Learning Deterministic Spiking Neuron Feedback Controllers

Tae Seung Kang and Arunava Banerjee
Computer & Information Science & Engineering
University of Florida
Gainesville, FL 32611
Email: {tsk, arunava}@cise.ufl.edu

***Abstract*—We consider the problem of feedback control when the controller is constructed solely of deterministic spiking neurons. Although spiking neurons and networks have been the subject of several previous studies, analysis has primarily been restricted to a firing rate model. In contrast, we construct a deterministic spiking neuron controller whose control output is one or multiple sparse spike trains. We model the problem formally as a hybrid dynamical system comprised of a closed loop between a plant and a spiking neuron network controller. The construction differs from classical controllers owing to the fact that the control feedback to the plant is generated by convolving the spike trains with a fixed kernel, resulting in a highly constrained and stereotyped control signal. We derive a novel synaptic weight update rule via which the spiking neuron controller *learns* to hold process variables at desired set points. We demonstrate the efficacy of the rule by applying it to the classical problem of the cart-pole (inverted pendulum). Experiments demonstrate that the proposed controller has a larger region of stability as compared to the traditional PID controller and its trajectories differ from those of the PID controller.**

I. Introduction

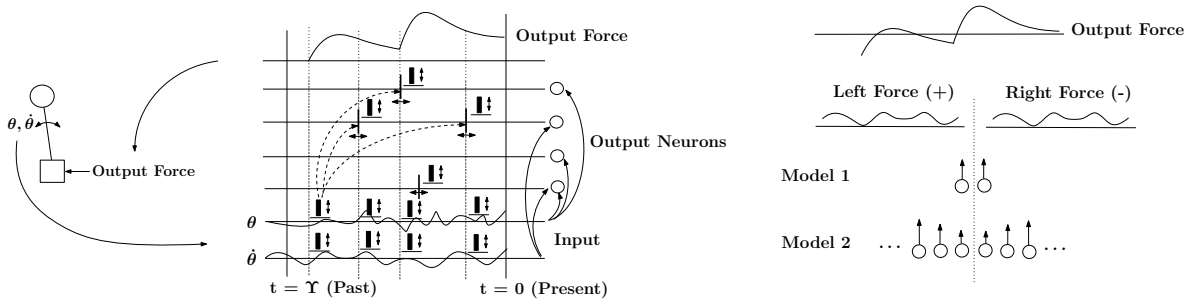
While there is considerable debate in the scientific community regarding the cognitive capacity of various animal species, there is general agreement that animals are exquisite control systems. Whether it be the flight of a dragonfly or the walking of a biped (such as a human), engineered systems pale in comparison to the versatility and robustness displayed by their animal counterparts. Even more intriguing is the fact that in many instances the particular skill, locomotion for instance, is *learned*. Our goal in this paper is to address this question of learning to control in the context of biologically motivated constraints—specifically, the fact that the constituent neurons of animal brains communicate with one another using action potentials (also known as spikes).

In the vast majority of biological systems, the control signal received by the muscles are in the form of spike trains generated by motor neurons. The controller itself is a network of spiking neurons that resides upstream from the motor neurons. The controller receives inputs, which in the case of a feedback controller are process variables that are to be maintained at fixed or dynamically varying set points. The process variable input into the controller is in turn computed elsewhere and incorporates the combined output of one or more sensory systems.

To bring the problem of learning a spiking neuron network controller into sharp relief, we abstract away all aspects of the system that are of secondary concern and replace them with simple, fixed, and predefined alternatives. In particular, we model the entire process beginning at the spike train output of the controller and culminating at the control signal generated (such as the force exerted by the muscle) using fixed convolution kernels. The impact of the control signal on the organism in its environment, we model using a fixed plant. Finally, we model the input of the process variables as postsynaptic potential inputs into specifically identified neurons of the controller. Our objective is to devise a formal synaptic weight update rule that when applied to the neurons of the controller, causes the controller to learn to perform the control task.

That the above problem differs from those previously studied in feedback control, can be discerned from the following observation. Traditional feedback controllers such as the proportional-integral-derivative (PID) controller [1] or its variants are designed to solve a control problem in the continuous domain with few restrictions. The process variable is a bounded continuous function of time, and so is the control signal generated by the controller; there is little else that constrains these functions. In contrast, the control output generated by the spiking neuron network controller is an ensemble of spike trains. The spike trains when convolved with the fixed convolution kernel referred to above, leads to a highly restricted and stereotyped signal. In particular, it is easy to observe that given a kernel, there exists a bound C such that any non-zero control signal f satisfies $\|f\|_\infty > C$—informally, the controller has the choice between generating no output or an output larger than a fixed strength. This has immediate implications for the stability of the fixed point (determined by the set point) of the combined (the controller and the plant in closed loop) dynamical system; the process variable can at best be made to oscillate around the set point.

The overall goal of the paper is to demonstrate that deterministic spiking neuron based controllers can be *learned*, and not to characterize a pre-given spike based controller. To our knowledge, this is the first deterministic spike based controller that has been demonstrated to *learn* a classical task. The controller does operate very much like a bang bang controller, and it has *learned* to operate that way. This work is a first



