Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Neuromorphic control of a simulated shape-memory-alloy-driven multi-legged robot using a spiking neural network

Daisuke Miki

miki.daisuke@chibatech.ac.jp

Chiba Institute of Technology

Hiroto Takigasaki

Chiba Institute of Technology

Article

Keywords:

Posted Date: July 8th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-10010940/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Version of Record: A version of this preprint was published at Scientific Reports on October 13th, 2026.
See the published version at https://doi.org/10.1038/s41598-026-75794-5.

Neuromorphic control of a simulated shape-memory-alloy-driven multi-legged robot using a spiking neural network

Daisuke Miki[1,2,*] **and Hiroto Takigasaki**[2]

[1]Department of Computer Science, Chiba Institute of Technology, Chiba, 275–0016, Japan
[2]Graduate School of Computer Science, Chiba Institute of Technology, Chiba, 275–0016, Japan
[*]miki.daisuke@chibatech.ac.jp

ABSTRACT

We propose a spiking-neural-network (SNN)-based deep reinforcement learning (DRL) method for efficient autonomous locomotion control of a starfish-inspired multi-legged soft robot driven by shape memory alloy (SMA) actuators. Conventional soft-robot controllers have predominantly relied on open-loop, pre-designed periodic signals; however, manually designing and tuning coordinated motion in systems with many degrees of freedom and strong nonlinearities remains difficult. Here, we train an SNN controller for SMAs, whose nonlinear contractile responses resemble key aspects of biological muscle. Specifically, we propose a Spiking Actor Network (SAN), in which the actor network of Twin Delayed Deep Deterministic Policy Gradient (TD3), an algorithm well suited to continuous action spaces, is replaced by an SNN. To account for the membrane potential, an internal state of SNNs, we extend the replay buffer to store and reuse membrane-potential information during training. To improve inference-time energy efficiency, we also add a regularization term that minimizes the squared membrane potentials, thereby encouraging action generation with fewer spike firings. In physics-simulation experiments on autonomous locomotion, the proposed method enabled the agent to acquire the leg coordination required to move toward a target. Analysis of the output spike trains showed that the learned firing patterns developed periodicity matched to the physical characteristics of the SMA, forming a coordinated gait that generated continuous propulsion while avoiding excessive firing. These results demonstrate the potential of autonomous neuromorphic control for energy-efficient soft-robot locomotion.

Introduction

Rigid robots, typified by industrial manipulators, offer high speed, repeatability, accuracy, and durability, and perform exceptionally well in structured environments. However, designs based on rigid components and conventional joint actuation limit their ability to adapt to unknown environments or to reproduce the flexible, compliant motions of living organisms. Consequently, robots that mimic biological body structures and locomotion mechanisms are attracting attention for applications that require direct physical interaction with humans, exploration in confined spaces, or manipulation of deformable objects.

Soft robotics has therefore advanced rapidly in recent years. Soft robots are typically composed of highly stretchable and flexible materials such as silicone and rubber. These materials allow the robot body to absorb external impacts and passively deform to conform to objects and surroundings. Moreover, soft robots employ actuation principles that differ fundamentally from those of conventional robots, using actuators such as pneumatic artificial muscles or shape memory alloys (SMAs) rather than rigid motors and gears[1–3]. In particular, SMAs are materials that generate contractile force by exploiting a thermally induced phase transformation of their crystal structure, and they are characterized by a very high output density per unit volume. Because the actuator itself is lightweight and flexible, facilitating miniaturization and quiet operation of the entire system, SMAs are widely adopted as the driving source for soft robots that mimic a wide variety of organisms. For example, Umedachi et al. developed an inchworm-like soft robot combining a flexible silicone body with SMA coils and achieved crawling locomotion using open-loop periodic SMA energization[4]. Mao et al. developed a starfish-shaped soft robot in which independently driven SMAs enabled locomotion on flat ground as well as various posture changes[5].

However, in these studies, the SMA inputs were manually designed periodic signals and the controllers operated in open loop. Because soft robots have, in principle, infinitely many degrees of freedom and SMAs exhibit strong nonlinearities, conventional model-based and open-loop controllers have limited ability to adapt to environmental changes such as obstacles and slopes, or to material degradation over time[6,7]. An SMA is an actuator that generates contractile force by exploiting a thermally induced phase transformation of its crystal structure. Its internal physical and chemical mechanism differs fundamentally from that of biological skeletal muscle, which is based on the sliding of actin and myosin filaments[8]. Nevertheless, from the viewpoint of robotic control, SMAs and biological muscle share several mechanical response characteristics. The first commonality is

that an electrical pulse input functions as the trigger for mechanical contraction. Biological muscle receives action potentials (spikes) transmitted from the axon terminals of motor neurons via the neuromuscular junction, and contraction begins when the muscle fibers depolarize. Similarly, artificial muscles using SMAs are driven directly by the Joule heat generated by externally applied current. From a control-systems perspective, despite their different energy-conversion mechanisms, both systems convert electrical input pulses into contractile force. The second commonality is that the output response to input intensity is nonlinear and exhibits threshold and saturation characteristics. Individual biological muscle fibers obey the all-or-none principle, whereby no contraction occurs unless the input stimulus exceeds a certain threshold. Moreover, once the motor units recruited by commands from the central nervous system reach their maximum, the output of the whole muscle saturates. Likewise, an SMA shows no macroscopic deformation until the temperature rise reaches the transformation start temperature, and once the transformation finish temperature is reached, the contraction ratio saturates even with further energization[9]. Such nonlinear input–output behavior is a salient characteristic of both systems. The third commonality is that intermediate states and tensions can be maintained and controlled through the temporal summation of intermittent pulse inputs. In biological muscle, when high-frequency spikes are input continuously from motor neurons, the individual twitches overlap, producing a sustained and powerful contractile state. Living organisms achieve smooth and precise control of muscle tension by adjusting this spike-firing frequency (rate coding). Similarly, in the control of SMAs, it is possible to maintain an intermediate contraction length or a constant tension by intermittently energizing the wire with high-frequency pulses such as pulse-width modulation (PWM), thereby accumulating and maintaining thermal energy within the system[10].

To adaptively control SMAs with such characteristics in unknown environments, a control method that can autonomously acquire appropriate inputs according to the environment is required. As a robot-control method for unknown environments, deep reinforcement learning (DRL)[11,12] has attracted particular attention in recent years. DRL has been actively studied in game AI[13,14] and robot control[15,16], and is known to be capable of acquiring action strategies that are robust to environmental changes. Haarnoja et al. proposed the Soft Actor-Critic (SAC) algorithm[17] and showed that, in learning to walk with multi-legged robots, control with high robustness to noise and external disturbances is possible. Furthermore, they reported that a policy learned in a simulation environment can be transferred directly to a real-world quadruped robot without resorting to techniques such as domain randomization[18], achieving stable walking even on unknown terrain[15]. In addition, when using neural networks for robot control, it is necessary to pay attention to energy efficiency at inference time. This is because the inference of the increasingly deep and complex neural networks that have achieved remarkable results in recent years is known to consume substantial power[19], and this problem becomes particularly pronounced in the control of robots whose onboard energy is limited. As one effective solution to this problem, spiking neural networks (SNNs), which mimic the information-processing mechanisms of the biological brain, have been attracting attention[20,21]. Because SNNs process input information as time-series spikes (binary signals), they have a high affinity with event-based processors (neuromorphic chips). Owing to this characteristic, it has been shown that they can perform inference with lower power consumption than conventional artificial neural networks (ANNs). A prior study[22] reported that, in a speech-recognition task, Intel Loihi[23], a neuromorphic chip, achieved substantially greater power efficiency than a conventional GPU. Tang et al. proposed a method that replaces the actor network of an actor–critic DRL algorithm with an SNN[24], and showed that performance comparable to or better than that of conventional DRL can be achieved in robot-walking tasks. On this basis, we propose an SNN-based deep reinforcement learning method for controlling SMA-driven soft robots.

Methods

SMA-driven multi-legged robot agent

We modeled a starfish-inspired multi-legged robot in the MuJoCo physics simulator[25] and implemented an SMA-based drive mechanism (Figure 1(a) and (b)). The proposed robot model has an axisymmetric structure consisting of a central torso and five legs arranged radially at 72-degree intervals on the horizontal plane. To allow free translational and rotational motion in space, the torso is connected to the world coordinate frame via a six-degree-of-freedom free joint. Each leg is a multi-joint link mechanism consisting of six hinge joints connected in series; the rotation axis of each joint is fixed in the lateral (width) direction of the leg, so that each is a one-degree-of-freedom joint that performs only pitch-direction bending. To realize a curling motion toward the robot's center while preventing structural self-interference, the range of motion of each joint is limited to 0–30 degrees, allowing a continuous bending of up to 180 degrees per leg. Thus, the robot as a whole has a total of 30 internal-joint degrees of freedom. Each leg is actuated by an SMA wire acting as a flexor and a passive elastic element acting as an extensor. The SMA wire is routed from the base to the tip of each leg along a path offset by a fixed distance from the rotation axis of each joint, so that the linear contractile tension of the SMA is converted into bending-direction torque at each joint. Meanwhile, each joint incorporates a passive restoring torque with a fixed spring constant, so that when energization of the SMA is stopped and the contractile force decreases, this elastic force passively extends the leg back to its initial straight posture.

