Reinforcement co-Learning of Deep and Spiking Neural Networks for Energy-Efficient Mapless Navigation with Neuromorphic Hardware

Guangzhi Tang, Neelesh Kumar, and Konstantinos P. Michmizos

***Abstract*— Energy-efficient mapless navigation is crucial for mobile robots as they explore unknown environments with limited on-board resources. Although the recent deep reinforcement learning (DRL) approaches have been successfully applied to navigation, their high energy consumption limits their use in many robotic applications. Here, we propose a neuromorphic approach that combines the energy-efficiency of spiking neural networks with the optimality of DRL to learn control policies for mapless navigation. Our hybrid framework, Spiking deep deterministic policy gradient (SDDPG), consists of a spiking actor network (SAN) and a deep critic network, where the two networks were trained jointly using gradient descent. The trained SAN was deployed on Intel's Loihi neuromorphic processor. The co-learning enabled synergistic information exchange between the two networks, allowing them to overcome each other's limitations through a shared representation learning. When validated on both simulated and real-world complex environments, our method on Loihi not only consumed 75 times less energy per inference as compared to DDPG on Jetson TX2, but also had a higher rate of successfully navigating to the goal which ranged by 1% to 4.2%, depending on the forward-propagation timestep size. These results reinforce our ongoing effort to design brain-inspired algorithms for controlling autonomous robots with neuromorphic hardware.**

I. Introduction

The ability of a mobile robot to navigate autonomously becomes increasingly important with the complexity of the unknown environment that it explores. Traditionally, navigation has been relying on global knowledge in the form of maps of the environment [1]. Yet, for many applications in need of effective navigation, the construction of an informative map is prohibitively expensive, due to real-time requirements and the limited energy resources [2], [3]. The recent introduction of deep reinforcement learning (DRL) methods, such as deep deterministic policy gradient (DDPG) [4], enabled learning of optimal control policies for mapless navigation, where the agent navigates using its local sensory inputs and limited global knowledge [5], [6], [7]. The optimality of DRL, however, also comes at a high-energy cost. Given that the growing complexity of mobile robot applications is hard to be continuously offset by equivalent increases in on-board energy sources, there is an unmet need for low-power solutions to robotic mapless navigation.

Energy-efficiency is currently the main advantage demonstrated by spiking neural networks (SNN), an emerging brain-inspired alternative architecture to deep neural networks (DNN) in which neurons compute asynchronously and communicate through discrete events called spikes[8]. We and others have recently shown how the realization of SNNs in a neuromorphic processor results in low-power solutions for mobile robots, ranging from localization and mapping of mobile robots [9] on Intel's Loihi [10] to planning [11] and control [12]. For mapless navigation, most SNN-based approaches employ a reward modulated learning, where a global reward signal drives the local synaptic weight updates [13], [14]. Despite the biological plausibility of this learning rule, it suffers from catastrophic forgetting and a lack of policy evaluation [13], which limit the learning of policies in complex real-world environments.

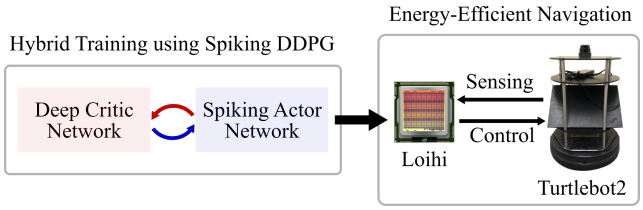

Fig. 1. Overview of the proposed hybrid framework, Spiking DDPG, that consisted of a spiking actor network and a deep critic network that were co-trained. The trained SAN was deployed on Intel's Loihi neuromorphic chip for controlling a mobile robot in simulated and real-world complex environments. The resulted mapless navigation compared favorably with the state-of-the-art methods, in both energy-efficiency and accuracy.

Interestingly, DRL methods are well-versed in overcoming catastrophic forgetting through memory replay, and provide systematic evaluation of policies [4]. That led us to wonder if, and to what extent, could we combine the advantages of two emerging methodologies, namely the energy-efficiency of an SNN with the computational capabilities of a DNN. Recent efforts to combine these two architectures have relied on directly converting the trained DNNs to SNNs using techniques such as weight-scaling [15]. Such methods, though, require larger time-steps for inference [16], which becomes particularly problematic in mobile robots that need to make decisions in real-time. To overcome this limitation, one possibility is to directly train SNNs using gradient-descent techniques such as spatiotemporal backpropagation (STBP) [17]. This method has demonstrated faster inference while exhibiting state-of-the-art performance for a wide range of classification tasks [18], [17]. However, the limited ability of spiking neurons to represent high precision action-values would result in the prediction of the same action-values for different inputs and prevent the generation of

*This work is supported by Intel's NRC Grant Award

GT, NK and KM are with the Computational Brain Lab, Department of Computer Science, Rutgers University, New Jersey, USA. konstantinos.michmizos@cs.rutgers.edu

corrective errors during training. Given the ability of the DRL approaches to represent high precision action-values, a fascinating possibility for developing a neuromorphic solution to mapless navigation in complex environments is to train the SNN in conjunction with a deep network.

In this paper, we propose Spiking DDPG (SDDPG), an energy-efficient neuromorphic method that uses a hybrid SNN/DNN framework to learn optimal policies for mapless robotic navigation in real-world environments (Fig. 1). Like its deep network counterpart, SDDPG has separate networks to represent policy and action-value: a spiking actor network (SAN) to infer actions from the robot states and a deep critic network to evaluate the actor. The two networks in this architecture were trained jointly using gradient-descent. To train the SAN, we introduce an extension of STBP, which allowed us to faithfully deploy the trained SAN on Intel's Loihi. We benchmarked our method through comparative analyses for performance and energy-efficiency with respect to DDPG in simulated and real-world complex environments. The SDDPG on Loihi consumed 75 times less energy per inference when compared against DDPG on Jetson TX2, while also achieving a slightly higher rate of successfully navigating to the goal.

II. Methods

A. Spiking Deep Deterministic Policy Gradient (SDDPG)

We propose the SDDPG algorithm to learn the optimal control policies for mapping a given state of the robot $s = \{G_{dis},\ G_{dir},\ \nu,\ \omega,\ S\}$ to the robot action $a = \{\nu_L,\ \nu_R\}$, where G_{dis} and G_{dir} are the relative distance and direction from the robot to the goal; ν and ω are the linear and angular velocities of the robot; S is the distance observations from the laser range scanner; ν_L and ν_R are the left and right wheel speeds of the differential drive mobile robot.

