2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 25-29, 2020, Las Vegas, NV, USA (Virtual)

Event-based PID controller fully realized in neuromorphic hardware: a one DoF study.

Rasmus Karnøe Stagsted[1], Antonio Vitale[2], Alpha Renner[3], Leon Bonde Larsen[1], Anders Lyhne Christensen[1], and Yulia Sandamirskaya[4]

***Abstract*— Spiking Neuronal Networks (SNNs) realized in neuromorphic hardware lead to low-power and low-latency neuronal computing architectures. Neuromorphic computing systems are most efficient when all of perception, decision making, and motor control are seamlessly integrated into a single neuronal architecture that can be realized on the neuromorphic hardware. Many neuronal network architectures address the perception tasks, while work on neuronal motor controllers is scarce. Here, we present an improved implementation of a neuromorphic PID controller. The controller was realized on Intel's neuromorphic research chip Loihi and its performance tested on a drone, constrained to rotate on a single axis. The SNN controller is built using neuronal populations, in which a single spike carries information about sensed and control signals. Neuronal arrays perform computation on such sparse representations to calculate the proportional, derivative, and integral terms. The SNN PID controller is compared to a PID controller, implemented in software, and achieves a comparable performance, paving the way to a fully neuromorphic system in which perception, planning, and control are realized in an on-chip SNN.**

I. INTRODUCTION

Biological systems have always inspired robotics research. Animals, from insects to humans, are amazing in their ability to move safely in complex and dynamic environments, even if the environments are not known in advance and are perceived with limited local sensing. In many cases, movement is generated with non-ideal, e.g. soft, redundant, or underactuated, motor systems. Obviously, biological neuronal networks are able to solve the complex perception, planning, and control problems with their limited energy resources, even in simple animals. Thus, many robotic systems were developed that use neuronal circuits for control of, e.g., insect-like robots [1], lizards [2], or vehicles [3], [4].

In parallel to the development of brain-inspired algorithms for robot control, in the field of neuromorphic engineering, a new type of computing hardware was proposed, in which biologically realistic neuronal networks are realized directly in hardware [5]–[8]. Neuromorphic hardware computes the dynamics of spiking neuronal networks in real-time at a low power consumption due to efficient event-based asynchronous architecture and co-location of memory and computing units in neurons and synapses. A distinctive property of neuromorphic hardware is on-chip plasticity that enables continual learning in real-time [8], [9]. These properties make neuromorphic hardware an attractive platform for neurorobotics. Neuromorphic platforms were used in proof-of-concept experiments for, e.g., robot navigation, path planning, and SLAM [10]–[15]. More recently, several architectures for spike-based motor control were suggested, and some of them realized in neuromorphic hardware, showing promising results. This includes bio-inspired neuronal circuits to control a single effector [16]–[19] or approximations of closed-form controllers of multi-DoF effectors with spiking neurons [20]–[23].

In our previous work, we have proposed a spiking neural network to realize a well-understood proportional, integral, derivative (PID) controller in neuromorphic hardware. The first version of this controller only realized the P-term in a mixed-signal ultra low power prototype neuromorphic device [24]. A more recent realization extended this controller to a full PID controller running on Intel's neuromorphic research chip Loihi [8] and was used to control a UAV constrained to one axis, as in this work, and to control a robotic vehicle [25]. In that work, we realized a nested cascade of two controllers: one for the pose and the other one for the speed of the robots. In this work, we further improve the PID controller on Loihi. First, we re-designed the integral path of the controller to cope with a limited resolution of value representation, which led to fast saturation of the I-path. Second, we simplified the network, removing the inner control loop. Finally, we improved the I/O interface, decreasing the time step duration of the control loop. These changes allowed us to increase the control frequency and improve performance. We compare the performance of our spiking PID controller, implemented in neuromorphic hardware, to an equivalent conventional controller implemented on a CPU. We demonstrate that our controller achieves a comparable performance to the conventional controller that is realized with full floating-point arithmetics. Additionally, we perform power measurements of the SNN controller on Loihi and estimate the power consumption that can be achieved in future, more tightly integrated implementations. In our closed-loop hardware setup, the neuromorphic chip is connected to the drone's flight controller and to the IMU sensor using a computing board UP^2, which can be omitted in a more embedded setup.