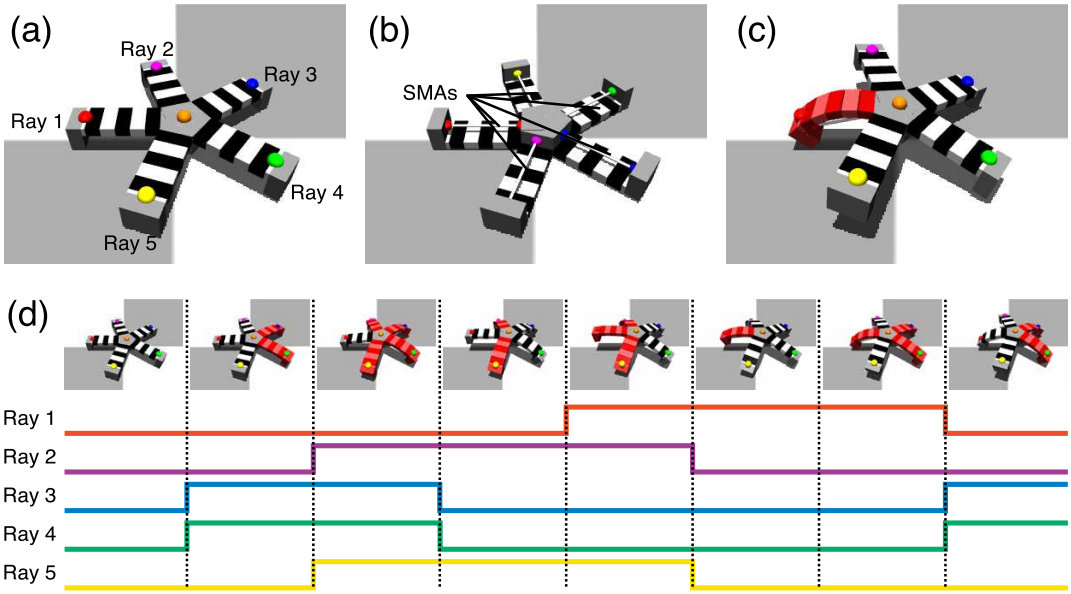


Figure 1. Multi-legged robot constructed in MuJoCo. (a) Top view and the arrangement of each ray. (b) Bottom view and the arrangement of the SMAs. (c) Behavior when the SMA of Ray 1 is contracted; red indicates that current is being applied to the SMA. (d) Locomotion of the robot when the rays are contracted and extended in sequence; red indicates that current is being applied to the SMA.

The SMA tension-control model used in this study is a simplified form of the one-dimensional constitutive model proposed by Brinson[9]. The Brinson model uses the phase-transformation progress ξ as an internal variable and is based on the constitutive relation $\varepsilon = \varepsilon_e + \varepsilon_{tr}(\xi)$, which describes the coupling among stress σ, temperature T, and phase fraction. Here, ε_e is the elastic strain and ε_{tr} is the transformation strain; ξ is given as a function of stress and temperature, $\xi(\sigma, T)$, and bidirectional thermomechanical coupling based on the Clausius–Clapeyron relation is explicitly considered. For use in real-time physics simulation and reinforcement-learning environments, however, we prioritized computational efficiency and numerical stability. We therefore adopted a simplified model that neglects explicit stress dependence and approximates the phase-transformation progress as a function of temperature alone. Because the aim of this study was to evaluate robot-level locomotion generation and control rather than to reproduce the detailed material behavior of SMAs, we omitted the full elastic–transformation strain constitutive law of the Brinson model[9]. Instead, we used the following mechanical approximation, in which the generated SMA force F is proportional to the phase-transformation progress $\xi \in [0, 1]$:

$$F = F_{\max}\,\xi \tag{1}$$

Here, $F_{\max}$ is the maximum generated force, which was set to 10 N. Furthermore, to represent the thermal hysteresis characteristic of SMAs, the temperature-dependent phase state ξ is defined by the following equation, using a sigmoid function with different transition temperatures for the heating and cooling processes, following the method of Jayender et al.[26]:

$$\xi(T) = \frac{1}{1 + \exp\left(-\frac{T - T_{\text{trans}}}{k_\sigma}\right)} \tag{2}$$

Here, T_{trans} is the central temperature of the phase transition; it was set to 50 °C during heating ($dT/dt \geq 0$) and to 40 °C during cooling. k_σ determines the steepness of the transition and was set to 3.0. For computational efficiency and numerical stability, the phase-transformation progress was assumed to instantaneously follow the temperature variation ($\xi = \xi(T)$). This assumption is reasonable when the time scale of the phase transformation is sufficiently short relative to the macroscopic motion of the robot. The time evolution of the internal temperature T of the wire is approximated by the following energy-conservation equation, which accounts for Joule heating from the input current and heat dissipation to the environment:

$$C\frac{dT}{dt} = \alpha u^2 - k_{th}(T - T_{\text{amb}}) \tag{3}$$

Here, the control input $u \in [0,1]$ is a normalized input assumed to be proportional to the drive current as $I = uI_{\mathrm{nom}}$. C is the heat capacity of the SMA wire (0.005 J/K), k_{th} is the heat-transfer coefficient (0.02 W/K), and T_{amb} is the ambient temperature (20 °C). $\alpha = R_{\mathrm{tot}} I_{\mathrm{nom}}^2$ is the electrical-to-thermal conversion coefficient calculated from the electrical resistance per unit length (900 Ω/m) and the rated drive current I_{nom} (0.3 A). To avoid numerical instability associated with the reinforcement-learning time step, the computed raw temperature gradient $\dot{T}_{\mathrm{raw}}$ was clipped to obtain the effective temperature change rate $\dot{T}$:

$$\dot{T} = \begin{cases} \min(\dot{T}_{\mathrm{raw}},\ \dot{T}^{+}_{\mathrm{max}}) & (\dot{T}_{\mathrm{raw}} \geq 0) \\ \max(\dot{T}_{\mathrm{raw}},\ -\dot{T}^{-}_{\mathrm{max}}) & (\dot{T}_{\mathrm{raw}} < 0) \end{cases} \tag{4}$$

The upper limit during heating, $\dot{T}^{+}_{\mathrm{max}}$, was set to 150 °C/s, and the upper limit during cooling, $\dot{T}^{-}_{\mathrm{max}}$, was set to 20 °C/s. This clipping emulates the physical upper limit on the temperature gradient while also preventing the simulation from diverging. Finally, the instantaneous phase state ξ is determined from the updated temperature T, and the posture change and locomotion of the robot are generated through the tendons in the physics engine. Figure 1(c) shows the contraction of the SMA of a single ray. Figure 1(d) reproduces the ray contraction–extension pattern used in the study by Mao et al.[5], demonstrating that the robot is capable of locomotion.

Reinforcement learning with spiking actor network

We used an SNN to control the starfish-shaped robot and adopted Twin Delayed Deep Deterministic Policy Gradient (TD3)[27], which is well suited to continuous-action control tasks, as the reinforcement-learning algorithm. In conventional TD3, the actor network π_ϕ takes only the agent's state s as input and estimates the corresponding action a based on $\pi_\phi(s)$. To train the SNN within TD3, the actor network must account for the membrane potential m, which reflects the dynamics of the spiking neurons; the action is therefore estimated from $\pi_{\phi,m}(s)$. In this respect, the proposed actor differs from that used in conventional TD3[27]. The proposed method also differs from prior work[24,27] in two main respects. First, the actor is implemented as a spiking actor network. Second, the contents stored in the replay buffer are extended to include SNN internal states. These details are described in the following subsections.

