Fully neuromorphic vision and control for autonomous drone flight

Paredes-Vallés, F.; Hagenaars, J.J.; Dupeyroux, J.; Stroobants, S.; Xu, Y.; de Croon, G.C.H.E.

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

AUTONOMOUS VEHICLES

Fully neuromorphic vision and control for autonomous drone flight

F. Paredes-Vallés†, J. J. Hagenaars*†, J. Dupeyroux†, S. Stroobants, Y. Xu, G. C. H. E. de Croon

Biological sensing and processing is asynchronous and sparse, leading to low-latency and energy-efficient perception and action. In robotics, neuromorphic hardware for event-based vision and spiking neural networks promises to exhibit similar characteristics. However, robotic implementations have been limited to basic tasks with low-dimensional sensory inputs and motor actions because of the restricted network size in current embedded neuromorphic processors and the difficulties of training spiking neural networks. Here, we present a fully neuromorphic vision-to-control pipeline for controlling a flying drone. Specifically, we trained a spiking neural network that accepts raw event-based camera data and outputs low-level control actions for performing autonomous vision-based flight. The vision part of the network, consisting of five layers and 28,800 neurons, maps incoming raw events to ego-motion estimates and was trained with self-supervised learning on real event data. The control part consists of a single decoding layer and was learned with an evolutionary algorithm in a drone simulator. Robotic experiments show a successful sim-to-real transfer of the fully learned neuromorphic pipeline. The drone could accurately control its ego-motion, allowing for hovering, landing, and maneuvering sideways—even while yawing at the same time. The neuromorphic pipeline runs on board on Intel's Loihi neuromorphic processor with an execution frequency of 200 hertz, consuming 0.94 watt of idle power and a mere additional 7 to 12 milliwatts when running the network. These results illustrate the potential of neuromorphic sensing and processing for enabling insect-sized intelligent robots.

INTRODUCTION

Over the past decade, deep artificial neural networks (ANNs) have revolutionized the field of artificial intelligence. Among the successes has been the substantial improvement of visual processing, to an extent that computer vision can now outperform humans on specific tasks (1). The field of robotics has also benefited from this development, with deep ANNs achieving state-of-the-art accuracy in tasks such as stereo vision (2, 3), optical flow estimation (4–6), segmentation (7, 8), object detection (9–11), and monocular depth estimation (12–14). However, this high accuracy typically relies on substantial neural network sizes that require heavy and power-hungry processing hardware [tens of watts, hundreds of grams (15)]. This limits the number of tasks that can be performed by larger (ground) robots and even prevents deployment on smaller robots with highly stringent resource constraints, like small flying drones.

Neuromorphic hardware may provide a solution to this problem because it mimics the energy-efficient, sparse, and asynchronous nature of sensing and processing in biological brains (16, 17). For example, the pixels in neuromorphic event-based cameras only transmit information on brightness changes (18). Because typically only a fraction of the pixels change in brightness substantially, this leads to sparse vision inputs with subsequent events that are on the order of a microsecond apart. The asynchronous and sparse nature of visual inputs from event-based cameras represents a paradigm shift compared with traditional, frame-based computer vision. Ideally, processing would exploit these properties for quicker, more energy-efficient processing. However, currently, learning-based approaches to event-based vision involve accumulating events over a substantial amount of time, creating an "event window" that represents extended temporal information. This window is then processed similarly to a

traditional image frame with an ANN (19–22). Although there is work that used much shorter event windows (23–25), latency could be further reduced by processing events asynchronously as they come in. This could be performed by neuromorphic processors designed for implementing spiking neural networks (SNNs) (26, 27). These networks have temporal dynamics more similar to biological neurons. In particular, the neurons have a membrane voltage that integrates incoming inputs and causes a spike when it exceeds a threshold. The binary nature of spikes allows for more energy-efficient processing than the floating point arithmetic in traditional ANNs (28, 29). The energy gain is further improved by reducing the spiking activity as much as possible, as is also a main property of biological brains (30). Coupling neuromorphic vision to neuromorphic processing promises low-energy and low-latency visual sensing and acting, as exhibited by agile animals such as flying insects (31).

However, to achieve these properties, several challenges related to present-day neuromorphic sensing and processing have to be overcome. For example, training is currently still much more difficult for SNNs than for ANNs (32, 33), mostly because of their sparse, binary, and asynchronous nature. The most well-known difficulty of SNN learning is the non-differentiability of the spiking activation function, which prevents naive application of backpropagation. Currently, this is tackled successfully with the help of surrogate gradients (34, 35), although longer sequences (as would be the case for event-by-event processing) can still lead to gradient vanishing. Moreover, although the richer neural dynamics can potentially represent more complex temporal functions, they are also harder to shape, and neural activity may saturate or dwindle during training, preventing further learning. The causes for this are hard to analyze because there are many parameters that can play a role. Depending on the model, the relevant parameters may range from neural leaks and thresholds to recurrent weights and time constants for synaptic traces. A solution may lie in learning these parameters (36, 37), but this further increases the dimensionality of the learning problem. Last, when targeting a robotics application, SNN training

Micro Air Vehicle Laboratory, Faculty of Aerospace Engineering, Delft University of Technology, Delft, Netherlands.
*Corresponding author. Email: j.j.hagenaars@tudelft.nl
†These authors contributed equally to this work.

and deployment are further complicated by the restrictions of existing embedded neuromorphic processing platforms. These restrictions were recently highlighted in a study on neuromorphic route learning (38) and include hardware and software challenges such as interfacing with neuromorphic cameras, the limitations of available neural models, and—importantly—the still-limited numbers of available neurons and synapses. To illustrate this latter point, the ROLLS (Reconfigurable Online Learning Spiking) chip (39) accommodates 256 spiking neurons, the Intel Kapoho Bay [featuring two Loihi chips (40) in a universal serial bus (USB) stick form factor] features 262,100 neurons (41), and the SpiNNaker (SNN architecture) version in (42) has 768,000 neurons. Although these chips differ in many more aspects than the number of neurons, this small sample already shows that current state-of-the-art SNNs cannot be easily embedded in robots. SNNs that have recently been trained on visually complex tasks, such as optical flow determination (23, 24), still featured network sizes far too large for implementation on current neuromorphic processing hardware for embedded systems. The smallest size SNN in these studies is LIF-FireFlowNet (Leaky-Integrate and Fire) for optical flow estimation (24), which still has 3.7 million neurons (at an input resolution of 128 pixels by 128 pixels).

As a consequence, pioneering work in this area has been limited in complexity. Very early work involved the evolution of SNN connectivity to map the 16 visual brightness inputs of a wheeled Kephera robot to its two motor outputs (43). The evolved SNN, simulated in software, allowed the robot to avoid the walls in a black-and-white-striped environment. Most work exploring SNNs for robotic vision focused on simulation. For example, in (44), the events from a simulated event-based camera with 128 pixels by 128 pixels were accumulated into frames, compressing them over time into eight-by-four Poisson input neurons. These inputs, which captured the clear white lines of the road border, were then directly mapped to two output neurons for staying in the center of the road with the help of reward-modulated spike time–dependent plasticity (R-STDP) learning. Robotic examples of in-hardware neuromorphic processing for vision are more rare. An early example is the one in (42), in which an event-based camera with 128 pixels by 128 pixels was connected to a SpiNNaker neuromorphic processor to allow a driving robot to differentiate between two lights flashing at different frequencies with a 128-neuron winner-takes-all network. In (45), an SNN was designed for following a light target in the top half of the field of view while avoiding regions with many events in the bottom half of the field of view. This network was successfully implemented in the ROLLS neuromorphic chip (39) and tested in an office environment. Recent years have seen an increasing focus on flying robots (drones) because they need to react quickly while being extremely restricted in size, weight, and power (SWaP). In (41), an SNN was implemented on a bench-fixed dual rotor to align the roll angle with a black-and-white disk located in front of the camera. The SNN involved both a visual Hough transform (46) for finding the line and a proportional-derivative (PD) controller for generating the propeller commands. Last, in (47), an SNN was first evolved in simulation and then implemented in Loihi for vision-based landing of a flying drone. This control network only consisted of 35 neurons because the visual processing was still performed with conventional, frame-based computer vision methods. In addition, only the vertical motion of the drone was controlled with the SNN; its lateral position was controlled using traditional control algorithms and an external motion capture system. The current work represents a step

up in complexity by performing three-dimensional (3D) visual ego-motion estimation and control of a flying drone with a fully neuromorphic vision-to-control pipeline.

In this article, we present a fully neuromorphic vision-to-control pipeline for controlling a flying drone, demonstrating the potential of neuromorphic hardware. The presented pipeline consists of an SNN that was trained to accept raw event camera data and output low-level control actions for performing autonomous vision-based ego-motion estimation and control at approximately 200 Hz. A core property of our learning setup is that it splits vision and control, which provides two major advantages. First, it helps to prevent the reality gap on the camera event input side because the vision part was trained on the basis of raw events from the actual event camera on the drone. We used self-supervised learning because this foregoes the need for ground-truth measurements that are difficult to obtain for event-based vision. Second, because the output of the vision part is an ego-motion estimate, we can learn the control in a simple and extremely fast simulator. This allows us to evade the high-frequency generation of visually realistic images for event generation (48), something that would lead to excessive training times in an end-to-end learning setup.

RESULTS

A fully neuromorphic solution to vision-based navigation

The fully neuromorphic vision-to-control pipeline, illustrated in Fig. 1C, was implemented on the Loihi neuromorphic processor (40) and used on board a small flying robot (see Fig. 1B) for vision-based navigation. A schematic of the hardware setup used is shown in Fig. 1A. The system successfully followed ego-motion set points (target values) in a fully autonomous fashion, without any external aids such as a positioning system. Figure 1D shows an example of a landing experiment with our neuromorphic pipeline in the control loop of the drone. The figure shows the smoothly decreasing height of the drone above the ground (blue line) and the estimated optical flow divergence (orange line), which is the vertical component of the velocity vector divided by this height. The divergence curve is typical of an optical flow divergence landing, first approaching the set point -0.5 s^{-1} and then becoming more oscillatory when getting very close to the ground (49).

As mentioned, the main challenge of deploying such a pipeline on embedded neuromorphic hardware is that, because of the preliminary state of this technology, one has to work within very tight limits regarding the available computational resources. In this project, several design decisions were made to adapt to these limitations. First, the vision processing pipeline assumes that the event-based camera on the drone, the DVS 240 (Dynamic Vision Sensor) (50), looks down at a static, texture-rich, flat surface. Knowing the structure of the visual scene in advance simplifies the estimation of the ego-motion of the camera (and hence of the drone) with the help of optical flow information, as in (47, 49, 51–53). Optical flow, or the apparent motion of scene points in the image space, can be estimated from the output of an event-based camera with a wide variety of methods, ranging from sparse feature-tracking algorithms (54) to dense (per-pixel) machine learning models (19, 21, 24). In the search for an efficient and high-bandwidth vision pipeline that achieves the desired 200-Hz operating frequency, the second design decision was to reduce the spatial resolution of the event-based vision data by only processing information from the image corner
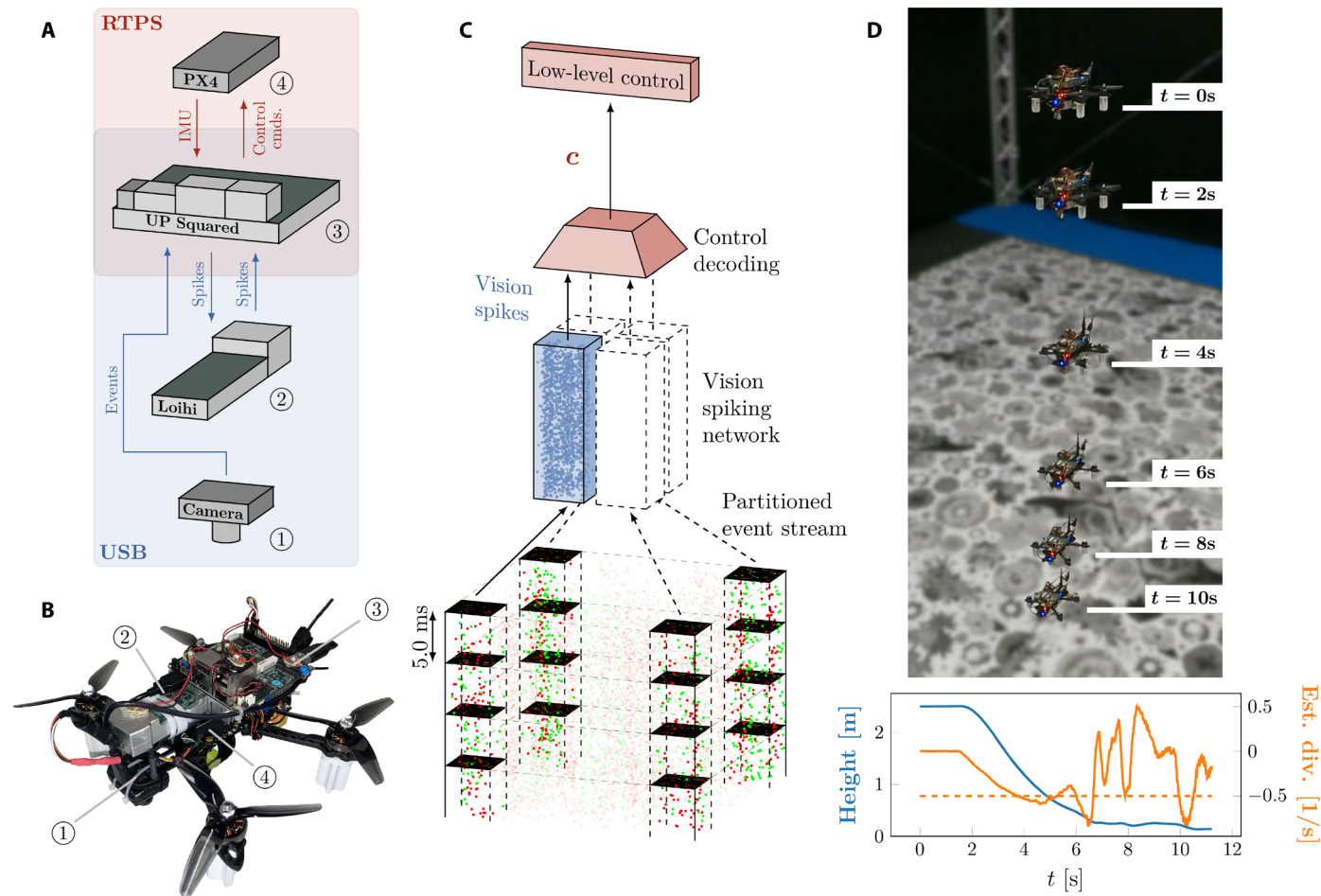
Fig. 1. Overview of the proposed system. (**A**) Hardware overview showing the communication among event camera, neuromorphic processor, single-board computer, and flight controller. RTPS and USB refer to the communication protocols used. (**B**) Quadrotor used in this work (total weight of 994 g; tip-to-tip diameter of 35 cm), with numbers indicating the components from (A). (**C**) Pipeline overview showing events as input, processing by the vision network, and decoding into a control command. (**D**) Composite picture and graph demonstrating the system for an optical flow constant divergence landing, with estimated divergence (solid orange line) oscillating around a set point (dashed orange line) while height is decreasing (solid blue line).