The on-chip plasticity on the Loihi can be used to tune the controller gains automatically. We realized the control gains as synapses in our architecture, which, in future work,

[1] RS, LL and AC are with SDU rk@mmmi.sdu.dk
[2] AV are a student at Department of Mechanical Engineering of ETH Zurich, Switzerland avitale@ethz.ch
[3] AR is with the Institute of Neuroinformatics, University of Zurich and ETH Zurich, Switzerland alren@ini.uzh.ch
[4] YS is with Neuromorphic Computing Lab, Intel Labs, Germany yulia.sandamirskaya@intel.com

can be made plastic to allow them to adapt according to local, reinforcement-driven on-chip learning rules [8] and thus implement an adaptive PID controller on-chip.

II. METHODS

A. Hardware

1) Neuromorphic hardware (Loihi): For the simulation of the Spiking Neural Network (SNN), Intel's neuromorphic research chip Loihi, in the form of a Kapoho Bay device, interfaced with a host computer over USB, was used. The Kapoho Bay features two Loihi chips, each chip simulating up to 128K neurons in its 128 cores. Furthermore, three x86 co-processors are integrated on the Loihi chip, used for monitoring, input/output, and configuration.

2) Constrained UAV: For testing the controller, we used the hardware setup developed in [25]: a 1-DoF constrained drone, shown in Fig. 1. An Inertial Measurement Unit (IMU) inside a DAVIS240C [26] mounted on the drone was used to measure the angle and angular acceleration while torque was produced with propellers on the motors of the drone.

The two motors were each controlled by an Electronic Speed Controller (ESC), driven by RC PWM. To generate the RC PWM signals, a Navio flight controller connected to a Raspberry Pi was used. This enabled high-level, but fast control of the motors from Linux.

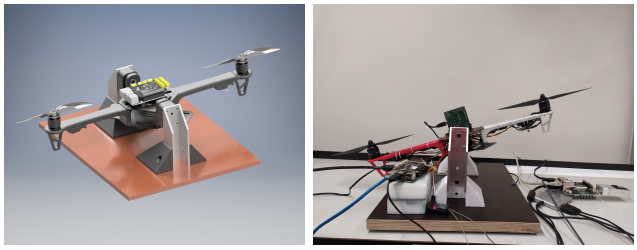

Fig. 1: **Left**: A 3D model of the constrained to 1DoF drone. **Right**: The physical setup.

3) Interfaces, system overview: An Intel UP^2 single-board computer[1] was used to interface the Kapoho Bay. For communication between the Raspberry Pi and the Intel UP^2, a light-weight peer-to-peer communication library *YARP* [27], which is based on the TCP/IP protocol, was used. This enabled communication between the Intel UP^2, used as Kapoho Bay host, and the Raspberry Pi, responsible for the motor interface.

The software architecture consists of four programs communicating through YARP (see Fig. 2). The *IMU node* extracts the angle measurements from the IMU in the DAVIS sensor, the *ESC node* writes motor commands to the ESC, the *Control Setpoint node* generates a predefined sequence of setpoints for the *Kapoho Bay node* which sends the received setpoint- and IMU-values to the neuromorphic chip and receives the motor commands that are converted and sent to the *ESC node*.

[1]Intel Pentium N4200: 1.1/2.5GHz w/4GB LPDDR4 RAM, running Ubuntu 16.04 and NxSDK 0.9.

This architecture enables other software PIDs that are communicating using YARP to be swapped with the *Kapoho Bay interface* for comparison against other controllers.

