

# Learning inverse kinematics using neural computational primitives on neuromorphic hardware

**Journal Article****Author(s):**

Zhao, Jingyue; Monforte, Marco; Indiveri, Giacomo ; Bartolozzi, Chiara; Donati, Elisa

**Publication date:**

2023-10-26

**Permanent link:**

<https://doi.org/10.3929/ethz-b-000655693>

**Rights / license:**

[Creative Commons Attribution 4.0 International](#)

**Originally published in:**

npj Robotics 1(1), <https://doi.org/10.1038/s44182-023-00001-w>

## ARTICLE OPEN



# Learning inverse kinematics using neural computational primitives on neuromorphic hardware

Jingyue Zhao<sup>1</sup>✉, Marco Monforte<sup>2</sup>, Giacomo Indiveri<sup>1</sup>, Chiara Bartolozzi<sup>2</sup> and Elisa Donati<sup>1,2</sup>

Current low-latency neuromorphic processing systems hold great potential for developing autonomous artificial agents. However, the variable nature and low precision of the underlying hardware substrate pose severe challenges for robust and reliable performance. To address these challenges, we adopt hardware-friendly processing strategies based on brain-inspired computational primitives, such as triplet spike-timing dependent plasticity, basal ganglia-inspired disinhibition, and cooperative-competitive networks and apply them to motor control. We demonstrate this approach by presenting an example of robust online motor control using a hardware spiking neural network implemented on a mixed-signal neuromorphic processor, trained to learn the inverse kinematics of a two-joint robotic arm. The final system is able to perform low-latency control robustly and reliably using noisy silicon neurons. The spiking neural network, trained to control two joints of the iCub robot arm simulator, performs a continuous target-reaching task with 97.93% accuracy, 33.96 ms network latency, 102.1 ms system latency, and with an estimated power consumption of 26.92  $\mu$ W during inference (control). This work provides insights into how specific computational primitives used by real neural systems can be applied to neuromorphic computing for solving real-world engineering tasks. It represents a milestone in the design of end-to-end spiking robotic control systems, relying on event-driven sensory encoding, neuromorphic processing, and spiking motor control.

*npj Robotics* (2023)1:1; <https://doi.org/10.1038/s44182-023-00001-w>

## INTRODUCTION

Neuromorphic engineering aims to develop adaptive and efficient artificial neural processing systems by implementing models of neural computation and brain-inspired processing mechanisms with electronic circuits<sup>1,2</sup>. The emulation of neural and synaptic dynamics in compact and energy-efficient mixed-signal circuits supports spike-based information encoding and processing with fast response and low-power consumption<sup>3</sup>. Spike-based neuromorphic architectures are therefore well suited for embedded low-power applications, such as autonomous robotics, prosthetics, and always-on wearable biomedical devices<sup>4–8</sup>. Within this context, the research community has developed neuromorphic modules for sensing<sup>9</sup>, perception<sup>10,11</sup>, and decision making<sup>12</sup> that are exploited in robotic applications<sup>8,13,14</sup>. However, research in neuromorphic motor control is still lagging behind, hindering the design of a fully autonomous embodied neuromorphic agent that would feature ultra-low latency and power consumption. Spiking low-level control of single joints was first demonstrated in simulation<sup>15</sup> using pulse-frequency modulation (PFM), then implemented on neuromorphic hardware<sup>16</sup>. Spiking neural network (SNN) on-chip implementations of the classical proportional-integral-derivative controller (PID) were then proposed<sup>17–19</sup>. implemented a spike-based PID controller using PFM by developing basic spike-processing modules and interfaces on field programmable gate array (FPGA), but focused less on the algorithms for coordination between joints and on its learning. Therefore, the missing piece in neuromorphic implementations is a high-level controller that coordinates multiple joints to drive the end-effector to perform specific tasks, such as target-reaching or trajectory tracking. To this aim, the inverse kinematics of a robot, that is, the relationship between joint configurations and corresponding target spatial coordinates of the end-effector needs to be found.

Analytical methods to solve the inverse kinematics involve deriving an explicit mathematical model of the robotic system based on simplified assumptions and parameters, which can differ from the real system and require iterative calibration. For some robots (e.g., with high degrees of freedom), deriving a closed-form solution is complicated or even impossible. Numerical methods (e.g., using the Jacobian inverse) rely on iterations to find an approximation of the solution, involving computationally intensive optimizations. Both the analytical and numerical methods are limited to known systems and cannot adapt to unknown situations. With the development of machine learning, learning-based methods are used to find an approximate mapping between high-level control parameters (i.e., end-effector positions) and low-level joint configuration<sup>20–22</sup>. The advantage of model-free, data-driven, learning-based methods is their intrinsic adaptation to the robotic plant, comprising unmodelled non-idealities that usually require calibration.

Inverse kinematics can be learned by a feedforward SNN endowed with spike-timing dependent plasticity (STDP) for moving the end-effector in the desired spatial direction<sup>23</sup>. Spiking reinforcement learning enables the SNN to learn the mapping between muscle lengths and muscle activation required to reach a fixed target in 2D space<sup>24</sup>. In ref. <sup>25</sup>, an SNN learns to move a robotic arm in three directions: left-right, up-down, and far-near, by combining simple motor actions in a hierarchical fashion to perform complex movements, rather than explicitly learning the inverse kinematics. However, most of those SNNs are still implemented in simulation and are not directly transferable onto neuromorphic hardware. The reason is that network parameters such as weights and connectivity probabilities are expressed as floating-point variables that do not meet the limited bit precision imposed by hardware implementations<sup>23,26,27</sup>. Moreover, some

<sup>1</sup>Institute of Neuroinformatics, University of Zurich and ETH Zurich, Winterthurerstrasse 190, 8057 Zurich, Switzerland. <sup>2</sup>Event-Driven Perception for Robotics, Istituto Italiano di Tecnologia, Via San Quirico 19D, 16163 Genova, Italy. ✉email: [jingyue-zhao@foxmail.com](mailto:jingyue-zhao@foxmail.com)

methods require custom neural models with fine-tuned neural and synaptic parameters that can hardly be reproduced on hardware<sup>24,26</sup>. A recent example<sup>28</sup> of using SNN for online learning of the inverse kinematics on the digital neuromorphic processor Loihi<sup>29</sup> is based on the Neural Engineering Framework (NEF)<sup>30</sup>. The control variables (e.g., target and current joint configurations, target end-effector position and the current distance to it, etc.) are represented by neuron ensembles, and the weights are modulated by an error-driven learning rule. NEF is also used to implement force control on the mixed-mode neuromorphic processor Neurogrid<sup>31</sup> with populations of spiking neurons that translate the desired forces of the end-effector and current joint angles into torque commands for the joints. A comprehensive learning-based method using SNNs for solving inverse kinematics<sup>32</sup> extends<sup>28</sup> and compares the online-learning method in ref. <sup>28</sup> with an offline Stochastic Gradient Descent (SGD)-based algorithm. The online-learning method shows the advantage of faster network convergence and more successful-reaching end-effector positions. The NEF-based methods<sup>28,31,32</sup> focus less on using brain-inspired neural circuits and learning algorithms to solve the inverse kinematics problem and perform some crucial non-spiking processing outside the neuromorphic hardware. The offline training of the simulated architectures before their conversion to the desired SNN (with a number of parameters ranging from 5000 to 300,000) may still require large-scale computing infrastructure and high energy consumption. In addition, the high firing rate of the SNNs during inference further increases the power budget. In general, the inference time of the SNNs in ref. <sup>32</sup> is long (from 400 ms to 3.8 s).

Multi-joint control, hence, can be solved or learned using SNNs, but there is still a need for their deployment on neuromorphic hardware toward the implementation of end-to-end neuromorphic robotic platforms<sup>13</sup> with ultra-low latency and power consumption. To this end, we present an SNN trained using a mixed-signal analog-digital neuromorphic processor—the Dynamic Neuromorphic Asynchronous Processors 1 (DYNAP-SE1)<sup>33</sup>—which controls two joints of the iCub robot<sup>34</sup> arm to perform target-reaching and trajectory tracking. For the first time, we trained the on-chip SNN weights with a computer in the loop, which allows for taking into account the system non-idealities and implementing any possible learning rule based on spike timings. The system architecture is based on neural populations that encode the desired input Cartesian coordinates of the end-effector and the corresponding joint angles. The relationship between them is learned by two hidden populations connected by trainable synapses. To learn the correct mapping, we introduced a disinhibition mechanism inspired by the basal ganglia<sup>35–37</sup> and recurrent connectivity that selects the closest possible configuration of the joints among the multiple possible solutions, based on the current robot state. This work adds to evidence of the relevance of using neural computational primitives to solve the complex problem of finding the correct solution among multiple possible solutions. This was first demonstrated and implemented on neuromorphic hardware in the context of false correspondences in stereo vision<sup>38,39</sup> and is now demonstrated in the context of the coordination of multiple joints in a motor task.

The system has been trained through motor babbling, imposing joint angles with a random order, and measuring the corresponding Cartesian positions. The training uses the ground truth known joint angles as the teaching signal. The same supervised learning procedure could be used in learning by demonstration, where a human teacher positions the robot and, therefore, the Cartesian position and joint angles are both known. Motor babbling is an approach to generate data to train data-driven learning methods that can solve the inverse kinematics as a regression problem and find an approximate mapping between high-level control

parameters (e.g., end-effector positions) and low-level joint configuration<sup>20–22,40,41</sup>.

Although this work solves a relatively simple problem, it represents an important milestone for neuromorphic robotics, towards scaling up to more complex behaviors – that can also adapt over time and to different conditions—using motor babbling as a self-supervised learning technique, neural computational primitives, and meta-learning.

## RESULTS

In this work, the correspondence between the iCub's end-effector Cartesian position  $(x, y)$  and the shoulder pitch and elbow joints  $(\theta_1, \theta_2)$  (Fig. 1A) is first learned and then used to drive the end-effector to reach various target positions  $(x^*, y^*)$  sequentially in 2D space (Fig. 1B).

