Kratzer et al. [4] employ a Recurrent Neural Network to forecast short-term human dynamics.

Historically, when it comes to estimating contact forces, the field has been heavily influenced by model-based techniques, particularly those rooted in residual estimation methods [5]. These methodologies heavily rely on dynamical models to deduce forces. However, one of the key challenges with such approaches lies in their limited adaptability to different individuals. While model-free techniques for force estimation have begun to surface [6], the field of predicting forces during physical contact remains relatively uncharted territory.

The inherent complexity associated with the estimation and prediction of forces exchanged in human-robot interactions emphasized the pressing need for innovative methodologies that can not only enhance safety but also boost the efficiency of such interactions.

In this thesis, we introduce a novel framework designed to forecast both the forces exchanged between a human operator and a robot and the accompanying human movements. To achieve this, we gathered data from interactions involving a human operator wearing a haptic suit and interacting with the end effector of a robotic manipulator. During these interactions, the human operator led the end effector in a pre-determined direction. The data collected from the haptic suit formed the basis for our Neural Network dataset, while the robotic arm recorded the true force values during the interactions. These recorded data served as a means to assess the accuracy of our model.

Our model can predict contact forces solely by analyzing kinematic information from the human arm, including joint angles, velocity, and acceleration data collected using a motion capture suit. We then compared these predicted forces with the ground truth forces measured by the manipulator with which the human is in contact.

One notable advantage of our model lies in its independence from explicit force measurements for force prediction in the inference phase. In numerous applications, obtaining a direct force measurement is often impractical, primarily because the introduction of Force/Torque sensors might be unfeasible due to their susceptibility to noise and environmental disturbances.

Given the absence of input force data, our model comprises two distinct networks.

The first network is a multi-layer perceptron designed for regression tasks. It takes, as input, the kinematic data at a specific time instant and estimates the corresponding contact force for that same instant.

The second component in our model takes charge of the prediction task and employs a Transformer network [7]. These networks were originally designed for natural language processing (NLP) as they adopt the innovative technique of attention to solve the vanishing gradient problem that affects both traditional Recurrent Neural Network models and Long Short Term Memory networks (LSTM) as comprehensively examined in [8].

Transformers also have been effectively adapted to time series prediction, especially in the analysis of long-term time-series predictions, and have proven to be effective when applied to long series.

In our setup, the Transformer takes as input both the contact forces estimated by the first element and the kinematic data over a window of adjustable length. It has the capacity to generate predictions up to 800 milliseconds into the future and can identify multiple movements in various directions.

In this thesis, we present the following key contributions:

- We introduce a predictive model capable of foreseeing both human movements and the contact forces that arise during interactions between humans and robots, anticipating events in a future time frame. This predictive capability holds the potential for enhancing the performance of robots in understanding and adapting to human intentions. It is worth noting that the exploration of the controller aspect, such as a model predictive controller [9], is beyond the scope of this thesis.

- During our testing phase, our network demonstrates the ability to estimate contact forces exclusively based on kinematic data related to the human. This feature significantly enhances the versatility of our framework, especially in situations where dedicated Force/Torque sensors might not be accessible.

This thesis is structured as follows: In section 1.2, we provide an introduction to the LAAS laboratory in Toulouse, where this research was conducted. In Chapter 2, we delve into the essential theoretical foundations required for a comprehensive understanding of this thesis. Following that, Chapter 3 presents the framework employed for data collection, and offers a detailed description of the architecture and the training strategies employed. Lastly, in Chapter 4, we discuss the results obtained.

## 1.2 Research Laboratory

The National Center for Scientific Research, better known by its acronym CNRS, is the largest public French organization for scientific research. It conducts its activities in all fields of knowledge. Founded in 1939, with the aim to "coordinate the activity of laboratories in order to achieve a higher yield from scientific research,". Its scientific activities are divided among ten national institutes specialized in various fields of knowledge.

These institutes oversee approximately a thousand units or "laboratories" and labeled services, most of which are managed in collaboration with other entities such as universities, other research establishments, higher education institutions, industries, and more.

According to the Scimago Institutions Rankings, CNRS ranks third globally as a research center. Webometrics confirms this third-place global ranking and adds that it also holds the first position in Europe.

The Laboratory of Analysis and Architecture of Systems (LAAS) is one of the independent research units (UPR) of the CNRS. Established in 1968, in association with the University of Toulouse, LAAS conducts research in the fields of computer science, robotics, automation, and micro and nanosystems. Additionally, LAAS promotes interdisciplinary research through 4 strategic axes: Ambient Intelligence, Living (biology, environment, medicine), Space, and Energy.

# Chapter 2

# State of the Art

In this chapter, we introduce the theoretical concepts that underlie the research presented in this thesis.

Then in section 2.1 we delve into the theory of Shared Control and Autonomy, which serves to justify the rationale behind our work.

The subsequent section, 2.2 begins by providing an overview of Neural Networks.

In Section 2.3, we delve into the specifics of Transformers, while Section 2.4 offers a concise recap of the challenges associated with time series prediction and how Transformers have been applied to address them.

The following section, 2.5, shifts to the field of Force measurements. To be more precise, we scrutinize the Residual method, which serves as a means to establish a ground truth for our model. Following this, we provide a short overview of Impedance control, in 2.6, a strategic approach we have implemented on the selected manipulator to ensure a secure and efficient collaboration with humans.

Finally, in Section 2.7, we introduce related works that have explored the issue of Force prediction in the field of Human/Robot Interaction.

## 2.1 Shared Control

In the field of Human-Robot interaction, the level of autonomy in robots can vary significantly across different applications. To address this variability, several control architectures have been designed to allow humans to interact with partially autonomous robots.

One of these design methodologies is Shared Control (SC) [1], which aims to strike a balance between human intervention and autonomous decision-making. In SC, the human and autonomous controller inputs are combined through a predetermined function, which determines the robot's behavior and level of autonomy. However, over the last decade, significant progress in sensing, inference, modeling, and

learning methods has paved the way for an evolution beyond Shared Control. This advancement has led to the emergence of Shared Autonomy (SA) approaches, where robots exhibit the capability to adapt their autonomy levels seamlessly based on their understanding of human actions, intentions, and the surrounding environment. The key aspect of SA is its adaptability, which becomes especially desirable in dynamic and ever-changing environments and in situations where human behavior evolves over time. Unlike SC, where the robot's behavior is fixed according to the human's design, SA incorporates a higher level of intelligence that enables the robot to continuously adjust its autonomy, offering a more intuitive and human-like interaction.

The dynamics of a robotic system can be described through the equation

$$\dot{x}(t) = f(x(t), u(t)) \tag{2.1}$$

$$u(t) = h_\theta(u_h(t), u_a(t); \theta(t)) \tag{2.2}$$

where $x$ represents the robot environment state, $u$ denotes the control input, and $\theta$ models the robot's understanding of the human and/or the environment. The arbitration function $h_\theta$ plays a pivotal role in combining and modulating the autonomous control input $u_a$ and the human input $u_h$. In SC approaches this function is often designed as a linear combination of the two inputs.

$$h(u_h, u_a) = \alpha u_h + (1 - \alpha) u_a \tag{2.3}$$

where $\alpha \in [0,1]$ is a weight that distributes the control authority between the human and the controller.

In Shared Autonomy, the concept of autonomy classification becomes more nuanced. In the past, some research papers attempted to propose discrete classifications of autonomy levels for SC applications in specific domains, such as telerobotics, autonomous vehicles, and surgical robotics. However, these classifications were domain-specific and often challenging to generalize.

In contrast, for SA approaches, a continuous spectrum of autonomy levels appears to be more appropriate, as it accommodates the robot's ability to continuously adjust its autonomy level based on real-time signals and feedback from the human, the task at hand, and the surrounding environment. This adaptability ensures that the robot can effectively respond to ever-changing conditions and deliver optimal performance.

Originally, Shared Control was introduced primarily as a control architecture for remotely operated robots. The incorporation of some degree of autonomy in the robot was necessary to overcome communication delays between local and remote sites. Traditionally, architectures corresponding to different human interaction modalities were categorized into three classes:

1. direct control, where the robot's degrees-of-freedom were controlled directly by the user through local interfaces;

2. supervisory control, where user commands and feedback occurred at a higher level, with the robot relying on stronger local autonomy to refine and execute tasks;

3. shared control, which encompassed all the intermediate levels where the robot was controlled through a combination of direct user commands and autonomy.

While SC was an important step in enhancing human-robot interactions, it still lacked the finesse of mutual adaptation and seamless cooperation found in human-human interactions.

SA architecture revolutionized the landscape of human-robot interactions by enabling a level of interaction much closer to human-human interaction. In a SA system, the robot can infer, in a probabilistic sense, the action that the human is performing. Based on this information, the robot can compute the appropriate action it must undertake, regulating its autonomy level and providing assistance as needed. This adaptability is crucial in achieving a human-like interaction experience and ensuring a harmonious collaboration between humans and robots.

To enhance the interaction experience further, this project will leverage machine-learning techniques, such as transformers, to predict the expected interaction force. By analyzing information from past interactions, the system can infer the current force and predict it within a time horizon. This anticipatory capability is reminiscent of how two humans collaborate during tasks like object transportation. Humans instinctively anticipate each other's future behavior, adapt accordingly, and work harmoniously. Similarly, robots equipped with SA capabilities should estimate humans' future intentions and actions to offer intuitive and natural assistance.

## 2.2 Neural Networks

In the field of machine learning, Neural Networks have emerged as a key technology, fundamentally altering the way we approach complex computational problems. These mathematical constructs, inspired by the biological Neural Networks of the human brain, have revolutionized various domains, ranging from image analysis and natural language understanding to robotics and autonomous systems [10].

Neural Networks are composed of interconnected nodes, also known as artificial neurons, organized into layers. Each connection between nodes is associated with a weight, which is adjusted during training to optimize the network's performance.

The output produced by each neuron is obtained as the result of a non-linear function named activation function. The choice of activation function is a key problem in the architecture of a Neural Network, some of the most common choices can be:

- Rectified Linear Unit (ReLU): $f(z) = max(0, z)$

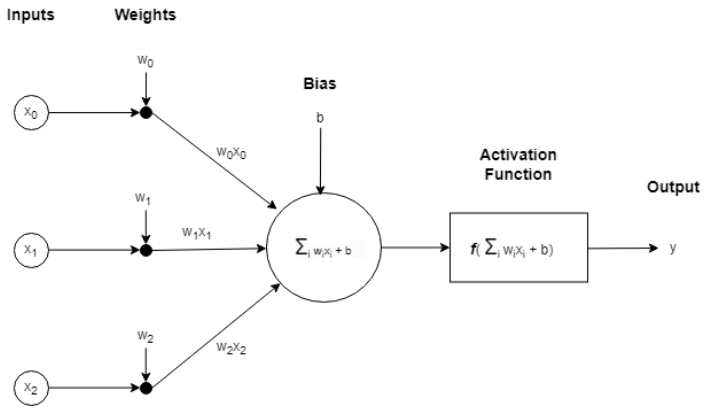



**Figure 2.1:** Neuron structure

- Leaky ReLU: $f(z) = \begin{cases} z & \text{if } z > 0 \\ \alpha z & \text{otherwise} \end{cases}$
- Exponential Linear Unit (ELU): $f(z) = \begin{cases} z, & \text{if } z > 0 \\ \alpha(e^z - 1), & \text{otherwise} \end{cases}$

### 2.2.1 Layer classifications

In a Neural Network, layers are fundamental building blocks that organize and process data as it flows through the network. Each layer is composed of a collection of artificial neurons and plays a specific role in transforming input data into meaningful output. We can classify the layers in a Neural Network based on their position in:

1. **The Input Layer**, it is the first layer of the Neural Network, and its primary function is to receive the raw input data.
2. **Hidden Layers**, which are intermediary layers between the input and output layers. These layers are responsible for extracting and learning relevant patterns and features from the input data. Each neuron in a hidden layer takes input from all neurons in the previous layer and applies a weighted sum of these inputs, followed by an activation function, to produce an output. Hidden layers enable the Neural Network to capture complex relationships and representations in the data.
3. **Output Layer**, this is the final layer of the Neural Network, responsible for producing the network's predictions or outputs. The number of neurons in the output layer depends on the specific task the Neural Network is designed for.

However, multiple naming systems can be employed to categorize the layers within Neural Networks. For instance, they can be categorized according to the activation function executed by the neurons or according to the connections established between the neurons within the current layer and those in other layers. Some common examples include:

1. **Fully Connected Layers** where each neuron in the layer is connected to every neuron in the next layer. They can become computationally expensive as their input grows. As such, they are commonly used for specific purposes within Neural Networks such as classifying image data.

2. **Convolutional layer** applies a convolution operation to the input, passing the result to the next layer. They are very used in the processing of images because they convert all the pixels in their receptive field into a single value.

3. **Recurrent Layers** where both input and prior states are fed into the neurons in order to create a sort of memory, they are used in recurrent and LSTM Neural Networks.

The transformation implemented by each layer is parameterized, based on the weights and bias of each neuron, the number of parameters is also known as the number of features of layer l. The value of these parameters is assigned during training to achieve optimal performance.

### 2.2.2 Neural Network Fine-tuning

The weights and biases of a Neural Network are continuously updated until the minimum error is reached. The error, in a supervised problem, is defined as the mismatch between the output produced by the Neural Network and its actual value. To quantify and systematically minimize this error during the training process, a mathematical function known as the loss function is employed. The loss function measures the extent of this error and guides the optimization algorithms to adjust the Neural Network's parameters for improved performance.

In the initial stage of training, each weight and bias is initialized to a random value near zero. To enhance the network's performance, a widely adopted approach is the utilization of the backpropagation algorithms. These optimization algorithms involve the systematic propagation of errors from the output layer backward through the network, taking into careful consideration the chain rule. Consequently, weight adjustments are made based on the gradient of the loss function concerning each of its parameters. This methodology, named gradient descent, is employed iteratively to determine the optimal parameters that lead to a local minimum within the loss function. The rate of parameter updates during the gradient descent process is finely controlled by a critical hyper-parameter termed the "learning rate."

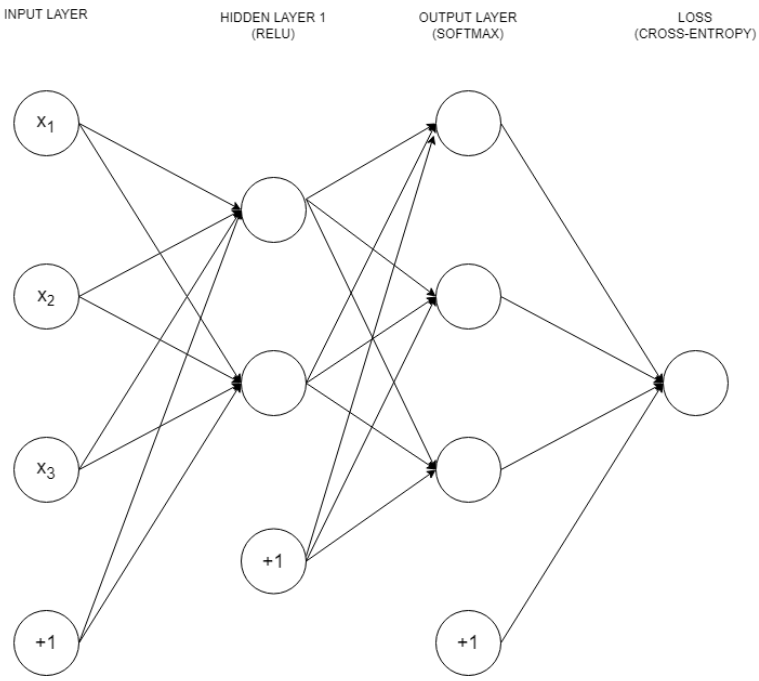



**Figure 2.2:** Neural Network structure

The learning rate plays a crucial role in the convergence and stability of the optimization process. If the learning rate is too small, the model's parameter updates will be tiny, and the optimization process will be slow, requiring many iterations to converge to a good solution. On the other hand, if the learning rate is too large, the parameter updates may overshoot the optimal values, leading to instability and failure to converge.

