

Ingeniería en Sistemas Audiovisuales
2016-2017

Trabajo Fin de Grado

“Diseño e implementación de un afinador polifónico de guitarra”

Sergio Sánchez Rodríguez

Tutores

David Estévez Fernández

Juan González Víctores

Madrid, Octubre de 2017



Esta obra se encuentra sujeta a la licencia Creative Commons
Reconocimiento – No Comercial – Sin Obra Derivada

Contenido

1.	Introducción	11
1.1	Motivación	11
1.2	Objetivos del trabajo.....	17
1.3	Fases del proyecto	18
1.4	Estructura del documento	20
2.	Estado del arte.....	21
3.	Análisis acústico	27
3.1	Análisis acústico.....	28
3.1.1	Análisis cuerda a cuerda	29
3.1.2	Análisis del rasgueo.....	31
3.1.3	Conclusiones	32
3.2	Método de procesamiento.....	33
3.2.1	Fast Fourier Transform (FFT)	34
4.	Prototipado software	39
4.1	Cálculo de la frecuencia de muestreo mínima	39
4.2	Lectura de audio en tiempo real y decisor monofónico	43
4.2.1	Decisor monofónico	43
4.2.2	Captura del audio	45
4.2.3	Cálculo del tiempo de ejecución	46
4.2.4	Conclusiones	47
4.3	Decisor polifónico	47
4.4	Aumento de la resolución	52
4.4.1	Errores de la FFT.....	52
4.4.2	Interpolación polinómica	54
4.4.3	Asegurar la precisión	57
4.4.4	Calibrado del script Python.....	59

5. Implementación física	61
5.1 Elección del microcontrolador	61
5.2 Circuito de entrada	63
5.2.1 Adaptación del voltaje.....	64
5.2.2 Adaptación de impedancias.....	66
5.3 Programación del afinador	68
5.3.1 Enrutamiento de audio.....	68
5.3.2 Frecuencia de muestreo	69
5.3.3 Adaptación del código	71
5.3.4 Comprobación de la precisión	72
5.3.5 Calibración.....	73
5.3.6 Sistema de representación	74
5.3.7 Sistema de alimentación.....	77
6. Conclusiones	79
6.1 Resultados	79
6.2 Trabajos futuros	80
Anexos	83
1. Marco regulador.....	85
1.1 RoHS.....	85
1.2 ISO 16:1975	85
1.3 Real Decreto 110/2015	86
2. Presupuesto.....	87
3. Esquemático del montaje físico	89
4. Diagrama de Gantt del proyecto	91
5. Fotografías del sistema físico	93
6. English summary	97
Bibliografía	103

Índice de tablas

Tabla 1-1 Nota y frecuencia de cada cuerda de la guitarra.....	17
Tabla 3-1 Frecuencias a detectar para cada cuerda.	33
Tabla 4-1 Frecuencias detectadas por le script mono_tuner.py	42
Tabla 4-2 Límites en hercios de los intervalos de búsqueda para cada cuerda en el caso polifónico según el criterio a priori utilizado por los afinadores comerciales.	49
Tabla 4-3 Extensión de los intervalos de búsqueda en hercios de cada cuerda en formato polifónico.	51
Tabla 4-4 Resultados del script para los audios a la frecuencia de referencia y para los audios a la frecuencia de referencia menos 1 cent.	58
Tabla 4-5 Resultados del script poly_tuner_rt.py al introducir los audios para la calibración.	59
Tabla 5-1 Límites de búsqueda en bins para cada cuerda.....	72
Tabla 5-2 Media y varianza obtenidas para los audios de prueba a la frecuencia de referencia y a la frecuencia de referencia menos 1cent.	73
Tabla 5-3 Definición de los límites en Hz del intervalo de frecuencias en los que se considera afinada una cuerda.	74
Tabla anexa 2-A Costes de los materiales del prototipo.	87
Tabla anexa 2-B Costes de la mano de obra.	88

Índice de figuras

1-1 Teclas del piano con la nota correspondiente a cada una. Al ser un piano extendido, se cuenta con una octava extra, de modo que el La de referencia no es el cuarto. Imagen obtenida de https://sites.google.com/site/proyectodigital2013un/ [Último acceso: 20/9/2017]	11
1-2 Tabla de correspondencias entre las notas en notación clásica (negro) y en notación americana (rojo). Imagen obtenida de http://mrdanilodawson.blogspot.com.es/2015/04/ [Último acceso: 20/9/2017]	12
1-3 Esquema de la distribución de tonos y semitonos entre las notas musicales. Imagen obtenida de https://www.guitarristas.info/foros/leccion-teoria-musical-1-tonos-semitonos-escala-mayor/245549 [Último acceso: 20/9/2017]	12
1-4 Imagen de una guitarra española con las partes identificadas. Imagen obtenida de http://aprendeenlinea.udea.edu.co/lms/moodle/mod/page/view.php?id=105865 [Último acceso: 21/9/2017]	13
1-5 Pastilla de 12 polos o "Humbucker" de una guitarra eléctrica. Imagen obtenida de https://socratic.org/questions/an-acoustic-guitar-uses-a-resonance-cavity-behind-the-strings-to-project-the-sound [Último acceso: 20/9/2017]	14
1-6 Esquema de funcionamiento de una pastilla de guitarra. 1: Imán central. 2: Líneas de campo magnético. 3: Cuerda de guitarra. 4: Bobina de cobre. 5: Circuito de tratamiento de la señal. Imagen obtenida de http://www.explainthatstuff.com/electricguitars.html [Último acceso: 20/9/2017]	15
1-7 Notas correspondientes a cada cuerda de la guitarra. Imagen obtenida de http://yoguitarra.blogspot.com.es/ [Último acceso: 20/9/2017]	15
1-8 Espectro frecuencial de la cuerda 6 (Mi a 82Hz) de una guitarra. La frecuencia fundamental se encuentra marcada con un círculo rojo. El resto de los picos se corresponden con los armónicos.	16
2-1 Diagrama de funcionamiento de un diapasón.	21
2-2 Ejemplo de afinador electrónico monofónico. Modelo Korg TM-50.	22
2-3 Afinador polifónico Polytune 2 de TC Electronic.	23
2-4 La banda de robots Compressorhead.	24
2-5 Izquierda: clavijeros automatizados de la Gibson Robot. Imagen obtenida de: https://www.wired.com/2007/11/self-tuning-rob/ [Último acceso: 23/9/2017] Derecha: Guitarras Gibson Robot. El potenciómetro verde ejerce la función de control del afinador. Imagen obtenida de http://archive.gibson.com/RobotGuitar/sp/story6.html [Último acceso: 23/9/2017].	25
3-1 Forma de onda de la grabación 1	27
3-2 Forma de onda de la grabación 2.	29
3-3 FFT de cada cuerda.	30
3-4 FFT de todas las cuerdas una a una superpuestas.	30

3-5 En línea azul de puntos, la FFT del rasgueo completo. En línea verde, las FFTs de cada cuerda.	31
3-6 Detalle de los armónicos de las cuerdas 1 (derecha) y 2 (izquierda)	32
3-7 Fragmento de una señal con un número entero de periodos.....	35
3-8 Señal con número entero de periodos replicada.....	35
3-9 FFT de la señal con periodos completos.....	35
3-10 Fragmento de señal con un número no entero de periodos.....	36
3-11 Réplica de la señal sin periodos completos.	36
3-12 FFT de la señal sin periodos completos.....	36
3-13 Fragmento de una señal enventanada.....	36
3-14 FFT aplicando la ventana.....	37
3-15 Ventana tipo Hann.	38
4-1 Cejilla o capo puesta en una guitarra (Imagen obtenida de https://www.thaliacapos.com/ [Último acceso: 20/9/2017])	40
4-2 FFT en Python del rasgueo de guitarra.....	41
4-3 Izquierda: funcionamiento del script mono_tuner_rt.py. Cada línea muestra, de izquierda a derecha, la frecuencia detectada, la cuerda asociada y la indicación sobre su afinación. Derecha: imagen del funcionamiento del Polytune Clip. Indica únicamente si la cuerda está demasiado grave, demasiado aguda o correctamente.	44
4-4 Ejecución del script mono_tuner_rt.py. Además de la información anterior, muestra el tiempo de ejecución de las operaciones de procesamiento.	47
4-5 Izquierda: Imagen del afinador Fender California. Derecha: imagen del afinador Yamaha YTC5.	48
4-6 Representación del espectro frecuencial del rasgueo de guitarra. Los valores marcados corresponden, de izquierda a derecha, a las cuerdas 6, 5, 4 y 3 y a un armónico de nivel notable.....	50
4-7 Espectro en frecuencia del rasgueo de guitarra en escala no logarítmica. Los puntos corresponden, de izquierda a derecha, a un armónico de nivel notable, a la frecuencia de referencia de la cuerda 2 y a la frecuencia de referencia de la cuerda 1.	50
4-8 FFT de una onda senoidal de frecuencia 82Hz. La FFT marca un pico en el bin 82 y los valores adyacentes resultan prácticamente despreciables.	52
4-9 FFT de un seno de 82.41 Hz. Se puede observar la dispersión producida al no coincidir la frecuencia con el valor exacto de un bin.	53
4-10 En azul, FFT de un seno a 82.41 Hz. En rojo la interpolación cuadrática basada en los 3 valores más altos. La marca corresponde al punto más alto de la estimación cuadrática.	55
4-11 Valores frecuenciales obtenidos por la estimación polinómica para las frecuencias de 81.5 a 82.5. Eje vertical: número de muestras. Eje horizontal: frecuencia calculada.....	56
5-1 En azul, ejemplo de una señal de guitarra eléctrica con una amplitud de 350mV. En rojo, los límites operativos de la placa de audio Teensy.....	64

5-2 En negro, el valor de continua de 1.65V que ha de sumarse a la señal. En verde, la señal una vez añadido ese valor y adaptada para el funcionamiento de la placa Teensy.....	65
5-3 Divisor de voltaje.....	65
5-4 Divisor de voltaje con filtro paso alto.....	66
5-5 Divisor de voltaje resistivo.....	67
5-6 Etapa de adaptación de impedancias	67
5-7 Esquema del enrutamiento de audio en la placa Teensy.....	69
5-8 Pantalla LCD mostrando el layout fijo.	75
5-9 Arriba izquierda: símbolo 1. Arriba derecha: símbolo 2. Abajo izquierda: símbolo 3. Abajo centro: símbolo 4. Abajo derecha: símbolo 5.....	76
5-A Prototipo. Montaje completo.	93
5-B Conector JACK 6,3mm estéreo de entrada.	93
5-C Etapa de adaptación de la señal de la guitarra para su lectura en la placa Teensy.	94
5-D Placa Teensy 3.2 montada sobre la Teensy Audio Board.	94
5-E Conversor lógico I2C usado para conectar la pantalla LCD a la placaz Teensy.	95
5-F Conexionado del LCD a la placa Teensy. Cable amarillo: SCL. Cable Azul: SDA.....	95
5-G Conversor de voltajes de 9V a 5V para el sistema de alimentación del LCD y la placa Teensy.	96

Agradecimientos

Me gustaría agradecer a mi familia todo el esfuerzo que han realizado por mi durante todos estos años, ya que sin ellos no habría podido llegar hasta aquí.

1. Introducción

Los instrumentos musicales son herramientas que necesitan estar perfectamente afinadas para poder desarrollar su función.

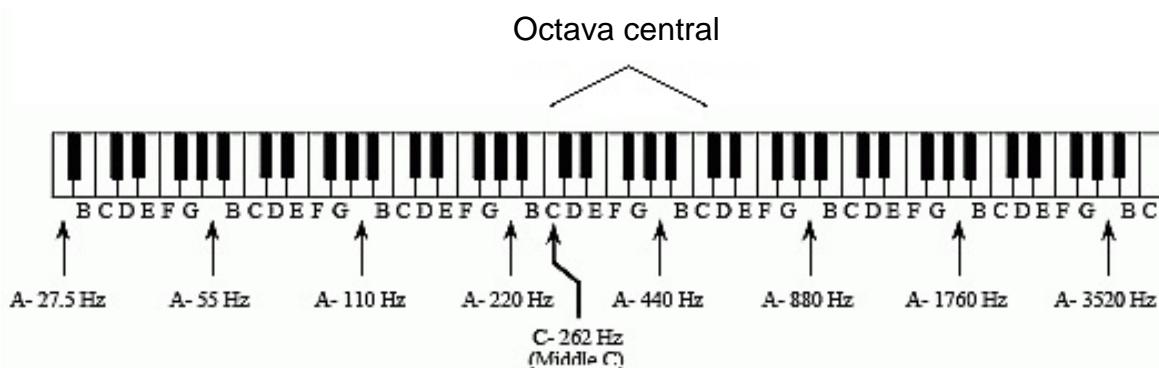
Para comprender cómo se afinan los instrumentos en este apartado se explica la teoría musical necesaria, para comprender cómo se construyen los intervalos entre notas y la referencia usada.

Como el proyecto se orienta a un afinador de guitarra se explican también los detalles de construcción y funcionamiento relacionados con la teoría musical explicada y con la generación de la señal acústica usada por el afinador. Estas características serán las que marquen ciertas decisiones con respecto al diseño del afinador e influirán en su funcionamiento.

También se explican las fases del desarrollo del proyecto y los objetivos que se intentan conseguir.

1.1 Motivación

La frecuencia de referencia más común utilizada por los instrumentos y la escala musical está definida en el estándar ISO 16^[1], que establece que la frecuencia de la nota *La₄* (Nota *La* de la octava del piano que comienza con el *Do* central, es decir, la cuarta octava, según el índice musical científico) sea de 440 Hz.



1-1 Teclas del piano con la nota correspondiente a cada una. Al ser un piano extendido, se cuenta con una octava extra, de modo que el *La* de referencia no es el cuarto. Imagen obtenida de <https://sites.google.com/site/proyectodigital2013un/> [Último acceso: 20/9/2017]

En la música occidental popular, partir de esa nota se fijan las frecuencias de las demás siguiendo lo que se conoce como *escala temperada*. Esta escala divide una octava en doce notas que cumplen que la distancia en términos logarítmicos entre dos notas consecutivas es igual, siendo la octava el intervalo creado entre dos notas cuyas frecuencias están separadas por una relación 2:1.

Do Re Mi Fa Sol La Si Do
C D E F G A B C

1-2 Tabla de correspondencias entre las notas en notación clásica (negro) y en notación americana (rojo).
Imagen obtenida de <http://mrdanilodawson.blogspot.com.es/2015/04/> [Último acceso: 20/9/2017]

Es decir, si La₄ son 440Hz, la nota que formaría una octava con ella sería La₅, cuya frecuencia sería 440Hz * 2 = 880Hz.

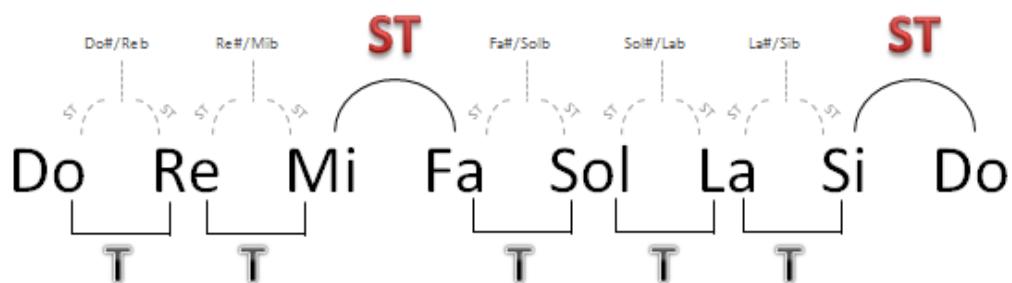
Por tanto, al dividir la octava en 12 porciones logarítmicas, si la frecuencia de una nota la denotamos como F_n , y la de la nota siguiente como F_{n+1} se obtiene la relación entre ellas como se presenta en la siguiente ecuación:

$$F_{n+1} = F_n \cdot 2^{(1/12)}$$

Por ejemplo, la nota siguiente a La₄, llamada La#₄, tendría una frecuencia igual a:

$$440\text{Hz} \cdot 2^{(\frac{1}{12})} = 466,16\text{Hz}$$

Esta distancia o intervalo entre dos notas contiguas se conoce también como *semitono*, siendo un *tono* la distancia equivalente a 2 semitonos.



1-3 Esquema de la distribución de tonos y semitonos entre las notas musicales. Imagen obtenida de <https://www.guitarristas.info/foros/leccion-teoria-musical-1-tonos-semitonos-escala-mayor/245549> [Último acceso: 20/9/2017]

Cada semitono, además, se suele dividir en 100 unidades (*centésimas de tono*) que siguen también una relación logarítmica de modo que, siendo n el número de centésimas entre dos frecuencias f_1 y f_2 :

$$f_2 = f_1 \cdot 2^{\frac{n}{1200}}$$

Siguiendo esta relación se pueden obtener las frecuencias de las notas utilizadas por cada instrumento para comparar las frecuencias medidas por el afinador. Por ejemplo, un afinador de guitarra dará seis referencias diferentes, una para cada cuerda. Esta relación logarítmica provoca también que los intervalos tengan tamaños menores cuanto menor sea la frecuencia a la que están.

La razón por la que se divide en intervalos logarítmicos tiene que ver con la percepción del sonido que permite el oído humano. Este, por su estructura interna, aporta la capacidad de distinguir mejor las variaciones en frecuencia cuanto más graves sean esas frecuencias. Es decir, se podría diferenciar entre 70 y 71 Hz, pero en cambio entre 10.000 y 10.001 Hz sería imposible detectar diferencias. Tal y como se ha construido la escala logarítmica de intervalos se puede tomar como referencia de la resolución del oído unos 5 cents, independientemente de la frecuencia. Es decir, a cualquier frecuencia el oído humano puede diferenciar dos tonos separados 5 cents de tono.

La guitarra aplica esta teoría musical para su construcción.

Una guitarra es un instrumento que utiliza generalmente seis cuerdas de diferente grosor para generar el sonido.



1-4 Imagen de una guitarra española con las partes identificadas. Imagen obtenida de <http://aprendeenlinea.udea.edu.co/lms/moodle/mod/page/view.php?id=105865> [Último acceso: 21/9/2017]

Estas cuerdas están distribuidas longitudinalmente sobre el *mástil* o *diapasón* y sobre parte del *cuerpo* de la guitarra (llamado *caja de resonancia* en la imagen 1-4). Cada una dispone de dos puntos de apoyo constantes, el *puente*, situado en el cuerpo y la *cejuela*, situada en el mástil. Además, el mástil dispone de puntos de apoyo llamados *trastes* que permiten al pulsar una cuerda variar su frecuencia siguiendo la relación comentada anteriormente en este apartado (factor $2^{1/12}$ entre traste y traste) para generar las notas de la escala musical, haciendo que sea más aguda cuanto más cerca del cuerpo se pulse.

Las guitarras acústicas tienen un cuerpo hueco con un agujero situado bajo las cuerdas, de modo que el sonido rebota en el interior de la guitarra amplificándolo. Dependiendo del diseño las cuerdas pueden ser metálicas o de nylon, pasando en este último caso a denominarse como “guitarra española”.

Las electroacústicas son similares a las acústicas en cuanto al cuerpo, pero incorporan debajo del puente un receptor piezoelectrónico que transforma las vibraciones de las cuerdas en señal eléctrica. Esta señal tiene un nivel muy bajo y una impedancia muy alta, lo que hace necesario la presencia de un preamplificador electrónico cuya función es la de adaptar esa señal para ser recibida por un amplificador de guitarra convencional. Este preamplificador por lo general incorpora control de ecualización y de volumen y ofrece una salida por conexión JACK de 6,3mm.



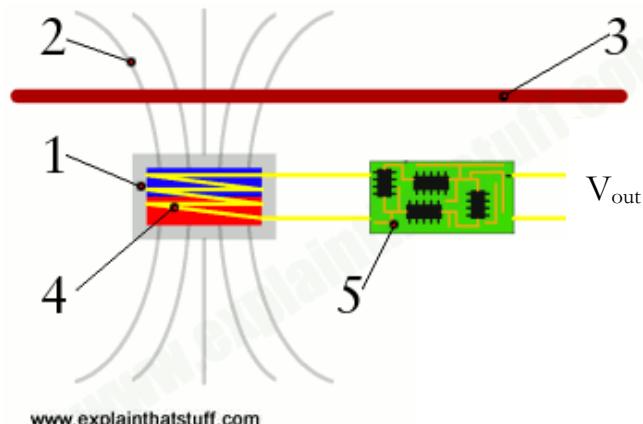
1-5 Pastilla de 12 polos o "Humbucker" de una guitarra eléctrica. Imagen obtenida de <https://socratic.org/questions/an-acoustic-guitar-uses-a-resonance-cavity-behind-the-strings-to-project-the-sound> [Último acceso: 20/9/2017]

Las guitarras eléctricas, al contrario que los otros dos tipos, tienen un cuerpo sólido o, en caso de tener cavidades acústicas, no son cavidades destinadas a aumentar el volumen acústico sino a la ecualización del sonido. Para transmitir el sonido de las cuerdas, este tipo de guitarras utiliza uno o varios dispositivos denominados *pastillas* ubicados en el cuerpo debajo de las cuerdas.

Estas pastillas están formadas por una bobina de cobre y uno o dos imanes por cuerda que generan un campo magnético estático no uniforme alrededor de ella. La cuerda, por ser metálica, al vibrar produce una modificación de ese campo magnético con una frecuencia igual a la frecuencia de oscilación.

Esa variación del campo provoca el fenómeno conocido como *inducción electromagnética* en la bobina, creando una corriente eléctrica con la frecuencia de oscilación de la cuerda.

Esta señal, después de pasar por un circuito que puede ser pasivo o activo y que permite controlar el volumen y ciertos parámetros de ecualización, sale de la guitarra por un cable JACK de 6,3mm.



