Variable Range Effector Guitar Pedal

Capstone Proposal

Team Members:
Nicolas Binford
Sara Davila
Timothy Eames
Alexander Fleetwood
Karan Marwah
Ethan Neidhart

Advisor:
Bahram Shafai

Table of Contents

1. Introduction

Effects pedals are a popular tool that electric guitar players use to change the sound of their instrument while playing. There are numerous types of effects pedals that give guitarists a near-infinite possibility of ways to customize their sound. However, each pedal that exists in today's market changes the guitar's entire range of notes and a separate pedal is needed for each effect. It is quite cumbersome for a guitarist to switch back and forth between turning on and off separate pedals to achieve a robust sound.

Another way guitarists can implement multiple effects is by recording and editing a piece of music by changing the Musical Instrument Digital Interface (MIDI) in a Digital Audio Workstation (DAW). This can give the guitarist more capability with more effects, but it affects the musician's songwriting and ability to play the music live the same way it was edited. To recreate the edited piece of music live, a guitarist would have to use a laptop running a DAW that uses digital loops that temporarily record a section of audio and continuously play it back.

This is what we aimed to change. For our project, we are trying to design a guitar pedal that allows guitarists to change their sound by choosing one of several effects for a particular range of notes. This will give them even greater customizability and possibility for artistic expression with a single pedal. Having this functionality in an effects pedal would allow music with this type of sound to be played live, potentially changing the way songwriting and live performing is done by guitarists.

2. Problem Formulation

Our main design goal is to create a functional guitar pedal that allows a musician to customize the sound of the guitar by applying different effects to different ranges of notes along the neck.

As a group that consists of several musicians, we decided we wanted to create this pedal to allow guitarists to have more customizability while playing music live. It is easy to create a unique piece of music after it has been recorded by adding several effects on a DAW, but this greatly limits a musician’s ability to play this piece live or to assist with songwriting. By creating this pedal, we are infinitely increasing a guitarist’s creative expression by allowing multiple effects to be applied over different ranges in real-time.

Our solution is unique. No pedal currently on the market can provide the output signal with frequency-dependent effects. Such a standard piece of musical equipment, which we have redesigned to provide this benefit, is applicable to a wide array of musical situations. A guitarist using this single pedal can augment their live music by applying effects like reverb, distortion, or tremolo each to a different range of notes, as opposed to using the many pedals they would need to track throughout the performance. This greatly expands their capacity for artistic expression, not only because the effects and the ranges on the pedal are modifiable, but also because our pedal reduces the effort a musician needs to perform to their liking.

While developing this pedal, we proposed the concept to other musicians outside our group to see if such a pedal would be useful to them. Every one of them was interested in our project, noting that they had never heard of a pedal or any other piece of equipment like ours they could provide frequency-dependent effects. A few musicians even asked if they could test the final product.

3. Analysis

The following sections outline the different components of each subsystem. Through our research and proof of concept testing, we have narrowed down our final design which will be discussed in future sections of this report.

3.1 Pitch Detection

In order to split the guitar's frequencies into different ranges, our group researched how pitch detection was performed on current digital music hardware. We found that there are various signal processing algorithms that are to detect the frequency of a signal, which corresponds to the note being played on the guitar in our use case. Some of the algorithms are polyphonic, meaning that they can detect identify multiple frequencies being played simultaneously (corresponding to musical chords), and others are monophonic, meaning they can only detect one frequency being played at a time. We found that every polyphonic algorithm was proprietary, while the monophonic ones were open source and commonly used. It would have been impossible for us to write our own polyphonic algorithm given our time constraints and our limited knowledge of signal processing, so we decided to use an existing monophonic algorithm for our pedal, giving us the limitation that it would only work with our guitar playing one note at a time. After further investigation, we decided on the autocorrelation algorithm because of its simplicity to implement.

Autocorrelation works by taking a signal and delaying it multiple times. The original signal is then compared with each of the delayed copies, starting with a delay of zero, and a correlation is found between the two. The delay of zero will have a maximum correlation, since there is no change. As the delay approaches the period of the signal, the correlation approaches a peak, and leaves the peak as the delay increases past the period. This is because a signal delayed by its period is equivalent to the original signal. Once this peak occurs, its exact location is detected, giving us a delay equal to the period of the signal. Since frequency is the inverse of period, it becomes trivial at this point to determine the frequency. We wrote a small C program on our Arduino that runs this algorithm continuously.

3.2 Latency

To decrease latency and prevent one computer from having to do pitch detection and effecting in tandem, we decided to balance the workload across a Raspberry Pi (RPi) and Arduino. The Arduino was responsible for pitch detection (this was made simple as the Arduino already has a built in ADC) and the RPi was responsible for effecting the signal. By having these two operations happen in parallel rather than serially on one system, our delay was reduced to about 5ms.