The hybrid framework consisted of a spiking actor network (SAN) and a deep critic network (Fig. 2). During training, the SAN generated an action a for a given state s, which was then fed to the critic network for predicting the associated action-value $Q(s, a)$. The SAN was trained to predict the action for maximizing this Q value. The critic network, in turn, was trained to minimize the temporal difference (TD) error for action-value, as described in [4]. To update the action-value, we used a reward function adopted from [5]:

$$R = \begin{cases} R_{goal} & \text{if } G_{dis} < G_{th} \\ R_{obstacle} & \text{if } O_{dis} < O_{th} \\ A * (G_{dis}(t) - G_{dis}(t-1)) & \text{otherwise} \end{cases} \quad (1)$$

where R_{goal} and $R_{obstacle}$ are the positive and negative rewards, respectively; O_{dis} is the distance to the obstacle; A is an amplification factor; $G_{th},\ O_{th}$ are the thresholds. The reward function encourages the robot to move towards the goal during exploration, which facilitates training.

For inference, we deployed the trained SAN on Loihi (see *II.D*), to predict the action to navigate the robot to the goal. We give the mathematical formalism for the inference and training phases in the next two sections.

B. Spiking Actor Network (SAN)

The building block of the SAN was the leaky-integrate-and-fire (LIF) model of a spiking neuron. Specifically, we updated the states of the i^{th} neuron at timestep t in two stages. First, we integrated the input spikes into synaptic current as follows,

$$c_i(t) = d_c \cdot c_i(t-1) + \Sigma_j w_{ij} o_j(t) \quad (2)$$

where c is the synaptic current, d_c is the decay factor for the current, w_{ij} is the weight of the connection from the j^{th} presynaptic neuron and o_j is a binary variable (0 or 1) which indicates the spike event of the j^{th} presynaptic neuron.

Second, we integrated the synaptic current into the membrane voltage of the neuron as per equation (3). Subsequently, the neuron fired a spike if the membrane voltage exceeded the threshold.

$$\begin{gathered} v_i(t) = d_v \cdot v_i(t-1) + c_i(t), \quad \text{if } v_i(t-1) < V_{th} \\ o_i(t) = 1 \ \&\ v_i(t) = 0, \quad \text{otherwise} \end{gathered} \quad (3)$$

where v is the membrane voltage, d_v is the decay factor for the voltage and V_{th} is the firing threshold.

The LIF neurons formed a fully connected multilayered SAN (Fig. 2). The network was driven by discrete Poisson spikes that encoded the continuous state variables. This was done by generating a spike at each timestep with probability proportional to the value of the state variables at that time. After T timesteps, we decoded the rescaled average spike count of the output layer neurons (**Action**) to left and right wheel speeds of the robot (Algorithm 1).

Algorithm 1: Forward propagation through SAN

Output: Left and right wheel speeds ν_L, ν_R
Require: Maximum timestep, T; Network depth, l
Require: Min and max wheel speeds ν_{min}, ν_{max}
Require: $\mathbf{W}^{(i)}$, $i \in \{1, ..., l\}$, the weight matrices
Require: $\mathbf{b}^{(i)}$, $i \in \{1, ..., l\}$, the bias parameters
Require: $\mathbf{X}^{(i)}$, $i \in \{1, ..., T\}$, the input spike trains
for *t=1,...,T* **do**
  $\mathbf{o}^{(t)(0)} = \mathbf{X}^{(t)}$;
  for *k=1,...,l* **do**
    $\mathbf{c}^{(t)(k)} = d_c \cdot \mathbf{c}^{(t-1)(k)} + \mathbf{W}^{(k)} \mathbf{o}^{(t)(k-1)} + \mathbf{b}^{(k)}$;
    $\mathbf{v}^{(t)(k)} = d_v \cdot \mathbf{v}^{(t-1)(k)} \cdot (1 - \mathbf{o}^{(t-1)(k)}) + \mathbf{c}^{(t)(k)}$;
    $\mathbf{o}^{(t)(k)} = Threshold(\mathbf{v}^{(t)(k)})$;
  end
  $\mathbf{SpikeCount}^{(t)} = \mathbf{SpikeCount}^{(t-1)} + \mathbf{o}^{(t)(l)}$;
end
$\mathbf{Action} = \mathbf{SpikeCount}^{(T)} / \ T$
$\nu_L = \mathbf{Action}[0] * (\nu_{max} - \nu_{min}) + \nu_{min}$
$\nu_R = \mathbf{Action}[1] * (\nu_{max} - \nu_{min}) + \nu_{min}$

C. Direct Training of SAN with Back-propagation

We extended the STBP to directly train our SAN for learning the optimal policy. The original STBP is limited to training networks containing simplified LIF neurons that

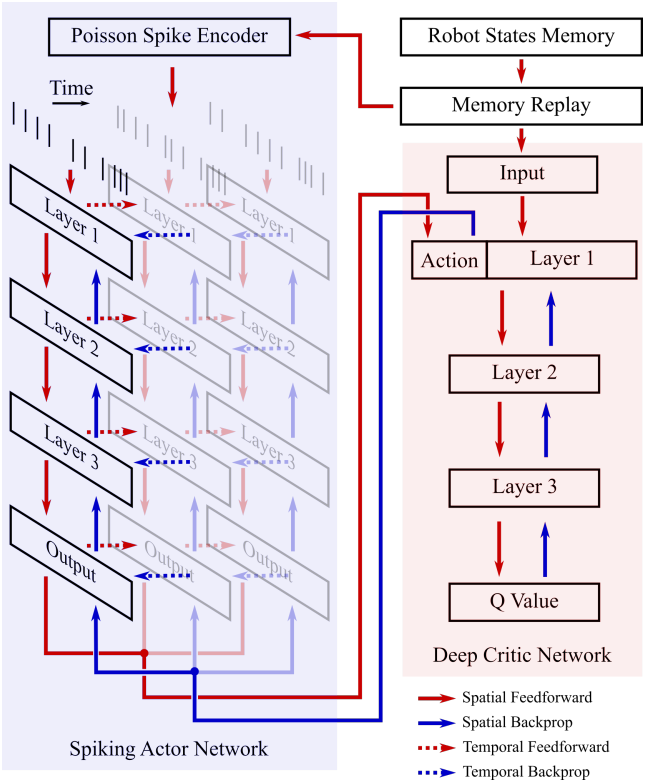

Fig. 2. Hybrid training of multilayered SAN and Deep Critic Network.

only have one state variable (voltage). Here, we extended it to a broader class of LIF neurons with two internal state variables (current and voltage), defined in the equations (2) and (3). We did this to be able to deploy our trained model on Loihi, which implements such a two-state neuron model.

