2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 14-18, 2024. Abu Dhabi, UAE

CASRL: Collision Avoidance with Spiking Reinforcement Learning Among Dynamic, Decision-Making Agents

Chengjun Zhang[1], Ka-Wa Yip[1], Bo Yang[1], Zhiyong Zhang[1], Mengwen Yuan[1], Rui Yan[2], Huajin Tang[3,4]

***Abstract*— Developing an efficient collision avoidance policy with Spiking Reinforcement Learning for dynamic, decision-making agents remains challenging. Moreover, the implementation of energy-efficient collision avoidance is important for mobile robots that operate with limited on-board computing resources. Most existing energy-efficient methods via spiking reinforcement learning are predominately concerned with the navigational capabilities of a single agent, and are unable to handle a large, and possibly varying number of agents. To overcome these limitations, we propose a model called collision avoidance with spiking reinforcement learning (CASRL), based on proximal policy optimization algorithms. This proposed model consists of an actor with spiking neural networks (SNNs) and a critic with deep neural networks (DNNs). Our spiking reinforcement learning algorithm is advantageous to handle an arbitrary number of other agents by virtue of a spiking-gated transformer (SpikeGTr) architecture and an accumulate-to-fire (ATF) module. Extensive experimental results demonstrate that CASRL obtains a competitive success rate of navigation and exhibits higher time-efficiency for navigation in crowded scenarios compared to traditional DNN-based methods.**

I. Introduction

Collision avoidance algorithms have gained significant interest in the fields of robotics and artificial intelligence, and have found applications in various real-world scenarios, including multi-robot search and rescue, navigation through human crowds, and autonomous warehouse. As shown in Fig. 1, this work aims to solve the collision avoidance problem faced by robots operating in a real-world environment alongside an arbitrary number of other agents by developing a high time-efficiency and high success-rate SNN approach.

The conventional approach involves the establishment of a central server responsible for planning the movements of all agents. Prior centralized methods [1], [2] for collision avoidance necessitate comprehensive knowledge about the intentions and workspace of all agents to facilitate centralized control. Based on the states and goals of all agents, these methods can generate the optimal collision avoidance navigation paths for all agents. However, they require substantial computational cost and communication bandwidth, and are not robust to failures or uncertainties. Thus, instead of planning all agents' paths centrally, several works [3], [4], [5] proposed agent-level decentralized collision avoidance policies, where each agent independently makes its own decision based on its local sensor measurements, such as the shapes, positions and velocities of other agents.

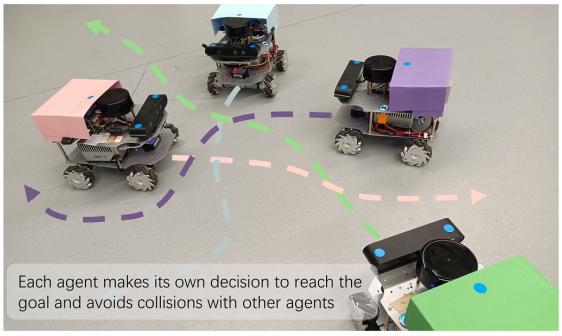


Fig. 1: CASRL is an SNN-based strategy for each agent to generate its own navigation path to the goal, while avoiding collisions with other agents.

The recent decentralized methods applied deep reinforcement learning (DRL) framework for robotics [6], [7]. Inspired by DRL, SNNs have demonstrated energy-efficiency and effectiveness in mobile robot planning and control [8]. Consequently, SNN-based collision avoidance policies can allow for better adaption to resource-constrained mobile robots. Moreover, the recent SNN-based navigation methods [9], [10] utilize spiking deep deterministic policy gradient (SDDPG) to construct a spiking actor network, focusing on enabling one single robot to navigate to its target position.

The challenge of collision avoidance further extends to navigation among dynamic agents featuring a varying number of other agents. Existing methods either establish a maximum number of agents that the model can accommodate [11], or leverage fixed-length lidar sensor data as the input [7]. Another method [6] introduced long short-term memory (LSTM) [12] cells to encode the varying size state of other agents into a fixed-length vector. Building upon the established sequence-to-sequence architecture Transformer [13], [14], this work introduces a Spiking-Gated Transformer (SpikeGTr) architecture to encode the states of other agents with arbitrary lengths more effectively.

Our major contributions are as follows: (i) we use novel SpikeGtr architecture and accumulate-to-fire (ATF) modules to encode arbitrary-length observation data of other agents; (ii) we introduce multi-scenario and multi-agent training strategies for reinforcement learning; (iii) we propose the first SNN-based collision avoidance policy among dynamic agents, which achieves a competitive rate of successful navigation and higher time-efficiency for navigation.

[1]Research Institute of Intelligent Computing, Zhejiang Lab, Hangzhou 311100, China
[2]College of Computer Science and Technology, Zhejiang University of Technology, Hangzhou 310014, China
[3]College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China
[4]The State Key Lab of Brain-Machine Intelligence, Zhejiang University, Hangzhou 310027, China
Corresponding author: H. Tang (htang@zju.edu.cn).

II. BACKGROUND

A. Problem Formulation

The multi-agent collision avoidance problem can be described in the following manner. Consider a scenario with N agents at time t. Each agent i has access to its state $s_{i,t}$ and the sequence state of other agents $\widetilde{S}_{i,t}$, where $i = 1, \cdots, N$. The problem is to output a collision-free action command $\mathbf{a_t}$ that drives each agent i to approach the goal g_i from its current position $p_{i,t}$.

Based on the observation, each agent can independently compute its action command $a_{i,t}$ sampled from a stochastic policy π shared by all agents:

$$\mathbf{a_t} \sim \pi_\theta(\mathbf{a_t}|[\mathbf{s_t}, \widetilde{\mathbf{S}}_\mathbf{t}]), \tag{1}$$

where θ denotes the policy parameters. The policy π guides the agent in approaching its goal, while avoiding collision with other agents.

