



## **UWL REPOSITORY**

**repository.uwl.ac.uk**

Neuromorphic robotic platform with visual input, processor and actuator,  
based on spiking neural networks

Cheng, Ran, Mirza, Khalid B. and Nikolic, Konstantin ORCID logoORCID: <https://orcid.org/0000-0002-6551-2977> (2020) Neuromorphic robotic platform with visual input, processor and actuator, based on spiking neural networks. Applied System Innovation, 3 (2). pp. 28-43.

<http://dx.doi.org/10.3390/asi3020028>

**This is the Published Version of the final output.**

**UWL repository link:** <https://repository.uwl.ac.uk/id/eprint/7083/>

**Alternative formats:** If you require this document in an alternative format, please contact:  
[open.research@uwl.ac.uk](mailto:open.research@uwl.ac.uk)


**Copyright:** Creative Commons: Attribution 4.0

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

**Take down policy:** If you believe that this document breaches copyright, please contact us at [open.research@uwl.ac.uk](mailto:open.research@uwl.ac.uk) providing details, and we will remove access to the work immediately and investigate your claim.

## Article

# Neuromorphic Robotic Platform with Visual Input, Processor and Actuator, Based on Spiking Neural Networks

Ran Cheng <sup>1</sup>, Khalid B. Mirza <sup>1</sup> and Konstantin Nikolic <sup>1,2,\*</sup> 

<sup>1</sup> Department of Electrical and Electronic Engineering, Imperial College London, London SW7 2BT, UK; ran.cheng18@imperial.ac.uk (R.C.); k.mirza@imperial.ac.uk (K.B.M.)

<sup>2</sup> School of Computing and Engineering, University of West London, St Mary's Road, Ealing, London W5 5RF, UK

\* Correspondence: konstantin.nikolic@uwl.ac.uk

Received: 18 March 2020; Accepted: 22 June 2020; Published: 24 June 2020



**Abstract:** This paper describes the design and modus of operation of a neuromorphic robotic platform based on SpiNNaker, and its implementation on the goalkeeper task. The robotic system utilises an address event representation (AER) type of camera (dynamic vision sensor (DVS)) to capture features of a moving ball, and a servo motor to position the goalkeeper to intercept the incoming ball. At the backbone of the system is a microcontroller (Arduino Due) which facilitates communication and control between different robot parts. A spiking neuronal network (SNN), which is running on SpiNNaker, predicts the location of arrival of the moving ball and decides where to place the goalkeeper. In our setup, the maximum data transmission speed of the closed-loop system is approximately 3000 packets per second for both uplink and downlink, and the robot can intercept balls whose speed is up to 1 m/s starting from the distance of about 0.8 m. The interception accuracy is up to 85%, the response latency is 6.5 ms and the maximum power consumption is 7.15 W. This is better than previous implementations based on PC. Here, a simplified version of an SNN has been developed for the 'interception of a moving object' task, for the purpose of demonstrating the platform, however a generalised SNN for this problem is a nontrivial problem. A demo video of the robot goalie is available on YouTube.

**Keywords:** neuromorphic engineering; SpiNNaker; DVS; robotic goalkeeper

## 1. Introduction

Neuromorphic engineering is inspired by the human neural system to address abstraction and automate human-type activities [1]. Spiking neuronal networks (SNNs) can be implemented in neuromorphic systems and together they are a part of the third generation of artificial intelligence [2,3]. The information in SNNs is encoded by the signal itself and its timing, and the signal only typically has two states: logical '1' and logical '0'. This encoding works towards reduced computational complexity in comparison to conventional deep neural networks which deal with real values. This type of encoding is also in coherence with address event representation (AER) which is often used in sensory neuromorphic systems [4].

The inherent sparseness of SNNs make them suitable for event-based image processing. Conventional cameras capture 25 or more still frames per second to present motion. Each frame in this representation is independent and normally the identical background is repeatedly recorded, which increases computational complexity and generates excessive and often useless data. However, pixels in event-driven cameras only generate data when they detect changes that are above a pre-defined threshold. This presentation enables neural networks to process the information it needs,

without spending time and power to process irrelevant parts. Each pixel in an event-driven camera, such as dynamic vision sensor (DVS) [5], is independent; the resulting data are sparse, and information is encoded by the location of events and its timing. These features of event-driven cameras are in line with the characteristics of SNNs. Each neuron in the input population of SNNs can correspond to a pixel of the DVS. Once a pixel generates an event, the corresponding neuron can be injected with a spike. This encoding method makes SNNs an appropriate platform to process event-driven image processing. Compared to conventional deep convolutional networks (CNNs), and the task of video processing, SNNs require much less computational resources, hence lower power consumption and lower reaction latency compared to CNN.

Hardware solutions for neuromorphic computing have been developed to simulate large-scale SNNs in real-time, such as TrueNorth [6], Neurogrid [7] and SpiNNaker [8,9], and they use asynchronous processing of spike-events. In conventional chip designs, the speed of all computations is set by a global clock. However, SNNs are inherently sparse and calculations are only performed when an event signal is present. Thus, the asynchronous model is more suitable for SNNs computations. Furthermore, neuromorphic chips have a routing network, which applies time-division multiplexing to send data packets. This networks-on-chip design increases the extent of connectivity since the multi-layered two-dimensional connection topologies are mainly used in silicon chip [8]. Additionally, neuromorphic chips use distributed on-chip memory to reduce the influence of memory bottleneck, and non-volatile technology to implement synaptic strength and plasticity [9].

The neuromorphic processor used is SpiNNaker [8], which has one ethernet port to communicate with the host PC and it does not support direct communication with external devices. To solve this, advanced processor technologies (APT) group at University of Manchester (UM) uses a PC to convert communication protocols between SpiNNaker and external devices [10], which is not suitable for mobile robotics. APT group also designed a field programmable gate array (FPGA) interface to convert the data received from DVS to spikes and then inject these spikes into SpiNNaker [11]. It is a unidirectional data flow and cannot be applied for closed-loop execution.

Apart from the FPGA interface, UM and Technical University of Munich (TUM) developed a microcontroller interface that supports bidirectional data transmission [12]. In this solution, an additional complex programmable logic device (CPLD) is used for the converting protocol between SpiNNaker and the microcontroller. The data bus in SpiNNakerLink is 7 bits, and it becomes 9 bits after the conversion of the CPLD [11]. This solution also has five pre-defined ports for two DVSs and three motors, which simplifies the connection of DVSs and motors, but also results in a limited capacity for other external devices and sensors, such as optical flow sensor. Furthermore, the two chips applied in this solution also mean higher power consumption and increased difficulty for further development.

TUM applies their interface to the robot called 'PushBot' or 'SpOmnibot', which uses Wi-Fi access point to relay the robot and a host machine [13,14]. The Wi-Fi connection increases the mobility of the robot at the expense of higher reaction latency. Another event-driven platform which combines SpiNNaker with a spiking silicon retina (such as ATIS) has been demonstrated on real-time object recognition task, specifically to classify 26 class characters [15], as well as for object tracking [16].

In addition to the applications where SpiNNaker was combined with a vision sensor, it has been used in multiple physical autonomous robots, such as musculoskeletal robotic hardware with neural control implemented on SpiNNaker [17]. Another application area of SNNs run on SpiNNaker is in the context of cognitive robotics, where the goal is identifying the approach and strategy how to compute. It has been demonstrated on a robot called 'iCub' and SpiNNaker, which has been developed to grasp the task of object-specific attention [18].