Since the threshold function that defines a spike is non-differentiable, the STBP algorithm requires a pseudo-gradient function to approximate the gradient of a spike. We chose the derivative of the rectangular function (defined in equation (4)) as our pseudo-gradient function since it demonstrated the best empirical performance in [17].

$$z(v) = \begin{cases} a_1 & \text{if } |v - V_{th}| < a_2 \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

where z is the pseudo-gradient, a_1 is the amplifier of the gradient, a_2 is the threshold window for passing the gradient.

At the end of the forward propagation of the SAN, the computed **Action** was fed to the n^{th} layer of the critic network, which in turn generated the predicted Q value. The SAN was trained to predict the action for which the trained critic network generated the maximum Q value. To do so, we trained our SAN to minimize the loss L defined in equation (5), using gradient descent.

$$L = -Q. \tag{5}$$

The gradient on the n^{th} layer of the critic network was:

$$\nabla_{\textbf{Action}} L = \mathbf{W}_c^{(n+1)'} \cdot \nabla_{a^{(n+1)}} L, \tag{6}$$

where a^{n+1} is the output of the $(n+1)^{th}$ layer of the critic network before being passed through the non-linear activation function, such as $ReLU$, and $\mathbf{W}_c^{(n+1)}$ are the critic network weights at the $(n+1)^{th}$ layer.

This gradient was then backpropagated to the SAN. To describe the complete gradient descent, we separate our analysis into two cases: i) the last forward propagation timestep, $t = T$, and ii) all the previous timesteps, $t < T$.

Case 1: for $t = T$.

At the output layer l, we have:

$$\begin{aligned} \nabla_{\textbf{SpikeCount}^{(t)}} L &= \frac{1}{T} \cdot \nabla_{\textbf{Action}} L \\ \nabla_{\mathbf{o}^{(t)(k)}} L &= \nabla_{\textbf{SpikeCount}^{(t)}} L \end{aligned} \tag{7}$$

Then for each layer, $k = l$ down to 1:

$$\begin{aligned} \nabla_{\mathbf{v}^{(t)(k)}} L &= z(\mathbf{v}^{(t)(k)}) \cdot \nabla_{\mathbf{o}^{(t)(k)}} L \\ \nabla_{\mathbf{c}^{(t)(k)}} L &= \nabla_{\mathbf{v}^{(t)(k)}} L \\ \nabla_{\mathbf{o}^{(t)(k-1)}} L &= \mathbf{W}^{(k)'} \cdot \nabla_{\mathbf{c}^{(t)(k)}} L \end{aligned} \tag{8}$$

Case 2: for $t < T$.

At the output layer l, we have:

$$\begin{aligned} \nabla_{\textbf{SpikeCount}^{(t)}} L = \nabla_{\textbf{SpikeCount}^{(t+1)}} L &= \frac{1}{T} \cdot \nabla_{\textbf{Action}} L \\ \nabla_{\mathbf{o}^{(t)(k)}} L &= \nabla_{\textbf{SpikeCount}^{(t)}} L \end{aligned} \tag{9}$$

Then for each layer, $k = l$ down to 1:

$$\begin{aligned} \nabla_{\mathbf{v}^{(t)(k)}} L &= z(\mathbf{v}^{(t)(k)}) \cdot \nabla_{\mathbf{o}^{(t)(k)}} L + \\ & \quad d_v(1 - \mathbf{o}^{(t)(k)}) \cdot \nabla_{\mathbf{v}^{(t+1)(k)}} L \\ \nabla_{\mathbf{c}^{(t)(k)}} L &= \nabla_{\mathbf{v}^{(t+1)(k)}} L + d_c \nabla_{\mathbf{c}^{(t+1)(k)}} L \\ \nabla_{\mathbf{o}^{(t)(k-1)}} L &= \mathbf{W}^{(k)'} \cdot \nabla_{\mathbf{c}^{(t)(k)}} L \end{aligned} \tag{10}$$

In this case, the gradients with respect to voltage and current had additional terms as compared to case 1, reflecting the temporal gradients backpropagated from the future timesteps.

By collecting the gradients backpropagated from all the timesteps (computed in the above two cases), we can compute the gradient of the loss with respect to the network parameters, $\nabla_{\mathbf{W}^{(k)}} L$, $\nabla_{\mathbf{b}^{(k)}} L$ for each layer k, as below:

$$\begin{aligned} \nabla_{\mathbf{W}^{(k)}} L &= \sum_{t=1}^{T} \mathbf{o}^{(t)(k-1)} \cdot \nabla_{\mathbf{c}^{(t)(k)}} L \\ \nabla_{\mathbf{b}^{(k)}} L &= \sum_{t=1}^{T} \nabla_{\mathbf{c}^{(t)(k)}} L \end{aligned} \tag{11}$$

We updated the network parameters every T timesteps.