regions of interest (ROIs), rather than the entire image space, and to limit the number of events to 90 per ROI. More specifically, as depicted in Figs. 1C and 2A, we propose the use of a small SNN that is applied independently at each ROI, with each ROI being 16 pixels by 16 pixels in size after a nearest-neighbor downsampling operation. Each network consists of 7200 neurons and 506,400 synapses distributed over five spiking layers: one input layer, three self-recurrent encoders, and a pooling layer. Its parameters (weights, thresholds, and leaks) are identical for the four ROIs, and it estimates the optical flow, in pixels per millisecond, of the corresponding ROI. Because of the static and planar scene assumption, the apparent motion of the scene points at the four corner ROIs encodes nonmetric information about the velocity of the camera (divided by the distance to the surface along the optical axis) and its rotational rates in a linear manner (*55*).

On the basis of this information, the robot performs ego-motion control. To keep the pipeline fully neuromorphic (minimum required processing happening outside of Loihi) and performant (sending more spikes to Loihi decreases execution frequency), we

trained a linear controller in simulation and merged it with the decoding of the spikes coming from the vision network (representing optical flow). In other words, the linear controller takes vision spikes, a user-given ego-motion set point, and attitude of the drone and maps these linearly to thrust and attitude control commands. Although opting for a linear controller allows for a fully neuromorphic vision-to-control pipeline, it also means that we had to make some assumptions. For instance, angles in pitch and roll should be small, and the optical flow variables taken as input should be derotated in pitch and roll (*51*, *56*). Furthermore, we should keep in mind that a linear controller will be unable to compensate for any drift or steady-state offset through integration (as could be done by a common proportional integral derivative, or PID, controller). We show that, despite all this, we can successfully control a flying drone to perform certain ego-motion maneuvers.

We split the training of our vision-to-control pipeline into two separate frameworks. On the one hand, the vision part of the pipeline, in charge of mapping input events to optical flow, was trained in a self-supervised fashion using the contrast maximization
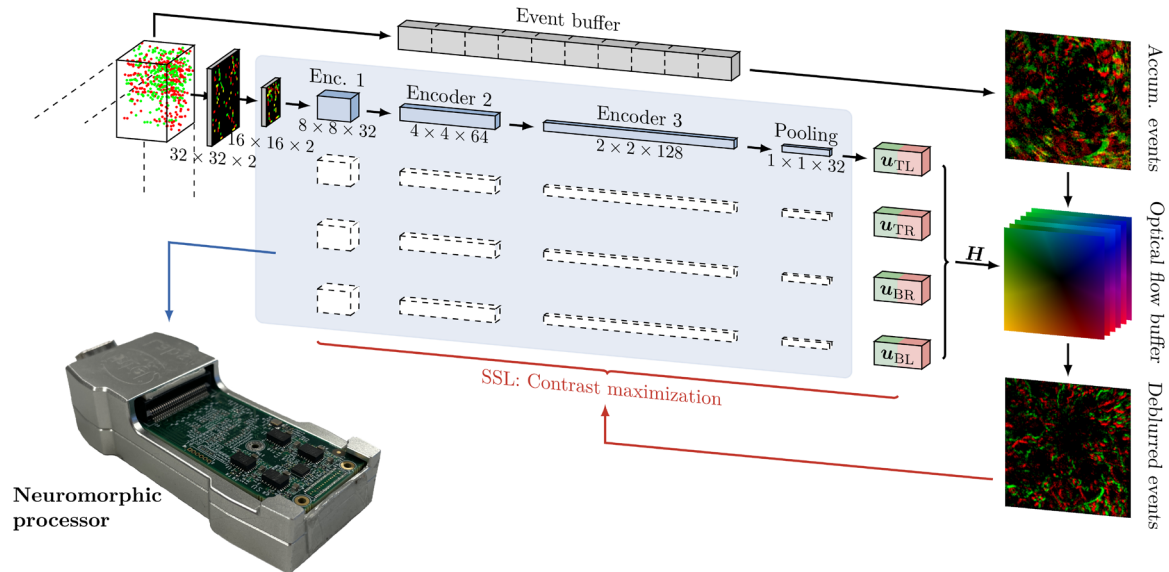
Fig. 2. Overview of the spiking vision network. Running at approximately 200 Hz, events are accumulated (max of 90 events per corner ROI) and then fed through the vision network consisting of three encoders (kernel size three by three, stride 2) and a spiking pooling layer. Spikes are decoded into two floats representing flow for that corner ROI. This network is replicated to the three other ROIs to end up with four ROI optical flow vectors. During training, these are used in a homography transformation to derive dense flow, which is then used for the self-supervised loss. The full network was running on the neuromorphic processor during the real-world flight tests.

framework (57, 58). The idea behind this approach is that, by compensating for the spatiotemporal misalignments among the events triggered by a moving edge (event deblurring), one can retrieve accurate optical flow information. In this work, we used the formulation proposed in (24) and shown in Fig. 2. Corner ROI events within nonoverlapping temporal windows of 5 ms were processed sequentially by our spiking networks, which provide optical flow estimates at every time step. Only during training, we used the motion information of the four corner ROIs to parameterize a homography transformation that, under the assumption of static planar surface, allowed us to retrieve dense optical flow, as in (55, 59–61). Following (24), we accumulated event and optical flow tuples over multiple time steps for contrast maximization to be a robust self-supervisory signal and only computed the deblurring loss function and performed a backward pass through the networks (using backpropagation through time) once 25 ms of event data were processed. To cope with the non-differentiable spiking function of our neurons, we used surrogate gradients (34).

On the other hand, the control part of the network, consisting of a linear mapping from the motion of the four corner ROIs to thrust and attitude control commands, was trained in a drone simulator using an evolutionary algorithm. Evolutionary algorithms work by evaluating all the individuals in a population, where the best-performing (or fittest) individuals are varied upon to form the population of the next generation. Over generations, the individuals will get an increasingly high fitness, which in our case means that they became better at ego-motion control. Figure 3 gives an overview of the simulator setup as was used in evolution. To avoid the need to incorporate an event-based vision pipeline in simulation, we used the ground-truth state of the simulated drone to generate the expected flows per corner ROI using the continuous homography transform (62) and used these to construct the unscaled velocity (velocity divided by height above ground) and yaw rate estimates

that make up the visual observables of the camera's ego-motion. The velocity was divided by height because optical flow vectors capture the ratio of velocity and distance (51, 56). The inputs to the linear control mapping were then these visual observables: absolute roll and pitch (from the drone's inertial measurement unit) and a desired set point for the visual observables. The outputs of the controller (desired collective thrust, pitch and roll angles, and yaw rate) were subsequently applied to the simulated drone model to control it. During evolution, the fitness of a controller was determined on the basis of the accumulated visual observable error in an evaluation. We evaluated each of the individuals in the population on a set of (repeated) ego-motion set points representing horizontal and vertical flight, created offspring through random mutations, and selected the best individuals for the next generation. The trained controller was transferred directly to the real robot, without any retraining.

Because of the split between the vision and control parts of the pipeline, we could evaluate their performance separately. The estimated corner ROI flows of the vision part were compared against ground-truth data obtained from a motion capture system, and the control part was evaluated in simulation. After connecting vision and control together, we then demonstrated the performance of our fully neuromorphic vision-to-control pipeline through real-world flight tests. To further illustrate the robustness of our vision-based state estimation, we performed real-world tests with changing ego-motion set points and tests in various lighting conditions. Last, we compared energy consumption against possible onboard GPU (graphics processing unit) solutions.

Vision-based state estimation
To prevent reality-gap issues when simulating an event-based camera, we trained and evaluated the vision part of our pipeline using real-world event sequences recorded with the same platform (drone
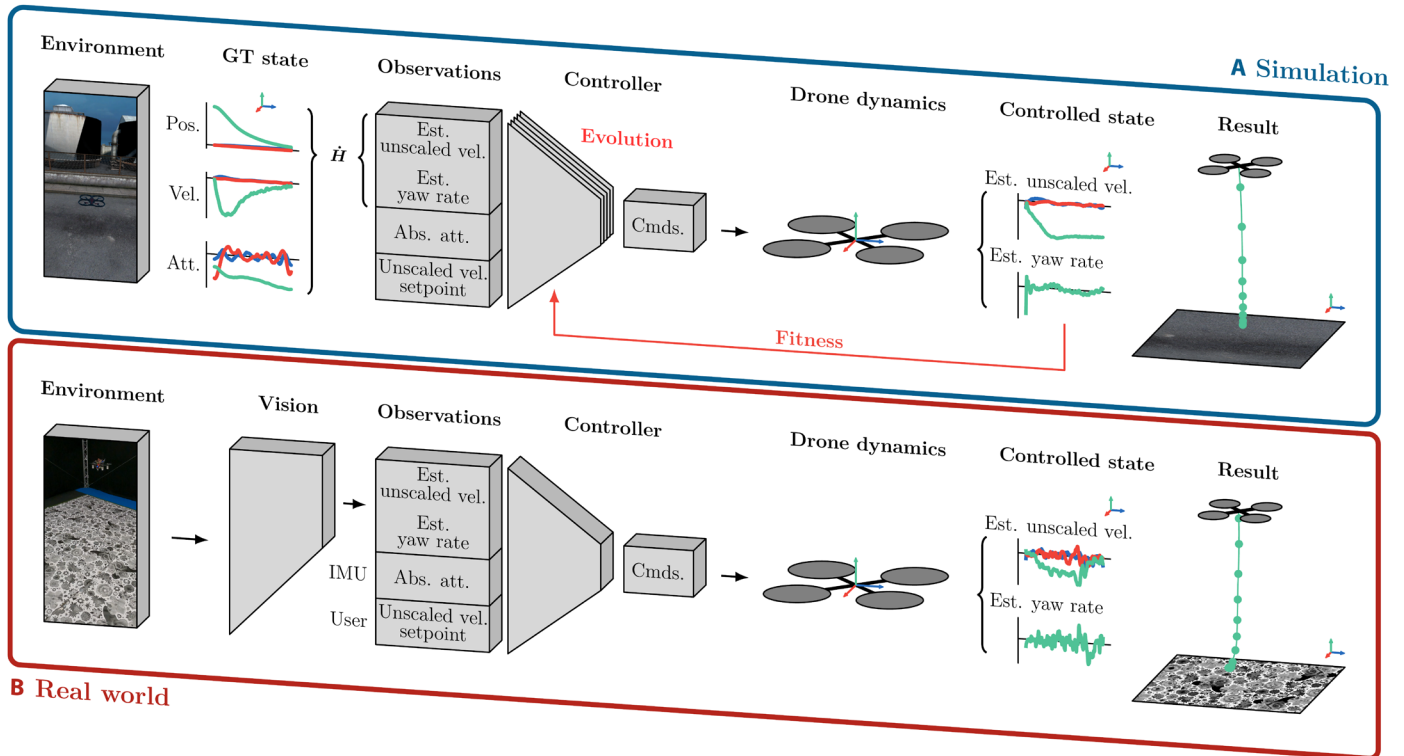
Fig. 3. Overview of the control pipeline for simulation and real-world tests. During training in simulation (**A**), we constructed visual observables (unscaled velocities and yaw rate) from ground truth using the continuous homography transform. The control decoding takes these observables together with roll and pitch and an ego-motion set point to output commands, which control the drone dynamics. We have trained the controller using evolution on the basis of a fitness signal that quantifies how well the controller can follow ego-motion set points for horizontal and vertical flight. In the real world (**B**), we receive flows of the corner ROI from the vision network and transform these to visual observables and control commands in a single matrix multiplication to send low-level control commands thrust and attitude to the autopilot.

and downward-facing event-based camera) and in the same indoor environment (static and planar, constant illumination). This dataset (available in the Supplementary Materials) consists of approximately 40 min of event data, which we split into 25 min for training and 15 min for evaluation, and its motion distributions are shown in Fig. 4A. In addition to the visual data, the ground-truth pose (position and attitude) of the drone over time was provided at a rate of 180 Hz and was used solely for evaluation. Examples of this ground truth, which can be converted to dense optical flow using the camera calibration, are shown in Fig. 4A alongside the floor texture of the indoor environment. Furthermore, to demonstrate that using the same texture for training and evaluation does not lead to overfitting (and resulting degraded control performance), we performed additional experiments with different types of texture that can be found in the Supplementary Materials.