Neuromorphic systems controlled by SNNs, but implemented on VLSI chips (instead of SpiNNaker), have been also demonstrated, such as an open-loop motor controller [19], which offer lower power consumption and simplified motor control. For robotic motion tracking using neuromorphic vision sensor, algorithms have been developed to distinguish the background and the moving object, to be used in combination with Hexapod robot [20] or iCub humanoid robot [21].

Deep neuronal networks applied in low-latency conventional vision-based robotics require high computational power and are dependent on a powerful, graphics processing unit (GPU)-driven PC. For a camera with say 50 frames per second, the response latency is not less than 20 ms, which is far from enough to support a fast-reacting robot. If the frame rate is increased to 5 kHz, the corresponding data rate will be 234.375 MB/s for a resolution of  $128 \times 128$  pixels. An alternative is to use an AER type camera device [22]. A robotic goalkeeper based on the DVS128, designed by Delbruck and Lang [23], achieves 3 ms reaction time. However, this system is using a PC to process data, which is reducing its mobility and power efficiency, and it cannot capture rich non-linear structures of visual input since neuronal networks are not applied in the system.

In order to utilise SNN for motion processing and implement the interface between SpiNNaker and external devices, a neuromorphic robotic platform consisting of a DVS, SpiNNaker and a servo motor has been designed. It is a ready-to-use neuromorphic platform for running any SNN to process motion in real-time, for example in the field of neuroprosthetics such as motor neuroprosthetics, which allows people to bypass the spine and directly control the prosthesis, which is vital for patients with spinal cord injury. In order to evaluate the performance of its closed-loop execution and also demonstrate the benefit of using SNN in such applications, the platform is implemented on a robot goalkeeper. A similar problem of trajectory prediction for robotics, which utilises an event-based camera and a neural network trained to predict the ball's position has been also investigated by Monforte et al. [24].

In this paper, the architecture of the system and the principle of hardware communication are described and demonstrated. An evaluation based on efficiency and accuracy is also presented and discussed. The interface developed in this paper uses a single microcontroller to establish low-latency communication between DVS, SpiNNaker and a servo motor. It is the first neuromorphic interface based on SpiNNaker that supports a servo to precisely control the position of the robotic arm. It does not require a second chip to convert the communication protocol between microcontroller and SpiNNaker, which reduces power consumption. Furthermore, an Arduino microcontroller is used. It has a number of built-in libraries for external devices, which means more sensors and actuators supported by the Arduino platform can be added easily into the platform. The components that comprise the system are off-the-shelf, and the software implemented only uses the standard libraries. This neuromorphic system offers a platform for many other applications at low-cost and simple for further development.

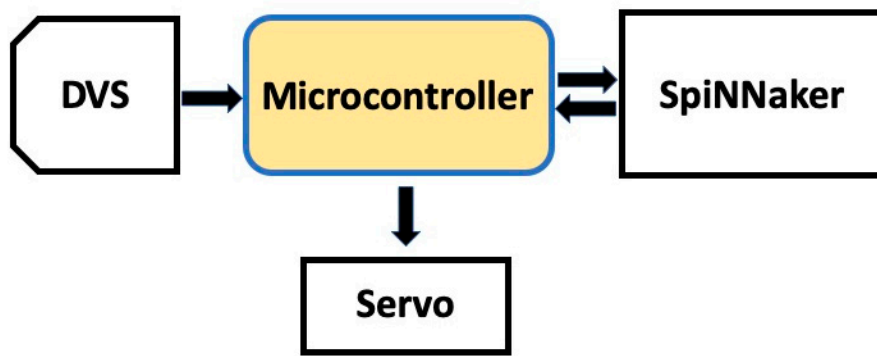
## 2. Materials and Methods

In the designed system, the microcontroller performs communication protocol conversion between DVS, SpiNNaker and the servomotor. The data formats are different for the three components, and are referred to as follows:

- *Events*: visual information captured by DVS and represented by AER protocol (between microcontroller and DVS).
- *Packets*: spikes injected to and received from SpiNNaker (between microcontroller and DVS).
- *Commands*: data used to control the servo (between microcontroller and servo).

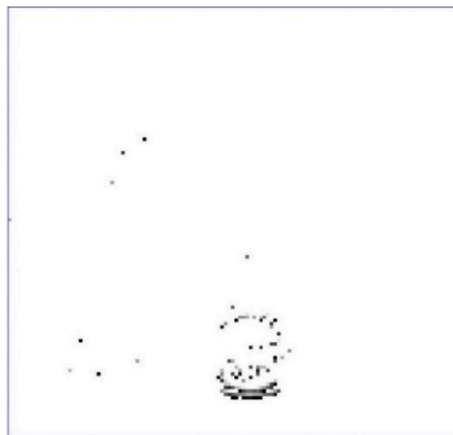
As shown in Figure 1, the microcontroller receives events from DVS, the events are then converted to packets and injected into SpiNNaker. The microcontroller receives packets (spikes) back from the SNN running on SpiNNaker. Finally, it generates servomotor control commands based on the received spikes.

The neuromorphic processor used is SpiNNaker SpiNN-3 hosted on a development board. It is a multi-chip platform, where each chip has 18 identical processing cores and six bidirectional inter-chip communication links. The SpiNN-3 has 4 SpiNNaker chips and can simulate a SNN with up to 6400 neurons. These chips share a 128 MB SDRAM, which gives over 1 GB/s sustained block transfer rate [9]. The router can route the spikes from a neuron to many connected neurons. SpiNNaker is designed to optimise the simulations of SNNs, and implements some common neuron models such as leaky integrate and fire model.



**Figure 1.** Functional blocks of the neuromorphic platform consisting of dynamic vision sensor (DVS), SpiNNaker board, and servo motor, with the microcontroller implemented as an interface between these blocks.

The DVS used in our platform is DVS128 from iniLabs, Zurich, which has a resolution of  $128 \times 128$ . The DVS camera has low latency down to  $15 \mu\text{s}$  and high dynamic range of up to 120 dB and can work at low power at 20 mW [5,22]. Figure 2 shows a DVS recording when a ping-pong ball is moving. The outline of the ball will produce spikes, but also some random noise is generated, which can be decreased by increasing the threshold of events.



**Figure 2.** DVS recording of a ping-pong ball passing.

### 2.1. From DVS to SpiNNaker

DVS only sends the addresses of pixels whose temporal change of intensity is higher than a preset threshold [5]. Therefore, the microcontroller receives the horizontal (x) and vertical (y) coordinates of events. The resolution of DVS is  $128 \times 128$ , which requires 7 bits to encode the x and y coordinates of the event address. For example, in Table 1, the packet corresponding to the event at pixel at the position (68,98) is encoded as:

**Table 1.** Encoding scheme for DVS events.

1000 0000	0010001	0100011	00	0010 1100 0100 1000
header	x	y	unused	virtual routing key

The first bit of header is the parity bit, it is set to ensure the whole packet has odd parity. The rest of header bits and the two unused bits in the packet data are set to 0.

