Article

System integration of neuromorphic visual (DVS), processing (SpiNNaker) and servomotor modules into an autonomous robot controlled by a Spiking Neural Network

Ran Cheng[1] and Konstantin Nikolic [2,*]

[1] Department of Electrical and Electronic Engineering, Imperial College London, London SW7 2BT, UK; ran.cheng18@imperial.ac.uk
[2] School of Computing and Engineering, University of West London, St Mary's Road, Ealing, London W5 5RF, UK; Konstantin.Nikolic@uwl.ac.uk
* Correspondence: Konstantin.Nikolic@uwl.ac.uk;

Abstract: This paper describes the design and modus of operation of a neuromorphic robotic system based on SpiNNaker platform, and its implementation on the goalkeeper task. The robotic system utilizes an Address Event Representation (AER) type of camera (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 microprocessor (Arduino Due) which facilitates the communication between different robot parts. A spiking neural network, 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 85%, the response latency is 6.5 ms and the maximum power consumption is 7.15 W. A demo video of the robot goalie is available on YouTube: https://www.youtube.com/watch?v=135fH21QXtg .

Keywords: Neuromorphic engineering; SpiNNaker; DVS; Robotic Goalkeeper

1. Introduction

Neuromorphic engineering, and it is inspired by the human neural system to address abstraction and automate human-type activities [1]. Spiking neural 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', and 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. Thus, neuromorphic engineering exhibits low power consumption and low reaction latency.

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, cells 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), is independent; the resulting data is 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.

Hardware solutions for neuromorphic computing have been developed to simulate large-scale SNNs in real-time. These neuromorphic chips, such as TrueNorth [4], Neurogrid [5] and SpiNNaker [6], use asynchronous AER. 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 AER 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 multilayered two-dimensional connection topologies are mainly used in silicon chip. In addition to this, neuromorphic chips use distributed on-chip memory to reduce the influence of memory bottleneck, and non-volatile technology to implement synaptic strength and synaptic plasticity.

The neuromorphic processor we used is SpiNNaker SpiNN-3 development board. It is a multi-chip platform; 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 128MB SDRAM, which gives over 1 GB/s sustained block transfer rate [7]. The router can route the spikes from a neuron to many connected neurons. SpiNNaker is designed to optimize the simulations of SNNs, and implements some common neuron models such as leaky integrate and fire model.

Currently, SpiNNaker only has one ethernet port to communicate with a 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 [7], 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 [8]. 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 [9]. 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 [9]. The interface board 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.

Deep neural networks applied in low-latency conventional vision-based robotics require high computational power and the dependence on a powerful 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 128×128 pixels. An alternative is to use an AER type camera device [10]. A robotic goalkeeper based on the DVS128, designed by Delbruck and Lang [11], 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 neural networks are not applied in the system.

In this paper, we describe and demonstrate a neuromorphic robotic system that consists of a DVS, SpiNNaker and a servo motor, and then implement it on robotic goalkeeper. The architecture of the system and the principle of hardware communication are shown. Finally, the evaluation based on efficiency and accuracy will be 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 protocol between

microcontroller and SpiNNaker, which reduces power consumption. Furthermore, we use an Arduino microcontroller is based on that has lots of built-in libraries for external devices.

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 devices, and they are referred as:

- *Events*: visual information captured by DVS and represented by AER protocol
- *Packets*: spikes injected to and received from SpiNNaker
- *Commands*: data used to control the 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 a SNN running on SpiNNaker. Finally, it generates servomotor control commands based on the received spikes.

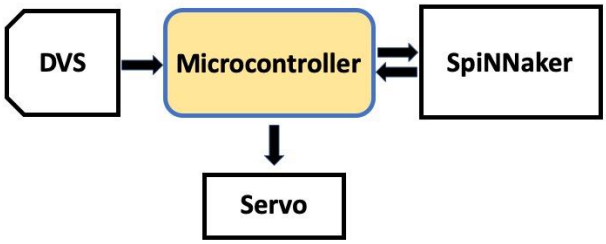


Figure 1. Architecture of the interface.