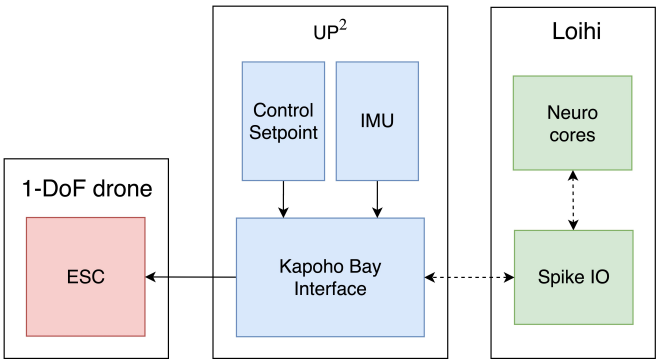

Fig. 2: Overview of the overall controller setup. The *IMU-* and *Control Setpoint* YARP nodes feed values into the *Kapoho Bay Interface* node. From here, the values are passed into the neuronal cores on Kapoho Bay, which responds with the output of the SNN PID. Finally, the output of the PID is sent to the *ESC* node and to the motors. The solid arrows are YARP connections, the dashed arrows are hardware- and software interfaces provided by Intel.

B. The Spiking Neural Network (SNN) PID

1) SNN Input encoding: The measured angular velocity, the measured angle, and the desired angle of the system, all have to be input into the spiking neural network. In this project, we used populations of neurons that represent inputs by "place code": neurons in each input population form a 1D array, the index of an active (spiking) neuron in the array at each time step encodes the input value. We used a one-hot encoding here, meaning that at most one neuron is active per time step in any population.

Each input population receives exactly one spike per time step from spike-generators on Loihi. Which neuron receives the spike is determined by the input value (from IMU or the setpoint), based on the formula in Eq. (1):

$$\mathbf{idx} = \left\lfloor \frac{x - x_{min}}{x_{max} - x_{min}} \cdot n \right\rfloor. \quad (1)$$

TABLE I: Input ranges

Input	x_{min}	x_{max}
Angular velocity	$-80deg/s$	$80deg/s$
Angle	$-40deg$	$40deg$
Setpoint	$-25deg$	$25deg$

In Eq. (1), $\mathbf{idx}$ is the calculated neuron index, x is the input value, x_{min} and x_{max} are the minimum- and maximum possible input values, and n is the number of neurons in the 1D population. Eq. 1 linearly maps an input value to a specific neuron in the neuron population. All input populations in our network have 63 neurons. The ranges of input values are listed in Table I.

The encoded by the spikes input values are passed through a series of 2D arrays to calculate the error value and the derivative value, as proposed in [25]. Each 2D array detects coincident spikes from the two 1D input arrays and sends a single spike to the 1D output array according to the desired operation defined by the connectivity between the 2D array and the output.

2) P-, I-, and D controller: The PID controller is defined by Eqs. (2):

$$u(t) = K_p e(t) + K_i \int e(t)dt + K_d \frac{de(t)}{dt}, \qquad e(t) = r(t) - y(t), \tag{2}$$

where $u(t)$ is the calculated command, $y(t)$ is the feedback signal from the IMU, $r(t)$ is the setpoint signal (target value for $y(t)$), $e(t)$ is the error, and K_p, K_i, and K_d are the P, I, and D gains, respectively.

On Loihi, the neural computation and spike exchange are synchronized by global time steps. Thus, each connection in the SNN introduces a one time step delay. The calculated variables e_t and d_t are thus defined as differences of values at specific time steps. The architecture of the neural PID controller is schematically shown in Fig. 3.

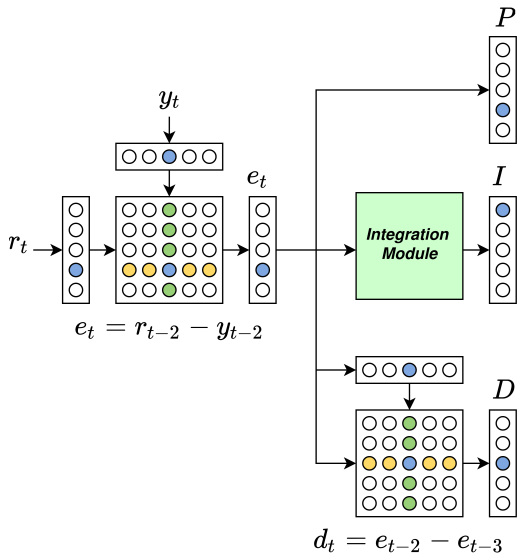

Fig. 3: The schematics of the SNN PID. Two arithmetic operations are performed in 2D operation arrays: 1) the sensor feedback (y) is subtracted from the setpoint (r) to obtain the error value (e); and 2) the error on one time step is subtracted from the error on the previous time step to obtain the derivative term. A new integration module computes the integral of the error.

