Robotics: Science and Systems 2020
Corvalis, Oregon, USA, July 12-16, 2020

Towards neuromorphic control: A spiking neural network based PID controller for UAV

Rasmus Karnøe Stagsted*, Antonio Vitale †, Jonas Binz †, Alpha Renner †, Leon Bonde Larsen *, Yulia Sandamirskaya †
* Mærsk Mc-Kinney Møller Instituttet, University of Southern Denmark (SDU), Odense, Denmark.
Email: rk@mmmi.sdu.dk, lelar@mmmi.sdu.dk
† Institute of Neuroinformatics (INI), University of Zurich and ETH Zurich, Switzerland.
Email: antnio.vitale@outlook.de, jbinz@student.ethz.ch, alren@ini.uzh.ch, yulia.sandamirskaya@intel.com

Abstract—**In this work, we present a spiking neural network (SNN) based PID controller on a neuromorphic chip. On-chip SNNs are currently being explored in low-power AI applications. Due to potentially ultra-low power consumption, low latency, and high processing speed, on-chip SNNs are a promising tool for control of power-constrained platforms, such as Unmanned Aerial Vehicles (UAV). To obtain highly efficient and fast end-to-end neuromorphic controllers, the SNN-based AI architectures must be seamlessly integrated with motor control. Towards this goal, we present here the first implementation of a fully neuromorphic PID controller. We interfaced Intel's neuromorphic research chip Loihi to a UAV, constrained to a single degree of freedom. We developed an SNN control architecture using populations of spiking neurons, in which each spike carries information about the measured, control, or error value, defined by the identity of the spiking neuron. Using this sparse code, we realize a precise PID controller. The P, I, and D gains of the controller are implemented as synaptic weights that can adapt according to an on-chip plasticity rule. In future work, these plastic synapses can be used to tune and adapt the controller autonomously.**

I. Introduction

Artificial neural networks have transformed the field of computer vision and are increasingly used in robotics to support, e.g. perception [1]–[4], simultaneous localisation and mapping [5]–[8], planning [9], [10], and end-to-end control [11]–[13]. To achieve good performance, large neural networks are required that need to be deployed on special-purpose hardware to meet the latency and power-consumption constraints of autonomous robots. Neuromorphic hardware offers a particularly favorable trade-off in latency and power, due to its parallel, asynchronous, event-based (spiking) way to compute [14]–[17]. In contrast to other neural network accelerators, neuromorphic hardware features plasticity that enables online on-chip learning in real-time, after deployment of the network [14], [18].

Thus, neuromorphic hardware offers a computing substrate for neuronal robot control architectures with adaptation and a close link to advanced perception – e.g., object recognition, segmentation, or shape estimation. In order to enable the seamless integration of different components of the overall architecture, we develop algorithms and computing primitives to solve different robotic tasks using SNNs. Such a pervasively neuronal approach is required to reduce computing bottlenecks on the interface between neuromorphic and conventional, sequential computing hardware. Conventional computing can be reduced to the bare minimum to support configuration, monitoring, and documentation during system development and can become obsolete at deployment.

The SNNs that we develop are complimentary to the state of the art deep neural networks (DNNs) and alleviate a number of challenges of their use in robotics: (1) training a DNN relies on a large amount of data that captures the task; since the characteristics of a specific robot do not translate perfectly well to another robot and are subject to fluctuations and drift, online learning and adaptation are required to enable precise neuronal control; (2) Robot control has stricter constraints on latency, power budget, and size of computing hardware than typical image processing applications, thus simple networks are preferable; (3) The relatively low (compared to high-resolution images) dimensionality of input and output space also lead to preference for smaller NN architectures in motor control applications. Despite the expanding research in applications of deep learning in motor control, most motor controllers deployed today use computationally much lighter conventional algorithms. Simple PID controllers are the most basic and widely used controllers for many control tasks [19].

In this work, we demonstrate how a nested PID controller can be realized in a spiking neuronal network (SNN) on-chip, achieving comparable performance to a conventional digital controller realized with floating-point arithmetic. Recent work on motor control in an SNN on-chip [20] implemented a neural network that learns to generate a corrective signal if the controller starts to fail, e.g., due to wear. In our work, the controller itself is realized as an SNN, making the whole system neuromorphic and subject to optimal performance and power consumption trade-off on neuromorphic hardware.

We implement our controller on Intel's neuromorphic research chip Loihi [14]. We demonstrate the controller performance on two robotic systems – a bi-motor UAV and a wheeled vehicle – in a closed-loop hardware setup with the neuromorphic chip and robots. The on-chip plasticity creates a possibility for autonomous tuning and adaptation of the controller [21]–[23].

