PLOS Computational Biology

RESEARCH ARTICLE

Spiking neurons as predictive controllers of linear systems

Paolo Agliati[1*], **André Urbano**[2], **Pablo Lanillos**[2], **Nasir Ahmad**[1], **Marcel van Gerven**[1], **Sander Keemink**[1]

1 Department of Machine Learning and Neural Computing, Donders Institute for Brain, Cognition and Behaviour, Radboud University, Nijmegen, The Netherlands, **2** Neuro AI and Robotics group, Cajal International Neuroscience Center, Spanish National Research Council, Madrid, Spain

* paolo.agliati@donders.ru.nl

Citation: Agliati P, Urbano A, Lanillos P, Ahmad N, van Gerven M, Keemink S (2026) Spiking neurons as predictive controllers of linear systems. PLoS Comput Biol 22(7): e1014432. https://doi.org/10.1371/journal.pcbi.1014432

Editor: Gunnar Blohm, Queen's University, CANADA

Received: October 20, 2025

Accepted: June 11, 2026

Published: July 9, 2026

Peer Review History: PLOS recognizes the benefits of transparency in the peer review process; therefore, we enable the publication of all of the content of peer review and author responses alongside final, published articles. The editorial history of this article is available here: https://doi.org/10.1371/journal.pcbi.1014432

Abstract

Neurons communicate with downstream systems via sparse and incredibly brief electrical pulses, or spikes. Using these events, they control various targets such as neuromuscular units, neurosecretory systems, and other neurons in connected circuits. This gave rise to the idea of spiking neurons as controllers, in which spikes are the control signal. Using instantaneous events directly as the control inputs, also called 'impulsive control', is challenging as it does not scale well to larger networks and has low analytical tractability. Therefore, current spiking control usually relies on filtering the spike signal to approximate analog control. This ultimately means spiking neural networks (SNNs) have to output a continuous control signal, necessitating continuous energy input into downstream systems. Here, we circumvent the need for rate-based representations, providing a scalable method for task-specific spiking control with sparse neural activity. In doing so, we take inspiration from both control theory and neuroscience, and define a spiking rule where spikes are only emitted if they bring a dynamical system closer to a target. From this principle, we derive the required connectivity for an SNN, and show that it can successfully control linear systems. We show that for physically constrained systems, predictive control is required, and the control signal ends up exploiting the passive dynamics of the downstream system to reach a target. Finally, we show that the paradigm scales to both high-dimensional systems and bio-inspired motor control tasks. Importantly, in all cases, we maintain a closed-form mathematical derivation of the network connectivity, the network dynamics and the control objective. This work advances the understanding of SNNs as biologically-inspired controllers, providing insight into how real neurons could exert control, and enabling applications in neuromorphic hardware design.

Data availability statement: There are no primary data in the paper; All relevant data are within the manuscript and its Supporting Information files. All simulations were run with Python 3.11.5. The source code and all the necessary dependencies are available at https://gitlab.socsci.ru.nl/2023-phd-paolo-agliati/spiking-neurons-as-predictive-controllers-of-linear-systems.

Funding: This publication is part of the project Dutch Brain Interface Initiative (DBI2) (024005022 to SK, MvG, PA) of the research programme Gravitation, which is financed by the Dutch Ministry of Education, Culture and Science (OCW) via the Dutch Research Council (NWO). The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Competing interests: The authors have declared that no competing interests exist.

Author summary

Action potentials, or spikes, are the key events that allow neurons in the vertebrate brain to govern the movement of the body, its internal state, and its interactions with the world. Control theory provides a formal framework for regulating processes toward desired states, but is traditionally separated from the study of neural networks. This work provides a rigorous account of how spiking neural network implementations can be formulated within the language of control theory. From the neural perspective, our work develops and expands the existing view of SNNs as biologically-inspired controllers that are modulating their own internal dynamics. We generalize this principle to explicitly control any downstream linear system using spikes. From the control theory perspective, we circumvent the problem of applying instantaneous events directly as control inputs, which has been analytically challenging and difficult to scale. We maintain a closed-form mathematical derivation of the network connectivity, the network dynamics and the control objective, thus obtaining a model that can be valuable for both hypothesis testing in neuroscience, and applications in neuromorphic systems.

Introduction

In the vertebrate nervous system, action potentials are one of the universal principles of communication between neurons [1–6]. Spiking behavior of cortical neurons grants efficient and robust computational properties, providing a basis for the realization of the complex adaptive functions of the cortex [1–4,7,8]. At the level of single neurons, spike events are incredibly brief and sparse, and through synaptic communication can affect the dynamics of neurons in downstream areas. This property, where neurons are not only communicating information but modifying behavior, settled the basis for viewing neurons as controllers [9,10] and spikes as control signals. We can find clear examples of spike-driven control in movement control studies, where spike generation allows animals to orient their head in space [11,12], as well as control the movement of their eyes [12,13] or limbs [12,14]. This control perspective can offer important insights into neural computation, but is also increasingly relevant for neuromorphic applications where spiking networks are used to control downstream plants [15–18].

Although the theory and application of control with spiking networks have recently seen some advances, it remains unclear how to generate each spike to successfully control a plant's dynamics. Traditionally, control theory relies on optimal control methods, such as the linear quadratic regulator (LQR) [19], which continuously adjusts control inputs to drive a system toward a target state while minimizing a cost function (Fig 1B). Although effective, these continuous control paradigms are difficult to reconcile with the discrete nature of SNNs. Attempts at bridging this gap include the use of a filtered spikes approach. These studies have applied classic optimal control principles to SNNs by approximating LQR signals through rate codes or filtered

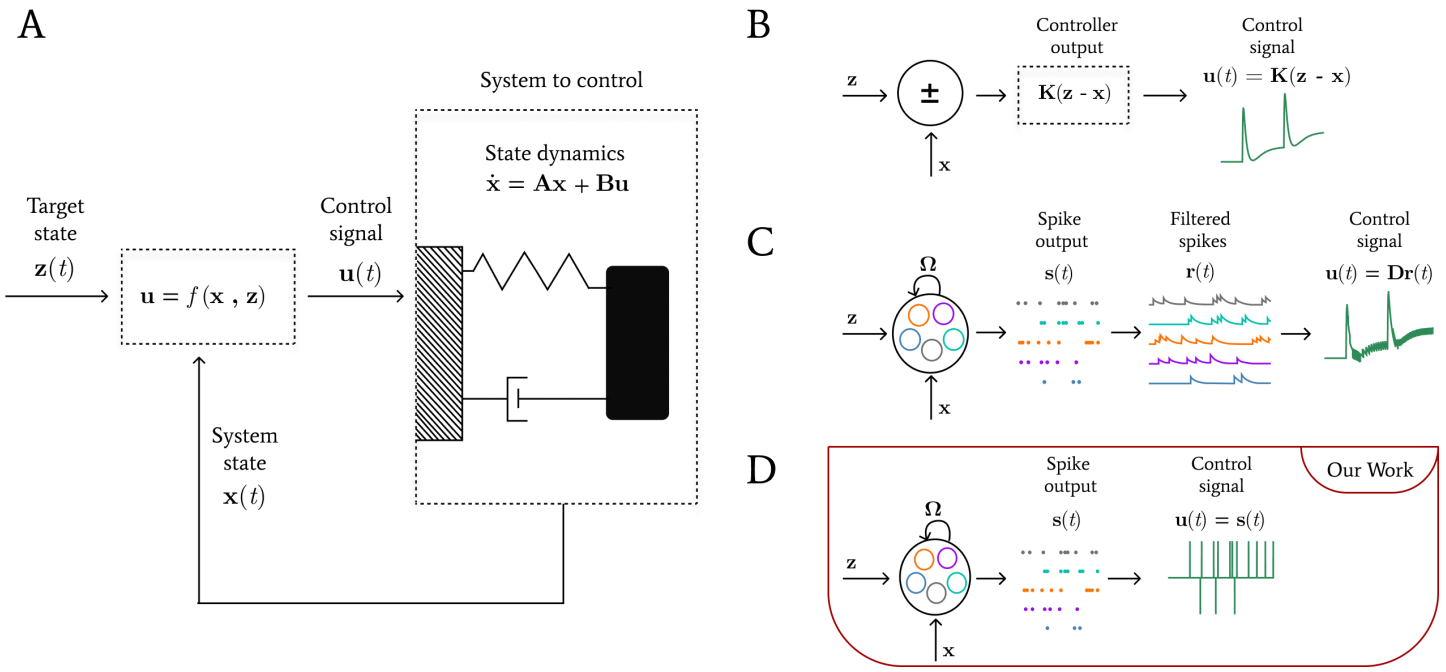

Fig 1. Overview of feedback control approaches. A: Scheme of a feedback controller and a downstream system. The current state of the system $\mathbf{x}(t)$ is given as input to the controller, along with a target value $\mathbf{z}(t)$. The controller outputs a control signal $\mathbf{u}(t)$, as a function of both inputs. $\mathbf{u}(t)$ is then mapped onto the dynamics of the downstream system via the **B** matrix. The system's dynamics are governed by the **A** matrix. B-D: Examples of three feedback control paradigms. B: The continuous LQR uses a gain matrix **K** to directly compute the control signal for a given state and target pair. C: In the filtered spiking case, the output spikes of the network are filtered to obtain $\mathbf{r}(t)$, which is decoded by the matrix **D** to form the control signal $\mathbf{u}(t)$ (adapted from [24]). D: In our spiking control paradigm, the output spikes $\mathbf{s}(t)$ represent the control input itself, and are therefore directly mapped onto the system dynamics through the **B** matrix.

https://doi.org/10.1371/journal.pcbi.1014432.g001

spike trains [20–24], as shown in Fig 1C. Importantly, in these filtered spikes approaches, spikes serve as the network's representation of the system state but are not actively computing control signals, which are still continuous in nature. This makes it difficult to trace the effect each single spike has on the trajectory of the controlled system. Alternative approaches for spike-based control in the literature focus on individual neuron-based controllers [25], or implement forms of instantaneous control [26]. Such impulsive control methods have been widely studied, but they mostly focus on the control theory perspective, sometimes even bypassing the need of a network [27–30]. Only recently, some works started integrating this framework with theoretical neuroscience concepts, to explicitly interpret the control impulses as the neurons' action potentials [31]. Still, network architectures are imposed rather than derived from control objectives. For this reason, the integration of bio-informed SNNs with (feedback) control remains an under-explored area. A new paradigm is needed to account for direct spiking control and its seamless integration with computational neuroscience theories.

Our study investigates how spike events can influence the dynamics of linear systems, using spikes directly as control signals. We consider this problem from the perspective of control theory, by formally deriving how neurons should spike to control a linear dynamical system (Fig 1). This equates to generating control inputs that can successfully drive the dynamics of a downstream plant to minimize a certain loss associated with the state of the system. When deriving a solution for this problem (see Materials and Methods), we obtain, without needing to introduce further assumptions, a feedback control scenario like the one depicted in Fig 1A. Our work embeds this instantaneous spiking control within theoretical neuroscience constraints.

Inspired by the neuroscience theory of spike coding networks (SCNs) [5,20,21,23], we derive, from control theory principles, a network of recurrent integrate-and-fire (IF) neurons with sparse activity and low-rank connectivity. By directly incorporating spikes as control signals (Fig 1D), our model ensures that each neuron's contribution to the system's trajectory is explicit. Between each spike event $s(t)$, the system evolves following its dynamics in an open loop, and when a control signal is produced, the system switches to a closed-loop control (with $s(t)$ being the control input). Our solution naturally exploits the intrinsic dynamics of the plant, instead of continuously inputting a signal over time. This approach could help understanding how neural circuits can execute control tasks efficiently while preserving key features of biological networks. In the following sections, we present our framework, analyze the model's performance on a linear dynamical system, and discuss its implications for both neuroscience and control theory.

Results

Formulating the control problem

To investigate the role of spiking activity in controlling linear dynamical systems, we start by revisiting the fundamental components of a control task. We then introduce our approach to modeling spike-based control, and finally explore the consequences of applying this method to physically constrained systems.

General control setup

The control of a generic K-dimensional linear dynamical system through an N-dimensional control signal, as illustrated in Fig 1A, can be expressed as:

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}\,, \tag{1}$$

where $\mathbf{x} \in \mathbb{R}^K$ is the system state, $\mathbf{A} \in \mathbb{R}^{K\times K}$ is the system matrix, describing the system's uncontrolled dynamics, $\mathbf{B} \in \mathbb{R}^{K\times N}$ is the input matrix which maps the control input onto the system, and $\mathbf{u} \in \mathbb{R}^N$ is the control input.

In classical control, $\mathbf{u}$ is typically a continuous signal, as in LQR control, where the optimal control input is computed to minimize a quadratic cost function (Fig 1B). For control systems that make use of neural networks, the control signal can instead be derived from neural activity. In spiking networks, a filtered-spike signal can be employed, where the firing rates of neurons determine the value of the control input for each timestep (Fig 1C). Here, the spikes are aiding the network's internal representation of the systems, but the control input still approximates a continuous control signal [20–25,32]. In our work, we instead consider a sparse and discrete control signal, where the input is made of brief pulses or 'kicks' (Fig 1D) [26–31]. The system then becomes:

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{s}\,, \tag{2}$$

where $\mathbf{s} \in \mathbb{R}^N$ describes the on/off state of each of N neurons in the network. A spike signal is emitted every time the voltage V_i of neuron i reaches its threshold T_i, resulting in the spike train $s_i(t) = \sum_j \delta(t - t_j)$ with t_j indexing the spike times. Importantly, here δ is the Dirac delta function, which describes an instantaneous pulse (whose area integrates to one) at spike times t_j and is zero otherwise. This is different from other forms of impulsive control like bang-bang control or chattering control [33], where the instantaneous input is maintained for a certain period of time.

Spike-based control

To formalize our control approach, we define a loss function that expresses the discrepancy between the current state of the linear dynamical system and a desired target state, by taking the squared L^2 norm of their difference. Specifically, we consider the loss:

$$L = \|\mathbf{x}(t+f) - \mathbf{z}(t+f)\|^2 + \mu\Gamma(\mathbf{s}) + \alpha\|\mathbf{r}\|^2 \tag{3}$$

where $\mathbf{x}(t+f)$ is the system state some time f in the future, $\mathbf{z}(t+f) \in \mathbb{R}^K$ the target state, μ is a regularization parameter that determines the spiking cost, $\Gamma(\mathbf{s}) = \int_t^{t+\delta t} \mathbf{s}(t)\mathrm{d}t$ counts the spikes within a short time window (in practice within a simulation timestep), $\mathbf{s}$ are the spike trains, α is a parameter that scales the cost for the neuron's activity, and $\mathbf{r}$ are the neurons' filtered spike trains. We consider both reactive control ($f=0$) and predictive control ($f>0$).

Classically, such a loss might be minimized by taking the gradient with respect to certain system parameters, or by finding the optimal control input that minimizes this loss for some future time window, as in standard LQR control. However, the use of a spiking control signal makes both of these approaches challenging to solve mathematically. Here, we instead take inspiration from the neuroscientific theory of spike coding networks. SCN theory assumes spikes should only happen when they improve on an underlying coding loss and derives the required spiking dynamics. From this principle, spiking neural networks that represent their inputs [20,21] and perform a variety of computations [23,24,34] can be defined.