1-6 Esquema de funcionamiento de una pastilla de guitarra. 1: Imán central. 2: Líneas de campo magnético. 3: Cuerda de guitarra. 4: Bobina de cobre. 5: Circuito de tratamiento de la señal. Imagen obtenida de <http://www.explainthatstuff.com/electricguitars.html> [Último acceso: 20/9/2017]

La frecuencia *fundamental* (f_1) de vibración de cada cuerda depende de la densidad lineal (μ) y longitud (L) de la cuerda y de la tensión (T) que se le ha aplicado, relacionadas según la siguiente ecuación:

$$f_1 = \frac{1}{2L} \sqrt{\frac{T}{\mu}}$$

Esta frecuencia se denomina *fundamental* porque, al ser la frecuencia más grave producida por la cuerda, es la que se usa como indicador de la nota a la que está sonando esa cuerda. Esta característica causa que sea una frecuencia importante de cara a la detección de la frecuencia de trabajo de la cuerda.



1-7 Notas correspondientes a cada cuerda de la guitarra. Imagen obtenida de <http://yoguitarra.blogspot.com.es/> [Último acceso: 20/9/2017]

Como en una guitarra por construcción la longitud y la densidad lineal de la cuerda son fijas, la frecuencia de la cuerda se ajusta modificando la tensión. La tensión de cada cuerda se controla mediante una *clavija* situada, por lo general, en el mástil, a la que va enrollada y que, mediante un tornillo sin fin, provoca que la cuerda se enrolle más, aumentando la tensión y haciendo el sonido más agudo, o se suelte, disminuyendo la tensión y haciendo el sonido más grave.

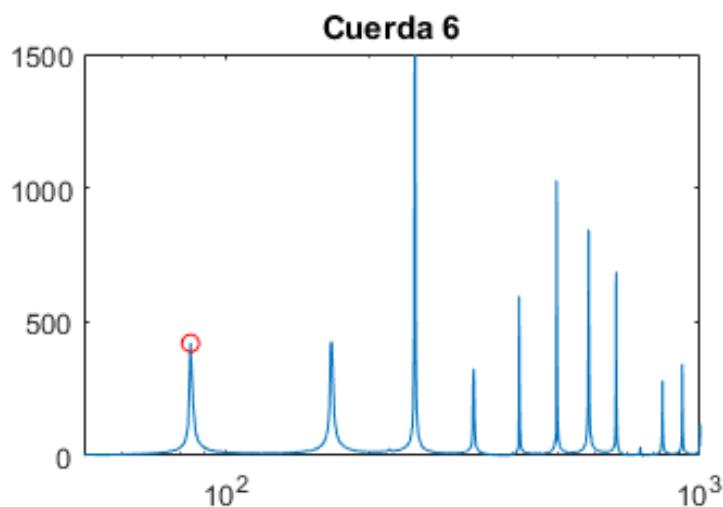
Cada cuerda, además de la frecuencia fundamental genera infinitas frecuencias diferentes conocidas como *armónicos* y que siguen la siguiente expresión:

$$f_n = \frac{n}{2L} \sqrt{\frac{T}{\mu}} = n \cdot f_1$$

Como se deduce de la ecuación de los armónicos, la frecuencia de éstos depende únicamente de la frecuencia fundamental de la cuerda.

La amplitud de cada uno de los armónicos es mucho más difícil de calcular porque depende de innumerables factores como pueden ser la densidad de la guitarra, los nudos de la madera, de la cuerda, la temperatura, y el circuito eléctrico por el que pase la señal entre otros. Esta cantidad de factores hacen que cada instrumento modifique la amplitud de esos armónicos de una manera diferente. Esta cualidad del sonido nos permite diferenciar no sólo una guitarra de otra, sino también una guitarra de un piano o de un trombón y se conoce como *timbre* musical.

La guitarra se puede afinar de muchas maneras posibles en función de la nota que se asigne a cada cuerda, pero la afinación más común es la *estándar*, que asigna las notas por cuerda especificadas en la Tabla 1-1.



1-8 Espectro frecuencial de la cuerda 6 (Mi a 82Hz) de una guitarra. La frecuencia fundamental se encuentra marcada con un círculo rojo. El resto de los picos se corresponden con los armónicos.

Durante este documento, para facilitar las referencias y por ser una denominación más común, se hace referencia a las cuerdas por su número o por la nota asignada en la notación inglesa salvo la cuerda 1, que para evitar la confusión con la cuerda 6 pasará a denominarse “e” o “he” (high E en inglés, donde *high* se refiere a la altura musical por ser la cuerda con la nota Mi más aguda).

Dado que las referencias usadas son muy precisas y la guitarra es un instrumento bastante inexacto por construcción, es necesaria la utilización de afinadores que implementen esas referencias de algún modo para calibrarla.

Tabla 1-1 Nota y frecuencia de cada cuerda de la guitarra.

Número de cuerda	Nota musical (notación clásica)	Nota musical (notación inglesa)	Frecuencia (Hz)
1	Mi ₄	E	329,63
2	Si ₃	B	246,94
3	Sol ₃	G	196,00
4	Re ₃	D	146,83
5	La ₂	A	110,00
6	Mi ₂	E	82,41

1.2 Objetivos del trabajo

El objetivo principal del trabajo es la realización de un prototipo totalmente funcional de un afinador polifónico de guitarra *open source*.

Todo afinador electrónico tiene que desempeñar tres funciones:

- Captar el sonido del instrumento. Ya sea mediante un micrófono (para instrumentos acústicos), una entrada de línea de audio (para instrumentos eléctricos) o un receptor piezoelectrónico (convierte la vibración del instrumento en una señal eléctrica. Se usa en instrumentos tanto acústicos como eléctricos).
- Procesar el sonido. Para obtener la *frecuencia fundamental* del sonido captado, con el fin de compararla con las referencias necesarias.
- Mostrar la diferencia entre la referencia y el sonido captado. Normalmente se usan pantallas LCD para los modelos más básicos y conjuntos de LEDs o galvanómetros para los modelos pensados para su uso profesional.

Por tanto, el afinador polifónico diseñado deberá cumplir con esas exigencias.

Al ser un prototipo no es requisito indispensable que tenga encapsulado propio. Por lo tanto, el sistema de visualización no será en principio similar al de un

afinador convencional utilizando hardware específicamente diseñado para ello. En su lugar se usará una pantalla LCD genérica.

En cuanto al método de captación del audio, se hará por una entrada de línea de audio JACK de 6,3mm con el fin de usarse en instrumentos eléctricos principalmente. Esto permite evitar ruidos e interferencias como podrían ocurrir al usar un micrófono. En principio este sistema podría funcionar también con una pinza piezoelectrica que contara con una conexión JACK de 6,3mm.

Para que el sistema sea válido para usarse como afinador será necesario que tenga cierta precisión detectando el sonido que haga que al afinar la cuerda el instrumento suene correctamente. Observando las especificaciones técnicas de algunos modelos del mercado como el Polytune 3^[2], el Korg TM-50TR^[3] y el Boss TU-3^[4], se puede observar que la precisión mínima que se necesitará es de 1 centésima de tono si bien el Polytune demuestra que se puede superar ofreciendo 0.5 centésimas de tono en el modo polifónico. Por tanto, el objetivo a cumplir será conseguir una precisión de una centésima de tono como mínimo.

Aunque las versiones comerciales de los afinadores polifónicos llevan cierto tiempo en el mercado (desde 2010), se encuentran pocas referencias a su funcionamiento debido a que los fabricantes guardan en secreto el código, componentes y esquemáticos. Además, la búsqueda de versiones libres de derechos o patentes de este tipo de afinadores no arroja resultados.

Por tanto, para hacer más accesibles este tipo de productos se pretende que el diseño cumpla ciertos requerimientos relacionados con el *open source* y que sea apto para realizarse por usuarios que no tienen por qué ser expertos en electrónica o audio, formando parte del movimiento *Do It Yourself (Hazlo tú mismo)*. Por estos motivos se intentará usar componentes lo más genéricos posible y se hará público el código usado y los esquemáticos necesarios para que cualquiera pueda tener acceso a ello.

Para resaltar esta faceta DIY se espera que en un futuro se pueda incorporar el afinador a los talleres que realiza en la UC3M¹ la asociación UC3Music² para enseñar a los estudiantes conceptos tanto de audio como de electrónica de una manera práctica.

1.3 Fases del proyecto

El diseño del afinador polifónico ha constado de varios pasos orientados a reducir el problema a unidades más sencillas que puedan tratarse de manera separada. De este modo se pudieron aislar los problemas que ocurrieron en cada fase y se

¹ <https://www.uc3m.es/Inicio> [Último acceso: 20/9/2017]

² <http://music.uc3m.es/> [Último acceso: 20/9/2017]

pudieron solucionar de manera más sencilla al poder identificar los procesos que los causaron.

Las fases que se realizaron fueron:

- **Análisis acústico.** Conocer las características del audio a tratar es crucial para poder seleccionar la mejor forma de tratarlo. En esta fase se realizó un estudio y se escogió el algoritmo de procesamiento y se intentó fijar el modo de detectar las frecuencias de cada cuerda en un espectro completo.
- **Prototipado software.** Para conocer el funcionamiento del algoritmo en un caso real de uso sin tener problemas relacionados con el hardware se utilizó el lenguaje Python³. Se realizó en dos pasos:
 - **Afinador monofónico:** este primer paso estuvo orientado a la comprobación del funcionamiento del algoritmo seleccionado en un caso de detección de una sola frecuencia. También se comprobó la viabilidad del sistema en tiempo real ya que, si un PC no es capaz de ejecutar el algoritmo seleccionado, no hay apenas microcontroladores en el rango de precios objetivo capaces de hacerlo.
 - **Afinador polifónico:** una vez asegurado el funcionamiento del algoritmo y su capacidad para realizarse en tiempo real, se probó el funcionamiento de la detección de tonos en un ámbito polifónico. En este paso se pusieron a prueba los resultados del análisis acústico realizado anteriormente. Además, se buscaron modos de aumentar la resolución del sistema que pudieran ser válidos no sólo en Python, sino también en un microcontrolador.
- **Desarrollo hardware.** Una vez comprobado que el algoritmo escogido era viable y se podía conseguir la suficiente resolución se implementó el programa en un microcontrolador. Se le dotó además de un sistema de representación con el fin de conseguir un sistema completamente independiente y funcional.

Durante la realización del diseño se trabajó utilizando la plataforma GitHub⁴. Todos los archivos utilizados durante el proceso y mencionados en esta memoria se encuentran disponibles en el repositorio PolyTuna⁵ de la asociación UC3Music.

³ <https://www.python.org/> [Último acceso: 20/9/2017]

⁴ <https://github.com/> [Último acceso: 20/9/2017]

⁵ <https://github.com/UC3Music/PolyTuna> [Último acceso: 20/9/2017]

1.4 Estructura del documento

Esta sección presenta una vista general del documento con el objetivo de que le sea más sencillo al lector la localización de los puntos que sean más de su interés.

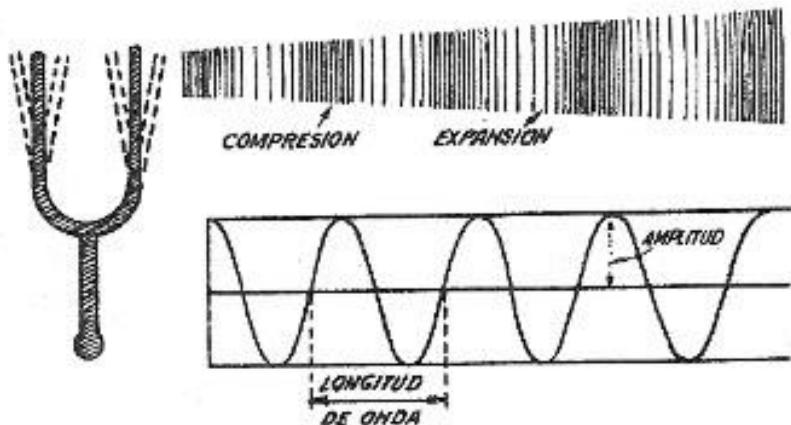
- **Capítulo 3.** Este capítulo se centra en la realización del análisis acústico de la señal de guitarra. Aquí se muestran los resultados de los análisis cuerda a cuerda y del rasgueo y se escoge el método de procesamiento a usar por el afinador, explicando brevemente las características de la FFT (*Fast Fourier Transform*).
- **Capítulo 4.** En este capítulo se realiza el prototipado *software* del afinador. Muestra el estudio para elegir la mínima frecuencia de muestreo que se puede utilizar para el afinador y cómo se implementa un afinador que funciona usando audios grabados. También se explica cómo se consiguió la lectura en tiempo real, el decisor polifónico y cómo se aumenta la precisión para conseguir resultados válidos para afinar.
- **Capítulo 5.** Aquí se describe la implementación física del afinador basándose en el *software* final del capítulo 3. Se comenta cómo se consigue leer en un microcontrolador la señal de la guitarra, explicando el circuito necesario. Además, se explica cómo se adaptó el código al microcontrolador y las modificaciones que se debieron realizar en las librerías para su funcionamiento. También se incluye en este capítulo los datos relacionados con el sistema de alimentación y de visualización final del afinador.
- **Capítulo 6.** En este apartado se muestran datos sobre el funcionamiento del afinador en un caso de uso real y se proponen futuras mejoras para el sistema.
- **Anexos.** En los anexos se incluyen esquemáticos completos del montaje, así como el presupuesto del prototipo, el marco regulador que debería seguir el producto final para poder comercializarse, el desarrollo temporal inicial del proyecto e imágenes del sistema final montado. El resumen en inglés se incluye también como el último anexo del documento.

2. Estado del arte

En este capítulo se estudia la evolución histórica de los sistemas de afinación y su estado actual con el fin de encontrar sistemas similares de los que tomar ejemplo para solucionar las dificultades que se presentan.

Hasta mediados del siglo XX la afinación debía realizarse usando una herramienta denominada diapasón.

Esta herramienta en apariencia sencilla se construye de forma que al golpearlo las vibraciones producidas generan una resonancia a una frecuencia de referencia muy concreta. Usando esa referencia los músicos deben, escuchando su instrumento musical a la vez, afinar este último hasta que los dos sonidos tengan la misma frecuencia. En el caso de los instrumentos de cuerda, a partir de la cuerda que se afina con el diapasón se deben afinar las siguientes aprovechando las relaciones que haya entre las frecuencias de cada una.



2-1 Diagrama de funcionamiento de un diapasón.

Esta operación requiere de un ambiente bastante silencioso para poder comparar ambos sonidos. El diapasón debe estar en perfectas condiciones, ya que cualquier pequeña variación física puede variar la frecuencia emitida y por lo general, cada diapasón sólo permite afinar directamente una cuerda. El resto deben afinarse después. Además, requiere de cierto entrenamiento del oído musical, lo cual supone un problema especialmente a músicos principiantes.

Si bien en ambientes más clásicos como en algunas orquestas y para afinar los pianos se sigue usando el diapasón, su uso general es prácticamente inexistente en comparación con los afinadores digitales.

Un afinador es un dispositivo electrónico que indica la diferencia entre la nota tocada por un instrumento y una nota de referencia predefinida, permitiendo ajustar la *altura* musical o frecuencia de trabajo de dicho instrumento.

Los afinadores digitales monofónicos simplificaron enormemente la tarea de afinación. En la guitarra, por ejemplo, basta con tocar una cuerda sin pulsarla para que el afinador detecte de qué cuerda se trata e informe, mediante una pantalla, si la cuerda pulsada es más grave (menor frecuencia), más aguda (mayor frecuencia) o igual que la frecuencia de referencia para esa cuerda. Realizando esta acción con cada una de las cuerdas se puede afinar la guitarra entera sin necesidad de compararlas a oído, eliminando la necesidad de tener un oído musical entrenado.



2-2 Ejemplo de afinador electrónico monofónico. Modelo Korg TM-50.

Los afinadores monofónicos implementaron también varias posibilidades de captura de la señal aparte de la recepción acústica por micrófono: la recepción eléctrica mediante un cable y la recepción de vibraciones mediante una pinza piezoelectrónica. Estas dos opciones permiten la afinación más sencilla de instrumentos eléctricos, ya que, al no tener caja de resonancia, el volumen producido es únicamente el de la cuerda y no se puede oír fácilmente.

Además, el hecho de no tener que escuchar el instrumento para afinar permite una reacción mucho más rápida en entornos de directo donde el tiempo tiene un valor muy alto y el nivel de ruido suele ser muy elevado constantemente.



2-3 Afinador polifónico Polytune 2 de TC Electronic.

Los afinadores polifónicos son una mejora reciente del sistema monofónico introducida por la empresa TC Electronic en 2010 con el modelo Polytune.

Con el sistema anterior las cuerdas debían ser afinadas tocándolas una a una lo cual, en ciertos momentos como en un concierto, requería de cierto tiempo y obligaba a hacerlo entre canciones.

El nuevo sistema permite conocer el estado de afinación de todas las cuerdas tocándolas a la vez (acción que se denomina *rasgueo*). Esto permite identificar a simple vista la cuerda o cuerdas desafinadas en unos segundos, permitiendo incluso realizar la afinación durante algún silencio de una canción para asegurar la calidad de la representación.

Esta facilidad y rapidez de uso ha convertido al Polytune en uno de los más usados en el entorno profesional.

Sin embargo, el alto precio de venta (111€⁶) hace difícil que muchos músicos que, si bien no son demasiado conocidos, también realizan actuaciones en directo, puedan acceder a comprarlo.

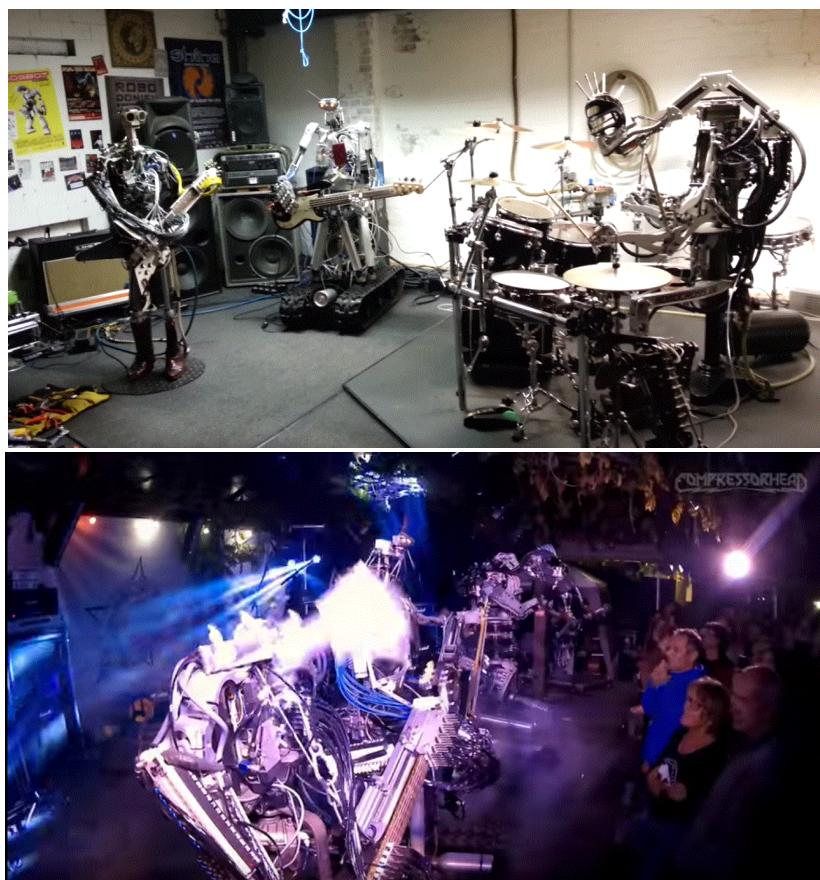
⁶ Precio en la tienda online Thomann:

https://www.thomann.de/es/tc_electronic_polytune_3_tuner_buffer.htm [Último acceso: 20/9/2017]

Normalmente este tipo de músicos suelen recurrir a copias o aprovechar los proyectos Open Source y Do It Yourself para conseguir aparatos con un funcionamiento similar y considerablemente más baratos.

Sin embargo, tras una búsqueda por la web se puede comprobar que no existen proyectos enfocados a la construcción física del afinador, si bien se pueden encontrar algunos proyectos software más o menos funcionales. Este proyecto de afinador intentará encajar en ese espacio que de momento se encuentra vacío.

El punto de partida es complejo dado que no hay información sobre los algoritmos usados por los afinadores comerciales, ya que las patentes que usan se centran sólo en los encapsulados^[5], ya que son mucho más sencillos de copiar que el software.



2-4 La banda de robots Compressorhead.⁷

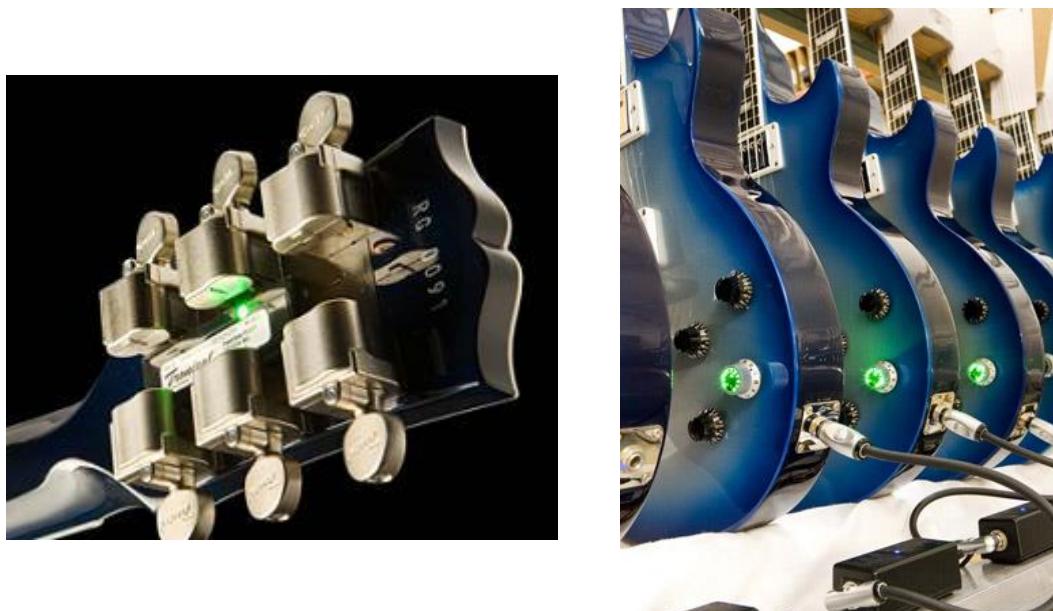
Las investigaciones y patentes se centran más bien en aplicar la afinación monofónica a sistemas automáticos. Por ejemplo, el artículo *Robot: Tune Yourself! Automatic Tuning in Musical Robotics*^[6] comenta cómo aplicar técnicas de afinación monofónica a “robots musicales”, es decir, a sistemas autónomos

⁷ Imagen superior obtenida de https://www.youtube.com/watch?v=3RBSkq_St8 y la inferior de https://www.youtube.com/watch?v=9gMX_hR-RoM. [Último acceso de ambos videos: 23/9/2017]

que producen música ejecutando movimientos programados previamente, como los mostrados en la imagen 2-4.

