Neuromorphic Robotic System

Ran Cheng

Supervisor: Dr Konstantin Nikolic

A Thesis submitted in fulfilment of requirements for the degree of
Master of Science
Communication and Signal Processing
of Imperial College London

Department of Electrical and Electronic Engineering
Imperial College London
September 4, 2019

Abstract

This project has designed, implemented and evaluated a neuromorphic robotic system based on SpiNNaker. The system utilizes a DVS to capture events of a moving object, a SNN run on SpiNNaker to predict the trajectory of the object, and a servo motor to intercept it. To establish communication between the three parts, an MCU is used. It receives events recorded by the DVS through serial port on a PC because the eDVS does not work. After receiving the events, the MCU converts the events to 40-bit packets and then transmits them to SpiNNaker by using 2-of-7 self-timed coding and 2-phase handshaking protocol. Next, the SNN processes the injected spikes and exports the output spikes to the MCU using the same coding and protocol. Finally, the MCU generates servo commands based on received spikes to control the motor to intercept the moving object. The maximum speed of the closed-loop system is $\sim$3000 $packets \cdot s^{-1}$ for both uplink and downlink, and the robot can intercept a slowing moving ball with an accuracy of $\sim$85%. Additionally, the latency from capturing events to generating the corresponding command is 6.847 ms. The robotic system has high efficiency due to its adaptive transmission speed and the inherently sparse events and spikes. For further development, the system should work independently without a PC and a more powerful MCU could be used to increase the transmission speed.

Acknowledgment

I would like to express my sincere gratitude to my supervisor: Dr Konstantin Nikolic. It is his guidance that inspires me to explore the field of neuromorphic engineering. It is his instructions that enlighten me to think in an academic research way. I always get quick and positive replies for him.

I would also like to appreciate Mr Vic Boddy, the Project Lab manager. His excellent support enables me to monitor the status of the whole robotic system designed in this project.

I would like to appreciate Mr Andrew Rowley, a PhD of SpiNNaker Team. He showed me the detailed communication protocol between SpiNNaker and external devices.

Finally, I would like to thank my classmates for the time spent with them in the lab and the central library.

Contents

Abstract **3**

Acknowledgment **5**

Contents **7**

List of Figures **9**

List of Tables **11**

Abbreviations **13**

Chapter 1. Introduction **15**

1.1 Motivation 15

1.2 Objective 16

1.3 The MCU board used in this project 17

1.4 Report overview 17

Chapter 2. Literature Review **19**

2.1 SNNs applied in classification 19

2.2 Robotics based on SpiNNaker 20

2.3 Other interfaces 22

2.3.1 FPGA interface 22

2.3.2 MCU interface 22

2.3.3 Comparison 23

2.4 Goalkeeper without SpiNNaker 24

Chapter 3. Methodology for Hardware Communication **25**

3.1 Communication between MCU and SpiNNaker 25

3.1.1 2-of-7 coding 26

3.1.2 Decoding SpiNNaker packets 27

3.1.3 Encoding SpiNNaker packets 29

3.2 Communication between MCU and DVS 30
3.3 Motor control 32
3.4 Synchronize events, packets and commands 33

Chapter 4. Implementation **37**
4.1 The robotic system 37
4.2 The order of booting devices 38
4.3 Inform SpiNNaker to receive and send spikes 39
4.4 SNN runs in SpiNNaker 40
4.5 The mechanism of controlling the servo motor 41

Chapter 5. Result **43**
5.1 Receiving packets from SpiNNaker 43
5.2 Sending packets to SpiNNaker 46
5.3 The closed-loop data transmission 48
5.3.1 Latency of the system 50

Chapter 6. Evaluation **53**
6.1 Accuracy 53
6.1.1 Resolution 53
6.1.2 Horizontal projection 54
6.1.3 Payload 55
6.2 Efficiency 55
6.2.1 Improve motor control mechanism 55
6.2.2 Independence 56
6.2.3 adaptive transmission speed 56
6.3 Further development 57

Chapter 7. Conclusion **59**

Bibliography **61**

List of Figures

1.1 Architecture of the hardware communication 16
1.2 Arduino Due board 17

2.1 8 classes of the locations of arrival 19
2.2 Architecture of the line follower robot system 20
2.3 The PushBot (The MCU connected to external devices) 21
2.4 The FPGA interface board 22
2.5 Layout of the interface board designed by TUM 23

3.1 The 8-channel bidirectional logical level shifter (Model: TXS0108E) 25
3.2 2-phase handshake 26
3.3 Formats of the two different types of SpiNNaker packets 28
3.4 The format of header 28
3.5 Recorded events of still picture (left) and recorded events of a rolling ball (right) 30
3.6 Mapping an event to a 40-bit packet 31
3.7 Downsampling using a 8×8 average mask 32
3.8 The position of a servo motor (right) and its theoretical corresponding pulse width (left) 33
3.9 Simplified logic for synchronizing events, packets and commands when the max speed of storing events is 2000 events per second (Grey lines do not represent data flow) 34

4.1 The Neuromorphic Robotic System 37
4.2 The DVS128 (Left) and the embedded DVS (Right) 38
4.3 The architecture of the neuromorphic robotic system with PC 38
4.4 Software connection for SpiNNakerLink 39
4.5 The SNN running on SpiNNaker 40
4.6 The trajectory of the robotic arm 41
4.7 The mechanism of controlling the servo motor (the process of decoding packets is not included) 41

5.1 Oscilloscope reading of Lout[6] ∼ Lout[0] (the $2^{nd} \sim 8^{th}$ lines in the Figure) and LoutACK (the first line) 44
5.2 Oscilloscope reading of Lout[6]∼Lout[0] (the $2^{nd} \sim 8^{th}$ lines in the Figure) and LoutACK (the first line) with a smaller time scale 44
5.3 The initial states of Lin[6]∼Lin[0] (the $2^{nd} \sim 8^{th}$ lines in the Figure) and LinACK (the first line) 47
5.4 The spikes received by SpiNNaker at the speed of 50 $packets \cdot s^{-1}$ 47
5.5 The spikes received by SpiNNaker at the maximum speed 48
5.6 The spikes received by SpiNNaker in two simulations 49
5.7 The maximum speed of downlink (yellow) and uplink (green) 50
5.8 Parallel execution of the system 51

6.1 Spikes received by SpiNNaker with the input neuron population size of 1024 (Top) and 4096 (Bottom) 53
6.2 Horizontal projections of the 8 positions 54

List of Tables

3.1 Connection between MCU and SpiNNakerLink for data input 26
3.2 Connection between MCU and SpiNNakerLink for data output 26
3.3 2-to-7 coding: converting a symbol into 2 changing bits 27

5.1 Spike source array of the input layer 43

Abbreviations

SNN:	Spiking Neural Network
DVS:	Dynamic Vision Sensor
eDVS:	embedded Dynamic Vision Sensor
ACK:	Acknowledgement
MCU:	Microcontroller Unit
FPGA:	Field Programmable Gate Array
CPLD:	Complex programmable logic device
EOP:	End of Packet
AER:	Address-Event Representation
PWM:	Pulse Width Modulation
AI:	Artificial intelligence
GPIO:	General-Purpose Input/Output
DAC:	Digital-to-Analogue Converter
IDE:	Integrated Development Environment
STDP:	Spike-Timing-Dependent Plasticity
SWAT:	Synaptic Weight Association Training

Chapter 1

Introduction

1.1 Motivation

Neuromorphic engineering corresponds to human cognition, which is the third generation of AI. The first generation of AI is rules-based and suited to narrowly defined problems [1], such as process monitoring and efficiency improving. The second generation is focused on sensing and perception [2]. It can be applied to understand the content of a video, for instance. The third generation is designed to emulate the human brain to automate human activities and address abstraction [3].

Spiking neural networks are performed in neuromorphic engineering to map synapses between neurons with regards to stimuli dynamically. Contrastive to activate all neurons in multi-layer perceptron neural networks, a neuron in SNNs is only fired independently when its membrane potential reaching a threshold [4]. Once a neuron is activated, it will transmit an impulse signal to connected neurons that directly increase or decrease their membrane potential. The information in SNNs is encoded by the signal itself and its timing [5]. Due to these features, neuromorphic computing exhibits low response latency and low energy consumption.

To utilize the advanced capacity and efficiency of neuromorphic engineering, this project applies it to a robotic system. For the system, SNNs run in a neuromorphic processor, SpiNNaker [6]. It can simulate a SNN of thousands of spiking neurons, and

process sparse input and generate spikes to control a motor in real-time. The system will have a deeper perception and judgment of input signals, and a high-efficiency control mechanism for the robotic arm.

1.2 Objective

This project aims to build a neuromorphic robotic system that running spiking neuron networks. For the system, the input signal is from a DVS, the signal is then passed to the processing unit, SpiNNaker, which simulates SNN based on hardware. Finally, the output data from SpiNNaker is used to control a servo motor. To establish the communication between DVS, SpiNNaker and servo motor, a MCU is used. As shown in Figure 1.1, the MCU should perform four communications in real-time.

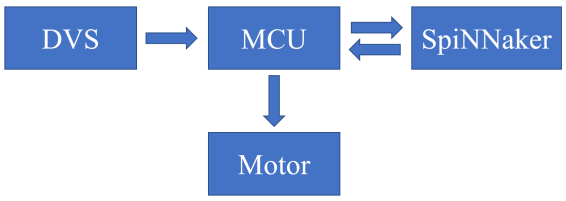

Figure 1.1: Architecture of the hardware communication

The order of data transmission through the MCU is

1. Receive data from DVS
2. Send data to SpiNNaker
3. Receive data from SpiNNaker
4. Release motor control commands

After the system is built, it will be applied to a real-time application to evaluate its performance. In the application, the system captures the movement of a moving ball and then predicts its location of arrival. Finally, the servo motor moves a baffle to intercept the moving ball when it arriving the destination.

1.3 The MCU board used in this project

To accomplish the objective, the MCU board should have enough GPIO pins for connecting SpiNNaker, DVS and motor, and have sufficient computing power that can parallelly receive, transmit, decode and encode data in real-time. Considering these aspects, an Arduino Due board is adopted (Figure 1.2). It is the first board in Arduino based on a 32-bit ARM core [7].

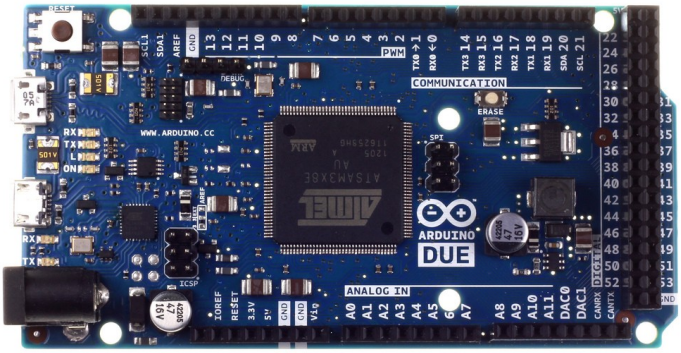

Figure 1.2: Arduino Due board

This board has 54 GPIO pins that are not only met the requirement of connecting SpiNNaker, DVS and motor, but also can connect more external devices simultaneously. Its clock speed is 84 MHz, which is 5.25 times faster than the Arduino UNO board used in the previous project. Furthermore, the board has 2 built-in DACs that enables it to offer the power supply to the neuromorphic robotic platform at various voltages. Additionally, the Arduino IDE has a built-in library for controlling servo motors.