D. SAN Realization on Loihi Neuromorphic Processor

We realized our trained SAN on Intel's Loihi. Given that Loihi only supports 8 bits integer weights, we introduce a layer-wise rescaling technique for mapping the trained SNNs with higher weight precisions onto the chip. Specifically, we rescaled the weights and the voltage threshold of each layer using equation (12), while maintaining their spike outputs by fixing the weight-threshold ratio. As a benefit of training our network with the neuron model that Loihi supports, all

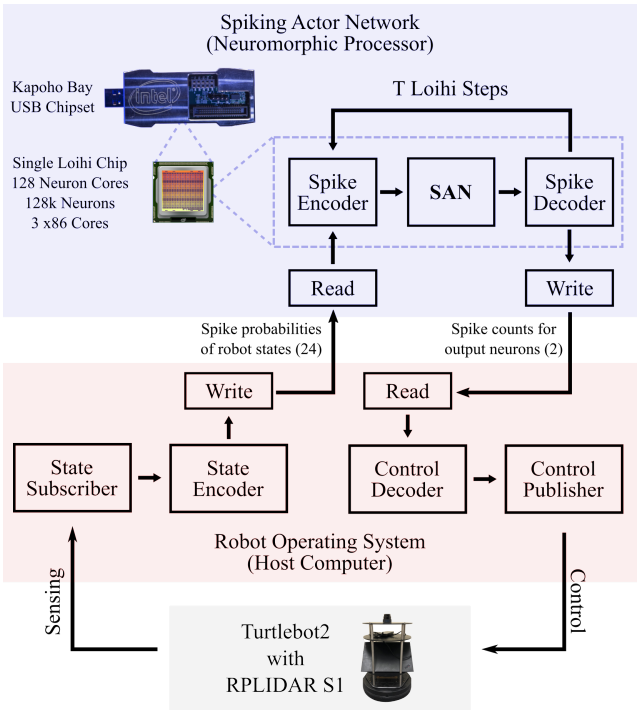

Fig. 3. Interaction between Loihi and the mobile robot through the robot operating system (ROS), for a single SNN inference.

other hyperparameters remained the same as the ones used in training.

$$
\begin{aligned}
r^{(k)} &= \frac{W_{max}^{(k)(loihi)}}{W_{max}^{(k)}} \\
\mathbf{W}^{(k)(loihi)} &= round(r^{(k)} \cdot \mathbf{W}^{(k)}) \\
V_{th}^{(k)(loihi)} &= round(r^{(k)} \cdot V_{th})
\end{aligned}
\tag{12}
$$

where $r^{(k)}$ is the rescale ratio of layer k, $W_{max}^{(k)(loihi)}$ is the maximum weight that Loihi supports, $W_{max}^{(k)}$ is the maximum weight of layer k of the trained network, $\mathbf{W}^{(k)(loihi)}$ are the rescaled weights on Loihi, and $V_{th}^{(k)(loihi)}$ is the rescaled voltage threshold on Loihi.

We also introduce an interaction framework for Loihi to control the mobile robot in real-time through the robot operating system (ROS) (Fig. 3). We encoded the robotic states obtained from ROS into Poisson spikes, and decoded the output spikes for robot control. The encoding and decoding modules were deployed on the low-frequency x-86 cores that Loihi has for interfacing with the on-chip networks during runtime. This avoided the need for communicating between Loihi and ROS directly through spikes, which reduced the data transfer load between Loihi and ROS.

III. Experiments and Results

A. Experimental Setup

We trained and validated our method on Turtlebot2 platform equipped with an RPLIDAR S1 laser range scanner (range: 0.2-40 m). The robot's field of view was set to front-facing 180 degrees with 18 range measurements, each with 10 degrees of resolution. Training was performed in the Gazebo simulator and validation was done in both the simulation and real-world environments. We used the ROS as a middleware for both the training and validation. The neuromorphic realization was performed on Intel's Kapoho-Bay USB chipset, with two Loihi chips.

B. Training in Simulator

During training, the agent sequentially navigated 4 environments of increasing complexities (Fig. 4a). The start and goal locations were sampled randomly from specific places in the 4 environments. The increase in difficulty across the 4 environments was due to the added obstacles and the different start-goal pairs. This encouraged the robot to build upon previously learned simple policies in easier environments (Env 1 and 2) and gradually learn complex policies for navigating in difficult environments (Env 3 and 4). This form of curriculum training has been shown to result in better generalization and faster convergence [19], [20].

Each episode in training could result in 3 possible outcomes: i) success: robot successfully navigated to the goal; ii) collision: robot collided with an obstacle; iii) timeout:

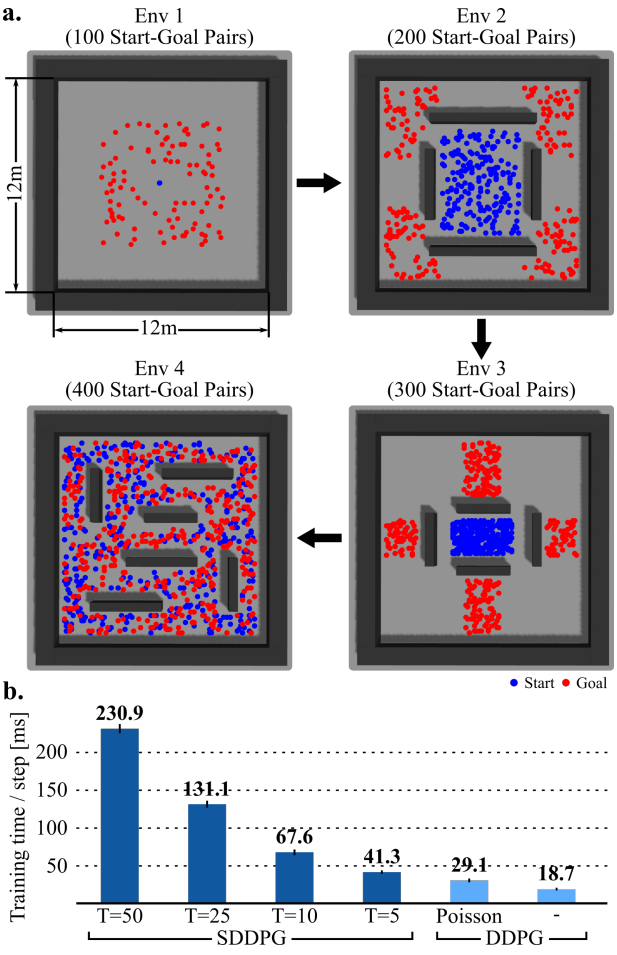

Fig. 4. **a.** The four training environments of increasing complexity, with randomly generated start and goal locations enabled curriculum learning. **b.** Training time per execution step of SDDPG decreased with decreasing T and approached the training time for DDPG.