2.1 From DVS to SpiNNaker

DVS only sends the addresses of pixels whose temporal change of intensity is higher than a threshold [12]. Therefore, the microcontroller receives the horizontal (x) and vertical (y) coordinates of events. The resolution of DVS is 128×128, which requires 7 bits to encode the x and y of the event address. For example, the packet corresponding to the event at pixel at the position (68,98) is encoded as:

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

Figure 2. DVS event data encoding.

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:

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

Figure 3. SpiNNaker data packet structure.

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 [13]. 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 [14]. 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 (‘0x1’, ’0x2’, ‘0x3’ and ‘0x4’).

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 pre-processed 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 4, we have applied an average mask of 8×8 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.

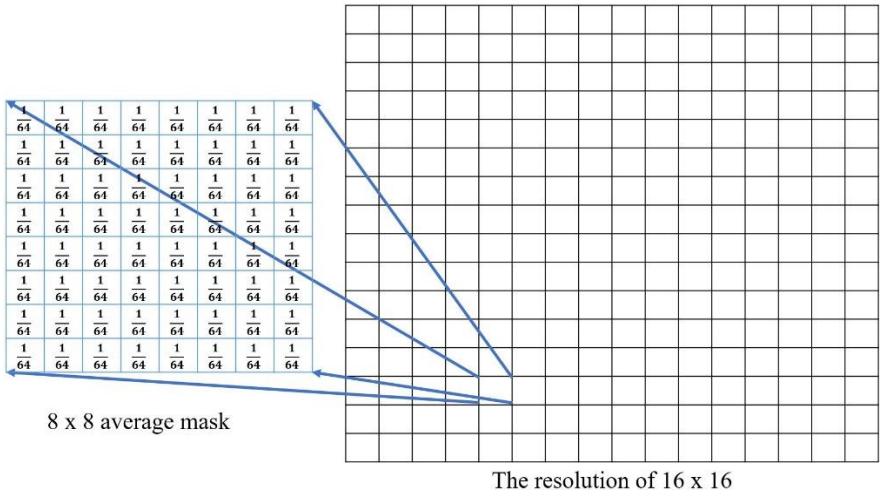


Figure 4. DVS data is pre-processed, and overall resolution reduced from 128×128 to 16×16, where each 'superpixel' represents a block of 64 (8×8) original pixels. The decision whether there is spike at a 'superpixel' is based on having a set number of spikes in the pixels that it covers within a specific 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 Figure 5.

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

Figure 5. DVS resolution reduction event data encoding.

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 2 wires and keeps logical levels of the rest 5 wires unchanged, i.e. uses the 2-of-7 coding method [15]. In Table 1, `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 [16]. This mechanism increases the transmission speed and reduces power consumption.

Table 1. 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 symbolize change of state ('1') or no change of state ('0'). If less confusing, we could have used symbols 'C' for change and 'N' for no-change, which 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 injecting spikes into the SNN, and the IDs of spiking neurons in output population are also sent in 40-bit packets. In our case, we set the single-axis robotic arm of the neuromorphic system to have 8 different movable positions. Thus, the output layer of the SNN has 8 neurons that corresponding to the 8 positions. What the microcontroller received is the spiking neuron ID, once the neuron is spiking. It is possible to set any other number of the goalkeeper positions using this digital motor.