1.4 Report overview

This report focuses on the design, implementation, and evaluation of the neuromorphic robotic system. Firstly, the literature review on SNNs, robotics that based on SpiNNaker and other interfaces are given to show current research on this area. Chapter 3 will then state the methodology for the hardware communication between DVS, SpiNNaker and the servo step by step. After describing the design for the robotic system, its

implementation of the goalkeeper will be shown in Chapter 4. The result of the hardware communication and the implementation will be presented in Chapter 5. Since the result is obtained, it will be evaluated and discussed in Chapter 6 based on accuracy, efficiency, and independence. Finally, Chapter 7 will conclude the project and this report, and give the inspirations for further development. Additionally, 3 data formats will be involved in this report. To avoid confusion, the meaning of the 3 formats is explained here.

- Events: the data sent from DVS
- Packets: the data sent to and received from SpiNNaker
- Commands: the data used to control the servo motor

Chapter 2

Literature Review

2.1 SNNs applied in classification

The designed neuromorphic robotic system will be implemented to intercept a moving ball, its location of arrival to the interception point can be divided into different classes (Figure 2.1). Thus, the SNN run in SpiNNaker performs a classifier.

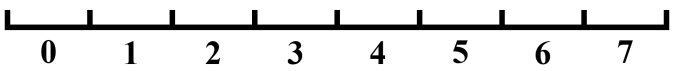

Figure 2.1: 8 classes of the locations of arrival

To train a SNN classifier, algorithms based on STDP rule are proposed. SpikeProp is the first supervised algorithm for training SNNs [8]. It applies a modified gradient-descent-based error-backpropagation method [9], which solves the inherent problem of SNNs by allowing every neuron to fire only once in the gradient descent training. However, this algorithm is not suitable for the SNNs training with multiple-spikes patterns, because neurons are allowed to fire at most once. For another mathematical training algorithm, a probabilistic gradient descent approach is performed [10]. Its training process has an extra supervisory signal, but it is also unable to learn patterns since only single spikes are utilized.

Apart from the mathematical perspective, SNNs training algorithms should be in

line with Hebbian theory [11], which training SNNs in a biological approach. These algorithms using the temporal correlation of postsynaptic and presynaptic spikes to update synaptic weights. Based on this mechanism, the SWAT algorithm is proposed for training SNNs for classification [12]. In each epoch of SWAT, the training data is passed to every training neuron and the synapse associated with it. The weights will be fixed once the average firing activity of training neurons meets the stopping criteria. The SWAT algorithm achieves an accuracy of 95.3 % and 96.7% for the Iris and Wisconsin testing sets [12]. Considering the high accuracy exhibited by SWAT, the algorithm can be applied to training the SNN classifier for this project.

2.2 Robotics based on SpiNNaker

Currently, robots based on SpiNNaker can only perform simple tasks, these robots do not have complex logic and deep perception. The APT group, University of Manchester developed a line follower robot based on SpiNNaker and DVS [13]. Firstly, the input signal from DVS is sub-sampled to fit the processing speed of SpiNNaker. Then, a three-layer leaky integrate-and-fire SNN is running on SpiNNaker to control the robot to move in the central axis of the line. As shown in Figure 2.2, the system uses a PC to convert protocols between DVS, SpiNNaker and wheel motors, but the data transmission speed is not stated in the published paper. Compare to this system, the platform designed in this project uses MCU as a communication protocol converter, which is more suitable for mobile robots.

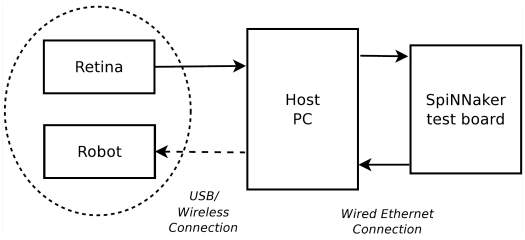

Figure 2.2: Architecture of the line follower robot system [13]

Apart from this project using MCU, a SpiNNaker robotic platform, called SpOmnibot, uses MCU for data format conversion and it is developed by Technical University of

Munich [14]. SpOmnibot is applied to an application that moving the robot to the most active stimulus. For example, two lights are flashing with different frequencies, the robot will approach the light with higher frequency. To perform this application, SpiNNaker of the robot runs a "Winner Takes All" SNN [15]. The network winner is the most active neuron, the only neuron can reach its spiking threshold. The result shows the network performs a robust max selection, and the robot system is running in real-time [14].

Different from the above robot system that uses one MCU for communication, a mobile robot called PushBot uses two MCUs, one MCU is connected to SpiNNaker, the other is connected to external devices [16]. PushBot has an extra communication, which is the wireless communication between the two MCU through WI-FI.

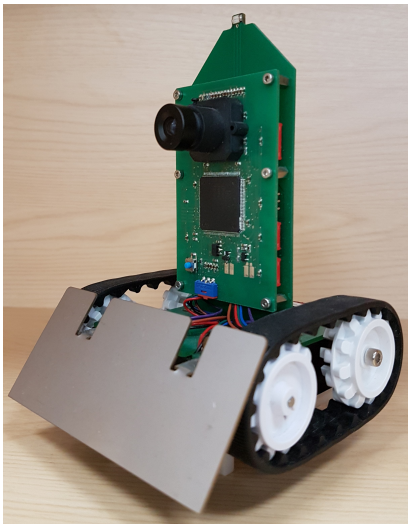

Figure 2.3: The PushBot (The MCU connected to external devices)

As shown in Figure 2.3, a MCU of PushBot is connected to an eDVS and two wheel motors. It sends DVS data to and receives motor commands from the other MCU through WIFI. The other MCU works as a translator. It converts the communication protocol between the MCU connected to external devices and SpiNNaker. The size of the robot is reduced and the movement of PushBot becomes more flexible by applying this separation mechanism.

2.3 Other interfaces

2.3.1 FPGA interface

There is a FPGA solution to interface SpiNNaker with DVS. This solution only feeds AER data into SpiNNaker, it cannot receive any information from SpiNNaker. As shown in Figure 2.4, a RaggedStone2 FPGA board is used for the unidirectional data transmission.

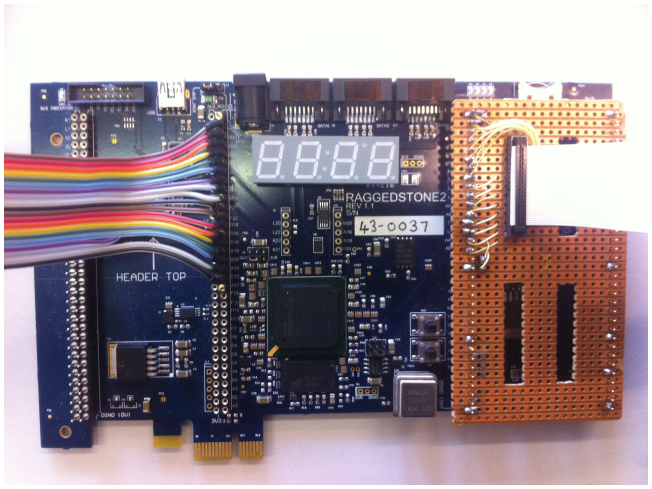

Figure 2.4: The FPGA interface board

2.3.2 MCU interface

Apart from FPGA, Technical University of Munich (TUM) and The University of Manchester have designed a MCU interface with bidirectional data transmission. In this solution, a CPLD is used as a communication protocol converter between SpiNNaker and MCU. The data to or from SpiNNaker is encoded through a 7-bit data bus, and it will be changed to a 9-bit data bus after the conversion of CPLD.

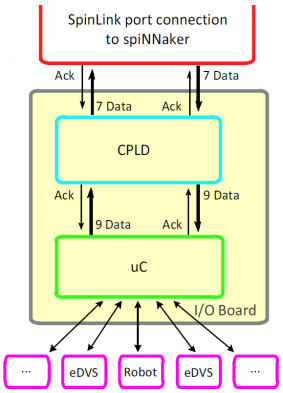

Figure 2.5: Layout of the interface board designed by TUM [14]

As described in Figure 2.5, the MCU keeps receiving available data from CPLD and connected devices, and then exchanging the two formats of data into the other format, respectively. The solution has been applied to a "Winner-Takes-All" SpiNNaker robot task. The robot is able to identify the most salient stimulus from the DVS input.

2.3.3 Comparison

The FPGA solution is based on a RaggedStone2 board, which is a high-cost development kit. Furthermore, the interface only can send data to SpiNNaker, and the data flow is unidirectional. Compared to this solution, the platform designed in this project has bidirectional data flow and can receive data from various sensors that support Arduino and feed data to multiple actuators.

The MCU interface with CPLD requires two data format conversions and two chips to transfer symbols, which means a higher power consumption. Furthermore, this mechanism also increases the difficulty of debugging and further development. The interface board has 5 peripheral ports pre-defined for 2 DVSs and 3 motors, which simplifies the connection of DVSs and motors, but also resulting in a limited capacity for other external devices and sensors, such as optical flow sensor. Contrastively, the platform designed in this project uses an Arduino Due board that has 54 GPIO pins and plenty of built-in libraries for external devices. Additionally, the platform does not need an assistant from the second chip, such as CPLD. On the other side, the MCU used in the designed plat-

form is based on ARM Cortex-M3 with a clock speed of 84 MHz, while the MCU interface designed by TUM and the University of Manchester is based on ARM Cortex-M4 with a clock speed of 168MHz [14]. Compared to the MCU interface, the designed platform has a better capacity of external devices, but a lower processing speed. According to the published paper, the total uploading and downloading speed of the MCU interface is 1.1 Mega packets per second [14].

2.4 Goalkeeper without SpiNNaker

The neuromorphic robotic platform designed in this project will be applied to predict the trajectory of a moving ball and then intercept it. A similar application, called robotic goalie, is implemented by University of Zurich and ETH Zurich [17]. The application uses a desktop computer to process the data from DVS and control a servo motor. According to the published paper, the latency from recording ball movement to generating motor commands is 2.2 $\pm$ 2 ms [17]. SpiNNaker and SNN are not performed in the robotic goalie. Since neuron networks are not used in the application, it is only suitable for linearly moving objects. Furthermore, the robotic goalie is not portable, because it requires a computer for processing data.

