Cristóvão Diniz Trevisan, Victor Volochtchuk de Araujo

Guitar Digitizer

Brazil

2017, v-0.0.1

Cristóvão Diniz Trevisan, Victor Volochtchuk de Araujo

Guitar Digitizer

Project presented as graduation material for the course of Electronic Engineering at UTFPR

Federal University of Technology - Paraná – UTFPR

Electronic Engineering

Graduation Program

Supervisor: Gustavo Benvenutti Borba

Brazil

2017, v-0.0.1

Cristóvão Diniz Trevisan, Victor Volochtchuk de Araujo

Guitar Digitizer/ Cristóvão Diniz Trevisan, Victor Volochtchuk de Araujo. – Brazil, 2017, v-0.0.1-

56 p. : il.

Supervisor: Gustavo Benvenutti Borba

Graduation Final Project – Federal University of Technology - Paraná – UTFPR
Electronic Engineering
Graduation Program, 2017, v-0.0.1.

1. Hexaphonic Guitar 2. Digitizer 3. MIDI 4. Pitch Detection I. Guitar Digitizer II. Gustavo Benvenutti BorbaIII. Federal University of Technology - Paraná IV. Electronic Engineering

CDU 02:141:005.7

Cristóvão Diniz Trevisan, Victor Volochtchuk de Araujo

Guitar Digitizer

Project presented as graduation material for the course of Electronic Engineering at UTFPR

Project Approved. Brazil, October 23, 2017:

Brazil

2017, v-0.0.1

This work is dedicated to those who supported our way into through enginnering course, even more when ourselfs tried to give up.

Acknowledgements

We give our special thanks to Eng. Mikhail Anatholy Koslowski, who gave us both intellectual (since he once used an equipament with the same purposed) and physical support (with compenents importation and equipaments). Also to our advisor who was present even in the moment that the initial idea came to life, in a situation nobody else would have stayed.

Abstract

Guitar are one of most popular instruments today, but there is one big disadvantage to use it: there is no good and affordable way to digitalize it's music. The biggest problem with this is the cost to annotate music, as it needs to be done by manually. This project tries to build one such system, building from passive hardware (hexaphonic pickup) to modern signal processing (pitch detection), attempting to produce a cheap and effective equipment for guitar music annotation by means of generating MIDI format data.

Key-words: guitar. digitizer. MIDI. pitch. detection. hexaphonic.

Resumo

Violões e guitarras estão entre os instrumentos mais populares da atualidade, mas existe uma grande desvantagem em os utilizar: não há um meio barato e eficaz para digitalizar sua música. O grande problema com isso é o alto custo para transcrever partituras, que atualmente é um processo manual. Esse projeto tenta construir um sistema com esse propósito, criando desde sensores passivos (captador hexafônico) até processamento digital de sinais moderno (detecção de nota), visando um produto barato e eficaz para anotação musical através da geração de dados no format MIDI.

Key-words: guitarra. digitalizador. MIDI. nota. detecção. hexafônico.

List of Figures

List of Tables

List of abbreviations and acronyms

MIDI	Musical Instrument Digital Interface
UI	User Interface
GUI	Graphical User Interface
API	Application Programming Interface
USB	Universal Serial Bus
DOM	Document Object Model
n.d.	No Date
fft	Fast Fourier Transform
ifft	Inverse Fast Fourier Transform
npm	Node Package Manager
DOM	Document Object Model
ADC	Analog to Digital Converter
DMA	Direct Memory Access
DIY	Do It Yourself
IDE	Integrated Development Environment

List of symbols

Ω	Ohm resistance unit
μC	Microcontroller
V	Volts
mV	milivolts

Contents

Introduction

Part I

Hardware

1 Pickup Project

Some electric guitars uses pickups consisted by magnets wrapped on copper thin wire, they *work* (WALLACE, 2004) using the variation of magnetic field, as the *Faraday Law* (NAVE, n.d.), created by the string vibration to generate the electrical signal correspondent to each tone frequency.

1.1 Requirements

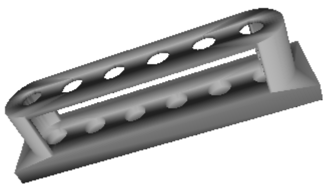
For assembly the pickup it was verified the requirements:

- pickup base to assembly the set up magnet+coil
- 6 magnets
- copper wire to wrap the magnets
- cover to attach the set up on the guitar

1.2 Pickup Base

After some studies it was decided to print the pickup base using 3D printer, this decision was due the reduction of the cost of the project. Than it was projected a 3D model using the software TO DO and printed this on 3D printer as the project showed on Figure 1.

Figure 1 – 3D pickup base project



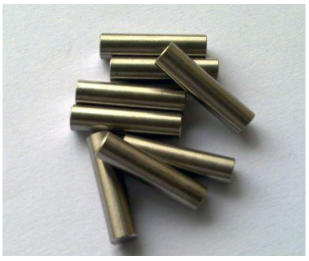
Source: made by authors

There was two materials to print the model, *PLA* (Polylatic Acid) (3D..., 2013) and *ABS* (Acrylonitrile Butadiene Styrene) (3D..., 2013). It was decided to print the model on PLA because it attends the requisites of robustness of the project and it is faster to print and have a lower cost when compared to the other material.

1.3 Set up Magnets+Coils

After print the pickup base it was assembled the coils around the guitar Figure 2 using thin copper wire. There was two dimensions of wire, 0.25mm and 0.5mm of diameter. It was tested the set up with both wires and it was chosen to use on the project the 0.5mm diameter wire because this dimension attended the pickup dimension restriction because it was decided to give 500 turns on each magnet to reach a minimal desired voltage value of TO DO mV.

Figure 2 – Magnets used on the project



Source: magnets (n.d.)

With the pickup already assembled it was verified the output signal. This signal pick was around TO DO mV and this value was so weak to send to the Analog-digital converter present on the microprocessor. With this dungeon it was verified that it is requested a circuit responsible for the signal amplification to be possible to read the signal with quality on the microprocessor.

2 Amplifier Circuits

Amplifiers circuits (AMPLIFIER. . . , n.d.) are circuits which increases an electrical input signal according to a specific transformation function (gain) for each different topology on the output. The circuit for small input signals normally is composed by an operational amplifier, resistors, trimmer potentiometers to adjust the gain and capacitors.

It was required by the project an amplifier which have a gain of 100 ad should be possible to supply with a sigle supply, ideally 5V provided by the USB port. The circuit should be compact to be a product differential, when compared to the existent systems which needs to connect on greater modules to perform the same action as the proposed on this project.

It was researched some types of amplifiers circuits on *book* (MILMANN; HALKIAS, 1981) and on *supplier catalogue* (CARTER, 2000). After verify some amplifiers topologies it was selected two different circuits which attends the project requirements. The first circuit uses the Texas Instruments INA 326 Instrumental Amplifier and the second uses the Texas Instruments TLV 4316 Operational Amplifier. These topologies are possible to use with single supply of 5V and the reach the desirable gain value without distortion on the project frequency range of work (audible frequency range).

2.1 INA 326

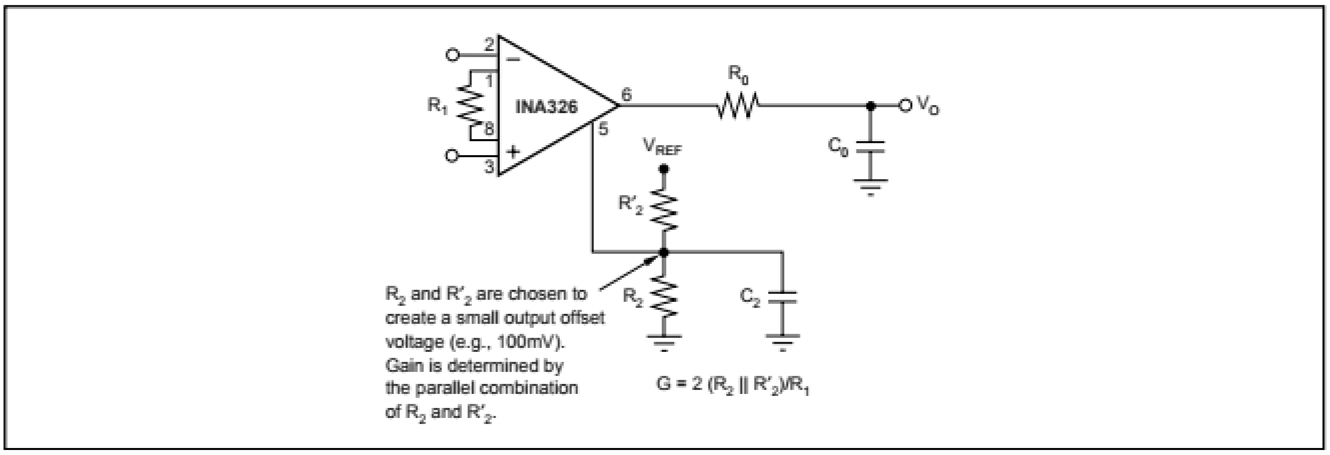
The project using the INA 326 IC, started with a research at the *component datasheet* (INA. . . , 2004) of this component and the *supplier catalogue* (CARTER, 2000). On theses documents it was verified the circuit topology Figure 3 which provides the desired Gain to the electric signal from the pickup respecting the project's initial requests.

