Spiking Neural Network Discovers Energy-Efficient Hexapod Motion in Deep Reinforcement Learning

KATSUMI NAYA, KYO KUTSUZAWA, (Member, IEEE), DAI OWAKI, (Member, IEEE), AND MITSUHIRO HAYASHIBE, (Senior Member, IEEE)
Neuro-Robotics Laboratory, Department of Robotics, Graduate School of Engineering, Tohoku University, Sendai 980-8579, Japan
Corresponding author: Mitsuhiro Hayashibe (mitsuhiro.hayashibe@inria.fr)

This work was supported by the JSPS Grant-in-Aid for Scientific Research on Innovative Areas Hyper-Adaptability Project under Grant 20H05458.

ABSTRACT In Deep Reinforcement Learning (DRL) for robotics application, it is important to find energy-efficient motions. For this purpose, a standard method is to set an action penalty in the reward to find the optimal motion considering the energy expenditure. This method is widely used for the simplicity of implementation. However, since the reward is a linear sum, if the penalty is too large, the system will fall into local minima and no moving solution can be obtained. In contrast, if the penalty is too small, the effect may not be sufficient. Therefore, it is necessary to adjust the amount of the penalty so that the agent always moves dynamically, and the energy-saving effect is sufficient. Nevertheless, since adjusting the hyperparameters is computationally expensive, we need a learning method that is robust to the penalty setting problem. We investigated on the Spiking Neural Network (SNN), which has been attracting attention for its computational efficiency and neuromorphic architecture. We conducted gait experiments using a hexapod agent while varying the energy penalty settings in the simulation environment. By applying SNN to the conventional state-of-the-art DRL algorithms, we examined whether the agent could explore for an optimal gait with a larger penalty variation and obtain an energy-efficient gait verified with Cost of Transport (CoT), a metric of energy efficiency for gait. Soft Actor-Critic (SAC) + SNN resulted in a CoT of 1.64, Twin Delayed Deep Deterministic policy gradient (TD3) + SNN resulted in a CoT of 2.21, and Deep Deterministic policy gradient (DDPG) + SNN resulted in a CoT of 2.08 (1.91 for normal SAC, 2.38 for TD3, and 2.40 for DDPG). DRL combined with SNN succeeded in learning more energy efficient gait with lower CoT.

INDEX TERMS Spiking neural network, deep reinforcement learning, energy efficiency, hexapod gait, spatio-temporal backpropagation.

I. INTRODUCTION
Energy-efficient control is an important aspect in the field of robotics as the energy resource is limited for autonomous mobile robots. Several studies have been conducted to minimize the energy consumption of legged robots. One method changes the gait to match the terrain using CPG [1] and other methods transit the gait according to its own energy consumption [2], [3]. On the other hand, Deep Reinforcement Learning (DRL), which learns the optimal behavior under unknown environments by end-to-end learning, has recently attracted considerable attention in robotics for its high capability on solution space exploration.

In DRL, it adopts another approach to obtain energy-efficient behavior patterns by learning. One standard way is to add an action penalty term to the reward function by multiplying the agent’s action by a weight coefficient for considering the energy expenditure. This method can be practically applied to any DRL algorithm because it only adds a term to the reward function, and it is reported to be effective in preventing overfitting [4]. However, the weight coefficients need to be somewhat larger to achieve a sufficient effect. Because the reward is a linear sum, if the penalty is too large, the system falls into local minima, and no moving solution can be obtained. In contrast, if the penalty is too
small, the effect may be insufficient. Therefore, adjusting the hyperparameters generally requires many trials, and the computational cost and manual tuning effort become a central issue of the DRL technique. In the motion generation task using continuous control, it is necessary to search for a control input that enables dynamic movement at all times. Besides, we need a learning method that is robust to the reward setting problem without falling into the local minima where the agent stops, even if we penalize the energy expenditure.