Spiking actor network

SNNs are third-generation neural networks that mimic the information-processing mechanisms of biological nervous systems[28]. Whereas conventional ANNs process information using continuous-valued activation functions, SNNs transmit information through discrete pulse signals, or spikes, that emulate neuronal firing. Information in an SNN is represented by the presence or absence of spikes, their firing timing, or the firing frequency within a given time window. This representation enables event-driven computation, in which operations are performed only when spikes occur, thereby reducing the use of computational resources[20]. The binary nature of spike-based inference can be exploited particularly effectively by neuromorphic chips specialized for SNN computation. For example, neuromorphic processors typified by Intel Loihi[23] implement spike-based asynchronous processing at the hardware level and have been reported to enable computation with extremely high power efficiency compared with ANN inference using conventional GPUs or CPUs[22]. In addition, in an SNN each neuron holds a membrane potential as an internal state and exhibits dynamic behavior that varies continuously over time. For this reason, SNNs have properties close to those of recurrent neural networks (RNNs) and can reflect the history of past inputs in their internal state[29]. Such characteristics are advantageous in control problems that require time-series data processing and information representation based on temporal coding, and their application to robot control using reinforcement learning and to continuous-time control tasks is particularly anticipated[24].

In SNNs, various neuron models are used to describe the temporal state changes and spike firing of neurons. We used the leaky integrate-and-fire (LIF) model[30], which provides a useful balance between computational efficiency and biological plausibility and is widely adopted in engineering applications of SNNs. In the LIF neuron model, the membrane potential $V_i(t)$ is the internal state and the input current $I_j(t)$ is the input; the membrane potential evolves through the weighted connections between neurons. A neuron fires when its membrane potential exceeds a threshold ϑ, propagating a spike to external neurons. The temporal change of the membrane potential is expressed by the following differential equation:

$$\tau \frac{dV_i(t)}{dt} = -V_i(t) + \sum_j w_{ij} I_j(t). \tag{5}$$

Here, $V_i(t)$ is the membrane potential of the i-th neuron at time t, $I_j(t)$ is the current due to the spike output from the j-th neuron, and τ is the time constant. Furthermore, when the membrane potential exceeds ϑ, the potential is set to $V_i - \vartheta$:

$$V_i \leftarrow V_i - \vartheta \text{ when } V_i \geq \vartheta. \tag{6}$$

By approximating the solution of Eq. (5) using the Euler method, the membrane potential can be expressed as follows:

$$V[t] = \beta V[t-1] + (1-\beta) I[t], \tag{7}$$

where $\beta = e^{-1/\tau}$ is the decay rate of $V[t]$. For simplicity, $(1-\beta)$ is replaced by a learnable weight W, and the input current $I[t]$ is expressed as $I[t] = WX[t]$. Here, $X[t]$ is a single input spike. Furthermore, taking into account the reset of the membrane potential due to spike firing, the following equations are obtained:

$$V[t] = \beta V[t-1] + X[t] - \vartheta S[t-1], \tag{8}$$

$$S[t] = \Theta(V[t] - \vartheta), \tag{9}$$

where $S[t] \in \{0,1\}$ is the output spike generated by the spiking neuron using the step function Θ. This formulation allows the LIF neuron to internally retain the history of past inputs as the membrane potential, giving it properties close to those of a recurrent neural network. These characteristics are advantageous in control problems that deal with time-series information and in tasks involving continuous-time state transitions in reinforcement learning.

An overview of the network structure of the proposed method is shown in Figure 2. We adopted an actor–critic deep reinforcement learning algorithm based on TD3[27] and configured the TD3 actor network as an SNN. The Spiking Actor Network (SAN) proposed in this study consists of three components: an encoder that converts continuous-valued state information into spike trains, a fully connected spiking neural network (FC SNN) that processes the spike information, and a decoder that generates continuous-valued actions from the spike activity. The details of each component are described below.

First, the encoder receives the observations obtained from the environment as input, applies a linear transformation with learnable weights and biases, and then activates them using the hyperbolic tangent function (tanh). The activation output is then thresholded at 0, and only positive values are retained. Next, based on population coding, these values are replicated across 256 spiking neurons. Noise sampled from the uniform distribution $U(0,1)$ is added to each replicated value, and a spike train is generated in which a spike fires (value 1) when a prescribed threshold is exceeded and does not fire (value 0) otherwise.

Next, the fully connected SNN (FC SNN) part consists of an input layer, two hidden layers, and an output layer, with all layers fully connected to one another. Each hidden layer contains 256 channels of LIF spiking neurons. The input layer receives the spike train of dimension $N_{env} \times 256$ output from the encoder. Here, N_{env} denotes the dimensionality of the environmental observation data. Meanwhile, the output layer uses population coding to generate a spike train of dimension $N_{act} \times 256$. Here, N_{act} is the dimensionality of the action space.

Finally, the decoder converts the spike train output from the FC SNN into continuous-valued actions. The spike train corresponding to each action dimension is converted into a continuous value by rate coding. These outputs are used as the intensity of the control signals applied to the actuators mounted on each leg of the soft robot.

Replay buffer

In off-policy deep reinforcement learning, including the standard TD3 algorithm, transitions containing the state s_t, action a_t, reward r_t, next state s_{t+1}, and terminal flag d_t at each time step t are stored in a replay buffer. The replay buffer enables the agent to randomly sample and reuse past experiences for learning, reducing temporal correlations between samples and stabilizing training[27]. Because this study uses an SNN as the actor network, the membrane potential m_t, which is the internal state of the neurons, must also be considered during learning. In an SNN, the membrane potential functions as a hidden state that transitions over time, and the current output depends on the history of past inputs. For this reason, the conventional TD3 framework, which stores only the single-step physical state transition (s_t, a_t, r_t, s_{t+1}), cannot accurately reproduce the dynamic behavior of the SNN at sampling time, which can destabilize learning[24,31].

Therefore, in addition to $(s[t], a[t], r[t], s[t+1], d[t])$, which are stored in the replay buffer in conventional TD3, we also stored membrane-potential information. Specifically, the membrane potential $m[t]$ at time t and the membrane potential $m[t+1]$ at the next time $t+1$ are stored in the replay buffer and reused during the learning process. By using the stored membrane potentials during learning, the internal state of the SNN in past states can be accurately reproduced, allowing computations involving temporal dependencies to be performed appropriately.

Actor-critic network parameter update

The critic network (Q-network) takes the state **s** and the action **a** as input and outputs the expected return (Q-value) for that state–action pair. The loss function of the critic network in this study is defined as follows, based on the TD3 algorithm:

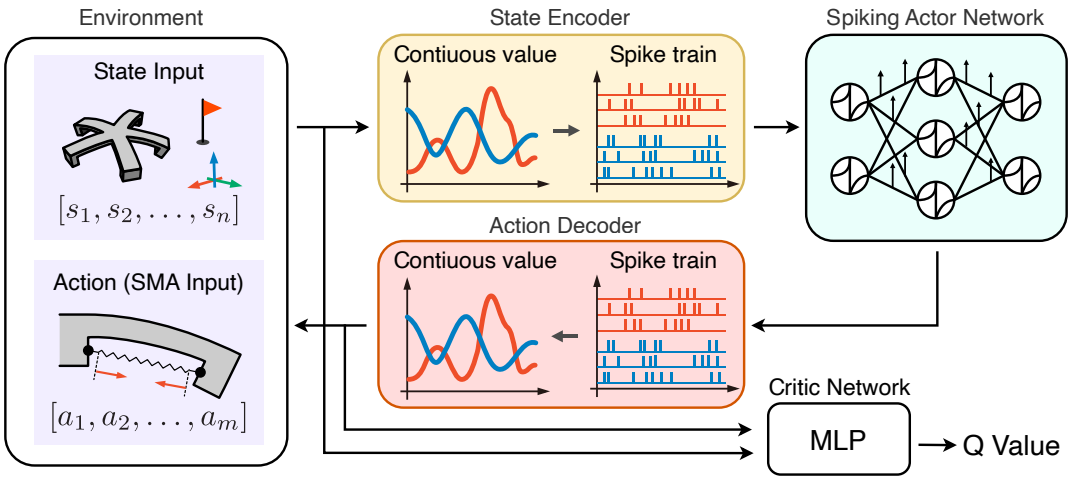


Figure 2. Interaction between the environment and the actor. The continuous-valued state observed from the environment is converted into a spike train by the state encoder, and the action is estimated by the SAN. The spike train output from the SAN is converted into continuous values by the action decoder, which updates the state of the environment. The state and action are input to the critic network, composed of an MLP, to estimate the Q-value.

$$L(\theta_i) = \frac{1}{B}\sum_{n=1}^{B}\left(Q_{\theta_i}(\mathbf{s}_n, \mathbf{a}_n) - y_n\right)^2 \quad (i = 1, 2) \tag{10}$$

