CLOSING THE LOOP: OPTIMAL STIMULATION OF NEURONAL NETWORKS VIA ADAPTIVE CONTROL ALGORITHMS

Julia Santos

A thesis
submitted in partial fulfillment of the
requirements for the degree of

Master of Science

University of Washington
2015

Committee:
Eli Shlizerman
Mark Kot

Program Authorized to Offer Degree:
Applied Mathematics

University of Washington

Abstract

CLOSING THE LOOP: OPTIMAL STIMULATION OF NEURONAL NETWORKS VIA ADAPTIVE CONTROL ALGORITHMS

Julia Santos

Chair of the Supervisory Committee:
Professor Eli Shlizerman
Applied Mathematics

The *Caenorhabditis elegans (C. elegans*) worm is a well-studied biological organism model. The nervous system of *C. elegans* is particularly appealing to study, since it is a tractable fully functional neuronal network for which electro-physical connectivity map (connectome) is fully resolved [1,2]. In this work, we use a recently established computational dynamical model of the *C. elegans* nervous system, which incorporated the static connectome data with intrinsic properties of neurons and their interactions. With this model, it has been demonstrated that robust oscillatory movements in motor neurons along the body can be invoked by constant current excitation of command sensory neurons (e.g., PLM neurons associated with forward crawling), and that their activation corresponds to low-dimensional Hopf bifurcation [3]. While these first results validated the model, it is exciting to learn and visualize how the nervous system transforms its oscillatory dynamics to the muscles to support robust full body movements (e.g., forward crawling) [4]. Moreover, it is intriguing to understand the optimal sensory stimulations that cause these movements to persist.

We explore these questions by developing methods to visualize network activity in a physical space and creating a model for *C. elegans* musculature as a viscoelastic rod with discrete rigid segments [5]. We map the neuronal dynamics such that they activate the muscles and deform the rod. When motor neuron activity stimulates muscles [2], this activation is translated into force applied to the rod, which moves in accordance with the physical properties of *C. elegans*. By stimulating the command PLM neurons, we establish for the first time that motor neuron dynamics are indeed producing coherent oscillatory full body movements that resemble forward crawling.

We utilize our computational full body model to determine the appropriate sensory input for behavior, such as crawling, to persist after explicit external stimulation (touch) has ceased, as observed in experiments [5]. Since such persistence could be explained by a feedback loop between the environment and sensory neurons, we propose an adaptive control algorithm that extends existing recursive least squares-based algorithms (e.g., FORCE [6]). The RLS algorithm is divided into training and operational phases. In the training phase, we reduce the error between desired and actual outputs by making small, rapid modifications to the weights which are applied to the network input (feedback). When the weighted feedback into sensory neurons prompts the system to produce the desired output without significant weight modification between iterations, a correct set of weights has been found [6]. We use a low-dimensional projection of motor neuron dynamics to calculate expected and actual output, and our algorithm is capable of finding sensory input patterns that will lead to the desired movement.

Supplementary Videos

1. Visualization of neuronal activity:

   celegans_neuronal_activity.avi

2. Visualization of movement using low-dimensional coefficients:

   celegans_low_dim_neuronal_activity.avi

3. Viscoelastic rod-based movement simulations

   celegans_combined_simulations.avi

Acknowledgements

Foremost, I would like to express my sincere gratitude to my advisor, Professor Eli Shlizerman, for his instruction, guidance and support throughout the research and writing process. I could not imagine a more contagiously enthusiastic and inspiring advisor.

I would like to thank Professor Mark Kot for his time and willingness to be on my thesis committee.

I also greatly appreciate the help of my classmate, Judith Moore, and her valuable editing skills.

Last, but certainly not least, I would like thank my family: my parents, Carolyn and Lewis Merryman, for encouraging my interests in math and science, and my husband, David Santos, for his endless support and confidence in my abilities.

Table of Contents

I. INTRODUCTION

With its well described and classified repertoire of physical behaviors and its neural network comprised of only 302 neurons, *Caenorhabditis elegans* is an ideal organism for the study of neuron stimulation and physical response [7]. Through cell staining and electron microscopy, all of *C. elegans'* neurons have been identified and neuron types mapped, allowing it to become the first organism to have its full connectome constructed. More recently, a full computational model of the *C. elegans* neural network was implemented to represent the system's dynamics in response to time-dependent stimuli [1]. This model provides an interactive dynamical system through which we can simulate network behavior, enabling further study of neuronal connectivity and interaction, along with broader system dynamics [1,3].

Even with *C. elegans'* fully-mapped connectome and a computational model of its neural network, the path from sensory stimulation to physical response is complex; it requires integrating the model of the complex nonlinear dynamic network with a representation of muscle activation and subsequent physical action. As such, we focus on understanding the interactions between network dynamics, muscle movement, and the physical environment in the particular behavior of forward crawling.

Links between full stimulation of neurons in the *C. elegans* network and specific physical behaviors have been established in the case of forward motion (crawling) [8]. Research has shown that when the pair of posterior lateral microtubule (PLM) touch receptor neurons are stimulated at specific levels, a supercritical Hopf bifurcation occurs and the system enters a limit cycle. When PLM neurons are stimulated at slightly higher levels, analysis of the network dynamics shows low-dimensional oscillatory patterns indicative of forward motion. Thus, the

behavior of the *C. elegans* neural network during forward motion is simulated by applying specific constant external input to the PLM neurons [3].

To gain further intuition into the behavior of the *C. elegans* neural network, we create dynamic visualizations of the activity of individual neurons during PLM excitation. At small intervals of time, we calculate the level of activity of each neuron relative to its equilibrium and plot the approximate location of the neuron based on its excitation status and approximate body location [9]. We sequentially combine these plots to create videos that allow us to observe the way the activity of individual neuron changes over time. These videos also help us recognize general trends in neuron behavior that we subsequently will investigate through filters and data analysis.

Moving beyond visualizations of the activity of the neural network itself, we seek to model the impact of neural excitation on muscle contraction and ensuing body movement. Representing the *C. elegans* body and muscle structures as a viscoelastic rod with discrete segments [4], we use existing maps between neurons and muscles [10] to approximate muscle stimulation based on neuron activity. To physically imitate muscle movement, we apply force to segments of the rod based on the input from connected neurons. We once again update neuron activity and muscle stimulation at small intervals in time and visualize the sequential motion of the viscoelastic rod as it models the physical behavior of *C. elegans* based on the state of its neural network.

Lastly, we strive to close the loop between neuron stimulation and forward movement in *C. elegans*. Computational simulations of prolonged forward motion can be generated by supplying constant external input to PLM neurons [3]; however, it is clear that in reality *C. elegans* continues to swim for a substantial amount of time after external stimulation (touch) has

ceased [5]. This extended period of movement could be explained by proprioceptive excitation, or a feedback loop in which physical movement excites sensory neurons like PLM, which in turn activate motor neurons that sustain the swimming motion [11,12]. Leveraging our physical model of *C. elegans* movement, we implement an algorithm based on recursive least squares [6] to strategically search for a feedback relationship between motor neuron activity and sensory neuron stimulation. Specifically, we seek a map from motor neuron output to sensory neuron input that will perpetuate forward motion dynamics in the neural network.

II. VISUALIZATION OF *C. ELEGANS* NEURONAL ACTIVITY DURING FORWARD MOTION

In order to gain better understanding of the behavior of the *C. elegans* neural network, we first model the physical form of the neural network. As a basis for this physical model, we leverage neuron activity data (Figure 1A) from simulations of *C. elegans* forward motion based on the single-compartment membrane equation (1) as defined in [3]. The equation

$$C\dot{V} = -G^C(V_i - E_{cell}) - I_i^{Gap}(\vec{\boldsymbol{V}}) - I_i^{Syn}(\vec{\boldsymbol{V}}) + I_i^{Ext}, \quad (1)$$

where $I_i^{Gap} = \sum_j G_{ij}^g(V_i - V_j)$ (2)

and $I_i^{Syn} = \sum_j G_{ij}^s s_j(V_i - E_j)$, (3)

uses total cell capacitance C, leakage conductance G^C, leakage potential E_{cell}, external input I^{ext}, and neural interactions through synapses and gap junctions, I^{Syn} and I^{Gap} (2,3), to model membrane potential over time. In equation (2), G_{ij}^g is the total conductivity of gap junctions between neurons i and j defined in the connectome data, and in equation (3), G_{ij}^s is the maximum total conductivity of the synapses between neurons i and j regulated by the synaptic activity variable

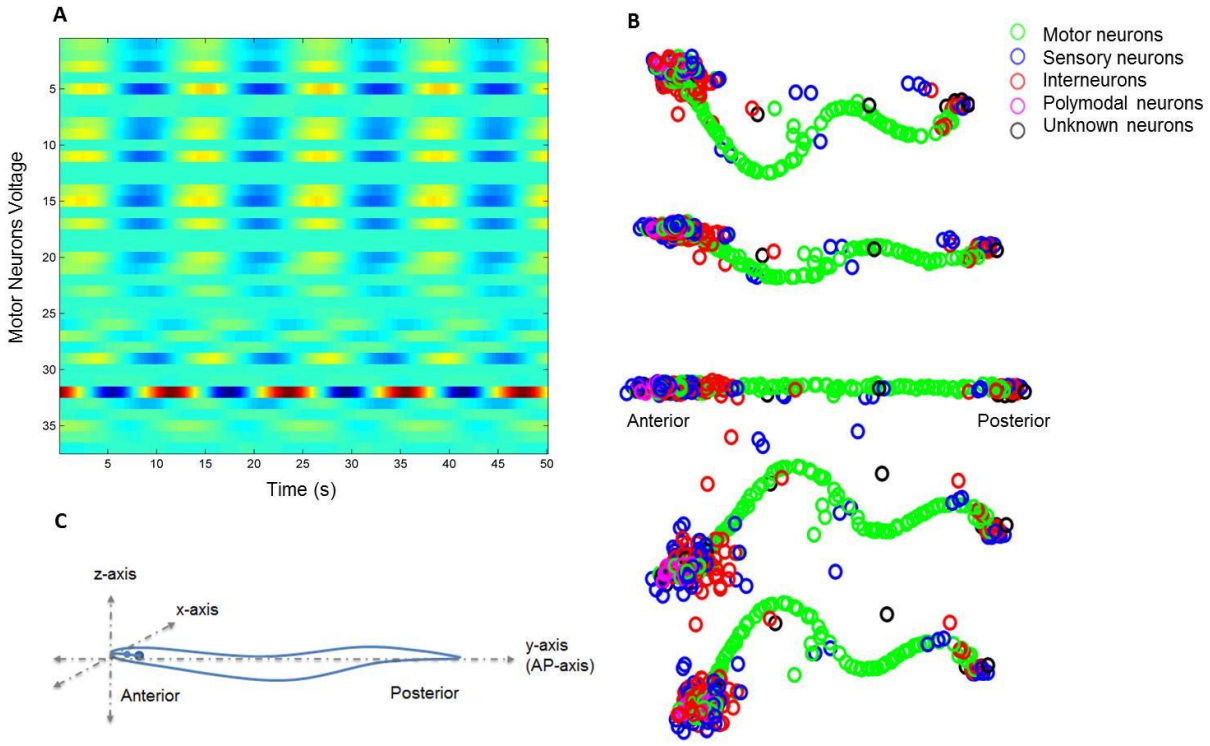

Figure 1. A) Motor neuron voltage oscillations during forward motion **B)** Layout of the axes used in transforming the position of neurons in *C. elegans*. During forward motion, the worm is undulating in the z-direction. The position of the neurons relative to the y-axis changes as the worm undulates. Since no rotation is included in the simulations, the position of the worm relative to the x-axis remains constant. **C)** Five physical representations of the *C. elegans* neural network. Undulation with the head upward corresponds with positive motor neuron activity (red stripes in figure A) and undulation with the head downward corresponds with negative motor activity (dark blue stripes in figure A).

$$s_i = a_r \phi(V_i; \beta, V_{th})(1 - s_i) - a_d s_i. \tag{4}$$

In Eq. (4), a_r and a_d are the activity's rise and decay time scale coefficients respectively, and ϕ is the sigmoid function

$$\phi(v_i; \beta, V_{th}) = \frac{1}{1 + \exp\left(-\beta(V_i - V_{th})\right)}. \tag{5}$$

We then seek to transform simulated voltage activity into a physical representation of network dynamics, allowing us to visualize neuronal activity along with neuron type and connectivity. The visual interpretation of individual neurons is particularly helpful in

investigating the voltage oscillations that occur in *C. elegans* during forward motion. Our visualizations are also valuable in analyzing the dominant patterns (modes) derived from the simulated forward motion neuronal activity.

A. Calculation of Neuron Coordinates in Varying Body States

The *C. elegans* neuronal map is first visualized by creating three-dimensional plots of all of the neuron locations based on data from [9]. This data provides coordinates for each *C. elegans* neuron in Euclidean space (Figure 1B), forming a caricature of the worm undulating in the z-direction. By combining physical location with neuron names, types and connectivity, we have the ability to create a variety of informative visualizations.

Our derived neuron coordinates assume that *C. elegans* is undulating with its head upward, which is a body form associated with forward motion. Since these coordinates assume a specific body position, we transform them to approximate neuron locations when the body of the nematode is in different states. First, we differentiate between an actively swimming and a resting *C. elegans* by approximating the location of the neurons when the body is in a straightened equilibrium state.

To straighten the upwardly undulating form, we first calculate the approximate physical length of the nematode's neural network. As there are hundreds of neurons in the network, we smooth the shape by averaging each z-coordinate with its four closest neighbors. This allows us to generate a better approximation of the high-level form of the network, from which we derive its length. We transform the y-coordinate of each neuron to the straightened form based on the calculated network length and data describing the location of each neuron along the anterior-posterior (AP) axis of the body [10]. With this information, we scale the y-coordinates from [10] to match our existing coordinate range.

We generate z-coordinates for the resting network by evaluating the z-coordinate of each neuron relative to the neurons surrounding it. For example, if one neuron has a higher z-value than its neighbors in the undulating state, it continues to be positioned higher than proximal neurons in the relaxed state. The relative z-coordinates are calculated by averaging z-coordinates of small groups of neurons and considering the z-value of each neuron relative to the group's average. The mean of the z-coordinates is removed from each z-value, and the resulting z-value is scaled by the standard deviation to avoid exaggerated scattering in the z-direction. The x-coordinates of the *C. elegans* neurons are untouched as we convert from upwardly undulating to the resting state since no rotation is applied in this transformation.

From observations of forward motion in *C. elegans* [2], we know that the swimming movement often follows a periodic wave, such that the head can point either upward or downward. As we want to be able to model both cases, we next approximate the coordinates when *C. elegans* is undulating with is head down. We continue to assume that the body of the nematode is not rotating in its transformations, which means that all of the downward undulation coordinates are not a reflection of the upward undulation coordinates. To achieve consistency in representing the relative location of the neurons, we classify certain neurons as representing AP-axis of the *C. elegans* body, such that their downward z-coordinates are a reflection of the upward coordinates about the y-axis. We then calculate the z-coordinates of the remaining neurons relative to the AP-axis, such that they remain on the same side of the axis in both the upward and downward undulation models.

B. Transformations Modeling *C. elegans* Forward Motion

In order to create dynamic visualizations of the activity of the *C. elegans* neural network during forward motion, we combine our three physical models of the C. elegans neural network

(upward/downward undulation and resting) with simulated neuron activity data [3] and form a map between neuron activity and position (Figure 1C). We consider highly active neurons to be represented by their undulating coordinates, whereas inactive neurons are represented by their resting coordinates. To accurately position a neuron based on its simulated activity, we calculate the equilibrium (a_{eq}) and the largest positive (a_{pos}) and negative (a_{neg}) activity levels for each neuron, such that $a_{neg} < a_{eq} < a_{pos}$. We then map a_{pos} to the upward undulation coordinates, a_{neg} to the downward undulation coordinates, and a_{eq} to the resting coordinates.