To overcome these issues, we investigated a spiking neural network (SNN), which has attracted attention for its computational efficiency and neuromorphic architecture. An SNN is a model of neurons in the brain that transmits information by spiking signals. It can handle spatio-temporal information, and in biological systems, noise induces the generation of regularity in excitable systems such as neurons and cells [5]. An SNN has discontinuous potentials and contains noise in the system. Therefore, it is expected to yield better results [6], [7]. The spikes are binary and do not need to transmit analog values; thus, SNNs can perform efficient computation [8], [9], and recently, it has been reported that they can rapidly search for movements that adapt to the environment during walking motions [10]. The potential for higher exploration performance of SNNs can be an attractive function for the learning process. However, this exploration ability aspect of SNNs has not been well studied in robotics so far, compared to other well-known aspects.

In this study, we performed walking experiments with a larger penalty variation for a hexapod agent and with the state-of-the-art different DRL algorithms. We verify whether the combination of DRL and SNNs can explore the optimal motion and obtain energy-efficient behavior patterns, even when the energy penalty is larger than that for conventional DRL. In addition to the investigation of its reward acquisition, we also verified the cost of transport (CoT), which is a metric of energy efficiency for gait.

II. RELATED WORK

SNNs have been focused on their ability to perform efficient computations because of their ability to handle binary spikes. Thus, several studies have already been performed to apply SNNs to mobile robots [11]–[13]. However, it is known that the backpropagation method used in artificial neural networks (ANNs) cannot be applied to the training of SNNs; thus, spike-timing-dependent plasticity (STDP) was used to train SNNs in these studies. STDP performs well only in low-dimensional tasks, but it has difficulty in high-dimensional tasks. SNNs use other learning rules to solve high-dimensional tasks, and some methods have been proposed, such as converting a trained ANN into an SNN [14], which approximates the backpropagation method of an ANN (SpikeProp [15], SuperSpike [16], spatio-temporal Back-propagation [17]). In a study using these learning methods to tackle high-dimensional continuous control tasks, a method of applying SNNs to DRL algorithms was proposed [18], [19]. They applied SNNs to the actor part of each DRL algorithm and used ANNs for the critic. They also showed that they were able to deploy to Loihi, a neuromorphic processor, and performed efficient computation.

In contrast, although these studies focused on the benefit of efficient computation, another point of view is that the discontinuous potential of the model contributes to the robustness of the model [7]. Furthermore, a musculoskeletal biped simulation study was reported [10] that enabled immediate adaptation to environmental changes in its gait by using an SNN-based controller with the contribution of spike-induced ordering.

III. BACKGROUND

In this section, we describe a brief reinforcement-learning problem setup and the algorithm we used. Next, we describe the concept of spiking neural networks (SNNs), model, and PopSAN, which combines DRL and SNNs.

A. DEEP REINFORCEMENT LEARNING

In DRL problems, we consider an infinite-horizon Markov decision process (MDP), defined by $\langle S, A, p, r \rangle$, where the state space $S$ and the action space $A$ are continuous, the state transition probability $p : S \times S \times A \rightarrow [0, 1]$ represents the probability density function of the current state $s_t$ and action $a_t$ to the next state $s_{t+1}$, and $r : S \times A \rightarrow \mathcal{R}$ represents the reward given by interacting with the environment. In addition, we use trajectory $\rho_t$ that is obtained by the policy $\pi(a_t|s_t)$.

We used three different model-free DRL algorithms, SAC [20], TD3 [21], and DDPG [22], which are widely used in continuous control tasks. SAC and TD3 are now known as state-of-the-art DRL algorithms.

SAC is a stochastic DRL algorithm that learns a policy $\pi(a_t|s_t)$ that maximizes the objective function (1) considering the entropy term $H$ of the policy.

$$J(\pi) = \sum_{t=0}^{\infty} E_{(s_t, a_t) \sim \rho_t} [r(s_t, a_t) + \gamma \cdot H(\pi(\cdot | s_t))]$$

By maximizing the expected policy entropy term, the learned policy can maximize the reward obtained while maintaining the diversity of the behaviors for better exploration ability. In addition, it can be trained off-policy, resulting in high sample efficiency. More details of the theorem are provided in [20].