One particularly widespread optimization algorithm is known as Adam, based on backpropagation, short for Adaptive Moment Optimization. Adam employs the following equations for each parameter $\omega_i$ to minimize error rates.

$$v_t = \beta_1 v_{t-1} - (1 - \beta_1) g_t \tag{2.4}$$

$$s_t = \beta_2 s_{t-1} - (1 - \beta_1) g_t^2 \tag{2.5}$$

$$\Delta\omega_t = -\eta \frac{v_t}{\sqrt{s_t + \epsilon}} g_t \tag{2.6}$$

$$\omega_{t+1} = \omega_t + \Delta\omega_t \tag{2.7}$$

with $\eta$ initial learning rate, $g_t$ gradient of the loss function at time t with respect to $\omega_i$, $v_t$ exponential average of gradients along $\omega_i$, $s_t$ exponential average of squares of gradients along $\omega_i$ $\beta_1$ and $\beta_2$ are the adjustable hyper-parameters.

The data sets also play a crucial role in ensuring the success of a Neural

Network. Since the weight update is determined by the derivative of the activation function, there is a possibility of an asymmetrical update process, particularly in heterogeneous data sets like the one we used in our project. This can result in certain weights being updated at a faster rate than others, to mitigate this phenomenon, we apply a process of input data normalization to standardize features. Furthermore, a data set that is not adequate for the problem under analysis can cause ulterior problems such as underfitting or overfitting.

Underfitting, also known as "type I error," occurs when a model cannot effectively capture the patterns in the training data. It often results from overly simple models, such as those with too few layers or neurons. A typical sign of underfitting is poor performance on both the training and testing data, indicated by high bias in the model's output.

Overfitting is a more common problem of Neural Networks and it occurs when a model learns the intricacies and noise in the training data so precisely that it adversely affects its performance on new, unseen data. This happens because the model mistakenly treats the noise in the training data as meaningful patterns or concepts. The challenge here is that these learned concepts may not be applicable to new data, hampering the model's ability to generalize effectively.

Overfitting often arises in scenarios with a small data set that contains a high level of noise. To mitigate overfitting, potential solutions include simplifying the model's complexity or, where feasible, increasing the size of the data set. One common indicator of overfitting is a high level of variance during the testing phase.

An alternative approach to counteract overfitting is to employ regularization algorithms, which are effective tools for reducing the model's tendency to fit noise in the training data and promote better generalization.

These algorithms introduce additional constraints to the learning process, discouraging the model from becoming too complex and fitting the noise in the training data. Among the most common techniques, we can cite:

- L1 Regularization (Lasso): it adds a penalty term proportional to the absolute value of the weights to the loss function. It encourages the model to shrink less important features' coefficients to zero, effectively performing feature selection.

- L2 Regularization (Ridge Regression): it adds a penalty term proportional to the square of the weights to the loss function. It forces the model to spread the influence of each feature across all output units, making the model more stable and less sensitive to individual data points.

- Dropout: this technique was specifically designed for Neural Networks. During training, a subset of neurons in a layer are randomly deactivated (set to zero). This encourages the network to learn redundant representations and reduces co-adaptation among neurons. At test time, all neurons are used, but their

weights are scaled by the dropout probability to ensure that the expected output remains the same.

- Batch Normalization: it normalizes the output of each neuron in a layer to have zero mean and unit variance over a mini-batch of training samples. This helps in avoiding the vanishing/exploding gradient problem during training. Batch normalization acts as a form of regularization by reducing internal co-variate shifts, and it can improve the training speed and generalization performance.

- Data Augmentation: it is a technique used to artificially increase the size of the training data set by applying random transformations to the original data. For example, in image data, this could involve random rotations, translations, flips, or brightness adjustments. Data augmentation helps the model generalize better by exposing it to different variations of the same data.

- Early Stopping: it is a regularization technique that monitors the model's performance on a validation data set during training. When the validation performance stops improving, training is halted before overfitting occurs. This prevents the model from overtraining on the training data and helps it generalize better to unseen data.

These regularization techniques can be used individually or in combination to prevent overfitting and improve the performance of Neural Networks.

In order to accurately test the data set cross-validation is the most common approach. This is a statistical method used to estimate the performance of machine learning (ML) models where the data set is randomly divided into two categories: the training and the test set. The model is trained using a training set. At the end of this phase, the data that was removed can be used to test the performance of the learned model on new data. The technique just described can present slight variation in its application, hence the techniques applied can be classified as follows:

- The holdout cross-validation consists of dividing the data set into two subsets: training set and test set sometimes called validation set.

- K-fold cross-validation where the data set is divided into "k" subsets, and the holdout method is, then, repeated k times. During each test, one of the k subsets is used as the test set while the remaining k-1 subsets are combined to form a training set. In the end, the average error across all k trials is computed. The advantage of this method is that the result is not affected by the way training and test sets are chosen. However, the downside is that the training algorithm has to be rerun from scratch k times.

- Leave-one-out cross-validation is similar to the method previously analyzed, but it is taken to its logical extremes. In this case, the number of subsets chosen equals the number of elements in the data set minus one. That means that the function is trained on all the data except for one point and a prediction is made for that point.

Neural Networks encompass a wide array of structures, each tailored to specific tasks and challenges. These networks exhibit different architectures, catering to a wide spectrum of applications. Some networks, like Feedforward Neural Networks (FNNs), are designed for straightforward tasks such as pattern recognition. Recurrent Neural Networks (RNNs) excel in sequential data analysis, making them ideal for tasks like natural language processing. Convolutional Neural Networks (CNNs), on the other hand, are optimized for image and spatial data. Lastly, transformer-based models, like BERT and GPT, have revolutionized language understanding and generation tasks. The diversity of Neural Network architectures allows us to leverage the right tool for the right job, unlocking the potential of artificial intelligence (AI) across various domains. In the next section, we will delve deeper into Transformer Neural Networks, a critical component of our project.

## 2.3 Transformer Neural Network

Transformers represent a category of Neural Networks first introduced in 2017 through the paper "Attention is All You Need" by Vaswani et al. [7]. These networks have experienced a significant rise in popularity, notably gaining prominence with the launch of ChatGPT by OpenAI.

These models were originally designed to address the challenges of sequence transduction, including tasks like neural machine translation. Consequently, their most common applications revolve around language models, especially in scenarios involving lengthy sequences.

Before the advent of transformers, Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM) networks, and Convolutional Neural Networks (CNNs) were the predominant choices. However, they exhibited limitations, particularly when dealing with lengthy sequences.

Recurrent Neural Networks (RNNs) can retain past information by maintaining a hidden state that captures and stores information from previous time steps, allowing it to influence future predictions. This feature empowers the network to not only process new data but also retain and utilize context from the past. While RNNs excel at handling shorter sequences, their efficiency diminishes as the time gap between relevant information and its application grows larger. The reason behind this inefficiency lies in the sequential nature of information flow. As sequences get longer, there is a heightened risk of losing vital information along

the way due to the sequential processing.

LSTM Networks are a type of recurrent Neural Networks designed to address some of the challenges associated with modeling long sequences. They introduce modifications to the way information is processed within the network, primarily through multiplicative and additive operations. LSTMs incorporate a mechanism known as cell states to facilitate selective retention and forgetting of information, distinguishing between what is important and less relevant. Nevertheless, due to the sequential nature of input data processing, the network faces an increasing risk of processing delays or potentially losing critical information as the input sequence lengthens.

To mitigate these challenges, RNNs, including LSTMs, have adopted attention mechanisms. Instead of encoding the entire sentence into a single hidden state, each word in the sequence has its corresponding hidden state, which is retained throughout the decoding stage.

However, the sequential nature of processing remains a limitation. To address this, CNNs have proven to be useful. These networks can process elements in parallel and are well-suited for capturing both short-range and long-range dependencies within sequences, making them a valuable tool for handling long input sequences efficiently. Moreover, the "distance" between the output word and any input word for a CNN is approximately logarithmic in nature (log(N)), which represents a substantial improvement compared to the distance between the output of a Recurrent Neural Network and an input, which typically scales linearly with the input sequence length (N). This attribute of CNNs contributes to their efficiency in handling sequential data and makes them particularly suitable for tasks like machine translation.

The challenge with Convolutional Neural Networks lies in their limitation in effectively capturing dependencies when translating sentences. This limitation prompted the development of Transformers, which integrate the capabilities of CNNs with attention mechanisms.

Transformers tackle this challenge by employing a combination of encoders and decoders enhanced with attention mechanisms.

Attention mechanisms play a crucial role in significantly accelerating the model's ability to perform efficient sequence translation. The key to this accelerated processing lies in the utilization of self-attention.

Self-attention, also known as scaled dot-product attention, is a mechanism used to weigh and capture the relationships between elements in a sequence, such as words in a sentence or tokens in a document. It is a key component for modeling contextual information and dependencies within sequences. The main steps of self-attention can be summarized as follows:

1. Each input vector is multiplied by three weight matrices, that are defined during the training phase, often referred to as Query, Key and Value.

2. Self-attention is responsible for computing similarity scores, often referred to as attention scores, between each query vector and all key vectors within the sequence. This computation is typically achieved by taking the dot product of the query vector and each key vector. These scores essentially indicate the degree of attention that each element should allocate to the others.

3. The scores are then divided by the square root of the dimension of the key vector, in order to normalize the output.

4. A softmax function is applied to the self-attention scores.

5. The value vector is multiplied by the score computed in the last step.

6. The weighted value vectors obtained are summed to produce the self-attention value for a given element of the input.

The structure of the Transformer is composed of two main blocks: encoders and decoders as we can see in Fig 2.3. The encoder processes input sequences. It uses an embedding layer to convert elements into vectors and employs self-attention to capture relationships between elements. The elements are then sent to a feedforward layer that applies a linear transformation and a non-linear activation function to each position independently, further enhancing the representation of each token. Multiple layers of the encoder stack allow the model to understand complex patterns in the sequence, producing rich contextual representations. These representations are then used for various tasks, including sequence translation and understanding. The Transformer decoder is responsible for generating sequences of output based on input information from the encoder. It has a similar structure to the encoder but in addition, it contains a self-attention mechanism that focuses only on relevant parts of the input, known as masked self-attention. It uses multiple stacked layers for complex pattern capture and incorporates positional encodings for word order awareness. The decoder generates output sequences step by step, producing probabilities for each output element, typically using autoregressive generation.

A fundamental characteristic of the Transformer is that each element traverses its dedicated path within the encoder. While dependencies exist between these paths in the self-attention layer, the feed-forward layer manages each element independently, allowing multiple paths to be executed in parallel as they pass through it.

The resulting output from self-attention is used to enrich the representations of the input elements, incorporating context and relationships between them. Self-attention has proven highly effective in various natural language processing tasks, enabling models like Transformers to understand complex dependencies within sequences and excel in tasks such as translation and text summarization.

However, certain challenges may surface when Transformers are applied to time series prediction, as we will explore in the following section.

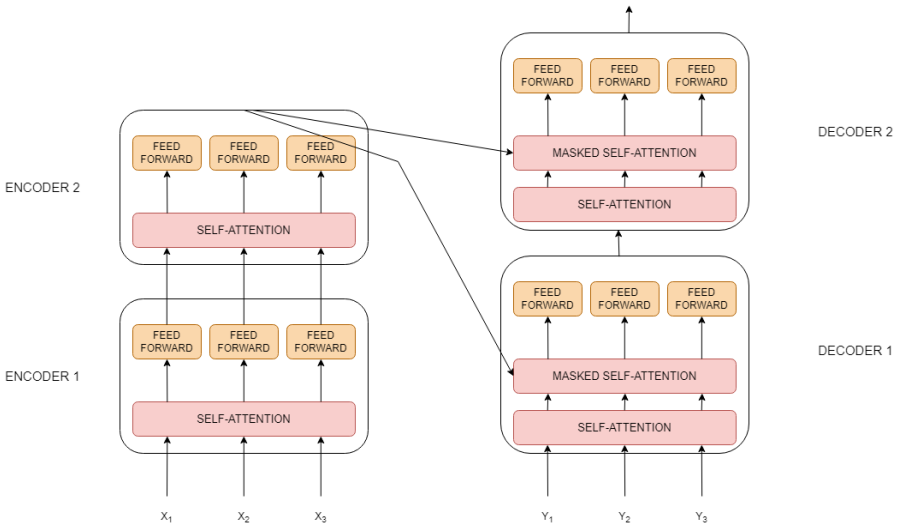



**Figure 2.3:** Transformer structure

## 2.4 Time Series Prediction

Time series forecasting is a methodology employed to predict future events by analyzing historical trends, operating under the premise that future patterns will resemble those observed in the past. Traditionally, statistical techniques like autoregressive models and exponential smoothing methods have dominated the field. However, in recent times, the adoption of Machine Learning models has surged in popularity, often outperforming traditional methods. These approaches, encompassing decision trees, random forests, and Neural Networks, have proven to be highly effective in capturing intricate patterns and unraveling non-linear relationships within time series data. Furthermore, deep learning architectures such as RNNs and LSTM networks have showcased their adeptness in capturing sequential dependencies and deciphering long-term trends present in time series data [11].

In this project, we have utilized time series analysis through Machine Learning. However, before delving into the aspects related to Machine Learning, it is crucial to gain a thorough understanding of the inherent challenges associated with time series analysis.

One of the primary obstacles is dealing with temporal dependencies and patterns, which can be intricate and require sophisticated modeling techniques. Handling

irregularities and outliers in the data is crucial to ensure accurate predictions, the choice of appropriate features, and the determination of suitable time windows for analysis demand careful consideration.

Several standard metrics are commonly used to evaluate the performance of the prediction model depending on the specific goals and characteristics of the forecasting problem. The Mean Absolute Error (MAE) is often favored when outliers or extreme values in the data need to be treated with less sensitivity. Mean Square Error (MSE) and Root Mean Square Error (RMSE), on the other hand, are sensitive to outliers and can penalize large errors more heavily, making them suitable for scenarios where extreme errors are of concern.

An additional significant challenge pertains to the choice of the method employed for forecasting time series data. Statistical methods require a priori knowledge about the data distribution to build a predictive model. We can divide statistical methods in two groups according to their mathematical complexity: exponential smoothing models and AutoRegressive Integrated Moving Average (ARIMA) models.

In exponential smoothing models, the time series are decomposed into components, their values are smoothed by weights over time. ARIMA models are usually of three types: autoregression based on establishing the correlation between observations, integration based on the differences required to guarantee stationarity and moving averages.

The moving average is a widely used statistical method that relies on calculating the arithmetic average of the most recent 'r' values to predict the next value. The choice of 'r' determines the level of smoothing in the prediction, with larger values of 'r' resulting in a more uniform prediction.

Machine Learning prediction methods do not require prior knowledge of the distribution, these methods show reliable information even in complex and nonlinear series. They can be divided into two approaches: global and local. In global approaches, all observations of the training series are considered, global approach's main limitation is to assume all pairs of inputs and outputs to be independent and identically distributed. In local approaches, Machine Learning algorithms have been adapted to include temporal information. Such methods partition the time series into subsequences based on the closeness and importance of the current prediction.

In recent years the Transformers, originally designed for Natural Language Processing, have undergone subtle architectural modifications to make them suitable for time series forecasting. There is no unanimous agreement regarding the application of traditional Transformers for time series forecasting. As explained in [12], this architecture may encounter challenges when it comes to capturing intricate temporal relationships within time series data.

The self-attention mechanism that makes the network so accurate in natural language processing causes, to some extent, a loss of temporal information, this

is not a concern in NLP as the semantic meaning is usually preserved even if the words are reordered.

Furthermore, the decoder design can contribute to poor performance of transformers because the high level of complexity required to perform long-term time series forecasting tends to generate an error accumulation and cause the divergence of the predicted with reference to the expected result.

An interesting modification has been presented in the paper [13] where a sparse self-attention mechanism is proposed to solve the complexity problem of the Transformer, and the encoder is modified so that it trims the input's time dimension, the decoder structure remains unchanged but in the self-attention computation we apply masked multi-head attention. This method has proved to outperform similar networks in univariate and multivariate series prediction.

An alternative approach is presented in the paper [14] which served as the basis for our project. This paper introduces a Transformer Neural Network designed for predicting humidity levels. In the study, a comparison is drawn between the predictions generated by the Transformer and those produced by an LSTM network trained on the same data set. The results indicate that the Transformer excels in preserving historical trends, outperforming the LSTM network. In this paper, the problem of losing temporal information is solved by encoding a timestamp as additional information for each input token, in doing so, the self-attention block will have the context of the relative distance between a given timestamp and the current one.

## 2.5 Residual method

In this thesis, we compared the force predictions generated by our network with the forces estimated by the manipulator. The estimation of actual forces was conducted using a residual-based approach, as outlined in the work by Magrini et al. [15].