Here, B is the mini-batch size and θ_i are the parameters of the two critic networks. y_n is the target value to be learned. By combining the suppression of overestimation bias in TD3[27] with target policy smoothing, y_n is computed as follows:

$$\begin{aligned} \tilde{\mathbf{a}}_n &= \mathrm{clip}(\pi_{\phi'}(\mathbf{s}'_n, \mathbf{m}'_n) + \varepsilon_n, -c, c), \quad \varepsilon_n \sim \mathcal{N}(0, \sigma^2\mathbf{I}) \\ y_n &= r_n + \gamma(1 - d_n) \min_{j\in\{1,2\}} Q_{\theta'_j}(\mathbf{s}'_n, \tilde{\mathbf{a}}_n) \end{aligned} \tag{11}$$

Here, ϕ' and θ'_j are the parameters of the target actor and target critic networks, respectively. $\mathbf{s}_n, \mathbf{a}_n, r_n, d_n$ denote the state, action, reward, and episode-termination flag (1 at the terminal state and 0 otherwise) in the n-th transition sampled from the replay buffer. Moreover, $\mathbf{m}'_n$ is the membrane-potential vector when the next state $\mathbf{s}'_n$ is input to the target actor network (SNN). Meanwhile, the actor network updates its parameters ϕ so as to maximize the Q-value evaluated by the critic network. The loss function of the spiking actor network proposed in this study combines the deterministic policy gradient with a regularization term for suppressing spike firing, and is defined as follows:

$$L(\phi) = \frac{1}{B}\sum_{n=1}^{B}\left(-Q_{\theta_1}(\mathbf{s}_n, \pi_\phi(\mathbf{s}_n, \mathbf{m}_n)) + \alpha\|\mathbf{m}_n\|_2^2\right) \tag{12}$$

The first term of the loss function, $-Q_{\theta_1}(\cdot)$, is the policy-gradient term that maximizes the expected Q-value by minimizing its negative value; Q_{θ_1} was used for the actor update. The second term, $\alpha\|\mathbf{m}_n\|_2^2$, is an L2 regularization term on the membrane potentials of the neurons in the actor network, where $\|\mathbf{m}_n\|_2^2 = \sum_{k=1}^{M} m_{n,k}^2$ represents the sum of squared membrane potentials of all M neurons when the state $\mathbf{s}_n$ is input. In the LIF neuron model, penalizing excessive increases in membrane potential reduces how frequently the membrane potential reaches the firing threshold. As a result, spike firings at inference time are suppressed, promoting energy-efficient action generation that exploits the characteristics of the SNN. The coefficient α controls the contribution of the regularization term and was set to suppress spike firing without impairing policy learning.

The weight parameters of the SNN are learned using the surrogate gradient (SG) method[29]. In supervised learning using backpropagation (BP) and in policy-gradient computation for reinforcement learning, the gradient with respect to the weight parameters W must be computed in order to minimize the loss function L. The gradient in backpropagation through time (BPTT), which involves unrolling along the time direction, is described by the chain rule as follows:

$$\frac{\partial L}{\partial W} = \sum_t \frac{\partial L}{\partial S_t}\frac{\partial S_t}{\partial V_t}\frac{\partial V_t}{\partial W} \tag{13}$$

Here, S_t is the binary output representing the presence or absence of a spike at time t, and V_t is the membrane potential. Because spike firing in the LIF model is defined by $S_t = \Theta(V_t - \vartheta)$ (where Θ is the Heaviside step function and ϑ is the firing threshold), its derivative is given by the following equation:

$$\frac{\partial S_t}{\partial V_t} = \delta(V_t - \vartheta) \tag{14}$$

Because this equation contains the Dirac delta function δ, $\partial S_t / \partial V_t$ is zero except when the value of V_t exactly coincides with the threshold ϑ. As a result, the gradient vanishes, and standard BP cannot be applied directly. To circumvent this non-differentiability problem, we adopted the SG method in this study. Specifically, during forward propagation, the step function Θ is used to generate exact spikes, whereas during backpropagation, a smooth surrogate function is applied to approximate the gradient computation. Although various surrogate functions have been proposed, we adopted the fast sigmoid function, which is widely used in methods such as that proposed by Zenke et al.[32]. The fast sigmoid function $\tilde{S}(V_t)$ and its derivative are defined as follows:

$$\tilde{S}(V_t) = \frac{V_t - \vartheta}{\frac{1}{k} + |V_t - \vartheta|} \tag{15}$$

$$\frac{\partial \tilde{S}(V_t)}{\partial V_t} = \frac{k}{(1 + k|V_t - \vartheta|)^2} \tag{16}$$

Here, k is a scale parameter that determines the steepness of the function's gradient. Whereas the derivative of the standard sigmoid function requires exponential computation, the fast sigmoid function can be computed using only absolute values and basic arithmetic operations, reducing computational cost in tasks with many time steps, such as deep reinforcement learning. Furthermore, because the fast sigmoid function has broad tails, the gradient is unlikely to vanish completely even when the membrane potential V_t is far from the threshold ϑ. Using this surrogate derivative mitigates gradient vanishing across deep layers and long time horizons, enabling stable weight updates both over time and across layers. We used the step function in Eq. (9) for forward propagation and the derivative of the fast sigmoid function in Eq. (16) for gradient computation during backpropagation, thereby enabling stable and efficient SNN learning in a continuous action space.

Experiments and results

Motion evaluation of the multi-legged robot

We first examined the relationship between the control signal applied to the soft robot, represented by the spike-firing rate, and the resulting postural change. First, we randomly generated spike trains based on population coding and supplied them as input to the SMA actuator of a single ray constituting the robot. In the experiment, one cycle consisted of applying the input at a constant firing rate for 20 seconds and then waiting (input off) for 30 seconds, and the response was measured while increasing the firing rate from 0.1 to 1.0 in increments of 0.1. Figure 3 shows the relationship between the firing rate of the applied spike train and the bending angle that the base of the ray (the junction between the main body and the SMA) makes with the plane of motion. Marked ray contraction emerged when the firing rate exceeded approximately 0.5, and the contraction saturated at firing rates of 0.8 and above. Figure 4 summarizes the effects of the firing rate and the input application time on the bending angle under each input condition. As shown in Figure 4(a), after the input was turned off, under all firing-rate conditions the ray extended (relaxed) almost back to its original initial state, reaching 2.0% of the maximum bending angle at $T_{ext} = 20$ s and 1.2% at $T_{ext} = 25$ s. Furthermore, Figure 4(b) shows that the output response (bending angle) to the input intensity (firing rate) exhibits strong nonlinearity and has clear threshold and saturation characteristics. This response resembles the nonlinear mechanical behavior of biological muscle described in the Introduction. These response characteristics suggest important constraints when designing periodic motions such as walking. That is, if contraction and extension are repeated with a period longer than T_{ext}, a non-actuated interval arises between the time the SMA has fully extended and the start of the next contraction, reducing the robot's locomotion speed. Conversely, if contraction and extension are performed with a period shorter than T_{ext}, the next contraction begins before the SMA has fully extended; this not only limits the range of motion of the ray and reduces the displacement per cycle but also leads to wasteful spike firing (excessive energy consumption). Therefore, to achieve highly efficient locomotion, it is necessary to find an optimal drive period that matches the actuator dynamics of the SMA. In the following experiments, we examined whether the reinforcement-learning agent could autonomously acquire this drive timing and the coordinated motion of each ray through interaction with the environment. Because these experiments were performed

in a physics simulator, parameters such as the time constant may differ in a real system owing to SMA thermodynamics, including cooling rate, silicone-body elasticity, ambient temperature, and floor friction. Thus, the learned policy cannot be transferred directly to a physical robot without further validation. Nevertheless, the simulator captures the essential behavioral characteristics required to evaluate whether a learning-based controller can exploit the nonlinear dynamics of SMA actuators.

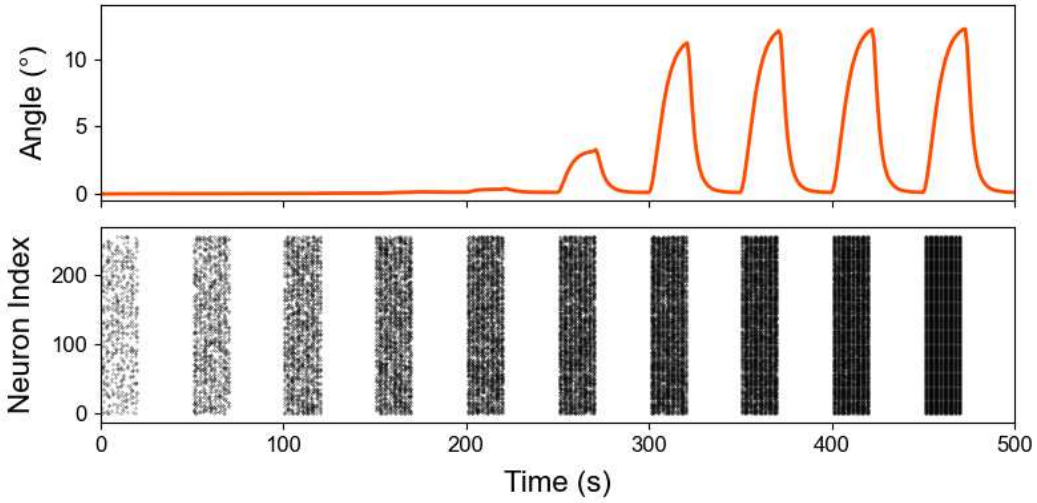


Figure 3. Relationship between the firing rate and the bending angle.