We trained our vision SNN with the self-supervised contrast maximization framework from (*24*) and a quantization-aware training routine that simulates the neuron and synapse models in the target neuromorphic hardware. Afterward, we evaluated the performance of our spiking network on the task of planar event-based optical flow estimation using sequences with varying amounts of motion. Qualitative results are presented in Fig. 4B, where the estimated visual observables (unscaled velocities and yaw rate, constructed from the estimated optical flow vectors at the image corner ROIs) are compared with their ground-truth counterparts. These results confirm the validity of our approach. Despite the architectural limitations of

the proposed solution (for instance, lower resolution because of binary activations, limited field of view, only self-recurrency, weight, and state quantization) and that it does not have access to ground-truth information during training, it was able to produce optical flow estimates that accurately capture the motion encoded in the input event stream (the ego-motion of the camera), even for the fast rotation of approximately 4 rad/s at the end of the shown sequence. Similarly to any other optical flow–based state-estimation solution, our SNN is subject to the aperture problem not only because of the limited receptive field of the corner ROIs but also because of the use of event cameras as vision sensors (*18*).

In Fig. 4C, we show the internal spiking activity of our vision SNN as it processes the top-left corner ROI from the fast sequence shown in Fig. 4B, along with the decoded optical flow vectors. These qualitative results provide insight into the type of processing carried out by the proposed architecture, which is spike-based and therefore sparse and asynchronous. Despite the rapid motion in the input sequence, all layers of the SNN maintained activation levels below 50% of the available neurons. The network was not explicitly trained to promote sparse activations but could perhaps be rendered even more energy efficient by including sparsity measures in the loss function. Furthermore, we can distinguish layers with activity levels that are highly correlated with the input activity (encoder 1 and pooling) and layers that rely on their explicit recurrent connections to maintain activity levels that are relatively independent of the input statistics (encoder 2 and encoder 3). This observation could be
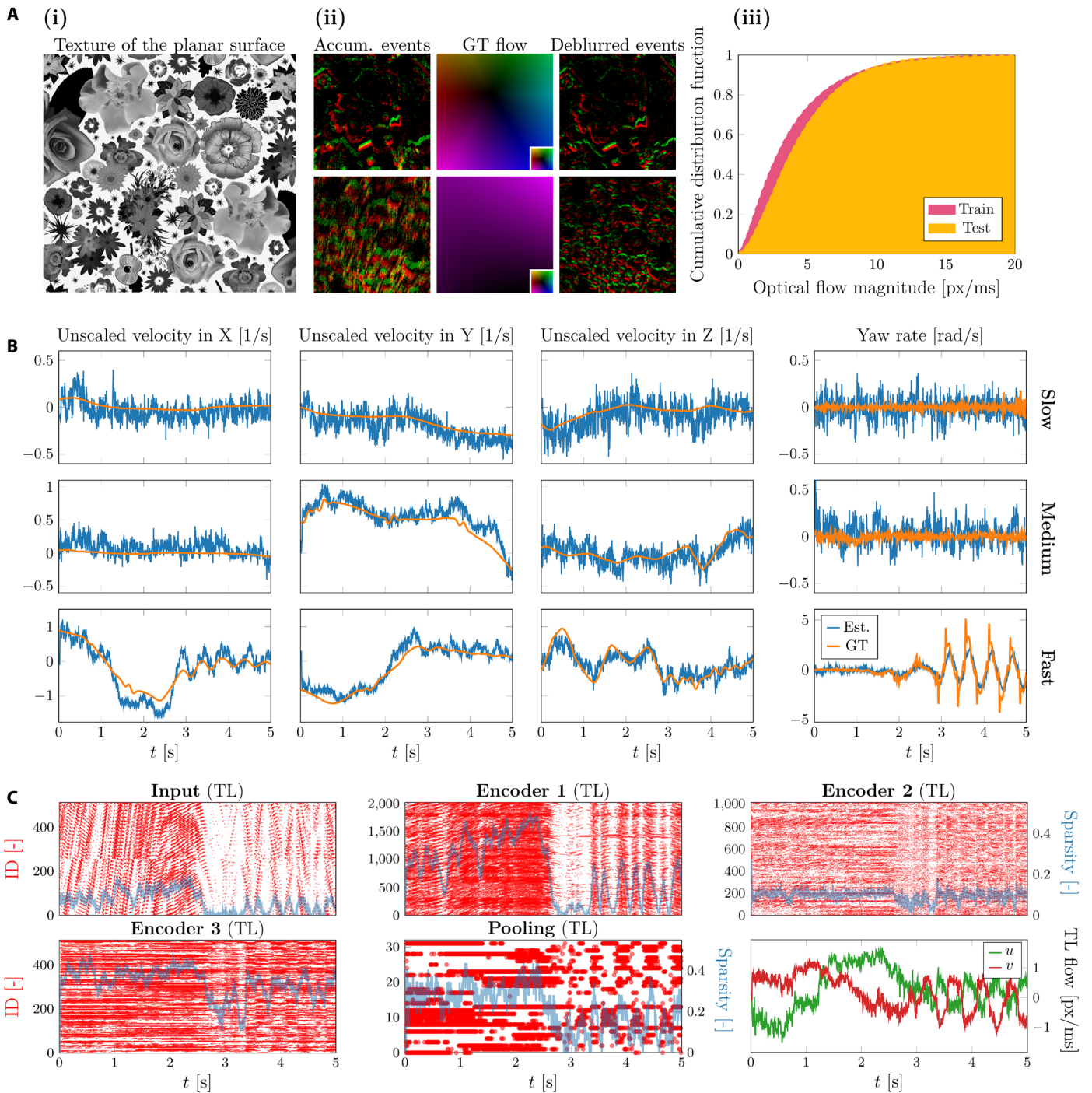
Fig. 4. Overview of results for the vision-based state estimation. (**A**) Characteristics of the dataset for estimating planar optical flow. (i) Grayscale flower texture used to cover the floor. (ii) Accumulated event windows showing the blur arising from motion, ground-truth flow fields as determined with a motion tracking system and based on a flat floor assumption (only for evaluation) and the result when using the flow fields to deblur the event windows (only for illustration). (iii) Ground-truth optical flow distributions for the training and test datasets. (**B**) Comparison of estimated and ground-truth visual observables for sequences with different motion speeds (slow, medium, and fast). (**C**) Network activity resulting from the events in the top-left corner ROI in the fast motion sequence.

the starting point for a future investigation into how this type of network determines optical flow [as in (63)].

In Table 1, we provide a quantitative comparison of our solution with other similar recurrent architectures (that are not compatible or do not fit in the Intel Kapoho Bay) on the basis of the average endpoint error (EPE; the Euclidean distance between predicted and ground-truth optical flow vectors averaged over all pixels in the image). This evaluation not only demonstrates the accuracy of our spiking network but also assesses the effect of each mechanism that was incorporated into the pipeline to achieve a solution that could be deployed on Loihi at the target frequency of 200 Hz (downsampling the image space, only processing the events from the corner ROIs, limiting the maximum number of processed events per corner, and adding quantization; see also Supplementary Methods). Several observations could be made. First, the ANN outperformed its spiking variants by a large margin. This is likely because of the higher resolution of the floating point activations and the direct relation between the backpropagation gradient and the output error. Second, self-recurrency (Self-RNN) was the weakest form of explicit recurrency among those tested. The other networks seemed to successfully exploit their capability of representing more complex temporal functions. Third, deploying one architecture to each image corner ROI instead of processing the entire image space at once was beneficial for our architecture (lowering the EPE from 9.80 to 8.22) but had a slight detrimental effect on the baselines. This may be due to the dataset containing ample texture on the floor. A dataset with much less texture might have shown a different result. Fourth, limiting the number of events that can be processed at once to 90 per corner ROI was also helpful for the evaluated SNNs because it helped reduce the internal activity levels. Last, the incorporation of the Loihi-specific weight and state quantization led to an error increase for our architecture.

Control through visual observables: From sim to real

Separately from the vision part, we trained and evaluated the control part of our pipeline. Because of limitations in the hardware setup (see Materials and Methods), this is a linear mapping from a visual observable estimate, absolute roll and pitch, and a visual observable set point to thrust and attitude commands. The visual observable estimate is made up of unscaled velocity $\hat{v}^{\mathcal{B}}$ and yaw rate $\hat{\omega}_z^{\mathcal{B}}$; the visual observable set point consists of the corresponding set points in unscaled velocity $v_{sp}^{\mathcal{B}}$ and yaw rate $\omega_{z,sp}^{\mathcal{B}}$. A population of these linear mappings was evolved in simulation for a set of 16 unscaled velocity and yaw rate set points. Each unscaled velocity set point $v_{sp}^{\mathcal{B}}$ had at most one nonzero element $\in \{ \pm 0.2, \pm 0.5, \pm 1.0 \}$ 1/s. In other words, they represented hover, vertical flight in the form of landing at three speeds (no ascending flight), and horizontal flight in four directions at three speeds. Unless mentioned otherwise, the set point for yaw rate $\omega_{z,sp}^{\mathcal{B}} = 0$. Figure 5 shows the performance of the evolved linear network controller in simulation in terms of the estimated unscaled velocities $\hat{v}^{\mathcal{B}}$ (Fig. 5A) and the world position $\mathbf{p}^{\mathcal{WB}}$ over time (Fig. 5B) for all set points. The control performance was satisfactory; the controller reached the set point in all cases and was capable of keeping the unscaled velocities for the nonflight direction close to zero. Especially for $v_{*,sp}^{\mathcal{B}} = \pm 1.0$ 1/s, some overshoot was witnessed, but this could be expected given that this is a linear mapping without any kind of derivative control.

Figure 5 furthermore shows the results obtained by deploying this controller in the real world and replacing the ground-truth visual observables with those estimated by the vision network (also see movie S3). Overall, the results demonstrated successful deployment of the fully neuromorphic vision-to-control pipeline. Looking at the unscaled velocity plots for the different set points, we see that these became less noisy for higher set points and faster flight, as can also be seen from the 3D position plots. This is due to the signal-to-noise ratio of the vision-based state estimation increasing with motion magnitude (little motion means that most events are due to noise, as can be seen in Fig. 4). In addition, the inertia of the drone provided some stability at higher speeds. Nevertheless, apart from several set points (for example, landings, $v_{\{x,y\},sp}^{\mathcal{B}} = \pm 0.2$ 1/s), the controller was not always able to reach the desired set point: The steady-state error looked to be proportional to the set point magnitude. This could be attributed to the fact that although the controller is a linear mapping, the relationship between attitude angle and resulting forward/sideways velocity is nonlinear (especially at larger attitude angles) and additionally affected by drag. Providing absolute attitude input to the network and simulating the drag [as in (64)] during training turned out not to be enough to compensate, and small errors were accumulated over time. Furthermore, there can be mismatches between the dynamics of the simulated drone (body characteristics and motor dynamics) with which the controller was trained and the real drone on which the flight tests were performed, although we abstracted the control outputs to attitude commands. Last, inaccuracies of the drag model can also be a source of error (in this case, it seems that drag was higher in reality than in simulation). A linear or proportional controller, such as we have here, cannot integrate all these small errors into an additional

Table 1. Quantitative comparison between different architectures. The value in the bottom right corner (asterisk) indicates the final architecture. Row-wise architecture choices and column-wise design decisions affected test accuracy in average EPE (Euclidean distance between predicted and ground-truth vectors, averaged over all pixels in the image; lower is better) in pixels per input window (5 ms). Reported EPEs were averaged over the test sequences and based on a single training run. Baseline architectures feature the same feedforward connectivity pattern as ours but vary the neuron model and the type of recurrent connections. Values in the table should be multiplied by 10^{-2}.