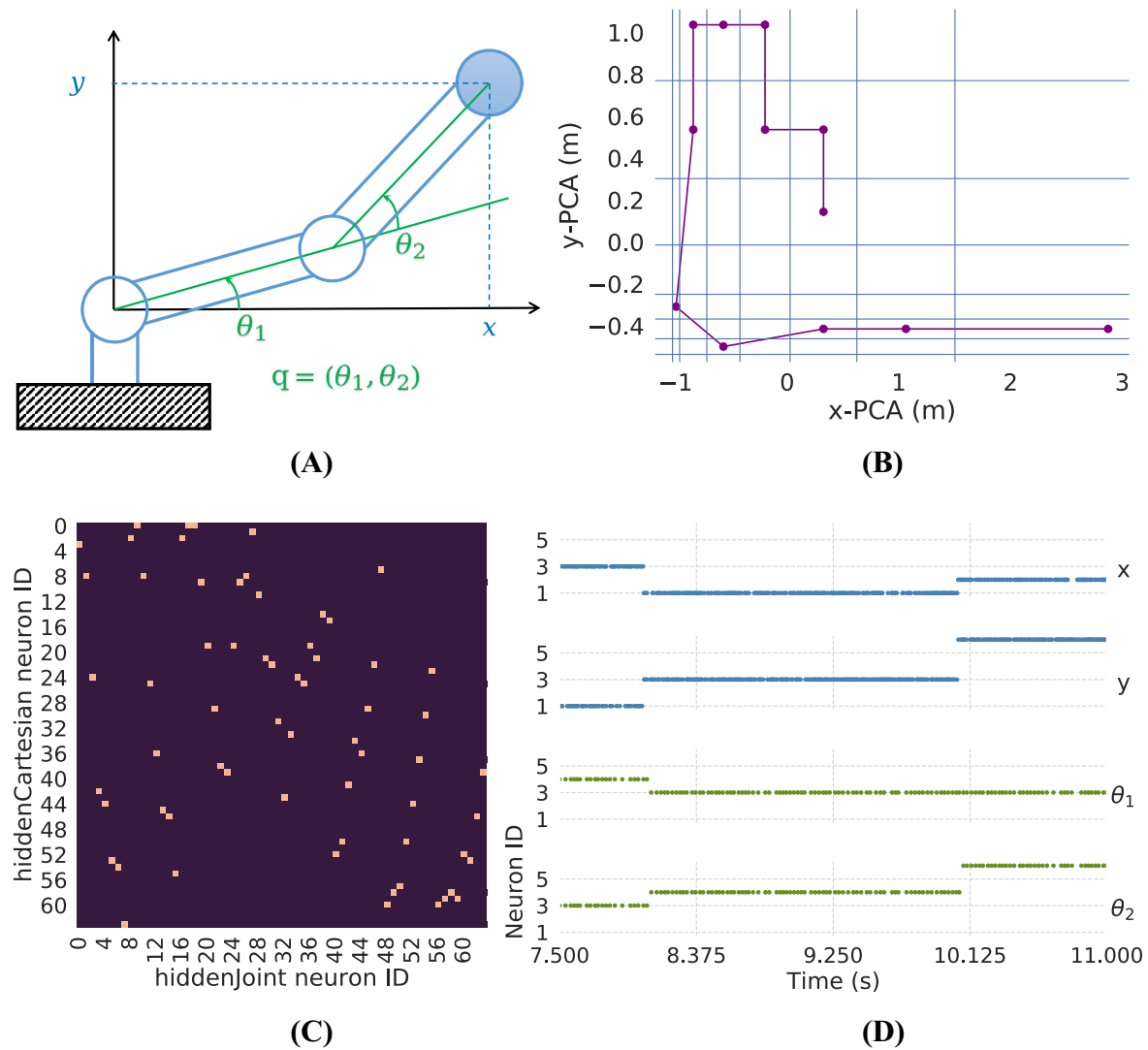
The inverse kinematics is learnt through spike-driven synaptic learning in the form of a weight matrix that connects two hidden populations that represent the Cartesian coordinates (*hiddenCartesian*) of the 2D working space of the robot and the joint space (*hiddenJoint*). The learnt weight matrix representing the connections from the presynaptic neurons in *hiddenCartesian* layer (vertical axis) to the postsynaptic neurons in *hiddenJoint* (horizontal axis) one is shown in Fig. 1C, where each dot represents the potentiated synapse between the corresponding pair of (pre, post) neurons. Given a target position—encoded as one-hot population code of the input  $x$  and  $y$  populations—the solver module (i.e., the trained SNN) uses learnt connectivity to drive the output neurons encoding  $(\theta_1, \theta_2)$ , such that only those that represent the correct solution are active, as shown in Fig. 1D. The neurons' activity in the output populations  $\theta_1$  and  $\theta_2$  actuates the shoulder and elbow joints to move the arm end-effector continuously. The spikes in the raster plots can look continuous when the time range of the  $x$ -axis is large because the intervals between the spikes are much smaller compared to the entire time range in Figs. 1D, 3A and S1.

The resulting solver module shows 97.93% accuracy, 33.96 ms on-chip network latency, 102.1 ms system latency, and 26.92  $\mu$ W on-chip power consumption in the continuous target-reaching task.

The response time of the controller to produce corrective actions is a crucial factor for the functioning of the system. It adds to the robot's actuation latency, defining a maximum target update rate for the robot end-effector to follow the target online.

### Role of disinhibition during training

Disinhibitory input connections to the two hidden populations construct selective and stable firing patterns in both the presynaptic (*hiddenCartesian*) and postsynaptic (*hiddenJoint*) neurons for each training sample, making the noisy silicon neurons learn the correct inverse kinematics (weight matrix) robustly over time through STDP. During the inference, both the disinhibition from the input signal to *hiddenCartesian* and the well-trained inter-populations connections are crucial for the SNN to generate reliable joint commands using noisy neurons for the control. The neurons' instantaneous firing rates after 400 ms of stimulation with the teaching signals corresponding to the target end-effector position  $(x, y) = (3, 0)$  are shown in Fig. 2A, B. The neurons firing rates are tracked using exponentially-decaying spiking traces (see Eq. (3)) with a time constant of 100 ms. As expected, neurons 3 in  $x$  and 0 in  $y$  are active. The former stimulates row 3 with direct excitation, while the latter disinhibits column 0 of *hiddenCartesian*. Only  $y$  gate neuron 0 is silenced, so that *hiddenCartesian* neurons in column 0 are not inhibited (turned off) by the inactive gate and can fire when stimulation comes from excitatory synapses. In the meantime, all the other  $y$  gate neurons keep firing to suppress the



**Fig. 1** Learning to control a two-joint arm with an on-chip spiking neural network (SNN). **A** The abstract model of a two-joint arm. The two hollow blue circles represent the controllable joints and the solid blue circle is the end-effector. **B** Test trajectory of a continuous target-reaching task. The blue lines are the boundaries of discrete end-effector space, while the purple dots are the target positions the arm needs to reach one by one, starting from the bottom-right point. **C** After training, the learned inverse kinematics can be represented with a classical connectivity matrix, where the vertical axis shows presynaptic neuron (*hiddenCartesian*) IDs and the horizontal one shows postsynaptic neuron (*hiddenJoint*) IDs. Each orange dot represents one connection between a pair of *hiddenCartesian* and *hiddenJoint* neurons. The sparsity of the connections hints at future possibilities of pruning the network to either make the model more compact or reuse some neurons to increase the encoding resolution or to perform other tasks. **D** Neurons activity of the SNN during the control task. The blue spikes are from the input populations, which encode target end-effector positions ( $x^*$ ,  $y^*$ ) while the green ones generated by the output populations are decoded as joint commands ( $\theta_1$ ,  $\theta_2$ ) that are used to drive the joints.

corresponding columns in *hiddenCartesian*, which can hardly fire even with the input coming from  $x$ . A similar disinhibition process also happens in *hiddenJoint* where neuron (4,2) encodes the target joint configuration  $(\theta_1, \theta_2) = (4, 2)$ . The correlation between the presynaptic neuron (*hiddenCartesian* neuron (3,0)) and postsynaptic neuron (*hiddenJoint* neuron (4,2)) can be learnt through triplet-STDP.

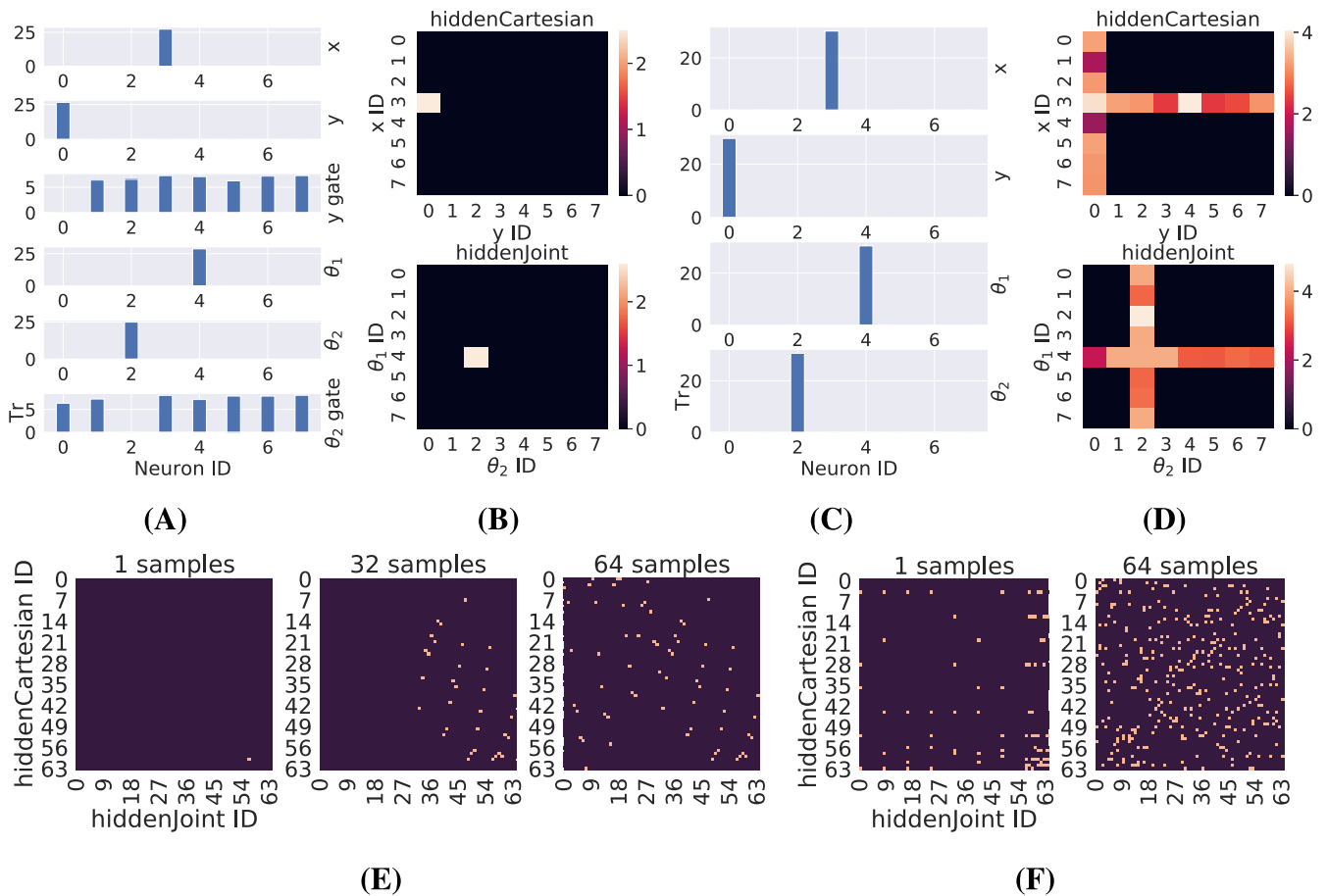
If disinhibition is replaced by direct excitation, the firing patterns in *hiddenCartesian* and *hiddenJoint* will become chaotic, as in Fig. 2C, D. The chaotic firing patterns in the pre- and postsynaptic populations drive the network to learn undesired connections, which in turn leads to more noisy neurons activity over time. As a result, the learnt weight matrix becomes very noisy (Fig. 2F) and fails to form the specific connectivity pattern that encodes the inverse kinematics. This vicious circle can be broken via disinhibition by creating selective and clean firing activities

during training. Figure 2E shows the potentiated connections resulting from disinhibition-driven synaptic plasticity.

During training, triplet-STDP supports the absence of potentiation at low frequencies and increased potentiation with frequency, that cannot be obtained with simple STDP. Both mechanisms eliminate unwanted synapse growth caused by low-firing noisy postsynaptic neurons in *hiddenJoint*.

#### Neurons activity and joints readout during the control task

The trained SNN with disinhibition drives the end-effector of the robotic arm to reach the 12 target positions in the trajectory of Fig. 1B. A Supplementary Video showing the simulated movements of the iCub robot during one trial of the control experiment is available.



