UKRAINIAN CATHOLIC UNIVERSITY

MASTER THESIS

Real-time simulation of arm and hand dynamics using SNN

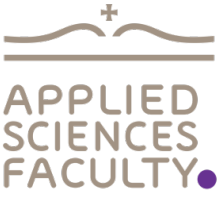
Author:
Andrii KAPATSYN

Supervisor:
Dr. Sergiy YAKOVENKO

A thesis submitted in fulfillment of the requirements for the degree of Master of Science

in the

Department of Computer Sciences
Faculty of Applied Sciences



Lviv 2024

Declaration of Authorship

I, Andrii KAPATSYN, declare that this thesis titled, "Real-time simulation of arm and hand dynamics using SNN" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

UKRAINIAN CATHOLIC UNIVERSITY

Faculty of Applied Sciences

Master of Science

Real-time simulation of arm and hand dynamics using SNN

by Andrii KAPATSYN

Abstract

The nervous system solves the complex problem of body dynamics in real-time, both during the planning and execution of movements. The resulting neural commands are expressed as spike trains formed by the interaction of the feed-forward and feedback pathways that encode limb posture and movement. While body biomechanics and neural recordings have been well characterized at different levels in the nervous system, we have no integrated view of this process.

In this study, we propose a feed-forward Spiking Neural Network (SNN) to tackle the inverse and forward dynamics problems for an arm and hand model. We designed human hand and arm models with 1 and 3 degrees of freedom (DoF) to simulate movements and developed corresponding SNNs. These SNNs predict joint angles, velocities, and accelerations based on the joint torques for the forward dynamics problem, and predict torques based on joint angles for the inverse dynamics problem.

We evaluated our models by comparing them with ideal controllers derived from kinematic equations for kinematic simulation and employed the state-of-the-art MuJoCo physical engine for dynamics simulation. Our 3-DoF model for dynamics simulation operates in real-time with a latency of 0.44 ms and achieves a mean absolute error of 0.0545 m. This performance demonstrates potential for integration with brain-computer interfaces for neuroprosthetics.

Acknowledgements

I'm grateful to Dr. Sergiy Yakovenko for his guidance and support throughout this research. I was fortunate to have a supervisor who calmly guided me through the challenges of this research and provided me with the tools to overcome them. I appreciate the detailed explanations from Serhii Bahdasariants, who helped me to understand the concepts of neuroscience and spiking neural networks better. Also, I want to thank Yurii Prima for his well-documented work on the Nengo framework, which was a valuable source of advanced techniques for my research.

I extend my gratitude to the Ukrainian Catholic University and the Faculty of Applied Sciences for their exemplary organization of the Master's Program in Data Science, which has greatly enriched my educational journey. My heartfelt thanks also go to Oleksii Molchanovsky and Ruslan Partsey for their outstanding leadership and their exceptional skill in managing organizational aspects of the program. I appreciate the time with my classmates, who have inspired and motivated me throughout this program.

I cannot conclude without expressing my heartfelt gratitude to my family and friends, whose unwavering support and encouragement have been my constant source of strength and inspiration. Their endless love and belief in me have been invaluable throughout this journey.

Contents

List of Figures

List of Tables

List of Abbreviations

SNN	**S**piking **N**eural **N**etwork
DoF	**D**egrees **o**f **F**reedom
BCI	**B**rain-**C**omputer **I**nterface
NEL	**N**eural **E**ngineering **L**ab
WVU	**W**est **V**irginia **U**niversity
EMG	**E**lectro**m**yo**g**raphy
EEG	**E**lectro**e**ncephalo**g**raphy
ECoG	**E**lectro**c**orticog**g**raphy
MRI	**M**agnetic **R**esonance **I**maging
fMRI	**F**unctional **M**agnetic **R**esonance **I**maging
RNN	**R**ecurrent **N**eural **N**etwork
LSTM	**L**ong **S**hort-**T**erm **M**emory
GRU	**G**ated **R**ecurrent **U**nit
IDM	**I**nverse **D**ynamics **M**odel
FDM	**F**orward **D**ynamics **M**odel
LIF	**L**eaky **I**ntegrate and **F**ire

List of Symbols

q	joint positions	(rad)
$\dot{q}$	joint velocities	$(\frac{\text{rad}}{\text{s}})$
$\ddot{q}$	joint velocities	$(\frac{\text{rad}}{\text{s}^2})$
τ	joint torques	(N m)
t	time	(s)
F_g	gravitational force	(N)
F_{fr}	joint friction force	(N)
I	moment of inertia	(kg m^2)
g	gravitational constant	$(9.81\ \frac{\text{m}}{\text{s}^2})$
m	links mass	(kg)
l	links length	(m)
r	links radius	(m)
u_x	task-space force	(N)

To the heroes of AFU and their families.

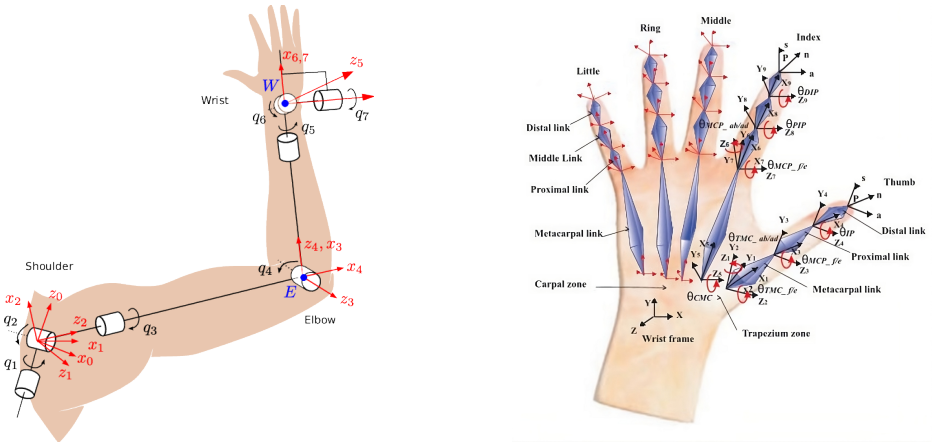
Chapter 1

Introduction

1.1 Motivation

The realistic musculoskeletal model of hand and arm consists of about 27 degrees of freedom (DOF) actuated by 52 force-generating musculotendon units scaled by limb segment geometry (Agur, Dalley, and Grant, 2008). This complexity is a challenge for real-time simulations, for example, for brain-computer interfaces (BCI), and can benefit from approximations of musculoskeletal transformations and limb physics (Sobinov et al., 2020). Neuroprosthetics require similar computations for the decoding of intent and encoding of sensory feedback. Previously, Manukian, Bahdasariants, and Yakovenko, 2023 have shown that the inverse dynamics problem can be solved by a relatively shallow network expressed as RNN and LSTM. But the real-time accurate and stable solution of the forward dynamics problem remains an open question, since it involves predicting the motion of a system based solely on the forces and torques applied, without prior knowledge of the resulting movement. This requires solving complex differential equations that describe the system's dynamics, which can be computationally intensive and sensitive to the initial conditions and system parameters of the complex mechanical linkage along the chains forming the multi-body system (Leeuwen, Aerts, and Otten, 2003).

Addressing the stated challenge holds the potential for future advances in prosthetic limb control. We could improve the robustness of human-machine interactions by solving limb dynamics problem with the help of Spiking Neural Networks (SNN) formulation. Ultimately, optimizing the efficiency and responsiveness of prosthetic during daily tasks can significantly enhance the quality of life for amputees.



(A) Zanchettin et al., 2010 (B) Jaworski and Karpiński, 2017

FIGURE 1.1: Kinematic models of the human arm (A) and hand (B).

1.2 Research Goals

The goal of this project is to develop a feed-forward Spiking Neural Network (SNN) that will be trained, based on the rigid-body dynamics equations (Featherstone, 2008) to predict the arm and hand position in response to the input from the musculoskeletal model (forward dynamics). We will use the musculoskeletal model created by Neural Engineering Lab (NEL) at West Virginia University (WVU) to provide the input (inverse dynamics feedback) as well as the target position data for the SNN training. The SNN will be implemented using the Nengo framework proposed by Bekolay et al., 2014, that takes advantage of the Neural Engineering Framework (NEF) to build and simulate large-scale neural models.

We anticipate that the neurons within the SNN will demonstrate directional tuning analogous to that observed in biological counterparts. Furthermore, it is expected that the motor transformations facilitated by the SNN will maintain robustness in the presence of sensorimotor noise. Additionally, we project that the SNN will offer a more computationally and energy-efficient solution, particularly for applications in wearable technologies.

1.3 Structure of Master Thesis

The remainder of the thesis is structured as follows. In Chapter 2 we provide an overview of the related work in the field of inverse and forward dynamics problems and perspective of using SNN for solving these problems. In Chapter 3 we present our methodology and research approach, including the experiment setup and evaluation details. Chapter 4 contains the details of the conducted experiments and the results obtained. Finally, we provide conclusions and outline future work directions in Chapter 5.