This method, based on the framework introduced by De Luca et al. [5], defines a vector of residuals. This residual vector represents a filtered version of the vector of equivalent joint torques resulting from the external forces applied to the robot. Once the residuals are estimated, calculating the applied forces becomes a straightforward process.

The dynamic model of a robot is typically formulated using the Euler-Lagrange formalism, which is a well-established framework in robotics and mechanics for describing the dynamics of a mechanical system [16].

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q}) = \boldsymbol{\tau} + \boldsymbol{\tau}_{\text{ext}} \tag{2.8}$$

Where $\mathbf{M}(\mathbf{q}) > 0$ is the symmetric inertia matrix, $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ collects the Coriolis and centrifugal contributions, $\mathbf{g}(\mathbf{q})$ is the gravitational vector, $\boldsymbol{\tau}$ is the vector of

commanded joint torques and $\boldsymbol{\tau}_{ext}$ is the vector of equivalent joint torques due to the applied external force $\mathbf{F}_{ext}$:

$$\boldsymbol{\tau}_{ext} = \mathbf{J}_C^T \mathbf{F}_{ext} \tag{2.9}$$

Where $\mathbf{J}_C$ is the geometrical Jacobian of the manipulator at the contact point. The matrix $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ is defined with the following property:

$$\mathbf{x}^T(\dot{\mathbf{M}}(\mathbf{q}) - 2\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}))\mathbf{x} = 0 \quad \forall \mathbf{x} \in \mathbb{R}^N \tag{2.10}$$

The residual vector is defined as follows

$$\mathbf{r}(\mathbf{t}) = \mathbf{G}_I \left[ \mathbf{p}(\mathbf{t}) - \int_0^t (\boldsymbol{\tau} + \mathbf{C}^T(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} - \mathbf{g}(\mathbf{q}) + \mathbf{r}))ds \right] \tag{2.11}$$

Where $\mathbf{G}_I$>0 is a gain diagonal matrix and $\mathbf{p}(t) = \dot{\mathbf{M}}(\mathbf{q})\ddot{\mathbf{q}}$ is the generalized momentum of the manipulator. We can notice now that, making the derivative with respect of time of the generalized momentum, we obtain the following expression:

$$\dot{\mathbf{p}} = \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \dot{\mathbf{M}}(\mathbf{q})\dot{\mathbf{q}} = \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{C}^T(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} \tag{2.12}$$

It follows that

$$\dot{\mathbf{p}} = \boldsymbol{\tau} - \boldsymbol{\tau}_{ext} - \mathbf{g}(\mathbf{q}) + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} \tag{2.13}$$

Assuming the robot is at rest at time t=0 and $\mathbf{r}(0)$=0, namely no external force is applied at the beginning, the evolution of the residual $\mathbf{r}$ is

$$\dot{\mathbf{r}} = \mathbf{G}_I(\boldsymbol{\tau}_{ext} - \mathbf{r}) \tag{2.14}$$

and for every component of the residual we can write a transfer function:

$$\frac{r_i(s)}{\boldsymbol{\tau}_{ext}(s)} = \frac{g_{ii}}{s + g_{ii}} \tag{2.15}$$

Where $g_{ii}$ is the $i_{th}$ element of the diagonal of the matrix $G_I$. For sufficiently high gain the dynamics of every component of the residual will reproduce the dynamics of the external torque, therefore $\mathbf{r} \approx \boldsymbol{\tau}_{ext}$. Once the residual vector is estimated, then the vector of corresponding external forces can be found as:

$$\mathbf{F_{ext}} = (\mathbf{J}_C^T))^{-1}\boldsymbol{\tau}_{ext} \approx (\mathbf{J}_C^T))^{-1}\mathbf{r} \tag{2.16}$$

In our case, since we assume that the contact is always at the robot end-effector, $\mathbf{J}(\mathbf{q})$ is the geometric Jacobian of the robot $\mathbf{J}_C(\mathbf{q})$.

## 2.6 Impedance Control

Impedance control is a widely used control strategy in robotics that focuses on regulating the interaction between a robot and its environment. The term "impedance" comes from the field of electrical engineering, where it represents the opposition that a circuit offers to the flow of alternating current. In robotics, impedance control serves a similar purpose, but instead of current, it deals with forces and motions.

In the context of collaborative robotics, impedance control enables robots to be more sensitive to human interactions, making them safer and more efficient when working alongside humans. By modulating the robot's stiffness, it can offer appropriate resistance to external forces, preventing accidents and damage while maintaining smooth and coordinated movements.

The impedance control calculates the control force required to maintain the gripper's position in alignment with the desired position. This solution proves effective, particularly when the contact forces are relatively small, and high accuracy is not a critical requirement. The robot's performance is generalized as a spring-damper system in each direction of the Cartesian space.

This process begins with the development of the dynamic model of a robot in contact, leveraging the linearized Lagrangian equations expressed in the cartesian space:

$$\mathbf{M}_x(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}_x(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{g}_x(\mathbf{q}) = \mathbf{J}_a^T(\mathbf{q})\mathbf{u} + \mathbf{F}_a \tag{2.17}$$

In the equation $\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}$ represent the angular position, angular velocity, and angular acceleration of each joint in joint space, respectively. The term $M_x$ corresponds to the Cartesian Mass matrix, while $\mathbf{C}_x$ stands for the Cartesian and Coriolis matrix in the cartesian space, $\mathbf{g}(\mathbf{q})$ is the gravitational vector and u represents the generalized control forces produced by the actuators, $\mathbf{J}_a$ is the analytical jacobian. Additionally, $\mathbf{F}_a$ signifies the interaction forces exerted on the robot by its environment in the cartesian space.

The implementation of the impedance controller encompasses two pivotal steps: feedback linearization control and the imposition of a dynamic impedance model. This process begins with the development of the dynamic model of a robot in contact, leveraging the linearized Lagrangian equations expressed in Cartesian coordinates:

Let's delve into these steps and their significance:

- In the first step we choose accordingly the value of u so that the cartesian gravity vector, Coriolis matrix, and the measured forces are canceled, also a new signal is added the acceleration $\mathbf{a}$. The control input appears as follows

$$\mathbf{u} = \mathbf{J}^T(\mathbf{q})[\mathbf{M}_\mathbf{x}(\mathbf{q})\mathbf{a} + \mathbf{S}_\mathbf{x}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{x}} + \mathbf{g}_x(\mathbf{q}) - \mathbf{F}_a] \tag{2.18}$$

- In the second step in order to impose the desired dynamic behavior, we consider the equation

$$\mathbf{M}_m(\ddot{\mathbf{x}} - \ddot{\mathbf{x}}_\mathbf{d}) + \mathbf{D}_m(\dot{\mathbf{x}} - \dot{\mathbf{x}}_d) + \mathbf{K}_m(\mathbf{x} - \mathbf{x}_d) = \mathbf{F}_a \tag{2.19}$$

  where $\mathbf{K}_m$ and $\mathbf{D}_m$ denote the stiffness and damping of the closed-loop system, respectively while $\mathbf{M}_m$ is the desired apparent inertia. The signal $\mathbf{a}$ is chosen as

$$\mathbf{a} = \ddot{\mathbf{x}}_d + \mathbf{M}_m^{-1}[\mathbf{D}_m(\dot{\mathbf{x}} - \dot{\mathbf{x}}_d) + \mathbf{K}_m(\mathbf{x} - \mathbf{x}_d) + \mathbf{F}_a] \tag{2.20}$$

By substituting the equations into each other, we can derive the control law of an impedance control:

$$\begin{aligned}\mathbf{u} =& \mathbf{M}(\mathbf{q})\mathbf{J}^{-1}(\mathbf{q})\{\ddot{\mathbf{x}}_d - \dot{\mathbf{J}}_a(\mathbf{q})\dot{\mathbf{q}} \\ &+ \mathbf{M}_m^{-1}[\mathbf{D}_m(\dot{\mathbf{x}}_d - \dot{\mathbf{x}}) + \mathbf{K}_m(\mathbf{x}_d - \mathbf{x})]\} \\ &+ \mathbf{S}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q}) + \mathbf{J}_a^T(\mathbf{q})[\mathbf{M}_x(\mathbf{q})\mathbf{M}_m^{-1} - \mathbf{I}]\mathbf{F}_a\end{aligned} \tag{2.21}$$

where $\mathbf{M}$ is the inertia matrix in the joint space. A simplification that can be applied is to set the apparent inertia equal to the natural Cartesian inertia. However, to accurately represent a real mechanical system, it is necessary to include Coriolis and centrifugal terms in the dynamical model:

$$\mathbf{M}(\mathbf{q})(\ddot{\mathbf{x}} - \ddot{\mathbf{x}}_d) + (\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{D}_m)(\dot{\mathbf{x}} - \dot{\mathbf{x}_d}) + \mathbf{K}_m(\mathbf{x} - \mathbf{x}_d) = \mathbf{F}_a \tag{2.22}$$

Notably, this option eliminates the need for a force/torque sensor, making the control process a pure motion control. This approach proves effective as long as the contact forces at the end effector remain limited. So the control law appears:

$$\begin{aligned}\mathbf{u} = \mathbf{M}(\mathbf{q})\mathbf{J}_a^{-1}(\mathbf{q})\{\ddot{\mathbf{x}}_d - \dot{\mathbf{J}}_a(\mathbf{q})\mathbf{J}_a^{-1}(\mathbf{q})\dot{\mathbf{x}}_d\} \\ +\mathbf{S}(\mathbf{q}, \dot{\mathbf{q}})\mathbf{J}_a^{-1}(\mathbf{q})\dot{\mathbf{x}}_d + \mathbf{g}(\mathbf{q}) \\ +\mathbf{J}_a^T[\mathbf{D}_m(\dot{\mathbf{x}}_d - \dot{\mathbf{x}}) + \mathbf{K}_m(\mathbf{x}_d - \mathbf{x})]\end{aligned} \tag{2.23}$$

This model ensures the asymptotic convergence of errors to zero when contact forces are equal to zero. Moreover, when the desired position is constant, and the desired velocities and accelerations are zero, we obtain the following control law:

$$\mathbf{u} = \mathbf{g}(\mathbf{q}) + \mathbf{J}_a^T(\mathbf{q})[\mathbf{K}_m(\mathbf{x}_d - \mathbf{x}) - \mathbf{D}_m\dot{\mathbf{x}}] \tag{2.24}$$

This form of control represents a Cartesian PD control with gravity cancellation. To analyze the convergence of this system when contact forces are nonzero, we

can assume $\mathbf{F}_a = \mathbf{K}_e(\mathbf{x}_e - \mathbf{x})$, where $\mathbf{K}_e$ is the contact stiffness and $\mathbf{x}_e$ is the rest position. The control law exhibits a unique solution under these conditions:

$$\mathbf{x} = (\mathbf{K}_m + \mathbf{K}_e)_{-1}(\mathbf{K}_m\mathbf{x}_d + \mathbf{K}_e\mathbf{x}_e) = \mathbf{x}_E \tag{2.25}$$

where $\mathbf{x}_E$ is the position of the equilibrium point.

In conclusion, the implementation of impedance control and the various control laws discussed above play a crucial role in facilitating safe and efficient collaboration between humans and robots in diverse applications and scenarios. By carefully considering the dynamics of the system and imposing desired behaviors, we can optimize the performance and responsiveness of robots while ensuring their adaptability to varying force interactions with their human counterparts [17].

## 2.7 Related works

The field of research concerning the prediction of safe human-robot physical interactions is extensive. Within this domain, studies can be broadly categorized into three main groups: (*i*) studies concentrating only on motion prediction; (*ii*) research focusing on the prediction of interaction forces; (*iii*) hybrid approaches that forecast motions and forces.

A substantial portion of research belongs to the first group, with a primary focus on deriving models for human motion. For instance, in the work by Luber et al. [18], a model is proposed based on social forces and environmental constraints. Antonucci et al. [19] present a deep Neural Network rooted in the social force model, while Zhang et al. [20] introduce a physics-based network. Liu et al. [3] employ Hidden Markov Models (HMM) for motion sequence modeling, and Lasota et al. [21] propose a multiple predictor for human motion, learned directly from the task being performed.

Building upon human motion models, some works delve into control strategies. For instance, Balan et al. [22] propose a collision-avoidance algorithm based on human intention, while Huo et al. [23] introduce a variable impedance controller that models human intention through an adaptive Neural Network. Kim et al. [24] present a technique for real-time estimation of joint torque for humans, treating them as humanoid robots.

Finally, several approaches in the field explore shared autonomy, which adjusts the level of autonomy in a robot based on estimated human actions and intentions. Nikolaidis et al. [25] propose an adaptive human-robot collaboration scheme, and Li et al. [26] formulate a game-theoretical approach integrated within an impedance controller. Milliken et al. [27] employ a partially observable Markov decision process to determine the expertise level of a human operator and provide assistance accordingly. Peternel et al. [28] use wearable electromyography sensors to measure

human fatigue and employ a variable impedance controller to minimize human effort.

A recent survey by Selvaggio [1] provides an overview of shared autonomy. Our work builds upon these existing studies by extending the estimation of human motion to include the prediction of contact forces.

It's noteworthy that most existing literature focuses on estimating current contact forces, whereas our model distinguishes itself by predicting future contact forces

An observer combining environmental forces and robot velocities is presented in Hacksel et al.'s work [29].

In Ko et al.'s study [6], a vision-based deep-learning method is proposed for estimating interaction forces during grasping tasks. Su et al. [30] introduce a deep learning-based algorithm to map electromyography sensor data to one-step-ahead force magnitude exchanged with the environment. An exception to contact force prediction is presented in Noohi et al.'s work [31], where a constant force model is utilized for dyadic cooperative object manipulation tasks.

Recent research efforts aim to jointly predict human movements and contact forces, enhancing prediction accuracy in human-robot interaction scenarios. For example, Maithani et al. [32] use a Recurrent Neural Network with Long Short-Term Memory units to predict human position, velocity, and force. These predictions are used to estimate parameters for an impedance controller. Yu et al. [33] propose a Bayesian-based method to estimate human stiffness and motion intention, which is combined with an impedance controller utilizing a Neural Network to compensate for uncertainties in robotic dynamics. However, neither of these approaches considers the prediction of future human behavior.

Lastly, Ghadirzadeh et al. [34] employ a reinforcement learning (RL) approach, where the human is not explicitly modeled but is treated as an environmental uncertainty in the RL problem formulation. In contrast, our framework explicitly models human motion.

# Chapter 3

# Method

Our system consists of two primary components: a human and a robot, as depicted in Figure 3.1. The human participant is equipped with a motion capture suit that enables the recording and streaming of kinematic data, including position, velocity, and acceleration, for each joint and link of the human body.

The robotic manipulator is controlled through a Cartesian impedance controller, as detailed in section 2.6. This controller ensures a safe interaction between the human and the robot as the system is reduced to a mass-spring-damper system and the robot is instructed to maintain its initial configuration for the duration of the experiment. On the manipulator, a residual-based technique is also implemented for estimating contact forces at its end-effector.

## 3.1 Framework setup

We collected data from an interaction involving a human operator equipped with a motion capture (mocap) suit interacting with the end effector of a robotic manipulator. The operator guided the end effector towards one of eight planar directions, as depicted in Figure 3.1. The mocap suit was utilized to capture kinematic data pertaining to the motion of the human arm, while the manipulator recorded the force applied by the human on the end effector.

In this section, we provide a brief overview of the two primary data sources used to train our Transformer-based network: (i) the mocap suit data and (ii) the contact forces from the manipulator

### 3.1.1 Xsens suit

We employed the Xsens MVN motion capture suit [35] to build our data set, this is a portable full-body inertial measurement system.

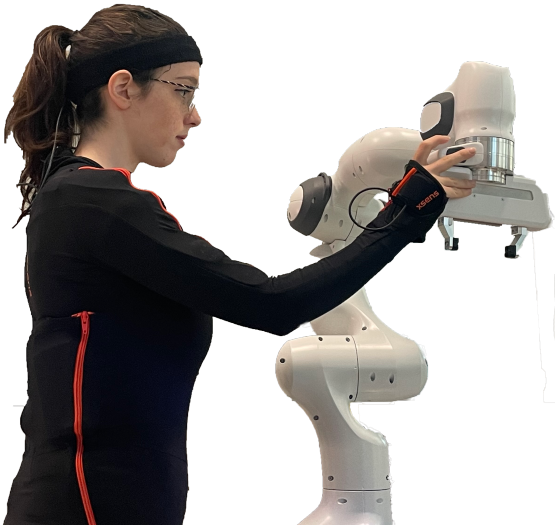


**Figure 3.1:** Experimental setup

The suit integrates 17 MTx sensors, each equipped with continuous updates designed to emulate the biomechanical attributes of the human body. These sensors are interconnected in a daisy-chain configuration, connected to the Xbus Master—a portable device responsible for data management. As a result, each limb requires only a single cable connection. The Xbus Master assumes a crucial role in synchronizing the sampled values from each sensor, providing power to the sensors, and overseeing wireless communication with the PC.

