HW3 - Joystick Controller

1 Introduction

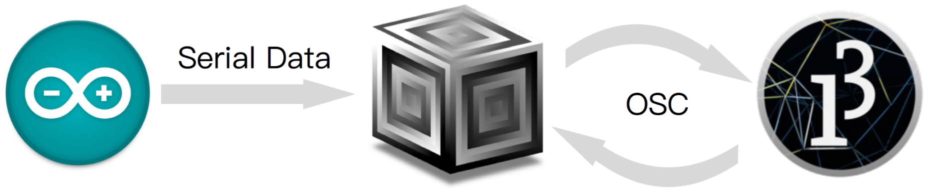
This project aims to design and implement an easy-to-operate multi-effect controller. By manipulating the joystick to different positions, the user can generate different sound effects on instruments, audio or white noise in real-time. This project has also designed and implemented a graphical interface that reflects the sound effects visually. The idea of this project could be applied to the following scenarios：

- Simplifying the equipment for live performances：With a joystick to control all the effects, the performers save a lot of effects monoblocks and connection cables.
- Integration with audio playback devices：Making it easier for users to adjust the audio playback they want.

2 Implementation

2.1 Overall Structure

In this project, the Arduino sent the values(x-axis, y-axis and z) read from the analog and digital pins to the SuperCollider by serial port. In Supercollider, different equations for using the x and y axis were set in the algorithm of three different effects. And then retrieve the three values in the array sent from a serial port and set them as arguments in the three effects. Finally, the 3 values from SuperCollider and the values of buttons on GUI are transferred mutually by the OSC protocols.



2.1.1 Serial communication (Arduino-SuperCollider)

Arduino:

- Serial.begin(9600) initializes serial communication with a baud rate of 9600.

- Use ‘Serial.print’ to read the x, y and z values through the serial communication.

SuperCollider:

- Use ‘p = SerialPort.new("COM3",9600)’ to open a port so that SuperCollider can read the message sent from Arduino via serial port.
- The message received from the serial port is a string. E.g. “516,518,1”.
- Use ‘arr=str.split(Char.comma).asInteger’ to convert the string into an integer array.
- Use ‘arr.at()’ to retrieve the first, second and third values to the x axis, y axis and z.

2.2.2 OSC Protocol (SuperCollider – Proccessing)

SuperCollider:

- var recAddr=NetAddr("127.0.0.1",57120);   //receiver
- var sendAddr=NetAddr("127.0.0.1",12000); //sender

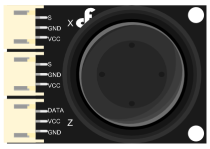
Processing:

- sender = new NetAddress("127.0.0.1",57120); //send to SC recAddress
- oscP5 = new OscP5(this,12000); //rec

In these two parts, loopback IP "127.0.0.1" is set to send messages.

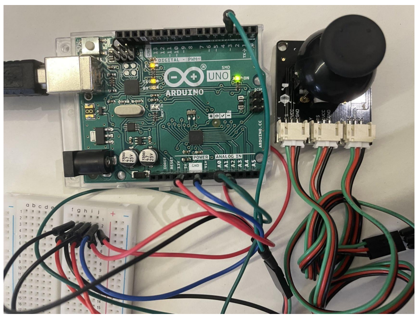
Port 12000 is for transferring the data from SuperCollider to Processing, and port 57120 is for transferring backwards.2.2 Arduino and Joystick

2.2.1 Joystick



A standard joystick module that allows the user to alternate 3-dimension values.In the Joystick, S pin of X and Y axes are the output of data for two axes separately,and connect to analog input A0 and A1.DATA pin of Z axis is the output of data, and connect to digital input 3.

2.2.2 Arduino



- 1）Serial.begin(9600) initializes serial communication with a baud rate of 9600.
- 2) ‘analogRead’ to read the analog pin connected to the x and y values. The range of x and y is from 0 - 1023.
- 3) ‘digitalRead’ to read the digital pin connected to the z value, which is ‘0’ or ‘1’.
- 3) ‘Serial.print’ to print the x, y and z values read through the serial communication.