3.3 Arduino & Raspberry Pi Communication

We used serial communication to allow our RPi to communicate with the Arduino over a serial bus. With a baud rate of 115200, communications were fast enough that the latency was barely noticeable. When the Arduino would determine a pitch, it would then send a corresponding bit to the RPi over serial.

3.4 Effect Processing

Instead of reinventing the wheel and writing our own algorithms to implement individual effects, we decided to use Pure Data. “Pure Data is a visual programming language developed by Miller Puckette in the 1990s for creating interactive computer music and multimedia works.” The program allows one to connect devices in modular fashion and make use of existing patches and modules created by fellow musicians for effects. Additionally, it supports serial communication and complex logic to route signals, communicate with other devices, and use both ADC and DAC.

4. Design

Block Diagram

Figure 1: High level blocks and workflow of the system

There are four main subsystems to the workflow: input (green), output (purple), analog to digital conversion (red), and DSP (blue). The input of the guitar signal is fed to the Arduino and Raspberry Pi simultaneously. The Arduino determines the pitch of the signal and through serial communication talks to the RPi. The RPi receives the pitch frequency (and thus within which range of frequency) from the Arduino. Combined with the user defined conditions from the switches, the RPi applies the appropriate effect. This processed signal is then sent out through a DAC to the speaker/amplifier.

4.1 Preamp

The pre amplifier consists of a non-inverting LM358 op-amp. Input from the guitar signal is read from an ⅛ inch 3.5mm female plug. This input is fed into the positive terminal on the op-amp that is DC biased with a large megaohm voltage divider. There is an AC gain of 16 and a DC gain of 1 from the amplifier. The analog input on the Arduino reads the change in AC signal between 0 and 3.3V, sitting centered at the middle of the range, 1.65V. These voltages are read by the 10-bit ADC and converted into a number between 0 and 1023

Figure 2: Preamp Schematic

4.2 Switch Network

The toggle switches used in this subsystem allow for the user input to the guitar pedal. In this prototype, 9 toggle switches were used for 3 effects with 3 different frequency ranges applicable to each. Toggle switches were implemented for the ease of sending sustained high or low signals to the RPi general purpose input-output (GPIO). As long as a GPIO is held high, the effect will be applied in the designated range. LEDs offer a visual indication to the user of what effects and ranges are being affected.

Figure 3: Switch Network

4.3 Arduino Pitch Detection

The pitch detection takes place on the Arduino in a C program, using the autocorrelation algorithm mentioned earlier. First, the signal data is read and saved to an array. Next, autocorrelation is run for each entry in the array (the delay being 1 entry in the array). The correlation is measured by the sum of the product of the two signals. Next, this sum is used to detect a peak correlation. If a peak correlation is found, then the period is the current entry. The loop then continues, running autocorrelation and peak detection on the next entry.

Peaks are detected using a state machine. After the sum is calculated with autocorrelation, the peak detection uses that sum and its current state to either take an action and advance the state, or do nothing. The loop then continues to the next entry, where the state is once again already determined. In state 0 (the default state), a threshold is set using the sum. Data is ignored for values under this threshold, and the state is immediately advanced to 1. In state 1, the peak detection looks for data above the threshold with a positive slope. Once this data is found, the state is advanced to 2. In state 2, the peak detection looks for data with either a negative slope, or a flat slope. When this occurs, the peak has been located. The period is set, and the state is advanced to 3, for which no functionality is defined because the peak has been detected.

When the loop is finished and the period of the signal has been detected, its inverse, which is the frequency in hertz, is stored in a variable. That variable is then checked against 3 if-else blocks that define the 3 frequency ranges (below 100 Hz, 100-200 Hz, and above 200 Hz). Depending on the range that the last note played was in, the Arduino will continuously write either a 0, 1, or 2 to its communication port. This is done to avoid an effect change if the signal fades out; for example, if a note is held for a long time, which the program will interpret as a different note being played. There is additional logic to just write the last value if an invalid frequency is detected; for example, an unexpected hum coming out of the guitar.

4.4 Pure Data Signal Processing

The selection and application of effects to the guitar signal takes place on the Raspberry Pi in a Pure Data program. The program our group wrote reads the guitar signal and the Arduino's communication port output simultaneously. Depending on frequency range returned by the pitch detection algorithm running on the Arduino, the Arduino writes either a 0, a 1, or a 2 to its communication port. The Pure Data program uses the "comport" object to read this value and uses it as the selector input to a demultiplexer. The demultiplexer's main input is the guitar signal coming from the ADC, read with the "adc~" object. Depending on the selector input, the signal is routed to one of three paths where the effects are applied. Each path allows one of three effects to be applied to that frequency range by interfacing with the GPIO. A separate "wiringPi_gpio" object is constantly reading the value of each physical switch in the switch network telling that effect to be applied to that range or not. This is done by multiplying the guitar sigal by the output of the GPIO. If the GPIO output is 1, the signal will be multiplied by 1 and sent to the appropriate effect patch. If the GPIO output is 0, the signal will be multiplied by 0, so nothing will be sent to that effect patch and that effect will not be applied. Thus, the multiplier for each effect in each range acts like a logical AND gate. The output of each effect patch is sent to the "dac~" object in the program, sending the final signal to the DAC.