Therefore, the multi-agent collision avoidance problem can be formulated as a sequential decision making problem [5], [15]. The sequential decisions can be described as observations and actions $([s_{i,t}, \widetilde{S}_{i,t}], a_{i,t})_{t=0:t_i^g}$ made by agent i, where t_i^g is the traveled time to its goal. Hence, the set of trajectories $\mathbb{L}$, which consists of each agent's trajectory l_i from its start position $p_{i,t=0}$ to its desired goal $p_{i,t=t_i^g} \equiv g_i$, can be defined as

$$\begin{aligned}\mathbb{L} =& \{l_i, i = 1, \cdots, N| \\ & a_{i,t} \sim \pi_\theta(a_{i,t}|[s_{i,t}, \widetilde{S}_{i,t}]), \\ & p_{i,t} = p_{i,t-1} + \Delta t \cdot a_{i,t}, \\ & \forall j \in [i, N], j \neq i : \|p_{i,t} - p_{j,t}\| \geq r_i + r_j \\ & \wedge \|a_{i,t}\| \leq a_i^{max}\}.\end{aligned} \tag{2}$$

The policy π shared by all agents is developed with the objective of minimizing the expected time to goal, which is defined as

$$\arg\min_{\pi_\theta} \mathbb{E}\left[\frac{1}{N}\sum_{i=1}^{N} t_i^g \,\middle|\, \pi_\theta\right]. \tag{3}$$

B. Related Work

To ensure multi-agent collision avoidance, early methods, regarded as reaction-based approaches, model the world as a static entity and use one-step interaction rules based on geometry or physics. Reciprocal Velocity Obstacle (RVO) [16] avoids collisions by calculating velocity obstacles, which refer to the obstacles formed by the speed and direction of moving objects at the current moment. An extension of the RVO algorithm, called Hybrid Reciprocal Velocity Obstacle (HRVO) [3], combines tracking error estimation with velocity obstacles to adapt to agents with kinodynamic constraints and unreliable velocity estimates. It can provide a more stable solution for all agents while ensuring collision-free navigation and efficiency. The Optimal Reciprocal Collision Avoidance (ORCA) framework [17], [18] is another advanced algorithm that builds upon the RVO approach, offering more refined velocity adjustments, consideration of multiple obstacles, and adaptability to various agent sizes and movement patterns. Although these reaction-based approaches are computationally efficient, they may sometimes generate unnatural trajectories and their complex parameters are tedious and difficult to be tuned properly.

Learning-based approaches which use reinforcement learning framework to solve Eqs. (2)-(3) have been extensively studied for multi-agent collision avoidance problem. Chen et al. [5] presented a decentralized multi-agent collision avoidance algorithm based on a novel application of DRL, which effectively offloads the online computation to an offline learning procedure. Their other work [15] introduced socially aware collision avoidance with DRL (SA-CADRL) for explaining/inducing socially aware behaviors, and another work [6] adopted LSTM cells to enable their GA3C-CADRL (GPU/CPU Asynchronous Advantage Actor-Critic for Collision Avoidance with Deep Reinforcement Learning) algorithm to use observations of an arbitrary number of other agents. Other learning-based approaches [7], [19] generate actions directly from raw sensor readings such as 2D laser-scans or depth images with end-to-end learning. However, these raw sensor approaches lack the ability to accurately distinguish between dynamic and static obstacles.

With advances in energy-efficient hardware, SNN has become a popular field of robotic applications over the last few years. Directly training SNNs by using gradient-descent techniques such as spatiotemporal backpropagation (STBP) [20], [21] has demonstrated its state-of-the-art performance for a wide range of visual tasks. Moreover, SNNs have been proven to be effective in reinforcement learning [22]. Tang et al. [9] introduced SNNs to deep deterministic policy gradient (DDPG) and demonstrated that a spiking actor network (SAN) achieves good performance in single robot navigation tasks. Furthermore, Yang et al. [10] introduced Spiking-GRU units into the spiking neural networks, endowing them with memory ability, which resulted in improved performance in dynamic and partially observable environments. Saravanan et al. [23] investigated the effectiveness of SNNs in single and multi-agent reinforcement learning environments. In this paper, we focus on learning a collision-avoidance policy for multi-agent environments through the spiking reinforcement learning framework.

III. THE CASRL MODEL

We begin this section with our reinforcement learning framework. Secondly, we introduce the details about our SNNs architecture. Finally, we elaborate on the training protocols used to optimize the policy.

A. Reinforcement Learning Setup

An RL framework can be used to solve the sequential decision making problem defined in Section II-A. The training process of RL seeks to find the optimal policy $\pi : [\mathbf{s_t}, \widetilde{\mathbf{S}}_\mathbf{t}] \mapsto \mathbf{a_t}$, which maps an agent's observation of the environment to a probability distribution across actions. We use the agent's own local coordinate frame [6], [15] to describe the state of the agent and the information of other agents. Specifically, the state of the agent itself is described

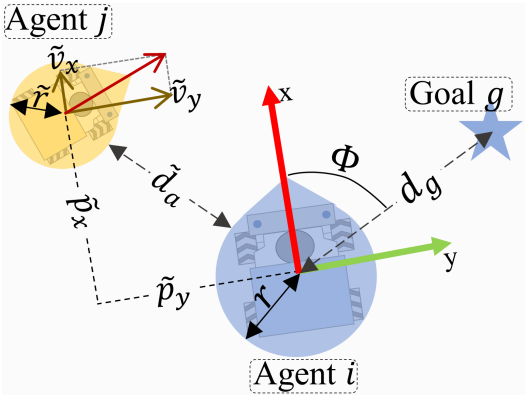


Fig. 2: The observation is defined in the agent's local coordinate frame.

by a single $\mathbf{s}$ vector, and the information of other agents is a sequence state which consists of several $\widetilde{\mathbf{s}}^o$ vectors for each agent in the vicinity. The observation data defined in agent's local coordinate frame is illustrated in Fig. 2. The $\mathbf{s}$ and $\widetilde{\mathbf{s}}^o$ vectors can be formulated as follows:

$$\mathbf{s} = [d_g, \phi, v_{pref}, r] \ , \tag{4}$$

$$\widetilde{\mathbf{s}}^o = \left[\widetilde{p}_x, \widetilde{p}_y, \widetilde{v}_x, \widetilde{v}_y, \widetilde{r}, \widetilde{d}_a, \widetilde{r}+r\right] \ , \tag{5}$$

where d_g is the agent's distance to goal. The angle to goal in the agent's local coordinate is denoted by ϕ. The agent's preferred speed is v_{pref} and r is the agent's radius. The converted data in the agent's local coordinate are $\widetilde{p}_x, \widetilde{p}_y, \widetilde{v}_x, \widetilde{v}_y$. The distance to the other agents is denoted by $\widetilde{d}_a$.