For SpiNNaker, the spikes are encoded as packets, in the following form (Table 2):

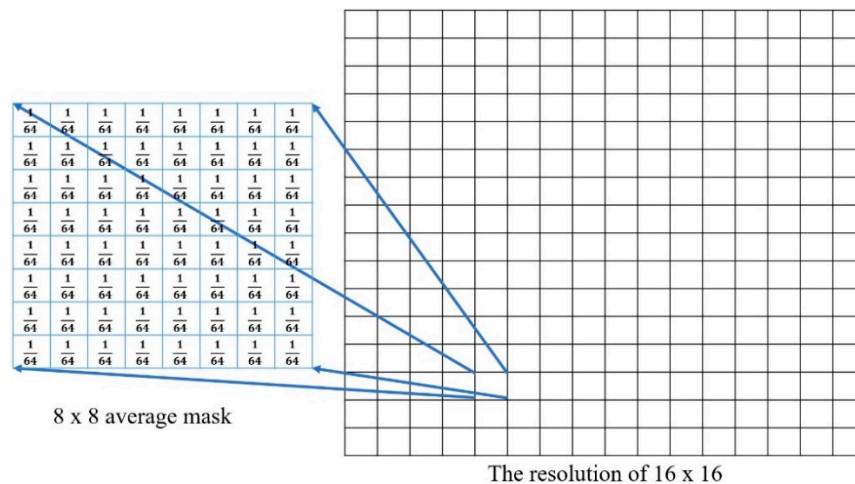
**Table 2.** SpiNNaker data packet structure.

Header (8 bits)	Packet Data (32 bits)	PayLoad (32 bits, optional)	EOP
-----------------	-----------------------	-----------------------------	-----

A packet contains: Header to indicate its specification, Packet Data which stores information about the spikes, and an optional PayLoad to carry more information, such as membrane voltage [8]. The packet format used in the designed system is 40-bit multi-cast packet without payload. Furthermore, SpiNNaker regards external devices as virtual chips on its system [25]. Thus, a virtual routing key should be defined as the delivery address of the injected spikes. In the designed system, the virtual routing key consists of four Hex numbers ('0 × 1', '0 × 2', '0 × 3' and '0 × 4').

The decimal to binary conversion of SpiNNaker packets starts from the least significant bit, and the order of virtual routing key is reversed. The reason is that SpiNNaker reads packets from the end.

The events received by the microcontroller can be sub-sampled or preprocessed to increase the processing speed and filter random noise. The resolution reduction is also important for reducing the size of the neural network needed to handle the DVS events. As shown in Figure 3, an average mask of  $8 \times 8$  has been applied to the received events and a new threshold is set to the averaged pixel value to define whether the 'superpixel' is generating an event or not.



**Figure 3.** DVS data are preprocessed, and overall resolution reduced from  $128 \times 128$  to  $16 \times 16$ , where each 'superpixel' represents a block of 64 ( $8 \times 8$ ) original pixels. The decision whether there is spike at a 'superpixel' is based on having a set number of spikes in the pixels that it covers within a specific time window. If the first layer of the SNN has one neuron for each pixel, then required number of neurons is reduced from 16,384 to 256—some loss of resolution but a significant advantage for the neural network training phase.

Through the resolution reduction, only 4 bits are needed to encode x and y of the events, respectively. For instance, the packet corresponding to the event at superpixel (3,15) is shown in Table 3.

**Table 3.** DVS resolution reduction event data encoding.

0000 0000	1100	1111	0000 0000	0010 1100 0100 1000
header	x	y	unused	virtual routing key

After receiving the raw pixel data from the DVS, the microcontroller is converting them first into superpixel events, and then into 40-bit packets appropriate for the SpiNNaker system. The microcontroller will send the packets into SpiNNaker by using 2-of-7 coding and 2-phase



handshaking protocol. Each packet is sent as 10 symbols followed by an ‘end-of-packet’ (EOP) symbol. Except for the EOP symbol, the 10 symbols are selected from 16 Hex values, and each Hex value is 4-bit.

Both SpiNNaker input and output data buses are 7 bits. To send a symbol, the microcontroller only changes the state of two wires and keeps logical levels of the rest five wires unchanged, i.e., uses the 2-of-7 coding method [26]. In Table 4, ‘1’ is represented by change of the state on the that line and ‘0’ is represented with no change of the state. In other words, logical-1 is physically represented by a change of the voltage level and not by a specific value for the digital signal. Similarly, logical-0 is represented by no-change of the physical voltage on the wire, not by a specific value. After sending a symbol, the voltage levels of the 7 wires will not return to the initial state [27]. This mechanism increases the transmission speed and reduces power consumption.

**Table 4.** 2-of-7 coding: converting a symbol into 2 changing bits. ‘1’ and ‘0’ do not represent physical voltage levels corresponding to logical-1 and logical-0, but symbolise change of state (‘1’) or no change of state (‘0’). If less confusing, the symbol ‘C’ could be used for change and ‘N’ for no-change, to encode ‘1’ and ‘0’.

Symbol.	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	EOP
L [0]	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	1	0
L [1]	0	1	0	0	0	1	0	0	0	1	0	0	1	1	0	0	0
L [2]	0	0	1	0	0	0	1	0	0	0	1	0	0	1	1	0	0
L [3]	0	0	0	1	0	0	0	1	0	0	0	1	0	0	1	1	0
L [4]	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
L [5]	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	1
L [6]	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	1

After the microcontroller changed the logical levels of two wires, SpiNNaker will send back an acknowledge (ACK) signal to indicate the symbol (data) has been received. For the microcontroller, it will send the next symbol, once it receives the ACK signal from SpiNNaker. Through this 2-phase handshaking protocol, packets are sent serially.

## 2.2. From SpiNNaker to Servomotor

After injecting spikes into SpiNNaker, the next hardware communication is to receive spikes back from SpiNNaker i.e., the output layer of the SNN that it runs, and control the servo on the bases of the received spikes. The coding method and communication protocol is the same as for injecting spikes into the SNN, and the IDs of spiking neurons in output population are also sent in 40-bit packets. In our case, the single-axis robotic arm of the neuromorphic system is set to have eight different movable positions. Thus, the output layer of the SNN has eight neurons that correspond to eight positions. What the microcontroller receive is the spiking neuron ID. It is possible to set any other number of the goalkeeper positions using this digital motor.

The rotation range of the servomotor is  $120^\circ$ , from  $-60^\circ$  to  $60^\circ$ . It is divided into eight equal steps, with the step angle of  $15^\circ$ , and each position corresponds to a neuron ID. The speed of the servomotor is  $120^\circ/150 \text{ ms} = 0.8^\circ/\text{ms}$  (for the weight of our ‘goalkeeper’). Thus, servo commands are executed at most every 150 ms. The position of the robotic arm is precisely controlled by using pulse width modulation (PWM) [28]. The range of pulse widths is from 1 ms to 2 ms, which is also divided into eight equal parts. The method of controlling the servo is described in the algorithm (Figure 4) below. The position of the robotic arm is changed after minimum 20 neuron IDs. The microcontroller keeps storing the received neuron IDs into a buffer until the number of IDs is 20. Next, the number of the most frequent ID in the buffer is found. If the number is equal to or greater than 10, a servo command corresponding to the ID will be generated. Finally, the command will be executed if the time difference from the latest execution is equal to or greater than 150 ms.

