An Energy-Efficient Lane-Keeping System Using 3D LiDAR Based on Spiking Neural Network

Genghang Zhuang[1], Zhenshan Bing[1], Zhen Zhou[1], Xiangtong Yao[1], Yuhong Huang[1], Kai Huang[2], and Alois Knoll[1]

***Abstract*— Lane keeping, as a fundamental functionality of autonomous navigation, remains a challenging task for autonomous robots and vehicles. Recently, spiking neural networks (SNNs) have gained attention and research interest due to their biological plausibility and application potential on neuromorphic processors. SNNs have also been successfully deployed on robots to solve autonomous navigation problems. However, lane keeping with a LiDAR sensor is still an open problem for SNNs. In this work, we propose an end-to-end approach based on an SNN to solve the lane-keeping problem using a 3D LiDAR sensor. For the first time, we explore the capability of the proposed SNN controller to perceive the LiDAR input and exploit the features to perform reward-based feedback learning. To ensure the effectiveness of the controller, the proposed method is deployed and evaluated on two high-fidelity simulators. The experimental results demonstrate the high applicability and performance in different scenarios. Furthermore, experiments show that the SNN is capable of performing lane keeping in a simulated urban environment with only 18 control neurons and 32 synapse connections, producing on average only a 17cm deviation from lane center, which is 4.3% of the lane width.**

I. Introduction

Autonomous navigation poses a formidable challenge for autonomous robots and vehicles, which involves algorithms for path planning, obstacle perception and avoidance. High autonomy for robots hinges on precise perception based on sensors to solve navigation tasks. To this end, various sensors have been successfully deployed on robots, including cameras, sonar sensors, and LiDARs. Among the utilized sensors, LiDAR has the advantages of high accuracy, low noise, and high robustness with luminance changes in the environment, which make it a commonly used sensor for autonomous driving [1]. Although the algorithm complexity can increase due to the large data scale of LiDAR, LiDARs have been extensively used for autonomous driving tasks such as object detection and semantic segmentation [1], [2].

With the advent of deep representation learning and artificial neural networks, deep neural networks (DNNs) have become a powerful end-to-end approach and framework for solving autonomous navigation problems due to their extensive applications and the capability of learning intricate policies in high-dimensional environments. However, DNNs that only rely on deep convolutional layers, devoid of temporal information from sensor data, are often prone to noisy and inaccurate input [3]. In addition, the high complexity of DNNs engenders computationally expensive and power-consuming training and inference, which makes them barely suited to deployment on power-constrained vehicles.

In recent years, spiking neural networks (SNNs) have gained attention due to the feature that an SNN is able to more closely mimic natural neural networks based on the temporal synaptic model and spike-timing-dependent plasticity. Besides, SNNs have the potential to be deployed on neuromorphic processors, which enable the networks to perform highly efficient computation [4]. Many studies have investigated the efficacy of SNNs on autonomous navigation tasks based on different sensors [5]–[7]. However, rare research exists on the utilization of SNNs to process 3D LiDAR sensor data for lane-keeping tasks. Mitchell et al. in [5] and Shalumov et al. in [8] respectively proposed two SNN models with 2D LiDARs to perform basic collision avoidance. However, without reward-modulated feedback learning, the models are not able to perform some advanced tasks, such as lane keeping on a bidirectional road without physical lane separators. Bing et al., in the previous work [9], [10], proposed a SNN for the dynamic vision sensor (DVS) to solve the lane-keeping problem. By utilizing the reward-modulated spike-timing-dependent plasticity (R-STDP) rule, this method performs feedback learning for the SNN [11]. Despite that, the method is not suited to unstructured point clouds from LiDARs. In contrast to DVS, LiDAR presents relatively high performance and robustness for lane feature extraction in noisy and complex urban scenes [12], [13], and LiDAR remains a more prevalent sensor on autonomous vehicles. Hence, utilizing LiDAR can further improve the performance and robustness of spike-based navigation. However, there is a dearth of research with regard to learning-based lane keeping SNNs using LiDARs.

In this work, for the first time, we propose an end-to-end approach to address the lane-keeping problem with a 3D LiDAR sensor based on a spiking neural network. The primary contributions are distilled and listed as follows: First, we propose an end-to-end learning-based SNN controller that is suited and targeted for the LiDAR sensor. The proposed SNN controller is capable of perceiving the LiDAR input using the convolutional layers as the lane feature extractor and exploiting the feature input to perform feedback learning from the environment to solve the lane-keeping problem. Second, to evaluate the effectiveness and performance of the proposed method, the experimental environments and testing scenarios are built on the high-fidelity simulators CoppeliaSim and CARLA, in which the proposed approach is deployed and analyzed. The experimental results demonstrate the high generalizability and performance of the proposed method in different environments and scenarios. One of the experiments shows that the SNN controller is capable of performing lane-keeping in the CARLA urban scenario with only 18 control neurons and 32 synapses.

[1]School of Computation, Information and Technology, Technical University of Munich; Corresponding author: Zhenshan Bing.
[2]School of Computer Science and Engineering, Sun Yat-sen University.