This circuit amplifies the input signal and the gain is obtained by the Equation 2.1 provided by the *component datasheet* (INA. . . , 2004):

$$G = 2 * \frac{(R_2||R_2')}{R_1} \tag{2.1}$$

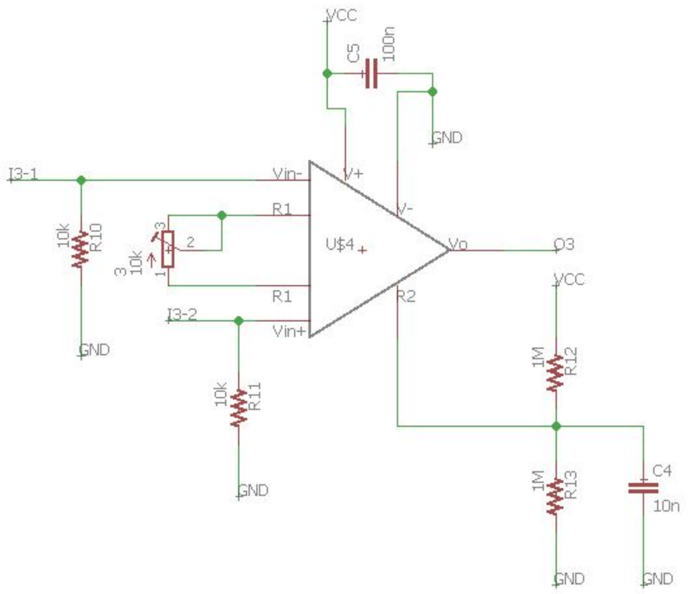
On the project the Resistor R_0 and the Capacitor C_0 was excluded because it was not relevant on the output to this project. It was calculated the values for the components and then developed a schematic model to perform some simulation tests to verify the functionalism of the circuit for the desired application. The schematic circuit Figure 4 was developed using the software CadSoft Eagle Professional 7.6.0 until the final version.

Figure 3 – INA 326 topology


INA326
R$_1$
R$_0$
V$_O$
C$_0$
V$_{REF}$
R'$_2$
R$_2$
C$_2$
R$_2$ and R'$_2$ are chosen to create a small output offset voltage (e.g., 100mV). Gain is determined by the parallel combination of R$_2$ and R'$_2$.
G = 2 (R$_2$ || R'$_2$)/R$_1$

Source: INA. . . (2004)

Figure 4 – INA 326 Schematic Circuit



Source: made by authors

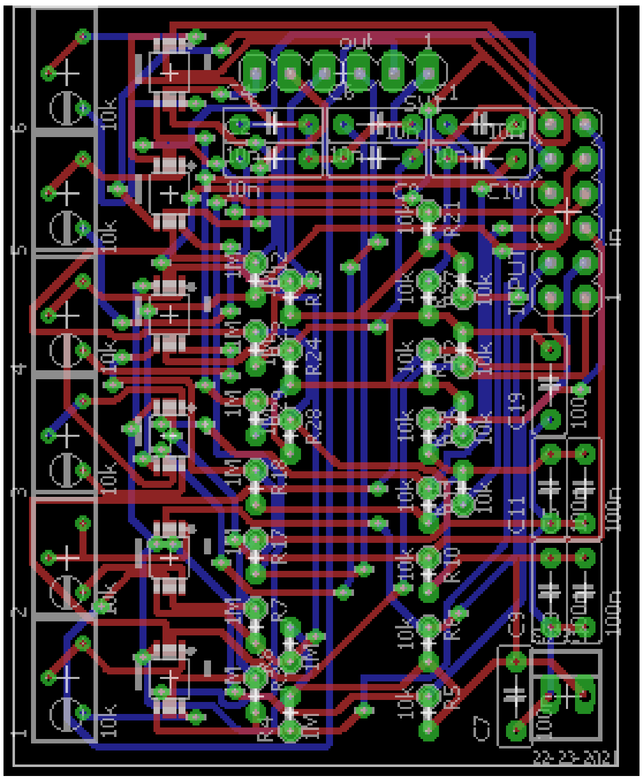
Using the components values showed on the circuit Figure 4 it is possible calculate the Gain of the circuit as the Equation 2.2 shows

$$G = 2 * \frac{(1M\Omega||1M\Omega)}{10k\Omega} = 2 * \frac{500k\Omega}{10k\Omega} = 2 * 50 = 100 \tag{2.2}$$

The amplifier circuit was projected to each channel as showed on **??**. So the circuit needed 6 IC INA 326 to be complete. After project the schematic circuit to each channel it was developed the PCB project, using the same software described on the schematic modeling. The PCI project Figure 6

This PCB project was projected on a dual layer board scheme, using 15 mils of minimum width for the conductive tracks. It was assembled on FR4 dual layer copper board, combining the through-hole and surface-mount technologies. This choice was made due the facility of the assembly of the through-hole components and the availability of the IC only on surface (SOP-8) encapsulation. It was sent the files to one person who works with printing PCB boards. After the processes the PCB was like showed on **??**.

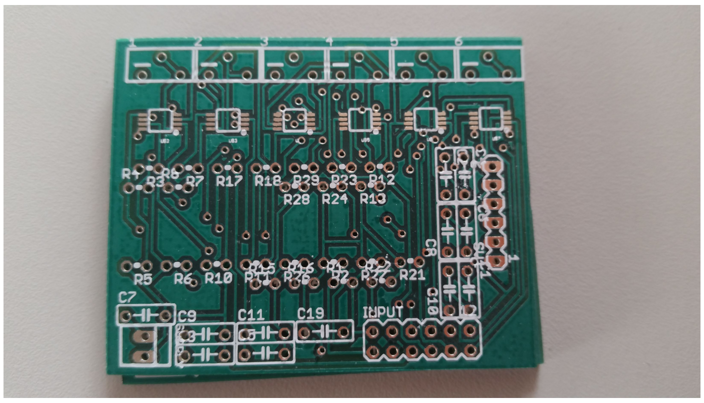
Figure 5 – Projected INA PCB



Source: made by authors

On this project it was used:

Figure 6 – PCB Project

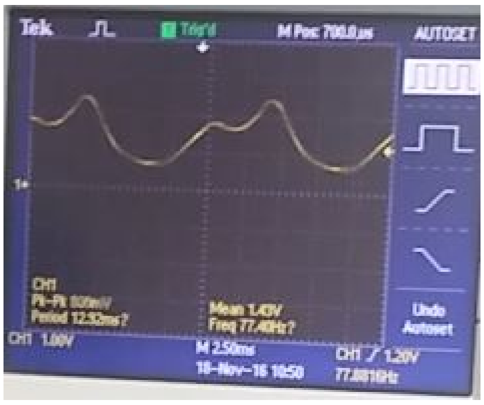


Source: made by authors

- 6 Instrumental Amplifiers INA 326;
- 6 10kΩ Trimmer Potentiometer;
- 6 10nF Ceramic Capacitor;
- 6 100nF x 50V Electrolytic Capacitor;
- 12 10kΩ 5 Percent Tolerance Resistor;
- 12 1MΩ 5 Percent Tolerance Resistor;
- 1 6 positions Pin bar;
- 1 12 positions dual track Pin bar;
- 1 2 positions Pin bar;

After soldering all the components it was performed some bench tests to verify the functionality of the amplifier circuit. The result showed that the circuit attends the required function, amplifying the signal from a pick of 10mV to a pick of 1V, proving that the circuit is working perfectly and attending the demand of the project. After the bench test the system was connected to the pickup to verify if the guitar signal would be amplified as needed to the conversion process. The result was satisfactory and attended well the purpose to the project, as showed on Figure 7.

