APPLICATION OF CEREBELLUM INSPIRED CONTROLLERS TO BALANCE RELATED TASKS

Thesis

Submitted to

The School of Engineering of the

UNIVERSITY OF DAYTON

In Partial Fulfillment of the Requirements for

The Degree of

Master of Science in Electrical Engineering

By

Ricardo Evora Mota

Dayton, Ohio

December, 2022

APPLICATION OF CEREBELLUM INSPIRED CONTROLLERS TO BALANCE RELATED TASKS

Name: Mota, Ricardo Evora

APPROVED BY:

Raúl Ordóñez, Ph.D.
Advisory Committee Chairman
Professor and Graduate Programs Director, Electrical and Computer Engineering

Temesguen Kebede, Ph.D.
Committee Member
Adjunct Professor, Electrical and Computer Engineering

Terek Taha, Ph.D.
Committee Member
Professor, Electrical and Computer Engineering

Robert J. Wilkens, Ph.D., P.E.
Associate Dean for Research and Innovation
Professor
School of Engineering

Gül E. Kremer, Ph.D.
Dean, School of Engineering

ABSTRACT

APPLICATION OF CEREBELLUM INSPIRED CONTROLLERS TO BALANCE RELATED TASKS

Name: Mota, Ricardo Evora
University of Dayton

Advisor: Dr. Raúl Ordóñez

Despite impressive advancements in the field of robotics, tasks such quick reaching movements, bipedal locomotion, and balance maintenance have shown to be a challenge. A possible reason for this is the predominance of feedback controls in robotics, which provide robust controllers at the expense of a slower response. The part of the human brain responsible for the performance of such tasks is the cerebellum, which functions exclusively in a feedforward way. Prior studies have shown cerebellum inspired controller's capabilities in movement learning, performing quick reaching movements, and functioning in uncertain environments. This thesis focuses on supervised learning cellular-level cerebellum computational models and its capability of performing balance related tasks. Through computer simulations, the innovative design was tested for the first time on the balancing of the inverted pendulum and double inverted pendulum. Another concept investigated in this work is the effect of cerebellum network size on performance, where among four different network sizes, the largest network ever simulated by the EDLUT spiking neural network simulator was created. Lastly, the controller's capability to transfer knowledge to another model performing the same task with different dynamics was evaluated. All controller sizes tested displayed impressive results on the inverted pendulum, quickly learning how to balance the pole. For the double inverted pendulum, all but the smaller sized network were able

to achieve learning. The larger networks displayed better performance in both tasks, but the creation of even larger networks might be necessary to properly define the cerebellum network size effect on performance. The bio-inspired design was also shown to be capable of transferring knowledge, with an initially trained controller outperforming an initially naive controller on inverted pendulum models with different dynamics. The findings of this experiment show that cerebellum computational models are capable of learning balance related tasks and of transferring knowledge.

To my mother

ACKNOWLEDGMENTS

First and foremost, I would like express sincere gratitude to my advisor, Dr. Raul Ordonez, who introduced me to adaptive controls and believed in the project, allowing me to be able to focus full time on my thesis. His expertise, in-depth feedback, and precise suggestions played a major role throughout this project. I would also like to thank the thesis committee members: Dr. Temesguen Kebede and Dr. Tarek Taha, for taking the time to be part of my thesis defense and providing insightful comments that helped improve this work. Additionally, I would like to thank my mother, Cecilia, my sister, Luiza, and my sister-in-law, Elizabeth, who have been a constant source of encouragement and support. This achievement would not be possible without them.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER I

INTRODUCTION

Robots have been created to simplify the human experience. They are able to quickly move an item from point A to point B with a precision that cannot be met by a human. They can carry enormous weights without a single sign of exhaustion. They can almost instantly solve the most complex of algorithms which take humans hours to solve. More importantly, robots are able to repeat tasks continuously for as long as needed. They never feel tired, emotionally drained, or have any other ambitions aside from the task they were created to do. Yet, when it comes to replicating some of the most essential tasks that define us as humans, robots have difficulties. Tasks such as bipedal locomotion, performing quick reaching movements, maintaining balance in uncertain environments, and safely interacting with humans are examples of a few tasks that have proven to be challenging for robots. Which brings us to the question. How do we, as humans, learn? How can we perform tasks easily mastered by robots such as moving items while effortlessly maintaining balance on two feet? How can we adapt so quickly to different environments and scenarios? The answers to these questions can be found by studying the human brain.

The concept of applying biological knowledge of the brain to robotics is not new. The most widely known example of this is the use of artificial neuron networks (ANNs) [7]. ANNs are a type of computational mechanism inspired by the functioning of biological neurons. This mechanism is immensely common in adaptive control systems and computer vision applications. This is only a small concept of a broader field of neurorobotics, but illustrates the technological knowledge that can be acquired through the implementation of neuroscience to robotics. There is far more to explore when it comes to this topic. A brain region that has caught the attention of many researchers in this field is the cerebellum.

This small region of the brain is believed to be responsible for balance, movement learning, among other responsibilities and might hold the key to a more human-like robotics.

1.0.1 Cerebellum

The cerebellum is located in the lower back of the brain. Despite constituting only 10% of the brain's total volume, the cerebellum contains over 50% of all neurons in the brain. It plays an important role in motor control, being responsible for movement learning, balance and movement precision. It is also believed the cerebellum may play an important role in cognitive functions, such as attention, language, and emotional control. The cerebellum consists of numerous units containing the same simple microcircuit. Different regions of the cerebellum receive projections from different parts of the central nervous system (CNS) and projects to different motor systems. Despite receiving different inputs, each section of the cerebellum performs similar computational operations [8].

The role of the cerebellum in CNS is not to initiate a movement but to adjust a command to improve its performance. Unlike most of the CNS, the cerebellum performs exclusively in a feedforward manner, allowing it to provide extremely quick responses [9]. This might be the reason modern robots struggle to perform the tasks previously mentioned. Most modern control systems use feedback loops, meaning that in order to calculate the next output the controller depends on the previous one. This results in an extremely precise controller but with a feedback delay. For tasks such as maintaining balance or bipedal locomotion, being able to provide instant response is essential. For example, if a human that is standing up suffers a sudden force impact, they must readjust instantly in order to maintain balance, as a delayed response may cause the human to lose their balance and fall. A similar aspect can be seen when it comes to quick reaching movements. A group of researchers have tested

the performance of reaching movements by both feedback and feedforward control models. While both models performed well in slow reaching movements, the feedback controller was not able to accurately perform quick reaching movements. The feedforward model on the other hand, was able to perform this type of movement with ease [10]. Clinical data further proves the cerebellum importance in such tasks. Humans with cerebellar damage can suffer with symptoms such as inability to walk, to maintain balance, and incapability of performing rapid movements [8].

1.0.2 Cerebellum Computational Models

As for many regions of the brain, the inner workings of the cerebellum is not yet certain. Though, there does exist a widely accepted theory for the functioning of the cerebellum. This theory was initially proposed by a neuroscientist called David Marr in 1969 [11]. A majority of Marr's theory is still accepted today, with the exception of the cerebellum's learning rule, which became a topic of debate. Marr proposed that learning in the cerebellum is done by the strengthening of a specific interconnection between two types of neurons called PFs. Two years later an engineer called James Albus released his own theory of the functioning of the cerebellum [12]. Despite not having any prior knowledge of Marr's work, Albus proposed a similar theory but instead of learning through the strengthening of PFs, Albus believed that the cerebellum learns through the weakening of it. Despite his suspicion he was not able provide any proof of this behavior. In 1989, a neuroscientist by the name of Masao Ito was able to provide proof of the weakening of PFs in the cerebellum [13]. Thus, the Marr-Albus-Ito theory for the functioning of the cerebellum was created.

The first computational model of the cerebellum ever created was called the Cerebellum Model Articulation Controller, most commonly known as CMAC [1]. CMAC was created

by Albus and drew heavy inspiration from the perceptron. Despite being an extremely simplified model of the cerebellum, his model served as the inspiration for the numerous computational models to follow. These designs varied from models that are extremely faithful to the biology of the cerebellum to models that focus solely on its functioning [2]. Designs that focus on biological plausibility provide the most accurate representation of the cerebellum and can help further the understanding of the functioning of the brain, but they are extremely computationally complex to implement. Models that focus solely on functioning, on the other hand, are easy to implement and very accessible to people with no prior knowledge of neuroscience, but it is, at the end of the day, only a rough estimation of the functioning of the cerebellum. A few notable computational models models are the Fujita model [3], which uses adaptive filters to simulate the cerebellum, the Multiple Paired Forward and Inverse Model (MPFIFM) [4], which uses pairs of forward and inverse models, and the Schweighofer–Arbib model [14], which uses biologically plausible models of neurons to stimulate the cerebellum.

Due to computational constraints, models that attempt to biologically faithfully recreate the cerebellum have not been widely implemented before the last decade. In recent years computers have immensely improved its computational capabilities allowing for larger implementations of biologically plausible models of the cerebellum. These models have been tested in various applications such as eyelid conditioning and robotic arm movement, displaying promising results [15] [16]. These models have also shown to deal well in uncertain environments and safely interact with humans [6].

1.0.3 Objective

Lack of computational capabilities have stunned the development of cellular-level cerebellum models, thus this technology is still in its embryonic stages. The overarching goal of this thesis is to further study this technology, explore its capabilities, and understand its limitations.

As there have been many prior experiments that focus on the cerebellum controller's performance in limb movement, this work solely focuses on the controller's capabilities in balancing tasks. There have been a few studies that explored the cellular-level cerebellum computational model performance in balancing tasks, such as the inverted pendulum, but these were done through a reinforcement learning approach [15] [17] [5]. The cerebellum's role in reinforcement learning is highly debated, with some researchers arguing that it is inexistent, while others propose that the cerebellum plays a major role in this type of learning [18]. The cerebellum is widely accepted as being responsible for supervised learning in humans [19]. In this thesis, we focused exclusively on the supervised learning design of the cellular-level cerebellum computational model, applying it for the first time to the inverted pendulum. As part of this work, we also tested a cerebellum computational model for the first time on a double inverted pendulum. Aside from testing its performance in these tasks, the effect of network sizes on the biologically inspired controller was evaluated by testing each task on three network sizes, including a network which is two times the largest network ever simulated on the simulator used for this project. Lastly, we sought to understand the cerebellum inspired controller's ability to transfer knowledge by analyzing how a controller that was trained on an inverted pendulum behaves while controlling an inverted pendulum model with different dynamics.

1.0.4 Chapter Organization

The following chapters of this thesis will be organized in the following manner: chapter 2 will describe the questions we intend to answer throughout this thesis and provide our expected results. In chapter 3 we will provide any background information needed for this thesis. We will provide an in depth description of the functioning of the cerebellum, provide more details on prominent cerebellum computational model designs, and explain all the algorithms and concepts used for our implementations. Chapter 4 will consist of information on the simulation set up. There we will describe the computational specifications, parameters of the pendulum models used, controller design, and what defines a successful experiment. Chapter 5 will present and discuss the test results acquired. In the last chapter, we will provide a conclusion on the findings and talk about possible future endeavors.

CHAPTER II

PROBLEM STATEMENT

While cerebellum computational models have been studied for over fifty years, it is only recently that they have become a more accessible technology. With the advances in computational capacity in the last decade, bigger cerebellum networks have been able to be created allowing for larger and more complex applications. A study that used to be predominantly theoretical has become increasingly more practical. That said, since this technology is still in an early phase of applications, there are many aspects of these designs that are not yet known. In order to gain a deeper understanding of this innovative control design, we will be focusing on answering a few questions relating to cerebellum computational models and the cerebellum network throughout this thesis.

Many recent cerebellum controller applications tend to focus on movement learning, where every joint of the system is controllable. This thesis will focus on another key aspect of human existence that the cerebellum is responsible for, which is equilibrium. In order to maintain equilibrium, human beings must be able to quickly adapt to immense nonlinearity caused by numerous disturbances. Which brings us to the first question we will be focusing on in this thesis, how well do cerebellum computational models perform in balancing tasks?

Based on theory alone, it would be expected that the cerebellum controller would be easily able to handle balance tasks. It is good to remember that the model used in this work is not an exact representation of the cerebellum but a simplified version of it, which could affect its overall performance. Despite that, we are now able to simulate larger cerebellum networks, allowing us to expect that it will be capable of performing well in this type of task.

The second question we intend to answer with this work is how network size affects learning in the cerebellum. In the engineering side of the discussion, there have been stud-

ies that have proposed that the larger the cerebellum computational model network, the more effective it is at learning a task, as it is closer to the 50 billion neuron network size of a real cerebellum [15] [17]. While this could be true, the human cerebellum is responsible for learning numerous tasks, thus does not fully focus on a single task like in most experimental applications. The model used in these experiments uses the third generation ANN, which is the most recent ANN generation. In prior generations, as an ANN size increases, learning capability grows up to a point. Using an ANN that is oversized for an application may actually negatively impact its learning potential [20]. In the biological part of the discussion, a similar occurrence has been seen. According to a recent research, oversized biological neural networks can negatively affect learning [21]. We plan to further investigate this topic by testing the performance of different size cerebellum networks.

The expected results to this inquiry will largely depend on our computational limitations. Our goal is to build the biggest cerebellum network that we are computationally able to build. We will then compare its performance to a medium size and smaller size network. Considering that it is unlikely that we will be able to build an extremely large network, we expect that the large and medium size network will perform similarly.

The last question to be addressed is how well cerebellum inspired controllers transfer knowledge. In robotics, there are two types of knowledge transfer problems [22]. The first focuses on transferring knowledge from one task to another. The second centers on transferring knowledge from one plant to another, where the controller is trained to control a model and then tested performing the same task on a model with different dynamics. Our research focuses on the latter.

Past applications of this type of controller have shown impressive results when it comes to adapting to new environments. This might not be the case for our application, as we will be focusing exclusively on tasks with high degrees of nonlinearity. Taking that into

consideration, we still expect our cerebellum computational model to excel in transferring knowledge, as high capability of quick adaptation is one of the main features of this controller.

CHAPTER III

BACKGROUND

In order to fully understand the topics talked about in this thesis, this chapter will provide background information on important concepts used throughout this work. We will begin by providing information on the current understanding of the functioning of the cerebellum network. We will then talk more in depth on the cerebellum computational models providing a few prominent examples and discussing recent applications. Lastly we will explain important concepts that were important in order to understand the cerebellum simulation approach used in this project.

3.0.1 Biological Model of the Cerebellum

The most widely accepted theory for the cerebellum's behavioral function is the Marr-Albus-Ito theory [12] [13] [11]. This theory is the main influence for the control models based on the cerebellum. According to it, the cerebellum network consists mainly of 6 components. . These are the mossy fibers (MFs), climbing fibers (CFs), parallel fibers (PFs), granule cells (GCs), Purkinje cells (PCs) and deep cerebellar nuclei (DCN). With MF and CF being responsible for the network's inputs and the DCN for the network's output.

The cerebellum network works in the following manner. Signals holding the current sensorimotor information and the desired state are sent from other sections of the central nervous system through MFs. MFs make excitatory synapses with GCs and DCN, bringing them closer to their threshold potential. The GCs then, through PFs, make excitatory synapse with PCs, which also receive excitatory synapse from another system input, the CFs. CFs carry teaching signals coming from the inferior olive, which is found in the medulla oblongata. A single CF makes connections with various PCs. The only output connection

that PCs make is an inhibitory synapse with the DCN, bringing the DCN further away from reaching its threshold potential. Along with the excitatory input from MFs and inhibitory input from PCs, the DCN also receives excitatory input from CFs.

