UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

# ELECTRIC GUITAR PICKUP CARACTERIZATION AND MODELLING

A Degree's Thesis

Submitted to the Faculty of the

Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona

Universitat Politècnica de Catalunya

by

Javier Vallés Tomàs

In partial fulfilment

of the requirements for the degree in

TELECOMMUNICATIONS ELECTRONIC SYSTEMS ENGINEERING

Advisor: Miguel Garcia

Barcelona, June 2014

## Abstract

The purpose of this project is make it easier for guitarists to modify any electronic component of their instrument, as well as to help them decide if any of the changes worth the investment of money and time.

The sound of an electric guitar or bass guitar is determined by two factors: the instrument body and its electronics. This project is focused on the electronics, aiming to analyze and recreate them through a software platform without any physical change needed. For this purpose, a circuit has been designed as an Arduino shield, based on using the guitar’s pickup as a RLC resonant in an oscillator circuit to obtain the resonance frequency of the system and therefore the pickup parameters with the Arduino microcontroller.

In addition, a MATLAB application has been designed as a prototype in order to be the reference for the next project’s web application. It can record guitar audio clips, modify them using the graphic user interface and listen the result of the modifications. Every configuration and recording can be saved and loaded, in order to be shared by the users when the application is online and open-software.

## Resum

El propòsit d’aquest projecte és simplificar als guitarristes qualsevol modificació de components electrònics del seu instrument, així com també ajudar-los a decidir si els hi és rentable la inversió de temps i diners.

El so d’una guitarra o baix elèctric està determinat per dos factors: un és l’instrument i l’altre els components elèctrics. Aquest projecte està centrat en l’electrònica per tal d’analitzar-la i reproduir-la mitjançant programari sense la necessitat de fer canvis físics. Per això, s’ha dissenyat un circuit en forma de “shield” d’Arduino que es basa en utilitzar la pastilla de ressonant RLC en un circuit oscil·lador per calcular la freqüència de ressonància i posteriorment els paràmetres de la pastilla amb el microcontrolador Arduino.

A més, s’ha dissenyat una aplicació en MATLAB com a prototip a partir del qual dissenyar l’aplicació web amb la qual es pot grabar clips d’audio de la guitarra i, modificant-ne el circuit a la pantalla d’usuari, escoltar-ne la diferència. Totes les configuracions i grabacions es poden guardar i carregar, pensant en que els usuaris puguin compartir-les quan l’aplicació sigui online i oberta.

## Resumen

El propósito de este proyecto es simplificar a los guitarristas la modificación de cualquier componente electrónico de su instrumento, así como ayudarles a decidir si merece la pena la inversión de tiempo y dinero.

El sonido de una guitarra o bajo eléctrico viene dado por dos factores: el instrumento y los componentes electrónicos. Este proyecto se centra en la electrónica para analizarla y poder reproducirla mediante software sin necesidad de hacer cambios físicos. Para ello, se ha diseñado un circuito en forma de "shield" para Arduino que se basa en utilizar la pastilla de resonante RLC en un circuito oscilador para calcular la frecuencia de resonancia y a partir de ahí los parámetros de la pastilla con el microcontrolador Arduino.

Además, se ha diseñado una aplicación en MATLAB como prototipo a partir del cual diseñar la aplicación web con la que se puede grabar clips de audio de guitarra y, modificando el circuito en la interfaz de usuario, escuchar la diferencia de sonido. Todas las configuraciones y grabaciones se pueden guardar y cargar, pensado para que los usuarios puedan compartirlas cuando la aplicación sea online y abierta.

*To Laura and all my friends for all the support away from home. To my parents for the restless support.*

## Acknowledgements

I would like to thank Miguel García in the first place for all the hours he has spent to help me with this project, not to mention all what I have learnt thanks to him.

Special thanks also to Eva Vidal for introducing me to this project and Jose M. Miguel for recommending me such an important book for this task as Helmut Lemme's 'Electric Guitar'.

Lots of thanks to Rodrigo Snider for helping me with the PCB design. Thanks to Vicente Ruiz also for having printed the PCBs in such a small time lapse.

## Revision history and approval record

| Revision | Date | Purpose |
|---|---|---|
| 0 | 04/01/2015 | Document creation |
| 1 | 26/01/2015 | Document revision |
| | | |
| | | |
| | | |

DOCUMENT DISTRIBUTION LIST

| Name | e-mail |
|---|---|
| Javier Vallés Tomàs | javiervalles92@gmail.com |
| Miguel J. Garcia Hernández | miguel.j.garcia@upc.edu |
| | |
| | |
| | |
| | |

| | | | |
|---|---|---|---|
| Written by: | | Reviewed and approved by: | |
| Date | 25/01/2015 | Date | 30/01/2015 |
| Name | Javier Vallés Tomàs | Name | Miguel J. García Hernández |
| Position | Project Author | Position | Project Supervisor |

# Table of contents

# List of Figures

## List of Tables:

# 1. Introduction

## 1.1. Project Background

The original idea for this project comes from Marc Alier (Director - Institut Ciències de l'Educació – UPC) when he starts building his own guitars and bass guitars. The market offers a wide range of pickups, but also a wider range of prices, which go a couple of euros to hundreds of euros, and this variation, awakens the idea: “Why some pickups are so more expensive than others?”

The construction of magnetic pickups can be different from one to another, but share the same principle, and the materials are mostly the same. So, is it all marketing that sets the price difference, or there is something more?

Starting from this point, the author thinks of not only an online benchmarking of pickups, but also an application which allows guitarists to share their entire guitar circuit configurations and simulation of the sound differences when changing any parameter.

Thus, the Arduino pickup analyzer and the MATLAB simulator are conceived.

## 1.2. Statement of Purpose

The purpose of this project is designing and implementing an analyzer for electric guitar’s pickups. In addition, a software-based simulator is intended to recreate the sound filter of the analyzed pickups, using recordings of the user’s guitar (accomplishing the aim of letting know the user how his guitar would sound if he changed his guitar’s pickups without having to change them).

Finally, the project is intended to let users around the world share their guitar configurations and the sound attached to them, as well as their pickup’s parameters to simulate them inside any other guitar. This goal is reached in the continuation of this project (done by a FIB student) which takes the MATLAB application and turns it into an open software web application that allows the connection between all users.

Therefore, this project main goals are:

1- Design of an Arduino-based pickup analyzer
2- Implementation of an Arduino shield easy to reproduce, as a DIY kit.
3- Design of a MATLAB Software-based filter simulator of the electric circuit in a guitar.
4- Design of a Reverse filter simulator – get the original signal before the circuit filter.
5- Create an application to save the information and allow to share it between guitarists (this goal is fully reached in the FIB project which is a continuation of this one).

## 1.3. Requirements and specifications

**Project requirements:**

- Low price circuitry in order to make the product attractive.
- Offer the tools to elaborate an exhaustive pickup benchmarking.
- Offer the ability to listen a guitar with other pickups in a realistic way.
- Offer a program usable for everyone regardless his knowledge in electronics.

**Project specifications:**

Pickup Analyzer:

- Precision up to an error of 5% measuring every pickup parameter (inductance, capacitance, series resistance, parallel resistance and signal amplitude).
- As an oscillator, the resonance frequency of a pickup may be between 2kHz and 10kHz. The maximum error allowed to achieve the parameters' precision should be of 5% as well.

Filter simulation:

- Similarity between a real guitar sound and the simulated one (human perception). Therefore, the filtering processes, forward and reverse, need certain accuracy, although it is subjective.
- Accuracy calculating the transfer functions, using circuit analysis formulas.

## 1.4. Work Plan

The Work Plan of this project is consists of two main blocks: the Arduino Analyzer (Analyzer) and the MATLAB application (called Filter and Reverse Filter in here, due its main function):

**1. Analyzer Design**
1.1 Design of the circuit
1.2 Simulation using PSPICE
**2. Analyzer implementation**
2.1 Breadboard construction and test
2.2 Arduino firmware and integration
**3. Filter design & Implementation**
3.1 Analysis of the circuit
3.2 MATLAB simulation
3.3 Software modeling
**4. Reverse Filter design & Implementation**
4.1 Application design & Implementation

All work packages are described in the tables below:

*Table I.1.* *Work Package #1*

| | | |
|---|---|---|
| Project: Analyzer design | WP ref: 1 | |
| Major constituent: Design & Simulation | Sheet 1 of 4 | |
| Short description:<br>Design and simulation using PSPICE of the main circuit of the analyzer, the pickup-based oscillator. Design of the other peripheral circuitry to measure the other parameters (series resistance and signal amplitude). | Planned start date: 01/07/14<br>Planned end date: 25/07/14<br><br>Start event: Final Product Meeting.<br>End event: Simulation working correctly. | |
| Internal task T1: Design of the circuit<br><br>Internal task T2: Simulation using PSPICE | Deliverables:<br>Schematics &<br>Simulation files | Dates:<br>25/07/14 |

*Table I.2.* *Work Package #2*

| | | |
|---|---|---|
| Project: Analyzer implementation | WP ref: 2 | |
| Major constituent: Hardware prototype & Software | Sheet 2 of 4 | |
| Short description:<br>Implementation of the main circuit (oscillator) and test it on breadboard and PCB. Arduino firmware implementation and integration to the whole circuit | Planned start date: 29/09/14<br>Planned end date: 28/11/14<br><br>Start event: Simulation working correctly.<br>End event: PCB board connected to Arduino working correctly. | |
| Internal task T1: Breadboard construction and test<br><br>Internal task T2: Arduino firmware and integration<br><br>Internal task T3: PCB design, construction and test | Deliverables:<br>Circuit boards<br>Arduino files | Dates:<br>28/11/14<br>31/10/14 |

***Table I.3.*** *Work Package #3*

| | | |
|---|---|---|
| Project: Filter design & Implementation | WP ref: 3 | |
| Major constituent: Design & Analysis, Software | Sheet 3 of 4 | |
| Short description:<br>Analysis of the filter generated out of a guitar's circuit. Simulation using MATLAB of the transfer function and evaluation of the simplification. Software modeling of the filter using a second order low-pass simplification. | Planned start date: 15/09/14<br>Planned end date: 22/12/14 | |
| | Start event: Analysis of the circuit in a guitar<br>End event: Software modeling of the circuit filter | |
| Internal task T1: Analysis of the circuit | Deliverables: | Dates: |
| Internal task T2: MATLAB simulation | MATLAB output files. | 06/10/14 |
| Internal task T3: Software modeling | MATLAB output files | 22/12/14 |

***Table I.4.*** *Work Package #4*

| | | |
|---|---|---|
| Project: Reverse Filter design & Implementation | WP ref: 4 | |
| Major constituent: Design & Analysis, Software | Sheet 4 of 4 | |
| Short description:<br>Finding the inverse filter of the second order simplification of the incoming signal and save the original guitar signal. Designing the Graphic User Interface in MATLAB used for filtering signals. | Planned start date: 01/12/14<br>Planned end date: 22/12/14 | |
| | Start event: Analysis of the filter<br>End event: Software modeling of the filter | |
| Internal task T1: Filter design & Implementation | Deliverables: | Dates: |
| | MATLAB output files | 22/12/14 |

## Milestones

***Table I.5.*** *Milestones table*

| WP# | Task# | Short title | Milestone / deliverable | Date (week) |
|---|---|---|---|---|
| 1 | 0 | Final Product Meeting | | 30/06/14 |
| 1 | 2 | PSPICE simulation | Schematics &Simulation files | 25/07/14 |
| 2 | 2 | Arduino firmware | Arduino files | 31/10/14 |
| 2 | 3 | PCB design | PCB boards | 28/11/14 |
| 3 | 2 | MATLAB simulation | MATLAB output files | 06/10/14 |
| 3 | 3 | Software modeling | MATLAB output files | 22/12/14 |
| 4 | 1 | Filter modeling | MATLAB output files | 22/12/14 |

Fortunately, every milestone could be reached in schedule, leaving the last month of the project term dedicated to the writing of this document.

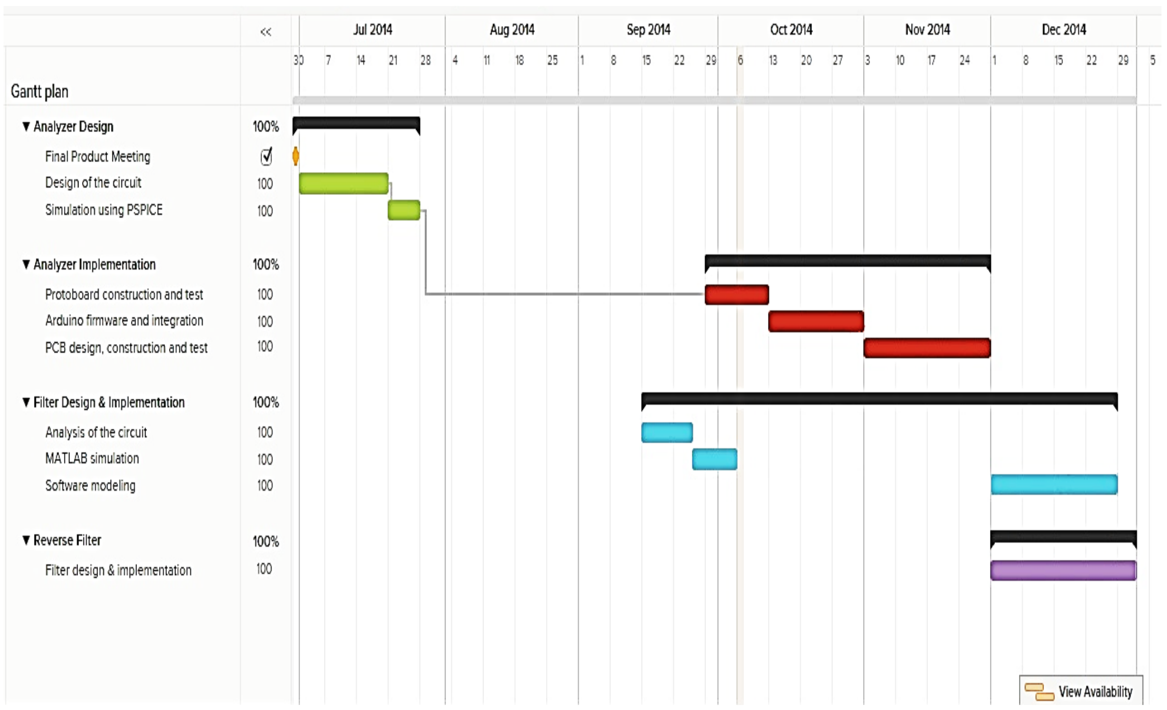


***Figure I.1.*** *Gantt Diagram*

# 2. State of the art of the technology used or applied in this thesis:

## 2.1. The Guitar Sound

The most important question before starting this project was: What determines a guitar sound?

The sound of an electric guitar or bass is a chain of several subsystems where everyone is important and could be the ruin of the whole tone if it failed. Mostly the blocks are:

- Guitar body
- Pickup and electronics
- Amplifier
- PA, Public Address system (i.e. sound system and room acoustics)

These are the main subsystems of a guitar tone and they are all important. However, the PA system is not a part of the instrument and, although the amplifier is very important and should be considered as a part of the instrument (a part of this project's next steps and improvements is focused on including the amplifier as a variable in the application), this project is focused in the guitar itself.

As it has been stated, the instrument comprises two main subsystems: the body and the electronics.

## 2.2. Guitar Body

Every string instrument depends heavily on its construction. Obviously, the strings are the most important part of the instrument. Electric guitars have plane wire strings for the three thinner ones and wounded strings with additional wire for the thicker. Bass guitars even have double wounded strings, since they are much thicker than guitar strings. The thickness of a string affects its sound being more powerful and full of tones if the string is thicker. However it may be harder to play and it becomes a trade-off for the musician.

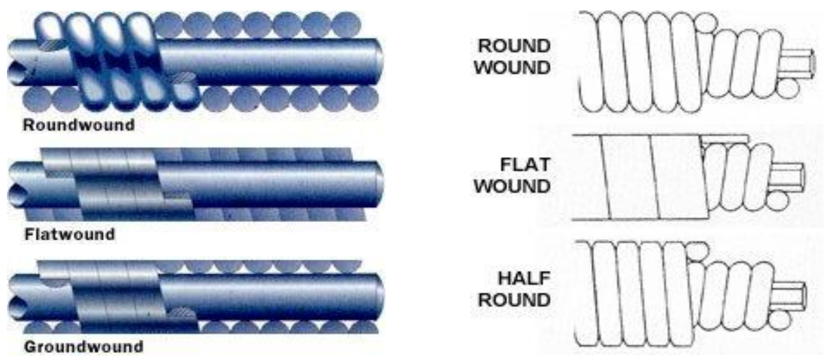


***Figure II.1.*** *Different wound types for Guitar and Bass Guitar strings*

Nevertheless, any string by itself is pointless, since what really gives the instrument its tone is the body. The prime material, wood, is what lets the string resonate easier or harder in any of its frequency of the spectrum, so it is a very important choice before construction. The problem is that every block of wood is unique, because every tree is unique. Its growing characteristics determine the wood's properties, and so does its treatment after the cutting.

The weight of the guitar, the type of the body (hollow, semi-solid, solid), the number of wood pieces used and their junction affect to the tone, there are infinite variables and that's why this project does not focus on the body physics. The guitar is considered a part of the user which will not be changed, but its electronics will.

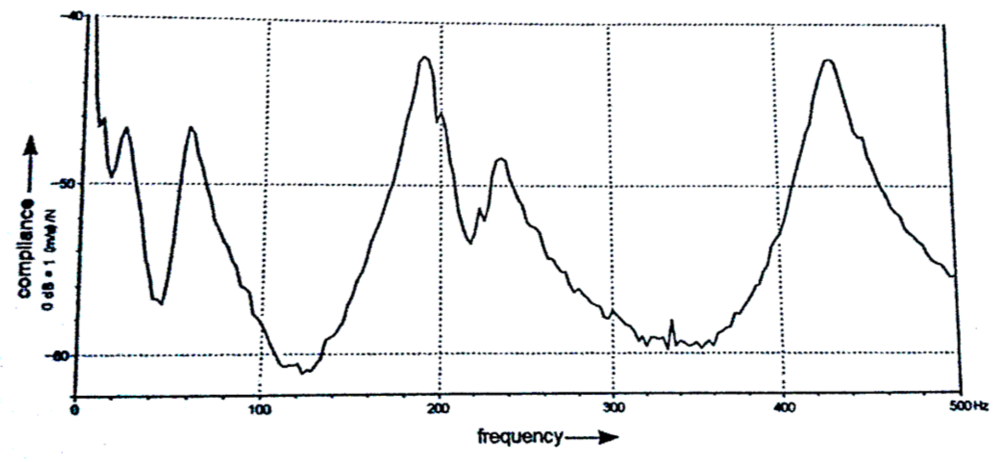


***Figure II.2.*** *Mechanical vibration spectrum of a Fender 'Stratocaster'. Peaks are the frequencies where the body is easily stimulated to vibrate (Electric Guitar, Helmuth Lemme)*

## 2.3. <u>Guitar Pickup</u>

The most important part of the electronics of a guitar is the magnetic pickup. It is like the microphone of a singer for a guitar, and this comparison is very good one since it voices the guitar, and mostly because it doesn't have a particular sound by itself. A 'good' pickup will not enhance the sound of a 'bad' guitar, while a 'bad' pickup could sound pretty acceptable if the guitar is a first class. The only difference between microphones is that sometimes while speaking of guitars, some nonlinearities are appreciated and even sought in a guitar's sound, and that's why a pickup may be studied not only looking for maximum linearity (which is ironically quite poor for a guitar's sound).

First the magnetic pickup is going to be explained. A pickup is made with any magnet material and a coil of some thousands turns of thin wire wrapping it. Its operation is based in two electromagnetic principles. The idea is to create a magnetic field which is going to be interfered by the strings. Every oscillation of a string is going to create a magnetic variation in the field with the same frequency harmonics as the mechanical vibration. Then, since the magnetic field crosses the coil center, an electric voltage is generated with the same frequency harmonics as the magnetic field, thus becoming the mechanical vibration into an electric signal.

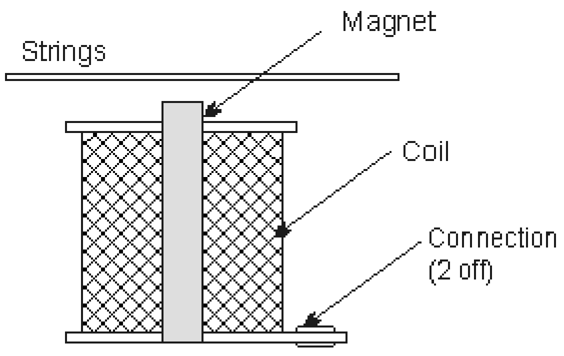


***Figure II.3.*** *String above an electric pickup, side sight.*

Since the magnetic material can be of any type, there are lots of construction types, which are showed in Figure II.4.

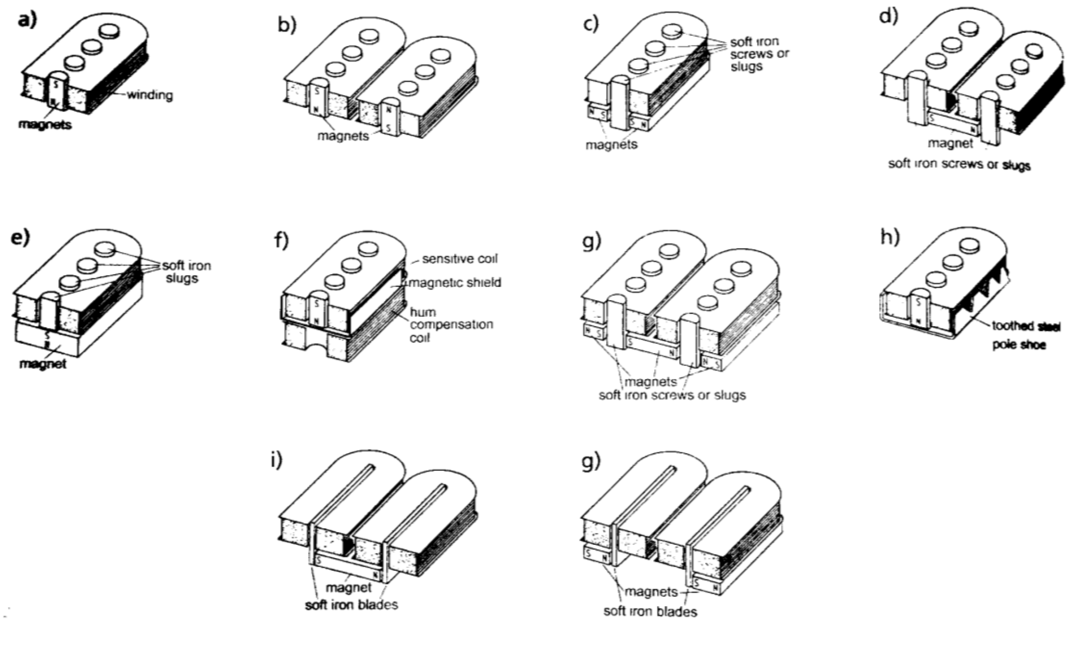


***Figure II.4.*** *List of pickup construction types (Electric Guitar, Helmuth Lemme).*

As can be seen, there are some pickups with two coils, whose inner magnets are polarized inverted (one N and the other S). These are called 'humbucker' pickups and were invented to cancel the hum that electronic devices generated and the pickup captured. The idea is to cancel the hum inverting the coil polarity, but adding the signal thanks to inverting also the magnetic field (double negative equals positive). They have some specific issues which have been considered as future development, mainly the physical differences of having two separated pickups (see Figure II.5).