Figure 7 – Amplified pickup signal



Source: made by authors

2.2 TLV 4316

The circuit using TLV 4316 started on the same way of the INA 326 project. It started verifying the *component datasheet* (TLV. . . , 2016) and the *supplier cataloge* (CARTER, 2000). On these documents it was verified some strategies, which are used on this project. For this IC it was needed use the concept of *Virtual gounding* (CARTER, 2000), neede to the operation of a comon Operational Amplifier on a single supply configuration

Part II

Firmware

3 Specifications

The firmware is basically an analog sampler, all it has to do is sample six analog channels, add a header (to identify the beginning and check continuity) and send it through USB. Lets start by listing the requirements for the hardware.

3.1 Requirements

a) Super cheap
b) 6 analog channels (more precision is better)
c) High sample rate (at least 10 kHz for each channel, but ideally 44 kHz or more)
d) Fast USB support, to send the data with headers

Based on this requirements the minimum transfer speed can be calculated, let's consider that a header will be set for every 252 samples (42 for each channel) and it has 4 bytes (3 of identification - to assure it is the header and not some data - and a counter). Previewing the worst case, each sample is 2 bytes long. The transfer rate given by Equation 3.1.

$$transfer\ rate = \left(\frac{channels * \frac{bytes}{sample} * \frac{samples}{package} + header\ size}{\frac{samples}{package}} \right) * f_s[B/s] \tag{3.1}$$

Considering that f_s has to be somewhere between 10 kHz and 50 kHz the transfer rate numerical result is given by Equation 3.2.

$$transfer\ rate = \left(\frac{6 * 2 * 252 + 4}{252} \right) * f_s = 12.0159 * f_s = 120.159 - 600.794[kB/s] \tag{3.2}$$

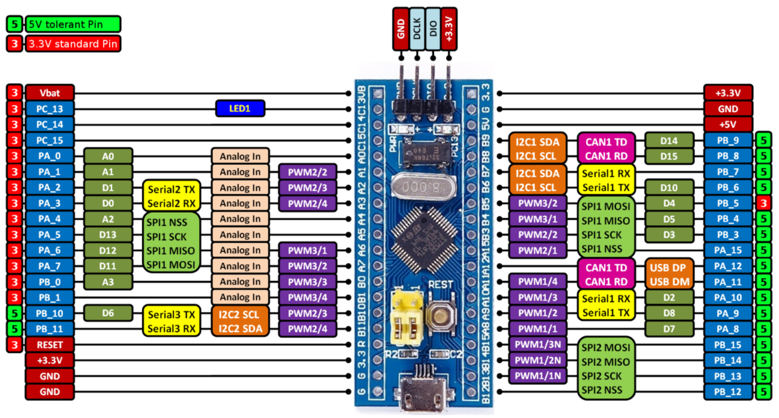
USB transfer speed is usually referred in Mbps, which gives us a range between 961.27 and 4806.34 Mbps. This is too high for serial communication (typical max of 1 Mbps) so we need to add a requirement for raw USB support, which allows bulk transfer that can a have transfer rate up to 12Mbps (USB full-speed standard).
We still need to choose or exact sample rate (f_s), but first let's select which hardware will be used.

3.2 Microcontroller Selection

There are too many microcontrollers that fit our requirements, but the most popular and cheap one is clearly the ARM from ST called STM32F103C8T6 (STM32F103XX. . . ,

2015), and that is why it was selected.
It has eight 12 bits ADC inputs (with 2 parallel channels), DMA for the ADCs and USB full-speed support. It's also relatively fast (72 MHz clock, 32 bits architecture). All this for under 2 U$D in a developing board from China (the actual μC is under 0.2 U$).
The board an it's pinout can be seen in Figure 8.

Figure 8 – STM32F103C8T6 Board


5 5V tolerant Pin
3 3.3V standard Pin
GND
DCLK
DIO
+3.3V
3 Vbat
3 PC_13 LED1
3 PC_14
3 PC_15
3 PA_0 A0 Analog In
3 PA_1 A1 Analog In PWM2/2
3 PA_2 D1 Serial2 TX Analog In PWM2/3
3 PA_3 D0 Serial2 RX Analog In PWM2/4
3 PA_4 A2 SPI1 NSS Analog In
3 PA_5 D13 SPI1 SCK Analog In
3 PA_6 D12 SPI1 MISO Analog In PWM3/1
3 PA_7 D11 SPI1 MOSI Analog In PWM3/2
3 PB_0 A3 Analog In PWM3/3
3 PB_1 Analog In PWM3/4
5 PB_10 D6 Serial3 TX I2C2 SCL PWM2/3
5 PB_11 Serial3 RX I2C2 SDA PWM2/4
3 RESET
+3.3V
GND
GND
+3.3V
GND
+5V
I2C1 SDA CAN1 TD D14 PB_9 5
I2C1 SCL CAN1 RD D15 PB_8 5
I2C1 SDA Serial1 RX PB_7 5
I2C1 SCL Serial1 TX D10 PB_6 5
PWM3/2 SPI1 MOSI D4 PB_5 3
PWM3/1 SPI1 MISO D5 PB_4 5
PWM2/2 SPI1 SCK D3 PB_3 5
PWM2/1 SPI1 NSS PA_15 5
CAN1 TD USB DP PA_12 5
PWM1/4 CAN1 RD USB DM PA_11 5
PWM1/3 Serial1 RX D2 PA_10 5
PWM1/2 Serial1 TX D8 PA_9 5
PWM1/1 D7 PA_8 5
PWM1/3N SPI2 MOSI PB_15 5
PWM1/2N SPI2 MISO PB_14 5
PWM1/1N SPI2 SCK PB_13 5
SPI2 NSS PB_12 5
REST
PWR
PC13

Source: Hudak (n.d.)

3.3 Sample Frequency

Usage of the ADCs for the selected μC can be optimized by using continuous sampling mode in conjunction with DMA (STM32F103XX. . . , 2015, ch. 11). In this mode the sample frequency is controlled by a register that sets the convolution time, which gives more precision the more ADC clock cycles (longer) it takes to sample.
The first variable chosen for this setup is the ADC clock, which is set by dividing the μC clock by 2, 4, 6 or 8. The ADC also has a maximum clock of 14 MHz. Taking in account the μC clock of 72 MHz, the highest possible value for the ADC clock is 12 MHz, which is set by a divider value of 6.
The last value to be chosen is the mentioned convolution time (T_c is calculated as the selected value plus 12.5 ADC clock cycles), which gives a sample frequency calculate by Equation 3.3.

$$f_s = \frac{ADC\ clock}{T_c + 12.5} * \frac{ADCs}{channels} = \frac{12}{T_c + 12.5} * \frac{2}{6}\ [MHz] \tag{3.3}$$

Table 1 shows the calculated results for each possible register value (using Equation 3.3). Based on it the chosen sampling frequency is 47.619 kHz. By using Equation 3.2 we can also calculate the actual data transfer rate (for all channels, including header), resulting in a total of 572,185 kHz. This last value will be used to test the USB communication.

Table 1 – ADC Sampling Frequencies

Register Value	**Convolution Time [cycles]**	**Sampling Frequency [kHz]**
000	1.5	285.71
001	7.5	200
010	13.5	153.85
011	28.5	97.56
100	41.5	74.07
101	55.5	58.82
110	71.5	47.62
111	239.5	15.87

Source: made by authors

4 Implementation

4.1 First Attempt

We first tried to build our firmware from scratch, using the tools given by the manufacturer, essentially a set of driver abstractions (HAL drivers). The problem found is that these abstractions are too slow, and don't work when the firmware uses the hardware close to it's limits (as we do for both transfer and sampling rates).

4.2 Second Attempt

In our research we found an amazing open source project called MiniScope (MINISCOPE, n.d.), in which a few options of low budget DIY digital scope (using different μCs) are presented. One of the μCs used by MiniScope is the one we selected, so for our implementation we took it's firmware as a base project. In this project the author claims to sample and transfer two channels at 461 kHz (but 8 bits only), which is very close to our needs (we need a little more transfer but much less sampling speed).

4.3 IDE

As we take MiniScope as a base project we will be using the same IDE as it, named CooCox (COOCOX, n.d.). It has a full set of tools, and it's completely free (no limitations).

