Simulating Cerebellar Learning of Internal Models

Grant R. Vousden-Dishington
Department of Bioengineering
University of California, San Diego
La Jolla, CA 92092
gvousden@ucsd.edu

Abstract

Many motor system impairments are debilitating not because they destroy previously acquired motor control policies but because they impair the ability to learn and incorporate error signals in moving. The cerebellum is of chief importance in nearly all motor learning tasks, and is a hypothesized location of storage for internal models important to movement. This model hypothesizes that plasticity in the parallel fibers that connect granule cells and Purkinje cells in the cerebellar cortex are the biological mechanism of policy learning. The model implements the granule cells and Purkinje cells as a neural network connected to a 2-degree-of-freedom simulated arm and two plasticity rules, one for long-term potentiation and one for long-term depression. The activity of the network and behavior of the simulated arm are captured and analyzed.

1 Introduction

Effective motor control is one of the nervous system's preeminent responsibilities. Mammals, especially primates, are embodied with dozens, if not hundreds, of degrees of freedom controlled by tens of billions of neurons; the inability to coordinate movement of a single body-part can have devastating effects on the health and survival chances of the animal. Even tunicates, a well-known aquatic filter feeder, possess cerebral ganglia during their larval stages, when they are mobile. After finding a suitable location, the tunicate becomes sessile and no longer requires its nervous system, so the organism digests its cerebral ganglion for nutrition instead. The motor tasks controlled by the nervous system range from limb movements to point, reach, and propel to vocal cord vibrations to utter specific sounds and speech. In spite of the innumerable examples of neural motor control, the description of the responsible neural mechanisms is largely incomplete.

No central nervous system (CNS) equipped organism is the same from birth to death; its motor systems (i.e. its body) grows and changes over the course of its lifetime, and it's because of this that a great deal of motor control is learned, not innate. From the perspective of an engineer, the brain must encode a *policy* that allows it to obtain a desired result. One of the simplest - but not trivial - desired actions many organisms can perform is to reach for or point to an object. This is the task explored in this project: given a target, a simulated neural controller must learn to guide a 2-degree-of-freedom simulated arm along a minimum-jerk trajectory with minimal error. The trajectory planner, playing the role of the motor cortex, will have its output added to a correction from a cerebellar model to improve its ability to follow the planned path. Learning is simulated by changes in the connection strength of particular synapses in the cerebellum. Section 2 discusses motor control in general, including the variables control problems most relevant to biological motor control. In addition, the section details the anatomy of the cerebellum and hypothesized functions that are implemented in this experiment. Section 3 explains the simulator, the global system architecture, and the recently proposed model of the cerebellum that is used in this experiment. Section 4 summarizes the results of the experiment and suggests interpretations of them, while section 5 reflects on the successes

and shortcomings of the experiment, as well as reviews the topics addressed in the course of the investigation.

2 Motor Control and the Cerebellum

Motor control can be conceptualized as a process with at least four steps: planning, enacting, observing, and learning. The action begins with a plan to achieve the desired task. In reaching tasks, the planning stage includes determination of the desired position and creating a trajectory to smoothly move the end effector from the starting position to target position; these trajectories often describe the desired acceleration and velocity of the end effector at each point as well, though both are often zero. An *end effector* is the general term for the object or body part performing the movement, but it is not constrained by a specific definition. The end effector in a reaching motion, for example, the end effector could just as well be an individual digit, the whole hand, or a tool, such as a wrench or pliers. Once a plan is made, the appropriate muscles can be activated to follow the trajectory. The resulting position and path the limb takes, however, can be very different from what was planned, especially if the limb's characteristics have changed since its last use or it has been injured in some way. Thus, the actor must observe is own movement, usually through vision or proprioception (i.e. the ability to sense and estimate the position of one's own body parts), and 'compute' the difference between the actual result and the intended result. If the difference between plan and reality is small, then the motor system has done its job, but if there is noticeable difference, then learning must take place to prevent similar errors in the future. Before explaining the learning implementation and the cerebellum's role in it, we first take a deeper look at what it means to plan a movement and trajectory.

