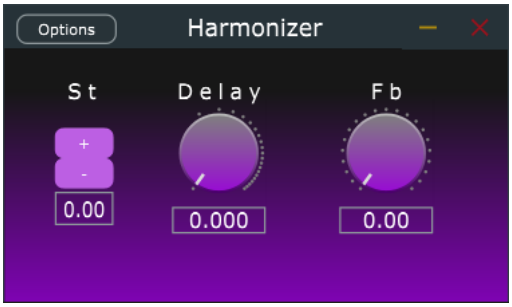
CMLS HW3 - Design and implementation of a Computer Music System

Xenharmonics Group:
Baroli Gabriele
Ferrando Alessandro
Mauri Noemi
Passoni Riccardo



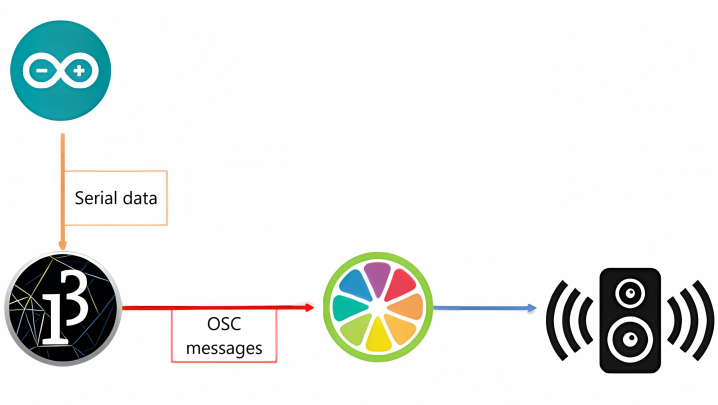
Kinetic Harmonizer

Abstract

Kinetic Harmonizer is an effect plugin developed with the **JUCE** framework and partially controlled by a **triple-axis accelerometer**.
It implements an *harmonizer* through a variable delay line, the cent shift in pitch is controlled by the accelerometer's movements in real time, employing interaction design techniques in its development.
The interactive section of the GUI has been implemented in **Processing**.

0.1 Introduction

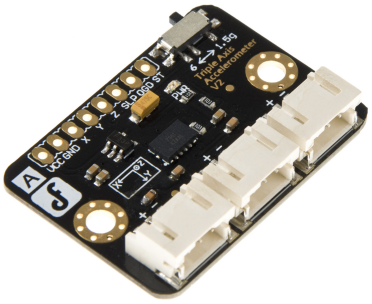
The plugin that's been developed is an *audio effect*, more specifically it's an harmonizer, implemented through a *delay line*. It allows for the control of each parameter that influences the pitch and placement in time of the delayed signal in relation to the input audio.



Interaction is an essential part of this plugin, the main parameters of the effect (something, something else and also this other thing) are controlled by an accelerometer. The sensors share the information through **Serial** communication, this data is then sent to the plugin by the **Arduino** microcontroller, through the use of the **OSC** protocol to send the data in a way that can be processed by the JUCE framework.
We'll now explain how the data is collected in this next section of the report.

1 Interaction System Unit - Board and Accelerometer

1.1 MMA7361 Triple-Axis Accelerometer



The **MMA7361** is a *triple-axis accelerometer*, meaning it gathers acceleration data for each one of the three dimensions of space: x, y and z. It's manifactured by Freescale, a company specialized in microcontrollers and analog sensors. Indeed, the MMA7361 outputs a 3.3V analog voltage for each of the three outputs, corresponding to the three available coordinates. This output voltage is proportional to the acceleration measured by the sensors and to the power supply voltage. Its has sensitivity can be selected using a dip switch, allowing to choose from either 1.5g or 6g. The sensitivity of an accelerometer refers to the greatest amount of acceleration the sensor can measure and accurately represent as an output.

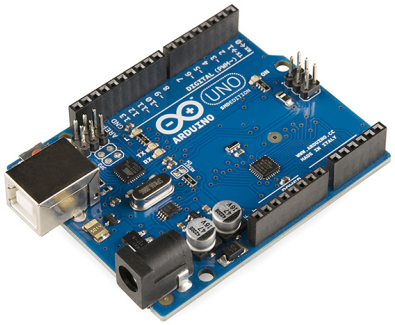
This will be the component that is used to control the plugin's parameters in real time. Which parameters in particular are controlled by the component's outputs will be explained in detail in the second section of this report.