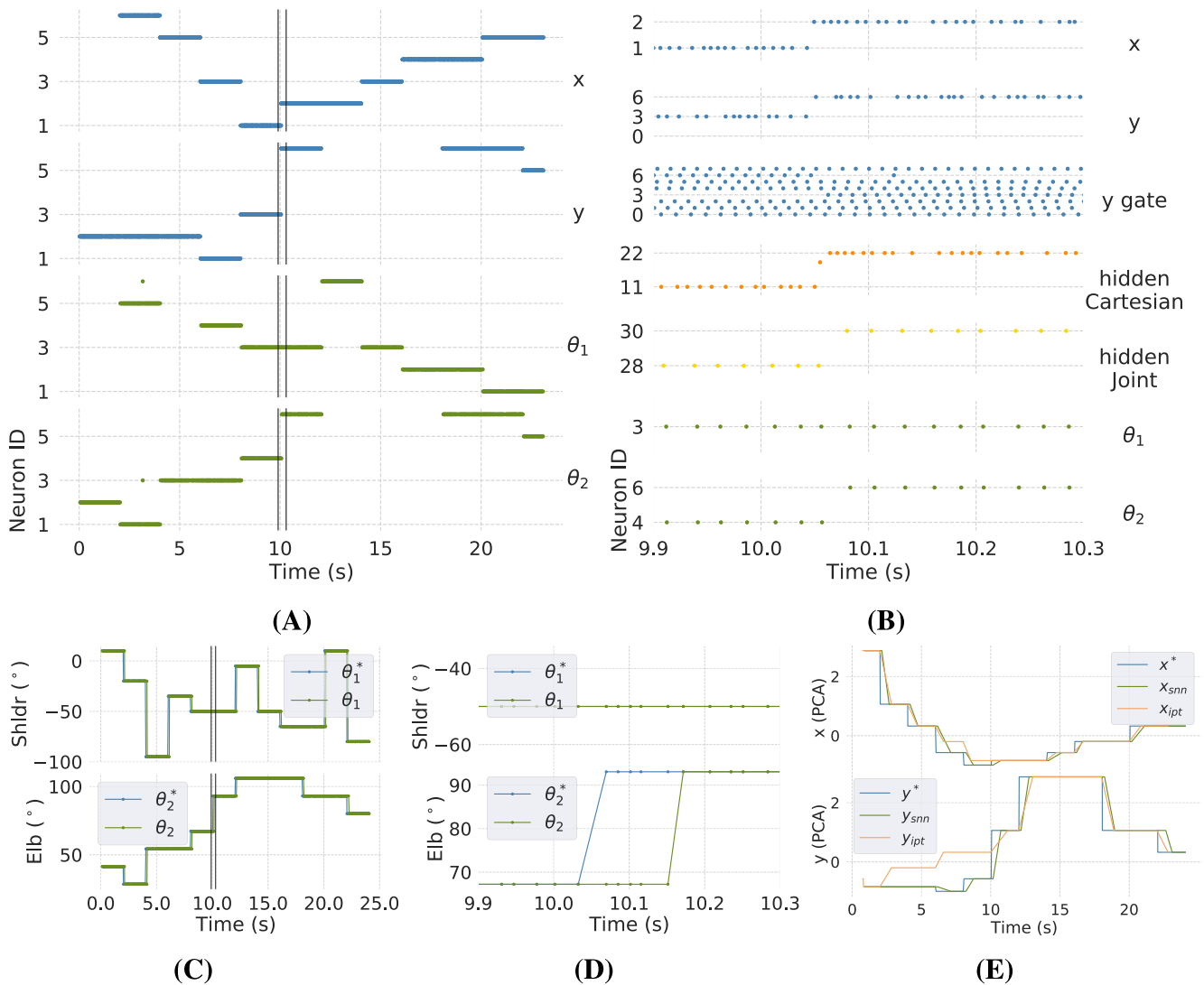
**Fig. 2 Function of disinhibition in training.** **A, B** and **C, D** report instantaneous firing rates of the neurons with and without disinhibition mechanism, respectively, after the learning of a sample where  $(x, y) = (3, 0)$ ,  $(\theta_1, \theta_2) = (4, 2)$ . **A, C:** Firing rates of  $x$ ,  $y$ ,  $y$  gate,  $\theta_1$ ,  $\theta_2$ , and  $\theta_2$  gate. Tr is the instantaneous firing states of the neurons tracked by exponentially-decaying spiking traces. **B, D:** Firing rates of *hiddenCartesian* and *hiddenJoint* in 2D layout. **E** Weight development over time during learning of the inverse kinematics with disinhibition mechanism. The vertical and horizontal axes represent presynaptic (*hiddenCartesian*) and postsynaptic (*hiddenJoint*) neuron IDs, respectively. Each orange dot indicates a connection is constructed between the corresponding pair of *hiddenCartesian* and *hiddenJoint* neurons. **F** Noisy weight matrix learnt without disinhibition after the first and last training samples, respectively.

The SNN running on DYNAP-SE1 receives time-varying target end-effector positions as input stimulation and generates joint solutions continuously. Neurons activity of the input and output populations during the control task is shown in the raster plot Fig. 3A. The spikes of output neurons are read out and decoded periodically into joint position commands that are sent to the low-level controllers of the simulator of the iCub robot (iCubSim<sup>42</sup>), actuating the shoulder and elbow joints. During the actuation, the SNN maintains a steady state and keeps the output joint solution constant until the motion is completely performed. When the target changes, the network converges to the new solution after 34 ms on average (Fig. 3B).

The joint command trajectories decoded from the SNN match the desired ones, generated using the dataset recorded during motor babbling, as shown in Fig. 3C. The solver module takes about 102 ms (see Fig. 3D) to generate a new joint configuration given a new target input, due to the latency of sequential setup of the spike generators and communication interface between DYNAP-SE1 and iCubSim. Fig. 3E compares the desired end-effector trajectories to those driven by the spiking controller and a nonlinear optimizer named *IpOpt*<sup>43</sup>. *IpOpt* is a C++ package for solving nonlinear problems (the inverse kinematics here), which generates joint configurations given target end-effector positions in the Cartesian space. Wrapped by an iCub control library API, it receives as input the desired pose in the Cartesian space, the

initial joints configuration, a preferred joints configuration to exploit the arm redundancy (e.g., elbow up or down), and a priority preference on either the position or the orientation of the end-effector to speed up the computation (limited at a low level by an error positioning threshold and a maximum number of iterations for the solver), outputting the desired joints configuration. Under the control of the SNN, the end-effector reaches all correct Cartesian positions with a latency that depends on the iCubSim low-level controllers, while *IpOpt* is unable to reach four target positions (testing samples 2 to 5) because its control accuracy fails to fit in the arm space discretization (see Fig. 1B).

When a new desired end-effector position is sent to the network, both the spatiotemporal patterns in the neural populations and the resulting joint commands decoded from output spikes change. Figure 3B and D show the transient behavior of the SNN during the target transition phase. Disinhibitory connections from  $y$  to *hiddenCartesian* create selective firing patterns in *hiddenCartesian* and thus in *hiddenJoint*, which generates reliable joint commands. In Fig. 3B, when the input changes (at 10.051 s), the  $x$  and  $y$  populations switch between active neurons (from #1 to #2 and from #3 to #6, respectively), which start inhibiting neuron #6 and stop inhibiting neuron #3 in the  $y$  gate population, that in turn start inhibiting neuron #11 and stop inhibiting neuron #22 in the *hiddenCartesian* population. The new active *hiddenCartesian* neuron stimulates its postsynaptic *hiddenJoint* neuron via



**Fig. 3** Neurons activity and joints readout during control. Shldr shoulder, Elb elbow. **A** Neurons activity during one control trial. The raster plot shows the spikes generated by  $x$  and  $y$  in blue and spikes of  $\theta_1$  and  $\theta_2$  in green. The two vertical lines mark the transition phase of the SNN when the input ( $x, y$ ) changes (zoomed in **B**). **B** Neurons activity during the target transition phase. The vertical axis only shows the ID labels of the firing or inhibited neurons. The red lines mark the first spike in each layer of the SNN that encodes the new state, indicating the convergence of that layer. The latency from input to *hiddenCartesian*, *hiddenCartesian* to *hiddenJoint* and *hiddenJoint* to output layer is 13, 16, and 3 ms, respectively. **C** Actual and ground truth joint commands during one control experiment. The joint commands ( $\theta_1, \theta_2$ ) (green) are continuously decoded from the output  $\theta_1, \theta_2$  spikes in **A**. The ground truth joint solutions ( $\theta_1^*, \theta_2^*$ ) correspond to the desired end-effector positions ( $x, y$ ) represented in the input populations (blue). The transition phase of the joint output marked by the two vertical black lines is zoomed in **D**. **D** Decoded and desired joint commands during the target transition. **E** Desired end-effector positions ( $x^*, y^*$ ) (blue) are fed as input to both the neuromorphic kinematics solver and a classical solver *IpOpt* during one control trial. Positions ( $x_{snn}, y_{snn}$ ) (green) and ( $x_{ipt}, y_{ipt}$ ) (orange) are the resulting trajectories driven by the spiking controller and *IpOpt*, respectively.

the connections learnt during the training process. The firing *hiddenJoint* neuron #30 activates new output neurons in  $\theta_1$  and  $\theta_2$  at about 10 s, which generate the new joint commands. The change of neuron activity from the input to the output layer takes 32 ms. The resulting state transition of joint solutions is shown in Fig. 3D, which takes about 102 ms due to the systematic delay mentioned above.

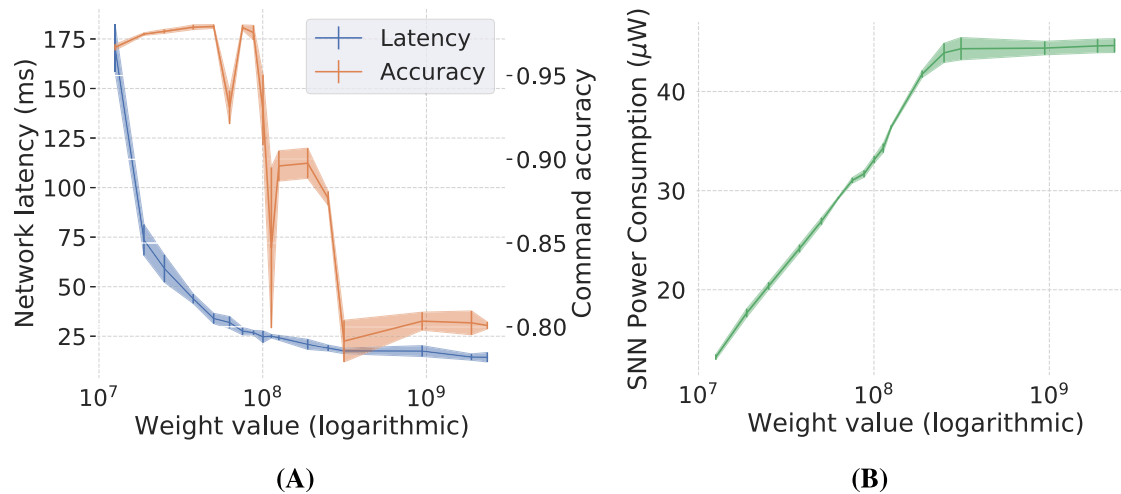
#### Latency and accuracy trade-off

To quantify the speed-accuracy trade-off, accuracy, and latency are measured with different key network parameters. Latency depends on the speed of propagation of spiking activity across the different layers of the network. When a new target position ( $x, y$ ) is sent as input to the SNN, the network needs time to

converge to the new solution. This is visible in the raster plots, as the previously active neurons (corresponding to the previous target position) in the output population stop firing and neurons corresponding to the new solution ( $\theta_1, \theta_2$ ) become active. In Fig. 3B, the delay from the input to the output populations takes 13, 16, and 3 ms, respectively, layer by layer, among which the transmission from *hiddenCartesian* to *hiddenJoint* consumes the longest time. The weight of the inter-population connections between *hiddenCartesian* and *hiddenJoint* can speed up the transition, at the cost of reduced accuracy.

Figure 4A shows the average network latency and accuracy for different weight values. When the synapses between *hiddenCartesian* and *hiddenJoint* populations are weak, it takes more input spikes—hence more time—to elicit activity in the postsynaptic neurons. The corresponding lower activation of the *hiddenJoint*





**Fig. 4 Latency, accuracy, and power consumption. A** Average network latency and accuracy for different weights. The average transition time of all target changes in Fig. 3A was calculated across five repetitions. We computed the accuracy by comparing the decoded joint commands with the desired ones in the experiment of Fig. 3C. The network output is correct only if the two vectors are equal at a certain time point, and the accuracy is the ratio of the right commands to the total ones. **B** Average power consumption of the SNN during the control task with standard deviations marked as vertical lines.

neurons, in turn, slows down the activity of the output populations  $\theta_1$  and  $\theta_2$ . On the contrary, stronger weights make the SNN react faster to the input change, but with an accuracy drop due to increased instability of the Winner-take-all (WTA) (see Fig. S1). Each of the active neurons in *hiddenJoint* represents one of the possible correct joint configurations for a given input. If one of the two populations  $\theta_1$  and  $\theta_2$  only has one firing neuron, and the other has two stable winners (see testing samples 4, 6, 9, and 11 in Fig. S1), the decoded  $(\theta_1, \theta_2)$  is correct because both combinations are correct. However, if in both  $(\theta_1$  and  $\theta_2)$  populations there is more than one active neuron, only part of the active configurations are correct, decreasing the accuracy of the controller.