When a CF is activated, learning occurs through a phenomenon called synaptic plasticity. Synaptic plasticity is when the strength of a synapse is changed. As previously mentioned, whether the PF synapse is strengthened or weakened has been a topic of debate in the past. Marr initially proposed that non-activated PF synapses would strengthen when a CF is activated [11]. Albus believed that the active synapse would weaken but had no concrete evidence for this hypothesis [12]. Ito then discovered the occurrence of a synaptic phenomenon called Long Term Depression (LTD), which confirmed that CFs are a type of teaching signal that weakens the active PFs [13]. While LTD is the most commonly used learning method of cerebellum inspired controllers, it has been recently proposed that another phenomenon occurs in the cerebellum called metaplasticity [23], which is the plasticity of synaptic plasticity. This means that the same synaptic plasticity can at some points strengthen the synapse and at others weaken it. A metaplasticity occurrence called Spike Timing Dependent Plasticity (STDP) has recently been proposed as the method of learning occurring on the cerebellum, where LTD occurs in PFs when the teaching signal has been activated but if CFs are not activated, Long Term Potentiation (LTP) slowly occurs, strengthening the synapse. While most cerebellum inspired control models only mention synaptic plasticity in PFs, there has been evidence of plasticity in other parts of the cerebellum as well, such as the synapse between MFs and DCN [24].

Aside from the main components previously stated, there also exists 3 types of interneurons which are sometimes mentioned in control models and play smaller roles in the cerebellum network. These interneurons are the Golgi cells (GOs), stellate cells (SCs) and basket cells (BCs). The GOs receive excitatory inputs from MFs and PFs while serving as

an inhibitory input to GCs. BCs and SCs both receive excitatory input from PFs and serve as an inhibitory input to the PCs. The main difference between BCs and SCs is the region of the PCs that they supply inhibitory synapse to, making them often interchangeable in control models. A schematic of the cerebellum network can be seen in Figure 3.9.

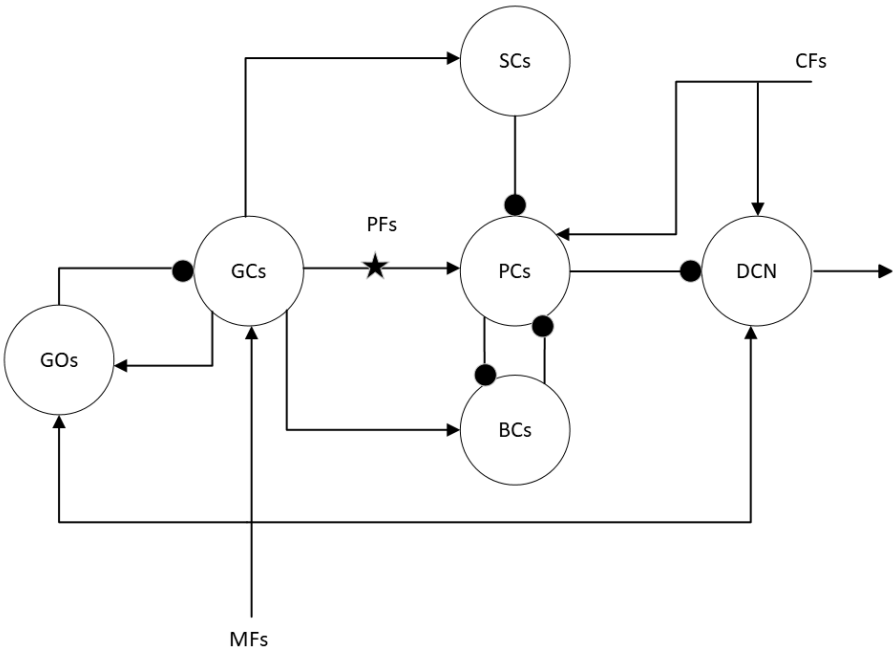

Figure 3.1: Marr-Albus-Ito Model. Arrows signify excitatory synapse, circles signify inhibitory synapse, and stars mark places of plasticity.

3.0.2 Computational Models of the Cerebellum

There exist several computational models of the cerebellum and numerous variations of each of them, but these models usually tend to fall within the following three categories: state-encoder-driven models, cellular-level models, and functional models [2]. While each category proposes vastly distinct models, there exist a few properties present in all models.

The first widely accepted property is that the cerebellum is a feedforward system. While all models are predominantly feedforward, some of them have feedback aspects that usually come into play in the initial attempts of learning a movement, before switching over to a feedforward control in the later stages of learning.

The second principle is that the cerebellum is a modular network. The cerebellum can be divided into independently functioning modules called microcomplexes. A microcomplex is defined as a microzone and an associated small group of DCN dedicated to a single function of motor control. A microzone is a set of PCs that project onto a distinct group of DCN [25]. In application to robotics, a microcomplex would be responsible for controlling one of the robot's degrees of freedom, whereas a robot arm with six degrees of freedom would require six microcomplexes.

The last property is that the cerebellum learns through synaptic plasticity. As mentioned earlier, there is evidence of plasticity happening in several parts of the cerebellum. In all computational models, learning happens either through plasticity or metaplasticity in the section that represents the PF-PC connection. Some variations of these models also include plasticity in other places like the MF-DCN connection.

As mentioned previously, cerebellum inspired computational models usually fall within three categories which each will be thoroughly explained in the following sections.

State-encoder-driven Models

State-encoder-driven models are usually defined as models that assume that GCs are on–off types of entities that split up the state space [2]. The issue with this definition is that it excludes models that do not use digital signals but also does not represent the GCs in a biologically plausible way. A better definition for this type of model would be a model that is biologically inspired by the architecture of the cerebellum but implemented

in a non-biologically plausible way. These models tend to represent the GC layer as a state encoder or an adaptive filter and take heavy inspiration from the perceptron. Even though these are the oldest of the three computational model types, they are still often used today.

The most widely known model of this type is the Cerebellum Model Articulation Controller (CMAC) [1]. The CMAC, shown in Figure 3.2, takes positional commands and feedback signals from the limb's sensors as inputs. It then computes the memory locations that should be activated for these inputs and activates them. Each memory location has its own assigned weight. The output of the model is defined as the sum of the weights of the activated memory locations. The output is then compared with the expected output. If there is an error, the weights of the activated memory locations are decreased, otherwise no changes are made. While, in CMAC, MFs are represented by the inputs, GCs are represented by the transformation of the input into a memory address, PFs are represented by the weighted addresses and PCs are represented by the sum of the weights, the role of the DCN is completely ignored. By doing this, CMAC also ignores the fact that PCs have exclusively inhibitory output and its main role is to stop the cerebellum from sending out a wrong command and not to be the source of its output. CMAC also ignores the direct influence that MFs have on the final output through the MF-DCN connection. The look-up table approach of imitating the cerebellum also suffers severely from the curse of dimensionality which may limit it in more complex applications [2].

A model called adjustable pattern generator (APG) was proposed as an improvement to CMAC [26]. Unlike in CMAC, APG acknowledges the key role that DCN plays in the cerebellum and that PC's main purpose is to inhibit it. Also in APG, the MFs make connections with both GCs and nuclear cells. A couple other previously mentioned components are also present in this model, such as BCs and GOs, which inhibits the PCs and GCs, respectively. APG also proposes a positive feedback loop between DCN and the motor cor-

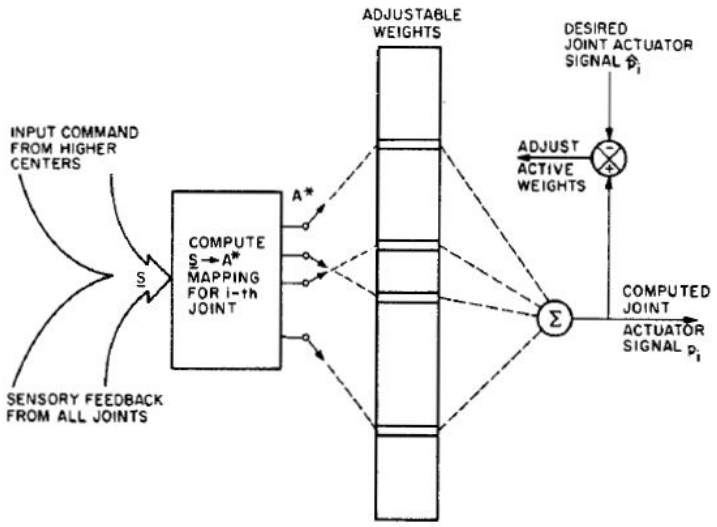

Figure 3.2: Schematic of the Cerebellum Model Articulation Controller. Source: Adapted from [1].

tex. When learning a new movement in APG, the BCs will initially inhibit PCs, giving the DCN-motor complex feedback loop full control of the output while plasticity is taking effect in PFs. After reaching a certain point, the BCs stop sending inhibitory signals to PCs. PCs then start sending inhibitory signals to DCN when its CF is activated, thus preventing the transmission of erroneous motor commands. The activation of the PCs also terminates the nucleus-motor cortex feedback loop. APG also addresses delays in updating weights by implementing eligibility traces, which acts as a local memory of recent synaptic activity so that the CF changes the weights of the PFs that activated it rather than the ones that are currently activated [27]. Since APG, like CMAC, uses the look up table approach to represent the GCs, it also suffers from the curse of dimensionality. Also like CMAC, APG works in discrete time which can limit its application. A schematic of the model can be seen in Figure 3.3.

Another model that uses CMAC as its foundation is the adaptive filter model of the

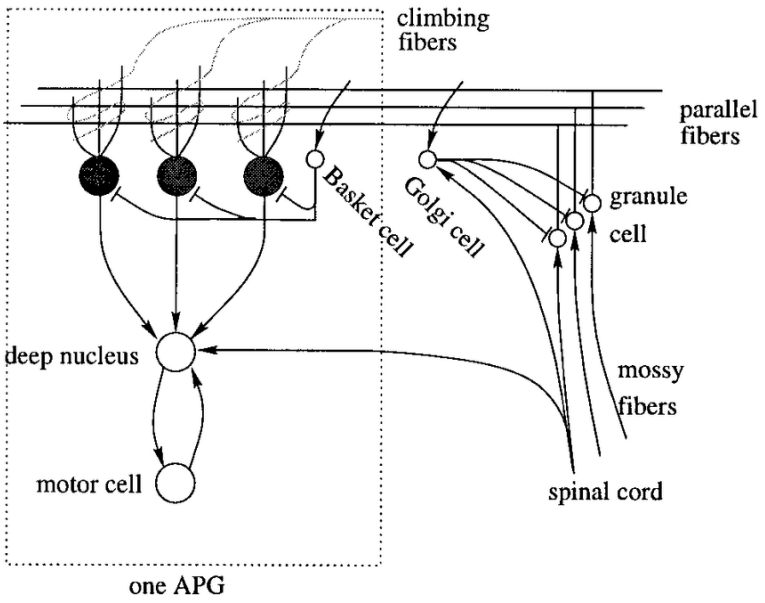

Figure 3.3: Schematic of the Adjustable Pattern Generator. Arrows represent excitatory synapse and horizontal lines represent inhibitory synapse. Source: Adapted from [2].

cerebellum [3]. Like CMAC, the adaptive filter model ignores the DCN and the inhibitory output of PCs. It functions very similarly to CMAC aside from a few additions and changes. The adaptive filter model abandons the look up table approach of simulating GCs and instead uses adaptive filters to split up the state space. This change causes a digital time control model to become continuous time. The adaptive filter model also proposes that GOs play a key role as a lag-compensator for GCs. Basket and SCs also play a key role in this model as they play an important factor in weight changes. A schematic of the model can be seen in Figure 3.4.

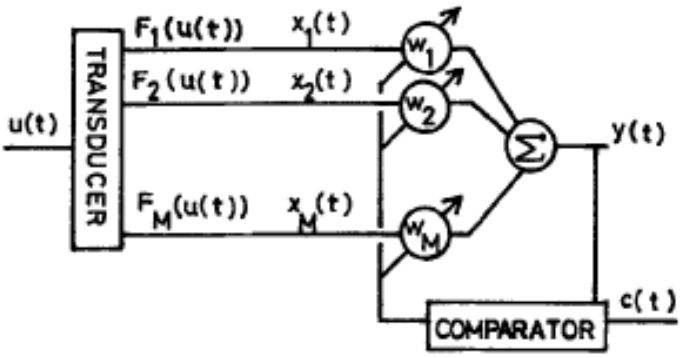

Figure 3.4: The Adaptive Filter Model. $u(t)$ represents the input, F_M represents the M^{th} filter, x_M is the output of $F_M(u(t))$, w_M represents the M^{th} weight, $y(t)$ is the output, and $c(t)$ is the desired output. Source: Adapted from [3].

Functional Models

Functional models ignore the architecture and main components of the cerebellum. Instead, they focus solely on replicating its functional understanding [2]. These models can be explained without any knowledge of neuroscience and use only control theory terms, thus making them more accessible. The use of both forward models and inverse models are common in the attempt to replicate the functioning of the cerebellum.

The most widely known functional model is the Multiple Paired Forward and Inverse Models (MPFIM) [4]. A MPFIM microcomplex usually consists of multiple modules that each contain a forward model, an inverse model and a responsibility predictor. Each module receives the same input signal. The simplest method of describing the inner workings of this model is to split it into three sections, multiple forward models, multiple inverse models and a responsibility predictor.

The multiple forward models section, shown in Figure 3.5, takes as an input the current motor command and the current state. Each forward model then makes a prediction for

the next state. After a delay, the previously estimated states are compared to the current state and the estimation error is calculated. The estimated error for each forward model is then used to calculate the amount of responsibility each module has in the output. Each model's predicted state is then multiplied by their responsibility factor and summed to form the multiple forward models section's state prediction output. Along with playing a larger role in the section's output, the module with the largest responsibility will adapt the most based on the output error.

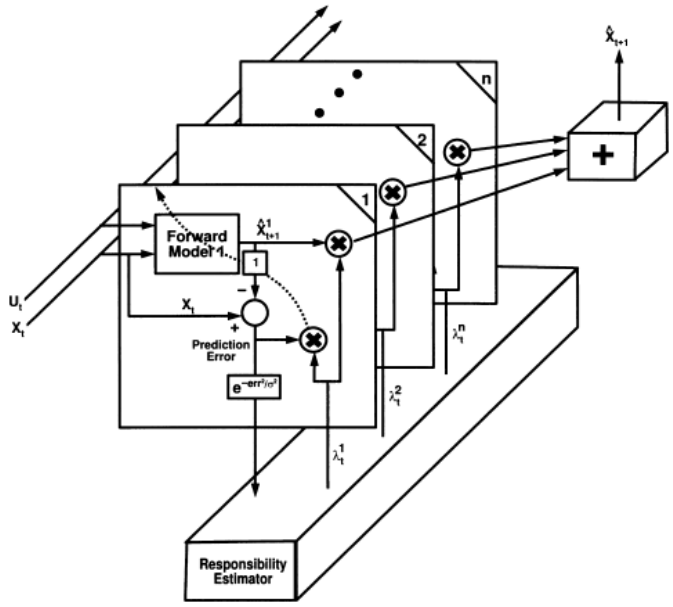

Figure 3.5: Multiple Paired Forward and Inverse Models: Multiple Forward Models Section. u_t represents the motor command, x_t represents the current state, $\hat{x}^1_{t+1}$ represents the first forward model's prediction of the next state, $\hat{x}_{t+1}$ is the section's output prediction of the next state, and and λ^1_t is the calculated responsibility of the first model. The dotted line is the training signal for learning. Source: Adapted from [4].

The multiple inverse models section, shown in Figure 3.6, takes as an input the desired state and the command feedback. The desired state is used in each inverse model to find the desired commands and the output of each model is then multiplied by the previously

calculated responsibility factor and summed to create the output command. The command feedback input is used solely to calculate the training signal. Similarly to the multiple forward models section, the more responsibility a module has, the more its inverse model adapts in the occurrence of an error.