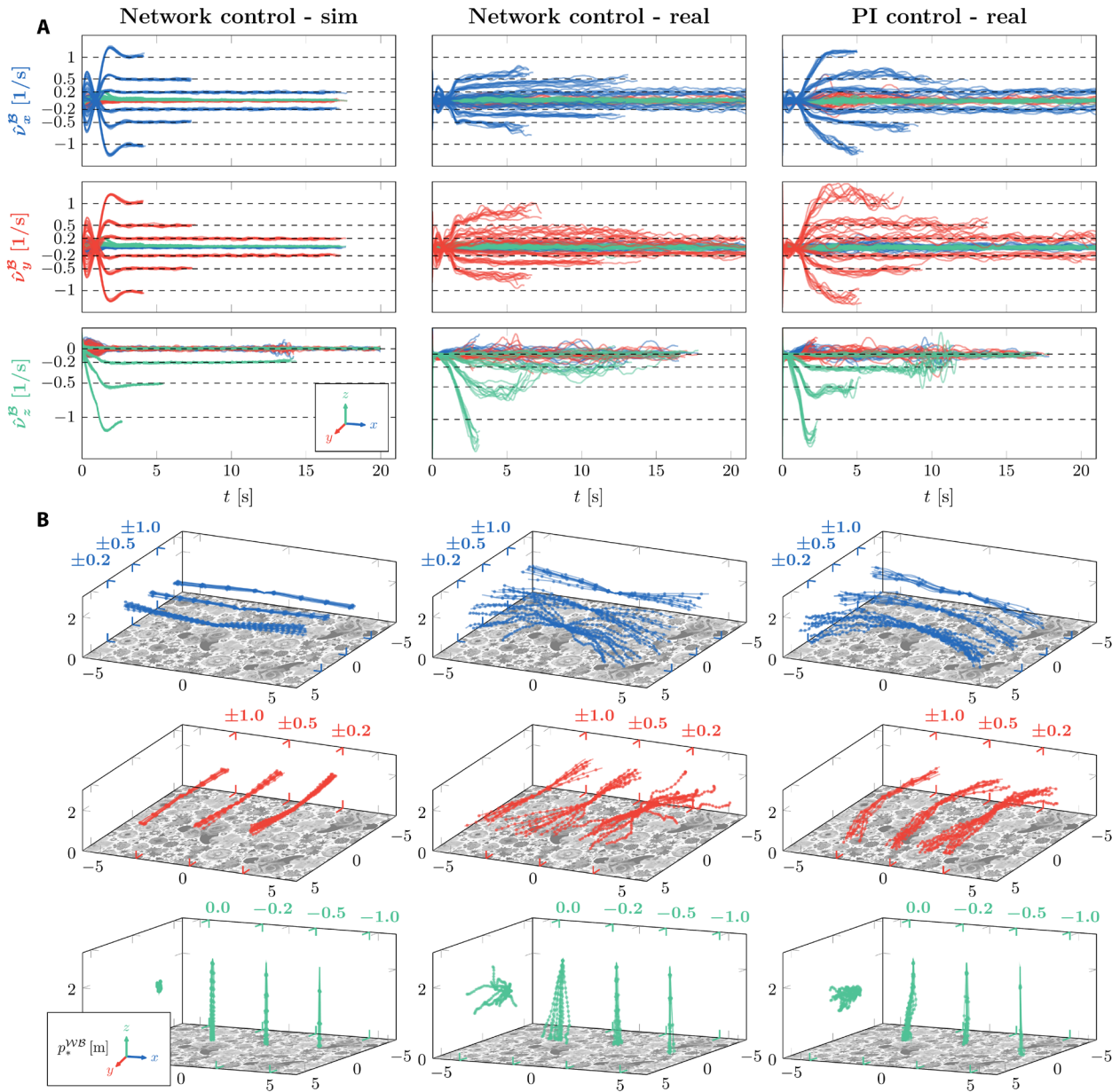|	Full image	+2× Down.	+Corner ROI crop	+Limit events	+Loihi quant.
Conv-GRU ANN	5.88	5.56	5.61	5.72	–
Conv-RNN SNN	7.92	7.89	7.91	6.97	6.96
Self-RNN SNN (ours)	10.79	9.80	8.22	7.71	8.34*

Fig. 5. Comparison of results obtained in simulation and during real-world flight tests. (**A**) Estimated unscaled velocities $\hat{v}^B$ for 16 different set points in three axes across three scenarios: linear network controller in simulation and the real world and a hand-tuned PI controller in the real world. Real-world tests used the vision network to obtain visual observable estimates. Set points were nonzero in one direction, indicated with dashed lines. Rows represent the different motion axes (*X*, *Y*, and *Z*). (**B**) 3D world position trajectories $\boldsymbol{p}^{WB}$ for the same flight tests. Each plot in (B) matches the plot in the corresponding location in (A).

control signal that compensates for them, leaving a steady-state error. The resulting drift in position and yaw for all runs in Fig. 5 were quantified in Supplementary Results.

Last, Fig. 5 shows the results obtained by connecting a hand-tuned proportional-integral (PI) controller to the vision-based state estimation (also see movie S4). We compared this with the linear network controller. Looking at all directions and set points, we see that the PI controller reaches the set point faster than the network controller. For horizontal flight, the network controller was not at all able to reach the set point $v^B_{\{x,y\},sp} = \pm 1.0$ 1/s and only just in the

case of ± 0.5 1/s, supposedly because of the limitations of linear control. The PI controller did not have this problem because it could increment its control command to eliminate the steady-state error. For vertical flight, both the network and the PI controller had substantial overshoot for $v^B_{z,sp} = -1.0$ 1/s. This had to do with the fact that at such speeds from such heights (2.5 m), the drone barely reached the set point before reaching the ground and therefore had little time to compensate for any overshoot (see Fig. 5A, PI controller for $v^B_{z,sp} = -0.5$ 1/s, for which overshoot was similar but was corrected shortly after).

Flickering, darkness, and squares: Examples of versatility and robustness

Although our main goal is to show a functional fully neuromorphic vision-to-control pipeline, for a broader applicability of this technology, it is informative to study the pipeline's robustness and versatility to conditions not encountered during learning. Figure 6 shows the results for these tests for the combination of the vision-based state estimation and linear network controller. Figure 6A displays the user alternating through different unscaled velocity set points in

X and Y (while keeping yaw constant) to let the drone fly a square. Apart from one section, the controller was able to reach the desired set point quite quickly, allowing for sharp corners, while keeping yaw drift small (0.06 rad).

Last, we have tested the robustness to lighting conditions for maneuvers in different directions. Here, we show the results of these experiments for the landing maneuver (see Supplementary Results for the other directions). Because landing involves a wider range of visual motion magnitudes than horizontal flight (flow is
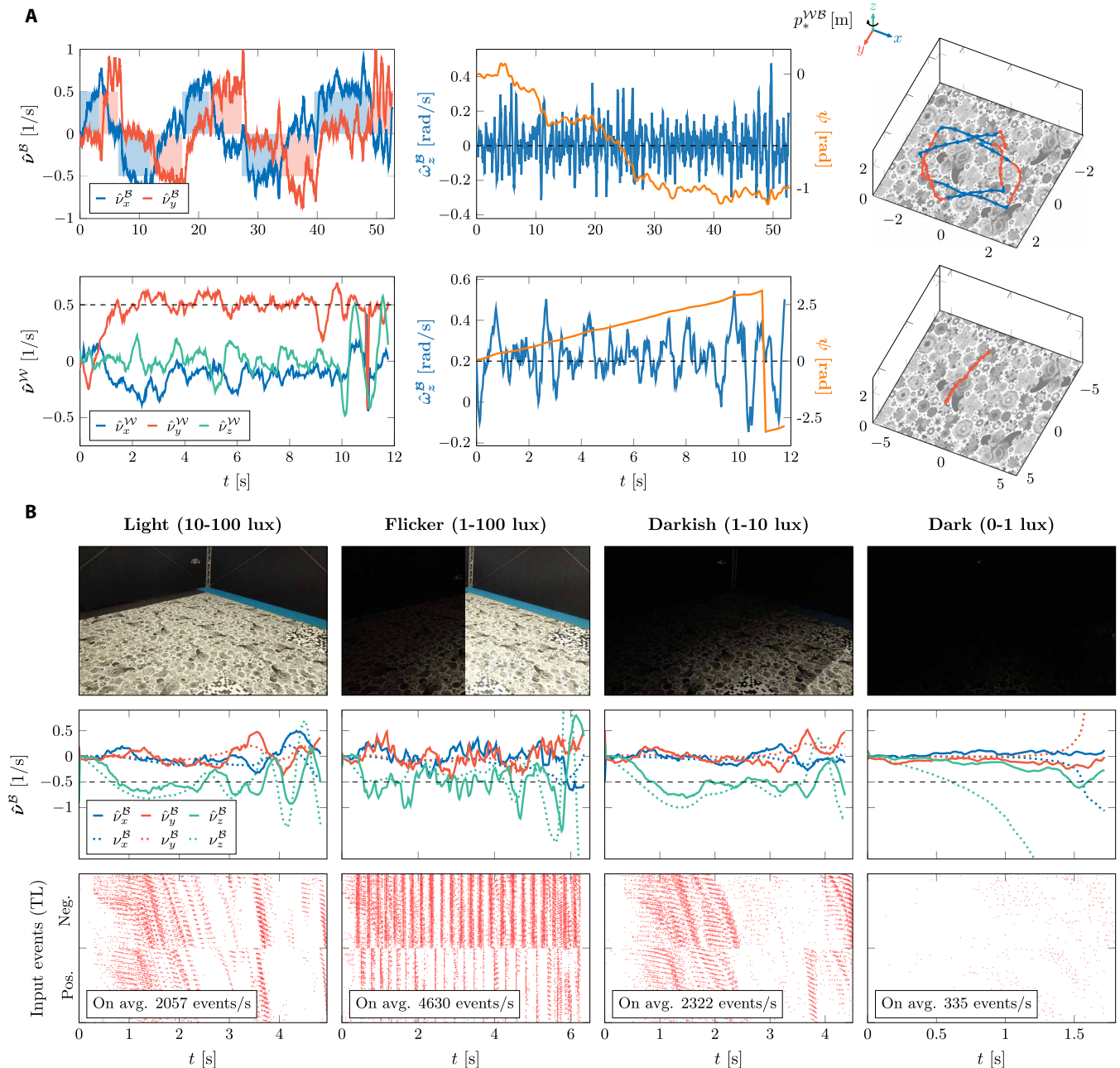

Fig. 6. Additional results with vision network and linear network controller. (**A**) By alternating set points in X and Y, we could fly a square. Plots show the estimated unscaled velocity $\hat{v}^B$ (with set points shaded in blue and red), estimated yaw rate $\hat{\omega}_{z}^{B}$, and position p^{WB}. (**B**) Landing experiments with different lighting conditions. Plots show the estimated and ground-truth unscaled velocity $\hat{v}^B$ and v^B, as well as the events received by the network in the top-left ROI. Although flickering lights led to many more events, visual observable estimates (and hence control) only diverged when it was so dark that there were almost no events.

inversely proportional to height), it allowed us to better see the effect of lower-light conditions. Furthermore, divergence-based landings inherently lead to oscillations (*49*), which makes for a more challenging scenario in the case of flickering lights. Figure 6B shows landing with divergence $\nu_{z,sp}^{B} = -0.5$ 1/s for various lighting conditions (quantified with lux measurements). With these experiments, we aimed to investigate the robustness of the approach to wildly varying event statistics. For instance, in a darker environment, contrasts are less visible, which means that motion will generate many fewer events and that there will be more spurious, noisy events—similar to our own human vision when we walk in the dark. When lights are switched on and off, this generates massive numbers of events that are unrelated to motion, hence violating the brightness constancy assumption underlying optical flow determination. The events for the top left corner ROI are shown in Fig. 6B. The results in the light and darkish settings look alike, but flickering lights led to a large increase in events, and the darkest setting gave almost no events. As Fig. 6B shows, despite the challenging light conditions, the controller was able to track the set point (black dashed line) well, and the estimated unscaled velocities approximated their ground truths. Only the darkest setting posed a real problem for the state estimation: In that case, the estimated unscaled velocities $\hat{\nu}^{B}$ diverged too much from the ground-truth unscaled velocities ν^{B} to perform a successful landing. For comparison, the flickering, darkness, and squares experiments have also been performed with the PI controller, which can be found in Supplementary Results.

Improved inference speed and energy consumption on neuromorphic hardware

Table 2 shows a comparison in terms of power/energy and run time between the Loihi neuromorphic processor and an NVIDIA Jetson Nano for running the vision network (both SNN and equivalent ANN) on sequences with varying amounts of motion and hence varying input event density. The SNN ran in hardware on Loihi and in software (PyTorch) on Jetson Nano. The tests for Loihi were performed on a Nahuku board, which contains 32 Loihi chips. We confirmed, insofar possible, that using two chips on Nahuku is representative of a Kapoho Bay (at least in terms of execution time), which is the two-chip form factor used on the drone. Still, neither of these benchmarks is completely representative of the tests performed in the real world: The benchmarks used data already loaded in memory and therefore only quantified the processing by the network without any bottlenecks because of input/output (I/O) and preprocessing, whereas the flight tests involved streaming event data that were coming in and being processed in an online fashion. This showed in Loihi's execution frequencies in Table 2, which were well above the 200 inferences (with one inference being the processing of one set of inputs by the network) per second (inf/s) achieved during flight tests.

Table 2 shows the power when the processors were idle and when they were running the networks. Because Jetson Nano does not provide static and dynamic power components, we compared the difference between idle and running power and used that to compute energy per inference. A main observation is that the Nahuku 32 board consumed 0.95 W when running the SNN, whereas Jetson

Table 2. Approximate energy and power characteristics for various devices on three sequences: slow, medium, and fast. On average, slow has 28.6 events/inf, medium has 106.9 events/inf, and fast has 186.6 events/inf. Delta power is the difference between idle and running (total) power and was used to compute energy per inference. Dynamic power is the power needed for switching and short-circuiting, and static power is due to leakage; together, they sum to running (total) power as well. Nahuku is a board with 32 Loihi chips (Kapoho Bay has 2). A Nahuku configuration where no spikes were sent and only chips and cores were allocated (no synapses) is included as "empty." Jetson Nano has a low-power (5 W) and high-power (10 W) mode. One inference (inf) was the processing of one set of inputs by the network, resulting in an output or prediction. On Jetson Nano, we tested both our vision SNN and the comparable Conv-GRU ANN. Dash entries indicate unmeasured configurations.