The controller can achieve 97.93% of accuracy with a latency of 33.96 ms, where the weight was chosen as the most balanced setup for the trajectory tracking task (see results in Fig. 3A–E). Smaller weights yield more stable neurons activity in *hiddenJoint* but not higher output accuracy because the transition to input change is too slow and the decoded joint solutions cannot switch to the new desired position quickly. With stronger weights, the controller can achieve a fast average reaction time of 14.44 ms while maintaining reasonable accuracy (over 80%).

As a comparison to classical robotics, we measured the computational time for solving the inverse kinematics problem as done in the iCub Cartesian control module, using the nonlinear optimization solver *IpOpt* to generate a new joints configuration given a desired end-effector position as the input of the corresponding C++ function. The average latency is 114.57 ms (99 to 156 ms) which depends on the central processing unit (CPU) real-time performance of the laptop.

The accuracy of *IpOpt* cannot be quantified since its ground truth joint solutions are unknown. However, *IpOpt* fails to reach some target end-effector positions (testing samples) in Fig. 3E because the accuracy of its generated joint solutions is not enough to guarantee that the end-effector falls into the expected grid in Fig. 1B. The failed testing samples are No. 2 to 5, which correspond to the purple dots No. 2 to 5 in Fig. 1B counting from the bottom-right one to the left, i.e., the left four dots at the bottom of the trajectory. The y coordinates of the four positions are close to each other, and the discrete partition along y is narrow in this area. Therefore, if the generated joint configuration is not accurate enough, the resulting end-effector position cannot

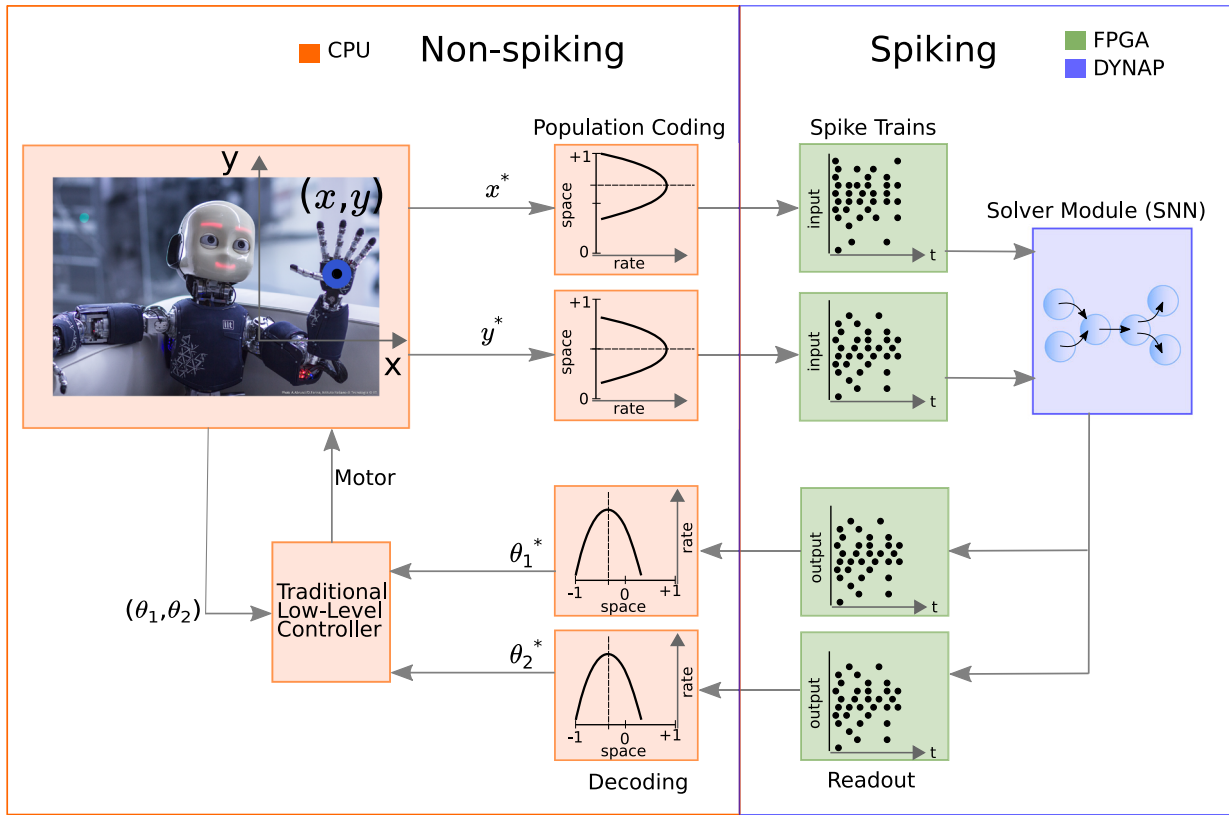
fall exactly into the expected partition. We are doing this to have a direct comparison between the SNN-based solver and *IpOpt* with the same encoding/discretization scheme of the end-effector space. Using more neurons can increase the neural coding resolution, and an improved neural encoding method can better map the Cartesian space to spiking neurons, making it possible to directly benchmark spike-based against classical methods without restrictions. It is true that a low-latency traditional solver could be engineered; however, we used the best implementation at hand as a baseline reference, which is currently integrated into the iCub and is optimized for this platform. We tuned the parameters of *IpOpt* to make it as fast as possible and turned off all the other applications apart from iCubSim running on the computer during the control experiments.

Also, we compared the spiking controller with the state-of-the-art (SoA) SNN-based inverse kinematics solvers implemented on neuromorphic hardware. The inference time (required for computing convergence) of the SNNs presented in ref. <sup>32</sup> corresponds to the network latency we measured here. In the best case, the deep SNN with 199,685 parameters, converted from a trained artificial neural network (ANN) and deployed on Loihi, needs 400 ms to generate the joint solution to reach the target end-effector position, which is slower than the setup we used for the target-reaching task (with 33.96 ms average latency) and the worst-case network latency of 170.37 ms with weak synaptic strength. The other SNNs in ref. <sup>32</sup> takes 2.6 to 3.8 s for the spiking activity in the networks to converge.

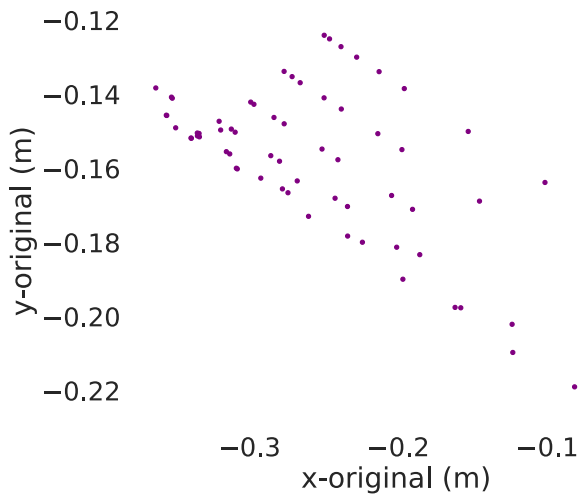
The system's accuracy corresponds to the discretization resolution of 6.4 and 7.5 cm in the most narrow grids in Figs. 1B and 5C, comparable to the mean error of the learning-based and SGD-trained SNNs<sup>32</sup> (6.3 and 3.8 cm), and can be improved by adding more neurons. The numbers reported here are only indicative of the quality of the overall system, as the error between the desired and final end-effector position also depends on the robot's actuation limits and accuracy.

### Power consumption

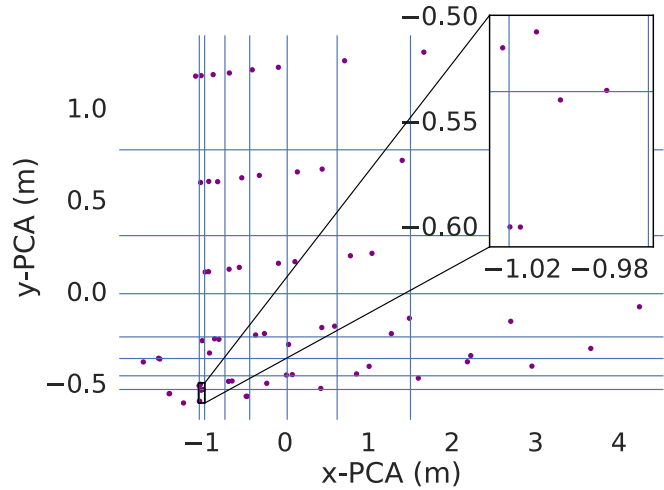
As the power consumption of the DYNAP-SE1 cannot be directly measured online during operation, we indirectly assess it as the sum of the power required for different operations relative to



(A)



(B)



(C)

**Fig. 5 System architecture and space discretization.** **A** The overall pipeline shows the implemented modules and the communication system between them. The components on the CPU are non-spiking, while the ones on the FPGA and DYNAP-SE1 are spiking. **B** End-effector positions in the original Cartesian space resulting from the applied joint configurations during motor babbling. **C** Discretisation of the end-effector space: the coordinates of the original end-effector positions in **B** are normalized, and principal component analysis (PCA) is applied to find a new plane that maximizes the variance of the position coordinates to split the dots as further from each other as possible. The blue lines are borderlines found using  $N$ -quantiles along the new x and y dimensions that divide the PCA arm space into discrete partitions.

spike generation and communication listed in Table S1<sup>33</sup>:

$$P = \sum_{n=1}^N r_n (E_{\text{spike}} + E_{\text{enc}} + N_{\text{cores}} (E_{\text{br}} + E_{\text{rt}}) + N_{\text{cam\_match}} E_{\text{pulse}}) \quad (1)$$

where  $E_{\text{spike}}$ ,  $E_{\text{enc}}$ ,  $E_{\text{br}}$ ,  $E_{\text{rt}}$ , and  $E_{\text{pulse}}$  represent the energy consumed by a DYNAP-SE1 operation to generate a spike, encode

a spike and append destinations, broadcast events to the same core, route events to a different core and extend generated pulse, respectively (see values in Table S1).  $N$  is the number of neurons in the SNN,  $r_n$  is the firing rate of neuron  $n$ ,  $N_{\text{cores}}$  is the number of destination cores of the neuron, and  $N_{\text{cam\_match}}$  is the total number of postsynaptic neurons that receive the input spikes.



The power consumption of the SNN and each network layer during the training and testing phases are reported in Table S2. The average power dissipation of the network during the training phase (3.46  $\mu$ W) is lower than during the control task (26.92  $\mu$ W) because (1) the cooling down phase used during training, which decreases the mean firing rates, constitutes half of the training procedure; (2) the inhibitory neurons *hiddenJoint\_inh* are not used during training.

The inhibitory neurons *hiddenJoint\_inh* of the WTA added during the testing phase contribute over 62% of the total energy consumption due to a large number of postsynaptic connections (to 64 neurons in *hiddenJoint*). As the power consumption of the network depends on the overall spiking activity of the neurons, it is affected by the weights between *hiddenCartesian* and *hiddenJoint*, as shown in Fig. 4B. During testing, this effect is amplified by the *hiddenJoint\_inh* population, as its dynamics follows *hiddenJoint*.

We are unable to measure the power consumption of the full chip due to the constraint of the DYNAP-SE1 processor and the available measurement device. We, therefore, estimated the power consumption of the chip using the total number of events produced by the neural populations in the model and their power budget. For larger SNNs, the power consumption of the chip would scale mainly with this figure, while other sources of power consumption on chip would not increase significantly. In the current prototype set-up, the interfacing FPGA and the algorithm running on the computer take much more power than the SNN on the chip. However, this component would decrease substantially in an optimized end-to-end neuromorphic control system. In addition, while it is true that the overall power consumption of the robot also depends on the power used by the actuation, the contribution to the power budget given by the processing power cannot be ignored, especially considering future scaled-up systems. Currently, most of the processing is performed on CPU and graphics processing unit (GPU) racks for space and power limitations. Optimizing computation (at all levels) will certainly improve the overall energy consumption figures. This applies to the iCub, and to most robotic platforms.

Unfortunately, it is not possible to compare the power consumption of our set-up with other SNN-based inverse kinematics solvers implemented on neuromorphic hardware proposed in the literature<sup>28,31,32</sup>, as those figures of merit are not reported. For a qualitative comparison, we can only use the reported firing rate, as a proxy of the energy used by the SNN, as in neuromorphic chips, power usually scales with the overall firing activity of the networks.