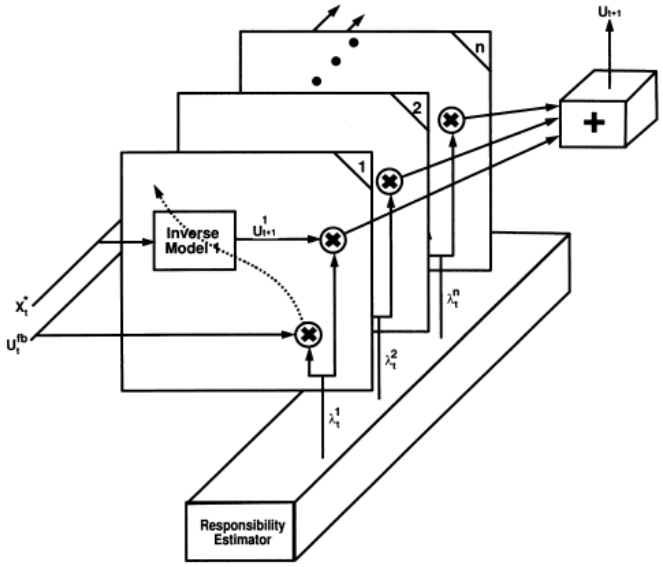

Figure 3.6: Multiple Paired Forward and Inverse Models: Multiple Inverse Models Section. u_{fb} represents the feedback motor command, x^* represents the desired state, $\hat{u}^1_{t+1}$ represents the first inverse model's calculated next command, u_{t+1} is the sections's output of the next command, and λ^1_t is the calculated responsibility of the first model. The dotted line is the training signal for learning. Source: Adapted from [4].

The responsibility predictor section, shown in Figure 3.7, attempts to predict the module's responsibility factors based on contextual information. Based on the prediction error, it adapts to provide better predictions in the future. A schematic of a single MPFIM module with all sections can be seen in Figure 3.8.

While most functional models tend to be computationally friendly and very accessible to

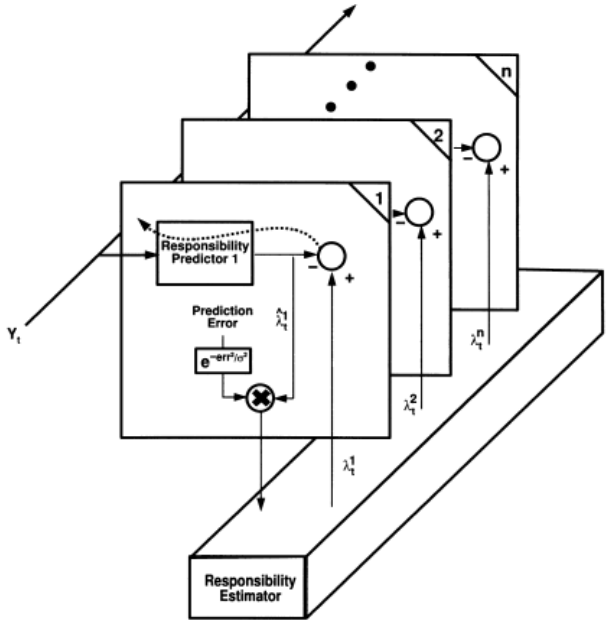

Figure 3.7: Multiple Paired Forward and Inverse Models : Responsibility Predictor Section. Y_t represents the contextual information, $\hat{\lambda}_t^1$ represents the first predictor's estimated responsibility, and λ_t^1 is the actual responsibility of the first model. The dotted line is the training signal for learning. Source: Adapted from [4].

audiences with no background in neuroscience, they are also only a rough estimation of the functionality of the cerebellum.

Cellular-level Models

Cellular-level models are computational models that are heavily biologically inspired by the cerebellum [2]. These models include all main cerebellum components and tend to also include all interneurons. Representations of other parts of the CNS such as the motor cortex and spinal cords are also common within these computational models. One of the main characteristics of cellular-level models is the use of Spiking Neural Networks (SNN). SNN is a type of Artificial Neural Network (ANN) that only fires once the neuron's

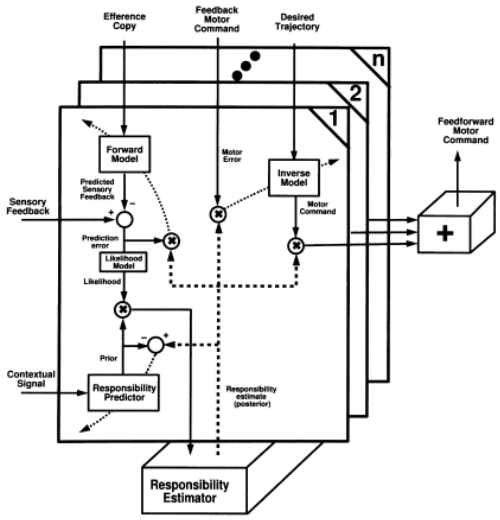

Figure 3.8: Multiple Paired Forward and Inverse Models : Schematic of a single module. The thick dotted line represents the role of the responsibility estimation's signal and the thin dotted line represents the training signal for learning. Source: Adapted from [4].

membrane potential has achieved a certain threshold [28]. The most commonly used spiking neural model is the Leaky Integrate-Fire [29], which is also the most common neuron model for cellular-level cerebellum models.

The most well known cellular-level computational model of the cerebellum is the Schweighofer–Arbib model, initially proposed by Schweighofer in 1995 [14]. This model was applied to explain known cerebellum system functions such as the saccadic eye movements [30][31] and reaching limb movement [10] [32]. Every interneuron plays an important role in this model. The GOs act as a feedforward-feedback inhibitory system that prevents the cerebellar cortex from receiving too much excitation and allows for better learning. Three GOs synapse with a single GC through the glomerulus, which is the point where the GOs' axon terminals come in contact with the GC's dendrites. The strength of each cell's inhibition will depend on the distance between the glomerulus and the GO. Similarly, the SCs act upon the dendrites of the PC to prevent their saturation when too many PFs have been activated, acting before

the slower GO inhibition takes place. BCs are pretty similar to SCs but inhibit several PC somas instead. In the Schweighofer–Arbib model there are two types of MFs. One of them carries information on the desired state of the plant while the other carries information on the current state. While this model tries to accurately replicate the cerebellum, it ignores the CF-DCN synapse due to CFs being the only part of the cerebellum that emits complex spikes rather than simple spikes. Due to this, the CF-DCN excitatory synapse would, in this model, essentially nullify the PC-DCN inhibitory synapse and prevent the cerebellum model to function as intended [32]. Learning in this model is usually implemented through LTD when a CF is activated and a gradual LTP when the error signal has not been activated [2].

Celular-level models provide the most realistic simulations of the cerebellum. From a biological point of view, it is the most important model type, since it allows for insight on the cerebellum function. Due to its complexity, it can be extremely challenging to implement these models [2].

3.0.3 Recent Applications

Possibly due to the recent advancement in computational capabilities, contemporary applications of cerebellum computational models tend to fall under cellular-level type. Different from designs such as the Schweighofer–Arbib, these models often simplify the cerebellum network by ignoring the interneurons and focusing on the six main components.

A substantial topic of debate in some recent applications is the type of learning the cerebellum performs. Supervised learning is a machine learning algorithm that is usually performed through an iterative process where a system's response to a given input is

compared to an expected output and the deviation from the expected output is used to adjust the system's adaptive elements. This machine learning method consists of three main components. The first is an input preprocessing which extracts features from raw data, providing an input signal that is more susceptible to learning. The second is an adaptive processor that generates an appropriate response to a provided input. This processor contains adaptive elements which will adjust based on the output error. The third component is a teaching signal which provides the processor with feedback on its performance. The cerebellum network contains all three of these components, with the input being preprocessed in the GC layer, the section beginning at the PCs and ending at DCN acting as the adaptive processor, and the CFs providing the teaching signal. Thus, the cerebellum is widely accepted as a supervised learning network [19].

The major topic of debate in recent research has been the cerebellum's role in reinforcement learning. Reinforcement learning is a machine learning technique that does not depend on any explicit teaching signal and learns using evaluative feedback based on a performed action. A common reinforcement learning technique is the actor-critic model. This model type consists of an actor which decides an action to take and a critic that evaluates the action and provides a reward [5]. Some researchers have attempted to show that the cerebellum focuses only on supervised learning, with the basal ganglia being solely responsible for reinforcement learning [17]. Other studies have proposed that the cerebellum works with other parts of the nervous system to perform reinforcement learning [18]. A group of researchers have recently proposed an alternative cerebellum model that is capable of reinforcement learning [5]. Differently from the most commonly used cellular-level supervised learning designs, which, as shown in Figure 3.9, focuses only on the cerebellum's six main components, the reinforcement learning paradigm includes a representation of the SC and BC interneurons named molecular layer interneurons (MLIs). In this set up, Purkenji

cells act as an actor, MLIs act as a critic, and CFs transport the reward. The cerebellum reinforcement learning paradigm is shown in Figure 3.10.

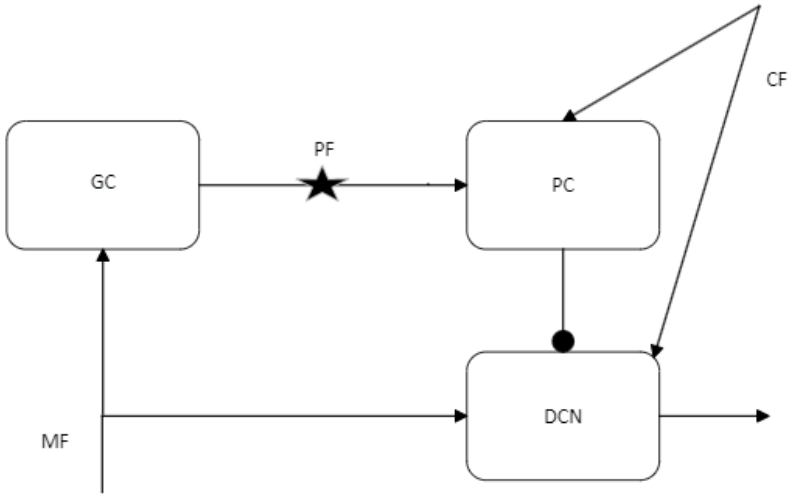

Figure 3.9: Cerebellum Supervised Learning Computational Model Design. Arrows signify excitatory synapse, circles signify inhibitory synapse, and stars mark places of plasticity.

3.0.4 Spiking Neural Networks

ANNs are computational systems that are inspired by the neural connections of the human brain. They are a tool that is often used in machine learning. ANNs can be separated into three generations [28], with the first being the perceptron. Based on the McCulloch-Pitts Neuron [7], the perceptron depends on a threshold function to decide if it fires an output or not. An example of a threshold function can be seen in Equation 3.1,

$$f(\vec{x}) = \begin{cases} 1, \text{if } \vec{w} \cdot \vec{x} > \theta, \\ 0, \text{otherwise.} \end{cases} \tag{3.1}$$

where $\vec{x}$ is an input vector, $\vec{w}$ is a weight vector, and θ is a scalar threshold.

A major limitation of the perceptron is that, while it can take both digital or contin-

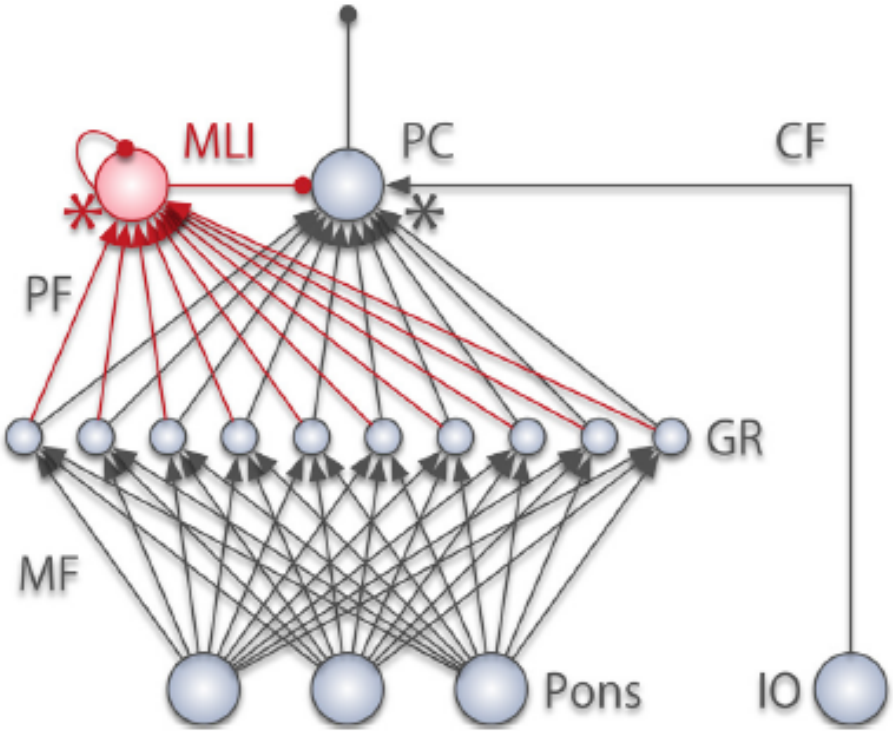

Figure 3.10: Cerebellum Reinforcement Learning Computational Model Design. Arrows signify excitatory synapse, circles signify inhibitory synapse, and asterisk mark places of plasticity. Source: Adapted from [5].

uous inputs, it can only provide digital outputs, making it unsuitable for continuous time applications. This issue was solved in the next generation of ANNs.

Second generation ANNs use activation functions instead of threshold functions, which provide continuous time output. A couple of commonly used activation functions are the sigmoid function and the saturated function, which are shown in Equations 3.2 and 3.3, respectively, where x is the input.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{3.2}$$

$$f(x) = \begin{cases} 1, & \text{if } x > 1 \\ x, & \text{if } 0 < x < 1 \\ 0, & \text{if } x < 0 \end{cases} \tag{3.3}$$

Second generation ANNs are also extremely versatile, being able to provide discrete time output with a threshold at the network's output. Due to this, second generation ANNs are currently the most widely used neural networks, being extremely common in fields such as machine learning and computer vision.

Despite being able to output either analog or digital signals, second generation ANNs run in discrete time. Operating in continuous time is extremely important for biological neurons, as they encode information using timing of spikes. Using temporal encoding allows for biological neurons to process more complex information using less neurons than second generation ANN [28]. The third and most modern generation of ANNs, SNN solves this issue. SNNs are modeled to imitate the functioning of a biological neuron network as much as computationally possible. The most commonly used SNN model is the leaky integrate fire (LIF) neuron [29]. In this model a neuron is represented by a parallel RC circuit driven by a current $I(t)$. The voltage of the circuit is compared to a threshold θ. If the voltage reaches the threshold value, a spike is fired, A diagram of this circuit can be seen in Figure 3.11.

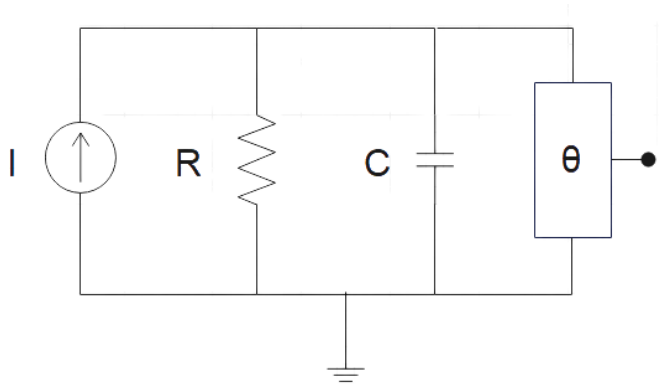

Figure 3.11: LIF neuron model circuit. The resistor is defined by R, C represents the capacitor, I is the input current, and θ is voltage comparator that fires once the threshold has been met.