The rotation range of the servomotor is 120^{O}, from -60^{O} to 60^{O}. We divide it into 8 equal steps, with the step angle of 15^{O}, and each postion corresponds to a neuron ID. The speed of the servomotor is $120^{O}/150\text{ms} = 0.8\ ^{O}/\text{ms}$(for the weight of our 'goalkeeper'). Thus, servo commands are executed at least every 150 ms. The position of the robotic arm is precisely controlled by using Pulse Width Modulation (PWM) [17]. The range of pulse width is from 1 ms to 2 ms, which is also divided into 8 equal parts. The method of controlling the servo is described in Algorithm 1 below. The position of the robotic arm is changed at least every 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
```

3. Implementation

The microcontroller must handle four tasks in parallel:

1. receiving and processing events from DVS,
2. encoding and injecting spikes to SpiNNaker,
3. receiving the output spikes from SpiNNaker, and
4. control the servomotor and the goalkeeper position.

Since the corresponding commands are processed virtually in parallel, it is important to synchronize 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.

To ensure the SNN receives the latest events and the command to the servo is generated based on the latest output spikes, a synchronization mechanism is proposed, shown in Figure 6. This mechanism sets a limit for the maximum transmission speed of injecting spikes. For example, we set this limit to 2000 packets per second. As shown in Figure 6, the microcontroller keeps receiving events but only stores one event into the buffer every 500 μs. For injecting spikes, the period is also 500μs and the microcontroller will reset the buffer if the time difference between current stored event and sent spike is greater than 1000 μs. 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 synchronized to reduce the response latency.

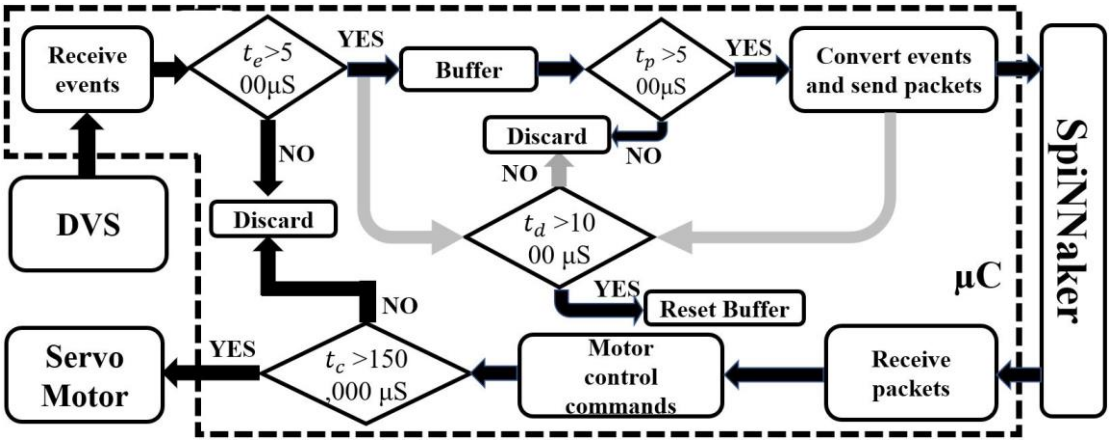
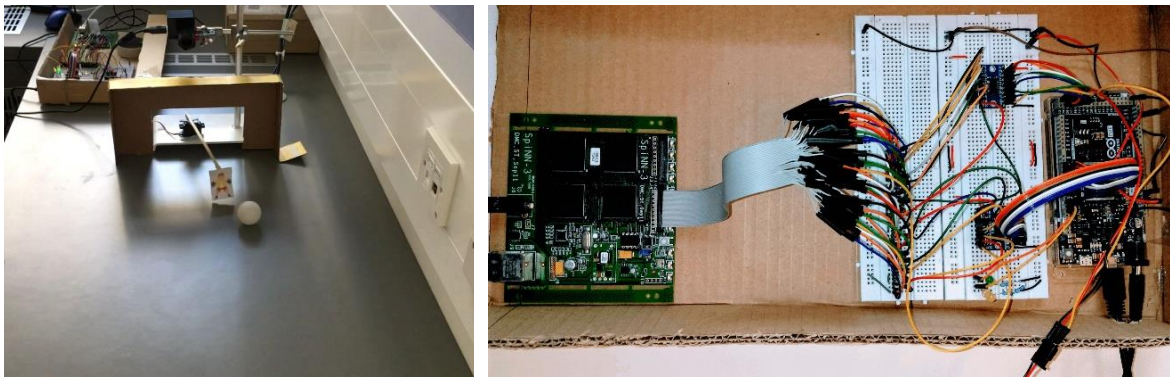


Figure 6. Synchronization of events, spikes and commands at the assumed maximum rate of 2000 packets/s for the SpiNNaker input spikes.

The buffering process for events enables the system to adapt to changing the speed of receiving events but also brings some defects. 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.

The setup of the neuromorphic goalkeeper is shown in Figure 7(a). The width of the baffle is approximately 4 cm and the movable range of the robot arm is 30 cm. Figure 7(b) shows a photo of the connections between the different modules.



(**a**) (**b**)

Figure 7. **(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 (1.8V to/from 3.3V), which are used between the microcontroller and SpiNNaker. The microcontroller (Arduino Due) is at the right end of the picture.

4. Results

Since we are predominantly concerned with the hardware solution in this paper, we are using a relatively simple SNN to capture the position and direction of motion of the ball and decide where to position the goalkeeper. The SNN has two layers, the input layer receives the injected spikes from the microcontroller, the output layer has 8 neurons corresponding to the 8 positions. This is a very simple SNN, but it is useful for our demonstration of the neuromorphic robotic system and surprisingly efficient. The goalkeeper only can focus on one ball, 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 SNN that we use rather than being a hardware limitation. For further development, an advanced SNN could be applied to increase the accuracy and the number of balls.