4.4 Modifications

The base project samples 2 channels at a different speed, bit rate and does not add any headers to the data. It also has some code to answer a few commands. It was as simple as setting up the registers for 6 channels, changing the sample size, placing our already chosen speed (section 3.3) and removing any unused code.
The act of changing the sample size was not done by registers. MiniScope was already sampling with 12 bits, but it was ignoring the least significant ones when filling the USB buffer. What we have done is to change the bits alignment and putting all the data received from the ADC to the USB buffer.

4.5 Testing

At first an attempt using the OS (Windows at that time, later Ubuntu) default driver as made. That did not work well, as it is too generic and thus slow.
At a second try, a simple libusb (LIBUSB, n.d.) program was built to test the transfer rate (calculated in section 3.3). The reported result is almost perfectly the calculated one.

4.5.1 Repository

Again, all code is available at GitHub (TREVISAN, 2017a).

Part III

Software

5 Tools Selection

5.1 Top Level Requirements

The exact implementation of each of the items will be discussed later, but to simply set our requirements a general list of them is:

a) Desktop GUI

b) Efficient signal processing (for pitch detection)

c) Real time graph visualization of the signals (oscilloscope like)

d) Access to *libusb* (LIBUSB, n.d.) API

e) Access to MIDI API

5.2 Language Choice

5.2.1 Java

The first choice was Java, as it meets all requirements. Desktop GUI can be done using Swing, a good library for pitch detection is also available (called TasosDSP (SIX; CORNELIS; LEMAN, 2014)). There is also a binding to libusb called usb4java and native MIDI support.
Following that idea a functional prototype was built, but a few problems came to rise. The first is that usb4java high-level API had bugs and was not working correctly. The solution was to fall back to the lower level API, but that made things much more complicated as threading and synchronization problems had to be dealt with. There was no good library for real time visualization either, which made really hard to both debug and tune the frequency detection algorithm. On top of that Swing is at least non-pleasant compared to more modern UI programming, so a second approach came to be.

5.2.2 JavaScript

In alignment with both current work experience and world programming tendencies JavaScript was taken as a choice. We will see that requirements fit much better now, for the following reasons.
For the desktop GUI, JavaScript has a few nice and mature Desktop GUI frameworks, like Electron and NW.js.
JavaScript is an interpreted language, and for that has a low efficiency when compared

to C++ or Java. That is huge problem, but there is an easy overcome. As this project tries to build a desktop application, Node.js will be used ultimately, and it has support for C++ bindings. That means the JavaScript code can call a compiled C++ library to calculate the pitch, thus solving the problem.
Graph visualization should not be a problem either as there are a lot of libraries for that. The most problematic requirement in Java was *libusb* support. It is available in JavaScript using *node-usb* (NODE-USB, n.d.), and a few simple tests returned good results with a much simpler API. MIDI was also tested and worked just fine.

5.3 Desktop Framework

Now that JavaScript is set as our final selection we need an environment to run it. There are two already listed really mature and popular choices: Electron and NW.js.
At first Electron was used to build a test application, because it is the most popular of the two (in fact even the editor used to write this words is built with it), but the pitch detection call was running slowly. As a mater of fact it was running much faster using pure JS code rather than the C++ library. A deeper research was needed, and the way Electron worked was getting in our way, but first it's necessary to know what Node.js is.

5.3.1 Interpreter

JavaScript is a interpreted language and thus needs an interpreter. The most common one is Google V8, which happens to be the same one used in most web browsers as well as in Node. The difference between browsers and Node is simply the API that comes with them. Web needs firstly to access the UI (html) and ways to modify it, it also needs secure and limited access to hardware and internet calls. Of course that means web JavaScript code cannot use C++ libraries directly.
On the other hand Node is a more pure version of V8, it also gives the possibility to write and call C++ code (feature needed for this project), which ultimately makes it as capable as any desktop program can be. Node also comes with a hand-full set of native resources (like file system and full communication access), but it does **not** provide any kind of GUI. Knowing that it is possible to have a better understanding on how the two desktop environments work.

5.3.2 Electron

Electron by design has at least two processes running (HOW. . . , 2017), one for the "web" and other for Node access. The answer for how the web process access native resources is also to why the execution of the C++ processing library was slow: it uses inter-process communication (IPC). IPC makes things a lot slower, which ultimately makes impossible to use Electron in this project.

5.3.3 Node Webkit

Differently from Electron, NW.js (NODE. . . , n.d.; HOW. . . , 2017) takes the Node environment and combines it with Chromium into a single process, removing the use of IPC. Initial tests reported that the pitch detection library has fast execution as expected.

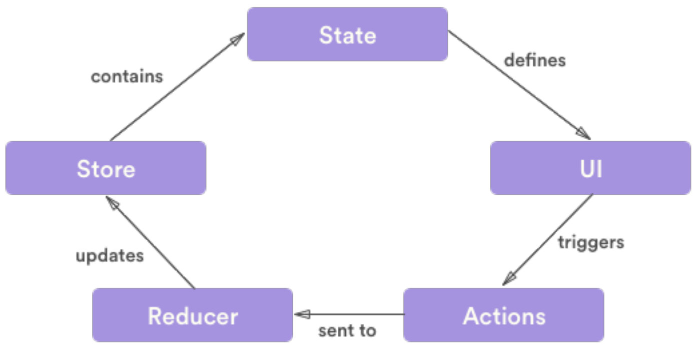
5.4 Architectural Tools

NW.js will go only as far as to give access to both Node and DOM API's. But that is too crude, and not what we wanted by given up on Java Swing. Again based on current work experience and world tendencies the setup chosen is React + Redux.
React (REACT, n.d.) is a library created by Facebook and world wide used for UI applications. It uses a declarative component-based system that makes it easy to build scalable and reusable code.
React only go as far to help building the UI, but we also need to pass the state of the application to the UI components, and that is where Redux (REDUX, n.d.) comes in. It keeps all the application state stored in a single place, described by transition functions. That makes the storage system easy to be tested and used, because all actions (that modify the state) must be well defined and it doesn't rely on the UI (React), making it easy to test.
Figure 9 shows the flow of an application that uses React + Redux. It is obvious to see the simplicity it has, a single path must be followed. This simplicity is what makes it much easier to use against other frameworks like Java Swing.
There is still the choice of the visual library to use, and the chosen one is Semantic UI React (SEMANTIC. . . , n.d.). It has some nice and robust React components to build a well designed application.

5.5 Fast Signal Processing

Pitch detection is a heavy problem to solve, and good implementations are time consuming, so we need efficiency to run it real-time. The library already said to be used didn't actually existed, the only one available was a pure JavaScript library (PITCHFINDER, n.d.) which is not suitable for this project. The solution was to build our own library based on both the pure JavaScript one and TarsosDSP (SIX; CORNELIS; LEMAN, 2014). Implementation details discussed further on chapter 6.

Figure 9 – React Redux Flow Diagram



Source: Getting. . . (2016)

5.6 Real Time Visualization

There are lots of charting libraries available for use with web interfaces (and by extension NW.js), unfortunately none was good fit for real time high density signals such as audio. The solution was again to build one, since all other things are looking to run smoothly in JavaScript, implementation details on chapter 7.

6 Pitch Detection

Pitch detection is simply frequency detection with the restriction of note quantization, Table 2 shows the base frequency for each of the 12 existent notes. Multiples of the same frequency are seen as the same note on a different range, known as octave. Even though the quantization makes things simpler it's still a hard task, even more for

Table 2 – Notes Frequencies

Note Name	Frequency
A	440.00
A#	466.16
B	493.88
C	523.25
C#	554.37
D	587.33
D#	622.25
E	659.25
F	698.46
F#	739.99
G	783.99
G#	830.61

Source: made by authors

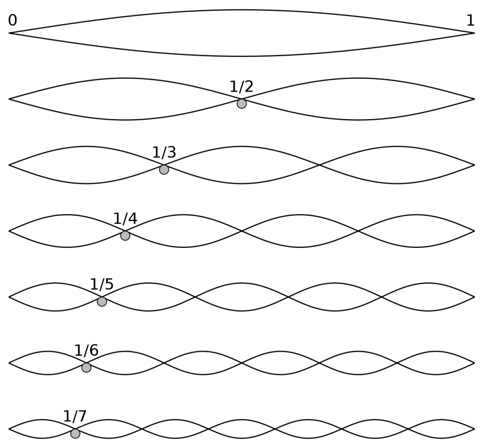
instruments where there is the presence of harmonic series. Harmonic series notes are multiples of the fundamental frequency (most important note) produced by integer sections of the instrument vibration. Figure 10 shows an visual representation of why they exist. The existence of them as well as the presence of both inter-signal and white noise makes necessary the use of non-trivial algorithms for pitch detection, and two of them will be discussed next.