---

**Algorithm 1:** Generate and execute servo commands

---

```

No. of ID = 0;
declare a buffer;
while True do
    receive one packet from SpiNNaker;
    decode the packet to obtain the neuron ID;
    if No. of ID ≤ 20 then
        store the ID into the buffer;
        Number of ID ++;
    else
        find the most frequent ID in the buffer;
        if No. of the found ID ≥ 10 then
            generate a command;
            if time difference ≥ 150 ms then
                execute the command;
            else
                discard;
            end
        else
            discard;
        end
        reset the buffer;
        No. of ID = 0;
    end
end

```

---

**Figure 4.** The algorithm/pseudo-code above is used to generate and execute servo commands.

### 3. Implementation

The microcontroller must handle four tasks in parallel:

- receiving and processing events from DVS,
- encoding and injecting spikes to SpiNNaker,
- receiving the output spikes from SpiNNaker, and
- control the servomotor and the goalkeeper position.

Since the corresponding commands are processed virtually in parallel, it is important to synchronise all of them. The transmission speed of the AER events is not fixed, but it is proportional to the movement captured by DVS. The resulting speed of injecting spikes into SpiNNaker is also not fixed. Furthermore, the output spikes from SNNs are sparse and not at a fixed rate, which causes a varying speed of generating commands.

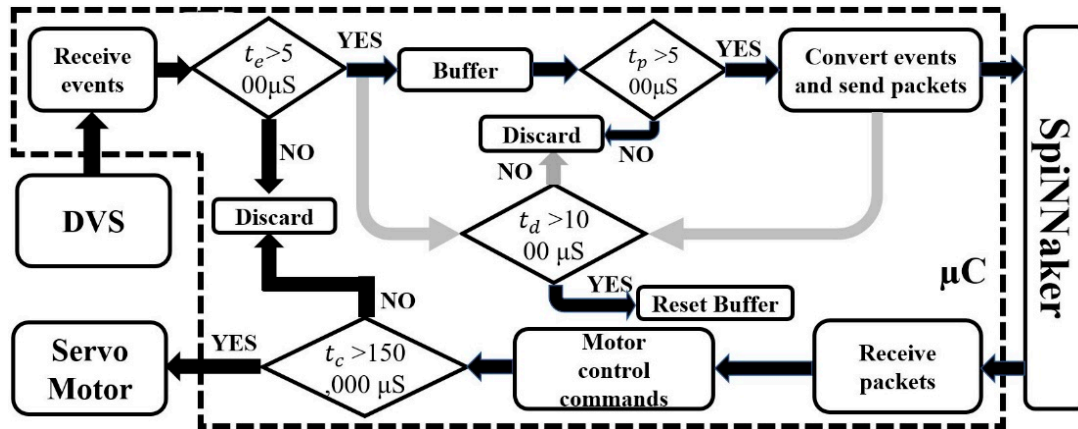
#### 3.1. Synchronization Mechanism

To ensure the SNN receives the latest events and the command to the servo is generated based on the latest output spikes, a synchronisation mechanism is proposed, as shown in Figure 5. This mechanism sets a limit for the maximum transmission speed of injecting spikes. For example, this limit is set to 2000 packets per second. As shown in Figure 5, the microcontroller keeps receiving events but only stores one event into the buffer every 500  $\mu$ s. For injecting spikes, the period is also 500  $\mu$ s and the microcontroller will reset the buffer if the time difference between current stored event and sent spike is greater than 1000  $\mu$ s. For servo command, it will be overwritten and not be executed until the time difference from last executed command is equal to or greater than 150 ms. Through this mechanism, events, spikes and commands are synchronised to reduce response latency.

The buffering process for events enables the system to adapt to changing the speed of receiving events but also brings some limitations. Compared to injecting events directly, the microcontroller spends computation power on writing, reading and resetting the buffer. Thus, the maximum transmission speed is further limited. Furthermore, information will be partially dropped out if the

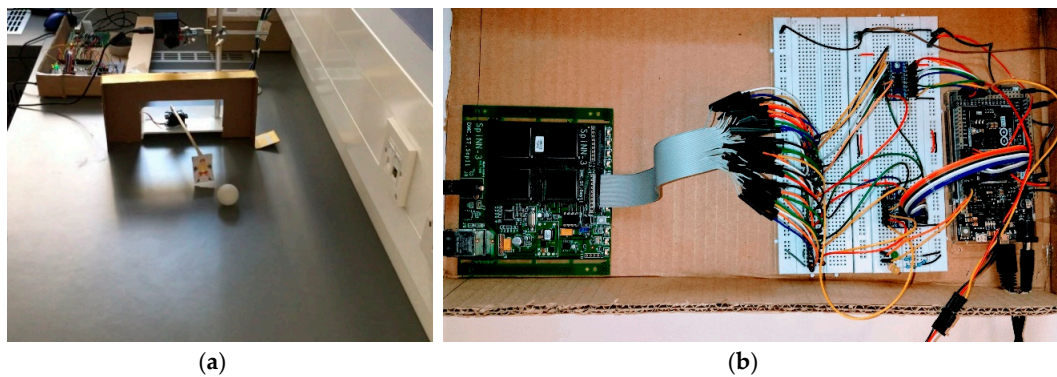


speed of receiving events is higher than the pre-defined transmission speed. This will reduce the accuracy of the system.



**Figure 5.** Synchronization of events, spikes and commands at the assumed maximum rate of 2000 packets/s for the SpiNNaker input spikes.

The setup of the neuromorphic goalkeeper is shown in Figure 6a. The width of the baffle is approximately 4 cm and the movable range of the robot arm is 30 cm. Figure 6b shows a photo of the connections between the different modules.



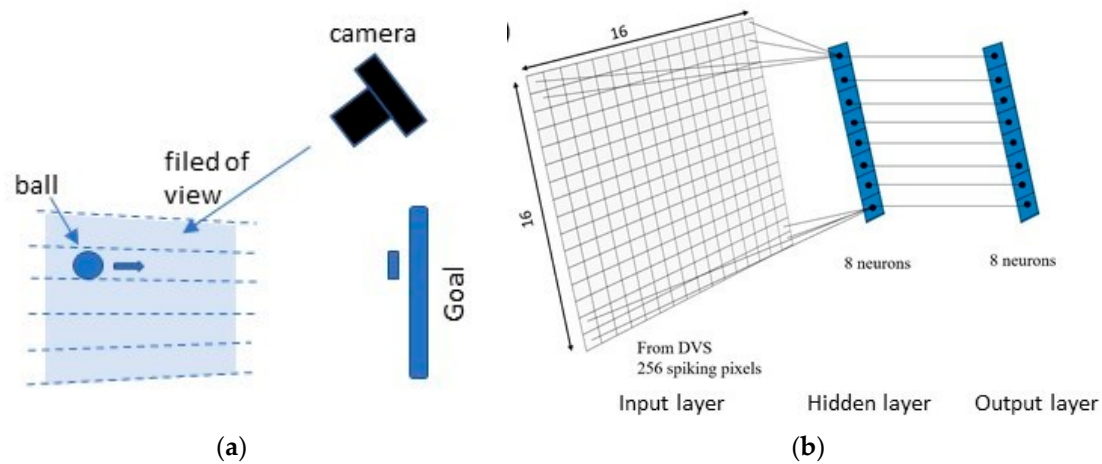
**Figure 6.** (a) Setup of the neuromorphic robotic goalkeeper. (b) Connections between the modules. One input/output port is used on SpiNNaker (SpiNN-3) board (shown left) and set in SpiNNakerLink class. Two chips on the protoboard are level shifters (1.8 V to/from 3.3 V), which are used between the microcontroller and SpiNNaker. The microcontroller (Arduino Due) is at the right end of the picture.