Following this same principle, but applied to the problem of controlling dynamical systems, we require that a spike should only happen if it improves the control loss (Eq. (3)). According to these SCNs-inspired approaches, whenever a neuron in the network spikes, it (directly or indirectly) influences the state of the plant, resulting in a different loss value. Therefore, we can compute two different losses L_i^{s} and L^{ns}, representing the distance to the target in the presence or absence, respectively, of a spike from neuron i. In order to control the plant's dynamics, the neurons in the network compare the two losses to determine whether inputting a spike is advantageous to reach the control target. The inequality $L_i^{\mathrm{s}} < L^{\mathrm{ns}}$ is describing how our loss can be improved by the control input, decreasing the distance towards the target and helping our network carry out the control task. Importantly, as described in the Discussion, the network will not compute nor approximate the optimal control solution for the problem. Differently from other methods (like the filtered spike controller illustrated in Materials and Methods), our spiking control algorithm does not track the LQR solution, but instead approaches the target by locally minimizing losses. These losses are describing a spiking condition, reminiscent of a voltage-threshold inequality $\mathbf{V} > \mathbf{T}$ (where $\mathbf{V}$ is the voltage of the neurons, and $\mathbf{T}$ is their firing threshold) that allows biological neural networks to produce action potentials. In Materials and Methods, we show that one can indeed derive voltage and thresholds values from our loss comparison. As a consequence, in our framework, the values of both voltage and threshold are composed of quantities that describe the (current and future) states of the plant. In particular, the voltage of neuron i is expressed by:

$$V_i = \mathbf{b}_i^T \mathbf{A}_f^T (\mathbf{z} - \mathbf{A}_f \mathbf{x}_0)\,, \tag{4}$$

where $\mathbf{b}_i$ is the ith column of the matrix $\mathbf{B}$, $\mathbf{A}_f \equiv e^{\mathbf{A}f} \in \mathbb{R}^{K\times K}$ is the matrix exponential of the state matrix $\mathbf{A}$ times the future time window f, and $\mathbf{x}_0 = \mathbf{x}(t)$ is the current state of the system. The threshold of neuron i is described as:

$$T_i = \frac{\mathbf{b}_i^T \mathbf{A}_f^T (\mathbf{A}_f \mathbf{b}_i) + \mu + \alpha(2\mathbf{r}_i + 1)}{2}, \tag{5}$$

where $\mathbf{r}_i$ indicates a vector containing only zeros except for the ith position, which holds the filtered spike traces of neuron i. A detailed derivation of the values of $\mathbf{V}$ and $\mathbf{T}$ can be found in Materials and Methods. Although a cost $\mathbf{C}$ on each dimensions of the system can be introduced to further modulate spiking activity, the values of V_i and T_i in Eq. (4) and Eq. (5) do not consider this weighted case (assuming $\mathbf{C}$ is an identity matrix), for simplicity. Further developing this simple spiking rule leads to a recurrent network of integrate-and-fire neurons, illustrated in Fig 2, and described by:

$$\dot{\mathbf{V}} = -\mathbf{V} + \mathbf{G}(\dot{\mathbf{z}} + \mathbf{z}) - \mathbf{F}\mathbf{x}_0 - \Omega\mathbf{s}\,, \tag{6}$$

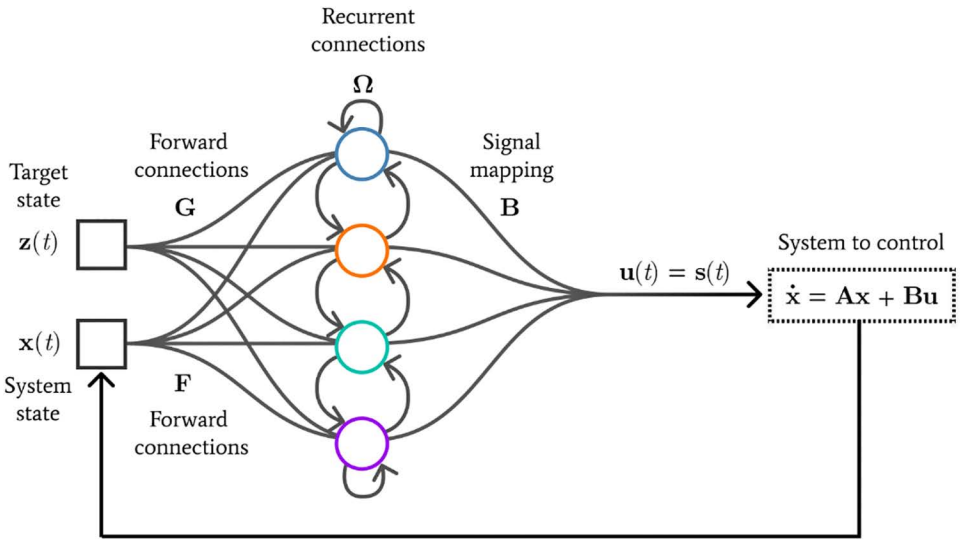

Fig 2. Illustration of the network-plant loop. Scheme of a recurrent spiking network composed of four neurons controlling a downstream system. Inputs and target are mapped onto the network via forward connections. **G** maps the target **z**, while **F** maps the system's current state, **x**. The matrix Ω represents the recurrent weights. The control inputs are the spikes **s**, which are mapped onto the system via the matrix **B**.

https://doi.org/10.1371/journal.pcbi.1014432.g002

where $\mathbf{V} \in \mathbb{R}^N$ represents the membrane potentials, $\mathbf{G} \in \mathbb{R}^{N\times K}$ are the forward connections that encode the target **z**, $\mathbf{F} \in \mathbb{R}^{N\times K}$ contains the forward connections that encode the current system's state $\mathbf{x}_0$, $\Omega \in \mathbb{R}^{N\times N}$ represents recurrent connections, and **s** are the spikes. Interestingly, the obtained recurrent network has low-rank connectivity with respect to the dimensions of the downstream system (see Materials and Methods). This network structure is not assumed, but rather derived from each neuron as a greedy spiking condition given our loss comparison (see Discussion). Consequently, although the network obtained is recurrent, each neuron computes the losses without access to network-level information. Under this light, our work can be seen as an extension of the same principle in [20], applied to controlling downstream systems. For a full derivation of the network, see Materials and Methods.

For a first proof-of-concept, we enforce an asynchronous firing scheme: at each timestep, all neurons independently compute their local loss comparisons, but only one is allowed to spike, even if several exceed threshold. The spiking neuron can be chosen randomly from the suprathreshold set, or, as we do here, selected as the one with the highest voltage above threshold. While for clarity we keep this firing policy in all the shown examples, it is not strictly required for achieving control. This is because networks that operate on local spiking rules can distribute their activity very effectively among neurons [20,35], which means no specific neuron is more relevant for the computation of the network. Such a simple asynchronous policy is only feasible in zero-delay systems, where the new state of the plant is inputted in the network without any time delay. In a system with delays, we could not impose this asynchronous firing. However, we can employ different methods that allow multiple neurons to spike while still being compatible with our local spiking rule, for instance, by scaling each neuron's threshold according to its past firing activity (see Materials and Methods). Examples of (failing and successful) spiking control of linear systems without the asynchronous firing constraint are shown in Fig A in S1 Appendix.

Reactive vs. predictive spiking control

Our control problem is centered around the comparison of losses (distances of the state to a target) under the conditions in which a spike is generated or not (see Results). By comparing the loss functions in the spiking and non-spiking cases,

we derived a firing condition for each neuron. Namely, the neuron's voltage exceeds its threshold and generates a spike only when the effect of this spike onto the system is to minimize the loss (reducing the distance between the system's state and the target). Importantly, in our control paradigm, the spiking rule is fully local: each neuron computes its losses, with no access to shared network-level information, and fires only when the inequality $L_i^s < L^{ns}$ is satisfied, allowing the dynamics of the plant to unfold independently.

In Results, we showed how starting with a spiking condition based on a loss comparison (spike if $L_i^s < L^{ns}$), one can derive the required network structure and dynamics to control a downstream system. Such a spiking rule can be defined relative to the immediate result of a spike, or the predicted effect that such spike will have on the system. In the first case, spikes are generated based on minimizing the loss function at the current time step. In the latter, spikes are generated based on a forward prediction of the system state, incorporating future expected loss minimization over a short horizon of f seconds. The idea of predictive spiking is also present in SCN-related works, usually considering the spike's influence on a future state [36], or on the difference between past and current state [37]. Adapting this predictive spiking idea in our work, means that the system would evolve in open loop after the spike, only following its intrinsic dynamics, until another spike is able to further improve the predicted loss.

Reactive and predictive spiking control could lead to significantly different spiking behavior. To illustrate this, we consider both cases for an example linear system, with its dynamics defined by velocity and position (Fig 3). Each spike produced by a neurons in the controlling network is multiplied by its corresponding column in the matrix **B**, which maps it onto the dynamics of the system. In Fig 3 we choose a **B** matrix that maps the spikes from four different neurons as vectors in the four different cardinal directions. When the system receives a spike control signal in the current timestep, the system state is instantaneously kicked in the corresponding direction. In the reactive case, only the immediate effect of the spiking is considered. This means the loss comparison that produces a spike control signal does not consider the future

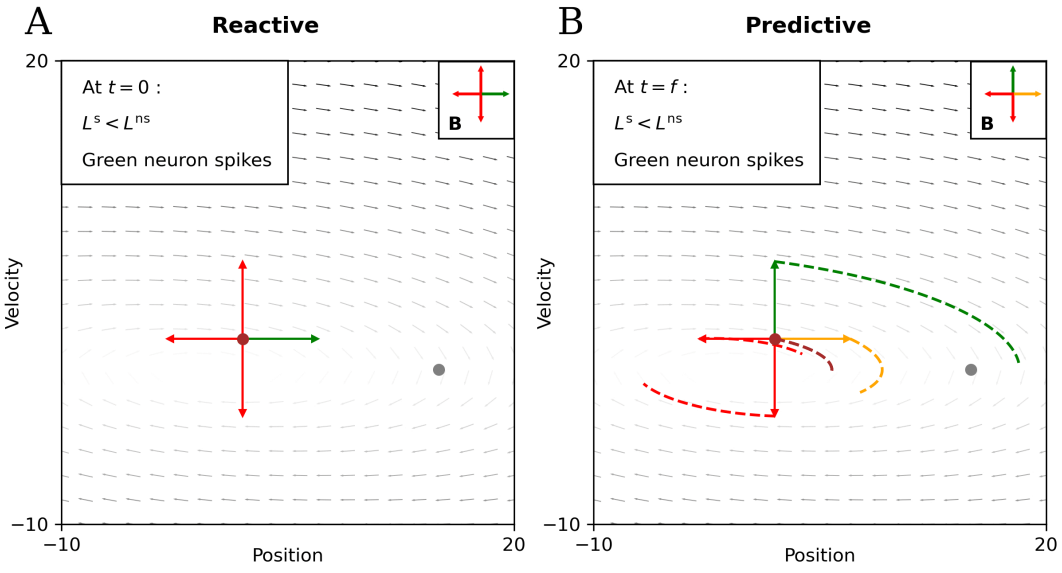

Fig 3. Examples of reactive and predictive spiking. A: Example of a linear dynamical system with its current state (brown dot), and four example states that can be reached with a spike. The gray dot represents a target state. We employ a reactive spiking rule. The neuron for which $L_i^s < L^{ns}$ spikes (green arrow), since that immediately shortens the distance between the system's state and the target. The other ones do not (red arrows). B: Example of a linear dynamical system, where the spike decision is taken based on the state of the system f seconds after the spike. In this predictive case, $L_i^s < L^{ns}$ is still the spiking condition, but the preferred spiking direction can differ. The condition $L_i^s < L^{ns}$ holds for both the upward (green) and rightward (orange) spikes, since the no-spike trajectory (brown) lies further from the target (gray dot) than either of the spiking ones. Here, an asynchronous firing rule (see Results) would select only one of these neurons to fire; either at random, or by considering the neuron with the highest voltage (green).

https://doi.org/10.1371/journal.pcbi.1014432.g003

state of the plant, but rather aims to improve the loss right after the spike is emitted. As shown by Fig 3A, in this arbitrary linear system, adopting a reactive spiking rule results in a spike that kicks the system directly towards the target (green arrow), without accounting for the future dynamics of the plant (which could later move the trajectory further away from it). All other spiking directions (red arrows) do not result in an improved loss right after the spike, and so the neurons whose spikes are mapped in those directions do not fire. On the contrary, predictive spiking control favors the spike that minimizes the loss f seconds into the future, considering how the system would evolve during this time window. In our predictive example (Fig 3B), either spiking rightward (orange trajectory) or upward (green trajectory) would give a lower loss in f seconds when compared to the non-spiking case (brown trajectory). If we employ an asynchronous firing policy, only one of these two neurons is selected for firing. For these examples, we employ asynchronous firing and we select the neuron with the highest voltage past its threshold (the one whose spike is mapped on the green arrow). Note that our paradigm can work outside of this simple asynchronous firing case (Fig A in S1 Appendix), by setting a spike-threshold adaptation mechanism (see Materials and Methods).

Application to physical systems: spring-mass-damper

To illustrate an application of our framework, we consider, as an example, the control of a spring-mass-damper system (SMD) [38] (Fig 1). This is a linear dynamical system composed of a mass attached to a spring (which causes oscillations whenever the mass is moved) and a dampener (which dampens the oscillations, giving the system a fixed stable state). The system can be entirely described within its two variables (the position and the velocity of the mass) which are both included in our system state, $\mathbf{x}$. This makes the SMD a very simple system that grants observable dynamics and clear analytical solutions, while also modeling a physical system with realistic state variables. As such, it is a standard benchmark in control literature and spiking control theories [24,39,40]. Furthermore, the SMD as a linear system can be expanded to approximate more complex plants, as shown by its applications in muscle models for motor control and neuro-mechanical studies [41,42]. In our proof-of-concept, we make use of a simple target $\mathbf{z}$, which evolves during the simulation, incrementing twice and then stabilizing. In all of our simulations, the target state is always relative to the position of the mass, while its velocity can range freely. Although this is the setup we will work with, our algorithm allows to constrain the values of either the position or the velocity of the mass, by changing the entries in the cost matrix $\mathbf{C}$, as shown in Materials and Methods. We will now illustrate the differences that emerge when controlling the system using a reactive versus a predictive paradigm, as described in Results (Fig 3). We consider both physically unconstrained control (where the control signal can affect both velocity and position), and physically constrained control (where the control signal can only affect the velocity).