Since the input current $I(t)$ can be defined in the following equation,

$$I(t) = I_R + I_C, \tag{3.4}$$

where I_R is the resistor's current and I_C is the current passing through the capacitor. Using Ohm's law, the definition of capacitance, and some rearrangement, the following differential equation can be derived,

$$C\frac{dV}{dt} = \frac{V(t)}{R} - I(t) \tag{3.5}$$

where $V(t)$ is the membrane potential, R is the membrane resistance, and C is the membrane capacitance. This model represents the LIF neuron until the membrane potential reaches the threshold θ. Once this happens, the neuron fires a spike and the membrane potential returns to the neuron's resting potential, where it remains until a refractory time t_r has passed.

There are also other neuron models that can be used in a SNN. There is the integrated fire neuron model [33] which is similar to LIF but does not take into consideration the membrane resistance, resulting in a slightly simpler and less biologically plausible model. There is also the Hodgkin–Huxley model [34] which attempts to create an artificial neuron that is as faithful as possible to its biological counterpart. This adds a lot of complexity to the model, thus, it is not suitable for larger operations. LIF neurons provide a good in-between point between biological plausibility and calculational simplicity, explaining its popularity. Regardless of which model is used, third generation ANNs are a lot more calculationally expensive than ANNs from the prior two generations, which can limit its applications.

3.0.5 Analog to Spike Conversion

In order to simulate an SNN, analog inputs must be converted to spikes. The method used to perform this conversion will depend on the biological characteristics of the input. Since MFs emit simple spike outputs it is quite simple to perform the analog to spike conversion, as it behaves like a regular spiking neuron. Like the first generation ANNs, spiking neurons have two possible outputs, the firing or non-firing of a spike. A MF input can then be modeled by dividing the dimensions of a state, such that each MF represents an equal sized portion of the space. The analog signal is converted to spikes by firing the MF that best describes the state. A way to do this is using a Gaussian radial basis function (RBF) to discretize the state dimensions. A Gaussian RBF can be defined by [35]:

$$\phi(x) = e^{-(\epsilon||x-x_i||)^2}, \tag{3.6}$$

where x is the input, x_i is the center of each region, ϵ is the smoothing factor. If x is located close to the center x_i, a larger RBF value will be obtained. On the other hand, points farther to x_i results in smaller RBF values. To convert the input, the MF assigned to the section with the highest RBF value will fire a spike.

CF inputs are a less simple conversion since it emits complex spikes. A previous study has been conducted to determine the most accurate way to model CF spikes [36]. This research showed that a deterministic approach can impair learning. CFs designed using stochastic processes displayed much better learning results. In specific the use of models where the spike interval probability follows a Poisson distribution proved to display the best results when it comes to CF analog to spike conversion. Poisson is a random variable distribution that follows a constant mean rate. A random variable follows the following

probability mass function [37]:

$$P(k,\tau) = e^{-\lambda\tau}\frac{(\lambda\tau)^k}{k!}, \tag{3.7}$$

where λ is the rate over a time interval τ and k represents the number of occurrences.

3.0.6 SNN Simulation Methods

Due to its complexity, SNN can be a challenge to simulate. There are currently two main approaches to SNN simulation, time-driven methods and event-driven methods. The main difference between the two approaches is when the neuron state is updated.

Time-driven Methods

Time-driven methods are SNN simulation approaches where neural states are updated based on time [38]. This is usually done by using differential equation solvers to solve for the membrane potential of each neuron. The most commonly used differential equations solvers are the Runge-Kutta methods [39]. These integration methods are designed based on Taylor Series. Its simplest form is known as the Euler method. Consider the following initial value problem:

$$\frac{dy}{dt} = f(t,y),\ y(t_0) = y_0, \tag{3.8}$$

where y is an unknown function, t is time, f is a function dependent on t and y, and t_0 and y_0 are the initial conditions of t and y, respectively. Using Euler's method this initial value problem can be solved through the following equation,

$$y_{n+1} = y_n + f(t_n, y_n)\Delta t, \tag{3.9}$$

where n represents sample number and Δt represents step time. A limitation of the Euler method is that the function's tangent is measured at the beginning of the step time. If the

function rapidly changes within a step time or if a step time is large, the Euler's method will likely approximate the function inaccurately. So this method is best applied to more simple differential equations. For more complex differential equations higher order Runge-Kutta methods are more effective. The higher the Runge-Kutta order, the higher its ability to accurately portray the function. That said, calculations also become more complicated as Runge-Kutta order becomes higher.

The most popular higher order Runge-Kutta methods are the second and fourth order, also known as RK2 and RK4, respectively. Using RK2, Equation 3.9 is expanded to include a term for the second derivative of $y(t)$ as follows,

$$y_{n+1} = y_n + (a_1 k_1 + a_2 k_2)\Delta t, \tag{3.10}$$

$$k_1 = f(t_n, y_n), \tag{3.11}$$

$$k_2 = f(t_n + b_1 \Delta t, y_n + b_2 k_1 \Delta t). \tag{3.12}$$

The a values and b values can vary depending which RK2 approach is used at this point. The two most commonly used approaches are the midpoint method and the Heun's method, which are both modifications of the Euler method [40]. In the midway method, instead of taking the derivative at the beginning of a step time, as it is done in the Euler method, the derivative is calculated at the midpoint of a time interval. Using this method, b_1 and b_2 will both be equal to $\frac{1}{2}$. Huen's method on the other hand, takes into consideration both the derivative at the beginning of step time and at the end. Using Huen's method b_1 and b_2 will be equal to 1. The values for a_1 and a_2 can then be found using the following equations

based on Taylor Series properties:

$$a_1 + a_2 = 1 \tag{3.13}$$

$$a_2 * b_1 = \frac{1}{2} \tag{3.14}$$

$$a_2 * b_2 = \frac{1}{2} \tag{3.15}$$

With the midpoint method, the value of a_1 and a_2 is found to be 0 and 1, respectively. Thus, the RK2 formula is derived to be

$$y_{n+1} = y_n + k_2 \Delta t \tag{3.16}$$

$$k_1 = f(t_n, y_n) \tag{3.17}$$

$$k_2 = f(t_n + \frac{\Delta t}{2}, y_n + k_1 \frac{\Delta t}{2}) \tag{3.18}$$

Using Huen's method, the value of a_1 and a_2 is found to be both $\frac{1}{2}$. Thus, through this approach the RK2 formula is defined as

$$y_{n+1} = y_n + \frac{\Delta t}{2}(k1 + k_2), \tag{3.19}$$

$$k_1 = f(t_n, y_n), \tag{3.20}$$

$$k_2 = f(t_n + \Delta t, y_n + k_1 \Delta t). \tag{3.21}$$

For the RK4 method, terms representing the third and fourth derivative of $y(t)$ are added. The equation for this method is much more complicated to derive. Thus, it is usually just simply defined in the following formula:

$$y_{n+1} = y_n + \frac{\Delta t}{6}(k1 + 2k_2 + 2k_3 + k_4), \tag{3.22}$$

$$k_1 = f(t_n, y_n), \tag{3.23}$$

$$k_2 = f(t_n + \frac{\Delta t}{2}, y_n + k_1 \frac{\Delta t}{2}), \tag{3.24}$$

$$k_3 = f(t_n + \frac{\Delta t}{2}, y_n + k_2 \frac{\Delta t}{2}), \tag{3.25}$$

$$k_4 = f(t_n + \Delta t, y_n + k_3 \Delta t). \tag{3.26}$$

Aside from the algorithm used, differential equation solvers tend to also differentiate in the type of step time used. Integration methods tend to fall under two step time categories: fixed-step integration and variable-step integration [41].

Fixed-step integrators are solvers that use a fixed step time. Meaning that numerous neurons are updated at a specific step time that does not change throughout the operation. This allows the solver to function in a relatively low calculation work load. This method does have its limitations, as neuron models defined by more complex differential equations will limit the maximum fixed step size that can be used.

Variable-step integration methods, on the other hand, iteratively adapts its step size in accordance to the neural dynamics. This process is ideal for neuron models defined by rather stiff differential equations. This iterative process can be quite computationally demanding, so its application is not recommended for networks with a high number of neurons.

Recently a new integration method has been developed that attempts to garner the positives and mitigate the negatives of fixed-step and variable-step integration methods. This method is called bi-fixed-step integration method [41]. In this approach two step times are fixed, a global step time T_g and a local step time T_l, such that

$$M_{gl} = T_g/T_l, \tag{3.27}$$

where M_{gl} is a positive integer. Neurons defined by the same differential model are then updated every T_g step time, similarly to a regular fixed-step integration method. What sets this method apart is that T_l can be used whenever needed to scale down the integration step size of a neuron by performing M_{gl} integrations of T_l within T_g.

Event-driven Method

An event-driven method is a SNN simulation approach where the neural state is updated when an event occurs [38]. An event is defined as a spike being received or being fired by a neuron. This method is significantly more difficult to implement than time-driven methods but may provide a faster simulation, since it does not need to calculate updates for a neuron with no events occurring. In its most basic version, an event-driven algorithm will, in the case of an event, calculate the time of the next event, update the state of the current neuron, and check for the threshold condition. The simulation will then wait until the time of the next event. This is, as mentioned, the most basic version of the algorithm, as it is only effective in the condition that there is no such thing as a non-instantaneous propagation. One of the most effective ways of performing event-driven simulations is by the use of look up tables [41]. A limitation of this approach is that more complex neuron models will demand higher dimension tables which increases look up times. Another concern might be networks that fire an excessive amount of spikes causing a slower simulation. Thus, the event-driven method is most suitable for simpler models with sparse firing times.

3.0.7 Event-Driven LookUp Table

Event-Driven LookUp Table (EDLUT) is a SNN simulator that was created using an innovative method of event-driven simulation in CPU platforms. This method, called event-driven look-up table, characterizes all neuron dynamics offline through the use of look-up tables. Numerous tables of neural characterization and synaptic dynamics are generated for each neuron type with more complex models requiring more tables with bigger dimensions. These tables are generated based on neuron type characteristics and network interconnection information provided by the user. Events coming from both external and internal

sources are sent to an event queue, where every event is organized chronologically. The most recent event is then extracted and the look-up tables are updated. Any generated event caused by this extracted event is then sent to the event queue. Despite being able to recreate any SNN of the user's choosing, EDLUT was created with the cerebellum neural network in mind. So, much of the provided documentation, neuron models, and example applications are based on the cerebellum. This makes it the ideal simulator for this type of SNN.

As mentioned previously in this work, event-driven simulations are more appropriate for smaller and simpler applications, which limited the simulator. Later on, time-driven techniques were added for both CPU and GPU platforms. Its time-driven simulation approach allowed for the use of different integration algorithms such as the Euler method, RK2, and RK4. EDLUT also allows the user to choose between a fixed step or a bi-fixed-step integration approach. Like in the event-driven model, the user must provide a description of the neuron types and the interconnections of the network, along with a description of the differential equation solver including the algorithm and the step type. It should also be noted that EDLUT is capable of running simulations where some neuron types are event-driven while others are time-driven and different differential equation solvers can be used for different neuron types.

In one of its most recent and most impressive applications, EDLUT was used to perform a time-driven simulation of cerebellum that controls the arm of a Baxter robot [6]. The robotic arm has six degrees-of-freedom, requiring six microcomplexes to be designed. Each microcomplex was designed with around ten thousand neurons, resulting in a over sixty-thousand neuron cerebellum network, making this the largest network simulation ever performed by EDLUT.

The experiment consisted of providing an input describing the robotic arm's current

position, desired position, current velocity, and desired velocity. The simulator would then output torque commands for each joint. The only synaptic plasticity present in the network was in the PFs using STDP plasticity. The computational model was tested in four scenarios. The first scenario the desired trajectory provided to the simulator was a circular trajectory. It took the controller 300 trials to learn the circular trajectory. A graphical representation of these results can be seen in Figure 3.12.

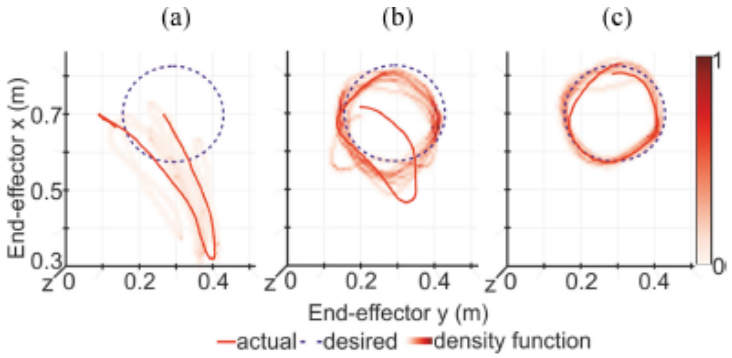

Figure 3.12: Circle Trajectory Test Results. Graph (a) shows the performance of the controller in early stages of learning. Graph (b) shows the controller's performance in the middle stage of learning. Graph (c) displays the controller's performance in the late stage of learning. Source: Adapted from [6].

An eight like trajectory was used for the second scenario. The controller had more difficulties learning this trajectory but was able to learn after 500 trials. A graphical representation of these results can be seen in Figure 3.13.

The last trajectory based experiment was for the controller to attempt to learn random fast target reaching movements. It took around 1000 trials for it to learn the reaching movements. A graphical representation of these results can be seen in Figure 3.14. For all three trajectory experiments, the movement accuracy of the trained robot was compared against the factory-built position controller. The cerebellum controller out performed the

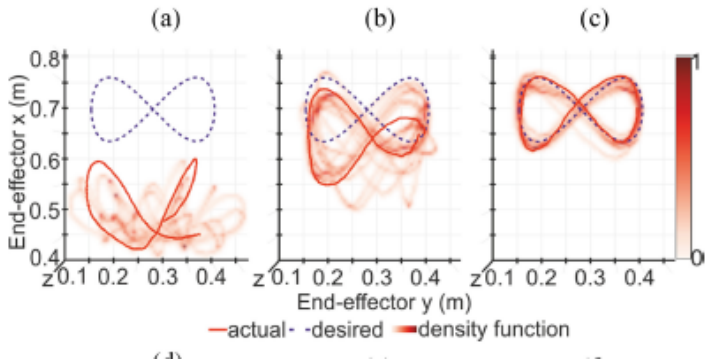

Figure 3.13: Eight like trajectory test results. Graph (a) shows the performance of the controller in early stages of learning. Graph (b) shows the controller's performance in the middle stage of learning. Graph (c) displays the controller's performance in the late stage of learning. Source: Adapted from [6].

built-in controller in every task.

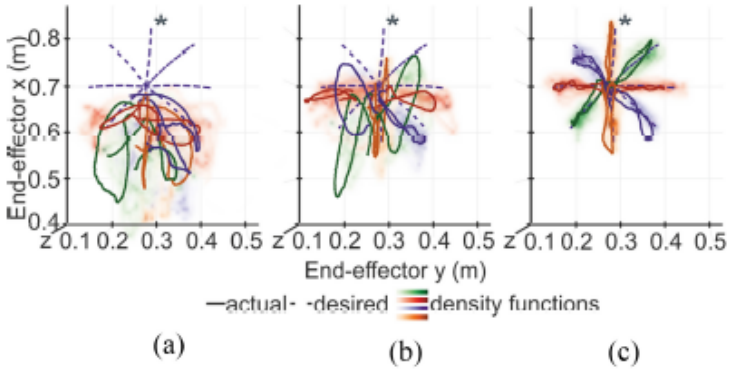

Figure 3.14: Random reaching movement test results. Graph (a) shows the performance of the controller in early stages of learning. Graph (b) shows the controller's performance in the middle stage of learning. Graph (c) displays the controller's performance in the late stage of learning. Source: Adapted from [6].