II. Methods

A. Hardware & software components

1) Neuromorphic hardware (Loihi): In this project, Intel's neuromorphic research chip Loihi [14] realized in the integrated device Kapoho Bay, was used to implement the SNN. It contains two Loihi chips, each integrating 128 neuron-cores, and three x86 cores. The x86 cores are used for monitoring and spike I/O. Each neuron-core simulates 1024 neuronal compartments and 4096 input and output "axons" (ports). We use 27ms time-steps in our experiments to synchronize neuronal processing with the overall handling of I/O between different devices (timestep on Loihi can be as fast as $10\mu s$). The Kapoho Bay has a USB interface, which gives easy access to the system via the NxSDK API provided by Intel.

2) Pushbot: The *Pushbot* [24] is a small differential drive robot with two propulsion belts (Fig. 1). The Pushbot features: an IMU to measure the angular velocity, one DC motor for each belt, and a WiFi module to communicate with a host computer, interfaced to the Kapoho Bay.

3) Constrained UAV: The UAV setup consists of a two-motor UAV mounted on a test bench, constraining the UAV to one degree of freedom. The torque around the horizontal axis is generated by two BLDC motors, mounted 23cm away from the rotational axis, with propellers facing upwards (Fig. 1). A mapping between motor velocity and torque was derived from the black box method, which enables direct torque control of the axis. We mounted a DAVIS240C sensor [25] on the UAV and used its integrated Inertial Measurement Unit (IMU) to obtain estimates of the UAV's pose and angular velocity. The event-based camera itself was not used here, but can be deployed for optic-flow based state estimation [26]. The IMU and Electronic Speed Controllers (ESCs) were interfaced using a Raspberri Pi communicating with the Kapoho Bay's host computer over the network.

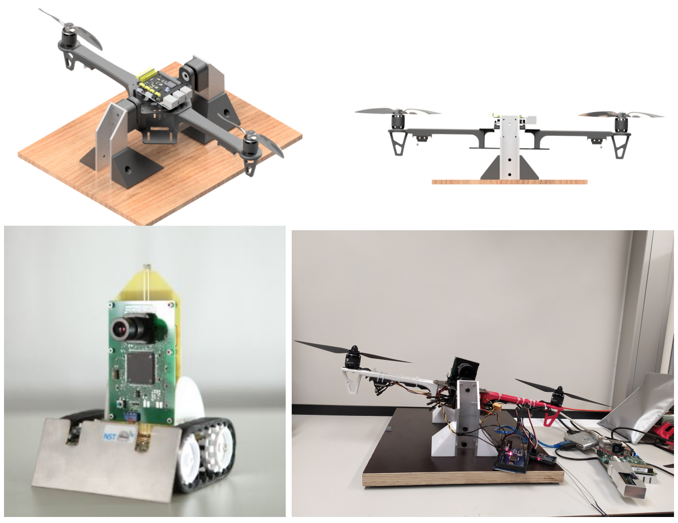

Fig. 1: **Top**: A 3D model of the physical test bench for UAV control. **Bottom**: The Pushbot robot and the complete UAV setup.

4) Interfaces, system overview: For the interface to the Kapoho Bay, we use an Intel UP^2 board. The high-level software components communicate using a light-weight peer-to-peer communication library *YARP* [27]. This enables a modular system design, in which we can easily swap a CPU-based PID controller with our Loihi implementation and the 1-DoF UAV with the Pushbot (Fig. 2). The injection of input spikes, extraction of SNN output, and configuration of the SNN are implemented using NxSDK.

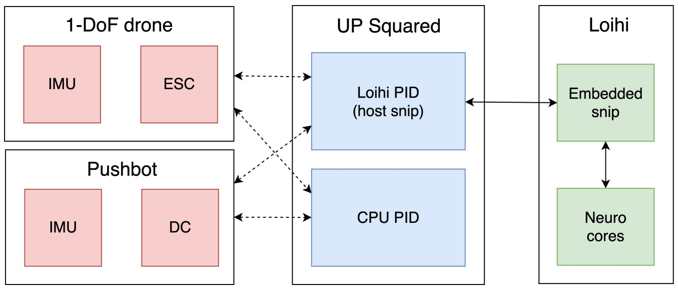

Fig. 2: Overview of the software-hardware setup: the IMU and the Electronic Speed Controllers (ESC) of the UAV or the IMU and the DC-motor encoders of the Pushbot communicate with the host computer (UP^2 board) over YARP. Communication with the Loihi chip is managed through YARP and a C-program running on the embedded x86 processor. The Loihi-based PID can be exchanged with a conventional CPU-based PID for comparison experiments.

B. Spiking Neural Network (SNN)

1) Neuronal representations: Our SNN consists of several groups of neurons – neuronal populations – arranged, conceptually, in arrays of different dimensionality. We use 1D populations of neurons to represent scalar values. E.g., a measured variable $a \in [-lim, lim]$ can be represented by a population of N neurons. Typically in population coding known from computational neuroscience, each neuron $n_i, i \in [0, N-1]$ is responsive to a range of the a values with a bell-shaped tuning curve: strongest response to a certain value and weaker response to nearby values. Thus, for any measured value a, several neurons in the population are active with different firing rates. In the network developed here, we use one-hot encoding: only one neuron is active at any time, representing a range of values, the whole population linearly mapping the interval $[-lim, lim]$ to $[0, N-1]$. The deterministic behavior of artificial neurons on Loihi allows us to use this more efficient code.