(a) Plant (b) Controller (c) Models

Fig. 1. The hybrid dynamical system that models the control problem. For more details, refer to the text. (a) Plant. The cart-pole plant has a state that can be changed by an external force. The state is described by the vertical angle θ and the angular velocity $\dot{\theta}$. The cart can only move left or right. For more details, refer to Section III. (b) Controller. The proposed spiking neuron controller is a feedforward neuron network that takes the plant state as input and produces an output force to control the plant. The synaptic update rules are set such that the weights on the synapses receiving the continuous process variable inputs change only when there are spikes generated by the "output neurons" of the controller. This is indicated by the vertical dotted lines. Perturbing the synaptic weights perturb the output spikes of the network in time (dotted arrows), which when convolved with the kernel creates a perturbation in the control signal. The control signal is optimized based on an error function that embodies the deviation of the process signal from its set point. (c) Models. We present two models, Model 1 and Model 2, based on the number of output neurons of the controller and their corresponding force magnitude. A circle indicates an output neuron of the proposed controller shown in (b). The vertical arrow stemming from a circle denotes a force produced by the neuron with its length representing the force magnitude. The longer the length, the larger the force magnitude. Model 1 has two output neurons to generate a force in the left direction (left force) and a force in the right direction (right force) with both forces having the same magnitude. Model 2 has 4 or more output neurons with half contributing to the left force and half right force, with forces having different magnitudes. In both models, the number of output neurons and the force magnitudes are symmetric for the left and right. The final output force that would be applied to the plant is shown at the top. It is generated by summing up the left and right forces. Note that the left force has a positive sign and the right force has a negative sign.

step toward control of more complex dynamical systems such as winged flight or bipedal walking.

Our objective is described schematically in Figure 1. We consider a hybrid dynamical system constructed out of a closed loop between a plant (we consider the classical problem of the inverted pendulum in this paper as described in Section III, but this could be replaced with any well defined plant), and a network of spiking neurons that models the controller. In the figure, the black vertical bars indicate the weights on spikes. That is, we virtually assign weights to spikes (denoted by the height of the bars) instead of the corresponding synapse. The conceptual underpinnings of this are described in section IV. The vertical double-headed arrows next to the black bars denote the perturbation of the corresponding weights. The four dashed arrows beginning at the second vertical lines from t = Υ (Past) denote the impact of the weight perturbations on the subsequent future spike times. We present two models, Model 1 and Model 2, based on the number of output neurons of the controller and their corresponding force magnitudes. Model 1 is defined as a network with two output neurons (for left and right directions) with the same force magnitude, and Model 2 is defined as a network with four or more output neurons (two or more left and two or more right) with distinct force magnitudes. The goal is to incrementally update the synaptic weights on the neurons of the network such that the network's output spike trains when convolved with the force kernel causes the process variables of the plant to deviate as little as possible from predefined set points. The process variables are in turn input into the spiking neuron network controller as postsynaptic potentials.

Our approach is based on the observation that if (a) synaptic weights are only updated at the times that the network generates spikes, and (b) one analyzes the past times at which the network generated spikes, one can then perform a perturbation analysis that would recommend a superior set of weights in the past. Since we can not reach into the past to change synaptic weights, the current synaptic weights of the network are updated to reflect these improvements. The synaptic update rule is used to train the network from randomly generated initial conditions of the plant in an online manner until failure. At failure, the plant is reinitialized to a different initial condition and the learning process continues.

The remainder of the paper is structured as follows. Section 2 describes the neuron model. Section 3 briefly describes the plant used in this paper, as well as the process variables and their corresponding set points. Section 4 comprises the core of our contribution where the synaptic weight update rule is derived. Section 5 describes experimental results from several variations of the controller, and Sections 6 and 7 present related work and conclusions.

II. Neuron model

We use a minor variation of the Spike Response Model (SRM) [2] for the neurons in our controller. The neuron receives continuous time process variable inputs at its synapses. Although our analysis seamlessly generalizes to postsynaptic potentials generated from afferent (incoming) spikes at synapses, we do not consider that here. The membrane potential at the soma of the neuron is the synaptically weighted sum of postsynaptic potentials (PSPs) generated by the current values of the process variables and afterhyperpolarizing potentials (AHPs) generated by the efferent (outgoing) spikes that have departed the soma of the neuron. The neuron generates a spike when the membrane potential crosses the threshold Θ

from below. Formally, the membrane potential of a neuron at the current time is given by

$$P(t) = \sum_{i \in \Gamma} w_i \; x_i(t) + \sum_{k \in \mathcal{F}} \eta(t_k^O). \tag{1}$$

where Γ is the set of synapses, w_i is the weight of synapse i, $x_i(t)$ is the continuous process variable input signal at synapse i, and $t = 0$ is the current time (with positive t indicating past). Similarly, η is the prototypical after-hyperpolarizing potential (AHP) elicited by an efferent spike of the neuron, t_k^O is the elapsed time since the departure of the k^{th} most recent efferent spike, and $\mathcal{F}$ is the set of past efferent spikes of the neuron. We assume in addition that all efferent spikes that were generated earlier than $t = \Upsilon$ in the past have no effect on the present membrane potential of the neuron (See Figure 1). The functional form of the AHP of a spike that we have used (and this can be modified without affecting the analysis) is

$$\eta(t) = Re^{-t/\gamma} \quad \text{for} \;\; 0 < t \leq \Upsilon \quad \text{and 0 otherwise} \tag{2}$$

where R denotes the instantaneous fall in potential after a spike and γ controls its rate of recovery.

III. Plant

The plant we consider in this paper is the classical control problem of the cart-pole (also known as the inverted pendulum). The cart-pole comprises of an inverted rigid pendulum, with the mass at the top. The pendulum is fulcrumed at its base to the cart which rests on a frictionless surface. Force can be applied to the cart to move it along the horizontal axis. The control problem is to apply forces to the cart to maintain the upright position of the pendulum. The process variables that we have considered in this paper are: θ, the angular deviation of the pendulum from the upright position, and $\dot{\theta}$, the angular velocity of the pendulum. The set points for the process variables are $\theta = 0, \dot{\theta} = 0$. The details of the system dynamics can be found in [3]. All quantities of interest as presented in the next section, we have derived through numerical computations.