Este sistema se centra en la comparación de las frecuencias emitidas por la cuerda con las frecuencias de referencia que se han grabado con anterioridad para esa posición, resultando en un algoritmo bastante sencillo de comparación de frecuencias realizando una FFT.

Uno de estos sistemas, pero orientado a instrumentos tocados por humanos, que además funcionaba tocando todas las cuerdas a la vez, fue el utilizado por las guitarras de una serie limitada emitida en 2008 llamada Gibson Robot^[7] de la compañía Gibson Guitar Corporation⁸. Este sistema, si bien funcionaba tocando todas las cuerdas a la vez, no se puede considerar realmente polifónico. Esto es debido a que la guitarra debía fabricarse con ese sistema integrado ya que contaba con una pastilla piezoeléctrica para cada cuerda. De este modo se separaba el problema polifónico a 6 problemas monofónicos mucho más sencillos de resolver, evitando interferencias entre las propias cuerdas.



2-5 Izquierda: clavijeros automatizados de la Gibson Robot. Imagen obtenida de: <https://www.wired.com/2007/11/self-tuning-rob/> [Último acceso: 23/9/2017] Derecha: Guitarras Gibson Robot. El potenciómetro verde ejerce la función de control del afinador. Imagen obtenida de <http://archive.gibson.com/RobotGuitar/sp/story6.html> [Último acceso: 23/9/2017].

Si bien de cara el procesamiento esa idea es muy práctica, el hecho de incluir una pastilla por cuerda lo convertía en un sistema muy caro y además necesitaba estar implementado en el instrumento en el momento de la fabricación, causando que sólo se pudiera aplicar a esas guitarras de serie limitada. De cara a un

⁸ <http://www.gibson.com/> [Último acceso: 23/9/2017]

sistema autónomo que se pueda utilizar con cualquier guitarra no es posible la implementación de este modo por las limitaciones físicas.

Un intento posterior de implementar este sistema en un formato menos invasivo para las guitarras es el Gibson Min-Etune^[8]. Este sistema requería sustituir las clavijas de afinación de la guitarra por unas robotizadas que incorporan receptores piezoelectríficos. Sin embargo, sigue sin ser un sistema polifónico dado que cada cuerda se recibe de forma independiente y, además, sólo está optimizado para detectar correctamente los modelos Gibson Les Paul⁹ y Gibson SG¹⁰, debido también a que sólo es compatible físicamente con estos tipos de guitarras. Esto causa que, si ese algoritmo se usara en guitarras con construcciones diferentes, como una guitarra electroacústica, la detección de frecuencias pueda funcionar.

Este proyecto intenta encajar en el nicho DIY de los afinadores polifónicos que aún se encuentra vacío. Además, el algoritmo abre la puerta a la realización *open source* y DIY de dispositivos de afinación automática como lo dos nombrados de Gibson al hacerlos más baratos al requerir menos materiales.

⁹ <http://www.gibson.com/Productos/Guitarras-El%C3%A9ctricas/Les-Paul.aspx> [Último acceso: 23/9/2017]

¹⁰ <http://www.gibson.com/Productos/Guitarras-El%C3%A9ctricas/SG.aspx> [Último acceso: 23/9/2017]

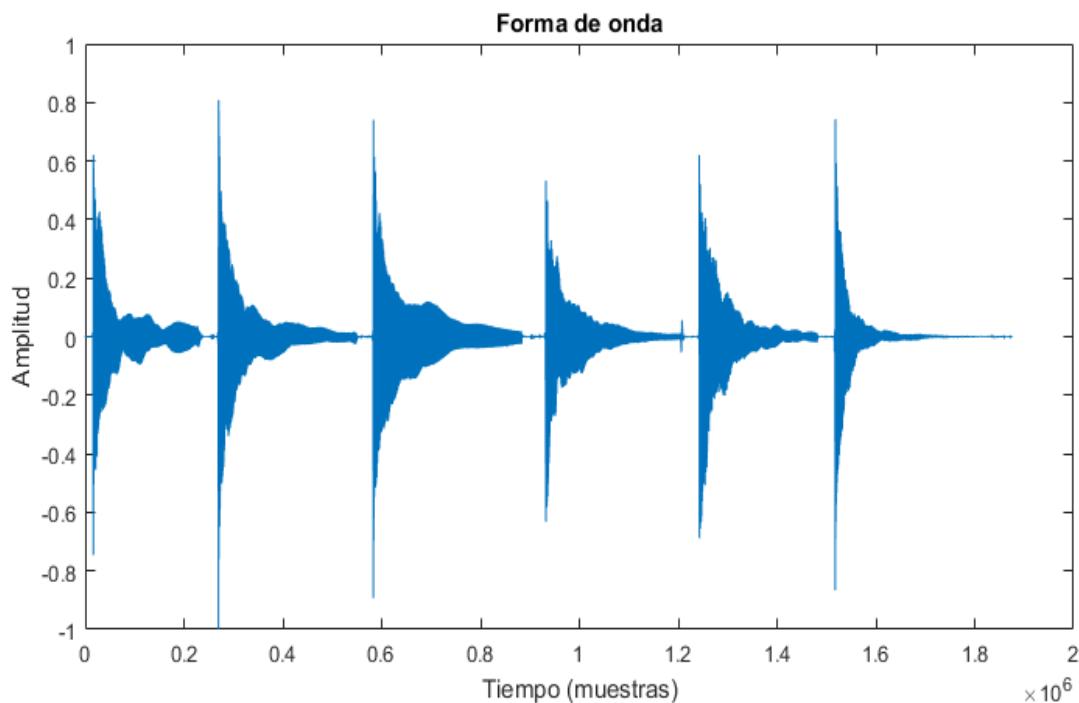
3. Análisis acústico

El proyecto que se va a abordar cuenta con pocos antecedentes que se puedan consultar debido a que sólo existen soluciones comerciales que guardan en secreto los detalles sobre sus diseños. Por tanto, resulta necesario hacer un estudio previo de las necesidades que marcarán su desarrollo.

Para el análisis acústico se usará el software MATLAB 2017¹¹ debido a que a la hora de plantear este apartado aún no se había elegido Python como lenguaje de desarrollo software y se contaba con mucha más experiencia programando en MATLAB.

El código utilizado en este apartado se encuentra en el archivo *mono_tuner.m*.

En este capítulo se analizaron dos audios de guitarra diferentes, uno con todas las cuerdas tocadas una a una y otro con todas tocadas a la vez.



3-1 Forma de onda de la grabación 1

¹¹ <https://es.mathworks.com/> [Último acceso: 20/9/2017]

Para realizar este análisis en el dominio de la frecuencia se usó la *Fast Fourier Transform* (FFT) debido a que ofrece mucha información de una manera bastante precisa. Usando esta información se eligió el algoritmo de cálculo de frecuencias más apropiado.

3.1 Análisis acústico

El primer paso realizado es un análisis del sonido producido por una guitarra con el fin de conocer sus características. De esta forma se podrá adaptar la detección de los tonos del afinador y conseguir que sea más precisa.

Para obtener muestras sobre las que trabajar se ha grabado el sonido de una guitarra tocada con púa y conectada mediante un cable JACK de 6,3mm a una interfaz de audio Focusrite Scarlett 2i4¹² usando el software Ableton Live 9¹³. No se han usado efectos de ningún tipo más que los controles de ganancia de la interfaz para asegurar que la señal grabada es lo más parecida posible a la que recibiría el afinador en condiciones de trabajo reales. La guitarra utilizada, una Gibson Songwriter Studio electroacústica, se ha afinado en afinación estándar (EADGBE) usando como referencia un afinador Korg GA-30¹⁴. Las grabaciones se han exportado con una profundidad de 16 bits y muestreadas a una frecuencia de 44100Hz.

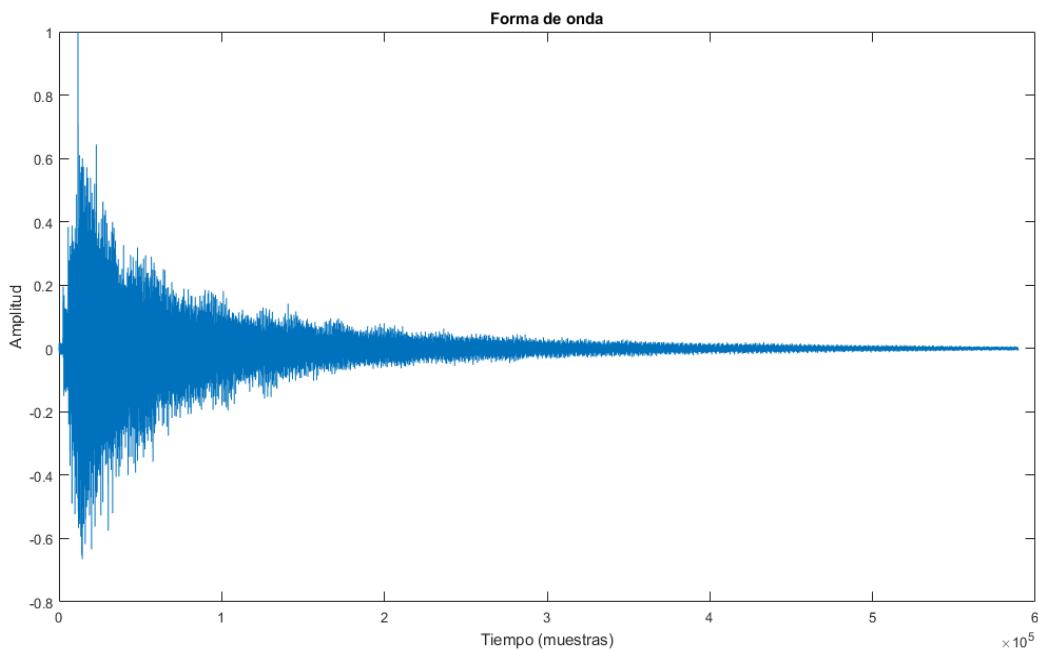
Se han realizado dos grabaciones:

- La primera consiste en las seis cuerdas tocadas una a una, manteniendo cada cuerda sonando al menos 3 segundos y sin superposición de sonidos entre cuerdas diferentes. Se muestra su forma de onda en la imagen 3-1.
- La segunda se corresponde a un *rasgueo*, es decir, a todas las cuerdas tocadas lo más al unísono posible comenzando a tocar desde la cuerda más grave (E) hasta la más aguda (h). Se produce superposición de sonidos entre todas ellas y se dejan sonar durante 10 segundos. Se muestra su forma de onda en la imagen 3-2.

¹² <https://us.focusrite.com/usb-audio-interfaces/scarlett-2i4> [Último acceso: 20/9/2017]

¹³ <https://www.ableton.com/en/> [Último acceso: 20/9/2017]

¹⁴ <http://www.korg.com/us/products/tuners/> (no se puede acceder a la página exacta del producto por estar descatalogado) [Último acceso: 20/9/2017]



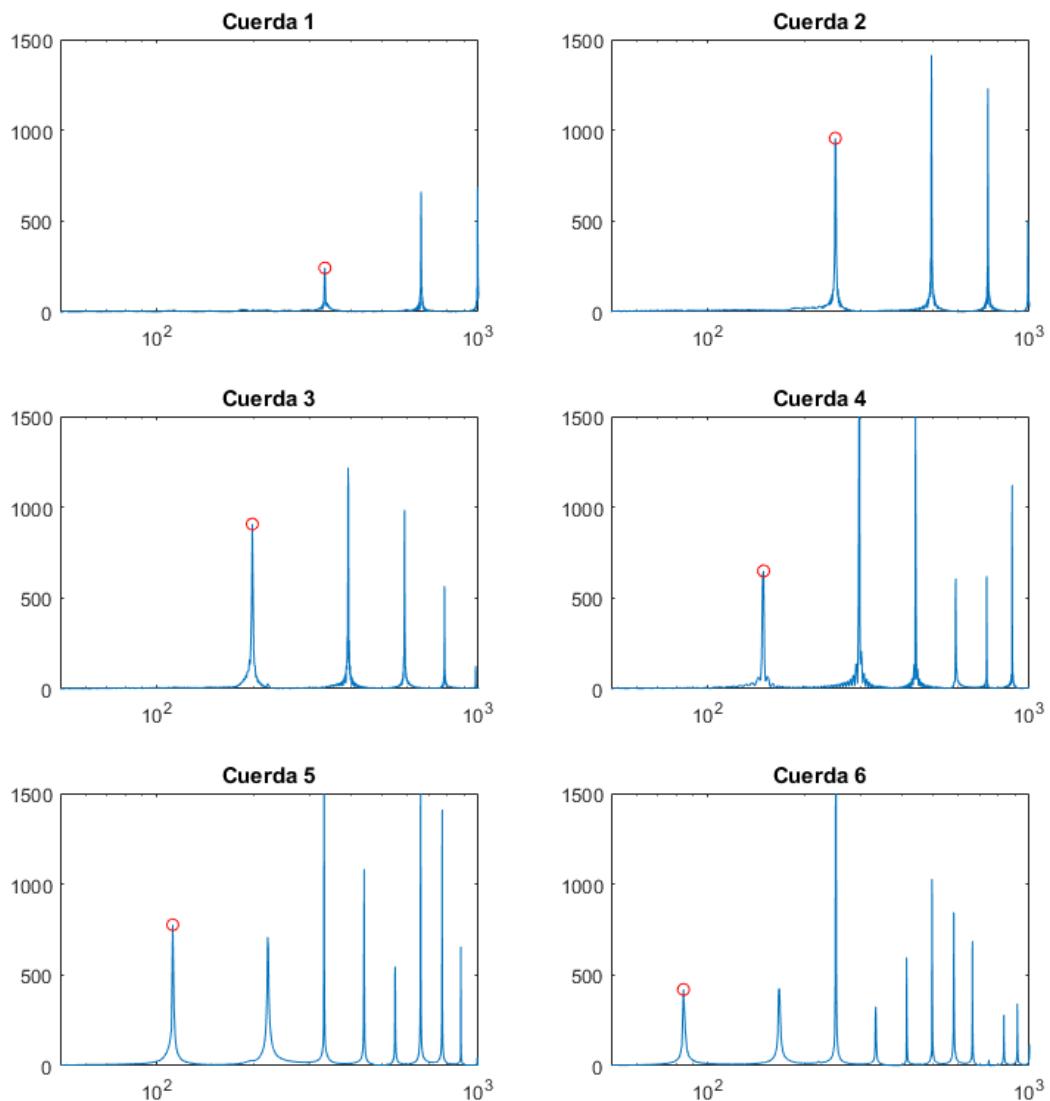
3-2 Forma de onda de la grabación 2.

3.1.1 Análisis cuerda a cuerda

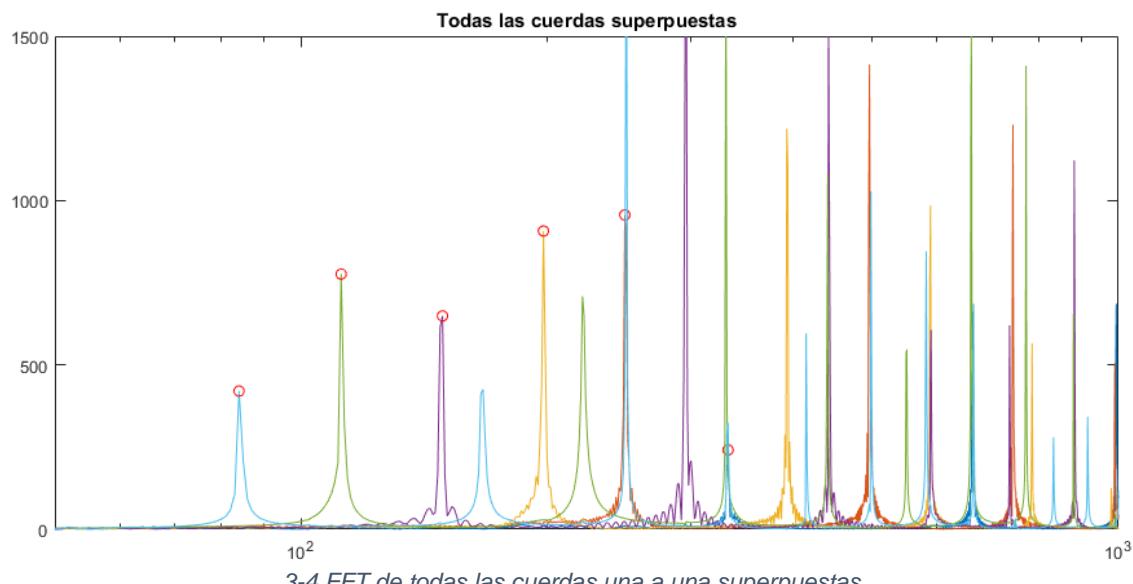
Para conocer el espectro frecuencial de cada cuerda de guitarra se aplica la *Fast Fourier Transform* (FFT) al fragmento de audio de la primera grabación correspondiente a cada una de las cuerdas, obteniendo el resultado mostrado en la figura 3-3.

Como se puede ver en las gráficas la frecuencia fundamental, marcada con un círculo rojo, es claramente diferenciable del resto de armónicos en cada cuerda. Sin embargo, también se aprecia cómo algunos armónicos tienen una amplitud mayor que la frecuencia fundamental de todas las cuerdas.

Esta característica de los armónicos puede afectar bastante a la detección de los tonos, ya que como se puede observar en la figura 3-4, al superponer las seis cuerdas, en las frecuencias fundamentales de las cuerdas 1 y 2 se pueden observar armónicos de las cuerdas 6 y 5 respectivamente con amplitudes superiores a las de la frecuencia fundamental de dichas cuerdas. Estando separadas las 6 cuerdas resulta sencillo detectar cuál es la frecuencia de cada cuerda, pero al tocarse todas a la vez pueden surgir problemas de enmascaramiento de las frecuencias entre las distintas cuerdas. Para comprobar si eso ocurre realmente se debe realizar un análisis del rasgueo completo.



3-3 FFT de cada cuerda.



3-4 FFT de todas las cuerdas una a una superpuestas.

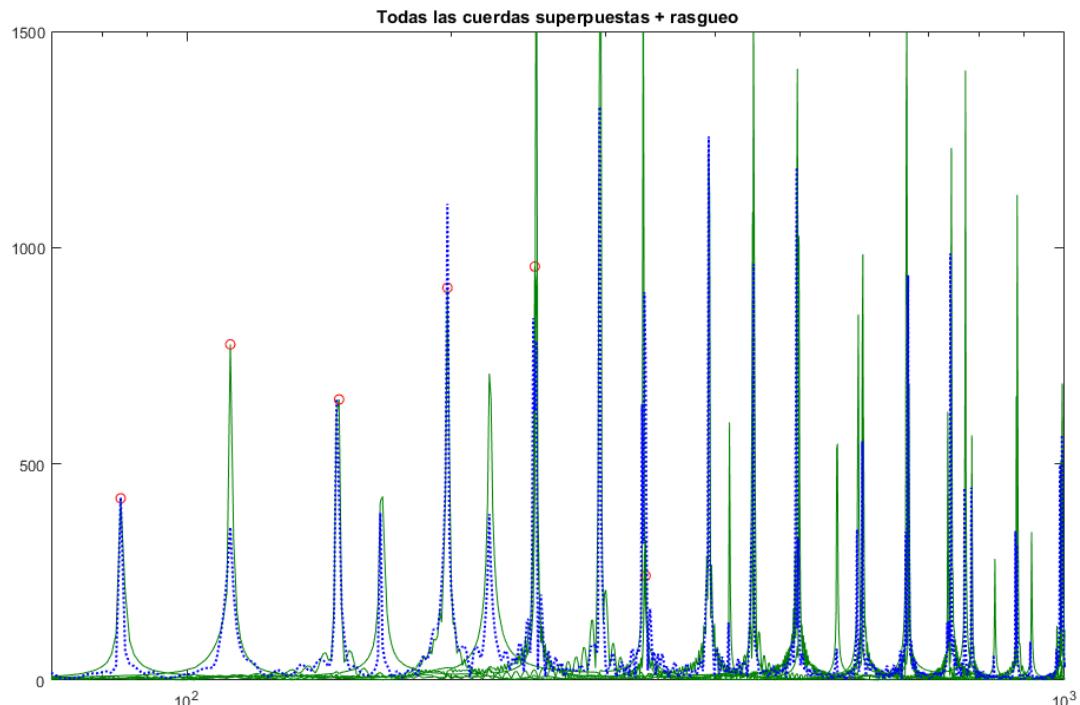
3.1.2 Análisis del rasgueo

Si bien en el análisis cuerda a cuerda se ve cómo se solapan algunas frecuencias fundamentales con armónicos, en el caso del rasgueo se puede observar cómo afecta ese solapamiento al espectro conjunto de todas las cuerdas.

Como se puede ver en la figura 3-5, la amplitud que se produce al rasguear en la frecuencia fundamental de la cuerda 1 (329 Hz) es mucho mayor a la que genera la propia cuerda tocada individualmente. Este hecho nos puede llevar a deducir que esa cuerda será enmascarada por el armónico y que por tanto la detección de un pico a esa frecuencia producirá problemas porque nos dará más información sobre el armónico que produce la cuerda 5 (observable en la imagen 3-4) que sobre la cuerda 1.