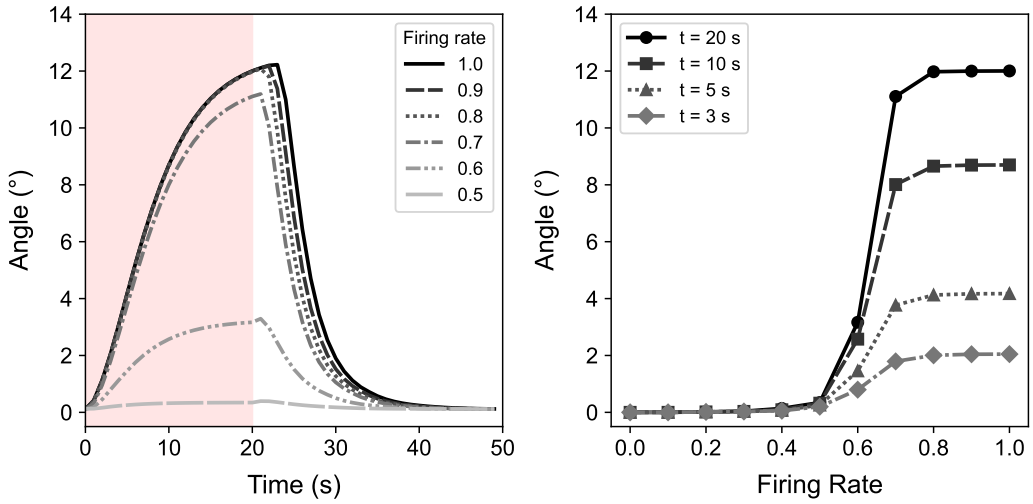


Figure 4. Response to spike inputs. (a) Time course of the bending angle for each firing rate; the shaded region indicates the application period of the input spike train. (b) Effect of the input intensity on the bending angle.

Policy evaluation in the autonomous locomotion task

To verify the effectiveness of the SNN-actor reinforcement-learning method for controlling the proposed SMA-driven multi-legged robot, we conducted learning experiments on autonomous locomotion tasks in MuJoCo. During learning, the agent selects actions based on the actor network and interacts with the environment at discrete time steps. State observation and the application of actions to the environment are performed at 1000-ms intervals; because the simulator's internal time step is 10 ms and the frame-skip count is set to 100, the agent maintains the same action during that interval. The maximum number of steps per episode was set to 1000, and at the end of each episode the robot's position and posture and the temperature state of the SMA (initial ambient temperature 20 °C) were initialized. At each step, the agent observes the current state of the robot from the environment. This state space is composed of a nine-dimensional real-valued vector consisting of the start-to-end distances

of the SMAs placed on each ray (5 dimensions), the relative planar position of the tip of the first ray with respect to the center of the torso (2 dimensions), and the relative planar vector from the center of the torso to the goal position (2 dimensions). To stabilize learning, the state vector was dynamically normalized using the mean and variance of the observations accumulated in the replay buffer before being input to the network. The Spiking Actor Network (SAN) encodes the input normalized state into a 256-dimensional spike train based on population coding. The hidden layers employ LIF neurons with 256 hidden nodes, and the output layer produces the control signal $a_t \in [0,1]^5$ for the SMA actuators of each of the five legs by decoding the firing frequency. The decay rate β of the membrane potential of the LIF neurons was initialized by random sampling from a uniform distribution over the interval $[0.3, 0.7]$; the scale parameter k of the surrogate gradient used in backpropagation was set to 5.0, and the weight α of the regularization term that prevents divergence of the membrane potential was set to 1.0×10^{-3}. Meanwhile, for the critic network used in the value estimation of TD3, we used a multilayer perceptron (MLP) with two fully connected layers of 256 hidden nodes, with ReLU as the activation function. The reward function $R^{(t)}$ for training autonomous locomotion was defined as the sum of three terms: a progress reward in the goal direction, a distance penalty based on the remaining distance to the goal, and an action penalty that suppresses energy consumption:

$$R^{(t)} = c_{\text{reward}} \left(\Delta \mathbf{x}_c^{(t)} \cdot \mathbf{n}_{\text{goal}} \right) - c_{\text{dist}} \left\| \mathbf{x}_{\text{goal}} - \mathbf{x}_c^{(t)} \right\|_2 - c_{\text{act}} \| \mathbf{a}^{(t)} \|_2^2 \tag{17}$$

Here, $\Delta \mathbf{x}_c^{(t)}$ is the displacement vector of the agent's center of mass at time t, $\mathbf{n}_{\text{goal}}$ is the unit vector pointing from the robot's center toward the goal, and $\mathbf{x}_c^{(t)}$ and $\mathbf{x}_{\text{goal}}$ represent the robot's center position and the goal position, respectively. The first term on the right-hand side is the main reward that promotes efficient movement in the goal direction, with a weight coefficient c_{reward} set to 1.0×10^3. The second term is a penalty term proportional to the Euclidean distance from the robot's center to the goal, introduced to continuously encourage behavior that approaches the goal; its weight coefficient c_{dist} was set to 1.0×10^{-1}. The third term is a penalty term proportional to the square of the L2 norm of the input control-signal vector $\mathbf{a}^{(t)}$ to each actuator, which serves to suppress wasteful power consumption and induce energy-efficient motion; its weight coefficient c_{act} was set to 1.0×10^{-3}.

Learning was carried out over 500,000 steps of exploration. To promote exploration of the environment, Gaussian noise with a standard deviation of 0.1 is added to the SNN output only during learning. For policy updates, an experience replay buffer of size 100,000 was used with a batch size of 100. The discount factor γ for future rewards was set to 0.99, and the smoothing update coefficient τ of the target network to 0.995. Adam[33] was used for network optimization, with learning rates of 1.0×10^{-4} for the SNN actor and 1.0×10^{-3} for the MLP critic. In addition, as parameters specific to TD3, the actor's delayed update interval (policy delay) was set to 2, and the standard deviation of the target policy noise to 0.2 (clipping range ± 0.5). In the following trajectory and spike analyses, to compare the behavioral changes accompanying the learning process, we visualized representative evaluation episodes using checkpoints at 30,000, 50,000, and 300,000 steps.

With the robot's initial position fixed near the origin, we analyzed the locomotion trajectory, the output spike trains, and the relative activity phases among the rays generated by the trained SAN.

Straight-line locomotion task First, the locomotion trajectory and the raster plot of the SAN output for the straight-line task, in which the goal position relative to the robot was set to $(\infty, 0)$ mm, are shown in Figure 5. Figure 5(a), (b), and (c) show the results using the policies at 30,000, 50,000, and 300,000 learning steps, respectively. At the early learning stage of 30,000 steps, no clear periodic structure was observed in the raster plot, and the position of the robot's center of mass barely changed. At 50,000 steps, forward movement began and a periodic firing pattern spanning multiple rays emerged, but the amount of movement was still limited. At 300,000 steps, the robot moved forward continuously and exhibited a large straight-line displacement compared with the early and middle stages of learning. From the raster plot at this point, an ordered, coordinated motion was confirmed in which, rather than all rays contracting simultaneously, multiple groups of rays fired with time offsets.