IV. Feedback control using spiking neurons

As described above, the desired state of the plant is to maintain a zero vertical angle and a zero angular velocity of the inverted pendulum. These process variables are input into different synapses of the controller neurons in the following form: the angles θ, and $-\theta$, and the angular velocities $\dot{\theta}$, and $-\dot{\theta}$. As described in Section 5, we have experimented both with the case where $\dot{\theta}$ is a process variable to be controlled at the set point 0, and the case where it is not. The control signal output of a neuron is generated by convolving the output spike trains of the neurons with a fixed force kernel $\kappa(t)$ as defined in the next section. Even number of neurons are used to generate forces, the first set generating forces to the right and the second set to the left. We analyze the general case of multiple neurons with different $\kappa(t)$ force kernels coming together to constitute the final control signal.

A. The Error Function

The proposed spiking neuron based controller depicted in Fig 1 can be formally modeled as follows. Consider a plant with process variables represented by vector $\langle x_1(0),\, x_2(0),\, ...,\, x_{\mathcal{D}}(0)\rangle$, where $\mathcal{D}$ is the number of process variables to be controlled. The desired state of the plant (i.e., the set point of the process variables) is represented by $\langle x_1^*(0),\, x_2^*(0),\, ...,\, x_{\mathcal{D}}^*(0)\rangle$. The error E can then be defined by $\frac{1}{2}\sum_{i=1}^{\mathcal{D}}(x_i(0) - x_i^*(0)^2$. The synaptic update rule that we derive next is based on minimizing this objective using gradient descent, which in the case of the full set of process variables and assuming set points of 0, reduces to:

$$E = \frac{1}{2}\sum_{i=1}^{D}(x_i(0))^2. \tag{3}$$

A traditional controller receives continuous time process signals from the plant and generates a continuous time control output. The proposed controller, however, generates spike trains, one for each neuron, instead of a continuous output. The spike train output of each neuron j is then convolved with the kernel

$$\kappa(t) = te^{-t/\tau_f} \tag{4}$$

to generate a force:

$$F_j = \mu_j \sum_{i \in \mathcal{F}_j} \kappa({}^j t_i^O) \tag{5}$$

where τ_f is the time constant, μ_j is the magnitude assigned to neuron j, ${}^j t_i^O$ is the time elapsed since the generation of the i^{th} most recent efferent spike of the output neuron j, and $\mathcal{F}_j$ is the set of past spikes of output neuron j. The final force F applied to the plant is

$$F = \sum_j \pm F_j \tag{6}$$

where $\pm$ represents the direction of F_j, $+$ for neurons that push to the right and $-$ for those that push to the left.

B. Gradients of the Error function

Our overall objective is to compute the gradient of the error with respect to the synaptic weights on the controller neurons. We do this in stages. We first compute the gradient with respect to the output spike times of the controller neurons. Applying chain rule, we then have

$$\frac{\partial E}{\partial({}^j t_l^O)} = \frac{\partial E}{\partial F}\frac{\partial F}{\partial({}^j t_l^O)} \tag{7}$$

where ${}^j t_l^O$ is the time elapsed since the departure of the l^{th} most recent efferent (outgoing) spike of neuron j, and

$$\frac{\partial E}{\partial F} = \sum_i \frac{\partial E}{\partial x_i}\frac{\partial x_i}{\partial F} = \sum_i x_i \frac{\partial x_i}{\partial F} \tag{8}$$

$$\frac{\partial F}{\partial({}^j t_l^O)} = \frac{\partial F}{\partial F_j}\frac{\partial F_j}{\partial \kappa({}^j t_l^O)}\frac{\partial \kappa({}^j t_l^O)}{\partial({}^j t_l^O)} = \pm\mu_j \frac{\partial \kappa}{\partial t}\Big|_{{}^j t_l^O} \tag{9}$$

In Eq (8), $\frac{\partial x_i}{\partial F}$ is drawn as a numerical derivative from the plant: $\frac{\partial x_i}{\partial F_j} \approx \frac{\Delta x_i}{\Delta F_j}$.

C. Perturbation analysis

Our goal now is to determine how perturbations in synaptic weights of the controller neurons translate to perturbations in the times of their output spikes. We achieve this by first assuming that synaptic weights are only perturbed at the times of the output spikes (see Figure 1). Consider the state of a neuron at the time of the generation of output spike t_l^O. The membrane potential of the neuron before perturbations of the weights on the input signals is given by

$$\tilde{\Theta} = \sum_{i\in\Gamma} w_{i,l} x_i(t_l^O) + \sum_{k\in\mathcal{F}} \eta(t_k^O - t_l^O). \tag{10}$$

where Γ is the set of synapses of the neuron and $w_{i,l}$ is the weight of synapse i immediately prior to output spike l. Note that we have replaced Θ with $\tilde{\Theta}$ to account for those output spikes that at the time of the generation of t_l^O were less than Υ old, but are now past that bound. If the synaptic weights were perturbed, this would cause the output spike t_l^O to be correspondingly perturbed according to

$$\tilde{\Theta} = \sum_{i\in\Gamma} (w_{i,l} + \Delta w_{i,l}) x_i(t_l^O + \Delta t_l^O) \tag{11}$$

$$+ \sum_{k\in\mathcal{F}} \eta(t_k^O - t_l^O + \Delta t_k^O - \Delta t_l^O). \tag{12}$$

Using a first order Taylor approximation, we get

$$\tilde{\Theta} = \sum_{i\in\Gamma} (w_{i,l} + \Delta w_{i,l}) \left(x_i(t_l^O) + \frac{\partial x_i}{\partial t}\Big|_{t_l^O} \Delta t_l^O \right) \tag{13}$$

$$+ \sum_{k\in\mathcal{F}} \left(\eta(t_k^O - t_l^O) + \frac{\partial \eta}{\partial t}\Big|_{(t_k^O - t_l^O)} (\Delta t_k^O - \Delta t_l^O) \right). \tag{14}$$

