Computer Music Languages and Systems

Homework n° 2
Flanger effect plugin with feedback

10574752

10751438

10612929

10486570

Master of Science in Music and Acoustic Engineering

POLITECNICO
MILANO 1863

Abstract

Our group worked on **Assignment 3**: implement a flanger effect plugin with feedback. We divided our work in steps as presented in the sections of this document. We start from the block diagram of Figure 1.1, first implementing the Delay, then adding the LFO and the feedback. We design the plug-in in JUCE, implementing the audio processor and the Graphical User Interface with Projoucer. During the implementation, based on our audio results, we customly set the parameters range that best represent a Flanger effect. As a reference, we used the "Material on delay lines" PDF and the related JUCE class-laboratories. Here is the link to the GitHub Repository containing all the code: `https://github.com/EllDy96/CarlGang/tree/Homework2`

Chapter 1

1.1 Theory recap: flanger with feedback

The Flanger is a Delay based plug-in. Its typical sound is produced by changing the delay length over time, creating a motion of regularly spaced notches in the frequency response. We modulated the delay-time with a low-frequency oscillator (LFO) using three waveforms: sine, triangle and sawtooth. To obtain this specific sound effect one important constraint is to set a short delay length (1-10 ms), that we fixed at 5ms for sound design purpose. Adding a **feedback control**, the output of the delay line is routed back to its input. It will result in many successive copies of the input signal spaced several milliseconds apart and gradually decaying over time producing an harmonic enrichment and a colouration of the sound effect! This feature can be controlled by the user with the feedback gain knob. However, since the delay times in the Flanger are below the threshold of echo perception (roughly 50–70 ms), these copies are not heard as independent sounds as with the feedback of a pure delay effect.
In the picture below a block diagram of a flanger with feedback unit is shown:

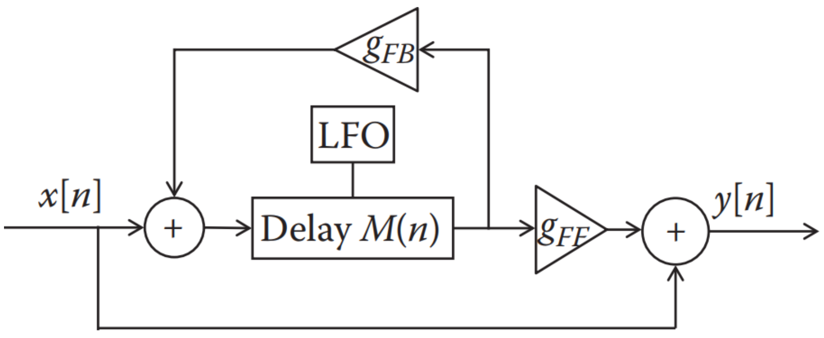


Figure 1.1: Flanger with feedback

As we can see from the block diagram, we have two gains, g_{FB}, which is the feedback gain, and g_{FF}, the gain related to the output of the delay line. For ensure the filter's stability we set the value of the feedback gain less then one. If $g_{FB} < 1$, the delayed copies of the sound will gradually decay, where a gain of 1 or more means they will grow indefinitely and the filter become unstable. So, the input/output relation for the flanger with feedback can be expressed in the time domain as:

$$y[n] = x[n] + g_{FF}d[n]$$

where $d[n] = x[n - M[n]] + g_{FB}d[n - M[n]]$ is the output of the delay line. $M[n]$ expresses the delay length over a sample n. There are different M functions, depending on the particular waveform we are using. Generally, we can describe M through the following parameters:

· M_0 (in samples), the delay given by the delay control;

· M_W (in samples), the delay given by the sweep width control;

· ϕ, the phase of the waveform (in rad).