Reactive control

We first consider a physically unconstrained system with reactive control (Fig 4A). In this case, the system can be successfully made to track a target (Fig 4A top right). This is achieved by continuously spiking directly towards the target, which is always the better choice since the neurons do not consider future states of the plant. In this example, the control is exerted by progressively kicking the mass in the same position as the evolving target, essentially ‘teleporting’ it without affecting its velocity. Of course, this is only possible because we are considering a physically unconstrained system, where the mass is allowed to be moved instantly in any direction of the state space. Also note that because of this property, the system ends up in a physically impossible state with a non-zero velocity (green curve in upper right plot), despite remaining in place according to the position variable. We therefore ask if a reactive control method could work on physically constrained systems. We employ reactive control to our physically constrained example, running the simulation of an SMD. In this context, we directly influence the velocity component of the state $\mathbf{x}$, while the position component evolves as a consequence. This reflects a key constraint of physical systems: the position cannot be instantaneously changed, as the mass cannot move from one position to another in no time. As shown in Fig 4B, a reactive approach fails to control

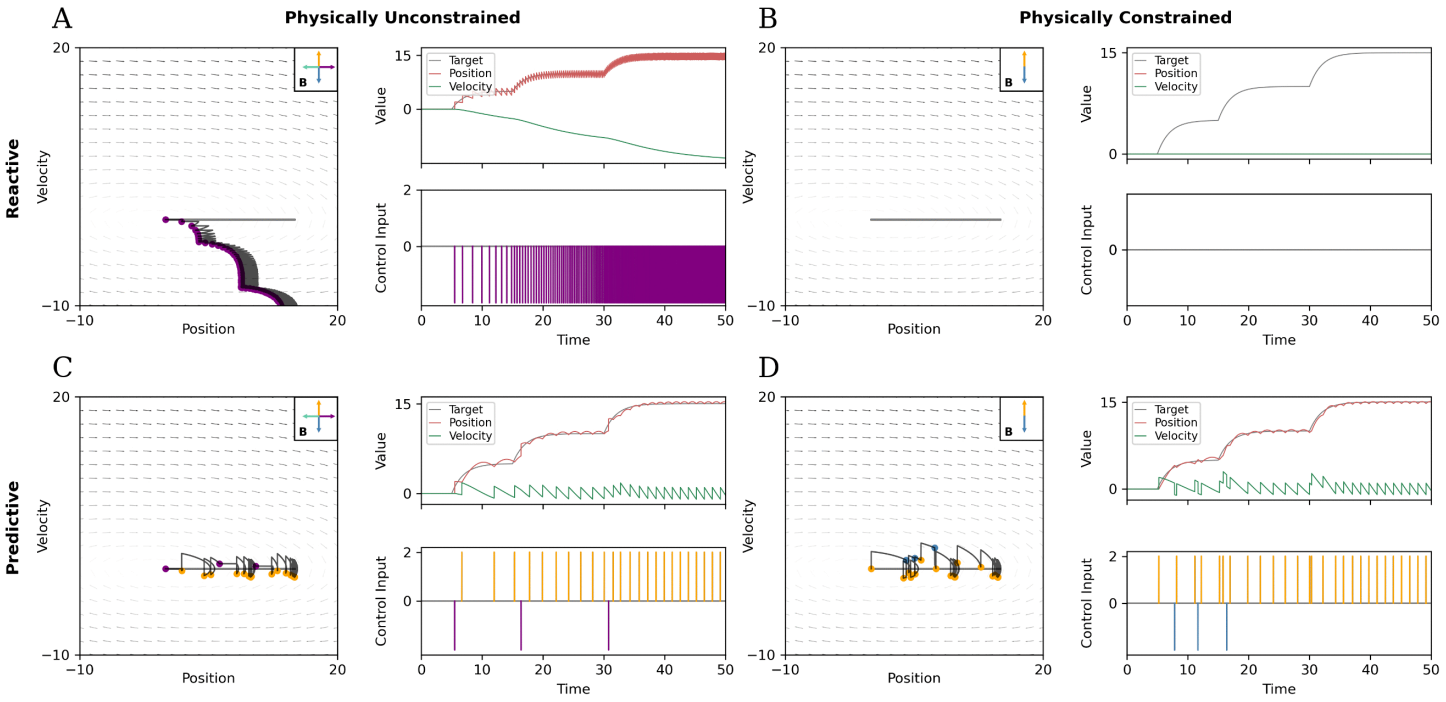

Fig 4. Spiking control simulations. The two rows denote reactive and predictive systems. The two columns represent a generic linear system, and the physical SMD. All the examples simulation are run with the simplest network possible of two neurons. A: On the left side, state space representation of a reactive spiking control approach for a generic unconstrained system. The gray line corresponds to the changing target. Although the matrix **B** allows for spikes in all four directions, the network continuously spikes rightwards, ignoring the intrinsic dynamics of the plant. On the right side, the top figure shows the changes of velocity, position and target over time, while the bottom figure shows the spike control input (each spike scaled by corresponding value in the matrix **B**). B: On the left side, the state space figure shows that without predictiveness, a physically constrained system cannot be controlled using our greedy spiking rule. In this example, **B** is constrained to only allow spikes on the velocity component, which do not cause any immediate effect on the position, preventing the loss to be decreased by the spike events. When controlling a physical system with a reactive control approach, the loss in the non-spiking case is always smaller than the loss in the spiking case, and the network never spikes. On the right side, no activity is present. C: In the predictive, unconstrained example, the system can spike in all four directions but considers its future state. In this case, the network either spikes directly towards the target or spikes upwards, exploiting the systems dynamics. D: Use of our predictive SNN approach for the control of an SMD. As shown in the state space representation, as the target position changes, the network spikes up or down, adjusting the velocity of the plant based on its predicted state in *f* seconds. On the right side, we highlight how the network's activity adjusts to the target. It employs downwards spikes to adjust for an eventual overshooting of the position relative to the target, and produces consecutive upwards spikes when the target is changing more rapidly.

https://doi.org/10.1371/journal.pcbi.1014432.g004

a physically constrained system. This is because influencing the velocity of the mass has no instant effects on its position. Therefore, the spikes can only take the velocity up or down in state space (see the red arrows in Fig 3A), increasing the distance from the target when compared to the non-spiking case (see brown dot Fig 3A). As a result, following our greedy spiking rule, the loss is never decreased by a spike, and the network remains silent. To appreciate these effects, the network should then be able to predict future states of the plant, accounting for the effect of the spiked velocity on the position of the mass.

Predictive control

We introduce our predictive spiking algorithm to a non-constrained system. In this scenario, the controller is allowed to spike any dimension of the system, but it is also able to predict the plant's future states, and it produces spikes that improve the loss *f* seconds from the current state. From Fig 4C, we can see how, when exerting the control, the position is sometimes directly moved towards the target (as in the reactive case), but optimizing the local losses often equates

to kicking the system in its velocity, exploiting the intrinsic dynamics of the plant. Our predictive algorithm exploits the intrinsic dynamics of the system even in the absence of physical constraints, although in some instances, the mass is still instantaneously moved towards the target. Finally, we apply the same predictive algorithm to the SMD simulation. In this scenario, the spiking network accounts for future states of the system, and is constrained to only spike the velocity either up or down, adjusting to the changing target position. The trajectory approaches the target and the loss f seconds after a spike is minimized. If the position then departs from the target, the losses of the neurons will change their value, eventually causing the neuron to reach its threshold. At this point, the velocity will be either spiked up or down, causing the network to trace the target correctly. As shown in Fig 4D, our predictive spiking rule introduces an element of anticipation, making the control strategy more robust in physically constrained dynamical systems, and allowing for the control of the SMD where a reactive control would fail.

The controller as a recurrent spiking neural network

We now focus on the example activity of a predictive control of an SMD with a network of four neurons, as shown in Fig 5. A visual representation of the structure of this network is shown in Fig 2. In this example, we want to show the network's performance, spikes, voltage traces, and errors. To bring all four neurons to spike, we choose a slightly more complex target $\mathbf{z}$, which spans positive and negative values. The control task and the network are set up in the same way as the previous SMD example (Fig 4D). In each timestep of the simulation, the neurons compute spiking and non-spiking losses, accounting for the future state of the system in f seconds, and its distance towards the target. The distance to the target considered in these losses is represented as the predictive error. When its value is large enough, the neuron's threshold is reached, and a spike is produced to minimize it. In turn, this minimizes the actual error, allowing the position of the mass to approach the target position. The sudden changes in value of the predictive error are caused by the voltage traces of the neurons reaching their threshold, and thus temporally coincide with the spike events. This means that the neuron's voltages are inherently encoding the predictive error of the control problem at hand. For more information on how this quantity is encoded in the voltages, see Materials and Methods. Each emitted spike directly impacts the downstream system. It is scaled by the matrix $\mathbf{B}$ by a different amount for each neuron, and mapped as a positive or negative contribution to the velocity in $\mathbf{x}$. Between spike moments, the system evolves as an open loop system, without any control signal in input, following its intrinsic dynamics. The control loop is closed each time a spike is produced and mapped onto the velocity. Since each spike is modeled as a Dirac delta function, the corresponding change in velocity is instantaneous and the control naturally exploits the intrinsic dynamics of the plant to minimize the predictive error. At time $T = 20$ seconds, we inject noise in the voltage of each neuron, which can cause them to reach their threshold independently from the computed loss. The local nature of the spiking rule (see Discussion) helps with preventing this noisy spiking from affecting the resulting control performance. We expand on this finding in the S1 Appendix, where we identify three regimes based on the voltage noise levels. Injecting progressively higher levels of noise will cause a loss of sparsity in the spiking activity, and eventually a failure in tracing the target (Fig C in S1 Appendix).

Varying meta-parameters

The control network we employ operates by expressing its neural parameters with elements of the downstream plant, which serves as a proof-of-concept for a spiking network that can directly compute control in a linear dynamical system. However, at this stage, other features of the network are still imposed as meta-parameters to adjust the control to our SMD example. For a detailed list of all the parameter setup used in this work, see Materials and Methods.

Number of neurons (N). The number of neurons in the network heavily influences the control of the plant. For instance, when operating on a system with very simple oscillatory dynamics as our SMD, employing more neurons can make the control more robust to different settings, but it raises the need to manage the spiking activity among neurons, especially since they do not have access to network-level information. This can be achieved by imposing asynchronous

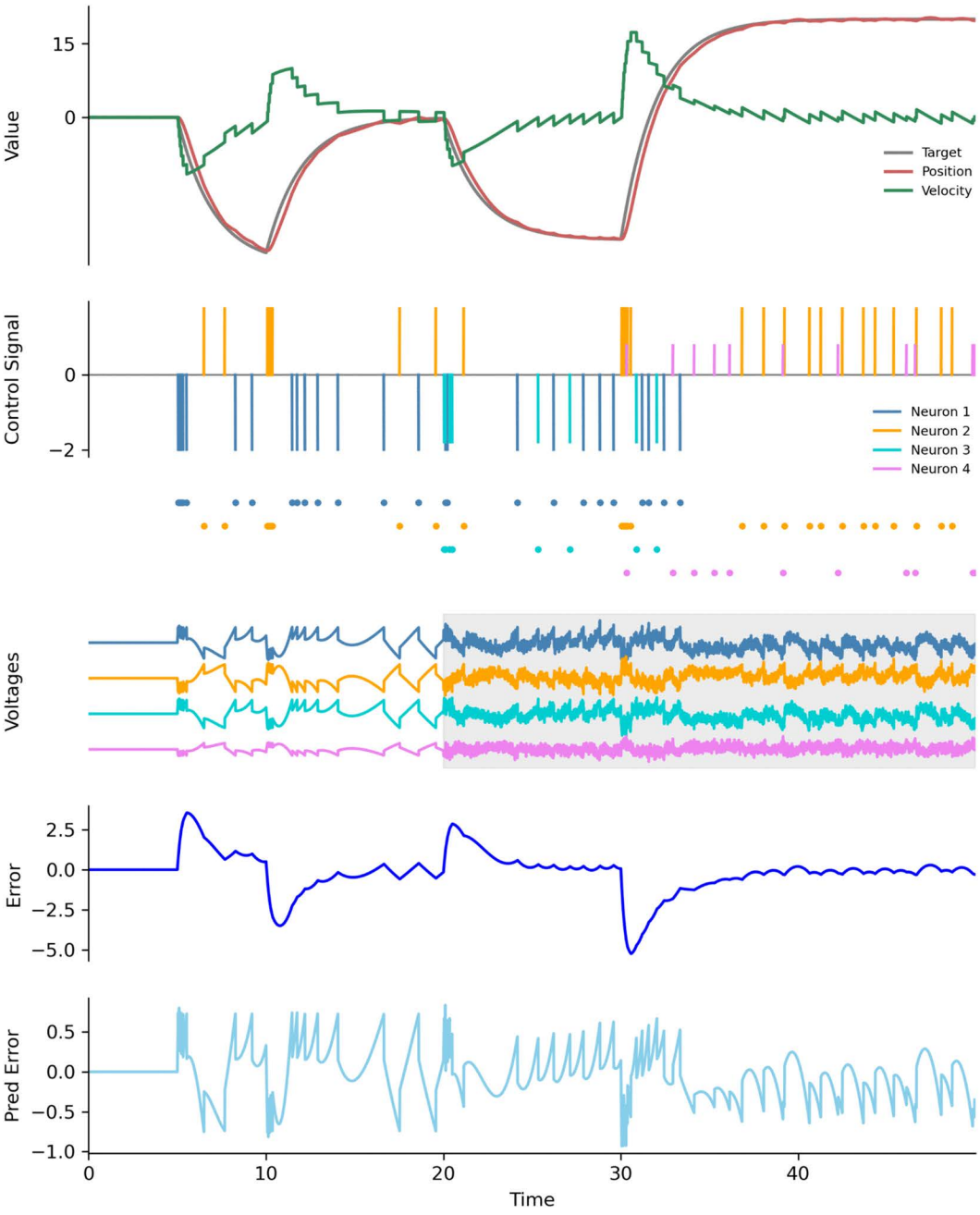

Fig 5. Example activity of a network of four neurons. Each spike plotted as control signal is modeled as a Dirac delta function that gets mapped onto the plant's dynamics through the matrix **B**, and instantaneously influences the velocity component of the system (see Results). Spike moments and the relative voltage traces are shown, as well as the resulting errors. The error trace (deep blue line) describes the current distance between the position and a target state, while the predictive error (light blue line) measures the distance between the target and a predicted position, *f* seconds into the future. At time $T = 20$ seconds into the simulation, we inject noise (drawn as random samples from a normal distribution with standard deviation $\sigma = 0.08$) in the voltages of the neurons. The network remains robust to noisy spiking and the control performance is not affected.

https://doi.org/10.1371/journal.pcbi.1014432.g005

firing or introducing an adaptive cost on spike, as mentioned in Results and Materials and Methods. While for our activity example in Fig 5 we showed the results for four neurons, in all other demonstrations we kept the network as simple as possible and employed only two neurons: spiking the velocity of the plant either up or down by the same amount. Indeed, for systems with more complex dynamics and targets, exerting effective control requires larger networks, as in the coupled oscillator case of Fig 8 or the arm reaching example of Fig 9, where $N = 500$.

Kick strength (B). The spikes of our SNN are Dirac delta functions that operate as control signals on the dynamics of the plant. These signals are mapped onto the downstream system via the matrix **B**, which can arbitrarily scale their value, resulting in smaller or larger kicks to the velocity of the system. This is the case for Figs 5, 8, and 9, where (since $N > 2$) each neuron's spike affects the velocity of the plant differently. In all the other examples, since we refer to the same SMD system with the same dynamics, we decide to keep **B** fixed, only allowing the system's velocity to be kicked up or down with the same strength. This makes our proof-of-concept as clear as possible. Of course, the value of **B** holds great relevance when varying the control objective, as it can let the same network control significantly different plants, as well as exert control on the same plant with different levels of accuracy. For this parameter exploration, since we only tackle the same SMD example, we keep a fixed **B** value that produces kicks of a compatible magnitude with the downstream plant.

Global threshold scaling (μ). When deriving the network, we can impose a scaling term μ on the loss, to introduce a cost for spiking (Eq. 3). This term influences the neuron's threshold value (see Materials and Methods). Scaling the neuron's threshold gives interesting results when controlling the plant (Fig 6), since it affects the number of spikes emitted as well as the accuracy in tracing the target position. Parameter α has a similar effect on the resulting control, since it scales the threshold based on the neuron's past activity.