TD3 and DDPG are deterministic DRL algorithms that learn policy $\pi(a_t|s_t) = \mu_\theta(s)$ that outputs the presumably optimal action for the current state. TD3 improves the exploration ability and overestimation of the estimated value. The objective function is shown in Equation (2).

$$J(\pi) = \sum_{t=0}^{\infty} E_{(s_t, a_t) \sim \rho_t} [r(s_t, a_t)]$$

In TD3, an overestimation of the value estimated in the DDPG is improved by using a method called clipped double Q-learning, which extends the double Q-learning in discrete actions. In addition, to improve the exploration capability,
Gaussian noise is added to the output of the deterministic policy (target policy smoothing).

**B. SPIKING NEURAL NETWORK**

SNNs transmit information through spike trains. Besides, it captures the characteristics of real spiking neurons. Various models have been proposed [23], [24], [25]. The spike train is shown in (3).

\[ S(t) = \sum s \delta(t - t^i) \]  

(3)

where \( s \) is the label of a spike and \( \delta \) is a Dirac function.

One of the most widely used models is the leaky integrate-and-fire (LIF) model for its computational simplicity. The LIF model is described in Equation (4).

\[ \tau \frac{du(t)}{dt} = -u(t) + I(t) \]  

(4)

where \( u(t) \) is the membrane potential at time \( t \), \( \tau \) is the time constant and \( I(t) \) is input signal that is induced by a presynaptic spike train. When membrane potential \( u(t) \) exceeds a given threshold \( V_{th} \), the neuron fires and resets its potential to \( u_{reset} \).

Although SNNs have the advantage of handling spatio-temporal information, it is known that the backpropagation of an ANN cannot be applied directly to the training of multilayered SNNs. Thus, we used the STBP method [17], which has shown high performance in SNN training.

**C. PopSAN**

In this study, we applied SNN to DRL using the PopSAN method [19]. Fig. 1 shows the architecture of PopSAN.

PopSAN is an application of SNNs to the actor part of the actor-critic in reinforcement learning and consists of an encoder module, a computation module of an SNN, and a decoder module. In the encoder module, the observation is encoded as a spike using population coding. The stimulation strength in the population, \( A_E \), is expressed by Equation (5).

\[ A_E = \exp(-1/2 \cdot ((s - \mu)/\sigma)^2) \]  

(5)

After being converted to the spike format, the SNN is trained using extended STDP [18]. LIF neurons were used in the SNN module. First, the presynaptic spike \( o \) is integrated and converted to a current \( c \) (6). The current \( c \) is then integrated and converted to a membrane potential \( v \) (7). When the membrane potential exceeds a threshold, the neuron fires (8).

\[ c(t) = d_c \cdot c(t - 1) + W \cdot o(t) + b \]  

(6)

\[ v(t) = d_v \cdot v(t - 1) \cdot (1 - o(t - 1)) + c(t) \]  

(7)

\[ o(t) = \text{Threshold}(v(t)) \]  

(8)

\( d_c \) and \( d_v \) are the current and voltage decay factors.

The decoder module converts the activity of the population output of the SNN layer into the action of the agent. It calculates the firing rate by summing up the number of spikes of the neurons in each defined timestep (9). Then, the \( i \)-th action is calculated using Equation (10).

\[ fr = \frac{\sum_{t=1}^{T} o(t)}{T} \]  

(9)

\[ a = W_d \cdot fr + b_d \]  

(10)

\( W_d \) and \( b_d \) are weight and bias for each action dimension.

**IV. PROPOSED METHOD**

We conducted walking experiments with DRL and SNN-driven DRL, and compared the results with the cost of transport (CoT), a measure of energy efficiency.

In this section, we explain the details of the agents used in the walking experiment, PopSAN, which applies SNNs to DRL, and the details of CoT.