In our architecture, we also use 2D arrays of $N \times N$ neurons to realize mathematical *operations*. These 2D arrays connect two input 1D arrays and one 1D output. The *operations* arrays can have different connectivity to their output array to realize different computing functions. For instance, the addition *operation* adds two numbers represented by two 1D populations and passes the result to the third 1D population, as shown in Fig. 3. The output population receives input along the diagonals of the 2D *operation* array, and the output dimensionality, consequently, is $2N-1$. Thus, if we concatenate such operations, the size of the populations will grow, which is undesirable for large, e.g., hierarchical controllers. Therefore, we constrain the output of an *operation* array to have the same size as its inputs, mapping several extreme diagonals to the first and last neuron of the output population. This leads to minor boundary effects, which, however, did not impact results in our experiments. Using the same principle, a subtraction operation can be implemented by connecting the opposite diagonals of the 2D neuronal array to the output vector.

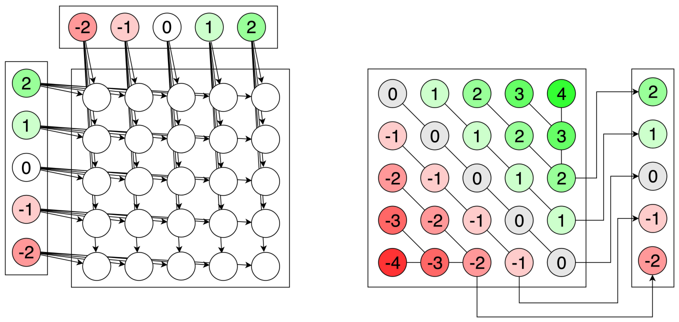

Fig. 3: **Left**: Connectivity from two 1D input arrays to the *operation* 2D array. Each neuron in the input array on the left is connected to all neurons on the respective row of the 2D array, and each neuron in the input array on top is connected to all neurons in the respective column. Numbers in the circles show the centers of the represented regions of the continuously measured variables. **Right**: The output connectivity of the same 2D *operation* array performing the addition operation. The size of the output 1D array is constrained to the size of the input arrays.

2) P-, I-, and D controller: Having realized the addition and subtraction operations with spiking neurons, the algebra of the PID controller can be implemented in the SNN network. The PID controller is defined by Eqs. (1):

$$\begin{aligned} u(t) &= K_p e(t) + K_i \int e(t)dt + K_d \frac{de(t)}{dt}, \\ e(t) &= r(t) - y(t), \end{aligned} \tag{1}$$

where $u(t)$ is the actuated command, $y(t)$ is the feedback/sensor signal, $r(t)$ is the control signal (desired value of $y(t)$), $e(t)$ is the error, and K_p, K_i, and K_d are the PID gains.

We reformulate the PID equations in a time-discrete way and define five values that will structure our SNN (e_t, p_t, i_t, d_t, and u_t) in Eqs. (2):

$$\begin{aligned} e_t &= r_{t-2} - y_{t-2} \quad \text{(error)}, \\ p_t &= e_{t-1} \quad \text{(proportional term)}, \\ i_t &= i_{t-2} + e_{t-2}\Delta t \quad \text{(integral term)}, \\ d_t &= \frac{e_{t-2} - e_{t-3}}{\Delta t} \quad \text{(derivative term)}, \\ u_t &= K_p p_t + K_i i_t + K_d d_t \quad \text{(command)}. \end{aligned} \tag{2}$$

Note, all computations on Loihi run in parallel, but the spike exchange is synchronized within a time-step. Thus, at least one time-step delay (Δt) is introduced by each connection (the delay can be longer, it is a parameter of a synapse). For simplicity, all connections in this network have a delay of one time-step. The system variables e_t, p_t, i_t and d_t can thus be defined with the exact delay in Eqs. (2). The architecture of the neural PID controller is visualized in Fig. 4.

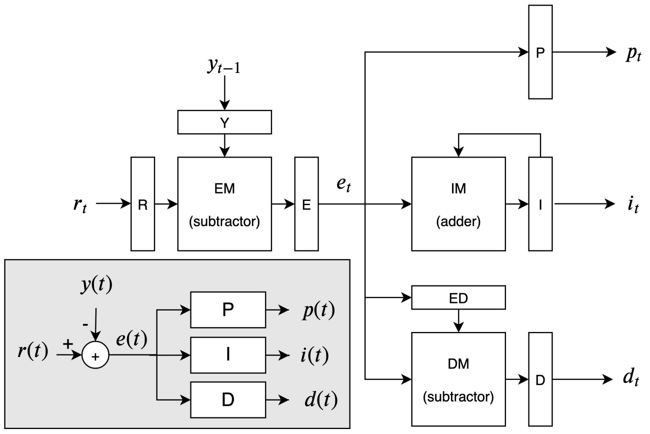

Fig. 4: The first part of the PID network consists of three 2D operation arrays (matrices): *EM* calculates the error/proportional signal by subtracting the control signal from the sensory feedback; *IM* calculates the integrated error signal by adding the current error with the last integrated error, fed back to the adder *IM*; and *DM* calculates the derivative of the error by subtracting the one time-step delayed error signal from the current error signal. Inset shows an overview of a PID controller.