The net global firing rate of the SNN running on DYNAP-SE1 during the inference phase is about 1.4 Hz (184 neurons, of which 16% are active with a mean firing rate of about 52 Hz). We derived a similar metric for the networks deployed on the Loihi set-ups<sup>32</sup>. The two networks running on Loihi<sup>32</sup> comprise four and five neural populations each. In the “best-case” scenario, assuming the mean activity of a single neuron is the reported figure of 1 kHz and assuming there is at least one neuron active in each of the 4/5 populations, the minimum activity would be roughly 4/5 kHz. The reason for the large difference in mean firing rates between our approach (about 52 Hz/neuron) and the Loihi-based ones (about 1 kHz/neuron) lies in the fact that we developed neural architectures that are inspired by their biological counterparts, which have been optimized by evolution to minimize power consumption, while in the Loihi-based SNNs, the authors argue that the neurons require a high spiking rate, in order to approximate ANN performance, when converting ANN to SNN.

## DISCUSSION

In this work, we trained an SNN on DYNAP-SE1 with a computer in the loop to learn the inverse kinematics of the iCub robot in a

simulated environment, constraining the movement to the shoulder pitch and the elbow. The SNN features a disinhibition mechanism inspired by the one found in basal ganglia’s neural circuits, which eliminates the noisy firing patterns in the neural populations with multiple input sources. The selective activation of specific neurons is crucial to both the event-driven STDP learning process and task execution. The trained SNN is used as the solver module to coordinate the shoulder pitch and the elbow joints to drive the end-effector (in our case, the palm of the hand) to reach 12 different positions continuously. In the limited conditions imposed by the neural coding used in the proposed system, the nonlinear optimization solver *IpOpt* embedded in the iCub control module for inverse kinematics achieves less accuracy than the spiking controller (all target end-effector positions are reached), that also shows lower latency and power consumption. The entire learning procedure is ultra-low power and only takes approximately 51.2s for 64 training samples. As a proof-of-concept, we trained a small but scalable spiking controller, which marks a significant step for neuromorphic robotics toward more complex and adaptive behaviors.

To scale up the SNN beyond the proof-of-concept two degrees of freedom demonstrator presented in this manuscript, more neurons can be used in each population to increase the task space, and more neural populations and larger connectivity matrices can be used to scale up the end-effector space from 2D to 3D and to increase the joint configuration space, including more degrees of freedom. Also, more space-efficient encoding schemes<sup>44</sup> can be used to ease the quadratic growth of the hidden populations due to the increasing size of the related input/output populations, at the cost of potentially slower network reaction.

Furthermore, real-world neuromorphic robotics applications would benefit from scalable and flexible neuromorphic processors, with more neurons and larger and more flexible input and output connectivity: an increased number of neurons could minimize the discretization error from encoding continuous analog variables using a limited number of individual neurons; a more flexible network topology, e.g., more input synapses and higher weight resolution, would increase the diversity of implementable SNNs to meet the requirements of the application. A user-friendly ecosystem including both the hardware and software infrastructure for neuromorphic algorithm developers is crucial for moving beyond proof-of-concept demonstrators: spike-based software libraries, toolboxes, and middleware for communication, processing, and analysis are lagging behind the requirements of emerging neuromorphic applications. Consequently, the performance (i.e., latency, throughput, power consumption, etc.) of the experimental neuromorphic setup suffers from redundant self-designed interfaces in the pipeline. The implementation of these interfaces can be challenging for individual researchers due to the time-consuming development process with even inadequate performance compared to those optimized by field experts. Based on this observation, building blocks for spiking robotic architectures with modularity, reusability, and plug-and-play features would benefit the deployment of robotic systems integrated with various neuromorphic sensors, computing substrates, and actuation modules. In particular, important blocks for real-time closed-loop motor control are fast input and output interfaces to speed up the sensorimotor loop of real-time motor control.

Finally, to scale up the implementation of the inverse kinematics model proposed here, to a fully spiking pipeline, the high-level controller can be interfaced with event-driven sensors and low-level controllers of single joints<sup>17,18</sup>: in the target-reaching task, an event camera<sup>45</sup> can be used to capture and encode the desired end-effector position in spike trains sent directly to the input populations in the SNN, and spiking tactile sensors<sup>46–48</sup> could be used for force feedback. The solver module SNN can be

interfaced with low-level spiking controllers<sup>17,18</sup> and PFM drivers<sup>15</sup>, creating an end-to-end spiking pipeline that does not require to waste of energy and time to convert signals from and to clocked representations<sup>13</sup>. End-to-end spiking systems would also reduce the power consumption required to transfer data between different systems. On-chip learning would further reduce power consumption during the learning phase by removing the need for a computer in the loop. This can be supported by the integration of memristive devices, that have been shown to support triplet-STDP rules<sup>49,50</sup>, and that allow scaling up the proof-of-concept two-joints control described here to higher dimensionality problems. This approach will come at the cost of higher device mismatch, which can be overcome using brain-inspired methods for achieving robust computation in heterogeneous mixed-signal neuromorphic processing systems<sup>51,52</sup>, at the cost of increasing the number of neurons, and power.

Besides the latency due to the system configuration that mostly depends on data conversion and transfer (from around 100 to 170 ms), there is an intrinsic latency due to the on-chip SNN convergence time (from 14.44 to 170.37 ms) that depends on the total drive of the network from the input and on the recurrent connectivity.

Since the accuracy in generating joint angles is mainly affected by the multiple winners in the output hidden population caused by the one-to-more connections from the input hidden population, a second run of training can be performed to do connection pruning based on the learnt inverse kinematics. More task-specific datasets (e.g., target trajectories instead of random motor babbling) can be collected to make the synapses between the hidden populations more selective to choose the optimal solution for the task out of all the possibilities. Also, a correlation between  $\theta_1$ ,  $\theta_2$  neurons in the output layer can be established to avoid invalid combinations by adding inhibitory connections across populations.

Latency, accuracy, and power also depend on the SNN configuration. These figures of merit mainly depend on the strength of synaptic connections between the hidden layers, so strong connections lead to high spiking rates and fast switching behavior of the network, hence lower latency, higher power consumption, and lower accuracy. The weights can be configured according to the features and requirements of different robotic systems and tasks. Learning to automatically adjust the weights on the fly can flexibly tune the SNN behavior and the latency/accuracy/power trade-off.

It is difficult to make a comprehensive quantified comparison with the SoA because (1) the benchmark task—e.g., a target-reaching task with the same DoF, target end-effector trajectories, robot kinematics, systematic errors, and even the same robotic platform - for the comparison of different neuromorphic motor controllers, is missing. Consequently, the selected robotic platform and the defined task have a significant impact on the presented results; (2) the metrics used to assess controller performance are not standardized. Comparison between the target and actual end-effector trajectories depicted in the figures is a typical measurement. However, the trajectory difference is either not quantified or done with various methods. Even with the same metric (e.g., RMSE), the values are highly affected by the robot (e.g., execution time) and the task (e.g., point-to-point distance), not only by the controller itself. Furthermore, latency and power usage, two crucial measures of neuromorphic controller performance, are rarely quantified or mentioned; (3) there are very few hardware-implemented spiking controllers to compare, and even if we consider the simulated ones, the issues listed above would still exist.

Therefore, we can only compare the available metrics reported in the literature<sup>32</sup> even if it solves the inverse kinematics for another robot, in a different task. We found that our SNN shows better performance in terms of network latency. Since power

consumption is not reported in the literature, we resort to relying on the overall firing rate, as a proxy for computation load and power consumption. Based on this, the SNN proposed in this paper may have better power efficiency, with an average firing rate of a few Hz, if compared to a few kHz of the Loihi implementation. In terms of learning capabilities, online systems support on-the-fly adaptation (e.g., to new environmental or geometrical constraints or tasks). The approach proposed in this manuscript is also less computationally expensive, as it adopts bio-plausible spike-based local learning rules in an event-driven fashion instead of SGD and backpropagation. Therefore, it can be replaced with event-based FPGA processing modules and on-chip learning circuits to further minimize power consumption. In addition, we compare the spiking controller with a classical inverse kinematics solver (*IpOpt*) in the same target-reaching task on the same robot and show that the SNN-based controller achieves comparable latency and control performance. Other solvers<sup>53</sup> report latency in the order of 0.1 ms, but are based on tailoring the solver to specific robotic platforms, where assumptions can be made to simplify the system by using model-based approaches, that however, do not generalize to all robots.

Reproducibility and robustness of the work are important aspects. The device mismatch of DYNAP-SE1 due to its analog nature is thoroughly measured and quantified in ref. <sup>52</sup>, which also proposes corresponding neural processing strategies for robust computation, given the hardware variability. Most of the strategies are adopted in the SNN proposed in this work, e.g., using population codes, recurrence and self-excitation, soft WTA networks, spike-based learning and plasticity, etc. Moreover, the training procedure is robust because the neural circuit exploited in the proposed system creates a bio-plausible disinhibition mechanism, which produces selective firing patterns of only a single pair of desired pre-post neurons simultaneously during training and triggers only the target neuron from the input side during inference. The inference error is not caused by the hardware mismatch but by the limited encoding resolution, which can be reduced using more neurons.

## MATERIALS AND METHODS

In this work, the shoulder pitch and elbow joints of the iCub are controlled in simulation to drive the end-effector to reach target positions in 2D space. A simplified model is shown in Fig. 1A. The joint angles  $q = (\theta_1, \theta_2)$  set the end-effector to the Cartesian position  $x = (x, y)$  following the forward (or direct) kinematics relationship  $x = f(q)$ . Conversely, to move the end-effector to a target position  $(x^*, y^*)$ , a solver module should generate the necessary joint angles  $(\theta_1^*, \theta_2^*)$  by solving the inverse kinematics relationship  $q = f^{-1}(x)$ . The latter can be solved analytically when the robot model and parameters are precisely known, numerically in the presence of limited errors in the model, or learnt through neural networks. In this work, an SNN is trained on a neuromorphic processor to learn the inverse kinematics and use it as the solver module to control the joints and drive the end-effector to the desired Cartesian positions.

Figure 5A shows the pipeline of the devised motor control system. As a testing environment, we resorted to iCubSim, DYNAP-SE1, and a computer in the loop for the training. The software modules on the laptop (including iCubSim) and on DYNAP-SE1 communicate via a Spartan-6 XC6SLX25 FPGA and a C++ event-driven library<sup>54</sup>, providing support for the integration of robotic modules with event-driven sensing and computing platforms. As the iCub robot relies on mainstream digital logic, an interface layer (on FPGA) is needed to encode Cartesian coordinates into input spike trains and decode output spike trains into digital joint values sent to motors.

The desired position  $(x^*, y^*)$  is discretized into one-hot population codes, where each neuron represents a sample of

the Cartesian space. To encode a desired position on the CPU, the neurons are stimulated with a Gaussian profile centered on  $(x^*, y^*)$ . The mean firing rates of the neurons are sent to the FPGA, where they are converted into Poisson spike trains and fed into the SNN.

The solver module, running on DYNAP-SE1, is a trained SNN that maps the inverse kinematics, continuously calculating the joint's configuration. The joint angles  $(\theta_1^*, \theta_2^*)$  corresponding to the desired Cartesian position are encoded by the neurons with the highest firing rate. The spikes of the output neurons in the SNN are streamed out from the DYNAP-SE1 chips via the FPGA. The instantaneous firing rates of the output neurons are converted into digital values (on CPU) and sent to the low-level motor controller of each motor to drive the end-effector to the target position.

### Population coding for encoding analog variables

To interface, the SNN mapping of the inverse kinematics with the non-spiking analog representation used in the robot, the Cartesian position and the joint angles are represented by dynamical neural populations. The joint angles are uniformly discretized following Eq. (2):

$$i = \left\lceil \frac{\theta - \theta_{\min}}{\theta_{\max} - \theta_{\min}} \times (N - 1) \right\rceil \quad (2)$$

where  $N$  is the size of the population,  $\theta_{\min}$  and  $\theta_{\max}$  are the minimum and maximum angles the joint can reach, and  $i$  is the neuron index. The angle can be decoded from the index of the maximally activated neuron using the same equation.