During forward motion many neurons have activity levels that oscillate between a_{pos} , a_{eq}, and a_{neg} over time. When activity is not at a maximal state and the neuron is not at rest, the physical position of the neuron is scaled based on the activity relative to the benchmark states. For example, if a neuron is at half of its a_{pos} activity level, we calculate the y and z-coordinates to be halfway between the upwardly undulating and resting locations, with x-coordinates remaining unchanged. Thus, the relationship between neuronal activity and physical location can be described by

$$y = \begin{cases} y_{eq} + \left(y_{pos} - y_{eq}\right) * \dfrac{a - a_{eq}}{a_{pos} - a_{eq}}, a > a_{eq} \\ y_{eq} - abs\left(\left(y_{eq} - y_{neg}\right) * \dfrac{a - a_{eq}}{a_{neg} - a_{eq}}\right), a < a_{eq} \\ y_{eq}, a = a_{eq} \end{cases} \tag{6}$$

$$\text{and } z = \begin{cases} z_{eq} + \left(z_{pos} - z_{eq}\right) * \frac{a - a_{eq}}{a_{pos} - a_{eq}}, a > a_{eq} \\ z_{eq} - abs\left(\left(z_{eq} - z_{neg}\right) * \frac{a - a_{eq}}{a_{neg} - a_{eq}}\right), a < a_{eq} \\ z_{eq}, a = a_{eq} \end{cases}. \tag{7}$$

We apply our map between neuron activity and position to data from simulations of *C. elegans* forward motion to generate visualizations of the network dynamics over time as videos. In these videos, neurons move up the positive z-axis as they approach a_{pos}, down the negative z-axis as

they approach a_{neg}, and stay near z = 0 as they near a_{eq}. With simulation data providing updated neuron activity values at intervals of 0.01s, we are able to clearly visualize the neurons oscillating over time.

C. Basic Network Activity Filtering for Visualization

Although we are easily able to see oscillations in the neurons from simulated forward motion data, it is clear that not all neurons are oscillating in phase with each other. To improve our understanding of how groups of neurons oscillate together, we filter simulation results to observe the behavior of neurons that have similar initial voltage changes. Neurons are filtered by their average voltage change over the first 0.1s of the simulations, and a neuron was only drawn if its initial mean voltage change was greater than the median voltage change, $n_{vis} = \{ n_j \; s.t. \; \sum_{t=1}^{10} vdiff_t(n_j) > \widetilde{vdiff}_{t=[1,10]} \}$.

The results of these visualizations show that, generally, neurons with activity levels that begin oscillating with voltage increases greater than the median change continue to oscillate together throughout the duration of the simulation. Likewise, we can see that the neurons with activity levels that begin oscillating with voltage decreases less than the median change each achieve their maximum and minimum voltages throughout the simulation at similar times. This broad filtering provides a high-level perspective of the oscillation trends in the *C. elegans* neural network, and motivates further investigation through principal component analysis.

D. Principal Component Analysis of Simulated Forward Motion Neural Activity

In order to more rigorously investigate the patterns of the *C. elegans* neural network during forward motion, we perform principal component analysis (PCA) on neuronal activity

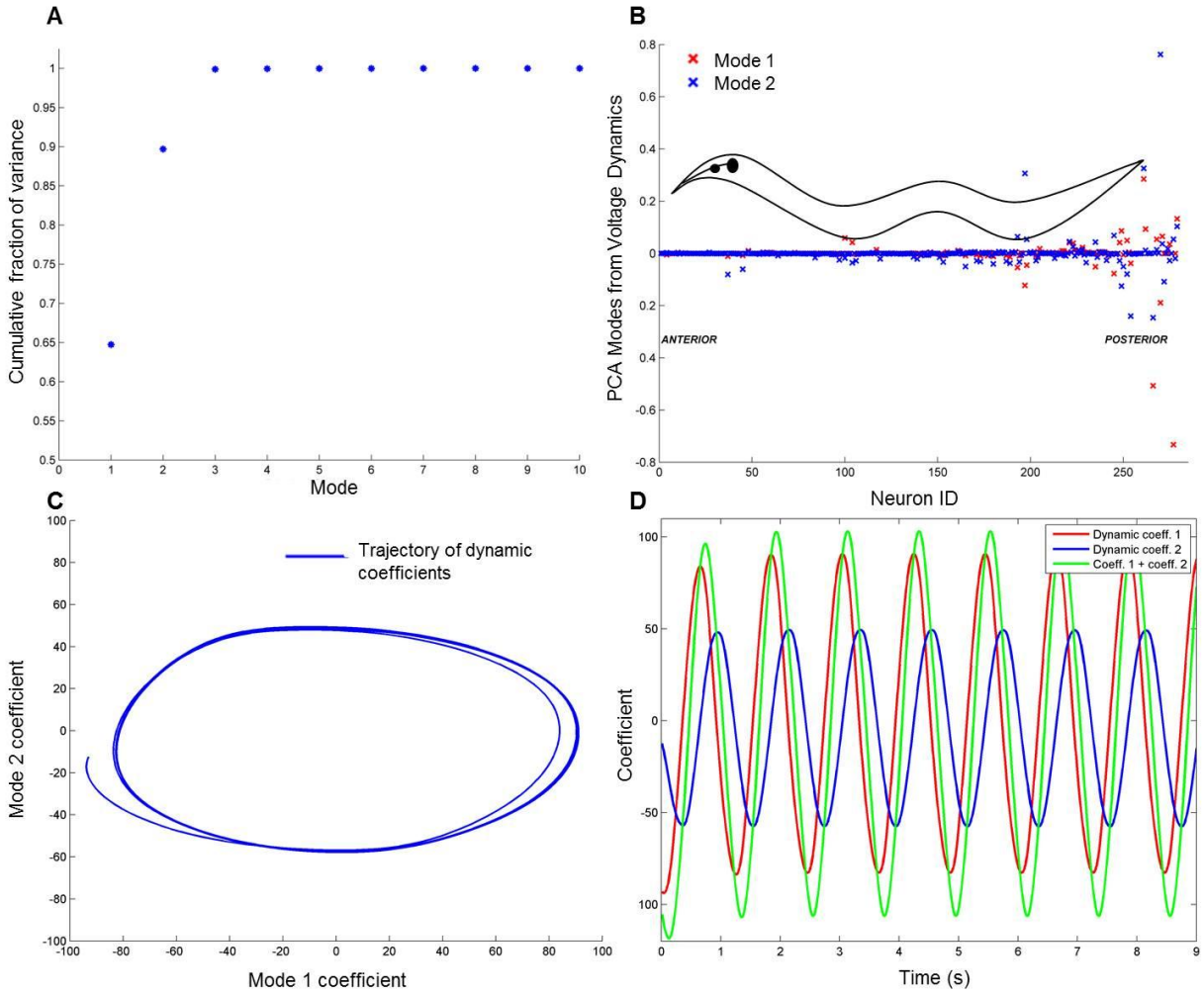

Figure 2. **A)** Cumulative fraction of variance explained by each mode. The system is dominated by the first two modes, which account for more than 90% of the variance. **B)** Pattern values in the first two modes for neurons along the body **C)** Two-dimensional trajectory of the low-dimensional oscillations during forward motion **D)** Oscillation of the time-dependent coefficients of the first two modes over time.

data from simulations of varying duration. We analyze simulation data generated by stimulating the PLM neurons with input of magnitude for which oscillatory dynamics exist (see [3]). In this discussion, we focus on analyzing ten-second simulations; however, these results are also representative of the findings using longer simulations. In all cases, the first second of simulation data was removed to ignore the effects of the initial forced perturbation.

We perform our singular value decomposition on simulation data from a 275-second simulation. Instead of using raw simulation data, we subtract the equilibrium value for each

neuron such that the data points at time *t* represent the difference between the activity at *t* and the equilibrium. For matrix factorization, we use singular value decomposition [13, 14] such that $D = \sum_{l=1}^{N} u_l \sigma_l v_l^T$, where u_l are the eigenvectors of DD^T, representing the pattern of each mode, v_l are the eigenvectors of D^TD, representing the time-dependent coefficients of each mode, and σ_l are the eigenvalues of both DD^T and D^TD, which are the singular values that act as stretching factors. We also consider the k-mode truncated decomposition of *D* is $D_k = \sum_{i=1}^{k} u_i \sigma_i v_i^T$ in which we take the dominant k modes. We find that only few modes (principal components) are significant for the decomposition: the first principal component explains approximately 65.24% of the energy and the first two components explain about 90.94% of the energy, where the energy explained by mode k is defined as $\sum_{i=1}^{k} \sigma_i^2 / \sum \sigma^2$ (Figure 2A). After the first eight components, the increase in the fraction of variance explained is not computationally measurable. Since the system is dominated by the first two principal components, we focus 2-mode truncated decomposition, and investigate the modes and time-dependent coefficients in this decomposition.

In reviewing the first two modes, it is evident that the neurons with stronger patterns values fall toward the posterior region of the body (Figure 2B), indicating that the posterior region contains neurons that have a more significant role in motivating *C. elegans'* forward movement. Our pattern plot also reveals that the ventral motor neurons (V*) have noticeably stronger pattern values than the dorsal motor neurons (D*), which implies that these neurons are responsible for a greater amount of the activity that creates the network oscillations.

To better identify the motor neurons responsible for the network behavior during forward motion, we look at the pattern values for motor neurons. We consider a neuron "active," or motivating forward motion, if its pattern value is above the median. Neurons with pattern values

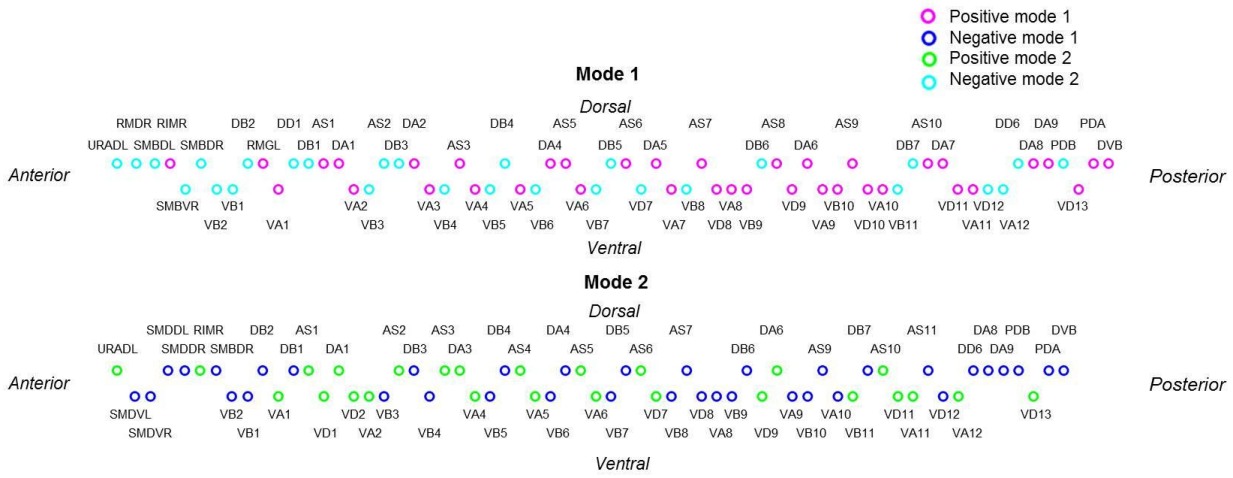

Figure 3. Visualization of the location and pattern values of active *C. elegans* motor neurons in the first two modes. Neurons are positioned according to their location on the AP-axis and the muscle group(s) they stimulate (dorsal or ventral). Colors indicate whether or not the neuron has a positive or negative pattern value in that particular mode.

below the median are not considered to be contributing substantially to the forward motion behavior, and are therefore labeled "inactive" (Figure 3). We identify 32 active ventral motor neurons in each of the first two modes, 21 active dorsal motor neurons in the first mode, and 18 active dorsal motor neurons in the second mode. In the first mode, we see 29 motor neurons that are positively activated (undulating upward), and 38 motor neurons that are negatively activated (undulating downward). In the second mode we find an opposite trend, with 37 motor neurons that are positively activated and 26 neurons that are negatively activated. Analysis of active neurons in each mode shows that there are a total of 74 active neurons, with 17 of them active only in one mode and 57 active in both modes. We will investigate the role of these motor neurons in generating forward motion as we explore the relationship between motor and sensory neuron activity in a forwardly-mobile *C. elegans*.

We also examine the first two time-dependent coefficients and see clear periodic oscillations in both. Summing these two coefficients, we see that oscillations with a period of about 1.2s (about two times the tail thrashing period found experimentally [15]), and note the

roughly circular trajectory of the coefficients in the principle component plane (Figure 4 C,D). The oscillations we see over time in the time dependent coefficients (projected voltages onto the modes) support the relationship between the activity of the neural network and the physical form of the nematode; when *C. elegans* is physically swimming forward with its body following a periodic wave pattern, and the activity of the neural network is also undergoing periodic oscillations.

III. VISCOELASTIC ROD MODEL

To gain further intuition into the relationship between neuronal activity and physical movement in *C. elegans*, we investigate how excitation of the neural network translates into muscle contraction. Modeling the physical state of *C. elegans'* muscles allows us to create a more accurate model of proprioception and environmental interaction, giving us the opportunity to estimate sensory input based on the approximate physical form of the nematode.

A. Physical Model for Anguilliform Swimmers

In order to realistically represent *C. elegans* movement, we apply a discrete viscoelastic rod model that has been used to describe motion in anguilliform swimmers [4]. This rod representation is chosen based on the anguilliform motion seen in *C. elegans* as it reacts to nose touch and similar stimuli [2,3]. In this model, the swimmer's body is represented as a rod composed of discrete rigid segments and joints [4], and the swimmer's muscle activity is represented by forces acting on the rod. The environment of the organism is modeled through damping, which may be varied to imitate different media.

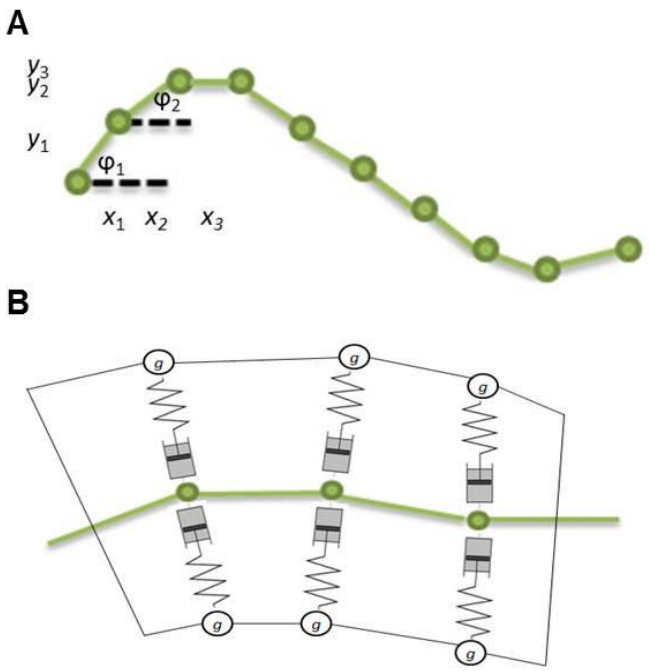

Figure 4. **A)** Representation of *C. elegans* body as a viscoelastic rod with discrete rigid segments. Segment position is governed by its length and the angle it forms with the horizontal plane **B)** The viscoelastic rod behavior is based on a spring and dashpot model with force g applied in the y-direction.

The state of the viscoelastic rod at any point in time is described by the x-y coordinates of the midpoint of each segment of the rod, and by the angle, φ, of each rod segment relative to the horizontal plane (Figure 4A) .

We consider the joints connecting the segments of the rod to be actuated by passive springs, dashpots, and time-dependent force generators (Figure 4B). Given this model, force applied near one end of the rod can lead to movement that travels through the other end of the rod, depending on the magnitude of initial force and parameters governing the rod and its environment [4].