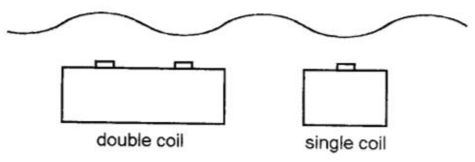


***Figure II.5.*** *A humbucker coil would not pick up overtones half a wavelength of the magnetic pole distance, while single coil would (Electric Guitar, Helmuth Lemme).*

A single coil pickup can be modelled like a second order RLC low-pass filter (Figure II.6). This circuit element has a very interesting property which is the superelevation in the resonance frequency. The ratio between the maximum value of the superelevation and the low frequencies value is called quality factor, Q factor or only Q. This behavior and the frequency where it appears is what determines the difference of sound between a pickup and another.

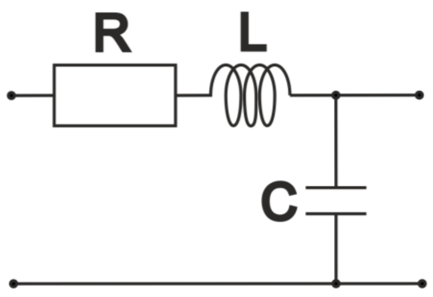


***Figure II.6.*** *Ideal pickup without consideration of the eddy current losses.*

If a string vibrates over the pickup, an AC voltage source appears in the model, and this is the first approximation model to the pickup (see Figure II.7).

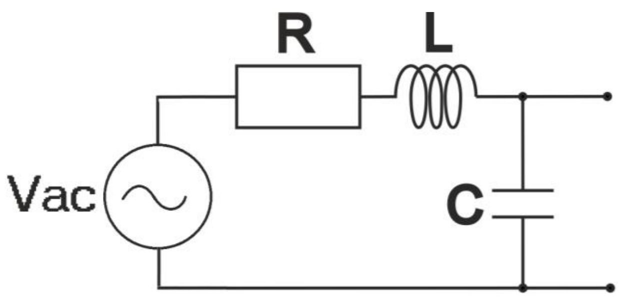


***Figure II.7.*** *Same equivalent as before with the AC voltage source.*

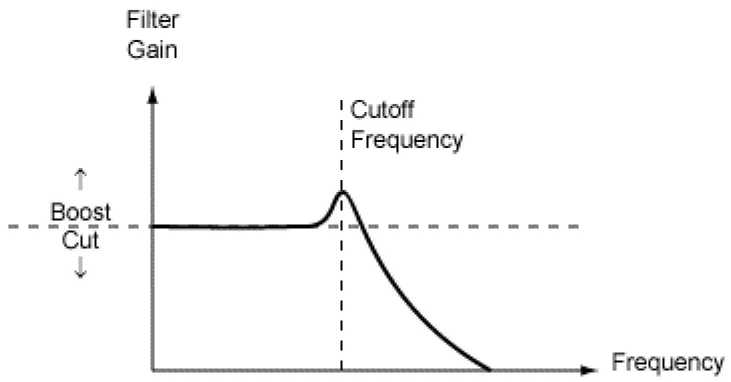


***Figure II.8.*** *Frequency response of a second order low pass filter.*

The reason of the pickup having a parallel capacitor is that between every turn there is a capacitance effect. Since there are thousands of turns, the inductance of the coil has a large value, between 2 and 10H (and larger if it is a humbucker pickup with two coils in series), and the capacitor, even it has a quite small value in the order of a few hundreds of pF, its effects appear in the human's ear audible frequency range, giving the pickup this transfer characteristic to be considered.

However, the real behavior of the pickup has nothing to do with the ideal one. Using an impedance analyzer, every tested pickup had its Q far below the ideal calculation. Measuring its inductance and capacitance, calculating the ideal value and comparing it to the measured value, real Q's were in the order of 1-2 while ideal Q's were between 10 and 20 or even higher. These results showed that eddy current losses in magnetic cores were a high concern for the pickup model and it had to be taken into account. That's why the author added to the models found in previous bibliography a parallel resistance. The complete model of a pickup can be seen in Figure II.9.

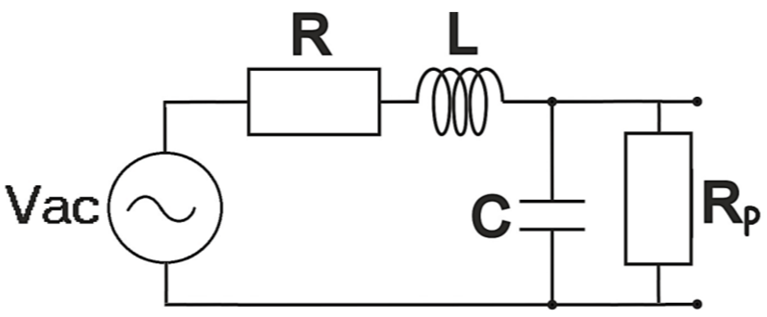


***Figure II.9.*** *Final pickup equivalent used in this project.*

The difference between ideal and real pickups can be seen in Figure II.10 and Figure II.11 with a test pickup simulated in PSPICE. Inductance, capacitance, series and parallel resistance values were measured using the Agilent 4294A Precision Impedance Analyzer.

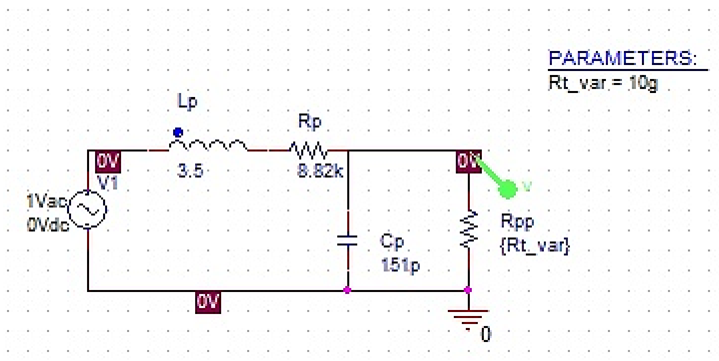


***Figure II.10.*** *PSPICE Schematic used for simulation of ideal and real pickup. For ideal simulation Rpp value is very high, can be considered infinite.*

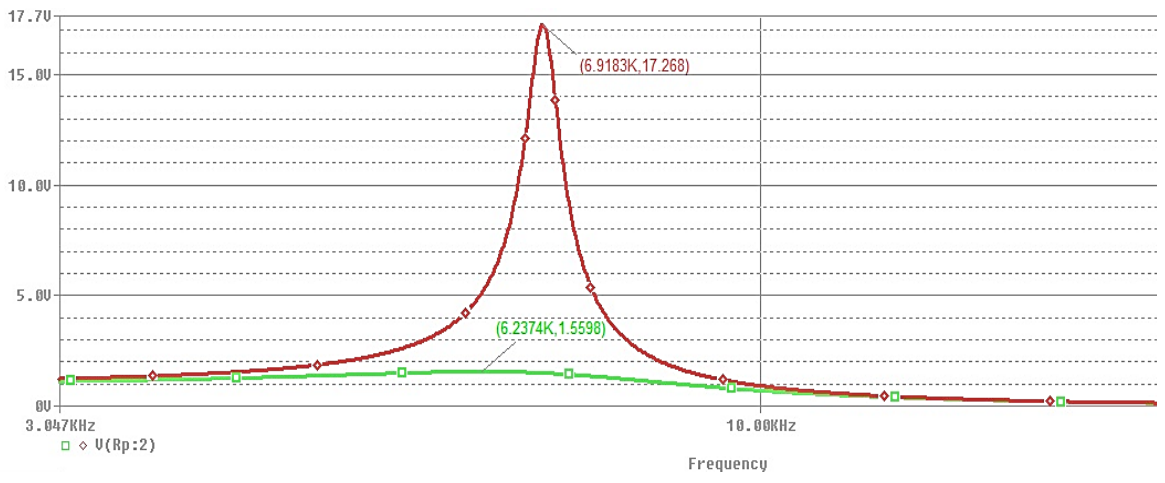


***Figure II.11.*** *Ideal Q (red) is 17.27, much higher than 1.56 real Q (green).*

## 2.4. <u>Guitar Electronics</u>

To complete an electric guitar circuit, there are usually two more simple elements, a tone control and a volume control. The volume control is a potentiometer used as a voltage divider. The tone control is a potentiometer connected to a capacitor, making a variable resistance first order low pass filter.

There are a lot of different circuit configurations for electric guitars, since the number of pickups and potentiometers can change. A guitar may have one, two, three or even more pickups while volume and tone controls may be shared by two or three pickups while others may be specific for a single pickup. However, once selected the configuration by the user, the circuit usually is a pickup connected to a tone control and a volume control, as shown in Figure II.12.

This circuit is the one has been used to recreate the guitar filter in software, which mathematical development can be seen in Annex 1.

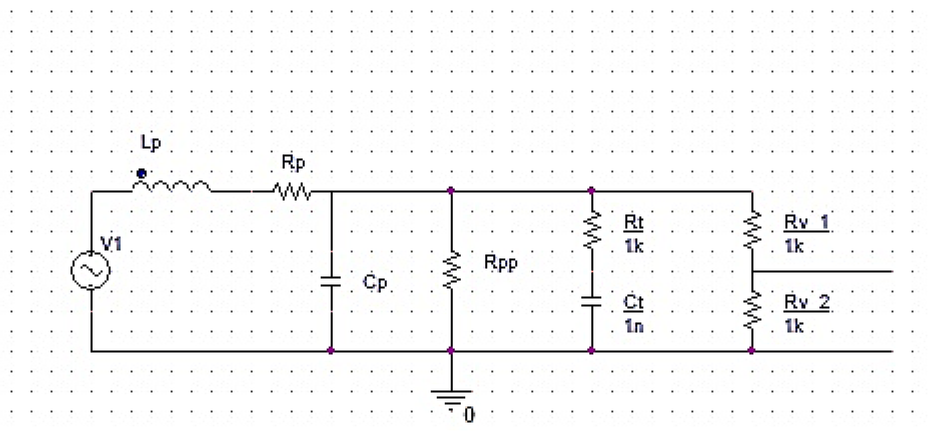


***Figure II.12.*** *Electric Guitar most common circuit.*

## 2.5. Pickup Controversy

One of the reasons this project was thought out was the wide range of prices of pickups the market offers. Manufacturers rarely give any technical information of their product, one because most of guitarists wouldn't understand this information and therefore it would be a useless task to characterize any pickup, and two, because if they did, prices should have to go a lot cheaper because almost perfect imitations would appear quickly.

Let's not forget that, even pickups really change a guitar's sound and give it a characteristic tone, they are only a bunch of turns of wire and some magnets. Then, how come could a pickup cost as much as hundreds of euros while others (made in the Far East mostly) made with the same materials cost a couple of euros?

The reason is simple. Since there are no knowledge among the buyers of the principles acting when using a pickup and there is not enough information and research among producers due to what has been stated before, everything ends up being a marketing merciless war between high quality (and prices) producers. There are lies such as a guitar will sound like any other, mostly one celebrity-owned guitar only by changing the pickup, even if they are not even the same model.

A guitar body is as important as a pickup, in the whole guitar tone, and so is the amplifier used and its tonal configuration. Sometimes changing a pickup, if one does not know what he is looking for and does it only for advertisement reasons, results in a waste of money.

The purpose of the project is not discrediting any manufacturer company but lending the consumer a tool to help him find what he is looking for the sound of his guitar, also avoiding a waste of time and money if he is not sure about what to change.

## 2.6. Bounds of the Project

As has been explained before, any pickup model is just an approximation to the real life component, and there are some issues not considered with enough accuracy.

One is the difference between single coils and humbucker coils. There are some issues such as the volume difference and the usually lower resonance frequency that are taken into account, but the main tonal differences are still to be studied and are part of future development.

Another main issue is the magnet type and construction. We can assure that the magnet strength is related to the volume and that it may not affect to the tone but there has not been any study of the tone relationship with the magnet type and shape, so it has to be stated and it could be a following step for the project.

There are nonlinear effects such as magnetic distortion due to deflection of the string, if a string is too close to the magnets and therefore the distance of the strings to the pickup, but if it is not extremely close, distance is taken into account with the volume measure. However there is no correction in the measures, so it is recommended to set the distance between the pickup and the string at 4mm (standard common value).

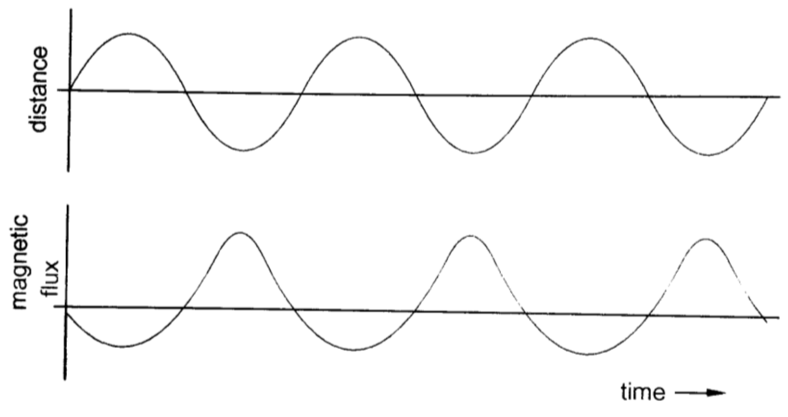


***Figure II.13.*** *Nonlinear distortion: string deflection and magnetic flux over time (Electric Guitar, Helmth Lemme).*

# 3. Methodology / project development:

## 3.1. Pickup Characterization Solution

The main idea was to determine with accuracy all five parameters of the pickup model that had been proposed, namely:

- Inductance
- Capacitance
- Series Resistance
- Parallel Resistance
- Volume

Anyway, first step was determining the resonance frequency parameters, i.e. inductance and capacitance values.

There were three first proposals, which had some pros and cons based on difficulty of implementation, price, microcontroller limitations, energy supply conditions, etc. but in the end one remained as the best overall solution.

### 3.1.1. VCO and Frequency Sweep

First idea was using a Voltage Controlled Oscillator (VCO) controlled by the microcontroller and measuring the pickup impedance to a certain number of frequencies to search the maximum value where the resonance is and the frequency where it is.

The IC selected for this task was not properly a VCO but a DDS (Direct Digital Synthesizer) wave generator. However, it was impossible to find one encapsulated in a DIP package in order to be easy to solder in a DIY PCB. It has to be stated though, that if this project had any chance of becoming a commercial product with the enough production resources, the author thinks this solution would be the optimal.

Making a VCO using OPAMPs and DIP ICs was too difficult and the circuit was way too large to fit in an Arduino shield (as an example, see Figure III.1).

These drawbacks added to the poor optimization of the software functionality led to think other solutions before choosing it for good.

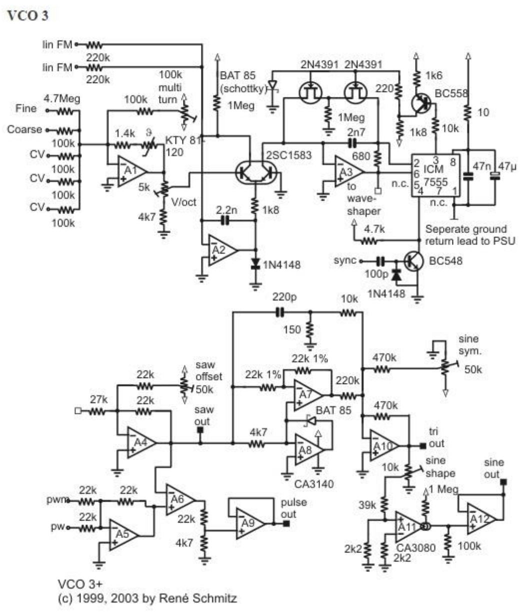


***Figure III.1.*** *VCO schematic designed with typical components.*

### 3.1.2. Time Domain Analysis

In order to get a pickup's parameters, a frequency domain solution had been already proposed, not suiting very well. So a time domain analysis was posed.

The possibility of making this proposal relied on the great difference of order of magnitude between the two main values (inductance and capacitance). While the inductance value was in the order of Henries, the capacitance value was in the order of pF ($10^{-12}$ Faradays).

This gap was useful for example to measure each parameter's behavior in a step response analysis (see Figure III.2). The low value capacitor had a low value tau with a test resistance, in the order of few microseconds, while the inductance had a higher tau in the order of hundreds of microseconds. Capacitor's tau was too low to measure it directly with the Arduino microcontroller and it could not be changed, since it depended on the pickup series resistance in parallel with the test resistance.

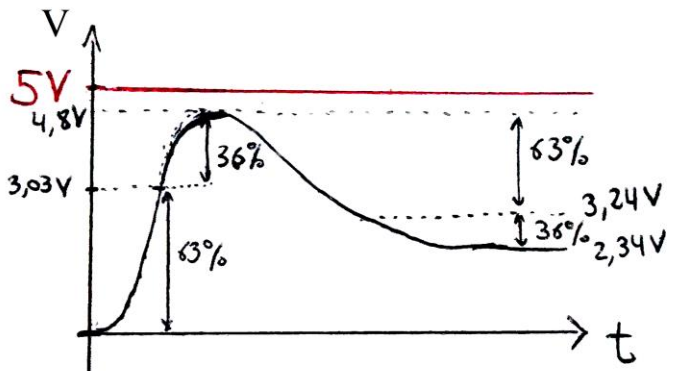


***Figure III.2.*** *Test pickup step response. The tau is the time passed until reaching 63% of the maximum value.*

As can be seen in Figure III.2, the maximum value is never reached because the inductance effect interferes slightly with the capacitance effect and this could lead to measuring errors.

These results were not very promising, and another problem was detected when calculating the values from the measured results. Pickups have a nonlinearity when they are measured as a passive load which does not show when are used as an AC voltage generator (capturing the string deflection in the magnetic field).

The inductance value at low frequencies is larger than the theoretical value that determines the resonance frequency (see Figure III.3). Thus the filter would be simulated incorrectly. These drawbacks required mathematical corrections too accurate and led to rejecting the proposal.

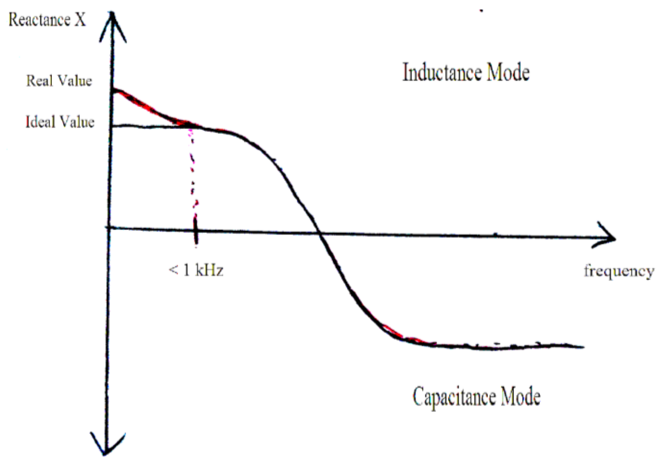


***Figure III.3.*** *Ideal and real pickup reactance comparison. Ideal calculated from resonance frequency.*

But how come did this issue not affect to the transfer characteristic? The answer is simple: The impedance of an inductor at those low frequencies (even a large inductor like the one present in a pickup) has a low value, since impedance for inductors is $jL\omega$, being $\omega = 2\pi f$. An impedance which compared to the 250kΩ potentiometers, or even to the pickup series resistance with a value of a couple of kΩ, is not significant. Therefore the variation between real and theoretical inductance value is not significant either.

### 3.1.3. Oscillator and Frequency counter

The last proposal made is also the one developed. The idea is to use the pickup as a feedback network in the classic schematic (see Figure III.4) of an oscillator and an OPAMP as the gain amplifier. Also, there has to be a gain regulation control to maintain the oscillation without distortion. Thus the oscillation frequency is the one that determines the pickup. Using the Arduino microcontroller as a frequency counter, we can calculate connecting and disconnecting some test capacitors the original inductance and capacitance value of the pickup, as well as the parallel resistance value with some phase shifting formulas.

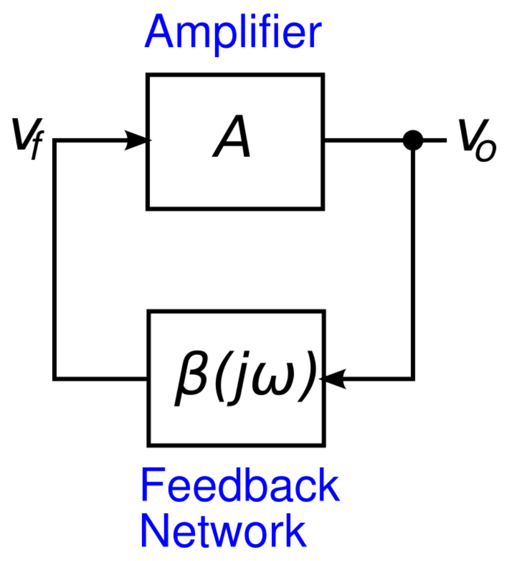


***Figure III.4.*** *Electronic oscillator basic scheme. The principles are unitary gain and no phase shifting in the whole loop.*

It was the chosen proposal to determine a pickup's parameters because it features cheap discrete electronic components easy to solder by anyone and it fits really well in an Arduino shield. Furthermore the Arduino features a comparator with which it is very easy to implement a frequency counter. It can also be used with a slight modification to measure also series resistance. Only extra circuitry needed is for the pickup's volume measure, which will be explained later.

The following chapter is dedicated to explaining how the oscillator is designed and the problems and solutions to them encountered.

## 3.2. Oscillator Design

The principal characteristics of this oscillator are (see Figure III.5):

- The guitar circuitry as a low Q band-pass filter
- A very rare and simple loop gain control called “diode throttle”
- Test capacitors to change the oscillation frequency controlled by BJT transistors with the Arduino GPIO pins
- OPAMP and circuit supply limited to 0-5V single supply from the Arduino board

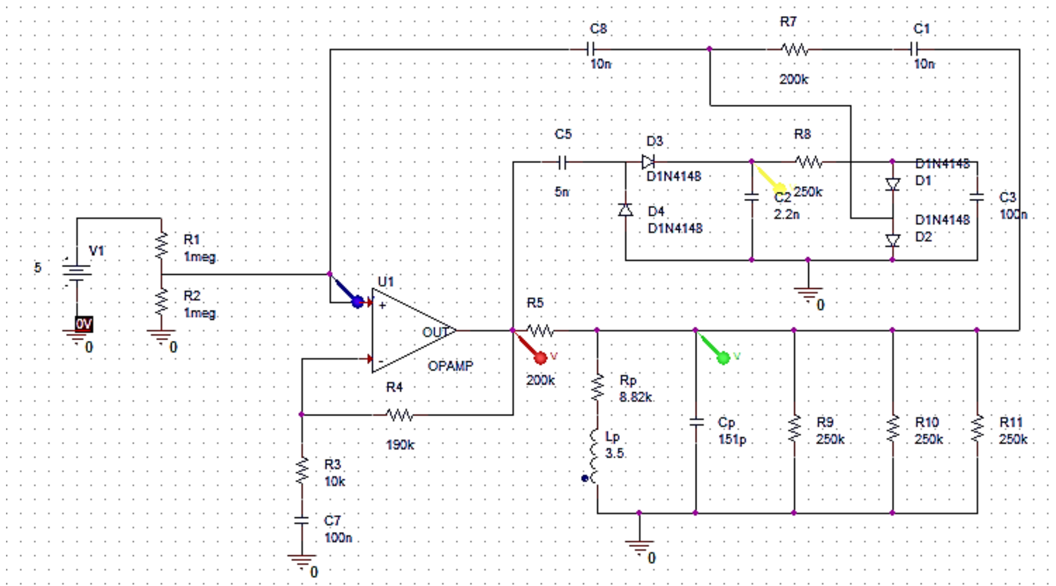


***Figure III.5.*** *Complete oscillator schematic.*

The program used for this design is OrCAD Capture CIS Lite, a free lite version of the program with PSPICE lite simulator. In this section the oscillator main features are going to be explained in detail.