To train the network, the manipulability space of the end-effector is sampled in random order through motor babbling<sup>55</sup>, by applying different joint angles  $(\theta_1, \theta_2)$  (across the joint space) to the arm. The generated Cartesian positions are non-uniformly distributed (Fig. 5B) due to the combination of rotations around the two axes and the different lengths of the arm's links. As a result, unlike in Eq. (2), the mapping from Cartesian coordinates to neuron index cannot be a uniform distribution. Because of the error introduced by the discretization (highlighted box in Fig. 5C), multiple positions fall in each partition and, given the non-uniform distribution, some regions are populated by more positions (resulting in denser regions). Therefore, we need to tailor the size of partitions to the density of the sampling. To do so, non-uniform Cartesian space discretization is obtained by applying normalization, principal component analysis (PCA), and  $N$ -quantiles division to the original sampled Cartesian coordinates resulting from the uniform discretization of the joint space (Fig. 5C). PCA rotates the  $x$  and  $y$  dimensions and finds a new 2D plane that maximizes the variance of the coordinates (thus the distribution of the dots is more sparse). The new end-effector coordinates in the PCA plane are then divided along  $x$  and  $y$  dimensions into  $N$  equal partitions (with  $1/N$  data points in each partition), respectively, using  $N$ -quantiles, which corresponds to  $N$  neurons in two neural populations. The discretization of the arm space is marked by the blue lines along the  $x$  and  $y$  axes in Fig. 5C. Non-uniform discretization reduces the discretization error, and the error can be further minimized by adding more neurons.

### Spiking neural network as inverse kinematics solver

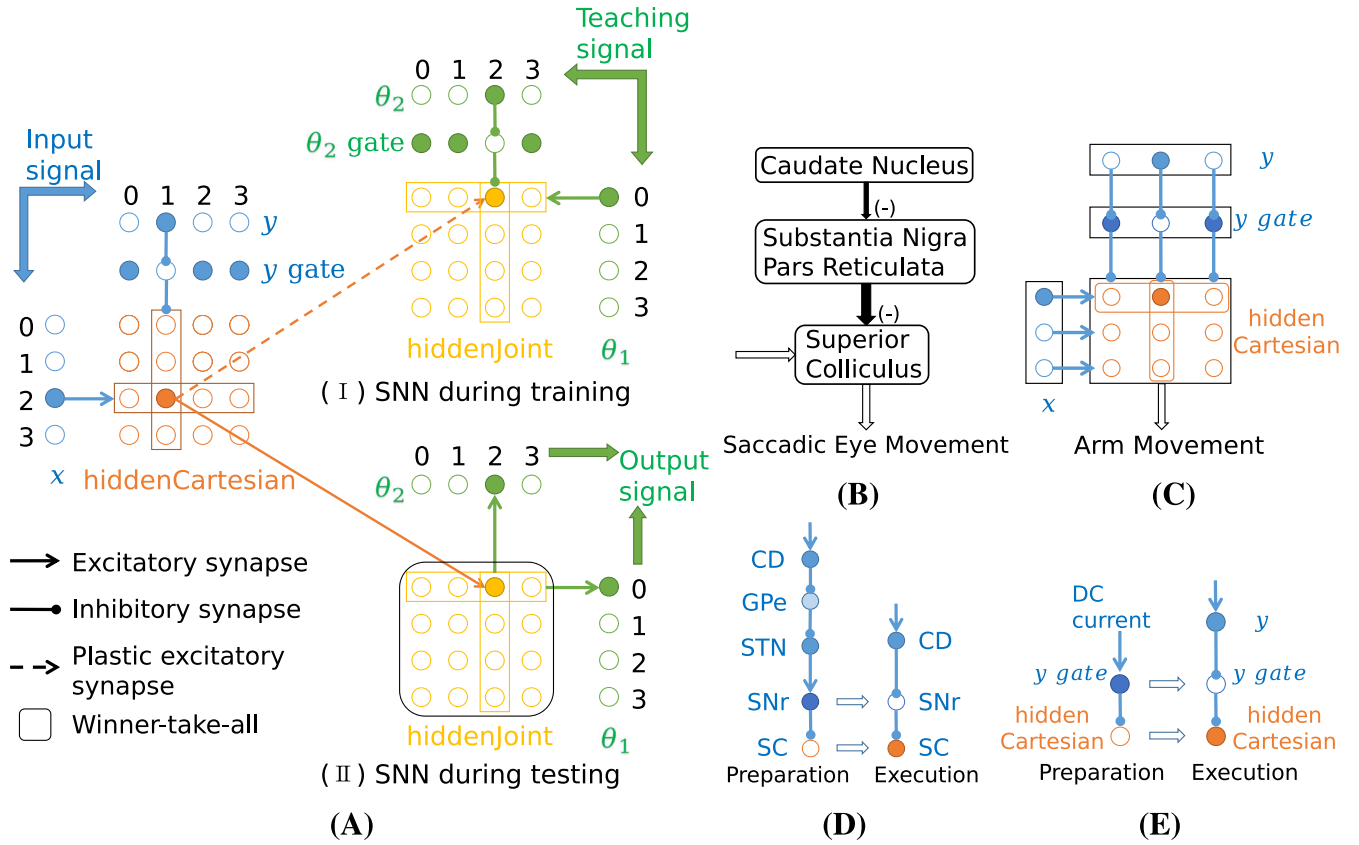
The solver module SNN is shown in Fig. 6A, both during training and inference. The neural populations encoding the Cartesian coordinates  $(x, y)$  feed the hidden layer *hiddenCartesian* representing the Cartesian space. Each neuron in populations  $x$  and  $y$  is connected with excitatory synapses to one row and one column, respectively. This results in the activation of all the neurons in the row and column, with higher activation of the neuron that corresponds to the input but introduces noise that disrupts the learning. To suppress the activation of neurons that do not exactly

match the  $(x, y)$  input during training, in *hiddenCartesian*, excitatory synapses are replaced by disinhibitory connections from  $y$  to *hiddenCartesian*, through a layer of gating neurons ( $y$  gate layer).

The neural circuits found in basal ganglia (Fig. 6B) inspired the disinhibition structure in the proposed network (Fig. 6C). Basal ganglia contribute to the learning and selection of actions via disinhibition, to control skeletal and saccadic eye movements<sup>35–37</sup>. To suppress involuntary saccadic eye movements, the substantia nigra pars reticulata (SNr) neurons fire at 50–100 Hz, stimulated by the sustained activity in the subthalamic nucleus (STN) neurons, constantly inhibiting pre-saccadic neurons in the superior colliculus (SC). This inhibition is removed by another inhibition from the caudate nucleus (CD) to the SNr, which results in the disinhibition of the SC<sup>37</sup>. Similarly to the SC, the *hiddenCartesian* layer receives multiple excitatory inputs that elicit excitatory activity, and disinhibition is used to selectively activate the correct neurons. To emulate the continuous drive of SNr by STN neurons, the  $y$  gate neurons receive a constant input current. As CD inhibits SNr,  $y$  modulates  $y$  gate through one-to-one connections, and each  $y$  gate inhibits a column of neurons in *hiddenCartesian*. When the input signal from the Poisson spike generators stimulates the input populations, the  $y$  neuron inhibits its  $y$  gate neuron. Since this single  $y$  gate neuron is inhibited, the corresponding column of *hiddenCartesian* neurons are disinhibited and get the chance to fire. However, only the neuron at the crossing point of the row - stimulated by the  $x$  population and the disinhibited column—can fire. The activity of the *hiddenCartesian* population, therefore, represents the desired end-effector position in the Cartesian plane.

The transition from one target end-effector position to another also benefits from the disinhibition mechanism. There are two pathways in basal ganglia (see Fig. 6D). The direct pathway (right) creates the selective inhibition of the SNr neurons, which releases the SC neurons and initiates movements, while the indirect (left) one leads to less selective facilitation of SNr which inhibits the SC neurons and suppresses movements<sup>37</sup>. These two pathways dominate sequentially to produce the switching of behavior from preparation (suppression) to execution (initiation). Similarly, in our network (Fig. 6E), the constant input current to the  $y$  gate neurons plays the role of the preparation (indirect) pathway, while the inhibition from  $y$  to  $y$  gate corresponds to the execution (direct) pathway. During task execution, when switching from one target  $(x, y)$  to another, the old  $(x, y)$  input signal is removed, leading the preparation pathway to be dominant due to the constant input current. Then the new input is given by first stimulating  $y$  neurons to apply the selective disinhibition and then activating  $x$  to trigger *hiddenCartesian* neurons. The slightly earlier stimulation to  $y$  opens the gate for the target *hiddenCartesian* neuron by silencing its  $y$  gate neuron. This disinhibition makes the selective *hiddenCartesian* neuron ready to receive activation from  $x$ , i.e., initiates the network state for generating a new movement. And then  $x$  stimulation kicks in and triggers the target *hiddenCartesian* neuron, which fully translates the network to the execution phase.

During the training phase (Fig. 6A), similar excitatory and disinhibitory connections are created from  $\theta_1$  and  $\theta_2$  to the hidden population *hiddenJoint* so that the active *hiddenJoint* neuron encodes the desired output state  $(\theta_1, \theta_2)$ , sent as a teaching signal. The inverse kinematics is learnt in the plastic connections from *hiddenCartesian* to *hiddenJoint*, through triplet-STDP<sup>56</sup>. Selective firing patterns in *hiddenCartesian* and *hiddenJoint* via the biologically plausible disinhibition mechanism are crucial to the training performance. At inference time, the activity of the *hiddenCartesian* population drives the correct neurons in the *hiddenJoint* population, which are then decoded as  $\theta_1$  and  $\theta_2$  from the output populations (Fig. 6A). The neurons' activity in  $\theta_1$  and  $\theta_2$  is continuously decoded as joint angles to drive the motors using 1-hot population decoding (Eq. (2)).



**Fig. 6 Network architecture.** **A** The SNN as inverse kinematics solver. The circles represent neurons in color coding that blue, orange, yellow, and orange stand for the input layer (including  $x$ ,  $y$ , and  $y$  gate), hidden layer  $hiddenCartesian$  and  $hiddenJoint$  and output layer (including  $\theta_1$ ,  $\theta_2$ , and  $\theta_2$  gate, respectively). The solid circles indicate firing neurons, while the empty ones are inactive. **B** Adapted from ref. <sup>37</sup>. In basal ganglia, the caudate nucleus (CD) neurons selectively inhibit the spontaneously firing substantia nigra pars reticulata (SNr) neurons, which suppresses the inhibition from SNr to some superior colliculus (SC) neurons. Consequently, only the disinhibited SC neurons can be activated by stimulus from other brain areas to generate saccadic eye movements. **C** In the proposed network, the selected  $y$  neuron disinhibits  $hiddenCartesian$  neurons via  $y$  gate so that the target  $hiddenCartesian$  neuron can be triggered by the stimulation of  $x$  to generate arm movements. **D** Two pathways in preparation and execution phases in basal ganglia. The indirect pathway (left) suppresses involuntary movements, while the direct one (right) disinhibits selective SC to generate voluntary motions. The indirect and direct pathways dominate sequentially during the preparation and execution phases; **E** Emulating biology, active  $y$  gate inhibits  $hiddenCartesian$  to creating the preparation phase, while  $y$  disinhibits  $hiddenCartesian$  to form specific firing patterns for action selection and execution.

Due to the discretization of the arm space, different end-effector positions ( $x$ ,  $y$ ) correspond to the same discretized joint angles ( $\theta_1$ ,  $\theta_2$ ) (partition in Fig. 5C). During training, the synapses corresponding to these multiple solutions are learnt, and a WTA network is used at inference to select a single joints configuration from the multiple possible solutions. WTA is implemented with a global inhibitory population with an excitatory vs. inhibitory neuron ratio of 4:1<sup>57</sup>.

Each input, output, and gate populations comprise  $N = 8$ , each hidden population has  $N^2$  neurons, and  $\frac{N^2}{4}$  inhibitory neurons are used to create the WTA network in  $hiddenJoint$ . In total, 176 and 184 neurons are used for the training and control networks, respectively.