En la frecuencia correspondiente a la fundamental de la cuerda 2 (246Hz), al rasguear todas las cuerdas a la vez se atenúa el armónico causado por otras cuerdas, causando que la amplitud total en esa frecuencia al tocar todas las cuerdas sea similar a la que causaría sólo la cuerda 2 tocada por separado. Esto lleva a pensar que el hecho de rasguear afecta a las vibraciones a esa frecuencia. Como es probable que afecte igual a todas las fuentes que produzcan esa frecuencia (a todas las cuerdas), la cuerda 2 tampoco ofrecerá más información que las que causen armónicos en ese punto porque, como se puede observar en la figura 3-4, otras cuerdas producen más amplitud ahí.

Por tanto, tampoco es una frecuencia que nos ofrezca demasiada información sobre la afinación.

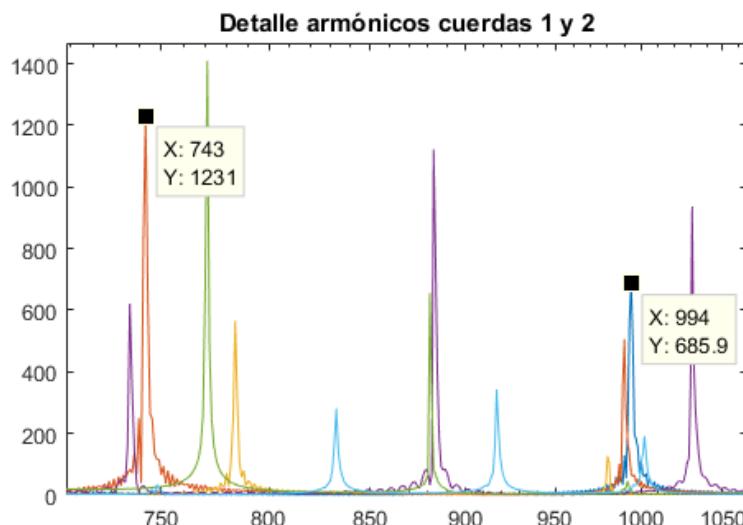


3-5 En línea azul de puntos, la FFT del rasgueo completo. En línea verde, las FFTs de cada cuerda.

3.1.3 Conclusiones

Si bien las cuatro cuerdas más graves (cuerdas 3, 4, 5 y 6) parecen que se pueden detectar satisfactoriamente buscando su frecuencia fundamental, con las cuerdas 1 y 2 no ocurre lo mismo.

Aunque las frecuencias fundamentales de esas cuerdas son enmascaradas sus armónicos nos siguen dando información sobre la nota original de la cuerda, ya que su frecuencia depende directamente de la frecuencia original. Por este motivo se ha buscado un punto donde algún armónico de cada una de estas dos cuerdas cause una amplitud mayor que los armónicos del resto de cuerdas que puedan estar presentes en ese punto.



3-6 Detalle de los armónicos de las cuerdas 1 (derecha) y 2 (izquierda)

La inexactitud provocada por la vibración de las cuerdas de la guitarra, que causa variaciones más grandes a altas frecuencias, provoca que los armónicos puedan estar desplazados en la gráfica. Por tanto, se calculan sus frecuencias teóricas eligiendo el armónico más cercano a las frecuencias medidas.

Para la cuerda 2, por la posición se puede deducir que se trata del tercer armónico ($n=3$):

$$f_3 = n * f_1 = 246,94 * 3 = 740.82 \text{ Hz}$$

Y para la cuerda 1, por proximidad, también podría tratarse del tercer armónico:

$$f_3 = n * f_1 = 329,63 * 3 = 988.89 \text{ Hz}$$

De este modo se han obtenido unos valores que a priori parecen más fiables de cara a la detección, causando que las frecuencias a detectar para cada cuerda sean las indicadas en la siguiente tabla:

Tabla 3-1 Frecuencias a detectar para cada cuerda.

Número de cuerda	Frecuencia fundamental (Hz)	Frecuencia a detectar (Hz)
1	329,63	988,89
2	246,94	740,82
3	196,00	196,00
4	146,83	146,83
5	110,00	110,00
6	82,41	82,41

3.2 Método de procesamiento

Como el objetivo del proyecto es desarrollar un prototipo funcional en formato portátil la complejidad del cálculo debería ser la menor posible para alcanzar la resolución necesaria. Esto se debe a que los procesadores potentes aumentan el precio del sistema y se pretende que sea lo más barato posible siempre y cuando cumpla los requisitos mínimos. Además, también se aumentaría el consumo y el tamaño, afectando a la portabilidad del sistema.

Aunque existen muchos algoritmos diferentes como el *zero crossing*^[9] o el *cross correlation*^[10], la mayoría son sólo válidos para el caso de detectar una frecuencia aislada y menos aún vienen implementados nativamente en un microchip. Esta implementación nativa es importante porque indica que el algoritmo está optimizado para funcionar en ese microprocesador lo cual mejora tanto el rendimiento como el consumo.

El algoritmo que se presenta más comúnmente integrado en los microcontroladores y que además resulta válido para el análisis frecuencial polifónico es la FFT.

Para comprender el comportamiento de la FFT es necesario saber cómo funciona, para lo cual se ha desarrollado el apartado 3.2.1.

3.2.1 Fast Fourier Transform (FFT)

La FFT (*Transformada Rápida de Fourier*) es un algoritmo de cálculo de la Transformada Discreta de Fourier y de su inversa^[11].

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi k \frac{n}{N}} \quad k = 0, \dots N - 1$$

Este algoritmo permite calcular de una manera muy eficiente computacionalmente las frecuencias que componen una señal discreta. Debido a la complejidad del algoritmo y a que no resulta imprescindible conocerlo para la realización del afinador, no se explica el algoritmo completo en esta memoria. Los detalles sobre el proceso más relevantes de cara al desarrollo del afinador y a la comprensión de ciertos resultados sí se explican más adelante en esta misma sección.

A pesar de que el audio es una señal continua, se convierte en discreta al introducirse en cualquier dispositivo digital, como un microcontrolador, mediante un conversor analógico-digital, permitiendo el funcionamiento de este algoritmo.

Este algoritmo se puede implementar de muchas formas, con lo cual, si bien calculan lo mismo, pueden diferir ligeramente en los resultados. Además, la mayoría de los dispositivos modifican el algoritmo para realizarlo de una manera más eficiente con un hardware concreto, aumentando las diferencias. Esto causa que haya variaciones entre diferentes dispositivos o librerías y sea necesario comprobar cómo funcionan los que se van a usar para conocer la mejor forma de interpretarlos.

Además, independientemente de la forma de calcular este algoritmo, surgen otros problemas derivados de su aplicación a señales reales.

Si bien teóricamente para calcular la Transformada de Fourier se debería ejecutar un cálculo infinito, en el mundo real esto no es posible dado que no existen señales infinitas y que se necesitarían infinitos cálculos para realizar el algoritmo.

Para conseguir una ejecución real sobre la señal de entrada $s(t)$ se obliga a que sea finita analizándola en fragmentos del mismo tamaño (al que llamaremos T). Dependiendo del método de ejecución estos fragmentos pueden o no solaparse entre sí, siendo bastante común que se solapen $T/2$.

Para separar la señal de entrada en esos fragmentos se multiplica por una función ventana $h(t)$ que toma valores diferentes de 0 sólo en un intervalo de tamaño T .

La función ventana más simple es la ventana rectangular, cuya expresión se indica en la ecuación siguiente.

$$h(t) \begin{cases} 1 & \text{si } t \in [0, T] \\ 0 & \text{resto} \end{cases}$$

Esta ventana simplemente tomaría un fragmento de la señal sin ponderar de ningún modo los valores. Sin embargo, esta ventana causa problemas derivados de cómo se realiza el algoritmo. Para el cálculo de la FFT se coge el intervalo elegido de la señal de entrada y se replica para conformar una señal más larga de la que sea más sencillo extraer información.

Si en el fragmento escogido la señal tiene un número entero de períodos la unión entre esta y sus réplicas es perfecta.

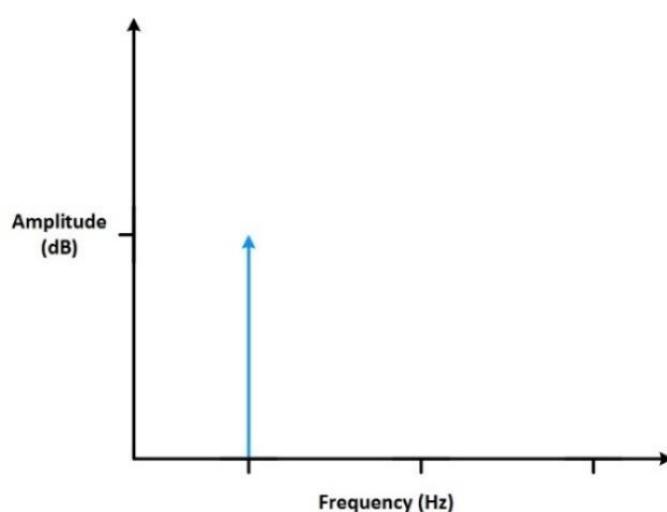


3-7 Fragmento de una señal con un número entero de períodos.¹⁵



3-8 Señal con número entero de períodos replicada.

Si esto se cumple la FFT es capaz de calcular de manera satisfactoria las frecuencias que conforman la señal, como demuestra la imagen 3-9.



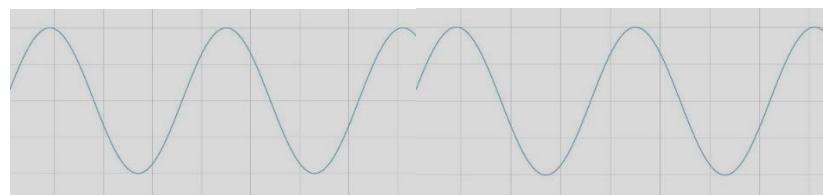
3-9 FFT de la señal con períodos completos.

¹⁵ Las gráficas de este apartado se han obtenido de <http://www.ni.com/white-paper/4844/es/#toc2>. [20/9/2017]

Sin embargo, si no hay un número completo de períodos, se producen discontinuidades que causan la aparición de componentes que no estaban presentes en la señal original y que modifican el resultado de los cálculos.

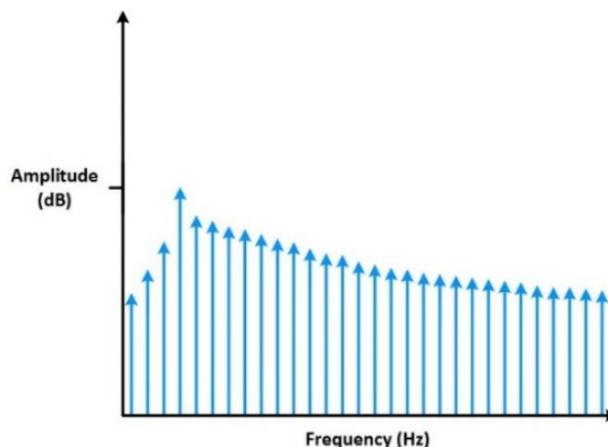


3-10 Fragmento de señal con un número no entero de períodos.



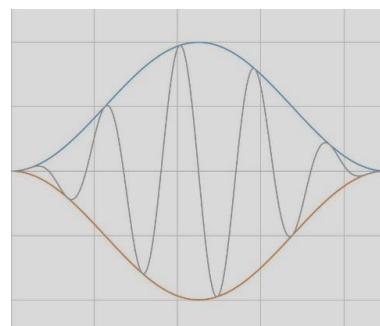
3-11 Réplica de la señal sin períodos completos.

Con estas discontinuidades la FFT quedaría como se indica en la imagen 3-12.



3-12 FFT de la señal sin períodos completos.

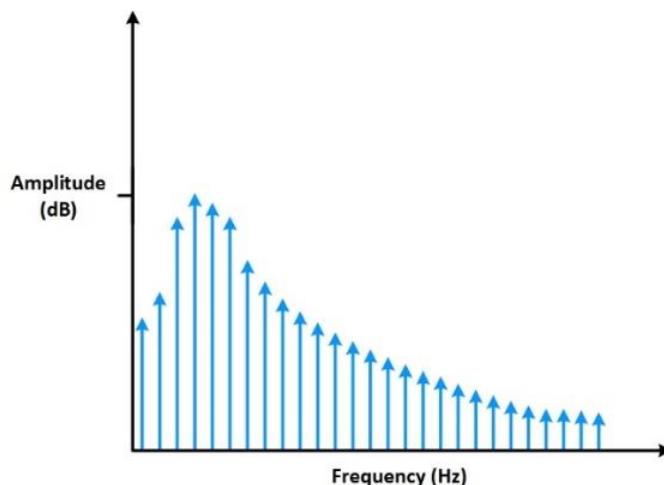
Debido a este efecto, para intentar eliminar las discontinuidades las ventanas usadas en la práctica fuerzan a 0 los valores de los extremos del intervalo.



3-13 Fragmento de una señal enventanada.

De este modo la unión entre dos réplicas siempre será en un punto en el que la función valga 0, causando que no haya discontinuidad.

Con este sistema se consigue paliar hasta cierto punto los errores en la detección de frecuencias, aunque como se muestra en la gráfica 3-14, sigue habiendo cierta dispersión en las frecuencias cercanas:



3-14 FFT aplicando la ventana.

Esto se debe a que la multiplicación en el dominio del tiempo de la señal y la ventana, por las propiedades de la transformada de Fourier, se convierte en la convolución de la transformada de ambas en el espacio frecuencial. Esto causa que la función ventana aplicada afecte al espectro frecuencial de la señal original.

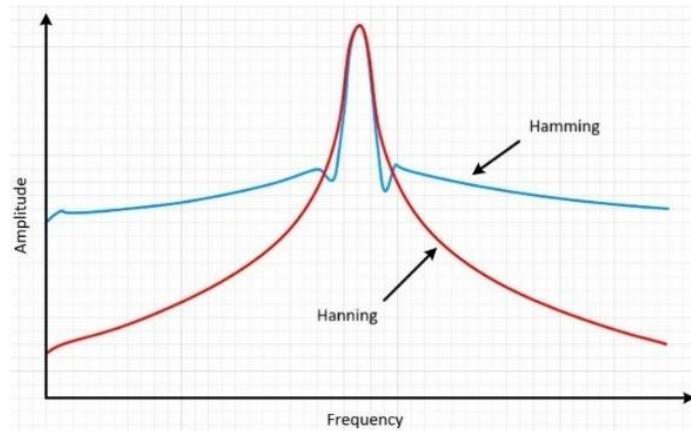
$$s_h(t) = s(t)h(t) \leftrightarrow S_h(f) = S(f) * H(f)$$

Por tanto, en función del destino que se le vaya a dar se necesitará una ecuación de ventana u otra. Como en la señal de guitarra se busca encontrar valores concretos de frecuencia con una resolución lo más alta posible se buscará usar una ventana cuya transformada de Fourier sea estrecha alrededor de la frecuencia central.

La ventana más común para este tipo de aplicaciones es la conocida como *coseno alzado* o *Hann* y sigue la expresión indicada en la ecuación siguiente:

$$v(n) = a_0 - a_1 \cos\left(\frac{2\pi n}{N-1}\right)$$

Esta ventana tiene un lóbulo central relativamente estrecho y atenúa bastante las frecuencias más lejanas. Por tanto, parece apropiada para detectar tonos y será la usada en las FFT del afinador.



3-15 Ventana tipo Hann.

4. Prototipado software

Una vez elegida la FFT como sistema para detectar las frecuencias y antes de comenzar la implementación física, se desarrolló un script en Python 3.5¹⁶ para comprobar la necesidad real de recursos y el funcionamiento del algoritmo. De este modo los problemas que surgieron pudieron ser identificados como problemas propios del algoritmo y se hace más sencillo resolverlos al no haber incertidumbre en cuanto al funcionamiento del hardware.

El principal motivo de usar Python frente a MATLAB como en el apartado del análisis acústico reside en la capacidad de Python para ser ejecutado en cualquier ordenador sin necesidad de contar con un software de pago como MATLAB. De este modo, si el programa es funcional, se podrá utilizar en proyectos futuros para desarrollar una versión *software* del afinador que también entre en el modelo *open source* que se intenta alcanzar con el dispositivo físico.

Para la realización de este prototipo se realizó en primer lugar una comprobación del funcionamiento del algoritmo con frecuencias de muestreo bajas para aumentar la resolución. Una vez completado ese paso se realizó un afinador monofónico que implementa la lectura de audio en tiempo real. Después se realizó el afinador polifónico completo y se diseñó un sistema para aumentar la resolución con una frecuencia de muestreo fija de cara a su implantación en el sistema físico.

4.1 Cálculo de la frecuencia de muestreo mínima

Para asegurar que el algoritmo desarrollado en Python pueda ser funcional en un microcontrolador es necesario adaptar algunos parámetros que lo ajusten a las limitaciones propias del micro, como el tamaño de la FFT. Para ello también resulta necesario comprobar que, bajo esos parámetros, el sistema puede funcionar correctamente, ya que en ocasiones al muestrear a frecuencias demasiado bajas se pueden producir errores en la lectura de frecuencias.

El código Python en el que se implementan las pruebas sobre la frecuencia de muestreo de la FFT se encuentra en el fichero *fft_test.py*.

¹⁶ <https://www.python.org/> [Último acceso: 20/9/2017]

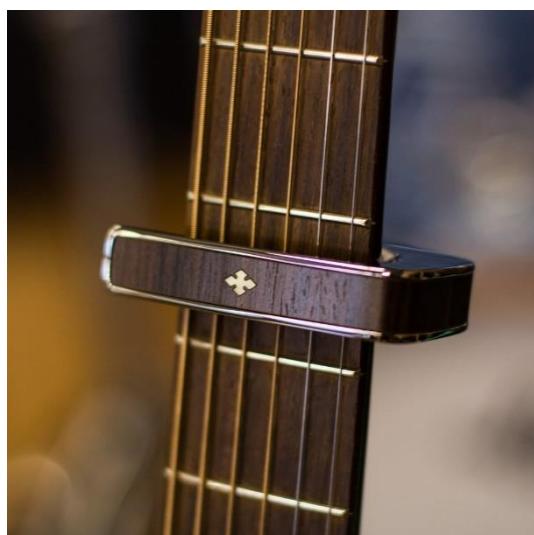
La mayoría de los microcontroladores del mercado ofrecen algoritmos propios para la realización de la FFT. Aunque eso consigue que la FFT se realice mucho más rápido que programándola directamente, también ofrecen resoluciones limitadas. Las resoluciones máximas más comunes son de 512 y de 1024 puntos (o *bins*).

Esto significa que sólo hay 1024 valores frecuenciales posibles para el resultado de la FFT, lo que causa que la resolución frecuencial de la FFT esté fuertemente limitada.

Por poner un ejemplo: la frecuencia de muestreo habitual de los sistemas de audio es de 44100 Hz. Si se realiza la FFT sobre un audio muestreado a esa frecuencia, según el criterio de Nyquist, se obtiene un valor máximo de frecuencias posible de $44100/2 = 22050\text{Hz}$.

El microprocesador debería representar todo el rango de frecuencias posibles, que van desde -22050 hasta 22050 con sólo 1024 puntos en el mejor de los casos. Esto causaría que entre dos puntos contiguos de la FFT hubiera $44100/1024 = 43,07\text{ Hz}$ de diferencia. Esa resolución resulta demasiado baja para un afinador, ya que implicaría por ejemplo que las cuerdas 5 y 6 estuvieran en el mismo *bin*, lo cual no ofrece información sobre la frecuencia de ninguna de las dos cuerdas.

Dado que no se pueden aumentar los puntos de la FFT en sí misma, la solución puede ser modificar la frecuencia de muestreo para que, al ser menor la frecuencia máxima a representar, los *bins* generados estén más juntos, mejorando la resolución.



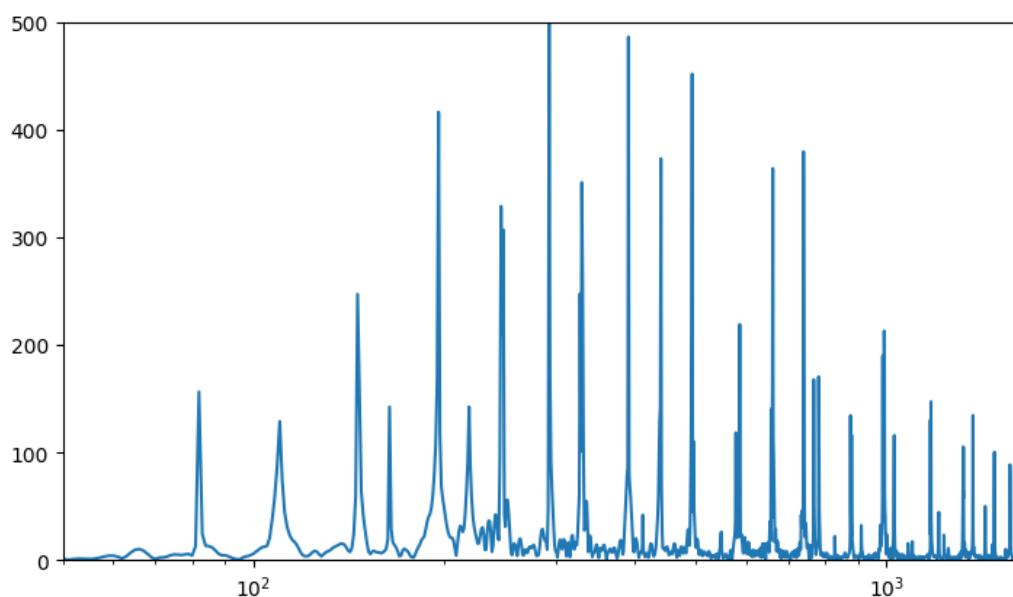
4-1 Cejilla o capo puesta en una guitarra (Imagen obtenida de <https://www.thaliacapos.com/> [Último acceso: 20/9/2017])

Aunque la frecuencia máxima a detectar son 988,89Hz es conveniente dejar un margen superior por si la cuerda a afinar es demasiado aguda y para permitir en un futuro la implementación de sistemas como reconocimiento de afinaciones diferentes que usen frecuencias mayores. Estas afinaciones son las afinaciones con *cejilla* o *capo*, llamadas así porque se consiguen usando una herramienta con ese nombre. Esta herramienta permite conseguir que las cuerdas den notas más agudas sin necesidad de pulsarlas con los dedos, ya que las mantiene siempre pulsadas a una altura elegida por el intérprete. Si bien se puede poner esta herramienta en cualquier parte del mástil de la guitarra lo más común es usarla entre los *trastes* 1 y 7, lo cual aumentaría la frecuencia de la cuerda hasta 7 semitonos.