Predictive window (f). When computing the spiking decision, each neuron compares spiking and non-spiking losses. As mentioned, these are predictive losses: they take into account the effect of a spike on the system f seconds into the future (see Results). The width of this future time window is crucial for the control results (Fig 6). Looking further into the future allows neurons to better assess whether the trajectory will approach or depart from the target, often preventing unnecessary spikes. On the other hand, if the window f is very large, the predicted state used in the loss comparison might approach (or depart from) the target way before the actual state of the plant does. This influences the accuracy with which the network represents the effect of its own control on the system's dynamics, as we will explain below. As mentioned in Discussion, for most of the simulations in this work we select a value for f that fits the scale of the downstream plant to showcase the results with clarity.

Control regimes

Different combinations of these meta-parameters can give different results in the control of the plant. Quantifying the optimality of the control strategy in comparison to other paradigms (e.g., standard LQR, or filtered-spikes) is very difficult. As described in the Discussion, each neuron in our network follows a greedy spiking rule given the control set, but like in other approaches, this does not necessarily produce the best overall controller. It is of interest to vary some control parameters, considering the cases where the changes in target value are traced by the system's state (and therefore the SMD is successfully controlled). Within these cases, we identified different regimes of control for our network. In these explorations we decided to maintain fixed the number of neurons ($N = 2$) and the kick strength **B**. We therefore vary the future time window f, influencing the predicted state based on which the network computes the spiking decision, and μ, the global scaling factor for threshold. As shown in Fig 6A, networks with a lower μ tend to spike more, using more energy, as defined by the integrated acceleration onto the system (for this system, this equates to the sum of the total spike count of the simulation, with each spike scaled by the corresponding value in the matrix **B**, as described in S1 Appendix). Given the same f, these networks also tend to have a lower error compared to high μ networks (Fig 6B), since a lower threshold allows to adjust more often to the target position. In general, it emerges from Fig 6B how a relatively wide range of parameters allows for the control of the plant, giving comparable error measures. Only more extreme parameters combinations

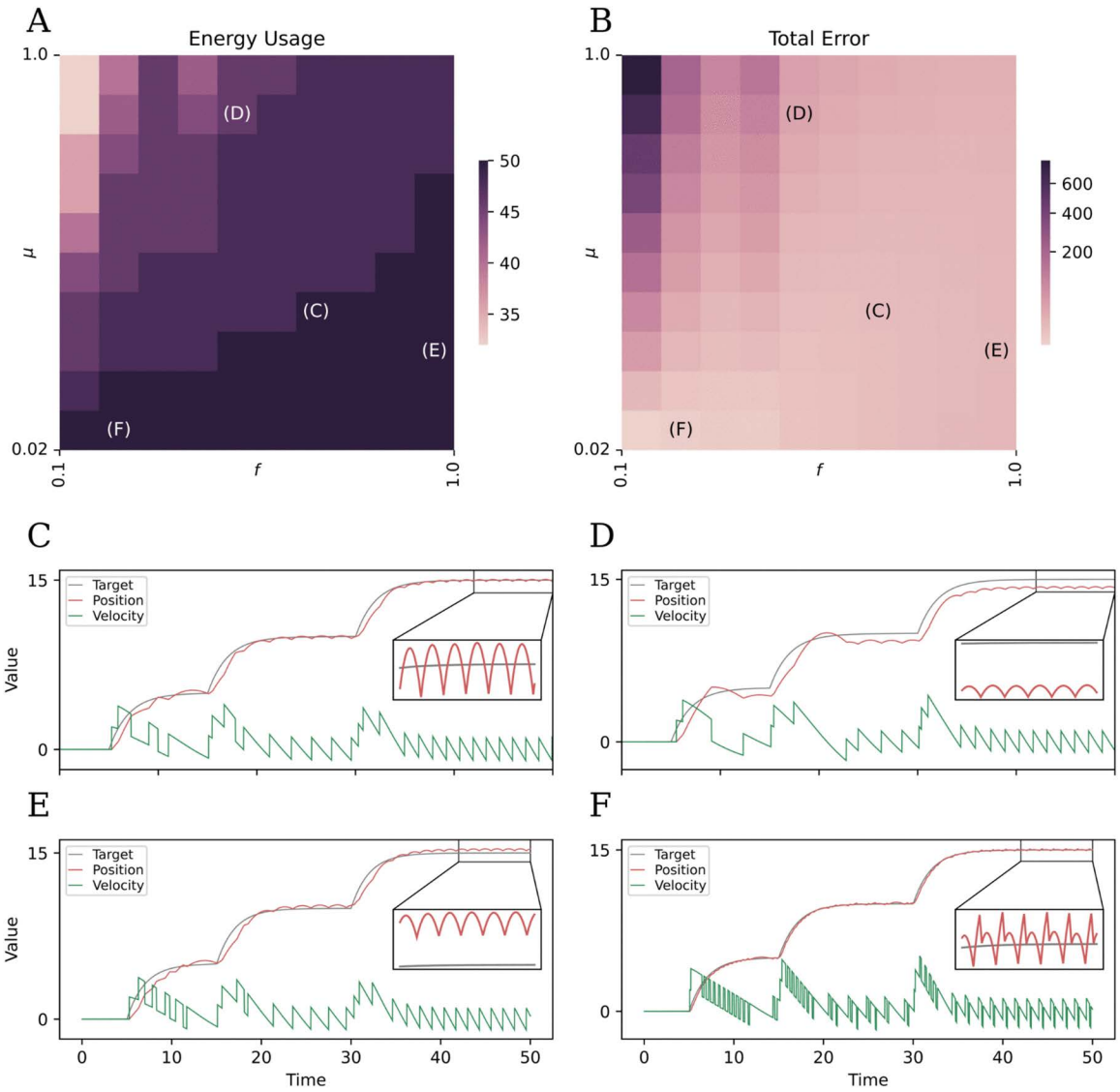

Fig 6. Parameter sweeps and control performance. Changing two meta-parameter (μ and *f*) in simulating SMD with spiking control. A: Energy Usage, calculated as integrated acceleration onto the system in each simulation (**Bs**), for various combinations of μ and *f* (linear scale). As described in S1 Appendix, the measure of acceleration is chosen for simplicity of interpretation in our framework. However, measures of work done on the plant can also be adopted to describe the energy input in the system. B: Total Error, calculated as cumulative distance from the state **x** to the target **z** in each simulation, for different combinations of μ and *f* (logarithmic scale). C-F: Different control regimes for different combinations of μ and ***f***. C: Efficient control regime. D: Under-control regime. E: Over-control regime. F: Inefficient control regime.

https://doi.org/10.1371/journal.pcbi.1014432.g006

(a high threshold scaling, especially with small *f*) result in the network not tracing the target, yielding significantly higher cumulative error.

Efficient and inefficient control. Within the boundaries of these trade-offs, we identified four regimes (Fig 6C-6F). With an adequate future time window *f* and relatively low μ, the network can operate in an efficient regime (Fig 6C, where the target position is followed precisely, while keeping a relatively low spike count. If the spike count is lowered any further from this point, the control input becomes too sparse, and the SMD mass does not closely orbit the target

position. On the contrary, decreasing the value of both these meta-parameters yields the inefficient regime, where the target is still traced with comparable precision, but the spike count, and as a consequence the energy input, is considerably higher (Fig 6F).

Under- and over-control. We then have the case where the network is allowed to look far enough in future states of the system, but μ is also very high, making it difficult for the neurons to spike enough, and to trace the target accurately (Fig 6D). In this regime of under-control, the position approximates the target but never manages to reach it. Finally, in the last case, the neurons are relatively free to spike as μ is low, and they are bound to look very far into future states of the system (Fig 6E). In this regime of over-control, the neurons fire based on a predicted position that is far into the future (large *f*) and thus differs significantly from the actual one. For instance, while the current state of the system might have not reached the target yet, the predicted state is already well past it, which causes the neurons to fire earlier than needed. This over adjustment makes the position hover slightly past the actual target.

Comparing control approaches

Comparing control approaches Lastly, we want to showcase the differences in the resulting control for three approaches: the continuous LQR control, the filtered-spikes control, and our predictive spiking control. As for previous examples, we consider a physically constrained simulation of an SMD, with a progressively changing target. In Fig 7 we directly compare the same measures for all three methods. As shown in the error trace (bottom row), the filtered-spikes algorithm (Fig 7B) approximates the control strategy of the continuous LQR (Fig 7A), as described in Materials and Methods. Spiking control (Fig 7C) has similar increases and decreases of the error (contingent to the changes in the value of the target), although its profile is very different, as no optimal control signal is being traced. Furthermore, the non-continuous nature of its control inputs only allows the position to orbit the target with a certain accuracy, without ever stationing on the exact target value. Analogous differences are highlighted when showing the activity over time of these networks (Fig 7, second row). In the continuous and filtered cases, the velocity is not spiked, and the position slowly adjusts to the changes in target, stabilizing once the target is reached. On the contrary, spiking control is progressively adjusting the velocity up or down, tracing the target as soon as it changes value. Once the target is reached, the controller orbits the target position by spiking regularly until the end of the simulation. Other fundamental differences are highlighted in the control input (Fig 7, third row), and in the consequent trajectory in state space (Fig 7, first row). As shown, the filtered spikes controller is able to smoothly control the state of the system by approximating the control input of the LQR. In this sense, the spikes are only representing the system's state, but the control signal is still continuous, as it is inputted in each timestep of the simulation. This approximation requires the network to spike much more than our spiking control algorithm, which exclusively inputs a kick when it needs to adjust the position to the target, and is not approximating any optimal control solution. Finally, on the comparison between filtered and spiking control, the role of a filtering element in the paradigm depends on the modeling choice we operate and the hypothesis we consider for the specific research question. In the filtered spikes approach, the filter is present within the network itself, so that the spikes represent but do not compute the control signal directly. In our spiking algorithm, the filtering element is placed outside of the network and into the plant: here, the matrix **B** directly maps spike events onto the system state, which makes the control inputs impulsive rather than continuous. Indeed the choice of positioning and defining a certain filtering element in different component of the control paradigm is partly artificial, but it does determine the nature of the control input and the interpretation of each element of the paradigm itself. As described in S1 Appendix, there is no universal energy advantage of our spiking algorithm from a control-theoretic perspective when compared to filtered spikes or continuous approaches (Fig B in S1 Appendix). Rather, the strength of our contribution lies primarily in the formulation of the spiking control scheme, which guarantees sparsity of the resulting spiking activity, interpretability of the model's components, and adaptability to spike-dependent control applications (for instance, when using neuromorphic hardware).

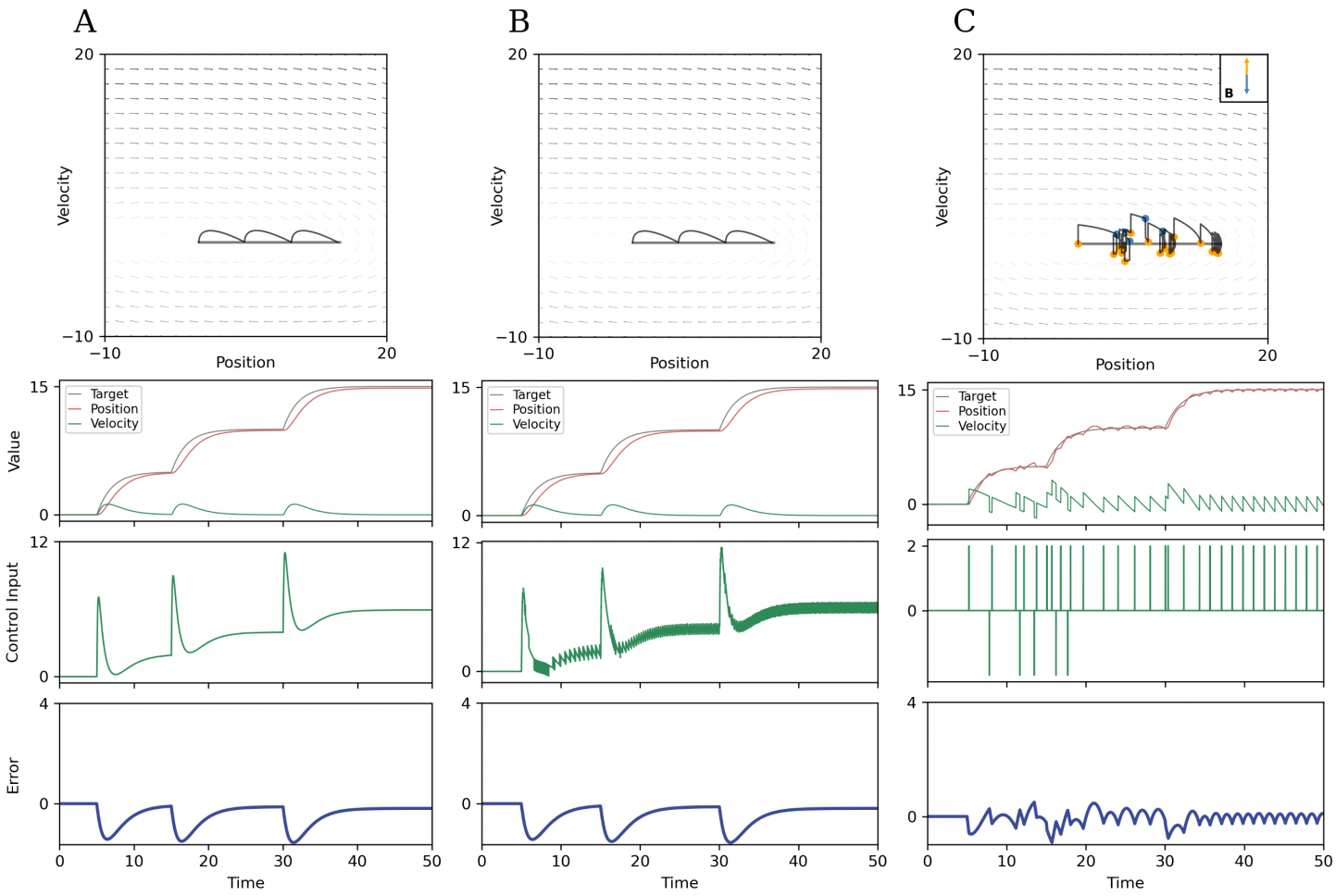

Fig 7. Control example of an SMD with different approaches. Simulation of the control of an SMD with continuous control (LQR), filtered-spikes control, and predictive spiking control: A: LQR control example: the controller continuously adjusts the position to the target by modulating the velocity. This is shown in the phase plane (first row) and in the activity plot (second row). Each time the target value changes, the position slowly adjusts to it, stabilizing when the target is reached. The control signal (third row) is larger when the target changes value, and the velocity is modulated to adapt to it. During these moments, the mass is the furthest away from the target, which is reflected in the error profile (bottom row). B: The filtered spikes control is approximating the LQR signal. Therefore, the trajectory in space, the activity over time and consequently the error (first, second, and last rows) have the same profile as in panel **(A)**. The approximation is seen in the control input (third row). Since the system is represented through spikes, the network keeps spiking to trace the LQR signal. C: In our predictive spiking algorithm, spikes directly influence the velocity of the system. This is shown in the upwards and downwards kicks in state space and in the activity plot (first and second row). On the one hand, this helps the controller adjust faster to changes in target values. On the other hand, when the target is reached, the controller can only orbit its position by spiking. This fundamental difference is also highlighted in the error profile (bottom row). In the control input plot (third row) we can see how the spike count is drastically lower in spike control than in the filtered-spikes one.

https://doi.org/10.1371/journal.pcbi.1014432.g007

Controlling coupled oscillators

We now scale up the same algorithm described in previous sections to control a higher dimensional linear system. This time, the plant we chose is a coupled oscillator system, where M number of SMDs are connected in series, such that the position of one mass affects the acceleration of the ones connected. The network has to control multiple SMDs, while taking into account the coupled effects between them. For the example shown in Fig 8, we consider the control of $M = 10$ coupled masses using a network of $N = 500$ neurons. As shown, the control algorithm scale naturally to this complex