2.3 Supercollider

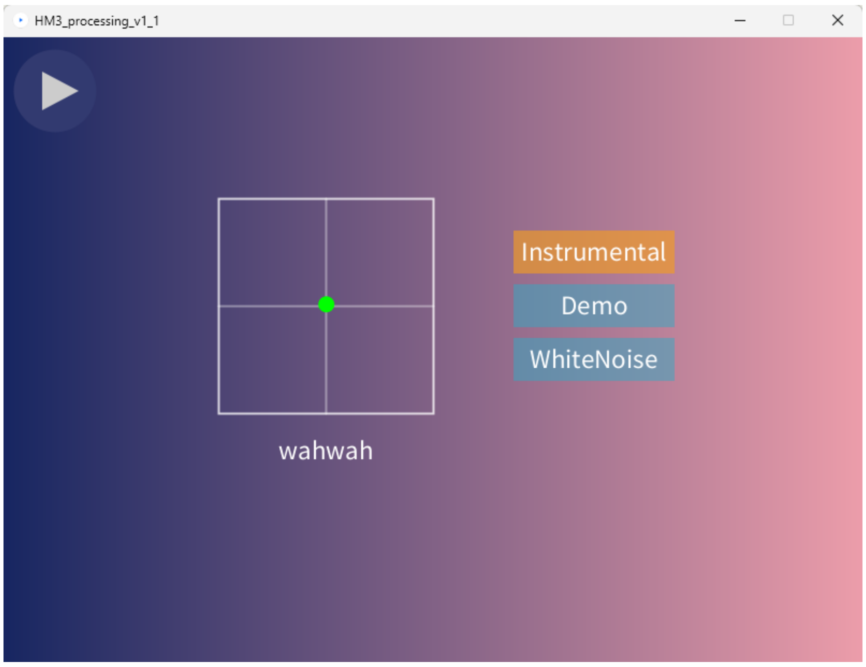
The basic logic of this part is to read the input data through the serial port and control the parameters of the synthesiser according to the value of the data to achieve real-time audio processing and control.

- Setting the initial state of synthesisers: By creating and instantiating Synth, the initial state of synthesisers such as Wah, Phaser, and Flanger can be set.
- Creating and running the loop to handle serial port input: creating a loop, the p.read function is used to read input data from the serial port, and it is stored as a string.
- Updating synthesiser parameters: Based on the data sent from the serial port, the synthesiser parameters are updated. For example, modifying the frequency band (\freqBand) and bandwidth (\bw) of the Wah synthesiser, modifying the frequency (\kfreq) and multiplier (\kmul) of the Phaser synthesiser, and modifying the rate (\rate) and mix level (\mix) of the Flanger synthesiser.
- Switching synthesiser operating modes based on button states: By comparing the current button state with the previous button state, the synthesiser's operating mode is switched. Different synthesiser operating modes are selected based on the change in button state.
- Sending OSC messages: Based on the updated parameter values, OSC messages are sent to processing using sendAddr.

2.4 Processing

The processing part of the application is responsible for showing the user's interactions with a joystick and control buttons for changing modes and sound playback. We used the OSC (Open Sound Control) library to interact with SC.

The application has three modes: Instrumental, Demo (Guitar solo), and White noise. Users can switch between these modes by interacting with graphical elements on the screen. The joystick controls some filters: wah-wah, flanger, and phaser, and users can change the active filter by clicking the joystick.



The application's functionality is divided into several key components:

Setup: The setup() function initialises the application by setting up the canvas size and other parameters. It establishes communication with the OSC server and defines variables for colours and visual elements.

OSC Event: The oscEvent() function is responsible for processing incoming OSC messages. It listens for messages with the address pattern "/data" and typetag "iii," indicating the reception of x, y, and z values from the joystick. Upon receiving these values, the function updates the internal variables accordingly.

User Interface: The draw() function handles the visual rendering of the user interface. It utilises the received joystick data to update the positions of graphical elements, such as the joystick visualisation, rectangles representing different modes, and a play button. The function also sends OSC messages with the current mode and play button state to the OSC server.

Interaction: The mousePressed() function detects mouse clicks and handles the corresponding actions. When the user clicks on a mode rectangle or the play button, the function updates the mode state or play button state, respectively.

3 Conclusion and further improvements

This project fulfills the function for which it was originally designed. However, to further enhance the sophistication and efficiency of this project, several aspects could be improved:

- Incorporate more effects, thereby expanding the selection of available effectors for users.
- Implement a feature that enables users to custom-define their desired effects.
- Explore sound synthesis capabilities to enhance the potential for a broader range of sound variations.