3) Spike-based operation of the network: We use a *one-spike* approach in our SNN, in which only one neuron in each population can fire at any time-step. This behavior is achieved by setting the maximum voltage- and current decays in neuronal compartments of all populations. This resets the membrane potential of neurons to the resting state at every time-step. Furthermore, the threshold of neurons in the 2D arrays is set to value of 3, and all the input weights to the 2D arrays are set to 2. This means that a neuron in any 2D array can only fire if it receives two spikes within the same time-step. At each time-step, at most one spike is output in each of the two input populations (e.g., *R* and *Y*) of the 2D *operation* array, e.g. *EM* (Error Matrix). This forces one neuron in the *EM* to fire. This, in turn, triggers firing of one neuron in the output population, *E* (error). In this way, spikes propagate through the network and maintain the one-spike behavior throughout all neuronal populations. Which neuron in each population fires determines the represented value at that time-step.

4) Output network: rate-coded populations: To enable future adaptive learning of the PID constants, two approaches are possible. A gain constant can be implemented as weights between two 1D neuronal arrays that represent values with population encoding (e.g., one-hot sparse encoding in our network). This gain representation will allow for an arbitrary mapping between the pre-gain values and the post-gain values, i.e., any non-linear scaling of P, I, and D terms can potentially be realized. Being a more powerful representation, we anticipate that learning such a gain matrix will be more difficult.

Thus, although we realized the network following both approaches, our results shown here are based on the second approach, in which the gain is represented by the weights between two rate-coded populations. A rate-coded population is a population of neurons with a homogeneous distribution of resting level membrane potentials. The overall firing rate of the whole population represents the current value of the variable. Note how this stays in contrast to the sparse population code, in which the value is represented by the identity (index, or address) of the firing neuron(s). Synaptic weights scale input to a rate-coding population, increasing or decreasing the number of neurons that get above threshold in response to input spikes (remember, there is only one spike per time-step per neuron) and thus controlling the overall firing rate. A simple learning rule that increases or decreases the weights depending on the controller's behavior can adjust the P, I, and D gains.

Thus, in our architecture, we convert the sparse coded *P/I/D* representations to population-rate code before the three values go through weights that encode the P, I, and D gains and are summed up in the control output population, U. In the sparse-coding *P/I/D* arrays, the represented values are in the range $a \in [-lim, lim]$, with $a = 0$ at the central neuron. Each neuron in the *P/I/D* 1D arrays is connected to either *P/I/D+* or *P/I/D-* rate-coding populations, depending on the sign of the represented value. The weight that connects each neuron in the sparse-coded populations is proportional to the magnitude of the encoded value (i.e., its index in the array, counted from the center). This connectivity produces high activity in the *P/I/D+* or *P/I/D-* neurons when edge neurons in the *P/I/D* arrays are active, and, respectively, low activity if a neuron close to the center of the *P/I/D* arrays is active (see Fig. 5).

It is important that neurons in the population-rate coded populations fire asynchronously: if all neurons fire synchronously, the overall firing rate will not depend on input strength in our single-spike network. Thus, small bias and noise currents are injected into the *P+* and *P-* neurons to ensure uniformly distributed voltage potentials that enable asynchronous behavior of the neurons. Both *P+* and *P-* are connected to the summing-up population *A* with the weight of respectively K_p and $-K_p$. Note that the transfer function of weights between two rate-coded populations may have a non-linear form; this requires further investigation and will be alleviated if weight adaptation is used to tune the gains.

The same steps as for *P* populations are made for *I* and *D* arrays (Fig. 6). A small bias is injected into the summing population *A* to ensure a firing activity of *A* as an offset. From this offset *P-*, *I-*, and *D-* lower the activity and *P+*, *I+*, and *D+* raise the activity.

The population-rate coded sum of control signals, represented in *A*, is converted back into sparse code in *B* by introducing linearly distributed thresholds of the neurons in *B*. *A* and *B* are connected in an all-to-all manner. The neuron of *B* representing the lowest value $(-lim)$ spikes in response to a single input spike and the neuron representing the highest value (lim) spikes only, when the activity of *A* is the highest. Every level of activity between $[-lim, lim]$ is linearly mapped along the *B* population. To get back to the one-spike encoding, each neuron in *B* is connected in a one-to-one manner to neurons in *U* with a positive weight, and to all neurons of *U* with lower indices with negative weights (Fig. 5).