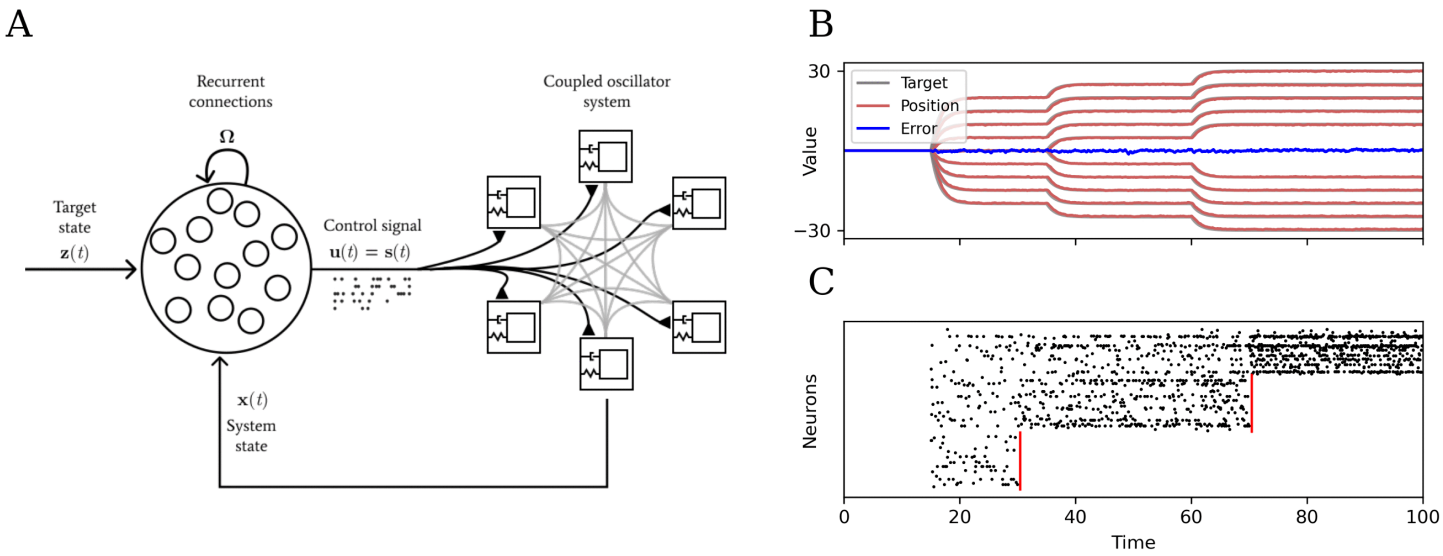

Fig 8. Control simulation of a coupled oscillator system. A: Scheme of the feedback spiking control for a downstream coupled-oscillator system. The current state of the system $\mathbf{x}(t)$ is given as input to the network, along with target state values $\mathbf{z}(t)$. The SNN outputs spikes $\mathbf{s}(t)$. These represent the control signal $\mathbf{u}(t)$, computed as a function of both inputs. As for the other networks, $\mathbf{u}(t)$ is then mapped onto the dynamics of the downstream system via the **B** matrix. This time, the downstream system is composed of M masses connected to each other, such that the acceleration of each mass influences the position of the next one. The plant dynamics are still linear, but they are significantly up-scaled in dimensionality when compared to the single SMD examples. B: Activity for the control of a system with $M = 10$ coupled masses. Each mass has a different target position over time $\mathbf{z}(t)$. For this example, the SNN makes use of a randomly initialized **B** matrix to map the spikes onto the high-dimensional system and guide the position of each mass to its designated target. Note: for clarity, here we do not show the spiked velocity over time like in previous figures. In blue we show the cumulative error of all neurons over time. C: Spiking activity of the controlling network. We demonstrate the robustness of the control by progressively silencing portions of the $N = 500$ neurons during the simulation. The vertical red lines indicate the silencing of neurons in certain timesteps. The network successfully redistributes the spiking activity among the non-silenced neurons and as a result, the control performance is not affected (as shown in panel **B**).

https://doi.org/10.1371/journal.pcbi.1014432.g008

linear system: even though the masses are coupled with each other, our local spiking rule is able to coordinate the spikes as to allow all masses to reach their respective target (Fig 8B). Neurons are assigned an element in the matrix **B** (now scaled up to the correct dimensionality based on the number of masses M) with a random normalized value. The spike of each neuron then maps either as a positive or negative kick on the velocity of a mass by the corresponding amount. This results in some neurons having a higher kick strength than others, but as before, only the neurons that allow the system to decrease the distance from the target in f second will fire. During the simulation we observe how spiking activity is sparse and distributed among neurons (Fig 8C). We then silence random neurons to test the robustness of the system. After each silencing event, the activity is distributed among the remaining active neurons and there is no effect on the control performance. This shows how our control algorithm scales to complex linear system and is robust to cell loss: given a large enough network, no specific neuron is determining the outcome in the control performance. The robustness observed in this system is a direct consequence of the greedy spiking rule and the redundancy inherent in the SCN framework, described in [20]. In this architecture, the membrane potential of each neuron tracks a projection of the global prediction error (distance to the target). If a specific neuron is silenced, the prediction error it would have corrected continues to accumulate in the membrane potentials of the remaining active neurons. Thus, neurons with a similar alignment (in our case, the neurons whose spikes are closely mapped in state space) will fire, redistributing the work across the remaining neurons in the population. Another example of spiking control over more complex linear system can be found in the Results, where we employ our algorithm to control a series of muscle fiber unit models to move a point arm in space (Fig 9).

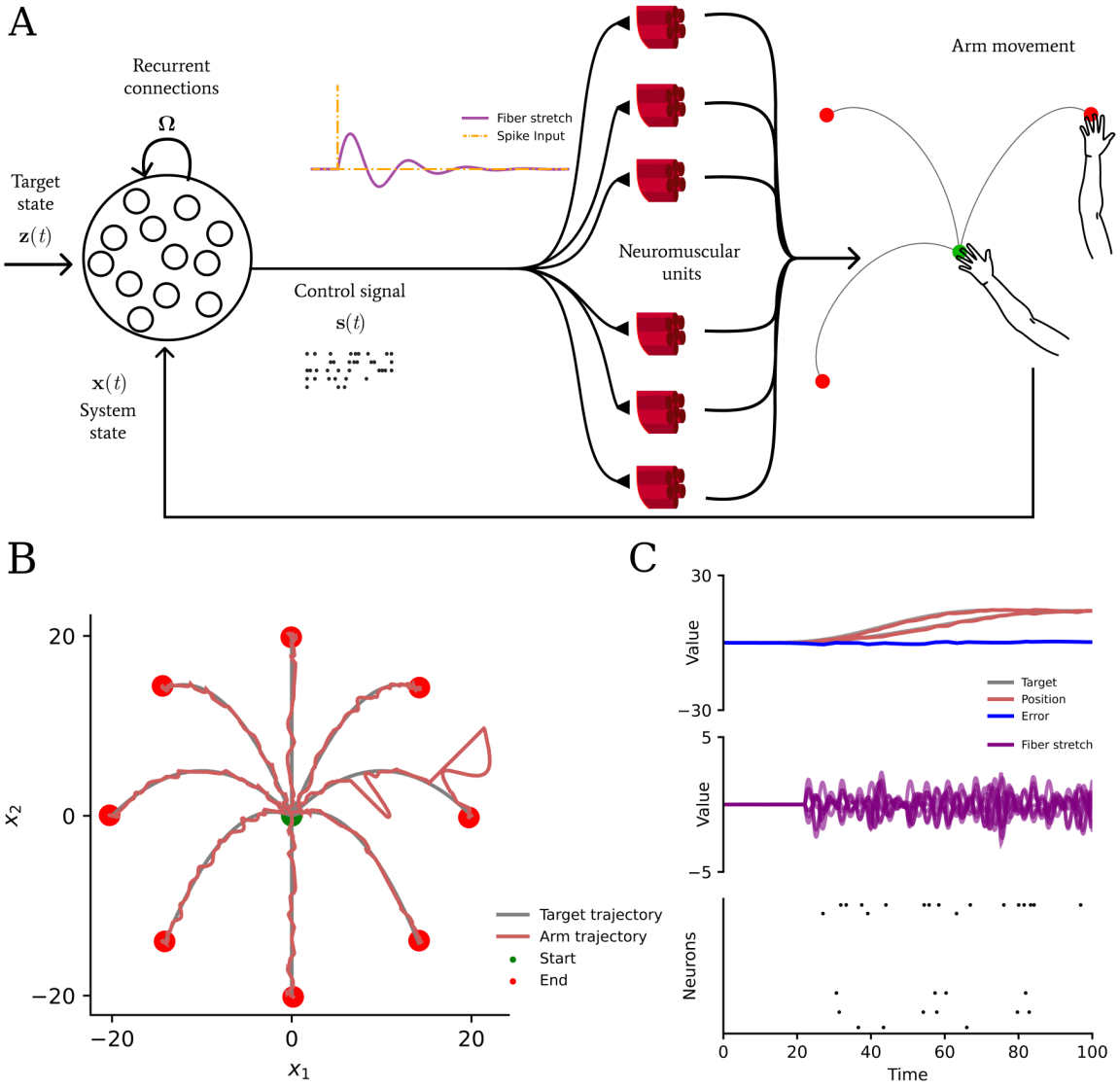

Fig 9. Arm reaching simulation. A: Scheme of the feedback spiking control for a single point arm reaching task. The current state of the system $\mathbf{x}(t)$ includes the states of the neuromuscular units and of the point arm. It is given as input to the network, along with target state values $\mathbf{z}(t)$ which are only relative to the arm position (the state of each unit is not constrained by the target). The SNN outputs spikes $\mathbf{s}(t)$, are then mapped through **B** onto the dynamics of the fiber units. The spikes affect the velocity component of each SMD, causing an oscillatory displacement. In these examples, the $N = 500$ neurons are randomly assigned to eight uncoupled SMDs. The state of such units are then linked to a 2D point mass. B: Examples of eight separate reaching tasks where the network controls the fiber units to guide the point arm through eight different target trajectories. In the rightward trajectory task, we perturb the velocity of the arm in two points during the simulation. After the perturbations, the network is able to redirect the trajectory and trace the target. C: Activity measures for one of the eight tracing tasks. In the top panel, the two coordinates x_1 and x_2 of the point arm correctly trace the target trajectory, as the error is kept around zero. In the middle panel, the eight SMDs oscillate according to the spike input, and influence the arm movement. In the bottom panel, little spike activity is needed to generate the necessary oscillations in the unit and cause the point arm to move in space.

https://doi.org/10.1371/journal.pcbi.1014432.g009

Arms reaching task simulation

We now use our paradigm to propose a bio-inspired example of motor control. In this example, the spiking network aims at controlling a point arm model through oscillating SMD elements that represent simplified fiber units. The use of

controlled oscillators as models of muscle fiber displacement is present in the literature [14,41,42]. The overall dynamical system is described by a single linear state equation, whose state vector $\mathbf{x}(t)$ contains both the kinematic variables of the arm (its position and velocity in time) and the internal mechanical states of the SMDs. Although all the state variables transformations are included in the same system matrix $\mathbf{A}$, they play fundamentally different physical roles. The arm states correspond to observable coordinates of a point-arm extremity moving in two dimensions. The SMD states represent internal muscle-like fiber extensions (fiber stretches) relative to their equilibrium lengths. These displacements are not spatial coordinates in the external workspace; rather, they quantify mechanical deformation within each fiber unit. Through standard SMD dynamics, a fiber extension and its velocity generate force, which in turn drives the arm's motion. The spikes of the network directly control the velocity with which the fiber is extended (or contracted). The SMD dynamics then evolve continuously according to their intrinsic spring stiffness, damping, and inertial properties. The collective state of these units is then linked to the position and velocity of the point arm in space (Fig 9A). Importantly, the control objective (and consequently the loss) is defined exclusively in terms of the arm's target trajectory. The target $\mathbf{z}(t)$ specifies desired time-varying coordinates in two-dimensional space, and no explicit target is assigned to the SMD states. Although the full system state includes both arm and fiber units variables, the cost function penalizes only deviations of the arm position from the desired trajectory, leaving the internal unit states unconstrained except through their mechanical coupling. In the example provided in Fig 9B, eight units oscillate in response to spike control signals and consequently affect the position of a single point in 2D space to trace a certain target. Starting from a point in the origin (green dot) we set eight different target trajectories to be traced consecutively and their respective end points (red dots). As shown, the network successfully controls the state of the fiber units to trace the target movements. During the rightward trajectory task (from the origin (0,0) to the (20,0) coordinates) we perturb the arm on its way to the target. To do so, we set the velocity of the arm to fixed values during certain timesteps, after which we let the network resume the control. As shown, the network manages to redirect the trajectory and trace the target. In Fig 9C, we show some measures of activity for one of the example trajectories. We illustrate how both components of the trajectory (x_1 and x_2) are correctly traced, and the error is minimized. This is caused by the fiber displacement (stretch) of the units over time. Each spike mapped onto these units is reflected in a dampened oscillatory behavior. As a result, as shown in the bottom panel, very sparse spiking behavior is sufficient to generate the correct SMD responses that drive the point arm to the target.

Discussion

Proposed spiking control framework

As described in Introduction, biological neurons use spikes for signaling and control of downstream system. Control of dynamical systems has been widely studied and applied in different contexts, including artificial neural networks [20–22,24,25], but generally doesn't consider spikes directly as control signals. In this work we remedied this by showcasing how control can be computed directly through the use of single spike events. This allowed us to obtain a model that uses spikes to actively kick the dynamics of the downstream plant, and exploits its intrinsic dynamics to reach a certain target. The emitted spikes are instantaneous events (modeled as Dirac delta functions, see Results). The spike generation mechanisms in the network and the mappings to the system need to be adjusted to the various constraints the plant might have in its dynamics (as explained in Results). In order for this control paradigm to work on physically constrained systems, we developed a predictive algorithm, that takes into consideration the effect of our control inputs onto the system within a certain future window, f. Our spiking rule is fully local and is computed for each neuron in every timestep of the simulation. Therefore, the target can be reached without a network-level objective, and without any explicit learning paradigm. Of course, introducing learning could still be a very useful addition to adjust all the meta-parameters of the network when adapting the control to different plants, or to tackle plants with unknown dynamics. From this simple spiking rule, we obtain, without imposing it, a recurrent network of IF neurons that exerts control by only inputting a spike if it improves the defined loss. This rule does not guarantee the optimality of the control as a whole (as explained in Discussion), but allows

for the control to work without the need for network-level information. Lastly, we scaled up this framework to control a coupled oscillator system, with various connected masses (each with its different target) and a larger network. We showcase that not only our control algorithm is easily scalable to complex linear systems and higher-dimensional networks, but it is also robust to neuron loss.

Assumptions and constraints

Greedy spiking rule. An important clarification about our proposed algorithm is that the spiking rule by which the network produces control signals does not guarantee any optimality on the controller as a whole. Each timestep in the simulation, the neurons operate the spike decision by independently trying to decrease the distance towards a target. That is why the neurons produce control signals (spikes) in a "greedy" way, as defined in the work of [20]. If spiking improves the loss, the neuron always spikes, as that is the only information that it has available. No other network-level information is used in the process. In other words, computing this local greedy spike decision for each neuron in the current timestep (albeit considering the effect of the spike in f seconds), does not equate to finding the best set of spikes over a whole future window, let alone over the whole simulation. As a result, our approach does not guarantee that the control actuated by the network is necessarily the optimal control solution for reaching a certain target. Furthermore, as described in the Results, many of the meta-parameters in the network are manually set rather than learned or optimized for the control task at hand. In most of our examples, we impose the values of f, μ, $\mathbf{B}$, and N that best fit the scale of the system to control, to showcase our proof of concept with clarity.