Planning is believed to take place in the motor cortex, and it involves a (crude) transformation from desired end-point, and time allowed for the action, to a trajectory that describes where we expect the end effector to be at each point in time. When reaching between a start and end point, there are uncountably many paths that could be taken, but not all of them will be reasonable. Furthermore, there are many valid criteria possible for choosing which path to take: we may wish to minimize the amount of energy expended in movement, use the smallest number of muscle groups, or we may want to maximize the velocity of the end effector at a particular part of the trajectory (e.g. throwing a javelin). In this simulation, we use a common path planning method called a minimum-jerk model. The goal of such models is to minimize the 'jerk' in the movement, which means to minimize the number of sudden changes in direction of the limb's parts; minimum-jerk trajectories often have smooth position, velocity, and acceleration profiles. Fortunately, there is a straightforward way to mathematically define jerk: the third derivative of position or, equivalently, the (first) derivative of acceleration. With the jerk now defined, the cost of a given movement can be defined as the integral of squared-jerk over the time interval of the movement. The two preceding definitions are summarized in Equation 1. The square of the jerk is used because jerk can have positive or negative values. The scaling factor $\frac{1}{2}$ is only for derivation of the analytical solution, with no other special purpose.

$$C(t) = \frac{1}{2} \int x^{(3)}(t)^2 dt \tag{1}$$

To find the minimum-jerk trajectory then becomes a process of finding the appropriate parameters to minimize Equation 1, which is an integral over the temporal duration of the movement, typically 0.5 seconds. The values can be found using the calculus of variations technique - similar to using the derivative to find the minimum of a function as well as the technique of gradient descent - and integration by parts. A full explanation of the derivation is beyond the scope of this report, but the full proof can be found in the web resources associated with Shadmehr and Wise (2005). In general, the derivation reveals that the minimum jerk trajectory for a n-variable problem, with x and y position variables, is the same for many problems. 2 gives the general form of the minimum-jerk model, which we use to compute the trajectory in simulation. The variable d is the duration of the movement. Note that an important consequence of this solution is that, in two and three dimensions, the minimum-jerk trajectory is a straight line.

$$\vec{x}(t) = \begin{bmatrix} x_{1i} + (x_{1f} - x_{1i})(10(t/d)^3 - 15(t/d)^4 + 6(t/d)^5) \\ x_{2i} + (x_{2f} - x_{2i})(10(t/d)^3 - 15(t/d)^4 + 6(t/d)^5) \\ \vdots \\ x_{ni} + (x_{nf} - x_{ni})(10(t/d)^3 - 15(t/d)^4 + 6(t/d)^5) \end{bmatrix} \quad (2)$$

After planning, the action is performed while the cerebellum predicts the sensory outcome through the use of *internal models*. The output of the motor cortex is a trajectory, which may be in Cartesian coordinates but is more commonly converted to joint-angles and torques, but the motor system does not wait until after the movement is performed to assess how well the plan was followed. The cerebellum acts as an anterograde corrector; shortly after the motor command is sent to the muscles, the cerebellum transmits a corrective signal that is 'added' to the command from the motor cortex, with the result being a corrected motor command absent of any errors that were predicted to occur in the uncorrected movement. There are two types of internal models the cerebellum can use to assist in motor control: forward and inverse. As the names imply, they involve the same variables, joint-angles and torques, but the inputs and outputs are switched. Forward models are those that take torques (or some other analog of force) as input and compute what the sensory outcome will be (i.e. what will be the final joint-angles of the limb, and in more complex models, the expected sights, sound, and other percepts). Conversely, the inverse model receives an input of joint-angles, usually associated with the target position, and outputs the torques - more generally, the motor commands - that the model calculates will achieve that position. Several studies, including Passot, Luque, and Arleo (2010), from which the cerebellar model used in this simulation was derived, make the case that both types of models are encoded in the cerebellar cortex.

Specifically, the inverse model receives a copy of the desired state and efferents to motor corrective signal to the muscle, while a copy of the corrected motor command, known as an efference copy, is fed back as input to the forward model and used to predict the state outcome. The predicted position is made available to the trajectory planner, the motor cortex in biological interpretations, and can be used to recompute the minimum-jerk trajectory from the current, mid-movement position if it is much different from the intended position at that point in time. To be completely biologically faithful, the model should also take into account the timing delays of signals being passed along the axons, especially those going into and out of the cerebellar component. For the sake of simplicity, the simulations described in the next section omit axonal delays. The motor control and learning scheme just described is presented in Figure 2.