For our plugin we are going to use the following waveforms, with $\phi \in [0, 1]$:

· Sine:

$$M(\phi) = M_0 + M_W(\frac{1}{2} + \frac{1}{2}\sin(2\pi\phi))$$

· Triangle:

$$M(\phi) = \begin{cases} M_0 + M_W(\frac{1}{2} + 2\phi) & \text{if } 0 \leq \phi < \frac{1}{2} \\ M_0 + M_W(1 - 2(\phi - 1/4)) & \text{if } \frac{1}{2} \leq \phi < \frac{3}{4} \\ M_0 + 2M_W(\phi - \frac{3}{4}) & \text{if } \frac{3}{4} \leq \phi \leq 1 \end{cases} \tag{1.1}$$

· Sawtooth:

$$M(\phi) = \begin{cases} M_0 + M_W(\frac{1}{2} + \phi) & \text{if } \phi \leq \frac{1}{2} \\ M_0 + M_W(\phi - \frac{1}{2}) & \text{if } \frac{1}{2} < \phi \leq 1 \end{cases} \tag{1.2}$$

1.2 Parameters choice

The parameters we chose for our flanger plugin are the following:

- **Feedforward (Mix)** $\rightarrow$ KNOB
  It represents the amount of delayed signal that is mixed in with the original one. Having a value of 0 means that we are considering only the dry signal. Increasing this value (up to 1), one can adjust the balance between the processed signal and the dry signal.

- **Feedback** $\rightarrow$ KNOB
  Through this parameter it's possible to control, in practice, how much of the output signal from the delay line we want to send back through the device input. The range of possible values for this plugin is $[0, 0.50]$. We decided to set the end of the range at half of his theoretical maximum for personal tastes to reach a specific sound effect, because the more it approaches 1 the more emerges a metallic sound due to the sharpening of peaks and notches in the frequency response.

- **Delay** $\rightarrow$ KNOB
  It lets the user adjusts the minimum amount of delay of the LFO, in a range of $[1.00, 5.00][\text{ms}]$. For higher delay times our flanger behaved as a chorus, so we fixed the max at 5 ms.

- **LFO Width (Sweep Width)** $\rightarrow$ KNOB
  It allows the user to control the total amplitude of waveform of the LFO, in a range of $[1.00, 20.00][\text{ms}]$.

- **LFO Frequency (Speed)** $\rightarrow$ KNOB
  The LFO frequency can be set in a range of $[0.05, 2.00][\text{Hz}]$.

- **Shape of the LFO Envelope** $\rightarrow$ COMBO BOX
  It allows one to select which shape use for the LFO. For this plugin there are three possible waveform shapes: Sine, Triangle and Sawtooth.

1.3 Plugin implementation in JUCE

1.3.1 Processor implementation

The first thing we decided to implement is the Value Tree State, a class used to manage all the parameters of the plugin, or so to say, the entire plugin's state. It is very helpful in order to handle the connection between the objects in the Editor and the Processor via the instantiation of specific classes called Attachments, one for each type of graphical objects (i.e., slider, comboBox). An identification string is used for retrieving a parameter, and by using the Value Tree State, the post-condition of the get method ensures that the parameter is the newest value up to date with the user interface.

As a delay-based effect, the flanger is implemented using circular buffers, which can be considered as FIFO buffers (First In First Out). The dimension of the buffer is fixed and it is large enough to fit the amount of the maximum delay (sweep width + delay parameters) at any point in the LFO cycle. The actual length of delay at any time is controlled by the distance between the read pointer and the write pointer in the buffer. The increase or decrease of the delay is represented by the speed of the read pointer with respect to the movement of the write pointer. If the read pointer moves faster (slower), the amount of delay will decrease (increase).

In addition, we need to use low order polynomial interpolation to calculate the output of the delay line, including the case in which the delay length, expressed by the function M[n], is not an integer. First, we tried to implement a simple linear interpolation, but then we opted instead for the cubic interpolation because the former was causing some artefacts. In either case, we left the linear interpolation commented in the code for a quick comparison.