Chapter 2

Related Work

2.1 Arm and Hand Dynamics Simulation

The problem of arm and hand dynamics simulation can be divided into two subproblems. The first one is the collection of signals from the human body to decode the limb movements or its intentions. The second one is the prediction of the applied joint torques (inverse dynamics problem) to control the arm and hand model as well as joint angles, joint velocities and accelerations (forward dynamics problem).

For the limb movement decoding problem, electromyography (EMG), electroencephalography (EEG), electrocorticography (ECoG), magnetic resonance imaging (MRI), and functional MRI (fMRI) signals are used (Tiwari et al., 2018). The most popular are EMG and EEG sensors for this task since they are relatively non-invasive and can be easily integrated into wearable devices for real-time applications, making them practical choices for limb movement decoding studies (Yoshimura et al., 2017).

Some studies propose using bioelectrical signals directly to solve the inverse dynamics problem (Ren et al., 2019). However, using those signals for such a complex task is challenging and requires thorough data processing and feature extraction approaches (Olmo and Domingo, 2020; Garg et al., 2021). But, the bioelectrical signals are often used to solve simpler limb movement decoding problems such as gesture recognition (Garg et al., 2021; Gautam et al., 2020; Ceolini et al., 2020).

Also, researchers suggest using systems for the musculoskeletal model dynamics simulations of movement that already implemented inverse and forward dynamics simulation and supported work with the bioelectrical data Pizzolato et al., 2016; Yough et al., 2021, such as OpenSim[1] or Matlab Simulink[2]. Such musculoskeletal models can be used for data generation to be passed to high-level algorithms to solve inverse dynamics or forward dynamics problems. These models significantly simplifies task, allowing researches to focus on solving directly inverse dynamics problem (Manukian, Bahdasariants, and Yakovenko, 2023; Porsa, Lin, and Pandy, 2015) or forward dynamics problem (Kvrgic and Vidakovic, 2020).

2.2 Inverse Dynamics and Forward Dynamics Problems

In the Manukian, Bahdasariants, and Yakovenko, 2023 work, three different architectures were considered to solve the inverse dynamics problem: RNN, GRU and LSTM. Authors showed that a single-layer LSTM could maintain prediction accuracy of 0.1 Nm with only 20 ms input sequences. Authors have proven that the proposed approach of using ANN is suitable for real-time applications for predicting kinetics in complex multijoint systems. Despite the low latency and great prediction

[1] https://opensim.stanford.edu/

[2] https://www.mathworks.com/products/simulink.html

accuracy was achieved, the proposed approach has some limitations. The limited exploration of behavioral space and training dataset specification are weak spots of approaches that uses ANN. Such a model can't accurately solve inverse dynamics for the unseen behavior, such as object interactions or pathologies like tremor.

The Polydoros, Nalpantidis, and Kruger, 2015 work proposes the online learning based approach, that uses a reservoir layer for the deep learning model to serve as a memory and Bayesian linear regression as a learning rule to optimize the model weights. Results show that the proposed algorithm can adapt to real-time changes of the inverse dynamics models significantly better than other ANN models, such as Manukian, Bahdasariants, and Yakovenko, 2023. The proposed model presents a limitation due to its computationally intensive nature. Although the authors state about real-time applications of the model, it models only 6 DoF robotic manipulator arm.

The inverse dynamics are in fact easier to compute than the forward dynamics, because the optimization problem becomes diagonal and decomposes into independent optimization problems over individual contacts - which can be solved analytically, as shown by Todorov, 2014, which describes the implementation details of MuJoCo. MuJoCo[3] stands for **Mu**lti-**Jo**int dynamics with **Co**ntact. It is a physics engine that is widely used in robotics and biomechanics research and represents the state-of-the-art implementation of the dynamics simulations. Todorov, 2014 presents real-time performance for the both inverse and forward dynamics problems. The forward dynamics is slower by an order of magnitude but still fast enough to run in real-time on single core of a desktop processor (see Figure 2.1) using the projected Gauss-Siedel method (PGS). The recursive Newton-Euler algorithms (RNEA) is used for the forward dynamics problem and considered to be the most efficient algorithm (Todorov, 2014; Kvrgic and Vidakovic, 2020).

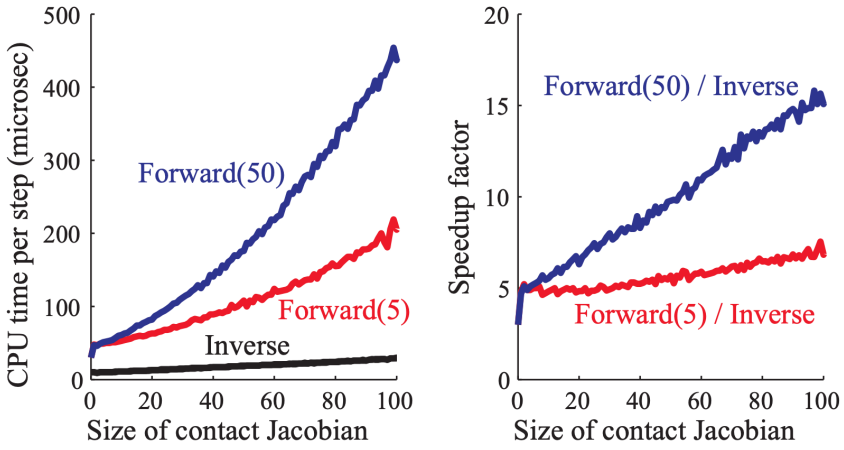


FIGURE 2.1: Comparison of the speed of the forward and inverse dynamics computations. Left: CPU time per simulation step is shown for different sizes of the contact Jacobian. The PGS iterative solver in the forward dynamics was run for 5 or 50 iterations. Right: the speedup factor is the ratio of forward dynamics CPU time to inverse dynamics CPU time. Todorov, 2014.

[3] https://mujoco.org/

2.3 Spiking Neural Networks Perspective

Spiking Neural Networks aims to mimic the remarkable properties of the human brain, such as online learning, massive parallelism, low power consumption, and analog computations. Similarly to human brains, SNNs don't use all neurons simultaneously, but only regions needed for the current tasks (Stone, 2018). Thus, it is possible to create low energy hardware based on the property that information is sparse in time and concentrated in spikes (Rueckauer et al., 2017; Pande et al., 2013). It is one of the biggest advantages of SNNs. Also, there are some disadvantages for SNNs. Because spikes signals are sparse and not differentiable, we can not apply backpropagation to train a model. There are three common ways for SNN to learn: unsupervised learning such as spike timing-dependent plasticity (STDP); indirect supervised learning such as ANNs-to-SNNs conversion; direct supervised learning such as gradient descent-based backpropagation. Another option to train SNN is to use Neural Engineering Framework, which is expalined in details in the section 3.2.

Most of the researches on the inverse and forward dynamics problem solved by SNN focused on the robotics domain to control the robotic arm. The primary motivation for using SNNs is to reduce the power consumption of embedded devices required to run the robot DeWolf, Jaworski, and Eliasmith, 2020; Hwu et al., 2016. The work of DeWolf et al., 2016 presents the recurrent error-driven adaptive control hierarchy (REACH) model. This model is neurally plausible and implements the fundamental biological process of dynamic adaptation for limb control similar to the one in the cerebellum. As well as Polydoros, Nalpantidis, and Kruger, 2015 work, this one implements the adaptive learning rule and online-learning approach. The biggest limitation of this work is that the model is significantly less complex than any of the reviewed previously. Authors presented only 3 DoF model and didn't discuss speed of the solution or the real-time constraints.

Chapter 3

Methodology and Research Approach

3.1 Problem Statement

The arm and hand dynamics simulation consists of two models that should compensate for each other. The Inverse Dynamic Model (IDM) predicts joint torques given the current body state and the desired position in the task space. The Forward Dynamic Model (FDM) predicts the next arm state by given joint torques (Featherstone, 2008).

The FDM accounts for gravitational force F_g and the moment of inertia I, and some constant joint friction force F_{fr} to update corresponding joint angles, velocities, and accelerations $q, \dot{q}, \ddot{q}$ by given applied joint torques τ for every degree of freedom. The moment of inertia I and the gravitational force F_g for the FDM are calculated by equations 3.1 and 3.2, where g is the gravitational constant, m is the vector of links mass, l is the vector of links length, and r is the vector of links radius.

$$I = \frac{ml^2}{12} + \frac{mr^2}{4} \tag{3.1}$$

$$F_g = sin(q) \cdot m \cdot g \cdot \frac{l}{2} \tag{3.2}$$

The corresponding model state variables are calculated as follows:

$$\ddot{q} = \frac{\tau - F_g - F_{fr}}{I} \tag{3.3}$$

$$\dot{q} = \int \ddot{q}\, dt \tag{3.4}$$

$$q = \int \dot{q}\, dt \tag{3.5}$$

The IDM model uses a Jacobian matrix J that relates the movement of the joint angles q to the movement of the task space position x. The Jacobian is a function of q, so it can represent a complex relationship, including the joint movement constraints as described by Featherstone, 2008. The Jacobian also defines an approximate relationship between forces in the task space and joint space (Place, 2017). Also, it should estimate the inertia matrix in task space M_x, allowing the effects of inertia to be canceled out (Khatib, 1987).