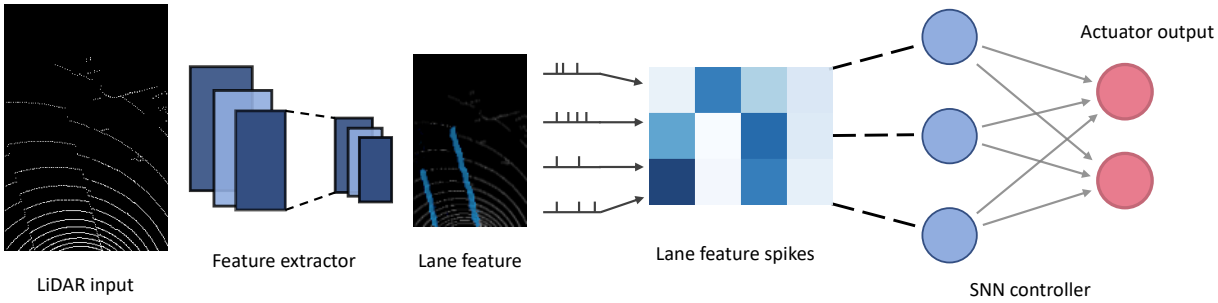


Fig. 1. Architecture of the lane-keeping system, which consists of the lane feature extractor and the SNN controller. The SNN controller perceives the LiDAR input extracted by the convolutional lane feature extractor and utilizes the lane feature to infer the actuator output.

II. Methodology

The proposed lane-keeping system consists of two main components, the lane feature extractor, and the SNN controller, as shown in Fig. 1. The SNN controller perceives the LiDAR sensor input that is extracted and transformed by the lane feature extractor consisting of convolutional layers. The input lane feature is utilized by the SNN controller to activate and generate the actuator output to drive the robot.

A. Lane Feature Extractor

In this work, we adapted a convolutional network to roughly segment the lane area from the LiDAR input to generate the lane features. The perception results are encoded to activate the sensory neurons of the SNN controller.

1) Point Cloud Data Preprocessing

The convolutional layers for lane feature extractor expects a 2D input of a fixed size. To adapt the unstructured point cloud into a fixed-size input format for the network, preprocessing of the LiDAR point cloud is performed. The serial point cloud data is transformed into a 2D top view grid-based representation as the input of the lane feature extractor network. The point cloud data is first filtered based on the specified region of interest (ROI). For an ROI that covers $15m$ for the front and $5m$ for the left and right sides of the robot, the points $(x, y, z)^T$, in the left-handed coordinate system with the x-axis facing forward, are retained if $x \in [0, 15],\ y \in [-5, 5]$. The filtered points are then downsampled into a grid based on the grid size. For a $0.1m$ grid size, the preprocessing generates the top view representation with a size of 150×100. The top view representations for the point cloud data are subsequently scaled to the input size of the lane feature extractor.

2) Convolutional Feature Extractor

The lane feature extractor is implemented based on a lightweight fully convolutional network designed for lane segmentation [13]. The original lane segmentation network consists of 5 convolutional layers, 2 pooling layers, 7 dilated convolutional layers, and an unpooling layer. These layers compose the encoder, context module, and the decoder of the network to perform the semantic segmentation task. In order to extract a lane feature suited to the SNN controller, the network is re-trained to detect only the lane boundary features to strengthen the sparsity of the network output, which subsequently facilitates the training of the SNN controller. Moreover, to additionally reduce the network scale, the original decoder module is removed, where the unpooling layer is bypassed, and the encoder and context module are retained from the trained model and connected with a pooling layer to further shrink the output size.

To make use of the lane feature results and feed them into the sensory neurons in the SNN-based controller, the continuous values of the lane feature must be properly converted into spikes. In the proposed method, frequency encoding based on a Poisson spike generator [14] is employed to convert the LiDAR rough lane feature output to spikes, which is a common approach that has been successfully deployed for dynamic vision sensors and camera sensors [11]. The result from the lane feature extractor is first cropped based on the receptive field of the sensory input of the SNN controller, which can be defined as the ROI for the SNN input. Subsequently, the continuous values of the lane feature are connected and fed to the spike generators to generate the sparse spikes for the lane boundary. In the process, a many-to-one connection fashion for the connections from the lane feature and spike generators is implemented in order to perform downsampling for the lane feature and modulate the spike frequency.

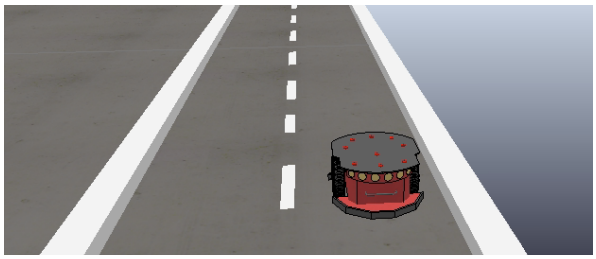
B. SNN Controller

The spiking neural network of the controller consists of sensory neurons, two actuator neurons, and the fully connected synapses between the sensory neurons and the actuator neurons. The sensory neurons are Poisson neurons that accept activation from the LiDAR perception input and emit spike trains to the actuator neurons via the synapses. The number of sensory neurons varies depending on the LiDAR configuration and environment settings. The two actuator neurons correspond to the left and right motors for a differential drive model. To exploit the temporal information from the sensor input, the leaky integrate-and-fire (LIF) neuron model is set up for the actuator neurons.