For the purpose of our experiments, we selected only the data from the four sensors positioned on the right arm and upper back. They provided us with joint angles, Cartesian velocity, and Cartesian acceleration.

When attaching sensors to the body, the initial alignment between the sensors and body segments is unknown. Therefore, a calibration procedure, composed of three steps, is necessary to establish the correct alignment and determine body dimensions.

In the first step, the subject is instructed to assume a predefined pose, such as the T-pose or N-pose, shown in Fig 3.2. The rotation from the sensor to the body segments $\mathbf{q}_{BS}$ is determined by aligning the sensor's orientation in the global frame $\mathbf{q}_{GS}$ with the known orientation of each segment in the chosen pose $\mathbf{q}_{GB}$ in the global frame. This relationship can be represented as follows.

$$\mathbf{q}_{GB} = \mathbf{q}_{GS} \otimes \mathbf{q}_{BS}^{*} \tag{3.1}$$

where $\otimes$ denotes a quaternion multiplication and * the complex conjugate of the quaternion.

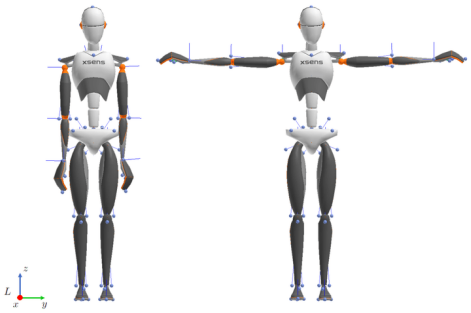


**Figure 3.2:** Calibration positions for Xsens [36]

In the second step, the subject is asked to perform a series of movements aligned with predefined axes. The measured orientation and angular velocity data are then utilized to deduce the sensor's orientation with respect to the segment's functional axes.

In the final step, the sensor-to-segment alignment and segment length are further adjusted by leveraging prior knowledge of the distances between two points in a kinematic chain.

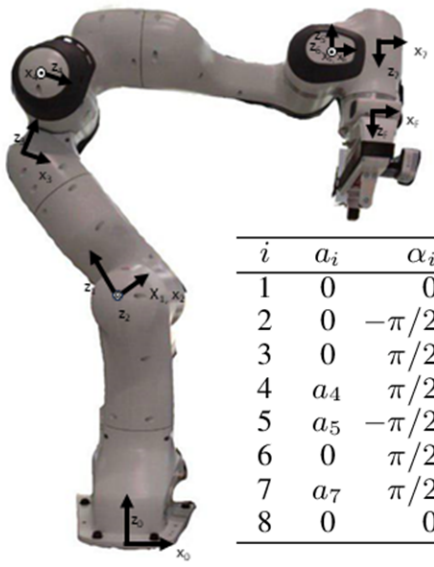
Once the calibration is complete, the collected data is translated into kinematic information for individual body segments, each of these components is interconnected by joints. The origin of each joint depends on the anatomical frame of the human body, with their respective X, Y, and Z axes corresponding to the functional movements of these joints.

However, the precision of joint position measurements, which are extensively used in our model, can be susceptible to various factors, including sensor noise and the influence of soft tissues surrounding the joints. To address this variability, a Kalman filter is employed by the Xbus Master, to refine the measurements.

To rectify orientation errors stemming from gyro integration issues, we utilize accelerometer data to define the gravity vector, which provides inclination stability. Additionally, a magnetic sensor is employed to correct these errors. It is important to note that the Earth's magnetic field can be locally disrupted by metallic objects, and this disturbance is continually estimated at each time step as part of the correction process. This robust approach ensures a high degree of immunity against distortions in the measurements.

### 3.1.2 Franka Emika Panda manipulator

The manipulator used is a Panda robot [37] developed by the German company Franka Emika. It is an advanced collaborative robot with seven degrees of freedom and an interchangeable end-effector. Designed to work seamlessly alongside humans,



| $i$ | $a_i$ | $\alpha_i$ | $d_i$ | $\theta_i$ |
|---|---|---|---|---|
| 1 | 0 | 0 | $d_1$ | $q_1$ |
| 2 | 0 | $-\pi/2$ | 0 | $q_2$ |
| 3 | 0 | $\pi/2$ | $d_3$ | $q_3$ |
| 4 | $a_4$ | $\pi/2$ | 0 | $q_4$ |
| 5 | $a_5$ | $-\pi/2$ | $d_5$ | $q_5$ |
| 6 | 0 | $\pi/2$ | 0 | $q_6$ |
| 7 | $a_7$ | $\pi/2$ | 0 | $q_7$ |
| 8 | 0 | 0 | $d_f$ | 0 |

**Figure 3.3:** Denavit Hartenber parameter of the Panda manipulator

the Panda robot boasts a maximum payload capacity of three kilograms, making it ideal for a wide range of applications.

The Panda manipulator's kinematics are well-defined by the Denavit-Hartenberg model, which accurately describes the robot's configuration and movement, as shown in Figure 3.3. In this model, the parameters are as follows: $a_4 = 0.0825$, $a_5 = -0.0825$, $a_7 = 0.0880$, $d_1 = 0.330$ m, $d_3 = 0.3160$, $d_5 = 0.3840$, and the flange offset $d_f = 0.1070$.

Libfranka, a C++ implementation of the client side of the FCI (Franka Control Interface), simplifies control and communication with the Panda robot. Libfranka efficiently manages network communication with the control system and provides a range of interfaces to achieve various objectives:

- Execution of non-realtime commands for Hand control and Arm parameter configuration.

- A library for computing desired kinematic and dynamic parameters, enhancing motion planning precision and efficiency.

Within the libfranka repository, multiple examples can be found to illustrate the diverse functionalities offered by the Panda robot. Among these examples, we will

analyze the cartesian impedance file that was a key element in our data-collection protocol.

This code shows an implementation of an impedance controller without the inclusion of inertia, where the equilibrium point is the initial configuration. It enables the robot to interact gently with the environment, especially when handling delicate or fragile objects, preventing potential damage. Similar to the flexibility of a human arm, the Panda can adjust its rigidity by modulating its impedance, ensuring robustness while executing various tasks.

In the context of collaborative robotics, impedance control enables robots to be more sensitive to human interactions, making them safer and more efficient when working alongside humans. By modulating the robot's stiffness, it can offer appropriate resistance to external forces, preventing accidents and damage while maintaining smooth and coordinated movements.

The initial modifications focused on the integration of a framework for the computation of the Mass and Coriolis Matrices. These dynamic parameters were derived using data and methodologies outlined in the research paper titled "Dynamic Identification of the Franka Emika Panda Robot With Retrieval of Feasible Parameters Using Penalty-Based Optimization" by Gaz et al. [37].

This paper provides a feasible set of parameters that characterize the dynamics of the robot. Typically, such a problem falls within the realm of nonlinear problems, often characterized by an infinite set of potential solutions. Nevertheless, these solutions can be effectively constrained by implementing upper and lower boundaries to eliminate any infeasible options.

The first step is to derive a symbolic dynamic model of the robot, because the FCI controller of the Panda can return the estimation of the link side torques elasticity can be neglected so the equation Eulero Lagrange can be used.

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q}) = \boldsymbol{\tau} \tag{3.2}$$

where $\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}} \in \mathbb{R}^n$ are, respectively, the joint positions, velocities, and accelerations. The dynamic model represented in Equation 3.2 typically incorporates nonlinear functions involving the variables $\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}$ along with the dynamic parameters, which will be elaborated on in more detail later. For each link $l_i$, i=1,...,n, let $m_i$ be the mass and let

$$\mathbf{r}_{i,ci} = \begin{bmatrix} r_{ix} \\ r_{iy} \\ r_{i1z} \end{bmatrix} \tag{3.3}$$

and

$$\mathbf{J}_{li} = \begin{bmatrix} J_{ixx} & J_{ixy} & J_{ixz} \\ J_{iyx} & J_{iyy} & J_{iyz} \\ J_{izx} & J_{izy} & J_{izz} \end{bmatrix} \tag{3.4}$$

be the position of the center of mass and the symmetric inertia tensor with respect to the i-th link frame, respectively. At this stage, the dynamic parameters of all the robot links can be gathered in the three vectors:

$$\begin{aligned} \mathbf{p}_1 &= (m_1 \dots m_n)^T, \\ \mathbf{p}_2 &= (c_{1x}m_1 \; c_{1y}m_1 \; c_{1z}m_1 \dots c_{nx}m_n \; c_{ny}m_n \; c_{nz}m_n), \\ \mathbf{p}_3 &= (ß_1^T \dots ß_n^T) \end{aligned} \tag{3.5}$$

where $\mathbf{p}_1 \in \mathbb{R}^n$, $\mathbf{p}_2 \in \mathbb{R}^{3n}$, $\mathbf{p}_3 \in \mathbb{R}^{6n}$, and

$$\mathbf{I}_i = (J_{ixx} \; J_{ixy} \; J_{ixz} \; J_{iyy} \; J_{iyz} \; J_{izz}) \tag{3.6}$$

It is possible to rearrange (3.2) as

$$\mathbf{Y}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})\pi(\mathbf{p_1}, \mathbf{p_2}, \mathbf{p_3}) = \tau \tag{3.7}$$

where $\pi(\mathbf{p_1}, \mathbf{p_2}, \mathbf{p_3}) = \begin{bmatrix} \mathbf{p_1^T} \\ \mathbf{p_2^T} \\ \mathbf{p_3^T} \end{bmatrix} \in \mathbb{R}^p$

Because $\mathbf{Y}$ can be pruned to obtain a matrix with full column rank the coefficient of $\pi$ can be identified.

$$\hat{\pi}_R = \mathbf{Y}_R^+(q, \dot{q}, \ddot{q})\bar{\tau} \tag{3.8}$$

This estimation might lead to physically inconsistent parameters,in order to avoid these solutions the following constraints must be applied.

$$m_i > 0 \tag{3.9}$$

$$\frac{tr(\mathbf{I_l})}{2} - \lambda_{max}(\mathbf{I_l}) > 0 \tag{3.10}$$

Despite the reliability of the solution, the parameters that have been computed cannot be directly introduced into the code provided by libfranka as it would cause a violation of the real-time constraint. For this reason, parallel computation of the parameters has been introduced through the use of threads.

In the following paragraph, we will provide a brief description of threads, as a comprehensive exploration of this topic falls outside the scope of this thesis.

A thread in C++ refers to a fundamental unit of execution within a program, allowing for concurrent and parallel execution of tasks. They enable a program to perform multiple operations simultaneously, enhancing efficiency and responsiveness. Each thread represents an independent sequence of instructions that can execute concurrently with other threads within the same process.

Threads are crucial for multitasking and handling tasks that can be performed independently. They can be thought of as separate paths of execution, with their

own program counters, registers, and stacks. They can be created and managed using various C++ standard library functions and classes, such as std::thread. Threads can execute different functions or methods, and their execution can be controlled using synchronization mechanisms like mutexes, semaphores, and condition variables. Multi-threading is particularly useful in applications that need to handle tasks simultaneously, like graphical user interfaces, server applications, data processing, and real-time systems.

In this code, there are three separate threads, the first one computes the correct values of torques in order to obtain an elastic response of the Franka to the human contact. The second and the third ones are respectively used to compute the values of the mass matrix and Coriolis matrix given the current configuration of the Franka.

An additional enhancement has been incorporated into the original code, with the introduction of an equilibrium point that is different from the starting one. Upon compilation and execution of the code, the Franka manipulator initiates its motion towards the specified target position, all the while exhibiting a responsive and spring-like behavior. This feature allows the human operator to introduce modifications to the robot's trajectory, as needed, in real-time. This adaptability can prove exceptionally valuable in collaborative environments where safety takes precedence, mitigating potential risks associated with contact between the human operator and the robotic arm.

Additionally, the newly imposed trajectory by the human is automatically stored. When the robot encounters familiar initial and final conditions, it will autonomously adopt the learned trajectory to reach the target point.

In the pursuit of enhancing the arm's capabilities, we introduced an additional feature into the system. In the modified code, when a subtle force is applied to the end effector while it is in its equilibrium position, the end effector does not display the expected spring-like behavior. Instead, the position to which it is guided becomes the new equilibrium point, while the manipulator still maintains a spring-like response if greater forces are applied. However, after a thorough evaluation, we made the deliberate choice to exclude this specific addition. We reasoned that this feature had the potential to unintentionally compromise the objective of our project, as it could interfere with the force estimation.

To acquire the essential data required for training our Neural Network, we leverage the Panda's capability to estimate external forces utilizing the residual approach, as detailed in Section 2.5. These force values, accompanied by their corresponding timestamps, are periodically recorded in a text file. They serve as the ground truth data to compute the loss function for our network.

In Figure 3.4, an illustration of our data set's sample forces can be observed. The first line contains the timestamp in the Linux format, while the subsequent six values represent the estimated forces and torques acting on the end effector. These

values are sampled at a frequency of 125 Hz.

```
1692694375512

 -2.55672
  3.38664
 0.241366
 0.252163
0.0617826
-0.514228

1692694375520

 -2.45665
   3.4542
 0.350723
 0.267495
0.0455337
-0.513808
```

**Figure 3.4:** Force sample

## 3.2 Data set description

Different experiments were collected in order to compose the training data set. They comprise variable intensities and directions of the contact force. For each experiment, the positions, velocities and accelerations of all the joints from the haptic suit were recorded, together with the Cartesian velocity and acceleration of the human arm interacting with the robot. Concurrently, the intensity and direction of the contact forces were collected from the robot.

The data set contains comprehensive information about the kinematics of the human's right arm. This encompassed a comprehensive array of metrics, including joint angles, Cartesian velocities, and accelerations across the arm's forearm, upper arm, and shoulder.

This data set was compiled to encapsulate the diverse range of human-induced forces that the robot could potentially experience. The values were collected by the Franka at a frequency of 125 Hz, a good balance of this parameter was crucial, on one side choosing a too-high value could cause of infraction of the real-time constraint applied on the Franka and also generate a data set too homogeneous that would risk overfitting when used on a Neural Network, on the other side a value too small might prevent us from learning significative insights on the human intentions.

On the Xsens, the data were sampled at a frequency of 250 Hz through the MVN Analyze software tool, as shown in Fig 3.5. The data extracted from the Xsens suit is stored in the MVN Open XML format, denoted by the file extension

".mvnx". These files can be conveniently opened and processed in various software applications, including Microsoft Excel, Access, MATLAB, and C-Motion Visual 3D. The parsing of these values is facilitated by a lightweight Python parser, which effectively extracts the relevant data.

An illustrative example of this file extension can be observed in Figure 3.6. Notably, this file comprehensively encapsulates a full description of body motion, offering a vector of 69 entries for each of the analyzed elements. These entries represent the 23 segments of the Xsens biomechanical model, providing a rich and detailed account of motion kinematics.

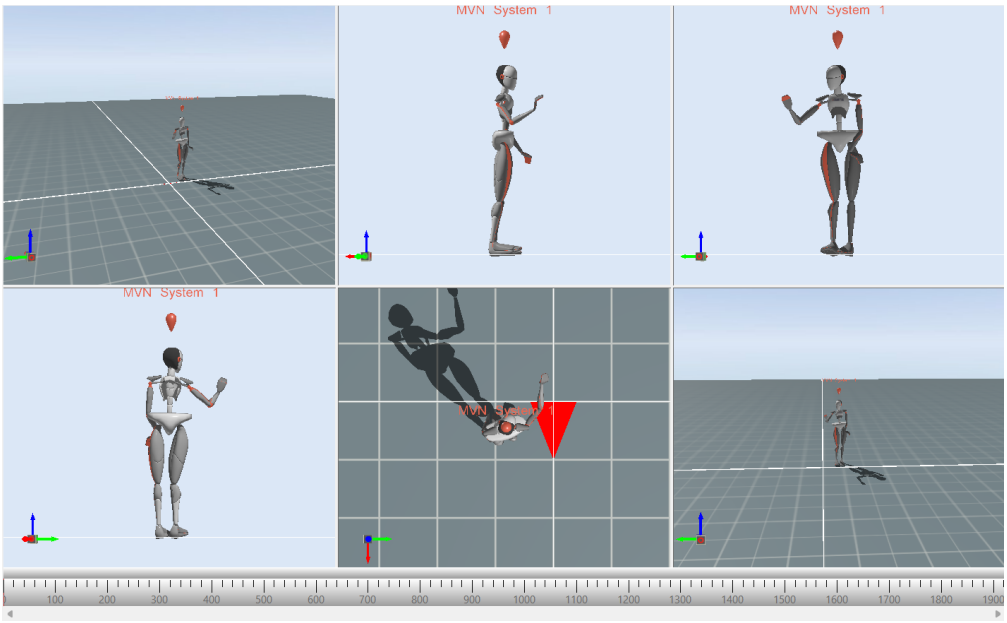



**Figure 3.5:** MVN xsens suit

The two sets of data were gathered from two distinct machines due to the specific requirements of each tool; libfranka necessitates a real-time Linux kernel, while Xsens is limited to operating on a Windows environment. Synchronizing these two data sources posed a challenge, but it was successfully achieved using Network Time Protocol (NTP).