The force to correct the position error is calculated in the task-space as

$$u_x = k_p(x' - x), \tag{3.6}$$

where x is the hand's current position, x' is the desired position, and k_p is the position gain term. Also, the IDM should account for the velocity error term $k_v I\dot{q}$ and corresponding gravity compensation term g_q, so the final control signal equation, which is the joint torques applied to the FDM, is calculated as follows:

$$u = J^T M_x u_x - k_v I\dot{q} - g_q, \tag{3.7}$$

3.2 SNN Simulation Toolkit

Neural Engineering Object (Nengo) is a graphical and scripting software for simulating large-scale neural systems. As neural network software, Nengo is a tool for modeling neural networks with cognitive science, psychology, artificial intelligence, and neuroscience applications. Nengo is based on the Neural Engineering Framework (NEF) Eliasmith and Anderson, 2002. This framework proposes three quantitatively specified principles that enable the construction of large-scale neural models. Briefly, this mathematical theory defines:

1. Representation: A population of neurons collectively represents a time-varying vector of real numbers through non-linear encoding and linear decoding.

2. Transformation: Linear and non-linear functions on those vectors are computed by linear decodings that are used to compute the connections between populations of neurons analytically.

3. Dynamics: The vectors represented by neural populations can be considered state variables in a (linear or non-linear) dynamical system, and recurrent connections can be computed using principle 2.

The summary of the NEF principles is shown in Figure 3.1. (A) According to the representation principle, neural signals are encoded within populations of neurons, each defined by its tuning curve (top). This curve indicates the level of neuronal activity in response to an input signal. When the input signal in the middle panel stimulates the eight neurons shown in the top panel, their activity generates the spike trains observed in the bottom panel. (B) The representation principle also allows for the decoding of neural population activity to retrieve the original input signal or its transformation. First, the firing pattern in the top panel is processed using a decaying exponential filter (middle panel). This filtered activity is then combined using a set of weights, approximating the input signal (bottom panel, green) and the cosine of the input signal (bottom panel, purple). (C) A sine wave encoded by population A (top panel) is transformed and projected as its negative to population B (middle panel), while its square is projected to population C (bottom panel). The transformation principle allows neurons to send signals to another population by decoding the desired function from one population and encoding it into the next. This process can be simplified into a single step by calculating connection weights between neurons in the two populations. (D) In a neurally implemented dynamical system, negative feedback between its two dimensions creates a harmonic oscillator. This oscillator's behavior is plotted over time (top) and within a state space (bottom). The dynamics principle suggests that signals represented by neural populations can be understood as state variables within a dynamical system

Based on these principles, Nengo allows to train any ensemble object, a group of neurons collectively representing a vector, by connecting it to any other object. This connection won't directly compute the function defined for that connection, but

instead, the function is approximated by solving the set of decoding weights. The output of a decoded connection is the sum of the ensemble neural activity weighted by the decoding weights solved in the build process.

Mathematically, it can be formulated as the following equation:

$$y(t) = \sum_{i=0}^{n} d_i^f a_i(x(t)), \tag{3.8}$$

where $y(t)$ is the output of the connection at time t, n is the number of neurons in the ensemble, d_i^f is the decoding weight associated with neuron i given the function f, and $a_i(x(t))$ is the activity of neuron i given $x(t)$, the input at time t.

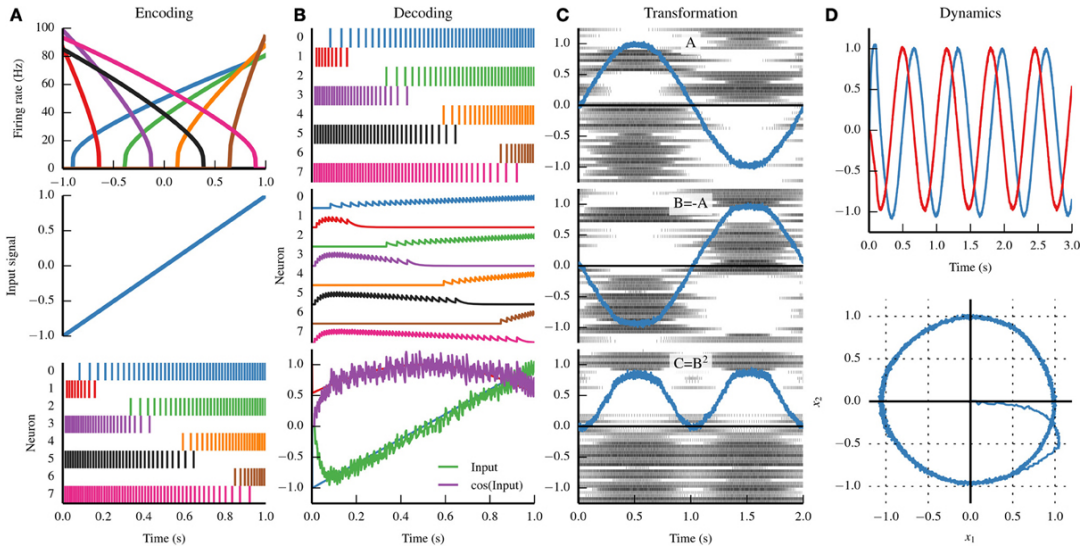


FIGURE 3.1: The visualization of the Neural Engineering Framework principles. (Bekolay et al., 2014)

These principles allow us to use the nengo and NEF as the white-box technique, an opposite to the black-box (Manukian, Bahdasariants, and Yakovenko, 2023) in terms of model definition approach (DeWolf, Jaworski, and Eliasmith, 2020). We can set connections between nengo objects to approximate the functions described in the section 3.1, to implement the inverse-forward relationship for arm and hand dynamics simulation.

We propose to use leaky integrate-and-fire (LIF) neurons (Bekolay et al., 2014) to build the SNN. In LIF neurons, the synaptic weight modulates pre-spikes which are then integrated as a current influx into the membrane potential, decaying exponentially. If the membrane potential exceeds the firing threshold, the post-neuron emits a post-spike and the membrane potential is reset (Lee et al., 2020). This model is biologically plausible and computationally efficient and implemented in the most of the SNN frameworks, including nengo.

The sub-threshold dynamics of a LIF spiking neuron can be expressed as:

$$\tau_m \frac{dV_{mem}}{dt} = -V_{mem} + I(t) \tag{3.9}$$

where V_{mem} represents the membrane potential of the post-neuron, and t_m is the time constant for the decay of the membrane potential. The input current $I(t)$ is the

weighted sum of pre-synaptic spikes at each time step, and defined as:

$$I(t) = \sum_{i=0}^{n^l}(w_i \sum_k \theta_i(t - t_k)) \tag{3.10}$$

where n^l denotes the number of pre-synaptic weights, w_i is the synaptic weight connection the i-th pre-neuron to the post-neuron, and $\theta(t - t_k)$ represents a spike event from the i-th pre-neuron at time t_k and formulated as:

$$\theta(t - t_k) = \begin{cases} 1, & \text{if } t = t_k \\ 0, & \text{otherwise} \end{cases} \tag{3.11}$$

where t_k is the time of the k-th pre-synaptic spike. Figure 3.2 illustrates the dynamics of an LIF neuron. The influence of each pre-synaptic spike $\theta_i(t - t_k)$ is modulated by the corresponding synaptic weight w_i, generating a current that flows into the post-neuron. These units do not have a bias term. The input current is integrated into the post-neuronal membrane potential V_{mem}, which decays exponentially over time with the time constant t_m. When the membrane potential exceeds the firing threshold V_{th}, the neuron fires a spike and resets its membrane potential to the initial value.

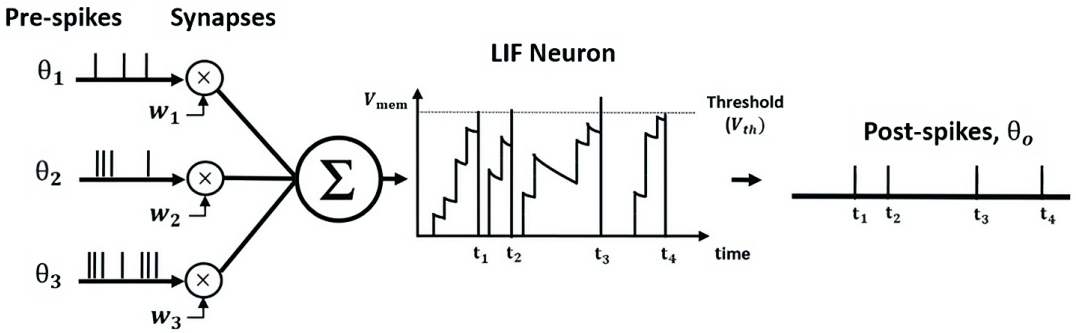


FIGURE 3.2: The illustration of LIF neuron dynamics. (Lee et al., 2020)

3.3 Experiment Setup

The arm and hand dynamics simulation can be mostly implemented using the nengo framework, as it allows to visualize the connections between objects and store and process data during the simulation in the form of graphs and custom data files.