### 3.2.1. Guitar Circuitry as a low Q band-pass filter

As it has been stated before, the idea is to use the guitar’s pickup as a resonant in an oscillator loop. The problem is that it cannot be separated from the volume and tone controls, since one of the most important requirements is for the user to characterize a pickup without having to disassemble anything. So the potentiometers of each control knob will appear in the loop and they will dampen the quality factor of the bad-pass filter as any parallel resistance would. However, the user is asked to set the potentiometers at their maximum value (and then introduce this value in the application) in order to have the best possible Q factor. In Figure III.6 an example made with a test pickup can be seen.

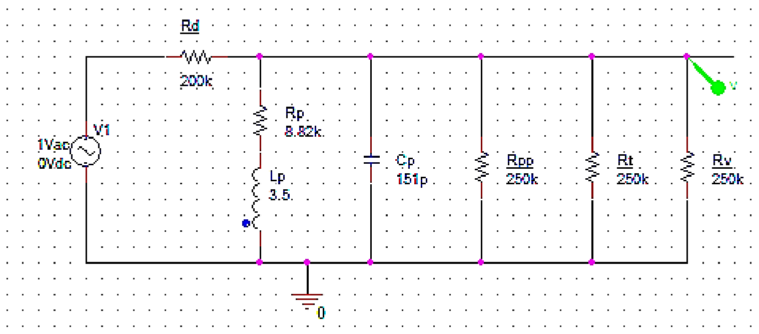


***Figure III.6.*** *Guitar circuit as passive load. $R_d$ is part of the oscillator circuit, while $R_t$ and $R_v$ are the guitar potentiometers.*

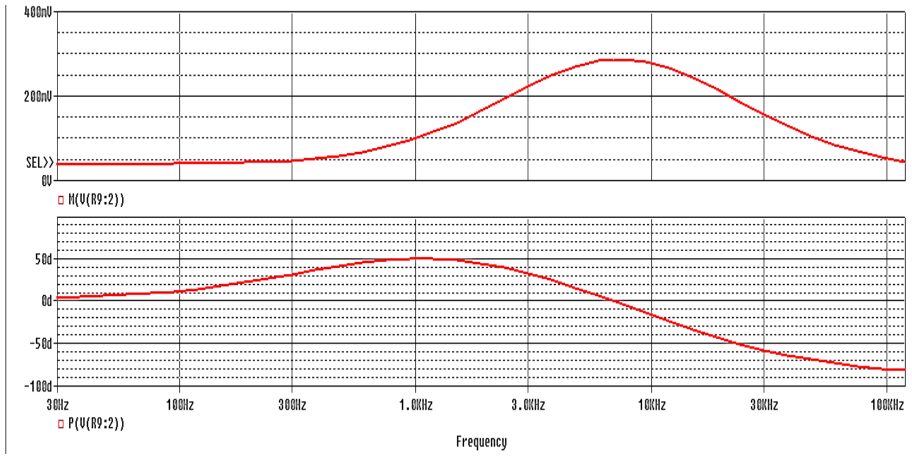


***Figure III.7.*** *Bode representation of the circuit. The low Q is noticed in the phase slope, making a little change in the phase loop a high difference in the frequency measured.*

### 3.2.2. Gain Control

To obtain and maintain the oscillation in the loop, there has to be:

- Amplification to generate the oscillation out of the white noise generated by the OPAMP.
- A gain control to avoid the circuit from saturating the oscillation.

The amplification stage was there, since ideal OPAMPs model equation for output voltage is $v_o = A(v_+ - v_-)$ where A is a very large gain (even $10^6$). Real OPAMPs equation depends on the frequency, decreasing $20\,{}^{dB}/_{dec}$ usually. This behavior is shown in what is called gain-bandwidth product. This parameter tells at what frequency the amplifier has unitary gain and also maximum gain in DC (see Figure III.8). When the loop is closed, gain remains constant for all frequencies if the open loop gain-bandwidth product is higher than the closed loop product. It is nearly impossible to stabilize a circuit with exactly unitary gain. Therefore, a control loop is needed or it will saturate easily.

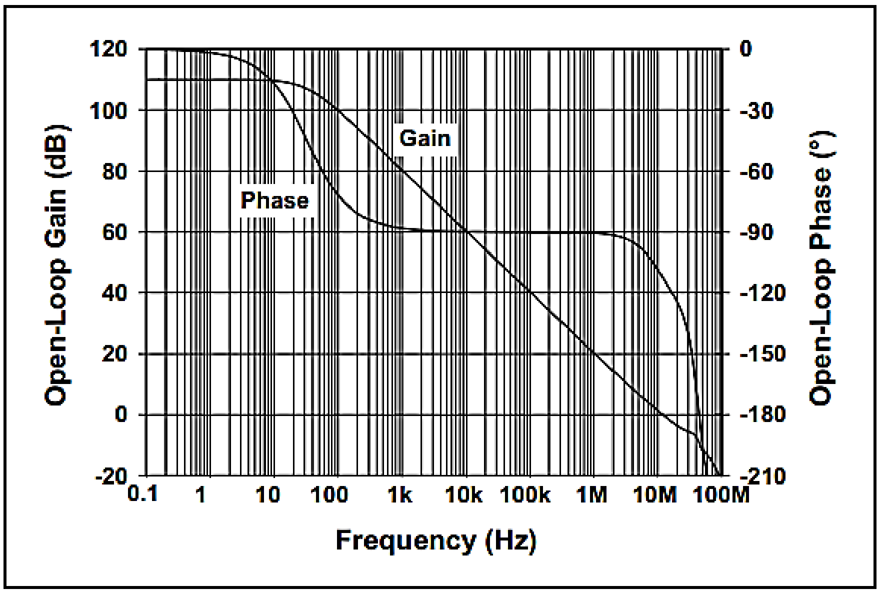


***Figure III.8.*** *Gain-Bandwidth product and phase versus frequency of the amplifier used in this oscillator.*

The amplifier circuit is shown in Figure III.9. It has a non-inverting gain of 20 ($V_-$ closed loop) and a voltage divider of a third more or less –The one made out of $R_{div}$ and $R_{eq}$ (pickup equivalent resistance) in parallel with $R_7$ ($R_{diodes}$ can be neglected)–, so the global gain depends on the voltage divider of $R_7$ and $R_{diodes}$. When the attenuation is 20, the loop has unitary gain.

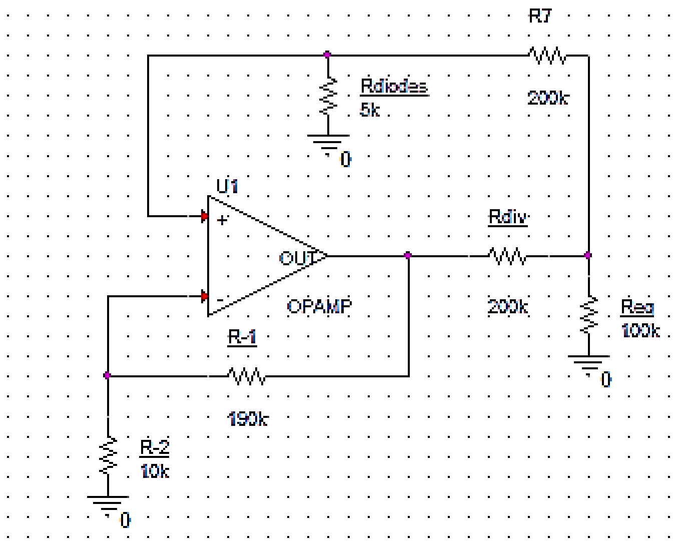


***Figure III.9.*** *Differential amplifier used as oscillator.*

The amplitude depends on the control resistance ($R_{diodes}$). When $V_{OUT}$ is getting higher, it has to make $R_{diodes}$ lower in order to make the attenuation of the voltage divider higher and compensate. This is accomplished using the circuit in Figure III.10.

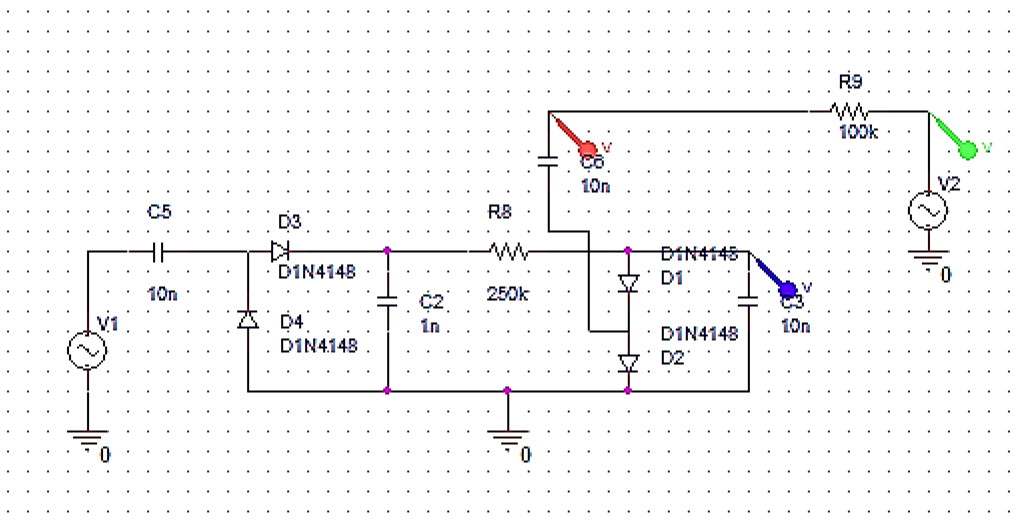


***Figure III.10.*** *Gain control 'Diode Throttle'. $V_1$ is OPAMP's $V_{OUT}$ while $V_2$ is voltage after the divider.*

The main idea is to use the diodes as a resistance. The I-V characteristic of the diode is shown in Figure III.11. Depending on the biased current ($I_{bias}$), its resistance (the inverse of the slope of the current versus voltage) changes.

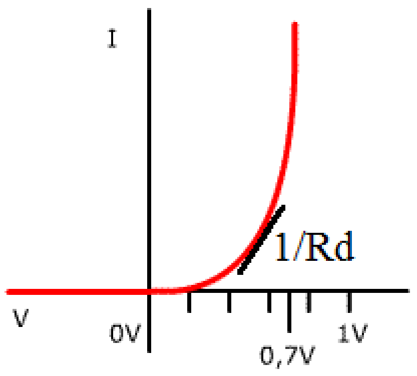


***Figure III.11.*** *The slope of the characteristic is the inverse of the resistance.*

$D_3$, $D_4$, and $C_2$ are used to rectify the sinusoidal wave and double its voltage value. Then, $I_{bias}$ depends on the voltage and $R_8$ value ($I = {}^{V}\!/_{R}$). The value of $R_d$ is $Rd = {}^{Vt}\!/_{I}$, where $V_t$ is the thermic voltage of a diode, usually 26mV. To simulate the lineal behavior of a resistance, two diodes in antiparallel configuration are used, making $Rdiodes = {}^{Rd}\!/_{2}$ (as two equal resistances in parallel configuration). $R_8$ value is calculated to have $V_{OUT}$ value between 1.5V –$R_{diodes}$ value is 32kΩ and therefore A > 1– and 2V –$R_{diodes}$ value is 5.4kΩ and therefore A < 1–, enough for the Arduino frequency counter.

Values of the capacitors used in this circuit, especially $C_5$, $C_2$, and $C_3$, they affect to the settling time of the oscillation. Their final value can be seen in Figure III.5 and Figure III.13. The bigger they are, the best filtering is accomplished, but since a quick settling time is needed to make the oscillator work properly, the value is set using the lowest possible without distortion.

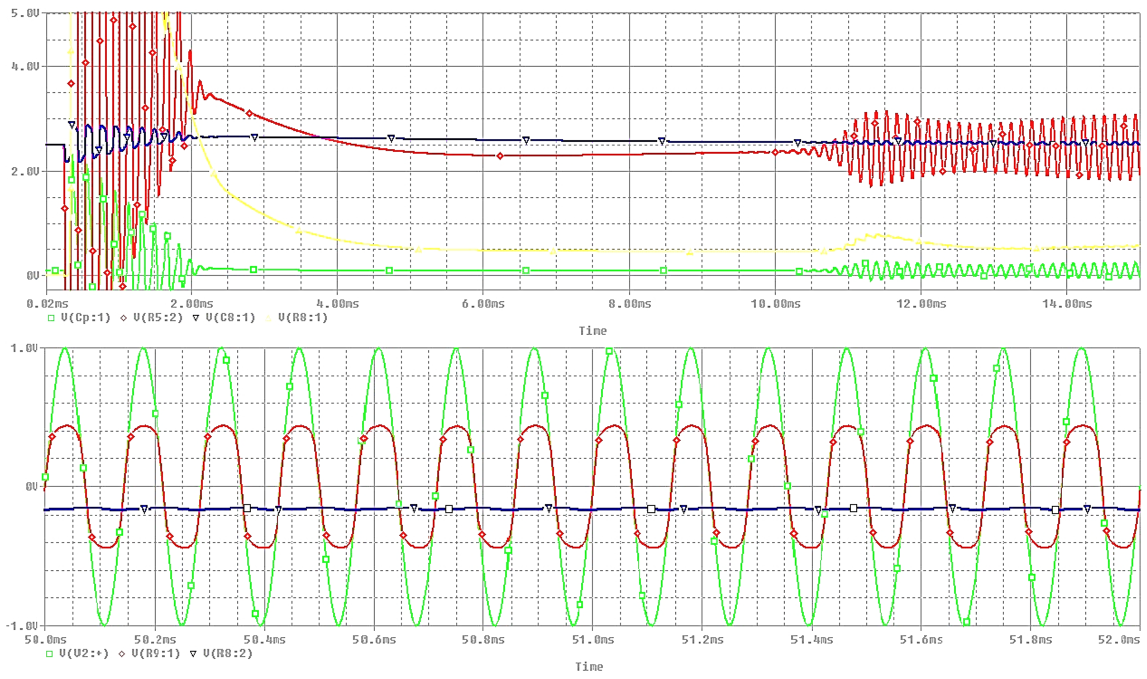


***Figure III.12.*** *Minimized settling time of the oscillator circuit (above) with optimum capacitor values without distortion corresponding to Figure III.5 schematic and distortion of the signal (below) due to undersized value capacitors corresponding to Figure III.10 schematic.*

### 3.2.3. OPAMP specifications

Choosing the best OPAMP for this project is important. It has to be 5V single supplied, its gain-bandwidth product the higher possible to prevent phase delay in the loop (which would change the frequency of the signal), its package must be DIP and its price must be very affordable. The best choice is Microchip MCP6291, with a GB product of 10MHz, enough for this application. However, to make a single supply amplifier work with sinusoidal waves, a bias voltage must be set, in this case at half $V_{DD}$, i.e. 2.5V.

### 3.2.4. Test Capacitors and Phase Shifting

Taking in to account that three values have to be obtained from this circuit –Inductance, capacitance and parallel resistance–, three equations are needed at least. To obtain inductance and capacitance value, a test capacitor in parallel with the pickup is needed. To switch it on and off, it is connected to a transistor used as a controlled gate by the microcontroller.

The way to calculate resistance in parallel is a little more complex, but it is explained widely in Annex 2. The idea is to introduce a previously calculated phase delay. Using another switching capacitor and splitting $R_7$ –Its value is 200kΩ and it is split in two 100kΩ resistors–, the phase delay in the loop is determined by the equivalent resistance connected to the capacitor and the capacitor values.

The transistors used to switch the capacitors have an unexpected importance. First idea was using MOS technology, but they present several problems such interferences in the frequency signal due to external electromagnetic fields and high and non-measurable values of capacitance in OFF mode. This led to use BJT technology because they are not interfered by electromagnetic fields and also have a maximum value of capacitance in OFF mode of 6pF.

Finally, A low pass filter is connected to the output pin of the OPAMP in order to use this signal (without the DC component) to compare it to GND in the Arduino comparator used as frequency counter. The final oscillator schematic can be seen in Figure III.13.

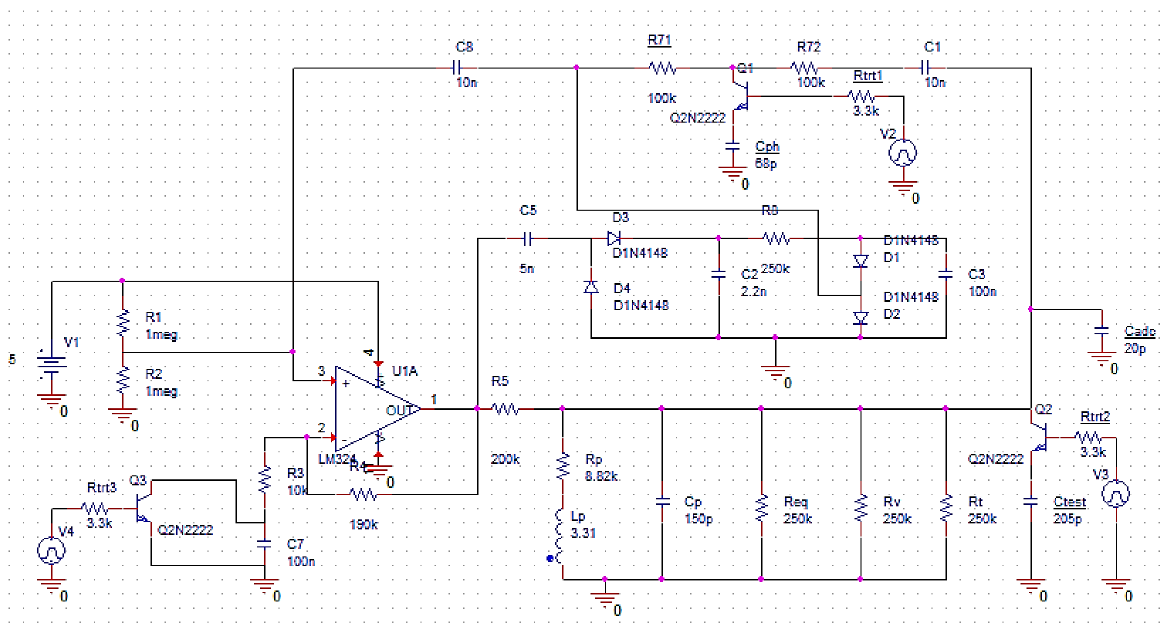


*Figure III.13. Oscillator schematic including test capacitors and series resistance measuring modification. The square signal generators represent Arduino GPIO pins.*

## 3.3. Complete Circuit Design

### 3.3.1. Series resistance measure

As it has been said, with the oscillator circuit the system can measure inductance, capacitance and parallel resistance. For the volume measure a whole new circuit is needed, but for the series resistance a single modification to the oscillator circuit is enough.

In Figure III.13 there is a transistor connected in parallel to the capacitor which makes the OPAMP amplify for AC current and breaks the amplifying configuration and turns the OPAMP into a voltage follower for DC. This capacitor is used in order not to amplify bias voltage $V_{bias}$, which would disable the circuit, making $V_{OUT}$ saturate and being $V_{DD}$ (5V) always.

However, for measuring series resistance it is alright to have a constant DC current flowing through a voltage divider –$R_5$ and the guitar's equivalent load resistance shape the divider–.

Thus, measuring where the pickup connects to the circuit with the Arduino ADC, the series resistance of the circuit can be obtained with a simple equation (see Annex 2).

Connecting the ADC to the resonant filter affects the frequency value, as many other devices would. Fortunately the manufacturer includes the parallel capacitance value of the ADC module in the product's datasheet, namely 20pF, so it only has to be taken into account when calculating the resultant frequency.

### 3.3.2. Volume Measuring Circuit

Most difficult part of measuring the pickup's volume is that the only elements to excite it are the strings. But they have a decreasing exponential behavior when strummed, and their voltage amplitude depends highly on the strength with which the player has strummed and the position where they have been strummed.

Therefore, a standard strumming method is intended to be performed so any player will have the most equal volume measure, and the difference of value will come mostly from the different pickups. The idea is to strum the third string (G in standard tuning) in the middle of it, i.e. in fret number 12. The strumming way is to lower it to touch next string (B in standard tuning) and then release it. This leads to an overshot oscillation, which decays as seen in Figure III.14.

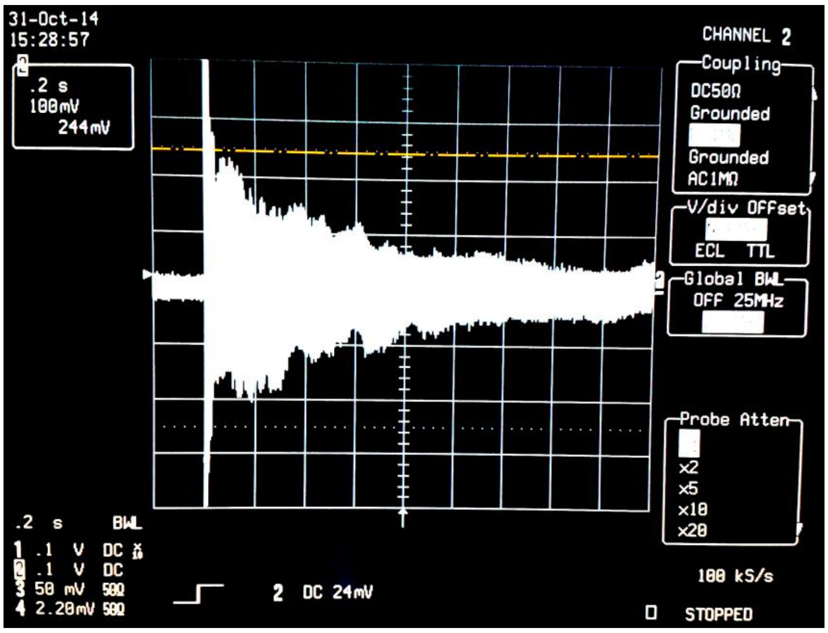


***Figure III.14.*** *Voltage captured from a pickup after strumming a string.*

The Arduino best performance is using A/D converters, but they only work for positive voltages. The circuit fixes this problem in two stages. It rectifies the sinusoidal wave in one stage and filters the rectified waveform in the other, amplifying the signal in both stages and leaving the amplified mean DC component. The rectification is done using a rail-to-rail

OPAMP using its non-linearity, not letting any negative signal appear in $V_{OUT}$. The filter is a simple low-pass filter with a very low cut-off frequency in order to only let the DC component of the signal pass (see Figure III.15).