The action space is a set of permissible velocities in discrete space. A natural choice for the action space for a mobile robot includes linear and angular speeds, that is, $a = [v, \omega]$. In this work, we discretize the action space into 11 actions. For a linear speed of v_{pref}, we include 6 angular speeds evenly spaced between $\pm\pi/6$; for linear speeds of $\frac{1}{2}v_{pref}$ and 0, the angular speed choices are $\{-\pi/6, 0, \pi/6\}$.

The main objective of collision avoidance is to avoid collisions with other agents during navigation, while minimizing the expected time to goal of all agents as shown in Eq. (3). We design a reward function $R_t = R_t^g + R_t^c$ to achieve this objective. The reward R_t^g is used to guide the agent toward its goal and prevents it from wandering randomly:

$$R_t^g = \begin{cases} r_{arrival} & \text{if } d_g(t) < G_{th} \ , \\ \omega_g(d_g(t-1) - d_g(t)) & \text{otherwise} \end{cases} \tag{6}$$

where $r_{arrival}$ is the reward at the goal, G_{th} is the goal reaching threshold, and ω_g is the amplitude coefficient for approaching the goal, which can minimize the time required to attain the goal with the objective of maximizing this expected reward. When the agent collides with others or the distance to the closest other agent, d_{min}, is in the danger zone, it is penalized by R_t^c:

$$R_t^c = \begin{cases} r_{collision} & \text{if } d_{min}(t) < 0 \ , \\ \omega_c(d_{min}(t) * 10 - 1.0) & \text{if } 0 \le d_{min}(t) \le 0.1 \ , \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

where $r_{collision}$ is a negative reward to punish the occurrence of collision during the RL process, and the punishment will be linearly decreased until $d_{min} = 0.1$. The symbol ω_c denotes the amplitude coefficient for punishment.

We set $r_{arrival} = 1$, $\omega_g = 0.2 \sim 0.08$, $r_{collision} = -1$ and $\omega_c = 0.05$ in the training procedure.

B. SNNs Architecture

We adopt a hybrid framework with an SNN actor and a DNN critic to form our CASRL, as suggested by previous work [9], this hybrid framework offers evidence that can facilitate overcoming each other's limitations through a shared representation learning. The leaky-integrate-and-fire (LIF) model of a spiking neuron is the base block of our SNNs, which can be described as follows:

$$\mathbf{v}_{l,\bar{t}} = \begin{cases} d_v \cdot (1 - \mathbf{o}_{l,\bar{t}-1}) + h_c(\mathbf{o}_{l-1,\bar{t}}) & \text{if spiking,} \\ d_v + f_c(\mathbf{o}_{l-1,\bar{t}}) & \text{otherwise} \end{cases} \tag{8}$$

$$\mathbf{o}_{l,\bar{t}}[j] = \begin{cases} 1 & \text{if } \mathbf{v}_{l,\bar{t}}[j] > V_{th}, \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

where $\mathbf{v}_{l,\bar{t}}$ is the membrane voltage of layer l at timestep $\bar{t}$, j represents the neuron index in layer l, $\mathbf{o}_{l,\bar{t}}$ is the spike, d_v is the decay factor, V_{th} is the firing threshold, and $h_c(\cdot)$ represents an operation that connects the spikes $\mathbf{o}_{l-1,\bar{t}}$ of the neurons in the previous layer to the membrane voltage of neurons in the current layer.

The LIF neurons with different types of connection are employed in our SNN actor (see Fig. 3). We first use a fully connection layer with LIF neurons to encode the continuous state variables ($N \times 7$) into discrete spikes ($N \times 512$). As feedforward neural networks typically used require a fixed-size input, we apply a sequence encoder (green box in Fig. 3) to modify the arbitrary-length sequence state of other agents to a fixed-size spike input. After concatenating the spikes of the self-state and the fixed-size spikes of N other agents' observable states, we feed this new vector to an SNN with two fully-connected layers to generate the last membrane voltage as our output. Meanwhile, the LIF neurons of the last neuron layer will not generate spikes as defined in Eq. (8). The DNN critic of our CASRL shares a similar architecture with that of SNN actor, by simply replacing all the spike neuron layers in SNN actor with Relu activations.

To enable the sequence encoder to handle a varying number of other agents, we introduce the popular sequence-to-sequence model, the Transformer, to form our Spiking-Gated Transformer (SpikeGtr). A SpikeGtr consists of a Spiking Self Attention (SSA) [24], two SpikeGating blocks and a SpikeMLP block. SSA offers an efficient method to model the information among other agents. The sequence state has been sorted by distance $\widetilde{d}_{a,i}$ from the nearest to the farthest. Considering the primary constraint for agent's decision making is the closest other agent, we apply an $N \times N$ upper triangular matrix as the attention mask (attn-mask) of SSA to enable subsequent sequence to observe the front sequences, while the sequences of the first and the most dangerous (i.e. the closest) other agent only focuses on

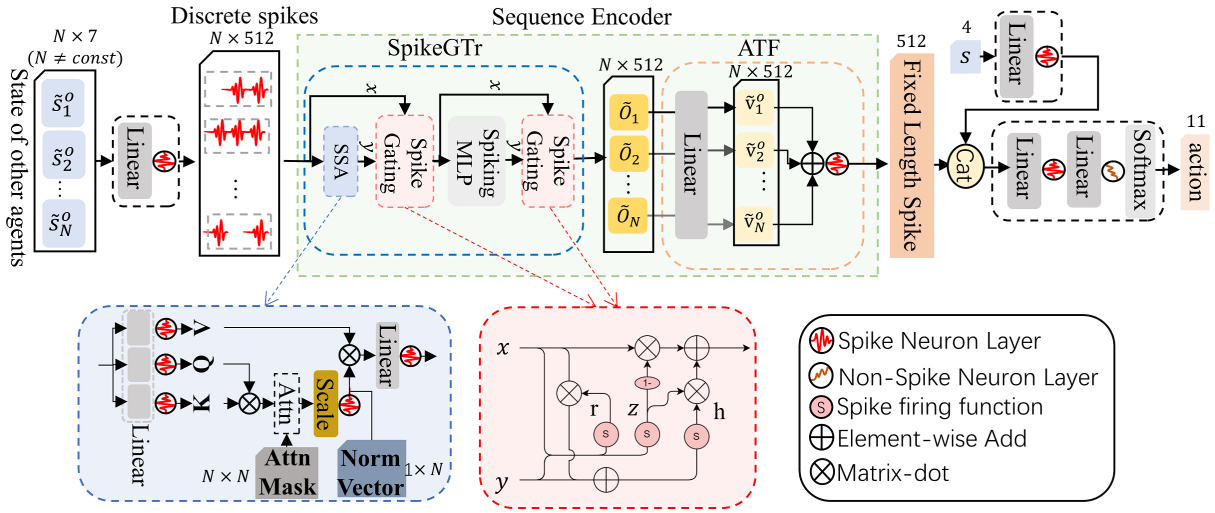


Fig. 3: SNN actor architecture of our CASRL. We employ SpikeGtr architecture and ATF module to construct our sequence encoder to modify the arbitrary-length sequence state of other agents to a fixed length spike. The intricacies of SSA and Spiking Gating unit in the SpikeGtr architecture are further illustrated in the blue and red boxes, respectively. The encoded spike, combined with its own state spike retrieved from s, is fed into two fully-connected layers to generate its final action.