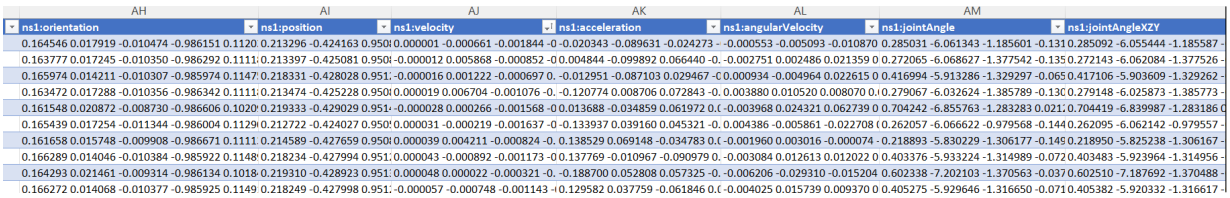
An NTP is a widely used networking protocol designed to synchronize the clocks of computers and devices within a computer network or across the internet. The primary purpose of NTP is to ensure that various systems maintain accurate and synchronized time, even when they are distributed across different geographical locations and interconnected through various network paths.

NTP achieves its synchronization goal by employing a hierarchical architecture consisting of different tiers of timekeeping sources, called NTP servers. It operates

using a client-server model.

The client devices, also known as NTP clients, periodically send time synchronization requests to one or more NTP servers. The server responds with its own timestamp, allowing the client to calculate the time difference, or offset, between its local clock and the server's clock. By exchanging multiple such messages with different servers, the client can calculate a more accurate time offset and adjust its local clock accordingly.

The implementation of Network Time Protocol (NTP) has demonstrated a significant reduction in clock drift between two devices. In the absence of NTP synchronization, the clocks of these two machines could potentially drift apart by several seconds. However, with the introduction of NTP synchronization, this drift has been reduced to an average of 20 milliseconds. The timestamps of the collected values are now synchronized and can be correctly associated [38].



| AH | AI | AJ | AK | AL | AM |
|---|---|---|---|---|---|
| ns1:orientation | ns1:position | ns1:velocity | ns1:acceleration | ns1:angularVelocity | ns1:jointAngle | ns1:jointAngleXZY |
| 0.164546 0.017919 -0.010474 -0.986151 0.1120 | 0.213296 -0.424163 0.950( | 0.000001 -0.000661 -0.001844 -0 | -0.020343 -0.089631 -0.024273 - | -0.000553 -0.005093 -0.010870 | 0.285031 -6.061343 -1.185601 -0.131 | 0.285092 -6.055444 -1.185587 - |
| 0.163777 0.017245 -0.010350 -0.986292 0.1111 | 0.213397 -0.425081 0.950( | -0.000012 0.005868 -0.000852 -0 | 0.004844 -0.099892 0.066440 -0. | -0.002751 0.002486 0.021359 0 | 0.272065 -6.068627 -1.377542 -0.135 | 0.272143 -6.062084 -1.377526 - |
| 0.165974 0.014211 -0.010307 -0.985974 0.1147 | 0.218331 -0.428028 0.951( | -0.000016 0.001222 -0.000697 0. | -0.012951 -0.087103 0.029467 -0 | 0.000934 -0.004964 0.022615 0 | 0.416994 -5.913286 -1.329297 -0.065 | 0.417106 -5.903609 -1.329262 - |
| 0.163472 0.017288 -0.010356 -0.986342 0.1111 | 0.213474 -0.425228 0.950( | 0.000019 0.006704 -0.001076 -0. | -0.120774 0.008706 0.072843 -0. | 0.003880 0.010520 0.008070 0.( | 0.279067 -6.032624 -1.385789 -0.130 | 0.279148 -6.025873 -1.385773 - |
| 0.161548 0.020872 -0.008730 -0.986606 0.1020 | 0.219333 -0.429029 0.951( | -0.000028 0.000266 -0.001568 -0 | 0.013688 -0.034859 0.061972 0.( | -0.003968 0.024321 0.062739 0 | 0.704242 -6.855763 -1.283283 0.021( | 0.704419 -6.839987 -1.283186 0 |
| 0.165439 0.017254 -0.011344 -0.986004 0.1129 | 0.212722 -0.424027 0.950( | 0.000031 -0.000219 -0.001637 -0 | -0.133937 0.039160 0.045321 -0. | 0.004386 -0.005861 -0.022708 ( | 0.262057 -6.066622 -0.979568 -0.144 | 0.262095 -6.062142 -0.979557 - |
| 0.161658 0.015748 -0.009908 -0.986671 0.1111 | 0.214589 -0.427659 0.950( | 0.000039 0.004211 -0.000824 -0. | 0.138529 0.069148 -0.034783 0.( | -0.001960 0.003016 -0.000074 - | 0.218893 -5.830229 -1.306177 -0.149 | 0.218950 -5.825238 -1.306167 - |
| 0.166289 0.014046 -0.010384 -0.985922 0.1148 | 0.218234 -0.427994 0.951( | 0.000043 -0.000892 -0.001173 -0 | 0.137769 -0.010967 -0.090979 0. | -0.003084 0.012613 0.012022 0 | 0.403376 -5.933224 -1.314989 -0.072 | 0.403483 -5.923964 -1.314956 - |
| 0.164293 0.021461 -0.009314 -0.986134 0.1018 | 0.219310 -0.428923 0.951( | 0.000048 0.000022 -0.000321 -0. | -0.188700 0.052808 0.057325 -0. | -0.006206 -0.029310 -0.015204 | 0.602338 -7.202103 -1.370563 -0.037 | 0.602510 -7.187692 -1.370488 - |
| 0.166272 0.014068 -0.010377 -0.985925 0.1149 | 0.218249 -0.427998 0.951( | -0.000057 -0.000748 -0.001143 -0 | 0.129582 0.037759 -0.061846 0.( | -0.004025 0.015739 0.009370 0 | 0.405275 -5.929646 -1.316650 -0.071 | 0.405382 -5.920332 -1.316617 - |

**Figure 3.6:** Kinematic values

Before saving the data in an appropriate format, some preprocessing is required in order to cancel some of the noise that might reduce the performance of the Neural Network.

The data obtained from the suit appears relatively homogeneous except for the presence of some abrupt spikes. In response to this, we implemented an algorithm designed to mitigate the impact of these high-amplitude spikes, effectively addressing the requirements of our experiments.

In contrast, the data associated with the interaction forces exhibited a considerably higher level of noise. To enhance the signal quality and reduce noise interference, we employed a Moving Average filter, resulting in a smoother and more reliable signal for analysis.

The moving average filter is a valuable tool for noise reduction due to its inherent ability to smooth out fluctuations in a data set. By calculating the average of a window of adjacent data points, the filter effectively suppresses random noise that may be present in individual measurements. This process minimizes the impact of sudden, short-term variations, resulting in a clearer and more consistent signal representation. The moving average filter's simple yet effective approach helps reveal underlying trends or patterns in the data while attenuating high-frequency noise components. The data collected and cleaned from any residual noise is then saved into two separate CSV files, one for training and one for testing.

For the Neural Network's data preparation, a careful partitioning strategy is adopted. Specifically, we have chosen to allocate 80% of the data set to the training set, reserving the remaining 20% for the testing set. This partitioning ratio adheres to one of the most commonly used approaches in the hold out method, a popular technique for data set splitting in machine learning explained in section 2.2.2. This division ensures that the network is trained on a substantial portion of the data while retaining a separate subset for unbiased evaluation during testing. By maintaining this ratio, we strike a balance between training and testing that helps in building a robust and effective Neural Network model.

Each entry in the time series is indexed, enabling the differentiation of individual recorded actions. To prepare the input for the model, a specific sub-portion of the collected time series is selected based on its index. However, prior to being fed into the model, it is essential to normalize each element within the series. This normalization step ensures that the data is on a consistent scale, preventing any disproportionate influences on the model's learning process.

For this purpose, the chosen normalization technique is the MinMax scaler. This technique scales the data to a fixed range, from 0 to 1. This approach is commonly favored because it typically leads to a smaller standard deviation, effectively mitigating the impact of outliers. This stands in contrast to other popular normalization methods like Z-score normalization. The MinMax scaling is performed using the following equation:

$$X_{\text{scaled}} = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}} \tag{3.11}$$

Where: $X_{scaled}$ is the scaled value, $X$ is the original value, $X_{min}$ is the minimum value in the data set, $X_{max}$ is the maximum value.

This equation ensures that the data is transformed to fit within the desired range while preserving the original relationships between the values. It is important to note that we exclusively applied normalization to the input series, leaving the target series untouched. This approach was chosen to prevent any inadvertent leakage of information regarding future events to the model.

The inclusion of timestamps in the inputs is crucial for the proper functioning of the transformer architecture. While transformers are widely recognized for their effectiveness in natural language analysis—outperforming previous solutions—these architectures have also showcased their adaptability across various tasks. However, a challenge arises when deploying transformers for time series forecasting. The issue pertains to potential loss of temporal information due to simultaneous processing of multiple data batches. This concern is not as pronounced in the realm of natural language processing (NLP) where semantic meaning often remains intact even with reordering of words. Therefore, to ensure accurate time series forecasting, it becomes imperative to provide the model with the temporal information that could

otherwise be lost during parallel processing of data batches. A summary of all the data forming the data set for training our model is given in table 3.1.

| Data type | Description | Symbol | Units | Device | Input/Target |
|---|---|---|---|---|---|
| Joints | shoulder roll<br>shoulder pitch<br>shoulder yaw<br>elbow roll<br>elbow pitch<br>elbow yaw | $q^h_{s\phi}$<br>$q^h_{s\theta}$<br>$q^h_{s\psi}$<br>$q^h_{e\phi}$<br>$q^h_{e\theta}$<br>$q^h_{e\psi}$ | $rad$ | mocap suit | Input&Target |
| Cartesian vel. | shoulder<br>upper arm<br>forearm | $\dot{\boldsymbol{p}}^h_s$<br>$\dot{\boldsymbol{p}}^h_u$<br>$\dot{\boldsymbol{p}}^h_f$ | $m/s$ | mocap suit | Input&Target |
| Cartesian acc. | shoulder<br>upper arm<br>forearm | $\ddot{\boldsymbol{p}}^h_s$<br>$\ddot{\boldsymbol{p}}^h_u$<br>$\ddot{\boldsymbol{p}}^h_f$ | $m/s^2$ | mocap suit | Input&Target |
| Contact force | | $\boldsymbol{f}_{\text{ext}}$ | $N$ | robot+eq. (??) | Target |

**Table 3.1:** Training Input Data

## 3.3 Architecture

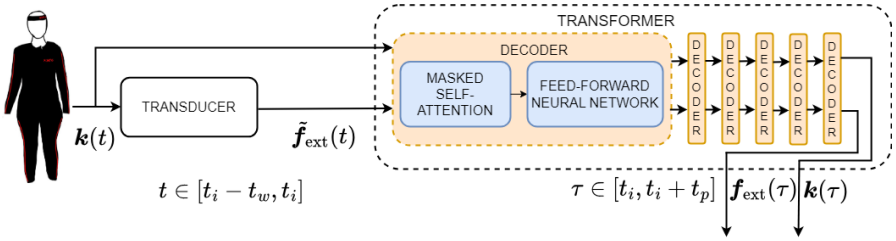



**Figure 3.7:** The complete Architecture.

As depicted in Fig. 3.7, our Neural Network is composed of two main parts: a Transducer and a Transformer. The former is a fully-connected Multi-Layer Perceptron (MLP) [39], the latter is a Transformer that takes a multivariate time series as input at time $t_i$. Both networks were implemented using the PyTorch library for training and testing functions.

### 3.3.1 Transducer Architecture

As our model does not rely on a Torque/Force sensor, we employ two networks to provide a force prediction. The first network, the Transducer, provides a reliable estimation of the contact forces through the analysis of kinematic data.

The Transducer takes as input the kinematic data $\boldsymbol{k}(t_i)$ from the suit at the current time instant $t_i$, and estimates the force $\boldsymbol{f}_{\text{ext}}(t_i)$ exerted at $t_i$. The Neural

Network architecture is defined using PyTorch's nn.Sequential module. The architecture consists of several fully connected (linear) layers interspersed with Rectified Linear Unit (ReLU) activation functions as shown in Fig 3.8. The input layer has 36 neurons, corresponding to the 36 features in the data set listed in Table 3.1. The architecture is designed in a feedforward manner, where the output of each layer is passed as input to the next layer. The ReLU activation functions introduce non-linearity into the model, allowing it to capture complex relationships in the data. Adam optimizer is used to update the network's weights during training with a learning rate of 0.0001.

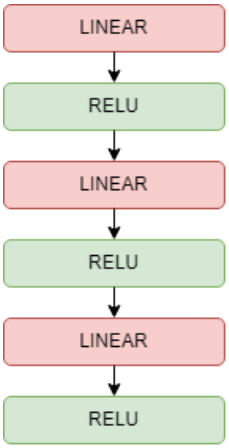



**Figure 3.8:** Transducer Architecture.

### 3.3.2 Transformer Architecture

The architecture we have implemented for the Transformer is based on an adapted version of GPT-2 [40] specifically designed for time series applications. Unlike other Transformers like BERT [41], which are bidirectional, GPT-2 is a generative model. This architecture incorporates the autoregression concept from RNNs, generating one output at a time and feeding it into the next sequence's input. This mechanism enhances the network's suitability for time series prediction.

In the traditional setup, a transformer model comprises two main components: an encoder and a decoder. Within the encoder, two fundamental elements exist a self-attention mechanism and a Feedforward Neural Network, as shown in 2.3. The self-attention mechanism enables each element within the input to establish connections with various segments of the entire sequence, based on the information encapsulated in three matrices known as Queries, Keys, and Values. On the other hand, the decoder incorporates an additional layer called Masked Self-Attention before the Feedforward Neural Network. The term masking indicates that an

element at a specific position in the sequence can only attend to past input data, preserving the autoregressive nature of the decoder.

A significant difference from the original Transformer architecture is that GPT-2 relies solely on a decoder structure. Unlike the conventional Transformer, which was initially designed for sequence-to-sequence tasks like language translation, GPT-2 is more versatile and can be applied to time series forecasting due to its generative capabilities.

Therefore, our architecture exclusively utilizes decoder layers. Specifically, we employ six consecutive decoder blocks, a value chosen empirically to strike a balance between mitigating the risks of both underfitting and overfitting.

## 3.4 Neural Network Training

In this section, we analyze the training procedures for our dual-network architecture, encompassing the Transducer and the Transformer. We delineate the training process for each network independently. Initially, the Transducer network undergoes standalone training. Following this, during the subsequent training of the Transformer, the Transducer is integrated into the architecture, as illustrated in Fig. 3.7, with its weight parameters held constant.

### 3.4.1 Transducer Training

The transducer is trained on a multivariate input consisting exclusively of kinematic parameters denoted as $\boldsymbol{k}$ from the mocap suit. The training data set comprises 86,000 data points. Before feeding into the model, the data is shuffled and normalized using a MinMax scaler. To evaluate the model's performance during the training phase, we employ the Mean Square Error criterion, comparing its predicted outputs with the values estimated by the Franka Emika manipulator.

$$\text{loss} = \frac{1}{n_t} \sum_{i=1}^{n_t} \cdot (y_i - \hat{y}_i)^2 \tag{3.12}$$

In the equation, $n_t$ represents the number of data in the training set, $y_i$ represents the actual value of the $i$-th data point, and $\hat{y}_i$ represents the predicted value of the $i$-th data point.