3) The Integration Module: The 2D arrays that perform arithmetics in this work are able to perform any mathematical operations on two input neuron populations, such as addition and subtraction. In our previous work, we used an addition array to realize error integration. However, due to the low resolution of value representation with the one-hot encoded populations (63 neurons), when we reduced the time step duration in the optimized architecture, the integration increments got too large. The summed signal saturated quickly both in positive and negative directions, leading to oscillations.

Thus, in this work, the error signal integration was based on a path integration network developed for neuromorphic SLAM in [15], [28]. Here, two different levels of integration are introduced: (1) on the level of an individual spiking neuron and (2) on the level of a population of neurons. Mathematically, spiking neurons behave like integrators in the way that their state variable – the membrane potential – is obtained by integrating the incoming current over time. One can exploit this behavior to accumulate values over time, and increment the integral representation (change the position of an active neuron in the I population) only when the integration-neuron's threshold is reached. A schematic of this integration network is shown in Fig. 4.

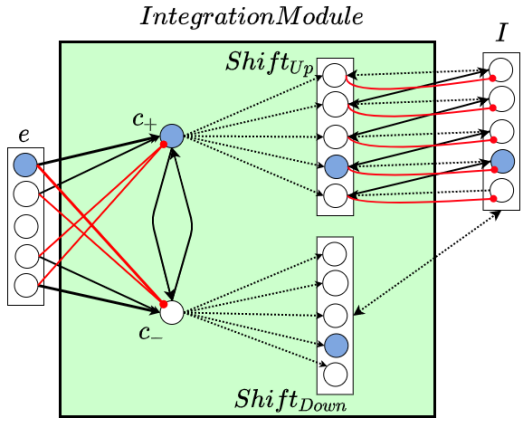

Fig. 4: Schematic of the network used to integrate a signal represented by a one-hot encoded spiking neuron population. Black are excitatory, red inhibitory connections. For clarity, only the connections from the $Shift_{Up}$ population to the I population are shown: the $Shift_{Down}$ is connected to I in an analogous way. c_+ and c_- are counter neurons that integrate the positive and negative errors, respectively, before integral increments are encoded in the I population.

Four auxiliary neuron populations are required to integrate the error signal (for integrating positive and negative error values): two neurons are responsible for accumulating the error signal over time in counter neurons (c_+ and c_-), and two populations shift the active neuron in the integral populations (the $Shift_{Down}$ and $Shift_{Up}$ layers. In the final, I, population, values are represented using place code, where greater neuron indices represent positive values while smaller neuron indices represent negative values. The two counter neurons and two shift populations are required to integrate both positive and negative values of the error signal. In Fig. 4, c_+ and the $Shift_{Up}$ populations account for positive errors while c_- and the $Shift_{Down}$ for negative errors.

The top half of the neurons in the error population represent positive values and are connected to the positive counter

population with synaptic weights according to their neuron index. In this way, larger errors cause a larger increase in the membrane potential of the counter neuron. Furthermore, the counter neurons have a high threshold and no leak, which allows them to perfectly integrate the incoming error signal over time. If the threshold potential is reached, the counter population activates all neurons in the shift layer, but not sufficiently to make them fire. For neurons in the shift layer to fire, both the counter population and the corresponding neuron in the I population have to fire at the previous time step. If this occurs, the active neuron in the shift layer activates the neuron in the I population either above ($Shift_{Up}$) or below ($Shift_{Down}$) the current firing neuron, simultaneously inhibiting the currently active neuron. The shifting position of the active neuron in the I layer represents the integral of the error signal.

The counter neurons are implemented as soft-reset neurons, where a spike in either neuron leads to subtraction of its threshold potential from the membrane potential of both counter neurons. In this way, the two neurons maintain a mirrored potential at all times and thus do not fire at the same time.

4) Output network: In order to combine the three outputs of the PID SNN signals, the P, I, and D signals are first split into two different populations representing positive and negative values, respectively. Then, the current P, I, and D values are scaled with their respective controller output and summed together. This combined value is stored in an auxiliary neuron population, which is denoted as population B. The activity is then passed on to the output population U, and the index of the firing neurons is read out and passed on to the host computer, where it is converted to motor commands.