6.1 YIN algorithm

Autocorrelation is a well know function to calculate a signal's fundamental frequency, but it gives too much error for this project use case. YIN (CHEVEIGNÉ; KAWAHARA, 2002) is a method that uses a few improvements to the autocorrelation method, achieving a much higher precision. It can also be implemented with logarithmic growth as the autocorrelation can be calculated using the fft and ifft algorithms. The algorithm can be divided in 6 steps, as follows:

1. Autocorrelation
2. Difference

Figure 10 – Harmonic Series



Source: Wikipedia (2017)

3. Cumulative mean normalized difference
4. Absolute threshold
5. Parabolic interpolation
6. Best local estimate

It's important to notice that the absolute threshold is a controlled attempt to regulate the error introduced by the harmonic series (as in Figure 10), it thus gives preference to lower frequencies (below the threshold).

6.2 MacLeod algorithm

MacLeod (MCLEOD; WYVILL, 2005) goes for another approach, using the square difference function. More precisely it uses a special normalized version of it. The best result is then calculated by means of using a parabolic interpolation of the highest peak and its two neighbors, this process also gives a threshold constant that limits the detection of the neighbors, thus the possibility of some tuning. As we will see this gave the best results for our project after some tuning.

6.3 Implementation

Implementation for both algorithms follow the same pattern, taking the pretty Java code of TarsosDSP (SIX; CORNELIS; LEMAN, 2014) and replacing the syntax and data structures with C++ ones (using standard library for containers). There is also a

JavaScript bridge for data type conversion, so we can use the library calls with simple arrays of numbers.
The implementation is not using fft for logarithm growth yet (but quadratic growth instead), following the TarsosDSP library. The faster implementation is kept as a goal for future improvement. All code is available as an open source project at Github and npm (NODE-PITCHFINDER, n.d.).

7 Data Visualization

The objective of data visualization is to live debug and tune both hardware gain and algorithms control constants. The ideal case is to have both real time chart as well as a buffered/triggered one, essentially something like an oscilloscope. That has to be performant for real time audio signals, sampled at more than 47 kHz.
There are a great amount of JavaScript DOM libraries for charting, but most of them are too automatic or have too much details, making them too slow for our need. The solution was to build our own library for that, again available as an open source project at both GitHub and npm (REACT-PLOTTER, n.d.).

7.1 Requirements

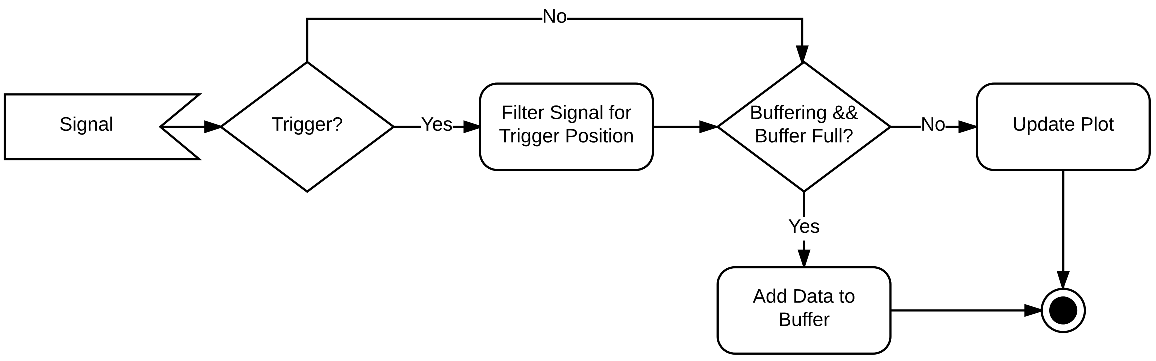
a) Be a React Component
b) Automatic calculations for array input
c) Option for triggering
d) Option for buffering
e) Minimum Redraw
f) Fixed Height/Width

7.2 Algorithm

Triggering and buffering are achieved by using a filter that only calls the plotting function when the options are met. This filter is simply the function called to add data, and it is represented by Figure 11.
For the actual drawing a triple buffer technique is used, one for holding the last state (called plotBuffer), on for drawing (called drawingBuffer) and finally the one actually rendered (called canvas). That last one is needed so the arrows don't get saved on the drawing scene.
For linear plot time a translation is established, in a way that only the new points will be drawn, the past ones are only translated to the left. The steps of the algorithm are as follow:

1. Clear drawingBuffer
2. Copy plotBuffer to drawingBuffer translating (removing) extra data

Figure 11 – Add Function Diagram



Source: made by authors

3. Draw new data on drawingBuffer

4. Copy drawingBuffer to plotBuffer

5. Copy plotBuffer to canvas

6. Draw arrows on canvas

7.3 Implementation

Using the listed requirements (section 7.1) a minimum API is built as a React component. Being such all it gives is a set of properties, for which the chart is drawn (when needed), they are listed in Table 3. The style property is a function that is called to

Table 3 – React Plotter Props

Property	**Type**	**Description**
style	Function	Style function (called to print the data)
[trigger]	number	Use trigger
[onlyFull=true]	bool	When using trigger it tells if the view should wait for a complete dataset before updating
[width=300]	number	
[height=150]	number	
[initialData=[ ]]	number[ ]	
[appendData=[ ]]	number[ ]	
[dataSize=100]	number	
[pixelSkip=1]	number	Pixels between points
[max=100]	number	Maximum Y Value
[min=-100]	number	Minimum Y Value
[useMean=true]	bool	Use mean calculation, otherwise median

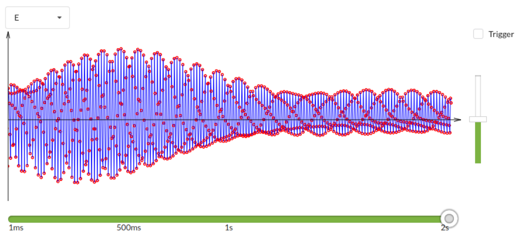
Source: react-plotter (n.d.)

render each point. Two styles were built, a line plot (points are connected by a straight line) and a digital plot (digital signal standard chart, not used in the final version of this project).

7.4 Results

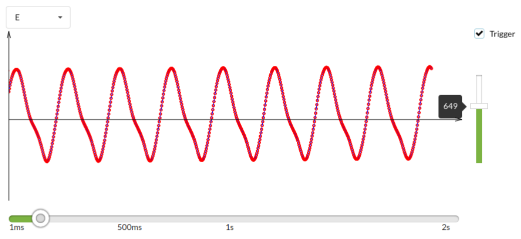
The results are more than satisfactory, tested to be able to run multiple plots of audio speed signals at the same time without much effort. Figure 12 and Figure 13 show how the visualization looks on the project, but full details and working examples are also available at GitHub (REACT-PLOTTER, n.d.).

Figure 12 – Real Time Plot



Source: authors

Figure 13 – Triggered Plot

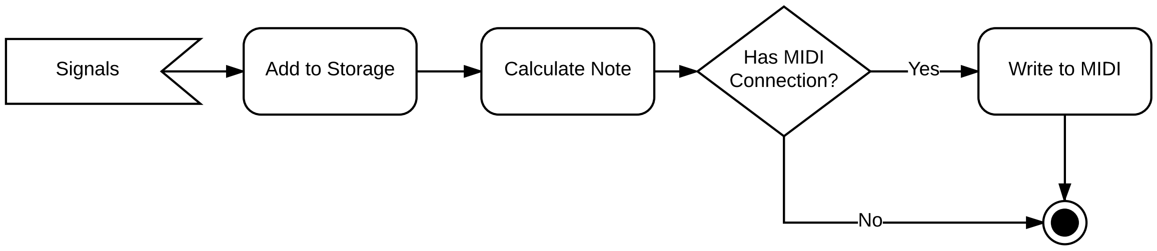


Source: authors

8 Main Program

The general idea is to build an UI with three main divisions: Home (with device selection), midi selection and signal-to-MIDI connections, Plot (with the signal visualization) and Options (with the algorithm tuning options, as well as virtual MIDI creation). When a device is selected it's signals will go through a simple process, as in Figure 14.