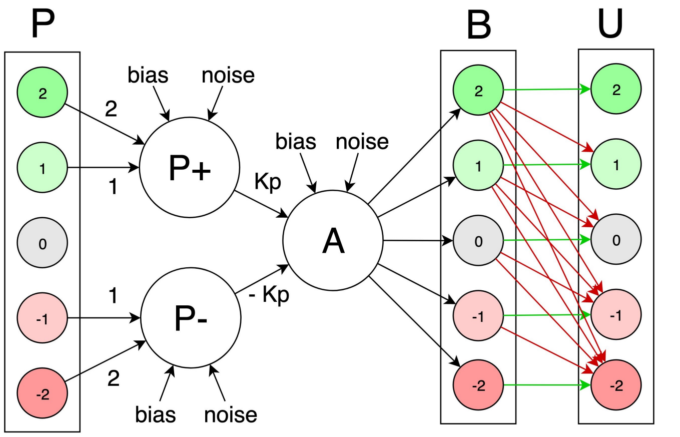

Fig. 5: Conversion from the sparse place-coded array (one-hot coding) (P) to the rate-coded neuronal populations (P+, P-). The same structure is used for I and D pathways. Adaptive gain can be applied to the weights from P+ and P- to the summing population A (weights +/-K$_p$). The rate-coded values are transferred back to the space-coded control output population U through an auxiliary population B. The space-code in the U population simplifies connection to the motors in our setup (the address of the firing neuron can easily be accessed in an embedded snip, alleviating the need to calculate firing rates).

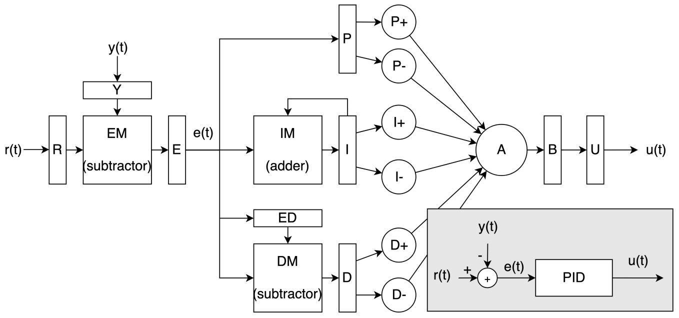

Fig. 6: Overview of the full PID controller and the output network with the space-to-rate code conversion, required to simplify learning of the controller gains.

C. Implementation on robots

1) Pushbot: For simplicity, a single PID controller was initially tested with the *Pushbot*[1]. The Pushbot's angular velocity was controlled to the target angular velocity (R) using angular velocity (Y), measured with an IMU or wheel encoders.

2) 1-DoF UAV: In order to control the angular position of the *1-DoF UAV*, two nested PID controllers are needed. The outer PID outputs the target angular velocity based on the target angular position and the current angular position. The inner PID computes the target torque based on the target angular velocity obtained from the outer PID and the current angular velocity.

D. Constraints of neuromorphic hardware and network size

Each neuronal core on Loihi can simulate up to 1024 neurons, which enables 2D populations of size 32×32 neurons. To increase population size, we divide the 2D populations into multiple segments and distribute them across multiple neuronal cores, increasing the size of available 2D operation arrays. Each connection in our network is initialized with its own input- and output axon (weight sharing is not supported in the current version of the API, NxSDK 0.9). The number of axons per core is limited to 4K. For each 2D-population's segment, $2n^2$ input- and n^2 output-axons are needed, where n^2 is the size of the 2D array's segment. To initialize two (nested) PID controllers for the UAV on Loihi, we need six 2D arrays. Thus, $6m^2$ neuro-cores are used for simulation of the 2D populations, where m^2 is the number of segments, in which each 2D array is divided. Since 128 neuro-cores are available on one chip, arrays of size 63^2 can be used (each 2D array consists of 3^2 segments of size 21^2). These populations use 54 neuro-cores and leave 74 neuro-cores for the rest of the network.

E. Experiments

We tested the SNN-based PID controller running on Loihi with both robotic systems and compared the performance against a conventional PID controller implemented on a CPU (the UP^2 board[2]). Furthermore, for the UAV, we implemented a constrained-PID (CPID) on the CPU that uses the same numeric precision as the SNN on Loihi (63 different values). This shows the impact of lowering the resolution of value representation without introducing the probabilistic behavior of the rate-coded populations. All connections in the SNNs were static, the PID gains of both systems were tuned by hand. [3]

F. Pushbot

The IMU integrated in the eDVS sensor of the Pushbot robot outputs the angular velocity, measured by a combination of a 3-axis gyroscope and the linear accelerometer. The velocity values were sampled at $20Hz$ (note, Loihi can support input at $> 10kHz$). The IMU-measured values were sent to the embedded program on Loihi and used to generate input spikes to the Y population. The two input spikes – to the Y and R populations – generate a cascade of spikes in the SNN architecture, eventually leading to a single spike in the output U population 12 time-steps later. The index of the firing neuron of the U population is recorded in the embedded program and is interpreted as the angular-velocity command sent to the robot. This command is directly translated into motor commands making the robot turn with the set velocity.