### 3.2. Neural Network Model

For the target application described in this paper, a relatively simple SNN has been developed to run on SpiNNaker, since the aim is to demonstrate our hardware platform. Development of a general SNN for the task of predicting the end position of a ball based on its initial motion (the ‘goalkeeper’ task) is a non-trivial problem and it is beyond the scope of this paper. A generalised methodology for developing an SNN to run on a DVS-SpiNNaker platform, using either supervised or unsupervised training algorithm, is described in Appendix A. Here the developed SNN is described to demonstrate our platform, and the model is shown and explained in Figure 7a.

The network has three layers as shown in Figure 7b. The first (input) layer of the SNN is connected to processed DVS output, which has a resolution of  $16 \times 16$  after downsampling (explained in Figure 3). The second (hidden) layer has eight neurons, and each neuron is connected to two columns of the DVS layer, and hence has 32 input synapses. The third (output) layer also has eight neurons, which correspond to eight possible positions of the goalkeeper. The neurons in the layers two and

three are connected using one-to-one connectivity. The total number of synapses in the SNN is  $32 \times 8 + 8 = 264$ . The SNN has been developed using PyNN [29] with sPyNNaker extension [30] and run on the SpiNNaker. The neuron model used in the SNN is standard leaky integrate-and-fire (LIF).



**Figure 7.** (a) Model: the neural network identifies in which of the possible  $N$  'lanes' the ball is approaching the goal ( $N = 8$  in the demonstration). This model covers only a subset of all possible trajectories of the ball towards the goal, but it is sufficiently good approximation for the platform hardware demonstration purposes. The field of view of the DVS camera is approximately 35 cm  $\times$  50 cm (not a square because of perspective). (b) The spiking neuronal network (SNN) topology.

For the input real-time captured events from DVS has been used, which was recording a ball rolling (pushed by a hand towards the goal from the opposite side of the table). The weights of all synapses were adjusted manually, without using any training algorithms, based on the model explained in Figure 7.

As shown in Figure 7b, lateral inhibition is not implemented in the output layer. This means the output neurons can spike simultaneously, and in that case the resulting output could not guide the robot arm to a unique location. Therefore, a voting mechanism (Figure 4) has been introduced. It is setting a threshold of 10 spikes as a criterion for deciding which neuron will determine the goalkeeper's position. Additionally, SpiNNaker will not send any packets unless a neuron in the output population is spiking. Furthermore, this threshold increases the robustness of the network to random spikes in the input layer representing the DVS pixels.

### 3.3. The Workflow

The workflow for the proposed method is shown in Figure 8. There are five main steps in the closed-loop execution:

1. DVS captures the events related to a moving object;
2. The microcontroller receives the events in AER format, converts them to SpiNNaker packets and sends to SpiNNaker;
3. SpiNNaker receives packets and injects spikes into the SNN, and after processing by SNN sends the outputs back to the microcontroller;
4. The microcontroller receives the IDs of output spikes and generates the servo command by using the algorithm (Figure 4);
5. The position of the robotic arm is adjusted by changing the pulse width of control signal received by the digital servomotor.

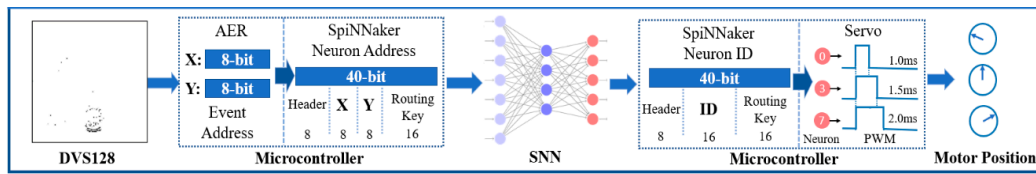


Figure 8. Workflow of the robotic system used for interception of a moving object.

#### 4. Results

The SNN has been coded in PyNN and downloaded to SpiNNaker. After the network was downloaded, the SpiNNaker board (i.e., the ethernet link) has been disconnected from the laptop, to demonstrate PC-independent operation. Once the simulation starts, the SNN will run for arbitrary time (typically 60 s in our experiments) with a time step of 1 millisecond. The input layer receives the injected spikes from the microcontroller, and the output layer has eight neurons corresponding to the eight positions. The network will not send any packets unless a neuron in the output population is spiking. A demo video of the robot goalie action can be found at the following link: <https://www.youtube.com/watch?v=135fH21QXtg>. The neuromorphic goalkeeper achieves an accuracy of up to 85% on the condition that the speed of the ball is up to 1 m/s and the initial distance more than 80 cm.

For the hardware communication, the logical levels of the ACK signals of injecting spikes (uplink) and receiving spikes (downlink) are monitored to measure the transmission speed, Figure 9. As can be observed both ACK signals of uplink and downlink have changed 11 times to send a packet, which indicates that data is in the form of 40-bit packets followed by an EOP symbol.

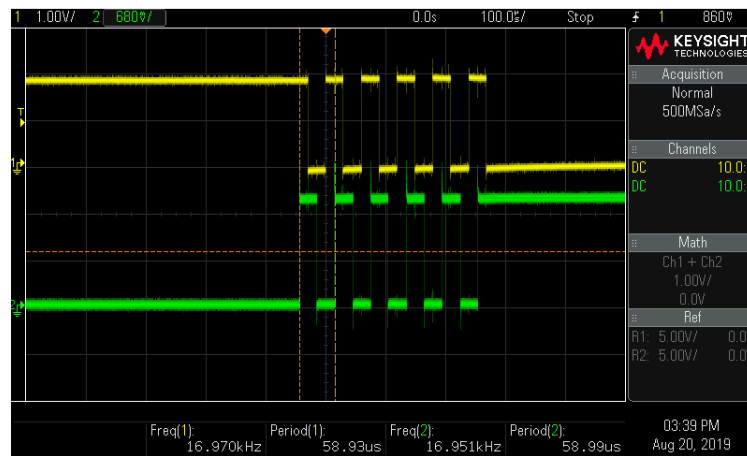


Figure 9. The logical levels of the acknowledge (ACK) signals (Yellow: downlink, Blue: uplink) during transmission at the maximum speed.

According to the reading of the oscilloscope, the frequencies of uplink and downlink ACK signals are 16.951 kHz and 16.970 kHz, respectively. Two symbols are sent in one period, since the logical levels of ACK signals change twice every period. Therefore, the uplink speed is:

$$V_{uplink} = 16.951 \text{ kHz} \times 2 = 33.902 \text{ Ksymbol/s} \approx 3082 \text{ packets/s}$$

The download speed is:

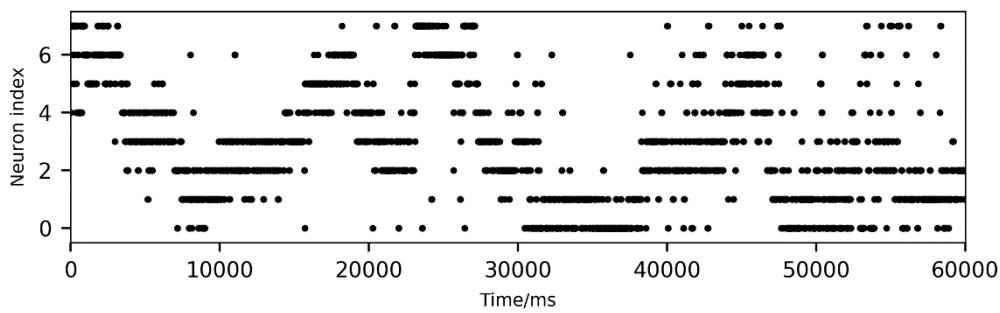
$$V_{downlink} = 16.970 \text{ kHz} \times 2 = 33.940 \text{ Ksymbol/s} \approx 3085 \text{ packets/s}$$

The microcontroller receives events in parallel but injects and receives spikes in serial. Thus, the speed of receiving events is higher than the speed of injecting and receiving spikes.