### Learning using triplet-STDP

Learning is implemented using the minimal version of the triplet-STDP algorithm<sup>56,58</sup>, derived as models of learning observed in visual cortex<sup>59</sup> and hippocampus cultures<sup>60</sup>. The weight between a presynaptic and a postsynaptic neuron is updated using three exponentially-decaying traces: presynaptic trace  $r_1(t)$ , postsynaptic traces  $o_1(t)$  and  $o_2(t)$  (see Fig. 7), updated at each pre or post-

spiking times respectively:

$$\begin{aligned} \frac{dr_1(t)}{dt} &= \frac{r_1(t)}{\tau_{pre}}, & \text{if } t = t_{pre}, \text{ then } r_1 &\rightarrow 1 \\ \frac{do_1(t)}{dt} &= \frac{o_1(t)}{\tau_{post1}}, & \text{if } t = t_{post1}, \text{ then } o_1 &\rightarrow 1 \\ \frac{do_2(t)}{dt} &= \frac{o_2(t)}{\tau_{post2}}, & \text{if } t = t_{post2}, \text{ then } o_2 &\rightarrow 1. \end{aligned} \quad (3)$$

where  $\tau_{pre}$ ,  $\tau_{post1}$ , and  $\tau_{post2}$  are the time constants of the traces.

A presynaptic spike at time  $t_{pre}$  (the blue dotted line in Fig. 7) triggers the LTD of the weight:

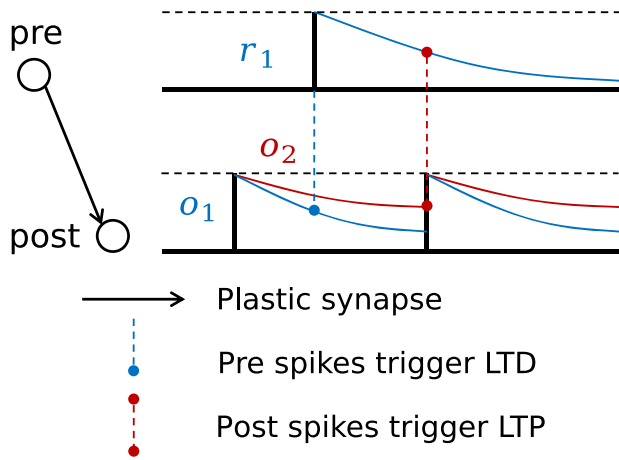
$$\begin{aligned} \Delta w(t) &= -A^- o_1(t) w(t)^{\mu_{pre}} \\ w(t) &\rightarrow w(t) + \Delta w(t), & \text{if } t = t_{pre}. \end{aligned} \quad (4)$$

where  $A^-$  is the amplitude of the weight decrease whenever there is a post-pre pair of spikes, and  $\mu_{pre}$  sets the weight dependence to the current weight. Similarly, a postsynaptic spike at time  $t_{post}$  (the red dotted line in Fig. 7) triggers the LTP of the weight:

$$\begin{aligned} \Delta w(t) &= A^+ r_1(t) o_2(t - \varepsilon) (w_{max} - w(t))^{\mu_{post}} \\ w(t) &\rightarrow w(t) + \Delta w(t), & \text{if } t = t_{post}. \end{aligned} \quad (5)$$

where  $A^+$  is the amplitude of the triplet potentiation term (i.e., 1-pre-2-post term) whenever there is a pre-post pair of spikes,  $\varepsilon$  is a very small positive constant to sample  $o_2$  before its reset at time





**Fig. 7 Triplet-STDP traces.** The solid black lines are the spikes generated by presynaptic and postsynaptic neurons which trigger long-term depression (LTD) (in blue) and long-term potentiation (LTP) (in red) processes, respectively. Spike times of trace  $r_1$  (i.e.,  $t_{pre}$ ) and the corresponding  $o_1$  values are needed by weight update in LTD, while trace values of  $r_1$  and  $o_2$  are read out at the firing times  $t_{post}$  of  $o_2$  for LTP.

$t_{post}$ ,  $\mu_{post}$  determines the weight dependence, and  $w_{max}$  clips the maximum weight of the plastic synapse.

The triplet-STDP learning rules are implemented with a computer in the loop using an event-driven framework that streams out the spikes of pre- and postsynaptic neural populations at run-time, calculates the required traces (Eq. (3)) and triggers LTD (Eq. (4)) and LTP (Eq. (5)) weight updates at pre- and postsynaptic spiking times, respectively.

The spikes of pre and postsynaptic neurons, streamed from the chip to the CPU, are used to compute the LTD and LTP **traceEvents**, respectively. The floating-point weights are updated at run-time driven by the generated LTP and LTD **traceEvents**, by modifying the weight matrix stored by the triplet-STDP algorithm running on the CPU. After each training sample, the analog weights are converted into binary weights that are applied to DYNAP-SE1 to adjust the connections at run-time during training. The weight is binary as it can only have two states: either depressed, i.e., the pre and postsynaptic neurons are non-connected, or potentiated, i.e., the pre and postsynaptic neurons are connected and their weight is determined by a global parameter, to meet the hardware constraints. The new binary weight matrix generated after a new training sample will be compared to the current one on DYNAP-SE1 and only the different connections will be updated on the chip. The detailed implementation is described in Section Event-driven Implementation of Triplet-STDP.

The training data was generated by selecting all possible  $N^2$  joint angles ( $\theta_1, \theta_2$ ) of the shoulder and elbow joints of the left arm of iCubSim, corresponding to  $N^2$  end-effector positions ( $x, y$ ) as shown in Fig. 5B, C. During training, the  $N^2$  training samples are fed in random order (motor babbling) into the network. Each sample lasts for 400 ms, then the floating weight matrix obtained with the triplet-STDP rule is converted into discrete (binary here) weights to comply with the chip constraints. Before training, all the initial floating-point weights are set to  $w_{init}$  (see Table S3). These analog weights are updated by the triplet-STDP rules, and only the ones that are potentiated to a certain level (stronger than  $w_{thr}$  in Table S3) will be applied to the DYNAP-SE1 chip after each training sample.

The weight discretization process happens in three steps: (i) thresholding: all floating weights are subtracted by a threshold value  $w_{thr}$  (see Table S3) so that the weights potentiated by weakly firing (noisy) neurons are filtered out; (ii) binarisation: all

non-negative values are set to 1, while the others are set to 0. Since all the remaining floating weights are equally important in the connectivity pattern representing the inverse kinematics, they are converted to the same discrete magnitude regardless of the analog value. This conversion punishes the overgrowing synapses and strengthens the weak ones to decrease the effect of mismatch across neural activation levels during the learning process and favors homogeneity in the network to remove biases in the selection of the control solution (more details in Section Weight Discretisation); (iii) Fusion: the weight matrices learnt after samples  $i$  and  $i + 1$  are merged into one so that new connections are learnt and the old ones are ensured to survive. In most cases (96.25%), the previous two steps have guaranteed that the weakly potentiated synapses in the earlier training samples will not fade away with more training data afterward. With fusion, the accuracy increases to 99.69% and can be further maximized by a second run of the training using the same dataset to strengthen the target connections encoding the inverse kinematics.

A new training sample is injected into the network after a 400 ms interval to cool down the network activity, and the same procedure is repeated for all the  $N^2$  samples. STDP and training parameters are shown in Table S3.

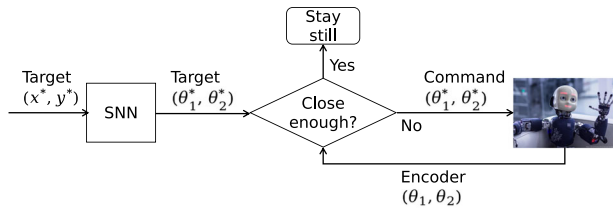
Here, we are not aiming at a faithful replica of how biological systems solve the inverse kinematic problem. Rather, we are using neural computational primitives to solve an engineering task. One of these primitives is indeed pruning, which happens in our implementation via weight discretization during training. Since we train the weight matrix on the CPU with computer-in-the-loop, the all-to-all connections are only maintained in the software weight matrix, and the weights will be thresholded and binarised in order to apply them on hardware after each training sample. This weight discretization procedure only keeps the strong synapses learnt in the weight matrix. Therefore, after training, we only retain the sparse connections learnt in the weight matrix (see Fig. 1C) applied to the neuromorphic hardware, so that neurons that do not have sufficiently strong synapses with other neurons could be re-allocated to other tasks. When moving to on-chip learning, pruning would lead to the optimization of neuromorphic chip resources, but a technological solution and the infrastructure to do so should be developed.

### Inference: to control a robotic arm

The trained network is used to control the two-joint arm to complete a continuous target-reaching task. Fig. 1B shows 12 end-effector positions in the PCA space, forming a target trajectory starting from the right-bottom point. For each testing sample, the desired end-effector position ( $x, y$ ) is encoded into spikes which stimulate the  $x$  and  $y$  populations of Fig. 6A(II). The firing neurons in  $x$  and  $y$  activate the corresponding *hiddenCartesian* neuron, and then action selection happens from *hiddenCartesian* to *hiddenJoint* via the inter-population connections, which represents the inverse kinematics. These trained synapses connect the end-effector space to the joint space. If multiple *hiddenJoint* neurons fire, the WTA in *hiddenJoint* will select a single winner neuron, which will then activate the corresponding neurons in  $\theta_1$  and  $\theta_2$ , representing the joint command.

Joint configurations are calculated continuously by the SNN and read out whenever the *hiddenJoint* neurons fire. The decoded joint commands are sent to iCubSim to drive the shoulder and elbow joints, as in Fig. 8. However, the actuation speed of iCubSim cannot keep up with frequent command inputs because it takes 0.3 ms to 1.8 s for a joint to move from one position to another precisely. To reduce the number of joint movements, once iCubSim receives a new joint configuration ( $\theta_1^*, \theta_2^*$ ), the current joint positions ( $\theta_1, \theta_2$ ) from the encoder will be compared with the target ones. If the current angles are close enough to the target ones, the joint configuration will not be applied to the robot





**Fig. 8 Control pipeline.** The SNN computed joint solution ( $\theta_1^*, \theta_2^*$ ) is compared with the current joint state of iCubSim to see whether the new command should be used to drive the robot joints to new angles. If the Euclidean distance between the current and command joint configuration is within a threshold (i.e., they are close enough), the command will not be executed.

because the joints have already reached the desired positions. Otherwise, the joint command ( $\theta_1^*, \theta_2^*$ ) will be used to drive the two joints. The Euclidean metric is used to check if the desired and current joint vectors are close enough. If the Euclidean distance is larger than a threshold ( $0.5^\circ$  in this work), the joints will be moved to the new positions.

### Reporting summary

Further information on research design is available in the Nature Research Reporting Summary linked to this article.

### DATA AVAILABILITY

The data that support the findings of this study are available from the corresponding author, J.Z., upon reasonable request.

### CODE AVAILABILITY

All software programs used in the presentation of the article are freely available upon request in a GitLab repository. Access to DYNAP-SE1 will be made available through remote connection upon request as well.

Received: 31 October 2022; Accepted: 6 March 2023;

Published online: 26 October 2023

### REFERENCES

- Mead, C. How we created neuromorphic engineering. *Nat. Electron.* **3**, 434–435 (2020).
- Christensen, D. V. et al. 2022 roadmap on neuromorphic computing and engineering. *Neuromorp. Comput. Eng.* **2**, 022501(2022).
- Chicca, E., Stefanini, F., Bartolozzi, C. & Indiveri, G. Neuromorphic electronic circuits for building autonomous cognitive systems. *Proc. IEEE* **102**, 1367–1388 (2014).
- Rahimiazghadi, M. et al. Hardware implementation of deep network accelerators towards healthcare and biomedical applications. *IEEE Trans. Biomed. Circuits Syst.* **14**, 1138–1159 (2020).
- Ma, Y. et al. Emg-based gestures classification using a mixed-signal neuromorphic processing system. *IEEE J. Emerg. Select. Topics Circuits Syst.* **10**, 578–587 (2020).
- Delbruck, T. & Lang, M. Robotic goalie with 3 ms reaction time at 4% cpu load using event-based dynamic vision sensor. *Front. Neurosci.* **7**, 223 (2013).
- Bartolozzi, C. et al. Embedded neuromorphic vision for humanoid robots. In *CVPR 2011 Workshops* 129–135 (IEEE, 2011).
- Sandamirskaya, Y., Kaboli, M., Conrad, J. & Celikel, T. Neuromorphic computing hardware and neural architectures for robotics. *Sci. Robot.* **7**, eabl8419 (2022).
- Liu, S.-C. & Delbruck, T. Neuromorphic sensory systems. *Curr. Opin. Neurobiol.* **20**, 288–295 (2010).
- Risi, N., Aimar, A., Donati, E., Solinas, S. & Indiveri, G. A spike-based neuromorphic architecture of stereo vision. *Front. Neurobot.* **14**, 93 (2020).
- Kreiser, R., Renner, A., Sandamirskaya, Y. & Pienroj, P. Pose estimation and map formation with spiking neural networks: towards neuromorphic slam. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* 2159–2166 (IEEE, 2018).