G. 1-DoF UAV

To control the angular position of the UAV in one degree of freedom, two nested PID controllers were used. The first PID receives the control signal (r_t) and the current angular position (y_t), measured by the IMU. The PID computes the target angular velocity u_t. u_t is passed-on into the second PID controller that also receives the current angular velocity ($\dot{y}_t$) from IMU as a 1D input and computes the target angular acceleration $\dot{u}_t$.

III. Results

To visually show the performance of the controller, for exemplary tests, we show a plot comparing the SNN implementation against CPU based- and constrained CPU based PID controllers. We also show tables with measures of overshoot, rise time, and settling time of the systems for quantitative comparison. A video of one of the test runs, along with the measured controller outputs, can be found in Supplementary.

A. Pushbot control

Fig. 7 shows the results of an exemplary run of the SNN controller on Loihi controlling the angular velocity of the Pushbot robot. The set value (blue line in the plot) was generated on the host computer and sent to the Loihi chip as input to the R population. The numerical values were interpreted in an embedded program on an x86 core, and a single spike per time-step was sent to one neuron in the R population as input. The full range of available speed ($\pm 150 deg/s$) was mapped onto 63 neurons of the 1D population. In Fig. 7, the red curve shows the angular velocity of the robot measured by IMU when the Pushbot is controlled by spikes from the U population in the SNN on Loihi. The green curve shows the performance of the software PID controller.

[1]http://inilabs.com/products/pushbot/

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

[3]All performance results are based on testing as of February 2020 and may not reflect all publicly available security updates. No product can be absolutely secure.

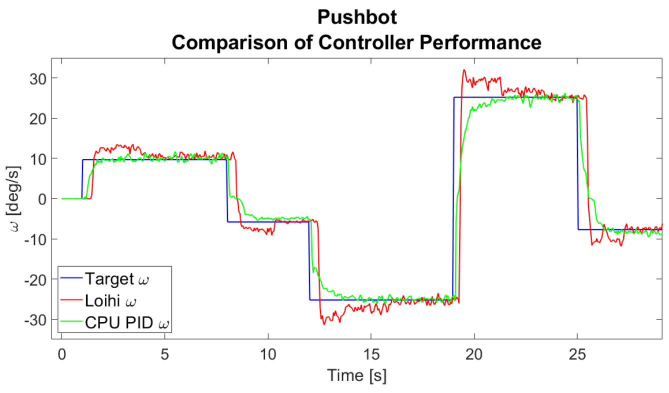

Fig. 7: Output of PID controllers for the Pushbot robot. **Blue**: the control signal for the PID (target angular velocity). **Red**: the angular velocity measured by IMU when UAV is controlled by the SNN-based PID running on Loihi. **Green**: the angular velocity measured by IMU when UAV is controlled by the software PID controller running on a CPU.

Table I shows the overshoot, rise time, and settling time of the SNN PID controller throughout four trails. Compared to the CPU-based PID values (Table II), it can be seen that there is no overshoot in the CPU PID compared to the mean $5.89 deg/s$ overshoot in the SNN PID. The rise time is shorter in the SSN PID and the settle time is longer. This is a sign that the D gain has a stronger effect in the CPU implementation. However, both controllers work well despite the delay in the system.

TABLE I: Performance of the SNN PID controller (Pushbot)

SNN PID	Overshoot $[deg/s]$	Rise time $[s]$	Settling time $[s]$
Trial 1	5.753	0.105	4.95
Trial 2	5.997	0.101	3.80
Trial 3	5.972	0.094	5.08
Trial 4	5.844	0.104	4.30
Mean	5.892	0.101	4.531
Std.	0.988	0.004	0.519

TABLE II: Performance of the CPU PID controller (Pushbot)

CPU PID	Overshoot $[deg/s]$	Rise time $[s]$	Settling time $[s]$
Trial 1	0	0.745	2.265
Trial 2	0	0.798	2.457
Trial 3	0	0.800	2.356
Trial 4	0	0.698	2.246
Mean	0	0.760	2.331
Std.	0	0.042	0.084

B. 1-DoF UAV

Fig. 8 shows the behavior of the SNN PID and software PIDs for the UAV, in an exemplary run.

Table III shows the average overshoot, rise time, and settling time for three PID controllers on the UAV, computed based on four trials. It can be noticed that the SNN PID overshoots more than the other controllers and has a faster rise time. This indicates again that the D gain has more effect in the CPU implementation than in the SNN implementation. Adaptation of gains in a learning process should be able to find a better parametrization of the SNN-based PID controller.