themselves. As SSA uses spikes for Query (Q), Key (K), and Value (V) without softmax, the value of $attn \times V$ spans the range $[0, N]$. Thus, a norm-vector $[1, 1/2, \cdots, 1/N]$ is used to limit the output value to the range $[0, 1]$. Previous work [14] has demonstrated that a Transformer architecture with a Gated Recurrent Unit (GRU) improves both performance and stability in RL tasks. We also introduce SpikeGating layers by replacing the residual connection in Transformer to enhance the stability of the RL process. The SpikeGating (as shown in the red box in Fig. 3) can be written as follows:

$$\begin{aligned} r &= f_s(W_r y + U_r x) \\ z &= f_s(W_z y + U_z x - b_g) \\ h &= f_s(W_g y + U_g(r \odot x)) \\ g(x, y) &= (1 - z) \odot x + z \odot h \end{aligned} \tag{10}$$

where W, U represent the weights, and $f_s(\cdot)$ is the spike firing function which is defined by Eq. (9).

Because the modeled information (shown as $\widetilde{O}_1 \sim \widetilde{O}_N$ in Fig. 3) of other agents by SpikeGtr is still an arbitrary-length sequence data, we create an accumulate-to-firing (ATF) module which utilizes the characteristics of SNN to encode these sequence data into fixed-length spikes. We treat sequence data as inputs at different timesteps into the SNNs. Moreover, as the sequence data of these other agents lack the frequency or time-dependent characteristics required for SNN inputs, we calculate the spike firing function after accumulating all membrane voltages of each timestep instead of calculating firing at every timestep. The voltage is written as follows:

$$\mathbf{v}_{l,\bar{t}} = d_v \cdot (1 - \mathbf{o}_{l,\bar{t}-1}) + \sum_{i=1}^{N} \widetilde{v}^o_{i,\bar{t}} \ . \tag{11}$$

C. Training Protocols

Since the spike firing function that defines a spike is non-differentiable, we extend spatiotemporal backpropagation (STBP) [20], [25] to directly train our SNN actor for learning the optimal policy. The STBP algorithm requires a pseudo-gradient function to approximate the gradient of a spike. We choose three functions as our pseudo-gradient function:

$$z_1(v) = \begin{cases} a_1 & \text{if } |v - V_{th}| < a_2 \,, \\ 0 & \text{otherwise.} \end{cases} \tag{12}$$

$$z_2(v) = \frac{\alpha}{\sqrt{\pi}} e^{-\alpha^2 v^2} \tag{13}$$

$$z_3(v) = \alpha \cdot (1 - sigmoid(\alpha v)) \cdot sigmoid(\alpha v) \tag{14}$$

where z is the pseudo-gradient, a_1 is the amplifier of the gradient and a_2 is the threshold window for passing the gradient for rectangular function. α is the parameter to control smoothness of gradient in both Gaussian error surrogate spiking function and sigmoid function.

A rectangular function (Eq. (12)) is applied for our main network since it has the lowest computational complexity and demonstrated the best empirical performance in [20]. We set $V_{th} = 0$ for both SSA and SpikeGating and choose a Gaussian error surrogate spiking function (Eq. (13)), which has been demonstrated to be competitive in training recurrent SNNs [26]. A sigmoid function is chosen for SSA according to the same selection in [24].

We adopt the proximal policy optimization (PPO) algorithm to our CASRL. In order to adapt our policy to environment with an arbitrary number of agents, we introduce a multi-scenario and multi-agent $(MS\text{-}MA)$ training strategy to PPO as shown in Fig. 4. The K scenarios involving a varying number of agents are treated as our training environment. During data collection with different scenarios, each agent exploits the same policy π to generate trajectories until they meet the states (i.e. reach their goals, collide with other agent or reach the training time T_{train}). Then we calculate the advantage $\hat{A}_t$ for each trajectory and push them into the replay memory M. The replay memory M is used to construct the surrogate loss $L(\theta)$ when M reaches the size

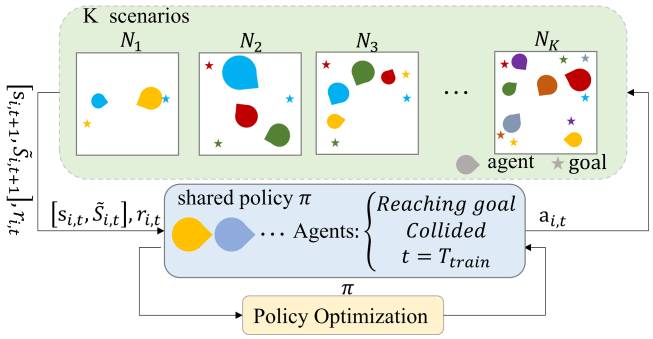


Fig. 4: An overview of our multi-scenario and multi-agent training strategy. The training environment comprises K scenarios with varying numbers of agents.

U_{thresh}. Finally, we update our SNN actor π_θ and DNN critic V_θ by minimizing surrogate loss $L(\theta)$ using gradient descent with Adam optimizer. The training process is summarized in Algorithm 1. This $MS\text{-}MA$ training strategy is suitable for training a policy to handle an arbitrary number of agents, since each agent i in different scenarios can collect trajectories with various lengths of other agents' state — $\tilde{S}_{i,t}$.