Chapter 3

Methodology for Hardware Communication

3.1 Communication between MCU and SpiNNaker

The SpiNNaker-3 board has two ports, named SpiNNakerLink0 and SpiNNakerLink1, for data transmission with external devices. Each port has 34 pins: 8 for data input; 8 for data output; the rest 18 pins are connected to ground. The logical high of SpiNNakerLink is 1.8 volt. It is different from the logical high of the MCU, which is 3.3 volt. In order to be compatible with the two logical levels, level shifters are used. As shown in Figure 3.1, the level shifter has 8 bidirectional channels. Thus, two level shifters are required: one for data input, the other for data output. The MCU has two build-in digital-to-analog converters, which are used to provide reference voltages for the level shifters.

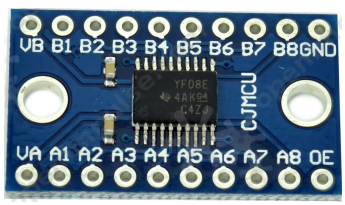

Figure 3.1: The 8-channel bidirectional logical level shifter (Model: TXS0108E)

Both input and output data buses of SpiNNakerLink are connected to the MCU pins after passing level shifters. The connections of the two buses are described in Table 3.1 and Table 3.2.

Table 3.1: Connection between MCU and SpiNNakerLink for data input

Signal	Lin[6]	Lin[5]	Lin[4]	Lin[3]	Lin[2]	Lin[1]	Lin[0]	LinACK
MCU Pin	22	23	24	25	26	27	28	29
SpiNNakerLink	2	4	6	8	10	12	14	16

Table 3.2: Connection between MCU and SpiNNakerLink for data output

Signal	Lout[6]	Lout[5]	Lout[4]	Lout[3]	Lout[2]	Lout[1]	Lout[0]	LoutACK
MCU Pin	2	3	4	5	6	7	8	9
SpiNNakerLink	33	31	29	27	25	23	21	19

3.1.1 2-of-7 coding

After the physical connection between the MCU and SpiNNakerLink. The communication protocol should be considered. SpiNNakerLink uses a 2-phase handshake protocol, which is an asynchronous communication. In 2-phase handshaking, the sender firstly sends a symbol to the receiver and then waits for the acknowledgement signal that sent from the receiver. For the receiver, once it receives a symbol from the sender, it will send an acknowledgement signal back to the sender (Figure 3.2).

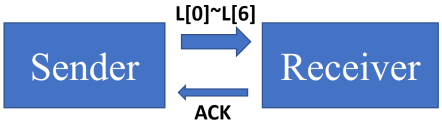

Figure 3.2: 2-phase handshake

There are total 17 symbols available to be sent in the data transmission. These symbols consist of 16 hexadecimal numbers (0 to F) and an 'End of Packet' symbol. In every symbol transmission, the sender only changes the logical level of 2 wires and the keep the rest 5 wires unchanged. This is the reason that the protocol named 2-of-7 coding [18]. Furthermore, after sending a symbol, the logical levels of the 7 data wires (L[0] L[6]) and the ACK wire will keep unchanged and not return to the initial state which is logical low

(0). Thus, this protocol is also named Non-Return-to-Zero [19]. Table 3.3 describes the relationship between a symbol and its corresponding two changing bits. In Table 3.3, '1' represents changing bits and '0' represents unchanged bits.

Table 3.3: 2-to-7 coding: converting a symbol into 2 changing bits

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

For example, assume the initial states of the 8 wires are:

L[6]	L[5]	L[4]	L[3]	L[2]	L[1]	L[0]	LACK
LOW	LOW	LOW	LOW	LOW	LOW	LOW	LOW

To send symbol '0', refer to Table 3.3, the changing bits are L[0] and L[4], the rest bits keep unchanged. Furthermore, after receiving the symbol '0', the receiver should change the logical level of the ACK bit to send acknowledgement. Thus, the states of the 8 wires after sending symbol '0' are:

L[6]	L[5]	L[4]	L[3]	L[2]	L[1]	L[0]	LACK
LOW	LOW	HIGH	LOW	LOW	LOW	HIGH	HIGH

3.1.2 Decoding SpiNNaker packets

Once symbols are received by the MCU, the next step is to decode the data sent from SpiNNaker. The information of spikes is sent in the form of SpiNNaker packets. There are two kinds of SpiNNaker packets: 40 bits and 72 bits. Each packet is followed by an 'EOP' symbols to indicate that the transmission of the current packet is finished.

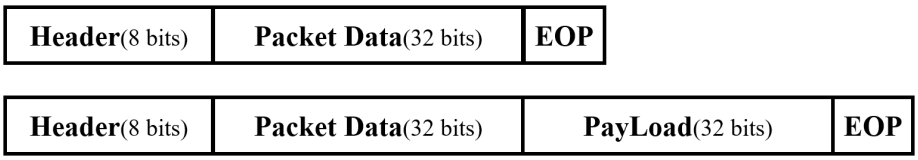

Figure 3.3: Formats of the two different types of SpiNNaker packets

As shown in Figure 3.3, the 72-bit packet has a 32-bit additional PayLoad than the 40-bit packet. Both packets contain a header and packet data. The spike information is in the packet data, and the header shows the specification of the packet. During simulation, SpiNNaker is sending two different types of packets to the MCU, namely nearest neighbour (NN) packets and Multi-cast (MC) packets. The NN packet does not contain any information about spikes, it is sent when SpiNNaker-3 board is initializing. Contrastively, MC packets contain the information of neurons that spiking. To distinguish the two types of packets, the last two bits of header are used (Figure 3.4). The last two bits indicate the packet's type: '01' for NN packets and '00' for MC packets.

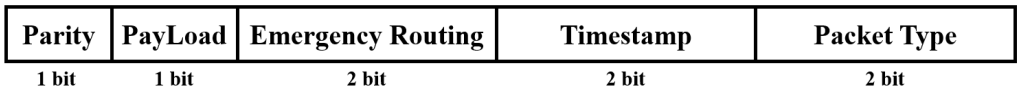

Figure 3.4: The format of header

The MCU keeps storing the received symbols into a buffer until it receives an EOP symbol. As mentioned in section 3.1.1, there are 17 different types of symbols: 16 Hex (Hexadecimal) numbers and EOP. A packet only contains Hex numbers; EOP symbols are used to separate packets. Thus, the MCU will do Hex to binary conversion after receiving an EOP symbol. Standard Hex to binary conversion starts from the most significant bit. For example, the Hex symbol 'A' will be converted to '1010'. However, the conversion of SpiNNaker packets is in the opposite way, which starts from the least significant bit. In decoding SpiNNaker packets, the Hex symbol 'A' will be converted to '0101'. For instance, symbols received by the MCU are:

2 8 3 0 8 9 D 0 3 8 7 A 6 E 8 F 5 B EOP

After Hex to binary conversion, the decoded packet is:

01000001–11000000000110011011000011000001–11100101011001110001111110101101

This is a 72-bit NN packet, and it is in the form of 'Header-packet_date-payload'. In its header, the parity bit is cleared (0) to ensure the whole packet has an odd parity of 1s. The payload bit is set (1), since it is a 72-bit packet that has a payload. The last two bits of its header are '01', which indicates it is a NN packet. This packet is sent when the SpiNNaker is booting. In simulation, the MCU is receiving 40-bit MC packets, such as:

1 0 D B 0 0 0 0 0 0 EOP

After Hex to binary conversion, the decoded packet is:

1000000–1011110100000000000000000000000

In the header of this packet, its parity bit is set (1) to ensure the number of '1' in the packet is odd. The payload bit is cleared (0), since it is a 40-bit packet. The last two bit of the header is '00', which indicates it is a MC packet. Both the emergency routing and timestamp in the header are '00', because the two parts are not used in the simulation. To obtain the neuron that spiking, the packet data is converted from binary to decimal: $10111101_2 \longrightarrow 189$. Therefore, the spiking neuron is the 189^{th} neuron.

After receiving all MC packets, the sequence of spiking is obtained. The next step is to determine when the neurons are spiking. It is found that the time at which a spike is received is related to the time of the spike generated. If the time difference between two adjacent spikes is 50 milliseconds, the time difference of receiving the two corresponding packets is also 50 milliseconds. Thus, the time of every spike can be determined if force the first spike happened at 0 millisecond.

3.1.3 Encoding SpiNNaker packets

To inject spikes to SpiNNaker, the information of spikes is encoded as 40-bits MC packets. The first thing is to define a routing key. The SpiNNaker regards the external device as a virtual chip on its system. Thus, it requires a virtual routing key as a delivery address of the injected packets. In this project, the virtual routing key is defined as '1234', which are 4 Hex numbers. For example, to send the spike of the 12^{th} neuron to SpiNNaker, the corresponding packet in Hex is:

0 0 D 0 0 0 4 3 2 1 EOP

The packet in binary is:

0000000–00000011000000000010110001001000

The order of the virtual routing key is reversed because SpiNNaker reads packets starts from the end of packets. The MCU is required to count the number of '1' in the binary format of the packet, and then determine to set or clear the parity bit to ensure the whole packet has odd parity.

3.2 Communication between MCU and DVS

DVS is only sensitive to movement. If an object move and the rest keep unchanged, DVS will only generate data of the location of the movement. The data sent from DVS is in the Address-Event Representation. In this representation, the movement of objects is also named event. The information of the event is sent in the form of abscissa (x) and ordinate (y).

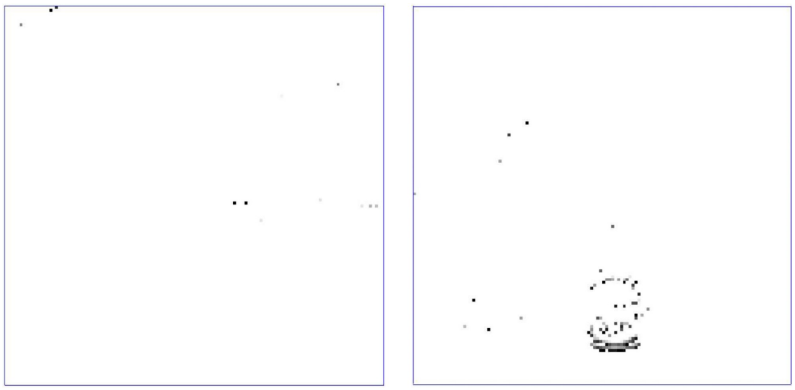

Figure 3.5: Recorded events of still picture (left) and recorded events of a rolling ball (right)

As shown in Figure 3.5, the recorded data has some noise, even when recording still images. Noise can be reduced by increasing the threshold of spikes, however, it can not be eliminated. In the recorded data, DVS will only send the location of the black dots and the blank area does not generate any data (Figure 3.5).

To sent events to the MCU, an embedded DVS should be used. However, the

embedded DVS does not work. As an alternative method, events are firstly received by a software called jAER, and then passed to the MCU. jAER is developed by the DVS team and its original edition does not support sending events out. The source code of jAER has been modified to send events to the MCU through a serial port in real-time. Once jAER receives an event from DVS, it will send the event to the MCU in the form of abscissa and ordinate. For the MCU, it should send events to SpiNNaker immediately, after receiving them.