In order to handle the multiple waveforms of the LFO, we defined the wave functions through a switch case in which the phase varies incrementally. Given the wave functions, it easy to compute the current delay with the user-defined parameters width and delay and hence the delay read pointer:

```
float currentDelay = delay + sweepWidth * waveformFunction;
```

1.3.2 GUI implementation

In order to create the knobs, we created two custom classes (BlueKnobStyle,MagentaKnobStyle)[1] in the *PluginEditor.h* that use the juce method **LookAndFeel_V4**. The class draw the knob starting from a rotary slider, drawing two concentric circles and rotating a rectangle using the method juce::AffineTransform.

Then we defined two elements (blueKnob,mageKnob) of the classes in the AudioProcessorEditor that gives the style defined to the slider using the function *setLookAndFeel* in the editor compiler.

[1]The classes are exactly the same except the colour of the smaller circle

Chapter 2

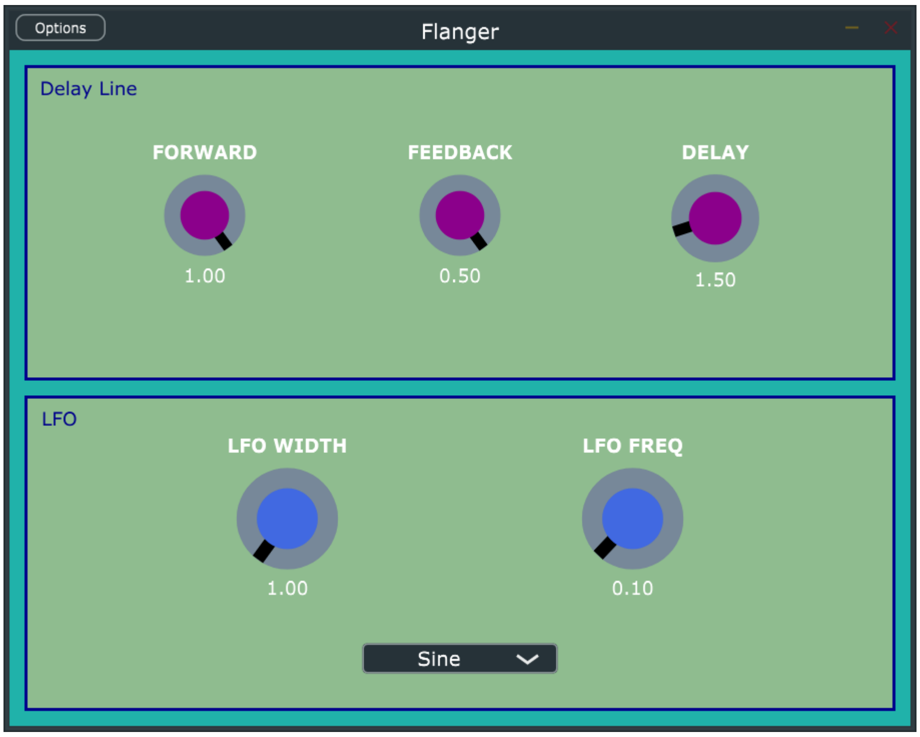
2.1 Result and Demo

To recreate the famous "jet passing overhead" characteristic sound of the flanger, we recorded a clean guitar and applied some distortion (adding higher frequencies enhance the effect). And finally, we fed the guitar recording to delay with the following parameters:

- **Feedforward** = 1
- **Feedback** = 0.25
- **Delay** = 1 ms
- **LFO Width** = 4.5 ms
- **LFO Freq** = 0.1 Hz
- **Waveform** = sine

The result can be played at the following link:
`inserire link`

Here bellow you can see the start-up window of the plugin with all the default parameters:



FORWARD 1.00
FEEDBACK 0.50
DELAY 1.50
LFO WIDTH 1.00
LFO FREQ 0.10

Figure 2.1: User interface