Since each process are parallelly executed and the speed of injecting spikes is higher than the speed of receiving spikes, the response latency is dominated by the time cost to inject spikes. Furthermore, the servo command is generated based on 20 spikes. Thus, the response latency is:

$$t_{\text{latency}} = 20 \times \frac{1}{3082} \text{ s} \approx 6.489 \text{ ms}$$

The SNN was running for 1 min in multiple trials, and its output spikes were recorded. Since the servo motor was set to have eight movable positions, the abscissa of the coordinate where the ball can move is also divided equally into eight parts. The microcontroller obtains current coordinate of the moving ball from DVS and inputs it to the SNN. The classification of the coordinate will then be output by the SNN. For example, neuron 6 in the output layer of the SNN will spike if current abscissa of the moving ball is in the 6th position. An example of the SNN activity for some randomly released balls is shown in Figure 10. The output spikes are concentrated at one or two neuron IDs in a short period, which gives strong and effective guidance for the microcontroller to generate servo commands. Because of the voting method applied in Figure 4, the effect of random noise has been reduced.



**Figure 10.** The spikes of the output layer of the SNN in one trial.

The neuromorphic robotic system has a maximum power consumption of 7.15 W, which is calculated in the following way. For DVS at high activity, its supply voltage is 5 V and the current is 100 mA. For the microcontroller, its operational voltage is 3.3 V and it has a maximum fuse current of 500 mA. For SpiNN-3 board, its power supply is 5 V, idle current is 400 mA and peak current is 1 A [31]. Therefore, the maximum power of the system is:

$$P_{\text{max}} = 5 \text{ V} \times 0.1 \text{ A} + 3.3 \text{ V} \times 0.5 \text{ A} + 5 \text{ V} \times 1 \text{ A} = 7.15 \text{ W}.$$

In our case the neural network can be run on only one chip (one of four) on SpiNNaker, which requires less than 2 mW of power, so the average power consumption in our example is approximately 4 W.

The developed system has high efficiency due to the inherently sparse events and spikes, impulse propagation of SNNs [32], and the synchronisation mechanism. The transmission speed of hardware communication is not fixed, it only has an assumed limit for the maximum speed in our realisation, but it could be relaxed. The speed is proportional to the events captured by DVS. Because of the adaptive speed, the efficiency is significantly improved.

## 5. Discussion

The platform developed in this paper has successfully demonstrated the closed-loop communication between DVS, SpiNNaker and a servo motor by using a single microcontroller. Compared to unidirectional communication of the FPGA interface, the developed solution supports bidirectional data flow and can control external actuators according to the received spikes from SNNs. The FPGA interface is based on a high-cost RoggedStone2 development kit, while our solution is based on the Arduino platform, which is more economical.

Apart from this solution, another microcontroller interface developed by the UM and TUM also supports bidirectional communication and control actuators. However, it requires the assistance of a CPLD, which increases the data transmission speed between SpiNNaker and the microcontroller but also increases the difficulty of further development. Their solution is based on an ARM Cortex-M4 microcontroller with a 168 MHz clock speed, whilst we use a Cortex-M3 microcontroller with an 86 MHz clock speed. Thus, the data transition of our solution is slower than the speed of the interface designed by UM and TUM. On the other side, our solution does not need the second chip, which means lower power consumption and higher stability. Additionally, the microcontroller developed by UM and TUM has 5 pre-defined peripheral ports for 2 DVSs and 3 motors. This design simplifies the connections with DVSs and motors but results in a limited capacity for other actuators and sensors. The microcontroller board used in our solution has 54 GPIOs and built-in libraries for external devices.

Compared to the robot goalie based on a PC (and DVS), the neuromorphic system that uses SpiNNaker as the processor has a lower power consumption, and it is completely controlled by an SNN running on a SpiNNaker, not using any PC processing power. Therefore, our system is portable and readily implementable on autonomous, battery-operated robotic applications. Admittedly for a small neural network as ours it would be possible to implement it on a microcontroller and thus the power consumption could be even lower than by using SpiNNaker. However, SpiNNaker is much easier to be used in developing a new SNN for our platform because of already existing PyNN software interface, and the power consumption is still acceptable for robotic applications.

Our solution can also be compared with other neuromorphic systems for tracking objects by using artificial neural networks. For example, Monforte et al. [24] have developed a recurrent neural network (RNN) for predicting the trajectory of a bouncing ball (the hardware used was ATIS event camera and iCub robot [16,21,24]). The input events of this neural network were also pre-processed by sub-sampling. This strategy reduces the computational complexity, but can affect the resolution precision. The RNN was using the long-short term memory (LSTM) learning rule [24], which indicates the current output is depended on previous input events. It is similar to our algorithm where the current prediction of the position of ball's arrival is based on 20 latest spikes received from SpiNNaker (Figure 4). Prediction of the upcoming trajectory of a tracked object was done in an asynchronous fashion. The RNN gives both horizontal and vertical prediction, but our algorithm only focuses on horizontal information. The training set and the testing set of the RNN are recorded data, and its performance of real-time events has not been investigated.

The camera (DVS) used to capture the information of moving objects is of the neuromorphic type, which exploits data-driven, event-based updates—the same as SNNs. It generates inherently sparse data, which is important for low latency and energy-efficient applications. Compared to frame-based cameras, bandwidth and computational complexity are greatly reduced. Furthermore, DVS is much better for the task of object tracking since the static background does not generate data.

The motor control mechanism used in the developed system is PWM, and it can precisely place the robotic arm. Compared to the pulse-frequency-modulation (PFM) motor control for neuromorphic robotics [19], PWM is in the standard library and does not require extra hardware. However, PFM has lower electromagnetic interferences and response latency at the expense of high-cost and complex design.

Although a relatively simple neural network model is used in this work, it is surprisingly efficient, as demonstrated in our YouTube video, as long as the incoming ball's trajectories are not deliberately chosen to play on the weakness of the model, which is to change the 'lanes' of motion. The goalkeeper can only focus on one ball at the time, hence the accuracy will be reduced if more than one moving ball appear in the field of view of DVS simultaneously. However, this constraint is more due to the simplicity of the used SNN rather than being a hardware limitation. For further improvement of the robot for this task, an advanced SNN could be developed, based on, for example, the human ball-catching models.



Another possible limitation of our SNN and potentially use of a DVS in general is the situation when the background is changing. In that case the number of events will increase rapidly what might overload the capacity of the microcontroller to handle them, and also to force all pixels to spike almost simultaneously after post-processing and resolution reduction. Some moderate change of the background including some other small objects moving in the field of view or some noise is currently eliminated by adjusting the spiking threshold in the DVS, as well as with our post-processing of events (decision when a super-pixel spikes) and our algorithm (Figure 4) which sets a minimum threshold in the number of spikes in the output layer for making the decision about where to move the goalkeeper.