All sparse coded populations, including the arrays representing P_t, I_t and D_t, represent values in the range $a \in [-lim, lim]$. In our architecture, the sparse coded values *P/I/D* are split into two populations: one for negative values and one for positive values. Each neuron in the *P/I/D* population is connected a certain number of neurons in either *P/I/D+* or *P/I/D-* populations. How many neurons receive input depends on the index of the neuron in the *P/I/D* population. For instance, the neuron representing +1 in P is connected to one neuron in P_+ while the neuron +2 is connected to two neurons. Each neuron in the *P/I/D+* and *P/I/D-* is then connected to the population B with respective synaptic weights of $\pm K_p/\pm K_i/\pm K_d$.

The neurons in population B have linearly increasing threshold potentials, where neuron 1 has the lowest and neuron 63 the highest threshold. Furthermore, a bias current equal to the middle neuron's threshold is injected in all neurons. In this way, all neurons up to the middle neuron, representing the value 0, are always firing, and the incoming activity from the *P/I/D+* and *P/I/D-*, respectively, activates or inhibits neurons in the B population.

To get back to one-spike sparse coding, *B* connects to a final 1D array *U*, with positive one-to-one connections. Thus, the neuron in U with the same index as the highest index of the active neurons in *B* spikes. Negative connections to all neurons of lower indices suppress spikes of those neurons (see Fig. 5).

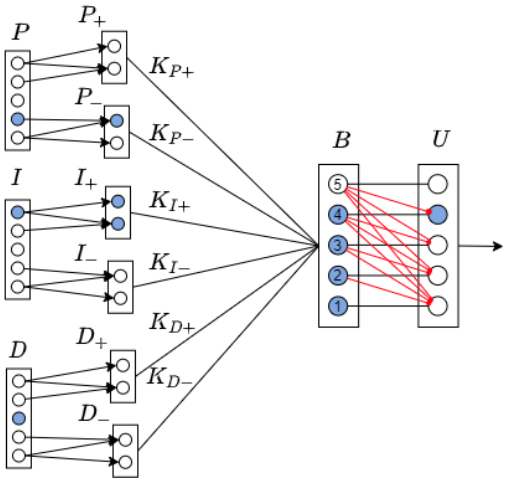

Fig. 5: Overview of the full output network: the three outputs (P, I, D) of the sparse coded network are summed in the output network using the auxiliary populations. Numbers inside the neurons of population B display the neurons' thresholds.

C. SNN Output Decoding

The embedded x86 cores on Loihi receive all output spikes from the SNN (u_t). The embedded program reads the spikes from the neural cores and linearly maps the spikes' source index to a value in the range $[-1, 1]Nm$.

D. Experiments

The SNN-based PID controller was implemented and run on Loihi, interfaced with the constrained drone platform and its performance was compared to a conventional PID controller implemented on a CPU[2] (Intel's UP[2] board). The gains (K_p, K_i, and K_d) for both PID controllers were systematically tuned by hand.

III. RESULTS

A. Timing

The sampling rate for the measured angular velocity, the measured angle, and the setpoint for the angle was approx $1kHz$. Since the whole setup is synchronized at each time step, PID provides a new result at $1kHz$. However, the PID controller delays the signal by ≈ 6 time steps due to the connections between neural populations, whereas the communication between the Kapoho Bay and the host computer delays the signal by two timesteps in each direction. This means that the control output has a latency of $\approx 10ms$ in the current setup, while the rate of the controller is $1kHz$.

[2]Intel Pentium N4200: 1.1/2.5GHz w/4GB LPDDR4 RAM running Ubuntu 16.04

B. Controller performance

For a quantitative comparison between the SNN and conventional PID controllers, Table II shows the overshoot, rise time, and settling time of the controllers. The overshoot was computed as the ratio between the maximum value and the target value (20 deg). The rise time is the time needed for the signal to rise from 10% to 90% of the steady-state value. The settling time is defined as the time needed for the signal to remain bounded within 20% of the final target value. Both a positive target amplitude change ($0° \rightarrow 20°$) and a negative target amplitudes change ($0° \rightarrow -20°$) were applied in order to assess the controller performance in both directions. Table II shows the two respective medians and the Inter Quartile Range for both the positive and negative steps over ten separate trials.[3].