Arguably the most impressive results from this experiment come from the last test where the controller's performance in unstructured environments was evaluated. First a hanging weight was attached to the robot's end-effector while performing a circular trajectory. In a

different setup an elastic band was added to the robot to restrict its movement while performing the same movement. The robotic arm was able to adapt to the new circumstances on both occasions. Robot-human interactions were then tested with a human grabbing the robotic arm and being able to move around the work-space with no restrictions. A human was also able to get in the way of the robot's trajectory with no risk of injuries.

CHAPTER IV

SIMULATION SETUP

4.1 Simulation Setup

In this chapter our simulation methods will be described. We first will explain the hardware and software specifications used for this project, then the tasks to be accomplished will be specified. Lastly, we will go into detail on the computational model used.

4.1.1 Hardware and Software Specification

To perform this simulation, an HP Z240 Workstation with a quad-core Intel Xeon E3-1225 processor was used. This machine's memory consisted of 32 GB of RAM and a NVDIA GeForce GTX 1660 with 6 GB of memory.

The simulation was performed using Gazebo 11 and ROS Noetic on a Linux Ubuntu 20.04 operating system. The codes used to perform this project were written in the C++ programming language.

4.1.2 Task Description

The inverted pendulum and double inverted pendulum were chosen as the equilibrium problems that the cerebellum computational models will be tested on. Both problems and the parameters used by each will be described in this section.

Inverted Pendulum

The inverted pendulum, also known as the cart and pole problem, is a classic nonlinear control problem where a pole is attached to a cart through a revolute joint. The cart is able

to move only in one dimension either using wheels or, in our case, a prismatic joint attached to a rail. In many applications, such as ours, only the cart movement is controllable. In our experiment this rail will have a size of 5 meters. The goal of the controller is to make the cart move either left or right in order to maintain the pole balanced. The dynamics of the inverted pendulum can be described with the state-space equations [42],

$$(M+m)\ddot{x} + ml cos\theta\ddot{\theta} - ml sin\theta\dot{\theta}^2 = F, \tag{4.1}$$

$$ml cos\theta\ddot{x} + ml^2\ddot{\theta} - mgl sin\theta = 0, \tag{4.2}$$

where M is the cart's mass, m is the pole's mass, l represents the pole length, θ is the pole angle, x is the cart position, g corresponds to the gravity constant, and F is the input force. The simulation model for this application can be seen in Figure 4.1. The values that were used for each parameter can be seen in Table 4.1.

Table 4.1: Inverted pendulum model parameters.

Parameter	Value
M (kg)	35.0
m (kg)	1.0
l (m)	0.5

Double Inverted Pendulum

The double inverted pendulum is a similar but far more complex version of the inverted pendulum. In this problem two poles are connected with each other through a revolute joint and the bottom pole is then attached to cart through another revolute joint. As with

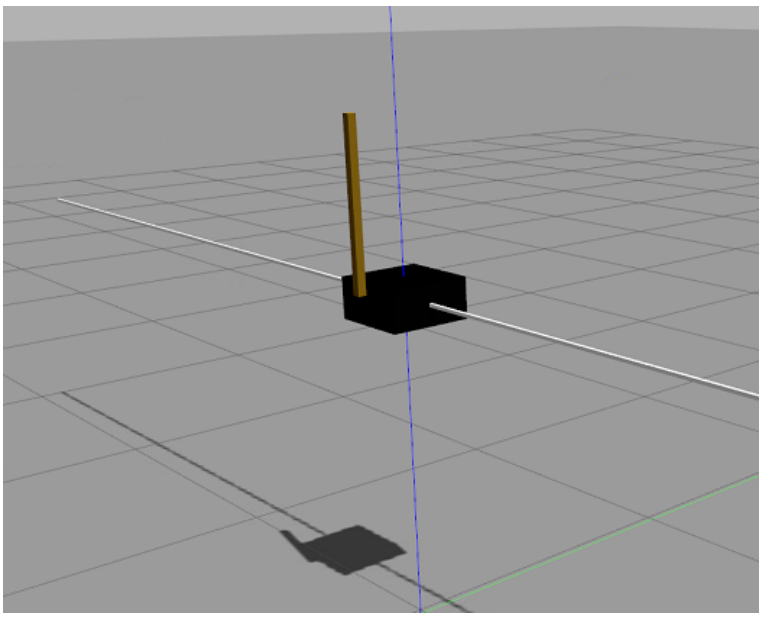

Figure 4.1: Inverted pendulum model.

the inverted pendulum, the cart is attached to a rail through a prismatic joint and is the only controllable part of the system. The size of the rail in this experiment will also be 5 meters. The double inverted pendulum can be defined by the state-space equations [43],

$$\begin{bmatrix} M+m_1+m_2 & \left(\frac{1}{2}m_1l_1+m_2l_1\right)cos\theta_1 & \frac{1}{2}m_2l_2cos\theta_2 \\ \left(\frac{1}{2}m_1l_1+m_2l_1\right)cos\theta_1 & \frac{1}{3}m_1l_1^2+m_2l_1^2 & \frac{1}{2}m_2l_1l_2cos(\theta_1-\theta_2) \\ \frac{1}{2}m_2l_2cos\theta_2 & \frac{1}{2}m_2l_1l_2cos(\theta_1-\theta_2) & \frac{1}{3}m_2l_2^2 \end{bmatrix} \begin{bmatrix} \ddot{x} \\ \ddot{\theta_1} \\ \ddot{\theta_2} \end{bmatrix} + \tag{4.3}$$

$$+ \begin{bmatrix} 0 & -\left(\frac{1}{2}m_1l_1+m_2l_1\right)\dot{\theta}_1sin\theta_1 & \frac{1}{2}m_2l_2cos\theta_2 \\ \frac{1}{2}m_1l_1cos\theta_1 & 0 & -\frac{1}{2}m_2l_1l_2\dot{\theta}_2sin(\theta_1-\theta_2) \\ \frac{1}{2}m_2l_2cos\theta_2 & -\frac{1}{2}m_2l_1l_2\dot{\theta}_2sin(\theta_1-\theta_2) & 0 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{\theta_1} \\ \dot{\theta_2} \end{bmatrix} +$$

$$+ \begin{bmatrix} 0 \\ -\left(\frac{1}{2}m_1+m_2\right)gl_1sin\theta_1 \\ -\frac{1}{2}m_2gl_2sin\theta_2 \end{bmatrix} = \begin{bmatrix} F \\ 0 \\ 0 \end{bmatrix},$$

where M represents the mass of the cart, m_1 is the mass of the bottom pole, and m_2 is the mass of the top pole. l_1 and l_2 are the bottom pole and top pole length, respectively. x represents the cart's position, θ_1 is the bottom pole's angle, and θ_2 is the top pole angle. g designates the gravity constant and F represents the input force. The simulated double inverted pendulum can be seen in Figure 4.2. The parameter values used for this problem

can be seen in Table 4.2.

Table 4.2: Inverted pendulum model parameters.

Parameter	Value
M (kg)	35.0
m_1 (kg)	0.5
m_2 (kg)	0.5
l_1 (m)	0.25
l_2 (m)	0.25

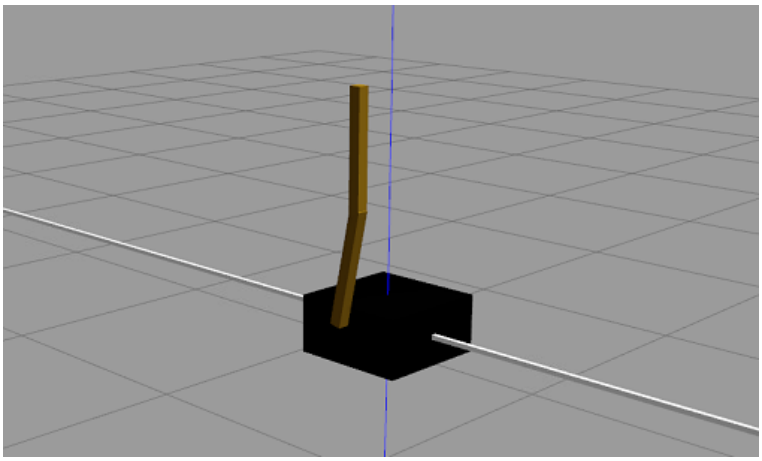

Figure 4.2: Double inverted pendulum model.

Transfer of Knowledge Experiment

To test the controller's ability to transfer knowledge, we will first train a controller to balance an inverted pendulum. The network size used will depend on the performances

of the inverted pendulum tests. The initially trained controller will then be tested on an inverted pendulum with different parameters. For comparison, an initially untrained version of the controller will be tested under the same circumstances. This is similar to the comparison of the performance of two individuals on a task, one with prior knowledge of it but performing it under different circumstances than it is used to and another who is completely unfamiliar with the task. Five trials will be run and a comparison of trial lengths acquired by the initially trained and initially untrained controller for each different parameter will be performed. The three parameters that will be altered for this experiment is the pole length, the pole weight and the rail length. Each parameter will be altered twice and only one of these parameters will be altered at a time. The altered parameter values can be seen in Table 4.3.

Table 4.3: Tested parameters.

Parameter	Value 1	Value 2
mass of pole (kg)	2.0	4.0
length of pole (m)	1.0	0.25
length of rail (m)	2.0	1.0

Trial Description

In the beginning of each trial, the cart will be set to its initial position in the center of the rail. The poles in both the inverted pendulum and double inverted pendulum will be set up right with a slight random angle deviation ranging from -0.001 and 0.001 radians for the inverted pendulum and between -0.00001 and 0.00001 radians for the double inverted pendulum. The controller will then attempt to balance the poles for as long as it is able to. Once the poles have fallen, the recorded time for the trial and the trial number are

displayed. The cart and poles will then be reset to its initial position and the next trial begins.

For the inverted pendulum task, a trial is considered successful if it lasts 60 seconds or more. For the double inverted pendulum a trial is considered successful if it lasts 15 seconds or more. For an experiment to be considered a success, ten consecutive successful trials must be performed. Each experiment will be allowed up to fifty trials to learn the task. The experiment will come to an end either when ten consecutive successful trials have been achieved or fifty trials have been conducted. For the transfer of knowledge tests, the inverted pendulum task will be executed for ten trials for each parameter by both the trained and untrained controller.

4.1.3 Cerebellum Computational Model

This thesis is meant to build on the experiments described in the end of last chapter [6]. The controller used in our applications is a slightly altered version of that controller. Thus many concepts of this section will be coming from that work. The outline of the controller used in our applications can be seen in Figure 4.3.

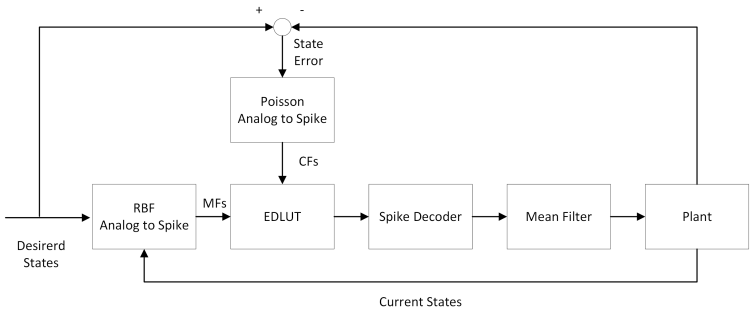

Figure 4.3: Controller diagram.

Analog to Spike Input Conversion

The controller takes as an input the current position, current velocity, desired position, and desired velocity of each joint. This information is then used to calculate both the MF and CF inputs. First, the network's MFs are equally split between each microcomplex. Each of the microcomplex's MFs are once again split equally between current position, current velocity, desired position and desired velocity inputs. For example, if a cerebellum network for a two degree-of-freedom robot has $2N$ MFs, each microcomplex is assigned N MFs. $\frac{N}{4}$ MFs are then assigned for each of the four microcomplex inputs. Each MF is defined by an unique index number, which in this example ranges from 0 to $2N - 1$.

Each microcomplex input's dimension is discretized into $\frac{N}{4}$ sections using a Gaussian RBF, where each section is assigned a MF index in order from left to right. The neuron indexes that describe the current position, current velocity, desired position, and desired velocity of a joint is then sent to the simulator as the MF input.

The CFs input, on the other hand, is created using an error estimator which compares the current and desired positions and velocities for each joint. CFs belonging to each microcomplex is split into two groups, positive CFs and negative CFs. If the value calculated by the error estimator is negative, a Poisson process generator will output random negative CF indexes. The amount of spikes that will be fired depends on how large the estimated error is, with a larger error resulting in more spikes being fired. If the error estimation is positive, the same process as the negative value occurs but instead of negative CFs, CFs from the positive group are fired.

Spiking Neural Network Model

LIF spiking neuron models were used to simulate this network. LIF models were chosen because this application demands vast amounts of artificial neurons. LIF's simplicity and efficiency make it perfect for such applications. The internal and external current of the LIF used in this project were calculated using the conductance of the neurotransmitters of the cerebellum. There are three main neurotransmitters responsible for neuron synapse in the cerebellum. These are AMPA, NMDA, and GABA neurotransmitters [44]. AMPA and NMDA are both neurotransmitters that cause excitatory synapse, while GABA is responsible for inhibitory synapses. The spiking neural networks used in this project can be defined by

$$C_m \frac{dV}{dt} = I_{internal} + I_{external}, \tag{4.4}$$

$$I_{internal} = -g_l(V + E_L), \tag{4.5}$$

$$\begin{aligned} I_{external} = &-(g_{AMPA}(t) + g_{NMDA}(t)g_{NMDA\_INF})(V - E_{AMPA}) \\ &- g_{GABA}(t)(V - E_{GABA}), \end{aligned} \tag{4.6}$$

$$g_{AMPA}(t) = g_{AMPA}(t_0)e^{\frac{t-t_0}{\tau_{AMPA}}} + \sum_{i=1}^{N} \delta_{AMPA_i}(t)w_i, \tag{4.7}$$

$$g_{NMDA}(t) = g_{NMDA}(t_0)e^{\frac{t-t_0}{\tau_{NMDA}}} + \sum_{i=1}^{N} \delta_{NMDA_i}(t)w_i, \tag{4.8}$$

$$g_{GABA}(t) = g_{GABA}(t_0)e^{\frac{t-t_0}{\tau_{GABA}}} + \sum_{i=1}^{N} \delta_{GABA_i}(t)w_i, \tag{4.9}$$

$$g_{NMDA\_INF} = \frac{1}{1 + e^{(62V)\frac{1.2}{3.57}}}, \tag{4.10}$$

where C_m is the membrane potential, V is the membrane potential, $I_{internal}$ is internal current, and $I_{external}$ is the external current,E_L is the resting potential, and g_L is the resting potential conductance.g_{AMPA}, g_{NMDA}, and g_{GABA} represent the conductance of

each receptor type. g_{NMDA_INF} is the NMDA activation channel and w_i is the synaptic weights. The value used for each parameter is displayed in Table 4.4.

Table 4.4: Neuron model parameters.