The equations used to calculate the position of the segments of the viscoelastic rod are [4]

$$x_{i+1} = \frac{h}{2}\left(cos\,(\varphi_i) + cos\,(\varphi_{i+1})\right) + x_i, \tag{8}$$

$$y_{i+1} = \frac{h}{2}\left(sin\,(\varphi_i) + sin\,(\varphi_{i+1})\right) + y_i. \tag{9}$$

The differential equation determining the change in φ, based on the forces **f** and **g** applied to the segments of the rod in the x and y directions, respectively, is

$$J\ddot{\varphi} = M_i - M_{i-1} + \frac{h}{2}(g_i + g_{i-1})cos\,(\varphi) - \frac{h}{2}(f_i + f_{i-1})sin(\varphi). \tag{10}$$

We define the components of (10) as the contact moment M_i (11), the moment of inertia J_i for link i (12), and the moment of inertia I for motions in the x-y plane (13), as

$$M_i = EI_i\left(\frac{(\varphi_{i+1}-\varphi_i)}{h}-k_i\right)+\delta_i\left(\frac{(\dot{\varphi}_{i+1}-\dot{\varphi}_i)}{h}\right), \tag{11}$$

$$J_i = \rho h\left(I_i+\frac{\pi}{12}r^2h^2\right), \tag{12}$$

$$I=\frac{\pi D^4}{64}. \tag{13}$$

In Eq. (11), we solve for the contact moment based on the preferred curvature of the rod k_i, the rod's elasticity E (Young's modulus), the environmental damping coefficient δ, and the segment lengths h_i. The parameters of equations defining the moments of inertia J and I (12-13) depend on the rod's material density ρ, rod radius r, segment length h_i and rod diameter D.

Expanding equation (10) by substituting in equations (11-13) we solve for $\ddot{\varphi}_\iota$,

$$J_i\ddot{\varphi} = M_i - M_{i-1}+\frac{h}{2}(g_i+g_{i-1})cos\,(\varphi_i)-\frac{h}{2}(f_i+f_{i-1})sin(\varphi_i), \tag{14}$$

$$\rho h\left(I_i+\frac{\pi}{12}r^2h^2\right)\ddot{\varphi} = EI_i(\frac{(\varphi_{i+1}-\varphi_i)}{h}-k_i)+\delta_i(\frac{(\dot{\varphi}_{i+1}-\dot{\varphi}_i)}{h})-EI_{i-1}(\frac{(\varphi_i-\varphi_{i-1})}{h}-k_i)$$
$$-\delta_{i-1}\left(\frac{(\dot{\varphi}_i-\dot{\varphi}_{i-1})}{h}\right)+\frac{h}{2}(g_i+g_{i-1})cos(\varphi_i)-\frac{h}{2}(f_i+f_{i-1})sin(\varphi_i), \tag{15}$$

$$\ddot{\varphi}_\iota = \{EI_i(\frac{(\varphi_{i+1}-\varphi_i)}{h}-k_i)+\delta_i(\frac{(\dot{\varphi}_{i+1}-\dot{\varphi}_i)}{h})-EI_{i-1}(\frac{(\varphi_i-\varphi_{i-1})}{h}-k_i)-\delta_{i-1}(\frac{(\dot{\varphi}_i-\dot{\varphi}_{i-1})}{h})$$
$$+\tfrac{h}{2}(g_i+g_{i-1})cos(\varphi_i)\ -\tfrac{h}{2}(f_i+f_{i-1})sin(\varphi_i)\}\cdot\frac{1}{\rho h\left(I_i+\frac{\pi}{12}r^2h^2\right)}. \tag{16}$$

We then transform Eq. (16) into a system of first order differential equations

$$p_1=\varphi;\ p_2=\dot{\varphi};\ \dot{p}_1=\dot{\varphi}=p_2 \tag{17}$$

$$\dot{p}_{2i}=\ddot{\varphi}_\iota=\{EI_i\left(\frac{\left(p_{1(i+1)}-p_{1i}\right)}{h}-k_i\right)+\delta_i\left(\frac{\left(p_{2(i+1)}-p_{2i}\right)}{h}\right)$$

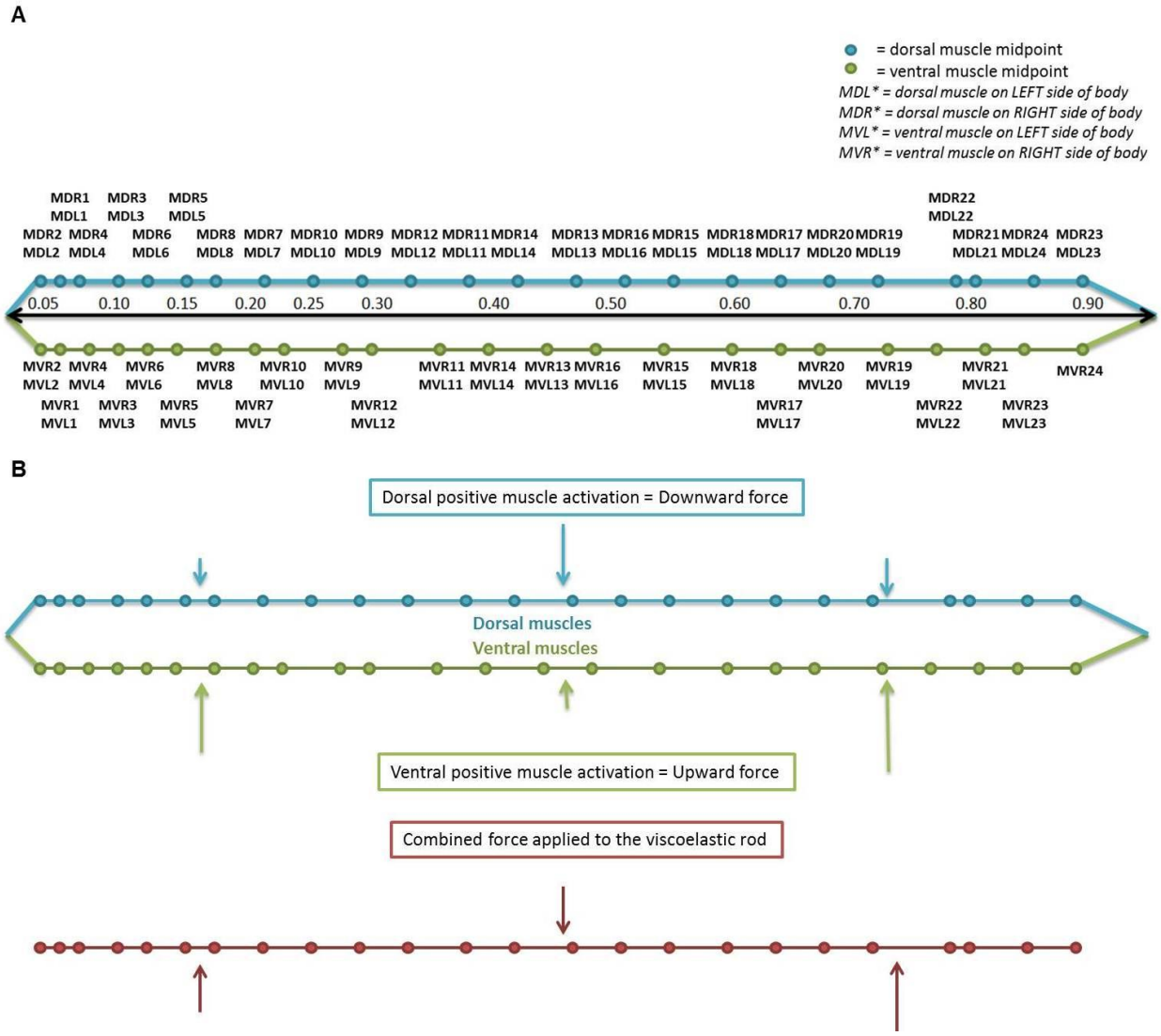

Figure 5. **A)** Map of the 95 muscles represented as segments of the viscoelastic rod. Segment length is determined by position of each muscle relative to the AP-axis, with the overall body length scaled to 1. **B)** Force is applied to segments of the viscoelastic rod to represent muscle activation. Dorsal and ventral muscles are combined based on their locations relative to the AP-axis, and summing the input into each group of muscles determines the overall force applied to a segment of the rod.

$$-EI_{i-1}\left(\frac{\left(p_{1i}-p_{1(i-1)}\right)}{h}-k_i\right)-\delta_{i-1}\left(\frac{\left(p_{2i}-p_{2(i-1)}\right)}{h}\right)+\frac{h}{2}(g_i+g_{i-1})\cos(p_{1i})$$

$$-\frac{h}{2}(f_i+f_{i-1})sin(p_{1i})\}\cdot\frac{1}{\rho h\left(I_i+\frac{\pi}{12}r^2h^2\right)} \tag{18}$$

that can be solved computationally.

B. Physical Model Applied to *C. elegans*

We apply the viscoelastic rod model to *C. elegans* by representing each muscle group in the nematode as a segment of the rod connected with joints. We approximate the forces applied to the rod segments using neuron activity data from forward motion simulations and a neuron-to-muscle map which are experimentally determined [10]. The combined activity of the neurons connected to a muscle is considered the muscle's input, which leads to activation and application of force.

In order to represent *C. elegans* most accurately, we model 95 muscles relevant to *C. elegans* forward motion and use their approximate sizes and locations to determine segment length and position. These 95 individual muscles are divided into four groups based on their physical location in the nematode: dorsal left, dorsal right, ventral left and ventral right. Every muscle is assigned a position along the AP- axis of the nematode, and the dorsal left and dorsal right muscles are coupled into single segments representing the dorsal muscle groups, and likewise for the ventral muscles. Dorsal (ventral) left muscles are paired only with dorsal (ventral) right muscles, and left-right pairs are determined from muscle location along the AP-axis, as in Figure 5A.

Differentiation between the dorsal and ventral muscle groups in this model is necessitated by the way these two muscle groups act in opposition to each other; in order for forward motion to occur, the dorsal muscles contract while the ventral muscles relax, and vice versa [2]. As such, we model activation of the muscles in the dorsal group through downward force, and the muscles in the ventral group through upward force. We then merge the dorsal and ventral muscle groups based on their location along the AP-axis to form a single discrete rod. We sum the dorsal (downward) and ventral (upward) forces over the dorsal-ventral muscle pairs to determine the

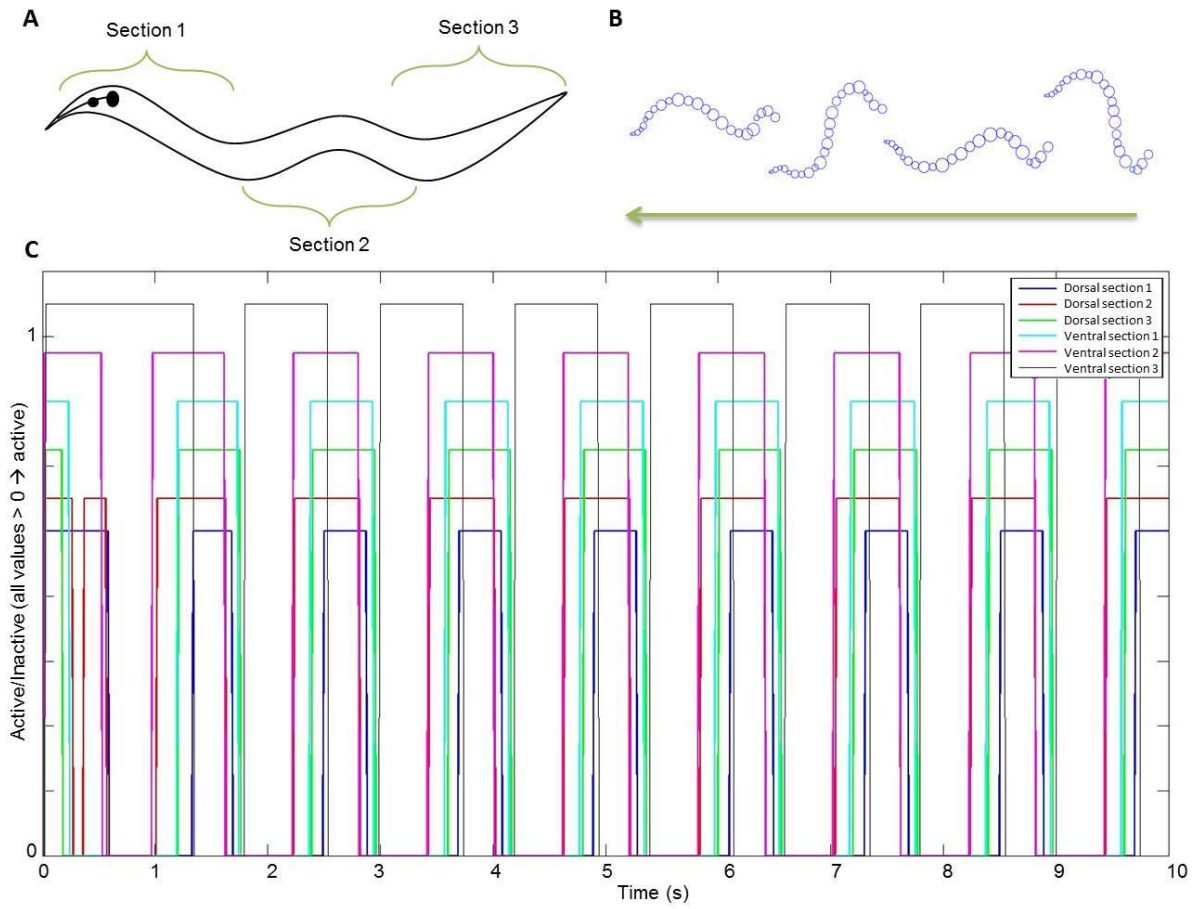

Figure 6. **A)** Three main contraction/relaxation sections of the worm. It is expected that dorsal and ventral muscles will not continuously be activated to contract simultaneously and that the stimulation of the three sections of the worm will be staggered **B)** Results of the simulations of viscoelastic rod movement over time during PLM stimulation. The worm is swimming from left to right, with the size of each rod segment represented by the diameter of each circle. **C)** Activation patterns of the groups of dorsal/ventral muscles based on input from connected motor neurons during PLM excitation showing the repeating activation pattern of the six muscle groups

overall force applied to each segment of the rod: for example, if a large downward force is applied by a dorsal muscle and a small upward force is applied by the corresponding ventral muscle, the net result will be application of downward force on the segment of the rod representing the merged muscles (Figure 5B).

Unlike the generic anguilliform swimmer model, the basic *C. elegans* viscoelastic rod representation assumes force is only applied in the y-direction, meaning **f** is a zero vector. Given

the absence of force in the x-direction, we eliminate the sine term from (18), resulting in the final system of equations

$$\dot{p}_{1i} = \dot{\varphi}_i = p_{2i} \tag{19}$$

$$\dot{p}_{2i} = \ddot{\varphi}_i = \{EI_i\left(\frac{\left(p_{1(i+1)} - p_{1i}\right)}{h} - k_i\right) + \delta_i(\frac{\left(p_{2(i+1)} - p_{2i}\right)}{h}) - EI_{i-1}\left(\frac{\left(p_{1i} - p_{1(i-1)}\right)}{h} - k_i\right)$$

$$- \delta_{i-1}(\frac{\left(p_{2i} - p_{2(i-1)}\right)}{h}) + \frac{h}{2}(g_i + g_{i-1})cos\,(p_{1i})\,\} \cdot \frac{1}{\rho h\left(I_i + \frac{\pi}{12}r^2h^2\right)} \tag{20}$$

for modeling *C. elegans* as a viscoelastic rod.