The results in Table II show that the controllers operate similarly well. Fig. 6 shows the behavior of the SNN PID and software PID for the drone in several exemplary runs (one of the runs being highlighted). In this work, the controller parameters were manually tuned. Future work will aim at adapting the gains in a learning process and, in this way, finding better parametrizations for the controllers.

TABLE II: Performance of PID controllers for UAV

Controller	Overshoot (%)		Rise time (s)		Settling time (s)	
Step	*+20°*	*-20°*	*+20°*	*-20°*	*+20°*	*-20°*
SNN PID						
Median	41.4	35.9	0.46	0.41	1.4	8.8
$\pm\frac{IQR}{2}$	*±13.8*	*±18.8*	*±0.11*	*±0.11*	*±0.65*	*±3.34*
CPU PID						
Median	24.0	20.1	0.53	0.49	1.3	4.3
$\pm\frac{IQR}{2}$	*±7.3*	*±6.4*	*±0.13*	*±0.07*	*±1.64*	*±4.4*

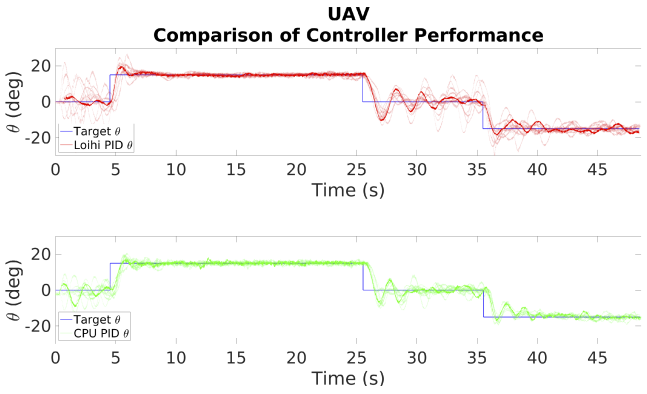

Fig. 6: Comparison between two different PID controllers controlling the 1-DoF drone to follow the setpoint (*Blue* square signal). *Red (top)*: the SNN PID running in SNN on the Loihi. *Green (bottom)*: the CPU PID running on an Intel Pentium N4200 CPU.

[3]All performance measurements are based on testing as of Jan. 2020 and may not reflect all publicly available security updates.

C. Power Measurements

Certain types of operations run faster and more efficiently on neuromorphic hardware [8]. Power measurements were done in an offline simulation of the SNN-PID network in order to quantitatively estimate how much power is consumed. Performing power measurements directly on the Kapoho Bay is currently not supported; thus the power measurements were done on Intel's Nahuku Board. Compared to the Kapoho Bay, the Nahuku Board features 32 Loihi chips with 128 neural cores each.

In a fully idle state, the Nahuku Board consumes a baseline power of $\approx 1000mW$. The SNN-PID requires only 38 neuro-cores, making it a relatively lightweight network that consumes additional power in the order of $< 10mW$. To get more robust measurements, the same model was initialized on more cores, distributed over more chips. Given that the SNN-PID model fits three times in one Loihi chip, the measurements were repeated for 1, 3, 6, and 15 models running simultaneously and, as expected, the resulting power consumption increased close to linearly to the number of models running on the board, as shown in Table III. Over these runs, an average of $25.2mW$was consumed by the network (estimated over 2000 time steps).

TABLE III: Power Consumption of the SNN PID Network

Number of Models	1	3	6	15
Power (mW)	20.3	59.5	131.4	418,7

This number is not indicative of the power consumption of the SNN controller in the control loop since it does not include the communication between the Loihi chip and the host computer, which is currently not optimized. Furthermore, we don't compare the power measurement with a CPU, since for such a simple controller, optimized microprocessors can be used, consuming similar (or even lower) power, which is hard to measure in isolation. We anticipate that due to linear growth of power dissipation with the network size, the advantage of neuromorphic hardware will become apparent for larger networks that combine the motor control with high-level perception of targets, obstacles, and error signals, and include online adaptation in the control loop.