Combining Eq (10) and (13), dropping higher order terms and rearranging, we get

$$\Delta t_l^O = \frac{\sum_{i\in\Gamma} \Delta w_{i,l} x_i(t_l^O) + \sum_{k\in\mathcal{F}} \frac{\partial \eta}{\partial t}\Big|_{(t_k^O - t_l^O)} \Delta t_k^O}{\sum_{i\in\Gamma} w_{i,l} \frac{\partial x_i}{\partial t}\Big|_{t_l^O} + \sum_{k\in\mathcal{F}} \frac{\partial \eta}{\partial t}\Big|_{(t_k^O - t_l^O)}} \tag{15}$$

We can now derive the final set of quantities of interest from Eq (15). If we perturb the weight $w_{i,l}$, there will only be a direct effect of the perturbation since $w_{i,l}$ does not impact spikes prior to t_l^O. Therefore, we have

$$\frac{\partial t_l^O}{\partial w_{i,l}} = \frac{x_i(t_l^O)}{\sum_{i\in\Gamma} w_{i,l} \frac{\partial x_i}{\partial t}\Big|_{t_l^O} + \sum_{k\in\mathcal{F}} \frac{\partial \eta}{\partial t}\Big|_{(t_k^O - t_l^O)}}. \tag{16}$$

If instead, we perturb $w_{i,p}$ where $p > l$ (so that $t_p^O > t_l^O$), there will only be an indirect effect of the perturbation through previously generated spikes. Therefore, for $p > l$ we have the recursion

$$\frac{\partial t_l^O}{\partial w_{i,p}} = \frac{\sum_{k\in\mathcal{F}} \frac{\partial \eta}{\partial t}\Big|_{(t_k^O - t_l^O)} \frac{\partial t_k^O}{\partial w_{i,p}}}{\sum_{i\in\Gamma} w_{i,l} \frac{\partial x_i}{\partial t}\Big|_{t_l^O} + \sum_{k\in\mathcal{F}} \frac{\partial \eta}{\partial t}\Big|_{(t_k^O - t_l^O)}}. \tag{17}$$

D. Learning rules

Learning is accomplished via gradient descent. The learning rule is a type of Spike Timing-Dependent Plasticity (STDP) [4]; the weight updates depend on spike times. The reason that the weights should be updated only when there are spikes is as follows. If there are no spikes generated, the pole is in a safe kinematic range and thus no control signal is necessary. This, in turn, indicates no need to update the weights. The weight updates are not independent. They get related to each other due to the common error functional on which gradient descent is performed. Applying chain rule, we get

$$\frac{\partial E}{\partial w_{i,p}} = \sum_{l\in\mathcal{F}} \frac{\partial E}{\partial t_l^O} \frac{\partial t_l^O}{\partial w_{i,p}}. \tag{18}$$

This is computed using Eq 7, 16, and 17. The weight modification rule for synapse i at t_p^O is defined as

$$w_{i,p} \leftarrow w_{i,p} - \alpha \frac{\partial E}{\partial w_{i,p}} \tag{19}$$

where α is the learning rate. Clearly we can not reach into the past to make these changes. We therefore institute a summed delayed update to the synapse at the current time.

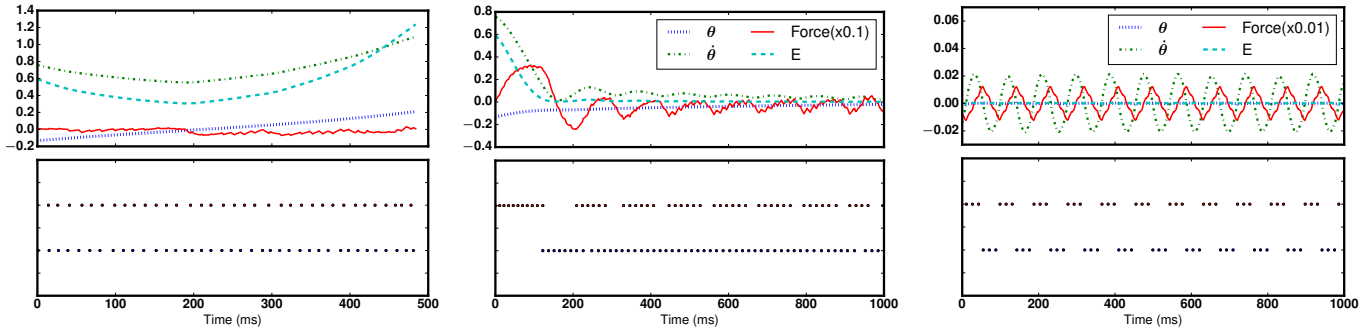
$$w_i \leftarrow w_i - \sum_{p\in\mathcal{F}} \alpha \frac{\partial E}{\partial w_{i,p}} \tag{20}$$

The weight update is performed immediately after the generation of a spike by any of the output neurons. This way we are guaranteed that the weights on the synapses remain constant between any two successive spikes (on any neurons).