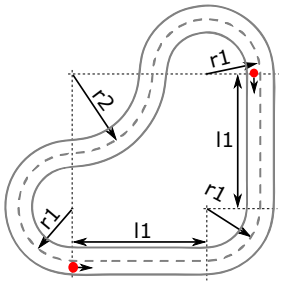
1) Feedback Learning

In order to control the robot for autonomous navigation in the environment, the parameters and weights of the synapse connections in the SNN controller should be precisely tuned and trained. In this work, a training method based on the spike-timing-dependent plasticity (STDP) rule is utilized for

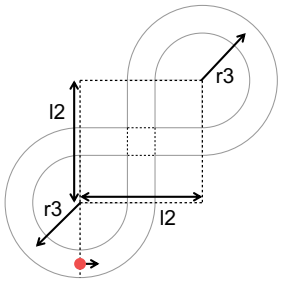
The source code of the proposed networks and resources can be found at: `https://github.com/tum-in-zhuang/lidar-snn-navigation`



(a) Robot agent in the CoppeliaSim simulator



(b) Scenario 1: Bidirectional loop with curb of $0.1m$ height. $r1 = 1.75m$, $r2 = 2.75m$, $l1 = 5.0m$



(c) Scenario 2: One-way loop with curb of $0.1m$ height. $r3 = 2.0m$, $l2 = 4.0m$

Fig. 2. Experiment scenarios on CoppeliaSim

the SNN controller. The STDP learning rule, which is derived from Hebbian learning theory, corresponds to the learning and refinement mechanisms of neuronal circuits in the brain and has been widely adopted in SNN training [15]. In the training process, the R-STDP method that combines STDP with the rewards from the environment is used to carry out the feedback learning for the controller [16], [17], so that the SNN can adjust the synapse weights actively based on the environment feedback and learn the temporal associations between the LiDAR sensor input and the proper speed output for the lane-keeping task. For the simulation environment scenarios involved in the experiments, the reward is determined based on the deviations of the robot's position from the center and the orientation from the lane direction. To drive the robot back to the center of the lane when it deviates, one motor should suppress the other motor or in the opposite way according to the deviation direction. Therefore, the reward for the left actuator is the additive inverse of the reward for the right actuator. The rewards are defined as follows:

$$\begin{aligned} r_l &= -(\gamma_d \cdot d + \gamma_\theta \cdot v \cdot tan\,\theta), \\ r_r &= \gamma_d \cdot d + \gamma_\theta \cdot v \cdot tan\,\theta, \end{aligned} \tag{1}$$

where d is the distance deviation from the lane center, θ is the orientation deviation, and v is the current velocity of the robot. The reward considers not only the position but also the real-time heading direction of the robot, which can help the controller react more promptly. When the robot deviates to the left of the lane but it is heading right, back to the center, the deviation rewards should be accordingly alleviated. In the other case, the rewards and the turning tendency should be further strengthened if the robot is heading left when it has already deviated to the left. γ_d and γ_θ are the constants to adjust the importance of the two kinds of deviations.

In the training process, the STDP rule updates the synapse

(a) Car agent in the scenario

(b) Scenario roadmap

Fig. 3. Experiment scenarios on CARLA

weights based on the time difference between the presynapse spike and the postsynapse spike. The update is given as:

$$\tau = t_{post} - t_{pre},$$
$$STDP(\tau) = \begin{cases} -A_- \, e^{\tau/\tau_-}, & \tau < 0, \\ A_+ \, e^{-\tau/\tau_+}, & \tau \geq 0, \end{cases} \tag{2}$$

where τ is the time difference between two spikes, A_- and A_+ are the constant multipliers applied to the weight update, and τ_- and τ_+ are the STDP time constant for weight changes [18]. To take the rewards from the environment into account, R-STDP first computes the eligibility trace c for the synapse as follows:

$$\dot{c} = -c/\tau_c + STDP(\tau) \cdot \delta(t - t_{pre/post}) \tag{3}$$

in which $t_{pre/post}$ is the time of the spike pairing, $\delta(t - t_{pre/post})$ is the Dirac delta function that filters $STDP(\tau)$ and step-increases the eligibility trace c [16]. Then the synapse connection weights w are updated based on the eligibility trace c and the rewards r:

$$\dot{w} = c \cdot r \tag{4}$$

2) Output Decoding

To drive the robot with the output from the SNN controller, the output spikes from the actuator neurons must be decoded into continuous motor speed values. In the decoding process, the output spikes for an actuator neuron are measured, and the spike rate n_t is linearly scaled to the output velocity v_t:

$$v_t = \gamma_v \cdot n_t / n_{max} \tag{5}$$

where n_{max} is the maximum possible spike rate and γ_v is the constant multiplier to adjust the maximum output speed. Subsequently, the output velocities are utilized to actuate the differential wheeled robot agent in the environment. To drive a normal four-wheel vehicle with the Ackermann steering model that accepts a translational velocity and a steering angle, a conversion from the differential velocities to the linear velocity v_{linear} and steering angle α is further performed as follows:

$$\begin{aligned} v_{linear} &= \frac{v_l + v_r}{2}, \\ R &= \frac{L}{2} \cdot \frac{v_l + v_r}{v_r - v_l}, \\ \alpha &= arctan\,\frac{W}{R} \end{aligned} \tag{6}$$

where v_l and v_r are the velocities of the left and right actuators, R is the turning radius, and L and W are the track width and wheelbase of the vehicle. With the conversion, the SNN controller is able to navigate a car agent in a high-fidelity simulator environment.