## 6. Conclusions

This paper focuses on the development of a neuromorphic robotic system and its demonstration on the robot goalkeeper task. The developed interface is the first neuromorphic interface that can precisely control the position of the robotic arm by using an SNN. It can allocate different positions corresponding to the behaviour of each neuron in the output population. The system is neuromorphic and controlled by an SNN run on SpiNNaker detached from a PC. It has high efficiency, due to the sparse data, use of SNNs and the adaptive transmission speed.

The designed system provides a low-cost and user-friendly platform for various neuromorphic robotics applications. The communication between the neuromorphic processor and the microcontroller is executed automatically. Developers only need to connect sensors to the microcontroller and get the output spikes through the microcontroller. The designed system is a prototype of general-purpose neuromorphic robotics platform. An SNN has been successfully applied in this closed-loop hardware system to solve a real-world problem: interception of a moving object. In further development, a microcontroller with a higher speed could be used to increase the data transmission speed and an advanced SNN could be created to increase the accuracy and responsiveness of the system. Regarding the applications of such a neuromorphic sensory-processing-actuation systems, one potential area is also in neuroprosthetic systems, such as sensory prosthesis (visual, cochlear, e.g., [33]) or sensory substitution systems [34]. Future work will also include development of deep learning realised on SNNs [35], and for that purpose our platform can serve as an excellent testing ground. Deep learning has been achieving exceptional progress, but for real-time reaction in real environment the efficiency and latency can be significantly improved by deploying the deep SNNs.

**Author Contributions:** Methodology, R.C., K.B.M., and K.N.; software, R.C.; supervision, K.N.; writing—original draft, R.C.; writing—review and editing, K.N. and K.B.M. All authors have read and agreed to the published version of the manuscript.

**Funding:** K.N. thanks the support from UK EPSRC grant EP/N002474/1.

**Acknowledgments:** We authors would like to express our sincere gratitude to The APT group, University of Manchester for the technical support of the communication between SpiNNaker and external devices.

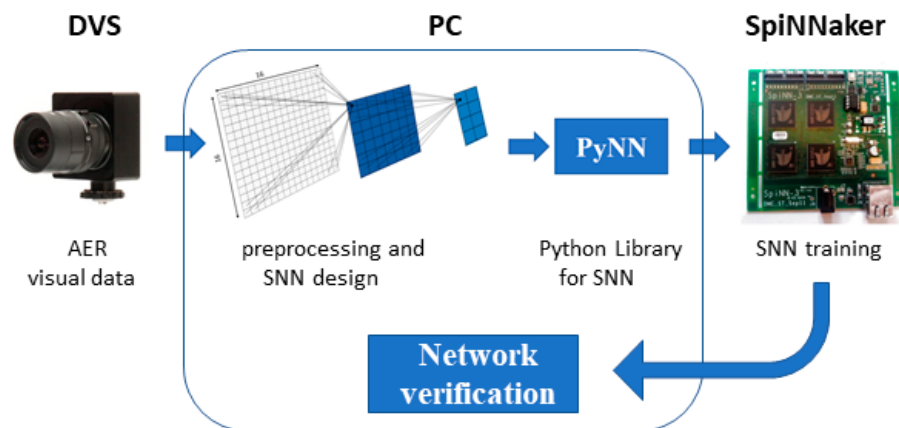
**Conflicts of Interest:** The authors declare no conflict of interest.

## Appendix A

Here a generalised methodology for training a neural network for our DVS-SpiNNaker-based platform is described, Figure A1. To run SNNs on SpiNNaker, sPyNNaker [30] software package is used, which is running PyNN [29] simulations of SNNs on SpiNNaker. PyNN is a Python interface to define SNN simulations for a range of simulator back-ends. PyNN defines a number of standard cell models, such as the LIF neuron, and the Izhikevich neuron including custom built models, as well as different types of synapses and synapse dynamics. For the learning algorithm it is possible to use the spike-timing-dependent plasticity (STDP) [36]. A demonstration of an SNN topology and how to use STDP to train an SNN using the example of counting the number of cars passing in each traffic lane can be found in [37,38]. As shown in Figure A1, to design and train an SNN in our platform, there are five main steps:



1. Recording motion by using DVS and generating the AER data;
2. Designing the SNN and writing it in PyNN;
3. Training the SNN (using the STDP learning rule) on SpiNNaker;
4. Reconstructing weights after training;
5. Verifying the SNN.



**Figure A1.** The process of designing and training SNN on our DVS-SpiNNaker-based platform.

## References

1. Maass, W. Networks of spiking neurons: The third generation of neural network models. *Neural Netw.* **1997**, *10*, 1659–1671. [CrossRef]
2. Ponulak, F.; Kasiski, A. Supervised learning in spiking neural networks with resume: Sequence learning, classification, and spike shifting. *Neural Comput.* **2010**, *22*, 467–510. [CrossRef] [PubMed]
3. Wei, D.; Harris, J.G. Signal reconstruction from spiking neuron models. In Proceedings of the 2004 IEEE International Symposium on Circuits and Systems, Vancouver, BC, Canada, 23–26 May 2004; Volume 5.
4. Indiveri, G.; Linares-Barranco, B.; Hamilton, T.J.; Van Schaik, A.; Etienne-Cummings, R.; Delbruck, T.; Liu, S.H.; Dudek, P.; Häfliger, P.; Schemmel, J.; et al. Neuromorphic silicon neuron circuits. *Front. Neurosci.* **2011**, *5*, 73. [CrossRef] [PubMed]
5. Lichtsteiner, P.; Posch, C.; Delbruck, T. A  $128 \times 128$  120 dB 15  $\mu$ s latency asynchronous temporal contrast vision sensor. *IEEE J. Solid State Circuits* **2008**, *43*, 566–576. [CrossRef]
6. Merolla, P.A.; Arthur, J.V.; Alvarez-Icaza, R.; Cassidy, A.S.; Sawada, J.; Akopyan, F.; Jackson, B.L.; Imam, N.; Guo, C.; Brezzo, B.; et al. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science* **2014**, *345*, 668–673. [CrossRef] [PubMed]
7. Benjamin, B.V.; Gao, P.; McQuinn, E.; Choudhary, S.; Chandrasekaran, A.R.; Bussat, J.M.; Alvarez-Icaza, R.; Arthur, J.V.; Merolla, P.A.; Boahen, K. Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations. *Proc. IEEE* **2014**, *102*, 699–716. [CrossRef]
8. Furber, S.B.; Lester, D.R.; Plana, L.A.; Garside, J.D.; Painkras, E.; Temple, S.; Brown, A.D. Overview of the SpiNNaker system architecture. *IEEE Trans. Comput.* **2013**, *62*, 2454–2467. [CrossRef]
9. Khan, M.M.; Lester, D.R.; Plana, L.A. Spinnaker: Mapping neural networks onto a massively-parallel chip multiprocessor. In Proceedings of the IEEE International Joint Conference on Neural Networks, Hong Kong, China, 1–8 June 2008; pp. 2849–2856.
10. APT Group. *SpiNNaker Datasheet Version 2.02*; University of Manchester: Manchester, UK, 2011.
11. APT Group. The University of Manchester Interfacing real-time spiking i/o with the spinnaker neuro mimetic architecture. *Aust. J. Intell. Inf. Process. Syst.* **2010**, *11*, 7–11.
12. APT Group, The University of Manchester. Interfacing AER Devices to SpiNNaker Using an FPGA. Available online: [http://spinnakermanchester.github.io/docs/fpga\\_aer/](http://spinnakermanchester.github.io/docs/fpga_aer/) (accessed on 10 March 2020).
13. Technische Universität München. PushBot (TUM SpiNNaker Robot). Available online: [http://spinnakermanchester.github.io/docs/push\\_bot/](http://spinnakermanchester.github.io/docs/push_bot/) (accessed on 10 March 2020).