V. Experiments

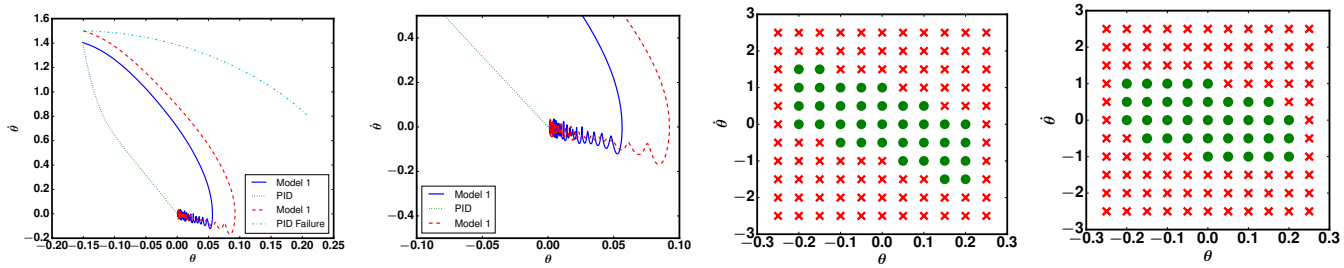
A. Setup

To demonstrate the efficacy of the proposed controller, we performed multiple simulations. For these simulations, we defined a successful learning event of the controller as having balanced the pole without failure for one hour of simulation time. A failure was defined as an event where the process variables of the pole was not within a certain predefined range. For example, given the 1ms time step, the number of steps for a successful run was 3600000 (1 hour). The predefined range was [-0.2094, 0.2094] for θ and [-2.01, 2.01] for $\dot{\theta}$. Training of the controller was continued until success. During the training phase, the controller learned by changing the synaptic weights. Once it had successfully learned the task, we fixed the weights and tested the controller with random initial plant states to evaluate its robustness. If the pole fell before training succeeded, we restarted training with random



(a) Start condition before training (b) Start condition after training (c) Stable condition after training

Fig. 2. Snapshots of the plant state (top) and spike trains of the controller (bottom) for Model 1 with two output neurons before and after training. In all top figures, the blue dotted line is the vertical angle θ, the green dash-dot is the angular velocity $\dot{\theta}$, the cyan dashed line is the error function E, and the red solid line is the force applied to the plant. (a) Before training, the synaptic weights are randomly chosen and fixed (no updates). As expected, the pole falls down quickly as the controller did not learn how to stabilize the plant. The spike train exhibits no particular pattern. The final force applied to the plant is scaled down 10 fold ($\times$0.1) of the actual values for improved visualization. (b) After training, the controller stabilizes the plant within a short amount of time (1000 ms). The spike trains are a bit dense but exhibit some patterns. The final force applied to the plant is scaled down 10 fold ($\times$0.1) of the actual values for improved visualization. (c) In the stable state, the pole oscillates around the set point (0, 0) and the error function E is also at around 0. The trained controller behaves like a bang-bang controller. This results from the patterns of the output spike trains. After one neuron fires three times, the other neuron also fires three times, then the first neuron fires again, and so on. Compared to the start condition, the spike trains are sparse. The final force applied to the plant is scaled down 100 fold ($\times$0.01) of the actual values for improved visualization. It can be seen that the patterns of the spike trains are different in the start condition from those in the stable state. In the start condition, one output neuron generates spikes continuously while the other does intermittently. This is because the controller attempts to repeatedly push the plant in one direction. In the stable state, however, both neurons produce spikes alternately.



(a) Trajectories (b) Zoomed in trajectories (c) Coverage of Model 1 (d) Coverage of PID

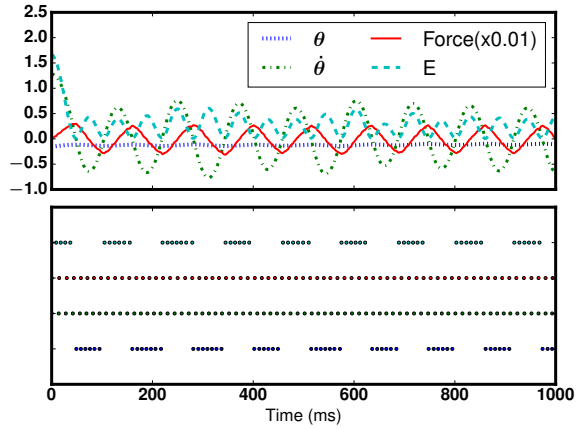
Fig. 3. Trajectories and coverages of Model 1 with two output neurons. (a) Trajectories of the plant state (θ, $\dot{\theta}$) for Model 1 after training and the PID controller over time with different initial settings. The trajectories start at two points (-0.15, 1.4) and (-1.5, 1.5). They are chosen to compare the performance of Model 1 against the PID controller. The blue solid line shows the trajectory of the plant state for Model 1 with the initial plant state (-0.15, 1.4). The green dotted line shows the trajectory of the plant state for the PID controller with the same initial state. While the plant for Model 1 eventually settles down, its trajectory is different from that of the PID controller. This demonstrates the fact that the proposed controller behaves differently suggesting a novel control mechanism. The red dashed line represents the trajectory of the plant state for Model 1 with the initial state (-1.5, 1.5). The cyan dash-dot is the trajectory for the PID controller with the same initial state. While Model 1 succeeds, the PID controller fails. (b) Trajectories in (a) zoomed in around the set point (0, 0). The plant for Model 1 oscillates between (0, -0.02) and (0, 0.02) in the stable condition. (c) Coverage of initial states (θ, $\dot{\theta}$) with Model 1 controller. A green circle indicates a success while a red 'x' mark indicates a failure for the corresponding initial state. (d) Coverage of PID controller. The number of dots covered (green circle) is 36 for Model 1 controller and 32 for the PID controller. The proposed controller covers a larger area than the PID controller.

initial weights since the same weights would yield the same results of the failed run.

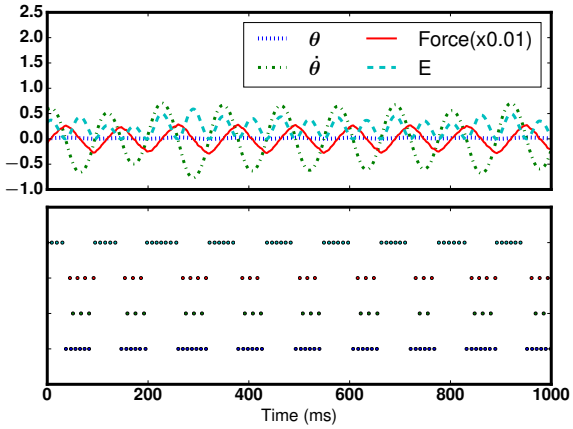
The plant was configured as: half-pole length $l = 0.5$ (m), pole mass $m = 0.1$ (kg), cart mass $M = 1.0$ (kg), and gravity $g = 9.8$ (m/s^2) . The configuration for the controller was: time step = 1ms, threshold = 0.0, $\tau_f = 20$ (ms), $R = -1000$, $\gamma = 1.2$, and $\alpha = 0.01$. The unit of R is the same as that of the membrane potential. The magnitude of R was set large to prevent a spike from being generated within 4-5 msec after a spike was generated by bringing down the membrane potential dramatically. We measured the firing rates of the output neurons in Hz (number of spikes per second). Specifically, the firing rate of a neuron is the total number of spikes generated by the neuron divided by the total running time of the simulation. In all the experiments, the same PID controller was used and its parameters were K_P = 20, $K_I = 0.01$, and $K_D = 1$.