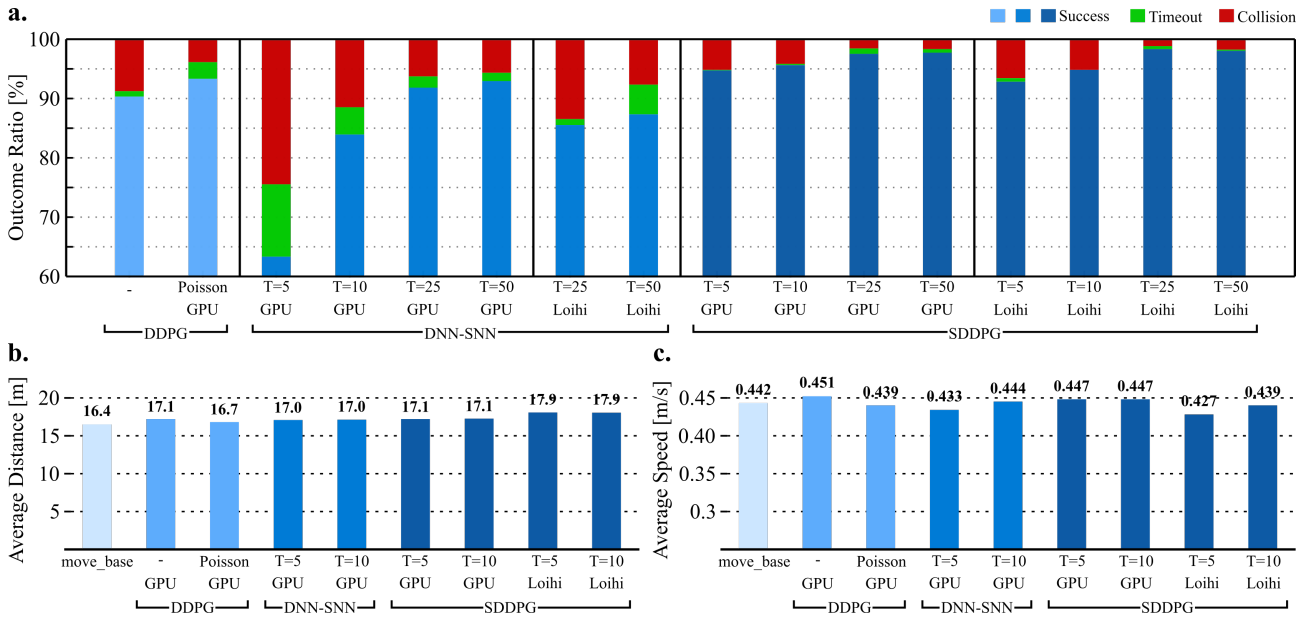

Fig. 5. Comparing SDDPG with other navigation methods in a complex test environment in the simulator, with 200 randomly generated start and goal positions. **a.** SDDPG has a higher rate of successfully navigating to the goal than the DNN-SNN and DDPG methods. There was no decrease in its performance when deployed on Loihi. **b**, **c.** SDDPG results in similar route quality (average distance and speed) as the other navigation methods. Results are averaged over five trained models for each method.

TABLE I
HYPERPARAMETERS FOR TRAINING SDDPG

Parameters	Values
Neuron parameters (V_{th}, d_c, d_v)	0.5, 0.5, 0.75
Pseudo-gradient function parameters (a_1, a_2)	1.0, 0.5
Neurons per hidden layer for SAN and critic net	256, 512
Batchsize	256
Learning rate for training SAN and critic net	10^{-5}, 10^{-4}
Goal and collision reward ($R_{goal}, R_{obstacle}$)	30, -20
Reward amplification factor (A)	15
Reward thresholds (G_{th}, O_{th})	0.5, 0.35 m
Wheel speed (ν_{min}, ν_{max})	0.05, 0.5 m/s

navigation exceeded 1000 execution steps, with each execution step being 0.1s. Training was performed for a total of 200,000 execution steps across all 4 environments. To average out the effect of exploration during training and for fair comparison with the baselines, we trained 5 models corresponding to 5 sets of randomly initialized start and goal pairs. Moreover, to investigate the effect of the forward propagation time, T, we trained SDDPG corresponding to 4 different values of $T = 5, 10, 25, 50$. Training time decreased with decreasing T (Fig. 4b), which partially overcame the limitation of high training time commonly associated with SNNs. The remaining hyperparameters used for training are shown in Table I.

C. Baselines for comparison

We compared SDDPG with the following approaches:

1) Map-based Navigation: We used the widely used ROS navigation package move_base consisting of a DWA (dynamic window approach) [21] local planner and a global planner based on Djikstra's algorithm. The map required for move_base was constructed using GMapping [22]. The robot's maximum speed was set to 0.5 m/s, same as SDDPG.

2) DDPG: The DDPG had the same network architecture as our SDDPG, with the SAN replaced by a deep actor network. To investigate the role of randomly generated Poisson spike inputs, we also compared our SDDPG against DDPG receiving inputs injected with Poisson noise (DDPG Poisson). To do this, we encoded the state inputs to Poisson spikes and then decoded it back to continuous state inputs. The baseline (DDPG/DDPG Poisson) and SDDPG methods were trained using the same hyperparameters. Training times for DDPG and SDDPG methods are shown in Fig. 4b.

3) DNN to SNN Conversion (DNN-SNN): We converted a deep actor network trained under the DDPG framework with Poisson noise, to an SNN, with same T values as the SDDPG, using weight rescaling [15]. We determined the optimum rescale factor by computing the layer's maximum output over the training duration, and then performing grid search around it [16].

D. Evaluation in Simulator

We evaluated our method in the Gazebo simulator in a $20m \times 20m$ test environment (Fig. 6). To test the generalization capability of our method, we designed the test environment to be sufficiently different from the training environments in the following aspects: i) differently shaped obstacles (triangular, L-shaped); ii) narrower traversal passages (min. 0.75m for test, 1.75m for training); iii) more densely organized obstacles. For an exhaustive evaluation, we generated 200 start and goal locations, sampled uniformly at random from all parts of the test environment with a minimum distance of 6m.