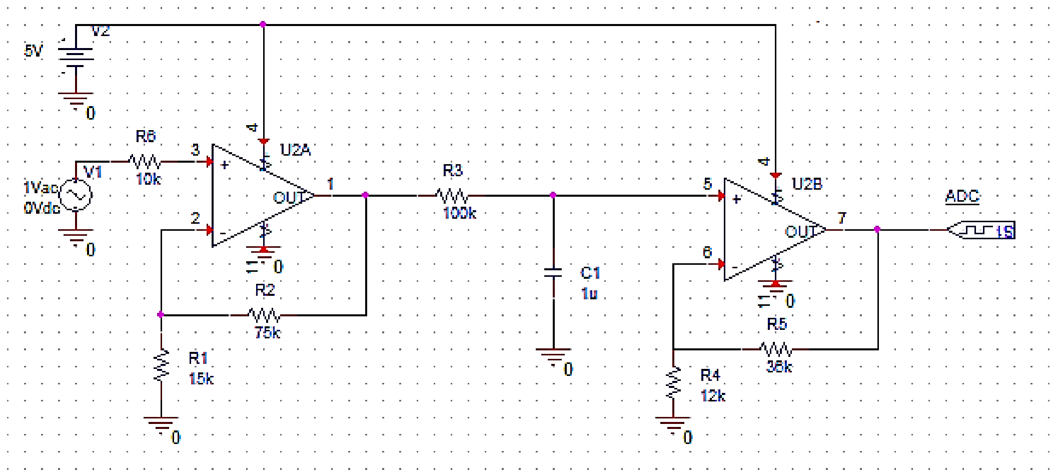


***Figure III.15.*** *Volume measuring circuit. Stage one rectifies and stage two amplifies. The low-pass filter is in the middle of both OPAMPs.*

OPAMP used for this application has to accomplish being rail-to-rail and having two units inside one package. These specifications are not very restrictive so TLC272 is used because it is available in the Sensors and Systems laboratory.

To switch from the oscillator to this circuit, a switch with two switching lines has been needed. One line is for the pickup to change circuit and the other one is connected to $V_{DD}$, GND and a GPIO pin of the Arduino, in order for the microcontroller to know if the pickup is connected to one circuit or another.

It is very difficult to make the same test for every pickup, so the results are distinguishable only when there is a huge difference between two pickups. Also human ear is logarithmically sensitive, so it doesn't matter when the error is a little percentage of amplitude due to variances of the strumming.

## 3.4. PCB Design

The different circuits of this project have been tested on a breadboard. When everything worked properly on breadboard, the PCB was designed.

Program used for the design of the board is P-CAD 2006. Its license is not free and other open software programs such as KiCad could have been used, but the author had already this program and some useful libraries edited by student partners and enlarged by the author himself.

First step to designing the PCB is configuring the circuit schematic with the proper elements. Operational amplifiers, transistors, switches were not defined in any library so they had to be defined by the author. Other components already were and they only had to be placed in the schematic.

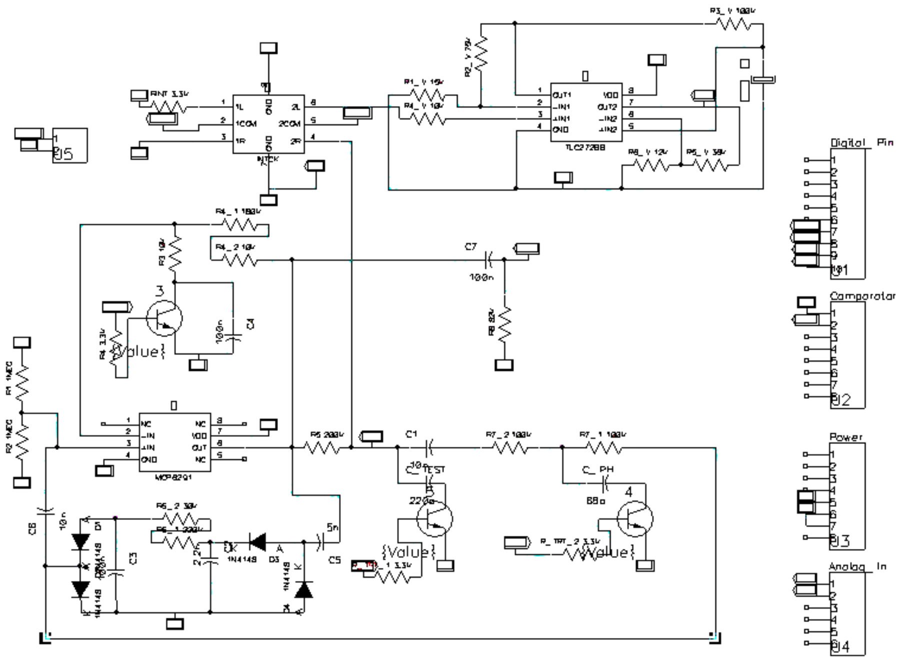


***Figure III.16.*** *PCB schematic design.*

Better schematics can be found in Annex 3. From this schematic a PCB layout is generated. Then all lines have to be drawn for every layer, connecting every component as the schematic determines. The result is shown in Figure III.17.

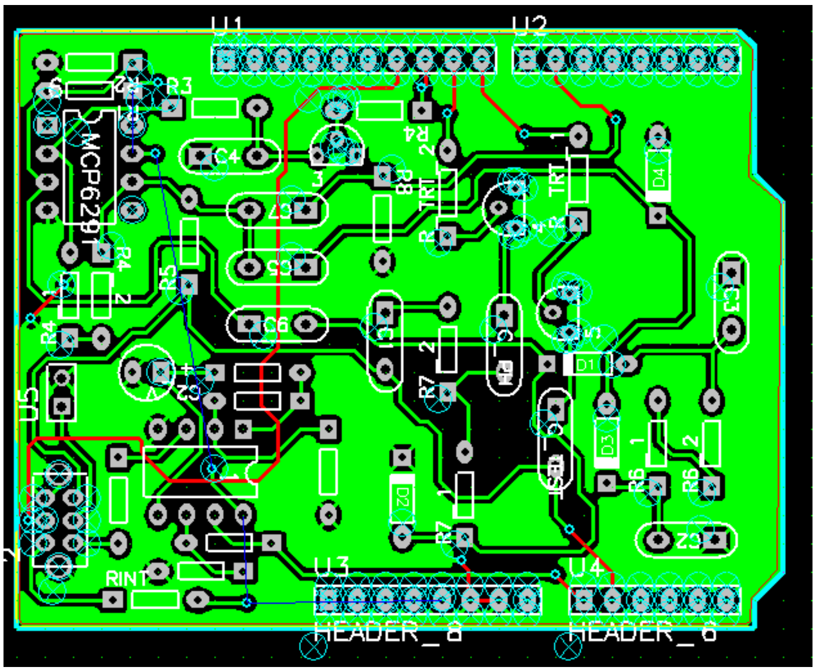


***Figure III.17.*** *Bottom layer of the PCB. The shape of the board is the same as the Arduino Uno.*

Finally, the electronics laboratory technician in the UPC Campus Nord is in charge of manufacturing the board. The final board can be seen in Figure III.18.

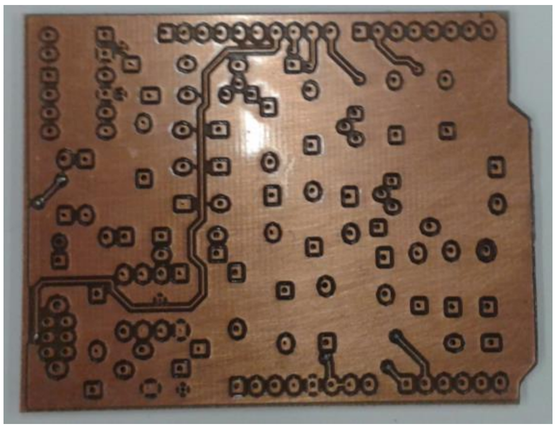


***Figure III.18.*** *Top layer of the real PCB.*

In Annex 3 all layers of both real and schematic boards can be seen in detail.

## 3.5. Arduino Firmware

The program run by the Arduino Uno microcontroller is widely explained in this section. The complete code can be found in Annex 4.

### 3.5.1. Frequency Counter

Modules used for the frequency counter are:

- Timer1, and its interruption
- Comparator module
- GPIO pins

The timer counts 1 second when the main loop sets a flag on. When the timer is counting, the main loop is always checking a register which its value is '1' when the comparator output is HIGH and '0' when it is LOW. Since the sample frequency is 1 second, the number of steps is already the frequency measured. This is done with every combination possible of switching both test capacitors (the resonance frequency change and the phase delay) ON and OFF. Thus, 4 frequencies are obtained. Using the formulas in Annex 2, the parameters are obtained. A "math.h" library is required for the mathematical operations.

### 3.5.2. Series Resistance

For the measure of the series resistance modules needed are:

- ADC module
- GPIO pins

Using an OUPUT pin to saturate the oscillator's OPAMP, the ADC measures a voltage from the voltage divider generated with the 200k resistance of the circuit and the guitar's circuit. Knowing the volume and tone controls resistance values, pickup's series resistance can be easily obtained.

### 3.5.3. Volume

Modules used for this measure are:

- ADC module
- GPIO pins

Taking into account the way of strumming the string, there is a delay of one second before acquiring data once the input signal is over a threshold. Then a sample is obtained every 10ms. The problem of accuracy is that variables and vectors are stored in the microcontroller's RAM memory, which is quite small. A vectors maximum value is of 200 positions more or less. That's why the signal is rectified first. In this case, the vector is 100 positions of length, making the acquiring span of one second long.

When the pickup is not connected to the circuit, the amplifier saturates. When the switching occurs, the high value capacitor is charged for a while, and could interfere with the measuring, the program could think the string is being strummed. That is why there is a 3 seconds delay after connecting the pickup to the circuit.

### 3.5.4. Communication with MATLAB GUI

For the communication with the computer, a USB serial communication is used, the same used with Arduino's compiler.

The protocol used consist in a dialog between the computer and the microcontroller. It can be seen in Figure III.19. The computer sends the potentiometers' resistance values for the microcontroller to use them in the formulas.

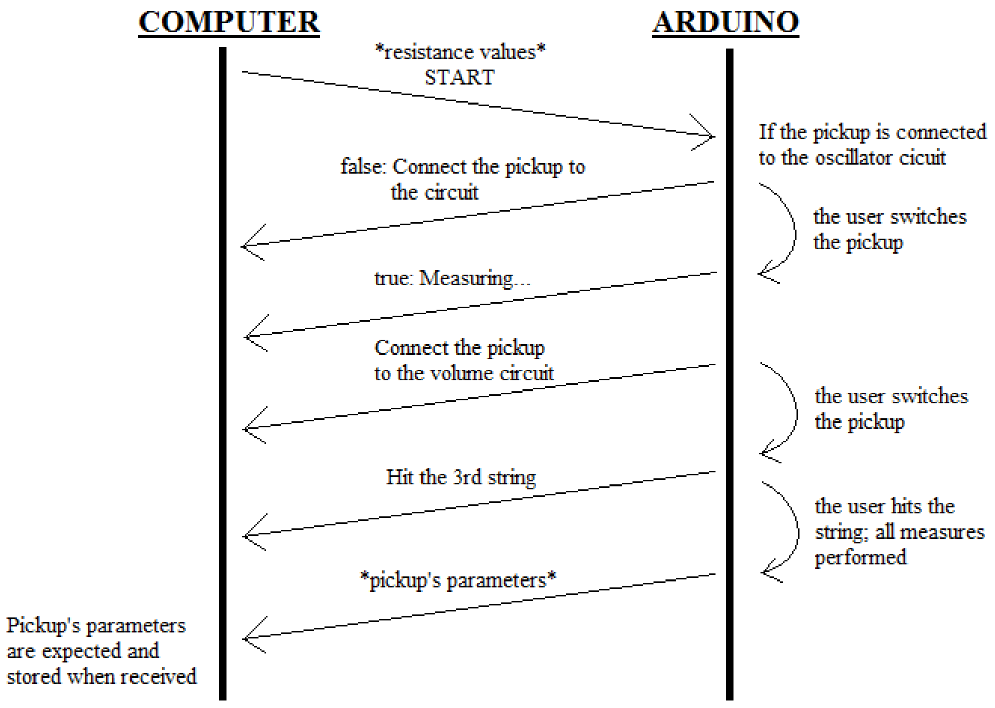

COMPUTER
ARDUINO
*resistance values* START
If the pickup is connected to the oscillator cicuit
false: Connect the pickup to the circuit
the user switches the pickup
true: Measuring...
Connect the pickup to the volume circuit
the user switches the pickup
Hit the 3rd string
the user hits the string; all measures performed
*pickup's parameters*
Pickup's parameters are expected and stored when received

***Figure III.19.*** *Communication protocol between the application and the microcontroller. The messages transmitted are strings of chars.*

## 3.6. MATLAB Graphic User Interface

The important thing about this project is the possibility of hearing comparisons between different circuitries with the same guitar body. This requires the following features in the application:

- Introducing different electric components.
- Saving and loading pickups and whole circuits.
- Recording audio clips from a known guitar.
- Reverse filtering of the recorded clip, in order to change the circuit in the simulation.
- Mixing a record with another circuit (filtering).
- Comparing two or more circuits by their Bode plot.
- Two guitar panels to compare the same record with two different circuits simultaneously.

The program used for this application is MATLAB. As a prototype, it is very useful using this program because it has a lot of pre-defined functions which have been used in this project, and also a very helpful IDE to create interfaces. However, it has been already stated that the goal of the project is to be an open software web application, allowing users to share their recordings and guitar configurations. It is part of the future work and development, since there is a project carried out at the FIB faculty based on this improvement.

In Figure III.20 the main window can be seen, where all features are easily identified.

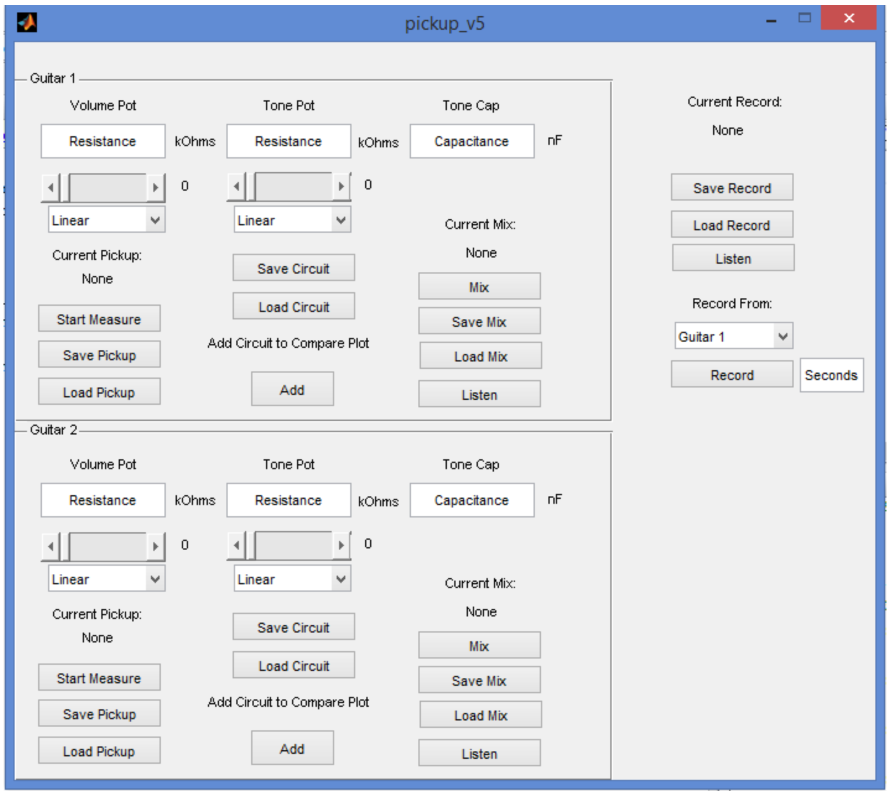
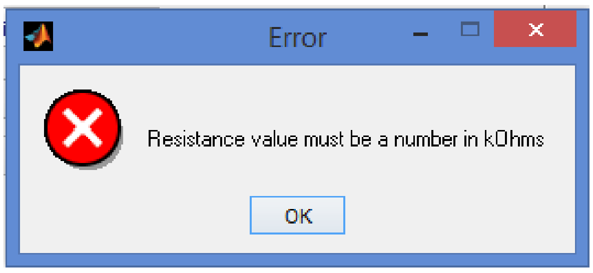


***Figure III.20.*** *Main window of the MATLAB application.*

To use the application properly, there are some steps to be followed. For example, the user has to introduce the resistance values before measuring a pickup, and has to define every circuit parameter before saving a circuit. He has to define the length of a recording before recording it. If the user does not proceed properly, an error pop-up window may appear, like the one in Figure III.21.



Resistance value must be a number in kOhms

***Figure III.21.*** *Error window example. This particular one will show when a value contains non-numerical characters.*

Since the market offers logarithmic type of potentiometers and they are very useful since human ear has also a logarithmic response (thus the control is a lot more instinctive for the user), the application offers them also. In Figure III.22 the selection of the potentiometer can be seen.

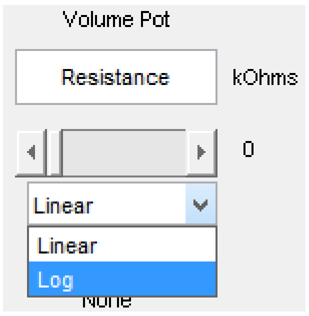


***Figure III.22.*** *The user can choose between a linear potentiometer ('Linear') and a logarithmic potentiometer ('Log').*

For more expert users in the field of electronics, the application offers also the ability to represent a circuit's transfer characteristic as a Bode plot and compare it to other circuits, using the values of the circuit's components to identify each trace (see Figure III.23).

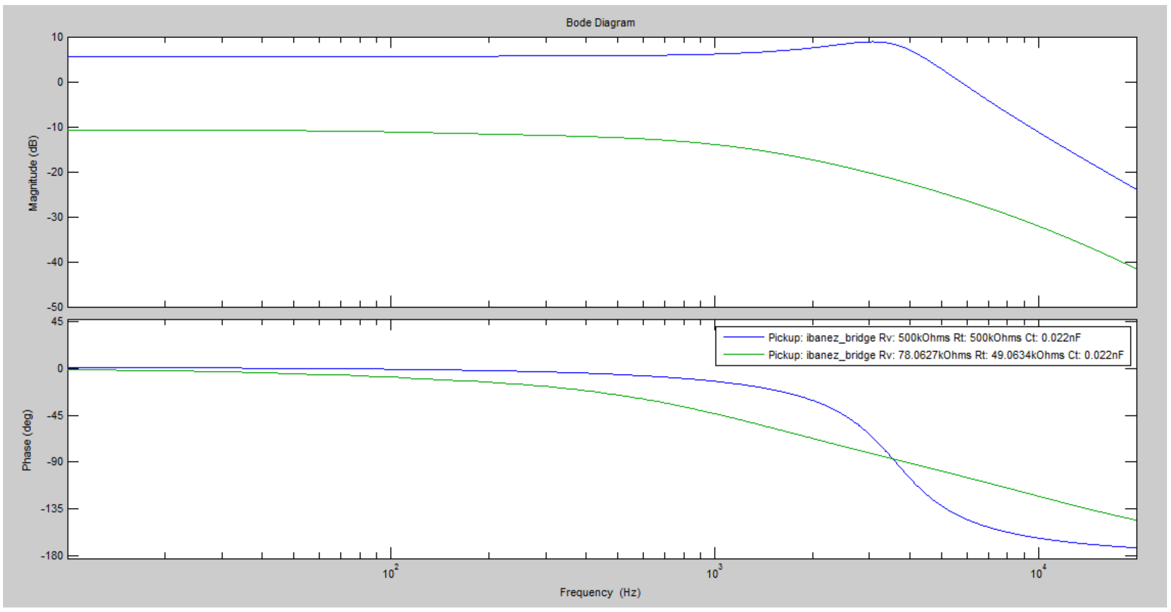


***Figure III.23.*** *Comparison between the same circuits only changing the potentiometers value.*

The code and the application user's guide can be seen in Annex 5 and Annex 6 respectively.

## 4. Results

Even though this project is not an analysis of choosing the 'best' pickup because everything involving tone 'goodness' is very subjective, some conclusions can be obtained analyzing the recordings and the pickup and the circuits used.

For example, comparing the same recording with two different circuits, show some difference in aggressiveness, and that is mostly because of the volume. One has a very high amplitude pickup (Seymour Duncan Humbucker) and some 500k resistors whether the other one has a lower amplitude pickup (Fender Stratocaster single coil) and some 250k resistors, which also lower the amplitude and therefore the aggressive sounding. The FFT of both filtered signals can be seen in Figure IV.1. They give the reader some information about the results. The shape of the FFT does not change very much from one to another. However, the amplitude and the amplitude ratio between harmonics change more and that's why the recognizable sound different.

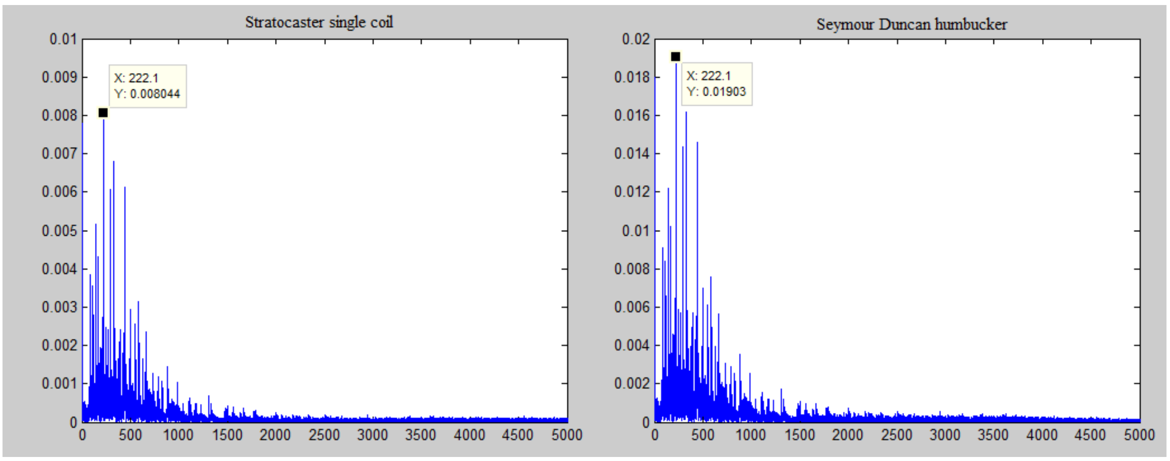


***Figure IV.1.*** *The shape is similar but the amplitude changes considerably in the two signals.*

However, the most valuable conclusion is that the position of the pickup in the guitar is very important to its tone. Figure IV.2 shows how amplitude ratio of the same chords are very different when captured with one pickup or another. The closer to the bridge, the higher amplitude in high frequencies.