Device	Seq.	Static (W)	Dynamic (W)	Idle (W)	Running (W)	Delta (W)	inf/s	mJ/inf
Nahuku 32 (empty)	Any	0.86	0.04	0.90	0.90	4×10^{-3}	60,496	71×10^{-6}
3*Nahuku 32: SNN	Slow	0.90	0.05	0.94	0.95	12×10^{-3}	1,637	7×10^{-3}
	Medium	0.90	0.04	0.94	0.95	8×10^{-3}	411	21×10^{-3}
	Fast	0.90	0.04	0.94	0.95	7×10^{-3}	274	27×10^{-3}
3*Jetson Nano: SNN (5 W)	Slow	–	–	1.05	2.23	1.18	14	86.11
	Medium	–	–	1.03	2.25	1.22	14	85.58
	Fast	–	–	1.03	2.24	1.21	14	86.19
3*Jetson Nano: SNN (10 W)	Slow	–	–	1.04	2.98	1.93	26	75.25
	Medium	–	–	1.06	2.98	1.92	26	75.35
	Fast	–	–	1.04	2.99	1.95	25	76.52
3*Jetson Nano: ANN (5 W)	Slow	–	–	1.05	2.66	1.61	59	27.46
	Medium	–	–	1.07	2.64	1.57	56	27.91
	Fast	–	–	1.06	2.64	1.58	57	27.52
3*Jetson Nano: ANN (10 W)	Slow	–	–	1.04	3.30	2.27	83	27.36
	Medium	–	–	1.07	3.33	2.26	80	28.09
	Fast	–	–	1.07	3.30	2.23	80	27.80

Nano consumed ~2.98 W when running in 10-W mode. This is a difference of a factor ~3.1×. Most of the energy expenditure of Loihi consisted of idle power. Table 2 also shows the difference between the idle power and the power when running the network ("Delta"). When only considering this extra power required for the inference of the network, Loihi outperformed Jetson Nano by three to four orders of magnitude, depending on the sequence. Hence, any possible reduction of the idle power would substantially affect the neuromorphic chip's energy efficiency advantages over other chips.

Moreover, Loihi provided a one- to two-order-of-magnitude improvement in execution frequency. Furthermore, the benefits of neuromorphic processing show in Loihi's increasing execution frequency as event sparsity increased (from fast to slow motion sequences). Because a GPU like Jetson Nano is not optimized to simulate SNNs, we also ran an equivalent ANN (Conv-GRU with downsampling, corner crop ROI, and event limiting from Table 1). The increased inference speed for the ANN on Jetson Nano confirms that this is a more suitable architecture for GPUs. Nonetheless, although energy consumption per inference has decreased compared with running the SNN on Jetson Nano, energy efficiency and inference speed still did not come close to those of the SNN on Loihi.

DISCUSSION

We presented a fully neuromorphic vision-to-control pipeline for controlling a flying drone. Specifically, we trained an SNN that takes in raw event-based camera data and produces low-level control commands. Real-world experiments demonstrated a successful sim-to-real transfer: The drone could accurately follow various ego-motion set points, performing hovering, landing, and lateral maneuvers—even under constant yaw rate.

Our study confirms the potential of a fully neuromorphic vision-to-control pipeline by running on board with an execution frequency of 200 Hz. Moreover, the chip spends 0.95 W, out of which 0.94 W was idle power and only 7 to 12 mW were used for inference. Hence, reducing the idle power can lead to substantial energy gains.

A major question is whether the energy gain of neuromorphic processing will make a difference on a system level even if future neuromorphic chips weigh on the order of grams and will have a negligible idle power. Compared with the power required for hovering, 277 W, a difference of a few watts (see Table 2) may seem like a small difference. However, on flying robots, such small differences can have substantial effects (65). A heavier, more power-consuming computing unit not only requires more power from the battery but also needs to be lifted in the air. This requires a bigger battery and possibly bigger motors that themselves also have to be lifted. Moreover, as argued in (65), a lower latency, as attained with neuromorphic processing, allows for faster flight. This in turn leads to drones accomplishing their missions with less flight time. Still, the main point is not necessarily what is gained on a ~1-kg drone if switched from a conventional embedded GPU to a lightweight neuromorphic processor (which is not negligible) but that neuromorphic processing will enable many more networks to run on such larger drones and even enable deep neural networks on much lighter platforms that cannot even carry an embedded GPU. An example of the latter are 30-g flapping wing drones, which use ~6 W to fly (66).

A particularly promising avenue to autonomous flight of such tiny drones is to make the entire drone sensing, processing, and actuation pipeline neuromorphic, from its accelerometer sensors to the processor and motors, allowing for sparse and event-driven/asynchronous computation all the way through. Because such hardware is currently not available, we have limited ourselves to the vision-to-control pipeline, ending at thrust and attitude commands.

Until then, advancements could come from improved I/O bandwidth and interfacing options for the neuromorphic processor and event camera. The current processor is limited to host boards with an x86 architecture (preventing other potentially lighter but equally performant architectures from being used) and can only be connected directly via AER (address-event representation) to a specific model of event-based camera (this is of course also a limitation on the part of the available event cameras). Although the former is limiting for all works implementing neuromorphic hardware on constrained systems like drones, the latter is especially relevant to our advanced use case, where we reached the limits of the number of spikes that can be sent to and received from the neuromorphic processor at the desired high execution frequency of 200 Hz. To achieve this frequency, we had to limit the number of events per input window to 90 per image corner ROI and limit ourselves to a linear network controller, which avoids having to send additional input spikes that encode the ego-motion set point and attitude. Improved interfacing could allow for more extensive vision processing and more complex spiking neuron control networks. This could make the vision-based ego-motion estimation more robust and considerably increase the control performance—even if SNN controllers currently perform worse than common PID controllers (67, 68). As long as both the vision and control networks still use a learned encoding and decoding, the proposed split-and-merge strategy would still work (see Materials and Methods). Ultimately, further gains in terms of efficiency might be obtained by moving from digital neuromorphic processors to mixed-signal hardware, but this will pose even larger development and deployment challenges given the noise sensitivity of such hardware (17, 69).

Despite the abovementioned limitations, the current work presents a substantial step toward neuromorphic sensing and processing for drones. The results are encouraging because they show that neuromorphic sensing and processing may bring deep neural networks within reach of small autonomous robots. In time, this may allow them to approach the agility, versatility, and robustness of small animals such as flying insects.

MATERIALS AND METHODS

Here, we explain the main components of the proposed fully neuromorphic vision-to-control pipeline, starting with the neuron model of our SNN and how this was trained in a self-supervised fashion using real event camera data. Next, we describe how the vision-based state estimate can be used for navigation and how we trained a controller on top of it. Last, we discuss the real-world tests and hardware and the performed energy benchmarks.

Neuromorphic state estimation: Spiking, sequential processing

We used a spiking neuron model based on the current-based leaky-integrate-and-fire (CUBA-LIF) neuron, whose membrane potential U and synaptic input current I at time step t can be written as

$$U_i^t = \tau_U \left(1 - S_i^{t-1} \right) U_i^{t-1} + I_i^t \qquad (1)$$

$$I_i^t = \tau_I I_i^{t-1} + \sum_j w_{ij}^{ff} S_j^t + w_{ii}^{rec} S_i^{t-1} \qquad (2)$$

where j and i denote presynaptic (input) and postsynaptic (output) neurons within a layer, $S \in \{0,1\}$ denotes a neuron spike, and w^{ff} and w^{rec} denote feedforward and self-recurrent connections (if any), respectively. The decays (or leaks) of the two internal state variables of this neuron model are learned and are denoted by τ_U and τ_I. A neuron fires an output spike if the membrane potential exceeds a threshold θ, which is also learned. The firing of a spike triggers a hard reset of the membrane potential. Note that, in this work, all neurons within a layer share the same decays and firing threshold.

Neurons on the Loihi neuromorphic processor also follow the CUBA-LIF model (40); however, several considerations must be taken into account to accurately simulate these on-chip neurons. First, the state variables are quantized in the integer domain. Hence, the parameters associated with these variables are also quantized in the same way: $w \in [-256 \ .. \ 256 - \Delta w]$, with Δw being the quantization step for the synaptic weights; $\tau_{\{U, I\}} \in [0 \ .. \ 4096]$ for the decays; and $\theta \in [0 \ .. \ 131071]$ for the threshold. We follow this quantization scheme with $\Delta w = 8$ (6-bit weights) in the simulation and training of our neural networks. Second, to emulate the arithmetic left (bit) shift operations carried out by the processor when updating the neuron states, we performed a rounding-toward-zero operation after the application of the decays. Taking these aspects into consideration, we obtained a matching score of 100% between the simulated and the on-chip spiking neurons. We used quantization-aware training (quantized forward pass and floating point backward pass) to minimize the performance loss of our SNN when deployed on Loihi. As a surrogate gradient for the spiking function σ, we opt for the derivative of the inverse tangent (37)

$$\sigma'(x) = \text{aTan}' = 1 / \left(1 + \gamma x^2\right) \qquad (3)$$

$$x = u - \theta \qquad (4)$$

with $\gamma = 10$ being the surrogate width.

Neuromorphic state estimation: Planar homography

Assuming that $\tilde{x} = \left[x^T, 1\right]^T$ and $\tilde{x}' = \left[x'^T, 1\right]^T$ are two undistorted corresponding points from a planar scene with pixel array coordinates $x = [x, y]^T$ and $x' = [x', y']^T$ expressed in homogeneous coordinates and captured by a pinhole camera at different time instances, a planar homography transformation is a linear projective transformation that maps $\tilde{x} \leftrightarrow \tilde{x}'$ such that

$$\lambda \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = H \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}; \quad \text{with } H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \qquad (5)$$

where H is a three-by-three nonsingular matrix, further referred to as the homography matrix, which is characterized by eight degrees of freedom and is defined up to a scale factor λ and from which we obtain the normalized form by setting $h_{33} = 1$.

From Eq. 5, we can formulate a system of linear equations for the kth point correspondence

$$A_k h = b_k \qquad (6)$$

$$A_k = \begin{bmatrix} x & y & 1 & 0 & 0 & 1 & -x'x & -x'y \\ 0 & 0 & 0 & x & y & 1 & -y'x & -y'y \end{bmatrix} \qquad (7)$$

$$h = \begin{bmatrix} h_{11} & h_{12} & h_{13} & h_{21} & h_{22} & h_{23} & h_{31} & h_{32} \end{bmatrix}^T \qquad (8)$$

$$b_k = \begin{bmatrix} x' & y' \end{bmatrix} T \qquad (9)$$

As shown in Fig. 2A, our vision network predicts the displacement of the corner ROI pixels in a certain time window. Using this information, we can solve for the components of the homography matrix through

$$h = A^{-1} b \qquad (10)$$

with A and b being the result of the concatenation of the individual A_k and b_k of each point correspondence $\forall k \in \{\text{TL, TR, BR, BL}\}$, resulting in a determined system of equations. This approach is referred to as the four-point parametrization of the homography transformation (55), and it has proven to be successful in the event camera literature for robotics applications (61, 70).

Once the homography matrix is estimated, we can estimate a dense (per-pixel) optical flow map as follows

$$u(x, H) = \begin{bmatrix} u(x, H) \\ v(x, H) \end{bmatrix} = \left(H \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \right)_{\text{Eucl}} - \begin{bmatrix} x \\ y \end{bmatrix} \qquad (11)$$

which encodes the displacement of pixel x in the time window of H. $(\cdot)_{\text{Eucl}}$ indicates the conversion from homogeneous to Euclidean coordinates.

Neuromorphic state estimation: Self-supervised learning

To train our spiking architecture to estimate the displacement of the pixels of the four corner ROIs in a self-supervised fashion, we used the contrast maximization framework for motion compensation (57, 58). Assuming constant illumination, accurate optical flow information is encoded in the spatiotemporal misalignments among the events triggered by a moving edge (blur). To retrieve it, one has to learn to compensate for this motion (deblur the event partition) by transporting the events through space and time. Once we get a per-pixel optical flow estimate $u(x, H)$ from Eq. 11, we can propagate the events to a reference time t_{ref} through the following linear motion model

$$x_i' = x_i + (t_{\text{ref}} - t_i) u(x_i, H) \qquad (12)$$

where t and t_{ref} are normalized relative to the time window between x and x'. The result of aggregating the propagated events is the image of warped events (IWE) at t_{ref}, and it having a high contrast indicates good motion compensation/deblurring.

As loss function, we used the reformulation from (24) of the focus objective function based on the per-pixel and per-polarity average time stamp of the IWE (20, 71). The lower this metric, the better the event deblurring and hence the more accurate the estimated

optical flow. We generated an image of the per-pixel average time stamp for each polarity p' via bilinear interpolation:

$$T_{p'}\left(\boldsymbol{x};\boldsymbol{u}\,|\,t_{\text{ref}}\right) = \frac{\sum_j{}'\kappa\left(x - x_j'\right)\kappa\left(y - y_j'\right)t_j}{\sum_j{}'\kappa\left(x - x_j'\right)\kappa\left(y - y_j'\right) + \epsilon} \tag{13}$$

$$\kappa(a) = \max(0, 1 - |a|)$$

$$j = \{i \,|\, p_i = p'\}, \quad p' \in \{+, -\}, \quad \epsilon \approx 0$$

Following (24), we first scaled the sum of the squared temporal images resulting from the warping process with the number of pixels with at least one warped event

$$\mathcal{L}_{\text{contrast}}\left(t_{\text{ref}}\right) = \frac{\sum_x T_+\left(\boldsymbol{x};\boldsymbol{u}\,|\,t_{\text{ref}}\right)^2 + T_-\left(\boldsymbol{x};\boldsymbol{u}\,|\,t_{\text{ref}}\right)^2}{\sum_x\left[n(\boldsymbol{x}') > 0\right] + \epsilon} \tag{14}$$

where $n(\boldsymbol{x}')$ denotes a per-pixel event count of the IWE. As in (20, 22, 24), we performed the warping process both in a forward ($t_{\text{ref}}^{\text{fw}}$) and in a backward fashion ($t_{\text{ref}}^{\text{bw}}$) to prevent temporal scaling issues during backpropagation. The total loss used to train our event-based optical flow networks is then given by

$$\mathcal{L}_{\text{contrast}} = \mathcal{L}_{\text{contrast}}\left(t_{\text{ref}}^{\text{fw}}\right) + \mathcal{L}_{\text{contrast}}\left(t_{\text{ref}}^{\text{bw}}\right) \tag{15}$$

$$\mathcal{L}_{\text{flow}} = \mathcal{L}_{\text{contrast}} + \lambda_{\mathcal{L}}\mathcal{L}_{\text{smooth}} \tag{16}$$

where $\mathcal{L}_{\text{smooth}}$ is a Charbonnier smoothness prior (72) applied in the temporal domain to subsequent per-corner-ROI optical flow estimates and $\lambda_{\mathcal{L}}$ is a scalar balancing the effect of the two losses. We empirically set this weight to $\lambda_{\mathcal{L}} = 0.1$.