**A. SIMULATED AGENTS**

We carried out experiments using MuJoCo [26], a physics simulation engine that is widely used for reinforcement learning of continuous control tasks. We chose a legged robot that is widely used as a mobile robot. The hexapod agent is shown in Fig. 2 and was created in the dm_control [27] environment, an open-source library by DeepMind. The agent has six legs. For each leg, the shoulder has two degrees of freedom of rotation, the elbow has one degree of freedom of rotation, and the wrist has one degree of freedom of linear motion, which is a passive spring. Three actuators for drive the shoulder and elbow. The joints of the agent are set with certain stiffness, so that the posture is maintained even when no torque is applied. However when it walks, the agent needs
FIGURE 3. For each algorithm, the temporal variation regarding reward during the learning is visualized while the value of $\alpha$ is varied from 0 to 1. 1 rollout is 1000 timesteps. If the reward converges at 1000 (the dotted line), it indicates that the agent is stopped for walking.

to apply torque to move its body. These are 112 observations: the position and velocity of the hinge, the output of the torque of the actuators, the velocity of the torso, the uprightness of the torso (the inner product of the z-axis of the torso and the z-axis of the absolute coordinate), the value of the IMU sensor, and the force and torque applied to the toes. Actions have 18 dimensions: the torque input of each leg actuator.
We set the reward function $R$ as in (11) and trained the agents using each algorithm.

$$R = v + s - \alpha \sum_{i=0}^{\text{actuator}} a_i(t)^2$$  (11)

where $v$ is the speed of the torso, $s$ is the survival reward, which takes the value 1 for each time step until it falls, $a_i(t)$ is the torque input to the actuator (action) and $\alpha$ is the coefficient for the sum of the squares of action over the number of actuators. $\alpha$ acts as a penalty for the energy expenditure. As the action penalty increases, the agent is required to acquire more energy-efficient walking patterns.

### B. WALKING EXPERIMENT

We trained the agent using each of the DRL algorithms and SNN-driven DRL algorithms. We chose PopSAN [19] as the method for adapting an SNN to each DRL algorithm. The DRL algorithm that we used was based on PFRL [28], a DRL library. We also implemented PopSAN in combination with DRL of PFRL, referring to the authors’ implementation. The source code used in this study can be found at [1]. We used a DNN and an SNN with 256 neurons in two layers. The other hyperparameters are described in Table 3.

### C. COST OF TRANSPORT

We used the energy efficiency metric, CoT, to evaluate the efficiency of walking obtained through learning to verify the physical performance rather than the computational reward.

$$\text{CoT} = \frac{\sum_{i=0}^{t} \left| a_i(t) \dot{\theta}_i(t) \right| dt}{mg\Delta d}$$  (12)

where the numerator is the energy consumption of the agent, $a_i(t)$ and $\dot{\theta}_i(t)$ indicate the torque input and angle of the $i^{th}$ joint, respectively. $m$ is the mass of the agent, $g$ is gravity, and $\Delta d$ is the distance traveled by the agent. The CoT indicates the amount of energy required to move a unit distance, and the smaller the CoT, the greater the energy-efficiency of walking.

### V. EXPERIMENTAL RESULT

To obtain more energy-efficient walking, we trained the agent using the DRL algorithms SAC, TD3, DDPG, and each algorithm with SNN, by varying the reward setting regarding the value of the weighting factor for energy expenditure $\alpha$ from 0 to 1. Subsequently, to evaluate the energy efficiency of the walking obtained from the training, we measured the CoT of a trained agent.

### A. WALKING EXPERIMENT

Table 1 shows the final reward for the walking experiment, which is calculated as the total reward minus the survival reward. Fig. 3 shows the learning process. We set 1000 timesteps for a rollout, and every 10 000 timesteps, 10 evaluations without exploration are run. We trained each algorithm for 500 000 timesteps using three seeds. SD is the standard deviation of the reward over 10 evaluations. If the agent falls down in the middle of the learning process, it starts the next rollout. The highest reward for each algorithm with and without the SNNs is shown in red. This means there is one red indication between SAC and SAC+SNN, one red indication between TD3 and TD3+SNN, and one red indication between DDPG and DDPG+SNN.