Applying this model with parameters specific to *C. elegans* [5,16] (see Appendix), we simulate the movement of *C. elegans* during constant PLM excitation of $2x10^4$ mV using the activity of the motor neurons responsible for forward motion (VD/VB/DD/DB) as input. The behavior of our physical model is validated by the known contraction and relaxation patterns found in the nematode's body during forward motion [3]. These patterns closely align with the simulated formations of the viscoelastic rod as it moves over time (Figure 6B).

The behavior of the viscoelastic rod as it models PLM excitation shows a clear relationship between stimulation of motor neurons along the body and muscle movement. Since our model is sensitive to the magnitude of motor neuron stimulation a muscle receives, we observe that greater force is applied in certain sections of the worm over time, creating the undulating form associated with *C. elegans* forward motion. Specifically, we often see the anterior and posterior sections receiving force in the opposite direction of the force applied to the middle section of the worm (Figure 6 A,B). This configuration represents dorsal (ventral) contraction in the anterior and posterior sections with ventral (dorsal) contraction in the middle section, which is precisely the expected muscle usage during forward motion [2].

IV. ADAPTIVE CONTROL ALGORITHMS

Leveraging our models of *C. elegans* neural activity and physical movement, we seek to identify feedback loops in the network that will output the prolonged oscillating signal associated with forward motion. In order to determine how sensory neurons can be stimulated to generate the desired activity in motor neurons, we develop two different adaptive control methods based on generic adaptive control methods such as gradient descent (Delta Rule) and recursive least squares (FORCE). Each algorithm uses training and operational phases. In the training phase, we reduce error between the desired and actual outputs by making small, rapid modifications to the weights, which are applied to the network input. When the system produces the desired output without significant weight modification between iterations, a correct set of weights has been found [6]. During the operational phase, weights are held at the static values found through training, and the resulting simulated dynamics demonstrate whether or not the weights converged upon function without modification.

A. Delta Rule - A Gradient Descent Algorithm

First method of supervised learning that we have implemented in this investigation was the gradient descent (or the Delta Rule [6, 14]). Traditionally, this method trains a system to produce a desired response by applying weights to the system output, with weights iteratively adjusted to minimize the difference between the desired system output and the estimated output [17]. The components of the gradient descent method for a generic neural network of N neurons are $\boldsymbol{w}(t)$, the column vector of weights w_n for $n = 1,\ldots,N$ at time t, and the column vector $\boldsymbol{r}(t)$ defining the activity of each neuron $r_n(t)$ at time t. We define scalar estimated network output as

$$z(t) = \boldsymbol{w}^{\boldsymbol{T}}(t)\boldsymbol{r}(t). \quad (21)$$

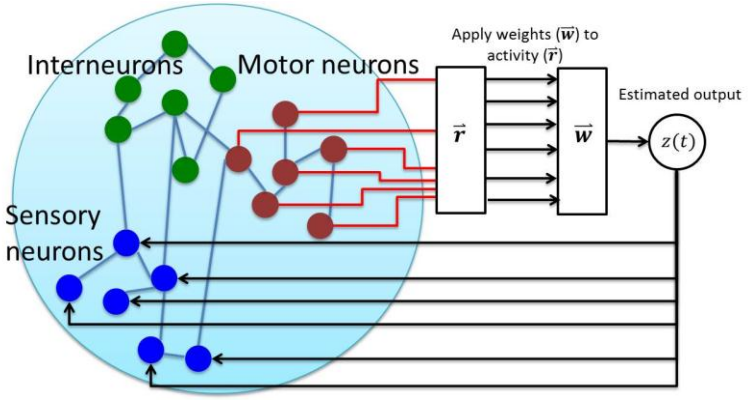

Figure 7. Basic RLS (FORCE) system structure in which dynamically updated weights are applied to system activity to generate estimated system output. This output is compared with the expected system output and weights are updated accordingly.

The scalar desired network output is denoted *f(t)*, define our learning rate as α, and calculate scalar estimation error as

$$e(t) = f(t) - z(t). \tag{22}$$

In the method of gradient descent, initial weight vector $\boldsymbol{w}(t_0)$ is randomly generated and learning rate α is generally in the interval [1,*N*] [17].

The goal of this method is to adjust the values of $\boldsymbol{w}$ during training such that after training is complete, applying the final weights, $\boldsymbol{w}_{\mathbf{final}}$ to the system output will minimize network error. In order to find the set of weights that will accomplish this, the weight-based cost function

$$E(\boldsymbol{w}(t)) = \frac{1}{2}\big(f(t) - \boldsymbol{w}(t)^T * \boldsymbol{r}(t)\big)^2 = \frac{1}{2}\big(f(t) - z(t)\big)^2 \tag{23}$$

is defined. In equation (23), we consider the network error to be the difference between the desired output, *f(t)*, and the estimated output, *z(t)*. We square the error to scale the cost for larger errors, and ensure that the cost function always yields non-negative values. We multiply the squared error by ½ in order to simplify the calculation of the gradient (see Eq. 24).

In order to determine how to adjust $\boldsymbol{w}$ to minimize cost, and therefore diminish network error, we calculate the gradient,

$$\nabla E = \frac{\delta E}{\delta z} * \frac{\delta z}{\delta \boldsymbol{w}} = -\big(f(t) - z(t)\big) * \boldsymbol{r}(t) = -e(t) * \boldsymbol{r}(t), \tag{24}$$

of $E(\boldsymbol{w})$. We then define our weight update function,

$$\boldsymbol{w}(t+\Delta t) = \boldsymbol{w}(t) + \alpha\Delta\boldsymbol{w}, \tag{25}$$

where $\Delta\boldsymbol{w} = -\frac{\delta E}{\delta w_n} = e(t) * \boldsymbol{r}(t)$ (26)

and $\boldsymbol{w}(t+\Delta t) = \boldsymbol{w}(t) + \alpha\big(e(t) * \boldsymbol{r}(t)\big)$, (27)

based on the gradient of the cost function and the learning rate (α). As can be seen in Eq. (27), smaller α values will keep $\boldsymbol{w}$(t+Δt) closer to $\boldsymbol{w}$(t), emphasizing the contribution of the previous weight. In contrast, larger α values will increase the impact of the gradient term on the updated weight value.

B. Recursive Least Squares Algorithm - Background

To enhance our ability to accurately adjust feedback weights to create the desired system dynamics, we leverage our RLS-based adaptive control algorithm. The advantage of this method is in its use of the estimated inverse correlation matrix of network activity to update weights instead of the computed derivative used in the method of gradient descent. The inverse correlation matrix models the neuron activity correlations over the entire duration of the training phase, with more emphasis given to recent activity.

The recursive least squares (RLS) algorithm proposed in [17] and developed by Sussillo and Abbott in [6] offers a training procedure specifically developed for recurrent neural networks and was called as FORCE – First Order Reduced and Controlled Error. The principal difference between the RLS-based algorithm and the method of gradient descent is that the RLS method is designed for recurrent networks and operates with network feedback, whereas the basic gradient descent algorithm simply applies weights on the network output to achieve the desired system behavior (Figure 7). Given a recurrent neural network and a function representing the desired network output, the RLS algorithm seeks a set of weights that can be applied to the network

feedback in order to create the desired output. Once RLS has found a set of correct weights (assuming they exist), they are applied to the network feedback without further modification, and the network will generate the desired output. In [6] the approach was developed and applied to training *output* or *internal* weights of recurrent networks. It was shown that the algorithm is most successful when it is applied to random networks that exhibit chaotic dynamics.
The consideration of the feedback loop as an entity separate from overall network output allows us to more accurately model the *C. elegans* sensory-motor feedback.

In order to find a functional set of weights to apply to the network feedback, the RLS algorithm employs a training phase in which small modifications are made to the weights. The goal of this phase is not only to reduce the error between the desired and estimated outputs, but also to reduce the amount of modification required to weights on each update. The minimization of weight change is a significant requirement, as the updates to the weights must be sufficiently small for the algorithm to converge; if large weight updates are required to keep the output error small, then the network will not produce the desired output once the training period ends and the weights are static.

The RLS algorithm has several factors that drive convergence of the weights to values that will produce the desired network behavior. First, the RLS algorithm does not clamp the feedback exactly to the *estimated* network output, as this could create longer term deviations from the expected (desired) output function. Instead, it implements small and rapid changes to the feedback weights based on output error to minimize negative delayed effects. Additionally, the algorithm holds feedback close to desired output function, but not fixed at the desired output function. This allows the network to stabilize during training, and is beneficial to the network

post-learning phase when the feedback may not be exactly as expected and weights are no longer updated by the training algorithm.

The RLS training method supports three network models, each with a different feedback method. The most basic version is when the RLS method is applied to a network with one readout unit that is used as a basis to provide feedback to the generator network, as seen in the model to the left.

The RLS training algorithm for a generic recurrent network of *N* neurons uses vectors $\boldsymbol{w}(t)$, $\boldsymbol{r}(t)$, scalar functions $z(t) = \boldsymbol{w}^T(t)\boldsymbol{r}(t)$ and $f(t)$, and learning rate α as defined in the method of gradient descent. Unlike the method of gradient descent, RLS employs two different error calculations. To represent the estimated error prior to the weight update at time *t*, we define

$$e_-(t) = \boldsymbol{w}^T(t-\Delta t)\boldsymbol{r}(t) - f(t). \tag{28}$$

To represent the estimated error after the weight update at time *t*, we define

$$e_+(t) = \boldsymbol{w}^T(t)\boldsymbol{r}(t) - f(t). \tag{29}$$

As in the method of gradient descent, we initialize $\boldsymbol{w}(t_0)$ randomly and expect α to be a positive whole number with $\alpha << N$ [6, 14].

During training, updates to the weights,

$$\boldsymbol{w}(t) = \boldsymbol{w}(t-\Delta t) - e_-(t)P(t)\boldsymbol{r}(t), \tag{30}$$

are calculated based on Recursive Least Squares results (see Appendix section *Derivation of RLS Equation and P Matrix* for details). Here, $P(t)$ is an NxN matrix,

$$P(t) = P(t-\Delta t) - \frac{P(t-\Delta t)\boldsymbol{r}(t)\boldsymbol{r}(t)^T P(t-\Delta t)}{1+\boldsymbol{r}(t)^T P(t-\Delta t)\boldsymbol{r}(t)}, P(0) = \frac{I}{\alpha}. \tag{31}$$

In Eqs. (30, 31), *P* is an approximation of the inverse of the correlation matrix of $\boldsymbol{r}$. In addition, the regularization term is used to ensure smoothness in the updates to the weights with network learning rate α. This inverse correlation matrix serves as a memory to emphasize activity in the

recent past, and give less weight to activity in the distant past. Smaller values of α lead to faster learning as the emphasis on the inverse correlation matrix is larger, but can simultaneously make the algorithm unstable. Increasing α can remedy the instability, however in this case the relationship $\alpha \ll N$should be maintained, as larger α allows the output to quickly diverge from the desired output, causing instability in the P and training failure [17].

Using the RLS algorithm, training ceases when $e_+(t)$ and $e_-(t)$ are approximately equal, or when $\frac{e_-}{e_+} \cong 1$, and the change in weights has converged to near zero value. We can see the relationship between these two requirements by rewriting the error in terms of *P*,

$$e_+(t) = e_-(t)\left(1 - \boldsymbol{r}^T(t)P(t)\boldsymbol{r}(t)\right). \quad (32)$$

Thus, we can see that as $\boldsymbol{r}^T(t)P(t)\boldsymbol{r}(t)$ asymptotically approaches zero, $e_+(t) \approx e_-(t)$ such that both convergence requirements are satisfied [6].

C. Implementation of Gradient Descent Algorithm for *C. elegans*

In the *C. elegans* neural network, the method of gradient descent is utilized to determine inputs to sensory neurons that instigate and maintain the network oscillation associated with forward motion. The ultimate goal of the gradient descent implementation is to find a set of weights that could provide functional sensory neuron inputs based on motor neuron output, however the model excluding network feedback is also considered as a basic case.

Since the method of gradient descent is not designed for recurrent networks, some modifications of the generic algorithm are required to best model the *C. elegans* sensory-motor feedback loop. In the *C. elegans* implementation of the gradient descent algorithm, network output is considered principally from motor neurons, and network input is allowed only to sensory neurons. The movement is a feedback loop since the input into sensory neurons is calculated based on the motor neuron output, the motor neuron output is defined by the motor

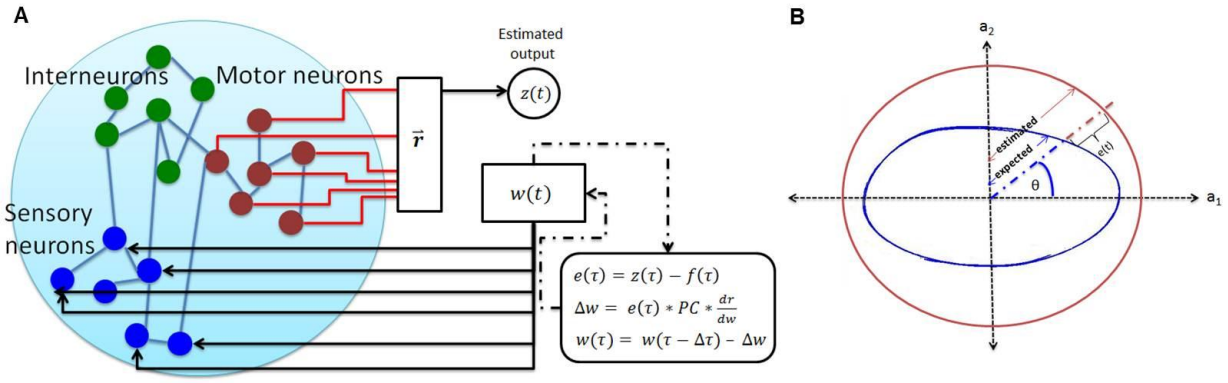

Figure 8. **A)** Diagram of gradient descent algorithm for the simple non-recurrent scenario. Weights are used as direct input into sensory neurons and are updated based on error and estimated derivative of *r(t)* with respect to *w(t)*. The use of the estimated output is solely for error calculation, as there is no feedback in this scenario. **B)** To determine output error, we compare the trajectory of the network activity during training with the expected network trajectory during forward motion. The error is calculated as the difference between the expected and estimated radius at angle θ.

neurons' activity, and motor neuron activity is affected by the input to sensory neurons (Figure 8A).

As the objective of this implementation is to determine functional inputs that perpetuate *C. elegans* forward motion, the traditional gradient descent algorithm is altered to apply and adjust weights on the *network inputs* instead of modifying weights on the network output. Thus, our algorithm weights the network input, which was formulated from motor neuron output. The method of gradient descent is implemented for *C. elegans* with vector $\boldsymbol{w(t)}$ representing sensory neuron input weights, which are initialized such that $\boldsymbol{w(t_0)} = 0$. We define our desired network output function as $f(t) = \sigma_1 a_1 + \sigma_2 a_2$,where σ_n is the singular value of the n^{th} PCA mode and a_n is the vector of time-dependent coefficients of the n^{th} PCA mode. We denote the n^{th} principal component vector as $\boldsymbol{PC_n}$. As in the generic model, we consider neuronal activity $\boldsymbol{r}(t)$, but base our implementation on $\boldsymbol{r_{eq}(t)} = \boldsymbol{r(t)} - \boldsymbol{v}_{equilibrium}$, the neuronal activity relative to equilibrium. Our overall estimated network output is

$$z\,(t) = \sqrt{(\boldsymbol{PC_1}^T \boldsymbol{r_{eq}}(t))^2 + \ (\boldsymbol{PC_2}^T \boldsymbol{r_{eq}}(t))^2} \tag{33}$$

and our motor neuron output is calculated as

$$z_{feedback}(t) = \sqrt{(\boldsymbol{PC_1}^T (\boldsymbol{m}.* \boldsymbol{r_{eq}}(t)))^2 + \ (\boldsymbol{PC_2}^T (\boldsymbol{m}.* \boldsymbol{r_{eq}}(t)))^2}\,, \tag{34}$$

where $\boldsymbol{m}$ is a mask vector that selects only motor neurons. Lastly, we define our estimation error and learning rate as in the generic method, such that $e(t) = f(t) - z(t)$ (or $\tilde{e}(\gamma) = \tilde{f}(\gamma) - \tilde{z}(\gamma)$) and $\alpha = 1$, and we consider $\boldsymbol{s}_{input}(t)$ to be vector of input into sensory neurons.