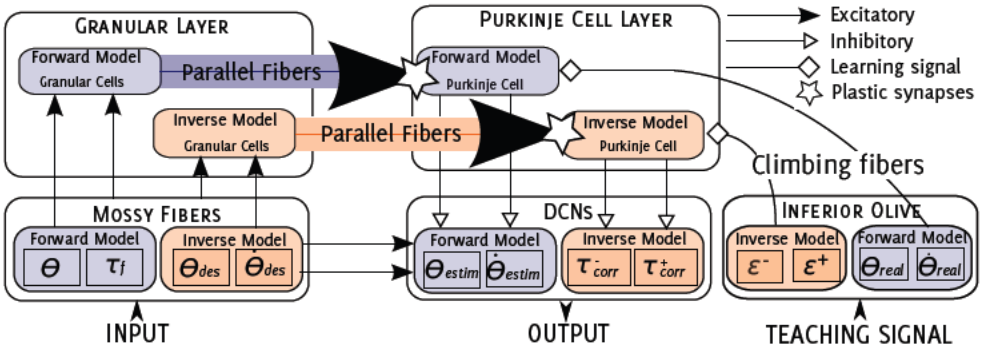

Figure 1: *A computational schematic of the cerebellar model. The inputs and outputs of the system are the mossy fibers and deep cerebellar nuclei (DCN), respectively. These areas are replaced with direct inputs and outputs to and from the simulated arm, in my model. The areas encapsulated in the neural network are the granule cells and Purkinje cells, connected by parallel fibers modeled as plastic synapses. Error feedback from the inferior olive is also given as a direct computation rather than a network representation.*

Having described the global architecture of the system, what remains is the internal anatomy of the cerebellar component. The cerebellar model is the primary component of interest in these simulations because of its well-observed adaptive learning, which are frequently interpreted as changes

in the forward and inverse models. Because both models are of interest for this experiment, every component of the cerebellum described herein is implemented twice, once for the forward model and once for the inverse model. The input neurons of the cerebellar model are mossy fibers[1]. In the inverse model, these fibers receive the desired state from the trajectory planning component of the motor cortex, while the forward model receives the efference copy of the command given to the limb. The primary role of the mossy fibers is to encode the input of the cerebellum via very specific activation; namely, each mossy fiber cell is made to fire at a maximum rate when the input has a specific value, known as its preferred direction, and to have a lower firing rate for values different from this preferred vector, with activation at or near zero for very distant values. This Gaussian activation profile is sometimes called a radial basis function, due to the activation of the neuron being essentially a function of the inputs 'radius' to its preferred direction. Many models, including the one presented in Passot, Luque, and Arleo (2010) and Carrillo et al. (2008), have only one neuron that responds at maximum firing rate to a given stimulus, though in the actual CNS there are probably redundant neurons having very similar activation profiles. This contributes to the robustness of the CNS encoding, but it can be ignored in small scale simulations like ours to increase performance.

After the mossy fibers receive input, they output to multiple *granule cells*. Biologically, the granule cells compose more of the CNS total neuron count (approx. 100 billion) than all other types of neurons, with an estimated 50 billion granule cells being present in the cerebellar cortex. Their vast numbers are compactly placed thanks to their extremely short axons and dendrites in comparison to neurons in the cerebrum, and a single mossy fiber can innervate several granule cells, and a single granule cell may be innervated by several mossy fibers, meaning a granule cell's input is a combination of inputs from several mossy fibers. It is possible that this allows the granule cells to encode 'combinations,' in the mathematical sense, of mossy fiber input, but as previously stated, hypotheses on the neural code are still speculative. However, what is known is that the granule cells play a significant role in learning for this study's model that will be explained after the remaining cerebellar regions are given.

In contrast to the small, densely packed granule cells, the *Purkinje cells* that receive output from them are among the largest neurons. Each Purkinje cell has a single axon, and they are the only known output of the cerebellar cortex, projecting to the deep cerebellar nuclei. Of additional note is that Purkinje cell axons are always inhibitory, meaning they reduce the likelihood of a deep nucleus's firing. Conversely, Purkinje cells have massive dendrites, and in vivo studies have shown a single Purkinje cell can receive inputs from as many as 100,000 *parallel fibers*, the axons of granule cells. Though possessing only one axon, it is capable of branching to many neurons after reaching the nuclei.

The deep cerebellar nuclei (DCN) are the output cells of the cerebellar component, delivering the forward model's predicted trajectory back to the motor cortex and transmitting the inverse model's corrective motor commands to the efferent nerves. The outputs of the models are in terms of Cartesian coordinates and torque, respectively. Aside from an inhibitory input from Purkinje cells, DCN receive excitatory input directly from the mossy fibers, so there is a direct connection between the input and output ensembles of the cerebellar model.