IV. CONCLUSION

In this work, we presented an improved version of the spiking neural network realization of a PID controller. Modifications in the SNN architecture and improved interface between neuromorphic cores and the host computer allowed us to improve the latency and frequency of the controller. We realized the SNN controller in neuromorphic hardware and used it to control a physical robot – a constrained to 1DoF drone robot – in real-time. The latency of our controller (the time it takes from a sensed IMU feedback to respective corrected motor command) was approx. 10ms, of which $0.1 - 1$ms were spent in neuronal computation on the neuromorphic cores. We compared the performance of our controller to a PID realized on a conventional CPU and

achieved comparable performance for overshoot, rise- and settling times.

ACKNOWLEDGMENT

We thanks SDU-UAV for funding all drone related hardware and Cao Dahn Do for building the test bench. Intel Labs and INRC provided the Loihi chip as well as a lot of support. This project was funded by SNSF grant Ambizione PZOOP2_168183 and was initiated during the Capo Caccia 2019 Workshop "Toward Neuromorphic Intelligence".

REFERENCES

[1] P. Manoonpong, U. Parlitz, and F. Wörgötter, "Neural control and adaptive neural forward models for insect-like, energy-efficient, and adaptable locomotion of walking machines," *Frontiers in Neural Circuits*, vol. 7, no. February, pp. 1–28, 2013. [Online]. Available: http://journal.frontiersin.org/article/10.3389/fncir.2013.00012/abstract

[2] A. J. Ijspeert, "Central pattern generators for locomotion control in animals and robots: A review," *Neural Networks*, vol. 21, no. 4, pp. 642–653, 2008.

[3] P. Tandon and Y. H. Malviya, "Efficient and Robust Spiking Neural Circuit for Navigation Inspired by Echolocating Bats," no. Nips, pp. 1–9, 2016.

[4] J. L. Krichmar and H. Wagatsuma, "Neuromorphic and brain-based robots," *Frontiers in Artificial Intelligence and Applications*, vol. 233, pp. 209–214, 2011.

[5] G. Indiveri, E. Chicca, and R. J. Douglas, "Artificial Cognitive Systems: From VLSI Networks of Spiking Neurons to Neuromorphic Cognition," *Cognitive Computation*, vol. 1, no. 2, pp. 119–127, 2009. [Online]. Available: http://www.springerlink.com/index/10.1007/s12559-008-9003-6

[6] S. B. Furber, D. R. Lester, L. A. Plana, J. D. Garside, E. Painkras, S. Temple, and A. D. Brown, "Overview of the SpiNNaker System Architecture," *IEEE Transactions on Computers*, vol. 62, no. 12, pp. 2454–2467, 2012.

[7] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, and A. S. e. a. Cassidy, "Artificial brains. A million spiking-neuron integrated circuit with a scalable communication network and interface." *Science (New York, N.Y.)*, vol. 345, no. 6197, pp. 668–73, 2014. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/25104385

[8] 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.

[9] N. Qiao, H. Mostafa, F. Corradi, M. Osswald, D. Sumislawska, G. Indiveri, and G. Indiveri, "A Re-configurable On-line Learning Spiking Neuromorphic Processor comprising 256 neurons and 128K synapses," *Frontiers in neuroscience*, vol. 9, no. February, 2015.

[10] 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," in *Robotics: Science and Systems*, vol. 13, 2017.

[11] F. Galluppi, C. Denk, M. C. Meiner, T. C. Stewart, L. a. Plana, C. Eliasmith, S. Furber, and J. Conradt, "Event-based neural computing on an autonomous mobile platform," *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2862–2867, 2014. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6907270

[12] D. Weikersdorfer and J. Conradt, "Event-based particle filtering for robot self-localization," *2012 IEEE International Conference on Robotics and Biomimetics, ROBIO 2012 - Conference Digest*, pp. 866–870, 2012.

[13] 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," *Proceedings - 2017 IEEE 5th International Symposium on Robotics and Intelligent Sensors, IRIS 2017*, vol. 2018-January, pp. 136–142, 2018.

[14] S. Koziol, S. Brink, and J. Hasler, "A neuromorphic approach to path planning using a reconfigurable neuron array IC," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 12, pp. 2724–2737, 2014.