In practice, we project our desired output $f(t)$ and estimated output functions $z(t)$ into the reduced principal component plane, such that $\tilde{f}$ and $\tilde{z}$ are functions of phase angle θ and yield the expected and estimated radii of the output trajectory of θ, respectively. We determine θ at time t by projecting the current network activity $\boldsymbol{r_{eq}}(t)$ onto the *PC* plane such that

$$a_1 = \boldsymbol{PC_1} * \boldsymbol{r_{eq}}(t), a_2 = \boldsymbol{PC_2} * \boldsymbol{r_{eq}}(t), z(v) = a_1 + ia_2, \text{and } \theta = Arg\,(z). \tag{35}$$

We perform this conversion in consideration of the fact we are training the system to produce the oscillations expected from forward motion, and these oscillations may begin at any point in time. Thus, instead of calculating error at time t based on the value of our desired output at time t, we model the network behavior in polar coordinates, which eliminates direct time dependency (Figure 8B).

Given our conversion of f and z to polar coordinates, we also adjust our weight update metric from a time-based interval to a Poincaré map. With this implementation, instead of updating weights every s seconds, we update weights each time the estimated output trajectory crosses a particular plane (e.g., the positive imaginary axis). Since there are scenarios, particularly early in training, in which the output trajectory would not cross the positive

imaginary axis, we also apply a time-based limit that forces the algorithm to update weights if necessary.

In order to define our weight updating function, we redefine our cost function, as in Eq. (25). Since we have modified the gradient descent method to apply weights $\boldsymbol{w}$ to the network input instead of output, we must reconsider our cost function and the calculation of its gradient from Eq. (30). We maintain the same overall form of our cost function, but take into consideration the fact that the definition of *z(t)* is now defined

$$z(t) = \sqrt{(\boldsymbol{PC_1}^T \boldsymbol{r_{eq}}(t))^2 + \ (\boldsymbol{PC_2}^T \boldsymbol{r_{eq}}(t))^2}, \tag{36}$$

$$\text{such that } E\big(w(t)\big) = \frac{1}{2}\left(f(t) - \sqrt{(\boldsymbol{PC_1}^T \boldsymbol{r_{eq}}(t))^2 + \ (\boldsymbol{PC_2}^T \boldsymbol{r_{eq}}(t))^2}\ \right)^2 = \frac{1}{2}\big(f(t) - z(t)\big)^2. \tag{37}$$

While the equation of the cost function (37) does not directly depend on the input weights, $\boldsymbol{w}(t)$, the network activity $\boldsymbol{r}(t)$ does depend on the input weights through simulation. Thus, we will define the gradient of the cost function as

$$\frac{\delta E}{\delta \boldsymbol{w}} = \frac{\delta E}{\delta z} * \frac{\delta z}{\delta \boldsymbol{r_{eq}}} * \frac{\delta \boldsymbol{r_{eq}}}{\delta \boldsymbol{w}} = -\big(f(t) - z(t)\big) .* \sqrt{\big(\boldsymbol{PC_1}^2 + \boldsymbol{PC_2}^2\big)}^T * \frac{\delta \boldsymbol{r_{eq}}}{\delta \boldsymbol{w}}. \tag{38}$$

Since analytical calculation of the derivative of the network activity with respect to the input weights cannot feasibly be used in the calculation of the cost function's gradient, we will computationally estimate $\frac{\delta \boldsymbol{r}}{\delta \boldsymbol{w}}$ as

$$\frac{\delta \boldsymbol{r_{eq}}}{\delta \boldsymbol{w}} = \frac{\delta \boldsymbol{r_{eq}}}{\delta t} * \frac{\delta t}{\delta \boldsymbol{w}} \approx \frac{\delta \boldsymbol{r_{eq}}}{\delta t} * \frac{\Delta t}{\boldsymbol{w}(t) - \boldsymbol{w}(t - \Delta t)}. \tag{39}$$

Substituting Eq. (39) into Eq. (38) yields the expanded form of the gradient of the cost function $E(\boldsymbol{w}(t))$,

$$\nabla E = -\big(f(t) - z(t)\big) * \sqrt{\big(\boldsymbol{PC_1}^2 + \boldsymbol{PC_2}^2\big)}^T * \frac{\boldsymbol{r_{eq}}\big(\boldsymbol{w}(t)\big) - \boldsymbol{r_{eq}}\big(\boldsymbol{w}(t - \Delta t)\big)}{\boldsymbol{w}(t) - \boldsymbol{w}(t - \Delta t)}. \tag{40}$$

We then update the input weights based on the gradient of the cost function and the learning rate (α),

$$\boldsymbol{w}(t+\Delta t) = \boldsymbol{w}(t) + \alpha\Delta\boldsymbol{w}, \tag{41}$$

$$\text{where } \Delta\boldsymbol{w} = -\nabla E = e(t) * \sqrt{\left(\boldsymbol{PC_1}^2 + \boldsymbol{PC_2}^2\right)}^T * \frac{\delta \boldsymbol{r_{eq}}}{\delta \boldsymbol{w}} \tag{42}$$

$$\text{and } \boldsymbol{w}(t+\Delta t) = \boldsymbol{w}(t) + \alpha\left(e(t) * \sqrt{\left(\boldsymbol{PC_1}^2 + \boldsymbol{PC_2}^2\right)}^T * \frac{\delta \boldsymbol{r_{eq}}}{\delta \boldsymbol{w}}\right). \tag{43}$$

We first use our weights to create a model of the network without using any explicit feedback between motor and sensory neurons. In this scenario, we simply use our weight vector $\boldsymbol{w}$ directly as sensory input, s

$$\boldsymbol{s_{input}}(t+\Delta t) = (\boldsymbol{w}(t+\Delta t).* \boldsymbol{s_{mask}}). \tag{44}$$

To add recurrence to this model by applying feedback directly from motor neurons to sensory neurons, we calculate $z_{feedback}(t)$ and apply this motor neuron feedback to the sensory neuron using a remapping matrix *A*. In this method, we use to approximate feedback into sensory neurons with *A* as a static map of the physical proximity of motor and sensory neurons. Through this map, active motor neurons provide greater input to nearby sensory neurons than to distant sensory neurons. This feedback model is based on the assumption that motor neurons activate proximal muscles, and nearby sensory neurons are subsequently stimulated by the activation and motion.

Using our static weight matrix *A*, we define $z_{feedback}(t) = A*z(t)$, yielding sensory neurons input

$$\boldsymbol{s_{input}}(t+\Delta t) = (\boldsymbol{w}(t+\Delta t).* \boldsymbol{s_{mask}}) * z_{feedback}(t). \tag{45}$$

Extending the basic proximity-based transformation, we implement a model that includes connectivity between motor neurons and muscles, and an approximation of *C. elegans* body movement based on calculated muscle activation. With these enhancements, we calculate the

input into a sensory neuron based on the state of the body in its immediate area. This implementation models proprioception and environmental interaction, as sensory neurons are activated based on the *C. elegans* body formation, which changes through movement over time.

The enhanced proprioceptive model is formed based on muscle-to-neuron connectivity map M and the viscoelastic rod model of *C. elegans* movement. We define the components of this system as muscle activity $a(t) = \boldsymbol{r_m}(t) * M$, viscoelastic rod-based physical model $v(a)$, and input weights $\boldsymbol{w}$. After calculating $v(a)$, we formulate our sensory input as

$$\boldsymbol{s_{input}}(t + \Delta t) = v(a) * \boldsymbol{w}(t). \quad (46)$$

D. Implementation of RLS Algorithm for *C. elegans*

In order to represent the specific feedback loop between motor and sensory neurons, some modifications to the original RLS-based FORCE algorithm were required. Unlike the generic algorithm, our RLS-based proprioceptive model considers the relationships between motor neuron output, muscle movement and sensory neuron input [18]. While the weight update algorithm remains unchanged, we calculate sensory neuron input by applying a transformation to the motor neuron output and weighting sensory neuron input instead of feeding the weighted output directly back into the network. As in the method of gradient descent, we calculate the estimated output using the motor neuron activity projected into the principal component space, and compare it with the expected trajectory during forward motion (Figure 8B). This methodology provides flexibility for training; since the expected output is not based on a particular point in simulation time, error can be accurately calculated regardless of when oscillations begin.

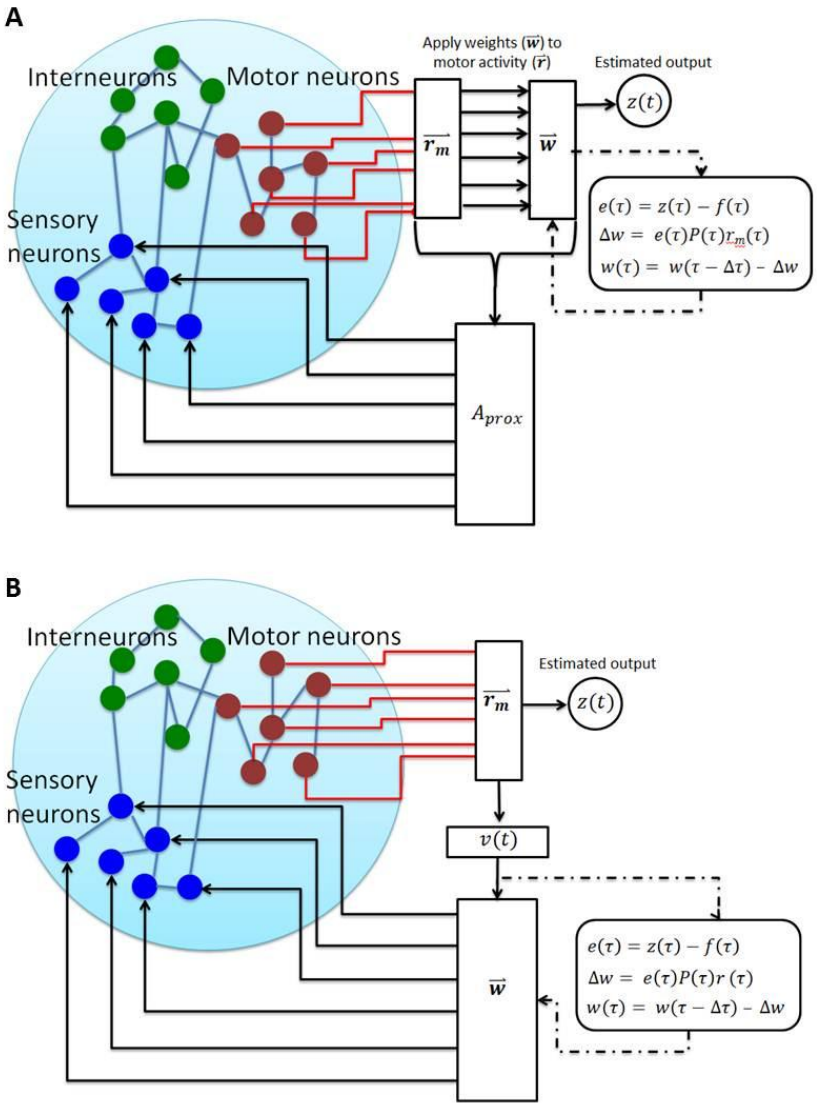

Figure 9. **A)** Adaptive control structure with weights applied to motor neuron output. Input is mapped to sensory neurons using a map based on the proximity of sensory and motor neurons **B)** Adaptive control structure with weights applied to sensory neuron input. Output is estimated based on motor neuron activity, which is then transformed through the viscoelastic rod model. The state of the viscoelastic rod is then weighted and used as input to sensory neurons.

We define the motor neuron network activity as $\boldsymbol{r_m}(t) = \boldsymbol{r_{eq}}(t).*\boldsymbol{m}$ and network output, exclusively based on motor neuron activity, as $z(t) = \boldsymbol{w}^T(t)*\boldsymbol{r_m}(t)$. The pre- and post-weight update error functions (Eqs. 28, 29) remain unchanged from the generic algorithm, except that they are based solely on motor neuron output.

As in the gradient descent algorithm, our first means of approximating sensory neuron input through a weighted map based on the physical proximity of motor and sensory neurons (matrix A). We apply this map to the weighted output of motor neurons such that the input into the sensory neurons is calculated as $\boldsymbol{s_{input}} = A*(\boldsymbol{r_m}(t).*\boldsymbol{w}(t))$ (Figure 9A).

We then progress to enhanced proprioceptive model, which includes the viscoelastic rod simulation of movement and body state (Figure 9B). In this model, we modify our algorithm to weight sensory neuron input, with weights modified based on the trajectory error and the inverse correlation matrix as before. As in the previously described methods, we update weights in order to minimize the cost function, which is defined as

$$E\big(w(t)\big) = \frac{1}{2}\Big(f(t) - \sqrt{(\boldsymbol{w}^T \boldsymbol{r_m}(t))^2}\ \Big)^2 = \frac{1}{2}\big(f(t) - z(t)\big)^2. \quad (47)$$

With our modifications to weight sensory input and use our motor activity-based viscoelastic rod model, $v(a)$, we define the weight update and sensory input functions as

$$w(t) = w(t - \Delta t) - e_m(t)P(t)r(t) \quad (48)$$

and $$\boldsymbol{s}_{input} = v(a) * w(t). \quad (49)$$

RESULTS

To investigate the performance of the algorithms we experiment with various training scheme to subsets of sensory and motor neurons. For network output we include the motor neurons known to be involved in forward motion and limit sensory neurons which receive feedback to a small set of neurons, known as tap responsive (PLM neurons), and then expand to consider additional neurons identified as integral to *C. elegans* forward crawling [15].