To send events to SpiNNaker, the coordinates of events should be encoded in the form of SpiNNaker packets. The resolution of DVS is 128×128. Therefore, 7 binary bits are needed for encoding abscissa and ordinate of events, respectively. For a 40-bit packet, 8 bits are used for header, 16 bits are used for the virtual routing key and the rest 16 bits can be used to encode the location of events (Figure 3.6).

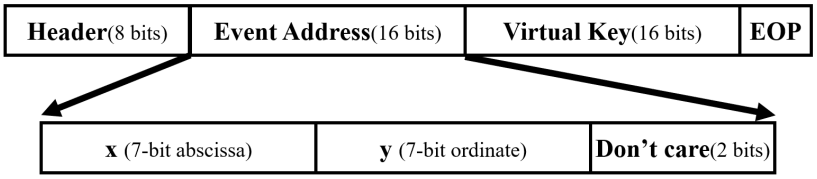

Figure 3.6: Mapping an event to a 40-bit packet

In the resolution of 128×128, mapping x and y coordinates of an event requires 14 bits. The rest 2 bits are unused and set to 0 in this project. As mentioned in section 3.1, the virtual routing key is '1234'. Additionally, both the decimal to binary conversion and Hex to binary conversion start from the least significant bit. Thus, to send the event whose coordinates is $(56, 78)$, the corresponding packet is:

1000 0000	0001110	0111001	00	0010 1100 0100 1000
header	x	y	unused	virtual routing key

The resolution can be reduced to increase events transmitting and processing speed. For example, an average mask of the size 8×8 can be applied to reduce the resolution from 128×128 to 16×16 (Figure 3.7). After applying the mask, each pixel is the average of 64 original pixels and a threshold can be set to determine the pixel is spiking or not.

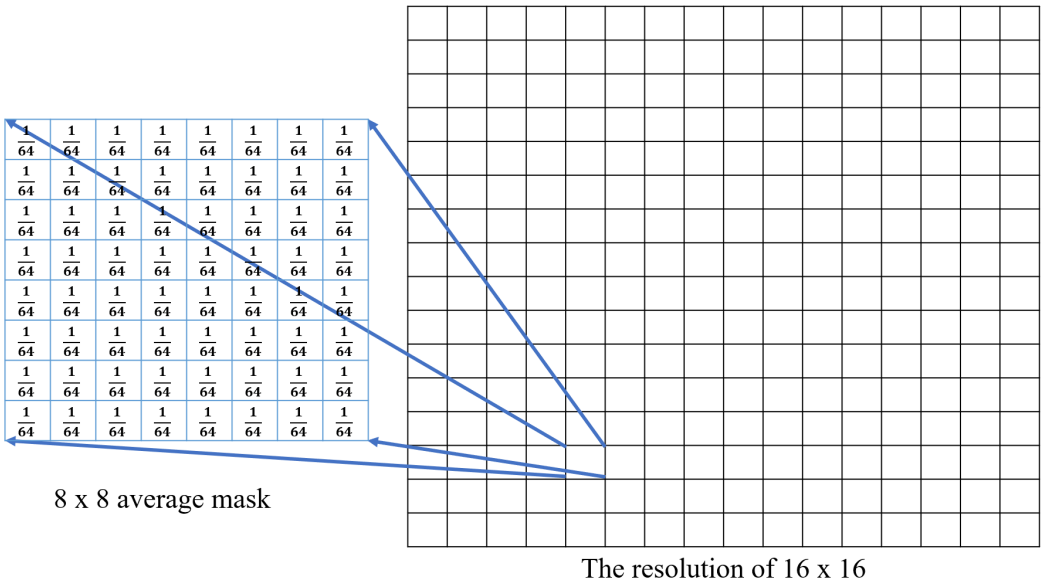

Figure 3.7: Downsampling using a 8×8 average mask

For the resolution of 16×16, 4 bits are needed to encode each coordinate. Thus, the rest 8 bits of the event address are not used and set to 0. For instance, the 40-bit packet of the event at $(5, 12)$ is:

0000 0000	1001	0011	0000 0000	0010 1100 0100 1000
header	x	y	unused	virtual routing key

3.3 Motor control

Events recorded by DVS can be sent to SpiNNaker through MCU and MCU can receive packets from SpiNNaker. The next step is to control the motor to rotate to the desired position based on packets received by MCU. To precisely control the position, a servo motor is used. The servo is controlled by a signal from MCU that using PWM. For servo motors, the position is proportional to the pulse width [20]. The minimum pulse width of the servo is 1 ms, and the maximum one is 2 ms. MCU controls the position through setting the pulse with of the control signal (Figure 3.8).

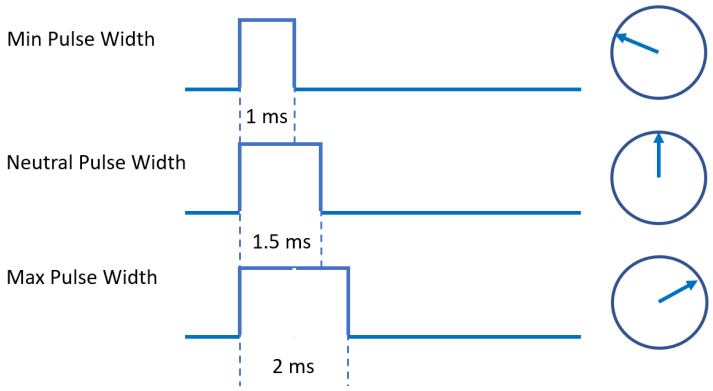

Figure 3.8: The position of a servo motor (right) and its theoretical corresponding pulse width (left)

It takes a certain time for the motor to rotate to the desired position, and the motor can not receive the next command during rotating. After measuring, the speed of the servo motor is 120^o per 150 ms. Thus, MCU will execute a control command every 150 ms. If the generating speed of commands is higher than this, the unexecuted commands will be stored into a buffer and the motor control is not in real-time. To ensure that the motor responds quickly to the packets received by MCU, only one command is stored in the buffer and it will be overwritten by newly generated commands. Detailed commands execution logic will be discussed in the next section.

3.4 Synchronize events, packets and commands

After the above three sections, the MCU can convert protocol between events of DVS and packets of SpiNNaker, and control the servo motor based on received packets. The next step is to synchronize events, packets, and commands, that is to ensure the SpiNNaker receives the latest events from DVS and ensure motor commands are generated based on the latest received packets.

Since DVS generates fewer events with capturing fewer movements, the MCU can not receive events at a specific speed. In extreme cases, DVS only generates several events per second. For SpiNNaker, it is only sending neuron information when the neuron is spiking. Therefore, the speed of receiving packets from SpiNNaker is uncertain. Further-

more, the speed of sending packets to SpiNNaker is dependent on the speed of receiving events from DVS, and it is also uncertain. Additionally, the motor commands are generated based on received packets, its generating speed is varying with the vary of receiving packets speed. Succinctly, four speeds are always changing, namely:

- The speed of receiving events
- The speed of sending packets
- The speed of receiving packets
- The speed of generating motor commands

To synchronize the four speeds, the built-in timer of the MCU with 1 microsecond precision is used. Once the neuromorphic robotic system is booted, the MCU keeps receiving events from DVS at varying speeds. Instead of immediately converting the received event into spike and sending it out, the MCU stores received events into a buffer periodically. The buffer will be overwritten from the beginning if an overflow happens. Assume the max speed of storing events is set to 2000 events per second, the MCU will keep ignoring the received events until the time difference between the currently received event and the last stored event is greater than or equal to 500 microseconds (Figure 3.9).

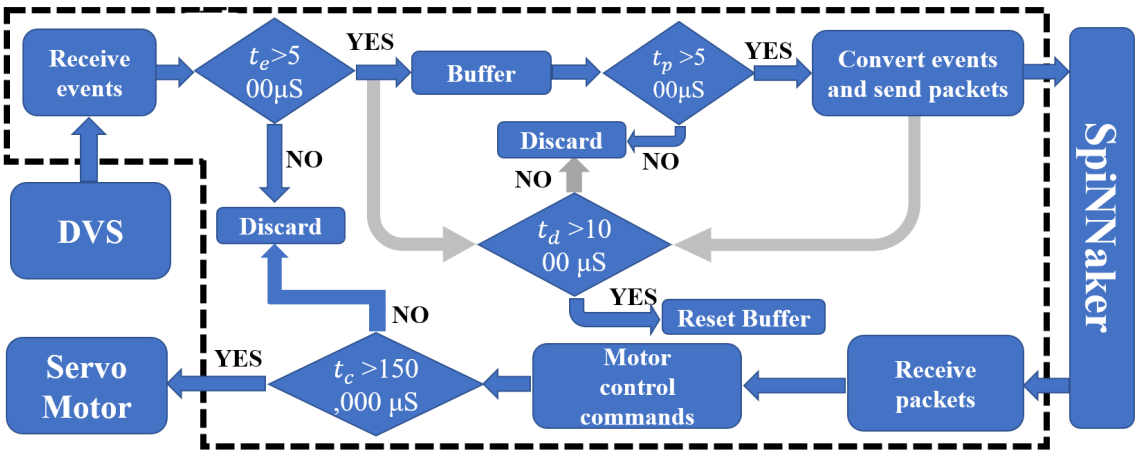

Figure 3.9: Simplified logic for synchronizing events, packets and commands when the max speed of storing events is 2000 events per second (Grey lines do not represent data flow)

The max speed of sending packets should be the same as or slower than the max speed of storing events, because the MCU firstly stores and converts events, and then sends packets. Again, assume the max speed of storing events is 2000 events per second, the MCU will not send packets until the time difference between the current time and the time of the last sent packet is greater than or equal to 500 microseconds (Figure 3.9). If all events in the buffer are converted to packets and sent, the MCU will not send packets until a new event is stored into the buffer. The final thing is to ensure the latest events are converted and sent to SpiNNaker. Since SpiNNaker requires time to boot and the MCU sends packets from the beginning of the buffer, there is a time difference between the stored events and the sent packets. To solve that, the MCU will clear all stored events and reset the buffer if the time difference is greater than 1000 microseconds.

The speed of receiving packets from SpiNNaker does not affect the system. Because the time step of SNN simulation is 1 millisecond, which is slower than the max speed of sending packets to SpiNNaker. Besides, the MCU will not execute the motor control command and keep overwriting it until time difference between the currently generated command and the last executed command is greater than or equal to 150,000 microseconds (Figure 3.9). Thus, the speed of generating commands also does not affect the system.

Chapter 4

Implementation

4.1 The robotic system

The circuit for connecting SpiNNaker and the MCU is built on breadboard. As can be observed in Figure 4.1, the circuit also contains two LEDs to monitor the transmission state of the system.