[15] R. Kreiser, A. Renner, V. R. Leite, B. Serhan, C. Bartolozzi, A. Glover, and Y. Sandamirskaya, "An on-chip spiking neural network for estimation of the head pose of the icub robot," *Frontiers in Neuroscience*, vol. 14, 2020.

[16] F. Perez-Peña, A. Morgado-Estevez, A. Linares-Barranco, A. Jimenez-Fernandez, F. Gomez-Rodriguez, G. Jimenez-Moreno, and J. Lopez-Coronado, "Neuro-inspired spike-based motion: From dynamic vision sensor to robot motor open-loop control through spike-VITE," *Sensors (Switzerland)*, vol. 13, no. 11, pp. 15 805–15 832, 2013.

[17] F. Perez-Pena, A. Morgado-Estevez, T. Serrano-Gotarredona, F. Gomez-Rodriguez, V. Ferrer-Garcia, A. Jimenez-Fernandez, and A. Linares-Barranco, "Spike-based VITE control with dynamic vision sensor applied to an arm robot," *Proceedings - IEEE International Symposium on Circuits and Systems*, pp. 463–466, 2014.

[18] F. Perez-Pena, J. A. Lenero-Bardallo, A. Linares-Barranco, and E. Chicca, "Towards bioinspired close-loop local motor control: A simulated approach supporting neuromorphic implementations," *Proceedings - IEEE International Symposium on Circuits and Systems*, pp. 1–4, 2017.

[19] E. Donati, F. Perez-Peña, C. Bartolozzi, G. Indiveri, and E. Chicca, "Open-Loop Neuromorphic Controller Implemented on VLSI Devices," *Proceedings of the IEEE RAS and EMBS International Conference on Biomedical Robotics and Biomechatronics*, vol. 2018-August, pp. 827–832, 2018.

[20] S. Menon, S. Fok, A. Neckar, O. Khatib, and K. Boahen, "Controlling articulated robots in task-space with spiking silicon neurons," in *Proceedings of the IEEE RAS and EMBS International Conference on Biomedical Robotics and Biomechatronics*, 2014, pp. 181–186. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6913773

[21] T. DeWolf, T. C. Stewart, J.-J. Slotine, and C. Eliasmith, "A spiking neural model of adaptive arm control," *Proceedings of the Royal Society B: Biological Sciences*, vol. 283, no. 1843, p. 20162134, 2016.

[22] J. Conradt, F. Galluppi, and T. C. Stewart, "Trainable sensorimotor mapping in a neuromorphic robot," *Robotics and Autonomous Systems*, vol. 71, pp. 60–68, 2015.

[23] T. DeWolf, P. Jaworski, and C. Eliasmith, "Nengo and low-power AI hardware for robust, embedded neurorobotics," pp. 1–11, 2020. [Online]. Available: http://arxiv.org/abs/2007.10227

[24] S. Glatz, J. Martel, R. Kreiser, N. Qiao, and Y. Sandamirskaya, "Adaptive motor control and learning in a spiking neural network realised on a mixed-signal neuromorphic processor," *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2019-May, pp. 9631–9637, 2019.

[25] R. K. Stagsted, A. Vitale, J. Binz, A. Renner, L. B. Larsen, and Y. Sandamirskaya, "Towards neuromorphic control: A spiking neural network based pid controller for uav." in *Robotics: Science and Systems*, 2020.

[26] J. A. Leñero-Bardallo, T. Serrano-Gotarredona, and B. Linares-Barranco, "A 3.6$\backslash\mu$ s latency asynchronous frame-free event-driven dynamic-vision-sensor," *IEEE Journal of Solid-State Circuits*, vol. 46, no. 6, pp. 1443–1455, 2011.

[27] G. Metta, P. Fitzpatrick, and L. Natale, "Yarp: yet another robot platform," *International Journal of Advanced Robotic Systems*, vol. 3, no. 1, p. 8, 2006.

[28] R. Kreiser, A. Renner, Y. Sandamirskaya, and P. Pienroj, "Pose estimation and map formation with spiking neural networks: towards neuromorphic slam," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 2159–2166.