Parameters	GC	PC	DCN
$C_m(pF)$	2.0	100.0	2.0
$g_l(nS)$	1.0	6.0	0.2
$E_L(mV)$	-65.0	-70.0	-70.0
$E_{AMPA}(mV)$	0.0	0.0	0.0
$E_{GABA}(mV)$			-80.0
$\tau_{AMPA}(ms)$	1.0	1.2	0.5
$\tau_{NMDA}(ms)$			14.0
$\tau_{GABA}(ms)$			10.0
$V_{thr}(mV)$	-50.0	-52.0	-40.0
$T_{ref}(mV)$	1.0	2.0	1.0

Synaptic Plasticity

In this model, the only synaptic plasticity happening is STDP in PFs. This learning rule is defined by

$$LTP\Delta w_{ij}(t) = \alpha\delta_{PFspike}(t)dt, \tag{4.11}$$

$$LTD\Delta w_{ij}(t) = \beta\int_{-\infty}^{t_{CFspike}} k(t - t_{CFspike})\delta_{PFspike}(t)dt, \tag{4.12}$$

$$k(x) = \begin{cases} \frac{-(x+d_k)}{\tau_{LTD}-d_k} e^{\frac{x+d_k}{\tau_{LTD}-d_k}}, & \text{if } x < -d_k \\ 0, & \text{if } x \geq -d_k \end{cases}, \tag{4.13}$$

where ΔW_{ij} is the synaptic weight between the ith PF and the jth PC, α is the synaptic efficacy increment, β is the synaptic efficacy decrement, δ_{PF} is the Dirac delta function representing a spike from a PF, τ_{LTD} corresponds to the time constant of the biological sensorimotor delay, and d_k is an adjustment of the kernel $k(x)$'s width. The definition for each one of these variables can be seen in Table 4.5.

Table 4.5: Plasticity plasticity parameters.

Parameters	Value
$\alpha(nS)$	0.002
$\beta(nS)$	-0.0005
$\tau_{LTD}(ms)$	100
$d_k(s)$	0.07

4.1.4 Neurons and Network Topology

EDLUT was used to simulate GCs, PCs, and DCN of each network. These neurons were all simulated using time-driven integration methods. Both the PCs and DCN neurons were simulated with the computer's CPU using a fixed-step RK4 differential equation solver. GCs were simulated with the GPU using a bifixed-step RK2 integration.

Four cerebellum networks of varying sizes were designed for this project. The basis for each network created is a cerebellum convergence and divergence ratio based on prior application of this controller [6]. In every designed network, each GC received four MFs, one from each of the four microcomplex subgroups, representing the desired joint position, desired joint velocity, current joint position, and current joint velocity. Every possible combination of four MFs was represented by a GC. Every PC received excitatory synapse from all GCs through PFs. Each CF was propagated to a single PC and DCN. A PC was only responsible for inhibiting a single DCN. Each DCN also received projections from every MF. Table 4.6 displays the convergence and divergence ratio for each neuron connection for a 10K neuron microcomplex.

Table 4.6: Base convergence divergence ratio for a 10K neuron microcomplex.

Neuron Connection	Quantity
MFs → GCs	4:1
MFs → DCN	40:1
GCs → PCs	10,000:1
PCs → DCN	1:1
CFs → PCs	1:1
CFs → DCN	1:1

20K Neuron Network

The first network created has around 20 thousand neurons. This network was created exclusively for the inverted pendulum. It provides each microcomplex with 10K neurons. This network is the smallest inverted pendulum network on this project. Further details on the total numbers of neurons and interconnections can be seen in Table 4.7.

Table 4.7: 20K Network Description.

Neuron		**Synapses**			
Pre-synaptic cells	Post-synaptic cells	Number	Type	Initial weight (nS)	Weight range (nS)
80 MFs	20K GCs	80K	AMPA	0.18	-
80 MFs	200 DCN	16K	AMPA	0.1	-
20K GCs	200 PCs	4M	GABA	1.6	[0,5]
200 PCs	200 DCN	200	AMPA	1.0	-
200 CFs	200 PCs	200	AMPA	0.0	-
200 CFs	200 DCN	200	AMPA	0.5	-
200 CFs	200 DCN	200	NMDA	0.25	-

30K Neuron Network

This network was created exclusively for the double inverted pendulum. Like the 20K, this network provides each microcomplex with 10K neurons. Since the double inverted pendulum is a three degree of freedom problem, it needs three microcomplexes. This network is

essentially the 20K network with an added microcomplex. The details on the total number of neurons and interconnections can be seen in Table 4.8.

Table 4.8: 30K Network Description.

Neuron		**Synapses**			
Pre-synaptic cells	Post-synaptic cells	Number	Type	Initial weight (nS)	Weight range (nS)
120 MFs	30K GCs	120K	AMPA	0.18	-
120 MFs	300 DCN	36K	AMPA	0.1	-
30K GCs	300 PCs	9M	GABA	1.6	[0,5]
300 PCs	300 DCN	300	AMPA	1.0	-
300 CFs	300 PCs	300	AMPA	0.0	-
300 CFs	300 DCN	300	AMPA	0.5	-
300 CFs	300 DCN	300	NMDA	0.25	-

60K Neuron Network

Unlike the previously mentioned networks, the 60K neuron network is used both for the inverted pendulum and the double inverted pendulum. Both tasks use the same number of each neuron but separated differently. For the inverted pendulum the 60K network is split into two, creating 30K neurons microcomplexes. On the other hand, for the double inverted pendulum, this network was split into three microcomplexes of 20K neurons. Despite being separated differently, this network is the same as the one used in the previous application of this controller [6], the largest network ever simulated by EDLUT prior to this work. Further details on the network can be found in Table 4.9.

Table 4.9: 60K Network Description.

Neuron		**Synapses**			
Pre-synaptic cells	Post-synaptic cells	Number	Type	Initial weight (nS)	Weight range (nS)
240 MFs	60K GCs	240K	AMPA	0.18	-
120 MFs	600 DCN	144K	AMPA	0.1	-
60K GCs	600 PCs	36M	GABA	1.6	[0,5]
600 PCs	600 DCN	600	AMPA	1.0	-
600 CFs	600 PCs	600	AMPA	0.0	-
600 CFs	600 DCN	600	AMPA	0.5	-
600 CFs	600 DCN	600	NMDA	0.25	-

120K Neuron Network

This is the largest network created for this project. It is also two times bigger than the largest network ever simulated by EDLUT. This network was applied to both the inverted pendulum and the double inverted pendulum. Like in the previous network, the neurons were split into two microcomplexes for the inverted pendulum, with each microcomplex containing six times the base neuron ratio. For the double inverted pendulum, the network was split in to three microcomplexes, with each containing four times the base neuron ratio. Table 4.10 contains further details, such as the neuron quantity and interconnections.

Table 4.10: 120K Network Description.

Neuron		**Synapses**			
Pre-synaptic cells	Post-synaptic cells	Number	Type	Initial weight (nS)	Weight range (nS)
480 MFs	120K GCs	480K	AMPA	0.18	-
480 MFs	1200 DCN	576K	AMPA	0.1	-
120K GCs	1200 PCs	144M	GABA	1.6	[0,5]
1200 PCs	1200 DCN	1200	AMPA	1.0	-
1200 CFs	1200 PCs	1200	AMPA	0.0	-
1200 CFs	1200 DCN	1200	AMPA	0.5	-
1200 CFs	1200 DCN	1200	NMDA	0.25	-

4.1.5 Controller Output

Before being applied to the task being performed, the DCN' s output needs to be processed. First a spike decoder is used to convert the spiking neuron output to a controller command. EDLUT's output comes in the form of DCN neuron indexes. To translate this, first all DCN neuron indexes of each microcomplex need to be equally split into two groups, positive DCN neurons and negative DCN neurons. If the output neuron index is part of the positive DCN group, it will be converted to a predefined force output. If the output neuron index is part of the negative DCN neurons, it will be converted to a force output of the same magnitude but in the opposite direction. The amount of force exerted varies depending on the microcomplex. This allows for results that focus more on one of the states than the other. For example, in an inverted pendulum, maintaining the angle of the pole up right is far more important than maintaining the cart in the center of the rail, so the output force exerted from the microcomplex focused on the pole angle should be larger than the one focusing on cart position. The values for the force exerted from the positive and negative DCN of each microcomplex in both tasks can be seen in Table 4.11 and Table 4.12, respectively. The states in both tables are described in the same symbols used to describe Equation 4.4, with θ_1 representing the bottom pole angle, θ_2 representing the top pole angle, and x representing the cart position. The output force of all microcomplexes will then be summed to form the network's output force. Following this, the summed output will then go through a mean filter. The mean filter will wait for nine more outputs and supply the cart with the mean force of the last ten commands.

Table 4.11: Positive DCN force output.

Task	θ_1 Focused Force Output (N)	θ_2 Focused Force Output (N)	x Focused Force Output (N)
Inverted Pendulum	400	-	100
Double Inverted Pendulum	-50	50	25

Table 4.12: Negative DCN force output.

Task	θ_1 Focused Force Output (N)	θ_2 Focused Force Output (N)	x Focused Force Output (N)
Inverted Pendulum	-400	-	-100
Double Inverted Pendulum	50	-50	-25

CHAPTER V

RESULTS

In this chapter, the simulation results will be shown and analyzed. We will first discuss the inverted pendulum simulation results for each of the three network sizes. Then the double inverted pendulum simulation results will be analyzed. Lastly we will discuss the results for the transfer of knowledge tests.

5.0.1 Inverted Pendulum

The first controller tested on the inverted pendulum was the 20K neuron network controller, the smallest of the three. This controller performed well with the inverted pendulum, being able to get ten consecutive successful trials within fourteen trials. In the first three trials, the controller was only able to balance the pole for less than 13 seconds. After achieving the first successful trial in the fourth try, the controller displayed a crescent trend up until the 10th trial where it achieved its highest trial length, lasting over 3 minutes. Upon reaching this peak, the controller displayed a large drop in performance with an attempt that lasted a little over one third of the previous trial. Upon that point a new crescent trend began ending with the last trial where the pole was balanced for over 2 minutes. The graphical representation of this experiment can be seen in Figure 5.1.

The second controller tested was the 60K neuron controller. This controller also performed well on the inverted pendulum task. Unlike the previous controller, this cerebellum model did not need to go through any failed trials. Despite only achieving successful trials, this controller did display slightly less consistent trends with large drops after longer trials happening twice throughout the series. The most clear example of this behavior happens between trials 6 and 7, where the controller achieved its best trial with an astonishing 13

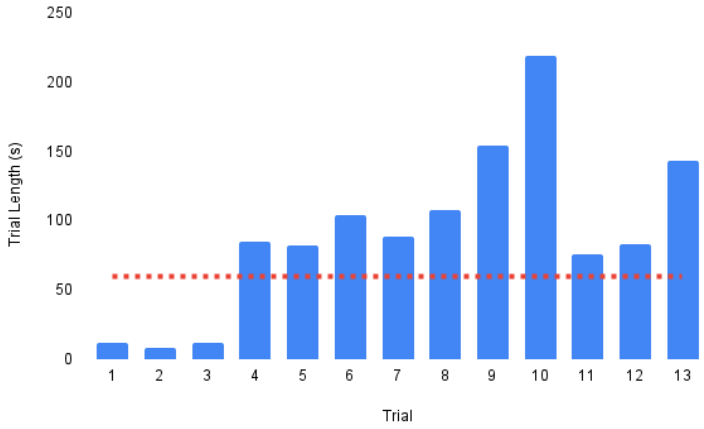

Figure 5.1: 20K network sized computational model's performance balancing an inverted pendulum. The red dotted line represents the successful trial mark of 60 seconds.

minute trial and then followed it with its worst trial, barely hitting the 60 second mark. The controller then displayed a new increasing trend achieving the second highest length in the series three trials later balancing the pole for over 3 minutes. More details on this part of the experiment can be seen in Figure 5.2.

The last controller tested for the inverted pendulum was the 120K network. Similar to the 60K network, this controller did not have any failed trials.The controller performed its longest trial in its first attempt, balancing the pole for over 7 minutes. It did also underperform a couple times following longer trials, similarly to the 60K neuron network size tests. This network did behave slightly differently than the other networks after a moment of underperformance. The first occasion that this can be seen is in the second trial, where a new crescent trend begins. Since this occurs after a successful first attempt of balancing the pole, it is understandable for a moment of underperformance to occur and a new increasing trend to begin from this trial length. The second occasion happens in trial 7, following a 7 minute long trial. Unlike the behavior previously seen, it seems like this trial did not affect

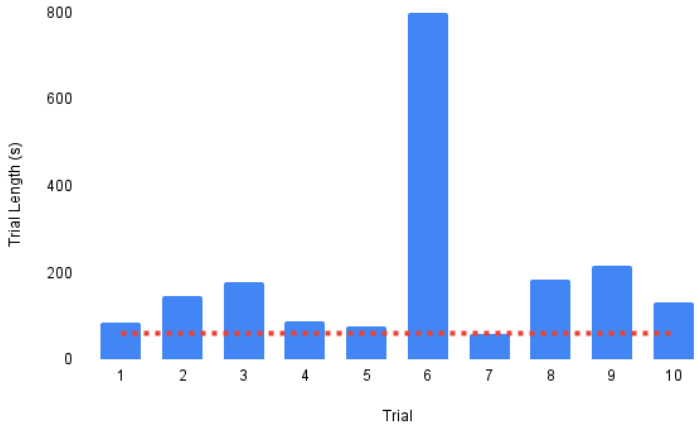

Figure 5.2: 60K network sized computational model's performance balancing an inverted pendulum. The red dotted line represents the successful trial mark of 60 seconds.

the prior upward trend, with trial 8 continuing the trend that had been shown up to trial 6. Further details of this network performance can be seen in Figure 5.3.

Overall every controller performed well in these tasks, requiring a few trials if any to start obtaining successful results. Once a successful trial was obtained, each network was capable of consecutively performing nine more trials lasting over a minute. While results were positive, there were multiple occasions where the controller was able to perform a longer trial than usual and then in the next trial it would perform one of the shortest trials of a series. It would be interesting to investigate how this pattern changes if more trials are conducted. The best performing network was the 120K neuron network since it had the highest trial length average and did not fail any attempts. The 60K neuron network came next as it achieved the highest trial length of all networks, acquired the second highest trial time average, and had no failed trials. The worst performing network was the smallest, as it failed the first few trials and obtained the lowest average trial length. Table 5.1 displays more details on each network's performance.

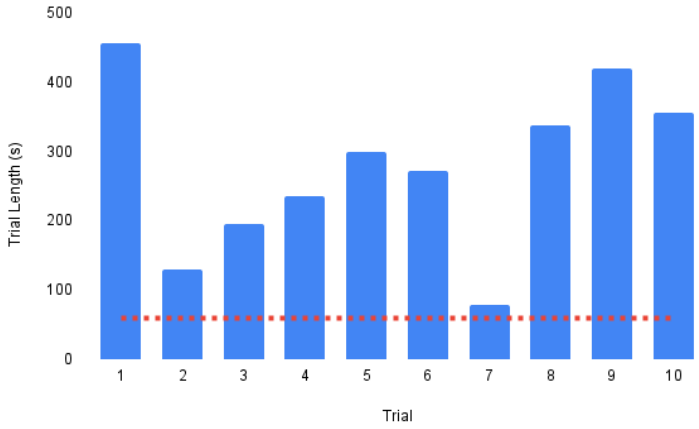

Figure 5.3: 120K network sized computational model's performance balancing an inverted pendulum. The red dotted line represents the successful trial mark of 60 seconds.

Table 5.1: Trial Performance Comparison

Network Size	Trials Needed	Max. Trial Time (s)	Min. Trial Time (s)	Average Trial Time (s)
20K	14	219.75	9.21	91.05
60K	10	799.5	76.12	197.05
120K	10	455.41	78.69	278.01

5.0.2 Double Inverted Pendulum

The first controller tested in the double inverted pendulum was the 30K neuron network computational model. This controller did not perform well in this task, failing to obtain ten consecutive successful trials. Out of the fifty trials conducted only eight were longer than 15 seconds, and often far apart. Upward trends can be seen throughout the fifty trials but were not enough to reach and maintain over the target time. Successful trials were more common in the later stages of the series but the trials in between were far from successful, displaying some of the lowest times in the series. The double inverted pendulum is a difficult task to master. It appears that this network size is too small to be able to handle this task.