Algorithm 1: CASRL

Input: Input parameters: Initialize SNN actor π_θ , DNN critic V_θ, replay memory M

for *each epoch* **do**
  // Collect data
  for *environment* $j = 1, 2, \cdots K$ **do**
    for *agents* $i = 1, 2, \cdots N_j$ **do**
      Run policy π_θ for time $T_{i,j}$
      if $d_g(i,j,t) < G_{th}$,$d_{min}(i,j,t) < 0$, *or* $T_{i,j} = T_{train}$ **then**
        Estimate advantages using GAE [27]
        $\hat{A}_{i,j,t} = \sum_{l=0}^{T_{i,j}} (\gamma\lambda)^l \delta_{i,j,t}$, where
        $\delta_{i,j,t} = r_{i,j,t} + \gamma V_{i,j,t+1} - V_{i,j,t}$
        collect $\{[\mathbf{s}, \tilde{\mathbf{S}}], \mathbf{r}, \mathbf{a}, \hat{\mathbf{A}}\} \rightarrow M$
      end
      break if $Size(M) > U_{thresh}$
    end
  end
  $\pi_{old} \leftarrow \pi_\theta$
  // Update policy and value net
  for $k = 1, \cdots E_{update}$ **do**
    sample training data from M
    $L^{CLIP}(\theta) =$
    $\hat{\mathbb{E}}_t[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)]$,
    where $r_t(\theta) = \frac{\pi_\theta(a_t|[s_n, \tilde{S}_t])}{\pi_{\theta_{old}}(a_t|[s_n, \tilde{S}_t])}$
    $L^{VF}(\theta) = \hat{\mathbb{E}}_t[V_\theta([s_n, \tilde{S}_t] - V_t^{targ})^2]$
    $L(\theta) = L^{CLIP}(\theta) - L^{VF}(\theta) + c\hat{\mathbb{E}}_t D[\pi_\theta]$,
    where D denotes an entropy bonus
    Optimize surrogate $L(\theta)$ wrt θ
  end
  clear memory M
end

IV. Results

A. Implementation Setup

The implementation of our algorithm is in Pytorch and the simulator is gym-collision-avoidance [6], [28]. We train multi-agent collision avoidance policy on a computer with an AMD Ryzen ThreadRipper pro 3995WX CPU and a Nvidia RTX A6000 GPU. We train our policy over 50,000 epochs, divided equally into two stages Since scenarios with fewer agents can lead to faster policy convergence, we set the maximum agent number in the first stage (scenario N_{max}) to 4. Then, to achieve robust performance in more complex scenarios, we set N_{max} to 10 in the second stage. The number of scenarios in training environment is set to $K = 16$ for both stages. To improve the solution with RL, agents are running a random assortment of policies (Non-Cooperative, Zero Velocity, or the learned CASRL policy) as in a previous work [6]. The agent parameters are chosen to be close to pedestrian values in real-world application: $r \in [0.2, 0.8]$m, and $v_{pref} \in [0.5, 2.0]$m/s.

Other key parameters are as follows. SNN's time-window $\bar{T} = 3$; in LIF neurons: $V_{th} = 0.5, d_v = 0.5$, learning rate $l_r = 2 \cdot 10^{-5}$; in Algorithm 1: $\lambda = 0.95, \gamma = 0.99, \epsilon = 0.2, c = 0.02, U_{thresh} = 1024, T_{train} = 32, E_{update} = 10$.

B. Simulation Results

We compare our proposed CASRL algorithm to several other approaches: 1) ORCA [17]: ORCA makes a minimal adjustment to agents' desired velocities at each step to make sure they do not collide with each other. The time horizon parameter in ORCA is set to 5 seconds and the collaboration coefficient is set to 0.5; 2) SA-CADRL [15]: SA-CADRL algorithm is capable of accepting up to 3 nearby agents' states to adapt to environments with varying numbers of agents; 3) GA3C-CADRL [6]: To perform a fair comparison, GA3C-CADRL is trained in scenarios of up to 10 agents; 4) SDDPG [9]: We re-trained the weights of SDDPG in multi-agent environments with $agents = 10$.

We created test sets consisting of 500 random scenarios with different numbers of agents. Each algorithm is evaluated on the same 500 test cases. The performance metrics used for comparison are as follows:

a. **Failures**: It accounts for the percentage of cases with a collision, and the percentage of cases where an agent gets stuck or does not reach the goal.

b. **Extra time to goal** $\hat{t}_g^e$: It is the average extra time to goal beyond a straight path at the speed of v_{pref}.

The results of various algorithms are summarized in Table I. Since SA-CADRL can accept up to 3 other agents' states as input, it performs worse on scenarios involving $agents \geq 8$ (the success rate drops below 80%). GA3C-CADRL provides highly competitive performance with success rate above 90% in both few-agent scenarios and crowded scenarios; however, it leads to longer navigation times. As SDDPG method utilizes LiDAR data as input, it struggles

TABLE I: **Performance on 500 random test cases.** CASRL outperforms other algorithms in extra time to goal when $agents \geq 4$ and has a competitive navigation success rate. Notably, it achieves the highest success rate among all methods for $agents = 2, 6$.

Agents	2	3	4	6	8	10
Method	% failures (% collisions / % stuck)					
ORCA	4.4 (4.4 / 0.0)	11.0 (8.6 / 2.4)	12.4 (9.2 / 3.2)	21.2 (16.8 / 4.4)	27.0 (19.2 / 7.8)	33.2 (23.4 / 9.8)
SA-CADRL	0.6 (0.6 / 0.0)	1.2 (1.0 / 0.2)	3.4 (0.6 / 2.8)	10.2 (3.0 / 7.2)	21.8 (8.6 / 13.2)	26.8 (12.4 / 14.4)
GA3C-CADRL	0.6 (0.6 / 0.0)	**0.2 (0.2 / 0.0)**	**1.2 (0.8 / 0.4)**	7.8 (7.2 / 0.6)	**8.8 (4.6 / 4.2)**	**10.0 (5.8 / 4.2)**
SDDPG	25.6 (25.6 / 0.0)	43.4 (40.4 / 3.0)	55.8 (51.2 / 4.6)	74.4 (69.6 / 4.8)	92.0 (81.8 / 10.2)	93.2 (86.4 / 6.8)
CASRL	**0.4 (0.0 / 0.4)**	0.8 (0.0 / 0.8)	4.6 (0.0 / 4.6)	**6.2 (0.4 / 5.8)**	12.8 (2.4 / 10.4)	14.0 (3.8 / 10.2)
Method	Extra time to goal $\hat{t}_g^e$(s) (50th / 75th / 90th percentile)					
ORCA	0.53 / 1.05 / 3.04	0.96 / 1.51 / 2.33	1.46 / 2.42 / 4.34	2.24 / 3.91 / 10.78	2.80 / 4.34 / 6.94	2.74 / 4.08 / 5.87
SA-CADRL	**0.38 / 0.59 / 1.10**	**0.44 / 0.72 / 1.18**	0.73 / 1.16 / 1.68	1.07 / 1.48 / 2.01	1.29 / 1.73 / 2.25	1.46 / 1.87 / 2.31
GA3C-CADRL	0.47 / 0.83 / 1.80	0.67 / 1.23 / 1.92	0.95 / 1.44 / 2.09	1.45 / 2.06 / 2.73	1.60 / 2.08 / 2.62	1.83 / 2.33 / 2.83
SDDPG	1.53 / 2.15 / 2.80	1.76 / 2.54 / 3.13	2.14 / 2.78 / 3.56	2.64 / 3.18 / 3.95	2.84 / 3.38 / 4.80	3.26 / 3.80 / 4.07
CASRL	0.52 / 1.08 / 1.90	0.50 / 0.83 / 1.29	**0.68 / 0.95 / 1.32**	**0.94 / 1.35 / 1.78**	**1.11 / 1.43 / 1.83**	**1.11 / 1.45 / 1.76**