For such a simulation, it is necessary to implement both parts of the model - forward and inverse dynamics, which will interact with each other. The input to the system will be the target end-effector position coordinates. The hand state data that represented as joint angles, joint velocities, and joint accelerations, as well as the corresponding joint torques, will be calculated directly by the corresponding parts of the model. To visualize the current arm state, a custom nengo component which allows drawing html canvas elements was used. Also, we added plot components to display graphs and neuron states during the simulation to conduct experiments. All these components allow visualizing and analyzing data during the simulation to conduct experiments. The example of the nengo environment setup is shown in Figure 3.3.

To calculate the metrics described in the section 3.4 and prepare the graphs for the results analysis, the simulation mode without creating a graphical interface was

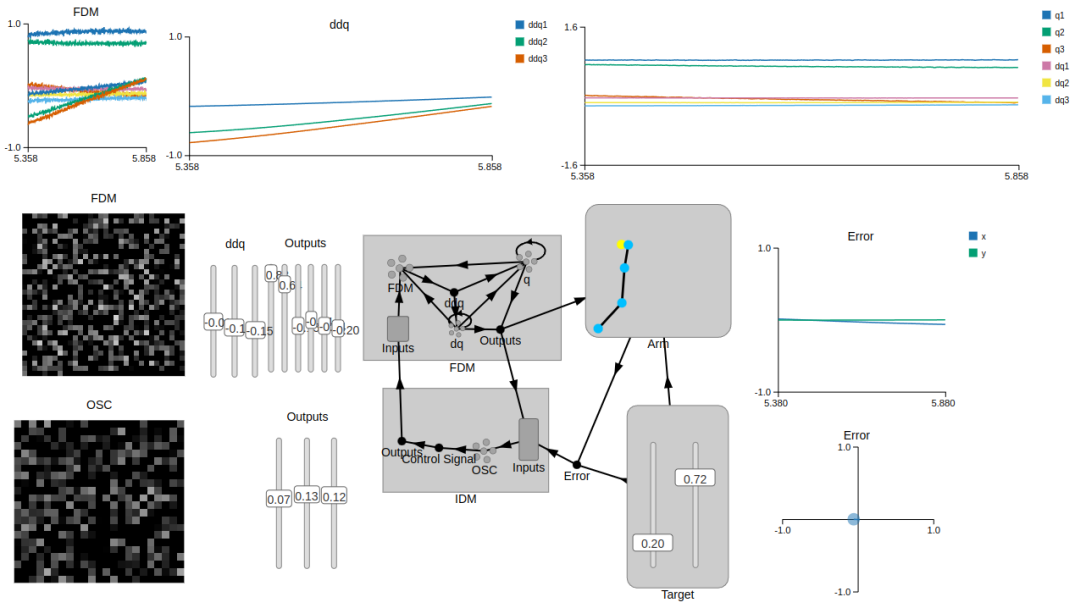


FIGURE 3.3: The example of the nengo experiment setup.

used, which allowed obtaining the results without overheads for visualization and properly compute model simulation time.

The proposed experiment setup allows conducting experiments with different arm and hand model complexity (number of degrees of freedom), and easily evaluate its performance. We suggest to start with a simple arm and hand model and then gradually increase the complexity of the model to evaluate the performance of the proposed approach and capabilities of the SNN. The further details and results of the experiments are described in the next chapter.

3.4 Metrics and Evaluation

The performance of the proposed approach can be expressed in terms of the accuracy of the predicted joint positions, velocities, and accelerations, as well as the joint torques, and the simulation time.

The accuracy of the predicted values can be calculated as the Root Mean Squared Error (RMSE) between the predicted and the ground truth values and Mean Absolute Error (MAE) of the predicted positions in the task space. The RMSE metric will allow us to compare the performance of the proposed approach with the related works (Manukian, Bahdasariants, and Yakovenko, 2023; DeWolf et al., 2016) and evaluate the IDM part of the model. The MAE metric will allow us to evaluate the FDM part of the model and compare it with the state-of-the-art physics engine, MuJoCo (Todorov, 2014). The proposed metrics are calculated as follows:

$$RMSE_\tau = \sqrt{\frac{1}{n}\sum_{i=0}^{n}(\tau_i' - \tau_i)^2} \tag{3.12}$$

$$MAE = \frac{1}{n}\sum_{i=0}^{n}|x_i' - x_i| \tag{3.13}$$

The real-time performance of the proposed approach can be evaluated using the simulation latency metric. This metric is determined by calculating the average time

difference between consecutive simulation steps. We will compare the simulation latency with the results of the corresponding MuJoCo simulation to conclude about the real-time capabilities of the proposed model.

3.5 Requirements

Defining clear requirements for the real-time Spiking Neural Network (SNN) dynamics model is crucial to ensure its efficacy and applicability in real-world scenarios. These requirements encompass the necessary accuracy, real-time constraints, and hardware specifications that the model must meet. Establishing these criteria helps in guiding the development process and evaluating the performance of the model under realistic conditions.

The model must reliably predict joint angles and torques with minimal error to ensure realistic and accurate simulations. For the inverse dynamics problem, the model should accurately predict the joint torques, with $RMSE_\tau \leq 0.1$ and the forward dynamics model should accurately predict the joint angles, with $MAE \leq 0.05$, which defines overall model capability to accurately move the arm and hand to the desired position.

The model must also operate in real-time, with a simulation latency (time of full pass for both FDM and IDM parts of the model) of less than 1 ms with a time step $\Delta t = 0.001$. This ensures that the model can update its predictions and adapt to new inputs almost instantaneously, providing a smooth and responsive user experience.

The real-time performance evaluation were conducted on the MacBook M1 Pro 2021 with 8 CPU cores and 16 GB of RAM. We haven't experimented with performance of the proposed model on the low-energy FPGA[4] or neuromorphic Loihi[5] devices, which is out of the scope of this work, and could be considered as a future work direction.

[4] https://www.nengo.ai/nengo-fpga/appendix.html
[5] https://en.wikichip.org/wiki/intel/loihi

Chapter 4

Experiments Implementation and Results

4.1 Neural Integrator Implementation

The integrator is an essential component of the IDM part of the model, as it should store the previous state of the system and update it accordingly to the accelerations produced by joint torques. In the SNN formalism, the integrator can be implemented as the recurrent connection between the ensemble of neurons. The goal of this experiment to show how to implement neural integrator and compare its response to the ideal integrator. The scheme of the integrator network is shown in Figure 4.2. From the Figure 4.1 we can see that the neural integrator is not perfectly follows the ideal integrator and has some noise in the response. The noisy response of neurons can be addressed by adding the low-pass filter to the connection outputs. The drift of the integrator values can be solved by increasing the number of neurons in the ensemble or by changing fire rates of neurons.

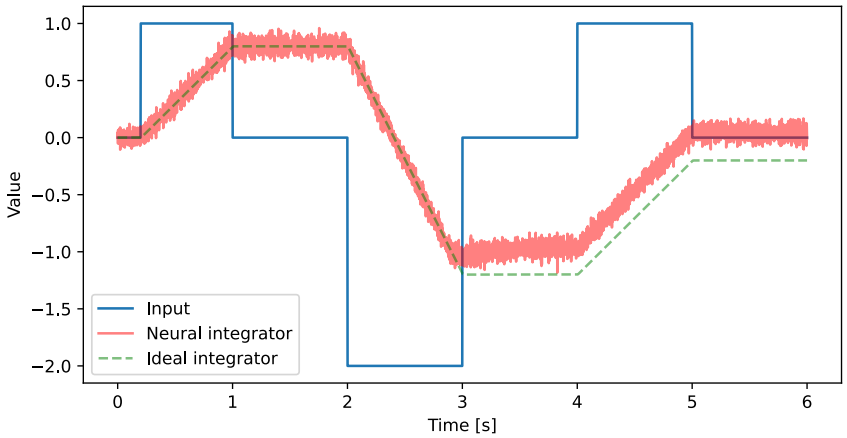


FIGURE 4.1: Integrator response comparison

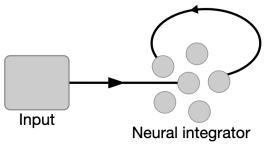


FIGURE 4.2: Scheme of the neural integrator network.

4.2 Model Parameters Selection

During the SNN models implementation, it's important to consider proper configuration of the neurons populations (Ensemble) and it's connections parameters for every component of the model.

For the **Ensemble**, the number of neurons, firing rates, number of dimensions, and radius should be considered.