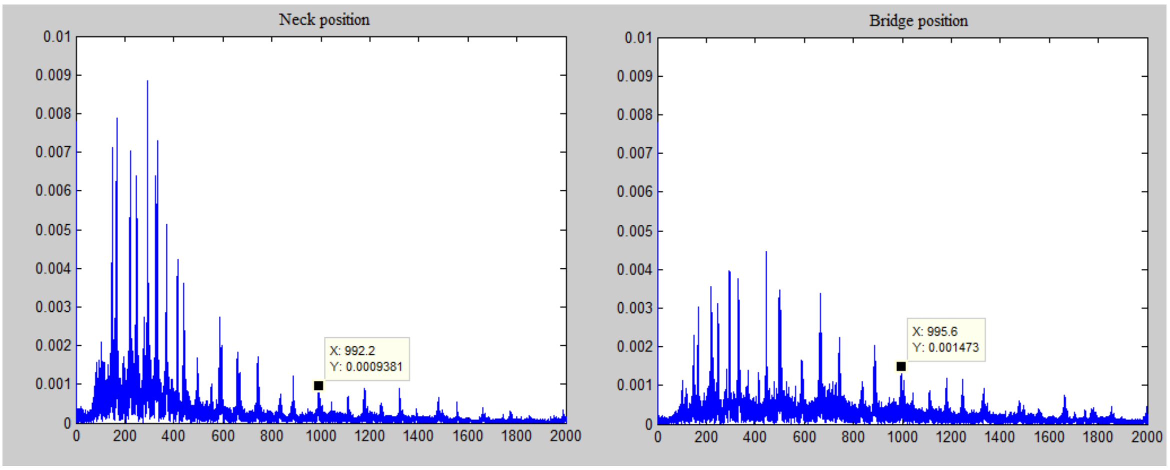


***Figure IV.2.*** *Same piece played in different positions. The amplitude ratio makes the tonal difference.*

Anyway, results are there for users to experiment and listen to achieve their own.

## 5. Budget

This project has not benefit purposes by itself. It is intended to exist as a DIY kit the user can get, and sales benefits should be used only for improving the project. However, benefits could come also from donations, but since the final goal has not been accomplished yet, it is too early to know the reception among guitarists.

It should be mentioned that if this project was intended to be produced massively, the microcontroller and the layout of the system would radically change, since what makes the most costly the project is the Arduino board, but it makes it simpler to construct as a DIY. To make this happen, the best scenario would be a company interested in showing its products to the user directly, so the products quality would be checked by the user instantly. The component list and the price is listed in the following tables.

***Table V.1.*** *Components and price of the Arduino shield.*

| Oscillator Circuit | | | | Volume Circuit | | | |
|---|---|---|---|---|---|---|---|
| **Type** | **Number** | **Price* (€)** | **Total (€)** | **Type** | **Number** | **Price* (€)** | **Total (€)** |
| Resistors | 14 | 0.01 | 0.14 | Resistors | 7 | 0.01 | 0.07 |
| Ceramic Capacitors | 9 | 0.05 | 0.45 | Electrolytic Capacitor | 1 | 0.10 | 0.10 |
| 1N4148 Diodes | 4 | 0.03 | 0.12 | TLC272ACP OPAMP | 1 | 0.74 | 0.74 |
| BC547 Transistors | 3 | 0.10 | 0.30 | Switch | 1 | 0.40 | 0.40 |
| MCP6291 OPAMP | 1 | 0.57 | 0.57 | Pins and Sockets | - | - | 1 |
| PCB | 1 | 10 | 10 | | | | |
| **TOTAL** | 13.89€ | | | | | | |

**Prices have been obtained in www.farnell.com on January 2015.*

What increases the price most are custom PCB orders. Cheapest dealers charge about 10€ for a board smaller than 100x100mm. Future work could include an extensively study of prices and pick up a best choice for manufacturing the kit's PCBs. It is intended for the user to order the board himself while there is no guitar components manufacturer willing to produce this analyzer to support its products. All needed files are intended to be available in the online application.

Cost of designing and prototyping in hours/person per cost is shown in Table V.2.

***Table V.2.*** *Work cost of the project.*

| **Task** | **Subtask** | **Span (working days)** | **Hours/person** | **Total cost* (€)** |
|---|---|---|---|---|
| **Analyzer design** | Design of the circuit | 15 | 60 | 480 |
| | Simulation using PSPICE | 5 | 20 | 160 |
| **Analyzer implementation** | Breadboard construction and test | 10 | 40 | 320 |
| | Arduino firmware and integration | 15 | 60 | 480 |
| | PCB design, construction and test | 20 | 80 | 640 |
| **Filter design & implementation** | Analysis of the circuit | 5 | 15 | 120 |
| | MATLAB simulation | 5 | 15 | 120 |
| | Software modeling, Filter design & implementation (MATLAB app) | 20 | 80 | 640 |
| **Reverse Filter** | | | | |
| **TOTAL** | - | 95 | 370 | 2960 |

**Cost is evaluated at cost of junior engineer hours (8€/h).*

Regarding to program licenses used, every task could be done with an open software program except for the MATLAB application, which student license cost 500€.

# 6. Conclusions and future development:

## 6.1. Conclusions

This project is an open door for guitarists to understand their instrument better and therefore it helps them achieve their tone goals as musicians. It has always been intended to provide the best simulator possible, but it is true that there is future development to be done. The main goal of this project was the electronic analyzer and, even though it is not perfect yet, it is now very complete and accurate.

What can be improved from this project –And it has been already started to be designed– is the platform where the guitar's simulations are held, and the simulation and recording processes as well.

## 6.2. Web application

This application is an extension of this project and is being performed by Aitor Terradellas, student of UPC Computer Science faculty (FIB).

His work consists in converting the MATLAB application into an open software free web application where every guitarist could share their experiments, changing circuits with components which parameters were measured by any other fellow guitarist (bassists also) and where advice and proofs of the sounds being discussed could be shared, thanks to the recordings filtered.

## 6.3. Other recommendations

For the global project, there is a need for studying the physical characteristics of humbucker pickups. Some tonal differences are taken into account in the scope of this project because they are related to volume, but physical differences such as those shown in Figure II.5 should be studied better in the future. Also, other circuit configurations should be analyzed, most importantly connecting two pickups in different positions to the same circuit, which has a very different tone than having any of the two connected singly.

Also, in a further scope, amplifiers should be studied and added to the variables and parameters guitarists could modify, so it would become in the future an application where anyone could find his dream sound without having to waste their money and time buying and trying to find it.

As an example, nowadays there are lots of guitar products reviews in YouTube. But if anyone is showing an amplifier in a video and playing a guitar which the watcher does not own, it is pointless to try to imagine how his guitar would sound with that amplifier. Same happens with pickups. So it would be like bringing a review of a product to the guitarists very home, and they could decide the worth of it based on the rest of their equipment interaction with it.

## Bibliography:

[1] H. Lemme. *Electric Guitar. Sound Secrets and Technology,* 1st ed. Amsterfoot, Netherlands: Elektor International Media BV, 2012.

[2] Professor S. Errede, “Measurement of EM Properties of Electric Guitar Pickups”. *UIUC Physics*, 2011 [Online] Available: http://www.ece.rochester.edu/courses/ECE140/resources/Guitar-Project/Electric_Guitar_Pickup_Measurements.pdf [Accessed: 20 September 2014].

[3] J. M. Miguel, “Notas de Clase – Diseño de Radioreceptores”. *UPC Telecom*, 2013 [Online] Available: http://designradio.blogspot.com.es/ [Accessed: October 2014].

[4] “Caffeinomane” profile, “Girino - Fast Arduino Oscilloscope”. *Instructables, 2013.* [Online] Available: http://www.instructables.com/id/Girino-Fast-Arduino-Oscilloscope/step11/How-the-Analog-Comparator-works/ [Accessed: 1 November 2014].

[5] Linear Technology©, “Half-Wave Rectifier”. *Linear Technology Solutions*, 1987 [Online] Available: http://www.linear.com/solutions/1607. [Accessed: 16 November 2014].

[6] Mathworks©, “MATLAB Documentation”. *Mathworks Product Help, 2014.* [Online] Available: http://es.mathworks.com/help/matlab/ [Accessed: December 2014].

[7] Wikipedia Contributors. "Digital Biquad Filter". *Wikipedia, the free encyclopedia*, 2014. [Online] Available: http://en.wikipedia.org/wiki/Digital_biquad_filter [Accessed: 3 December 2014].

# Annexes:

## Annex 1: Mathematical development of electric guitar's circuit.

To simulate the electronic filter of an electric guitar, the circuit has to be analyzed. Starting from the guitar circuit most common configuration, a Laplace transfer function is intended to be obtained, to later be converted into a Z transform in the MATLAB application.

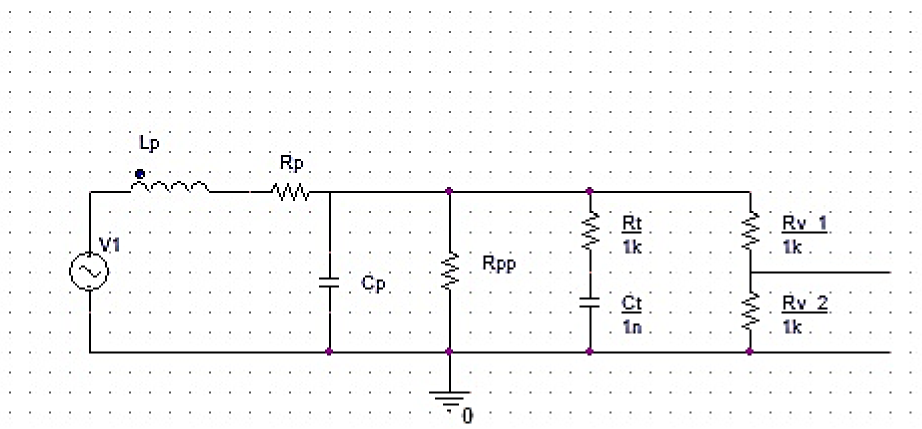
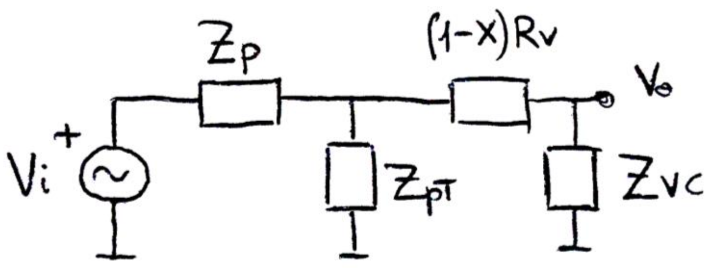


***Figure II.12.*** *Electric Guitar most common circuit.*

There is also a capacitor corresponding to the cable capacitance connected in parallel to $R_{V\_2}$, but only for future development purposes. The circuit can be simplified into two cascade voltage divider of impedances, as show next Figure:



Where every impedance has been transformed to frequency domain and potentiometers values are defined with two normalized variables y and x that present at value '1' the maximum resistance value for tone and volume potentiometers respectively:

$$Z_p = R_p + L_p s \tag{1}$$

$$Z_{pT} = \frac{R_{pp}R_T y C_T s + R_{pp}}{R_{pp}R_T y C_p C_t s^2 + R_{pp}C_p s + yR_T C_T s + R_{pp}C_T s + 1} \quad (2)$$

$$Z_{vc} = \frac{xR_v}{xR_v C_c s + 1} \quad (3)$$

Therefore, the transfer function is:

$$H(s) = \frac{Z_{pT}//((1-x)R_v + Z_{vc})}{Z_p + Z_{pT}//(Z_{vc} + (1-x)R_v)} \times \frac{Z_{vc}}{(1-x)R_v + Z_{vc}} \quad (4)$$

Where “//” defines the “parallel configuration of” for any two impedances. Developing the equation the following expression is obtained:

$$H(s) = \frac{Z_{pT}Z_{vc}}{Z_p Z_{pT} + Z_{vc}Z_p + (1-x)R_v Z_p + Z_{pT}Z_{vc} + Z_{pT}(1-x)R_v} \quad (5)$$

This leads to a numerator and a denominator:

NUM:

$$xR_v R_{pp} R_T y C_T s + xR_v R_{pp} \quad (6)$$

DEN:

$$\begin{gathered}
(R_p + L_p s)(R_{pp}R_T y C_T s + R_{pp})(xR_v C_c s + 1) + \\
xR_v(R_p + L_p s)(R_{pp}R_T y C_p C_T s^2 + (R_{pp}C_p + yR_T C_T + R_{pp}C_T)s + 1) + \\
(1-x)R_v(R_p + L_p s)(R_{pp}R_T y C_p C_T s^2 + (R_{pp}C_p + yR_T C_T + R_{pp}C_T)s + 1)(xR_v C_c s + 1) + \\
(R_{pp}R_T y C_T s + R_{pp})xR_v + (R_{pp}R_T y C_T s + R_{pp})(1-x)R_v(xR_v C_c s + 1)
\end{gathered} \quad (7)$$

Once multiplied every variable and grouped by the order of the s factor:

NUM:

$$xR_vR_{pp}R_TyC_T\boldsymbol{s} + xR_vR_{pp} \tag{8}$$

DEN:

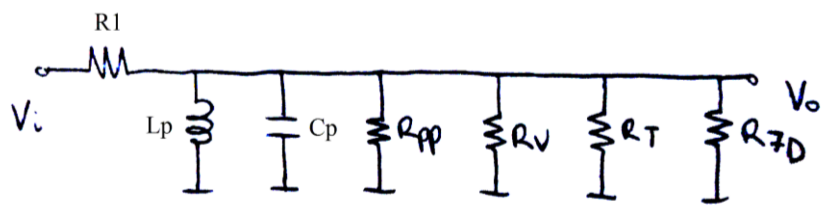
$$\begin{gathered}\Big((1-x)xR_v^2R_{pp}yR_TC_pC_TC_CL_p\Big)\boldsymbol{s^4} + \\ \Big(R_{pp}xR_vyR_TC_TC_cL_p + R_{pp}yR_TC_pC_TL_p \\ + (1-x)R_v\big(xR_vC_cL_p\big(R_{pp}C_p + yR_TC_T + R_{pp}C_T\big) + R_{pp}R_TyC_TC_pxR_vC_CR_p\big)\Big)\boldsymbol{s^3} + \\ \big(R_{pp}xR_vyR_TC_TC_c + R_{pp}xR_vC_cL_p + R_{pp}yR_TC_TL_p + R_{pp}R_pR_vyR_TC_TC_p \\ + R_vL_p\big(R_{pp}C_p + yR_TC_T + R_{pp}C_T\big) + (1-x)xR_v^2C_CR_p\big(R_{pp}C_p + yR_TC_T + R_{pp}C_T\big) \\ + (1-x)xR_v^2C_CL_p + (1-x)xR_v^2R_{pp}yR_TC_TC_c\big)\boldsymbol{s^2} + \\ \big(R_{pp}yR_TC_T + R_{pp}L_p + R_{pp}xR_vR_pC_C + R_pR_v\big(R_{pp}C_p + yR_TC_T + R_{pp}C_T\big) + R_vL_p \\ + (1-x)xR_v^2C_CR_p + R_{pp}R_vyR_TC_T + (1-x)xR_v^2R_{pp}C_C\big)\boldsymbol{s} + \\ R_{pp}R_p + R_vR_p + R_{pp}R_v\end{gathered} \tag{9}$$

This equation is inside the application and thus the transfer function is found, later transformed to Z domain.

# Annex 2: Mathematical formulas to determine pickup parameters

Phase shifting is the key to find all the pickup's parameters. In the oscillator circuit the phase is shifted on purpose using $C_{ph}$ capacitor connected and disconnected using a BJT transistor. But the transistor also has parasitic capacitance, so there is always phase shifting. Here can be seen the equation used for the correction of the phase shift and finding the real pickup values.

The circuit is the following:



With the following transfer function:

$$H(s) = \frac{\frac{L_p}{R_1} s}{\frac{s^2}{\omega_o^2} + \left(\frac{1}{R_1} + \frac{1}{R_p}\right) L_p + 1} \tag{1}$$

Where $R_p$ is the equivalent value of $R_{pp}$, $R_V$, $R_T$ and $R_{7D}$. This last one resistance represents the resistance value of the following resistances in the oscillator loop.

Phase of this transfer function is defined with the equation:

$$\phi(\omega) = \frac{\pi}{2} - \tan^{-1}\left(\frac{\frac{L_p \omega}{R_{eq}}}{\omega^2 + \omega_o^2}\right) \tag{2}$$

There are four measures using all combinations of both test capacitors $C_{test}$ and $C_{ph}$, with their corresponding phase shift and oscillation frequency:

| **Phase Shift** | **$C_{ph}$** | **$C_{test}$** |
|---|---|---|
| $\Phi_m$ | OFF | OFF |
| $\Phi_{ph}$ | ON | OFF |
| $\Phi_{testm}$ | OFF | ON |
| $\Phi_{testph}$ | ON | ON |

The phase shifting for every case is:

$$\phi_m = \frac{\pi}{2} - \tan^{-1}\left(\frac{\frac{\omega_m}{R_{eq}C}}{\omega_o^2 - \omega_m^2}\right) \tag{3}$$

$$\phi_{ph} = \frac{\pi}{2} - \tan^{-1}\left(\frac{\frac{\omega_{ph}}{R_{eq}C}}{\omega_o^2 - \omega_{ph}^2}\right) \tag{4}$$

$$\phi_{testm} = \frac{\pi}{2} - \tan^{-1}\left(\frac{\frac{\omega_{testm}}{R_{eq}(C + C_{test})}}{\omega_o^2 - \omega_{testm}^2}\right) \tag{5}$$

$$\phi_{testph} = \frac{\pi}{2} - \tan^{-1}\left(\frac{\frac{\omega_{testph}}{R_{eq}(C + C_{test})}}{\omega_o^2 - \omega_{testph}^2}\right) \tag{6}$$

Using equations 3 and 4 and dividing them, the resonance frequency of the pickup can be obtained and with equations 5 and 6 the resonance frequency with the test capacitor ON is obtained:

$$\frac{\omega_o^2 - \omega_{ph}^2}{\omega_o^2 - \omega_m^2} = \frac{\tan\left(\frac{\pi}{2} - \phi_m\right) {}^{\omega_{ph}}/_{\omega_m}}{\tan\left(\frac{\pi}{2} - \phi_{ph}\right)} = a, \tag{7}$$

$$\omega_o^2 = \frac{a\omega_m^2 - \omega_{ph}^2}{a - 1} \tag{8}$$

$$\frac{\omega_{otest}^2 - \omega_{testph}^2}{\omega_{otest}^2 - \omega_{testm}^2} = \frac{\tan\left(\frac{\pi}{2} - \phi_{testm}\right) {}^{\omega_{testph}}/_{\omega_{testm}}}{\tan\left(\frac{\pi}{2} - \phi_{testph}\right)} = b, \tag{9}$$

$$\omega_{otest}^2 = \frac{b\omega_{testm}^2 - \omega_{testph}^2}{b - 1} \tag{10}$$

The value of each phase shift is determined by the equation of a simple low pass filter, but the equivalent resistance of the low pass is the calculated with:

$$R_{ph} = (R_{71} + R_{diodes})//(R_{72} + R_{eq}) \tag{11}$$

Where R$_{eq}$ is the equivalent resistance of pickup, potentiometers and R$_1$ (see Figure III.13).

$$\phi_m = \tan^{-1}(\omega_m C_{trt} R_{ph}) \tag{12}$$

$$\phi_{ph} = \tan^{-1}(\omega_{ph} C_{ph} R_{ph}) \tag{13}$$

$$\phi_{testm} = \tan^{-1}(\omega_{testm} C_m R_{ph}) \tag{14}$$

$$\phi_{testph} = \tan^{-1}(\omega_{testph} C_{ph} R_{ph}) \tag{15}$$

Where C$_{trt}$ is the transistor's parasitic capacitance (6pF max.).

Finally, every parameter is obtained with the following formulas:

$$C = \frac{C_{test}}{\frac{\omega_o^2}{\omega_{otest}^2} - 1}, \qquad C_p = C - C_{ADC} - C_{trt} \tag{16}$$

$$L_p = \frac{1}{4\pi^2 f_o^2 C} \tag{17}$$

$$R_{eq} = \frac{\omega_{ph}}{C(\omega_o^2 - \omega_{ph}^2)\tan\left(\frac{\pi}{2} - \phi_{ph}\right)}, R_p = \frac{R_{eq}R_1}{R_{eq} - R_1}, \\ R_{pp} = \frac{1}{\frac{1}{R_p} - \frac{1}{R_v} - \frac{1}{R_T} - \frac{1}{R_{7D}}} \tag{18}$$

For series resistance measures, the formula is:

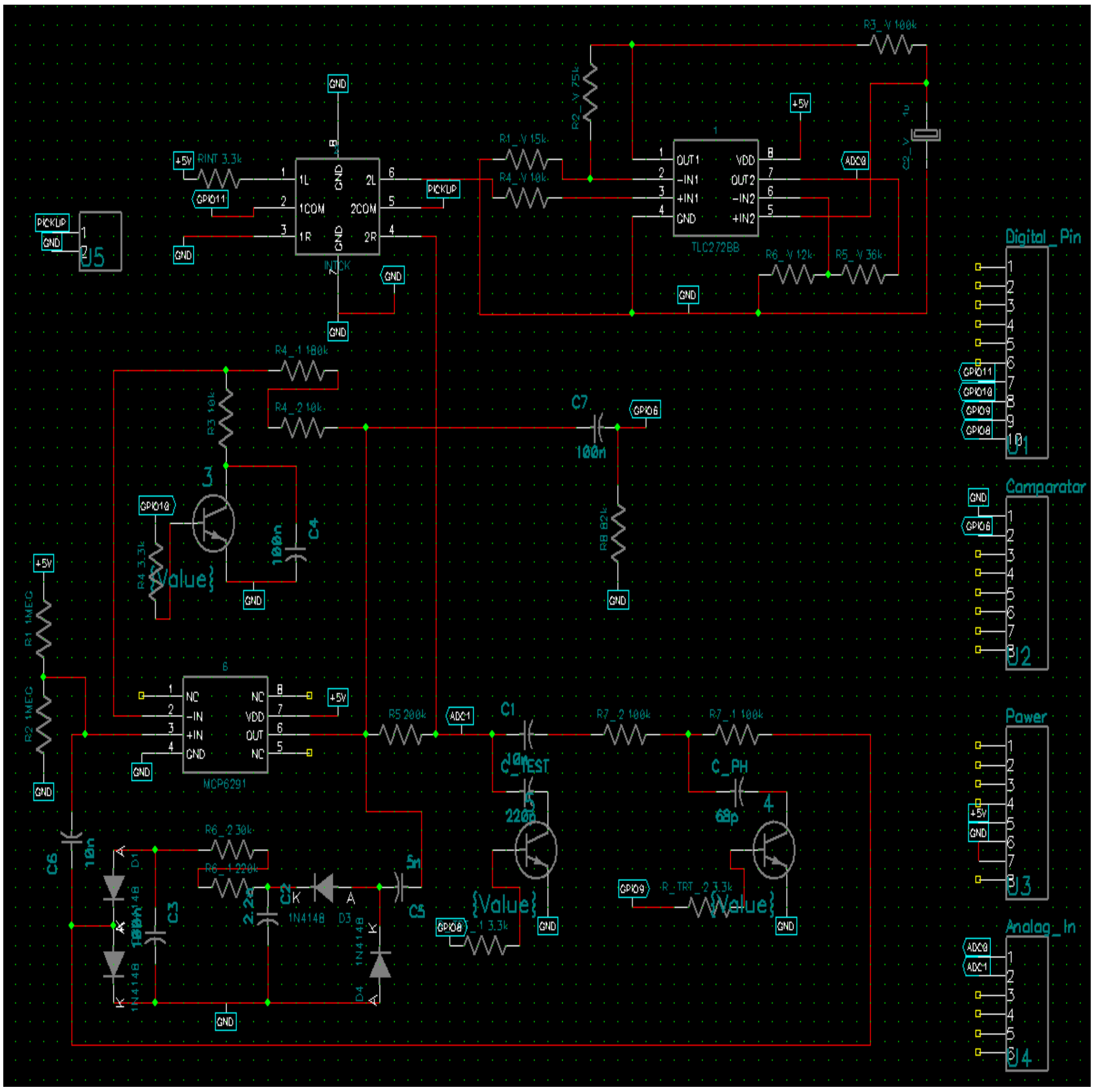
$$R_{s\_eq} = \frac{R_1 V_m}{V_{DD} - V_m}, R_s = \frac{1}{\frac{1}{R_{s\_eq}} - \frac{1}{R_v} - \frac{1}{R_T}} \tag{19}$$

Where V$_m$ is the voltage measured with the ADC module and V$_{DD}$ is the supply voltage.