The pedal has three possible effects that can be applied to the signal: fuzz, delay, and wah. The effects are implemented using patches, which are Pure Data blocks that use the language's built-in functions to modify the signal.

Fuzz Patch

The fuzz patch uses Pure Data's "clip" function to limit the amplitude of the signal to a certain range. Cutting off the peaks of the waveform like this produces the distorted signal that is commonly associated with rock music.

Delay Patch

The delay patch uses Pure Data's "delread" and "delwrite" functions to create delayed copies of the signal and send them to the output after the original signal. This results in hearing that note repeated multiple times until the signal fades out, like an echo. We experimented with how much time passes between each delayed copy until we found a value that seemed pleasant for the purpose of songwriting.

Wah Patch

The wah patch uses Pure Data’s “vcf” (voltage-controlled band-pass filter) and “osc” (oscillator) functions to create what is formally called a “spectral glide,” which makes the note mimic a human voice saying “wah wah.” This is achieved by oscillating between peak responses of a frequency filter. Like with the delay patch, we experimented with how much time one oscillation between the minimum and maximum of the frequency range took until we found a value that sounded pleasing to us.

5. Parts and Implementation

5.1 Project Timeline

Microcontroller, ADC, and Pitch Detection Research	■	■								
Build Preamp			■							
Test Pre-amp with Pitch Detection Code				■						
Test Pure Data DSP with Guitar					■					
Communication between Arduino and RPi						■				
Choose effects and frequency ranges							■			
Hard-coded prototype working								■		
Build & Test Switch Network									■	
Assemble Components in Box										■
	July 17	Aug 17	Sep 17	Oct 17	Nov 17	Dec 17	Jan 18	Feb 18	Mar 18	Apr 18

5.2 User Workflow

To use this pedal, one does not need to know know anything about Linux, Pure Data, or Arduino programming. A boot script on the Raspberry Pi ensures that every necessary program starts when it turns on, so all the user needs to do is plug the pedal into a wall outlet to start making music. The input and output are ¼” audio jacks, which is the standard for guitars and amplifiers. Once the pedal is turned on, a guitar is plugged into the input, and the output is plugged into an amplifier, the switch network can be used to select which effects are applied to which frequency range. Each switch represents an effect for a range. For example, there is a switch for the distortion effect for the low range, the middle range, and the high range. If no effects are applied to a certain range, it will just output a clean guitar signal.

Although the pedal is plug and play, it has HDMI and USB inputs available for debugging. If one connects a keyboard, mouse, and monitor to the pedal, they will see the RPi's Linux desktop environment and could modify the Pure Data program, which runs the effects, if desired. The pitch detection program running on the Arduino cannot be modified. The only way to modify the pedal's pitch detection would be to disconnect the Arduino from the RPi and flash a new program onto it.

6. Cost Analysis

This table includes our expenses from Summer 1 through the end of the Spring Semester and the completion of our project.

Item	Amount
Arduino Uno Rev3	$22.00
Raspberry Pi 3 Model B	$35.29
Sandisk 32GB Micro SD Card	$12.99
Sabrent USB Sound Card	$6.99
DFRobot Protoboard (2)	$11.60
HOSA ¼” to ⅛” Inch Adapter (2)	$11.40
Toggle Switches	$5.00
LM358 Op-Amp	$0.43
Pinfox Plastic Enclosure	$12.49
Total	**$118.19**

Ultimately, each of our materials are fairly inexpensive. The final cost of the materials of $118.19 is well below the allotted $1000 budget. Current guitar pedals for one effect cost upwards of $100, which allows our product to be competitive in the market as not only a multi-effect pedal, but one that provides a novel ability.

7. Conclusion

After meeting nearly every week to work on this project over the past 10 months, we were able to successfully create a guitar pedal that allows a guitarist to apply three different effects to three different ranges of notes on the neck. By the end of March 2018, we developed a fully functioning prototype that we were able to test extensively. Our pedal allows for an extremely simple setup requiring a user to plug the pedal into an outlet, plug the guitar into the input, plug a speaker or amplifier to the output, and toggle the switches for the effects. Our pedal allows both professional and hobbyist guitarists to customize their sound in real-time with no latency.

Moving forward, we are looking into getting a patent for our project. Since polyphonic pitch detection algorithms are currently proprietary, we look forward to them becoming publicly available to allow us to expand on our pedal.