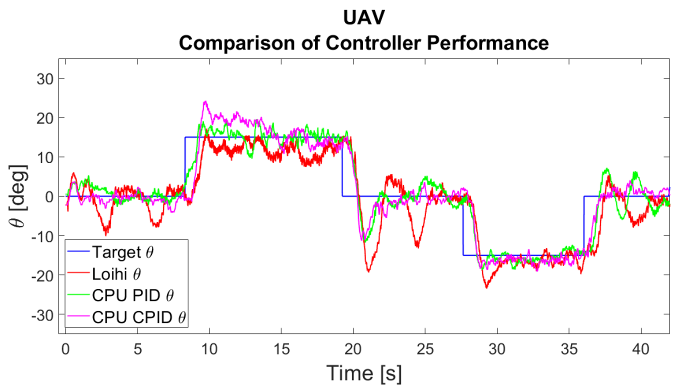

Fig. 8: **Blue**: control signal for the outer PID (target angle). **Red**: performance of the SNN-PID running on Loihi (measured UAV angle). **Green**: performance of a conventional PID controller on CPU. **Magenta**: performance of the CPU-PID constrained to the same value representations as the SNN-PID.

TABLE III: Performance of PID controllers for UAV

	Overshoot $[deg]$	Rise time $[s]$	Settling time $[s]$
SNN PID	8.319	3.419	21.781
CPU PID	4.128	4.173	21.772
CPID	4.876	3.872	21.764

IV. Discussion

Before implementing adaptation of the control gains using on-chip plasticity, several challenges of our setup must be addressed. Thus, the Pushbot followed the control signal with approx $450ms$ response latency. This response time is partly due to the delay from the sensor to the host program and back to the robot and the sensors through the physical system. This accounts for approximately $100ms$ of delay and affects the results for both the Loihi- and the CPU PID. The larger part of the delay is introduced by the long time-step on Loihi. It takes 12 time-steps of $27ms$ for the data to flow through the PID controller (this is approx. $323ms$). By optimizing I/O, the timestep duration could be reduced down to $10\mu s$ in a future implementation, considerably improving the results.

The performance of the Pushbot and the UAV differ since the two control problems are different. The Pushbot's controller is controlling the first derivative of the position, which is damped by the motors and gears, making the system easier to control and to tune since the D term becomes obsolete. The UAV controller is controlling the second derivative of a non-linear system, which is a harder control problem. Furthermore, the UAV was unbalanced in our test bench due to cable pulling. As can be observed in Fig. 8, there is more overshoot in the negative direction than in the positive direction.

V. Conclusion

We implemented a PID controller fully in a Spiking Neural Network on Intel's neuromorphic research chip Loihi. The

SNN was tested on two robotic platforms and was able to control both. The results show that the behavior of the controllers matches the behavior of conventional PID controllers, but the performance was worse due to long time-steps causing delays. A more direct low-level interface to Loihi cores will enable fast, responsive, and reliable communication between the chip and the robot as the hardware matures, significantly reducing the response time.

To our knowledge, this is the first attempt to implement a PID controller fully in neuromorphic hardware. Having access to an SNN implementation of a PID controller will lead to new applications of neuromorphic technology. Implemented in the concurrent neuromorphic hardware, the SNN controller can be scaled-up to control multiple degrees of freedoms and allows us to build nested and hierarchical controllers. The PID controller can be combined with SNN-based forward and inverse models with online learning and adaptation [28] and can be seamlessly integrated with SNN-based vision systems on chip [29]–[35]. A complete SNN-based robotic solutions without the need for conventional sequential hardware in the loop will reduce bottlenecks on the interface between the neuronal and conventional computing systems.

No learning was applied in this work. However, the output network in our SNN was designed to be compatible with online learning in order to enable autonomous adaptation of the controller gains, which is the subject of our further work.

VI. Acknowledgements

We would like to thank SDU-UAV for founding all drone related hardware and Cao Dahn Do for building the 1-Dof UAV test bench. We thank Intel Labs for providing the state-of-the art neuromorphic research chip Loihi as well as a lot of support. This project was funded by SNSF grant Ambizione PZOOP2_168183 ELMA and was initiated during the Capo Caccia 2019 Workshop "Toward Neuromorphic Intelligence".

References

[1] A. Giusti, J. Guzzi, D. C. Cireşan, F.-L. He, J. P. Rodríguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, G. Di Caro *et al.*, "A machine learning approach to visual perception of forest trails for mobile robots," *IEEE Robotics and Automation Letters*, vol. 1, no. 2, pp. 661–667, 2015.

[2] H. A. Pierson and M. S. Gashler, "Deep learning in robotics: a review of recent research," *Advanced Robotics*, vol. 31, no. 16, pp. 821–835, 2017.