Figure 14 – Signals Flow Diagram



Source: made by authors

8.1 GUI

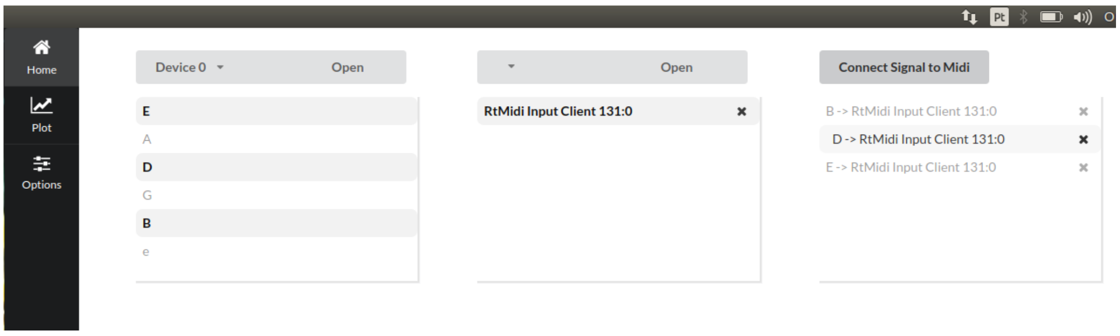
8.1.1 Home

The home page has three horizontally divided sections: Device and signals, MIDI selection and list, and finally connections, as in Figure 15.
The first is used to open the device (there can be only one used at a time). When it is open a list of signals will be displayed (six of them, named as each guitar note). Each signal can be selected so it can be connected to a MIDI device.
The second is to open any given number of existing MIDI devices, which will be listed bellow (can be also closed). The listed devices can also be selected, but only one at a time.
The final section is used to establish the connections, given the selected signals and MIDI, the connections are showed in the box bellow and can be deleted.

8.1.2 Plot

The Plot page is a simply a react-plotter (REACT-PLOTTER, n.d.) component with a few visual controls, being: a dropdown to select which signal is being displayed, a checkbox to enable trigger, a slider to control the time range and another slider to control the trigger value. The full page is as in Figure 16. It was chosen to show only one plot

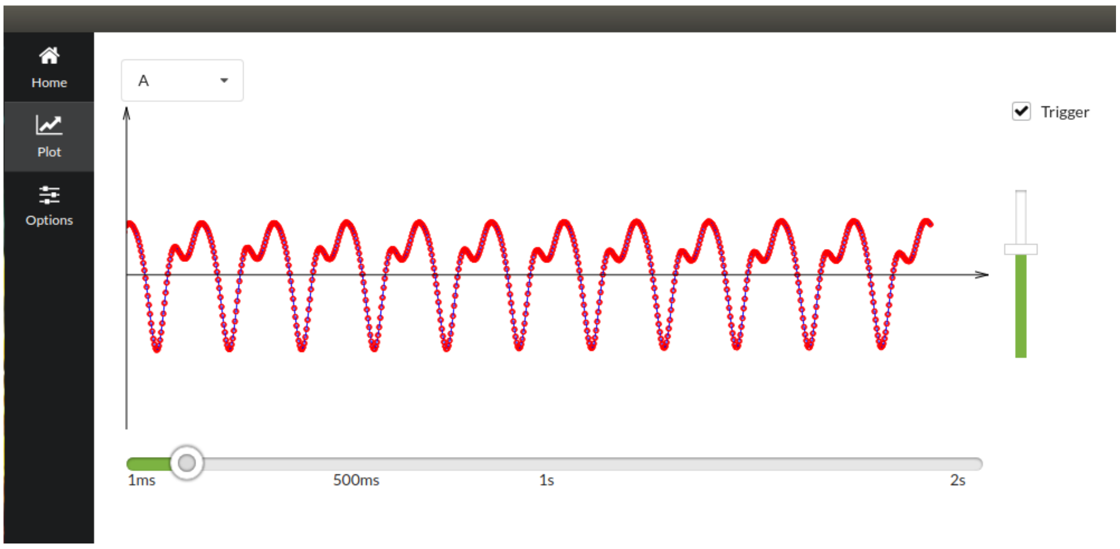
Figure 15 – Home Page


Home
Plot
Options
Device 0
Open
Open
Connect Signal to Midi
E
A
D
G
B
e
RtMidi Input Client 131:0
B -> RtMidi Input Client 131:0
D -> RtMidi Input Client 131:0
E -> RtMidi Input Client 131:0

Source: authors

at a time so it's size is bigger, easier to see. This also makes the program a little more performant.

Figure 16 – Plot Page

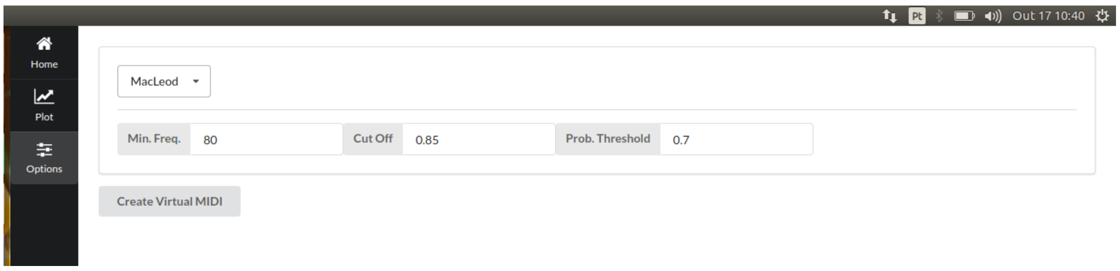


Source: authors

8.1.3 Options Page

The options page is simply an UI to control the used algorithm (from chapter 6) and it's parameters. For both Linux and macOS it's also possible to create virtual MIDI devices, from which our program can write and a synthesizer can read. For Windows this is still possible, but using a hacky solution (since Windows does not give any official API for this) - the easiest option being LoopBe1 (LOOPBE1, n.d.). The page can be seen at Figure 17.

Figure 17 – Options Page



Min. Freq.	80	Cut Off	0.85	Prob. Threshold	0.7

Source: authors

8.2 Implementation

8.2.1 Resources

At first let's take a look at our resorces:

a) USB device: only one can be connected

b) MIDI devices: a list of them can be used at any time

c) Signals: fixed for the only connected device

It's easy to see they are all global, so they can be represented as static classes. But JavaScript has good functional programming capabilities, which are very suitable for global resources. JavaScript imported modules are also scoped, by default, this means it works like a C++ namespace, keeping our static resources separated in a nice way. Taking these mentions in account three modules were built for resource easy access, being MIDI, USB and signal processing.

8.2.2 Entry Point

The entry point for our project is both a declarator and connector. It allocates all needed structures (or calls the module that does it), the most significant one being the Redux store (REDUX, n.d.) - store being a short for storage, which is where all of our application visible state is held.
The entry point also connects all callbacks and logic in a declarative way. In this single file all of the program's internal functionalities are declared, so much that if you read it you should also understand the entire program.

8.2.3 Reducers

Redux stored data is not defined by a set a properties, like typical OOP applications, instead it uses a more functional programming paradigm. The storage is defined by transfer

functions, each of them describing actions that can modify the current state by returning a new one. Each of these functions, called reducer, receive two parameters - the current state and the action to be processed - and should return the new state for the action (or the current one if there are no changes).
Our program has nine reducers, but five of them share the same transfer function, as in Table 4.

Table 4 – Reducers

Name	**Data Type**	**Actions**	**Used for**
device	string	set, remove	current selected device
devices	string[ ]	add, remove	list of devices
signals	string[ ]	set, clear	list of signals
signalsData	object[ ]: name: number[ ]	set, clear	list of signals data (name and values)
object	object: any	set, clear, remove	used for: Plot Page options General options MIDI devices Signal to MIDI connections

Source: made by authors

8.2.4 Note Calculation

To calculate the note a few steps are shown in Figure 18. The amplitude calculation is a simple absolute avareage removing the mean. The pitch within range is a function that limits each string frequency to being close to it's know possible values, as in Table 5.