The training process is organized into epochs and mini-batches. Here are the key components of the training loop:

- Epochs: The training process iterates over a fixed number of epochs. An epoch represents one complete pass through the entire training data set.

- Mini-Batches: To efficiently train the model, the data set is divided into mini-batches. Each mini-batch consists of 10 data points, which helps in parallelizing computations and avoiding memory issues
- Forward Pass: For each mini-batch, a forward pass is performed through the Neural Network. This means that the input data is propagated through the layers to compute the predicted values.
- Model Evaluation: After each epoch, the model's performance is evaluated on the test data set using the MSE loss. The best model weights are stored if the current MSE is lower than the previously recorded best MSE.

In this network, the raytune open-source library was used for the tuning of the hyperparameters dropout, learning rate and number of epochs [42].

### 3.4.2 Transformer Training

The Transformer takes as input a multivariate time series composed of the kinematic data coming from the suit and the forces estimated by the Transducer over the time sequence $[t_i - t_w, t_i]$ and produces in output a prediction of the kinematic and dynamic data over the time window $[t_i, t_i + t_p]$, where $t_p$ is the length of the prediction window that in our project is set to 800 ms .

The primary challenge that affects Neural Networks during training is the discrepancy between the training and inference phases. A common training technique, for time series prediction, is known as "teacher-forcing." In teacher-forcing, during each new time step of a multi-step prediction, the model is provided with access to all the true preceding values to infer the new element. This strategy serves to prevent the training sequence from diverging; however, it can introduce a higher error during inference, where the model can no longer rely on continuous correction during multi-step prediction. To address this challenge and enhance Transformer performance, we adopted a technique known as *scheduled sampling* [43].

This approach enables the Transformer to initially use the ground-truth (i.e., true) data during the early stages of training. However, as training progresses, the Transformer gradually incorporates more of its own predicted outputs into its input sequence, reducing its reliance on true values over time. This transition begins with a high probability of using true values as inputs and gradually shifts towards greater reliance on the model's predictions. This strategy strikes a balance between providing the model with valuable training information and preparing it for real-world scenarios where actual values may not be readily available. The transition function we employed follows a sigmoid pattern with inverse decay:

$$\mathrm{v} = \frac{l}{l + \exp\left(\frac{n_{ep}}{l}\right)} \tag{3.13}$$

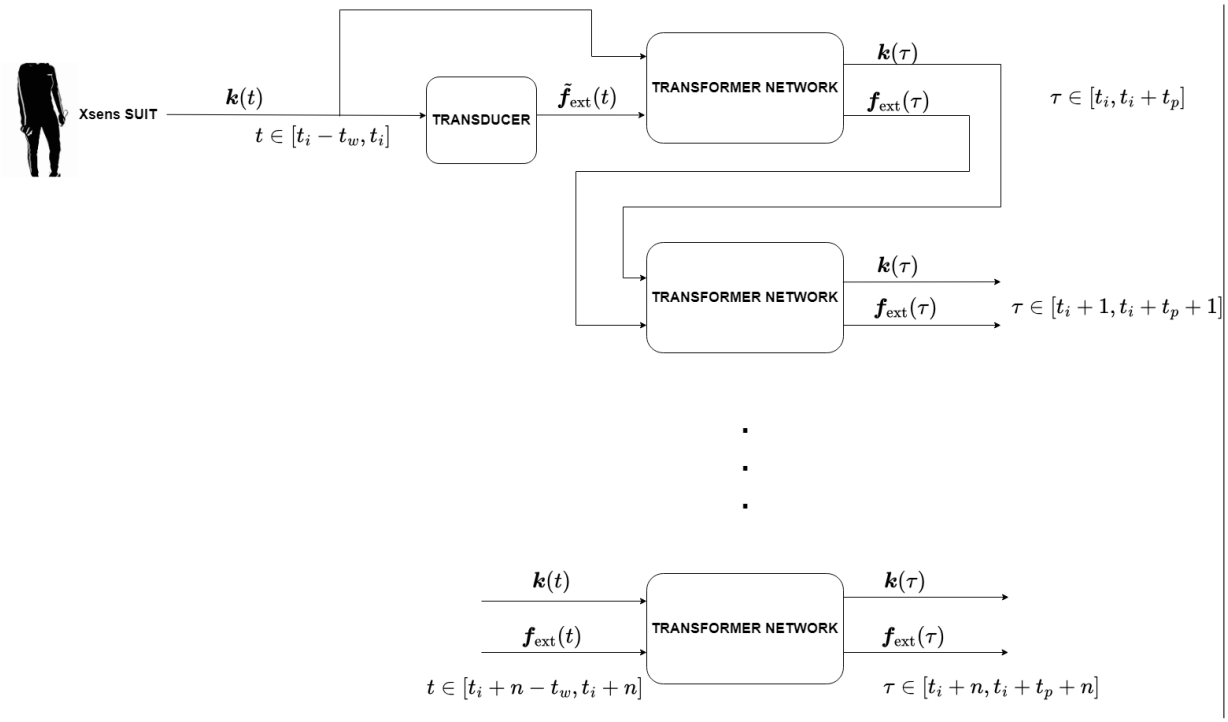



**Figure 3.9:** Recursive Transformer Architecture

where $v$ is the probability of selecting the ground-truth data, $n_{ep}$ is the epoch number in the training set, $l \geq 1$ is a user-defined parameter for the speed of convergence. Since our training data set includes both kinematic and dynamic (i.e., forces) values, we have chosen to employ a weighted mean square error as our loss function. This choice is guided by the network's continuous access to the accurately measured kinematic values, making them relatively easier to predict. Moreover, recognizing the heightened significance of accurate force prediction in the context of physical human-robot interaction, we have placed greater emphasis on minimizing errors in force predictions.

$$\text{loss} = \frac{1}{n_t} \sum_{i=1}^{n_t} w_i \cdot (y_i - \hat{y}_i)^2 \tag{3.14}$$

In the last equation, $n_t$ represents the number of data in the training set, $w_i$ represents the user-defined weight assigned to the $i$-th data point, $y_i$ represents the actual value of the $i$-th data point, and $\hat{y}_i$ represents the predicted value of the $i$-th data point.

Overfitting was one of the main challenges we encountered during the fine-tuning process. Different strategies were adopted to provide the best testing results.

1. Early Stopping was adopted and provided a significant improvement in reducing the loss in the testing phase. It helps prevent excessive training and overfitting to the training data.

2. Dropout, multiple experiments were required to fine-tune the parameters. A small dropout rate had little impact on the final results, while a high rate hindered the network's ability to learn the series features.

3. L1 Regularization was introduced to further improve the results. In this technique a penalty term is added to the loss function based on the absolute values of the model's weights, encouraging sparsity and mitigating overfitting.

4. Batch Size, experimentation with increasing the batch size was performed. However, it was found to be ineffective in our specific experiment.

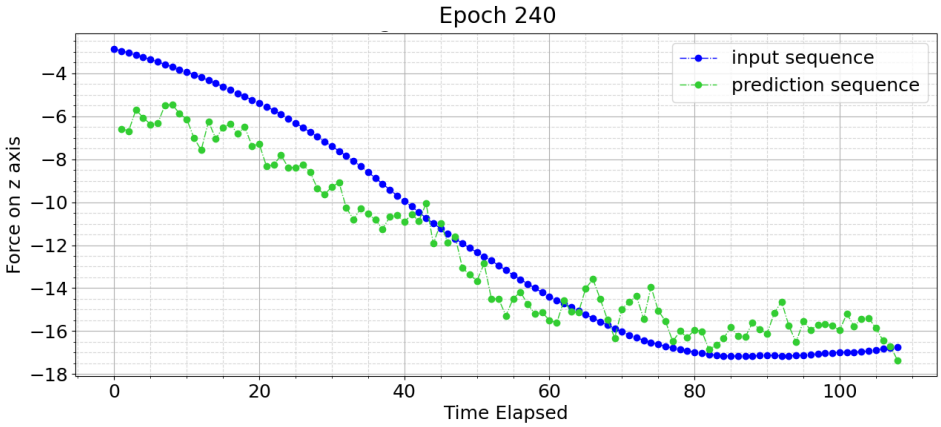
5. Learning Rate and Weight Decay, adjustments were made to reduce the learning rate and weight decay. These hyperparameter adjustments aimed to fine-tune the model's training process and improve generalization.

By implementing these strategies, we aimed to strike a balance between model complexity and generalization, ultimately addressing the challenge of overfitting and achieving better testing results.

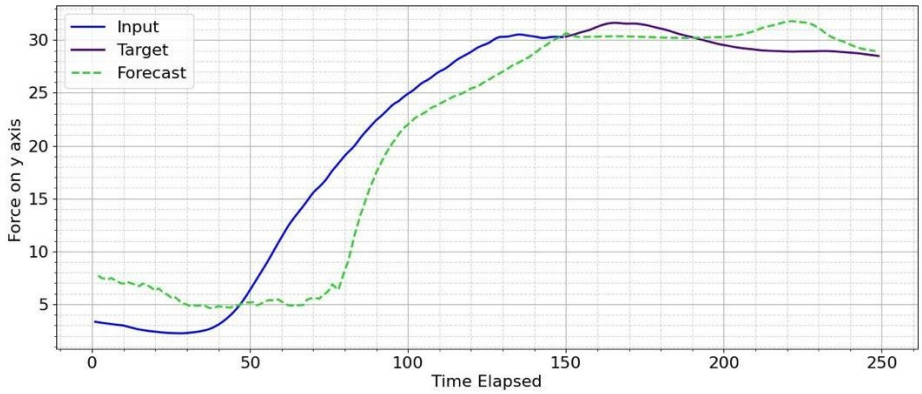
To ensure the precision of predictions, each input sequence is segmented into two distinct sections: the Training sequence and the Forecast sequence. The Training sequence represents the minimum number of samples that the network must analyze to generate dependable predictions. At each new step of this section, the input is updated with the last prediction as can be seen in Fig 3.9

In Fig. 3.10b, we can observe the distinction between the two sections, where the input sequence corresponds to the Training sequence. In this section, the predictions appear less stable, exhibiting a tendency for fluctuations. In contrast, the values in the second part of the sequence are notably smoother and devoid of spikes.

In our project we chose a different Training Length in the training phase than in the testing phase, in particular, we chose a longer forecast window than in the testing. We can see in the figures the results of training and testing.




**(a)** An example of training




**(b)** An example of testing

**Figure 3.10:** Example of force prediction in our model

# Chapter 4

# Experimental Results

In this section, we will delve into the data collection process, encompassing both training and testing phases. Subsequently, we will showcase the outcomes of our experiments conducted within the context of predicting contact forces and human motions using our specialized framework.

To underscore the predictive prowess of our architecture and the efficacy of the Transducer component, we will conduct a comparative analysis against a pure Transformer-based architecture.

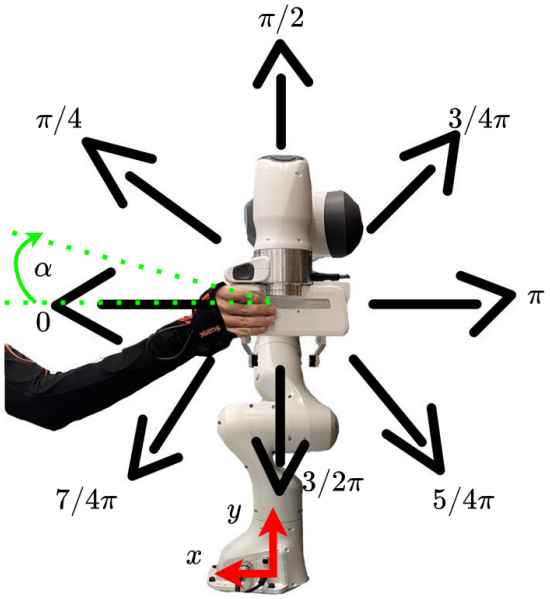
## 4.1 Data Collection Protocol

Our data set comprises 320 samples, each containing detailed information about the motion executed by a human operator during interactions with a robotic arm. The human operator wore an Xsens motion capture suit and was positioned in close proximity to the robotic arm. In this controlled environment, the human operator intentionally applied forces to the end-effector of the robotic arm using their right hand.

As previously mentioned, the robot was controlled using an impedance controller, with the stiffness, inertial, and damping matrices held constant throughout the data collection phase. The robot is tasked to maintain its initial configuration and to respond when a human operator applies force to its end-effector.

During these interactions, once a force is detected, both the kinematic human information described in Section 3.1.1 and the estimated contact forces from Section 3.1.2 are captured with a frequency rate of 125 Hz. This high sampling frequency allows for the precise recording of rapid movements.

Each movement in the training data set corresponds to one of eight equally spaced planar directions, as illustrated in Fig. 4.1. In each sample the robot starts from its resting configuration and moves along one of the directions denoted with




[H]

**Figure 4.1:** Visualization of the 8 directions of motions

$\alpha = \{0, \pi/4, \pi/2, 3\pi/4, \pi, 5\pi/4, 3\pi/2, 7\pi/4\}$.

We collected 40 samples for each direction of motion, covering a broad spectrum of force intensities ranging from 0 N to ±40 N. Consequently, the robot's end-effector exhibited displacements spanning from 0 cm to ±25 cm. The duration of each individual sample averages 2.5 seconds.

Following the data collection phase, we synchronized and filtered the data. This process was employed to enhance the overall data quality and to mitigate the impact of noise interference.

It is crucial to emphasize that we assume the active participation of the human operator in generating motion during each experiment, with the intent of following one of the previously mentioned directions. This intentional motion is essential to establish a meaningful correlation between the kinematic and force data within our training data set.

## 4.2 Comparison description

We compared the proposed model with a similar architecture that lacks a Transducer network, making it a pure transformer-based model. In this alternative network, no estimation of contact forces is provided. The input consists solely of timestamps and kinematic data, while the output remains consistent with the model that includes the Transducer.

The pure transformer-based network shares an identical structure with the one utilized in our framework, which is a GPT-2-based model comprising 6 decoder layers. We train this network using the same data set, as described in Section 4.1. For performance analysis, we employ a weighted mean square error metric with the same weights as previously employed. Specifically, in both cases, we assign four times more weight to the error on the force values compared to the error on the kinematic components, this value was chosen empirically as it provided the minimum loss.

Both networks utilize a prediction window of 1.6 seconds during the training phase. To mitigate overfitting, we employ Early Stopping and apply a dropout rate of 0.35.

During the testing phase, we optimize performance by reducing the prediction window to 800 ms.

Both networks are trained on the same computer, which is equipped with 64 GB of RAM and an Intel(R) i7 Core processor running at 3.20 GHz.

## 4.3 Results and discussion

For testing purposes, we evaluate the performance of these two networks on two data sets. The first data set contains directions already present in the training set, while the second data set comprises planar motions that have not been explored during training.

Both architectures demonstrated the ability to make precise predictions when evaluated on the first data set. The results for both networks can be found in Tables 4.1a and 4.1b, which include the average errors calculated for joint angles, cartesian velocity, cartesian acceleration, and contact forces.

However, our model exhibited superior generalization capabilities when tested in new directions. The second data set is composed of 7 distinct motions that were not part of the training set. In each sample, the robot starts from its resting configuration and moves along one of the new directions denoted with $\alpha = \{4\pi/3, \pi/3, 5\pi/3, 8\pi/9, 10\pi/9, \pi/6, 11\pi/6\}$.

These testing motions were acquired through the procedure outlined in Section 4.1, with the sole difference being that we gathered only one motion for each testing direction The prediction errors for these new directions are presented in Tables 4.1, 4.2, and 4.3. A detailed description of the tables is provided below:

- In Tables 4.1c and 4.1d, the columns labeled $q^h_{sr}$, $q^h_{sp}$, $q^h_{sy}$, $q^h_{er}$, $q^h_{ep}$ and $q^h_{ey}$ represent the errors for the shoulder and elbow joints across roll, pitch and yaw angles.

- In Tables 4.2a and 4.2b, the columns $\dot{p}^h_{sx}$, $\dot{p}^h_{sy}$, $\dot{p}^h_{sz}$, $\dot{p}^h_{ux}$, $\dot{p}^h_{uy}$, $\dot{p}^h_{uz}$, $\dot{p}^h_{fx}$, $\dot{p}^h_{fy}$, and

$\dot{p}^h_{fz}$ represent the errors computed along the spatial components ($x$, $y$, $z$) of velocity for the human's shoulder, upper arm, and forearm.