B. Results

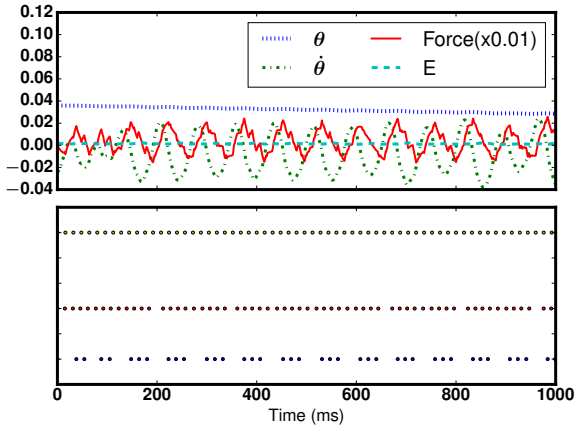
We start by describing a model that successfully learned the control response. Not surprisingly, when $\dot{\theta}$ as a state variable to control was removed, the controller failed to learn regardless of whether the controller was based on Model 1 or Model 2. To elaborate, although the controller was able to hold the pole upright for a short period in the training stage, it failed to hold



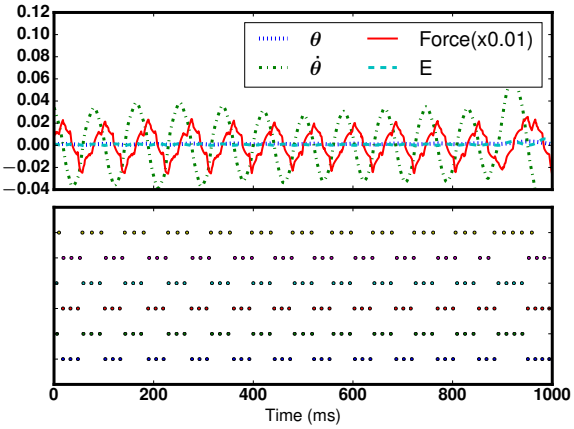
(a) Start condition for 4 output neurons



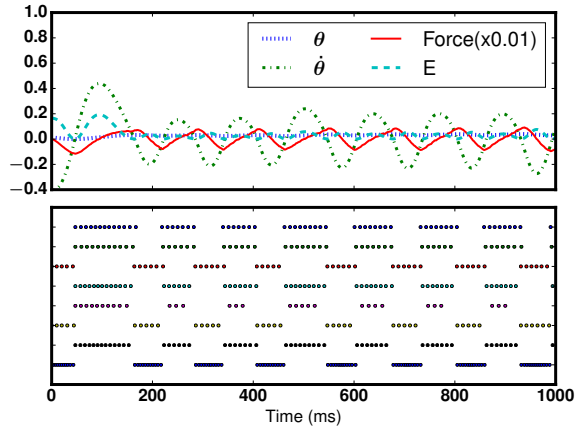
(b) Stable condition for 4 output neurons



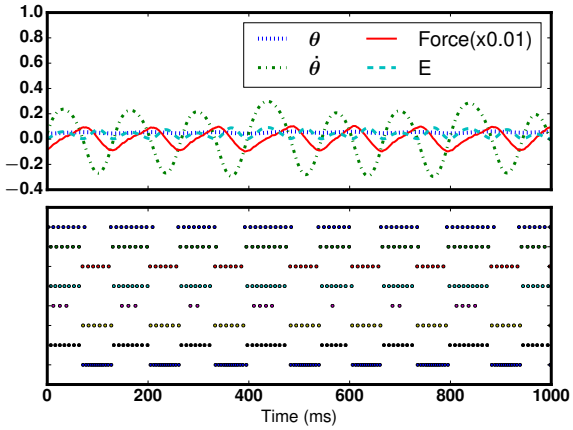
(c) Start condition for 6 output neurons