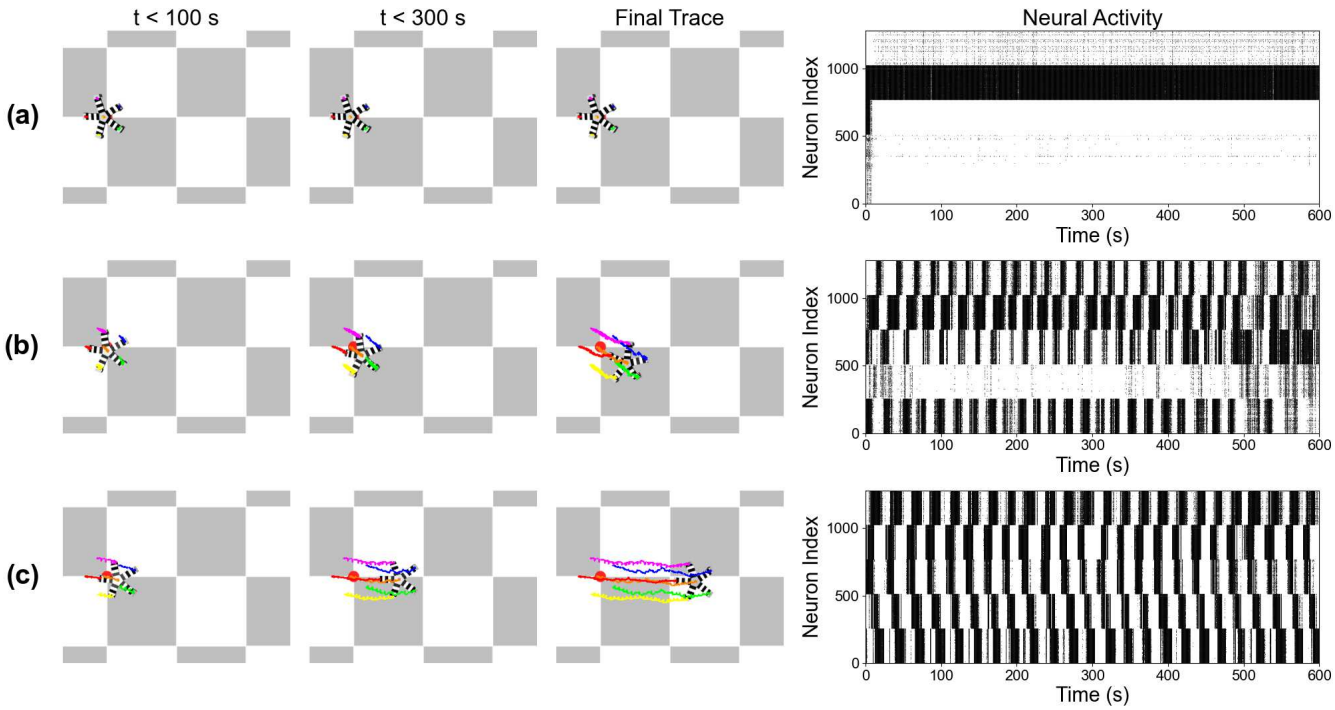


Figure 5. Locomotion trajectory of the robot and raster plot of the SAN output during the straight-line locomotion task. The red dot indicates the robot's initial position. Learning steps: (a) 30,000, (b) 50,000, (c) 300,000.

To elucidate the mechanism of this gait, we performed frequency analysis and cross-correlation analysis to quantify the activity dynamics of the neuron populations in each ray (Ray 1–5) and their relative phase relationships. In the frequency analysis, after removing the linear trend from each time series, we applied the fast Fourier transform (FFT) to compute the frequency spectrum. In addition, to identify the time delays among the rays, we computed the cross-correlation function with Ray 1 as the reference for each zero-baseline-normalized signal. To remove small noise components and extract macroscopic activity patterns, the firing rate was smoothed using a Gaussian filter with a standard deviation of $\sigma = 1.0$. The time lag at which the obtained cross-correlation waveform attained its maximum was evaluated as a fraction of the reference period and normalized to the range of $\pm 180^\circ$ to convert it into a phase difference. As shown in Figure 6, at 30,000 steps no clear common periodicity was confirmed among the rays. On the other hand, at 50,000 steps, Rays 1, 3, 4, and 5 showed a peak at 0.0467 Hz, corresponding to a period of approximately 21 s. However, Ray 2 showed a peak at 0.0017 Hz, indicating that at this stage not all rays were synchronized to the same period. At 300,000 steps, all rays showed a peak at 0.0400 Hz, with a dominant contraction period of approximately 25 s. Considered together with the results of the previous section, this suggests that the agent acquired a drive period that allows the next contraction to begin after the SMA has sufficiently extended, and formed a coordinated phase relationship among multiple rays based on that period. Visualizing the phase differences in a polar coordinate system, no clear phase relationship had formed at 30,000 steps, whereas at 50,000 steps activity with a common period began to appear in multiple rays except Ray 2. At 300,000 steps, Ray 2 and Ray 5 were located on the leading side relative to Ray 1, while Ray 3 and Ray 4 exhibited a large phase difference close to anti-phase. These results are similar to the contraction period and inter-ray phase relationships reported for the SMA-driven starfish-shaped robot of Mao et al.[5], indicating that the agent autonomously acquired, through reinforcement learning, both a contraction period suited to the physical characteristics of the SMA and a coordinated phase relationship among the rays.

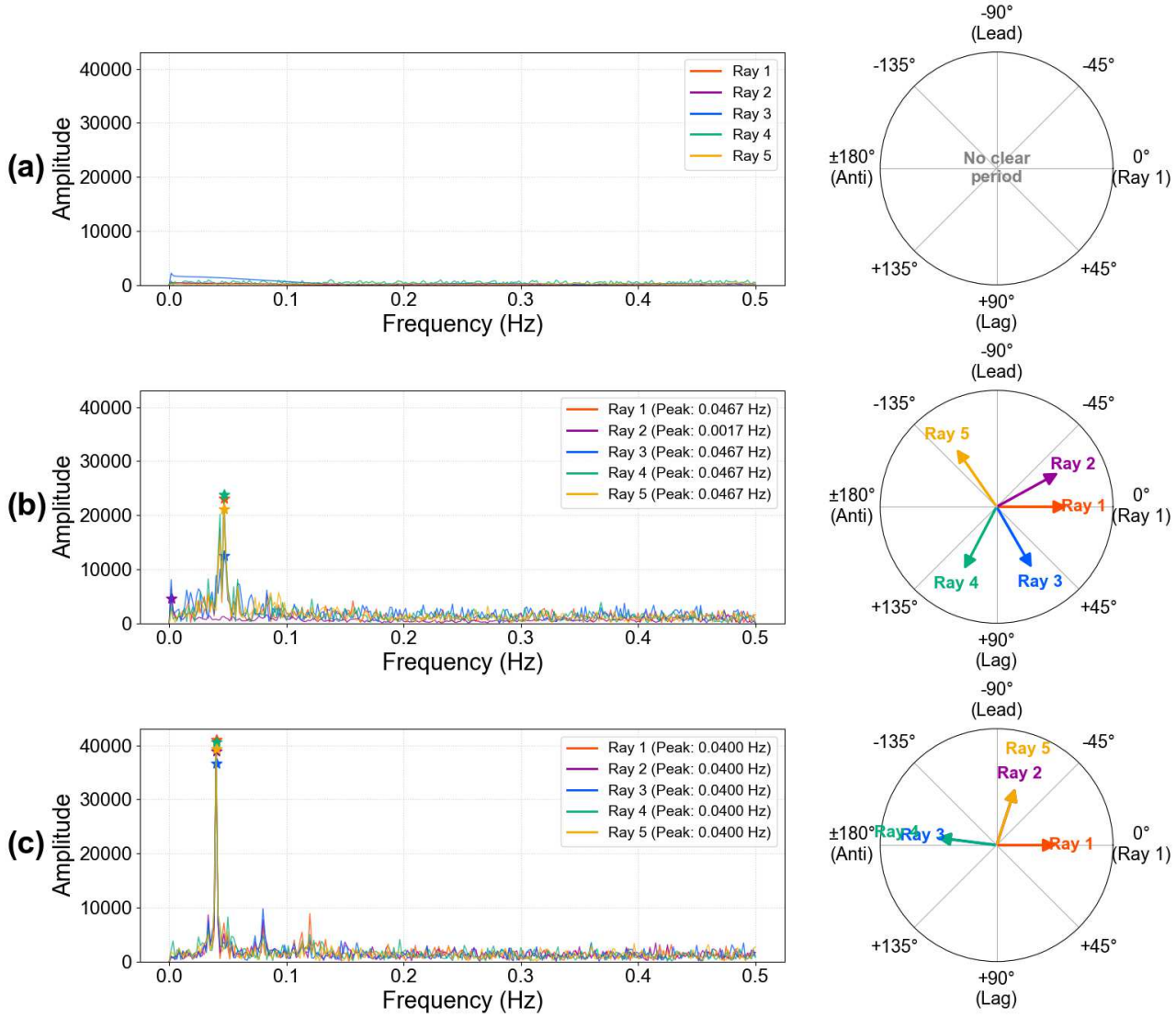


Figure 6. Frequency analysis and phase relationships of the SAN output when the goal position was set to $(\infty, 0)$ mm. The left column shows the FFT amplitude spectrum of the population firing activity of each ray, and the right column shows the polar plot of the phase difference relative to Ray 1. Learning steps: (a) 30,000, (b) 50,000, (c) 300,000.