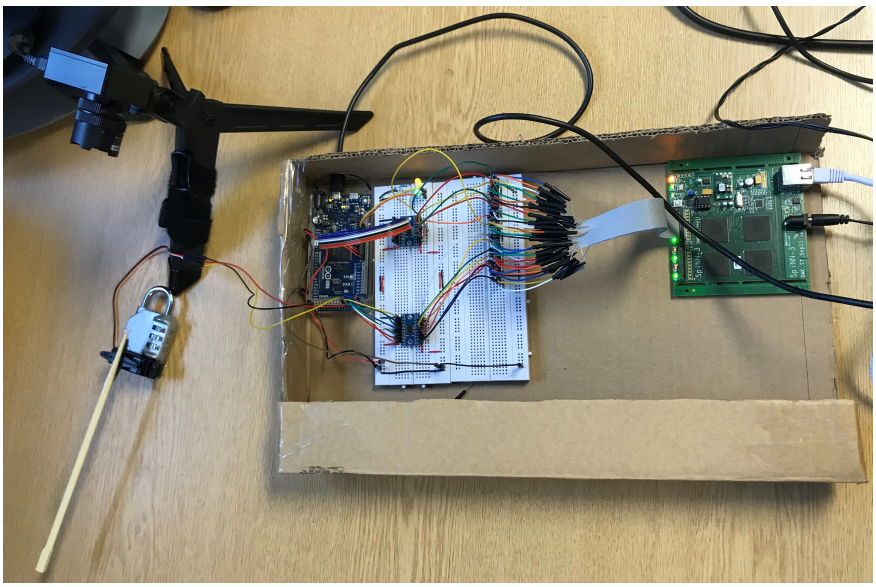

Figure 4.1: The Neuromorphic Robotic System

- The yellow LED flashes at the same speed as receiving events
- The green LED flashes at the same speed as sending packets

If the data transmission speed is faster than a threshold, the two LEDs are always on visually. Furthermore, the state of receiving packets is reflected in the motor rotation.

4.2 The order of booting devices

In the initial design, an embedded DVS should be connected to the MCU to transmit events directly. However, the embedded DVS does not work, probably because some parameters of its communication protocol have been modified by previous students. As an alternative solution, a DVS128 is used, which only has one USB port for data communication (Figure 4.2).

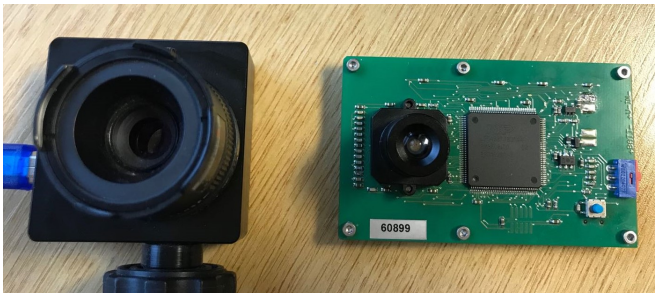

Figure 4.2: The DVS128 (Left) and the embedded DVS (Right)

Since the MCU does not have a USB port to connect DVS128, a PC is used to pass the events from DVS128 to the MCU. A new thread is added to the software jAER, which enables DVS128 to send events out through a serial port while displaying them. It is worth noting that the PC only performs events transmission, it does not process any data. As shown in Figure 4.3, the architecture of the system now consists of a PC between DVS and the MCU.

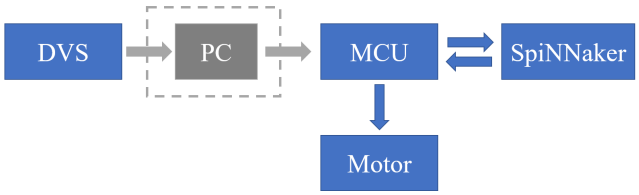

Figure 4.3: The architecture of the neuromorphic robotic system with PC

For the new architecture, the order of booting the system is

1. Power on SpiNNaker
2. Connect DVS128 and the MCU to a PC
3. Open the modified jAER on the PC (The yellow LED starts to flash)
4. Press the reset button on SpiNNaker (The green LED starts to flash)

At the moment of SpiNNaker is powered on, the logical levels of SpiNNakerLink pins will change randomly and then return to the initial values. Thus, the first step is to boot SpiNNaker. Then, DVS128 is booted to generate events for the MCU to convert to packets and prepare to be sent to SpiNNaker. Next, the reset button on SpiNNaker is pressed to give a logic level change of the ACK signal to start sending packets to SpiNNaker. After that, the robotic system is ready to use, because receiving packets from SpiNNaker and motor control will be performed automatically by the MCU.

4.3 Inform SpiNNaker to receive and send spikes

SpiNNaker regards external devices as virtual chips on its system, and it will not response to external devices if the program running on SpiNNaker does not define these virtual chips. To send spikes out, a class is defined for the output neuron population. As described in Figure 4.4, the class is the extension of the model "ApplicationSpiNNaker-LinkVertex" [21].

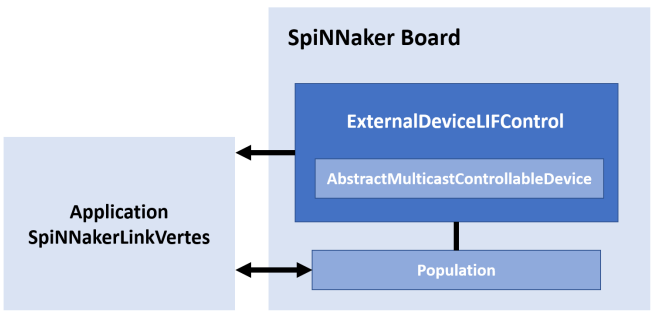

Figure 4.4: Software connection for SpiNNakerLink [21]

Since SpiNNaker-3 has two SpiNNakerLink ports, the class should declare which port is used. In this robotic system, either of the two ports can be used as long as the physical connected port is the same as the port defined in the class. Similar to sending spikes, another class is defined for injecting spikes. The difference is the class for injecting spikes should declare an additional virtual routing key as the delivery address of the injected spikes. As mentioned in Chapter 3, the virtual routing key used in this project is set to "1234". After defining the classes for the input and output neuron populations, SpiNNaker can send spikes to and receive spikes from external devices in the form of 40-bit packets.

4.4 SNN runs in SpiNNaker

Since this is a hardware project, a fairly simple SNN runs on SpiNNaker. The network only utilises the horizontal axis of the input events, which is divided into 8 equal parts, one for each input neuron. As shown in Figure 4.5, the output population also has 8 neurons that are directly connected to the input population. The connection method is the one-to-one connection.

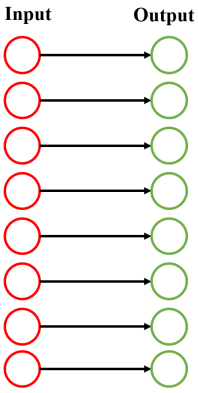

Figure 4.5: The SNN running on SpiNNaker

The network will not send any packets unless a neuron in the output population is spiking. Once the simulation starts, the SNN will run for 60 seconds with a time step of 1 millisecond.

4.5 The mechanism of controlling the servo motor

The maximum rotation range of the servo motor is 120^o, which is divided into 8 equal parts corresponding to the 8 neurons in the output population. The projections of the 8 parts on the horizontal axis have different lengths, because the trajectory of the robotic arm is a sector (Figure 4.6).

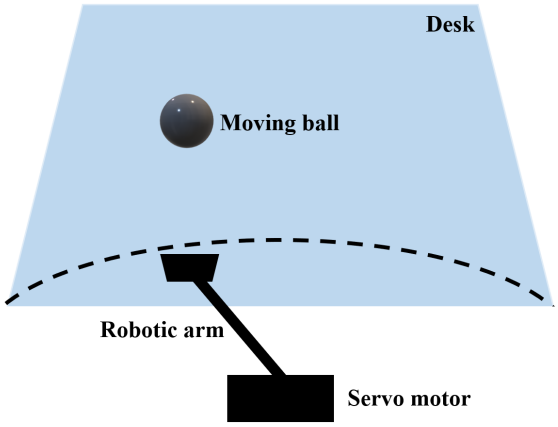

Figure 4.6: The trajectory of the robotic arm

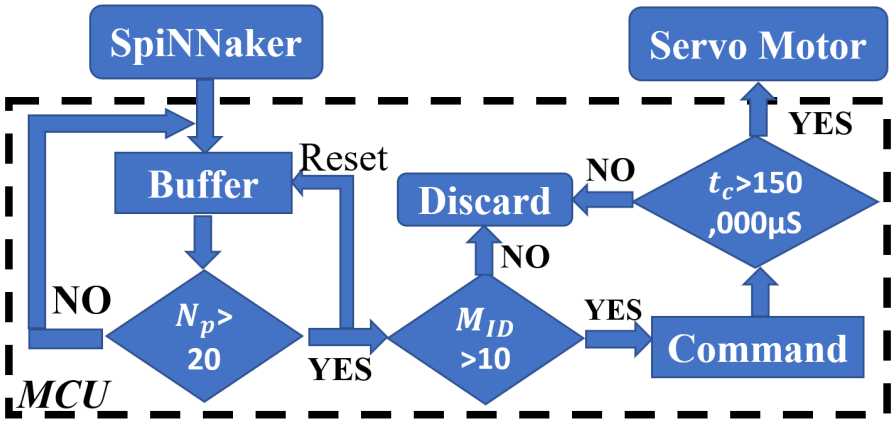

Figure 4.7: The mechanism of controlling the servo motor (the process of decoding packets is not included)

The MCU identifies spiking neurons by decoding packets received from SpiNNaker, and controls the position of the servo motor through these spikes. As described in Figure 4.7, the received packets are firstly decoded, and the obtained ID of the spiking neuron is stored in a buffer. Once the number of stored neuron ID is equal to 20, the MCU will

find the most frequent ID in the buffer and then reset the buffer. If the number of most frequent ID is greater than 10, a command is generated for moving the robotic arm to the position corresponding to the ID. Finally, the command will be executed if and only if the time difference from last executed command is greater than or equal to 150,000 microseconds.

This interception mechanism does not predict the trajectory of a moving ball, it simply moves the robotic arm to the position that has the most spikes. The SNN is expected to directly output the spike corresponding to the target position. This project does not aim to design and train a perfect SNN, and the SNN running on SpiNNaker is used for evaluating the hardware communication of the neuromorphic robotic system.

Chapter 5

Result

5.1 Receiving packets from SpiNNaker

The system is firstly tested by receiving pre-defined spikes from SpiNNaker. As mentioned in Chapter 4.2, SpiNNaker should be powered before the MCU due to the random logical levels change of SpiNNakerLink at power-on. The SNN running on SpiNNaker has two layers, an 8-neuron input layer and an 8-neuron output layer with one-to-one connection. As described in Table 5.1, 10 spikes are pre-defined for the input layer and the corresponding neurons will be forced to spike during simulation. The entire simulation process will last for one second.

Table 5.1: Spike source array of the input layer

Neuron ID	2	0	4	7	2	0	3	4	6	7
Spike time (ms)	0	50	100	150	240	270	279	333	680	855

During the whole process, the MCU has received 558 packets, of which 548 are 72-bit and the rest 10 packets are 40-bit. The 72-bit packets are sent when the SpiNNaker system is initializing, such as the processes “Running routing table compression on chip” and “Loading executables onto the machine”.