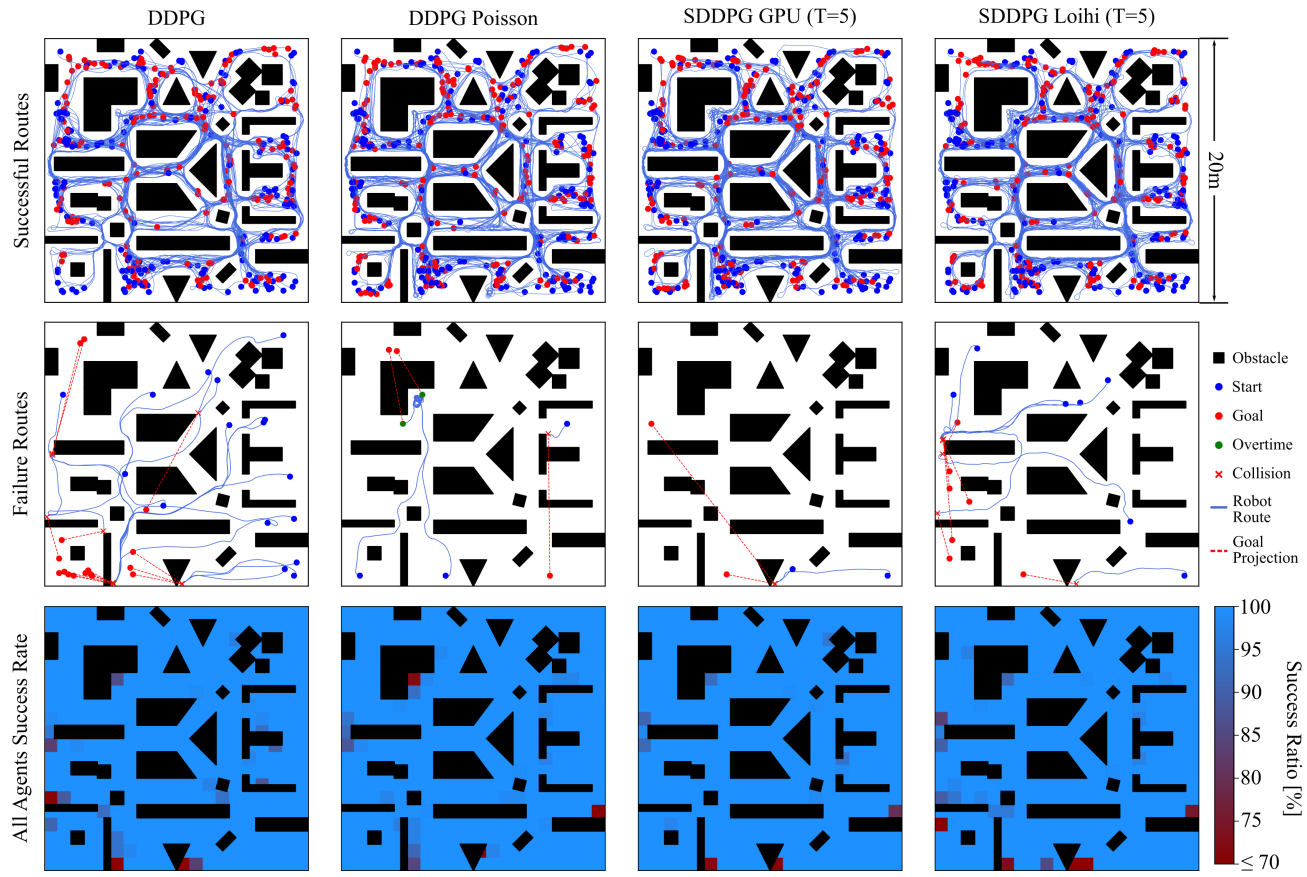

Fig. 6. Analyzing success and failure trajectories over 200 randomly sampled start and goal locations for the models with the highest success rate per method (upper and middle row). SDDPG fails at fewer locations than DDPG. The superior SDDPG performance is demonstrated in the heatmap showing the rate of successfully crossing each $1m \times 1m$ location in the test environment, computed across 5 models per method (bottom row).

We used the same start and goal locations to evaluate our method and all the baselines. We first compared the methods based on the rate of the three possible outcomes–success, collision, and timeout (Fig. 5a). Our method outperformed the DNN-SNN conversion method for all values of T, with the performance being substantially better for smaller values of T. Our method performed slightly better than DDPG, even when deployed on Loihi with low precision weights. To further inspect SDDPG's ability to navigate effectively, we compared its route quality against move_base, as well as all the other navigation methods(Fig. 5b,c). Specifically, we computed the average distance and speed corresponding to the successful routes taken by each method. For fairness, we only considered the successful routes that had common start and goal positions across all methods. Despite not having access to the map, SDDPG achieved the same level of performance as the map-based method, move_base.

We then analyzed the trajectories of the routes that resulted in failure (collision or timeout) (Fig. 6). The methods failed at the locations where it required the agents to move around the obstacles in the ways that it had never experienced in training. To further investigate the failure locations, we generated a heatmap of the environment where the intensity of pixel corresponding to each $1m \times 1m$ location was equal to the percentage of times the agent successfully crossed that location. The heatmap reveals that the SDDPG method failed at fewer locations than DDPG.

Strikingly, although we did not explicitly target performance improvement over state-of-the-art, our results indicates that SDDPG had a slightly higher rate of successfully navigating to the goal. A possible explanation is that the noise introduced by Poisson spike encoding of the state inputs helps the SDDPG networks in escaping the 'bad' local minima, in alignment with the results of [23]. The fact that DDPG Poisson performed better than DDPG further supports this reasoning. A theoretical analysis of this observation is intriguing but beyond the scope of this study.

E. Evaluation in Real-world

We evaluated the navigation methods in a real-world environment to test the generalization capability of SDDPG (Fig. 7a). The environment was an office setting consisting of cubicles and commonplace items such as chairs, desks, sofas, dustbins, and bookshelves. The space spanned over an area of approximately $215m^2$, with the shortest passage being $0.9m$ in length. The robot was required to navigate to 15 goal locations placed sequentially to cover all the areas of the environment. We estimated the pose of the robot using $amcl$ [24], based on the map generated by GMapping. Mapless