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 seconds in our experiments) with a time step of 1 millisecond. 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 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 8. 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.

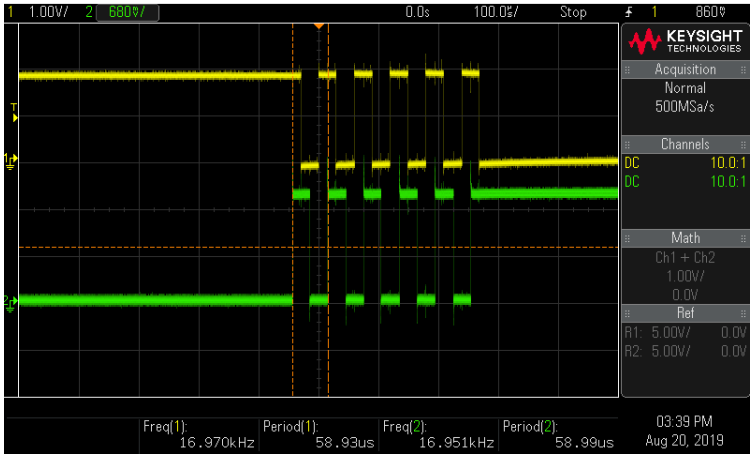


Figure 8. The logical levels of the ACK signals (Yellow: downlink, Blue: uplink) during transmission at the maximum speed

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:

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

The download speed is:

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

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:

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

The SNN was running for 1 minute in multiple trials, and its output spikes were recorded. Since the servo motor was set to have 8 movable positions, the abscissa of the coordinate where the ball can move is also divided equally into 8 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 6^{th} position. At present, we do not focus on the training of SNNs. Thus,

the SNN does not have predictive capabilities, it just gives the classification of the abscissa of the moving ball. The balls keep being randomly released at different times and directions, and the trajectory of each ball is different. The SNN activity is shown in Figure 9. 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. Due to the voting method applied in Algorithm 1, the effect of random noise has been reduced.

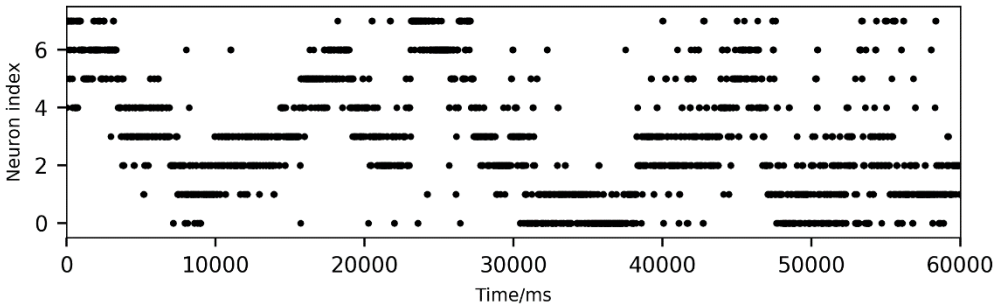


Figure 9. The spikes of the output layer of the SNN in one trial

The neuromorphic robotic system has a maximum power consumption of 7.15W, which is calculated in the following way. For DVS at high activity, its supply voltage is 5 V and 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 [18]. Therefore, the maximum power of the system is:

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

The developed system has high efficiency due to the inherently sparse events and spikes, impulse propagation of SNNs [19], and the synchronization mechanism. The transmission speed of hardware communication is not fixed, it only has an assumed limit for the maximum speed in our realization, 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 and Conclusions

The solution 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, while our solution uses 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 general-purpose inputs/outputs and built-in libraries for external devices

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 a SNN. It can allocate different positions

corresponding to the behavior of each neuron in the output population. The system is neuromorphic and controlled by a 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. Compared to the robot goalie without SpiNNaker [11], the system has a lower consumption, which is 7.15 W, and is completely controlled by a SNN running on a SpiNNaker, not using any PC processing power. It successfully applies SNNs in a closed-loop hardware system to solve real-world problems, although the SNN used here is very simple. 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 system, one potential area is also in neuroprosthetic systems, such as sensory prosthesis (visual, cochlear, e.g. [20]) or sensory substitution systems [21].

Author Contributions: Methodology, RC and KN; Software, RC; Supervision, KN; Writing – original draft, RC; Writing – review & editing, KN.