### TABLE 1. The training results for each algorithm at 200k and 500k steps. The reward is the value obtained by subtracting the survival reward from the reward used in training. SD represents the standard deviation for 10 evaluations.

|         | 200k step |         | 500k step |
|---------|-----------|---------|-----------|
| Reward  | 5174, 1635| 1340, 1992| 60.76, 116.92 | 88.87, 17.82 |
| SD      | 44.12, 43.03| 45.22, 15.80 | 2.53, 0.06 |
| SAC     | 5532, 3032| 2214, 1801| 494.9, -15.8 |
| TD3     | 1348, 545.9| 527.5, 856.8| -153.6, -190.3 |
| DDPG    | 242, 472 | 85.8, 12.01 | 0.85, 0.50 |
| SAC+SNN | 4461, 2699| 1488, 967.0| 880.6, -55.1 |
| SAC+SNN | 4636, 339.09| 27.64, 61.71| 7.92, 0.92 |
| SAC+SNN | 5369, 383.1| 2144, 1787| 1465, -17.7 |
| SAC+SNN | 30.11, 37.20| 37.00, 70.46| 22.49, 0.00 |
| SAC+SNN | 1056, 1406.0| 91.0, 551.4| -83.5, -145.9 |
| SAC+SNN | 2986.0, 629.9| 23.2, 212.2| 0.79, 0.84 |
| SAC+SNN | 4634.9, 1465| 1971.0, 1276| 331.1, -141.6 |
| SAC+SNN | 89.67, 146.22| 43.39, 32.64| 4.06, 0.45 |
| SAC+SNN | 1575.7, 356.7| 674.0, 611.1| 759.0, -141.1 |
| SAC+SNN | 15.80, 90.80| 80.27, 16.63| 9.14, 0.92 |
| SAC+SNN | 1386, 1680| 1315, 758.6| 1017, -35.7 |
| SAC+SNN | 652.49, 62.13| 58.80, 52.71| 29.73, 0.21 |

CoT is defined as (12).

### TABLE 2. Cost of Transport (CoT) obtained from each algorithm. Red indicates less than 2.5, bold indicates greater than or equal to 2.5 and less than 3.0. X indicates the situation where the agent did not walk.

|         | 200k step | 250k step | 500k step |
|---------|-----------|-----------|-----------|
| SAC     | 3.123, 3.487| 2.768, 2.099| X |
| SAC+SNN| 0.059, 0.045 | 0.025, 0.015 | 4.25, 1.95, X |
| SAC+SNN| 0.042, 0.027 | 0.015, 0.003 | 0.13, 0.13 |
| TD3     | 7.522, 3.419| 2.678, 2.496| X |
| TD3+SNN| 0.047, 0.028 | 0.034, 0.021 | 2.95, 2.55 |
| DDPG    | 3.621, 3.766| 2.942, 2.378| X |
| DDPG+SNN| 0.151, 0.067 | 0.028, 0.012 | 2.59, 1.45 |
| SAC     | 12.240, 5.475| 17.395, 2.551| X |
| SAC+SNN| 1.061, 1.458 | 11.429, 0.995 | X |
| SAC+SNN| 5.556, 3.887| 2.709, 2.397| X |
| SAC+SNN| 0.896, 0.079 | 0.043, 0.028 | 0.028, 0.028 |
| SAC+SNN| 3.916, 3.752| 2.586, 2.159, 2.354 | X |
| SAC+SNN| 0.121, 0.060 | 0.019, 0.003 | 0.02, 0.02 |
| SAC+SNN| 3.483, 3.421| 2.015, 1.969, 1.640 | X |
| SAC+SNN| 0.560, 0.024 | 0.035, 0.035 | 0.017, 0.017 |
| SAC+SNN| 4.875, 3.964| 2.761, 2.258, 2.639 | X |
| SAC+SNN| 0.352, 0.017 | 0.089, 0.041 | 0.028, 0.028 |
| SAC+SNN| 4.359, 3.388| 2.951, 2.207, 2.430 | X |
| SAC+SNN| 0.127, 0.058 | 0.064, 0.034 | 0.019, 0.019 |
| SAC+SNN| 9.665, 4.439 | 3.231, 2.699, 2.757 | X |
| SAC+SNN| 0.836, 0.229 | 0.375, 0.792, 0.012 | X |
| SAC+SNN| 3.948, 3.994| 2.801, 2.259, 2.077 | X |
| SAC+SNN| 0.960, 0.084 | 0.113, 0.014 | 0.013, 0.013 |
indication between TD3 and TD3+SN and so on. Here, for each algorithm, as the value $\alpha$ increases, the total reward collected is 1000, indicate that no walking progress has been made. When the robot stops, the first term (velocity) and the third term (penalty for the action) of the reward equation become zero, and only the second term (the survival reward) is given. The maximum number of timesteps is 1000 if the robot does not fall over, so the reward for stopping the robot is 1000. The agent is learning to stop. In each DRL, the reward did not exceed 1000 at $\alpha = 0.8$, and the agent stopped without walking. In contrast, DRL with an SNN succeeded in learning walking with a reward higher than 1000 in each case.