As discussed in (24, 25), there has to be enough linear blur in the accumulated input event partition for this loss function to be a robust supervisory signal (58, 73). Because we processed the event stream sequentially, with only a few events being considered at each forward pass, we defined the so-called training partition

$$\varepsilon_{k \to k+K}^{\text{train}} \doteq \left\{\left(\varepsilon_i^{\text{inp}}, \boldsymbol{u}_i\right)\right\}_{i=k}^{K} \tag{17}$$

which is a buffer that gets populated every forward pass with the input events and their corresponding optical flow estimates. This is illustrated in Fig. 2A. At training time, we performed a backward pass with the content of the buffer using backpropagation through time once it contained five successive event-flow tuples (25 ms of event data), after which we updated the model parameters, detached its states from the computational graph, and cleared the buffer. The selection of input and training partition lengths represents deliberate design choices (74) made in alignment with our target execution frequency of 200 Hz, the fact that we do not have direct connectivity between the event camera and the neuromorphic processor, and the statistical attributes of our dataset. We used a batch size of 16 and trained until convergence with the Adam optimizer (75) and a learning rate of 1×10^{-4}. We validated the quality of the estimated optical flow against the ground-truth optical flow using the average EPE metric, which is defined as the Euclidean distance between predicted and ground-truth flow values averaged over all pixels in the image

$$\text{EPE} = \frac{1}{N}\sum_{i \in I}^{N}\left\|\boldsymbol{u}_i - \boldsymbol{u}_i^{\text{GT}}\right\| \tag{18}$$

where I is the set of N pixels in the image and u^{GT} is the ground-truth flow.

From a vision-based state estimate to control

The corner ROI flows $\left[\boldsymbol{u}_{\text{TL}}^T, \boldsymbol{u}_{\text{TR}}^T, \boldsymbol{u}_{\text{BR}}^T, \boldsymbol{u}_{\text{BL}}^T\right]^T \in \mathbb{R}^{8\times 1}$ resulting from the vision-based state estimation can be used to control the drone. More specifically, we can transform the flows to visual observable estimates (51), consisting of unscaled velocities $\hat{\boldsymbol{v}}^C \in \mathbb{R}^{3\times 1}$ and yaw rate $\hat{\omega}_z^C$ in the camera frame C, as follows (56)

$$\boldsymbol{u} = \begin{bmatrix} -1 & 0 & x & x \\ 0 & -1 & y & -y \end{bmatrix}\begin{bmatrix} \boldsymbol{v}^C \\ \omega_z^C \end{bmatrix} \tag{19}$$

where $\boldsymbol{u}$ is the optical flow of a world point with pixel array coordinate $\boldsymbol{x} = [x, y]^T$, and where it is assumed that the scene is static and planar, angles in pitch and roll are small, and optical flow is derotated in pitch and roll (meaning the observed flow is only due to translation and yawing). Concatenating Eq. 19 for all four corners of the field of view ($\boldsymbol{u}_k, \boldsymbol{x}_k \,\forall\, k \in \{\text{TL}, \text{TR}, \text{BR}, \text{BL}\}$) allows us to do a least-squares estimation of the unscaled velocities $\hat{v}^C$ and the yaw rate $\hat{\omega}_z^C$, which can then be transformed to the body frame $\mathcal{B}$. To perform control, we can let a user select visual observable set points, $v_{\text{sp}}^{\mathcal{B}}$ and $\omega_{z,\text{sp}}^{\mathcal{B}}$, and use a trained or manually tuned controller to minimize the difference between the estimated visual observables and their set points.

Because Eq. 19 is a linear transformation, it can be "merged" with other transformations if these are also linear. This holds for the decoding from spikes to corner ROI flows in the vision SNN, meaning that we can use a single linear transformation from spikes to control commands if we use a linear controller. In a similar fashion, we can use this idea to connect separately trained SNNs, merging their linear decodings and encodings. If both are implemented on neuromorphic hardware, this would mean that no off-chip transfer is necessary. Figure 7 illustrates these concepts.

Training control in simulation

We performed control by linearly transforming the visual observable estimates $\hat{v}^{\mathcal{B}} \in \mathbb{R}^{3\times 1}$ and $\hat{\omega}_z^{\mathcal{B}}$, the drone's absolute roll $|\phi|$ and pitch $|\theta|$, and the unscaled velocity set point $v_{\text{sp}}^{\mathcal{B}} \in \mathbb{R}^{3\times 1}$ to a control command $\boldsymbol{c} \in \mathbb{R}^{4\times 1}$, which consists of an upward, mass-normalized collective thrust offset from hover $\bar{f}_{0,c}$ in the body frame $\mathcal{B}$, a roll angle ϕ_c and pitch angle θ_c, and a yaw rate $\omega_{z,c}^{\mathcal{B}}$, to reach a certain set point of unscaled velocities $v_{\text{sp}}^{\mathcal{B}}$ and yaw rate $\omega_{z,\text{sp}}^{\mathcal{B}}$ (always 0). The control part was trained separately from the vision part because of the cost of accurately simulating event-based camera inputs (this needs subpixel displacements between frames, hence high frame rate for fast motion). Simulation was done with a modified version of the drone simulator Flightmare (76). To mimic the output of the vision-based state-estimation network, we first computed the ground-truth continuous homography (62, 77) from the state of the drone

$$\dot{H} = K\left(\left[\boldsymbol{\omega}^C\right]_\times + \frac{1}{p_z^{\text{WC}}}\boldsymbol{v}^C\left(\boldsymbol{e}_{-z}^W\right)^T\right)K^{-1} \tag{20}$$

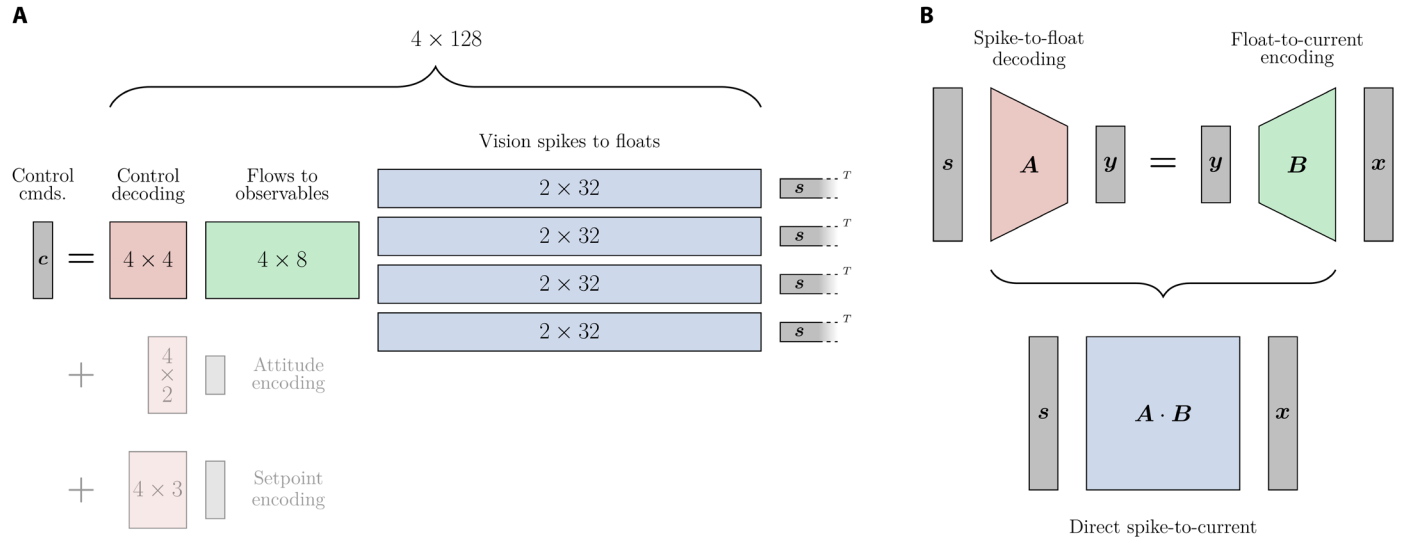**Fig. 7. Merging linear transformations.** (**A**) We go directly from output spikes s of the vision network to control commands c in a single linear decoding by multiplying the involved linear transformation matrices. This is not possible for the matrices used for encoding (shown faded), so we leave these as is. (**B**) The same principle can be applied to connect two separately trained spiking networks in a spiking manner, from spikes s to currents x, suitable for neuromorphic hardware.

where $\dot{H}$ is the continuous homography, K is the camera intrinsic matrix, $[\omega^C]_\times \in \mathbb{R}^{3\times3}$ is a skew-symmetric matrix representing infinitesimal rotations, p_z^{WC} is the Z component of the position vector from the world frame W to the camera frame C (representing perpendicular distance from the ground plane to the camera), v^C is the velocity of the camera, and e_{-z}^W is the unit vector in the negative Z direction of the world frame. To obtain angular rates and velocities in the camera frame, we used the camera extrinsics, consisting of a rotation R^{CB} and a translation T^{CB}

$$\omega^C = R^{CB}\omega^B \tag{21}$$

$$v^C = R^{CB}\left(v^B + [\omega^B]_\times T^{CB}\right) \tag{22}$$

where the right-hand sides of Eqs. 21 and 22 are known from the simulator. Next, we used the continuous homography to get the flow of the four corners (*62, 77*)

$$u_k = \left(-\left(1 - \tilde{x}_k(e_{-z}^W)^T\right)\dot{H}\tilde{x}_k\right)_{\text{Eucl}} \tag{23}$$

where u_k is the flow in Euclidean coordinates, 1 is the identity matrix, and $\tilde{x}_k = [x_k, y_k, 1]^T$ is the projection of the world points in the corners of the field of view onto the pixel array in homogeneous coordinates (so $\tilde{x}_{\text{BL}} = [0, 180, 1]^T$ and $\tilde{x}_{\text{TR}} = [180, 0, 1]^T$; note the difference with respect to Eq. 19). We added $\mathcal{N}(0, 0.025)$ noise to the flows u_k (based on a characterization of the vision SNN). Equation 19 was subsequently used to go from flows of the corner ROIs to visual observables in the camera frame, which were then transformed back to the body frame for control. Note that relating camera motion to the motion of points in the image (flow) can equivalently be done with the image Jacobian or feature sensitivity matrix (*78*).

We used a mutation-only evolutionary algorithm with a population size of 100 to evolve the weights of the linear controller matrix $\in \mathbb{R}^{4\times9}$, whose initial values were drawn from $U(-0.1, 0.1)$. More

specifically, we generated offspring by adding mutations drawn from $\mathcal{N}(0, 0.001)$ to all parameters of each parent and then evaluated the fitness of both parents and offspring. The next generation was composed of the best 100 individuals, and we repeated this process until convergence (approximately 25,000 generations). We used Flightmare to assess fitness at flying various visual observable set points: Every individual was evaluated across a set of 16 set points, with each unscaled velocity set point v_{sp}^B having at most one nonzero element $\in\{\pm0.2, \pm0.5, \pm1.0\}$ 1/s, skipping the positive set points for the Z direction, and including hover. The yaw rate set point was set to $\omega_{z,\text{sp}}^B = 0$ for all. Each set point was repeated 10 times, meaning a total of 160 evaluations per individual. Fitness F was computed as

$$F = \frac{1}{N_{\text{eval}}} \sum_{i \in N_{\text{eval}}} \sum_{j \in N_{\text{steps}}} \mathbf{w} \cdot \left(v_{\text{sp},i}^B - \begin{bmatrix} \hat{v}_x^B \\ \hat{v}_y^B \\ v_z^W \end{bmatrix}_j \right)^2 + \left(\hat{\omega}_z^B\right)^2 \tag{24}$$

Here, N_{eval} is the number of evaluations, $N_{\text{steps}} = 1000$ is the number of steps per evaluation, and $\mathbf{w} = [1, 1, w_z]^T$ is a vector weighing the fitness for different axes, where we set $w_z = 10$ for set points where $v_{z,\text{sp}}^B = 0$. Note that, for the Z direction, we used the ground-truth unscaled velocity in the world frame v_z^W instead of the one in the body frame because the latter was zero in the case of the drone ascending or descending at a slope equal to its attitude and would hence go unpunished, leading to extra vertical drift. Furthermore, if the simulated drone went out of bounds or crashed before the end of an evaluation, it would be reset without any additional fitness penalty.

We used domain randomization (*79*) to obtain a more robust controller and reduce the reality gap: For each of the 10 repeats, a random constant bias $U(-0.001, 0.001)$ rad was added to the absolute pitch and roll received by the control layer. This bias was shared among the population to keep things fair. Furthermore, for each of the 160 evaluations per individual, we randomly varied the initial

position $\boldsymbol{p}^{\mathcal{WB}} = [0, 0, 2]^T + U^{3\times1}(-1, 1)$ m (except for horizontal flight, where we fixed $p_z^{\mathcal{WB}}$ to 1.5 m, because of the linear nature of the controller we have here, as explained later), initial velocity $\boldsymbol{v}^{\mathcal{WB}} \sim U^{3\times1}(-0.02, 0.02)$ m/s, initial attitude quaternion $\boldsymbol{q}^{\mathcal{WB}} \sim U^{4\times1}(-0.02, 0.02)$ (normalized), and initial angular rates $\boldsymbol{\omega}^B \sim U^{3\times1}(-0.02, 0.02)$ rad/s.