Motor learning of the entire network is implemented by plasticity in the parallel fibers, which is the result of granule cell activity and modulation by the activity of the inferior olivary nucleus. This area is usually abbreviated 'inferior olive,' or just 'IO,' and is responsible for delivering a teaching signal to the Purkinje cells that influence the synaptic weights of the parallel fibers. Inferior olive activity encodes the real state (position and velocity) values observed at the end of the time step, which is used to drive plasticity in the forward model. The error signal delivered to the inverse model is merely the discrepancy between the real state and the desired state at that timestep. The parallel fibers undergo two kinds of plasticity, long-term potentiation (LTP) described in Equation 3 and long-term depression (LTD) described in Equation 4, which are event-driven increases and decreases in synaptic strength, respectively. The LTP rule is a homosynaptic, non-associative rule, increasing the strength of a given parallel fiber by alpha if that fiber's granule cell spiked in the present timestep. The LTD rule, conversely, is an associative, heterosynaptic rule, because it is a function of the difference in timing between the spike from the inferior olive and the spike from the granule cell. The kernel K quantifies the correlation between granule cell and inferior olive spikes. In the Passot, Luque, and Arleo (2010) and Carrillo et al. (2008) experiments, this kernel gave its highest value when the granule cell spiked 100 ms before the inferior olive, because there was a total of 100 ms delay between motor command generation and error feedback from the sensory systems.

However, in simulations that ignore delay, a maximum value should be given for a difference near 0 ms, because granule cell spikes will occur in the same timestep that the inferior olive receives the error signal related to that command. For other values, there is an exponential decrease in the value given by the kernel. Figure 2 shows the response function used by Carrillo et al. (2008), to give the reader an idea of what the kernel function yields. With the two rules, parallel fibers will undergo bimodal plasticity, triggered when the granule cells fire, for LTP, and when the inferior olivary cells spike, for LTD. The parameters alpha and beta are known as the learning rate in plastic models and, in general, quantify how quickly the model learns new experience and replaces old knowledge; higher learning rates implicitly place higher value on the most recent experience while lower rates maintain learning that has already taken place and keeps the total learned policy stable.

$$\omega_{GC_i-PC_j}(t) = \omega_{GC_i-PC_j}(t) + \alpha\delta_{GC_i}(t) \tag{3}$$

$$\omega_{GC_i-PC_j}(t_{IO}) = \omega_{GC_i-PC_j}(t) - \beta \int_{-\infty}^{t_{IO}} K(t - t_{IO})\delta_{GC_I}(t)dt \tag{4}$$

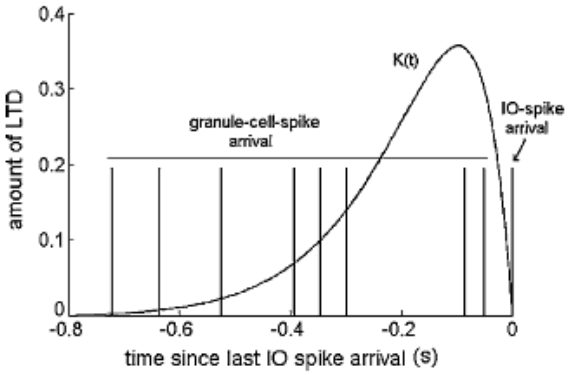

Figure 2: *The function determining the weight of LTD, modified from Carrillo et al. (2008). This model accounts for a 100 ms delay between motor command generation and sensory feedback which is not present in my model. As seen, the function is maximum when the granule cell spike takes place 100 ms before the spike from the inferior olive, the region that provides feedback.*

3 Experimental Setup: The Nengo Simulator

The simulations conducted for this experiment are all run in the Nengo simulator developed by the Centre for Theoretical Neuroscience at the University of Waterloo. Nengo is optimized for the simulation of networks containing hundreds or thousands of neurons; however, during testing it was noted that changes in the amount of memory allocated to the simulator were needed when components of the network contained 10,000 or more neurons. The workstation used for the simulations was not sufficient to do this, so the sizes of the networks used in the work by Passot, Luque, and Arleo (2010) were halved. Figure 3 shows the graphical front-end for Nengo, which allows visual, point-and-click style editing of networks. There are three essential levels of abstraction necessary to understand Nengo: nodes, ensembles, and networks. Nodes represent either individual neurons or computational variables (e.g. robot joint angles and velocities); the latter is only necessary if the user wants the neural network to interact with simulated, non-neural components, as we do here with the robotic arm. Ensembles are sets of nodes capable of having inputs (terminations) that are delivered to all constituent nodes and outputs (origins), that can be computed as direct node activity, decoded (i.e. computational) node values, or as electrical current. Single nodes also need not be encapsulated in an ensemble. Computational elements and functions can also be implemented with Nengo's input components. Networks are the highest level in the Nengo hierarchy, and every simulation must contain one network that encapsulates all other networks and ensembles. Networks cannot have terminations or origins unless one of its ensembles 'exposes' its termination or origin; this is how multiple networks can be connected together. Networks are useful for designating groups of neurons