(d) Stable condition for 6 output neurons

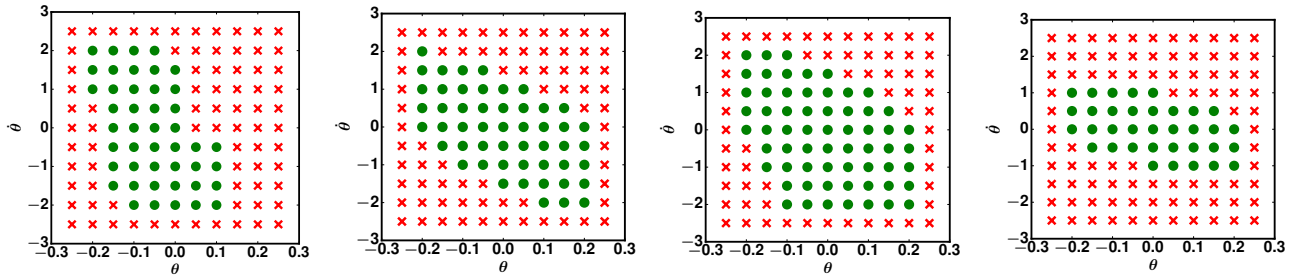


(e) Start condition for 8 output neurons



(f) Stable condition for 8 output neurons

Fig. 4. Snapshots of the plant state (top) and the output spike trains of the controller (bottom) in the start and stable condition for Model 2 with 4, 6, and 8 output neurons after training. In the top figures, the blue dotted line and the green dash-dot represent the vertical angle θ and angular velocity $\dot{\theta}$ of the pole in radian ($^\circ$), respectively. The red solid line is the force applied to the cart and the cyan dashed line is the error function E. Note that the force applied to the plant is scaled down 100 fold ($\times 0.01$) of the actual values for improved visualization. For each spike trains snapshot, the force magnitude assigned to each output neuron is symmetric: big to small and small to big. (a), (b) 4 output neurons. For the spike trains snapshot, the force magnitude assigned to each output neuron is 1000, 500, 500, and 1000 from top to bottom. (c), (d) 6 output neurons. The force magnitude assigned to each output neuron is 300, 200, 100, 100, 200, and 300 from top to bottom. (e), (f) 8 output neurons. The force magnitude assigned to each output neuron is 250, 200, 150, 100, 100, 150, 200, and 250 from top to bottom.



(a) 4 neurons (b) 6 neurons (c) 8 neurons (d) PID

Fig. 5. Stability coverages of the initial states (θ, $\dot{\theta}$) of Model 2 controllers. The PID coverage is reproduced from Fig. 3d for convenient visual comparison. A green circle indicates a success while a red 'x' indicates a failure for the corresponding initial state. The 6 neuron controller covers a larger region than the 4 neuron controller, and the 8 covers a larger region than the 6. This implies that controllers with more neurons are more robust. When compared to the PID controller, the proposed controller covers more in general. Especially 6 and 8 neurons cover the whole region of that of the PID. This indicates the proposed controller has better control capability than the traditional PID controller.

the pole upright for 1 hour. Surprisingly, the controller with either model worked once we added $\dot{\theta}$. In what follows, we discuss the experimental results for each model.

1) Model 1: Fig. 2 shows snapshots of the plant state (top) and the spike trains of the output neurons (bottom) of the proposed controller with 2 output neurons before and after training. The controller received 4 continuous inputs $(\theta, -\theta, \dot{\theta}, -\dot{\theta})$ from the plant and produced output spike trains at the two output neurons. The output spike trains correspond to the left direction force and right direction force, respectively. In the top panels, the blue dotted line and the green dash-dot represent the time evolution of the vertical angle θ and angular velocity $\dot{\theta}$ of the pole in radian (°) and rad/sec, respectively. The red solid line is the force applied to the cart and the cyan dashed line is the error E. The force magnitude assigned to each output neuron was 100. The average firing rates of the output neurons were 33.94Hz and 34Hz. Fig. 2a shows a snapshot before training where the synaptic weights were randomly chosen and not updated. As shown in the top panel, the pole fell down shortly after the start of the simulation as the controller was untuned. This is further obvious from the bottom panel where the spike trains were randomly generated regardless of the state of the plant. After training, the pole successfully stood upright for 1 hour. The trained controller was tested on random initial conditions. Fig. 2b shows the start condition in the testing phase to demonstrate how the proposed controller behaved in controlling the plant initially. Fig. 2c shows the stable state after a while. From the figures, it can be seen that the patterns of the spike trains are different in the start condition versus the stable state. In the start condition, one output neuron generates spikes continuously while the other does intermittently. This is because the controller attempts to repeatedly push the plant in one direction. In the stable state, however, both neurons produce spikes alternately. Fig. 3 shows the trajectories of the plant state (θ, $\dot{\theta}$) over time with two different initial settings. Further testing was performed over several initial states to determine the robustness and the coverage over initial states for the proposed controller with Model 1 as compared to the PID controller. In Fig. 3a, we can observe that the trajectory of the proposed controller is different from that of the PID controller. This implies that our controller behaves in a different manner in order to control the plant suggesting a novel control mechanism. Fig. 3b shows the trajectories in Fig. 3a zoomed in around the set point (0, 0) for better visualization. Fig. 3c and Fig. 3d show the coverages of stability for Model 1 and the PID controller, respectively. Although they do not exclusively include each other, Model 1's coverage is larger than the PID. For example, the initial state (-0.2, 1.5) covered by Model 1 is not covered by the PID controller.

2) Model 2: We performed the same learning experiments above for 4, 6, and 8 output neurons with successively larger force kernels. In this case, the force magnitude assigned to each output neuron was symmetric: pairs of equal magnitude for the left and right force. For 4 output neurons, the force magnitude assigned to each output neuron was 1000, 500, 500, and 1000. For 6 output neurons, it was 300, 200, 100, 100, 200, and 300. For 8 output neurons, it was 250, 200, 150, 100, 100, 150, 200, and 250. Fig. 4 shows snapshots of the plant and the controller. It shows the start and stable conditions after training. As shown in the figure, Model 2 controllers achieved the objective of stabilizing the plant. As in the case of Model 1, the spike trains exhibit regular patterns; neurons fired alternately periodically. In general, the spike train in the stable state was sparser than that in the start condition. It can be observed that there are unnecessary spikes in the stable state. To elaborate, since we have neurons generating large as well as small forces, we need only the small force neurons to fire in the stable state. This can be mitigated by adding communication between output neurons so that they are aware of one another's spike trains. Fig. 5 shows the coverages of initial conditions that are controlled. As shown in the figures, the stability coverage increases with the number of neurons in the controller. 4 has a larger region than 2, 6 has a larger region than 4, etc. Fig. 5d is reproduced from Fig. 3d to ease visual comparison with Model 2's coverage. It is clear that Model 2 covers a much wider area than the PID. It should be noted that the coverage of 6 or 8 neurons subsumes the entire

PID area. This suggests that the proposed controller is more robust than the traditional PID controller.