More details on this test can be seen in Figure 5.4.

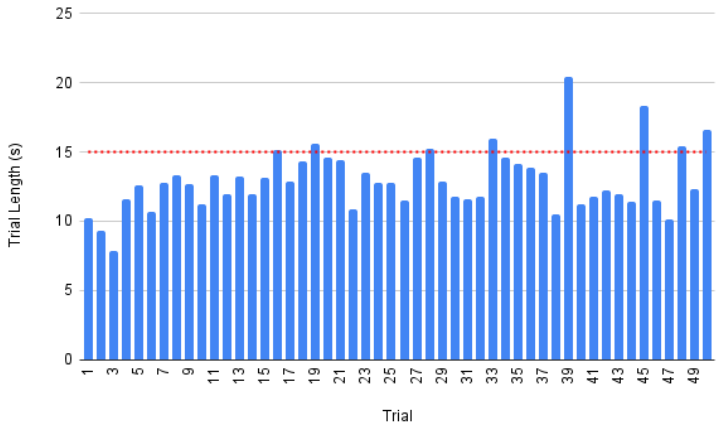

Figure 5.4: 30K network sized computational model's performance balancing an inverted pendulum. The red dotted line represents the successful trial mark of 15 seconds.

The second controller tested consisted of a 60K neuron network. This controller performed better than the 30K controller, achieving ten consecutive successful trials in sixteen attempts. The controller's first successful trial occurred in its second attempt with a 23 seconds trial, the longest trial achieved. The controller then obtained more successful trials until failing on its sixth trial. The controller was then able to achieve times that range between 15 and 21 seconds in the following trials. The trends throughout the series were not very consistent with moments of crescent and decrescent trends. More details on this series can be seen in Figure 5.5.

The last controller tested was the 120K neuron network controller. This controller had the best results in this task obtaining zero failed trials. Different from the previously tested networks where trials close to the 15 seconds mark were a common occasion, the lowest time trial achieved by the 120K controller was of 17 seconds which happened in the fourth trial. Similar crescent and decrescent trends to the 60K controller can be seen in this controller's

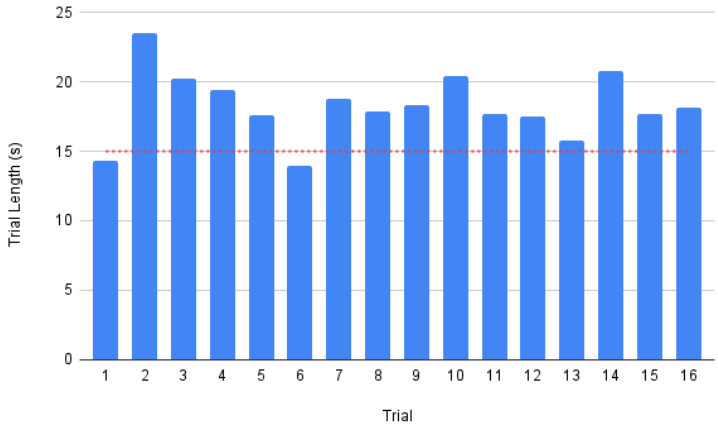

Figure 5.5: 60K network sized computational model's performance balancing an inverted pendulum. The red dotted line represents the successful trial mark of 15 seconds.

performance but with smaller variations. Its longest trial occurred in its second attempt as well, achieving an attempt that lasted around 21 seconds. Further details of this network's performance is shown in Figure 5.6.

The results of the double inverted pendulum show that it is a hard task to master. Network sizes that were able to obtain over 10 minute trials in the inverted pendulum were not even able to constantly achieve 20 seconds trials. That said, ignoring the expectations that come from the excellent inverted pendulum performance, the larger controllers achieved good results on the double inverted pendulum. The smaller network size, on the other hand, had a poor performance in this task. Like in the inverted pendulum, the largest network size performed better achieving the highest average trial length and achieving success with fewer trials. The second best was once again the 60K neuron network controller, obtaining the second highest average trial length and the longest trial of all the series. The 30K neuron network obtained the worst average trial length and failed to obtain consecutive

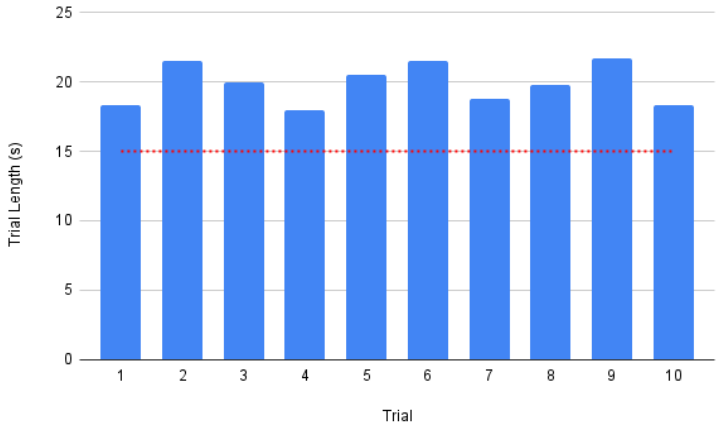

Figure 5.6: 120K network sized computational model's performance balancing an inverted pendulum. The red dotted line represents the successful trial mark of 15 seconds.

successful trials. Table 5.2 displays more details on each network's performance.

Table 5.2: Double inverted pendulum trial performance comparison.

Network Size	Trials Needed	Max. Trial Time (s)	Min. Trial Time (s)	Average Trial Time (s)
20K	-	20.38	7.85	12.95
60K	16	23.47	13.96	18.25
120K	10	21.65	17.99	19.82

5.0.3 Transfer of Knowledge

The controller chosen to test the computational model's ability to transfer knowledge was the 20K neuron network controller. This controller was chosen since it was the only controller that needed a couple trials before achieving trials over a minute. This will allow for a more clear display of how well the trained controller can transfer knowledge.

The first parameter altered was the pendulum's mass. In this first test, the pendulum

mass was set to double of the mass value in the initial model, which was the model used to train the controller. Overall the initially trained controller outperformed the initially untrained one in all trials except Trial 2. The initially trained controller also displayed much higher trial times than the initially untrained controller was able to obtain. The performance of both controllers in the first five trials can be seen in Figure 5.7.

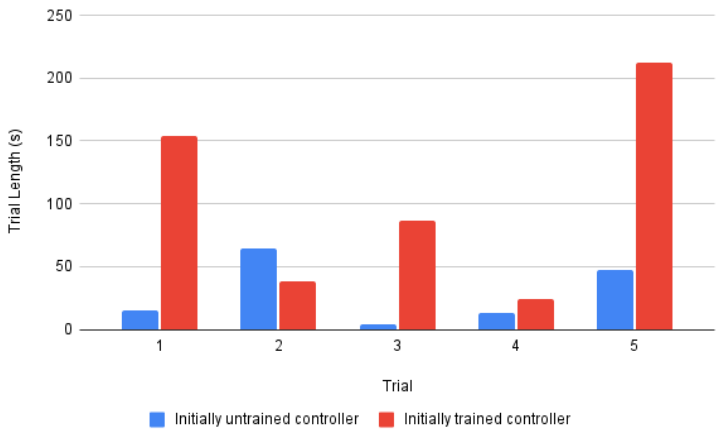

Figure 5.7: Performance of initially trained and initially untrained controller with 2 kg pole mass.

In the next model tested, the pole mass was doubled once again, becoming four times heavier than the model used to train the controller. The initially trained model once again outperformed the initially untrained model in these circumstances, obtaining higher times in all but one of the trials. Once again the initially trained controller was able to obtain much larger trial lengths than the initially untrained controller was capable of. More details of this series can be seen in Figure 5.8.

Next, the initially trained controller was tested on a model with the pole length double the size of the original. Like in the previous tests, the initially trained controller outperformed the initially untrained controller in all trials except one. In this scenario, the initially

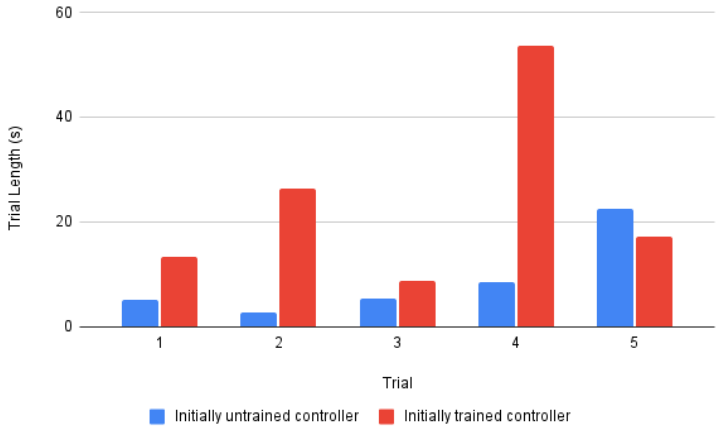

Figure 5.8: Performance of initially trained and initially untrained controller with 4 kg pole mass.

trained controller also displayed its capability of achieving much higher trial lengths than the initially naive controller. Figure 5.9 displays the performance of each controller under this circumstance.

As part of the second alternate pole length test, the original pole size was divided by two. In this scenario, the initially trained controller was able to balance the pole longer than the initially naive controller in every attempt. Once again the initially trained controller was overall able to achieve much higher trial times than the initially untrained controller, with its longest trial lasting three times the longest initially naive controller trial. More details on this series of trials can be seen in Figure 5.10.

Next, the rail length used by the cart was altered, restricting its movement. First the rail length was changed from 5 meters to 2 meters. Once again the initially trained controller outperformed the initially naive controller in almost every trial, only obtaining a slightly smaller trial time than the initially untrained controller in the last trial. Similarly to previous series, the initially trained controller was able to achieve overall much higher

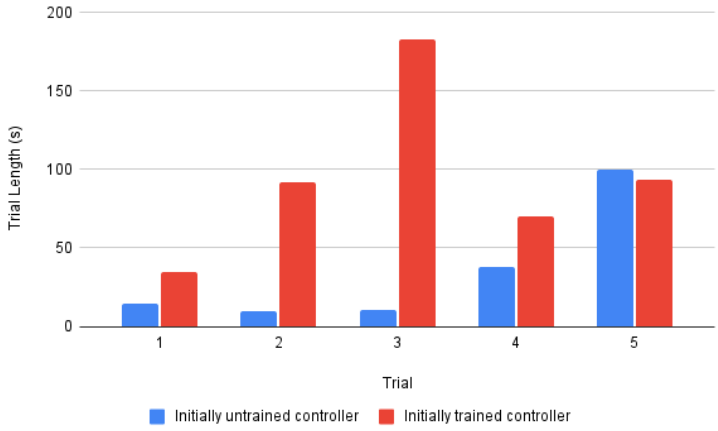

Figure 5.9: Performance of initially trained and initially untrained controller with 1 m pole length.

trial times than the initially naive controller. Figure 5.11 provides more details on this series of trials.

Lastly, the rail length was altered once more, this time to 1 meter. The initially trained controller performed better in all trials compared to the initially untrained controller. Like in the previous tests, the initially trained controller displayed a much higher capability of obtaining longer trials. Figure 5.12 displays the performance of each controller in the aforementioned context.

In every test that occurred in this section, the initially trained controller outperformed the initially naive controller in the majority of the trials. The initially trained controller consistently displayed the capability of performing longer trials than the initially untrained controller and also achieved higher average trial times in every experiment. This shows that the cerebellum computational model is indeed capable of transferring knowledge. More details of the comparison of the results obtained in each experiment can be seen in Table 5.3.

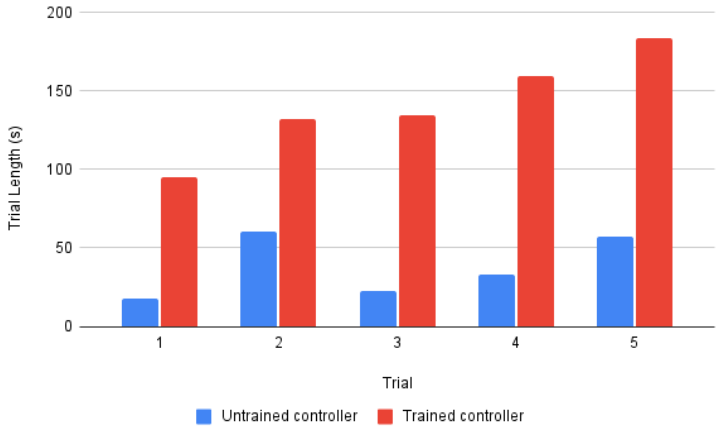

Figure 5.10: Performance of initially trained and initially untrained controller with 0.5 m pole length.

Table 5.3: Transfer of knowledge performance comparison.

Initially Trained?	Pole Mass (kg)	Pole Length (m)	Rail Length (m)	Max. Trial Time (s)	Min. Trial Time (s)	Average Trial Time (s)
No	2.0	0.5	5.0	64.47	4.53	29.04
Yes	2.0	0.5	5.0	212.75	38.04	103.10
No	4.0	0.5	5.0	22.39	2.67	8.77
Yes	4.0	0.5	5.0	53.51	8.67	23.74
No	1.0	1.0	5.0	99.94	9.36	34.41
Yes	1.0	1.0	5.0	182.72	38.04	94.61
No	1.0	0.25	5.0	60.65	22.9	38.29
Yes	1.0	0.25	5.0	183.70	95.4	141.04
No	1.0	0.5	2.0	108.85	5.86	58.52
Yes	1.0	0.5	2.0	206.16	64.93	115.22
No	1.0	0.5	1.0	13.21	5.75	8.94
Yes	1.0	0.5	1.0	83.75	42.98	60.76

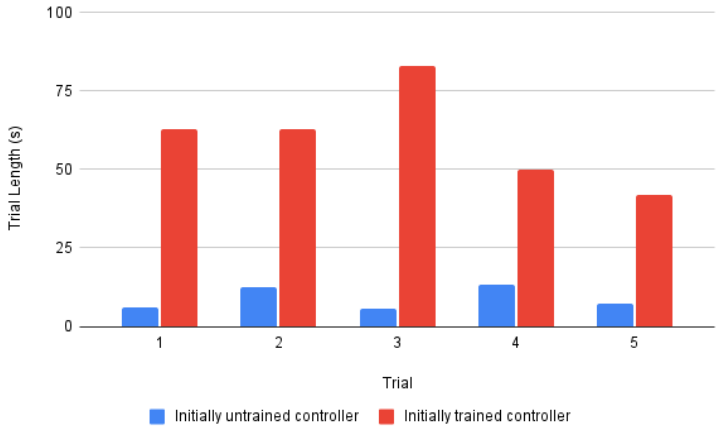

Figure 5.11: Performance of initially trained and initially untrained controller with 2 m rail length.

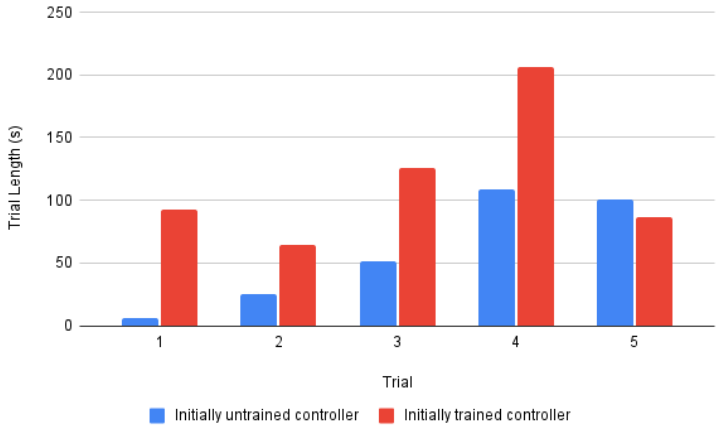

Figure 5.12: Performance of initially trained and initially untrained controller with 1 m rail length.