We modified Flightmare to include drag $\boldsymbol{f}_{\text{drag}}$ occurring as a result of translational motion, and we took it to be acting in the so-called "flat-body" frame $\mathcal{B}'$, which is the body frame rotated by the roll and pitch of the drone such that the z axis is aligned with the world z axis. Following (64), we used a drag model that is linear with respect to velocity in x and y but using a drag coefficient $k_{v,x} = k_{v,y} = 0.5$. This results in the following

$$\boldsymbol{f}_{\text{drag}}^B = -\boldsymbol{R}^{BB'}\left(\begin{bmatrix} k_{v,x} \\ k_{v,y} \\ 0 \end{bmatrix} \circ \boldsymbol{R}^{B'B}\boldsymbol{v}^B\right) \quad (25)$$

The outputs of the linear controller $\boldsymbol{c} \in \mathbb{R}^{4\times1}$ were clamped to $[-1,1]$ and fed to different parts of the cascaded low-level (thrust, attitude, and rate) controllers. To accommodate some of the shortcomings of the linear controller, we compensated thrust for the attitude of the drone.

All dynamics equations were integrated with fourth-order Runge-Kutta with a time step of 2.5 ms. The frequency of the simulation was 50 Hz. All remaining details can be found in Supplementary Methods.

From simulation to the real world
We achieved successful sim-to-real transfer through several strategies. One is domain randomization (79), which we did by adding noise to the observed flows, through a random bias on the attitude estimate, and by varying the initial conditions of the simulated quadrotor. Another is abstraction (80), which we did by making use of low-level controllers to go from thrust and attitude to rotor speeds and by calibrating the attitude and hover thrust biases before each flight to ensure that they were small/zero (as in the simulator). Last, we smoothed (low-pass filter) and scaled the computed visual observables and tuned the gains scaling the control layer outputs in the real world.

Equation 19 was used to transform the flows of the corner ROIs coming from the vision network into visual observables (unscaled velocities $\widehat{v}^B$ and yaw rate $\widehat{\omega}_z^B$), absolute roll $|\phi|$ and pitch $|\theta|$ were taken from the drone's accelerometers, and the unscaled velocity set point v_{sp}^B was provided by the user. The yaw rate set point $\omega_{z,\text{sp}}^B = 0$ was fixed.

Extra experiments were performed by connecting the vision network to a hand-tuned PI controller. All remaining details can be found in Supplementary Methods.

Hardware setup
Real-world experiments were performed with a custom-built quadrotor carrying the event-based camera (DVS 240), a single-board computer (UP Squared), and a neuromorphic processor (Intel Kapoho Bay with two Loihi neuromorphic research chips). A high-level overview can be found in Fig. 1, with all components listed in Table 3. We used PX4 as autopilot firmware and ROS (Robot Operating System) for communication. More specifically, events coming from the event-based camera were passed to the UP Squared over USB using ROS1. These events were processed (downsampling, cropping, and limiting to 90 per image corner ROI) on the UP Squared and sent as spikes to the vision network running on the Kapoho Bay over USB. After processing, the output spikes were sent back over USB to the UP Squared, where they were decoded into flows for each corner ROI. The ROI flows were then published by an ROS1 node and sent to ROS2 over a ROS1-ROS2 bridge. The linear controller (or PI controller, for that matter) and the processing around it, running as a ROS2 node, took the ROI flows together with the attitude estimate coming from PX4 (IMU) and the ego-motion set point provided by the user and output the control command. This command was then sent over ROS2 to PX4 and processed by the low-level controllers there. ROS2 makes use of RTPS (real-time publish-subscribe) for communication, which allows for high-frequency and high-bandwidth messaging between the UP and PX4, meaning that our entire pipeline can run at 200 Hz—approximately the frequency of PX4's attitude controller.

Table 3. List of hardware components used for the real-world test flights. Power consumption estimates were obtained from the Loihi energy benchmark*, a real-world hover test with the drone†, or component datasheets‡. Dash entries indicate components not using any power.

Component	Product	Mass (g)	~Power (W)
Frame	GEPRC Mark 4 225 mm		
Motor	Emax 2306 Eco II Series		
Propellor	Ethix S5 5 inch	508	259†
Flight controller	Pixhawk 4 Mini		
ESC	SpeedyBee 45A BL32 4in1		
Battery	Tattu FunFly 1800 mAh 4S	195	–
Single-board computer	UP Squared ATOM Quad Core 08/64	202	18‡
Event-based camera	DVS 240	27	1‡
Neuromorphic processor	Intel Loihi, Kapoho Bay form factor	62	1*

This attitude control frequency was a major driver for our choice to run the neuromorphic pipeline at the same frequency. This choice also depended on various other factors, ranging from the trade-off between a fast execution frequency and good optical flow estimation to the limitations of the spike interfacing over USB. A 5-ms event window turned out to give accurate optical flow estimates, as well as a very fast execution frequency (see Supplementary Methods). Moreover, this frequency could be attained reliably, and Table 2 shows that Intel's Loihi inference time was consistently faster than 5 ms. For position control between test runs and as failsafe, we used an OptiTrack motion capture system.

We estimated power required for hover flight by flying with a fully charged battery until a certain voltage, keeping track of time, and estimating spent milliampere hour using the value from the charger. During this flight, the UP Squared was powered on (needed for control), and we estimated a power consumption of 277 W. Subtracting the estimated 18 W for the UP Squared results in 259 W for the remaining components in Table 3. For the Kapoho Bay, we used the numbers from the performed energy benchmark, given that running power is almost equal to idle power. We got power estimates for the UP Squared (version UPS-APLX7-A20-0864) and DVS 240 from online available datasheets.

Energy benchmark

Energy benchmarks for Loihi were performed on a Nahuku board (host machine: Intel Xeon Platinum 8280 CPU at 2.7 GHz, 126 GB RAM, Ubuntu 20.04, NxSDK 1.0.0), which contains 32 Loihi chips, because Kapoho Bay does not support energy probing. These (software) probes report a variety of power measurements, as well as execution times. Following the documentation, static power is due to transistor leakage, dynamic power is due to switching, idle power is measured while the embedded CPU cores on Loihi are still clocked but the neuromorphic cores are inactive, and total/running power is due to all components together. Kapoho Bay does support probing execution times, which we used to confirm that these are almost identical between Nahuku and Kapoho Bay. Furthermore, documentation states that Nahuku shuts down unused chips, meaning that it can emulate energy consumption of a Kapoho Bay (which has two chips) with minimal overhead. Still, these benchmarks are not very representative of actual use on a drone because that involves receiving and processing streaming data in an online fashion, whereas here all data were loaded to memory beforehand and then processed as quickly as possible. Therefore, what these benchmarks represent is not the energy consumption and execution speed of the whole pipeline but rather that of the network alone, without any bottlenecks and influences because of I/O and preprocessing. This also explains the much higher execution frequencies of Loihi with respect to real world tests, where this was always around 200 inf/s.

On Jetson Nano, we simulated the SNN and ANN in PyTorch (ARM Cortex-A57 CPU at 1.43 GHz, 4 GB RAM, Ubuntu 20.04, PyTorch 1.12.0). Jetson Nano has two power modes: a low-power (5 W) mode and a max-power (10 W) mode, the difference being the number of active CPU cores (two versus four) and the frequencies of the active CPU and GPU cores. Power consumption was measured using the tegrastats utility, and execution time was measured in Python code. The split between static and dynamic power cannot be made here because these are not available as measurements. Idle power was measured for a period of time after running the benchmarks.

Statistical analysis

Unless mentioned otherwise, figures and tables show individual runs without any statistical method applied. Reported EPEs (as in Table 1) are averaged over sequences in the test dataset. Reported drifts (as in tables S2 to S4) are averaged over all runs in a certain flying direction. For investigating the influence of the tether on the drone dynamics (see Supplementary Results), we performed a Z test with sampling distributions created using bootstrapping (81), and a confidence interval of 95% was used to determine significance.

Supplementary Materials

This PDF file includes:
Results
Methods
Tables S1 to S6
Figs. S1 to S6
References (82–88)

Other Supplementary Material for this manuscript includes the following:
Movies S1 to S4

REFERENCES AND NOTES

1. D. Cireşan, U. Meier, J. Masci, J. Schmidhuber, A committee of neural networks for traffic sign classification, in *Proceedings of the 2011 International Joint Conference on Neural Networks* (IEEE, 2011), pp. 1918–1921.

2. X. Cheng, Y. Zhong, M. Harandi, Y. Dai, X. Chang, H. Li, T. Drummond, Z. Ge, Hierarchical neural architecture search for deep stereo matching, in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, H. Lin, Eds. (Curran Associates, 2020), vol. 33, pp. 22158–22169.

3. X. Gu, Z. Fan, S. Zhu, Z. Dai, F. Tan, P. Tan, Cascade cost volume for high-resolution multi-view stereo and stereo matching, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (IEEE, 2020), pp. 2495–2504.

4. E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, T. Brox, FlowNet 2.0: Evolution of optical flow estimation with deep networks, in *Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (IEEE, 2017), pp. 2462–2470.

5. D. Sun, X. Yang, M.-Y. Liu, J. Kautz, PWC-Net: CNNs for optical flow using pyramid, warping, and cost volume, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (IEEE, 2018), pp. 8934–8943.

6. Z. Teed, J. Deng, RAFT: Recurrent all-pairs field transforms for optical flow, in *Computer Vision—ECCV 2020, Lecture Notes in Computer Science* (Springer, 2020), vol. 12347, pp. 402–419.

7. Y. Yuan, X. Chen, X. Chen, J. Wang, Segmentation transformer: Object-contextual representations for semantic segmentation, in *Computer Vision—ECCV 2020, Lecture Notes in Computer Science* (Springer, 2020), vol. 12347, pp. 173–190.

8. Z. Liu, H. Hu, Y. Lin, Z. Yao, Z. Xie, Y. Wei, J. Ning, Y. Cao, Z. Zhang, L. Dong, F. Wei, B. Guo, Swin transformer V2: Scaling up capacity and resolution, in *Proceedings of the 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (IEEE, 2022), pp. 12009–12019.

9. R. Girshick, Fast R-CNN, in *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)* (IEEE, 2015), pp. 1440–1448.

10. J. Redmon, A. Farhadi, YOLOv3: An incremental improvement. arXiv:1804.02767 [cs.CV] (2018).

11. M. Xu, Z. Zhang, H. Hu, J. Wang, L. Wang, F. Wei, X. Bai, Z. Liu, End-to-end semi-supervised object detection with soft teacher, in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)* (IEEE, 2021), pp. 3060–3069.

12. R. Garg, V. K. B.G., G. Carneiro, I. Reid, Unsupervised CNN for single view depth estimation: Geometry to the rescue, in *Computer Vision—ECCV 2016, Lecture Notes in Computer Science* (Springer, 2016), vol. 9912, pp. 740–756.

13. C. Godard, O. Mac Aodha, G. J. Brostow, Unsupervised monocular depth estimation with left-right consistency, in *Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (IEEE, 2017), pp. 270–279.

14. W. Yuan, X. Gu, Z. Dai, S. Zhu, P. Tan, NeW CRFs: Neural window fully-connected CRFs for monocular depth estimation, in *Proceedings of the 2022 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (IEEE, 2022), pp. 3916–3925.

15. NVIDIA embedded systems for next-gen autonomous machines; https://nvidia.com/en-us/autonomous-machines/embedded-systems/.

16. G. Indiveri, R. Douglas, Neuromorphic vision sensors. *Science* **288**, 1189–1190 (2000).

17. Y. Sandamirskaya, M. Kaboli, J. Conradt, T. Celikel, Neuromorphic computing hardware and neural architectures for robotics. *Sci. Robot.* **7**, eabl8419 (2022).

18. G. Gallego, T. Delbrück, G. Orchard, C. Bartolozzi, B. Taba, A. Sensi, S. Leutenegger, A. J. Davidson, J. Conradt, K. Daniilidis, D. Scaramuzza, Event-based vision: A survey. *IEEE Trans. Pattern Anal. Mach. Intell.* **44**, 154–180 (2020).

19. A. Z. Zhu, L. Yuan, K. Chaney, K. Daniilidis, EV-FlowNet: Self-supervised optical flow estimation for event-based cameras, in *Robotics: Science and Systems XIV* (Robotics: Science and Systems Foundation, 2018).

20. A. Z. Zhu, L. Yuan, K. Chaney, K. Daniilidis, Unsupervised event-based learning of optical flow, depth, and egomotion, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPRW)* (IEEE, 2019), pp. 989–997.

21. M. Gehrig, M. Millhäusler, D. Gehrig, D. Scaramuzza, E-RAFT: Dense optical flow from event cameras, in *2021 International Conference on 3D Vision (3DV)* (IEEE, 2021), pp. 197–206.

22. F. Paredes-Valles, G. C. H. E. de Croon, Back to event basics: Self-supervised learning of image reconstruction for event cameras via photometric constancy, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (IEEE, 2021), pp. 3446–3455.

23. F. Paredes-Vallés, K. Y. W. Scheper, G. C. H. E. de Croon, Unsupervised learning of a hierarchical spiking neural network for optical flow estimation: From events to global motion perception. *IEEE Trans. Pattern Anal. Mach. Intell.* **42**, 2051–2064 (2020).

24. J. Hagenaars, F. Paredes-Vallés, G. C. H. E. de Croon, Self-supervised learning of event-based optical flow with spiking neural networks, in *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. S. Liang, J. Wortman Vaughan, Eds. (Curran Associates, 2021), vol. 34, pp. 7167–7179.

25. F. Paredes-Vallés, K. Y. W. Scheper, C. De Wagter, G. C. H. E. de Croon, Taming contrast maximization for learning sequential, low-latency, event-based optical flow. arXiv:2303.05214 [cs.CV] (2023).