Haciendo cálculos se obtiene una frecuencia máxima de referencia (la de la cuerda 1) de 1481Hz. Ofreciendo un margen de medio tono por encima de esa frecuencia para facilitar la detección se obtiene, por tanto, que una frecuencia máxima de 1569,8Hz podría ser lo mínimo necesario. Esto causaría que, redondeando, la frecuencia de muestreo fuera de 3200Hz. Con estos valores se obtendría una distancia entre *bins* de $3200/1024 = 3,125\text{Hz}$, que ya aporta más información sobre las cuerdas y con algo de procesamiento posterior se podría mejorar la precisión.

En la imagen 4-2 se muestra el resultado que ofrece la FFT del paquete NumPy¹⁷ de Python sobre el mismo audio usado anteriormente en MATLAB.

Para esta prueba se ha fijado la frecuencia de muestreo a 3200Hz, con lo cual la frecuencia más alta representada es de 1600Hz.



4-2 FFT en Python del rasgueo de guitarra

¹⁷ <http://www.numpy.org/> [Último acceso: 20/9/2017]

Se puede observar que se obtiene un resultado parecido al de la FFT de MATLAB, salvando algunas diferencias como la amplitud, debido a que los dos algoritmos de cálculo son diferentes.

En esta ocasión sí se puede observar cómo la FFT nos ofrece las frecuencias correctas de cada cuerda, redondeadas al entero más cercano debido a que la precisión actual del algoritmo corresponde a 1 *bin* por hercio.

Para comprobarlo se ha realizado un detector de picos para el caso monofónico que ofrece un funcionamiento que simula el de un caso real de comprobación de afinación. Este detector lee el audio pregrabado en fragmentos de 0,1 segundo y ejecuta la FFT sobre cada uno de ellos. De este modo se consigue la FFT en fragmentos de tiempo que pueden imitar a los intervalos que se conseguirían al obtener el audio en tiempo real.

El script *mono_tuner.py* busca los picos de la FFT de la señal de la guitarra tocada cuerda a cuerda, con lo cual, el primer pico de los detectados muestra la nota fundamental de la cuerda que corresponda a ese intervalo de la señal. En la Tabla 4-1 se muestran los resultados del primer pico detectado para cada cuerda.

Como se puede observar, todas las cuerdas son detectadas correctamente, dado que los resultados que se ofrecen en casi todos los casos son los enteros más cercanos a las frecuencias teóricas de la guitarra. Esa oscilación entre dos valores se debe a que la cuerda no emite una vibración perfecta y realmente varía ligeramente en valores cercanos a la frecuencia en la que fue afinada. En las cuerdas más agudas (1 y 2) se puede observar una variación algo mayor, pero es debido a que, como se ha comentado anteriormente, la misma desviación musical ocupa más hercios en frecuencias altas.

Tabla 4-1 Frecuencias detectadas por le script *mono_tuner.py*

Número de cuerda	Frecuencia fundamental (Hz)	Frecuencia detectada (Hz)
1	329,63	331
2	246,94	248
3	196,00	197
4	146,83	148
5	110,00	110
6	82,41	83

4.2 Lectura de audio en tiempo real y decisor monofónico

El siguiente paso del desarrollo en Python es permitir la lectura en tiempo real del audio que recibe el ordenador.

Para esta tarea se ha partido del script anterior (*mono_tuner.py*), al que se le ha cambiado la lectura del archivo .wav por el sistema de captura de audio.

Para asegurar el funcionamiento del sistema se debe mostrar por consola información relevante sobre el audio que se está recibiendo. Esta información es:

- Frecuencia obtenida: con el fin de comprobar la precisión del sistema.
- Cuerda pulsada y estado de la afinación: en este script se ha añadido un decisor que detecta la cuerda que se quiere afinar y devuelve el nombre y si se debe hacer más grave o más aguda para alcanzar el tono correcto. Si bien no es completamente necesario en este estado del desarrollo, es una información mucho más visual que sólo la frecuencia. Además, se introducen algunos conceptos que serán útiles en el afinador polifónico y facilita el trabajo futuro si se quisiera utilizar el script para programar un afinador monofónico funcional en Python.
- Tiempo de ejecución de la FFT: para comprobar la viabilidad del algoritmo.

4.2.1 Decisor monofónico

Este sistema funciona utilizando el primer pico detectado por la FFT. Como se ha explicado en el apartado 3.1.1, este primer pico debería corresponder a la frecuencia fundamental de la cuerda tocada.

Debido al sistema de afinación de la guitarra explicado anteriormente las cuerdas al desafinarse tendrán tendencia a tomar frecuencias cercanas a la original. Es decir, es más probable que una cuerda desafinada se encuentre en un entorno cercano a la frecuencia de referencia. Si bien no se puede establecer una ecuación probabilística concreta sin realizar un estudio de muchas situaciones diferentes de cuerdas desafinadas, se puede utilizar este conocimiento para establecer un decisor sencillo para detectar la cuerda tocada.

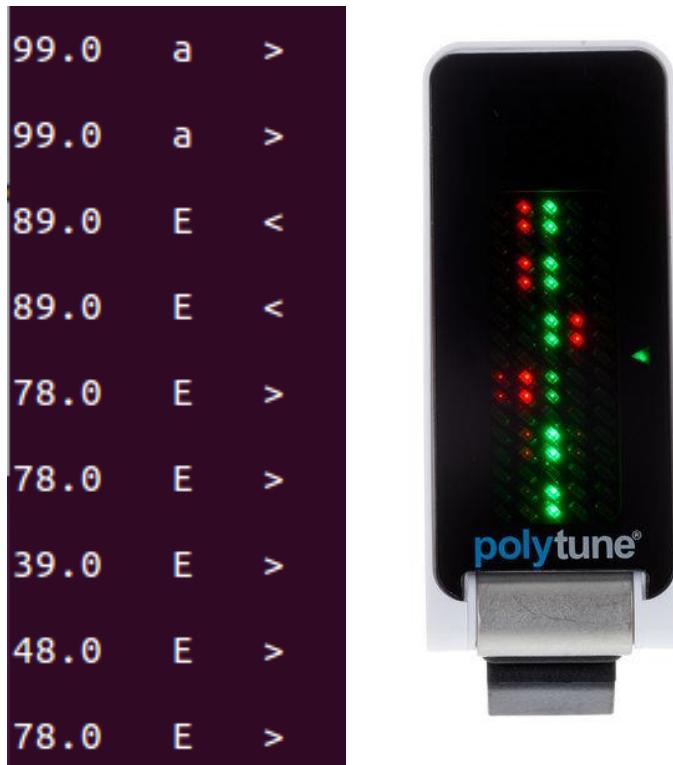
Por tanto, dado que la probabilidad de que una cuerda esté en una frecuencia cercana a su referencia es mayor que el hecho de que esté lejos, y dado que la probabilidad de que se quiera afinar una cuerda u otra es la misma, el problema se reduce a obtener la frecuencia de referencia más cercana al pico detectado.

Esto deja un decisor de la siguiente forma:

$$\arg \min_i \{ |f_{ref}^i - f_d| \}$$

Donde:

- i : es el número de cuerda. Toma valores entre 1 y 6.
- f_{ref}^i : es la frecuencia de referencia de la cuerda i .
- f_d : es la frecuencia fundamental detectada por la FFT.



4-3 Izquierda: funcionamiento del script `mono_tuner_rt.py`. Cada línea muestra, de izquierda a derecha, la frecuencia detectada, la cuerda asociada y la indicación sobre su afinación. Derecha: imagen del funcionamiento del Polytune Clip. Indica únicamente si la cuerda está demasiado grave, demasiado aguda o correctamente.

Una vez obtenida la cuerda con la referencia más cercana al pico detectado, en función del signo de la resta $f_{ref}^i - f_d$ se puede obtener información sobre si la cuerda está afinada más grave o más aguda de lo que debería.

- $f_{ref}^i - f_d > 0$: la frecuencia detectada es menor que la referencia. La consola mostrará el carácter “>” como indicación de que se debe aumentar la tensión de la cuerda.
- $f_{ref}^i - f_d < 0$: la frecuencia detectada es mayor que la referencia. Se mostrará el carácter “<” como indicación de que se debe reducir la tensión de la cuerda.
- $f_{ref}^i - f_d = 0$: la cuerda está bien afinada. Se mostrará el carácter “-“ para indicar que no es necesario afinarla.

En la imagen 4-3, se muestra un ejemplo del funcionamiento del decisor y de la representación por consola. Esta representación por consola es similar a la

usada en algunos afinadores del mercado y por tanto será la usada también en el sistema polifónico en Python.

4.2.2 Captura del audio

Para la captura del audio se ha usado el paquete de Python PyAudio¹⁸. Este paquete incorpora funciones para facilitar la programación de la captura de audio del sistema. Sin embargo, fija algunas limitaciones:

- **Frecuencia de muestreo.** La frecuencia de muestreo debe ser compatible con las frecuencias del sistema sobre el que se utiliza para evitar problemas de captura. Por tanto, no se podrá utilizar en las versiones en tiempo real la frecuencia de muestreo de 3.2 kHz que se probó con el archivo .wav. Para asegurar el funcionamiento se utilizará una frecuencia de muestreo de 44.1kHz, ya que es la frecuencia estándar de la industria del audio y está soportado por todas las tarjetas de audio.
- **Retardo.** El hecho de que el programa esté realizado sobre el lenguaje Python provoca que el audio sufra un retardo constante de unos cinco segundos entre su captura y su procesamiento debido a la cantidad de procesos que se realizan para su compilación y ejecución sobre un sistema operativo como es Ubuntu.

Para subsanar este comportamiento harían falta drivers específicos para la tarjeta de sonido para Ubuntu y no se tiene acceso a ellos. Esto permitiría eliminar los retardos introducidos por Python, pero tampoco es necesario para la realización de este proyecto debido a que los procesos que causan ese retardo no estarán presentes en el microcontrolador por el hecho de no contar con sistema operativo.

El objetivo del desarrollo en Python del afinador es el de comprobar el funcionamiento de los algoritmos utilizados. Por tanto, aunque el retardo no sea práctico para un uso real, el resto del sistema funciona correctamente y ofrece los mismos resultados que si no lo hubiera. Esto lo convierte en un programa válido para la consecución del objetivo marcado, por lo cual, se evitaron complicaciones innecesarias.

Si bien el retardo producido en la captura del audio no es relevante en este caso, los posibles retardos producidos por la ejecución de la FFT resultan cruciales de cara a determinar la posibilidad de aplicar el algoritmo en un microcontrolador. Esto se debe a que el algoritmo a aplicar en el ordenador y en el microcontrolador será muy similar. Si este algoritmo no es capaz de ser ejecutado por un ordenador en tiempo real, no se puede esperar que el microcontrolador, teniendo mucha menor potencia de procesamiento que el ordenador, sea capaz de ejecutarlo.

¹⁸ <https://github.com/jleb/pyaudio> [Último acceso: 20/9/2017]

Sin embargo, por esa diferencia de procesamiento, el hecho de que el ordenador sea capaz de realizar todas las operaciones en tiempo real tampoco asegura el funcionamiento en el microcontrolador. Para asegurar el funcionamiento se deberá conseguir un microprocesador que ejecute las FFT con una velocidad suficiente.

4.2.3 Cálculo del tiempo de ejecución

Como se ha explicado anteriormente, el tiempo de ejecución del algoritmo es crucial para asegurar el funcionamiento en un microcontrolador.

Si bien lo ideal sería que el tiempo de ejecución fuera el menor posible, si se estudia el uso que se hace de un afinador se puede calcular qué valores permitirían el funcionamiento del sistema. Esto ayudaría a concretar las necesidades reales de procesamiento.

Ejecutando el script *mono_tuner_rt.py* modificando el parámetro *CHUNK_SIZE* de PyAudio se puede conseguir modificar la cantidad de FFTs realizadas por segundo.

Este parámetro controla el tamaño del fragmento de audio que se envía a la FFT. Es decir, la FFT sólo se ejecuta cuando se completa uno de esos fragmentos. A más pequeño sea el fragmento, menos tiempo se tardará en llenar y antes se ejecutará la FFT. Dado que los datos a mostrar son el resultado de la FFT, el número de realizaciones marcará la frecuencia de actualización de los datos. Si esa frecuencia de datos permite una afinación fluida, será suficiente para permitir el funcionamiento.

Las diferentes ejecuciones del script marcan como ideal una frecuencia de actualización de 15 FFTs por segundo, si bien con 10 puede ser suficiente para el funcionamiento, aunque no se vea de manera tan fluida. Usando el valor de 15 FFTs por segundo, se obtiene que no se puede superar un tiempo de procesamiento del audio de $\frac{1}{15} = 67\text{ ms}$. Para el caso de requisitos mínimos, no se podrían superar $\frac{1}{10} = 100\text{ ms}$.

Para calcular la velocidad que consigue el ordenador se va medir el tiempo real de ejecución únicamente del algoritmo utilizando la librería *time* de Python.

Esta librería permite, del modo que muestra el fragmento de código a continuación, medir el tiempo en segundos entre dos puntos de código. Para hacerlo se resta al valor del reloj del ordenador en un punto inmediatamente posterior al algoritmo el valor del reloj en el punto inmediatamente anterior.

```
#COMIENZO DEL PERIODO DE TIEMPO A MEDIR
tiempo_inicial = time.time()
peak = fft_func(audio=audio_data)
string_name, string_status = comparator(peak=peak)
tiempo_final = time.time()
#FIN DEL PERIODO DE TIEMPO A MEDIR
tiempo_ejecucion = tiempo_final - tiempo_inicial
print(peak, ' ', string_name, ' ', string_status, ' Exec time: ',
      tiempo_ejecucion, '\n')
```

4.2.4 Conclusiones

Ejecutando de nuevo el script *mono_tuner_rt.py*, esta vez con la librería *time* se obtienen los tiempos de ejecución observados en la imagen 4-4.

123.0	a	<	Exec time:	0.012913703918457031
109.0	a	>	Exec time:	0.02039051055908203
98.0	a	>	Exec time:	0.026926755905151367
88.0	E	<	Exec time:	0.012660980224609375
99.0	a	>	Exec time:	0.012601137161254883
99.0	a	>	Exec time:	0.012871265411376953
89.0	E	<	Exec time:	0.01266336441040039
89.0	E	<	Exec time:	0.01896357536315918
78.0	E	>	Exec time:	0.020766735076904297

4-4 Ejecución del script *mono_tuner_rt.py*. Además de la información anterior, muestra el tiempo de ejecución de las operaciones de procesamiento.

Si bien el tiempo es variable, se puede observar cómo no se superan los 27ms segundos y obteniendo una media de 16,7ms, lo cual permitiría incluso algún margen para realizar procesamiento extra para aumentar la resolución del sistema. Además, dado que la FFT calculada en Python utiliza 44100 puntos y el microprocesador utilizará 1024, el margen se incrementa enormemente.

4.3 Decisor polifónico

Debido al uso de la FFT para detectar las frecuencias de la señal de audio que recibe el afinador, la única diferencia entre un detector monofónico y uno polifónico será el decisor aplicado.

Si bien a priori puede parecer que los intervalos definidos por el afinador monofónico del apartado 4.2 puedan ser útiles también para ejecutar la

búsqueda de los picos en el formato polifónico, el análisis acústico realizado en el capítulo 3 lleva a descartar esa opción.

En el caso monofónico, al encontrarse las cuerdas aisladas de las demás, no aparecen en frecuencias intermedias picos de niveles altos correspondientes a los armónicos de cuerdas más graves.

Si bien es cierto que, dada la característica de los armónicos de ser al menos de un valor de $2 \cdot f_0$, para las cuerdas de menor frecuencia como pueden ser la 6, la 5, la 4 y la 3 se pueden obtener buenos resultados usando esa misma región de decisión como se puede ver en la imagen 3-4.

En el caso de las cuerdas más agudas el caso es diferente debido a que las frecuencias de referencia fijadas están rodeadas de picos mucho mayores que se encuentran bastante cercanos.



4-5 Izquierda: Imagen del afinador Fender California. Derecha: imagen del afinador Yamaha YTC5.¹⁹

Por tanto, resulta necesario establecer regiones de búsqueda que permitan marcar los picos reales de las cuerdas como los más altos de esa región de cara a facilitar la detección.

Para intentar establecer un criterio para establecer las regiones realizaron comprobaciones usando varios afinadores comerciales. Como no hay datos sobre su funcionamiento concreto debido al carácter secreto de su diseño, se va a intentar obtener información a partir de los datos que muestran algunos modelos como el Fender California y el Yamaha YTC 5.

Como se puede observar en la figura 4-5, la pantalla de estos afinadores sólo muestra valores situados en el intervalo limitado por la frecuencia 50 centésimas

¹⁹ Fotos obtenidas de: https://www.thomann.de/es/fender_california_ft1620_clip_tuner_i.htm y https://www.thomann.de/es/yamaha_ytc_5_clip_tuner.htm respectivamente. [20/9/2017]

de tono menor y la frecuencia 50 centésimas de tono mayor que la referencia correspondiente. Esto puede llevar a pensar que la zona de detección de cada cuerda es ese intervalo de 50 centésimas a cada lado.

La lógica de este sistema se podría basar en que el momento en que la variación de frecuencia supera las 50 centésimas (valor que supone el punto medio entre dos notas) se estaría más cerca de la nota contigua que de la nota original. Además, este cálculo de las regiones permite adaptarlas a cada frecuencia, ofreciendo intervalos más grandes a medida que aumenta la frecuencia debido al modo en que se calculan las centésimas de tono explicado en el apartado 1.1 sobre la escala musical.

Por tanto, aplicando este criterio las zonas de detección para cada cuerda serían las siguientes:

Tabla 4-2 Límites en hercios de los intervalos de búsqueda para cada cuerda en el caso polifónico según el criterio a priori utilizado por los afinadores comerciales.

Número de cuerda	Frecuencia a detectar (Hz)	Límite inferior del intervalo de búsqueda (Hz)	Límite superior del intervalo de búsqueda (Hz)
1	988,89	960,73	1.017,86
2	740,82	719,73	762,53
3	196,00	190,42	201,74
4	146,83	142,65	151,13
5	110,00	106,86	113,22
6	82,41	80,06	84,82

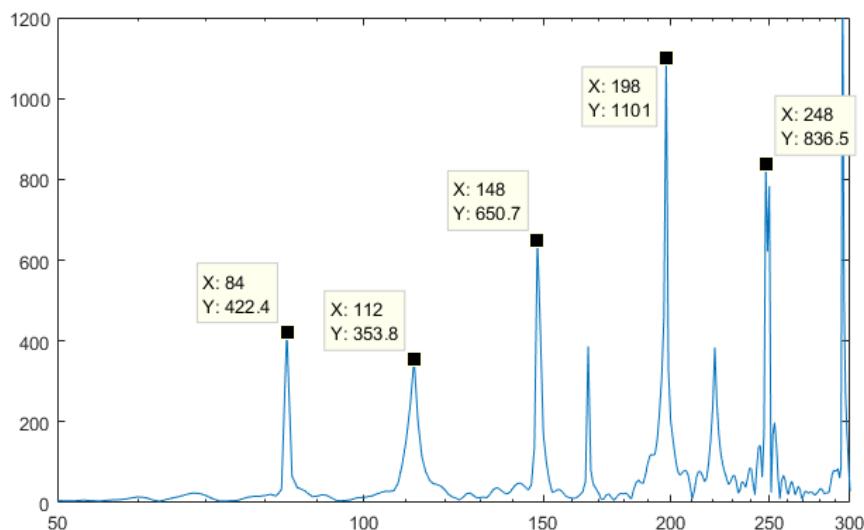
Estos márgenes, si bien parecen lógicos para evitar armónicos que puedan ser mayores que la frecuencia a detectar, ofrecen márgenes muy pequeños. El problema se agrava si se tiene en cuenta el cálculo aproximado que se ha hecho en la sección 4.1 sobre la resolución del microcontrolador. Este cálculo ofrecía, en el mejor de los casos, una precisión de 3,125Hz, lo cual reduce la zona de detección de la cuerda 6, por ejemplo, a 2 *bins*. Esta falta de datos puede causar problemas y si se puede, se debería aumentar el rango.

Para estudiar la viabilidad de aumentar esos intervalos, se recurrirá a los espectros medidos en el apartado sobre el análisis de audio.

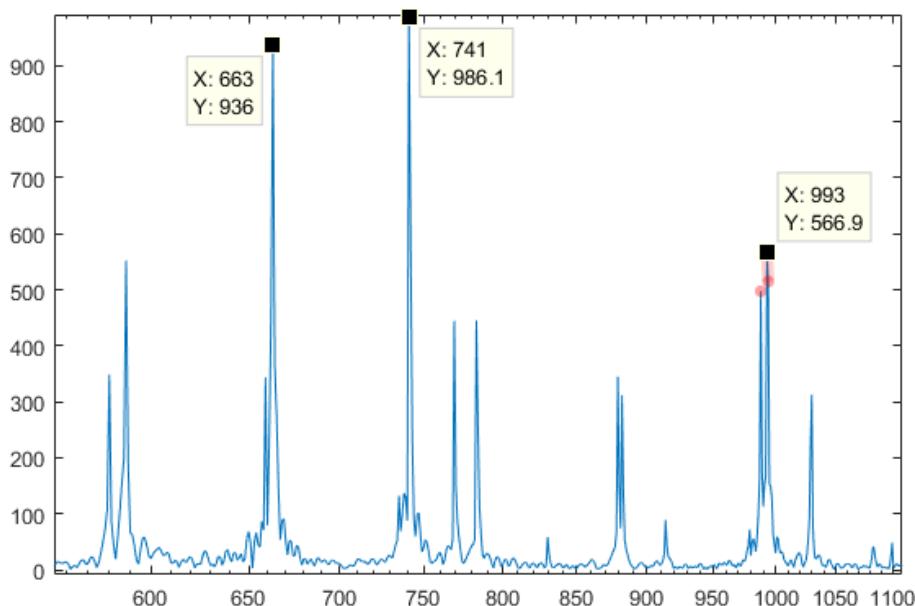
Como se puede observar en la figura, las cuerdas 6, 5, 4 y 3 tienen un margen bastante más amplio de detección alrededor debido a que no tienen armónicos

con mayor amplitud que su fundamental. El único armónico que podría causar problemas al detectar la cuerda 4 se encuentra a 248 Hz con lo cual el margen es bastante grande y aunque tiene mayor amplitud que la mayoría de las cuerdas, sigue siendo menor que la más cercana.