We first investigate whether or not any control algorithm is needed to elicit forward motion in *C. elegans*. Thus, we consider basic scenarios with constant random or uniform PLM input on the scale of the known solution ($2x10^4$). We also experiment with random or uniform weights on viscoelastic rod feedback to PLM neurons, justifying the need for a training algorithm. Next, we

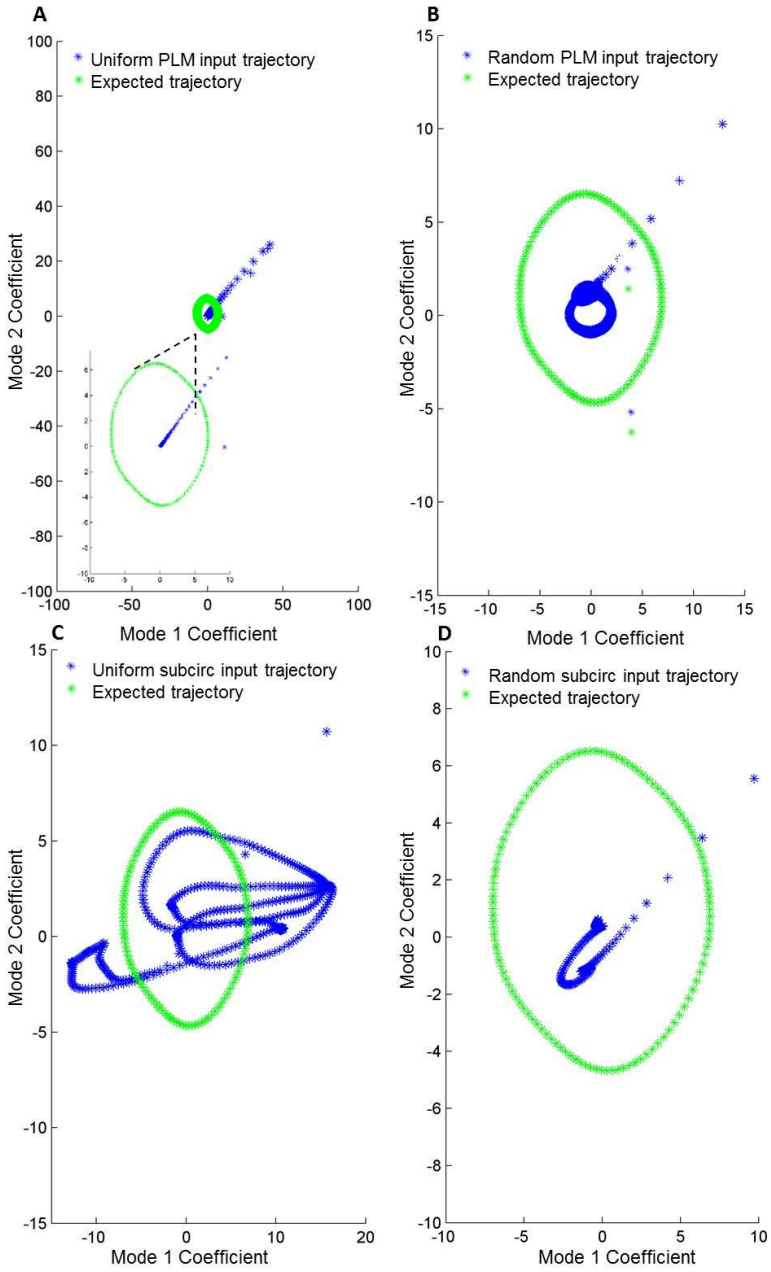

Figure 10. Examples of network dynamics in uniform and random input as compared to the expected forward motion dynamics. In PLM scenarios, input was required to be of the same magnitude as the known PLM solution. **A)** Simulation of network behavior with arbitrary constant uniform input into PLM neurons **B)** Simulation of network behavior with constant random input into PLM neurons **C)** Simulation of network behavior with arbitrary constant uniform input into the sub-circuit of seven sensory neurons associated with forward motion **D)** Simulation of network behavior with constant random input into the sub-circuit of seven sensory neurons

explore random or uniform input and weights on the seven sensory neurons identified as relevant to *C. elegans* crawling [15].

We apply adaptive control algorithms to two different models of the *C. elegans* neural network, one with feedback, and one without direct feedback. In the scenario without feedback, we first confirm the algorithm's ability to find known PLM input solution. We then use the algorithm search for a set of inputs into a sub-circuit of seven sensory neurons that generates

forward motion dynamics similar to the known PLM excitation. In the model with feedback, sensory neuron input is determined by feedback from motor activity, and we use the algorithms to find input weights such that forward motion dynamics are perpetuated after PLM excitation ceases.

A. Uniform/Random Weights

In order to determine whether or not forward motion could be instigated through arbitrary uniform input into tail tap response PLM neurons, we randomly choose positive input on the scale of $2x10^4$ and constantly provide this stimulation over several seconds. We determine that the forward motion solution we seek cannot be generated with generic uniform input, even if it is of the same magnitude as the known PLM excitation (Figure 10A). Likewise, we find that random (non-uniform) input into PLM neurons does not generate the neuronal patterns associated with forward movement (Figure 10B). We perform the same experiment providing input into the sub-circuit of seven sensory neurons identified as relevant to forward motion, with results failing to indicate any pattern associated with crawling (Figure 10 C,D).

Next, we investigate the neuronal behavior when feedback is used to determine sensory neuron input. We start with considering the basic cases of uniformly weighted feedback and randomly weighted feedback. We find that when uniformly weighted feedback into the PLM neurons follows 4s of constant PLM stimulation at $2x10^4$ mV, the feedback does not maintain the oscillations associated with forward motion. Although initial feedback evokes a response (see outliers in Figure 11A), the system slowly settles back toward equilibrium. Similarly, randomly weighted PLM feedback is unable to perpetuate the oscillations associated with forward motion (Figure 11B). In testing feedback on a larger set of seven sensory neurons, we find that the error is significantly larger than in the PLM feedback case (Figure 11 C,D). While providing feedback

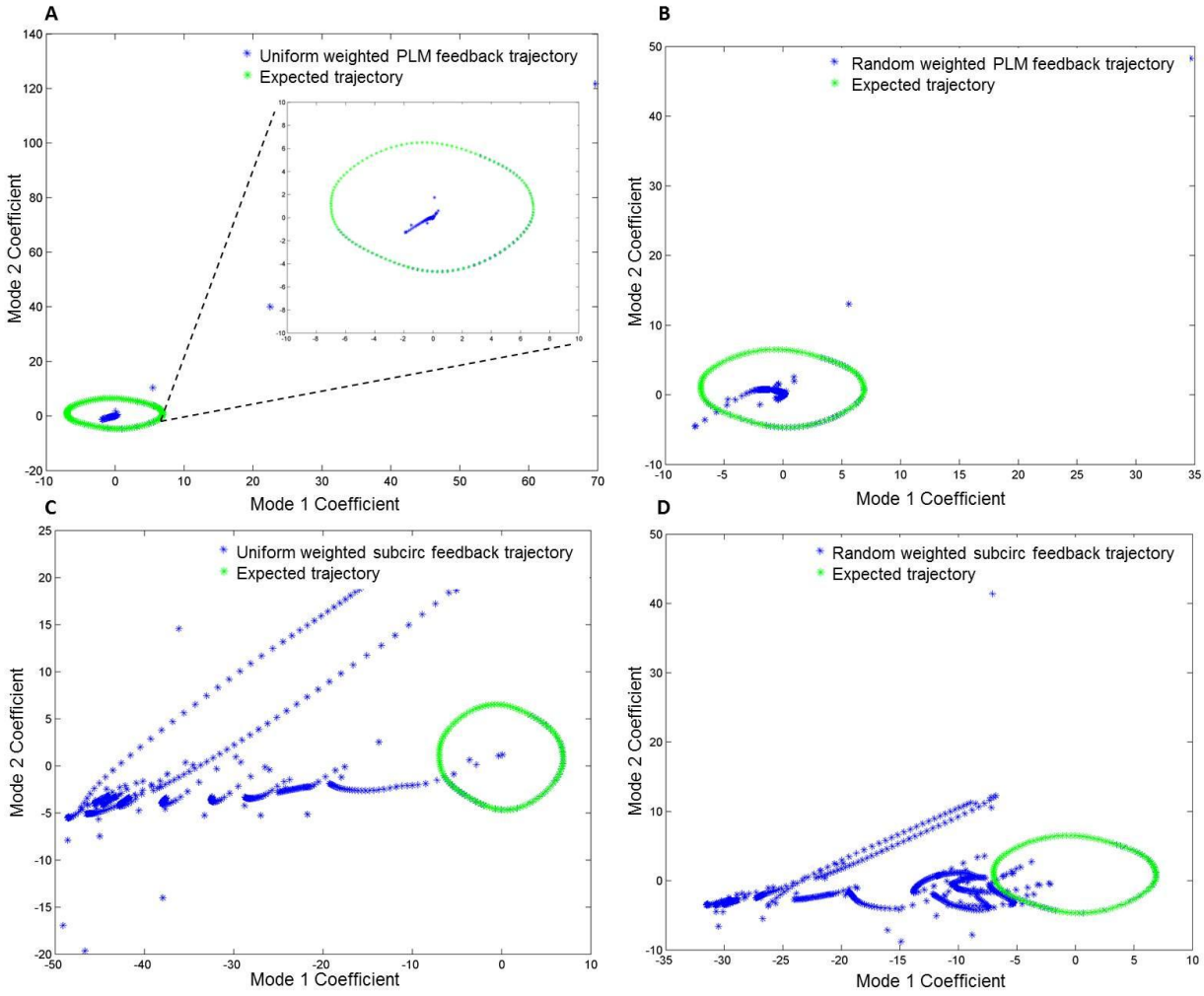

Figure 11. Examples of recurrent network dynamics with uniformly and randomly weighted feedback as compared to the expected forward motion dynamics. **A)** Simulation of network behavior with arbitrary constant uniform weights on feedback into PLM neurons **B)** Simulation of network behavior with constant random weights on feedback into PLM neurons **C)** Simulation of network behavior with arbitrary constant uniform weights on feedback into the sub-circuit of seven sensory neurons associated with forward motion **D)** Simulation of network behavior with constant random weights on feedback into the sub-circuit of seven sensory neurons.

into the seven neurons clearly elicits a neural response, the response does not align with the expected dynamics of forward motion and appears to be of random behavior.

B. Gradient Descent

Given our failure to create the dynamics of forward motion using random/uniform inputs or weights, we employ the method of gradient descent to search for appropriate system inputs

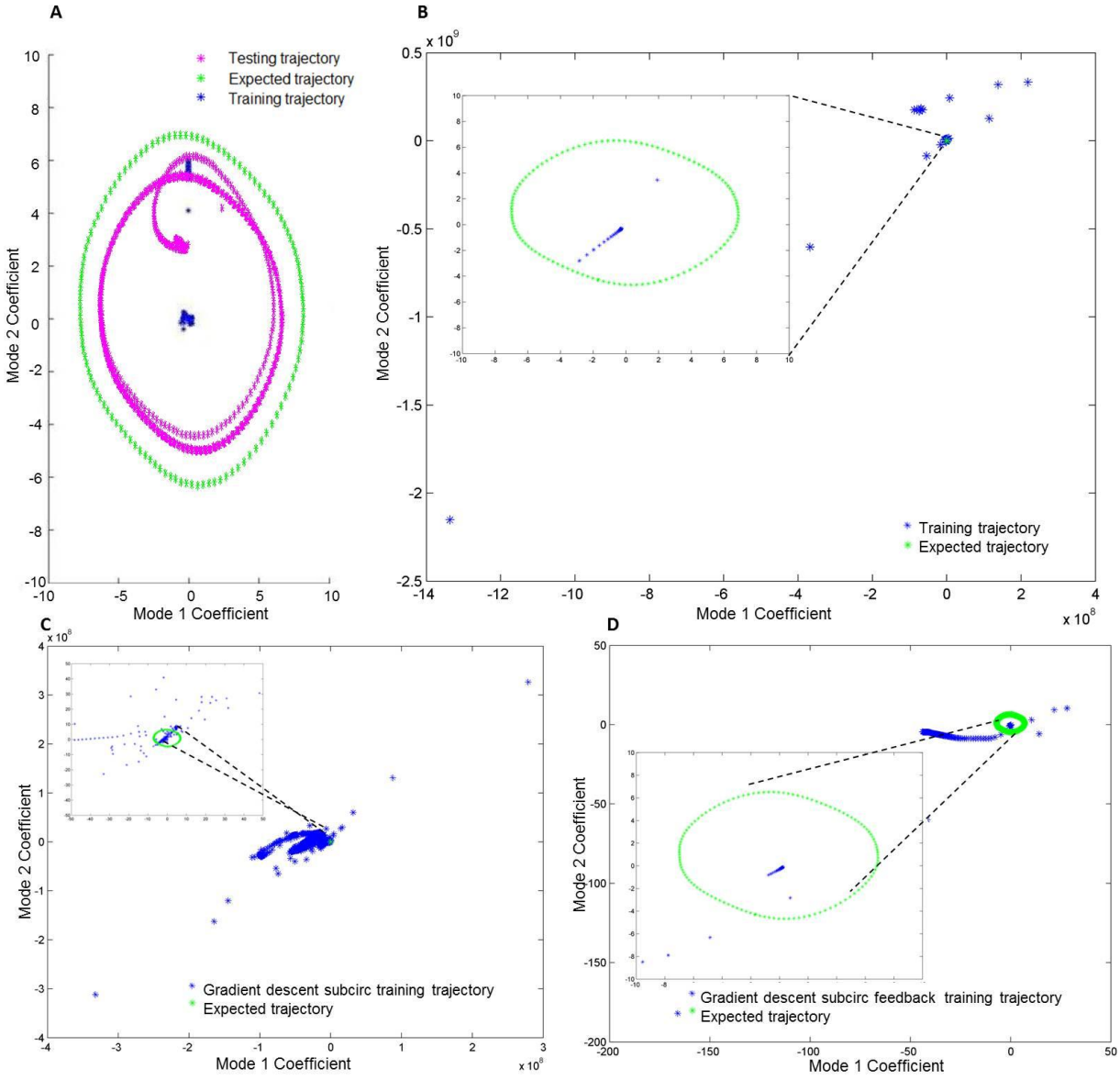

Figure 12. **A)** Convergence of motor output toward output expected during PLM excitation of $2x10^4$. Input modifications were made each time the trajectory of the activity crossed the positive imaginary axis (training points). After gradient descent training, PLML and PLMR ended at $1.8x10^4$ and $1.9x10^4$, respectively. **B)** Training of weights on feedback into PLM neurons, the algorithm did not converge and no oscillations were generated. **C)** Gradient descent results when training with input into seven sub-circuit sensory neurons without feedback **D)** Gradient descent results with training feedback input into seven sub-circuit sensory neurons.

and then to search for functional feedback weights. Using the previously described algorithm, we initialize the system with small random input into PLM neurons and search for values that will produce the network output expected during forward motion. The algorithmic converges on

values of $1.8x10^4$ and $1.9x10^4$ mV for the input into to PLMR and PLML, respectively. These input values, which are near the optimal known solution of $2x10^4$ mV, are determined over the course of approximately 70s of training (Figure 12A).

Although the method of gradient descent is able to correctly identify the input needed to PLM neurons to induce forward motion, when input is allowed into other neurons, no solution is found. In experiments introducing input into the sub-circuit of seven sensory neurons, our algorithm was unable to converge to the correct solution within 250s of training, which is about 200 periods of wave-like body deformations (Figure 12C). Furthermore, when we modify our algorithm to derive sensory input from feedback, such that the algorithm is setting weights on the feedback instead providing direct input values, the algorithm is again unable to converge (Figure 12 B,D). The addition of feedback causes significant disruption in the system output, which was previously avoided in the simple model by requiring low magnitude weight changes. Even with small weight changes, with feedback included, dramatic input changes can occur, which cause sharp irregularities in system output. Since the gradient descent method modifies weights based on the estimated derivative of network activity with respect to the weight change, these irregularities in output can cause erroneous weights changes and ultimately divergence.