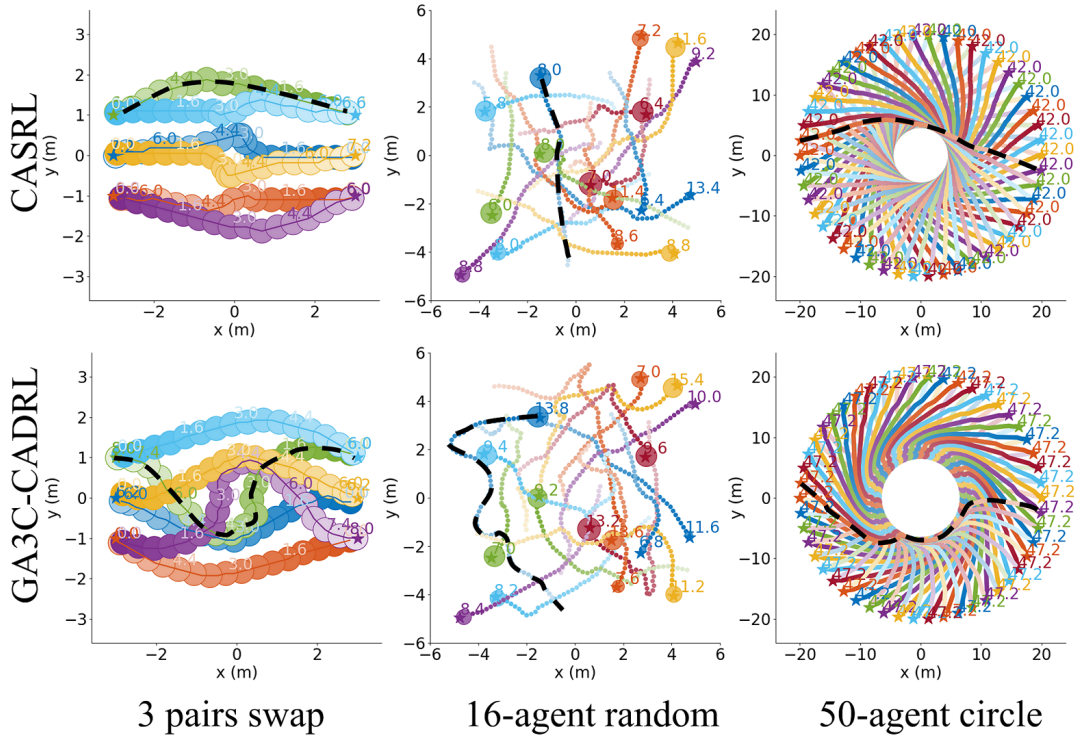


Fig. 5: We showcase three experiments to illustrate the learned policy's capability to manage a varying number ($\{6, 16, 50\}$) of agents. In all cases, the marked CASRL's navigation paths appear to be smoother than GA3C-CADRL, accounting for CASRL's time-efficiency for navigation.

to differentiate dynamic agents and predict their subsequent actions, resulting in a collision rate greater than 50% for $agents \geq 4$. Our CASRL (SpikeGtr+ATF) exhibits a high success rate (above 86% in all test scenarios), which is consistently superior to ORCA, SA-CADRL and SDDPG, and in some scenarios ($agents = 2, 6$) superior to GA3C-CADRL. The result $\hat{t}_g^e$ in Table I also indicates that our method demonstrates higher time-efficiency for navigation, particularly in crowded scenarios ($agents \geq 4$). Trajectories visualized in Fig. 5 indicate that our algorithm tends to choose trajectories close to a straight path which enables agents to reach their goals more quickly; whereas, the paths generated by GA3C-CADRL appear more complex and longer because its agents are more inclined to perform escape maneuvers to avoid collisions. Moreover, it is worth noting that the failures of CASRL often result from getting stuck rather than colliding with others (demonstrating the lowest collision failures among other algorithms). This is because agents are more inclined to choose straight paths to their goals for efficient navigation. However, executing an escape maneuver can help agents avoid getting stuck.

To demonstrate the effectiveness of our proposed sequence encoder (SpikeGtr+ATF), we also train our CASRL policies with four different types of sequence encoder: 1) SpikeGtr+ATF; 2) SpikeGtr (which uses the average mean to convert sequence data to fixed-length); 3) ATF; 4) Spiking Long Short-Term Memory (SpikeLSTM) [29]. For a fair energy-efficiency comparison, we also generate a traditional DNN collision avoidance policy with Gated Transformer architecture (GTr) through our $MS\text{-}MA$ training strategy.

TABLE II: **Performance with different sequence encoders.** SpikeGtr+ATF outperforms other encoders in navigation success rate and spends less time to goal for $agents = 6, 10$.