26. W. Maass, Networks of spiking neurons: The third generation of neural network models. *Neural Netw.* **10**, 1659–1671 (1997).

27. A. Grüning, S. M. Bohte, Spiking neural networks: Principles and challenges, in *ESANN 2014 Proceedings. European Symposium on Artificial Neural Networks* (ESANN, 2014).

28. M. Davies, A. Wild, G. Orchard, Y. Sandamirskaya, G. A. F. Guerra, P. Joshi, P. Plank, S. R. Risbud, Advancing neuromorphic computing with Loihi: A survey of results and outlook. *Proc. IEEE* **109**, 1–24 (2021).

29. F. Ottati, C. Gao, Q. Chen, G. Brignone, M. R. Casu, J. K. Eshraghian, L. Lavagno, To spike or not to spike: A digital hardware perspective on deep learning acceleration. *IEEE J. Emerg. Sel. Top. Circuits Syst.* **13**, 1015–1025 (2023).

30. P. Sterling, S. Laughlin, *Principles of Neural Design* (MIT Press, 2015).

31. F. T. Muijres, M. J. Elzinga, J. M. Melis, M. H. Dickinson, Flies evade looming targets by executing rapid visually directed banked turns. *Science* **344**, 172–177 (2014).

32. M. Pfeiffer, T. Pfeil, Deep learning with spiking neurons: Opportunities and challenges. *Front. Neurosci.* **12**, 774 (2018).

33. A. Tavanaei, M. Ghodrati, S. R. Kheradpisheh, T. Masquelier, A. Maida, Deep learning in spiking neural networks. *Neural Netw.* **111**, 47–63 (2019).

34. E. O. Neftci, H. Mostafa, F. Zenke, Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Process. Mag.* **36**, 51–63 (2019).

35. F. Zenke, T. P. Vogels, The remarkable robustness of surrogate gradient learning for instilling complex function in spiking neural networks. *Neural Comput.* **33**, 1–27 (2021).

36. S. S. Chowdhury, C. Lee, K. Roy, Towards understanding the effect of leak in spiking neural networks. *Neurocomputing* **464**, 83–94 (2021).

37. W. Fang, Z. Yu, Y. Chen, T. Masquelier, T. Huang, Y. Tian, Incorporating learnable membrane time constant to enhance learning of spiking neural networks, in *Proceedings of the 2021 IEEE/CVF International Conference on Computer Vision (ICCV)* (IEEE, 2021), pp. 2661–2671.

38. L. Zhu, M. Mangan, B. Webb, Neuromorphic sequence learning with an event camera on routes through vegetation. *Sci. Robot.* **8**, eadg3679 (2023).

39. N. Qiao, H. Mostafa, F. Corradi, M. Osswald, F. Stefanini, D. Sumislawska, G. Indiveri, A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128K synapses. *Front. Neurosci.* **9**, 141 (2015).

40. M. Davies, N. Srinivasa, T. H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C. K. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y. H. Weng, A. Wild, Y. Yang, H. Wang, Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro* **38**, 82–99 (2018).

41. A. Vitale, A. Renner, C. Nauer, D. Scaramuzza, Y. Sandamirskaya, Event-driven vision and control for UAVs on a neuromorphic chip, in *2021 IEEE International Conference on Robotics and Automation (ICRA) (ICRA)* (IEEE, 2021), pp. 103–109.

42. F. Galluppi, C. Denk, M. C. Meiner, T. C. Stewart, L. A. Plana, C. Eliasmith, S. Furber, J. Conradt, Event-based neural computing on an autonomous mobile platform, in *2014 IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, 2014), pp. 2862–2867.

43. D. Floreano, C. Mattiussi, Evolution of spiking neural controllers for autonomous vision-based robots, in *Evolutionary Robotics. From Intelligent Robotics to Artificial Life*, *Lecture Notes in Computer Science* (Springer, 2001), vol. 2217, pp. 38–61.

44. Z. Bing, C. Meschede, K. Huang, G. Chen, F. Rohrbein, M. Akl, A. Knoll, End to end learning of spiking neural network based on R-STDP for a lane keeping vehicle, in *Proceedings of the 2018 IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, 2018), pp. 4725–4732.

45. M. B. Milde, H. Blum, A. Dietmüller, D. Sumislawska, J. Conradt, G. Indiveri, Y. Sandamirskaya, Obstacle avoidance and target acquisition for robot navigation using a mixed signal analog/digital neuromorphic processing system. *Front. Neurorobot.* **11**, 28 (2017).

46. D. H. Ballard, Generalizing the Hough transform to detect arbitrary shapes. *Pattern Recognit.* **13**, 111–122 (1981).

47. J. Dupeyroux, J. J. Hagenaars, F. Paredes-Vallés, G. C. H. E. de Croon, Neuromorphic control for optic-flow-based landing of MAVs using the Loihi processor, in *2021 IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, 2021), pp. 96–102.

48. H. Rebecq, D. Gehrig, D. Scaramuzza, ESIM: An open event camera simulator, in *Conference on Robot Learning* (MLResearchPress, 2018), pp. 969–982.

49. G. C. H. E. de Croon, Monocular distance estimation with optical flow maneuvers and efference copies: A stability-based strategy. *Bioinspir. Biomim.* **11**, 016004 (2016).

50. C. Brandli, R. Berner, M. Yang, S.-C. Liu, T. Delbruck, A 240 × 180 130 dB 3 μs latency global shutter spatiotemporal vision sensor. *IEEE J. Solid State Circuits* **49**, 2333–2341 (2014).

51. G. C. H. E. de Croon, H. W. Ho, C. De Wagter, E. van Kampen, B. Remes, Q. P. Chu, Optic-flow based slope estimation for autonomous landing. *Int. J. Micro Air Veh.* **5**, 287–297 (2013).

52. B. J. Pijnacker Hordijk, K. Y. W. Scheper, G. C. H. E. de Croon, Vertical landing for micro air vehicles using event-based optical flow. *J. Field Robot.* **35**, 69–90 (2018).

53. J. J. Hagenaars, F. Paredes-Vallés, S. M. Bohté, G. C. H. E. de Croon, Evolved neuromorphic control for high speed divergence-based landings of MAVs. *IEEE Robot. Autom. Lett.* **5**, 6239–6246 (2020).

54. R. Benosman, S.-H. Ieng, C. Clercq, C. Bartolozzi, M. Srinivasan, Asynchronous frameless event-based optical flow. *Neural Netw.* **27**, 32–37 (2012).

55. S. Baker, A. Datta, T. Kanade, "Parameterizing homographies" (Tech. Rep. CMU-RI-TR-06-11, Robotics Institute, 2006).

56. H. C. Longuet-Higgins, K. Prazdny, The interpretation of a moving retinal image. *Proc. R. Soc. Lond. B Biol. Sci.* **208**, 385–397 (1980).

57. G. Gallego, H. Rebecq, D. Scaramuzza, A unifying contrast maximization framework for event cameras, with applications to motion, depth, and optical flow estimation, in *Proceedings of the 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (IEEE, 2018), pp. 3867–3876.

58. G. Gallego, M. Gehrig, D. Scaramuzza, Focus is all you need: Loss functions for event-based vision, in *Proceedings of the 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (IEEE, 2019), pp. 12280–12289.

59. D. DeTone, T. Malisiewicz, A. Rabinovich, Deep image homography estimation. arXiv:1606.03798 [cs.CV] (2016).

60. T. Nguyen, S. W. Chen, S. S. Shivakumar, C. J. Taylor, V. Kumar, Unsupervised deep homography: A fast and robust homography estimation model. *IEEE Robot. Autom. Lett.* **3**, 2346–2353 (2018).

61. N. J. Sanket, C. M. Parameshwara, C. D. Singh, A. V. Kuruttukulam, C. Fermüller, D. Scaramuzza, Y. Aloimonos, EVDodgeNet: Deep dynamic obstacle dodging with event cameras, in *Proceedings of the 2020 IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, 2020), pp. 10651–10657.

62. Y. Ma, S. Soatto, J. Košecká, S. S. Sastry, *An Invitation to 3-D Vision: From Images to Geometric Models* (Springer, 2004).

63. D. B. de Jong, F. Paredes-Vallés, G. C. de Croon, How do neural networks estimate optical flow? A neuropsychology-inspired study. *IEEE Trans. Pattern Anal. Mach. Intell.* **44**, 8290–8305 (2021).

64. C. De Wagter, F. Paredes-Vallés, N. Sheth, G. de Croon, The sensing, state-estimation, and control behind the winning entry to the 2019 Artificial Intelligence Robotic Racing Competition. *Field Robot.* **2**, 1263–1290 (2022).

65. B. Boroujerdian, H. Genc, S. Krishnan, W. Cui, A. Faust, V. Reddi, MAVBench: Micro aerial vehicle benchmarking, in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)* (IEEE, 2018), pp. 894–907.

66. M. Karásek, F. T. Muijres, C. De Wagter, B. D. Remes, G. C. De Croon, A tailless aerial robotic flapper reveals that flies use torque coupling in rapid banked turns. *Science* **361**, 1089–1094 (2018).

67. R. K. Stagsted, A. Vitale, A. Renner, L. B. Larsen, A. L. Christensen, Y. Sandamirskaya, Event-based PID controller fully realized in neuromorphic hardware: A one DoF study, in *Proceedings of the 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (IEEE, 2021), pp. 10939–10944.

68. S. Stroobants, J. Dupeyroux, G. De Croon, Design and implementation of a parsimonious neuromorphic PID for onboard altitude control for MAVs using neuromorphic processors,

in *Proceedings of the International Conference on Neuromorphic Systems 2022* (ACM, 2022), pp. 1–7.

69. C. Frenkel, D. Bol, G. Indiveri, Bottom-up and top-down approaches for the design of neuromorphic processing systems: Tradeoffs and synergies between natural and artificial intelligence. *Proc. IEEE* **111**, 623–652 (2023).

70. T. Ozawa, Y. Sekikawa, H. Saito, Accuracy and speed improvement of event camera motion estimation using a bird's-eye view transformation. *Sensors* **22**, 773 (2022).

71. A. Mitrokhin, C. Fermüller, C. Parameshwara, Y. Aloimonos, Event-based moving object detection and tracking, in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (IEEE, 2018), pp. 1–9.

72. P. Charbonnier, L. Blanc-Feraud, G. Aubert, M. Barlaud, Two deterministic half-quadratic regularization algorithms for computed imaging, in *Proceedings of 1st International Conference on Image Processing* (IEEE, 1994), pp. 168–172.

73. T. Stoffregen, L. Kleeman, Event cameras, contrast maximization and reward functions: An analysis, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (IEEE, 2019), pp. 12300–12308.

74. J. L. Valerdi, C. Bartolozzi, A. Glover, Insights into batch selection for event-camera motion estimation. *Sensors* **23**, 3699 (2023).

75. D. P. Kingma, J. Ba, Adam: A method for stochastic optimization. arXiv:1412.6980v9 [cs.LG] (2017).

76. Y. Song, S. Naji, E. Kaufmann, A. Loquercio, D. Scaramuzza, Flightmare: A flexible quadrotor simulator, in *Conference on Robot Learning* (MLResearch Press, 2020).

77. S. Zhong, P. Chirarattananon, Direct visual-inertial ego-motion estimation via iterated extended kalman filter. *IEEE Robot. Autom. Lett.* **5**, 1476–1483 (2020).

78. P. Corke, *Robotics, Vision and Control*, vol. 73 of *Springer Tracts in Advanced Robotics* (Springer, 2011).

79. J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, P. Abbeel, Domain randomization for transferring deep neural networks from simulation to the real world, in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (IEEE, 2017), pp. 23–30.

80. K. Y. W. Scheper, G. C. H. E. de Croon, Abstraction, sensory-motor coordination, and the reality gap in evolutionary robotics. *Artif. Life* **23**, 124–141 (2017).

81. P. R. Cohen, *Empirical Methods for Artificial Intelligence, Vol. 139* (MIT Press, 1995).

82. M. Denninger, D. Winkelbauer, M. Sundermeyer, W. Boerdijk, M. Knauer, K. H. Strobl, M. Humt, R. Triebel, BlenderProc2: A procedural pipeline for photorealistic rendering. *J. Open Source Softw.* **8**, 4901 (2023).

83. Y. Hu, S.-C. Liu, T. Delbruck, V2e: From video frames to realistic DVS events, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)* (IEEE, 2021), pp. 1312–1321.

84. A. Dosovitskiy, P. Fischer, E. Ilg, P. Hausser, C. Hazirbas, V. Golkov, P. Van Der Smagt, D. Cremers, T. Brox, FlowNet: Learning optical flow with convolutional networks, in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)* (IEEE, 2015), pp. 2758–2766.

85. E. Ilg, T. Saikia, M. Keuper, T. Brox, Occlusions, motion and depth boundaries with a generic network for disparity, optical flow or scene flow estimation, in *Proceedings of the European Conference on Computer Vision (ECCV)*, vol. 11216 of *Lecture Notes in Computer Science* (Springer, 2018), pp. 614–630.

86. R. J. Bouwmeester, F. Paredes-Vallés, G. C. H. E. de Croon, NanoFlowNet: Real-time dense optical flow on a nano quadcopter, in *2023 IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, 2023), pp. 1996–2003.

87. N. J. Sanket, C. D. Singh, C. Fermüller, Y. Aloimonos, Ajna: Generalized deep uncertainty for minimal perception on parsimonious robots. *Sci. Robot.* **8**, eadd5139 (2023).

88. E. Kaufmann, L. Bauersfeld, D. Scaramuzza, A benchmark comparison of learned control policies for agile quadrotor flight, in *2022 IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, 2022).