CHAPTER VI

CONCLUSION

The main goal of this thesis was to explore the capabilities of this innovative brain inspired controller. More specifically, it was to understand how it can perform tasks relating to maintaining balance. Due to the cerebellum's characteristics and role in human bipedal locomotion and maintenance of balance, expectations of its performance on these types of tasks were high. The cerebellum computational model was able to surpass our expectations in most tests, especially when it came to the inverted pendulum task. We initially expected that the controller would be able to balance the inverted pendulum for over a minute, but in reality this was an easy task for the innovative design as it obtained trails that were much longer, with the largest lasting over 13 minutes. The two largest networks used did not even obtain one trial under 60 seconds. On the double inverted pendulum, on the other hand, the controller performed closer to the expected. The two largest networks were able to relatively easily obtain trials over 15 seconds, but the smallest controller failed this task. The double inverted pendulum task is immensely more difficult than the inverted pendulum, so for a controller to be able to consistently balance two poles for over 15 seconds is quite impressive. Thus, our initial assumption that a cerebellum computational model would perform well in balancing tasks was correct.

Compared with currently commonly used reinforcement learning and supervised learning controllers, the cerebellum computational model displays a lot of potential. While the inverted pendulum is often considered a reinforcement learning problem [45], it can also be achieved using supervised learning adaptive controllers. Reinforcement learning controllers, such as the Q learning, can learn very complex tasks but tends to require lots of training in order to achieve this. Supervised learning adaptive controllers on the other hand, can learn quicker but tend to struggle to perform more complex tasks, such as the double inverted

pendulum. The cerebellum computational models' ability of quickly learning tasks along with its shown capability of learning highly complex tasks, such as the double inverted pendulum, highlights its potential in the field of robotics.

Another topic that was explored throughout this thesis is how network size affects the cerebellum performance. Our expectations were that the largest and second largest networks would perform similarly while the smallest would perform the worst of the three. This was partially true as the smallest network did not perform as well as the other two in both the inverted pendulum and the double inverted pendulum tasks, obtaining average trial times of 91.05 s and 12.95 s, respectively. On the other hand, the largest network showed to perform much better than the medium sized network in the inverted pendulum, with a 278.01 s average trial. On the double inverted pendulum, the improvement from the medium to large network is more subtle, achieving a 19.82 s average trial. With the inverted pendulum, both the medium sized and large sized network did not have any failed trials. On the double inverted pendulum, the larger network did not obtain any failed trials while the medium network obtained six. While the larger network clearly performed better than the medium sized network, the percent increase of the average trial time between the medium and the large network was smaller than the amount of growth between the small and medium network in both the inverted pendulum and double inverted pendulum tasks. A visualization of the average trial times for each network size for the inverted pendulum and double inverted pendulum can be seen in Figure 6.1 and Figure 6.2, respectively. Computational capabilities limited our efforts to obtain extremely large networks to be able to properly provide a definite answer to this question as it is likely that a 120 thousand neuron network is not large enough to possibly display signs of overfitting. We hope to address this question again in the future using a computer with more memory, allowing for much larger networks.

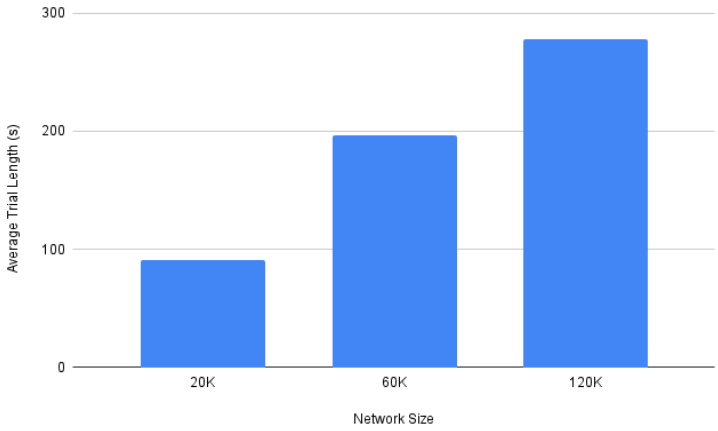

Figure 6.1: Comparison of average trial length for each network size on the inverted pendulum.

Lastly, the transfer of knowledge capabilities of the cerebellum inspired controls were analyzed. We expected the control design to excel in this application due to the biological characteristics of the cerebellum and prior experiments performed that displayed the controller's ability to adapt to different circumstances. This was tested by training a cerebellum inspired controller on the original model of the inverted pendulum. Parameters of the inverted pendulum were altered one at a time and the performance of the initially trained controller was compared to the performance of an initially naive controller of the same size. Our predictions were correct as the initially trained cerebellum computational model displayed an amazing performance, outperforming the initially naive controller in almost every trial. The trained model also obtained longer trails and a much higher average trial time in every test. This clearly shows that the cerebellum computational model, like its biological counterpart, is able to quickly adapt to new contexts and use prior knowledge to complete different tasks.

The natural follow up to this project would be to test the cerebellum computational

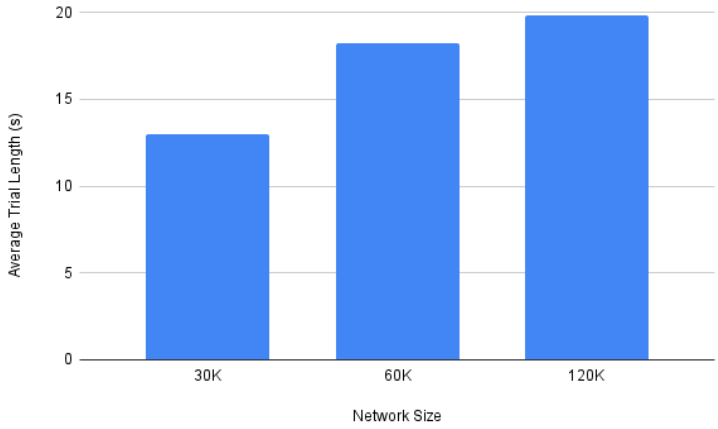

Figure 6.2: Comparison of average trial length for each network size on the double inverted pendulum.

model on bipedal locomotion. The inverted pendulum is many times the initial starting point in discussions of human balance and locomotion, as it is theorized that the human gait functions similarly to the inverted pendulum [46] [47]. The cerebellum's known role in maintaining balance and walking, makes this an even more interesting next step to take following this thesis. Another interesting next step for this technology would be to test it on maintaining the balance of a standing up humanoid robot.

Another concept of the cerebellum that could be more explored is its multitasking. As mentioned previously, the cerebellum can be split into several modules with the same simple circuitry, allowing for modules to focus on different tasks. This could be an interesting concept to perform experiments and attempt to understand how they learn and perform so many tasks at once. There are also many problems that are not particularly balance related that would be interesting to see this controller design being applied to. Some interesting future applications of this controller type would be on a prosthetic limb, attempting to allow for a more human-like movement or any robotic limb application.

Overall this innovative control design is quick, adaptable, and capable of learning complex tasks without much effort. With it being still in its embryonic phase of development, there are innumerable possibilities for the future of this technology.

BIBLIOGRAPHY

[1] J. S. Albus, "A new approach to manipulator control: The cerebellar model articulation controller (cmac)," *Journal of Dynamic Systems, Measurement, and Control*, vol. 97, 9 1975.

[2] B. Siciliano and O. Khatib, *Springer handbook of robotics*. Springer International Publishing, 1 2016.

[3] M. Fujita, "Adaptive filter model of the cerebellum," *Biological Cybernetics*, vol. 45, pp. 195–206, 10 1982.

[4] D. M. Wolpert and M. Kawato, "Multiple paired forward and inverse models for motor control," 1998. [Online]. Available: https://www.researchgate.net/publication/220361482

[5] T. Yamazaki and W. Lennon, "Revisiting a theory of cerebellar cortex," pp. 1–8, 11 2019.

[6] I. Abadia, F. Naveros, J. A. Garrido, E. Ros, and N. R. Luque, "On robot compliance: A cerebellar control approach," *IEEE Transactions on Cybernetics*, vol. 51, pp. 2476–2489, 5 2021.

[7] W. S. Mcculloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity* n," *Bulletin of Mothemnticnl Biology*, vol. 52, pp. 99–115, 1990.

[8] E. R. Kandel, J. H. Schwartz, T. M. Jessell, S. A. Siegelbaum, and A. Hudspth, *Principles of Neural Science, Fifth Edition*, 2013.

[9] J. C. Eccles, M. Ito, and J. Szentágothai, *The Cerebellum as a Neuronal Machine*. Springer Berlin Heidelberg, 1967.

[10] N. Schweighofer, M. A. Arbib, and M. Kawato, "Role of the cerebellum in reaching movements in humans. i. distributed inverse dynamics control," *European Journal of Neuroscience*, vol. 10, pp. 86–94, 1998.

[11] D. Marr, "A theory of cerebellar cortex," *J. Physiol*, vol. 202, pp. 437–470, 1969.

[12] J. S. Albus, "A theory of cerebellar function," 1971.

[13] M. Ito, "Long-term depression," *Annual Review of Neuroscience*, vol. 12, 3 1989.

[14] N. Schweighofer, "Computational models of the cerebellum in the adaptive control of movements," 1995.

[15] W. K. Li, M. J. Hausknecht, P. Stone, and M. D. Mauk, "Using a million cell simulation of the cerebellum: Network scaling and task articleity," *Neural Networks*, vol. 47, pp. 95–102, 11 2013.

[16] A. Antonietti, D. Martina, C. Casellato, E. D'Angelo, and A. Pedrocchi, "Control of a humanoid nao robot by an adaptive bioinspired cerebellar module in 3d motion tasks," *Computational Intelligence and Neuroscience*, vol. 2019, 2019.

[17] M. Hausknecht, W. K. Li, M. Mauk, and P. Stone, "Machine learning capabilities of a simulated cerebellum," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, pp. 510–522, 3 2017.

[18] M. Kawato, S. Ohmae, H. Hoang, and T. Sanger, "50 years since the marr, ito, and albus models of the cerebellum," pp. 151–174, 5 2021.

[19] J. L. Raymond and J. F. Medina, "Computational principles of supervised learning in the cerebellum," pp. 233–253, 7 2018.

[20] I. V. Tetko, D. J. Livingstone, and A. I. Luik, "Neural network studies. 1. comparison of overfitting and overtraining," *Journal of Chemical Information and Computer Science*, vol. 35, pp. 826–833, 1995.

[21] D. V. Raman, A. P. Rotondo, and T. O'Leary, "Fundamental bounds on learning performance in neural circuits," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 116, pp. 10 537–10 546, 2019.

[22] S. Zhou, M. K. Helwa, A. P. Schoellig, A. Sarabakha, and E. Kayacan, "Knowledge transfer between robots with similar dynamics for high-accuracy impromptu trajectory tracking," in *2019 18th European Control Conference (ECC)*, 2019, pp. 1–8.

[23] N. Schweighofer and M. A. Arbib, "A model of cerebellar metaplasticity," *Learning and Memory*, vol. 4, pp. 421–428, 1998.

[24] M. D. Mauk and N. H. Donegan, "A model of pavlovian eyelid conditioning based on the synaptic organization of the cerebellum," 1997.

[25] M. Ito, "The cerebellum and nural control," 1984.

[26] J. C. Houk, J. T. Buckingham, and A. G. Barto, "Models of the cerebellum and motor learning," *Behavioral and Brain Sciences*, vol. 19, 9 1996.

[27] A. H. Fagg, N. Sitkoo, A. G. Barto, and J. C. Houk, "A computational model of cerebellar learning for limb control," 1997.

[28] W. Maass, "Networks of spiking neurons: The third generation of neural network models," vol. 10, pp. 1659–1671, 1997.

[29] W. Gerstner and W. M. Kistler, *Spiking neuron models : single neurons, populations, plasticity*. Cambridge University Press, 2002.

[30] N. Schweighofer, M. A. Arbib, and P. F. Dominey, "A model of the cerebellum in adaptive control of saccadic gain i. the model and its biological substrate," *Biological Cybernetics*, vol. 75, pp. 19–28, 1996.

[31] ——, "Biological cybernetics a model of the cerebellum in adaptive control of saccadic gain ii. simulation results," *Biol. Cybern*, vol. 75, pp. 29–36, 1996.

[32] N. Schweighofer, J. Spoelstra, M. A. Arbib, and M. Kawato, "Role of the cerebellum in reaching movements in humans. ii. a neural model of the intermediate cerebellum," *European Journal of Neuroscience*, vol. 10, pp. 95–105, 1998.

[33] L. F. Abbott, "Lapicque's introduction of the integrate-and-fire model neuron (1907)," *Brain Research Bulletin*, vol. 50, pp. 303–304, 1999.

[34] A. L. Hodgkin and A. F. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve," *J. Physiol*, pp. 500–544, 1952.

[35] B. Fornberg, E. Larsson, and N. Flyer, "Stable computations with gaussian radial basis functions," *SIAM Journal on Scientific Computing*, vol. 33, pp. 869–892, 2011.

[36] S. Kuroda, K. Yamamoto, H. Miyamoto, K. Doya, and M. Kawato, "Statistical characteristics of climbing fiber spikes necessary for efficient cerebellar learning," *Biological Cybernetics*, vol. 84, pp. 183–192, 2001.

[37] D. P. Bertsekas and J. N. Tsitsiklis, *Introduction to probability.* Athena Scientific, 2008.

[38] R. Brette, M. Rudolph, T. Carnevale, M. Hines, D. Beeman, J. M. Bower, M. Diesmann, A. Morrison, P. H. Goodman, F. C. Harris, M. Zirpe, T. Natschläger, D. Pecevski, B. Ermentrout, M. Djurfeldt, A. Lansner, O. Rochel, T. Vieville, E. Muller, A. P. Davison, S. E. Boustani, and A. Destexhe, "Simulation of networks of spiking neurons: A review of tools and strategies," pp. 349–398, 2007.

[39] P. DeVries, *A first course in computational physics*, 1984.

[40] E. Suli and D. F. D. F. Mayers, *An introduction to numerical analysis.* Cambridge University Press, 2003.

[41] F. Naveros, J. A. Garrido, R. R. Carrillo, E. Ros, and N. R. Luque, "Event- and time-driven techniques using parallel cpu-gpu co-processing for spiking neural networks," *Frontiers in Neuroinformatics*, vol. 11, 2 2017.

[42] J. T. Spooner, M. Maggiore, R. Ordonez, and K. Passino, *Stable adaptive control and estimation for nonlinear systems : neural and fuzzy approximator techniques.* Wiley, 2002.

[43] S. Jadlovská and J. Sarnovský, "Classical double inverted pendulum-a complex overview of a system."

[44] M. F. Bear, B. W. Connors, and M. A. Paradiso, *NEUROSCIENCE EXPLORING THE BRAIN*, 4th ed., 2015.

[45] V. Gullapalli, "A comparison of supervised and reinforcement learning methods on a reinforcement learning task," in *Proceedings of the 1991 IEEE International Symposium on Intelligent Control*, 1991, pp. 394–399.

[46] J. Milton, J. L. Cabrera, T. Ohira, S. Tajima, Y. Tonosaki, C. W. Eurich, and S. A. Campbell, "The time-delayed inverted pendulum: Implications for human balance control," *Chaos*, vol. 19, 2009.

[47] A. D. Kuo, "The six determinants of gait and the inverted pendulum analogy: A dynamic walking perspective," *Human Movement Science*, vol. 26, pp. 617–656, 8 2007.