Table 5 – Pitch Range

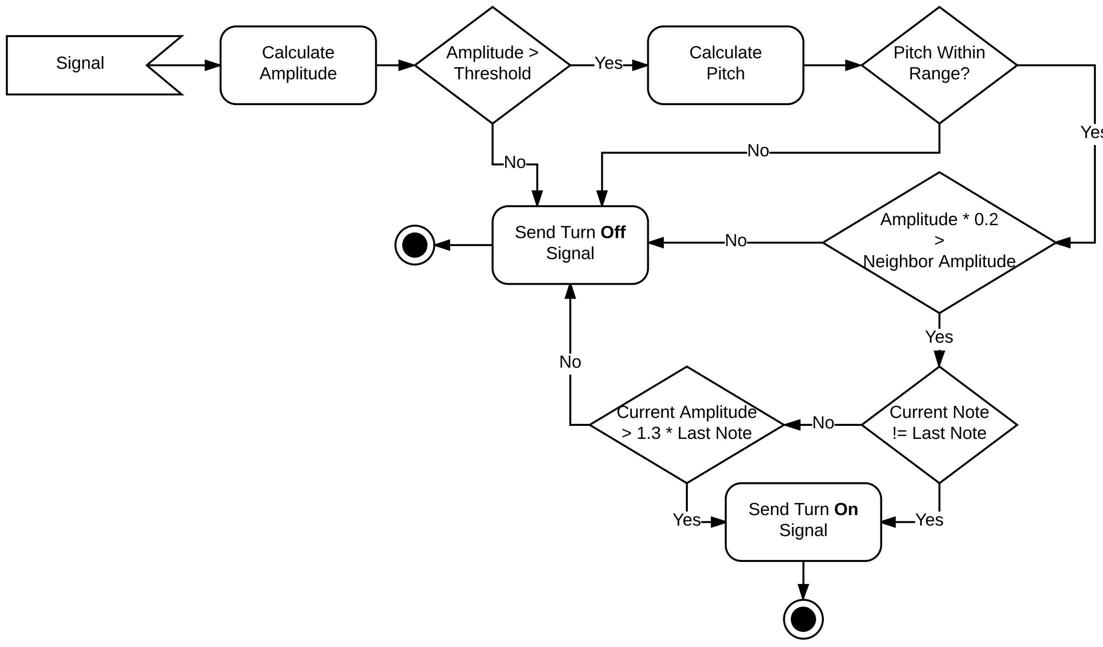
String	**Min. Freq.[Hz]**	**Max. Freq.[Hz]**
E	70	265
A	95	350
D	130	470
G	170	625
B	215	785
e	290	1050

Source: made by authors

8.2.5 Tests

This is for now a prototype of a real world project, so it has a limited amount of tests. The tests fall into one of two categories: unity or timing. Unity tests are made for the signal processing relating modules and also for every reducer, these are all automated and sum up to a total of 31 tests over 9 modules.
The timing tests were used to check if each separated functionality that may cause

Figure 18 – Note Detection Diagram



Source: made by authors

processing issues can run in real-time. There are timing tests for: signal average value calculation, signal window buffering, raw data conversion, pitch detection and USB polling.

8.2.6 Repository

Again, all code is available at GitHub (TREVISAN, 2017b).

Part IV

Results and Dicussions

9 Software Results and Discussions

9.1 Results

TODO: measurements and analysis

9.2 Future Improvements

9.2.1 Note Detection

Real time note detection proved not to be so accurate. Legato (connected) notes may cause a middle note detection and there are misdetections (mostly at frequency transitions). It works well enough for live play of MIDI instruments, but not for music notation.
For real world music notation a different solution is needed: record the signals and post-process them. This way it is possible to use a more detailed analysis of each signal and thus get very good results, as there is much more computing power available when not being limited by real-time processing.

9.2.2 Performance

Though real time analysis work, a few limitations were detected. Buffering can only process each data point one time (can't use a sliding buffer that recycles data). This means that we are close to the processing limits, due to two causes: single core processing and slow algorithms.
One step of the solution is to use multi-core processing for the pitch detection, which can be done using Node.js support for it.
As already stated (section 6.3) the current implementation uses quadratic growth algorithms, when they can be implemented with logarithmic growth. Fixing this will improve the performance to a point where multi-core processing won't be even needed.

Conclusion

Sed consequat tellus et tortor. Ut tempor laoreet quam. Nullam id wisi a libero tristique semper. Nullam nisl massa, rutrum ut, egestas semper, mollis id, leo. Nulla ac massa eu risus blandit mattis. Mauris ut nunc. In hac habitasse platea dictumst. Aliquam eget tortor. Quisque dapibus pede in erat. Nunc enim. In dui nulla, commodo at, consectetuer nec, malesuada nec, elit. Aliquam ornare tellus eu urna. Sed nec metus. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.

Phasellus id magna. Duis malesuada interdum arcu. Integer metus. Morbi pulvinar pellentesque mi. Suspendisse sed est eu magna molestie egestas. Quisque mi lorem, pulvinar eget, egestas quis, luctus at, ante. Proin auctor vehicula purus. Fusce ac nisl aliquam ante hendrerit pellentesque. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Morbi wisi. Etiam arcu mauris, facilisis sed, eleifend non, nonummy ut, pede. Cras ut lacus tempor metus mollis placerat. Vivamus eu tortor vel metus interdum malesuada.

Sed eleifend, eros sit amet faucibus elementum, urna sapien consectetuer mauris, quis egestas leo justo non risus. Morbi non felis ac libero vulputate fringilla. Mauris libero eros, lacinia non, sodales quis, dapibus porttitor, pede. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Morbi dapibus mauris condimentum nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Etiam sit amet erat. Nulla varius. Etiam tincidunt dui vitae turpis. Donec leo. Morbi vulputate convallis est. Integer aliquet. Pellentesque aliquet sodales urna.

Bibliography

3D printer materials. 2013. Description of the materials used on 3D printers. Available from Internet: <http://www.3dprinterhelp.co.uk/what-materials-do-3d-printers-use/>. Quoted in page 17.

AMPLIFIER Circuits. n.d. Simple theory about amplifier circuits. Available from Internet: <http://www.electronics-tutorials.ws/amplifier/amp_1.html>. Quoted in page 19.

CARTER, B. A Single-Supply Op-Amp Circuit Collection. 2000. Available from Internet: <http://electro.uv.es/asignaturas/ea2/archivos/sloa058.pdf>. Accessed: 30 sep. 2017. Quoted 2 times in pages 19 and 23.

CHEVEIGNÉ, A. de; KAWAHARA, H. YIN, a fundamental frequency estimator for speech and music. *The Journal of the Acoustical Society of America*, p. 111, jan. 2002. Available from Internet: <http://audition.ens.fr/adc/pdf/2002_JASA_YIN.pdf>. Accessed: 30 sep. 2017. Quoted in page 35.

COOCOX. n.d. Free/open ARM Cortex-M Development Tool-chain. Available from Internet: <http://www.coocox.org/>. Quoted in page 28.

GETTING Started with React, Redux and Immutable: a Test-Driven Tutorial (Part 2). 2016. Theodo. Available from Internet: <https://www.theodo.fr/blog/2016/03/getting-started-with-react-redux-and-immutable-a-test-driven-tutorial-part-2/>. Accessed: 8 oct. 2017. Quoted in page 34.

HOW does Node.js work with NW.js and Electron? 2017. Medium. Available from Internet: <https://medium.com/@paulbjensen/how-does-node-js-work-with-nw-js-and-electron-da9e50e2c3c1>. Accessed: 8 oct. 2017. Quoted 2 times in pages 32 and 33.

HUDAK, Z. *STM32F103C8T6 board, alias Blue Pill*. n.d. Available from Internet: <https://os.mbed.com/users/hudakz/code/STM32F103C8T6_Hello/>. Quoted in page 26.

INA 326, 327 datasheet. 2004. Supplier datasheet for INA 326 and 327. Available from Internet: <http://www.ti.com/lit/ds/symlink/ina326.pdf>. Quoted 2 times in pages 19 and 20.

LIBUSB. n.d. Cross-platform API for generic USB access. Available from Internet: <http://libusb.info/>. Accessed: 8 oct. 2017. Quoted 2 times in pages 29 and 31.

LOOPBE1. n.d. A Free Virtual MIDI Driver. Available from Internet: <http://www.nerds.de/en/loopbe1.html>. Accessed: 18 oct. 2017. Quoted in page 42.

MAGNETS. n.d. Image of a example of pickup magnets. Available from Internet: <http://tinderwetstudios.com/blog/wp-content/uploads/2015/07/guitar-pickup-magnets.jpg>. Quoted in page 18.

MCLEOD, P.; WYVILL, G. A Smarter Way to Find Pitch. *Proc. International Computer Music Conference*, Barcelona, Spain, p. 138–141, sep. 2005. Available from Internet: <http://miracle.otago.ac.nz/tartini/papers/A_Smarter_Way_to_Find_Pitch.pdf>. Accessed: 30 sep. 2017. Quoted in page 36.