### B. MEASURING THE COST OF TRANSPORT

Next, we measured the CoT using the agent that had trained for 200 000 timesteps and 500 000 timesteps, as shown in Table 2. We performed 1000 timestep walking experiments 30 times to calculate the CoT and its standard deviation. Red and bold letters indicate CoT less than 2.5, and within the range of 2.5–3, respectively. Table 2 reveals that, in most cases, the CoT decreased as the learning step progressed. As $\alpha$ increased, the agent was able to learn more energy-efficient walking with a lower CoT. The results show that it is effective to increase the penalty term of the action to learn an efficient movement. For SAC+SN and DDPG+SNN, walking with the smallest CoT was obtained when $\alpha = 0.8$. For each algorithm, the DRL with SNN obtained a lower CoT walking than the DRL alone.

Fig. 5-7 shows the position and velocity of the center of mass (CoM) in the z-axis (height direction) during the last learning phase for 1000 timesteps of walking for each algorithm at $\alpha = 0.6$. In the case of TD3 and DDPG, the temporal variation of the phase portrait was largely reduced by using the SNN and a consistent limit cycle based on the CoM. By using the SNN, the agent was able to learn to walk with less CoM shift, which is an energy-efficient walking with low CoT.

### C. ANALYSIS OF ENCODED OBSERVATIONS

To validate the SNN contributions to the acquisition of gait patterns, we examined how the agent’s observations were separated into spikes during encoding. For each algorithm using the SNN, we performed principal component analysis on the observations of the agent with 500 000 timestep training (a), the observations encoded into spike format using an untrained encoder (b), and the observations encoded using a trained encoder (c). The PCA results are presented in Fig. 4. Regarding (a) and (b), even when untrained encoders are used, they are transformed into a form that captures more periodic structures than the unencoded...
FIGURE 5. Phase portrait regarding CoM for SAC and SAC+SNN at $\alpha = 0.6$. This figure shows the relationship between the position of the CoM on the z-axis (height direction) and its velocity for the last learning phase (1000 timesteps).

TABLE 3. Hyperparameters for each DRL algorithms.

| Algorithm | SAC | TD3 | DDPG |
|-----------|-----|-----|------|
| Actor(Policy) | 256-ReLU-256-ReLU-U | 256-ReLU-256-ReLU-U | 256-ReLU-256-ReLU-U |
| Actor(Q function) | OcS | OcS | OcS |
| Critic(Q function) | 256-ReLU-256-ReLU-U | 256-ReLU-256-ReLU-U | 256-ReLU-256-ReLU-U |
| Learning Rate(Policy) | 0.0003 | 0.0005 | 0.0005 |
| Learning Rate(Q function) | 0.0001 | 0.0003 | 0.0005 |
| Batch size | 256/256 when $\alpha = 0.6$ | 256 | 256 |

observations, especially DDPG+SNN. Furthermore, when regarding (b) and (c), before and after learning the encoder, the distance between the observations in (c) is greater than that in (b). It is thought that the learning of the encoder has improved the representational capability of the network, and thus, the observations can better be separated. These results suggest that the learned encoder transforms its observations into a complex and periodic structure through learning, which is advantageous for learning gait.