For the volume measure, the voltage measured has a gain of $G = \frac{24}{\pi}$. This gain results from the OPAMPs gain (times 6 and times 4) and the DC component of a semi-rectified sine wave, which has a normalized amplitude of $\frac{1}{\pi}$.

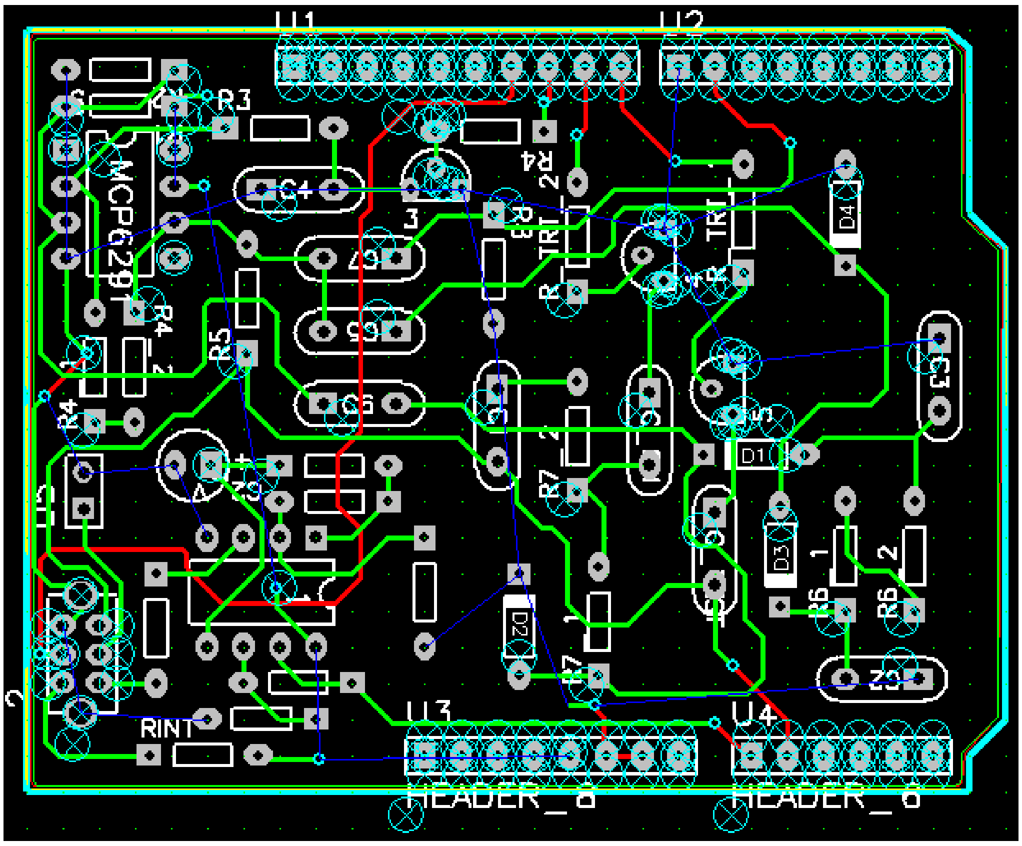
# Annex 3: PCB layouts and Components

All PCB captures are going to be shown in this Annex. The full resolution schematic appears next:

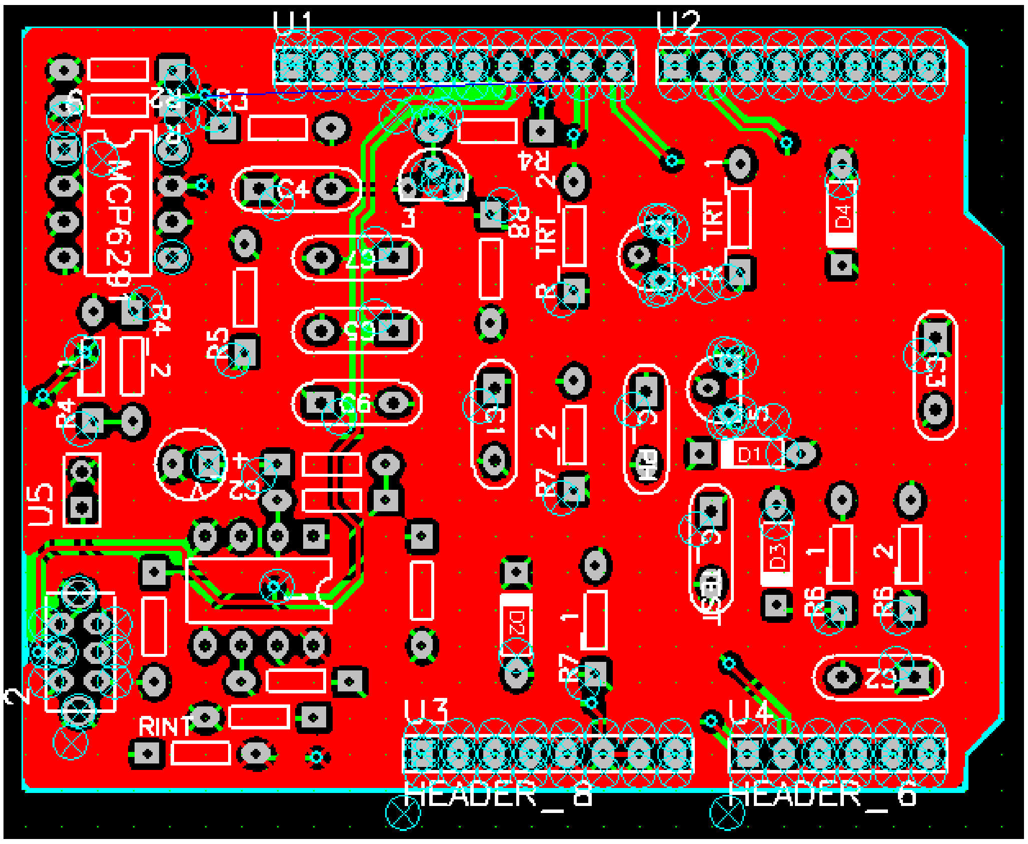


*Full Resolution Schematic.*

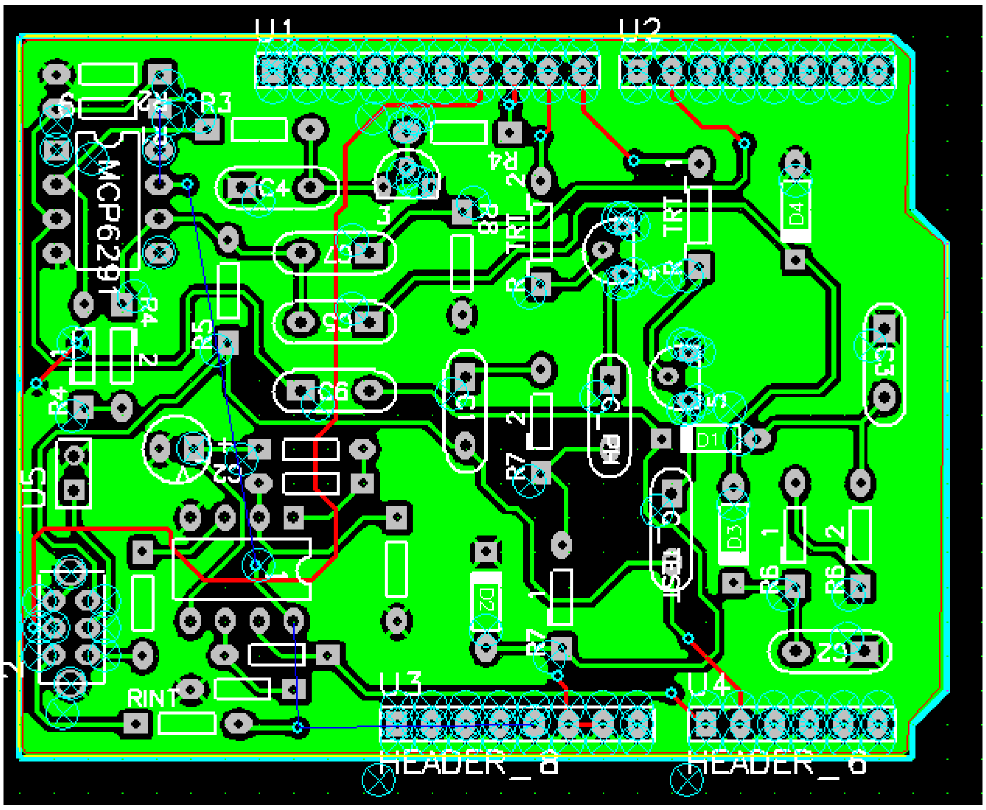
In the PCB design window lines are showed red for top layer and green for bottom layer. Also planes $V_{DD}$ on top layer is red and GND on bottom layer is green.



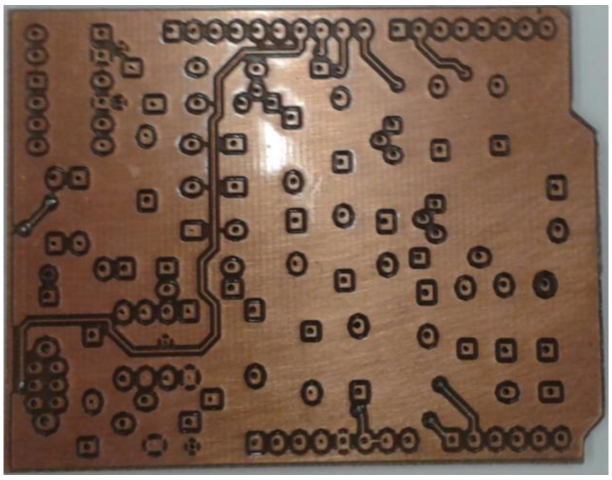
*Lines showed without the $V_{DD}$ and GND planes.*



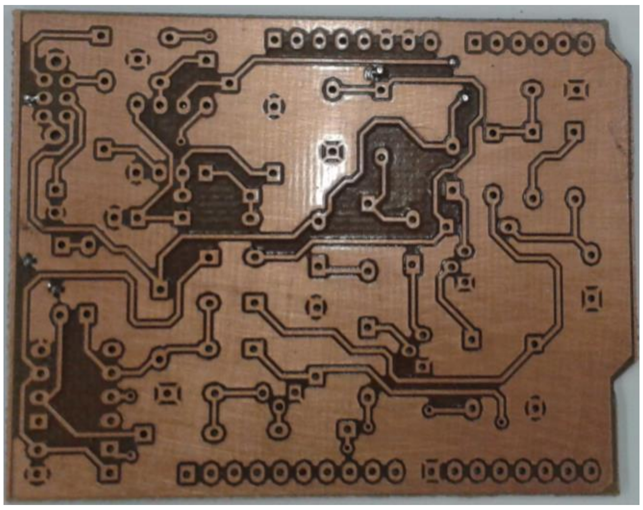
*Top Layer. All pin connections end in this layer because they are soldered the other way around the other components are.*



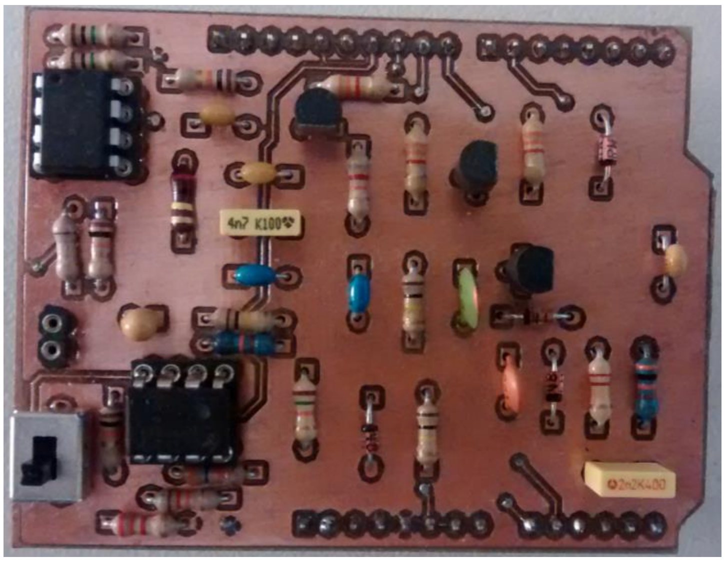
*Bottom Layer.*



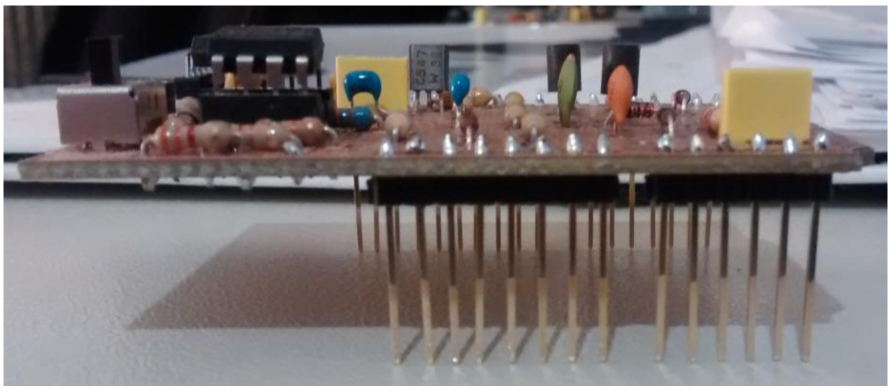
*Finished Top Layer.*



*Finished Bottom Layer.*



*Top view of the functional board.*



*Side view of the functional board.*

**List of Components:**

| Oscillator Circuit | | | Volume Circuit (and Board) | | |
|---|---|---|---|---|---|
| **Number** | **Part** | **Value** | **Number** | **Part** | **Value** |
| 3 | Resistor | 3.3k | 1 | Resistor | 3.3k |
| 1 | Resistor | 180k | 1 | Resistor | 10k |
| 2 | Resistor | 10k | 1 | Resistor | 15k |
| 1 | Resistor | 82k | 1 | Resistor | 75k |
| 2 | Resistor | 1M | 1 | Resistor | 12k |
| 1 | Resistor | 220k | 1 | Resistor | 36k |
| 1 | Resistor | 30k | 1 | Resistor | 100k |
| 1 | Resistor | 200k | 1 | Capacitor | 1µ |
| 2 | Resistor | 100k | 1 | OPAMP | TLC272 |
| 3 | Capacitor | 100n | 2 | Socket | 8-pin |
| 1 | Capacitor | 10n | 2 | Pin Header | 8x1 male |
| 1 | Capacitor | 2.2n | 1 | Pin Header | 6x1 male |
| 1 | Capacitor | 5n | 1 | Pin Header | 10x1 male |
| 1 | Capacitor | 10n | 1 | Pin Header | 2x1 female |
| 1 | Capacitor | 68p | 1 | Switch | DPDT, C&K OS202011MS2QS1 |
| 1 | Capacitor | 220p | | | |
| 3 | Transistor | BC547 | | | |
| 4 | Diode | 1N4148 | | | |
| 1 | OPAMP | MCP6291 | | | |

# Annex 4: Arduino firmware:

The code implemented for the Arduino microcontroller appears below:

```
#include <math.h>

#define M_PI_2 1.57079632679489661923
#define V_hold 70 //The value in order to start the acquisition
#define L 100 //Length of the volume vector
//Flags used for the timer
int flag_loop = 0;
int flag_switch = 0;
int timer = 0;
//Series Resistance variables
int V_adc[10];
double Vm = 0;
//Oscillator Circuit variables
int i = 0;
double f_m = 0;
double f_test = 0;
double f_ph = 0;
double f_test_ph = 0;
double phi_m = 0;
double phi_test = 0;
double phi_ph = 0;
double phi_test_ph = 0;
double a = 0;
double b = 0;
double wo2 = 0;
double wo2_test = 0;
double f0 = 0;
double C = 0;
double Cp = 0;
double Req = 0;
double Rp = 0;
double Rpp = 0;
double Rs = 0;
double Rs_eq = 0;
double R7_D = 300000;
double Rt = 0;
double Rv = 0;
double R1 = 200000;
double C_trt = 6E-12;
double C_ph = 68E-12;
double Ctest = 205E-12;
double Cadc = 20E-12;
double Lp = 0;
int Vol = 0;
int MUX = 0;
//flag used for counting the rising edge in the comparator only
boolean toggleCount = 0;
//sincronization variables with of the timer with the main loop
boolean flag_init = 0;
boolean flag_sincro = 0;

 ISR(TIMER1_COMPA_vect){ //timer1 interrupt 1Hz
   if(flag_init){
     if(flag_sincro){

         timer = 1;

     }
     flag_sincro = 1;
   }
 }
//Function that converts string vars into float vars
float stringToFloat(String x) {
  float res = 0;
  int i = 0;
  boolean decimal = false;
  while(i < x.length() && !decimal) {
    if(x[i] == '.') decimal = true;
    else if(!decimal) {
      int aux = x[i]-'0';
      res = (res*10) + aux;
    }
    ++i;
  }
  return res;
}
//Function that returns Rt and Rv as float variables
void stringToValues(String s) {
   int i = 0;
   int j = 0;
   float values[2];
   String aux = "";
   while(i < s.length()) {
      if(s[i] != '&') {
        aux = aux + String(s[i]);
      } else {
         values[j] = stringToFloat(aux);
         aux = "";
         ++j;
      }
     ++i;
   }
  values[j] = stringToFloat(aux);
  Rt = values[0];
  Rv = values[1];
}
//Function used to measure the volume of the pickup
int volumeMeasure(){
  int flag_end = 0;
  int flag_print = 0;
  int V_trigger  = 0;
  int V_pickup[L];
  float V_mean = 0;

  while(!flag_end){
    if(flag_print == 0){
      Serial.println("Connect the volume measuring circuit to the Guitar");
      flag_print =1;
    }
    flag_switch = digitalRead(11);
    if(flag_switch){
      delay(3000); //delay time in order to stabilize the Vout at 0V (when circuit disconnected Vout = Vdd)
      if(flag_print == 1){
        Serial.println("Hit the 3rd string (G), pushing it in the 12th fret touching the 2nd string (B) and releasing");
        flag_print = 2;
      }
      int local_flag = 0;
      while(!local_flag){
      V_trigger = analogRead(A0);
      if(V_trigger > V_hold){
        delay(1000);
        for(i = 0; i < L; i++){
        V_pickup[i] = analogRead(A0);
        delay(10);
        }
        for(i = 0; i < L; i++){
          V_mean = V_mean + V_pickup[i];
        }
        V_mean = V_mean / L;
        //Volume value before Circuitry
        V_mean = 5*24*V_mean/ M_PI/ 1024;
        flag_end = 1;
        local_flag = 1;
      }
     }
    }
  }
  return V_mean;
}

void setup(){
  Serial.begin(9600);
  pinMode(13, OUTPUT);
  pinMode(8,OUTPUT);
  pinMode(9,OUTPUT);
  pinMode(10,OUTPUT);
  cli();//stop interrupts

  //set timer1 interrupt at 1Hz
    TCCR1A = 0;// set entire TCCR1A register to 0
    TCCR1B = 0;// same for TCCR1B
    TCNT1  = 0;//initialize counter value to 0
    // set compare match register for 1hz increments
    OCR1A = 15624;// = (16*10^6) / (1*1024) - 1 (must be <65536)
    // turn on CTC mode
    TCCR1B |= (1 << WGM12);
    // Set CS12 and CS10 bits for 1024 prescaler
    TCCR1B |= (1 << CS12) | (1 << CS10);
    // enable timer compare interrupt
    TIMSK1 |= (1 << OCIE1A);

    sei();//allow interrupts

}

void loop(){
    //All the tasks begin when asked from the computer
    if(Serial.available() > 0){
      flag_switch = digitalRead(11);
      if(!flag_switch){
        flag_loop = 0;
        //Format of resistance values is: "Rt&Rv". Example: 250000&250000
        String s1 = Serial.readString();
        stringToValues(s1);
        //Start Measuring
        digitalWrite(8,LOW);
        digitalWrite(9,LOW);
        //The microcontroller informs the app that it has started
        Serial.println("Start");
        flag_init = 1;
        while(flag_init){
          if(MUX == 0){
            //The timer is sincronized with the loop
            while(flag_sincro){
              if(timer == 0){
                if((ACSR & 0x20) == 32){
                  if(toggleCount == 0){
                    f_m++;
                    toggleCount = 1;
                  }
                }
                else{
                 toggleCount = 0;
                }
              }
              else{
                MUX = 1;
                digitalWrite(8,HIGH);
                delay(100);
                flag_sincro = 0;
                timer = 0;
              }
            }
          }
          if(MUX == 1){
            while(flag_sincro){
              if(timer == 0){
                if((ACSR & 0x20) == 32){
                  if(toggleCount == 0){
                    f_test++;
                    toggleCount = 1;
                  }
                }
                else{
                 toggleCount = 0;
                }
              }
              else{
                MUX = 2;
                digitalWrite(8,LOW);
                digitalWrite(9,HIGH);
                delay(100);
                flag_sincro = 0;
                timer = 0;
              }
            }
          }
          if(MUX == 2){
            while(flag_sincro){
```

```
    if(timer == 0){
      if((ACSR & 0x20) == 32){
        if(toggleCount == 0){
          f_ph++;
          toggleCount = 1;
        }
      }
      else{
       toggleCount = 0;
      }
    }
    else{
      MUX = 3;
      digitalWrite(8,HIGH);
      digitalWrite(9,HIGH);
      delay(100);
      flag_sincro = 0;
      timer = 0;
    }
  }
}
if(MUX == 3){
  while(flag_sincro){
    if(timer == 0){
      if((ACSR & 0x20) == 32){
        if(toggleCount == 0){
          f_test_ph++;
          toggleCount = 1;
        }
      }
      else{
       toggleCount = 0;
      }
    }
    else{
      MUX = 4;
      digitalWrite(8,LOW);
      digitalWrite(9,LOW);
      delay(100);
      flag_sincro = 0;
      timer = 0;
    }
  }
}
if(MUX == 4){
  //When all four measures have been performed, the timer is "ignored"
  flag_init = 0;
  flag_sincro = 0;
  timer = 0;
  MUX = 0;

  ////////////////////////////////////////////////////
  //Series Resistance Measure
  digitalWrite(10,HIGH);
  delay(100);
  for(i = 0; i < 10;i++){
    V_adc[i] = analogRead(A1);
  }
  for(i = 0;i < 10; i++){
    Vm = Vm + V_adc[i];
  }
  Vm = Vm/10;
  Rs_eq = 200000 * Vm /(1000 - Vm); //1023 represents Vdd, 5V
  //The OPAMP Vout is about 4.88V, i.e. 1000
  Rs = 1 / (1/Rs_eq - 1/Rv - 1/Rt);

  digitalWrite(10,LOW);

  //Pickup's parameters calculation
  phi_m = atan(2*M_PI*f_m*C_trt*84000);
  phi_ph = atan(2*M_PI*f_ph*C_ph*84000);
  phi_test = atan(2*M_PI*f_test*C_trt*84000);
  phi_test_ph = atan(2*M_PI*f_test_ph*C_ph*84000);

  a = tan(M_PI_2 - phi_m)*f_ph / f_m / tan(M_PI_2 - phi_ph);
  b = tan(M_PI_2 - phi_test)*f_test_ph / f_test / tan(M_PI_2 - phi_test_ph);

  wo2 = (a*square(2*M_PI*f_m) - square(2*M_PI*f_ph)) / (a-1);
```

```
          wo2_test = (b*square(2*M_PI*f_test) - square(2*M_PI*f_test_ph)) / (b-1);

          f0 = sqrt(wo2) / (2*M_PI);
          C = Ctest / ((wo2/wo2_test) - 1);
          Cp = C - C_trt - Cadc;
          Req = 2*M_PI*f_m / (C*(wo2 - square(2*M_PI*f_m))*tan(M_PI_2 - phi_m));
          Rp = Req*R1 / (R1- Req);
          Rpp = 1/(1/Rp -1/Rv - 1/Rt -1/R7_D);
          Lp = 1 / (wo2*C);

          //Volume measure function call
          Vol = volumeMeasure();

          //Pickup's parameters sending to the computer
          Serial.println(Rs);
          Serial.println(Lp);
          Serial.println(Cp*1E+12);
          Serial.println(Rpp);
          Serial.println(Vol);

          //All variables used are reset
          f_m = 0;
          f_test = 0;
          f_ph = 0;
          f_test_ph = 0;
          Rs = 0;
          Vm = 0;
        }
      }
    }
    //The microcontroller knows if the proper circuit is connected. If not, it informs the user
    else{
     if(!flag_loop){
      Serial.println("Connect the oscillator circuit to the Guitar");
      flag_loop = 1;
    }
  }
 }
}
```