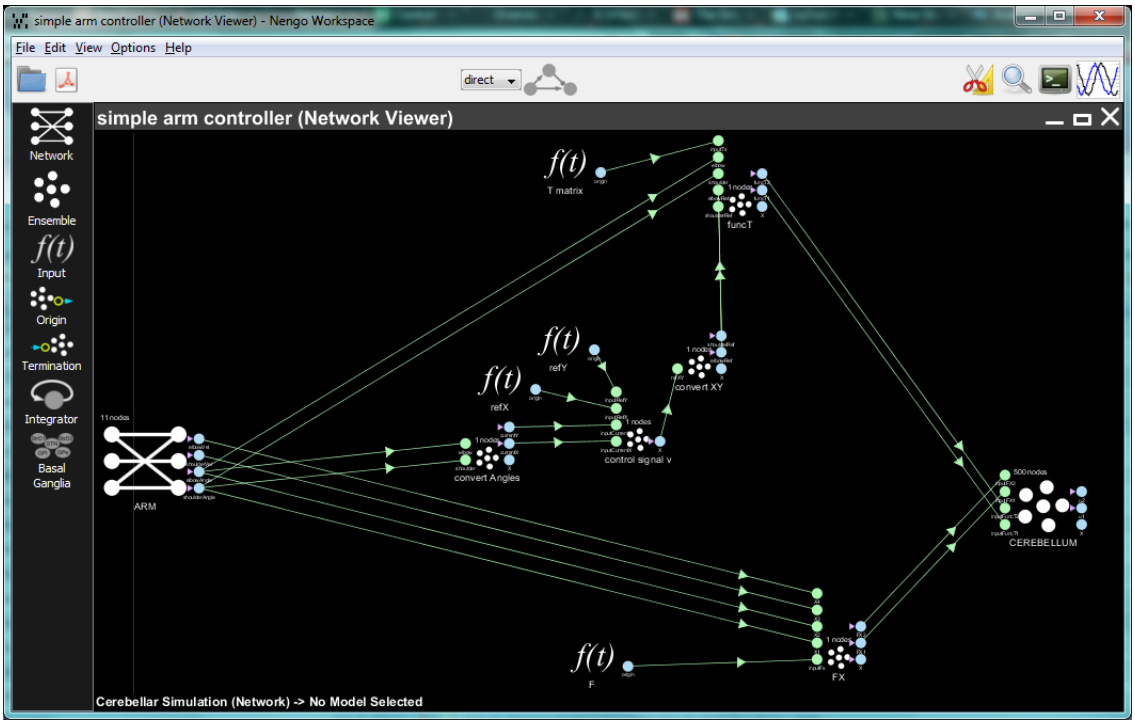

Figure 3: *The network architecture shown in nengo's visual interface. The cerebellar controller (far right) encapsulates the granule cells and Purkinje cells that control the simulated arm (far left). All other nodes are conversions from Cartesian coordinates to polar coordinates and representations of the arm location.*

the user wishes to simulate individually as well as in a larger, super-network, while ensembles are useful for separating neurons intended to perform different functions.

All of the components shown in Figure 2 are implemented as ensembles of varying sizes, encapsulated in a network that acts as the cerebellum, seen in Figure 3. The robotic arm and its dynamics are also implemented as a network containing nodes encoding its physical characteristics, position, and velocity. For all other components, while many design decisions are possible, the implementation is done using single-node ensembles because the present project is not interested in simulating these components in a neuromorphic manner. The only component being modeled as a neural network is the cerebellum. The reason that single-node representations wrapped in ensembles instead of just single nodes is that these components can receive inputs from other components and output their results to the next component in the loop; the node type in Nengo cannot have independent terminations and origins.

For all experiments, the circular target has unitless Cartesian coordinates between -1 and +1. In the same coordinate system, the simulated arm is comprised of two links, each possessing one degree-of-freedom, each of length 1. In the initial conditions of each experiment, unless otherwise stated, the arm is straightened vertically, placing the end effector at (0,2). As such, the target is horizontally 1 unit to the left of the arm's 'elbow.' For the end effector to reach this position, its elbow must bend 90 degrees. As with a real arm, the elbow can only bend one direction. To assess the activity of the neural network and arm, we will examine the model behaviorally, temporally, and potentiometrically. All recordings are taken from the ARM and CEREBELLUM components seen in Figure 3.

4 Results and Analysis

The activity of the network in terms of spikes and voltage are first examined, followed by the 'decoded' behavior of these networks.