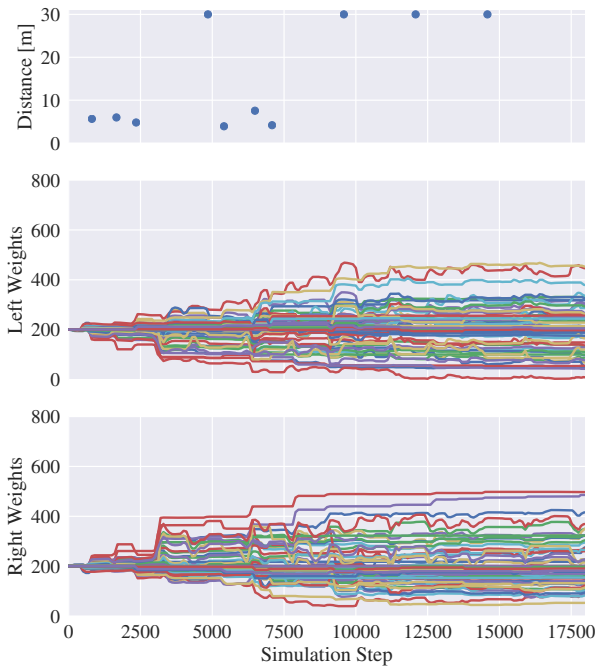


Fig. 4. Scenario 1 on CoppeliaSim. Training process with 8×8 sensory neurons and 2 actuator neurons.

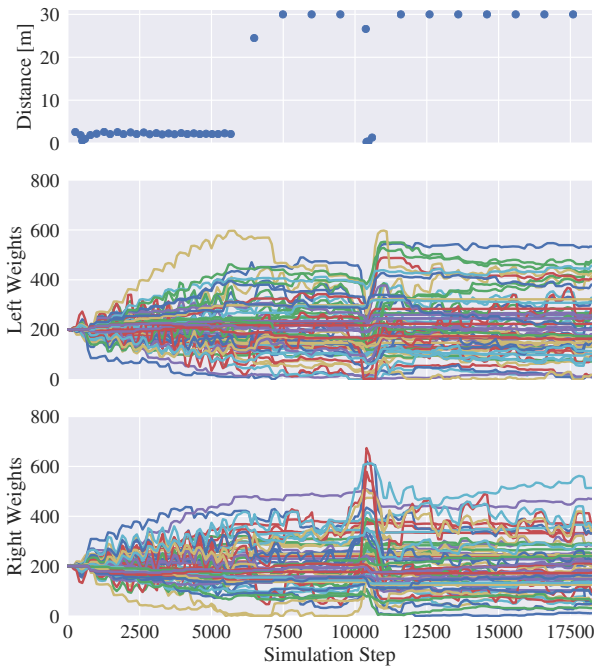


Fig. 6. Scenario 2 on CoppeliaSim. Training process with 8×8 sensory neurons and 2 actuator neurons.

Left Weights

133	296	167	236	208	209	206	205
98	136	207	234	204	263	175	197
110	111	128	286	244	266	205	200
68	53	148	120	260	242	228	191
207	170	331	43	336	293	248	244
197	235	318	55	165	325	255	214
87	305	232	100	77	379	214	201
45	446	199	187	9	452	196	199

Right Weights

323	130	265	164	182	171	191	181
322	271	175	136	193	113	194	190
294	303	323	132	153	144	200	197
349	275	259	227	186	123	131	177
219	273	107	485	102	167	169	154
201	132	130	497	214	94	150	184
349	91	159	322	227	82	186	199
415	84	184	217	368	52	200	200

Fig. 5. Scenario 1 on CoppeliaSim. Learned synapse weights for 8×8 sensory neurons and 2 actuator neurons.

Left Weights

130	11	217	470	199	114	477	203
202	102	106	153	171	156	381	162
270	72	152	77	430	182	457	173
143	8	93	132	225	283	425	205
164	14	251	259	102	265	257	308
131	92	317	284	146	151	402	321
163	170	328	266	170	88	434	245
165	17	414	245	166	19	533	223

Right Weights

268	469	136	105	185	285	57	216
207	376	198	211	176	157	85	210
104	290	201	513	135	137	136	240
174	330	363	189	61	94	63	162
252	56	238	102	353	135	173	132
332	290	49	126	242	217	65	125
285	282	52	131	229	362	31	149
227	333	43	146	240	340	11	171

Fig. 7. Scenario 2 on CoppeliaSim. Learned synapse weights for 8×8 sensory neurons and 2 actuator neurons.

III. Experiments

To evaluate the effectiveness of the proposed method, a series of lane-keeping experiments in three scenarios on two different simulators were carried out. In this section, we elaborate on the experiment settings for simulators and scenarios and analyze the experimental results to demonstrate the applicability and performance of the proposed system.