The **number of neurons** should be selected based on the complexity of the task and number of dimensions of the input data and is selected experimentally. It's recommended to start with some small number of neurons and gradually increase it to find the optimal number of neurons for the task.

The **firing rates** of neurons affects the number of spikes generated by the neurons and the response time of the ensemble. The neurons population with higher firing rates will be able to generate smoother and more accurate approximation. This is shown in the Figure 4.3, where two populations of neurons with firing rates of 10 Hz and 100 Hz are compared.

The **number of dimensions** is selected based on the number of the input dimensions that should be passed to the ensemble.

The **radius** scales the range of input values that the ensemble can represent. By default, this range is within a unit hypersphere. If this default value is maintained, the neural activity will saturate for input vectors with magnitudes exceeding 1, resulting in inaccurate vector representations and function approximations. While this issue is less significant in lower-dimensional spaces, it becomes problematic as the dimensionality of the state space increases, since input vectors with norms greater than 1 become more frequent. Ideally, we want to represent vectors of any dimension, where each element ranges between -1 and 1. To achieve this, we calculate the norm of a unit vector of size D, given by $\sqrt{D}$, which is the magnitude of a vector of size D with all elements set to one, where D is the number of dimensions.

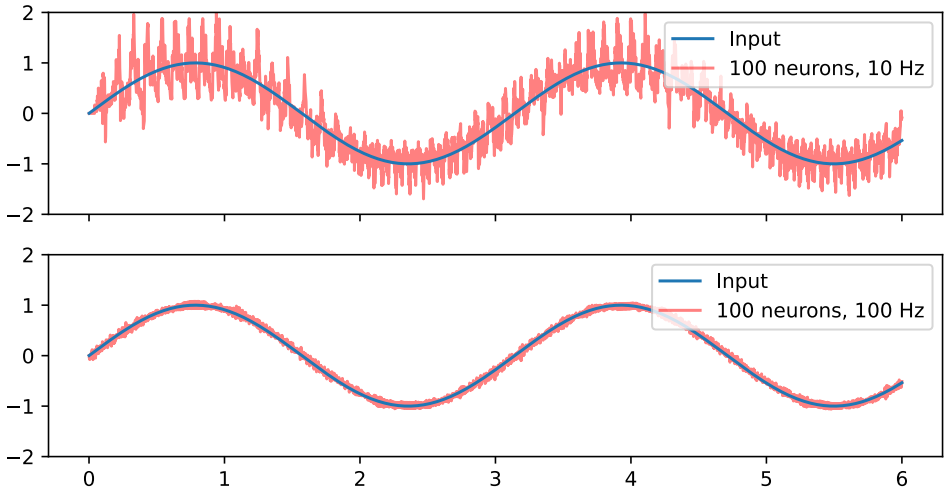


FIGURE 4.3: Two populations of neurons that represents a sine wave with firing rates of 10 Hz and 100 Hz

The **Connection** parameters that should be configured are the synapse time, transform and function.

The **synapse time** defines the time constant of the low-pass filter applied to the connection output. It should be selected experimentally to reduce the noise in the connection output.

The **transform** defines the scale the output of the connection, usually, we don't need to apply any transformations but for some cases we need to scale the outputs to achieve the desired behavior for the connection. For instance, we can scale the output of the connection to the integrator to implement the change of the state for the integrator over time.

The **function** that should be approximated by the connection should be defined for the NEF framework as it was described in the section 3.2.

4.2.1 Input preprocessing

The input vector to the neurons in the Ensemble should range from -1 to 1. To ensure this, we added additional input and output nodes to preprocess and postprocess the data appropriately. These nodes adjust the data by subtracting the mean and scaling it by three times the standard deviation, ensuring that the most of the values fall within the required range. An extra input node scales the input data and connects it to the corresponding Ensemble, and the Ensemble connects to an output node that scales the data back and approximates the desired function.

To compute the appropriate mean and standard deviation values, we used two approaches. For the joint torques q in our experiment setup we have calculated analytically the mean and the scaling factor, since the q values can be in the range of $[0, \pi/2]$. We selected the mean value and scaling factor equal to $\pi/4$. For all the other input data, we recorded the corresponding values during the simulation on the perfect controller via replacing the Ensembles by the Nodes that calculated the desired functions directly. For the simulation, we used randomly generated samples of joint positions and velocities as well as the target end-effector positions as the starting state, and we recorded the simulation until the arm and hand model reached the target.

4.2.2 Intercepts configuration

Another important property of the neurons population is the intercepts. The intercepts determine how much of state space a neuron fires for. By default, the intercepts in the NEF are uniformly distributed between -1 and 1, and it works well for 1-dimensional inputs. It means that a neuron with an intercept of 0 is active for 50% of the state space, and a neuron with an intercept of 0.5 is active for 25% of the space. But, even for the 2-dimensions, an intercept of 0.5 will be active for less than 20% of the state space. In the higher dimensions, the most of the neurons appears to be useless because they are not active for the most of the state space. To address this issue the area distribution concept was proposed by Dr. Terrence C. Stewart[6]. The main idea is to analytically find the proportion of the state space that a neuron should be active for and then find an inverse (intercepts) that gives that proportion. As result, we can achieve the more efficient use of the neurons in the Ensemble and reduce the number of neurons needed to represent the input data for the high-dimensional inputs. The Figure 4.4 shows the effectiveness of the proposed approach for the intercepts distribution selection.

[6] https://github.com/tcstewar/testing_notebooks/blob/master/Intercept%20Distribution%20.ipynb

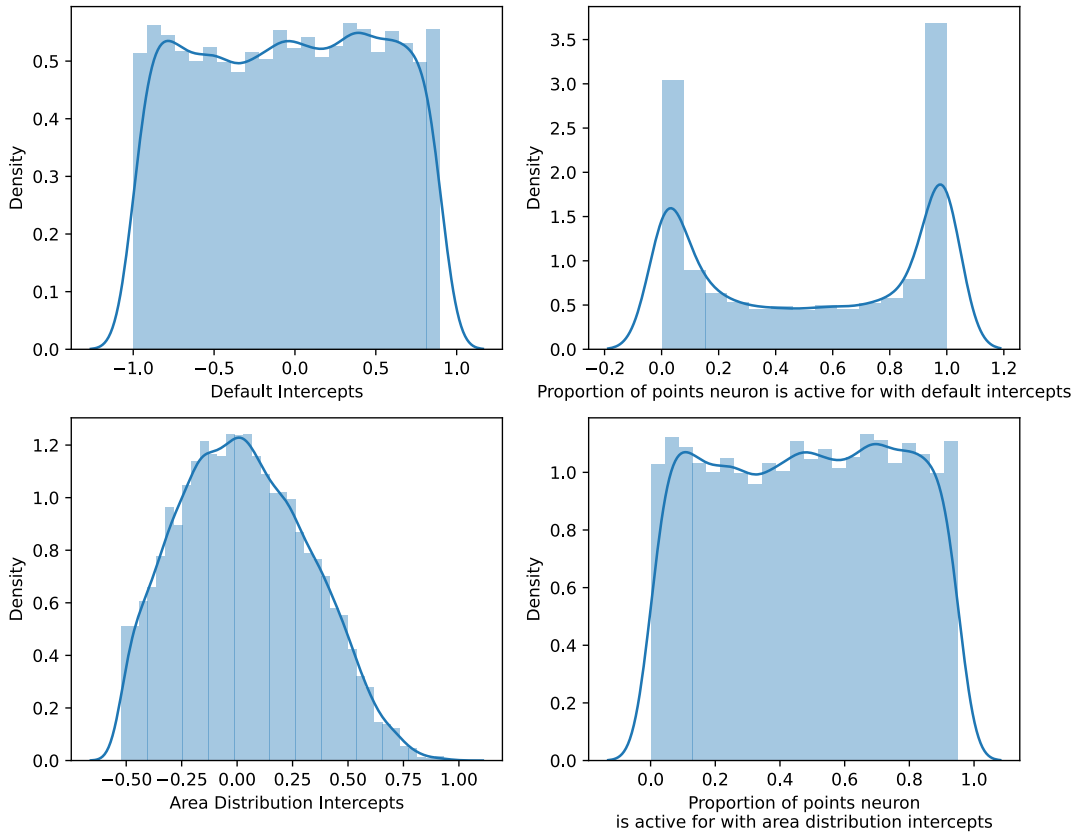


FIGURE 4.4: The density plots of the intercepts distribution for the 8-dimensional input and corresponding proportion of points neuron is active for with default approach (top) and area distribution approach (bottom).

4.3 Kinematics Model Implementation

We decided to start with the simpler kinematics problem to validate the proposed approach and develop the methodology for the further development of the dynamics model.