Goal-reaching task Next, the locomotion trajectory and the raster plot of the SAN output under the condition where the goal position relative to the robot was set to $(150, -50)$ mm are shown in Figure 7. Figure 7(a), (b), and (c) show the results using the policies at 30,000, 50,000, and 300,000 learning steps, respectively. At 30,000 steps, the position of the robot's center of mass barely changed, and no clear periodic structure was observed in the SAN output. At 50,000 steps, movement toward the goal appeared along with a periodic firing pattern, but the trajectory still stopped short of the target position. At 300,000 steps, the robot moved stably toward the goal and reached its vicinity. In the raster plot, the firing timings of the individual rays were temporally separated, and a coordinated firing structure was confirmed in which multiple groups of rays were driven sequentially.

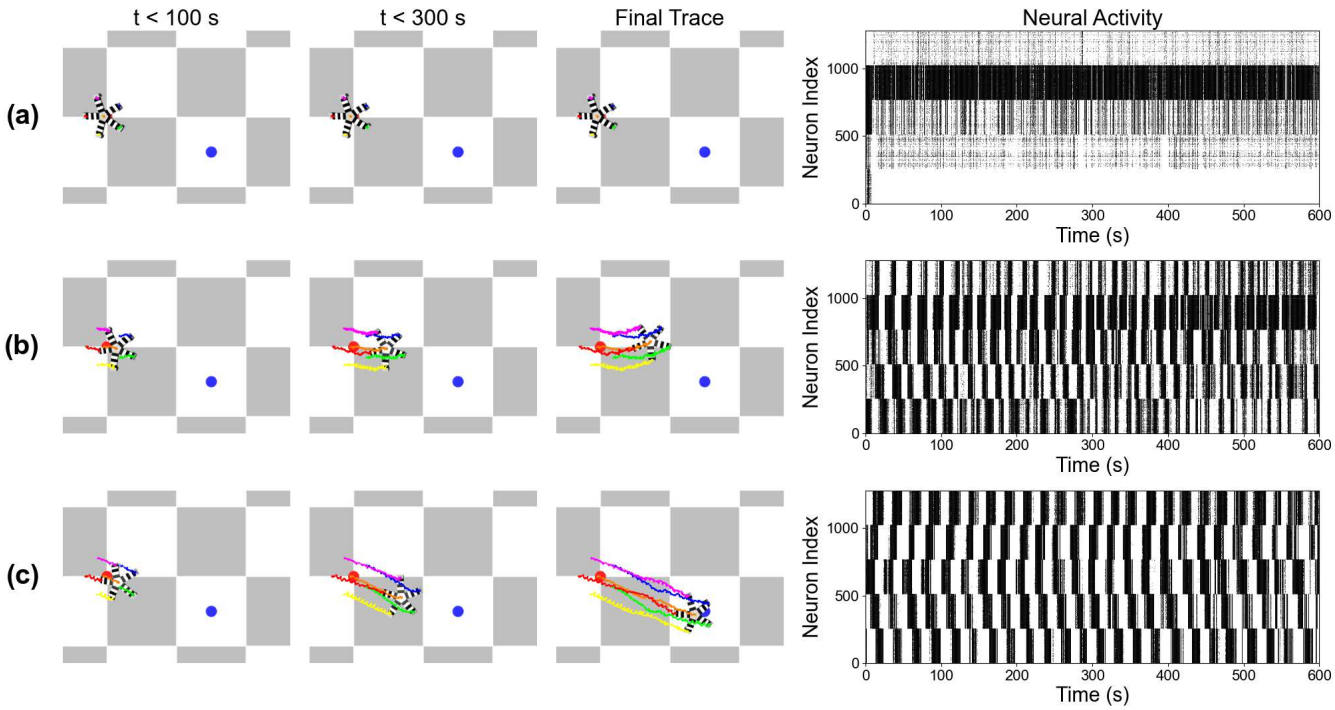


Figure 7. Locomotion trajectory of the robot and raster plot of the SAN output when the goal position relative to the robot was set to $(150, -50)$ mm. The red (blue) dot indicates the robot's initial (goal) position. Learning steps: (a) 30,000, (b) 50,000, (c) 300,000.

The frequency analysis and phase relationships under this condition are shown in Figure 8. At 30,000 steps no clear peak had formed, whereas at 50,000 steps Rays 1–4 showed a peak at 0.0550 Hz and Ray 5 at 0.0650 Hz, corresponding to periods of approximately 15–18 s. At 300,000 steps, Rays 1, 2, 4, and 5 showed a peak at 0.0417 Hz and Ray 3 at 0.0400 Hz, with a dominant contraction period of approximately 24–25 s. Regarding the phase relationships, at 300,000 steps Ray 2 and Ray 5 were located on the leading side relative to Ray 1, while Ray 3 and Ray 4 showed a phase difference close to anti-phase. This shows that, even when the direction of movement changes, the SAN does not merely reproduce a single fixed-period input but adjusts the contraction timing of each ray according to the relative relationship between the robot's posture and the goal position.

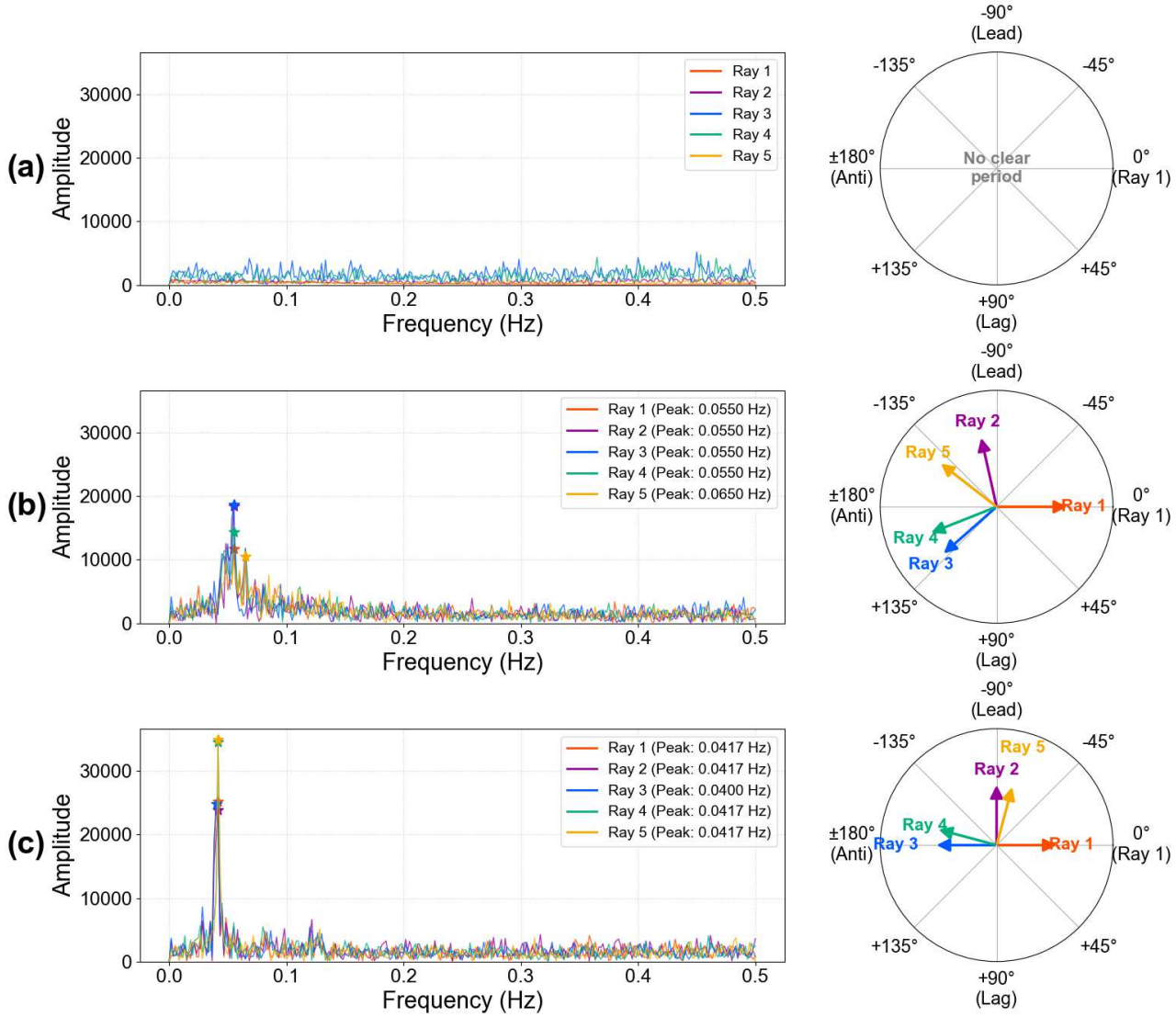


Figure 8. Frequency analysis and phase relationships of the SAN output when the goal position was set to $(150, -50)$ mm. The left column shows the FFT amplitude spectrum of the population firing activity of each ray, and the right column shows the polar plot of the phase difference relative to Ray 1. Learning steps: (a) 30,000, (b) 50,000, (c) 300,000.