Features of the control scheme. From the described spiking rule, we obtain a recurrent LIF network able to exert control over a linear dynamical system. Although such spiking rule is based on a loss evaluated over a time window of duration f, the control problem considered in this work is of infinite-horizon in nature. That is to say, there is no terminal time or terminal cost in the control problem, and control actions are generated progressively as the system evolves, without reference to a final target state. Therefore, the parameter f only specifies a fixed-time prediction window over which the loss is evaluated. The control decision at each time does not involve any optimization of future control actions.

Consequently, this work does not directly include an account for how the cumulative effect of spikes allows for the reduction of control error. However, the stability analysis provided by [43], traces analogies between our discrete impulses control method and an analogue continuous control, and in doing so, proves the control stability of our paradigm. The core of the work of [43] is the introduction of an auxiliary variable $x_c(t)$, defined as the sum of the plant state $x(t)$ and a linear combination of the internal neuronal variables $z(t)$ (analogue to our voltages $\mathbf{V}(t)$). The auxiliary variable can be seen as a continuous counterpart of our impulsive control, accounting for hypothetical impulses from the neurons, rescaled by how close their voltage is to its threshold. Applying these hypothetical impulses to the system state amounts to a jump from $x(t)$ to $x_c(t)$. In the section "Extension to connected neuronal units" of their work, [43] use Lemma 12 to prove that the internal neuronal variables $z(t)$ of the network remain uniformly bounded from above and below. Because the neuronal variables are bounded, the "perturbation" they introduce to the continuous auxiliary variable $x_c(t)$ is also bounded. Since the neuronal variables are encoding for the distance between the state and the target, that error (while not reaching zero due to the kicks strenght $\mathbf{B}$), will converge to and stay within a small region around the target. The spiking system is then guaranteed to be practically stable in the control of the plant.

Simplicity of use case. Following a similar reasoning, we purposefully chose a simple dynamical system, which is linear (fully observable) and two-dimensional. We also chose a simple target that progressively changes position and then stabilizes. This simplicity allowed us to showcase our results without any additional complication, demonstrating how this framework easily scales to any linear dynamical system. However, the limits of our paradigm still need to be tested, by including adaptive (or learned) meta-parameters, and expanding our control algorithm to operate on non-linear systems (see Discussion). Important to note that even in an optimal controller, these meta-parameters would need to be adjusted based on the control problem at hand. The differences in the control approach of our network is highlighted in Fig 7 and

explained in Results, where we control the same SMD system with a standard LQR (optimal controller), a filtered spiking algorithm, and our spiking controller. As shown, the filtered spiking controller (Fig 7B) approximates the control signal of an LQR (Fig 7A), making the resulting trajectories in state space equivalent, while our spiking algorithm (Fig 7C) differs significantly from the optimal trajectory in state space. However, by adjusting the meta-parameter correctly, the network still traces the target with good accuracy (as shown in the error profile), while producing significantly less spikes than the filtered spiking controller.

Considerations on spiking control

Relation to prior spike-based control models. Previous works underlined the importance of each spike event in the computation of neural networks [20,34,36]. Such spike-based computational models highlight the relevance of the temporal precision of spike events, where precise spike timing is determined by each neuron's contribution to a desired output. These models exhibit features similar to cortical networks, including irregular spiking and E/I balance [36]. Many of these models are developed from SCN theory, which gives them a strong overlap with our framework but also introduces important distinctions (as described in Materials and Methods). In SCNs, as in our approach, spike activity is used to generate a signal that controls a dynamical system. However, SCNs use this signal to control their own network read-out (thereby influencing the dynamics of their internal state). This read-out can be quite complex, and include synaptic kernels, requiring a form of predictive spiking [36,37] similar to the one employed in our paradigm. In short, despite SCN-related works share common aspects with out approach, they do not explicitly hypothesize that the computational role of spikes could be controlling a different downstream system. In contrast, our work generalizes to the control any downstream linear system, rather than focusing solely on modulating the network's own internal state. Moreover, when SCN principles are employed to solve realistic control tasks [22,24,25], the emphasis on a temporal, single-spike perspective is lost, favoring a rate-based or filtered-spike approach, which glosses over the role of each spike in the network's computation.

Future outlook. In neuroscience, single spike events for control could have a huge significance when modeling downstream actuator systems. Spike-based control seems to be a promising modeling angle when explaining the control of downstream muscle actuators [14], that ultimately allow movement to occur. This paradigm could be expanded to include models of body orientation in space as well as the control of more detailed models of limbs [12]. Spiking control is also compatible with existing dynamical system approaches to modeling complex inter-network phenomena like epilepsy [44]. Further down the line, spiking control could be a useful approach to interpret how the spikes influence the dynamics of the network that produce it. This has been theorized by previous works in SCNs [20,21], suggesting that the computational role of spikes in the network is to control its dynamics, and many physiological features of biological networks are a consequence of this necessity. In this case, the downstream system is the network itself (or a second downstream network), which means that our paradigm could serve as a tool to explain potential roles of the spike code in all bio-plausible networks. As mentioned above, various works emerged from this initial SCN perspective to investigate the biophysical and computational properties of these networks, and their potential values for interpreting detailed neural models [36,37]. Not only these works stem from the common root of SCN-based modeling, but they also incorporate elements that we employ in our algorithm, such as different forms of predictive spiking. Revisiting these topics with our control-focused perspective could give interesting insights, for instance, on the computational role of E/I balance [45] or provide fresh perspectives on various properties of SNNs [46]. Following a similar path, we could study the effect of bio-plausible additions in our framework, like noise or network perturbations, analyzing for instance the robustness of our system [35].

Expanding our proof-of-concept to more complex nonlinear systems could be relevant to investigate the role of spikes in controlling realistic downstream actuators. This could include detailed models of muscle fibers for the control of movement [14], or models of complex downstream networks. Furthermore, it could have use cases for neuromorphic control

applications where a better exploration of the full state space of the plant is needed. Tackling nonlinearity might require further refinements on the model by employing local linearization techniques [47,48] or a simultaneous prediction of multiple future states [49]. In the effort to apply this algorithm to more realistic use cases, another required step would be to find a way to adjust neural parameters according to the control problem. As mentioned in Results, a lot of important parameters of the network are currently imposed to best fit the control case. Ideally, we would want the same network to be able to control different plants. This could be achieved by applying a discount factor on some parameters, which would allow us to adapt, for instance, the value of f as the simulation progresses, finding the best time window to balance the number of spikes in input and the accuracy with which the target is traced. Finally, we can introduce learning on all the parameters of the network, to make our control as flexible as possible to a wide range of downstream systems. Not having to assume that we operate on completely observable plants would greatly expand the use cases of our algorithm, and would allow for testing of various neuroscience-derived hypotheses.

Materials and methods

Simulation and network setup

Simulation details. All the simulations in this paper were implemented in Python using the following packages. NumPy(version 1.23.5): for matrix operations and numerical computations. SciPy(version 1.10.1): for solving differential equations and simulating trajectories using `solve_ivp`. Matplotlib(version 3.7.1) and Seaborn(version 0.13.2): for visualizing results, including raster plots, heatmaps, and system trajectories. Python control system library 'control' (version 0.10.1): for the initializations of the continuous control example.

Code availability. All simulations were run with Python 3.11.5. The source code and all the necessary dependencies are available at https://gitlab.socsci.ru.nl/2023-phd-paolo-agliati/spiking-neurons-as-predictive-controllers-of-linear-systems.

Time settings. In each single SMD simulation (Figs 4 to 7), the total time T is set to 50 seconds, the timestep $\delta t = 0.01s$ divides each run into $nT = 5000$ timesteps. For the coupled SMD examples (Figs 8 and 9), $T = 100s$, $\delta t = 0.01s$, and $nT = 10000$.

Target settings. For every simulation, the target $\mathbf{z}$ is only defined for the position component of the system's state, and it is initialized at zero. For Figs 4, 6, and 7, three target values are set: $z_{base} = 5$, from five second into the simulation, $z_{base} = 10$ after 15 seconds, and $z_{base} = 15$ after 30 seconds, and until the end of the simulation. For Fig 8 we use similar z_{base} steps, but we initialize them so each mass is attributed a different set of targets, which are progressively more positive or negative. In Fig 5 we use bigger and irregular steps, to better show the activity of the four neurons: $z_{base} = -30$ from five second into the simulation, $z_{base} = 0$ after 15 seconds, $z_{base} = -25$ after 20 seconds, $z_{base} = 20$ after 30 seconds, and until the end of the simulation. In Fig 9, z_{base} is defined as pre-computed curved arch trajectories for the arm positions, set at different angles in 2D space. Between each value of z_{base}, the parameter $z_{leak} = 0.5$ manages the exponential behavior of the target curve. The value of z_{leak} is adjusted to 0.8 and 1 for Figs 8 and 9, respectively.

Neurons and spike mapping. For most runs, the number of neurons N is fixed to two, except for Fig 5 where $N = 4$. In the scaling up examples of Figs 8 and 9, $N = 500$. $\mathbf{B}$ is kept to spike the velocity component of a fixed value of 2 or −2 for most simulations (the values relative to the positions are kept at zero). One exception is the physically unconstrained cases of Fig 4, where $\mathbf{B}$ can also allow spikes to the position of the same amounts (2 and −2). For Fig 5, $\mathbf{B}$ values are sampled from a normal distribution, with the second column (velocity) normalized to have norm 1. The resulting value is then doubled. The same normalized $\mathbf{B}$ values are quadrupled in the coupled oscillator simulation of Fig 8, and multiplied by six in the examples of Fig 9. In this three cases, the dimensions of matrix $\mathbf{B}$ are also adapted to allow each one of the $N = 4$ and $N = 500$ neurons to have their spikes mapped onto the system.

Future time window and spike cost. The future window f is kept at 0.3 for the predictive simulations in Figs 4, 5, 7, and 8. Given the indirect effects of the muscle units on the arm's movement, we adjusted to a bigger value of $f = 2$ for Fig 9.

When using reactive control, f is set to zero. For Fig 6, f values in the subsequent simulations are linearly spaced from 0.1 to 1 (included), with the fourth value rounded up at 0.25. μ is fixed at 0.3 for the simulations in Figs 4, 7 and 9. For Fig 5, μ is set to 0.5 and for Fig 8, $\mu = 0.4$. In Fig 6, μ values in the subsequent simulations are linearly spaced from 0.02 to 1 (included), with the third value rounded up at 0.25.

Initial state and system matrix. The initial state of the simulation $\mathbf{x}_0$ (at $T = 0$) is always kept at [0,0]. The system matrix **A** is also kept the same for all the single-SMD simulations (since we control the same SMD). The first row is set at [0, 0.5], and the second at [–0.1, –0.1]. For Figs 8 and 9, the variable $M = 10$ is introduced to indicate the number of masses (or motor units) to control. In these examples, the dimensions of all other matrices are then scaled based on M (and N), and coupling values $\gamma = -0.3$ are introduced to make the position and acceleration of the units dependent on each other. To test robustness to cell silencing in Fig 8, we silence 180 random neurons 30 seconds into the simulation, and other 180 after 70 seconds.

Continuous and filtered-spiking parameters. The examples of continuous and filtered-spiking control in Fig 7A and 7B largely use the same settings as the other single-SMD simulations. In the continuous control case, we introduce an LQR gain matrix **K** to compute the control signal, initialized using the `control.lqr` function from the python control package. To initialize **K**, we use the matrix **B** that maps the control signal onto the velocity of system, set as [0, 0.25]. We will also need a control cost $R = 0.001$ (analogous to μ in the spiking control case), and finally a $K \times K$ matrix **Q** to introduce a cost on the dimensions of the system (analogous to **C** in our spiking paradigm). The first row of **Q** is set at [1, 0], while the second is [0, 1]. For our filtered spiking control we will need a matrix $\mathbf{D} = [1, -1]$ to map the filtered spike traces onto the network read-out. For our example of Fig 7B, we use a spiking cost $\mu_f = 0.1$, and a decay factor $\lambda = 1$ for the filtered spike traces.

Derivation

Network. In our work, we derive a spiking neural network (SNNs) of leaky integrate-and-fire (LIF) neurons from control principles. In general, a network of N such neurons is described by their voltage dynamics

$$\dot{\mathbf{V}}(t) = -\lambda \mathbf{V}(t) + \mathbf{F}\mathbf{x}(t) + \Omega \mathbf{s}(t), \tag{7}$$

where $\mathbf{V}(t) \in \mathbb{R}^N$ are the neurons' voltages, λ determines the membrane leak time-constant, $\mathbf{x}(t) \in \mathbb{R}^K$ are the inputs, $\mathbf{F} \in \mathbb{R}^{N \times K}$ are the forward weights, $\Omega \in \mathbb{R}^{N \times N}$ are the recurrent weights, and $\mathbf{s}(t) \in \mathbb{R}^N$ are the neural spike trains. A spike is generated whenever a neuron's voltage crosses its threshold, T_i, and a spike train is described as a sum of Dirac delta functions, $s_i(t) = \sum_{t_j} \delta(t - t_j)$. Whenever a neuron spikes, its voltage is updated by a reset value($-R_i$), throughout this paper implemented through the diagonal elements of Ω. We now will generally be leaving out the time notation (t) for notional clarity, unless it's important for a specific derivation step.

Dynamical system

We want to control a linear dynamical system. We consider a K-dimensional linear dynamical system, over which we want to directly exert control using spike events. The system is described by the following equation:

$$\mathbf{x} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{s}, \tag{8}$$

where $\mathbf{x} \in \mathbb{R}^K$ represents the state of the system (one value for each dimension of the system), $\mathbf{A} \in \mathbb{R}^{K \times K}$ is a matrix which defines the transformations in **x** that give rise to the dynamics (for instance, an oscillatory behavior), $\mathbf{B} \in \mathbb{R}^{K \times N}$ is a matrix that maps the spike events onto the system, and $\mathbf{s} \in \mathbb{R}^N$ are the spike trains, which represent our control signal. Since it's a linear system, we can solve it analytically to derive the state of the system in a future time step f, starting from the current state $\mathbf{x}_0$.

No spike condition. In the absence of any spiking control signal, we obtain:

$$\mathbf{x}_f = e^{\mathbf{A}f}\mathbf{x}_0,$$

where $\mathbf{x}_0 = \mathbf{x}(t)$ is the current state of the system, $\mathbf{x}_f = \mathbf{x}(t+f)$ is the predicted state after f seconds, and $e^{\mathbf{A}f}$ is the matrix exponential of $\mathbf{A}$ times the future observation f. We introduce the shorthand $e^{\mathbf{A}f} = \mathbf{A}_f$ for ease of writing, resulting in:

$$\mathbf{x}_f = \mathbf{A}_f\mathbf{x}_0.$$

Neuron *i* spikes condition. We now consider the effect of a single neuron i spiking on the current state of the system, and how that affects the state of the system f seconds into the future. For the condition where neuron i spikes, the spike vector $\mathbf{s}$ is zero for all elements and one for the element corresponding to the spiking neuron i. The operation $\mathbf{Bs}$ in Eq.8 results in considering the ith column of matrix $\mathbf{B}$. Here we denote that with $\mathbf{b}_i$. We can write the predicted state of the system after a spike as:

$$\mathbf{x}_f = \mathbf{A}_f(\mathbf{x}_0 + \mathbf{b}_i),$$

where $\mathbf{b}_i$ is the ith column of the matrix $\mathbf{B}$.