For the FDM, we have used the following equation instead of equation 3.3 to calculate the joint accelerations:

$$\ddot{q} = \tau, \tag{4.1}$$

and the following equation for the inverse dynamics part, instead of equation 3.7 to calculate the joint torques:

$$u = J^T u_x. \tag{4.2}$$

4.3.1 1 DoF kinematics model

The simplest arm and hand model possible is the 1 DoF model. The implementation of such a model will allow us to build the proper setup for the further experiments, since the structure of SNN model (neurons populations and its connections) will be the same or pretty similar.

The FDM part of the model contains the input node, that takes joint torques and passes it to the "FDM" ensemble, that supposed to predict the joint accelerations. The joint accelerations are then passed to the extra node that applies low pass filter

to the accelerations and them to the two separate integrators called "dq" for the joint velocities and "q" for the joint positions. The "dq" integrator is connected to the "q" integrator to calculate the joint positions and velocities. Finally, the joint positions and velocities are passed to the output node to generate the output of the FDM model.

The IDM part of the model contains the input node that takes joint angles, velocities and task space error and passes it to the "IDM" ensemble, which supposed to predict the joint torques, which then passed to the output node to generate the output of the IDM model.

The structure of the described SNN model as well as the visualization of the arm and hand model is shown in corresponding blocks in the Figure 4.5.

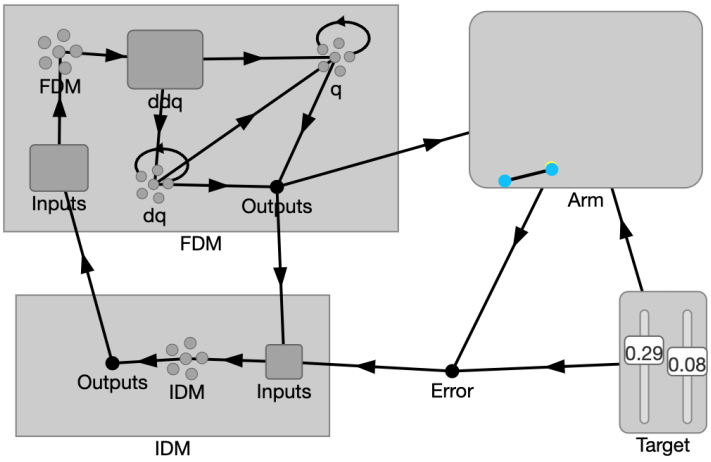


FIGURE 4.5: Visualization of the structure of the arm and hand kinematics model for 1 DoF.

During the development, we manually selected the best performing number of neurons for all the Ensembles to be 200 and the low pass filter to be 0.3. Finally, to evaluate the model performance, we executed the 10 simulations of 25 seconds with pre-defined target trajectory that follows the following rules:

$$d = sin(t \cdot s + o \cdot \pi)\frac{\pi}{4} + \frac{\pi}{4} \tag{4.3}$$

$$x = l \cdot cos(d) \tag{4.4}$$

$$y = l \cdot sin(d) \tag{4.5}$$

where t is the simulation timestep, s is the speed of the target, o is the offset of the target trajectory, and l is the total length of arm and hand model. The parameters of the target functions were set to $s = 0.2$ and $o = 1.5$. Such a trajectory allows us to evaluate the model performance and model stability for the different models. After running the simulations, we got average measures of $MAE_x = 0.011$, $RMSE_\tau = 0.0057$ and the simulation latency of 0.18 ms, which is much faster than the real-time. The corresponding simulation plots are available in Appendix A.1.

4.3.2 3 DoF kinematics model

The 3 DoF arm and hand model contains 3 joints which are shoulder, elbow and wrist and 3 links which are upper arm, forearm and hand respectively. The visualization of the 3 DoF arm and hand model is shown in the Figure 4.6.

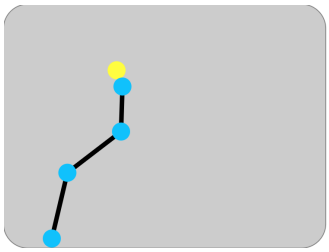


FIGURE 4.6: Visualization of the 3 DoF arm and hand model.

The parameters of the neural network were selected to be the same as for the 1 DoF model. The only difference is that we used Area distribution approach for the intercepts selection for the Ensembles as it was described in the section 4.2.2.

The average simulation measures for the 3 DoF model are $MAE_x = 0.0565$, $RMSE_\tau = 0.0454$ and the simulation latency of 0.23 msec. The corresponding simulation plots are available in Appendix A.2.

4.4 Dynamics Model Implementation

4.4.1 3 DoF dynamics model

The dynamics model is much more complicated that the kinematics model. To implement the dynamics model we used the MuJoCo physics engine to compute the forward dynamics and the ABR Control[7] library as the perfect controller to compute the inverse dynamics. The choice of the target dynamics IDM and FDM functions based on MuJoCo and ABR Control will allow to directly compare the SNN model performance with the perfect controller.

To follow the experiment setup, discussed in the section 3.3, we designed and implemented a 3-DoF arm and hand model from scratch for the MuJoCo engine. This involved defining the physical properties, joint limits, and kinematic chain to ensure accurate dynamic behavior. The development process included extensive parameter tuning to match real-world dynamics, guided by the MuJoCo documentation[8] and our experimental requirements. The model has the same structure as the 3 DoF kinematics model, but configured to account the dynamics properties. The visualization of the implemented model is shown in the Figure 4.7.

FIGURE 4.7: Visualization of the 3 DoF arm and hand model implemented in MuJoCo.

[7] https://github.com/abr/abr_control
[8] https://mujoco.readthedocs.io/en/stable/modeling.html

Next, we integrated both the MuJoCo and ABR Control with the implementation of the SNN model that should approach corresponding IDM and FDM functions.

During the implementation of the dynamics SNN model, we figured out that the model requires more neurons and more complex structure to approximate the dynamics functions. To optimize the model performance and number of the dimensions of the Ensembles of IDM, we divided terms of the IDM equation 3.7 into two separate parts. The Ensemble with name "JTMxu" approximated the $J^T M_x u_x$ term, and accepted joint angles and error in task space as the input. The "Mdq-g" Ensemble approximated $k_v I\dot{q} - g_q$ term and accepted joint angles and joint velocities as the input. Also, we reduced the number of dimensions for the "FDM" Ensemble, that allowed to approximate the joint accelerations without the integrator feedback. The resulting structure of the SNN model is shown in the Figure 4.8.

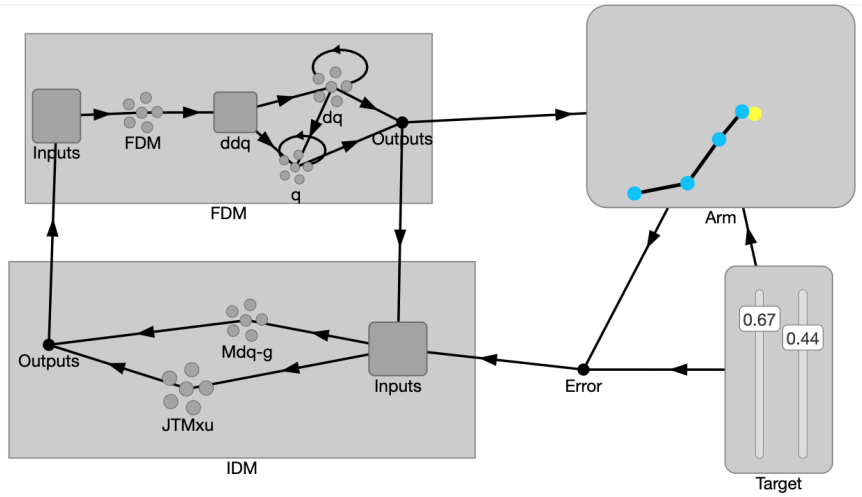


FIGURE 4.8: Visualization of the structure of the arm and hand dynamics model for 3 DoF.

To achieve better performance of the SNN model, we increased the number of neurons of both IDM and FDM ensembles to 1000, the "q" and "dq" integrator Ensembles was set to 500 neurons. After the evaluation, we got the average measures of $MAE_x = 0.0751$, $RMSE_\tau = 0.1060$ and the simulation latency of 0.45 msec. The corresponding simulation plots are available in Appendix A.3.

To compare the simulation latency of our approach and MuJoCo, we executed the same 25 seconds simulations but used only corresponding functions that calls MuJoCo forward dynamics interface and ABR Control inverse dynamics computation. As the result, the simulation latency of MuJoCo equals to 0.077 msec, which not that far from ours results in terms of real-time execution on desktop computers.

4.4.2 3 DoF dynamics model optimization

Previously, we have used the Neural Engineering Framework to optimize the connections of the SNN model, as it described in the equation 3.8 to approximate the target functions. The NengoDL[9] framework introduces the possibility to use regular deep learning techniques to optimize the SNN model further. The main idea is to use differentiable approximation of spiking neurons during training, and employing actual spiking neurons during inference. NengoDL automates these transformations if the user attempts to optimize a model that includes a spiking neuron model

[9] https://www.nengo.ai/nengo-dl/introduction.html

with an equivalent, differentiable rate-based implementation. Moreover, deep learning methods can optimize all network parameters (encoders, decoders, and biases), whereas NEF methods only optimize decoders.