- In the tables 4.2c and 4.2d, the columns $\ddot{p}^h_{sx}$, $\ddot{p}^h_{sy}$, $\ddot{p}^h_{sz}$, $\ddot{p}^h_{ux}$, $\ddot{p}^h_{uy}$, $\ddot{p}^h_{uz}$, $\ddot{p}^h_{fx}$, $\ddot{p}^h_{fy}$, and $\ddot{p}^h_{fz}$ represent the errors computed along the spatial components ($x$, $y$, $z$) of acceleration for the human's shoulder, upper arm, and forearm.
- In the tables 4.3a and 4.3b, the columns $f_{ext,x}$, $f_{ext,y}$ and $f_{ext,z}$ represent the error regarding the contact forces over the axis x, y and z

In each table, the final row represents the average of the values presented in the preceding rows. Notably, within the tables illustrating the results of the Transducer architecture, values that exhibit a lower average loss compared to their counterparts in the pure Transformer architecture are highlighted in bold.

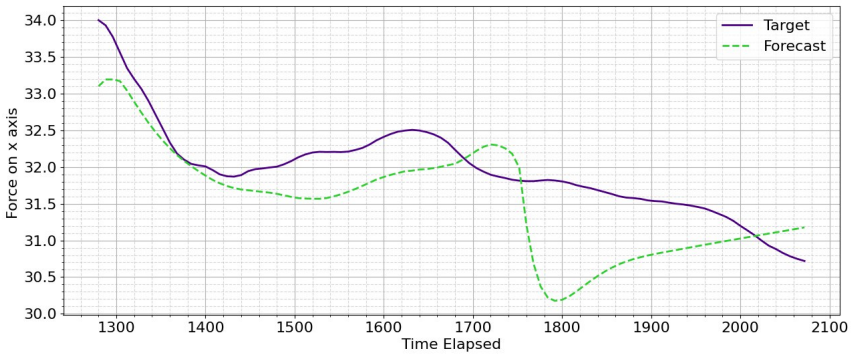
When comparing the tables, it becomes evident that our model exhibits a modest reduction in error when predicting the kinematic components compared to the model without Transducer, although this improvement is not highly significant. In contrast, a substantial enhancement is observed in the prediction of contact forces when a transducer is incorporated into the architecture.

An illustrative example of the output generated by our network, after the training period, is presented in Figures 4.2a, 4.2b, and 4.2c. Notably, this sample is associated with a direction that is not present in the training data set: a motion executed at an angle of $\frac{7\pi}{6}$, as per the reference frame illustrated in Figure 4.1.

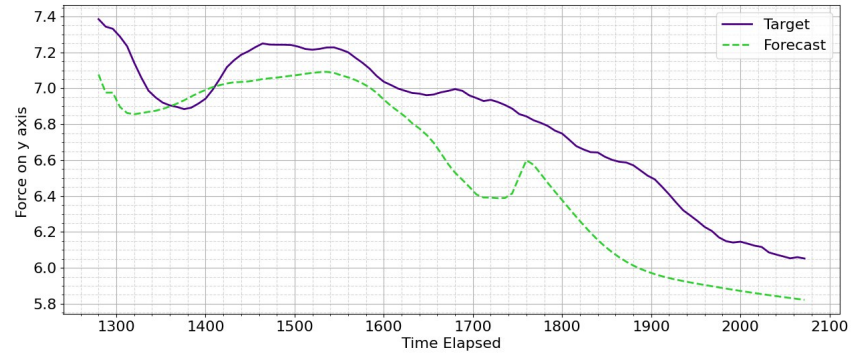
In these images, we notice that the prediction for the x-directional force accurately follows the expected trend. Furthermore, the prediction for the y-directional force is not only highly accurate in terms of the trend but also in terms of absolute values. In the z-direction, where no external force is applied, our model predicts a relatively constant trend, which aligns with the absence of significant variations.

The movements were confined to the x-y plane, and as a result, the force values measured along the z-axis often reflect measurement errors or human errors, typically averaging around 3 N. This accounts for the smaller error observed on the z-axis in both Table 4.3a and Table 4.3b. However, a substantial disparity between the two models emerges when we compare the errors in force measurements along the other two axes. Specifically, on the x-axis, the error is 9 N smaller on average, and on the y-axis, it is 6 N smaller.

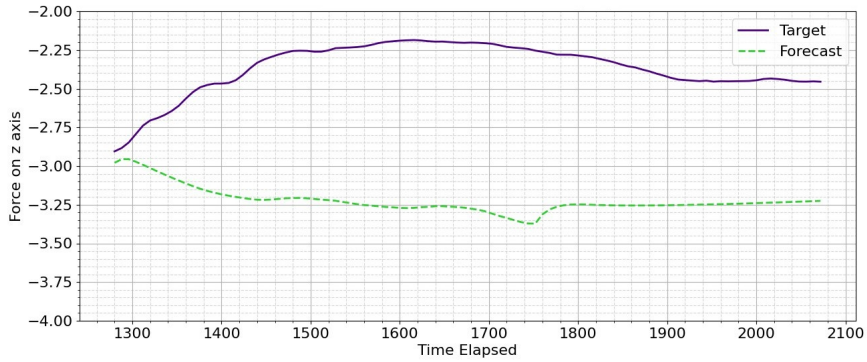
It is noteworthy that our framework achieves high accuracy in predicting contact forces thanks to the constant robot stiffness which facilitates the creation of a unique mapping between human kinematics and contact forces.




**(a)** Force on x axis in Newton




**(b)** Force on y axis in Newton




**(c)** Force on z axis in Newton

**Figure 4.2:** Prediction example of our model

| $\alpha$ | $\overline{q}^h$ [deg] | $\overline{\dot{p}_s^h}$ [m/s] | $\overline{\ddot{p}_s^h}$ [m/s$^2$] | $f_{ext,x}$ [N] | $f_{ext,y}$ [N] | $f_{ext,z}$ [N] |
|---|---|---|---|---|---|---|
| 0 | 6.366 | 0.029 | 0.19 | 0.63 | 2.56 | 1.72 |
| $-\pi/4$ | 2.0768 | 0.014 | 0.14 | 1.19 | 1.90 | 1.46 |
| $-\pi/2$ | 3.177 | 0.011 | 0.067 | 0.99 | 1.92 | 4.3 |
| $3\pi/4$ | 3.633 | 0.015 | 0.074 | 0.52 | 1.80 | 4.86 |
| $\pi$ | 8.29 | 0.018 | 0.065 | 0.15 | 0.94 | 3.73 |
| $5\pi/4$ | 3.45 | 0.020 | 0.15 | 1.68 | 3.15 | 1.53 |
| $\pi/2$ | 3.45 | 0.020 | 0.19 | 1.68 | 3.15 | 1.53 |
| $\pi/4$ | 5.26 | 0.019 | 0.18 | 1.38 | 1.32 | 4.11 |

**(a)** Prediction error on training directions with Transducer

| $\alpha$ | $\overline{q}^h$ [deg] | $\overline{\dot{p}_s^h}$ [m/s] | $\overline{\ddot{p}_s^h}$ [m/s$^2$] | $f_{ext,x}$ [N] | $f_{ext,y}$ [N] | $f_{ext,z}$ [N] |
|---|---|---|---|---|---|---|
| 0 | 3.70 | 0.02 | 0.018 | 0.51 | 3.51 | 0.66 |
| $-\pi/4$ | 0.013 | 0.074 | 0.016 | 1.95 | 1.91 | 2.50 |
| $-\pi/2$ | 2.76 | 0.01 | 0.06 | 0.72 | 1.58 | 5.15 |
| $3\pi/4$ | 3.40 | 0.014 | 0.07 | 0.86 | 1.67 | 5.06 |
| $\pi$ | 12.49 | 0.023 | 0.016 | 2.18 | 5.69 | 2.20 |
| $5\pi/4$ | 4.17 | 0.021 | 0.19 | 3.97 | 4.20 | 3.56 |
| $\pi/2$ | 2.61 | 0.013 | 0.11 | 1.37 | 0.65 | 3.92 |
| $\pi/4$ | 3.2 | 0.01 | 0.13 | 1.25 | 1.32 | 3.04 |

**(b)** Prediction error on training directions without Transducer

| $\alpha$ | $q_{sr}^h$ [deg] | $q_{sp}^h$ [deg] | $q_{sy}^h$ [deg] | $q_{er}^h$ [deg] | $q_{ep}^h$ [deg] | $q_{ey}^h$ [deg] |
|---|---|---|---|---|---|---|
| $4/3\pi$ | 22.90 | 19.10 | 7.19 | 9.88 | 5.00 | 1.51 |
| $\pi/3$ | 1.36 | 1.26 | 1.89 | 0.013 | 7.38 | 0.60 |
| $5/3\pi$ | 15.80 | 13.46 | 16.69 | 13.63 | 18.39 | 2.63 |
| $8/9\pi$ | 22.84 | 12.27 | 2.32 | 7.86 | 4.68 | 3.18 |
| $10/9\pi$ | 16.19 | 14.15 | 3.16 | 3.88 | 17.09 | 3.92 |
| $\pi/6$ | 4.65 | 12.12 | 1.90 | 2.87 | 21.90 | 12.12 |
| $11/6\pi$ | 1.18 | 1.04 | 0.68 | 1.72 | 9.85 | 3.81 |
| avg | **12.02** | **10.49** | **4.83** | 5.84 | **12.04** | **3.99** |

**(c)** Prediction error on joint angles with Transducer in degrees

| $\alpha$ | $q_{sr}^h$ [deg] | $q_{sp}^h$ [deg] | $q_{sy}^h$ [deg] | $q_{er}^h$ [deg] | $q_{ep}^h$ [deg] | $q_{ey}^h$ [deg] |
|---|---|---|---|---|---|---|
| $4/3\pi$ | 20.30 | 20.43 | 6.11 | 6.07 | 11.39 | 1.83 |
| $5/3\pi$ | 12.51 | 13.10 | 6.55 | 0.97 | 22.41 | 2.89 |
| $8/9\pi$ | 2.31 | 4.09 | 1.42 | 5.17 | 21.98 | 7.59 |
| $10/9\pi$ | 25.23 | 21.26 | 17.93 | 3.55 | 21.63 | 0.59 |
| $\pi/6$ | 15.38 | 18.59 | 4.67 | 3.99 | 20.90 | 12.12 |
| $11/6\pi$ | 1.47 | 2.36 | 2.20 | 5.67 | 14.20 | 7.07 |
| avg | 12.87 | 13.31 | 6.65 | 4.24 | 18.75 | 5.35 |



**(d)** NN prediction error on joint angles without Transducer in degrees

**Table 4.1:** Prediction error comparison on average values and on joint angles

| $\alpha$ | $\dot{p}^h_{sx}$ [m/s] | $\dot{p}^h_{sy}$ [m/s] | $\dot{p}^h_{sz}$ [m/s] | $\dot{p}^h_{ux}$ [m/s] | $\dot{p}^h_{uy}$ [m/s] | $\dot{p}^h_{uz}$ [m/s] | $\dot{p}^h_{fx}$ [m/s] | $\dot{p}^h_{fy}$ [m/s] | $\dot{p}^h_{fz}$ [m/s] |
|---|---|---|---|---|---|---|---|---|---|
| $4/3\pi$ | 0.0141 | 0.0168 | 0.0068 | 0.0269 | 0.0147 | 0.0202 | 0.0939 | 0.0189 | 0.077 |
| $5/3\pi$ | 0.0145 | 0.013 | 0.0026 | 0.0239 | 0.00692 | 0.00693 | 0.0306 | 0.0104 | 0.0098 |
| $8/9\pi$ | 0.0123 | 0.039 | 0.00690 | 0.01769 | 0.0155 | 0.0243 | 0.0831 | 0.1643 | 0.0636 |
| $10/9\pi$ | 0.01516 | 0.0608 | 0.0268 | 0.03108 | 0.0241 | 0.075 | 0.0746 | 0.0179 | 0.1391 |
| $\pi/6$ | 0.0297 | 0.0191 | 0.00329 | 0.0401 | 0.01446 | 0.0077 | 0.094 | 0.0276 | 0.0327 |
| $11/6\pi$ | 0.0195 | 0.0188 | 0.00239 | 0.0381 | 0.00876 | 0.00873 | 0.0647 | 0.083 | 0.01663 |
| avg | **0.0314** | 0.00826 | **0.0313** | 0.01707 | 0.01665 | **0.0612** | 0.0414 | **0.0492** | **0.0582** |

**(a)** NN prediction error on cartesian velocities with Transducer in $m/s$

| $\alpha$ | $\dot{p}^h_{sx}$ [m/s] | $\dot{p}^h_{sy}$ [m/s] | $\dot{p}^h_{sz}$ [m/s] | $\dot{p}^h_{ux}$ [m/s] | $\dot{p}^h_{uy}$ [m/s] | $\dot{p}^h_{uz}$ [m/s] | $\dot{p}^h_{fx}$ [m/s] | $\dot{p}^h_{fy}$ [m/s] | $\dot{p}^h_{fz}$ [m/s] |
|---|---|---|---|---|---|---|---|---|---|
| $4/3\pi$ | 0.014 | 0.019 | 0.064 | 0.02 | 0.0145 | 0.022 | 0.076 | 0.018 | 0.08 |
| $5/3\pi$ | 0.019 | 0.012 | 0.0028 | 0.047 | 0.012 | 0.032 | 0.102 | 0.074 | 0.15 |
| $8/9\pi$ | 0.025 | 0.024 | 0.007 | 0.04 | 0.016 | 0.009 | 0.079 | 0.032 | 0.033 |
| $10/9\pi$ | 0.01878 | 0.020 | 0.0033 | 0.027 | 0.008 | 0.002 | 0.096 | 0.04 | 0.041 |
| $\pi/6$ | 0.025 | 0.024 | 0.007 | 0.04 | 0.016 | 0.0097 | 0.079 | 0.032 | 0.033 |
| $11/6\pi$ | 0.018 | 0.015 | 0.0024 | 0.0209 | 0.015 | 0.0119 | 0.043 | 0.012 | 0.014 |
| avg | 0.01846 | 0.00511 | 0.035 | 0.01334 | 0.01167 | 0.076 | 0.0345 | 0.0624 | 0.06128 |

**(b)** Prediction error on cartesian velocities without Transducer in $m/s$

| $\alpha$ | $\ddot{p}^h_{sx}$ [m/s] | $\ddot{p}^h_{sy}$ [m/s] | $\ddot{p}^h_{sz}$ [m/s] | $\ddot{p}^h_{ux}$ [m/s] | $\ddot{p}^h_{uy}$ [m/s] | $\ddot{p}^h_{uz}$ [m/s] | $\ddot{p}^h_{fx}$ [m/s] | $\ddot{p}^h_{fy}$ [m/s] | $\ddot{p}^h_{fz}$ [m/s] |
|---|---|---|---|---|---|---|---|---|---|
| $4/3\pi$ | 0.2 | 0.24 | 0.13 | 0.11 | 0.059 | 0.066 | 0.27 | 0.12 | 0.27 |
| $5/3\pi$ | 0.052 | 0.052 | 0.021 | 0.097 | 0.08 | 0.12 | 0.08 | 0.14 | 0.74 |
| $8/9\pi$ | 0.149 | 0.24 | 0.269 | 0.255 | 0.064 | 0.094 | 0.479 | 0.257 | 0.33 |
| $10/9\pi$ | 0.256 | 0.408 | 0.344 | 0.142 | 0.133 | 0.132 | 0.293 | 0.142 | 0.663 |
| $\pi/6$ | 0.27 | 0.20 | 0.058 | 0.105 | 0.061 | 0.042 | 0.219 | 0.548 | 0.099 |
| $11/6\pi$ | 0.157 | 0.017 | 0.042 | 0.185 | 0.096 | 0.061 | 0.227 | 0.302 | 0.311 |
| avg | 0.164 | 0.2021 | **0.0127** | 0.0141 | 0.0838 | **0.087** | **0.2618** | **0.2285** | 0.3658 |