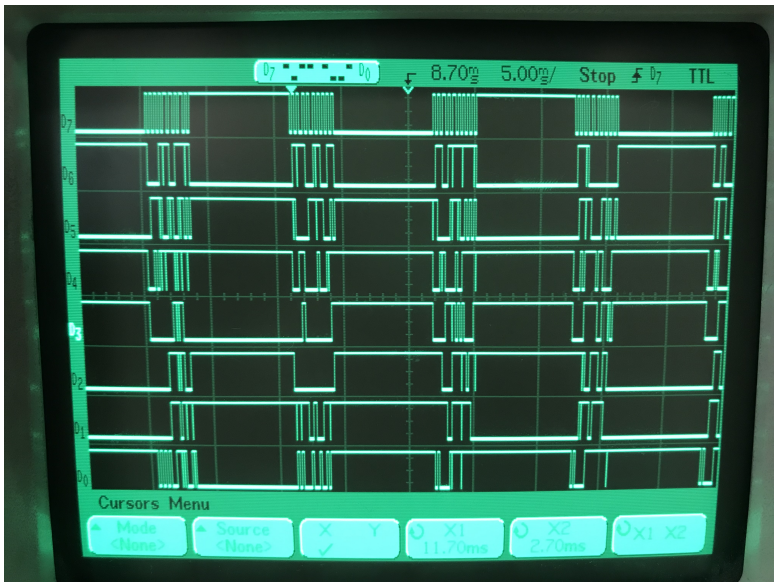

Figure 5.1: Oscilloscope reading of Lout[6] ~ Lout[0] (the $2^{nd} \sim 8^{th}$ lines in the Figure) and LoutACK (the first line)

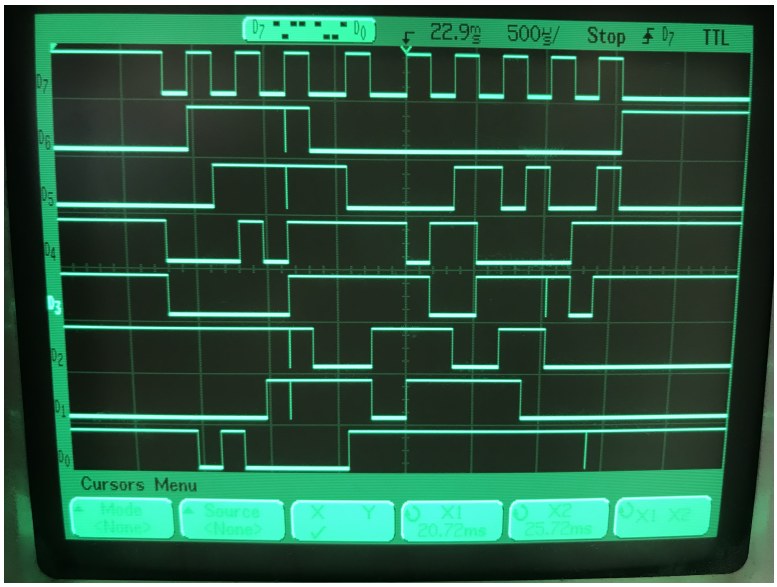

Figure 5.2: Oscilloscope reading of Lout[6]~Lout[0] (the $2^{nd} \sim 8^{th}$ lines in the Figure) and LoutACK (the first line) with a smaller time scale

An 8-channel oscilloscope is applied to monitor the logical levels of the 7 data wires and the acknowledgement wires of SpiNNakerLink. Due to the ephemeral 1 second simulation, the oscilloscope can not accurately locate the simulation process. Instead, the logical level change when SpiNNaker is initializing is recorded. As shown in Figure 5.1,

the period of ACK is not constant and the interval between each packet is longer than the transmission time of one packet. In a smaller time scale, the oscilloscope reading shows the frequency of ACK is also not constant during the process of transmitting one packet (Figure5.2). Since a 2-phase handshaking protocol is used, the period of transmitting each symbol is self-timed. Furthermore, the time taken by the MCU to decode each symbol is different. Thus, the period of ACK is not constant in transiting symbols.

The logical level of ACK has been changed 19 times during the transmission of a packet (Figure 5.2), which indicates the packet is 72-bit. Additionally, only 2 of the 7 data wire are changed when sending one symbol, which is in line with the 2-of-7 coding protocol. The 19 symbols sent in Figure 5.2 are:

3 8 0 4 1 3 A 4 D 1 3 6 3 6 5 6 3 7 EOP

The last symbol 'EOP' is used to separate packets, and it does not belong to the packet. After Hex to Binary conversion (starts from the LSB), the 72-bit packet is:

1100 0001	0000 0010 1000 1100 0101 0010 1011 1000	1100 0110 1100 0110 1010 0110 1100 1110
header	packet data	payload

The second bit of the header is the payload bit, and it is set in this packet. For the packets contain spike information, the payload is cleared because these are 40-bit packets that do not contain payload. Thus, the MCU utilizes the payload bit to distinguish 72-bit packets and 40-bit packets. When receiving 40-bit packets, the MCU will print the packets' index, spike time, symbols in the packets and decoded packets, if the MCU is connected to the Arduino IDE. The 40-bit packets printed by the MCU are:

546 Spike Time: 2
1 0 2 0 0 0 0 0 0 0
10000000–01000000000000000000000000000000

547 Spike Time: 50006
0 0 0 0 0 0 0 0 0 0
00000000–00000000000000000000000000000000

548 Spike Time: 100020
1 0 4 0 0 0 0 0 0 0
10000000–00100000000000000000000000000000

```
549      Spike Time: 150015
1   0   7   0   0   0   0   0   0   0
10000000–11100000000000000000000000000000
550      Spike Time: 240020
1   0   2   0   0   0   0   0   0   0
10000000–01000000000000000000000000000000

551      Spike Time: 270021
0   0   0   0   0   0   0   0   0   0
00000000–00000000000000000000000000000000

552      Spike Time: 279023
0   0   3   0   0   0   0   0   0   0

00000000–11000000000000000000000000000000
553      Spike Time: 333015
1   0   4   0   0   0   0   0   0   0
10000000–00100000000000000000000000000000

554      Spike Time: 680018
0   0   6   0   0   0   0   0   0   0
00000000–01100000000000000000000000000000

555      Spike Time: 855001
1   0   7   0   0   0   0   0   0   0
10000000–11100000000000000000000000000000
```

Since the first spike has happened at the beginning of the simulation, the MCU can identify the spike time for all received spikes with a precision of 1 microsecond. The MCU will set the time when the first 40-bit packet is received as the starting time of simulation and then set the time when the following 40-bit packets are received as their spike time. If the first spike has not happened at the beginning, the spike time recorded by the MCU will be its actual time minus the time of the first spike. Additionally, if two neurons are simultaneously fired, the MCU can also receive the two corresponding spikes.

5.2 Sending packets to SpiNNaker

After successfully receiving spikes from SpiNNaker, the system is then applied to send spikes to SpiNNaker. The initial states of SpiNNakerLink input pins are different from its output pins. All the output pins are logical low, while the logical level of input

ACK is high and the rest 7 data pins remain logical low (Figure 5.3).

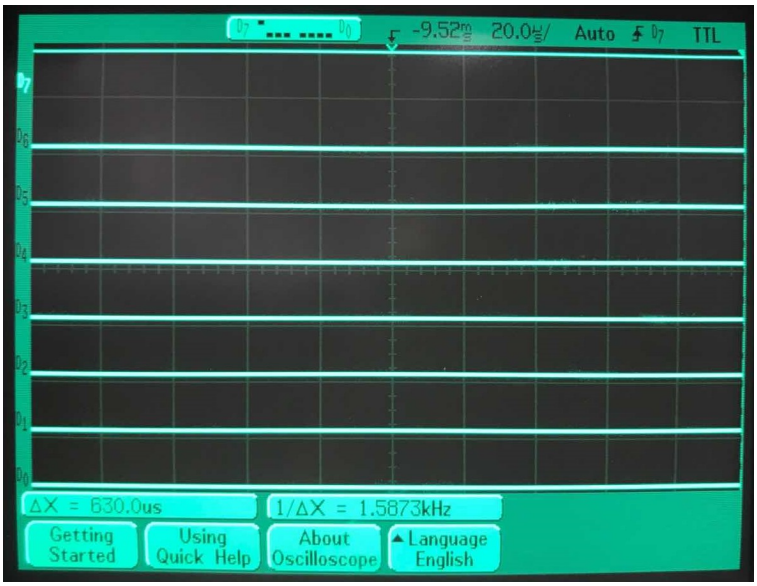

Figure 5.3: The initial states of Lin[6]~Lin[0] (the $2^{nd} \sim 8^{th}$ lines in the Figure) and LinACK (the first line)

To send symbols into SpiNNaker, the MCU firstly changes the logical levels of 2 pins and then waits for the logical level of LinACK changing from high to low. Once the logical level of LinACK is changed, the MCU will send the next symbol. After the interface is built, the MCU sends 5 spikes repeatedly to SpiNNaker at different speeds. The neuron IDs of the spikes are 6, 4, 3, 7 and 0, respectively. The transmission speed is firstly set to 50 packets per second and the simulation time is 1 second.

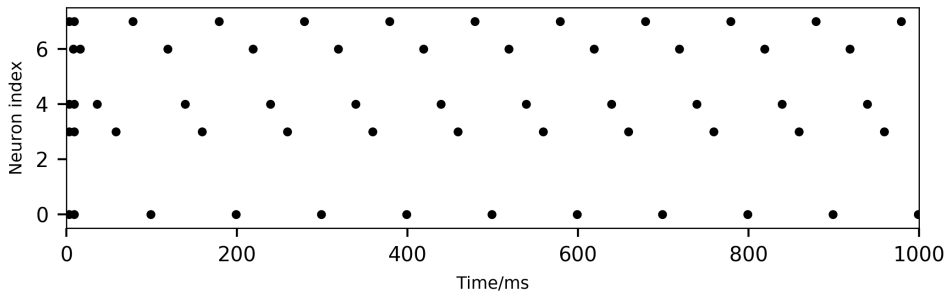

Figure 5.4: The spikes received by SpiNNaker at the speed of 50 $packets \cdot s^{-1}$

As shown in Figure 5.4, SpiNNaker can receive these spike in the correct order and at the correct time. Next, the speed limit is removed and the MCU is sending spikes

to SpiNNaker at the fastest speed it can reach. The result shows that the maximum transmission speed is higher than the speed which the SNN can afford (Figure 5.5).