14. Denk, C.; Llobet-Blandino, F.; Galluppi, F.; Plana, L.A.; Furber, S.; Conradt, J. Real-time interface board for closed-loop robotic tasks on the spinnaker neural computing system. In Proceedings of the Artificial Neural Networks and Machine Learning, Sofia, Bulgaria, 10–13 September 2013; pp. 467–474.
15. Orchard, G.; Lagorce, X.; Posch, C.; Furber, S.B.; Benosman, R.; Galluppi, F. Real-time event-driven spiking neural network object recognition on the SpiNNaker platform. In Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS), Lisbon, Portugal, 24–27 May 2015; pp. 2413–2416.
16. Glover, A.; Stokes, A.; Bartolozzi, C. ATIS + SpiNNaker: A fully event-based visual tracking demonstration. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, Madrid, Spain, 1–5 October 2018.
17. Christoph, R.; Jentzsch, S.; Hostettler, R. Musculoskeletal robots: Scalability in neural control. *IEEE Robot. Autom. Mag.* **2016**, *23*, 128–137. [[CrossRef](#)]
18. Rast, A.D.; Adams, S.V.; Davidson, S.; Davies, S.; Hopkins, M.; Rowley, A.; Stokes, A.B.; Wennekens, T.; Furber, S.; Cangelosi, A. Behavioral learning in a cognitive neuromorphic robot: An integrative approach. *IEEE Trans. Neural Netw. Learn. Syst.* **2018**, *29*, 6132–6144. [[CrossRef](#)] [[PubMed](#)]
19. Perez-Peña, F.; Linares-Barranco, A.; Chicca, E. An approach to motor control for spike-based neuromorphic robotics. In Proceedings of the IEEE Biomedical Circuits and Systems Conference, Lausanne, Switzerland, 22–24 October 2014.
20. Mishra, A.; Ghosh, R.; Goyal, A. Real-time robot tracking and following with neuromorphic vision sensor. In Proceedings of the IEEE International Conference on Biomedical Robotics and Biomechatronics, Singapore, 26–29 June 2016; pp. 13–18.
21. Glover, A.; Bartolozzi, C. Event-driven ball detection and gaze fixation in clutter. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, Daejeon, Korea, 9–14 October 2016; pp. 2203–2208.
22. Lichtsteiner, P.; Posch, C.; Delbruck, T. A 128 × 128 120 dB 30 mW asynchronous vision sensor that responds to relative intensity change. In Proceedings of the ISSCC Digest of Technical Papers, Visuals Supplement, San Francisco, CA, USA, 6–9 February 2006; pp. 508–509.
23. Delbruck, T.; Lang, M. Robotic goalie with 3 ms reaction time at 4% CPU load using event-based dynamic vision sensor. *Front. Neurosci.* **2013**, *7*, 223. [[CrossRef](#)] [[PubMed](#)]
24. Monforte, M.; Arriandiga, A.; Glover, A.; Bartolozzi, C. Exploiting event cameras for spatio-temporal prediction of fast-changing trajectories. In Proceedings of the IEEE International Conference on Artificial Intelligence Circuits and Systems, Genova, Italy, 31 August–2 September 2020; pp. 108–112.
25. APT Group. *External Devices on SpiNNaker*; University of Manchester: Manchester, UK, 2019.
26. Brouwer, A.E.; Shearer, L.B. A new table of constant weight codes. *IEEE Trans. Inf. Theory* **1990**, *36*, 1334–1380. [[CrossRef](#)]
27. Mori, H.; Satake, M.; Kishigami, T. Communication System with a Plurality of Nodes Communicably Connected for Communication Based on NRZ (Non-Return to Zero) Code. U.S. Patent Application No. 13/199,438, 1 March 2012.
28. Hagiwara, M.; Akagi, H. PWM control and experiment of modular multilevel converters. In Proceedings of the IEEE Power Electronics Specialists Conference, Rhodes, Greece, 15–19 June 2008; pp. 154–161.
29. Davison, A.P.; Brüderle, D.; Eppler, J.; Kremkow, J.; Müller, E.; Pecevski, D.; Perrinet, L.; Yger, P. PyNN: A common interface for neuronal network simulators. *Front. Neuroinform.* **2018**, *2*, 11. [[CrossRef](#)] [[PubMed](#)]
30. Rhodes, O.; Bogdan, P.A.; Brenninkmeijer, C.; Davidson, S.; Fellows, D.; Gait, A.; Lester, D.R.; Mikaitis, M.; Plana, L.A.; Rowley, A.G.D.; et al. sPyNNaker: A software package for running PyNN simulations on SpiNNaker. *Front. Neurosci.* **2018**, *12*, 816. [[CrossRef](#)] [[PubMed](#)]
31. APT Group. *SpiNNaker-3 Development Board*; University of Manchester: Manchester, UK, 2012.
32. Gerstner, W.; Kistler, W. *Spiking Neuron Models: Single Neurons, Populations, Plasticity*; Cambridge University Press: Cambridge, UK, 2002. [[CrossRef](#)]
33. Degenaar, P.; Hankins, M.; Drakakis, E.; Toumazou, C.; Nikolic, K.; Kennard, C. Retinal Prosthetic Devices. U.S. Patent Application No. 12/306,072, 17 June 2010.
34. Gaspar, N.; Sondhi, A.; Evans, B.; Nikolic, K. A low-power neuromorphic system for retinal implants and sensory substitution. In Proceedings of the 2016 IEEE Biomedical Circuits and Systems Conference (BioCAS), Shanghai, China, 17–19 October 2016; pp. 78–81.

35. Lee, J.H.; Delbruck, T.; Pfeiffer, M. Training deep spiking neural networks using backpropagation. *Front. Neurosci.* **2016**, *10*, 508. [[CrossRef](#)] [[PubMed](#)]
36. Jin, X.; Rast, A.; Galluppi, F.; Davies, S.; Furber, S. Implementing spike-timing-dependent plasticity on SpiNNaker neuromorphic hardware. In Proceedings of the IEEE World Congress on Computational Intelligence, Barcelona, Spain, 18–23 July 2010.
37. Bichler, O.; Querlioz, D.; Thorpe, S.J.; Bourgoin, J.P.; Gamrat, C. Extraction of temporally correlated features from dynamic vision sensors with spike-timing-dependent plasticity. *Neural Netw.* **2012**, *32*, 339–348. [[CrossRef](#)] [[PubMed](#)]
38. Bichler, O.; Querlioz, D.; Thorpe, S.J.; Bourgoin, J.P.; Gamrat, C. Unsupervised features extraction from asynchronous silicon retina through spike-timing-dependent plasticity. In Proceedings of the International Joint Conference on Neural Networks, San Jose, CA, USA, 31 July–5 August 2011.



© 2020 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).