Agents	2	6	10
Encoder	% failures (% collisions / % stuck)		
GTr	0.8 (0.2 / 0.6)	13.6 (5.4 / 8.2)	40.0 (23.6 / 16.4)
SpikeLSTM	1.0 (0.0 / 1.0)	6.4 (3.6 / 2.8)	17.2 (11.4 / 5.8)
SpikeGtr	0.8 (0.0 / 0.8)	9.2 (0.4 / 8.8)	16.2 (2.0 / 14.2)
ATF	9.2 (0.2 / 9.0)	14.2 (1.6 / 12.6)	24.0 (10.6 / 13.4)
SpikeGtr+ATF	**0.4 (0.0 / 0.4)**	**6.2 (0.4 / 5.8)**	**14.0 (3.8 / 10.2)**
Encoder	$\hat{t}_g^e$(s) (50th / 75th / 90th percentile)		
GTr	0.73 / 1.08 / 1.56	1.78 / 2.50 / 3.16	2.37 / 2.92 / 3.53
SpikeLSTM	0.61 / 0.89 / 1.32	1.31 / 1.85 / 2.65	1.69 / 2.20 / 2.76
SpikeGtr	0.45 / 0.70 / 1.98	0.95 / 1.38 / 1.78	1.22 / 1.59 / 1.94
ATF	**0.51 / 0.84 / 1.28**	1.45 / 2.03 / 2.61	1.78 / 2.19 / 2.91
SpikeGtr+ATF	0.52 / 1.08 / 1.90	**0.94 / 1.35 / 1.78**	**1.11 / 1.45 / 1.76**

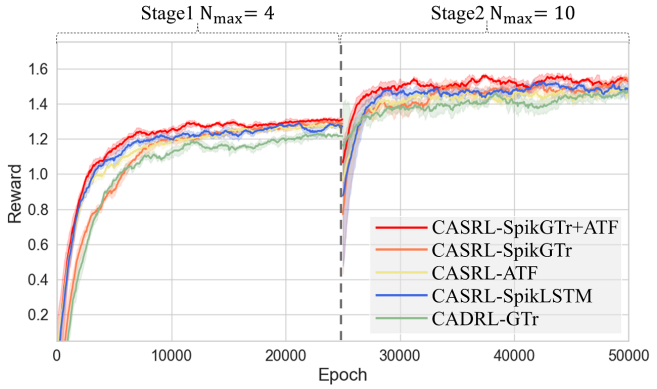


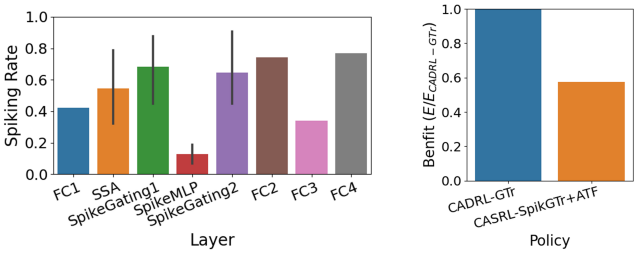
Fig. 6: Average rewards shown in epoch during the training process.

As ATF is a spiking based module, we only consider the average mean for GTr while not combining GTr+ATF. The reward curves during training process are shown in Fig. 6. Our proposed algorithm with SpikeGtr+ATF obtains higher rewards than other encoder types.

Compared to other SNN encoders, the success rate of using SpikeGtr+ATF in all scenarios is significantly higher than other approaches (See Table. II). Although previous works [30], [14], [31] have proven the applicability of transformers in DRL, our experiments indicate that DNN does not match the performance of SNN. This could possibly be due to the lack of training samples, or because the task is not suitable for the Transformer architecture of DNNs. Consequently, SpikeGtr+ATF has the best encoding performance for an arbitrary number of other agents among all other encoders.

C. Energy Efficiency

To examine energy efficiency, we consider the computing operations of CASRL-SpikeGtr+ATF and CADRL-GTr with similar numbers of neurons and connections. The multiply-accumulate (MAC) operations are primarily responsible during execution in DNNs. SNNs, however, perform accumulate (AC) operations because spike events are binary operations whose input is integrated (or accumulated) into a membrane potential only upon receiving spikes. According to standard CMOS technology [32], a 32-bit floating-point (FL) MAC operation consumes 4.6pJ $(0.9 + 3.7$pJ$)$, while an AC operation requires 0.9pJ. By multiplying the spiking rate of SNNs, we can determine the amount of AC operations and calculate the energy consumption. As shown in Fig. 7a, the spiking rate of every layer (and among all timesteps, with error bars representing the standard deviation) is lower than 1. By further accounting the MAC and AC operations of the architectures, we demonstrate that our SNN method achieves high energy efficiency, consuming only 57% of the energy consumed by DNN's counterpart, as depicted in Fig. 7b.



(a) Spiking rates (b) Energy efficiency

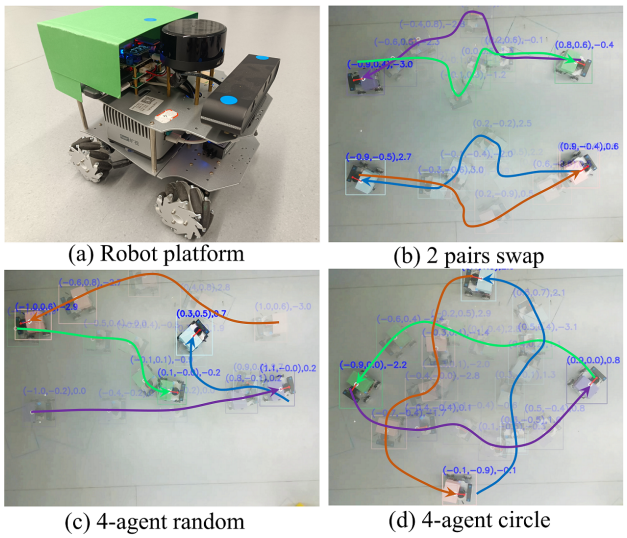
Fig. 7: (a) Spiking rates of each layer in CASRL. (b) SNN-based policy consumes 57% of DNN's energy consumption.

D. Robotic Experiment

The CASRL policy is implemented for multi-robot for autonomous navigation in an indoor environment. The robot platform, equipped with a 2D Lidar for localization, is shown in Fig. 8a. Robots share their positions and states via a wireless network. We adopt 4 robots on a $2m \times 3m$ ground and test with 3 scenarios (2-pair swap, 4-agent random and 4-agent circle) for 10 times each. A top-down camera is used to record the navigation trajectories of all robots. To avoid robot collisions caused by errors, the radius r of the robots is set to $0.35m$ in the model (the actual robot size is $0.22m \times 0.16m$). Robots successfully reach their goals in all test cases, as shown in Fig. 8b∼d. The navigation paths in real experiments also indicate differences from those in the simulation tests, owning to localization errors and inaccuracies in motion trajectories in real environments.