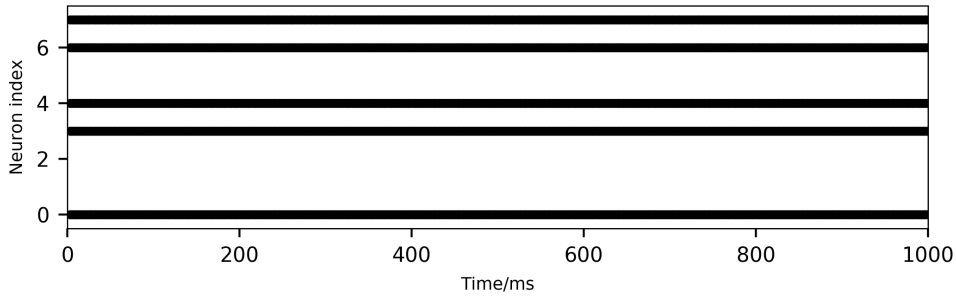

Figure 5.5: The spikes received by SpiNNaker at the maximum speed

Furthermore, the python program running on SpiNNaker issues a warning:

2019-08-27 19:19:20 WARNING: The weights from the synapses for Population 1:0:7 on 0, 0, 2 saturated 583 times. If this causes issues you can increase the spikes_per_second and / or ring_buffer_sigma values located within the .spynnaker.cfg file.

5.3 The closed-loop data transmission

The robotic system has successfully received spikes from SpiNNaker and sent spikes to SpiNNaker, the next step is to perform the whole closed-loop data transmission. The data flow direction is:

$$\text{DVS} \Rightarrow \text{PC} \Rightarrow \text{MCU} \Rightarrow \text{SpiNNaker} \Rightarrow \text{MCU} \Rightarrow \text{Servo Motor}$$

Since the embedded DVS does not work, a DVS has to be used. Events captured by DVS are sent to a PC through a serial port, and the PC then passes the event to the MCU through another serial port. The rest data transmission mechanism follows the description of Chapter 3, and the simulation time is 60 seconds.

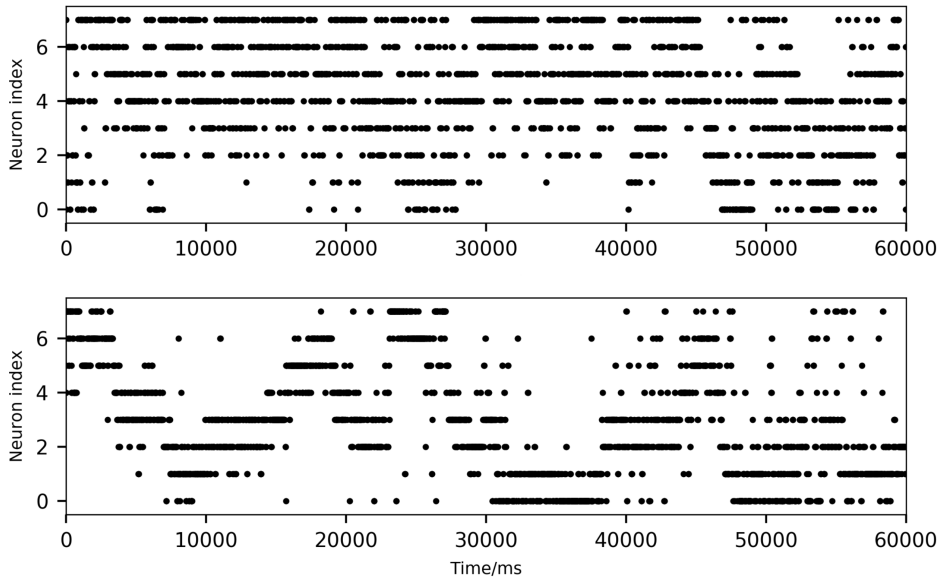

Figure 5.6: The spikes received by SpiNNaker in two simulations (The ball is moving to the left (Top) and moving to different directions (Bottom) in multiple trials)

As shown in Figure 5.6, SpiNNaker received events from DVS after the protocol conversion using the MCU. The horizontal position of the moving ball can be observed directly from the spikes sent to SpiNNaker, and neuron 0 to 7 are corresponding to the right end to the left end respectively. Furthermore, the MCU can control the servo motor to move the robotic arm to different positions based on the spikes sent from SpiNNaker. For a slow-moving ball, the robotic goalkeeper can intercept it with a success rate of ∼85%. However, the success rate is very low for fast-moving balls. Since the SNN simulated on SpiNNaker has not been trained to predict the trajectory of moving objects, the system can not move the robotic arm to the correct position before the fast-moving ball arrives.

To test the capacity of the system, the speed limit in the MCU program is removed and the MCU transfer data at the maximum speed that it can reach. During simulation, both the ACK signals of receiving packets from (downlink) and sending packets to (uplink) SpiNNaker are monitored by an oscilloscope. Figure 5.7 shows the two ACK signals in transmitting one packet. The logical levels of both signals are changed 11 times to transmit a packet, which indicates the packet is 40-bit (ten 4-bit symbols and an 'EOP' symbol that used to separate packets).

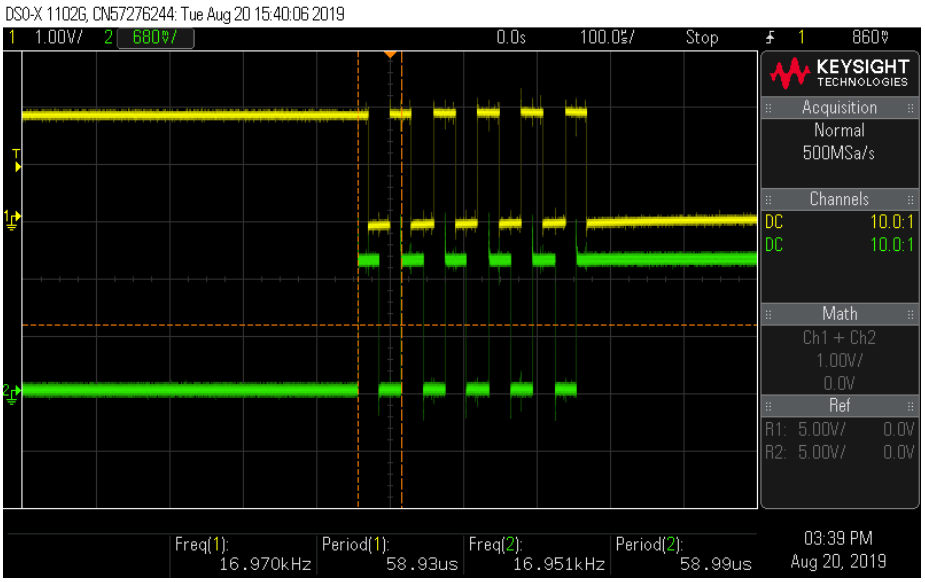

Figure 5.7: The maximum speed of downlink (yellow) and uplink (green)

Through the measurement using the oscilloscope, the frequencies of downlink ACK and uplink ACK are 16.970kHz and 16.951kHz, respectively. Since the logical levels of both ACK signal change twice in each period, 2 symbols are sent in one period. Thus, the downlink speed is:

$$\begin{aligned} V_{Downlink} &= 16.970kHz \times 2 = 33.940k\ symbols \cdot s^{-1} \\ &\simeq 3.085k\ packets \cdot s^{-1} \\ &= 16.97k\ bytes \cdot s^{-1} \end{aligned}$$

The uplink speed is:

$$\begin{aligned} V_{Uplink} &= 16.951kHz \times 2 = 33.902k\ symbols \cdot s^{-1} \\ &\simeq 3.082k\ packets \cdot s^{-1} \\ &= 16.95k\ bytes \cdot s^{-1} \end{aligned}$$

5.3.1 Latency of the system

The response latency measures the time between the DVS captures the movement of the ball and the servo motor moves to the target position. Both the time steps used

in jAER and DVS are 1 μs. Thus, DVS requires at least 1μs to generate an event and at least another 1μs to send the event to jAER. The time cost for the MCU to receive one event from jAER through serial port is measured by the built-in timer of the MCU. The average time taken to receive an event is 32 μs.

The transmission speeds between the MCU and SpiNNaker are already given in the above section. The time cost to send a packet to SpiNNaker and then receive the corresponding packet is:

$$t_{spike} = \frac{1}{3085}s + \frac{1}{3082}s \simeq 649\mu s$$

which is longer than the time cost between DVS captures an event and the MCU receives the event. Furthermore, the MCU checks the state of all processes every 1 μs. Thus, receiving events, sending packets, receiving packets, generating commands and executing commands are processed in parallel. During the MCU receiving the first packet from SpiNNaker, it is also sending the second converted event to SpiNNaker (Figure 5.8). Furthermore, the servo command is generated based on the latest 20 received packets, and the uplink speed is nearly the same as the downlink speed.

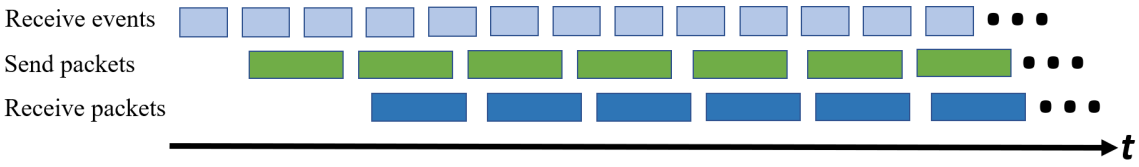

Figure 5.8: Parallel execution of the system

Therefore, the resulting time spent on the processes before generates servo control commands is

$$\begin{aligned} t_{command} &= 1\mu s + 1\mu s + 32\mu s + 20 \times \frac{1}{3082}s + \frac{1}{3085}s \\ &\simeq 6847\mu s \end{aligned}$$

Since the motor speed is 120^o per 150 ms and the motor rotation range is 120^o, the generated command will not be executed until the time difference from the last executed

command is greater than or equal to 150 ms. Assume the generated command is executed immediately and the motor is required to rotate 120^o to reach the target position, the response latency of the system is:

$$t_{latency} = t_{command} + 150ms = 156.847ms$$

During rotation of the motor, commands are still being generated continuously but will be overwritten until the end of the rotation. Therefore, motor rotation is also a parallel process of the system. Due to these parallel processes, the response latency is not a constant, and it is dominated by the time spent on motor rotation and the time to wait for the command to be executed.

Chapter 6

Evaluation

6.1 Accuracy

6.1.1 Resolution

The robotic system subsamples events from the original resolution 128×128 to 8×8, and only utilizes the horizontal axis of the events to intercept moving balls. The resolution used in the system can provide at most 8 different positions for the motor to move, which is insufficient for a long horizontal length. Additionally, the system does not have the prediction ability due to the unused vertical axis information. Furthermore, the communication between the MCU and SpiNNaker can operate under the resolutions of 32×32 and 64×64.