Taken together, these results show that the dominant contraction period acquired in the late stage of learning was on the same time scale as the SMA relaxation time T_{ext} measured in the basic experiment, suggesting that the agent learned a drive period suited to the physical constraints of the SMA to improve locomotion efficiency. Moreover, rather than synchronizing all SMAs, the agent generated ordered phase-propagation dynamics in which each leg maintained its own phase difference while producing propulsion. These dynamics resemble the contraction–extension pattern empirically designed for the starfish-shaped robot of Mao et al.[5]. Whereas prior work pre-designed this coordinated motion and controlled it in open loop, the present results show that an SNN-based actor network can autonomously acquire a comparable gait solely through interaction with the environment.

Conclusions

We proposed an SNN-based deep reinforcement learning method for efficient autonomous locomotion control of a multi-legged soft robot actuated by shape memory alloys (SMAs), which exhibit strong nonlinearity and characteristic thermal hysteresis. Conventional SMA-driven soft robots have predominantly been controlled in open loop using manually designed periodic signals, owing to their complex response characteristics and the time delay involved in relaxation, making adaptation to dynamic tasks difficult. To control the muscle-like mechanical dynamics of the SMA, we introduced a biomimetic actor network, the

SAN, with temporal information-processing capability and trained it to learn an SMA drive strategy solely through interaction with the environment.

Experiments in the MuJoCo physics simulator confirmed that the proposed method can autonomously acquire a control policy suited to the physical characteristics of the SMA. First, the agent learned a spike-firing period of approximately 24–26 s, comparable to the SMA contraction–relaxation time of approximately 20–25 s measured in the basic experiment. This indicates that, during policy learning, the agent implicitly captured physical constraints such as thermodynamic response delay and output saturation, and acquired drive timing that exploits the actuator dynamics. Second, from the phase analysis of the spike trains driving the rays, an ordered coordinated motion was confirmed in which the SMAs are driven sequentially while maintaining their own phase differences, rather than being synchronized all at once. Through these phase-propagation dynamics, an extremely rational gait is formed that produces continuous propulsion for the robot as a whole while compensating for the long relaxation time of each individual SMA. Third, even for different goal positions, the robot acquired a policy that observes the relative relationship between its own posture and the goal and moves toward the goal by dynamically changing the contraction timing (firing pattern) of each SMA.

These results demonstrate that learning-based neuromorphic control can help overcome the limitations of conventional open-loop control for artificial-muscle-type soft robots with high-dimensional, nonlinear dynamics. A promising direction for future work is to transfer the policy learned in simulation to a physical robot (sim-to-real transfer).

References

1. Rus, D. & Tolley, M. T. Design, fabrication and control of soft robots. *Nature* **521**, 467–475 (2015).
2. Laschi, C., Mazzolai, B. & Cianchetti, M. Soft robotics: Technologies and systems pushing the boundaries of robot abilities. *Sci. robotics* **1**, eaah3690 (2016).
3. Lee, C. *et al.* Soft robot review. *Int. J. Control. Autom. Syst.* **15**, 3–15 (2017).
4. Umedachi, T. & Trimmer, B. A. Design of a 3d-printed soft robot with posture and steering control. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2874–2879 (IEEE, 2014).
5. Mao, S. *et al.* Gait study and pattern generation of a starfish-like soft robot with flexible rays actuated by smas. *J. Bionic Eng.* **11**, 400–411 (2014).
6. George Thuruthel, T., Ansari, Y., Falotico, E. & Laschi, C. Control strategies for soft robotic manipulators: A survey. *Soft robotics* **5**, 149–163 (2018).
7. Kim, D. *et al.* Review of machine learning methods in soft robotics. *Plos one* **16**, e0246102 (2021).
8. Madden, J. D. *et al.* Artificial muscle technology: physical principles and naval prospects. *IEEE J. oceanic engineering* **29**, 706–728 (2004).
9. Brinson, L. C. One-dimensional constitutive behavior of shape memory alloys: thermomechanical derivation with non-constant material functions and redefined martensite internal variable. *J. intelligent material systems structures* **4**, 229–242 (1993).
10. Teh, Y. H. & Featherstone, R. An architecture for fast and accurate control of shape memory alloy actuators. *The Int. J. Robotics Res.* **27**, 595–611 (2008).
11. Li, Y. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274* (2017).
12. Arulkumaran, K., Deisenroth, M. P., Brundage, M. & Bharath, A. A. Deep reinforcement learning: A brief survey. *IEEE signal processing magazine* **34**, 26–38 (2017).
13. Mnih, V. *et al.* Human-level control through deep reinforcement learning. *nature* **518**, 529–533 (2015).
14. Hafner, D., Pasukonis, J., Ba, J. & Lillicrap, T. Mastering diverse control tasks through world models. *Nature* **640**, 647–653 (2025).
15. Haarnoja, T. *et al.* Learning to walk via deep reinforcement learning. *arXiv preprint arXiv:1812.11103* (2018).
16. Tang, C. *et al.* Deep reinforcement learning for robotics: A survey of real-world successes. *Annu. Rev. Control. Robotics, Auton. Syst.* **8**, 153–188 (2025).
17. Haarnoja, T., Zhou, A., Abbeel, P. & Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, 1861–1870 (Pmlr, 2018).
18. Tobin, J. *et al.* Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, 23–30 (IEEE, 2017).

19. Sze, V., Chen, Y.-H., Yang, T.-J. & Emer, J. S. Efficient processing of deep neural networks: A tutorial and survey. *Proc. IEEE* **105**, 2295–2329 (2017).

20. Roy, K., Jaiswal, A. & Panda, P. Towards spike-based machine intelligence with neuromorphic computing. *Nature* **575**, 607–617 (2019).

21. Bing, Z., Meschede, C., Röhrbein, F., Huang, K. & Knoll, A. C. A survey of robotics control based on learning-inspired spiking neural networks. *Front. neurorobotics* **12**, 35 (2018).

22. Blouw, P., Choo, X., Hunsberger, E. & Eliasmith, C. Benchmarking keyword spotting efficiency on neuromorphic hardware. In *Proceedings of the 7th annual neuro-inspired computational elements workshop*, 1–8 (2019).

23. Davies, M. *et al.* Loihi: A neuromorphic manycore processor with on-chip learning. *Ieee micro* **38**, 82–99 (2018).

24. Tang, G., Kumar, N., Yoo, R. & Michmizos, K. Deep reinforcement learning with population-coded spiking neural network for continuous control. In *Conference on robot learning*, 2016–2029 (PMLR, 2021).

25. Todorov, E., Erez, T. & Tassa, Y. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, 5026–5033 (IEEE, 2012).

26. Jayender, J., Patel, R. V., Nikumb, S. & Ostojic, M. Modeling and control of shape memory alloy actuators. *IEEE transactions on control systems technology* **16**, 279–287 (2008).

27. Fujimoto, S., Hoof, H. & Meger, D. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, 1587–1596 (PMLR, 2018).

28. Maass, W. Networks of spiking neurons: the third generation of neural network models. *Neural networks* **10**, 1659–1671 (1997).

29. Neftci, E. O., Mostafa, H. & Zenke, F. Surrogate gradient learning in spiking neural networks. *IEEE Signal Process. Mag.* **36**, 51–63 (2019).

30. Gerstner, W. & Kistler, W. M. *Spiking neuron models: Single neurons, populations, plasticity* (Cambridge university press, 2002).

31. Kapturowski, S., Ostrovski, G., Quan, J., Munos, R. & Dabney, W. Recurrent experience replay in distributed reinforcement learning. In *International conference on learning representations* (2018).

32. Zenke, F. & Ganguli, S. Superspike: Supervised learning in multilayer spiking neural networks. *Neural computation* **30**, 1514–1541 (2018).

33. Kingma, D. P. & Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

Data availability

The datasets generated and analysed during the current study are available from the corresponding author on reasonable request.

Funding

This work was supported by JSPS KAKENHI Grant Number JP25K07685.

Author contributions statement

D.M. and H.T. conceived the idea and designed the research. D.M. developed the algorithm, performed the simulations, analyzed the results, and wrote the manuscript. All authors reviewed and provided input on the manuscript.

Competing interests

The authors declare no competing interests.

Additional information

Correspondence and requests for materials should be addressed to D.M.