Para las cuerdas 6, 5 y 4 se observa que se puede dejar sin problema aparente un margen de 2,5 semitonos para permitir la detección. Se marca este límite debido a que la distancia entre ellas en afinación estándar es precisamente de 5 semitonos. Por tanto, se coge musicalmente el punto medio. Entre las cuerdas 4 y 3, sin embargo, este margen se va a limitar a 2 semitonos debido a que la distancia entre ellas es de 4 semitonos al contrario que entre las demás.



4-6 Representación del espectro frecuencial del rasgueo de guitarra. Los valores marcados corresponden, de izquierda a derecha, a las cuerdas 6, 5, 4 y 3 y a un armónico de nivel notable.



4-7 Espectro en frecuencia del rasgueo de guitarra en escala no logarítmica. Los puntos corresponden, de izquierda a derecha, a un armónico de nivel notable, a la frecuencia de referencia de la cuerda 2 y a la frecuencia de referencia de la cuerda 1.

Para las cuerdas 2 y 1 se muestra la zona de altas frecuencias con mayor resolución en la figura 4-7.

En el caso de la cuerda 2 se puede observar un armónico bastante alto en la zona de frecuencias inferiores a ella. Este armónico limita la zona de detección inferior de esta cuerda a 1 semitono para evitar el armónico. En el caso de la cuerda 1 no hay problemas por ningún lado, limitando la detección a 2,5 semitonos por ambos lados para no entrar en la zona de detección de la cuerda 2.

Las nuevas zonas de detección, por tanto, quedarían de la siguiente manera:

Tabla 4-3 Extensión de los intervalos de búsqueda en hercios de cada cuerda en formato polifónico.

Número de cuerda	Frecuencia a detectar (Hz)	Límite inferior del intervalo de búsqueda (Hz)	Límite superior del intervalo de búsqueda (Hz)
1	988,89	855,91	1.142,52
2	740,82	699,24	855,91
3	196,00	169,64	226,45
4	146,83	127,09	169,64
5	110,00	95,21	127,09
6	82,41	71,33	95,21

Estos intervalos ya ofrecen un margen más amplio para poder obtener datos válidos a bajas resoluciones.

Cabe destacar que son márgenes muy amplios para una guitarra eléctrica ya que, por experiencia, cuando se desafina una guitarra la variación suele estar normalmente por debajo de los 20 cents de tono. El motivo principal de este aumento es permitir obtener algún dato extra sobre las cuerdas más graves de cara al algoritmo de aumento de la resolución.

4.4 Aumento de la resolución

La versión Python del afinador con la precisión actual de 1Hz por *bin* no se puede usar para afinar.

Para que el oído humano detecte dos notas como iguales tiene que haber entre ellas una diferencia menor de 5 cents. En la cuerda 6 (que será la cuerda con peor precisión de todas al tener la menor frecuencia) se obtiene un ancho de banda de 0.24 Hz para esos 5 cents.

Como la distancia entre *bins* es de 1Hz no se puede obtener de ninguna manera directa una variación de como máximo 0.24Hz, haciendo imposible una comparación que no produzca un error demasiado alto.

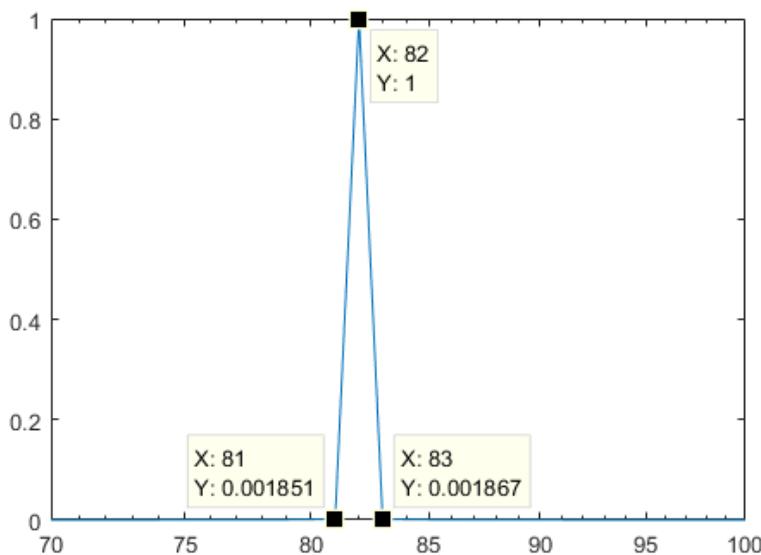
Sin embargo, con cierto procesamiento extra se puede obtener un valor aproximado de la posición real del pico que la FFT ha mostrado en ese *bin*.

4.4.1 Errores de la FFT

Debido al método de cálculo basado en ventanas en frecuencia que usa la FFT (como se explicó en el apartado 0 las frecuencias que están situadas entre dos *bins* diferentes no se ven representadas sólo en el más cercano de esos *bins*.

Cuando una frecuencia no coincide exactamente con un *bin* se produce una dispersión de su energía en torno a la frecuencia real. Por tanto, todos los *bins* situados alrededor, reflejarán parte de esa energía.

Para mostrar esta dispersión, en el script MATLAB *interp_test.m* se ha creado un seno de 82Hz y se ha calculado su FFT, mostrada en la imagen 4-8.

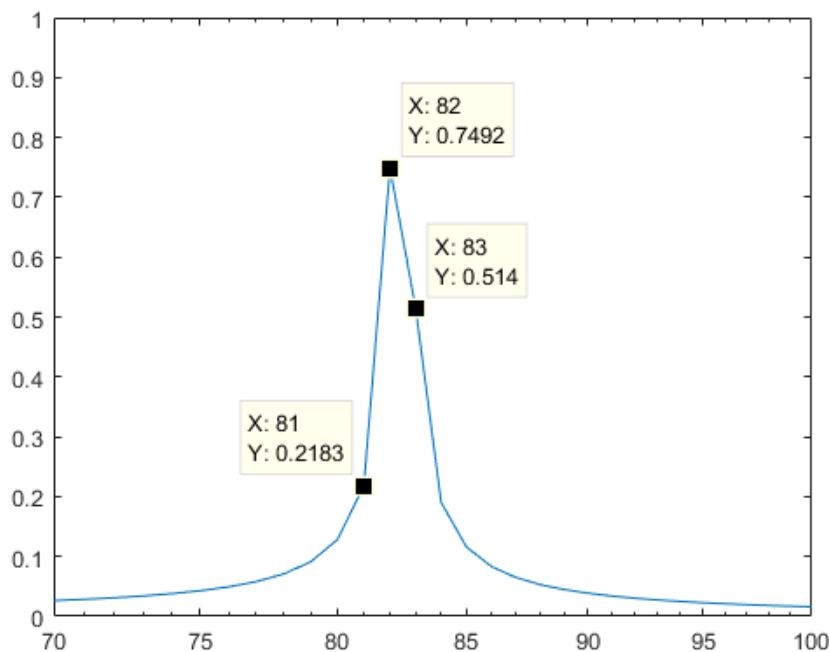


4-8 FFT de una onda senoidal de frecuencia 82Hz. La FFT marca un pico en el bin 82 y los valores adyacentes resultan prácticamente despreciables.

Como se puede observar, la FFT muestra el resultado esperado. Hay un pico en la frecuencia correspondiente al seno y valores del orden de 1000 veces inferiores al valor de ese pico en los puntos adyacentes. Esta representación es una aproximación bastante precisa de una delta de Dirac, que es la transformada de Fourier Teórica de la función seno.

Si el seno cambiara de frecuencia a 82.41 Hz se esperaría, visto el ejemplo anterior, que se creara una delta en 82.41 Hz. Como no hay un *bin* específico para esa frecuencia sería lógico pensar que se distribuyera su energía entre los dos *bins* más cercanos, 82 y 83 Hz. Al no estar exactamente en el centro de los dos, a priori también sería lógico pensar que el *bin* 82 será ligeramente más alto que el 83 al estar más cerca de él la pico. Además, el resto de *bins* toman valores muy bajos al estar más lejos de la frecuencia real.

Sin embargo, al ejecutar la FFT se obtiene la respuesta mostrada en la imagen 4-9.



4-9 FFT de un seno de 82.41 Hz. Se puede observar la dispersión producida al no coincidir la frecuencia con el valor exacto de un bin.

Aunque se cumple que los *bins* con más amplitud son los adyacentes (el 82 y el 83), se puede observar cómo también los *bins* cercanos toman valores no despreciables al contrario que en el primer caso. Si bien esta dispersión puede parecer un problema, en realidad resulta muy útil. El algoritmo de la FFT no asigna dos valores iguales a dos frecuencias diferentes, por tanto, los valores marcados por esos puntos identifican de manera indirecta la frecuencia concreta de 82.41 Hz. Aunque la relación entre esos valores y la frecuencia real es bastante compleja de calcular matemáticamente, se puede intentar simplificar.

4.4.2 Interpolación polinómica

Con el objetivo de identificar en esa zona el pico correcto con mayor precisión se va a realizar una interpolación de los máximos locales con el objetivo de conseguir más muestras. Como la interpolación debe ser llevada a cabo posteriormente por un microcontrolador con mucha menor potencia que la del ordenador no se pueden aplicar algoritmos que sean computacionalmente demasiado complejos. La interpolación más sencilla computacionalmente que pueda conseguir una forma a priori aproximada a la curva formada por la FFT es la interpolación cuadrática.

Esta interpolación se basa en calcular los coeficientes de un polinomio de segundo grado que pase por 3 puntos reales de la gráfica.

Este sistema se puede implementar mediante el uso del algoritmo de Lagrange para el cálculo de un polinomio $L(x)$ cuya expresión es:

$$L(x) = \sum_{j=0}^k y_j l_j(x)$$

Dado un conjunto de puntos de longitud $k+1$:

$$(x_0, y_0), \dots, (x_k, y_k)$$

Donde se asume que todos los x_i son diferentes y siendo $l_j(x)$:

$$l_j(x) = \prod_{i=0, i \neq j}^k (x - x_i) / (x_j - x_i)$$

Este algoritmo consigue una complejidad de computación bastante baja y se puede programar sin necesidad de librerías matemáticas con relativa facilidad debido a que está compuesto por operaciones que un microcontrolador puede realizar de manera nativa (suma y resta en todos los microcontroladores y multiplicación y división en algunos).

Sin embargo, ese algoritmo cuenta con ciertas limitaciones:

- La complejidad aumenta a mayor sea el grado del polinomio a calcular. Esto limitará la capacidad de ajustar la curva. La primera opción que se va a probar será usar un polinomio de grado 2. En función de los resultados se planteará aumentar al grado o usar otros algoritmos.
- El grado del polinomio está ligado a la cantidad de muestras tomadas. Para que el polinomio sea de grado 2, sólo se podrán tomar 3 muestras. Esto se debe a que este algoritmo obliga al polinomio a pasar por todos los puntos.

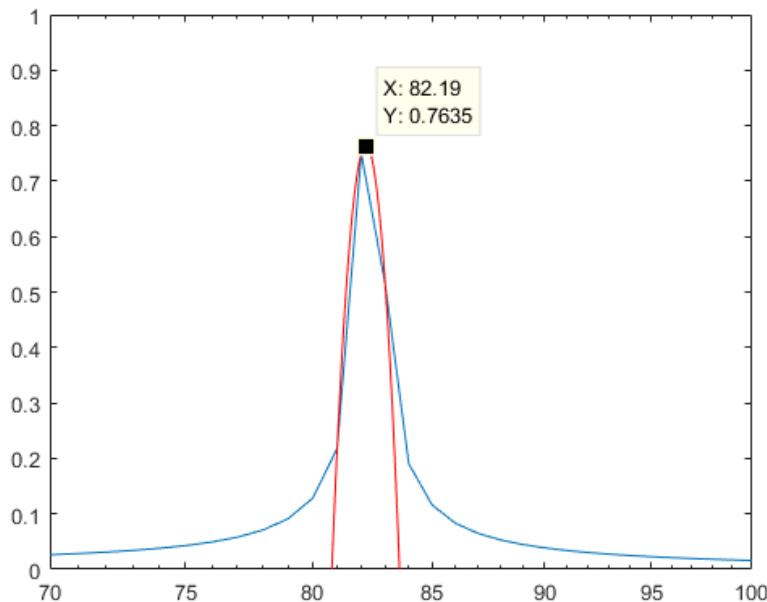
Para intentar obtener los datos más representativos se escogerá los puntos con valores más altos ya que en ellos se representa la mayor parte de la energía.

Como se puede observar en la imagen 4-10, el pico obtenido está en 82,19 Hz por lo cual se encuentra desviado con respecto al valor que debería tener de 82,41 Hz.

Para que un *bin* (f_0) de la FFT original sea el valor más alto debe ser el más cercano a la frecuencia calculada por la FFT (f_R), es decir, será el pico más alto si:

$$|f_R - f_0| < 0.5$$

Por tanto, se va a estudiar cómo afecta la estimación polinómica en ese rango para un *bin* concreto calculando el pico que se detecta para cada una de esas frecuencias.



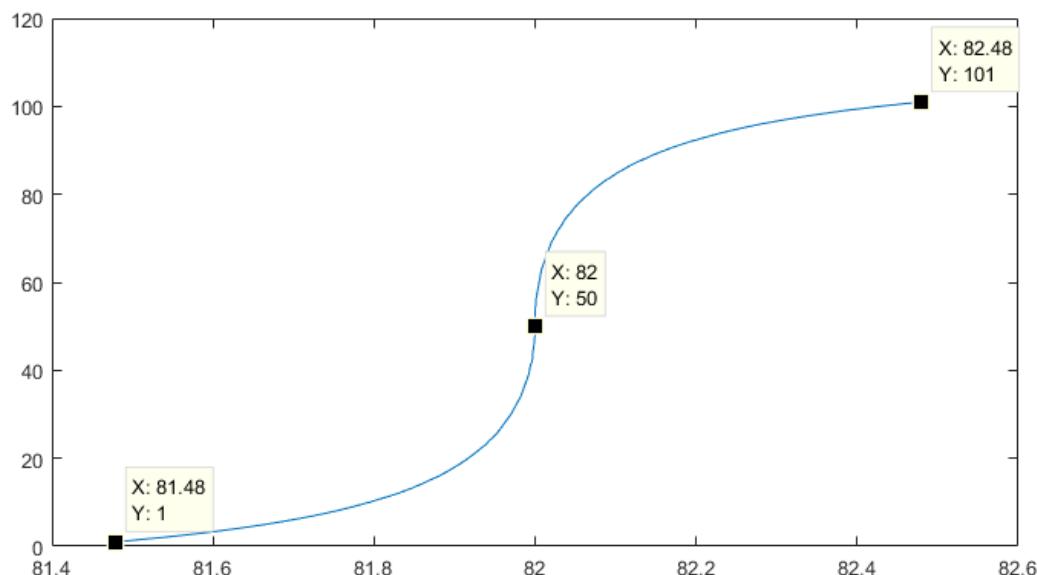
4-10 En azul, FFT de un seno a 82.41 Hz. En rojo la interpolación cuadrática basada en los 3 valores más altos. La marca corresponde al punto más alto de la estimación cuadrática.

Para la prueba se han tomado 100 muestras equiespaciadas entre 81.5 y 82.5 Hz obteniendo la gráfica 4-11.

Estudiando los resultados se puede comprobar que la desviación producida no es constante en todo el margen ya que las frecuencias generadas no se distribuyen linealmente, aunque se produce un *offset* que sí es constante de 0.02 Hz hacia frecuencias graves (antes comenzaba en 81,5 y acababa en 82,5 y ahora comienza en 81,48 y acaba en 82,8).

Además, a medida que se acerca al *bin “real”* (el generado por la propia FFT), la desviación provoca que las frecuencias estimadas estén más cerca de lo debido mientras que los valores más lejanos tienden a separarse más.

Dado que la gráfica muestra una forma similar a algunas funciones comunes, se podría pensar que se puede aproximar esta desviación a una función matemática que asignara a cada frecuencia leída su valor corregido. En un caso común de detección de frecuencias esa forma de solucionar el problema podría ser la más apropiada dado que ofrece la capacidad de corregir todas las frecuencias introducidas en el sistema con un procesamiento a priori bastante ligero, pero que podría variar según la forma que tomara la función que más se aproxime a la curva dibujada.



4-11 Valores frecuenciales obtenidos por la estimación polinómica para las frecuencias de 81.5 a 82.5.
Eje vertical: número de muestras. Eje horizontal: frecuencia calculada.

Este afinador, en cambio, no necesita conocer con exactitud todas las frecuencias leídas. Su funcionamiento sólo va a basarse en informar sobre si la cuerda está afinada más grave, más aguda o correctamente.

La gráfica 4-11 muestra una figura estrictamente creciente, lo que indica que la transformación ofrece valores estimados mayores conforme mayor es la frecuencia leída. Al no variar esta característica de las frecuencias leídas sólo resulta necesario para el funcionamiento del afinador conocer la desviación producida en la frecuencia de referencia usada.

Por tanto, aplicando una corrección a cada referencia en forma de *offset*, dado el funcionamiento del afinador, aunque no se conozcan los valores reales no se va a afectar a su capacidad para detectar si el tono leído es el correcto.

Este ahorro de cálculos y complejidad de computación conlleva la necesidad de calibrar el afinador introduciéndole como señales de prueba las referencias

exactas de cada cuerda. De este modo el afinador mostrará la frecuencia desviada que detecta y se podrá corregir la referencia para que se tome como tal la nota mostrada.

4.4.3 Asegurar la precisión

El hecho de estimar de la forma explicada en el apartado anterior las frecuencias plantea un problema para el cálculo de la precisión debido a la desviación desigual de las estimaciones con respecto al valor original.

Como se ha explicado en el apartado 1.1 sobre la escala musical, los intervalos musicales son mayores (ocupan un ancho de banda mayor) a mayor sea la frecuencia. En la FFT los *bins* tienen el mismo ancho de banda independientemente de la frecuencia. Estas cualidades causan que un mismo intervalo musical ocupe varios *bins* en frecuencias altas, pero sólo una fracción de *bin* en frecuencias bajas.

Por tanto, a priori, si se obtiene la precisión necesaria para el intervalo inmediatamente inferior a la sexta cuerda, se debería conseguir la precisión necesaria para todas las demás ya que es la más grave. Sin embargo, la cercanía de una frecuencia a un *bin* real de la FFT causa variaciones en la precisión debido a que, como se puede ver en la gráfica 4-11, en los puntos centrales la pendiente es muy alta, indicando que los valores de esa zona tienden a juntarse, causando que la distancia entre ellos sea menor.

Como el interpolador causa que las frecuencias detectadas más cercanas al *bin* real se acerquen aún más a ese *bin*, la precisión disminuirá enormemente, ya que tomarán valores más parecidos y se necesitará la obtención de más decimales para conseguir diferenciarlos.

En el caso de este programa Python, como la segunda cuerda tiene una frecuencia de referencia que coincide con un *bin* real (110,00Hz), puede ocurrir que un valor que se ha desviado en la realidad más de 5 cents el detector considere que está más cercano debido a esta desviación. Al ser la segunda cuerda más grave, hay que comprobar si esta variación supone un problema mayor que la de la cuerda 6. Si bien la precisión mínima para detectar diferencias es de 5 cents, se va a intentar conseguir una precisión más parecida a la de las aplicaciones comerciales, intentando conseguir una diferencia de 1 cent.

Dado que el problema reside en la cercanía de las muestras, cuantas más haya, mejor se diferenciarán entre sí. Por tanto, para conseguir aumentar la resolución habrá que aumentar el número de muestras creadas. Por tanto, realmente sólo es necesario asegurar que a la frecuencia de referencia y a la frecuencia que esté 1 cent por debajo se les asignarán dos muestras diferentes.

Para comprobarlo se deberá calcular primero qué frecuencia f_{-1} se encuentra 1 cent por debajo de la referencia para cada cuerda:

Cuerda 6:

$$f_{-1}^6 = \frac{f_0^6}{\frac{1}{2^{1200}}} = 82.35 \text{ Hz}$$

Cuerda 5:

$$f_{-1}^5 = \frac{f_0^5}{\frac{1}{2^{1200}}} = 109.94 \text{ Hz}$$

Con estos datos se pueden sintetizar usando MATLAB dos archivos .wav de audio (uno por cuerda) que se introducirán al script *poly_tuner.py*. Cada uno de estos archivos contendrá durante los 15 primeros segundos un seno a la frecuencia de referencia de una de las cuerdas y los 15 siguientes un seno a la frecuencia f_{-1} de la misma cuerda. De este modo se podrá ir probando con diferentes cantidades de muestras generadas. La cuerda que necesite una cantidad de muestras mayor para diferenciar ambas frecuencias será la que marque el número de muestras generadas.

La síntesis de los archivos de audio se encuentra implementada en el fichero MATLAB *calibrator_wav_generator.m*.

Para someter el afinador a las pruebas para comprobar la resolución se comenzará por 200 y se subirá duplicando el número de muestras para intentar que los números obtenidos sean más redondos. Al no limitar en Python el procesamiento se puede permitir esto.

Si bien con 200 no se consigue diferenciar en ninguna de las cuerdas, y con 400 sólo en la 5, al llegar a 800 muestras por *bin* se obtienen las siguientes salidas para ambas cuerdas:

Tabla 4-4 Resultados del script para los audios a la frecuencia de referencia y para los audios a la frecuencia de referencia menos 1 cent.

