1. PROBLEM DESCRIPTION

Wah-wah processing has been used since the 1960s to add colour and interest to electric instruments in music, most commonly the electric and bass guitar, but can potentially be used on any instrument or audio signal. The wah-wah effect was originally hardware in the form of a foot pedal which was rocked back and forth by the user to create the effect of the instrument mimicking the human vocal sound ‘waaaaah’.

The autowah function proposed below is software designed to create this effect, but with flexibility of selection of several more advanced parameters that can be added after recording and exported as a .wav file to be used for production in a digital audio workstation.

Built into the function is an envelope filter to individualize the wah effect to the input. The auto-wah function creates an output specific to the audio input as the envelope modulates the wah filter with respect to the attack and decay (amplitude) of the signal (Cabrera, 2016). This means the output wah effect will be individual to every input audio without the need for a manual foot pedal.

It also contains two formant oscillators, and choice of oscillation rate and wet/dry mix.

2. SPECIFICATION

Autowah can be used for use on any digital audio signal, but is intended for use mostly for music, on singular instruments. The software changes the tone colour of the instrument by mimicking acoustic properties of human vowel sounds, oscillating periodically between two or more vowels.

The second and third resonant frequencies of the vocal tract, called ‘formants’ are interpreted cognitively as the tone colour of vowel sounds (Keen, 1999). The two vowel formants produce vowel sounds that are approximate and vary depending on the language, but can be simplified to the four shown in figure 1: i as in pizza, æ as in hat, u as in juice and a as in party. Schwa, represented by a, is the name for the neutral vowel sound used in English when there is no emphasis on that vowel sound, for example, the second ‘e’ in the word ‘presentation’.

![Figure 1: Relative positions of the 4 extreme vowel sounds with respect to the first and second formants (Martens, 2016)](image)

The autowah software is designed so both of these formants can be implemented on the input signal, to replicate human vocal sounds. The autowah can also operate with only one formant, as with traditional wah-wah pedals and filters, or with two, to oscillate between whichever vowel sounds desired.

By fixing the centre frequency of one of the formant low frequency oscillators, the autowah can sweep between two adjacent vowel sounds. For example, to sweep between u \(\rightarrow\) a set min_freq1 = 300 Hz, max_freq1 = 1000 Hz (or thereabouts) and set min_freq2 = max_freq2 (1500 Hz). To sweep between a \(\rightarrow\) æ min_freq2 = 1500 Hz, max_freq2 = 2200 Hz and set
\[ \text{min}_\text{freq1} = \text{max}_\text{freq1} \ (1000 \text{ Hz}) \]. These minimum and maximum frequencies are of course approximate, as vowel sounds vary with languages and accents. The smaller the difference between minimum and maximum frequencies for each formant, the more suppressed the vowel sounds, which tend towards schwa.

3. IMPLEMENTATION

Autowah implements both an autowah and an envelope so that the output wah-wah effect is individualized to the input audio signal, therefore providing a stylistically flexible and versatile digital processing system. The envelope filter first analyses the signal using the absolute value of a Hilbert transform function. The Hilbert transform function computes an analytic signal to calculate the instantaneous amplitude and frequency information contained within the audio signal (Mathworks, 2016). This information is then used in the wah filter to modulate the oscillations to fit the audio.

The autowah function uses a low frequency oscillator (LFO) to sweep a bandpass filter between the formant’s cut-off frequencies (Pro Tools Production, 2016).

\[ G = 2 \sin \left( \frac{\pi f_c}{f_s} \right) \]

Where \( G \) = filter coefficient, \( f_c \) = cut-off frequency, \( f_s \) = sample rate.

The processing is applied to the envelope, and therefore its amplitude will change proportionally to that of the input. There are two LFOs to represent the first and second formant, and these are produced using sinusoidally oscillating bandpass filters. Three zero vectors of the same length as the envelope are created \( h \), \( b \), and \( l \). These are the vectors that become the spectral shaping of the envelope.

\[
\begin{align*}
h(n) &= e(n) - l(n - 1) - \text{Db}(n - 1) \\
b(n) &= G_h(n) + b(n - 1) \\
l(n) &= G_l(n) + l(n - 1)
\end{align*}
\]

Where \( e \) = envelope vector, \( d \) = damping coefficient, \( n \) = sample.

These are looped through every sample in the envelope and \( G \) is recalculated for each sample. The oscillating bandpass filter processing is repeated for the second formant. These are summed, and then multiplied by the wet/dry mix, to create the wah effect. Finally, this is added to the original input, and then normalised to produce the output. (Pro Tools Production, 2016).

\[
\text{output} = \text{input} + m(F_1 + F_2)
\]
\[
\text{output} = \max(\text{output})/|\text{output}|
\]

Where \( F_1 \) and \( F_2 \) are the formant oscillators, \( m \) = wet/dry mix.

Figure 2 outlines the signal processing steps undergone by the function to produce the output effect.

4. EVALUATION

The DSP solution produces output audio as specified, with a wide range of possibilities of sounds for the user to explore. Despite having many input variables, autowah can be called using only 2 variables: the input audio and sample rate. After the function is called, a dialogue box (figure 3) is displayed for the other inputs, with brief explanations and default settings as a guide to the user.

Figure 3: Dialogue box with default settings as a guide to the user.

Some further suggestions are outlined in the following table, for some common uses of wah-wah effects.
### Table 1: Input Settings

<table>
<thead>
<tr>
<th>Input</th>
<th>Clean bass</th>
<th>Overdrive solo guitar</th>
<th>Clean solo guitar</th>
<th>Fuzz rhythm guitar</th>
</tr>
</thead>
<tbody>
<tr>
<td>min 1 (Hz)</td>
<td>200</td>
<td>700</td>
<td>300</td>
<td>700</td>
</tr>
<tr>
<td>max 1 (Hz)</td>
<td>700</td>
<td>1500</td>
<td>1000</td>
<td>1500</td>
</tr>
<tr>
<td>min 2 (Hz)</td>
<td>1500</td>
<td>1500</td>
<td>0</td>
<td>1500</td>
</tr>
<tr>
<td>max 2 (Hz)</td>
<td>1500</td>
<td>2200</td>
<td>0</td>
<td>2200</td>
</tr>
<tr>
<td>wet/dry</td>
<td>0.7</td>
<td>0.5</td>
<td>0.3</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Examples of these are provided with this proposal. Generally, higher oscillators are better suited for instruments with the high frequency information so higher maximum and minimum formant frequencies are more appropriate. Lower oscillators are more appropriate for instruments played in a lower register. White noise and single note examples have also been provided to demonstrate the variety of vowel sounds that are able to be created with the software.

The output audio is automatically saved in the path folder as a .wav file, ready to be imported to a digital audio workstation such as Protools.

### 5. ACKNOWLEDGMENTS

The author would like to acknowledge associate professor William Martens and Ms. Ella Manor for their guidance in completing this project and getting my software to actually work this time. Thanks to classmates Nash Cameron-Jeffs and Daniele Natoli for helping me with Protools to record my sound examples. Thanks also go to Gaye Adlington for provision of said Protools facilities, and performers Jay Glennan and Hamish Adlington.

### 6. REFERENCES