- Liang, D. & Indiveri, G. A neuromorphic computational primitive for robust context-dependent decision making and context-dependent stochastic computation. *IEEE Trans. Circuits Syst. II: Express Briefs* **66**, 843–847 (2019).
- Bartolozzi, C., Indiveri, G. & Donati, E. Embodied neuromorphic intelligence. *Nat. Commun.* **13**, 1–14 (2022).
- Ma, S. et al. Neuromorphic computing chip with spatiotemporal elasticity for multi-intelligent-tasking robots. *Sci. Robot.* **7**, eabk2948 (2022).
- Perez-Peña, F., Leñero-Bardallo, J. A., Linares-Barranco, A. & Chicca, E. Towards bioinspired close-loop local motor control: a simulated approach supporting neuromorphic implementations. In *2017 IEEE International Symposium on Circuits and Systems (ISCAS)* 1–4 (IEEE, 2017).
- Donati, E., Perez-Peña, F., Bartolozzi, C., Indiveri, G. & Chicca, E. Open-loop neuromorphic controller implemented on vlsi devices. In *2018 7th IEEE International Conference on Biomedical Robotics and Biomechanics (Biorob)* 827–832 (IEEE, 2018).
- Stagsted, R. K. et al. Event-based pid controller fully realized in neuromorphic hardware: a one dof study. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* 10939–10944 (IEEE, 2020).
- Zhao, J. et al. Closed-loop spiking control on a neuromorphic processor implemented on the icub. *IEEE J. Emerg. Select. Topics Circuits Syst.* **10**, 546–556 (2020).
- Linares-Barranco, A., Perez-Peña, F., Jimenez-Fernandez, A. & Chicca, E. ED-Biorob: a neuromorphic robotic arm with fpga-based infrastructure for bio-inspired spiking motor controllers. *Front. Neurobot.* **14**, 590163 (2020).
- Csiszar, A., Eilers, J. & Verl, A. On solving the inverse kinematics problem using neural networks. In *2017 24th International Conference on Mechatronics and Machine Vision in Practice (M2VIP)* 1–6 (IEEE, 2017).
- Demby's, J., Gao, Y. & DeSouza, G. N. A study on solving the inverse kinematics of serial robots using artificial neural network and fuzzy neural network. In *2019 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)* 1–6 (IEEE, 2019).
- Gholami, A., Homayouni, T., Ehsani, R. & Sun, J.-Q. Inverse kinematic control of a delta robot using neural networks in real-time. *Robotics* **10**, 115 (2021).
- Bouganis, A. & Shanahan, M. Training a spiking neural network to control a 4-dof robotic arm based on spike timing-dependent plasticity. In *The 2010 International Joint Conference on Neural Networks (IJCNN)* 1–8 (IEEE, 2010).
- Dura-Bernal, S. et al. Cortical spiking network interfaced with virtual musculoskeletal arm and robotic arm. *Front. Neurobot.* **9**, 13 (2015).
- Tieck, J. C. V., Steffen, L., Kaiser, J., Roennau, A. & Dillmann, R. Controlling a robot arm for target reaching without planning using spiking neurons. In *2018 IEEE 17th International Conference on Cognitive Informatics & Cognitive Computing (ICCI\* CC)* 111–116 (IEEE, 2018).
- Chadderdon, G. L., Neymotin, S. A., Kerr, C. C. & Lytton, W. W. Reinforcement learning of targeted movement in a spiking neuronal model of motor cortex. *PLoS ONE* **7**, e47251(2012).
- Tieck, J. et al. Towards grasping with spiking neural networks for anthropomorphic robot hands. In *International Conference on Artificial Neural Networks* 43–51 (Springer, 2017).
- Zaidel, Y., Shalumov, A., Volinski, A., Supic, L. & Ezra Tsur, E. Neuromorphic nef-based inverse kinematics and pid control. *Front. Neurobot.* **15**, 631159. (2021).
- Davies, M. et al. Loihi: a neuromorphic manycore processor with on-chip learning. *IEEE Micro* **38**, 82–99 (2018).
- Eliasmith, C. *How to Build a Brain: A Neural Architecture for Biological Cognition* (Oxford Univ. Press, 2013).
- Menon, S., Fok, S., Neckar, A., Khatib, O. & Boahen, K. Controlling articulated robots in task-space with spiking silicon neurons. In *5th IEEE RAS/EMBS International Conference on Biomedical Robotics and Biomechanics* 181–186 (IEEE, 2014).
- Volinski, A. et al. Data-driven artificial and spiking neural networks for inverse kinematics in neurobotics. *Patterns* **3**, 100391 (2022).
- Moradi, S., Qiao, N., Stefanini, F. & Indiveri, G. A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (dynaps). *IEEE Trans. Biomed. Circuits Syst.* **12**, 106–122 (2017).
- Natale, L., Bartolozzi, C., Pucci, D., Wykowska, A. & Metta, G. icub: the not-yet-finished story of building a robot child. *Sci. Robot.* **2**, eaaq1026 (2017).
- Chevalier, G. & Deniau, J. M. Disinhibition as a basic process in the expression of striatal functions. *Trends Neurosci.* **13**, 277–280 (1990).
- Goenewegen, H. J. The basal ganglia and motor control. *Neural Plast.* **10**, 107–120 (2003).
- Hikosaka, O., Takikawa, Y. & Kawagoe, R. Role of the basal ganglia in the control of purposive saccadic eye movements. *Physiol. Rev.* **80**, 953–978 (2000).
- Marr, D. & Poggio, T. Cooperative computation of stereo disparity: a cooperative algorithm is derived for extracting disparity information from stereo image pairs. *Science* **194**, 283–287 (1976).
- Osswald, M., Ieng, S.-H., Benosman, R. & Indiveri, G. A spiking neural network model of 3d perception for event-based neuromorphic stereo vision systems. *Sci. Rep.* **7**, 1–12 (2017).

40. Athulya, P. et al. A computer vision approach for the inverse kinematics of 2 dof manipulators using neural network. In *2020 IEEE Recent Advances in Intelligent Computational Systems (RAICS)* 80–85 (IEEE, 2020).
41. El-Sherbiny, A., Elhosseini, M. A. & Haikal, A. Y. A comparative study of soft computing methods to solve inverse kinematics problem. *Ain Shams Eng. J.* **9**, 2535–2548 (2018).
42. Tikhonoff, V. et al. The icub humanoid robot simulator. In *IROS Workshop on Robot Simulators* (2012).
43. Wächter, A. & Biegler, L. T. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Math. Program.* **106**, 25–57 (2006).
44. Renner, A., Sandamirskaya, Y., Sommer, F. & Frady, E. P. Sparse vector binding on spiking neuromorphic hardware using synaptic delays. In *Proceedings of the International Conference on Neuromorphic Systems 2022* 1–5 (Association for Computing Machinery, 2022).
45. Gallego, G. et al. Event-based vision: a survey. *IEEE Trans. Pattern Anal. Mach. Intell.* **44**, 154–180 (2020).
46. Caviglia, S., Pinna, L., Valle, M. & Bartolozzi, C. Spike-based readout of posfet tactile sensors. *IEEE Trans. Circuits Syst. I Regul. Pap.* **64**, 1421–1431 (2016).
47. Tan, H. et al. Tactile sensory coding and learning with bio-inspired optoelectronic spiking afferent nerves. *Nat. Commun.* **11**, 1–9 (2020).
48. Birkoben, T., Winterfeld, H., Fichtner, S., Petraru, A. & Kohlstedt, H. A spiking and adapting tactile sensor for neuromorphic applications. *Sci. Rep.* **10**, 1–11 (2020).
49. Cai, W., Ellinger, F. & Tetzlaff, R. Neuronal synapse as a memristor: Modeling pair- and triplet-based stdp rule. *IEEE Trans. Biomed. Circuits Syst.* **9**, 87–95 (2014).
50. Yang, R. et al. Synaptic suppression triplet-stdp learning rule realized in second-order memristors. *Adv. Funct. Mater.* **28**, 1704455 (2018).
51. Bill, J. et al. Compensating inhomogeneities of neuromorphic vlsi devices via short-term synaptic plasticity. *Front. Comput. Neurosci.* **4**, 129 (2010).
52. Zendrikov, D., Solinas, S. & Indiveri, G. Brain-inspired methods for achieving robust computation in heterogeneous mixed-signal neuromorphic processing systems. Preprint at *bioRxiv* <https://doi.org/10.1101/2022.10.26.513846> (2022).
53. Hartl-Nesic, C. & Meiring, M. Computational performance of the forward and inverse kinematics of an anthropomorphic robot arm. In *Proceedings of the Joint ARW & OAGM Workshop 2019* 115–116 (2019).
54. Glover, A., Vasco, V., Iacono, M. & Bartolozzi, C. The event-driven software library for yarp-with algorithms and icub applications. *Front. Robot. AI* **4** (2018).
55. Caligiore, D. et al. Using motor babbling and hebb rules for modeling the development of reaching with obstacles and grasping. In *International Conference on Cognitive Systems* 22–23 (Citeseer, 2008).
56. Pfister, J.-P. & Gerstner, W. Triplets of spikes in a model of spike timing-dependent plasticity. *J. Neurosci.* **26**, 9673–9682 (2006).
57. Gabbott, P. & Somogyi, P. Quantitative distribution of gaba-immunoreactive neurons in the visual cortex (area 17) of the cat. *Exp. Brain Res.* **61**, 323–331 (1986).
58. Diehl, P. U. & Cook, M. Learning and inferring relations in cortical networks. Preprint at *arXiv:1608.08267* (2016).
59. Sjöström, P. J., Turrigiano, G. G. & Nelson, S. B. Rate, timing, and cooperativity jointly determine cortical synaptic plasticity. *Neuron* **32**, 1149–1164 (2001).
60. Wang, H.-X., Gerkin, R. C., Nauen, D. W. & Bi, G.-Q. Coactivation and timing-dependent integration of synaptic potentiation and depression. *Nat. Neurosci.* **8**, 187–193 (2005).

## ACKNOWLEDGEMENTS

The authors would like to acknowledge Carsten Nielsen, Nicoletta Risi, the 2019 Capocaccia Neuromorphic Workshop, and all its participants for fruitful discussions. This work was supported by the European Research Council (ERC) under the European Union's Horizon 2020 Research and Innovation Program Grant Agreement No. 724295 (NeuroAgents).

## AUTHOR CONTRIBUTIONS

J.Z. and E.D. conceived the idea. J.Z. implemented the SNN and the learning framework on the neuromorphic processor with discussions with C.B., G.I., and E.D. M.M. collected the training data for the SNN and helped J.Z. to realize the communication and control modules of the robot and perform control experiments over the robotic arm. J.Z. conducted experimental results analysis and performance evaluation of the spiking controller with the help of G.I. J.Z., E.D., and C.B. wrote the manuscript draft, and all authors reviewed, edited, and agreed with the final form of the manuscript. E.D. provided the overall supervision of the work.

## COMPETING INTERESTS

C.B. is the Associate Editor of NPJ Robotics, she has not participated in any decision about the peer-review and acceptance of the manuscript. The remaining authors declare no competing interests.

## ADDITIONAL INFORMATION

**Supplementary information** The online version contains supplementary material available at <https://doi.org/10.1038/s44182-023-00001-w>.

**Correspondence** and requests for materials should be addressed to Jingyue Zhao.

**Reprints and permission information** is available at <http://www.nature.com/reprints>

**Publisher's note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.



**Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this license, visit <http://creativecommons.org/licenses/by/4.0/>.

© The Author(s) 2023