A. Experimental Setup

In order to examine the performance and the capability of the proposed approach to generalize for different environments, two distinct simulators, CoppeliaSim [19] and CARLA [20], are utilized to carry out the experiments. In the CoppeliaSim simulator, the proposed method is evaluated on a differential wheeled Pioneer robot, as shown in Fig. 2a, with a 16-channel 3D LiDAR installed at the front of the robot agent. Besides, to further investigate the performance of the proposed method in the urban area, the subsequent experiments are performed in the high-fidelity simulator CARLA which is targeted for autonomous driving simulation. While a gap does exist between simulators and reality, studies show that conducting training and testing in the high-fidelity simulators provides models with good generalizability and transferability to the real world [21]. The proposed lane-keeping method is deployed on a car agent in the simulator with a 16-channel 3D LiDAR. The LiDAR sensor is mounted on the top of the vehicle to obtain the necessary point cloud data around the agent. The training for the SNN controller requires careful tuning for the parameters of the neurons and synapse connections. In the experiments, we have set the time step $50ms$ for SNN simulation, the parameters $A_- = A_+ = 1.0$ and $\tau_- = \tau_+ = 200.0$ for STDP learning, $\gamma_d = 3.0$, $\gamma_\theta = 5.0$ for the reward feedback, and the synapse weights were initialized at 200.0.

In the training process, the SNN controller is trained in a fashion to navigate the robot agent to drive within the lane area. In each training episode, the robot agent is first placed in the starting position to begin the training. During the navigation, the LiDAR point cloud data obtained from the

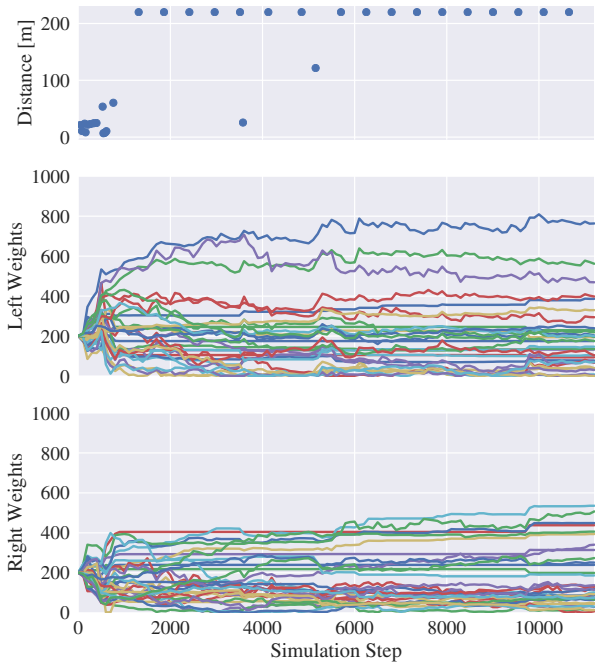


Fig. 8. Urban scenario on CARLA. Training process with 8×4 sensory neurons and 2 actuator neurons.

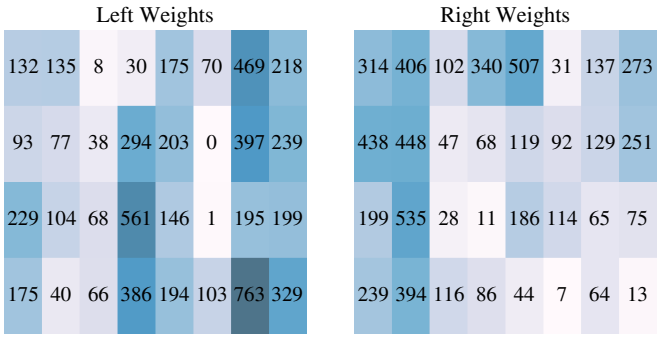


Fig. 9. Urban scenario on CARLA. Learned synapse weights for 8×4 sensory neurons and 2 actuator neurons after the training iterations.

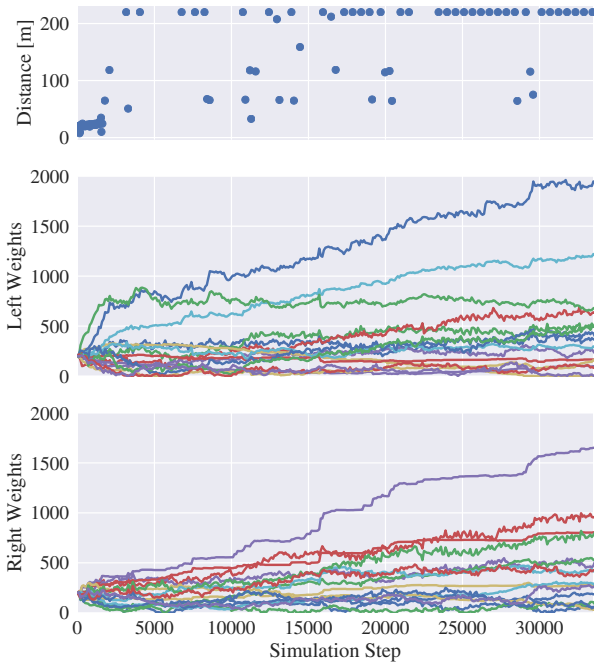


Fig. 10. Urban scenario on CARLA. Training process with 4×4 sensory neurons and 2 actuator neurons.