Losses

To control the system, we set a target state $\mathbf{z}$ which holds one value to be reached for each dimension of the system. To quantify the distance from this target, we define a loss as the L^2 norm of the difference between the target and the predicted state:

$$L = ||\mathbf{z} - \mathbf{x}_f||^2 + \mu\Gamma(\mathbf{s}) + \alpha||\mathbf{r}||^2,$$

where $\mathbf{z} \in \mathbb{R}^K$ is the target state, μ is a parameter that scales the cost for each spike, $\Gamma(\int_t^{t+\delta t}\mathbf{s}(t)\mathrm{d}t)$ counts the spikes within a short time window (in practice within a simulation timestep), $\mathbf{s}$ are the spike trains, α is a parameter that scales the cost for the neuron's activity, and $\mathbf{r}$ are the neurons' filtered spike traces. In the most general case, we can scale the part of this loss that describes the distance to the target. We introduce the matrix $\mathbf{C}$, which represents a cost on the dimensions of the system. This allows us to introduce a penalty on the values of the system's dimensions, which will weigh on the input to the controller. The loss then becomes:

$$L = (\mathbf{z} - \mathbf{x}_f)^T\mathbf{C}(\mathbf{z} - \mathbf{x}_f) + \mu\Gamma(\mathbf{s}) + \alpha||\mathbf{r}||^2, \tag{9}$$

where $\mathbf{C} \in \mathbb{R}^{K\times K}$ is a cost matrix on each dimensions of the system. Assuming the perspective of a single neuron in our network, we can express this loss in the presence and absence of a spike.

Loss with no spike. In the absence of spikes, the loss of equation (9) becomes:

$$L^{\text{ns}} = (\mathbf{z} - \mathbf{A}_f\mathbf{x}_0)^T\mathbf{C}(\mathbf{z} - \mathbf{A}_f\mathbf{x}_0) + \alpha||\mathbf{r}||^2.$$

The error component in this loss can be renamed to E for ease of writing, obtaining:

$$L^{\text{ns}} = E + \alpha\mathbf{r}^T\mathbf{r}. \tag{10}$$

Loss with neuron *i* spike. In our setup, neurons do not have access to the state of other neurons in the network, but rather compute a comparison between losses independently from each other. Thus, from the perspective of a single neuron *i*, only one spike can be produced at the current time, and the function $\Gamma(\mathbf{s})$ outputs a spike vector $\mathbf{o}$ which contains only zeros except for the *i*th position, which is one. We name this vector $\mathbf{o}_i$. Under this assumption, we can rewrite $\Gamma(\mathbf{s})$ as:

$$\Gamma(\mathbf{s}) = \Gamma\left(\int_t^{t+\delta t} \mathbf{s}(t)\mathrm{d}t\right) = \sum_i^N (\mathbf{o}_i) = 1,$$

the loss in Eq. (9) then becomes:

$$L_i^s = (\mathbf{z} - \mathbf{A}_f(\mathbf{x}_0 + \mathbf{b}_i))^T \mathbf{C}(\mathbf{z} - \mathbf{A}_f(\mathbf{x}_0 + \mathbf{b}_i)) + \mu + \alpha||\mathbf{r} + \mathbf{o}_i||^2.$$

First, we can expand the error term of this loss, obtaining:

$$L_i^s = E - 2(\mathbf{A}_f\mathbf{b}_i)^T\mathbf{C}(\mathbf{z} - \mathbf{A}_f\mathbf{x}_0) + (\mathbf{A}_f\mathbf{b}_i)^T\mathbf{C}(\mathbf{A}_f\mathbf{b}_i) + \mu + \alpha||\mathbf{r} + \mathbf{o}_i||^2.$$

We now also expand the square term $\alpha||\mathbf{r} + \mathbf{o}_i||^2$, obtaining:

$$L_i^s = E - 2(\mathbf{A}_f\mathbf{b}_i)^T\mathbf{C}(\mathbf{z} - \mathbf{A}_f\mathbf{x}_0) + (\mathbf{A}_f\mathbf{b}_i)^T\mathbf{C}(\mathbf{A}_f\mathbf{b}_i) + \mu + \alpha\mathbf{r}^T\mathbf{r} + 2\alpha\mathbf{o}_i{}^T\mathbf{r} + \alpha\mathbf{o}_i{}^T\mathbf{o}_i.$$

Substituting from Eq. (10):

$$L_i^s = L^{ns} - 2(\mathbf{A}_f\mathbf{b}_i)^T\mathbf{C}(\mathbf{z} - \mathbf{A}_f\mathbf{x}_0) + (\mathbf{A}_f\mathbf{b}_i)^T\mathbf{C}(\mathbf{A}_f\mathbf{b}_i) + \mu + 2\alpha\mathbf{o}_i{}^T\mathbf{r} + \alpha\mathbf{o}_i{}^T\mathbf{o}_i.$$

As mentioned previously, $\mathbf{o}_i$ and $\mathbf{o}_i{}^T$ contain only zeros except for the *i*th position, which is one. This means $\mathbf{o}_i{}^T\mathbf{o}_i = 1$. Let us consider the other term, $\mathbf{o}_i{}^T\mathbf{r}$.

$$\mathbf{o}_i{}^T\mathbf{r} = \sum \mathbf{o}_i\mathbf{r} = \mathbf{r}_i$$

Where $\mathbf{r}_i$ indicates a vector containing only zeros except for the *i*th position, which holds the firing rates of neuron *i*. We can write out loss with neuron *i* spiking as:

$$L_i^s = L^{ns} - 2(\mathbf{A}_f\mathbf{b}_i)^T\mathbf{C}(\mathbf{z} - \mathbf{A}_f\mathbf{x}_0) + (\mathbf{A}_f\mathbf{b}_i)^T\mathbf{C}(\mathbf{A}_f\mathbf{b}_i) + \mu + \alpha(2\mathbf{r}_i + 1). \tag{11}$$

Neural variables

As mentioned in Discussion, previous works have treated spiking neurons as systems that can directly encode dynamical variables. Based on this view, they derived a "greedy" spiking rule where a neuron *i* fires whenever this results in a decrease of a cost function, which usually measures the distance between the dynamical variable *x* and its estimate $\hat{x}$, or a target *z*. This is a fully local rule: a neuron's decision to fire depends solely on its current membrane potential and its threshold, which in turn depend on information locally accessible to the neuron at that moment. The neuron does not access global information about the activity of the network. This rule guarantees that each spike emitted by a neuron contributes to improving the representation (or the control) of the dynamical variable, and has therefore a clear interpretation within the computing objective of the network. We start deriving our network based on this simple firing condition:

$$L_i^{\mathrm{s}} \leq L^{\mathrm{ns}}.$$

Substituting from Eq. (10) and (11):

$$L^{\mathrm{ns}} - 2(\mathbf{A}_f\mathbf{b}_i)^T\mathbf{C}(\mathbf{z} - \mathbf{A}_f\mathbf{x}_0) + (\mathbf{A}_f\mathbf{b}_i)^T\mathbf{C}(\mathbf{A}_f\mathbf{b}_i) + \mu + \alpha\mathbf{r}^T\mathbf{r} + \alpha(2\mathbf{r}_i + 1) - L^{\mathrm{ns}} \leq 0,$$
$$-2(\mathbf{A}_f\mathbf{b}_i)^T\mathbf{C}(\mathbf{z} - \mathbf{A}_f\mathbf{x}_0) + (\mathbf{A}_f\mathbf{b}_i)^T\mathbf{C}(\mathbf{A}_f\mathbf{b}_i) + \mu + \alpha(2\mathbf{r}_i + 1) \leq 0.$$

Bringing the negative signed factors to the right side of the inequality, we obtain:

$$\mathbf{b}_i^T\mathbf{A}_f^T\mathbf{C}(\mathbf{z} - \mathbf{A}_f\mathbf{x}_0) \geq \frac{\mathbf{b}_i^T\mathbf{A}_f^T\mathbf{C}(\mathbf{A}_f\mathbf{b}_i) + \mu + \alpha(2\mathbf{r}_i + 1)}{2},$$

We can attribute the left side of this inequality to the threshold of the neuron i:

$$T_i = \frac{\mathbf{b}_i^T\mathbf{A}_f^T\mathbf{C}(\mathbf{A}_f\mathbf{b}_i) + \mu + \alpha(2\mathbf{r}_i + 1)}{2}.$$

As mentioned in Results, we showcase our simulations imposing an asynchronous firing mechanism on the network, so that only one neuron is allowed to fire in each timestep. However, our network derivation generalizes beyond this constraint. We introduce the cost term based on each neuron's firing rate in our losses ($\alpha||\mathbf{r}||^2$ and $\alpha||\mathbf{r} + \mathbf{s}_i||^2$), which is reflected on the neuron's threshold as $\alpha(2\mathbf{r}_i + 1)$, and limits the neuron's ability to spike based on its own past activity. This spike-threshold adaptation mechanism allows us to retain the local aspect of our spiking rule, without causing too many control signals being inputted into the plant, and thus retaining the sparsity of the network's spiking regime. The spiking mechanism has very strong biophysical bases and has been present in previous modeling paradigms, especially in combination with LIF neuron models [23,50].

On the right side, all the time dependent factors can be attributed to the voltage of that neuron:

$$V_i = \mathbf{b}_i^T\mathbf{A}_f^T\mathbf{C}(\mathbf{z} - \mathbf{A}_f\mathbf{x}_0).$$

This gives us a spiking condition for neuron i: $V_i \geq T_i$. If we do not pose any constraint on the dimensions of the system, we would make the matrix $\mathbf{C}$ an identity matrix $\mathbf{I}$, obtaining the voltage and threshold values described in Eq. (4) and Eq. (5). Stepping back from the single neuron perspective, and considering instead the voltages of all the neurons in the network, we obtain:

$$\mathbf{V} = \mathbf{B}^T\mathbf{A}_f^T\mathbf{C}(\mathbf{z} - \mathbf{A}_f\mathbf{x}_0).$$

From there, we pose:

$$\mathbf{G} = \mathbf{B}^T\mathbf{A}_f^T\mathbf{C},$$

obtaining:

$$\mathbf{V} = \mathbf{G}(\mathbf{z} - \mathbf{A}_f\mathbf{x}_0). \tag{12}$$

We now consider the rate of change of $\mathbf{V}$, as well as the rate of change of all time-dependent variables:

$$\dot{\mathbf{V}} = \mathbf{G}(\dot{\mathbf{z}} - \mathbf{A}_f\dot{\mathbf{x}}_0), \tag{13}$$

the change of the current state of the system over time $\dot{\mathbf{x}}_0$ can be expressed by Eq. (8):

$$\dot{\mathbf{x}}_0 = \mathbf{A}\mathbf{x}_0 + \mathbf{B}\mathbf{s}.$$

Substituting this into Eq. (13) yields:

$$\begin{aligned}\dot{\mathbf{V}} &= \mathbf{G}(\dot{\mathbf{z}} - \mathbf{A}_f(\mathbf{A}\mathbf{x}_0 + \mathbf{B}\mathbf{s})) \\ &= \mathbf{G}(\dot{\mathbf{z}} - \mathbf{A}_f\mathbf{A}\mathbf{x}_0 + \mathbf{A}_f\mathbf{B}\mathbf{s}) \\ &= \mathbf{G}\dot{\mathbf{z}} - \mathbf{G}\mathbf{A}_f\mathbf{A}\mathbf{x}_0 - \mathbf{G}\mathbf{A}_f\mathbf{B}\mathbf{s}.\end{aligned} \tag{14}$$

We then add and subtract a $\mathbf{V}$ on the right side of the equation. We leave the $-\mathbf{V}$ as is, while we substitute the $+\mathbf{V}$ using the definition of $\mathbf{V}$ from Eq. (12), obtaining:

$$\dot{\mathbf{V}} = -\mathbf{V} + \mathbf{G}\dot{\mathbf{z}} - \mathbf{G}\mathbf{A}_f\mathbf{A}\mathbf{x}_0 + \mathbf{G}\mathbf{z} - \mathbf{G}\mathbf{A}_f\mathbf{x}_0 - \mathbf{G}\mathbf{A}_f\mathbf{B}\mathbf{s};$$

which can be written as

$$\dot{\mathbf{V}} = -\mathbf{V} + \mathbf{G}(\dot{\mathbf{z}} + \mathbf{z}) - \mathbf{G}\mathbf{A}_f(\mathbf{A} + \mathbf{I})\mathbf{x}_0 - \mathbf{G}\mathbf{A}_f\mathbf{B}\mathbf{s}.$$

We can trace this back to the form of a recurrent network of integrate and fire neurons:

$$\dot{\mathbf{V}} = -\mathbf{V} + \mathbf{G}(\dot{\mathbf{z}} + \mathbf{z}) - \mathbf{F}\mathbf{x}_0 - \Omega\mathbf{s}, \tag{15}$$

where $\mathbf{F} = \mathbf{G}\mathbf{A}_f(\mathbf{A} + \mathbf{I})$, and $\Omega = \mathbf{G}\mathbf{A}_f\mathbf{B} = \mathbf{B}^T\mathbf{A}_f{}^T\mathbf{C}\mathbf{A}_f\mathbf{B}$.

This full network equation describes the changes in $\mathbf{V} \in \mathbb{R}^N$, which represents the membrane potentials. Here, $\mathbf{G} \in \mathbb{R}^{N\times K}$ are the target input weights: forward connections that encode for the target $\mathbf{z}$. Considering the changes in target value over time $\dot{\mathbf{z}}$ as well as the actual value of the target $\mathbf{z}$ allows for the network to trace rapid changes of the target. Only encoding for the current value of $\mathbf{z}$ would still allow to control the system, but with a slower tracing of the target. The matrix $\mathbf{F} \in \mathbb{R}^{N\times K}$ represents the state input weights, that contain the forward connections to encode the current system's state $\mathbf{x}_0$, $\Omega \in \mathbb{R}^{N\times N}$ are the recurrent weights, and $\mathbf{s}$ are the spikes. Note that the diagonal elements of Ω implement a reset in the membrane potential after each spike. Each individual neuron will have its self-reset implemented by a factor of $\boldsymbol{b}_i{}^T\boldsymbol{A}_f{}^T\boldsymbol{C}(\boldsymbol{A}_f\boldsymbol{b}_i)$, making it formally equivalent to the reset ($-R_i$) of LIF neurons. Starting from a greedy spiking condition, we obtained a network of LIF neurons in the form of Eq. (7). Such network has a voltage decay term, forward weights that map the target onto the network, forward weights that encode the system state in input, recurrent connectivity, and spikes, and is able to control our linear dynamical system. For a K-dimensional system to control and a network of N neurons, $\mathbf{B}$ is a $N \times K$ matrix, while $\mathbf{A}_f$ and $\mathbf{C}$ are $K \times K$ matrices. Thus, $\Omega = \mathbf{B}^T\mathbf{A}_f{}^T\mathbf{C}\mathbf{A}_f\mathbf{B}$ is of rank K. Hence, our spiking control solution uses a network with low-rank connectivity (with a rank given by the dimensionality of the controlled system).