Specification	Value
Voltage	3.3V or 8V
Sensitivity	$\pm$1.5g or $\pm$6g
Power Consumption	500μA
Output	Analog
Size	37x26mm

Table 1: MMA7361 spec table.

In order to work with the data gathered by its sensors, it's necessary to convert their analog signals to a usable digital one. The **Arduino UNO** is a microcontroller that can be used to achieve this necessary conversion.
The sensor's information in transmitted to the microcontroller as raw data, through **Serial** communication, Arduino will then format the data in a way that JUCE is able handle.

1.2 Arduino UNO

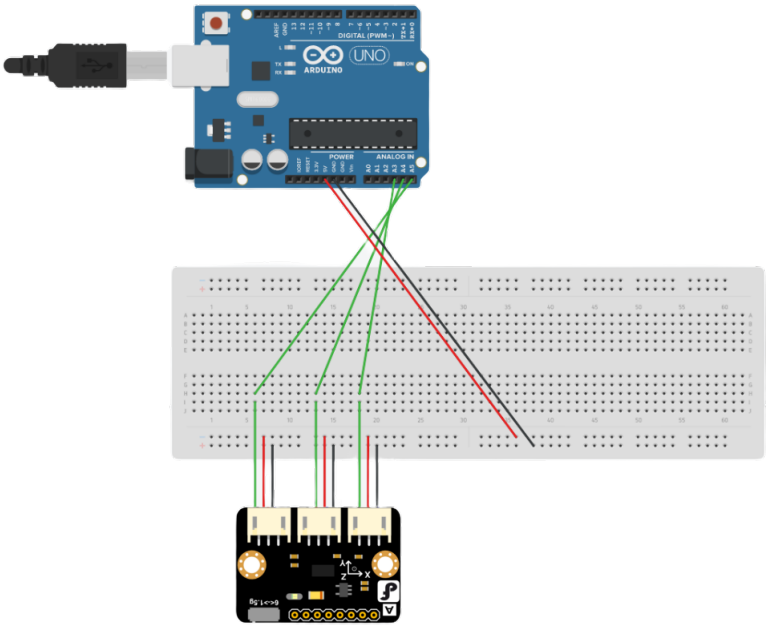


The **Arduino UNO** is a microcontroller board based on the *ATmega328P* chip. It contains provides a number of input sockets: 6 analog inputs, 14 digital input/output pins (6 of which may be used as PWM outputs) and a USB port. It can be connected with a variety of different sensors, such as the accelerometer that's being used in this particular project, and other electrical components that can be programmed using the *Arduino programming language*.

In order to use the data from the accelerometer, the sensor must be connected to the microcontroller, which will then transmit the signals to the plugin's host.
This is done through the **OSC protocol** (Open Sound Control), a content format intended for sharing music performance data.

1.3 Breadboard setup and connections

The MMA7361 used to collect the data, as it can be seen in the following illustration, is connected to the microcontroller using a *breadboard*.
Three connections are required for each of the three sensors, for a total of **nine**: one for *Serial communication* to the Arduino, one to provide *power* to the accelerometer and another for *ground* connection.



The data is transmitted to the Arduino through the breadboard *analogically*, the microcontroller converts this signal into a series of *10-bit values*, meaning it will map the received voltages between 0 and the maximum working voltage to a value between 0 and 1024.
This is accomplished through the use of the function ***analogRead()***.