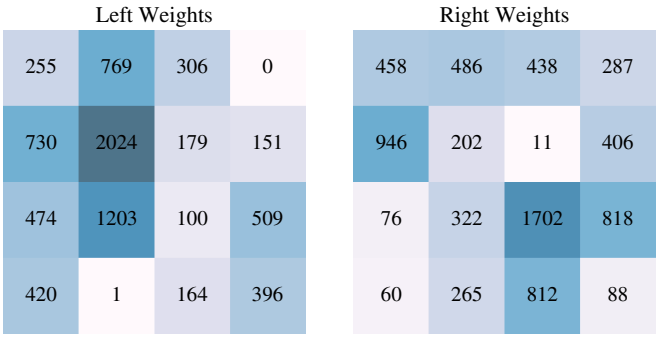


Fig. 11. Urban scenario on CARLA. Learned synapse weights for 4×4 sensory neurons and 2 actuator neurons after the training iterations.

simulator are fed into the lane feature extractor network to generate the sensory input for the SNN controller. The spikes from the SNN controller are detected and converted into the speed output, which is passed to the simulator to actuate the robot agent. In each simulation step, to perform the feedback learning for the SNN controller, the robot position relative to the lane center is computed to update the rewards for the R-STDP training. Once the robot drives away from the lane area, the episode fails, and the robot agent is reset to the starting point to start a new training episode. The experiment scenarios are designed to be complete loops allowing the robot agent to return to the starting point to conclude the training episode as a successful training iteration. The training process continues to perform the feedback learning for the controller until the robot agent stably completes loops and the synapse weights remain steady.

B. Experiments on CoppeliaSim

We first conduct experiments on CoppeliaSim. To evaluate the performance of the proposed method in different scenario settings, two scenarios with different roadmaps are set up and utilized in the CoppeliaSim experiments, as shown in Fig. 2. In the experiments, we train the SNN controller with 8×8 sensory input neurons and set the receptive field for the sensory input to $x \in [0, 5],\ y \in [-2, 2]$.

Scenario 1 contains a loop with bidirectional roads and curbs at the sides. The 3D LiDAR sensor is installed at the front of the robot to reduce the blind spot of the sensor. During the experiments, the robot agent is trained to drive within the right lane. The lane width is $0.5m$. If the robot agent deviates from the lane center by more than $0.25m$, it will be reset to one of the starting points. In the experiments, two starting points are chosen alternately to train the robot for both lanes. Fig. 4 shows the training process of approximately $17,500$ simulation steps for scenario 1. In the figure, the distance traveled for each training iteration is shown on the top, which can indicate the training progress over the simulation steps. During the beginning of $4,000$ simulation steps, the robot agent mostly resets around $5m$. Due to the fact that the synapse weights are evenly initialized with a constant value, the robot agent is not able to make the first

TABLE I

PERFORMANCE COMPARISON WITH DQN

		Training steps	Deviation		Neurons	Inference	
			Mean	RMSE		Time	Energy
DQN	Copp. 1	80,400	0.022m	0.027m	715	67ms	CPU: 166J GPU: 5.79J
	Copp. 2	97,500	0.071m	0.088m			
	CARLA	136,500	0.438m	0.665m	1027		
SNN	Copp. 1	10,000	0.025m	0.036m	66	50ms	Loihi: 1.87×10^{-3}J
	Copp. 2	12,500	0.042m	0.052m			
	CARLA 8x4	11,200	0.148m	0.180m	34		
	CARLA 4x4	33,500	0.166m	0.202m	18		

turn at $5m$ in the beginning. After $8,000$ simulation steps of learning, the robot starts to finish the complete loop for both lanes. The weight curves show the learning process for the synapses over the simulation steps. The weights are updated gradually and remain stable after $10,000$ steps. Fig. 5 shows the learned weights after approximately $17,500$ steps.

In order to investigate the capability of the proposed method to adapt to different environments, the experiments are also carried out in scenario 2. As shown in Fig. 2c, scenario 2 is a one-way loop in the shape of figure eight, with an intersection in the center. The training for scenario 2 is performed with the same parameter settings as scenario 1. Fig. 6 illustrates the learning process of approximately $17,500$ steps. In the beginning, resetting happens repeatedly at around $2m$, and the synapse weights are gradually updated over the simulation steps. At around step $6,000$, the SNN controller starts to turn the robot agent properly and is able to navigate the robot agent in the scenario stably after $12,500$ simulation steps. Fig. 7 shows the learned synapse weights after the approximately $17,500$ simulation steps. Interestingly, the robot agent can freely pass the intersection in the center. One explanation is that the open area with no boundary in the intersection will not trigger the sensory spikes to further unevenly activate the output actuator neurons. Hence, the robot agent will continue to follow the prior control and go straight ahead to pass the intersection.

For the purpose of comparative evaluation for the SNN controller, we additionally conduct aforementioned experiments based on a deep Q-learning network (DQN). The implemented DQN is composed of three fully-connected dense layers, a 32x16 input layer, a hidden layer with 200 neurons, an output layer with 3 neurons, and 103,000 connections between the layers. The DQN controller is trained and tested with the same scenario settings and sensory input. As shown in TABLE I, the DQN controller requires eight times more training steps than the SNN controller to achieve comparable results in scenario 1 and scenario 2. With fewer neurons and connections, SNN outperforms DQN in scenario 2 in terms of deviation errors, demonstrating the superior generalization and adaptability of the proposed method to different scenarios compared to the DQN controller.