	Frecuencia de referencia 800 muestras.	Frecuencia de referencia -1 cent. 800 muestras
Cuerda 5	109.9975	109.9775
Cuerda 6	81.9975	81.9850

Como se puede observar, con esta resolución ya se puede distinguir entre los dos tonos en esta cuerda, por tanto, el sistema toma 800 muestras por *bin* ya que se obtiene resolución suficiente para igualar a algunos sistemas comerciales.

4.4.4 Calibrado del script Python

Una vez conseguida la resolución necesaria para el afinador se procedió a su calibración.

Para realizar la calibración se introdujo por la entrada de audio del ordenador varios archivos de audio con cada una las frecuencias de referencia. Estas frecuencias serán tonos puros generados usando el software MATLAB y se guardarán en formato .wav para evitar pérdidas debidas a la compresión.

Introduciendo los audios se obtienen las salidas indicadas en la tabla, en la cual se han calculado además los offset correspondientes a cada cuerda.

Tabla 4-5 Resultados del script *poly_tuner_rt.py* al introducir los audios para la calibración.

Número de cuerda	Frecuencia teórica a detectar (Hz)	Frecuencia real a detectar (Hz)	Offset (Hz) ($F_{\text{teórica}} - F_{\text{real}}$)
1	988,89	988,9025	-0,125
2	740,82	740,8150	0,005
3	196,00	195,9575	0,0425
4	146,83	146,7950	0,035
5	110,00	109,9975	0,0025
6	82,41	81,9975	0,4125

Estos valores son los que introducirán al programa para usar como referencia. De este modo se asegurará que la frecuencia calculada coincide con la frecuencia correcta. Como estos offsets los causa el algoritmo de interpolación, son los mismos independientemente del ordenador o tarjeta de audio usados, aunque la calidad de los circuitos de audio de estos dispositivos pueda limitar la reoslución del afinador.

El calibrado, al depender de la referencia de *bins* usados por la interpolación será diferente en el caso del microcontrolador, ya que la distancia entre *bins* seguramente sea mayor. Por tanto, el método utilizado para la calibración se explicará también de una forma más orientada al microcontrolador en el apartado 5.3.5 para el calibrado del dispositivo final montado en un microcontrolador.

5. Implementación física

Una vez comprobado que se puede realizar de forma satisfactoria un afinador polifónico *software* con los parámetros elegidos se puede continuar con la implementación del sistema en un microcontrolador que pueda funcionar de forma autónoma.

El primer paso en este apartado fue la elección de un microcontrolador que fuera capaz de realizar el procesamiento necesario y diseñar un circuito que permita conectarlo a una guitarra. Una vez conseguido eso, se programó el afinador basándose en el prototipo en Python, calibrándolo y asegurando la resolución suficiente. Posteriormente se le incluyeron los sistemas de representación y de alimentación para conseguir un prototipo completamente funcional y portátil.

5.1 Elección del microcontrolador

El primer paso para la implementación consiste en comparar las opciones que hay en el mercado de cara a elegir la más apropiada.

La elección inicial se basó en cuatro criterios:

- Presencia de un ADC (conversor de analógico a digital) de buena calidad. Por lo general los integrados en los microcontroladores suelen ofrecer una conversión de una calidad baja. Si bien pueden servir para algunas aplicaciones de audio, dado que el proyecto necesita una resolución bastante alta se debe optar por un microcontrolador que ofrezca un ADC de la mejor calidad posible.
- Presencia de una FPU (unidad de punto flotante). Esta unidad permite a los microcontroladores ejecutar algunas operaciones de forma paralelizada, acelerando en gran medida la ejecución. Esta unidad es la encargada de realizar la FFT en tiempo real mientras se obtienen las siguientes muestras de audio, aprovechando todas las muestras posibles.

- Precio: como sistema orientado al DIY, cuanto más barato sea, más accesible será. Por tanto, se debe escoger el sistema *low-cost* que cumpla las especificaciones.
- Open source. Dado que el proyecto completo pretende formar parte del marco *open source*, un paso importante es que la propia placa sobre la que se integre el programa sea también de código abierto para facilitar la programación y las modificaciones futuras que se quieran realizar.

Tras una búsqueda sobre los sistemas Arduino²⁰, concretamente sobre el UNO²¹, MEGA²², ZERO²³ y M0²⁴, se descartaron todos ellos dado que ninguno de los chipsets que utilizan tiene FPU. Además, ninguna de las placas dispone de un ADC con suficiente resolución, si bien mediante una placa de expansión se podría añadir un ADC mejor del proporcionado de serie en esas placas.

Tras una investigación más profunda se llegó a la conclusión de que para conseguir una placa con FPU es necesario que el microcontrolador esté basado en una arquitectura ARM²⁵ Cortex M4 o M7 o sistemas Intel. Sin embargo, surgen dos problemas:

- **Capacidad de operar en tiempo real:** Si bien el procesamiento que ofrecen algunos sistemas es muy potente, como el de la Raspberry Pi²⁶, la necesidad de un sistema operativo que lo controle causa una latencia que, al igual que con el script Python, haría el sistema poco útil. La solución sería utilizar otra placa aparte que sirviera de interfaz de audio, pero el sistema sería demasiado grande, consumiría mucho y sería muy caro.
- **Precio:** una de las placas de desarrollo más baratas encontradas, basada en Cortex M4 con FPU es la placa Teensy²⁷ 3.5, cuyo precio de partida es de 24,25\$. El resto se encuentran bastante por encima y no están optimizados para audio como la Teensy.

Si bien la opción más barata a priori parece la Teensy 3.5, una búsqueda más exhaustiva lleva a encontrar la Teensy 3.2, basada también en ARM Cortex M4, pero sin FPU, por un precio de 19\$, inferior al de la 3.5.

A pesar de no cumplir la condición inicial de tener FPU, la Teensy 3.2 ofrece otras ventajas:

²⁰ <https://www.arduino.cc/> [Último acceso: 20/9/2017]

²¹ <https://store.arduino.cc/arduino-uno-rev3> [Último acceso: 23/9/2017]

²² <https://store.arduino.cc/arduino-mega-2560-rev3> [Último acceso: 23/9/2017]

²³ <https://store.arduino.cc/genuino-zero> [Último acceso: 23/9/2017]

²⁴ <https://store.arduino.cc/arduino-m0> [Último acceso: 23/9/2017]

²⁵ <https://www.arm.com/> [Último acceso: 20/9/2017]

²⁶ <https://www.raspberrypi.org/> [Último acceso: 20/9/2017]

²⁷ <https://www.pjrc.com/teensy/> [Último acceso: 20/9/2017]

- **Velocidad del procesador:** El procesador de la Teensy 3.2 permite el funcionamiento hasta a 96MHz. A más rápido sea el procesador, más instrucciones podrá procesar por segundo, con lo que tardará menos en ejecutar operaciones complejas. Esto supone una gran ventaja frente los 16 o 48 MHz soportados por las plataformas Arduino.
- **Optimización DSP (Procesamiento Digital de Señales):** Si bien la alta velocidad del procesador por sí sola no permitiría la realización de una FFT de 1024 muestras en tiempo real con una actualización de al menos 100ms junto con el resto de las operaciones necesarias, la optimización del chip para el procesamiento digital de señales mediante instrucciones específicas para el procesamiento de señales mejora enormemente la ejecución. Ciertos bancos de pruebas²⁸ muestran cómo a 512 muestras la placa Teensy 3.2 consigue realizar 130 FFTs por segundo frente a las 16 del Arduino M0. Si bien no será necesario realizar todas esas FFTs, el hecho de que se puedan realizar ofrece tiempo de procesamiento extra a los métodos de interpolación y al control de la pantalla.

Ambas placas Teensy necesitan un adaptador para las aplicaciones orientadas al audio, el Audio Adaptor Board for Teensy 3.0-3.6. Este módulo sirve de interfaz entre la guitarra y el microcontrolador ofreciendo un ADC con una calidad de 16 bits, frente a los 10 o 12 bits de las plataformas Arduino. Además, ofrece una frecuencia de muestreo de 44,1 kHz y operaciones integradas para el procesamiento del audio.

5.2 Circuito de entrada

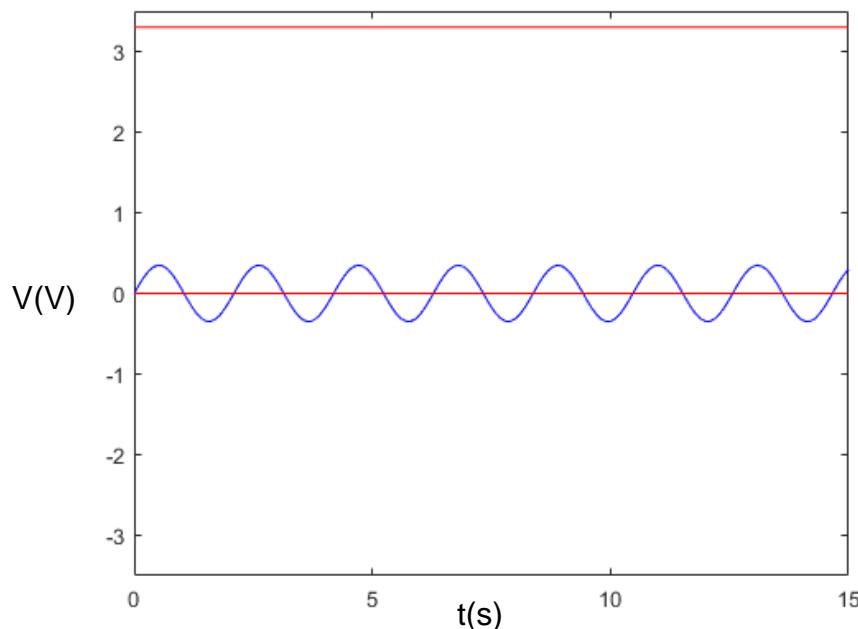
Para hacer el sistema aplicable al mayor número posible de guitarras es conveniente realizar un acondicionamiento de la señal de salida de la guitarra para adaptarla a la placa de audio.

La señal de salida de una guitarra eléctrica tiene, por lo general, una amplitud de entre 150mV y 350mV y una impedancia de salida de unos $8k\Omega$, sin embargo, algunos modelos de eléctricas o las electroacústicas superan esos valores de voltaje, llegando en torno a 1V y tienen menor impedancia. Además, es una señal sin componente continua y tiene valores positivos y negativos, es decir, oscila alrededor de 0V.

La entrada de línea (Line In) de la placa de audio requiere de una señal de entrada que oscile entre 0 y el voltaje de alimentación, en este caso 3.3V.

Para conseguir conectar guitarra y placa, por tanto, son necesarias dos etapas, una para adaptar el voltaje y otra para adaptar las impedancias.

²⁸ <http://openaudio.blogspot.com.es/2016/09/benchmarking-fft-speed.html> [Último acceso: 20/9/2017]



5-1 En azul, ejemplo de una señal de guitarra eléctrica con una amplitud de 350mV. En rojo, los límites operativos de la placa de audio Teensy.

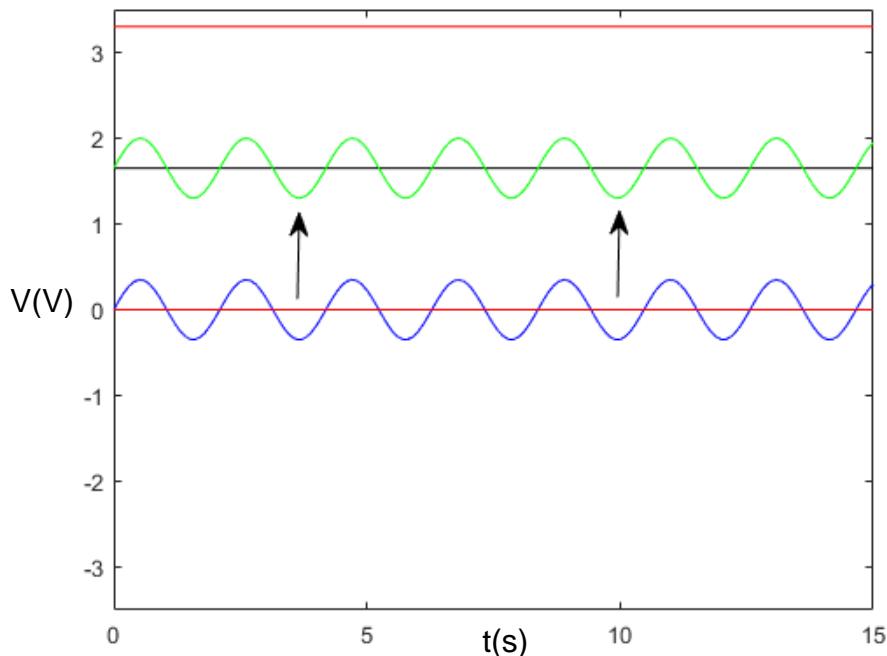
5.2.1 Adaptación del voltaje

Dado que la placa requiere un voltaje cuyo valor máximo sea 3.3V y cuyo valor mínimo sea 0V, y la señal de la guitarra tiene valores positivos y negativos, es necesario modificar el voltaje de entrada. La primera etapa se encarga de este cometido.

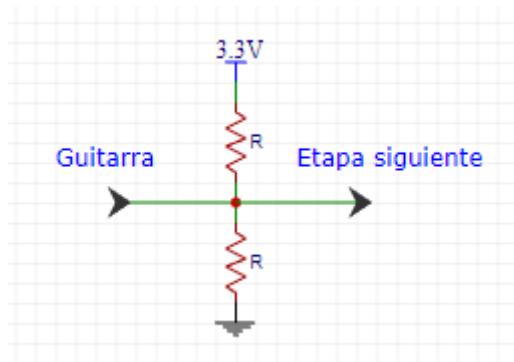
Para tener el máximo rango posible se debe introducir en la señal de la guitarra una componente continua cuyo voltaje sea la mitad del valor máximo admitido por la Teensy, es decir, de 1.65V. De este modo se deja a la señal de guitarra un margen de 1.65V tanto por encima, para los valores que antes eran positivos, como por debajo, para los valores que antes eran negativos. Dado que el voltaje máximo estándar de las guitarras eléctricas son 350mV, se quedaría un margen extra de 0.8V para algunas guitarras como las electroacústicas o las eléctricas activas, que pueden llegar a esos valores fácilmente porque cuentan con un preamplificador integrado.

Además, la placa de audio Teensy dispone de un amplificador programable previo al ADC que permite aumentar el voltaje de la señal de entrada sin continua, permitiendo aumentar el voltaje posteriormente si fuera necesario.

El circuito necesario para la realización de este ajuste a la señal está formado por dos resistencias con la configuración mostrada en la imagen 5-3.



5-2 En negro, el valor de continua de 1.65V que ha de sumarse a la señal. En verde, la señal una vez añadido ese valor y adaptada para el funcionamiento de la placa Teensy.

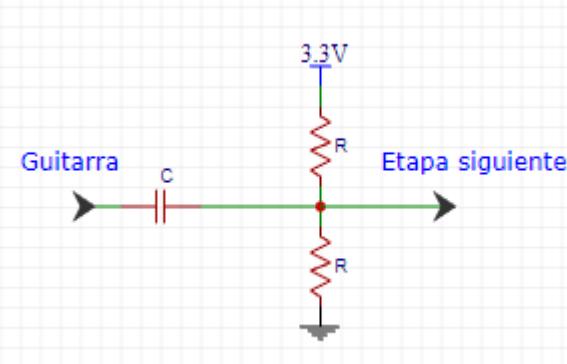


5-3 Divisor de voltaje

De esta forma, ejecutando un análisis en continua se obtiene que forman un divisor de voltaje que, independientemente del valor que tomen las resistencias siempre que sea el mismo en ambas, ofrece una tensión de 1.65V que se sumaría a la señal de la guitarra.

$$V_{out} = \frac{R_2}{R_1 + R_2} * V_{in} = \frac{1}{2} V_{in}$$

Para asegurar que la señal de la guitarra en el momento de llegar al divisor de tensión esté centrada en 0 y no tenga ningún tipo de ruido de baja frecuencia que pueda desviárla se introduce un condensador en serie, previo a las resistencias, que forma junto con estas un filtro paso alto. El circuito resultante se muestra en la imagen 5-4.



5-4 Divisor de voltaje con filtro paso alto

Para que este filtro no afecte a la señal de audio su frecuencia de corte debe ser lo suficientemente baja, concretamente por debajo de 20Hz para que no afecte al espectro audible, así que hay que elegir los valores de los componentes para que se cumpla esta condición.

Por tanto, aplicando la ecuación de la frecuencia de corte (f_c) de un filtro paso alto:

$$f_c = \frac{1}{2\pi R_{eq} C}$$

Donde:

- R_{eq} : es la resistencia equivalente del divisor de tensión en ohmios, es decir, la suma en paralelo de las dos resistencias que lo forman. Al ser iguales se puede demostrar que $R_{eq} = R/2$ mediante la siguiente expresión, partiendo de la ecuación para la suma en paralelo de dos resistencias:

$$\frac{1}{R_{eq}} = \frac{1}{R_1} + \frac{1}{R_2} = \frac{R_1 + R_2}{R_1 R_2}; \text{ si } R_1 = R_2 = R \rightarrow \frac{1}{R_{eq}} = \frac{2R}{R^2} = \frac{2}{R} \rightarrow R_{eq} = \frac{R}{2}$$

- C es la capacidad del condensador en faradios.

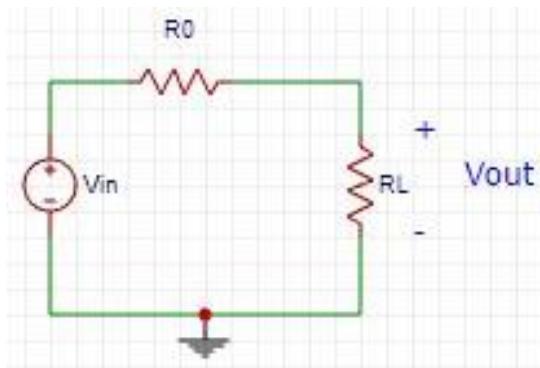
Eligiendo los valores de $C = 10nF$ y $R = 2M\Omega$ se obtiene el siguiente resultado:

$$f_c = \frac{1}{2\pi * 10nF * 1M\Omega} = 15.91Hz$$

5.2.2 Adaptación de impedancias

Para asegurar la correcta transmisión de la señal hay que conectar la guitarra a una impedancia lo más alta posible y conseguir que la impedancia que se conecta a la placa sea lo más baja posible. Además, si la impedancia que se conecta a la siguiente etapa es constante con independencia de la guitarra que se conecte se favorece el buen funcionamiento del dispositivo para cualquier instrumento.

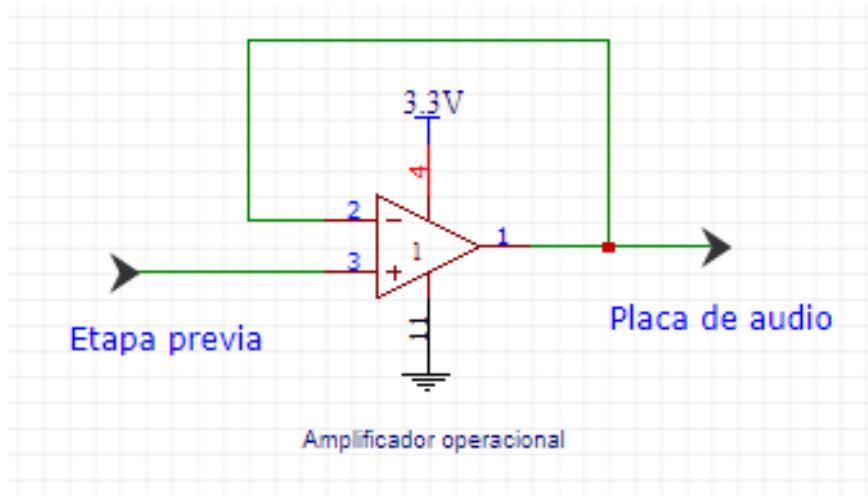
Esto se debe a que el generador de señales usado (la guitarra) no es ideal y tiene una impedancia de salida (R_0) con un valor diferente de 0. Cuando se conecta al generador una impedancia cualquiera (R_L) se podría visualizar la situación como un caso de un divisor de voltajes con un generador ideal como el de la figura 5-5.



5-5 Divisor de voltaje resistivo

Como lo que se busca es transmitir un voltaje al dispositivo representado por la impedancia de carga (R_L), dadas las ecuaciones del divisor de voltajes mostradas en el apartado 5.2.1, para aumentar la cantidad de voltaje que cae en R_L se debe conseguir que ésta sea lo mayor posible con respecto a R_0 .

Conseguir que esto se cumpla entre todos los elementos del circuito es lo que se conoce como adaptación de impedancias. El circuito que se encargará de esta labor será el indicado en la figura 5-6.



5-6 Etapa de adaptación de impedancias

Este circuito constituido por un amplificador operacional en modo seguidor de tensión.

Un amplificador operacional, teóricamente, tiene una impedancia de entrada infinita y una impedancia de salida de 0. A la entrada se conectaría la guitarra y a la salida la placa de audio, con lo cual se solucionarían ambos problemas.

El modo de conexión como seguidor de tensión asegura que las impedancias de entrada y salida del amplificador continúen con el mismo valor y provoca que la amplificación aplicada a la señal de entrada sea de 1, es decir, la dejaría igual.

Para la realización del dispositivo se ha utilizado un LM324-N. Aunque la hoja de especificaciones no informe sobre las impedancias del operacional, se informa sobre el valor de la intensidad de entrada (45nA a 1,4V) y la de salida (25mA a 2V) en situaciones de trabajo normales. Aplicando la ley de Ohm ($V=I \cdot R$) se obtienen las impedancias reales.

$$\frac{1.4 \text{ V}}{45 \text{ nA}} = 31.1 \text{ M}\Omega$$

$$\frac{2 \text{ V}}{25 \text{ mA}} = 80 \text{ }\Omega$$

Como se puede observar, este modelo de operacional aporta una impedancia de salida mucho menor que la que aportaría una guitarra y ofrece una impedancia de entrada conocida y mucho mayor que la impedancia de salida de la guitarra. Por estos dos motivos satisface las características buscadas previamente.