C. Recursive Least Squares

We use our RLS algorithm with feedback only into PLM neurons. Using the proximity-based map from muscles to neurons, we find that weights do not converge with training durations up to 100s. In our experiments, we observe that forward motion dynamics are maintained for a short period of time during training if weights and feedback are updated according to the Poincaré map implemented in the method of gradient descent, and initial weights are favorably set. Even in this case, after several simulated seconds of training weights

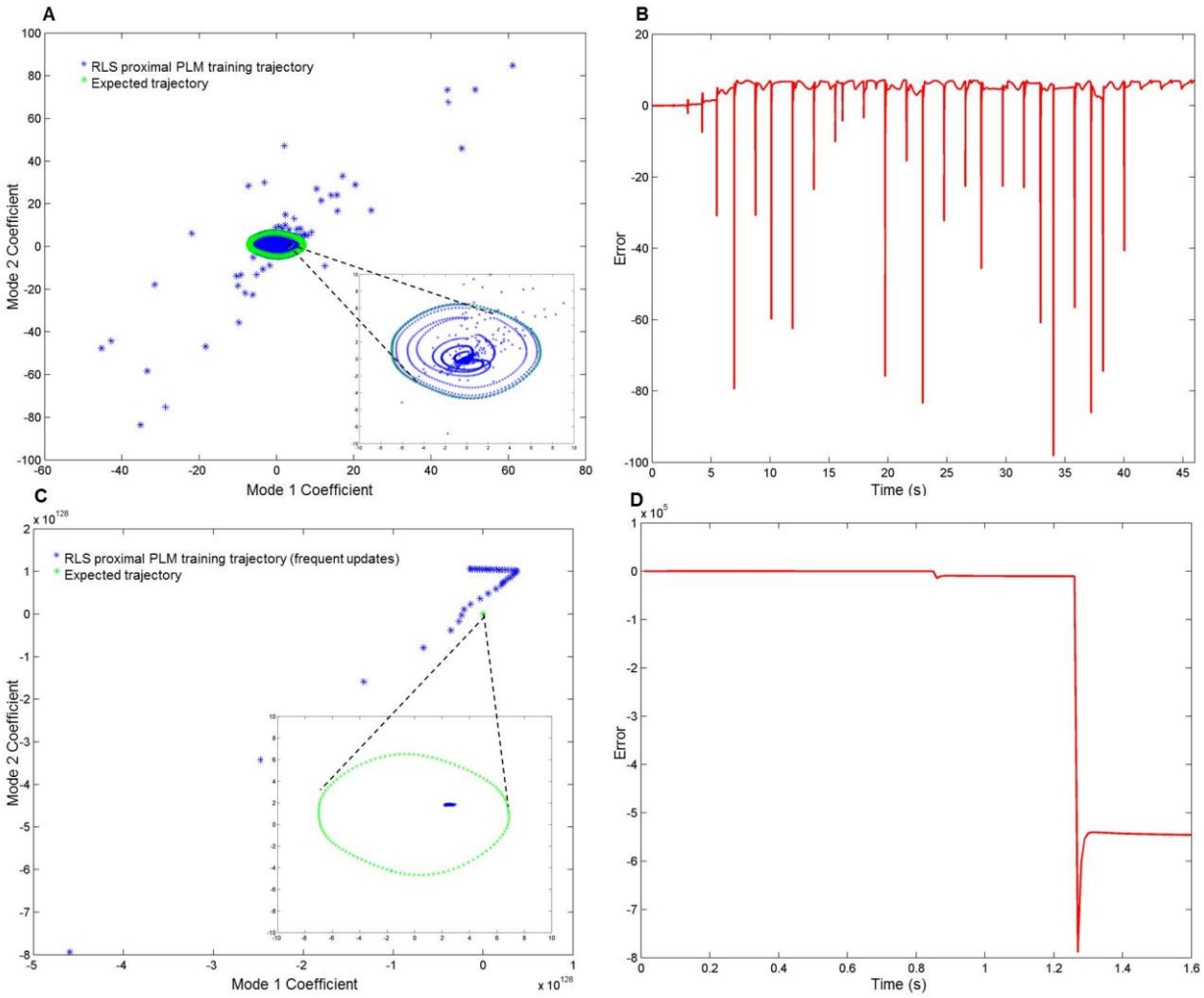

Figure 13. A) Training of PLM neurons using Poincaré map with proximity-based input from motor neurons. Weights do not converge and the system has repeated settling periods where activity approaches the resting equilibrium. **B)** Error during training shown in figure A. The large spikes in error are due to weight changes or forced stimulation due to system settling. **C)** Training of PLM neurons with more frequent weight and feedback updates (approximately 4 updates per cycle), also using proximity-based input from motor neurons. Weights quickly grow in magnitude due to large error. **D)** Error during training of PLM neurons in figure C. Error grows rapidly due to oscillations in motor neuron feedback into sensory neurons.

have not been adequately modified to maintain forward motion and ultimately the dynamics decay (Figure 13 A,B). A significant issue with our application of the RLS method using the proximal map is that the RLS method is intended to make frequent weight updates, but we are only updating weights once per cycle. This setup allows the network dynamics to diverge significantly from the desired behavior between weight updates, making it extremely difficult to

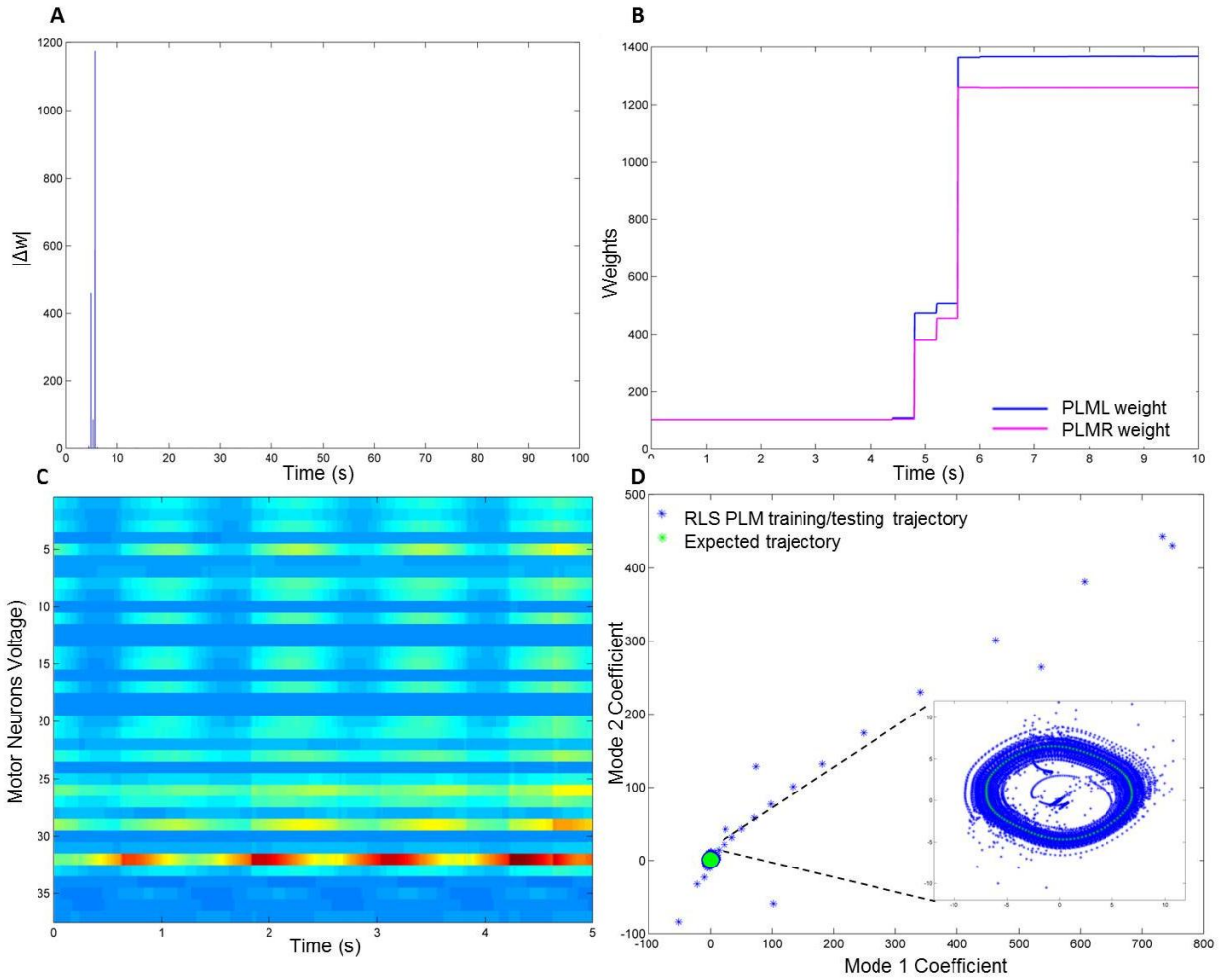

Figure 14. A) Magnitude of weight change over time. Large weight changes are made initially to reduce error, then significantly smaller weight changes are made until error is consistently near zero, at which point weight modification ceases. **B)** Actual values of the weights used on PLML and PLMR during training. Weights continue to change up through 10s of training, although the magnitude of weight change is very small, given the scale of the weights. **C)** Motor neuron voltage oscillations during operational phase (static weights), which can be compared with the oscillations during forward motion in Figure 1A. **D)** Trajectory of activity during training and testing as compared to the expected trajectory during forward motion.

correctly modify weights and converge to a solution within a reasonable training period. In particular, if PLM input has diminished, the network can settled to its resting equilibrium before the next weight update. We attempt to mitigate this scenario by providing extra stimulus when the system settles, but this jolt can cause a disjuncture between weights and sensory input. The overarching issues from infrequent weight updates could be resolved by updating weights and

feedback more frequently, but this modification introduces another challenge. As motor neuron activity oscillates during forward motion, feedback-based sensory neuron input also oscillates. However, oscillating input into PLM neurons does not generate oscillating network dynamics, as we have demonstrated through training and testing (Figure 13C). This oscillating input poses further challenges for the weight modification algorithm, as feedback alternates between positive and negative values, meaning weighted input switches from a large positive number to a large negative number, creating dramatic increases in error (Figure 13D). This rapid error increase leads to large weight modifications, which in turn create even larger swings in input, and the algorithm spirals toward infinite weights and infinite error.

We then enhance our algorithm to use the feedback based on viscoelastic rod, modeling the actual movement of the worm over time. We initialize the weights uniformly to a randomly chosen small number and find that weights generally converge within the first twenty simulated seconds of training when we consider input received from the viscoelastic rod into the PLM neurons (Figure 14 A,B). Once the weights have converged, the system enters the operational phase during which weights are held static. During this period of testing, we see that voltage oscillations in the motor neurons are very similar to those seen during forward motion (Figure 14C). We also map the trajectory of the motor neuron activity as compared to the expected forward motion trajectory during both training and testing, and we observe that the system settle to the known forward motion dynamics (Figure 14D).

Given our success in training PLM neurons to produce desired output using network feedback, we proceed to use the RLS algorithm with feedback into the sub-circuit of seven sensory neurons. With our static proximity-based map, we are unable to converge upon a set of weights that enables the system to produce the dynamics of forward motion (Figure 15A). We

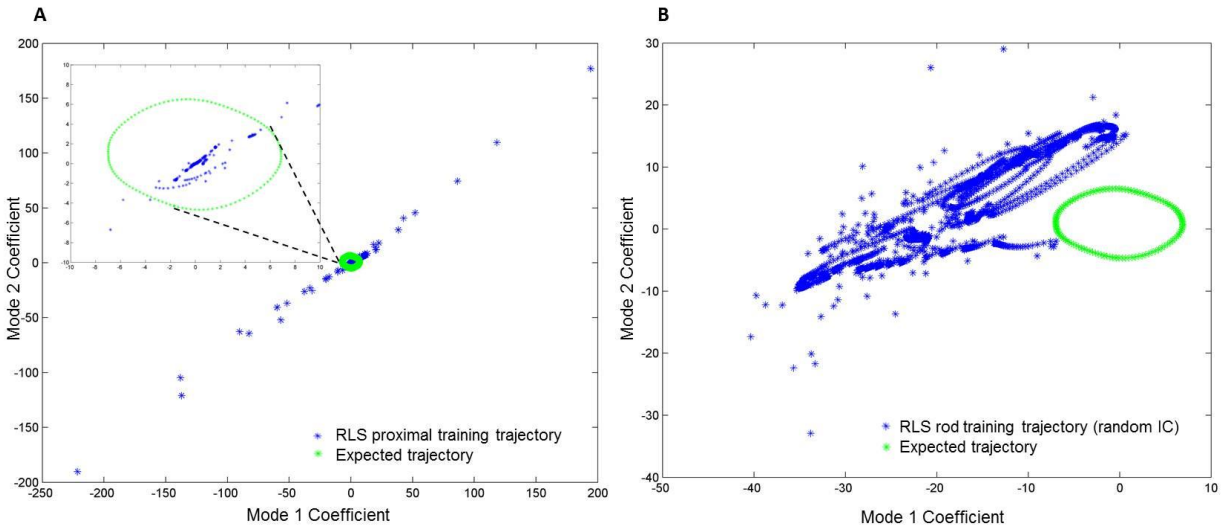

Figure 15. **A)** Results of training with input to seven sub-circuit sensory neurons using a static proximity-based map from motor neurons to sensory neurons to model feedback. **B)** Results of training with input into seven sub-circuit sensory neurons using feedback based on the viscoelastic rod model with randomly chosen small initial weights.

then advance our algorithm by calculate feedback based on our viscoelastic rod model, initializing weights with small randomly chosen values, as in PLM training. Over the course of training of up to one simulated hour (3600s), weights failed to converge, and the network activity did not generate the oscillations associated with forward motion. Although the resulting trajectories are centered in a region much closer to our desired trajectory, any oscillations that are occurring are not of the correct form or magnitude (Figure 15B).

In order to improve our algorithms ability to converge without continuing to extend our training period, we initialize weights based on observed neuron activity during forward motion. Thus, we sample voltage activity of the input neurons during standard simulated forward motion, and use these values as a basis for our initial weights, such that w_0 = *sample_voltages./initial_feedback.* With this methodology, we again see initial larger weight changes, followed by a period of small weight changes as error stays small (Figure 16A).

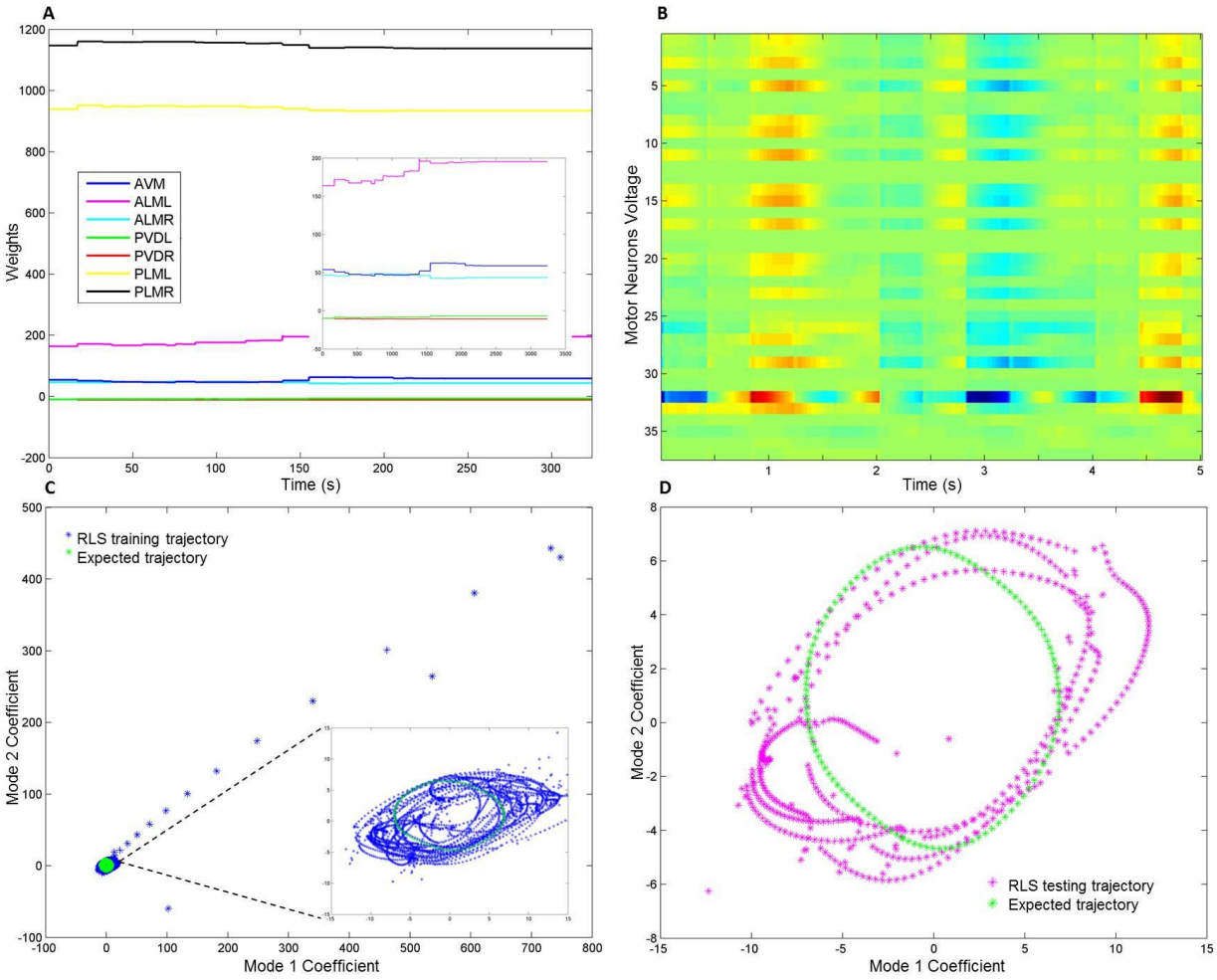

Figure 16. A) Weight changes during RLS training of the seven neurons associated with *C. elegans* crawling **B)** Motor neuron activity during testing using weights converged upon during RLS training **C)** Training trajectory as compared to the expected trajectory during forward motion **D)** Testing trajectory using weights converged upon during RLS training.