MILMANN, J.; HALKIAS, C. C. *Eletrônica Disositivos e Circuitos*. [S.l.: s.n.], 1981. Volume 2. Quoted in page 19.

MINISCOPE. n.d. A Free Virtual MIDI DriverVery cheap low-speed dual channel PC/USB oscilloscope with STM32 (STM32F103C8T6) microcontroller. Available from Internet: <http://tomeko.net/miniscope_v2c/>. Accessed: 19 oct. 2017. Quoted in page 28.

NAVE, R. *Faraday's Law*. n.d. Theory of the Faraday's Law. Available from Internet: <http://hyperphysics.phy-astr.gsu.edu/hbase/electric/farlaw.html>. Quoted in page 17.

NODE-PITCHFINDER. n.d. A compilation of pitch detection algorithms Node. Available from Internet: <https://github.com/cristovao-trevisan/node-pitchfinder>. Accessed: 8 oct. 2017. Quoted in page 37.

NODE-USB. n.d. Node bindings to libusb. Available from Internet: <https://github.com/tessel/node-usb>. Accessed: 8 oct. 2017. Quoted in page 32.

NODE Webkit. n.d. Available from Internet: <https://nwjs.io/>. Accessed: 8 oct. 2017. Quoted in page 33.

PITCHFINDER. n.d. A compilation of pitch detection algorithms for Javascript. Available from Internet: <https://github.com/peterkhayes/pitchfinder>. Accessed: 8 oct. 2017. Quoted in page 33.

REACT. n.d. A JavaScript library for building user interfaces. Available from Internet: <https://reactjs.org/>. Accessed: 8 oct. 2017. Quoted in page 33.

REACT-PLOTTER. n.d. Real Time (high speed) Plotter Component for React. Available from Internet: <https://github.com/cristovao-trevisan/react-plotter>. Accessed: 16 oct. 2017. Quoted 4 times in pages 38, 39, 40, and 41.

REDUX. n.d. Predictable state container for JavaScript apps. Available from Internet: <http://redux.js.org/>. Accessed: 8 oct. 2017. Quoted 2 times in pages 33 and 43.

SEMANTIC UI React. n.d. The official Semantic-UI-React integration. Available from Internet: <https://react.semantic-ui.com>. Accessed: 9 oct. 2017. Quoted in page 33.

SIX, J.; CORNELIS, O.; LEMAN, M. TarsosDSP, a Real-Time Audio Processing Framework in Java. In: *Proceedings of the 53rd AES Conference (AES 53rd)*. [S.l.: s.n.], 2014. Quoted 3 times in pages 31, 33, and 36.

STM32F103XX Reference Manual. [S.l.], 2015. Available from Internet: <www.st.com/resource/en/reference_manual/cd00171190.pdf>. Accessed: 19 oct. 2017. Quoted in page 26.

TLV x316 Operational Amplifier. 2016. Supplier datasheet for TLV x316 family. Available from Internet: <http://www.ti.com/lit/ds/symlink/tlv316.pdf>. Quoted in page 23.

TREVISAN, C. *Guitar Digitizer Firmware Codebase*. 2017. Available from Internet: <https://github.com/cristovao-trevisan/guitar-digitizer-firmware>. Quoted in page 29.

TREVISAN, C. *Guitar Digitizer GUI Codebase*. 2017. Available from Internet: <https://github.com/cristovao-trevisan/guitar-digitizer>. Quoted in page 45.

WALLACE, H. *How do guitar pickups work*. 2004. Theory behind the pickup working. Available from Internet: <http://zerocapcable.com/?page_id=219>. Quoted in page 17.

WIKIPEDIA. *Harmonic series (music)*. 2017. Available from Internet: <https://en.wikipedia.org/wiki/Harmonic_series_(music)>. Accessed: 9 oct. 2017. Quoted in page 36.

Appendix

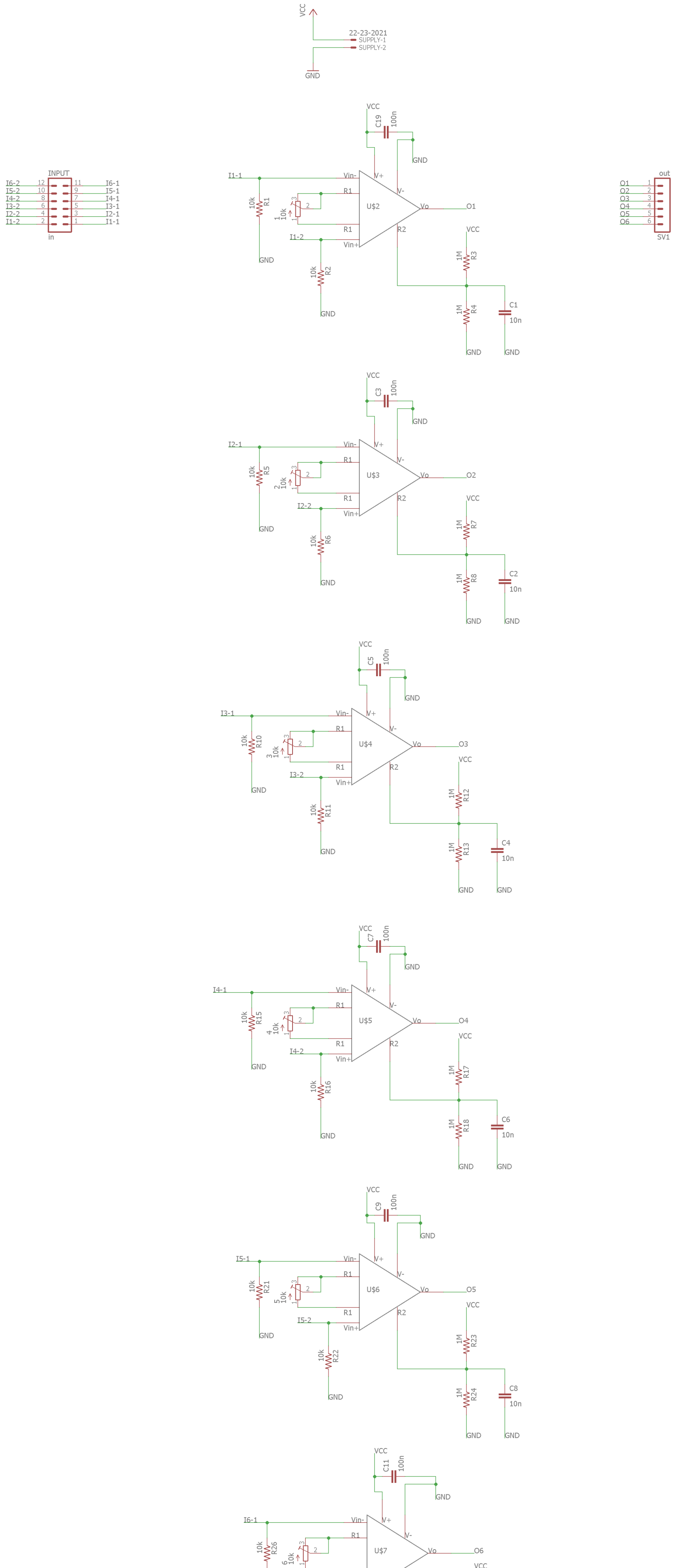
APPENDIX A – Quisque libero justo

Quisque facilisis auctor sapien. Pellentesque gravida hendrerit lectus. Mauris rutrum sodales sapien. Fusce hendrerit sem vel lorem. Integer pellentesque massa vel augue. Integer elit tortor, feugiat quis, sagittis et, ornare non, lacus. Vestibulum posuere pellentesque eros. Quisque venenatis ipsum dictum nulla. Aliquam quis quam non metus eleifend interdum. Nam eget sapien ac mauris malesuada adipiscing. Etiam eleifend neque sed quam. Nulla facilisi. Proin a ligula. Sed id dui eu nibh egestas tincidunt. Suspendisse arcu.

Annex

ANNEX A – INA 326 complete Schematic.

Figure 19 – INA 326 Complete Schematic


VCC
22-23-2021
SUPPLY-1
SUPPLY-2
GND
INPUT
in
out
SV1
U$2
U$3
U$4
U$5
U$6
U$7
Vin-
Vin+
V+
V-
Vo
R1
R2
O1
O2
O3
O4
O5
O6
I1-1
I1-2
I2-1
I2-2
I3-1
I3-2
I4-1
I4-2
I5-1
I5-2
I6-1
10k
1M
100n
10n