VI. DISCUSSION

As shown in Table 2, by using the SNN in all algorithms of SAC, TD3, and DDPG, the agent was able to learn walking without falling into the local minima of stopping at $\alpha = 0.8$. In contrast, when $\alpha = 0.8$, it could not learn walking by using TD3 and DDPG. It walked by using SAC but with a high CoT. In addition, for the same value of $\alpha$, the algorithms with the SNN exhibited a lower CoT than that without the SNN. Thus, we were able to obtain energy-efficient behavior patterns with low CoTs. Among all these tests, SAC+SNN achieved the lowest CoT of 1.64. This result is significant because SAC is already known for its good exploration ability; however, it can...
be further improved by combining it with an SNN. The good exploration ability of SAC could be confirmed when it was compared to TD3 and DDPG. Although DDPG did not show good performance, it was more effective when combined with the SNN. This can be observed from the numbers of bold and red rewards and CoT in the tables, as compared to the case of non-SNN DDPG.

SAC is known to have a higher exploration ability than other deterministic DRL algorithms, and it was reported to produce more energy-efficient motions than TD3 in our
group’s previous study [29], [30]. However, this is the first report on the case where the most energy-efficient gait was obtained when SAC was combined SAC with SNN. Potentially, this indicates that the type of exploration can be different and compensatory for both the SAC and SNN explorations. In addition, Fig. 3 and Table 1 show that gait patterns in TD3 and DDPG driven by SNNs are less variable than those without SNNs, resulting in an increasing trend in reward. Because these algorithms learn by deterministic policy, they exhibit fast computations; however, compared to algorithms such as SAC, they tend to fall into local minima.

Therefore, SNN facilitated the acquisition of periodic patterns, which in turn led to the acquisition of gait patterns with less CoM shift, as shown in Figs. 6 and 7. This probably results from obtaining a more accurate periodic representation of the observations via population coding, as shown in Fig. 4. In particular, DDPG showed a sudden drop in reward in the later stages of learning; however, by using SNN, the model acquired a more stable gait, which is a visible improvement incorporated with SNN.

In DDPG and TD3, noise is added to improve the exploration performance, but the size of the noise is a hyperparameter, so it needs to be adjusted for each task to obtain a sufficient effect. However, the noise effect caused by the discontinuous potentials of the SNN does not require hyperparameter adjustment; therefore, it can be used generally for various DRL algorithms.

In contrast, the CoT is sometimes lower for normal DRLs when either the learning step or $\alpha$ is small. In PopSAN, both the encoders and decoders are learnable. However, when the learning steps are small, they are not learned sufficiently, which may result in high CoT walking. The reason why the CoT is sometimes lower in normal DRL when $\alpha$ is small is that the influence of the energy efficiency term in the reward is small, and even SNN-driven DRL sometimes results in a gait with unnecessarily large leg movements.

VII. CONCLUSION

In this study, to obtain energy-efficient gait patterns, we trained a six-legged agent to walk with different reward settings by varying the weight to consider for the energy expenditure. The effect of SNN-driven DRL was investigated over different DRL algorithms and evaluated for the energy efficiency of the hexapod gait using CoT.

In both the stochastic algorithm such as SAC and the deterministic algorithm such as TD3 and DDPG, we succeeded in searching the walking pattern even with a larger energy penalty setting, when it was combined with SNN. SNN-driven DRL obtained a more energy-efficient gait when the learning step was considerable, and a larger energy penalty was set. To the best of our knowledge, this is the first report that the most energy-efficient motion could be obtained when SAC is driven with an SNN than with SAC only. In addition, we confirmed the increase in the reward in TD3 and DDPG.

Until now, SNNs have been mainly focused on their computational efficiency merits, but we have experimentally demonstrated that they also be beneficial in terms of acquiring periodic patterns for energy-efficient motor learning in legged agents in DRL.