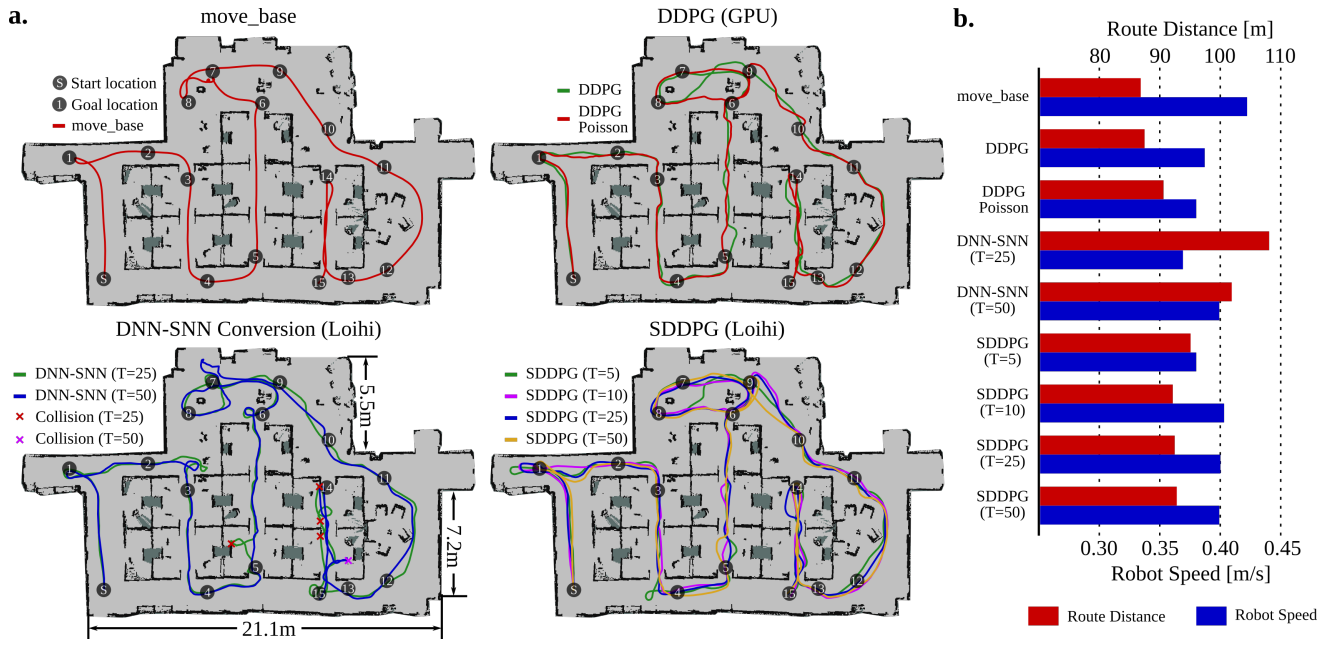

Fig. 7. Comparing SDDPG against other navigation methods in a complex real-world environment containing 15 sequential goal locations covering a large area of the environment. **a.** SDDPG successfully navigated to all the goal locations and took a similar route as the map-based navigation method (move_base) and the other baseline methods. **b.** Route distance and speed metrics are similar for all tested methods.

TABLE II
POWER PERFORMANCE ACROSS HARDWARE

Method	Device	Idle (W)	Dyn (W)	Inf/s	μJ/Inf
DDPG	CPU	13.400	58.802	6598	8910.84
DDPG	GPU	24.154	46.570	3053	15252.91
DDPG	TX2(N)	1.368	1.934	868	2227.41
DDPG	TX2(Q)	1.352	0.457	390	1171.70
SDDPG(50)	Loihi	1.087	0.017	125	131.99
SDDPG(25)	Loihi	1.087	0.014	203	67.29
SDDPG(10)	Loihi	1.087	0.011	396	28.47
SDDPG(5)	Loihi	1.087	0.007	453	**15.53**

navigation methods (DDPG, SDDPG) did not have access to this map. While the DNN-SNN method experienced several collisions in its route, the SDDPG method successfully navigated to all the goal locations and took a similar route as the map-based method (Fig. 7a). Interestingly, SDDPG (T=10, 25, 50) exhibited slightly smoother movements than DDPG and DDPG Poisson (speed comparison in Fig. 7b).

F. Power Performance Measurement

We performed a comparative analysis (Table II) of inference speed and energy consumption for the following mapless navigation solutions: i) DDPG on E5-1660 CPU, ii) DDPG on Tesla K40 GPU, iii) DDPG on Jetson TX2, and iv) SDDPG on Loihi. We measured the average power consumed and the speed of performing off-line inference for the robot states recorded during testing. We used tools that probed the on-board sensors to measure the power for each device: powerstat for CPU, nvidia-smi for GPU, sysfs for TX2, and energy probe for Loihi. The measurements for TX2 were taken for two of its power modes- the energy-efficient mode MAXQ (Q) and the high-performance mode MAXN (N). The energy cost per inference was obtained by dividing the power consumed over a period of time (1s) with the number of inferences performed in that time. Compared to DDPG running on the energy-efficient chip for DNNs, TX2 (Q), SDDPG ($T = 5$) was 75 times more energy-efficient while also having higher inference speed. There was a performance-cost tradeoff associated with SNNs of different timesteps T, suggesting that, for further improvement in navigation performance, SDDPG with larger T may be preferred, albeit with a higher energy cost.

IV. DISCUSSION AND CONCLUSION

Here, we propose a neuromorphic solution to mapless navigation that combines the low power consumption and high robustness capabilities of SNNs with the representation learning capability of DNNs. While recent efforts on integrating the two learning paradigms have focused on training the two networks separately[16], [25], we present a method to train them in conjunction with each other. Our training approach enabled synergistic information exchange between the two networks, allowing them to overcome each other's limitations through shared representation learning; This resulted in an optimal and energy-efficient solution to mapless navigation when deployed on a neuromorphic processor. Such efforts can complement the neuromorphic hardware that currently allow joint inference such as the Tianjic chip [25], and spur the development of hybrid neuromorphic chips for energy-efficient joint training.

Our method was 75 times more energy-efficient than DDPG running on a low-power edge device for DNNs (TX2). This superior performance comes not only from the asynchronous and event-based computations provided by the

SNNs, but also from the ability of our method to train the SNNs for lower values of T with very little loss in performance. This is, however, not the case with the DNN-SNN conversion method, which required 5 times more timesteps and 4.5 times more energy to reach the same level of performance as SDDPG (T=5). This gain in energy-efficiency could enable our method to effectively navigate mobile service robots with limited on-board resources in domestic, industrial, medical, or disaster-relief applications. Further decreases in energy cost may be achieved by coupling our method with low-cost mapless localization methods, such as the ones based on active beacons [26], and by utilizing analog memristive neuromorphic processors [27] that are orders of magnitudes more efficient than their digital counterparts.