**(c)** Prediction error on cartesian accelerations with Transducer in $m/s^2$

| $\alpha$ | $\ddot{p}^h_{sx}$ [m/s] | $\ddot{p}^h_{sy}$ [m/s] | $\ddot{p}^h_{sz}$ [m/s] | $\ddot{p}^h_{ux}$ [m/s] | $\ddot{p}^h_{uy}$ [m/s] | $\ddot{p}^h_{uz}$ [m/s] | $\ddot{p}^h_{fx}$ [m/s] | $\ddot{p}^h_{fy}$ [m/s] | $\ddot{p}^h_{fz}$ [m/s] |
|---|---|---|---|---|---|---|---|---|---|
| $4/3\pi$ | 0.162 | 0.271 | 0.128 | 0.099 | 0.053 | 0.092 | 0.233 | 0.124 | 0.226 |
| $5/3\pi$ | 0.054 | 0.074 | 0.042 | 0.089 | 0.071 | 0.796 | 0.385 | 0.632 | 0.678 |
| $8/9\pi$ | 0.115 | 0.037 | 0.024 | 0.0104 | 0.094 | 0.061 | 0.206 | 0.362 | 0.470 |
| $10/9\pi$ | 0.178 | 0.299 | 0.247 | 0.382 | 0.118 | 0.114 | 0.128 | 0.258 | 0.528 |
| $\pi/6$ | 0.226 | 0.2 | 0.046 | 0.115 | 0.047 | 0.052 | 0.215 | 0.392 | 0.076 |
| $11/6\pi$ | 0.166 | 0.022 | 0.035 | 0.125 | 0.078 | 0.036 | 0.177 | 0.168 | 0.233 |
| avg | 0.1386 | 0.0753 | 0.137 | 0.0778 | 0.1668 | 0.219 | 0.314 | 0.371 | 0.314 |

**(d)** NN prediction error on cartesian accelerations with Transducer in $m/s^2$

**Table 4.2:** Prediction errors on velocity and acceleration

| $\alpha$ | $f_{ext,x}$ [N] | $f_{ext,y}$ [N] | $f_{ext,z}$ [N] |
|---|---|---|---|
| $4/3\pi$ | 0.225 | 0.421 | 0.016 |
| $5/3\pi$ | 1.918 | 2.502 | 1.959 |
| $8/9\pi$ | 0.843 | 1.104 | 0.875 |
| $10/9\pi$ | 0.328 | 0.229 | 0.019 |
| $\pi/6$ | 0.833 | 1.098 | 0.142 |
| $11/6\pi$ | 0.570 | 2.150 | 0.227 |
| avg | **0.705** | **1.237** | **0.5411** |

**(a)** Prediction error on Force with Transducer in Newton

| $\alpha$ | $f_{ext,x}$ [N] | $f_{ext,y}$ [N] | $f_{ext,z}$ [N] |
|---|---|---|---|
| $4/3\pi$ | 1.091 | 11.230 | 4.402 |
| $5/3\pi$ | 9.851 | 10.728 | 5.038 |
| $8/9\pi$ | 6.860 | 15.795 | 5.484 |
| $10/9\pi$ | 12.634 | 3.925 | 7.154 |
| $\pi/6$ | 14.166 | 3.813 | 1.484 |
| $11/6\pi$ | 5.995 | 1.470 | 1.006 |
| avg | 9.962 | 7.764 | 3.676 |

**(b)** Prediction error on Force without Transducer in Newton

**Table 4.3:** Prediction errors on contact forces

# Chapter 5

# Conclusions

In this thesis, we have introduced a transformed-based architecture designed to predict kinematic human data and interaction forces within the context of physical human-robot interactions.

Our approach consists of two primary components: an MLP Transducer responsible for estimating contact forces based on kinematic data obtained from a mocap suit, and a Transformer that forecasts both kinematic and force parameters over a future time horizon.

In order to train our network we collected 320 samples of interaction between a human operator and a robotic manipulator. The kinematic data from the humans were collected through the Xsens motion capture suit. On the manipulator, an impedance controller was implemented to ensure safe collaboration between the parties.

When we compared our model to a purely Transformer-based network, our approach exhibited similar performance in predicting kinematic data and a notable improvement in the accuracy of contact force predictions.

Future research directions will focus on the following aspects:

1. Incorporating variable stiffness into the robot to enhance adaptability. In fact, the impedance-controlled model we presented maintained constant values of stiffness throughout the interaction. However, to achieve more effective results in human-robot interaction, a model with variable stiffness would be more appropriate for various tasks, allowing for a closer emulation of human-human interaction behavior.

2. Exploring additional directions of motion, including out-of-plane motions, to broaden the applicability of our approach. Currently, our model has only been tested on movements confined to the x-y plane. In the future, we aim to train our network on a more comprehensive dataset to encompass a complete range of possible human motions.

3. Testing the network on another robotic arm in conditions where a Force/Torque sensor or residual methods are not applicable. This technique could be integrated using the technique of Transfer Learning.

4. Integrating our proposed method with control strategies that account for future human behavior in robot motion planning. This would enable the design of a controller that allows the robot to react and assist a human in a meaningful way based on the computed predictions, as prescribed in the Shared Autonomy paradigm.

# Bibliography


[1] Mario Selvaggio, Marco Cognetti, Stefanos Nikolaidis, Serena Ivaldi, and Bruno Siciliano. «Autonomy in Physical Human-Robot Interaction: A Brief Survey». In: *IEEE Robotics and Automation Letters* 6.4 (2021), pp. 7989–7996. DOI: 10.1109/LRA.2021.3100603 (cit. on pp. 1, 5, 23).

[2] Marko Ackermann and Antonie J Van den Bogert. «Optimality principles for model-based prediction of human gait». In: *Journal of biomechanics* 43.6 (2010), pp. 1055–1060 (cit. on p. 2).

[3] Hongyi Liu and Lihui Wang. «Human motion prediction for human-robot collaboration». In: *Journal of Manufacturing Systems* 44 (2017). Special Issue on Latest advancements in manufacturing systems at NAMRC 45, pp. 287–294. ISSN: 0278-6125. DOI: https://doi.org/10.1016/j.jmsy.2017.04.009 (cit. on pp. 2, 22).

[4] Philipp Kratzer, Marc Toussaint, and Jim Mainprice. «Prediction of Human Full-Body Movements with Motion Optimization and Recurrent Neural Networks». In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. 2020, pp. 1792–1798. DOI: 10.1109/ICRA40945.2020.9197290 (cit. on p. 2).

[5] A. De Luca and R. Mattone. «Sensorless Robot Collision Detection and Hybrid Force/Motion Control». In: *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*. 2005, pp. 999–1004. DOI: 10.1109/ROBOT.2005.1570247 (cit. on pp. 2, 18).

[6] Dae-Kwan Ko, Kang-Won Lee, Dong Han Lee, and Soo-Chul Lim. «Vision-Based Interaction Force Estimation for Robot Grip Motion without Tactile/-Force Sensor». In: *Expert Syst. Appl.* 211.C (Jan. 2023). ISSN: 0957-4174. DOI: 10.1016/j.eswa.2022.118441. URL: https://doi.org/10.1016/j.eswa.2022.118441 (cit. on pp. 2, 23).

[7] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. «Attention is all you need». In: *Advances in neural information processing systems* 30 (2017) (cit. on pp. 3, 13).

[8] M. Onat Topal, Anil Bas, and Imke van Heerden. «Exploring Transformers in Natural Language Generation: GPT, BERT, and XLNet». In: *arXiv preprint* (2021). URL: https://arxiv.org/abs/2102.08036 (cit. on p. 3).

[9] Carlos E Garcia, David M Prett, and Manfred Morari. «Model predictive control: Theory and practice—A survey». In: *Automatica* 25.3 (1989), pp. 335–348 (cit. on p. 3).

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016 (cit. on p. 7).

[11] Neil Gershenfeld and Andreas Weigend. *Time Series Prediction: Forecasting The Future And Understanding The Past*. Westview Press, 2018 (cit. on p. 16).

[12] Ailing, Muxi Zeng, Lei Chen, Qiang Zhang, and Xu. «Are Transformers Effective for Time Series Forecasting?» In: *arXiv preprint arXiv:2205.13504* (2022). DOI: 10.48550/arXiv.2205.13504. arXiv: 2205.13504 [cs.AI]. URL: https://arxiv.org/abs/2205.13504 (cit. on p. 17).

[13] Haoyi, Shanghang Zhou, Jieqi Zhang, Peng, Jianxin Shuai Zhang, Hui Li, Wancai Xiong, and Zhang. «Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting». In: *arXiv preprint arXiv:2012.07436* (2022). DOI: 10.48550/arXiv.2012.07436. arXiv: 2012.07436 [cs.LG]. URL: https://arxiv.org/abs/2012.07436 (cit. on p. 18).

[14] Natasha and Klingenbrunn. *Transformers for Time-series Forecasting*. Accessed on Date (e.g., October 5, 2023). 2021. URL: https://medium.com/mlearning-ai/transformer-implementation-for-time-series-forecasting-a9db2db5c820 (cit. on p. 18).

[15] Emanuele Magrini, Fabrizio Flacco, and Alessandro De Luca. «Estimation of contact forces using a virtual force sensor». In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2014, pp. 2126–2133 (cit. on p. 18).

[16] Luigi Siciliano. *Modeling, Planning and Control of Robot Manipulators*. New York, NY: Springer, 2008. ISBN: 978-0-387-74314-7 (cit. on p. 18).

[17] A. De Luca, C. Siciliano, and L. Zollo. «A Framework for the Analysis and Control of Robotic Manipulators Influenced by Elastic Deformations». In: *IEEE Transactions on Robotics and Automation* 9.5 (Oct. 1993), pp. 531–541 (cit. on p. 22).

[18] Matthias Luber, Johannes A. Stork, Gian Diego Tipaldi, and Kai O. Arras. «People tracking with human motion predictions from social forces». In: *2010 IEEE International Conference on Robotics and Automation*. 2010, pp. 464–469. DOI: 10.1109/ROBOT.2010.5509779 (cit. on p. 22).

[19] Alessandro Antonucci, Gastone Pietro Rosati Papini, Paolo Bevilacqua, Luigi Palopoli, and Daniele Fontanelli. «Efficient Prediction of Human Motion for Real-Time Robotics Applications With Physics-Inspired Neural Networks». In: *IEEE Access* 10 (2022), pp. 144–157. DOI: 10.1109/ACCESS.2021.3138614 (cit. on p. 22).

[20] Zhibo Zhang, Yanjun Zhu, Rahul Rai, and David Doermann. «PIMNet: Physics-Infused Neural Network for Human Motion Prediction». In: *IEEE Robotics and Automation Letters* 7.4 (2022), pp. 8949–8955. DOI: 10.1109/LRA.2022.3188892 (cit. on p. 22).

[21] Przemyslaw A. Lasota and Julie A. Shah. «A multiple-predictor approach to human motion prediction». In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. 2017, pp. 2300–2307. DOI: 10.1109/ICRA.2017.7989265 (cit. on p. 22).

[22] Lucian Balan and Gary M. Bone. «Real-time 3D Collision Avoidance Method for Safe Human and Robot Coexistence». In: *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2006, pp. 276–282. DOI: 10.1109/IROS.2006.282068 (cit. on p. 22).

[23] Yingxin Huo, Xiang Li, Xuan Zhang, and Dong Sun. «Intention-driven variable impedance control for physical human-robot interaction». In: *2021 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*. IEEE. 2021, pp. 1220–1225 (cit. on p. 22).

[24] Wansoo Kim, Jinoh Lee, Luka Peternel, Nikos Tsagarakis, and Arash Ajoudani. «Anticipatory Robot Assistance for the Prevention of Human Static Joint Overloading in Human–Robot Collaboration». In: *IEEE Robotics and Automation Letters* 3.1 (2018), pp. 68–75. DOI: 10.1109/LRA.2017.2729666 (cit. on p. 22).

[25] Stefanos Nikolaidis, David Hsu, and Siddhartha Srinivasa. «Human-robot mutual adaptation in collaborative tasks: Models and experiments». In: *The International Journal of Robotics Research* 36.5-7 (2017), pp. 618–634 (cit. on p. 22).

[26] Yanan Li, Keng Peng Tee, Rui Yan, Wei Liang Chan, and Yan Wu. «A framework of human–robot coordination based on game theory and policy iteration». In: *IEEE Transactions on Robotics* 32.6 (2016), pp. 1408–1418 (cit. on p. 22).

[27] Lauren Milliken and Geoffrey A Hollinger. «Modeling user expertise for choosing levels of shared autonomy». In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 2285–2291 (cit. on p. 22).

[28] Luka Peternel, Nikos Tsagarakis, Darwin Caldwell, and Arash Ajoudani. «Robot adaptation to human physical fatigue in human–robot co-manipulation». In: *Autonomous Robots* 42 (2018), pp. 1011–1021 (cit. on p. 22).

[29] PJ Hacksel and Septimiu E Salcudean. «Estimation of environment forces and rigid-body velocities using observers». In: *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*. IEEE. 1994, pp. 931–936 (cit. on p. 23).

[30] H. Su, W. Qi, Z. Li, Z. Chen, G. Ferrigno, and E. De Momi. «Deep Neural Network Approach in EMG-Based Force Estimation for Human–Robot Interaction». In: *IEEE Transactions on Artificial Intelligence* 2.05 (Jan. 2021), pp. 404–412. DOI: 10.1109/TAI.2021.3066565 (cit. on p. 23).

[31] Ehsan Noohi, Miloš Žefran, and James L Patton. «A model for human–human collaborative object manipulation and its application to human–robot interaction». In: *IEEE transactions on robotics* 32.4 (2016), pp. 880–896 (cit. on p. 23).

[32] Harsh Maithani, Juan Antonio Corrales Ramon, and Youcef Mezouar. «Predicting human intent for cooperative physical human-robot interaction tasks». In: *2019 IEEE 15th International Conference on Control and Automation (ICCA)*. IEEE. 2019, pp. 1523–1528 (cit. on p. 23).

[33] Xinbo Yu, Wei He, Yanan Li, Chengqian Xue, Jianqiang Li, Jianxiao Zou, and Chenguang Yang. «Bayesian Estimation of Human Impedance and Motion Intention for Human–Robot Collaboration». In: *IEEE Transactions on Cybernetics* 51.4 (2021), pp. 1822–1834. DOI: 10.1109/TCYB.2019.2940276 (cit. on p. 23).

[34] Ali Ghadirzadeh, Judith Bütepage, Atsuto Maki, Danica Kragic, and Mårten Björkman. «A sensorimotor reinforcement learning framework for physical human-robot interaction». In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2016, pp. 2682–2688 (cit. on p. 23).

[35] Daniel Roetenberg, Henk Luinge, and Per Slycke. «Xsens MVN: Full 6DOF Human Motion Tracking Using Miniature Inertial Sensors». In: *Xsens Technology* (Apr. 2013). DOI: cc2ba84a4d6e06fd85ad434f5b1a8545c1cc993c (cit. on p. 24).

[36] M. Schepers, M. Giuberti, and M. Bellusci. «Xsens MVN: Consistent Tracking of Human Motion Using Inertial Sensing». In: *Xsens Technology* (Mar. 2018). DOI: 10.13140/RG.2.2.22099.07205 (cit. on p. 26).

[37] Claudio Gaz, Marco Cognetti, Alexander Oliva, Paolo Robuffo Giordano, and Alessandro De Luca. «Dynamic Identification of the Franka Emika Panda Robot With Retrieval of Feasible Parameters Using Penalty-Based Optimization». In: *IEEE Robotics and Automation Letters* 4.4 (2019), pp. 4147–4154. DOI: 10.1109/LRA.2019.2931248 (cit. on pp. 26, 28).

[38] Network Time Foundation. *NTP Documentation*. 2022. URL: https://www.ntp.org/documentation.html (cit. on p. 33).

[39] Matt W Gardner and SR Dorling. «Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences». In: *Atmospheric environment* 32.14-15 (1998), pp. 2627–2636 (cit. on p. 35).

[40] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. «Language models are unsupervised multitask learners». In: *OpenAI blog* 1.8 (2019), p. 9 (cit. on p. 36).

[41] Jacob, Ming-Wei Devlin, Kenton Chang, Kristina Lee, and Toutanova. «BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding». In: *arXiv preprint* (Year). DOI: 10.48550/arXiv.1810.04805. arXiv: 1810.04805 [cs.CL]. URL: https://arxiv.org/abs/1810.04805 (cit. on p. 36).

[42] Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Jordan Matheny, Alexey Radul, Shivaram Venkataraman, Martin Raison, and Vishal Senthil. *Ray Tune: A Distributed Hyperparameter Optimization Library*. Year. URL: https://docs.ray.io/en/master/tune.html (cit. on p. 38).

[43] Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. «Scheduled sampling for sequence prediction with recurrent neural networks». In: *Advances in neural information processing systems* 28 (2015) (cit. on p. 38).