# Annex 5: MATLAB GUI code

The MATLAB GUI code is written below:

```
function varargout = pickup_last_version(varargin)

gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn',
@pickup_last_version_OpeningFcn, ...
                   'gui_OutputFcn',  @pickup_last_version_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end



% --- Executes just before pickup_last_version is made visible.
function pickup_last_version_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to pickup_last_version (see
VARARGIN)

% Choose default command line output for pickup_last_version
handles.output = hObject;

handles.newPickup = [0 0 0 0 0];
handles.newRecord = audiorecorder(44100,8,1);
handles.newCircuit = {'pickup_name',0,0,0,0,0};
handles.newMix = [0];
%Guitar 1 parameters
handles.currentPickup = handles.newPickup;
handles.currentCircuit = handles.newCircuit;
handles.currentMix = handles.newMix;
handles.Ct = 0;
handles.Rt = 0;
handles.y = 1;
handles.Rv = 0;
handles.x = 1;
set(handles.slider_vol,'Value',handles.x);
set(handles.text_slider_vol,'String',num2str(handles.x));
set(handles.slider_tone,'Value',handles.y);
set(handles.text_slider_tone,'String',num2str(handles.y));
%Guitar 2 parameters
handles.currentPickup_2 = handles.newPickup;
handles.currentCircuit_2 = handles.newCircuit;
handles.currentMix_2 = handles.newMix;
handles.Ct_2 = 0;
handles.Rt_2 = 0;
handles.y_2 = 1;
handles.Rv_2 = 0;
handles.x_2 = 1;
set(handles.slider_vol_2,'Value',handles.x_2);
set(handles.text_slider_vol_2,'String',num2str(handles.x_2));
set(handles.slider_tone_2,'Value',handles.y_2);
set(handles.text_slider_tone_2,'String',num2str(handles.y_2));
%Record Parameters
handles.seconds = 0;
handles.currentRecord = [0];
% Update handles structure
guidata(hObject, handles);

% UIWAIT makes pickup_last_version wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = pickup_last_version_OutputFcn(hObject, eventdata,
handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;



function edit_vol_Callback(hObject, eventdata, handles)
% hObject    handle to edit_vol (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

handles.Rv = str2double(get(hObject,'String')) * 1000;

if isnan(handles.Rv)
    errordlg('Resistance value must be a number in kOhms','Error');
    set(hObject,'String',0);
    handles.Rv = 0;
end

if(mod(handles.Rv, 1) > 0)
    errordlg('Maximum three decimals','Error');
    set(hObject,'String',0);
    handles.Rv = 0;
end
guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function edit_vol_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_vol (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit_tone_Callback(hObject, eventdata, handles)
% hObject    handle to edit_tone (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.Rt = str2double(get(hObject,'String')) * 1000;

if isnan(handles.Rt)
    errordlg('Resistance value must be a number in kOhms','Error');
    set(hObject,'String',0);
    handles.Rt = 0;
end

if(mod(handles.Rt, 1) > 0)
    errordlg('Maximum three decimals','Error');
    set(hObject,'String',0);
    handles.Rt = 0;
end
guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function edit_tone_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_tone (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on slider movement.
function slider_vol_Callback(hObject, eventdata, handles)
% hObject    handle to slider_vol (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
val = get(handles.popup_vol,'Value');
str = get(handles.popup_vol,'String');
if strcmp(str{val},'Log')
aux = get(hObject,'Value');
    if aux == 0
        handles.x = 0;
    else
%f(x) = a*exp(b*x)
%     Coefficients (with 95% confidence bounds):
a = 0.009624; %coefficients in order to get y = 0.1 when x = 0.5
b = 4.644;
```

```
handles.x = a*exp(b*aux);
    end
else
handles.x = get(hObject,'Value');
end
set(handles.text_slider_vol,'String',get(hObject,'Value'));
guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function slider_vol_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider_vol (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Executes on slider movement.
function slider_tone_Callback(hObject, eventdata, handles)
% hObject    handle to slider_tone (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
val = get(handles.popup_tone,'Value');
str = get(handles.popup_tone,'String');
if strcmp(str{val},'Log')
aux = get(hObject,'Value');
if aux == 0
    handles.y = 0;
else
%f(x) = a*exp(b*x)
%     Coefficients (with 95% confidence bounds):
a = 0.009624; %coefficients in order to get y = 0.1 when x = 0.5
b = 4.644;
handles.y = a*exp(b*aux);
end
else
handles.y = get(hObject,'Value');
end
set(handles.text_slider_tone,'String',get(hObject,'Value'));
guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function slider_tone_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider_tone (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Executes on selection change in popup_vol.
```

```
function popup_vol_Callback(hObject, eventdata, handles)
% hObject    handle to popup_vol (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
val = get(hObject,'Value');
str = get(hObject,'String');
if strcmp(str{val},'Log')
aux = get(handles.slider_vol,'Value');
if aux == 0
    handles.x = 0;
else
    %f(x) = a*exp(b*x)
    %     Coefficients (with 95% confidence bounds):
    a = 0.009624; %coefficients in order to get y = 0.1 when x = 0.5
    b = 4.644;
    handles.x = a*exp(b*aux);
end
else
handles.x = get(handles.slider_vol,'Value');
end
guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function popup_vol_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popup_vol (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on selection change in popup_tone.
function popup_tone_Callback(hObject, eventdata, handles)
% hObject    handle to popup_tone (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
val = get(hObject,'Value');
str = get(hObject,'String');
if strcmp(str{val},'Log')
aux = get(handles.slider_tone,'Value');
if aux == 0
    handles.y = 0;
else
    %f(x) = a*exp(b*x)
    %     Coefficients (with 95% confidence bounds):
    a = 0.009624; %coefficients in order to get y = 0.1 when x = 0.5
    b = 4.644;
    handles.y = a*exp(b*aux);
end
else
handles.y = get(handles.slider_tone,'Value');
end
guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function popup_tone_CreateFcn(hObject, eventdata, handles)
```

```
% hObject    handle to popup_tone (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit_tone_cap_Callback(hObject, eventdata, handles)
% hObject    handle to edit_tone_cap (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.Ct = str2double(get(hObject,'String')) * 1e-9;

if isnan(handles.Ct)
    errordlg('Capacitance value must be a number in nF','Error');
    set(hObject,'String',0);
    handles.Ct = 0;
end
guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function edit_tone_cap_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_tone_cap (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in start_meas_push.
function start_meas_push_Callback(hObject, eventdata, handles)
% hObject    handle to start_meas_push (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if handles.Rt > 0 && handles.Rv > 0

delete(instrfind({'Port'},{'COM3'}));
arduino = serial('COM3','Timeout',60);
arduino.BaudRate = 9600;
fopen(arduino);
pause(2);
var = strcat(num2str(handles.Rt),'&',num2str(handles.Rv));
fprintf(arduino,'%s',var);

str = fgets(arduino);
msgbox(str, 'Info');
if(strcmp(str,'Start') == 0)
    str = fgets(arduino);
    msgbox(str, 'Info');
end
```

```
pause(2);
msgbox('Measuring...','Info');

str = fgets(arduino);
msgbox(str, 'Info');

str = fgets(arduino);
msgbox(str, 'Info');

handles.newPickup(1) = str2double(fgets(arduino));
handles.newPickup(2) = str2double(fgets(arduino));
handles.newPickup(3) = str2double(fgets(arduino));
handles.newPickup(4) = str2double(fgets(arduino));
handles.newPickup(5) = str2double(fgets(arduino));

handles.currentPickup = handles.newPickup;
set(handles.text_current_pickup,'String', 'New Pickup');

pickup_msg = {['Rs = ' num2str(handles.currentPickup(1))]  ...
                    ['Lp = ' num2str(handles.currentPickup(2))], ...
                    ['Cp = ' num2str(handles.currentPickup(3))], ...
                    ['Rpp = ' num2str(handles.currentPickup(4))], ...
                    ['Vol = ' num2str(handles.currentPickup(5))]};

msgbox(pickup_msg, 'Pickup');
guidata(hObject,handles);

fclose(arduino);
else
    errordlg('Set the circuit parameters first!','Error');
end

% --- Executes on button press in push_save_pickup.
function push_save_pickup_Callback(hObject, eventdata, handles)
% hObject    handle to push_save_pickup (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

 if strcmp(get(handles.text_current_pickup,'String'),('New Pickup'))
     name = inputdlg('Enter pickup Name:','Save Pickup',[1 30]);
     name = name{1}; %Get the string of the cell
     filename = strcat(name,'.pickup');
     csvwrite(filename,handles.currentPickup);
     set(handles.text_current_pickup, 'String',name);
 else
     errordlg('You have not measured a new Pickup!','Error');
 end
  guidata(hObject, handles);

% --- Executes on button press in push_load_pickup.
function push_load_pickup_Callback(hObject, eventdata, handles)
% hObject    handle to push_load_pickup (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
name = inputdlg('Enter pickup Name:','Load Pickup',[1 30]);
name = name{1}; %Get the string of the cell

filename = strcat(name, '.pickup');
```

```
handles.currentPickup = csvread(filename);

set(handles.text_current_pickup,'String', name);
pickup_msg = {['Rs = ' num2str(handles.currentPickup(1))],...
                    ['Lp = ' num2str(handles.currentPickup(2))],...
                    ['Cp = ' num2str(handles.currentPickup(3))],...
                    ['Rpp = ' num2str(handles.currentPickup(4))],...
                    ['Vol = ' num2str(handles.currentPickup(5))]};

msgbox(pickup_msg, 'Pickup');
guidata(hObject,handles);


% --- Executes on button press in push_save_circuit.
function push_save_circuit_Callback(hObject, eventdata, handles)
% hObject    handle to push_save_circuit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
 if strcmp(get(handles.text_current_pickup,'String'),'None') == 0
     if handles.Rt > 0 && handles.Rv > 0 && handles.Ct > 0
name = inputdlg('Enter Circuit Name:','Save Circuit',[1 30]);
     name = name{1}; %Get the string of the cell
     filename = strcat(name,'.circ');

fid = fopen(filename,'wt+');
pickup_name = get(handles.text_current_pickup,'String');
fprintf(fid, '%s\n',pickup_name);
fprintf(fid, '%s\n',num2str(handles.Rv));
fprintf(fid, '%s\n',num2str(handles.x));
fprintf(fid, '%s\n',num2str(handles.Rt));
fprintf(fid, '%s\n',num2str(handles.y));
fprintf(fid, '%s\n',num2str(handles.Ct));
fclose(fid);
guidata(hObject,handles);
    else
    errordlg('Set the circuit parameters first!','Error');
    end

 else
     errordlg('You have not selected a Pickup!','Error');
 end

% --- Executes on button press in push_load_circuit.
function push_load_circuit_Callback(hObject, eventdata, handles)
% hObject    handle to push_load_circuit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
name_circ = inputdlg('Enter Circuit Name:','Load Circuit',[1 30]);
name_circ = name_circ{1}; %Get the string of the cell

filename = strcat(name_circ, '.circ');

fid = fopen(filename,'rt+');

name_pickup = strcat(fgets(fid));
file_pickup = strcat(name_pickup,'.pickup');
handles.currentPickup = csvread(file_pickup);
handles.Rv = str2double(fgets(fid));
handles.x = str2double(fgets(fid));
handles.Rt = str2double(fgets(fid));
handles.y = str2double(fgets(fid));
handles.Ct = str2double(fgets(fid));

set(handles.text_current_pickup,'String', name_pickup);
set(handles.edit_vol,'String',num2str(handles.Rv / 1000));
set(handles.edit_tone,'String',num2str(handles.Rt / 1000));
set(handles.edit_tone_cap,'String',num2str(handles.Ct*1e9));
set(handles.slider_vol,'Value',handles.x);
set(handles.slider_tone,'Value',handles.y);
set(handles.text_slider_vol,'String',num2str(handles.x));
set(handles.text_slider_tone,'String',num2str(handles.y));

pickup_msg = {['Rs = ' num2str(handles.currentPickup(1))],...
                    ['Lp = ' num2str(handles.currentPickup(2))],...
                    ['Cp = ' num2str(handles.currentPickup(3))],...
                    ['Rpp = ' num2str(handles.currentPickup(4))],...
                    ['Vol = ' num2str(handles.currentPickup(5))]};

msgbox(pickup_msg, 'Pickup');
guidata(hObject,handles);

% --- Executes on button press in push_add.
function push_add_Callback(hObject, eventdata, handles)
% hObject    handle to push_add (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if strcmp(get(handles.text_current_pickup,'String'),'None') == 0
    figure(1);
    this_tf = transfer_function(handles);
    w = linspace(0,2*pi*20000,2000);
    P = bodeoptions;
    P.FreqUnits = 'Hz';
    bode(this_tf,w,P);
    Rv_actual = handles.Rv * handles.x/ 1000;
    Rt_actual = handles.Rt * handles.y/ 1000;
    Ct_actual = handles.Ct * 1e6;
    msg = ['Pickup: ' get(handles.text_current_pickup,'String')...
           ' Rv: ' num2str(Rv_actual) 'kOhms' ...
           ' Rt: ' num2str(Rt_actual) 'kOhms' ...
           ' Ct: ' num2str(Ct_actual) 'nF'];
    ch = get(gca,'Children');
    set(ch(1),'DisplayName',msg);
    legend('-DynamicLegend');
    hold all
 else
     errordlg('You have not selected a Pickup!','Error');
 end
   guidata(hObject, handles);

function edit_seconds_Callback(hObject, eventdata, handles)
% hObject    handle to edit_seconds (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.seconds = str2double(get(hObject,'String'));

if isnan(handles.seconds)
```

```
    errordlg('Seconds must be a number!','Error');
    set(hObject,'String',0);
    handles.seconds = 0;
end

guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function edit_seconds_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_seconds (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in push_record.
function push_record_Callback(hObject, eventdata, handles)
% hObject    handle to push_record (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if strcmp(get(handles.text_current_pickup,'String'),'None') == 0
   if handles.Rt > 0 && handles.Rv > 0 && handles.Ct > 0

    if handles.seconds > 0
       msgbox('Record will start in...','Info');
       pause(1);
       msgbox('3','Info');
       pause(1);
       msgbox('2','Info');
       pause(1);
       msgbox('1','Info');
       pause(1);
       msgbox('START','Info');
       recordblocking(handles.newRecord,handles.seconds);
       msgbox('End of Recording','Info');
       handles.currentRecord = getaudiodata(handles.newRecord);
       %Reverse Filtering taking into account which guitar is being
       %recorded
       val = get(handles.popup_guitar,'Value');
       str = get(handles.popup_guitar,'String');
       if strcmp(str{val},'Guitar 1')
           H = transfer_function(handles);
       elseif strcmp(str{val},'Guitar 2')
           H = transfer_function_2(handles);
       end
       Z = c2d(H,1/44100);
       %since it is reversed, b(num) is den and a(den) is num
       b = (Z.den{1});
       a = (Z.num{1});
       if a(1) == 0
           a = a(2:length(a));
       end
       handles.currentRecord = filter(b,a,handles.currentRecord);
       set(handles.text_current_record,'String', 'New Record');

    else
        errordlg('Select duration of recording!','Error');
    end
   else
    errordlg('Set the circuit parameters first!','Error');
   end

 else
     errordlg('You have not selected a Pickup!','Error');
 end
   guidata(hObject, handles);

% --- Executes on button press in push_save_record.
function push_save_record_Callback(hObject, eventdata, handles)
% hObject    handle to push_save_record (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if strcmp(get(handles.text_current_record,'String'),('New Record'))
     name = inputdlg('Enter Record Name:','Save Record',[1 30]);
     name = name{1}; %Get the string of the cell
     filename = strcat(name,'.rec');
     csvwrite(filename,handles.currentRecord);
     set(handles.text_current_record, 'String',name);
 else
     errordlg('You have not recorded a new Record!','Error');
 end
    guidata(hObject, handles);

% --- Executes on button press in push_load_record.
function push_load_record_Callback(hObject, eventdata, handles)
% hObject    handle to push_load_record (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
name = inputdlg('Enter Record Name:','Load Record',[1 30]);
name = name{1}; %Get the string of the cell

filename = strcat(name, '.rec');
handles.currentRecord = csvread(filename);

set(handles.text_current_record,'String', name);
guidata(hObject,handles);

% --- Executes on button press in push_listen_record.
function push_listen_record_Callback(hObject, eventdata, handles)
% hObject    handle to push_listen_record (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if strcmp(get(handles.text_current_record,'String'),'None') == 0
 p = audioplayer(handles.currentRecord,44100);
 playblocking(p);
else
     errordlg('You have not selected a Record!','Error');
 end
   guidata(hObject, handles);

% --- Executes on button press in push_mix.
```

```
function push_mix_Callback(hObject, eventdata, handles)
% hObject    handle to push_mix (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if strcmp(get(handles.text_current_pickup,'String'),'None') == 0
    if strcmp(get(handles.text_current_record,'String'),('None')) ==0
        if handles.Rt > 0 && handles.Rv > 0 && handles.Ct > 0
           %Current Circuit Filtering
           H = transfer_function(handles);
           Z = c2d(H,1/44100);
           b = (Z.num{1});
           a = (Z.den{1});
           handles.currentMix = filter(b,a,handles.currentRecord);
           set(handles.text_current_mix,'String', 'New Mix');
        else
            errordlg('Set the circuit parameters first!','Error');

        end
    else
     errordlg('You have not recorded a new Record!','Error');
    end
else
     errordlg('You have not selected a Pickup!','Error');
end
   guidata(hObject, handles);

% --- Executes on button press in push_save_mix.
function push_save_mix_Callback(hObject, eventdata, handles)
% hObject    handle to push_save_mix (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% --- Executes on button press in push_load_mix.
if strcmp(get(handles.text_current_mix,'String'),('New Mix'))
     name = inputdlg('Enter Mix Name:','Save Mix',[1 30]);
     name = name{1}; %Get the string of the cell

     %First audio file is saved
     filename = strcat(name,'.mix');
     csvwrite(filename,handles.currentMix);
     set(handles.text_current_mix, 'String',name);

     %Then circuit file is saved
     filename = strcat(name,'.circ');
     fid = fopen(filename,'wt+');
     pickup_name = get(handles.text_current_pickup,'String');
     fprintf(fid, '%s\n',pickup_name);
     fprintf(fid, '%s\n',num2str(handles.Rv));
     fprintf(fid, '%s\n',num2str(handles.x));
     fprintf(fid, '%s\n',num2str(handles.Rt));
     fprintf(fid, '%s\n',num2str(handles.y));
     fprintf(fid, '%s\n',num2str(handles.Ct));
     fclose(fid);
 else
     errordlg('You have not created a new Mix!','Error');
 end
    guidata(hObject, handles);

function push_load_mix_Callback(hObject, eventdata, handles)
% hObject    handle to push_load_mix (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
name = inputdlg('Enter Mix Name:','Load Mix',[1 30]);
name = name{1}; %Get the string of the cell

%First Audio file is loaded
filename = strcat(name, '.mix');
handles.currentMix = csvread(filename);
set(handles.text_current_mix,'String', name);

%Then Circuit file is loaded
filename = strcat(name, '.circ');

fid = fopen(filename,'rt+');

name_pickup = strcat(fgets(fid));
file_pickup = strcat(name_pickup,'.pickup');
handles.currentPickup = csvread(file_pickup);

handles.Rv = str2double(fgets(fid));
handles.x = str2double(fgets(fid));
handles.Rt = str2double(fgets(fid));
handles.y = str2double(fgets(fid));
handles.Ct = str2double(fgets(fid));

set(handles.text_current_pickup,'String', name_pickup);
set(handles.edit_vol,'String',num2str(handles.Rv / 1000));
set(handles.edit_tone,'String',num2str(handles.Rt / 1000));
set(handles.edit_tone_cap,'String',num2str(handles.Ct*1e9));
set(handles.slider_vol,'Value',handles.x);
set(handles.slider_tone,'Value',handles.y);
set(handles.text_slider_vol,'String',num2str(handles.x));
set(handles.text_slider_tone,'String',num2str(handles.y));

pickup_msg = {['Rs = ' num2str(handles.currentPickup(1))],...
                    ['Lp = ' num2str(handles.currentPickup(2))],...
                    ['Cp = ' num2str(handles.currentPickup(3))],...
                    ['Rpp = ' num2str(handles.currentPickup(4))],...
                    ['Vol = ' num2str(handles.currentPickup(5))]};

msgbox(pickup_msg, 'Pickup');

guidata(hObject,handles);

% --- Executes on button press in push_listen_mix.
function push_listen_mix_Callback(hObject, eventdata, handles)
% hObject    handle to push_listen_mix (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if strcmp(get(handles.text_current_mix,'String'),'None') == 0
 p = audioplayer(handles.currentMix,44100);
 playblocking(p);
else
     errordlg('You have not selected a Mix!','Error');
 end
   guidata(hObject, handles);
```

```
function H = transfer_function(handles)

Rp = handles.currentPickup(1);
Lp = handles.currentPickup(2);