The experiments on CoppeliaSim demonstrate that the proposed SNN controller is capable of learning the control policy from the sensory input to perform lane-keeping, and presents high efficiency and generalizability in two scenarios.

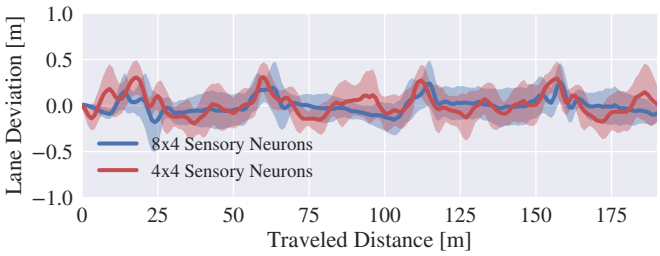


Fig. 12. Urban scenario on CARLA. Mean and standard deviation of lane keeping errors in testing. The errors remain low and slightly fluctuate when the vehicle turns at right-angle bends.

C. Experiments on CARLA

The experiments conducted on CARLA aim to analyze and demonstrate the applicability of the proposed method in the urban environment. In the experiments, the SNN controller is trained to navigate a vehicle within the lane area in the specified roadmap, as shown in Fig. 3, where the starting point is marked in red. The γ_v parameter for the velocity control is set to 3.0. The travel distance to complete a successful training iteration in the scenario is $220m$. The lane width of the scenario is $4m$, which is taken into account to reset a deviating car agent. The training iteration is reset if the agent is $2m$ away from the lane center or a collision is detected. We carried out experiments on CARLA with different parameter settings to investigate the performance of different numbers of sensory neurons.

We first evaluate the SNN controller with 8×4 sensory input neurons. In this setting, the receptive field for the sensory input is set to $x \in [0, 10],\ y \in [-5, 5]$. The SNN controller has been trained for $11,200$ iterations for the experiment setting. Fig. 8 illustrates the training process over the $11,200$ simulation steps. As shown in the graph, in the beginning, the training extensively resets around $20m$, along with the dramatic changes in the synapse weights. After 500 simulation steps, the reset distance increases to around $70m$. The SNN controller can successfully finish a complete navigation iteration in the scenario after around $1,000$ training steps. In the following iterations, the synapse weights are reinforced or adjusted based on the failures at simulation steps $3,700$ and $5,200$ approximately. Fig. 9 shows the synapse weights of the SNN controller after the $11,200$ training steps. Fig. 12 shows the deviations from lane center over distance traveled in each loop in the testing runs. The mean error and root-mean-square error (RMSE) of lane deviations in testing are $0.148m$ and $0.180m$, respectively, for 8×4 sensory neurons, only around 4% of the lane width.

In an attempt to reduce the involved neurons in the SNN controller, we conducted experiments with 4×4 sensory neurons in the same scenario on CARLA. Due to fewer sensory neurons, the receptive field for the sensory input is adjusted to $x \in [0, 7],\ y \in [-3, 3]$. As illustrated in Fig. 10, the SNN controller requires more training steps to achieve a stable state in comparison with the case of 8×4 sensory neurons. The SNN controller starts to successfully finish a complete navigation iteration after around $3,000$ training steps. During the simulation steps, approximately from $8,000$

to $30,000$, the synapse weights are adjusted based on the failures. The weights remain stable after $30,000$ training steps. Fig. 11 shows the learned weights after the $33,500$ simulation steps. As shown in Fig. 12, the deviations from lane center are slightly higher than that of 8×4 sensory neurons when the vehicle turns at right-angle bends. The mean error and RMSE of lane deviations in testing are $0.166m$ and $0.202m$, respectively. Due to the decline of the receptive field for the sensory input, compared with 8×4 sensory neurons, the reinforced connections are different, which indicate SNN learned the weights from exploiting different associations from lane perception to turning actions.

Comparative experiments were also carried out on the high-fidelity urban scenario for LiDAR-based DQN as well as the DVS-based SNN method [9]. However, the DVS-based controller failed to complete proper lane-keeping due to the high scene complexity. We only present the comparative results of LiDAR-DQN, which requires an enlarged hidden layer with 512 neurons to achieve proper navigation. As illustrated in TABLE I, with 8×4 and 4×4 input, the training steps required by the proposed SNN controller are only 8.2% and 24.5%, respectively, compared with the DQN controller. In spite of fewer involved neurons, the SNN controller achieves superior lane-keeping accuracy, showing only 33.8% and 37.9% of mean deviation error by the DQN controller, for 8×4 and 4×4 input, respectively. The experiments illustrate superiority of the SNN controller in terms of training efficiency and lane-keeping accuracy with limited control neurons and synapses, and demonstrate the high applicability in the urban scenario.