APPENDIX

HYPERPARAMETERS FOR EACH DRL ALGORITHMS

The hyperparameters used for each DRL algorithm are listed in Table 3. Some parameters that significantly affected the learning were experimentally determined by the validation experiments shown in Figs. 8 and 9. The other parameters were determined by referring to PFRL [28] and PopSAN [19].

Fig. 8 shows that none of the DRL algorithms succeed in learning. On the contrary, SNN-driven DRL succeeds in learning, except when the learning rate is 7e-4 in SAC+SNN and DDPG+SNN. Fig. 9 shows that all batch sizes, except when the SAC’s batch size is 128, fail to learn, whereas the SNN-driven DRL succeeds in learning for all batch sizes.

REFERENCES

[1] S. Zenker, E. E. Aksoy, D. Goldschmidt, F. Worgotter, and P. Manoonpong, “Visual terrain classification for selecting energy efficient gaits of a hexapod robot,” in Proc. IEEE/ASME Int. Conf. Adv. Intell. Mechatronics, Jul. 2013, pp. 577–584.
[2] G. Wang, L. Ding, H. Gao, Z. Deng, Z. Liu, and H. Yu, “Minimizing the energy consumption for a hexapod robot based on optimal force distribution,” IEEE Access, vol. 8, pp. 5393–5406, 2020.
[3] N. Kotegge, C. Parkinson, P. Moghadam, A. Elfes, and S. P. N. Singh, “Energetics-informed hexapod gait transitions across terrains,” in Proc. IEEE Int. Conf. Robot. Autom. (ICRA), May 2015, pp. 5140–5147.
[4] J.-C. Shi, Y. Yu, Q. Du, S.-Y. Chen, and A.-X. Zeng, “Virtual-taobao: Virtualizing real-world online retail environment for reinforcement learning,” in Proc. AAAI Conf. Artif. Intell., Jul. 2019, vol. 33, no. 1, pp. 4902–4909. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/4419
[5] B. Lindner, J. Garcia-Ojalvo, A. Neiman, and L. Schimansky-Geier, “Effects of noise in excitable systems,” Phys. Rep., vol. 392, no. 6, pp. 321–424, 2004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0370157303004228.
[6] N. Kasabov and E. Capecci, “Spiking neural network methodology for modelling, classification and understanding of EEG spatio-temporal data measuring cognitive processes,” Inf. Sci., vol. 294, pp. 565–575, Feb. 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025514006562.
[7] J. H. Lee, T. Delbruck, and M. Pfeiffer, “Training deep spiking neural networks using backpropagation,” Frontiers Neurosci., vol. 10, p. 508, Nov. 2016. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnins.2016.00508.
[8] E. Z. Farsa, A. Ahmadi, M. A. Maleki, M. Gholami, and H. N. Rad, “A low-cost high-speed neuromorphic hardware based on spiking neural network,” IEEE Trans. Circuits Syst. II, Exp. Briefs, vol. 66, no. 9, pp. 1582–1586, Sep. 2019.
[9] B. V. Benjamin, P. Gao, and E. McQuinn, “Neuroid: A mixed-analog-digital multichip system for large-scale neural simulations,” Proc. IEEE, vol. 102, no. 5, pp. 699–716, May 2014.
[10] S. Yonekura and Y. Kuniyoshi, “Spike-included ordering: Stochastic neural spikes provide immediate adaptability to the sensormotor system,” Proc. Nat. Acad. Sci. USA, vol. 117, no. 22, pp. 12486–12496, Jun. 2020. [Online]. Available: https://www.pnas.org/content/117/22/12486.
[11] Z. Bing, C. Meschede, G. Chen, A. Knoll, and K. Huang, “Indirect and direct training of spiking neural networks for end-to-end control of a lane-keeping vehicle,” Neural Netw., vol. 121, pp. 21–36, Jan. 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608019301595.
[12] G. Tang, A. Shah, and K. P. Michmizos, “Spiking neural network on neuromorphic hardware for energy-efficient unidimensional slam,” in Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS), Nov. 2019, pp. 4176–4181.