Cp = handles.currentPickup(3)  * 1e-12;
Rpp= handles.currentPickup(4);
Vol= handles.currentPickup(5);
Rt = handles.Rt;
Rv = handles.Rv;
Ct = handles.Ct;
Cc = 0;
x = handles.x;
y = handles.y;
%The Pickup amplitude is normalized with the reference of 50mV the
%circuitry effect is taken into account in the arduino code
A = Vol*5/1024/50e-3*pi/24;
n1 = A*x*Rv*Rpp*Rt*y*Ct;
n0 = A*x*Rv*Rpp;

d4 = (1-x)*x*Rv^2*Rpp*y*Rt*Cp*Ct*Cc*Lp;
d3 = Rpp*x*Rv*y*Rt*Ct*Cc*Lp + Rpp*Rv*y*Rt*Ct*Cp*Lp + (1-
x)*Rv*(x*Rv*Cc*Lp*(Rpp*Cp+y*Rp*Ct+Rpp*Ct) + Rpp*Rt*y*Cp*Ct*x*Rv*Rp*Cc);
d2 = Rpp*x*Rv*y*Rt*Ct*Cc + Rpp*x*Rv*Cc*Lp + Rpp*y*Rt*Ct*Lp +
Rpp*Rp*y*Rt*Rv*Cp*Ct + Rv*Lp*(Rpp*Cp+y*Rp*Ct+Rpp*Ct) + (1-
x)*x*Rv^2*Rp*Cc*(Rpp*Cp+y*Rp*Ct+Rpp*Ct) + (1-x)*x*Rv^2*Cc*Lp + (1-
x)*x*Rv^2*Rpp*y*Rt*Ct*Cc;
d1 = Rpp*Rt*y*Ct + Rp*Lp + Rpp*x*Rv*Rp*Cc +
Rv*Rp*(Rpp*Cp+y*Rp*Ct+Rpp*Ct) + Rv*Lp + (1-x)*x*Rv^2*Rp*Cc +
Rpp*Rv*y*Rt*Ct + (1-x)*x*Rv^2*Rpp*Cc;
d0 = Rp*Rpp + Rv*Rp + Rv*Rpp;

H = tf([n1 n0],[d4 d3 d2 d1 d0]);

function edit_vol_2_Callback(hObject, eventdata, handles)
% hObject    handle to edit_vol (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

handles.Rv_2 = str2double(get(hObject,'String')) * 1000;

if isnan(handles.Rv_2)
    errordlg('Resistance value must be a number in kOhms','Error');
    set(hObject,'String',0);
    handles.Rv_2 = 0;
end

if(mod(handles.Rv_2, 1) > 0)
    errordlg('Maximum three decimals','Error');
    set(hObject,'String',0);
    handles.Rv_2 = 0;
end
guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function edit_vol_2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_vol (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit_tone_2_Callback(hObject, eventdata, handles)
% hObject    handle to edit_tone (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.Rt_2 = str2double(get(hObject,'String')) * 1000;

if isnan(handles.Rt_2)
    errordlg('Resistance value must be a number in kOhms','Error');
    set(hObject,'String',0);
    handles.Rt_2 = 0;
end

if(mod(handles.Rt_2, 1) > 0)
    errordlg('Maximum three decimals','Error');
    set(hObject,'String',0);
    handles.Rt_2 = 0;
end
guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function edit_tone_2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_tone (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on slider movement.
function slider_vol_2_Callback(hObject, eventdata, handles)
% hObject    handle to slider_vol (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
val = get(handles.popup_vol_2,'Value');
str = get(handles.popup_vol_2,'String');
if strcmp(str{val},'Log')
aux = get(hObject,'Value');
    if aux == 0
        handles.x_2 = 0;
    else
%f(x) = a*exp(b*x)
%     Coefficients (with 95% confidence bounds):
a = 0.009624; %coefficients in order to get y = 0.1 when x = 0.5
b = 4.644;
handles.x_2 = a*exp(b*aux);
    end
else
handles.x_2 = get(hObject,'Value');
```

```
end
set(handles.text_slider_vol_2,'String',get(hObject,'Value'));
guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function slider_vol_2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider_vol (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Executes on slider movement.
function slider_tone_2_Callback(hObject, eventdata, handles)
% hObject    handle to slider_tone (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
val = get(handles.popup_tone_2,'Value');
str = get(handles.popup_tone_2,'String');
if strcmp(str{val},'Log')
aux = get(hObject,'Value');
if aux == 0
    handles.y_2 = 0;
else
%f(x) = a*exp(b*x)
%      Coefficients (with 95% confidence bounds):
a = 0.009624; %coefficients in order to get y = 0.1 when x = 0.5
b = 4.644;
handles.y_2 = a*exp(b*aux);
end
else
handles.y_2 = get(hObject,'Value');
end
set(handles.text_slider_tone_2,'String',get(hObject,'Value'));
guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function slider_tone_2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider_tone (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Executes on selection change in popup_vol.
function popup_vol_2_Callback(hObject, eventdata, handles)
% hObject    handle to popup_vol (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
val = get(hObject,'Value');
str = get(hObject,'String');
```

```
if strcmp(str{val},'Log')
aux = get(handles.slider_vol_2,'Value');
if aux == 0
    handles.x_2 = 0;
else
    %f(x) = a*exp(b*x)
    %     Coefficients (with 95% confidence bounds):
    a = 0.009624; %coefficients in order to get y = 0.1 when x = 0.5
    b = 4.644;
    handles.x_2 = a*exp(b*aux);
end
else
handles.x_2 = get(handles.slider_vol_2,'Value');
end
guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function popup_vol_2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popup_vol (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on selection change in popup_tone.
function popup_tone_2_Callback(hObject, eventdata, handles)
% hObject    handle to popup_tone (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
val = get(hObject,'Value');
str = get(hObject,'String');
if strcmp(str{val},'Log')
aux = get(handles.slider_tone_2,'Value');
if aux == 0
    handles.y_2 = 0;
else
    %f(x) = a*exp(b*x)
    %     Coefficients (with 95% confidence bounds):
    a = 0.009624; %coefficients in order to get y = 0.1 when x = 0.5
    b = 4.644;
    handles.y_2 = a*exp(b*aux);
end
else
handles.y_2 = get(handles.slider_tone_2,'Value');
end
guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function popup_tone_2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popup_tone (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit_tone_cap_2_Callback(hObject, eventdata, handles)
% hObject    handle to edit_tone_cap (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.Ct_2 = str2double(get(hObject,'String')) * 1e-9;

if isnan(handles.Ct_2)
    errordlg('Capacitance value must be a number in nF','Error');
    set(hObject,'String',0);
    handles.Ct_2 = 0;
end
guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function edit_tone_cap_2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_tone_cap (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in start_meas_push.
function start_meas_push_2_Callback(hObject, eventdata, handles)
% hObject    handle to start_meas_push (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if handles.Rt_2 > 0 && handles.Rv_2 > 0

delete(instrfind({'Port'},{'COM3'}));
arduino = serial('COM3','Timeout',60);
arduino.BaudRate = 9600;
fopen(arduino);
pause(2);
var = strcat(num2str(handles.Rt_2),'&',num2str(handles.Rv_2));
fprintf(arduino,'%s',var);

str = fgets(arduino);
msgbox(str, 'Info');
if(strcmp(str,'Start') == 0)
    str = fgets(arduino);
    msgbox(str, 'Info');
end

pause(2);
msgbox('Measuring...','Info');

str = fgets(arduino);
msgbox(str, 'Info');
```

```
str = fgets(arduino);
msgbox(str, 'Info');

handles.newPickup(1) = str2double(fgets(arduino));
handles.newPickup(2) = str2double(fgets(arduino));
handles.newPickup(3) = str2double(fgets(arduino));
handles.newPickup(4) = str2double(fgets(arduino));
handles.newPickup(5) = str2double(fgets(arduino));

handles.currentPickup_2 = handles.newPickup;
set(handles.text_current_pickup_2,'String', 'New Pickup');

pickup_msg = {['Rs = ' num2str(handles.currentPickup_2(1))]  ...
                    ['Lp = ' num2str(handles.currentPickup_2(2))], ...
                    ['Cp = ' num2str(handles.currentPickup_2(3))], ...
                    ['Rpp = ' num2str(handles.currentPickup_2(4))], ...
                    ['Vol = ' num2str(handles.currentPickup_2(5))]};

msgbox(pickup_msg, 'Pickup');
guidata(hObject,handles);

fclose(arduino);
else
    errordlg('Set the circuit parameters first!','Error');
end

% --- Executes on button press in push_save_pickup.
function push_save_pickup_2_Callback(hObject, eventdata, handles)
% hObject    handle to push_save_pickup (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

 if strcmp(get(handles.text_current_pickup_2,'String'),('New Pickup'))
     name = inputdlg('Enter pickup Name:','Save Pickup',[1 30]);
     name = name{1}; %Get the string of the cell
     filename = strcat(name,'.pickup');
     csvwrite(filename,handles.currentPickup_2);
     set(handles.text_current_pickup_2, 'String',name);
 else
     errordlg('You have not measured a new Pickup!','Error');
 end
  guidata(hObject, handles);

% --- Executes on button press in push_load_pickup.
function push_load_pickup_2_Callback(hObject, eventdata, handles)
% hObject    handle to push_load_pickup (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
name = inputdlg('Enter pickup Name:','Load Pickup',[1 30]);
name = name{1}; %Get the string of the cell

filename = strcat(name, '.pickup');
handles.currentPickup_2 = csvread(filename);

set(handles.text_current_pickup_2,'String', name);
pickup_msg = {['Rs = ' num2str(handles.currentPickup_2(1))],...
                    ['Lp = ' num2str(handles.currentPickup_2(2))],...
                    ['Cp = ' num2str(handles.currentPickup_2(3))],...
```

```
                          ['Rpp = ' num2str(handles.currentPickup_2(4))],...
                          ['Vol = ' num2str(handles.currentPickup_2(5))]};

msgbox(pickup_msg, 'Pickup');
guidata(hObject,handles);

% --- Executes on button press in push_save_circuit.
function push_save_circuit_2_Callback(hObject, eventdata, handles)
% hObject    handle to push_save_circuit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
 if strcmp(get(handles.text_current_pickup_2,'String'),'None') == 0
     if handles.Rt_2 > 0 && handles.Rv_2 > 0 && handles.Ct_2 > 0
name = inputdlg('Enter Circuit Name:','Save Circuit',[1 30]);
     name = name{1}; %Get the string of the cell
     filename = strcat(name,'.circ');

fid = fopen(filename,'wt+');
pickup_name = get(handles.text_current_pickup_2,'String');
fprintf(fid, '%s\n',pickup_name);
fprintf(fid, '%s\n',num2str(handles.Rv_2));
fprintf(fid, '%s\n',num2str(handles.x_2));
fprintf(fid, '%s\n',num2str(handles.Rt_2));
fprintf(fid, '%s\n',num2str(handles.y_2));
fprintf(fid, '%s\n',num2str(handles.Ct_2));
fclose(fid);
guidata(hObject,handles);
    else
    errordlg('Set the circuit parameters first!','Error');
    end

 else
     errordlg('You have not selected a Pickup!','Error');
 end

% --- Executes on button press in push_load_circuit.
function push_load_circuit_2_Callback(hObject, eventdata, handles)
% hObject    handle to push_load_circuit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
name_circ = inputdlg('Enter Circuit Name:','Load Circuit',[1 30]);
name_circ = name_circ{1}; %Get the string of the cell

filename = strcat(name_circ, '.circ');

fid = fopen(filename,'rt+');

name_pickup = strcat(fgets(fid));
file_pickup = strcat(name_pickup,'.pickup');
handles.currentPickup_2 = csvread(file_pickup);

handles.Rv_2 = str2double(fgets(fid));
handles.x_2 = str2double(fgets(fid));
handles.Rt_2 = str2double(fgets(fid));
handles.y_2 = str2double(fgets(fid));
handles.Ct_2 = str2double(fgets(fid));

set(handles.text_current_pickup_2,'String', name_pickup);
set(handles.edit_vol_2,'String',num2str(handles.Rv_2 / 1000));
set(handles.edit_tone_2,'String',num2str(handles.Rt_2 / 1000));
set(handles.edit_tone_cap_2,'String',num2str(handles.Ct_2*1e9));
set(handles.slider_vol_2,'Value',handles.x_2);
set(handles.slider_tone_2,'Value',handles.y_2);
set(handles.text_slider_vol_2,'String',num2str(handles.x_2));
set(handles.text_slider_tone_2,'String',num2str(handles.y_2));

pickup_msg = {['Rs = ' num2str(handles.currentPickup_2(1))],...
                    ['Lp = ' num2str(handles.currentPickup_2(2))],...
                    ['Cp = ' num2str(handles.currentPickup_2(3))],...
                    ['Rpp = ' num2str(handles.currentPickup_2(4))],...
                    ['Vol = ' num2str(handles.currentPickup_2(5))]};

msgbox(pickup_msg, 'Pickup');
guidata(hObject,handles);

% --- Executes on button press in push_add.
function push_add_2_Callback(hObject, eventdata, handles)
% hObject    handle to push_add (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if strcmp(get(handles.text_current_pickup_2,'String'),'None') == 0
    figure(1);
    this_tf = transfer_function_2(handles);
    w = linspace(0,2*pi*20000,2000);
    P = bodeoptions;
    P.FreqUnits = 'Hz';
    bode(this_tf,w,P);
    Rv_actual = handles.Rv_2 * handles.x_2/ 1000;
    Rt_actual = handles.Rt_2 * handles.y_2/ 1000;
    Ct_actual = handles.Ct_2 * 1e6;
    msg = ['Pickup: ' get(handles.text_current_pickup_2,'String')...
           ' Rv: ' num2str(Rv_actual) 'kOhms' ...
           ' Rt: ' num2str(Rt_actual) 'kOhms' ...
           ' Ct: ' num2str(Ct_actual) 'nF'];
    ch = get(gca,'Children');
    set(ch(1),'DisplayName',msg);
    legend('-DynamicLegend');
    hold all
 else
     errordlg('You have not selected a Pickup!','Error');
 end
   guidata(hObject, handles);

% --- Executes on button press in push_mix.
function push_mix_2_Callback(hObject, eventdata, handles)
% hObject    handle to push_mix (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if strcmp(get(handles.text_current_pickup_2,'String'),'None') == 0
    if strcmp(get(handles.text_current_record,'String'),('None')) ==0
        if handles.Rt_2 > 0 && handles.Rv_2 > 0 && handles.Ct_2 > 0
           %Current Circuit Filtering
           H = transfer_function_2(handles);
           Z = c2d(H,1/44100);
           b = (Z.num{1});
           a = (Z.den{1});
           handles.currentMix_2 = filter(b,a,handles.currentRecord);
           set(handles.text_current_mix_2,'String', 'New Mix');
        else
            errordlg('Set the circuit parameters first!','Error');

        end
    else
     errordlg('You have not recorded a new Record!','Error');
    end
else
     errordlg('You have not selected a Pickup!','Error');
end
   guidata(hObject, handles);

% --- Executes on button press in push_save_mix.
function push_save_mix_2_Callback(hObject, eventdata, handles)
% hObject    handle to push_save_mix (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% --- Executes on button press in push_load_mix.
if strcmp(get(handles.text_current_mix_2,'String'),('New Mix'))
     name = inputdlg('Enter Mix Name:','Save Mix',[1 30]);
     name = name{1}; %Get the string of the cell

     %First audio file is saved
     filename = strcat(name,'.mix');
     csvwrite(filename,handles.currentMix_2);
     set(handles.text_current_mix_2, 'String',name);

     %Then circuit file is saved
     filename = strcat(name,'.circ');
     fid = fopen(filename,'wt+');
     pickup_name = get(handles.text_current_pickup_2,'String');
     fprintf(fid, '%s\n',pickup_name);
     fprintf(fid, '%s\n',num2str(handles.Rv_2));
     fprintf(fid, '%s\n',num2str(handles.x_2));
     fprintf(fid, '%s\n',num2str(handles.Rt_2));
     fprintf(fid, '%s\n',num2str(handles.y_2));
     fprintf(fid, '%s\n',num2str(handles.Ct_2));
     fclose(fid);
 else
     errordlg('You have not created a new Mix!','Error');
 end
    guidata(hObject, handles);

function push_load_mix_2_Callback(hObject, eventdata, handles)
% hObject    handle to push_load_mix (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
name = inputdlg('Enter Mix Name:','Load Mix',[1 30]);
name = name{1}; %Get the string of the cell

%First Audio file is loaded
filename = strcat(name, '.mix');
handles.currentMix_2 = csvread(filename);
set(handles.text_current_mix_2,'String', name);

%Then Circuit file is loaded
filename = strcat(name, '.circ');
```

```
fid = fopen(filename,'rt+');

name_pickup = strcat(fgets(fid));
file_pickup = strcat(name_pickup,'.pickup');
handles.currentPickup_2 = csvread(file_pickup);

handles.Rv_2 = str2double(fgets(fid));
handles.x_2 = str2double(fgets(fid));
handles.Rt_2 = str2double(fgets(fid));
handles.y_2 = str2double(fgets(fid));
handles.Ct_2 = str2double(fgets(fid));

set(handles.text_current_pickup_2,'String', name_pickup);
set(handles.edit_vol_2,'String',num2str(handles.Rv_2 / 1000));
set(handles.edit_tone_2,'String',num2str(handles.Rt_2 / 1000));
set(handles.edit_tone_cap_2,'String',num2str(handles.Ct_2*1e9));
set(handles.slider_vol_2,'Value',handles.x_2);
set(handles.slider_tone_2,'Value',handles.y_2);
set(handles.text_slider_vol_2,'String',num2str(handles.x_2));
set(handles.text_slider_tone_2,'String',num2str(handles.y_2));

pickup_msg = {['Rs = ' num2str(handles.currentPickup_2(1))],...
                    ['Lp = ' num2str(handles.currentPickup_2(2))],...
                    ['Cp = ' num2str(handles.currentPickup_2(3))],...
                    ['Rpp = ' num2str(handles.currentPickup_2(4))],...
                    ['Vol = ' num2str(handles.currentPickup_2(5))]};

msgbox(pickup_msg, 'Pickup');

guidata(hObject,handles);

% --- Executes on button press in push_listen_mix.
function push_listen_mix_2_Callback(hObject, eventdata, handles)
% hObject    handle to push_listen_mix (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if strcmp(get(handles.text_current_mix_2,'String'),'None') == 0
 p = audioplayer(handles.currentMix_2,44100);
 playblocking(p);
else
     errordlg('You have not selected a Mix!','Error');
 end
   guidata(hObject, handles);

function H = transfer_function_2(handles)

Rp = handles.currentPickup_2(1);
Lp = handles.currentPickup_2(2);
Cp = handles.currentPickup_2(3) * 1e-12;
Rpp= handles.currentPickup_2(4);
Vol= handles.currentPickup_2(5);
Rt = handles.Rt_2;
Rv = handles.Rv_2;
Ct = handles.Ct_2;
Cc = 0;
x = handles.x_2;
y = handles.y_2;
```

```
%The Pickup amplitude is normalized with the reference of 50mV the
%circuitry effect is taken into account in the arduino code
A = Vol*5/1024/50e-3*pi/24;
n1 = A*x*Rv*Rpp*Rt*y*Ct;
n0 = A*x*Rv*Rpp;

d4 = (1-x)*x*Rv^2*Rpp*y*Rt*Cp*Ct*Cc*Lp;
d3 = Rpp*x*Rv*y*Rt*Ct*Cc*Lp + Rpp*Rv*y*Rt*Ct*Cp*Lp + (1-
x)*Rv*(x*Rv*Cc*Lp*(Rpp*Cp+y*Rp*Ct+Rpp*Ct) + Rpp*Rt*y*Cp*Ct*x*Rv*Rp*Cc);
d2 = Rpp*x*Rv*y*Rt*Ct*Cc + Rpp*x*Rv*Cc*Lp + Rpp*y*Rt*Ct*Lp +
Rpp*Rp*y*Rt*Rv*Cp*Ct + Rv*Lp*(Rpp*Cp+y*Rp*Ct+Rpp*Ct) + (1-
x)*x*Rv^2*Rp*Cc*(Rpp*Cp+y*Rp*Ct+Rpp*Ct) + (1-x)*x*Rv^2*Cc*Lp + (1-
x)*x*Rv^2*Rpp*y*Rt*Ct*Cc;
d1 = Rpp*Rt*y*Ct + Rp*Lp + Rpp*x*Rv*Rp*Cc +
Rv*Rp*(Rpp*Cp+y*Rp*Ct+Rpp*Ct) + Rv*Lp + (1-x)*x*Rv^2*Rp*Cc +
Rpp*Rv*y*Rt*Ct + (1-x)*x*Rv^2*Rpp*Cc;
d0 = Rp*Rpp + Rv*Rp + Rv*Rpp;

H = tf([n1 n0],[d4 d3 d2 d1 d0]);

%The choosing between Guitar 1 and Guitar 2 is taken into account when
%recording, in the push_record function

% --- Executes on selection change in popup_guitar.
function popup_guitar_Callback(hObject, eventdata, handles)
% hObject    handle to popup_guitar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes during object creation, after setting all properties.
function popup_guitar_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popup_guitar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

# Annex 6: Application User's Guide

Instructions to Measure and Simulate with the Guitar Simulator application. It is highly recommended to use this guide the first time the simulator is being used.

**Measuring a Pickup's parameters:**

1. Choose the value of the Tone and Volume potentiometers. It has to be a number with maximum three decimals and its units are kΩ.
2. Make sure the Arduino USB cable is connected to the computer and the jack from the circuit is connected to your guitar.
3. Your guitar's knobs must be set at the maximum value.
4. Click on 'Start Measure' and follow the instructions
5. When it is finished, 'New Pickup' will appear in the 'Current Pickup' field. Push 'Save Pickup' to save it with the name you want.
6. You can load any pickup measured pushing the button 'Load Pickup'

**Saving a Circuit:**

1. Choose a value for Tone capacitor, Tone potentiometer and Volume potentiometer.
2. Choose a pickup or measure one and save it first.
3. Click on 'Save Circuit' and save it with the name you want.
4. You can load any saved circuit pushing the button 'Load Circuit'.

**Recording an audio clip:**

1. Set a Guitar Panel with the configuration you have in the guitar is going to be recorded. If you have not measured the guitar's pickup do it.
2. Be sure that the panel selection above the 'Record' button points to the proper panel ('Guitar 1' or 'Guitar 2')
3. Select the duration of the recording in the 'Seconds' box. The maximum time is 20 seconds in this application version.
4. When the recording is finished, 'New Record' will appear in the 'Current Record' field. Push 'Save Record' to save it with the name you prefer.
5. You can load any saved record pushing the button 'Load Record'.

## Mixing and Saving a Mix

1. Once a circuit has been set or loaded in a Guitar panel and a record has been recorded or loaded, click on ‘Mix’ button.

2. Choose different potentiometer positions using the sliders and the potentiometers type, ‘Linear’ or ‘Log’, below the resistance value box.

3. When a mix has been performed, ‘New Mix’ will appear in the ‘Current Mix’ field. Push ‘Save Mix’ to save the mix and the circuit attached.

4. Push the ‘Listen’ Button to hear the result. Use both panels with different configurations and listen to the difference.

## Glossary

A list of all acronyms and the meaning they stand for.

**VCO**: Voltage Controlled Oscillator.

**DDS**: Direct Digital Synthesis.

**DIP**: Dual In-Line Package.

**IC**: Integrated Circuit.

**DIY**: Do It Yourself.

**OPAMP**: Operational Amplifier.

**GB**: Gain-Bandwidth.

**BJT**: Bi-Junction Transistor.

**GPIO**: General Purpose Input-Output.

**$V_{DD}$**: Supply Voltage.

**ADC**: Analog-to-Digital Converter.

**GND**: Ground.

**GUI**: Graphic User Interface.

**IDE**: Integrated Development Environment.

**FFT**: Fast Fourier Transform.

**DPDT**: Double Pole, Double Throw.