Funding: KN 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.

References

1. W. Maass, “Networks of spiking neurons: The third generation of neural network models” Neural Networks, vol. 10, no. 9, pp. 1659 – 1671, 1997
2. F. Ponulak and A. Kasiski, “Supervised learning in spiking neural networks withresume: Sequence learning, classification, and spike shifting” Neural Computation, vol. 22, no. 2, pp. 467–510, 2010
3. D Wei and J. G. Harris, “Signal reconstruction from spiking neuron models” in 2004 IEEE International Symposium on Circuits and Systems, vol. 5, 2004.
4. Merolla, P. A. et al. A million spiking-neuron integrated circuit with a scalable communication network and interface. Science 345, pp. 668–673, 2014.
5. Benjamin, B. V. et al. Neurogrid: a mixed-analog–digital multichip system for large-scale neural simulations. Proc. IEEE 102, pp.699–716, 2014.
6. M. M. Khan, D. R. Lester, and L. A. Plana, “Spinnaker: Mapping neural networks onto a massively-parallel chip multiprocessor” IEEE International Joint Conference on Neural Networks, pp. 2849–2856, 2008
7. ATP group, “SpiNNaker datasheet version 2.02”, University of Manchester, Jan 2011
8. APT group The University of Manchester, “Interfacing real-time spiking i/o with the spinnaker neuro mimetic architecture”, Australian Journal of Intelligent Information Processing Systems, vol. 11, no. 1, pp. 7–11, 2010.
9. APT group The University of Manchester, "Interfacing AER devices to SpiNNaker using an FPGA"[Online], Available: http://spinnakermanchester.github.io/docs/fpga_aer/
10. Technical University of Munich and University of Manchester, “Real-time interface board for closed-loop robotic tasks on the spinnaker neural computing system” Artificial Neural Networks and Machine Learning, pp. 467–474, 2013.
11. T Delbruck and M Lang, "Robotic goalie with 3 ms reaction time at 4% CPU load using event-based dynamic vision sensor", Frontiers in Neuroscience, 2013
12. P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128×128 120dB 30mW Asynchronous Vision Sensor that Responds to Relative Intensity Change" in ISSCC Dig. of Tech. Papers, Visuals Supplement, San Francisco, pp.508-509, 2006
13. S. B. Furber, D. R. Lester and etc, "Overview of the SpiNNaker System Architecture", in IEEE Transactions on Computers, Volume: 62, Issue: 12, 2013.
14. APT group, "External Devices on SpiNNaker", University of Manchester, Sep 2019.
15. A. E. Brouwer and L. B. Shearer, “A new table of constant weight codes”, IEEE Transactions on Information Theory, vol. 36, no. 9, pp. 1334–1380, 1990.

16. H. Mori, M. Satake, and T. Kishigami, “Communication system with a plurality of nodes communicably connected for communication based on nrz (non-return to zero) code” 2012.
17. M. Hagiwara and H. Akagi, “PWM control and experiment of modular multilevel converters,” IEEE Power Electronics Specialists Conference, pp.154–161, June 2008.
18. APT group, "SpiNNaker-3 Development Board", University of Manchester, 2012.
19. W. Gerstner and W. Kistler, "Spiking Neuron Models: Single Neurons, Populations, Plasticity", Cambridge University Press, doi:10.1017/CBO9780511815706, 2002.
20. P. Degenaar, M. Hankins, E. Drakakis, C. Toumazou, K. Nikolic, C. Kennard: ‘Retinal prosthetic devices’, US Patent App. 12/306,072 .
21. N. Gaspar, A. Sondhi, B. Evans, K. Nikolic: “A low-power neuromorphic system for retinal implants and sensory substitution”, Proc. of 2016 IEEE Biomedical Circuits and Systems Conference (BioCAS), 78-81.