We have selected hybrid approach when the Connections weights were optimized according to the NEF framework, but the Ensembles biases, gains and encoders were optimized using the deep-learning approach.

The optimization process was performed only on the IDM part of the model to validate the possibility of the optimization in the scope of our problem.

To perform the optimization, we have generated separate dataset of the joint angles, velocities, targets and the expected joint torques. The training dataset was generated by running the simulations of the 3 DoF arm and hand model using previously implemented perfect controller and forward dynamics interface. We have generated 10000 samples of the random joint angles, velocities and targets positions across the task space and recorded the corresponding joint torques through the 1000 steps of the simulation. The dataset was split into the training and validation sets with the 90/10 ratio. We used MSE as the evaluation function to compare the output joint torques with the ground truth values and optimize the model.

We used the Adam optimizer with the learning rate of 0.1 to optimize the model. Such a high value of the learning rate was selected empirically, as it provided better convergence of the optimization process. We also reduced the learning rate by a factor of 0.1 if the validation loss did not improve for 10 epochs during the training. Finally, we stopped the optimization process if the validation loss did not improve for 20 epochs.

We have used the NVIDIA RTX 2070 GPU to run the optimization process with the batch size of 256 samples. The optimization process early stopped after 130 epochs, and the final model achieved the validation loss of 0.0002.

We loaded the optimized parameters for the same evaluation process for the 3 DoF dynamics model. As result, we achieved the average measures of $MAE_x = 0.0545$, $RMSE_\tau = 0.0668$ and the simulation latency of 0.44 msec. The corresponding simulation plots are available in Appendix A.3.

Chapter 5

Conclusions and Future Work

In this master thesis, we approached the problem of real-time simulation of arm and hand dynamics using Spiking Neural Networks. We conducted an extensive review of related work to comprehend the project domain, evaluate current state-of-the-art solutions, and assess the potential of SNNs for addressing inverse and forward dynamics problems. To achieve the defined research goals, we developed a comprehensive methodology and research approach. This included designing custom experiments, configuring simulation environments, and developing SNN models, with it further optimization which includes dataset collection and training. For the experiments we have carefully approached the proper model parameters selection with possible improvements such as the use of area distribution for the intercepts and optimizing the model parameters on GPU supported by the NengoDL, according to the problem specifics. The gradual increase in model complexity from a simple 1 DoF kinematics model to much complex 3 DoF dynamics model demonstrated the scalability and effectiveness of the proposed apporach. We have shown that the implemented models can work in real-time speed and provide accurate predictions of the arm and hand dynamics. The evaluation results are presented in the Table A.1. The source code and all the related materials are available on demand in the GitHub repository[10].

This work only discusses the perspectives of using SNN for solving the inverse and forward dynamics problems. The next steps also could be the implementation of the SNN model for the more complex musculoskeletal models, with higher degrees of freedom. Also, we haven't discussed the application and energy efficiency of the SNN model for the low-energy hardware, which is one of the biggest advantages of SNNs. Future work could explore the integration of SNN models with neuromorphic hardware to leverage their low-power consumption benefits, making them suitable for portable and wearable devices.

In summary, this thesis has demonstrated the feasibility and effectiveness of using SNNs for real-time simulation of arm and hand dynamics. The results highlight the potential of SNNs in providing accurate and efficient solutions for dynamic modeling of human arm and hand. With further exploration and optimization, SNNs could become a powerful tool for advanced dynamic simulations and low-energy hardware implementations, leading to more sophisticated and practical applications in the field of neuroprosthetics.

[10] https://github.com/R1chrdson/snn-hand-dynamics

Appendix A

Experiment simulation results

A.1 1 DoF kinematics model simulation

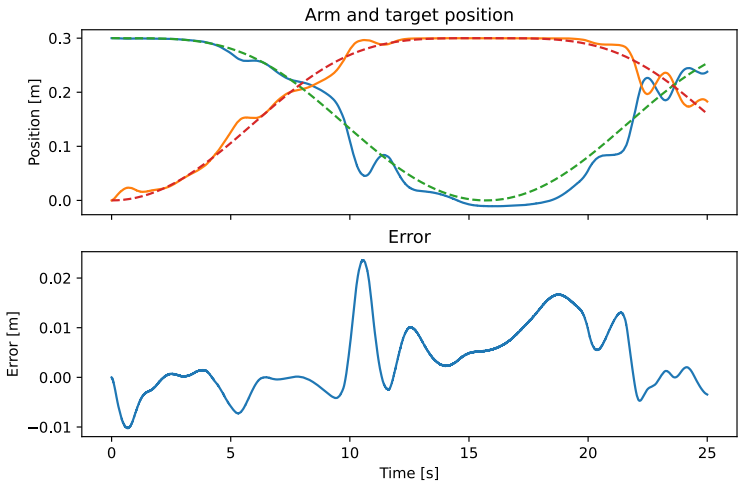


FIGURE A.1: The simulation for the 1 DoF kinematics model.

A.2 3 DoF kinematics model simulation

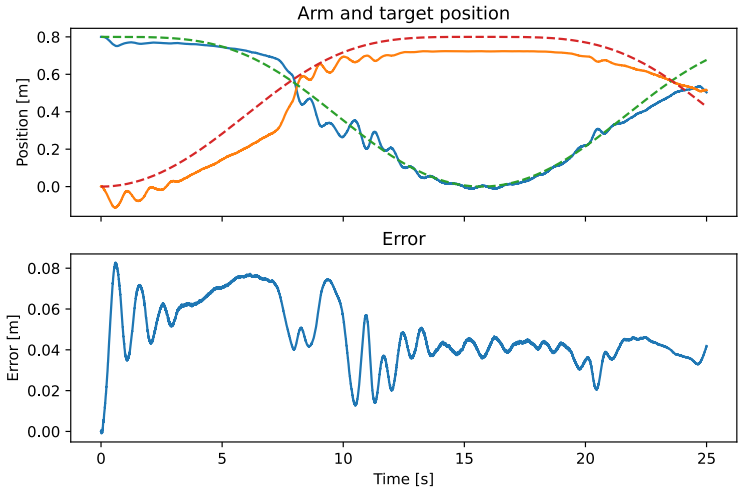


FIGURE A.2: The simulation for the 3 DoF kinematics model.

A.3 3 DoF dynamics model simulation

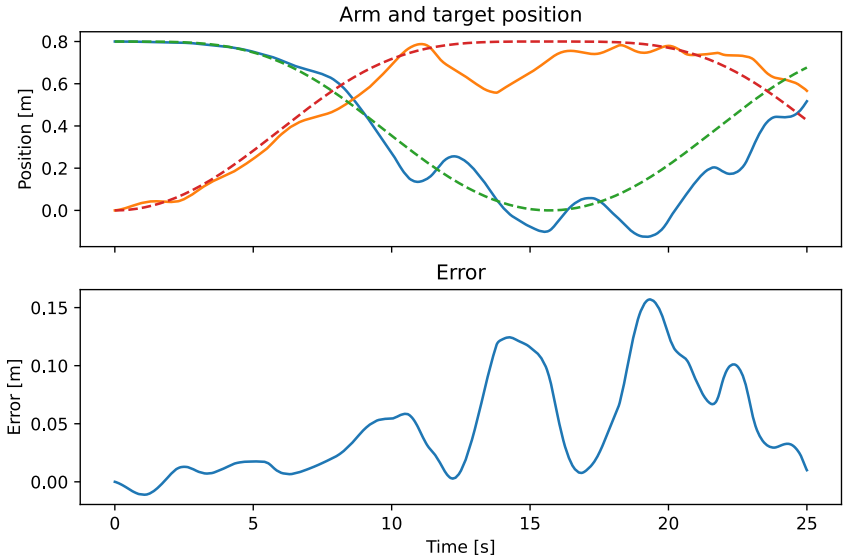


FIGURE A.3: The simulation for the 3 DoF dynamics model.

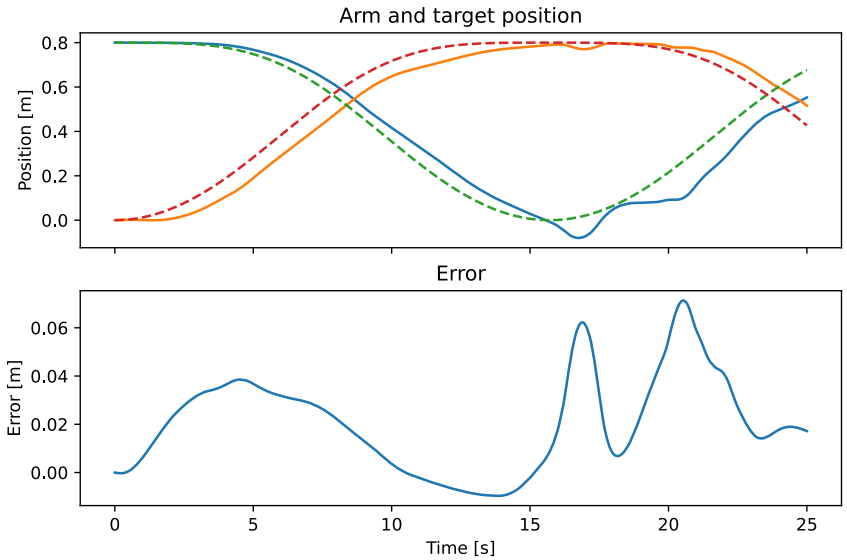


FIGURE A.4: The simulation for the optimized 3 DoF dynamics model.