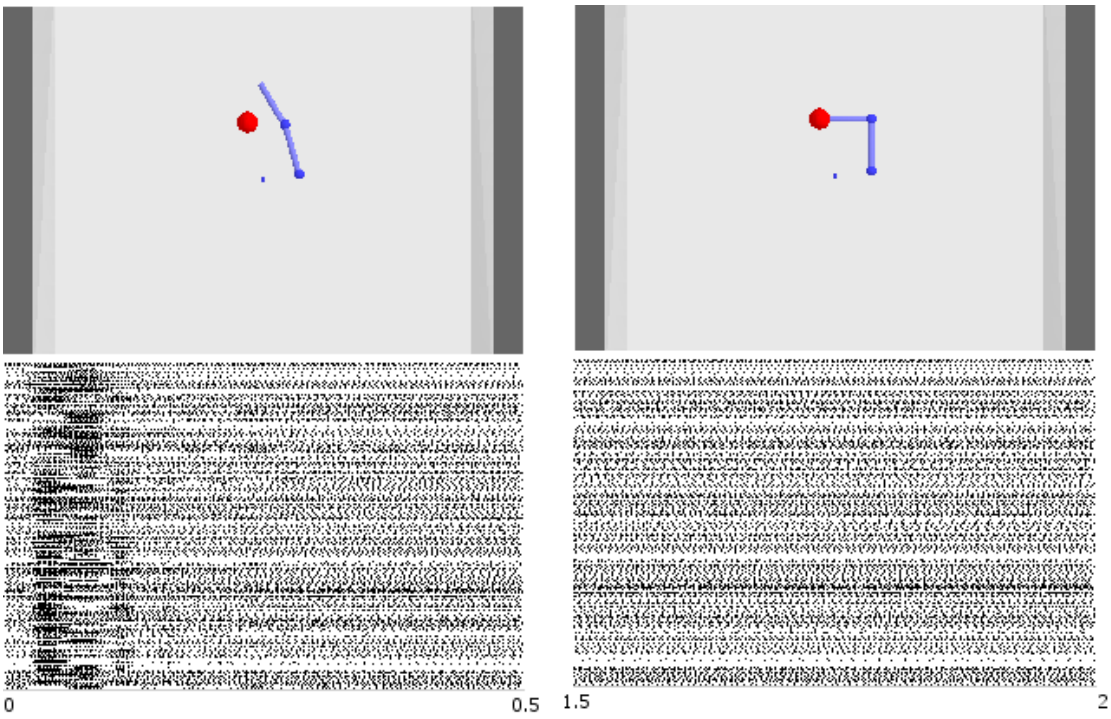

Figure 4: *The arm state and spiking activity of all neurons over the first 0.5 seconds of activity (left) and final 0.5 seconds of activity (right). Large shifts in activity take place immediately following initiation of movement, but spiking activity does not visibly change significantly during for most of the movement duration.*

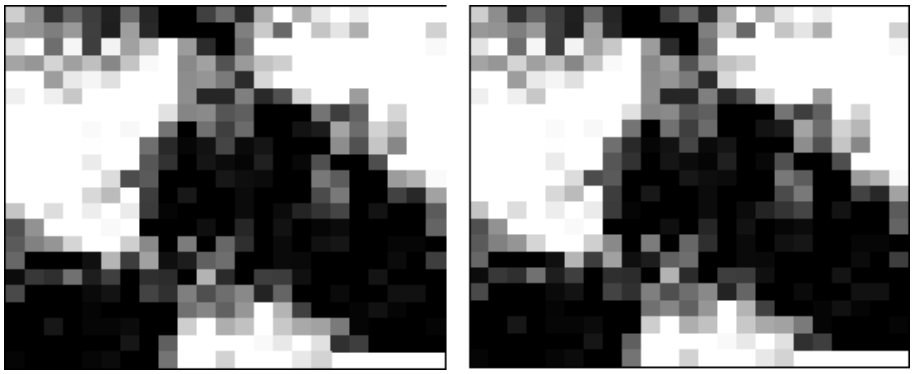

Figure 5: *The relative voltage of all neurons in the cerebellar network (left) vs. the rates of those neurons (right). The direct proportionality between the two is obvious.*