2 Graphical Feedback Unit - GUI and Processing

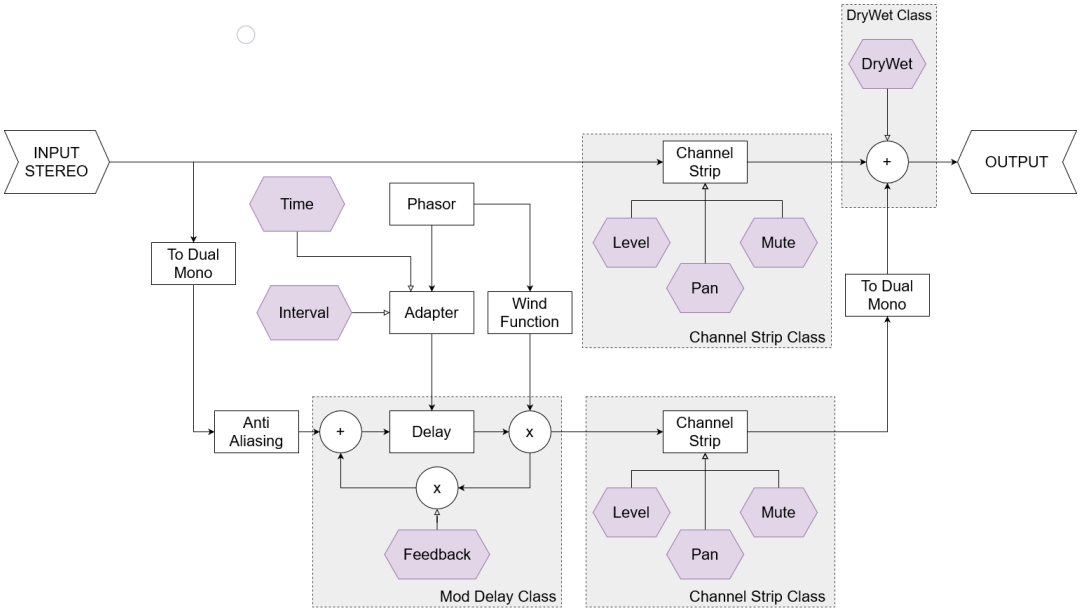
LaTeX is great at typesetting mathematics. Let $X_1, X_2, \ldots, X_n$ be a sequence of independent and identically distributed random variables with $\mathrm{E}[X_i] = \mu$ and $\mathrm{Var}[X_i] = \sigma^2 < \infty$, and let

$$S_n = \frac{X_1 + X_2 + \cdots + X_n}{n} = \frac{1}{n}\sum_{i}^{n} X_i$$

denote their mean. Then as n approaches infinity, the random variables $\sqrt{n}(S_n - \mu)$ converge in distribution to a normal $\mathcal{N}(0, \sigma^2)$.

3 Computer Music Unit - Kinetic Harmonizer

3.1 JUCE implementation



This block diagram shows the signal flow of the plug-in implemented in JUCE. First of all the signal is processed by the dual-mono block which function is to copy the signal in the dual-mono buffer and then process it. The dual mono buffer has two channel so if the signal is mono is copied in both channel otherwise, if the signal is stereo, the sum of the two original channels is memorized in each channel of the dual mono buffer.

The dual mono processing ensures that each channel is being processed with the same precision, making the result more accurate and it is performed in the **PluginProcessor.cpp** file.
In order to avoid the aliasing phenomena the signal is filtered in order to satisfy the *Nyquist Theorem*, which is processed in the **Filter.h** file. The following block implements the pitch shifting by the means of delay lines.
The delay time will determine the amount of pitch shifting that will occur. Increasing the delay time will lower the pitch of the delayed signal, while decreasing the delay time will raise the pitch of the delayed signal.

The delay time is determined by the adaptor block that take into consideration the interval which represents the number of semitones and cents of which the user want to increase or decrease the original signal and a phasor. The phasor consists of a saw up or saw down LFO depending if we want to increase or decrease the pitch (**Merger.h**, **Oscillator.h**).
The delay function, performed in the **Delay.h** file, iterates over the output buffer, processing one sample at a time for each channel. For each sample, it retrieves

the current feedback value and computes the delay time for the current modulation channel. It then computes the read and write indices for the delay memory buffer based on the current write index, delay time, and sample rate.
The function then computes the fractional part of the read index using the *"integerPart"* variable, which is dependent on the *alpha* value.

Finally, the function computes the output sample v4alue by performing linear interpolation between two adjacent samples in the delay memory buffer using the *alpha* value. It then writes the output sample to the output buffer and updates the delay memory buffer with the feedback value.
The *oldSample* array is used to store the previous output sample value for each channel, which is needed for the interpolation computation.
The signal is the processed by the Channel Strip block which is able to set the level, the pan and mute on/off. For the purpose of the homework, these parameters cannot be changed.

The signal is finally mixed with the original one through a dry wet. Same as before, for the purpose of the homework we have set it to totally wet.

3.2 Connection with Arduino