VI. Related Work

Neural networks have been the tool of choice for solving various problems since Rumelhart et al. introduced the gradient descent based error backpropagation algorithm [5]. Similar algorithms have also been used for spiking neural networks [6], [7]. [8] introduced *SpikeProp* to solve the XOR problem by applying the error backpropagation algorithm to spiking neuron networks based on temporal coding or spike timing based coding [9]. Their supervised learning rule generates a desired pattern of spikes with the constraint that each output neuron be allowed to fire only once in a prescribed time window. [10] extended this rule to multiple output spikes, albeit with the first output spike used in the error function. [11] presented a spiking neural network based controller to regulate a robot's arms with 4-degrees of freedom. The neuron model they used is the Izhikevich model [12] and the learning algorithm is Spike Timing-Dependent Plasticity (STDP) [4]. Their controller shows high firing rates due to rate-based coding.

Recently, Gerstner et al. [13], [14] studied control problems for motor systems using reinforcement learning. They used the actor-critic model [3], [15] to train the networks. [16] studied the perturbation analysis [17] to reveal how perturbations in the weights and times of the input spikes of a neuron translate to perturbations in the timing of its output spikes. Although our proposed controller can be described to be most similar to [16] and [8], unlike these, ours does not require the prescription of the desired spike train and the input to the network is continuous.

VII. Conclusion

We have proposed a spiking neuron network controller and have applied it to the classical cart-pole control problem to demonstrate its efficacy. The derivation presented is general and can be applied to any feedforward network. The primary advantage of our controller is that it has a larger region of stability as compared to the traditional PID controller. Furthermore, our controller behaves in a manner different from the traditional PID controller. As demonstrated in our experiments, the proposed controller succeeds in several initial conditions where the PID controller fails. The proposed controller produced different trajectories than that of the PID controller. We presented two controller models with different output neuron settings: two output neurons with the same force magnitude (Model 1) and 4 or more neurons with different force magnitude kernels (Model 2). From the experiments, we observe that more neurons with diverse force magnitudes can learn larger ranges and are thus more flexible and robust. In particular, the 6 or 8 output neuron controller performs substantially better than the PID controller. In future work, we plan to add a kernel for filtering the inputs and shall consider other control costs. The former can readily be added to the current controller keeping the derivation the same. The latter requires using a more general error function that included the number of control spikes and other output statistics. One of the issues in Model 2 is that some spikes are produced redundantly. We can mitigate this by adding recurrent inhibitory connections among the output neurons. This will lead to sparse spike trains which is more natural in biological systems as they signify higher energy efficiency. Finally, we plan to extend our controller to apply to the locomotion of a fish with 3 or more degrees of freedom which is a more realistic and complex control problem than the cart-pole.

Acknowledgment

The authors would like to thank the Air Force Office of Scientific Research (Grant FA9550-16-1-0135) for their generous support of this research.

References

[1] N. Minorsky, "Directional stability of automatically steered bodies," *Journal of the American Society for Naval Engineers*, vol. 34, no. 2, pp. 280–309, 1922.

[2] W. Gerstner and W. M. Kistler, *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university press, 2002.

[3] C. W. Anderson, "Strategy learning with multilayer connectionist representations," in *Proceedings of the Fourth International Workshop on Machine Learning*, 1987, pp. 103–114.

[4] S. Song, K. D. Miller, and L. F. Abbott, "Competitive hebbian learning through spike-timing-dependent synaptic plasticity," *Nature neuroscience*, vol. 3, no. 9, pp. 919–926, 2000.

[5] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Cognitive modeling*, vol. 5, p. 3, 1988.

[6] F. Rieke, *Spikes: exploring the neural code*. MIT press, 1999.

[7] P. Dayan and L. F. Abbott, *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. The MIT Press, 2005.

[8] S. M. Bohte, J. N. Kok, and H. La Poutre, "Error-backpropagation in temporally encoded networks of spiking neurons," *Neurocomputing*, vol. 48, no. 1, pp. 17–37, 2002.

[9] R. V. Florian, "The chronotron: a neuron that learns to fire temporally precise spike patterns," *PloS one*, vol. 7, no. 8, p. e40233, 2012.

[10] O. Booij and H. tat Nguyen, "A gradient descent rule for spiking neurons emitting multiple spikes," *INFORMATION PROCESSING LETTERS*, vol. 95, no. 6, pp. 552–558, 2005.

[11] A. Bouganis and M. Shanahan, "Training a spiking neural network to control a 4-dof robotic arm based on spike timing-dependent plasticity," in *Neural Networks (IJCNN), The 2010 International Joint Conference on*. IEEE, 2010, pp. 1–8.

[12] E. M. Izhikevich *et al.*, "Simple model of spiking neurons," *IEEE Transactions on neural networks*, vol. 14, no. 6, pp. 1569–1572, 2003.

[13] N. Frémaux, H. Sprekeler, and W. Gerstner, "Reinforcement learning using a continuous time actor-critic framework with spiking neurons," *PLoS computational biology*, vol. 9, no. 4, p. e1003024, 2013.

[14] G. Hennequin, T. P. Vogels, and W. Gerstner, "Optimal control of transient dynamics in balanced networks supports generation of complex movements," *Neuron*, vol. 82, no. 6, pp. 1394–1406, 2014.

[15] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. Sep, no. 5, pp. 834–846, 1983.

[16] A. Banerjee, "Learning precise spike train-to-spike train transformations in multilayer feedforward neuronal networks," *Neural Comput.*, vol. 28, no. 5, pp. 826–848, May 2016. [Online]. Available: http://dx.doi.org/10.1162/NECO_a_00829

[17] ——, "On the phase-space dynamics of systems of spiking neurons. i: Model and experiments," *Neural Computation*, vol. 13, no. 1, pp. 161–193, 2001.