V. Conclusion

This work presented CASRL, an efficient collision avoidance algorithm developed with our proposed multi-scenario and multi-agent training strategy. We introduced novel structures, SpikeGtr+ATF, to the SNN actor, capable of encoding the states of other agents with varying lengths into a fixed-length world representation. The new approach was demonstrated to have a highly competitive navigation success rate and outperform the existing methods in time-efficiency at crowded scenarios. Compared to DNN model, our proposed SNN-based method demonstrated energy-efficiency. The new algorithm was also implemented in multi-robot setups to verify its effectiveness in real-world experiments.



(a) Robot platform (b) 2 pairs swap

(c) 4-agent random (d) 4-agent circle

Fig. 8: (a) Robot platform. (b∼d) Visualization of navigation paths in 3 test scenarios with CASRL. A video is appended with this manuscript.

ACKNOWLEDGMENT

This work was supported by the National Natural Science Foundation of China under Grant 62236007, NSAF Grant U2030204 and Grant 62276235.

REFERENCES

[1] J. Yu and S. M. LaValle, "Optimal multirobot path planning on graphs: Complete algorithms and effective heuristics," *IEEE Transactions on Robotics*, vol. 32, no. 5, pp. 1163–1177, 2016.

[2] F. Augugliaro, A. P. Schoellig, and R. D'Andrea, "Generation of collision-free trajectories for a quadrocopter fleet: A sequential convex programming approach," in *2012 IEEE/RSJ international conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 1917–1922.

[3] J. Snape, J. Van Den Berg, S. J. Guy, and D. Manocha, "The hybrid reciprocal velocity obstacle," *IEEE Transactions on Robotics*, vol. 27, no. 4, pp. 696–706, 2011.

[4] D. Claes, D. Hennes, K. Tuyls, and W. Meeussen, "Collision avoidance under bounded localization uncertainty," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 1192–1198.

[5] Y. F. Chen, M. Liu, M. Everett, and J. P. How, "Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning," in *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2017, pp. 285–292.

[6] M. Everett, Y. F. Chen, and J. P. How, "Motion planning among dynamic, decision-making agents with deep reinforcement learning," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 3052–3059.

[7] P. Long, T. Fan, X. Liao, W. Liu, H. Zhang, and J. Pan, "Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning," in *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 6252–6259.

[8] 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.

[9] G. Tang, N. Kumar, and K. P. Michmizos, "Reinforcement co-learning of deep and spiking neural networks for energy-efficient mapless navigation with neuromorphic hardware," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 6090–6097.

[10] B. Yang, M. Yuan, C. Zhang, C. Hong, G. Pan, and H. Tang, "Spiking reinforcement learning with memory ability for mapless navigation," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2023, pp. 1–8.

[11] M. Kuderer, H. Kretzschmar, C. Sprunk, and W. Burgard, "Feature-based prediction of trajectories for socially compliant navigation." in *Robotics: science and systems*, 2012.

[12] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[13] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[14] E. Parisotto, F. Song, J. Rae, R. Pascanu, C. Gulcehre, S. Jayakumar, M. Jaderberg, R. L. Kaufman, A. Clark, and S. Noury, "Stabilizing transformers for reinforcement learning," in *International Conference on Machine Learning*, 2020.

[15] Y. F. Chen, M. Everett, M. Liu, and J. P. How, "Socially aware motion planning with deep reinforcement learning," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 1343–1350.

[16] J. Van den Berg, M. Lin, and D. Manocha, "Reciprocal velocity obstacles for real-time multi-agent navigation," in *2008 IEEE international conference on robotics and automation*. Ieee, 2008, pp. 1928–1935.

[17] J. Van Den Berg, S. J. Guy, M. Lin, and D. Manocha, "Reciprocal n-body collision avoidance," in *Robotics Research: The 14th International Symposium ISRR*. Springer, 2011, pp. 3–19.

[18] D. Bareiss and J. Van den Berg, "Generalized reciprocal collision avoidance," *The International Journal of Robotics Research*, vol. 34, no. 12, pp. 1501–1514, 2015.

[19] L. Tai, J. Zhang, M. Liu, and W. Burgard, "Socially compliant navigation through raw depth inputs with generative adversarial imitation learning," in *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 1111–1117.

[20] 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.

[21] M. Yuan, C. Zhang, Z. Wang, H. Liu, G. Pan, and H. Tang, "Trainable spiking-yolo for low-latency and high-performance object detection," *Neural Networks*, vol. 172, p. 106092, 2024.

[22] M. Yuan, X. Wu, R. Yan, and H. Tang, "Reinforcement learning in spiking neural networks with stochastic and deterministic synapses," *Neural computation*, vol. 31, no. 12, pp. 2368–2389, 2019.

[23] M. Saravanan, P. S. Kumar, K. Dey, S. Gaddamidi, and A. R. Kumar, "Exploring spiking neural networks in single and multi-agent rl methods," in *2021 International Conference on Rebooting Computing (ICRC)*. IEEE, 2021, pp. 88–98.

[24] Z. Zhou, Y. Zhu, C. He, Y. Wang, S. Yan, Y. Tian, and L. Yuan, "Spikformer: When spiking neural network meets transformer," *arXiv preprint arXiv:2209.15425*, 2022.

[25] P. Gu, R. Xiao, G. Pan, and H. Tang, "Stca: Spatio-temporal credit assignment with delayed feedback in deep spiking neural networks." in *IJCAI*, vol. 15, 2019, pp. 1366–1372.

[26] B. Yin, F. Corradi, and S. M. Bohté, "Effective and efficient computation with multiple-timescale spiking recurrent neural networks," in *International Conference on Neuromorphic Systems 2020*, 2020, pp. 1–8.

[27] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[28] M. Everett, Y. F. Chen, and J. P. How, "Collision avoidance in pedestrian-rich environments with deep reinforcement learning," *IEEE Access*, vol. 9, pp. 10 357–10 377, 2021.

[29] A. Lotfi Rezaabad and S. Vishwanath, "Long short-term memory spiking networks and their applications," in *International Conference on Neuromorphic Systems 2020*, 2020, pp. 1–9.

[30] W. Li, H. Luo, Z. Lin, C. Zhang, Z. Lu, and D. Ye, "A survey on transformers in reinforcement learning," *arXiv preprint arXiv:2301.03044*, 2023.

[31] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch, "Decision transformer: Reinforcement learning via sequence modeling," *Advances in neural information processing systems*, vol. 34, pp. 15 084–15 097, 2021.

[32] M. Horowitz, "1.1 computing's energy problem (and what we can do about it)," in *2014 IEEE international solid-state circuits conference digest of technical papers (ISSCC)*. IEEE, 2014, pp. 10–14.