A.4 Evaluation results

Model	MAE_x	$RMSE_\tau$	Simulation latency
1 DoF kinematics	0.0110	0.0057	0.18
3 DoF kinematics	0.0454	0.0565	0.23
3 DoF dynamics	0.0741	0.1060	0.45
3 DoF dynamics optimized	0.0545	0.0668	0.44

TABLE A.1: The evaluation results for the implemented kinematics and dynamics models.

Bibliography

Agur, A. M. R., Arthur F. Dalley, and 1886-1973. Grant J. C. Boileau (2008). *Grant's atlas of anatomy*. 12th ed. Philadelphia, Pa.: Lippincott Williams & Wilkins. Chap. xvi, 864 pages : illustrations ; 28 cm. ISBN: 0781770556; 9780781770552; 9780781796040; 0781796040.

Bekolay, Trevor et al. (2014). "Nengo: a Python tool for building large-scale functional brain models". In: *Frontiers in Neuroinformatics* 7. ISSN: 1662-5196. DOI: 10.3389/fninf.2013.00048.

Ceolini, Enea et al. (Aug. 2020). "Hand-Gesture Recognition Based on EMG and Event-Based Camera Sensor Fusion: A Benchmark in Neuromorphic Computing". In: *Frontiers in Neuroscience* 14. ISSN: 1662-453X. DOI: 10.3389/fnins.2020.00637.

DeWolf, Travis, Pawel Jaworski, and Chris Eliasmith (2020). *Nengo and low-power AI hardware for robust, embedded neurorobotics*. DOI: 10.48550/ARXIV.2007.10227.

DeWolf, Travis et al. (Nov. 2016). "A spiking neural model of adaptive arm control". In: *Proceedings of the Royal Society B: Biological Sciences* 283.1843, p. 20162134. ISSN: 1471-2954. DOI: 10.1098/rspb.2016.2134.

Eliasmith, Chris and C. H. Anderson (2002). *Neural Engineering: Computation, Representation, and Dynamics in Neurobiological Systems*, p. 376. ISBN: 0262050714.

Featherstone, Roy (2008). *Rigid Body Dynamics Algorithms*. Springer US. ISBN: 9780387743141. DOI: 10.1007/978-1-4899-7560-7.

Garg, Nikhil et al. (July 2021). "Signals to Spikes for Neuromorphic Regulated Reservoir Computing and EMG Hand Gesture Recognition". In: *International Conference on Neuromorphic Systems 2021*. ICONS 2021. ACM. DOI: 10.1145/3477145.3477267.

Gautam, Arvind et al. (2020). "MyoNet: A Transfer-Learning-Based LRCN for Lower Limb Movement Recognition and Knee Joint Angle Prediction for Remote Monitoring of Rehabilitation Progress From sEMG". In: *IEEE Journal of Translational Engineering in Health and Medicine* 8, 1–10. ISSN: 2168-2372. DOI: 10.1109/jtehm.2020.2972523.

Hwu, Tiffany et al. (2016). *A Self-Driving Robot Using Deep Convolutional Neural Networks on Neuromorphic Hardware*. DOI: 10.48550/ARXIV.1611.01235.

Jaworski, Łukasz and Robert Karpiński (June 2017). "Biomechanics of the human hand". In: *Journal of Technology and Exploitation in Mechanical Engineering* 3.1, pp. 28–33. DOI: 10.35784/jteme.536.

Khatib, O. (Feb. 1987). "A unified approach for motion and force control of robot manipulators: The operational space formulation". In: *IEEE Journal on Robotics and Automation* 3.1, 43–53. ISSN: 0882-4967. DOI: 10.1109/jra.1987.1087068.

Kvrgic, Vladimir and Jelena Vidakovic (2020). "Efficient method for robot forward dynamics computation". In: *Mechanism and Machine Theory* 145, p. 103680. ISSN: 0094-114X. DOI: 10.1016/j.mechmachtheory.2019.103680.

Lee, Chankyu et al. (Feb. 2020). "Enabling Spike-Based Backpropagation for Training Deep Neural Network Architectures". In: *Frontiers in Neuroscience* 14, p. 119. DOI: 10.3389/fnins.2020.00119.

Leeuwen, J. van, P. Aerts, and E. Otten (2003). "Inverse and forward dynamics: models of multi–body systems". In: *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences* 358.1437, pp. 1493–1500. DOI: 10.1098/rstb.2003.1354.

Manukian, Mykhailo, Serhii Bahdasariants, and Sergiy Yakovenko (Dec. 2023). "Artificial physics engine for real-time inverse dynamics of arm and hand movement". In: *PLOS ONE* 18.12, e0295750–.

Olmo, Manuel del and Rosario Domingo (Dec. 2020). "EMG Characterization and Processing in Production Engineering". In: *Materials* 13.24, p. 5815. ISSN: 1996-1944. DOI: 10.3390/ma13245815.

Pande, Sandeep et al. (Jan. 2013). "Modular Neural Tile Architecture for Compact Embedded Hardware Spiking Neural Network". In: *Neural Processing Letters* 38.2, pp. 131–153. ISSN: 1573-773X. DOI: 10.1007/s11063-012-9274-5.

Pizzolato, C. et al. (Oct. 2016). "Real-time inverse kinematics and inverse dynamics for lower limb applications using OpenSim". In: *Computer Methods in Biomechanics and Biomedical Engineering* 20.4, 436–445. ISSN: 1476-8259. DOI: 10.1080/10255842.2016.1240789.

Place, C.M. (Nov. 2017). *Dynamical Systems*. DOI: 10.1201/9781315141541.

Polydoros, Athanasios S., Lazaros Nalpantidis, and Volker Kruger (Sept. 2015). "Real-time deep learning of robotic manipulator inverse dynamics". In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. DOI: 10.1109/iros.2015.7353857.

Porsa, Sina, Yi-Chung Lin, and Marcus G. Pandy (Dec. 2015). "Direct Methods for Predicting Movement Biomechanics Based Upon Optimal Control Theory with Implementation in OpenSim". In: *Annals of Biomedical Engineering* 44.8, 2542–2557. ISSN: 1573-9686. DOI: 10.1007/s10439-015-1538-6.

Ren, Jia-Liang et al. (May 2019). "Deep Learning based Motion Prediction for Exoskeleton Robot Control in Upper Limb Rehabilitation". In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. DOI: 10.1109/icra.2019.8794187.

Rueckauer, Bodo et al. (Dec. 2017). "Conversion of Continuous-Valued Deep Networks to Efficient Event-Driven Networks for Image Classification". In: *Frontiers in Neuroscience* 11. ISSN: 1662-453X. DOI: 10.3389/fnins.2017.00682.

Sobinov, Anton et al. (Dec. 2020). "Approximating complex musculoskeletal biomechanics using multidimensional autogenerating polynomials". In: *PLOS Computational Biology* 16.12, pp. 1–26. DOI: 10.1371/journal.pcbi.1008350.

Stone, James (June 2018). *Principles of Neural Information Theory: Computational Neuroscience and Metabolic Efficiency*. ISBN: 978-0993367922.

Tiwari, Neha et al. (2018). "Brain computer interface: A comprehensive survey". In: *Biologically Inspired Cognitive Architectures* 26, pp. 118–129. ISSN: 2212-683X. DOI: 10.1016/j.bica.2018.10.005.

Todorov, Emanuel (2014). "Convex and analytically-invertible dynamics with contacts and constraints: Theory and implementation in MuJoCo". In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6054–6061. DOI: 10.1109/ICRA.2014.6907751.

Yoshimura, Natsue et al. (Sept. 2017). "Decoding finger movement in humans using synergy of EEG cortical current signals". In: *Scientific Reports* 7.1. ISSN: 2045-2322. DOI: 10.1038/s41598-017-09770-5.

Yough, Matthew G. et al. (May 2021). "A segmented forearm model of hand pronation-supination approximates joint moments for real time applications". In: *2021 10th*

International IEEE/EMBS Conference on Neural Engineering (NER). IEEE. DOI: 10.1109/ner49283.2021.9441405.

Zanchettin, Andrea Maria et al. (Jan. 2010). "Kinematic motion analysis of the human arm during a manipulation task". In: vol. 2, pp. 1–6. ISBN: 978-3-8007-3273-9.