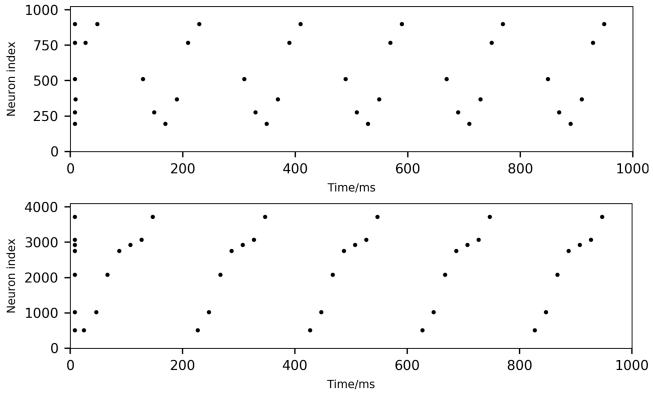

Figure 6.1: Spikes received by SpiNNaker with the input neuron population size of 1024 (Top) and 4096 (Bottom)

The size of the neuron populations in the SNN running on SpiNNaker is increased from 8 to 1024 or 4096, and the MCU is able to inject spikes into the increased input population and receive spikes from the output population (Figure 6.1). The original resolution of 128×128 requires 16384 neurons to simulate, SpiNNaker-3 board can not support the simulation and an error is issued by the Python program:

PacmanValueError: Unable to allocate requested resources to vertex 'Population 1 with 16384 atoms': No resources available to allocate the given resources within the given constraints

Although the MCU can communicate SpiNNaker under higher resolutions, the essence of the system is unchanged. It just counts the spikes from SNN to make a decision. The SNN applied in the system does not have the learning process and can not give the command directly. In summary, the hardware system can transmit data under 32×32 and 64×64 resolutions, but the SNN does not have the ability to process the data.

6.1.2 Horizontal projection

The trajectory of the robotic arm is a sector, the resulting projections of each position in x horizontal axis is different. As shown in Figure 6.2, positions that closer to the middle have larger projections. However, all the 8 positions have the same horizontal projection in the events captured by DVS and the spikes processed by SpiNNaker. This cognitive difference leads to errors in motor rotation, which reduces the accuracy of interception. For improvement, the positions should be realigned to have the same horizontal projection.

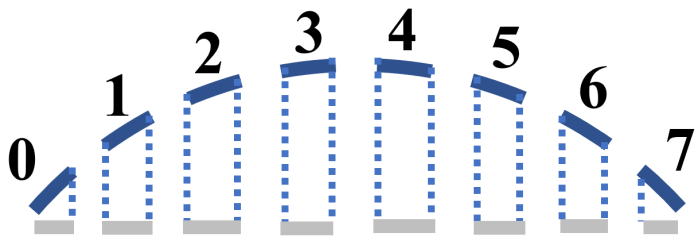

Figure 6.2: Horizontal projections of the 8 positions

6.1.3 Payload

The transmitted packets between the MCU and SpiNNaker are 40-bit, which do not include a 32-bit payload to carry more information about spikes. In the solution raised by TUM and University of Manchester, the payload is used to store the membrane potential of the spiking neuron [14]. The python program developed in this project does not define the payload in the classes for declaring input and output neuron populations. The system can transmit 72-bit packets, if the 32-bit payload is defined. However, the transmission speed will be decreased with larger packets. There is a trade-off between the 32-bit payload and the transmission speed.

6.2 Efficiency

The designed neuromorphic robotic system has high efficiency in terms of events generated by DVS and spikes processed by SpiNNaker. DVS only generate events for area where has light intensity change, and the rest area will not generate any data. Thus, the events received by the MCU are inherently sparse. Furthermore, the neuron in SNNs are only fired when their membrane potential is higher than a pre-defined threshold. In typical multi-layer perceptron networks, all neurons are fired at every propagation cycle [22]. Thus, SNNs are also inherently sparse and require less computational power compare to CNNs. Due to the sparse signals of DVS and SNNs, the robotic system has high efficiency.

6.2.1 Improve motor control mechanism

The system executes motor control commands every 150 ms, which dominates the response latency of the system. This mechanism could be improved to increase the efficiency of the system. For example, it takes only 30 ms for the motor to rotate to the target position, if the motor is in position 3 and the command requires the motor to move to position 4. After the rotation, the motor will stop moving in the next 120 ms. It should be noted that the average angular velocity is increasing with the increase of the rotated angle. For instance, it costs the motor 30ms to rotate 15^o and 50ms to rotate 30^o.

Therefore, the rotation time for different angles is measured rather than calculated. An improved mechanism can execute commands with different waiting times. The program will record the current position of the motor and calculate the angle required to rotate for the motor based on the executed command. Then, the waiting time for the next command execution is generated based on the rotated angle. That is the program will execute a command every 30 to 150 ms. Thus, the response latency is reduced and the efficiency of the system is increased.

6.2.2 Independence

There are two processes in the system that need to use a PC: transmitting events from DVS to the MCU and loading python program into SpiNNaker. For the latter, the PC can be detached once the SpiNNaker is set running. However, a PC is necessary for transmitting events, because an eDVS is not used. Thus, the robotic system can not run independently. The dependence on PC reduces the efficiency of the system and prevents it to be a mobile robot. To improve this, an eDVS or a board with USB ports could be used. For example, the Raspberry Pi Model B boards can replace the currently used MCU, due to its 4 USB ports and 40 GPIO pins.

6.2.3 adaptive transmission speed

The data transmission speed is not fixed, the system only has a limitation for the maximum speed. Each data transmission process has a waiting time before the next data transmission, and the interval between two adjacent transmissions can be longer than the waiting time. Thus, the transmission speed can be any value lower than the maximum speed. If DVS captures fewer events, the system will transmit data at a slower speed to reduce power consumption.

6.3 Further development

The designed neuromorphic robotic system can be improved from independence, accuracy and efficiency. Firstly, the system should work independently by using an eDVS or another board with a USB port. Next, a well designed and trained SNN should be performed to directly output the motor control commands. To increase the accuracy further, the resolution of events should be increased to 32×32 or 64×64. Current transmission speeds are $\sim$3000 $packets \cdot s^{-1}$ for both uplink and downlink. which are insufficient to support higher resolutions. A MCU with faster clock speed should be used instead of current MCU to increase transmission speed. Furthermore, 72-bit packets could be applied instead of 40-bit packets to carry more information, such as membrane potential. Additionally, humans have two eyes, but this project uses only one DVS. In the future, two DVSs could be used to compose a stereo vision system to capture 3D information.

Chapter 7

Conclusion

This project has successfully designed, implemented and evaluated a neuromorphic robotic system that consists of DVS, SpiNNaker and a servo motor by using a single MCU to establish the communication between them. The system is implemented to predict the trajectory of a moving ball and then intercept it which works as the same as a goalkeeper. The data flow direction in this application is:

$$\text{DVS} \Rightarrow \text{PC} \Rightarrow \text{MCU} \Rightarrow \text{SpiNNaker} \Rightarrow \text{MCU} \Rightarrow \text{Servo Motor}$$

The events captured by DVS should be directly sent to the MCU. However, the communication with the eDVS can not be established, the reason might be its parameters have been modified by previous students. Thus, a DVS128 is used and a PC is applied to pass the events from DVS128 to the MCU. Then, the project explored the communication between the MCU and SpiNNaker through SpiNNakerLink. The system used 16 GPIO pins to communicate SpiNNaker: 8 for the input bus and 8 for the output bus. The data transmission protocol used is 2-phase handshaking, and the symbols are transmitted in the form of 2-to-7 coding. The MCU transmitted symbols to SpiNNaker in serial and used 40-bit packets to transfer spikes. After receiving the spikes, the SNN run on SpiNNaker passed the spikes from input neuron population to output neuron population with one-to-one connection. Next, the MCU generated the servo command based on the latest received 20 spikes and performed PWM to precisely control the motor to move to the target position. At this point, a closed-loop execution is completed.

In the goalkeeper application, the system achieved an accuracy rate of ∼85% for slow-moving balls. The data is transmitted at an adaptive speed with the limit of the maximum speed of ∼3000 $packets \cdot s^{-1}$ for both uplink and downlink. Assume the data transmission is at the maximum speed, the time cost from capturing events to generating the corresponding command is 6.847 ms. For the servo motor, it spends 150ms to rotate the whole range, which is dominated in the response latency.

For further development, the robotic system should firstly be able to operate without a PC, and then improve its accuracy and efficiency. An improved SNN could be applied to precisely predict the trajectory of moving balls. A MCU with a higher clock speed could be used to increase the transmission speed, which enables the system can operate under the resolution of 64×64, and provide more information to the SNN using 72-bit packets. Finally, a PCB could be used instead of the breadboard to reduce the size of the robot and make the system more stable.

Bibliography

[1] D. Crevier, *AI: The Tumultuous Search for Artificial Intelligence*. ISBN-10: 0465029973, 1993.

[2] P. Chaudhary, J. Aronco, and J. Wegner, "Flood-water level estimation from social media images," *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, pp. 5–12, 2019.

[3] W. Maass, "Networks of spiking neurons: The third generation of neural network models," *Neural Networks*, vol. 10, no. 9, pp. 1659 – 1671, 1997.

[4] F. Ponulak and A. Kasiski, "Supervised learning in spiking neural networks with resume: Sequence learning, classification, and spike shifting," *Neural Computation*, vol. 22, no. 2, pp. 467–510, 2010.

[5] D Wei and J. G. Harris, "Signal reconstruction from spiking neuron models," in *2004 IEEE International Symposium on Circuits and Systems*, vol. 5, 2004.

[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] ARM, *ARM Cortex-M3 Processor Technical Reference Manual Revision r2p1 Documentation*. Arm Limited, 2015.

[8] S. M. Bohte, J. N. Kok, and H. L. Poutr, "Error-backpropagation in temporally encoded networks of spiking neurons," *Neurocomputing*, vol. 48, no. 1, pp. 17 – 37, 2002.

[9] D. E. Rumelhart and G. E. Hinton, "Learning internal representations by error propagation," *Parallel Distributed Processing*, 1985.

[10] J. Pfister, D. Barber, and W. Gerstner, "Optimal hebbian learning: A probabilistic point of view," *Artificial Neural Networks and Neural Information Processing*, pp. 92–98, 2003.

[11] D. O. Hebb, *The organization of behavior: a neuropsychological theory.* Science Editions, 1962.

[12] J. J. Wade, L. J. McDaid, and J. A. Santos, "Swat: A spiking neural network training algorithm for classification problems," *IEEE Transactions on Neural Networks*, vol. 21, no. 11, pp. 1817–1830, Nov 2010.

[13] APT group The University of Manchester, "Interfacing real-time spiking i/o with the spinnaker neuromimetic architecture," *Australian Journal of Intelligent Information Processing Systems*, vol. 11, no. 1, pp. 7–11, 2010.

[14] 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.

[15] M. Oster, R. Douglas, and S. Liu, "Computation with spikes in a winner-take-all network," *Neural Computation*, 2009.

[16] T. Stewart, A. Kleinhans, A. Mundy, and J. Conradt, "Serendipitous offline learning in a neuromorphic robot," *Frontiers in Neurorobotics*, vol. 10, 2016.

[17] 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.

[18] 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.

[19] 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.

[20] M. Hagiwara and H. Akagi, "Pwm control and experiment of modular multilevel converters," *IEEE Power Electronics Specialists Conference*, pp. 154–161, June 2008.

[21] ATP Group, "External devices on spinnaker," *The University of Manchester*, 2016.

[22] A. J. Shepherd, *Second-order methods for neural networks: Fast and reliable training methods for multi-layer perceptrons.* Springer Science and Business, 2012.