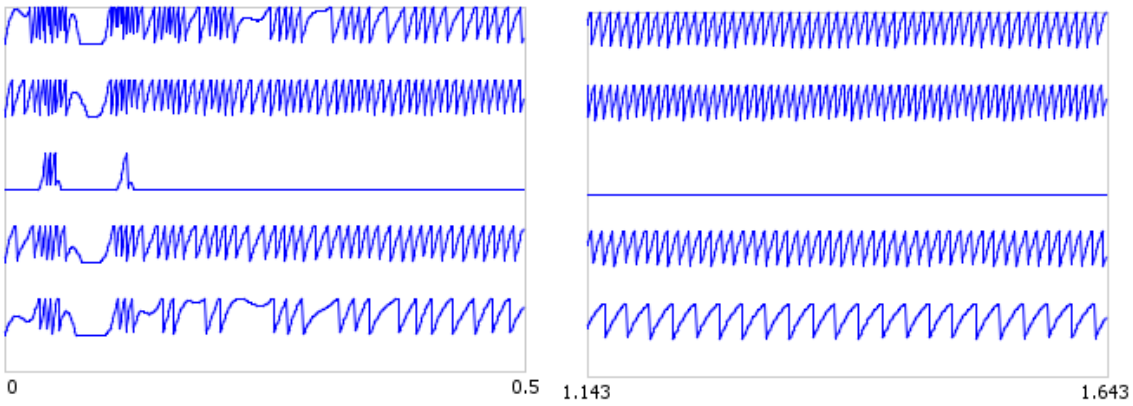

Figure 6: *The potential of five random neurons with respect to time. During the initiation of movement, there is much greater variability in the spiking pattern of each neuron. As the movement progresses and the end effector is closer to the target, the regularity of the patterns increase.*

4.1 Temporal Analysis

The Nengo simulator allows us to display a spike raster of every neuron in the cerebellar network while the arm is in motion. Figure 4 shows two intervals of activity: the first half-second of activity after the simulation is started and the last half-second, when the arm has stopped moving more than 0.001 units in either dimension. Not only is the activity in the 200 ms following initiation most noticable, there is a sequential nature to the activation. The spike raster has been automatically sorted to better show this activity, by placing neurons in rows closest to those neurons that match its rate and phase. The neurons respresented in the lower-half of the spike raster activate briefly, and as they return to previous levels of activity, the neurons in the upper-half of the raster increase in activity. One possible explanation for this concerns the motion of the arm itself.

As the arm begins to move, it plans on a straight-line trajectory, as mentioned in the background previously. However, the optimal movement is to rotate just the elbow until the end effector reaches the target. This final state is not apparent to the computation, so the controller moves the entire arm, making it necessary to reverse the movement of the first link in the arm. The opposing phase of firing for the lower-half of the raster versus the upper-half. Examining the decoded activity of the network can indicate the likelihood of this hypothesis. No cerebellar plasticity is observed in this portion of the movement.

4.2 Potentiometric Analysis

Figure 5 demonstrates the well-known relationship between each neuron's voltage and firing rate. The images are taken 0.1 seconds after the start of the simulation, placing the activity within the range seen in the spike raster of Figure 4. Once again, the neurons are arranged such that those that activate in tandem are spatially close. While this is not the arrangement of neurons we would expect to appear in a biological specimen, it is realistic to assume that neurons firing together in the same system will be closely linked in the network topology. To examine the change in voltage over time, we show the outputs in Figure 6. This shows with greater detail the potentials driving the spike rasters in Figure 4. The curved depolarization and sudden hyperpolarization is typical of leaky integrate-and-fire neuron models.

The network response functions have many different shapes, as shown in Figure 7. Multiple types response functions are observed in these five neurons. Those shown in red, black, and pink display a gap in responsiveness, having Gaussian/radial-basis response functions outside of this range. The other neurons also observe a Gaussian activation function, but the Gaussian is continuous and complete over a single interval with no gaps. The Gaussian response is a characteristic tuning curve of neurons involved in motor control. While a single neuron may fire only to a very specific input, there will rarely be only one neuron firing at any given time; this biological redundancy ensures that

the loss or defect of a single neuron will have minimal impact on the model. In addition, this makes it much more difficult for plasticity to shift an internal model encoded in the cortex.

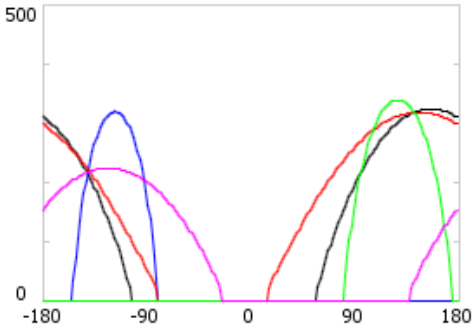

Figure 7: *The responses of the same five neurons to inputs, shown on the x-axis. There are a variety of response functions instantiated in the network, some which are more linear than others.*

4.3 Behavioral Analysis

To get a better idea of how these neural populations are being interpreted by the arm, we decode the activity using the neurons' tuning curves and sum their outputs to determine what the command to the arm will be. Figure 8 shows the decoded activity during the initial and final intervals of the movement. The commands are analogous to the activation of muscles and the force applied by them, not commands regarding the absolute position of the arm. The oscillations and sudden changes in activity during the 0 to 0.5 second interval are consistent with the changing spike rates seen in 4. The blue line corresponds to the command for the first link, before the elbow, and the black line corresponds to the activity of the second link. Neither link totally ceases activity until the end of the simulation, but the second link's ultimate displacement is much greater, as can be calculated by taking the integral over the black curve. While the total movement of the first link is similar to the second, it spends as much time rotating away from the target as toward, so it's displacement over the simulation is minimal. Over one trial, no learning could take place and this flawed policy remains unchanged.