Relation to spike coding networks

Works in SCNs use a very similar derivation as the one described in Materials and Methods. In these works, a read-out is usually defined in the following form:

$$\mathbf{y} = \mathbf{D}\mathbf{r}, \tag{16}$$

where $\mathbf{D} \in \mathbb{R}^{K \times N}$ contains the decoding or output weights of all neurons, and $\mathbf{y} \in \mathbb{R}^{K}$ is the K-dimensional network read-out (also sometimes seen as the network state). $\mathbf{r} \in \mathbb{R}^{N}$ represent some form of filtered spike trains, ranging from simple exponential filtering (such that $\dot{\mathbf{r}} = -\lambda\mathbf{r} + \mathbf{s}$), or more complex post-synaptic response kernels. In both cases a loss is optimized such that the output of the network progressively tracks its input, $\mathbf{x}$. A loss between these two quantities can be defined:

$$L = ||\mathbf{x} - \mathbf{D}\mathbf{r}||^2. \tag{17}$$

Much like in our derivation, from this loss voltage, dynamics are derived by considering the rule that spikes should only happen if L(neuron spikes) < L(no spikes), which we took inspiration from for the control case. For works that make use of complex kernels either some prediction of the future state was also employed [36], or a delayed signal was used to explicitly model the difference between past and current state [37].

These derivations are largely similar to ours, with the crucial differences that we consider downstream linear systems of a general form, and that previous SCN works have mainly considered a single read-out given by filtered spike trains. In this sense, our work can be seen as a generalization of standard SCN paradigms (which exert 'control' mainly over their own read-out), which expands the use of similar principles to explicitly control any downstream linear dynamical system.

Filtered spiking controller

For the filtered spiking controller in Fig 7B, we used a more direct extension of SCNs. There, we consider a network that should produce a read-out which matches the optimal control expected from an LQR controller. This means our network spikes to constrain the loss between the network output, and the LQR control signal, such that:

$$L = ||\mathbf{u} - \mathbf{D}\mathbf{r}||^2, \tag{18}$$

where $\mathbf{u} = -\mathbf{K}(\mathbf{x} - \mathbf{z})$ is the control signal as it would be generated by an LQR controller. When defining the LQR signal to approximate, we consider a K-dimensional system controlled by a W-dimensional control signal. The matrix $\mathbf{K} \in \mathbb{R}^{W \times K}$ is then the optimal feedback gain matrix can be found by solving an algebraic Riccati equation, given assumptions on the cost of state deviations and control actuation [51]. This solution ensures that the control law $\mathbf{u} = -\mathbf{K}(\mathbf{x} - \mathbf{z})$ optimally balances tracking performance and control cost when controlling the linear system. Spiking activity that constrains the loss described in Eq. (18) can be achieved by simply using a standard SCN network, while considering $-\mathbf{K}(\mathbf{x} - \mathbf{z})$ as an input instead of just $\mathbf{x}$.

Supporting information

S1 Appendix. Supplementary analyses of spiking control performance and robustness. In the supporting S1 Appendix, we address several aspects of the proposed spiking control framework that complement the main results. We demonstrate how control performance is affected when the assumption of asynchronous firing is relaxed, and show that a spike-threshold adaptation mechanism restores sparse activity and stable control under delayed or synchronous firing conditions. We further discuss alternative measures of control energy, including the total work performed on the controlled system. Finally, we investigate the robustness of the network to voltage noise over a wide range of amplitudes, characterizing the transition from sparse and accurate control to high-activity regimes in which noise degrades control performance. Supplementary figures provide illustrative examples of these effects and their implications for spiking control networks.
(PDF)

Acknowledgments

The authors thank Luc Selen and Yuzhen Qin for advice on the model's design and conceptualization.

Author contributions

Conceptualization: Paolo Agliati, André Urbano, Pablo Lanillos, Nasir Ahmad, Sander Keemink.

Data curation: Paolo Agliati, Sander Keemink.

Formal analysis: Paolo Agliati, Nasir Ahmad, Marcel van Gerven, Sander Keemink.

Funding acquisition: Marcel van Gerven, Sander Keemink.

Methodology: Paolo Agliati.

Project administration: Nasir Ahmad, Marcel van Gerven, Sander Keemink.

Resources: Paolo Agliati, Nasir Ahmad, Marcel van Gerven, Sander Keemink.

Software: Paolo Agliati.

Supervision: Nasir Ahmad, Marcel van Gerven, Sander Keemink.

Validation: Paolo Agliati, André Urbano, Pablo Lanillos, Nasir Ahmad, Sander Keemink.

Visualization: Paolo Agliati, Sander Keemink.

Writing – original draft: Paolo Agliati.

Writing – review & editing: Paolo Agliati, André Urbano, Pablo Lanillos, Nasir Ahmad, Marcel van Gerven, Sander Keemink.

References

1. Koch C. Biophysics of Computation: Information Processing in Single Neurons. Oxford University Press. 2004.
2. Gollisch T, Meister M. Rapid neural coding in the retina with relative spike latencies. Science. 2008;319(5866):1108–11. https://doi.org/10.1126/science.1149639 PMID: 18292344
3. Wolfe J, Houweling AR, Brecht M. Sparse and powerful cortical spikes. Curr Opin Neurobiol. 2010;20(3):306–12. https://doi.org/10.1016/j.conb.2010.03.006 PMID: 20400290
4. Boerlin M, Denève S. Spike-based population coding and working memory. PLOS Computational Biology. 2011;7(2):e1001080. https://doi.org/10.1371/journal.pcbi.1001080
5. Gütig R. To spike, or when to spike?. Current Opinion in Neurobiology. 2014;25:134–9. https://doi.org/10.1016/j.conb.2014.01.004
6. Srivastava KH, Holmes CM, Vellema M, Pack AR, Elemans CPH, Nemenman I, et al. Motor control by precisely timed spike patterns. Proc Natl Acad Sci U S A. 2017;114(5):1171–6. https://doi.org/10.1073/pnas.1611734114 PMID: 28100491
7. Mainen ZF, Sejnowski TJ. Reliability of spike timing in neocortical neurons. Science. 1995;268(5216):1503–6. https://doi.org/10.1126/science.7770778 PMID: 7770778
8. Bazhenov M, Rulkov NF, Fellous JM, Timofeev I. Role of Network Dynamics in Shaping Spike Timing Reliability. Physical Review E. 2005;72(4):041903. https://doi.org/10.1103/PhysRevE.72.041903
9. Cisek P. Beyond the computer metaphor: behavior as interaction. Journal of Consciousness Studies. 1999;6(11):125–42.
10. Ahissar E, Assa E. Perception as a closed-loop convergence process. eLife. 2016;5:e12830. https://doi.org/10.7554/eLife.12830
11. Zhang K. Representation of spatial orientation by the intrinsic dynamics of the head-direction cell ensemble: a theory. J Neurosci. 1996;16(6):2112–26. https://doi.org/10.1523/JNEUROSCI.16-06-02112.1996 PMID: 8604055
12. Eliasmith C. A unified approach to building and controlling spiking attractor networks. Neural Comput. 2005;17(6):1276–314. https://doi.org/10.1162/0899766053630332 PMID: 15901399
13. Seung HS. How the brain keeps the eyes still. Proc Natl Acad Sci U S A. 1996;93(23):13339–44. https://doi.org/10.1073/pnas.93.23.13339 PMID: 8917592
14. Berniker M, Penny S. A normative approach to neuromotor control. Biological Cybernetics. 2019;113(1–2):83–92. https://doi.org/10.1007/s00422-018-0777-7

15. Stagsted R, Vitale A, Binz J, Renner A, Bonde Larsen L, Sandamirskaya Y. Towards neuromorphic control: A spiking neural network based PID controller for UAV. In: Robotics: Science and Systems 2020, 2020. https://doi.org/10.15607/rss.2020.xvi.074
16. Polykretis I, Tang G, Balachandar P, Michmizos KP. A Spiking Neural Network Mimics the Oculomotor System to Control a Biomimetic Robotic Head Without Learning on a Neuromorphic Hardware. IEEE Trans Med Robot Bionics. 2022;4(2):520–9. https://doi.org/10.1109/tmrb.2022.3155278
17. Liu Y, Pan W. Spiking Neural-Networks-Based Data-Driven Control. Electronics. 2023;12(2):310. https://doi.org/10.3390/electronics12020310
18. Schmetterling R, Forni F, Franci A, Sepulchre R. Neuromorphic Control of a Pendulum. IEEE Control Systems Letters. 2024;8(28):1235–40. https://doi.org/10.1109/LCSYS.2024.3409093
19. Anderson BDO, Moore JB. Optimal Control: Linear Quadratic Methods. Courier Corporation. 2007.
20. Boerlin M, Machens CK, Denève S. Predictive coding of dynamical variables in balanced spiking networks. PLoS Comput Biol. 2013;9(11):e1003258. https://doi.org/10.1371/journal.pcbi.1003258 PMID: 24244113
21. Denève S, Machens CK. Efficient codes and balanced networks. Nat Neurosci. 2016;19(3):375–82. https://doi.org/10.1038/nn.4243 PMID: 26906504
22. Huang F, Ching S. Dynamical Spiking Networks for Distributed Control of Nonlinear Systems. In: 2018. 1190–5. https://doi.org/10.23919/ACC.2018.8430996
23. Guo W, Fouda ME, Eltawil AM, Salama KN. Neural coding in spiking neural networks: A comparative study for robust neuromorphic systems. Frontiers in Neuroscience. 2021;15. https://doi.org/10.3389/fnins.2021.638474
24. Slijkhuis FS, Keemink SW, Lanillos P. Closed-Form Control With Spike Coding Networks. IEEE Trans Cogn Dev Syst. 2024;16(5):1677–87. https://doi.org/10.1109/tcds.2023.3320251
25. Moore JJ, Genkin A, Tournoy M, Pughe-Sanford JL, de Ruyter van Steveninck RR, Chklovskii DB. The neuron as a direct data-driven controller. Proc Natl Acad Sci U S A. 2024;121(27):e2311893121. https://doi.org/10.1073/pnas.2311893121 PMID: 38913890
26. Yang T. Impulsive Control. IEEE Transactions on Automatic Control. 1999;44(5):1081–3. https://doi.org/10.1109/9.763234
27. Yang Z, Xu D. Stability Analysis and Design of Impulsive Control Systems With Time Delay. IEEE Trans Automat Contr. 2007;52(8):1448–54. https://doi.org/10.1109/tac.2007.902748
28. Raff T, Allgower F. Observers with impulsive dynamical behavior for linear and nonlinear continuous-time systems. In: 2007. 4287–92. https://doi.org/10.1109/CDC.2007.4434613
29. Wang X, Li C, Huang T, Pan X. Impulsive control and synchronization of nonlinear system with impulse time window. Nonlinear Dyn. 2014;78(4):2837–45. https://doi.org/10.1007/s11071-014-1629-1
30. Ouyang D, Shao J, Jiang H, Nguang SK, Shen HT. Impulsive synchronization of coupled delayed neural networks with actuator saturation and its application to image encryption. Neural Netw. 2020;128:158–71. https://doi.org/10.1016/j.neunet.2020.05.016 PMID: 32446193
31. Petri E, Scheres KJA, Steur E, Heemels WPMH. Analysis of a simple neuromorphic controller for linear systems: A hybrid systems perspective. In: arXiv, 2024. https://arxiv.org/abs/2409.06353
32. Huang F, Riehl J, Ching S. Optimizing the dynamics of spiking networks for decoding and control. In: 2017. 2792–8. https://doi.org/10.23919/ACC.2017.7963374
33. Felgenhauer U. On stability of bang-bang type controls. SIAM Journal on Control and Optimization. 2003;41(6):1843–67. https://doi.org/10.1137/S0363012901399271
34. Brendel W, Bourdoukan R, Vertechi P, Machens CK, Denève S. Learning to represent signals spike by spike. PLoS Comput Biol. 2020;16(3):e1007692. https://doi.org/10.1371/journal.pcbi.1007692 PMID: 32176682
35. Calaim N, Dehmelt FA, Gonçalves PJ, Machens CK. The geometry of robustness in spiking neural networks. eLife. 2022;11:e73276. https://doi.org/10.7554/eLife.73276
36. Schwemmer MA, Fairhall AL, Denéve S, Shea-Brown ET. Constructing Precisely Computing Networks with Biophysical Spiking Neurons. J Neurosci. 2015;35(28):10112–34. https://doi.org/10.1523/JNEUROSCI.4951-14.2015 PMID: 26180189
37. Zeldenrust F, Gutkin B, Denéve S. Efficient and robust coding in heterogeneous recurrent networks. PLoS Comput Biol. 2021;17(4):e1008673. https://doi.org/10.1371/journal.pcbi.1008673 PMID: 33930016
38. Gu DW, Petkov PH, Konstantinov MM. Robust control of a mass-damper-spring system. Robust control design with MATLAB. London: Springer. 2005. p. 101–62. https://doi.org/10.1007/1-84628-091-5_8
39. Vincent TL, Joshi SP, Lin YC. Positioning and active damping of spring-mass systems. Journal of Dynamic Systems, Measurement, and Control. 1989;111(4):592–9. https://doi.org/10.1115/1.3153099
40. Ahmadvand R, Sharif SS, Banad YM. Neuromorphic robust framework for concurrent estimation and control in dynamical systems using spiking neural networks. arXiv. 2023. https://arxiv.org/abs/2310.03873
41. Byadarhaly KV, Perdoor MC, Minai AA. A modular neural model of motor synergies. Neural Networks. 2012;32:96–108. https://doi.org/10.1016/j.neunet.2012.02.003
42. Pazzaglia A, Bicanski A, Ferrario A, Arreguit J, Ryczko D, Ijspeert A. Balancing central control and sensory feedback produces adaptable and robust locomotor patterns in a spiking, neuromechanical model of the salamander spinal cord. PLoS Comput Biol. 2025;21(1):e1012101. https://doi.org/10.1371/journal.pcbi.1012101 PMID: 39836708

43. Eilers L, Stapmanns J, Dias C, Pfister JP. On the Stability of Event-Based Control with Neuronal Dynamics [Preprint]. arXiv:2511.18015; 2025. https://arxiv.org/abs/2511.18015
44. Qin Y, El-Gazzar A, Bassett DS, Pasqualetti F, Van Gerven M. In: 2024. 583–8. https://doi.org/10.1109/CDC56724.2024.10886408
45. Podlaski WF, Machens CK. Approximating Nonlinear Functions With Latent Boundaries in Low-Rank Excitatory-Inhibitory Spiking Networks. Neural Comput. 2024;36(5):803–57. https://doi.org/10.1162/neco_a_01658 PMID: 38658028
46. Mancoo A, Keemink SW, Machens CK. Understanding spiking networks through convex optimization. Advances in Neural Information Processing Systems. 2020;33:8824–35.
47. Yeşildirek A, Lewis FL. Feedback linearization using neural networks. Automatica. 1995;31(11):1659–64. https://doi.org/10.1016/0005-1098(95)00078-B
48. Baratchart L, Pomet J-B. On local linearization of control systems. J Dyn Control Syst. 2009;15(4):471–536. https://doi.org/10.1007/s10883-009-9077-9
49. Nardin M, Phillips JW, Podlaski WF, Keemink SW. Nonlinear computations in spiking neural networks through multiplicative synapses. Peer Community In Circuit Neuroscience. 2021;:100003. https://doi.org/10.24072/pci.cneuro.100003
50. Jones DL, Johnson EC, Ratnam R. A stimulus-dependent spike threshold is an optimal neural coder. Front Comput Neurosci. 2015;9:61. https://doi.org/10.3389/fncom.2015.00061 PMID: 26082710
51. Brunton SL, Kutz JN. Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control. 1st ed. Cambridge University Press. 2019. https://doi.org/10.1017/9781108380690