In addition, in order to further investigate the energy efficiency of the proposed spike-based system, we estimate the approximate inference energy consumption with the Nengo simulator [22] based on the number of the involved operations of the modules. Listed in the last column in TABLE I, we evaluate the energy consumption per inference for the DQN-based solution on CPU and GPU as well as the proposed SNN-based system on the Loihi neuromorphic processor. The results exhibit the tremendous advantage and significant potential of the spike-based lane-keeping system in regard to energy efficiency on neuromorphic hardware.

IV. Conclusion

In this work, we propose an end-to-end approach to address the lane-keeping problem using a 3D LiDAR sensor based on a spiking neural network. The proposed SNN controller perceives the LiDAR input using the convolutional lane feature extractor and exploits the feature input to perform feedback learning from the environment to solve the lane-keeping problem. The experimental environments and testing scenarios are built and set up on two high-fidelity simulators. The experimental results on CoppeliaSim and CARLA demonstrate the high generalizability and performance of the proposed method in different environments and scenarios.

References

[1] Y. Li, L. Ma, Z. Zhong, F. Liu, M. A. Chapman, D. Cao, and J. Li, "Deep learning for lidar point clouds in autonomous driving: A review," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 8, pp. 3412–3432, 2020.

[2] X. Yao, Y. Shan, J. Li, D. Ma, and K. Huang, "Lidar based navigable region detection for unmanned surface vehicles," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 3754–3759.

[3] M. Lechner, R. Hasani, A. Amini, T. A. Henzinger, D. Rus, and R. Grosu, "Neural circuit policies enabling auditable autonomy," *Nature Machine Intelligence*, vol. 2, no. 10, pp. 642–652, 2020.

[4] M. Bouvier, A. Valentian, T. Mesquida, F. Rummens, M. Reyboz, E. Vianello, and E. Beigne, "Spiking neural networks hardware implementations and challenges: A survey," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 15.

[5] J. P. Mitchell, G. Bruer, M. E. Dean, J. S. Plank, G. S. Rose, and C. D. Schuman, "Neon: Neuromorphic control for autonomous robotic navigation," in *2017 IEEE International Symposium on Robotics and Intelligent Sensors (IRIS)*. IEEE, 2017, pp. 136–142.

[6] Z. Bing, A. E. Sewisy, G. Zhuang, F. Walter, F. O. Morin, K. Huang, and A. Knoll, "Toward cognitive navigation: Design and implementation of a biologically inspired head direction cell network," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 5, pp. 2147–2158, 2021.

[7] Z. Bing, D. Nitschke, G. Zhuang, K. Huang, and A. Knoll, "Towards cognitive navigation: A biologically inspired calibration mechanism for the head direction cell network," *Journal of Automation and Intelligence*, vol. 2, no. 1, pp. 31–41, 2023.

[8] A. Shalumov, R. Halaly, and E. E. Tsur, "Lidar-driven spiking neural network for collision avoidance in autonomous driving," *Bioinspiration & Biomimetics*, vol. 16, no. 6, p. 066016, 2021.

[9] 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. 4725–4732.

[10] 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 Networks*, vol. 121, pp. 21–36, 2020.

[11] Z. Bing, C. Meschede, F. Röhrbein, K. Huang, and A. C. Knoll, "A survey of robotics control based on learning-inspired spiking neural networks," *Frontiers in neurorobotics*, vol. 12, p. 35, 2018.

[12] W. Cheng, H. Luo, W. Yang, L. Yu, S. Chen, and W. Li, "Det: A high-resolution dvs dataset for lane extraction," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2019, pp. 0–0.

[13] L. Caltagirone, S. Scheidegger, L. Svensson, and M. Wahde, "Fast lidar-based road detection using fully convolutional neural networks," in *2017 ieee intelligent vehicles symposium (iv)*.

[14] P. U. Diehl and M. Cook, "Unsupervised learning of digit recognition using spike-timing-dependent plasticity," *Frontiers in computational neuroscience*, vol. 9, p. 99, 2015.

[15] H. Markram, W. Gerstner, and P. J. Sjöström, "Spike-timing-dependent plasticity: a comprehensive overview," *Frontiers in synaptic neuroscience*, vol. 4, p. 2, 2012.

[16] E. M. Izhikevich, "Solving the distal reward problem through linkage of stdp and dopamine signaling," *Cerebral cortex*, vol. 17, no. 10, pp. 2443–2452, 2007.

[17] R. V. Florian, "Reinforcement learning through modulation of spike-timing-dependent synaptic plasticity," *Neural computation*, vol. 19, no. 6, pp. 1468–1502, 2007.

[18] W. Potjans, A. Morrison, and M. Diesmann, "Enabling functional neural circuit simulations with distributed computing of neuromodulated plasticity," *Frontiers in computational neuroscience*, vol. 4.

[19] E. Rohmer, S. P. Singh, and M. Freese, "V-rep: A versatile and scalable robot simulation framework," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*.

[20] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.

[21] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. Al Sallab, S. Yogamani, and P. Pérez, "Deep reinforcement learning for autonomous driving: A survey," *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[22] T. Bekolay, J. Bergstra, E. Hunsberger, T. DeWolf, T. Stewart, D. Rasmussen, X. Choo, A. Voelker, and C. Eliasmith, "Nengo: a Python tool for building large-scale functional brain models," *Frontiers in Neuroinformatics*, vol. 7, no. 48, pp. 1–13, 2014.