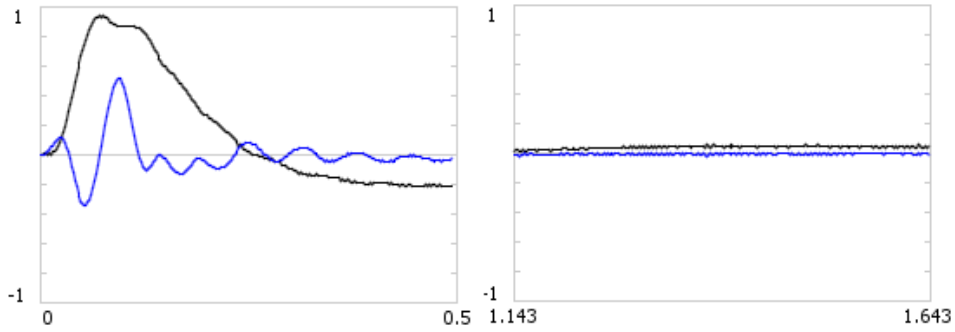

Figure 8: *The decoded motor commands sent to the arm by the cerebellar cortex. The oscillating pattern of the commands, especially the transitions between positive and negative early on, are indicative of changing motor intentions as the*

5 Discussion

The aim of this project was to test the change in accuracy of total movement as a function of neural network plasticity. Unfortunately, the implemented learning rules failed to work properly, so the investigation of motor learning in this network is not yet complete. Still, the simulation was able

to demonstrate why motor learning is needed at all. A smooth and accurate movement would have a bell-shaped velocity profile and a single-wavelength sinusoid acceleration profile. The forces generated in Figure 8 do not very well match either of these predictions, indicating that the neural controller has a flawed motor policy.

The simulations also demonstrated different types of tuning curves and activation functions for each neuron, indicating that the network has a heterogenous cluster of neurons that are not identical. This is a desirable feature biologically, because multiple neurons can partially encode the same information and thus be resistant to the error or destruction of any one part of the network. This heterogeneity also minimizes the waste of neurons, because rarely will any two neurons encode exactly the same information, even if their activations are heavily overlapped.

Although we have not shown an appreciable degree of learning in the network used for this experiment, there is a clearly presented need for an optimization policy. By implementing counterbalancing potentiation and depression functions, we expect those neurons that contribute to smooth and accurate movements to have their connections strengthened and those that do not, or those which may be suited to other kinds of movement than the one in progress, to ultimately be removed. However, there are many types of potentiation and depression rules available and being researched. Future work will thus implement these learning rules more carefully and measure them against the behavioral and psychophysical results found in humans and mammals.

Acknowledgments

I thank Dr. Gert Cauwenberghs for his instruction in and out of the classroom and Jeff Bush for all of his patience and help in answering questions for this and other class assignments.

References

[1] Trevor Bekolay. Using and extending plasticity rules in Nengo. Centre for Theoretical Neuroscience technical report. September 2010.

[2] Richard R. Carillo et al. A real-time spiking cerebellum model for learning robot control. Biological Systems, 94(1-2):18-27, 2008.

[3] Chris Eliasmith & Charles H. Anderson. Neural engineering: computation, representation, and dynamics in neurobiological systems. Computational neuroscience. MIT Press, 2003.

[4] Jean-Baptiste Passot, Niceto Luque, and Angelo Arleo. Internal models in the cerebellum: a coupling scheme for online and offline learning in procedural tasks. Lecture notes in computer science, From Animals to Animats, 11:435-446, 2010.

[5] Stephen H. Scott. Optimal feedback control and the neural basis of volitional motor control. Nature Reviews Neuroscience,5:532-546, 2004.

[6] Reza Shadmehr and Stephen P. Wise. The computational neurobiology of reaching and pointing: a foundation for motor learning. MIT Press, 2005.

[7] Thomas P. Trappenberg. Fundamentals of Computational Neuroscience, 2nd edition. Oxford University Press, 2008.

[8] Daniel M. Wolpert, R. Chris Miall, and Mitsuo Kawato. Internal models in the cerebellum. Trends Cognitive Science, 2: 33847, 1998.