The algorithm then converges on weights that will generate dynamics and voltage patterns similar to those of PLM-based forward motion for approximately five to ten simulated seconds after training ceases (Figure 16 B-D). When the operational period extends past ten seconds, oscillations become highly irregular and disappear.

Adaptive Control Algorithm	Feedback	Initial Conditions	Sensory Neurons Receiving Input	Outcome
Gradient descent	**None**	**Random input of smaller magnitude than known solution**	**PLML/R**	**Input converges near known PLM solution resulting in forward motion dynamics**
Gradient descent	None	Random uniform with magnitude smaller than PLM solution ($2x10^4$mV)	ALML/R, AVM, PLML/R, PVDL/R	Divergence with error of magnitude 10^8
Gradient descent	Viscoelastic rod-based feedback	Weights initialized such that early feedback is near known solution	PLML/R	Divergence with error of magnitude10^8
Gradient descent	Viscoelastic rod-based feedback	Weights initialized based on observed neuronal activity during forward motion	ALML/R, AVM, PLML/R, PVDL/R	Diverges or settles to resting equilibrium, depending on magnitude of weights
RLS	Proximity based feedback	Weights initialized based on observed neuronal activity during forward motion	PLML/R	Divergence with error of magnitude10^{5+} with frequent weight updates, with Poincaré map, diverges or settles to equilibrium
RLS	Proximity based feedback	Weights initialized based on observed neuronal activity during forward motion	ALML/R, AVM, PLML/R, PVDL/R	Divergence with error of magnitude10^2
RLS	**Viscoelastic rod-based feedback**	**Small/randomly initialized weights**	**PLML/R**	**Converges with error of magnitude 1**
RLS	Viscoelastic rod-based feedback	Small/randomly initialized weights	ALML/R, AVM, PLML/R, PVDL/R	Erratic/random dynamics, error of magnitude 10
RLS	**Viscoelastic rod-based feedback**	**Weights initialized based on observed neuronal activity during forward motion**	**ALML/R, AVM, PLML/R, PVDL/R**	**Converges with error of magnitude 1, but dynamics of forward motion cease after approximately 5-10 simulated seconds**

Figure 17. **A)** Table summarizing training results using the two adaptive control algorithms to elicit forward motion dynamics with various network configurations.

CONCLUSION

In conclusion, we have developed a realistic model of physical movement based on motor neuron activity, and we have devised an adaptive control method that transforms feedback from our physical model into sensory neurons input, such that the desired network dynamics are

generated. We created this system based on an existing computational model of the *C. elegans* neural network with known dynamics of forward motion [1,3], and the worm's known physiological traits [5,7,15].

Using simple visualizations of neuron movement based on voltage changes, we recognized high-level similarities between neuronal activity and physical activity. We then created a viscoelastic rod-based model with the physical attributes of *C. elegans* and its habitat, and translated motor neuron activity into force applied to the rod. Activating the rod based on motor neuron voltage data from forward motion simulations, we confirmed that the motion of the rod mimicked the known patterns of *C. elegans* crawling forward [2].

With our functional model of the relationship between motor neuron activity and physical movement, we proceeded to investigate how to close the loop, integrating motor neuron activity, physical movement, and sensory neuron stimulation. We sought a means to use the state of the viscoelastic rod model as feedback into sensory neurons, which would then stimulate motor neurons and perpetuate forward motion. As experiments with uniformly or randomly weighting feedback from the viscoelastic rod into sensory neurons failed to generate the desired dynamics, we employed adaptive control methods to identify input weights that would enable the feedback to maintain the dynamics of forward motion.

After exploring control via the method of gradient descent and RLS, we found that our RLS algorithm was able to find input weights for PLM neurons such that the feedback from the viscoelastic rod perpetuated the dynamics of forward motion. With the relationship between feedback and PLM input established, we applied our algorithm to a model with input into seven sensory neurons known to be associated with *C. elegans* crawling. In this case, we found that the algorithm converges if weights are given realistic initial values based on sensory neuron activity

during forward motion. Using the converged weights, we observed that forward motion is perpetuated for five to ten simulated seconds after initial PLM excitation, which is realistic given that *C. elegans* reaction to touch does cease after a period of time.

The methods for simulating *C. elegans'* physical movement based on its neuronal activity described in this research provide a strong foundation for future development. To build on the existing model, fluid dynamics could be integrated to provide a more realistic representation of the body's interaction with the environment. The accuracy of the model could be improved further through increased freedom in *C. elegans* movement, which could be achieved by allowing force to be applied in other directions (e.g., horizontally), depending on body state and muscle location. To expand the scope of the physical movement simulations, different actions like backward motion or the Omega-turns described in [8] could be simulated using a viscoelastic rod model.

In the realm of adaptive control algorithms, the research presented in this paper describes methods for identifying optimal stimulation that could be leveraged in several fields of neuroscience and biomechanics. The RLS-based training methods we used to model feedback could be modified to help identify ideal inputs for the feedforward controlled systems used in functional electrical stimulation research. This method could also be used in the development of the "next generation" of deep brain stimulation, in which researchers are working to develop closed-loop stimulators that can automatically adjust output based on neural activity. Lastly, the optimal control algorithms discussed here could be used in conjunction with advancements in brain-machine interfaces like non-invasive brain stimulation, which can not only monitor neuronal activity, but also alter it and observe the impact of the modifications.

REFERENCES

1. Varshney LR, Chen BL, Paniagua E, Hall DH, Chklovskii DB: Structural properties of the Caenorhabditis elegans neuronal network. *PLoS Computational Biology*, 2011, **7**(2):e1001066.
2. Sengupta P, Samuel AD: Caenorhabditis elegans: a model system for systems neuroscience. *Current Opinion in Neurobiology*, 2009, **19**(6):637–643.
3. Kunert J, Shlizerman E, Kutz JN: Low-dimensional functionality of complex network dynamics: neurosensory integration in the Caenorhabditis elegans connectome. *Physical Review E, Statistical, Nonlinear, and Soft Matter Physics*, 2014, **89**(5):052805.
4. McMillen T, Williams T, Holmes P: Nonlinear muscles, passive viscoelasticity and body taper conspire to create neuromechanical phase lags in anguilliform swimmers. *PLoS Computational Biology*, 2008, **4**(8):e1000157.
5. Backholm M, Ryu WS, Dalnoki-Veress K: Viscoelastic properties of the nematode Caenorhabditis elegans, a self-similar, shear-thinning worm. *Proceedings of the National Academy of Sciences of the United States of America*, 2013, **110**(12):4528–4533.
6. Sussillo D, Abbott LF: Generating coherent patterns of activity from chaotic neural networks. *Neuron*, 2009, **63**(4):544–557.
7. Maricq AV, Peckol E, Driscoll M, Bargmann CI: Mechanosensory signaling in C. elegans mediated by the GLR-1 glutamate receptor. *Nature*, 1995, **378**(6552):78–81.
8. Stephens GJ, Johnson-Kerner B, Bialek W, Ryu WS: Dimensionality and dynamics in the behavior of C. elegans. *PLoS Computational Biology*, 2008, **4**(4):e1000028.
9. OpenWorm, Palyanov A, Szigeti B, Idili G, Hokanson J, Cantarelli M, Currie M, Gleeson P, Khayrulin S, Larson S (ed.s) 2011–2014. http://www.openworm.org/
10. WormAtlas, Altun ZF, Herndon LA, Crocker C, Lints R, and Hall DH (ed.s) 2002–2015. http://www.wormatlas.org
11. Tytell ED, Holmes P, Cohen AH: Spikes alone do not behavior make: why neuroscience needs biomechanics. *Current Opinion in Neurobiology*, 2011, **21**(5):816–822.
12. Tytell ED, Hsu CY, Williams TL, Cohen AH, Fauci LJ: Interactions between internal forces, body stiffness, and fluid environment in a neuromechanical model of lamprey swimming. *Proceedings of the National Academy of Sciences of the United States of America*, 2010, **107**(46):19832–19837.
13. Shlizerman E, Schroder K, and Kutz JN: Neural activity measures and their dynamics. *SIAM Journal on Applied Mathematics*, 2012, **72**(4):1260–1291.
14. Sirovich L: Modeling the functional organization of the visual cortex. *Physica D: Nonlinear Phenomena*, 1996, **96**(1–4):355–366.
15. Wicks SR, Roehrig CJ, Rankin CH: A dynamic network simulation of the nematode tap withdrawal circuit: predictions concerning synaptic function using behavioral criteria. *The Journal of Neuroscience*, 1996, **16**(12):4017–4031.
16. Sznitman J, Purohit PK, Krajacic P, Lamitina T, Arratia PE: Material properties of Caenorhabditis elegans swimming at low Reynolds number. *Biophysical Journal*, 2010, **98**(4):617–626.
17. Haykin SS: Adaptive filter theory, 3rd edn. Upper Saddle River, N.J.: Prentice Hall; 1996.
18. Paoletti P, Mahadevan L: A proprioceptive neuromechanical theory of crawling. *Proceedings of the Royal Society of London B*, 2014, **281**(1790):20141092.

APPENDIX

Parameters of Viscoelastic Rod

The parameters defining the remaining physical characteristics of the rod were set based on published analyses of high-speed images of *C. elegans* forward-swimming in a controlled environment (Material Properties of *C. elegans*) and micropipette deflection experimentation on anesthetized *C. elegans* (Viscoelastic properties of the nematode *C. elegans*). We model an adult *C. elegans* with diameter $D = 65\mu m$, Young's modulus $E = 3.77kPA$ and material density $\rho = 1.0\frac{g}{cm^3}$. We assume the preferred curvature to be the equilibrium state of the nematode, represented by the zero vector, and assign damping coefficient $\delta = 1\frac{Ns}{m}$ as an approximate representation of the substrates making up *C. elegans* habitat (broadly ranging from rotting vegetation to animal intestines). As previously described, we assign segment lengths to vector $h_i$based on approximate muscle length.

Derivation of RLS Equation and *P* Matrix

We begin our derivation with the equations

$$P(t) = \left(\sum_t r(t)r^T(t) + \alpha I\right)^{-1} \tag{50}$$

and $P(t) = P(t - \Delta t) - \frac{P(t-\Delta t)r(t)r^T(t)P(t-\Delta t)}{1+ r^T(t)P(t-\Delta t)r(t)}$ (51)

defining matrix P from *General Patterns of Chaotic Networks*. We derive Eq. (50) from the recursive least squares algorithm applied to a system with desired output *f(t)*, actual output *z(t)*, activity *r(t)* and weights *w(t).* We define the relationship between output and weights as *z(t)* =

$w^T r(t)$ and calculate error as $e(t) = f(t) - w^T r(t) = z(t) - f(t)$. Thus, by the least squares criterion, we set $E(t) = (f(t) - z(t))^2$. We then find the least squares solution to the setting of weights, $\widetilde{w}$, by minimizing the sum of squared error (SSE),

$$SSE = \sum_n E(t) = e^T(t)e(t), \text{letting } \widetilde{w} = w^T$$

$$= (f(t) - \widetilde{w}r(t))^T (f(t) - \widetilde{w}r(t))$$

$$= (f^T(t) - r^T(t)\widetilde{w}^T)(f(t) - \widetilde{w}r(t))$$

$$= f^T(t)f(t) - f^T\widetilde{w}r(t) - r^T(t)\widetilde{w}^T f(t) - r^T(t)\widetilde{w}^T\widetilde{w}r(t)$$

$$= f^T(t)f(t) - 2f^T r\widetilde{w}r(t) - r^T(t)\widetilde{w}^T\widetilde{w}r(t).$$

Differentiating SSE with respect to $\widetilde{w}$, we find

$$\frac{\partial SSE}{\partial \widetilde{w}} = \frac{\partial}{\partial \widetilde{w}}(f^T(t)f(t) - 2f^T r\widetilde{w}r(t) - r^T(t)\widetilde{w}^T\widetilde{w}r(t))$$

$$= -2r^T(t)f(t) + 2r^T(t)r(t)\,\widetilde{w}.$$

Setting this result to zero, we find the optimal value of $\widetilde{w}$,

$$0 = -2r^T(t)f(t) + 2r^T(t)r(t)\widetilde{w}$$

$$r^T(t)r(t)\widetilde{w} = r^T(t)f(t)$$

$$\widetilde{w} = \frac{r^T(t)f(t)}{r^T(t)r(t)}.$$

We expand $\widetilde{w}$ over *t*, which yields

$$\widetilde{w}(t) = \sum_t r(t)r^T(t)^{-1} \sum_t r(t)f(t),$$

$$\widetilde{w}(t) = \sum_t r(t)r^T(t)^{-1} \sum_t r(t)f(t) = \Phi^{-1}(t)\psi(t),$$

where $\Phi(t) = \sum_t r(t)r^T(t)$

and $\psi(t) = \sum_t r(t)f(t)$.

Now, we observe that $\Phi(t)$ is the inverse of Eq. (50) without the regularization term. To recursively compute $\Phi(t)$ and $\psi(t)$ based on $t - \Delta t$, we rewrite our equations as

$$\Phi(t) = \Phi(t - \Delta t) + r(t)r^T(t)$$

and $\psi(t) = \psi(t - \Delta t) + r(t)f(t)$.

Setting $P(t) = \Phi^{-1}(t)$ as in Eq. (49), we can rewrite *P(t)* as

$$P(t) = \left(\Phi(t - \Delta t) + r(t)r^T(t)\right)^{-1}.$$

To verify the equality of *P(t)* from Eq. (50) and Eq. (51), we use the Woodbury Matrix Identity,

$$(A + UV^T)^{-1} = A^{-1} - A^{-1}U(C^{-1} + V^TA^{-1}U)^{-1}V^TA^{-1}.$$

Substituting Eq. (50) into the RHS of the Woodbury Matrix Identity yields

$$A^{-1} = P(t - \Delta t); U = V = r(t); C = I.$$

Since $\Phi(t - \Delta t) = P^{-1}(t - \Delta t)$,

$$I = P^{-1}(t - \Delta t) + r(t)r^T(t)\{P(t - \Delta t)$$

$$-P(t - \Delta t)r(t)(I + r^T(t)P(t - \Delta t)r(t))^{-1}r^T(t)P(t - \Delta t)\}$$

$$= I + r(t)r^T(t)\,P(t - \Delta t) - \frac{(r(t) + r(t)r^T(t)P(t - \Delta t)r(t))r^T(t)P(t - \Delta t)}{I + r^T(t)P(t - \Delta t)r(t)}$$

$$= I + r(t)r^T(t)\,P(t-\Delta t) - \frac{r(t)(I + r^T(t)P(t-\Delta t)r(t))r^T(t)P(t-\Delta t)}{I + r^T(t)P(t-\Delta t)r(t)}$$

$$= I + r(t)r^T(t)P(t-\Delta t) - r(t)r^T(t)P(t-\Delta t) = I.$$

Thus, we have verified the equality

$$P(t) = \sum_t r(t)r^T(t)^{-1} = \left(\Phi(t-\Delta t) + r(t)r^T(t)\right)^{-1} = P(t-\Delta t) - \frac{P(t-\Delta t)r(t)r^T(t)P(t-\Delta t)}{1 + r^T(t)P(t-\Delta t)r(t)}.$$