While most demonstrations of the SNN advantages focus on the energy gains and come at a cost of a drop in performance [9], [11], [12], this is perhaps the first time that an energy-efficient method also demonstrates better accuracy than the current state of the art, at least in the tested robotic navigation tasks. The accuracy increase is partly due to our hybrid training approach that helped overcome the limitation of SNNs in representing high precision values, which led to better optimization. Moreover, the SNN's inherently noisy representation of its inputs in the spatiotemporal domain might have also enabled it to escape 'bad' local minima. The superior SDDPG results suggest reinforcement learning as a paradigm where the energy-efficient SNNs may also realize their promises for computational robustness and versatility.

Overall, this work supports our ongoing effort to develop neuromorphic solutions for real-time energy-efficient robot navigation. Our mapless solution can complement the current map-based approaches for generating more reliable control policies in applications where maps can be easily acquired. It can also be extended to solve a variety of robotic tasks, paving the way for fully autonomous mobile robots.

REFERENCES

[1] J.-A. Meyer and D. Filliat, "Map-based navigation in mobile robots:: Ii. a review of map-learning and path-planning strategies," *Cognitive Systems Research*, vol. 4, no. 4, pp. 283–317, 2003.

[2] F. Niroui, K. Zhang, Z. Kashino, and G. Nejat, "Deep reinforcement learning robot for search and rescue applications: Exploration in unknown cluttered environments," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 610–617, 2019.

[3] M. Lahijanian, M. Svorenova, A. A. Morye, B. Yeomans, D. Rao, I. Posner, P. Newman, H. Kress-Gazit, and M. Kwiatkowska, "Resource-performance tradeoff analysis for mobile robots," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1840–1847, 2018.

[4] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[5] L. Tai, G. Paolo, and M. Liu, "Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 31–36.

[6] J. Choi, K. Park, M. Kim, and S. Seok, "Deep reinforcement learning of navigation in a complex and crowded environment with a limited field of view," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 5993–6000.

[7] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi, "Target-driven visual navigation in indoor scenes using deep reinforcement learning," in *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2017, pp. 3357–3364.

[8] W. Maass, "Networks of spiking neurons: the third generation of neural network models," *Neural networks*, vol. 10, no. 9, pp. 1659–1671, 1997.

[9] G. Tang, A. Shah, and K. P. Michmizos, "Spiking neural network on neuromorphic hardware for energy-efficient unidimensional slam," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 4176–4181.

[10] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[11] K. D. Fischl, K. Fair, W.-Y. Tsai, J. Sampson, and A. Andreou, "Path planning on the truenorth neurosynaptic system," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2017, pp. 1–4.

[12] H. Blum, A. Dietmüller, M. Milde, J. Conradt, G. Indiveri, and Y. Sandamirskaya, "A neuromorphic controller for a robotic vehicle equipped with a dynamic vision sensor," *Robotics Science and Systems, RSS 2017*, 2017.

[13] Z. Bing, C. Meschede, K. Huang, G. Chen, F. Rohrbein, M. Akl, and A. Knoll, "End to end learning of spiking neural network based on r-stdp for a lane keeping vehicle," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 1–8.

[14] X. Guan and L. Mo, "Unsupervised conditional reflex learning based on convolutional spiking neural network and reward modulation," *IEEE Access*, vol. 8, pp. 17 673–17 690, 2020.

[15] P. U. Diehl, G. Zarrella, A. Cassidy, B. U. Pedroni, and E. Neftci, "Conversion of artificial recurrent neural networks to spiking neural networks for low-power neuromorphic hardware," in *2016 IEEE International Conference on Rebooting Computing (ICRC)*. IEEE, 2016, pp. 1–8.

[16] D. Patel, H. Hazan, D. J. Saunders, H. T. Siegelmann, and R. Kozma, "Improved robustness of reinforcement learning policies upon conversion to spiking neuronal network platforms applied to atari breakout game," *Neural Networks*, vol. 120, pp. 108–115, 2019.

[17] Y. Wu, L. Deng, G. Li, J. Zhu, and L. Shi, "Spatio-temporal backpropagation for training high-performance spiking neural networks," *Frontiers in neuroscience*, vol. 12, p. 331, 2018.

[18] S. B. Shrestha and G. Orchard, "Slayer: Spike layer error reassignment in time," in *Advances in Neural Information Processing Systems*, 2018, pp. 1412–1421.

[19] C. Florensa, D. Held, M. Wulfmeier, M. Zhang, and P. Abbeel, "Reverse curriculum generation for reinforcement learning," in *Conference on Robot Learning*, 2017, pp. 482–495.

[20] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of the 26th annual international conference on machine learning*, 2009, pp. 41–48.

[21] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics & Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.

[22] G. Grisetti, C. Stachniss, and W. Burgard, "Improved techniques for grid mapping with rao-blackwellized particle filters," *IEEE transactions on Robotics*, vol. 23, no. 1, pp. 34–46, 2007.

[23] M. Plappert, R. Houthooft, P. Dhariwal, S. Sidor, R. Y. Chen, X. Chen, T. Asfour, P. Abbeel, and M. Andrychowicz, "Parameter space noise for exploration," in *International Conference on Learning Representations*, 2018.

[24] D. Fox, W. Burgard, F. Dellaert, and S. Thrun, "Monte carlo localization: Efficient position estimation for mobile robots," *AAAI/IAAI*, vol. 1999, no. 343-349, pp. 2–2, 1999.

[25] J. Pei, L. Deng, S. Song, M. Zhao, Y. Zhang, S. Wu, G. Wang, Z. Zou, Z. Wu, W. He, *et al.*, "Towards artificial general intelligence with hybrid tianjic chip architecture," *Nature*, vol. 572, no. 7767, pp. 106–111, 2019.

[26] J. Yun, S. Kim, and J. Lee, "Robust positioning a mobile robot with active beacon sensors," in *International Conference on Knowledge-Based and Intelligent Information and Engineering Systems*. Springer, 2006, pp. 890–897.

[27] A. Ankit, A. Sengupta, P. Panda, and K. Roy, "Resparc: A reconfigurable and energy-efficient architecture with memristive crossbars for deep spiking neural networks," in *Proceedings of the 54th Annual Design Automation Conference 2017*, 2017, pp. 1–6.