[3] N. Sünderhauf, O. Brock, W. Scheirer, R. Hadsell, D. Fox, J. Leitner, B. Upcroft, P. Abbeel, W. Burgard, M. Milford *et al.*, "The limits and potentials of deep learning for robotics," *The International Journal of Robotics Research*, vol. 37, no. 4-5, pp. 405–420, 2018.

[4] D. Palossi, A. Loquercio, F. Conti, E. Flamand, D. Scaramuzza, and L. Benini, "A 64-mw dnn-based visual navigation engine for autonomous nano-drones," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8357–8371, Oct 2019.

[5] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard, "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age," *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1309–1332, Dec 2016.

[6] S. Milz, G. Arbeiter, C. Witt, B. Abdallah, and S. Yogamani, "Visual slam for automated driving: Exploring the applications of deep learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2018, pp. 247–257.

[7] K. Tateno, F. Tombari, I. Laina, and N. Navab, "Cnn-slam: Real-time dense monocular slam with learned depth prediction," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 6243–6252.

[8] R. Li, S. Wang, and D. Gu, "Ongoing evolution of visual slam from geometry to deep learning: challenges and opportunities," *Cognitive Computation*, vol. 10, no. 6, pp. 875–889, 2018.

[9] P. Karkus, D. Hsu, and W. S. Lee, "Qmdp-net: Deep learning for planning under partial observability," in *Advances in Neural Information Processing Systems*, 2017, pp. 4694–4704.

[10] S. Toyer, F. Trevizan, S. Thiébaux, and L. Xie, "Action schema networks: Generalised policies with deep learning," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[11] V. Rausch, A. Hansen, E. Solowjow, C. Liu, E. Kreuzer, and J. K. Hedrick, "Learning a deep neural net policy for end-to-end control of autonomous vehicles," in *2017 American Control Conference (ACC)*. IEEE, 2017, pp. 4914–4919.

[12] S. Yang, W. Wang, C. Liu, and W. Deng, "Scene understanding in deep learning-based end-to-end controllers for autonomous vehicles," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 49, no. 1, pp. 53–63, 2018.

[13] T.-M. Hsu, C.-H. Wang, and Y.-R. Chen, "End-to-end deep learning for autonomous longitudinal and lateral control based on vehicle dynamics," in *Proceedings of the 2018 International Conference on Artificial Intelligence and Virtual Reality*, 2018, pp. 111–114.

[14] 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.

[15] 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

[16] 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.

[17] 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

[18] 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.

[19] K. J. Åström and T. Hägglund, *PID controllers: theory, design, and tuning*. Instrument society of America Research Triangle Park, NC, 1995, vol. 2.

[20] 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.

[21] S. Akhyar and S. Omatu, "Self-tuning PID control by neural networks," *Proceedings of the International Joint Conference on Neural Networks*, vol. 3, pp. 2749–2752, 1993.

[22] A. Guez, J. Eilbert, and M. Kam, "Neuromorphic architecture for adaptive robot control: A preliminary analysis." 12 1987, pp. iv/567–572.

[23] A. Guez and J. Selinsky, "A trainable neuromorphic controller," *Journal of robotic systems*, vol. 5, no. 4, pp. 363–388, 1988.

[24] https://inilabs.com/products/pushbot/.

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

[26] R. S. Dimitrova, M. Gehrig, D. Brescianini, and D. Scaramuzza, "Towards low-latency high-bandwidth control of quadrotors using event cameras," *arXiv preprint arXiv:1911.04553*, 2019.

[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] 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.

[29] X. Lagorce and R. Benosman, "STICK: Spike Time Interval Computational Kernal, a Framework for General Purpose Computation Using Neurons, Precise Timing and Synchrony," *Neural computation*, vol. 27, pp. 2261–2317, 2015.

[30] X. Lagorce, G. Orchard, F. Gallupi, B. E. Shi, and R. Benosman, "HOTS: A Hierarchy Of event-based Time-Surfaces for pattern recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 8828, no. c, pp. 1–1, 2016. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7508476

[31] D. Liang, R. Kreiser, C. Nielsen, N. Qiao, Y. Sandamirskaya, and G. Indiveri, "Neural state machines for robust learning and control of neuromorphic agents," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 4, pp. 679–689, Dec 2019.

[32] S. Baumgartner, A. Renner, R. Kreiser, D. Liang, G. Indiveri, and Y. Sandamirskaya, "Visual pattern recognition with on on-chip learning: towards a fully neuromorphic approach," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2020.

[33] J. H. Lee, T. Delbruck, and M. Pfeiffer, "Training deep spiking neural networks using backpropagation," *Frontiers in Neuroscience*, 2016.

[34] E. O. Neftci, H. Mostafa, and F. Zenke, "Surrogate gradient learning in spiking neural networks," *arXiv preprint arXiv:1901.09948*, 2019.

[35] F. Zenke and S. Ganguli, "Superspike: Supervised learning in multilayer spiking neural networks," *Neural computation*, vol. 30, no. 6, pp. 1514–1541, 2018.