5.3 Programación del afinador

Una vez se contó con un circuito de entrada de audio adaptado al instrumento que permita probar si el código funciona, se pudo comenzar a programar el afinador. Para esta tarea se utilizó la IDE Arduino²⁹ ya que es completamente compatible con Teensy y permite la programación y la carga del código de una manera sencilla.

En este apartado se realizó la configuración de audio de la placa y se adaptó el código existente incluyendo el aumento de resolución y la calibración.

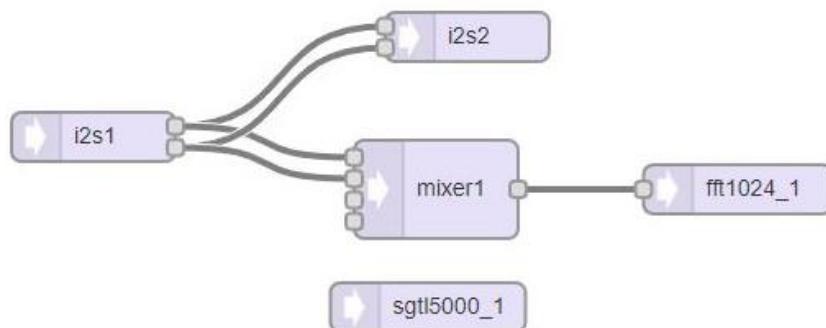
5.3.1 Enrutamiento de audio

El primer paso fue configurar las conexiones de audio interiores de la placa para conseguir que realice las operaciones requeridas.

Para este cometido Teensy dispone de una herramienta web³⁰ que permite configurar estas conexiones de forma gráfica para después generar el código necesario automáticamente. En la imagen 5-7 se puede observar el diseño utilizado para este afinador.

²⁹ <https://www.arduino.cc/en/main/software> [20/9/2017]

³⁰ <https://www.pjrc.com/teensy/gui/> [20/9/2017]



5-7 Esquema del enrute de audio en la placa Teensy.

Donde cada elemento tiene una función diferente:

- **I2s1**: Es el dispositivo que identifica la entrada de audio. Es el encargado de tomar la señal de la entrada Line In y enviarla al resto de dispositivos. A pesar de que la señal recibida de una guitarra tiene un solo canal, en el prototipo y en las pruebas se usó un montaje estéreo dado que los conectores usados lo eran también y resultaba más cómodo. En cada señal estéreo iba la misma señal mono de la guitarra que, por tanto, estaba duplicada.
- **I2s2**: es el dispositivo de salida de audio. Por defecto envía el audio a la salida JACK de 3.5 de la placa de audio Teensy.
- **Mixer1**: es un sumador de audio. Suma la señal obtenida por ambos canales de la entrada. Al ser la misma señal lo que se consigue es simplemente un aumento de la amplitud.
- **Fft1024_1**: es el módulo encargado de hacer la FFT. La placa cuenta con una FFT de 1024 puntos. Se comienza a calcular automáticamente en cuanto se inicializa el módulo vía software.
- **Sgtl5000_1**: este dispositivo es el controlador de la placa de audio. Es necesario incluirlo porque permite configurar parámetros como la ganancia y canales de entrada, el tamaño destinado a la memoria de audio y además permite inicializar la captura de audio.

5.3.2 Frecuencia de muestreo

Como se ha explicado anteriormente, la elección de la frecuencia de muestreo resulta crucial de cara a permitir al menos una diferenciación de las diferentes cuerdas debido a la baja cantidad de puntos usada en la FFT de la placa Teensy.

Esta placa muestra una limitación relacionada con la frecuencia de muestreo y es que, a priori, no se puede modificar porque el autor de la librería sólo proporciona una frecuencia de muestreo fija en la misma. El diseño utilizado fija esta frecuencia a 44.1kHz. Esto, como ya se ha calculado anteriormente, resulta en una precisión extremadamente baja, ya que las cuerdas más graves pasan a estar representadas en el mismo bin, imposibilitando la obtención de información sobre cada una de forma independiente.

Sin embargo, en la hoja de características del chip controlador de la placa, el SGTL5000, se indica que admite frecuencias de muestreo desde 8kHz. Por debajo de esta frecuencia se desconoce el funcionamiento.

Esta frecuencia permitiría obtener $8000/1024 = 7.8125$ Hz de distancia entre bins, resultando al menos suficiente para distinguir en bins diferentes las cuerdas más graves para realizar la interpolación.

Dado que la frecuencia de muestreo no viene fijada por hardware, ya que el chip permite otros valores, estará fijada por software en alguna de las librerías de la placa.

Tras una investigación sobre el funcionamiento de la placa se encontró que para modificar la frecuencia es necesario editar los archivos *AudioStream.h* y *output_i2s.cpp* que se pueden encontrar en las librerías de Teensy.

Para modificar los archivos correctamente fue necesario realizar ciertos cálculos.

El sistema calcula la frecuencia de muestreo basándose en el reloj del procesador. En este caso está configurado a 96Mhz. A partir de este valor, en el archivo *output_i2s.cpp* se calcula la frecuencia de muestreo de la siguiente forma:

$$96\text{Mhz} * \frac{\text{MCLK_MULT}}{\text{MCLK_DIV}} * \frac{1}{256}$$

Donde MCLK_MULT y MCLK_DIV son unas variables de tipo entero que se deben configurar en el fichero para conseguir el valor que se busque de frecuencia de muestreo. Igualando la expresión anterior a 8kHz y probando combinaciones de valores para MCLK_MULT y MCLK_DIV se obtuvieron los valores MCLK_MULT = 4 y MCLK_DIV = 188.

Por lo tanto, la frecuencia de muestreo real es:

$$96\text{Mhz} * \frac{4}{188} * \frac{1}{256} = 7978,72 \text{ Hz.}$$

Si bien no es exactamente 8kHz, dado que la frecuencia original generada tampoco era de 44.1kHz exactamente (como se pudo observar al leer los archivos originales) se puede suponer que el dispositivo admite pequeñas variaciones alrededor de los valores especificados.

Una vez calculada la frecuencia de muestreo resulta necesario definirla en el archivo *AudioStream.h* sobrescribiendo la variable *AUDIO_SAMPLE_RATE_EXACT*.

5.3.3 Adaptación del código

Para la programación de la placa Teensy se utilizó la versión adaptada de C/C++ utilizada por la IDE Arduino. Por tanto, se necesitó modificar el código Python anterior para adaptarla a este lenguaje.

Sin embargo, más que la propia adaptación del lenguaje, en este paso supuso un reto la sustitución de métodos como el de interpolación ya que, al no haber librerías específicas para cálculo matemático como en Python, fue necesario programarlos a mano.

Además, dado que en el caso de la placa Teensy la FFT y la captura de audio se realiza automáticamente no es necesario ejecutar una función de *callback* de manera explícita cuando se lee el fragmento de audio determinado.

En este caso se realizan todas las operaciones en el método *loop()*, que se ejecuta continuamente una y otra vez hasta que se apague el dispositivo. Como las operaciones realizadas sólo se ejecutan al tener información sobre la FFT, sólo se ejecuta el código dentro del método *loop()* cuando el dispositivo indique que existe una FFT disponible mediante el método *fft1024.available()*.

Como la placa permite la realización de 130 FFTs por segundo, para controlar que la tasa de actualización de los datos no sea molesta para el usuario tanto en el *serial port* como en la pantalla se fuerza una espera de 100ms entre dos análisis consecutivos.

Además de todo lo ya comentado, un punto muy importante que necesita un cálculo nuevo al cambiar de plataforma es el relativo a los márgenes de detección.

Como la placa Teensy presenta una frecuencia de muestreo diferente de la del programa Python la posición de los *bins* es diferente. Por este motivo los valores de los intervalos de búsqueda utilizados no coinciden con valores de frecuencia.

Para calcular los nuevos intervalos hay que dividir la frecuencia límite entre la nueva distancia entre *bins*:

$$f_{Teensy} = \frac{f_{Python}}{7.8125}$$

Realizando esta operación se obtuvieron los siguientes intervalos de *bins*:

Tabla 5-1 Límites de búsqueda en bins para cada cuerda.

Número de cuerda	Frecuencia a detectar (Hz)	Límite inferior del intervalo de búsqueda (Nº de Bin)	Límite superior del intervalo de búsqueda (Nº de Bin)
1	988,89	110	146
2	740,82	89	110
3	196,00	22	29
4	146,83	16	22
5	110,00	12	16
6	82,41	9	12

Como estos intervalos indican un número de *bin* se han redondeado al entero más cercano para obtener los valores reales a utilizar.

Dado que un afinador real no muestra las frecuencias concretas leídas, esta información, traducida a frecuencias, se muestra por *serial port* en la primera versión del prototipo. De este modo se puede ejecutar la calibración del afinador y se puede comprobar si se consigue la precisión esperada.

5.3.4 Comprobación de la precisión

Del mismo modo que en el programa Python, se analizó la capacidad del afinador físico para detectar como diferentes dos tonos separados por 1 cent.

Al contrario que en Python, debido a la resolución del algoritmo de la FFT (7,792 Hz), se obtiene que ninguna cuerda coincide exactamente con un *bin*, permitiendo calcular el número de muestras necesarias usando sólo la más restrictiva, es decir, la sexta.

Durante las pruebas con el archivo de audio³¹ generado en MATLAB, a partir de 100 muestras se consiguió diferenciar la variación de frecuencias. Sin embargo, la FFT no ofrece valores tan estables como en Python debido seguramente a la menor resolución inicial. Esto causa que los valores calculados oscilen entre varios valores cercanos. En el caso de 100 muestras algunos de estos valores son comunes para ambos tonos, pudiendo causar confusión.

Como el sistema, computacionalmente, aún permitía generar muchas más muestras, se aumentó el número de muestras hasta que se pudo obtener una

³¹ Mismo archivo de audio que el usado en el apartado 4.4.3 para la sexta cuerda.

resolución algo mayor que permitiera caracterizar mejor estos valores obtenidos mediante un cálculo de su media y su varianza.

En 200 muestras se decidió que la variación entre los dos valores era suficiente ya que el número de apariciones de los valores comunes había descendido hasta aproximadamente un 5% de los valores mostrados.

En la Tabla 5-2 se muestran la media y la varianza de los valores leídos durante la prueba para ambas frecuencias.

Tabla 5-2 Media y varianza obtenidas para los audios de prueba a la frecuencia de referencia y a la frecuencia de referencia menos 1cent.

	Frecuencia de referencia	Frecuencia de referencia -1 cent.
Media	82,6961	82,6324
Varianza	0,0057	0,0072

A la vista de los resultados, se eligió el número de 200 muestras generadas para la interpolación.

5.3.5 Calibración

Como se pudo observar en el apartado 5.3.4, los valores calculados por la FFT del microcontrolador son más variables que los obtenidos por el *script* en Python.

Como resulta complicado obtener un valor fijo aun usando un tono sintético, que es una de las señales más constantes que se pueden obtener, no se puede realizar la calibración del mismo modo que en Python.

En Python bastaba con calcular un *offset* que, aplicado a las frecuencias de referencia, permitía obtener un valor que se podía comparar. Al ser más difícil obtener un valor constante a comparar, este procedimiento no resulta válido para este caso, forzando a escoger un rango de valores entre los que cada cuerda se pueda considerar afinada.

Dado que la resolución objetivo del sistema es de 1 cent de tono, todas las frecuencias que se encuentren en un intervalo de 1 cent tanto por encima como por debajo de la frecuencia de referencia deben ser aceptadas.

Debido al sistema de interpolación no se pueden calcular las frecuencias correspondientes de forma directa, de modo que se generó un archivo de audio para cada cuerda que contenía en los primeros 15 segundos la frecuencia teórica 1 cent inferior a la referencia y los siguientes 15 la frecuencia 1 cent superior.

Calculando la media de los valores obtenidos para cada tono se pueden obtener ambos márgenes, tanto el superior como el inferior. El uso de la media permite

que, si la cuerda se aleja justo 1 cent de la frecuencia original, los valores variables que devuelve el afinador se encuentren en ocasiones por debajo y en ocasiones por encima del umbral elegido. Esta oscilación de los valores, al verse en la pantalla como una variación de los símbolos asignados a cada estado, ofrece información de una variación justa de 1 cent, lo cual cumple la resolución objetivo del sistema.

Tras la realización de las pruebas se obtuvieron los márgenes mostrados en la Tabla 5-3. Nótese que, al haber realizado la interpolación, los valores que limitan los intervalos se muestran en Hz porque se comparan valores frecuenciales, al contrario que ocurría en el apartado 5.3.3, donde se comparaban valores de *bin*.

Tabla 5-3 Definición de los límites en Hz del intervalo de frecuencias en los que se considera afinada una cuerda.

Número de cuerda	Límite inferior del intervalo “afinado” (Hz)	Límite superior del intervalo “afinado” (Hz)
1	988,6502	989,4708
2	740,4910	740,9928
3	195,6426	195,8060
4	147,0298	147,2196
5	109,7334	109,8457
6	82,6539	82,7871

5.3.6 Sistema de representación

El sistema de representación del afinador debe permitir conocer el estado de todas las cuerdas a la vez y ofrecer la mayor visibilidad posible. Además, dado que un pedal de guitarra se sitúa habitualmente en el suelo, deberá ser lo suficientemente grande como para poder leerse.

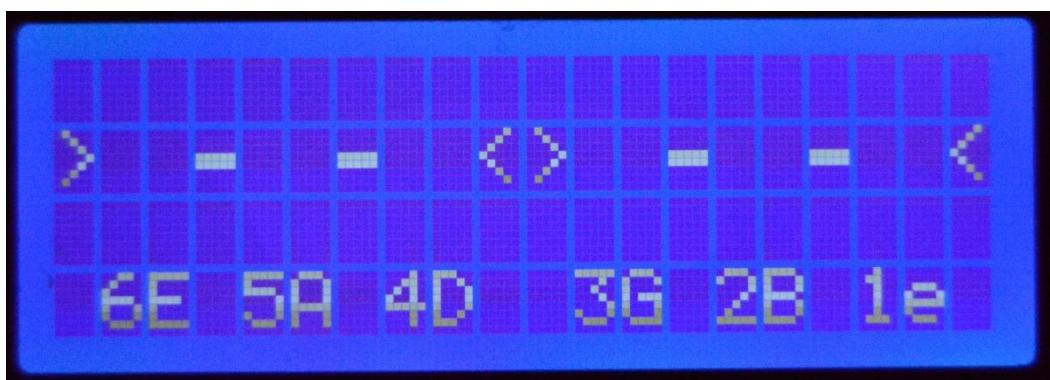
El sistema ideal actualmente y más utilizado son distribuciones de LEDs adaptadas al afinador, debido a su alto brillo y contraste si se colocan sobre fondo negro. Sin embargo, realizar una pantalla adaptada al afinador es bastante complejo ya que requiere piezas a medida. Este problema se hace más importante dado el carácter DIY del dispositivo, que intenta hacer el montaje lo más sencillo posible.

Para el prototipo se usará una pantalla de caracteres LCD de tamaño 20x4 retroiluminada.

Este sistema tiene un tamaño y un brillo suficientes para ser visualizado estando en el suelo por una persona de pie. Además, es un producto genérico y se puede conectar muy fácilmente a la placa mediante el protocolo I2C.

Este protocolo permite usar sólo dos pines de la placa Teensy para hacer funcionar el display debido al uso de un bus de datos. El problema de otros sistemas estaría en que la conexión de la placa con el microcontrolador Teensy y la placa de audio usa muchos pines y físicamente puede ser difícil de conectar. Con este sistema se reutilizan dos pines ya usados por la placa de audio, facilitando el montaje. Estos pines, al ser buses, permiten la comunicación con varios dispositivos a la vez sin interferencias asignándoles una dirección.

La representación del afinador cuenta con unos gráficos fijos que sirven de *layout* para ubicar cada cuerda y como referencia y 6 zonas de gráficos móviles que dependen del estado de cada cuerda.



5-8 Pantalla LCD mostrando el layout fijo.

Los gráficos fijos, mostrados en la imagen 5-8, incorporan el número y la nota en notación inglesa de cada cuerda y unas flechas y líneas que sirven para indicar dónde deben estar los cursores para considerar la cuerda correctamente afinada.

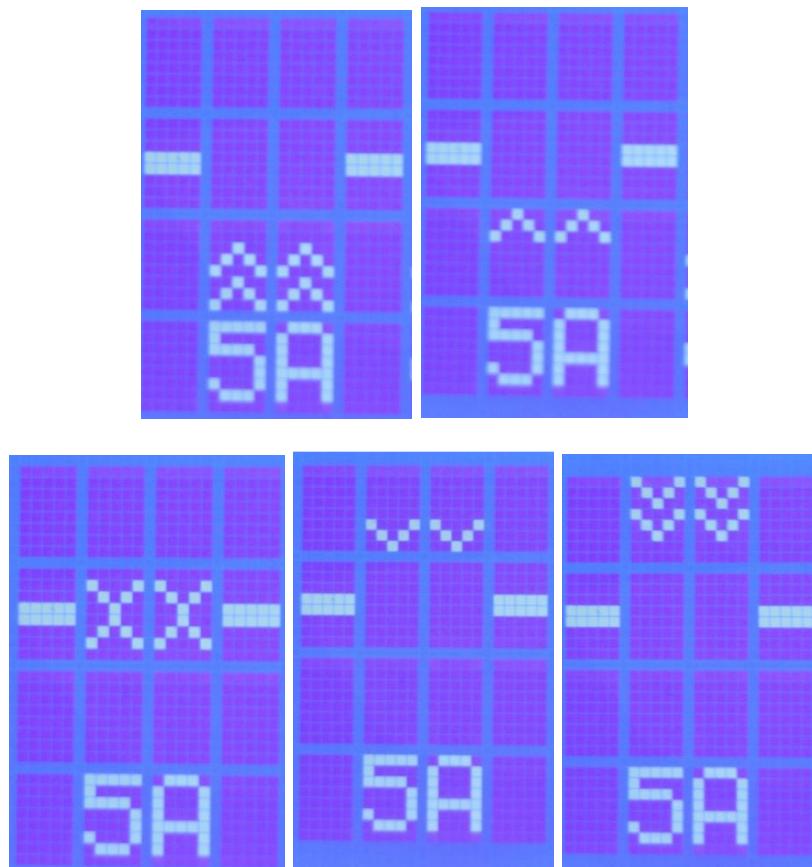
Los gráficos móviles están situados cada uno encima del nombre de la cuerda correspondiente y toman una forma u otra dependiendo del estado de la cuerda.

Como no se han encontrado símbolos prediseñados en el alfabeto del LCD que sean apropiados se usan seis nuevos símbolos que se definen como una matriz de píxeles de 8x5.

Estos símbolos se muestran en la imagen 5-9 y toman los siguientes significados:

- **Símbolo 1.** Indica que la cuerda se encuentra más de 20 cents por debajo de la frecuencia de referencia.
- **Símbolo 2.** La cuerda se halla menos de 20 cents por debajo de la referencia.

- **Símbolo 3.** La cuerda está afinada correctamente, es decir, en el intervalo definido en el apartado 5.3.5.
- **Símbolo 4.** La cuerda se halla menos de 20 cents por encima de la referencia.
- **Símbolo 5.** La cuerda se halla más de 20 cents por encima de la referencia.



5-9 Arriba izquierda: símbolo 1. Arriba derecha: símbolo 2. Abajo izquierda: símbolo 3. Abajo centro: símbolo 4. Abajo derecha: símbolo 5.

Se ha usado el valor de 20 cents debido a que, como se puede ver en los afinadores de la imagen 4-5, es el rango habitual de representación en los modelos comerciales.

El uso de este LCD requiere una adaptación de niveles lógicos entre él y la placa Teensy, ya que la placa envía datos a 3.3V mientras que el LCD los recibe a 5V, haciendo imposible una conexión directa. Si bien introducir valores de 3.3V a un dispositivo que funciona con 5V no cause un problema, dado que el bus I2C es bidireccional, los datos a 5V que envíe el LCD a la placa podrían causar daños por tener un voltaje mayor a 3.3V.

Esto se soluciona usando adaptador de niveles entre ellos, de forma que los 3.3V de la Teensy se convierten en 5V y viceversa, permitiendo que los datos sean leídos por el LCD y que los datos del LCD no quemen la placa.

5.3.7 Sistema de alimentación

Para conseguir que el afinador se pueda usar sin necesidad de estar enchufado a un ordenador fue necesario diseñar una etapa de alimentación.

Aunque no hay un estándar oficial en cuanto a la alimentación de los pedales de la guitarra, se suele usar una pila de 9V para alimentarlos. Por tanto, este tipo de pilas son la fuente utilizada en este dispositivo. Dado que tanto la pantalla LCD como la placa Teensy funcionan a 5V fue necesario utilizar un conversor de voltaje de 9V a 5V.

Existen dos tipos de conversores, cada uno con características diferentes:

- **Lineales:** estos conversores son pequeños y baratos. Sin embargo, tienen ciertas limitaciones relativas a la potencia que puede pasar por ellos debido al calor que generan para realizar la conversión. El calor generado es producido por la potencia que disipan, siendo poco eficientes en cuanto a consumo.
- **Comutados:** son ligeramente más grandes y caros que los anteriores. Sin embargo, no generan tanto calor, por lo que resultan más apropiados para usarlos en dispositivos que no vayan a contar con ventilación. Además, al no calentarse permiten alimentar a más dispositivos al no estar tan limitados en potencia y ser más eficientes en la conversión.

Los afinadores requieren de una pantalla para mostrar los resultados. Como estas pantallas suelen ser grandes y tener bastante brillo para mejorar la visibilidad son el elemento que más consume del pedal. Por tanto, será necesario que el conversor pueda suplir con la suficiente potencia al afinador y, si es eficiente, mejorará la autonomía al usarse con pila.

Además, el comutado al no calentarse permite que el dispositivo sea más pequeño y la caja no necesite ventilación, reduciendo los costes al hacerla más sencilla. Estos motivos causan que se use un conversor de tipo comutados para el afinador.

6. Conclusiones

En este apartado se comentan los resultados de las pruebas realizadas al prototipo físico del afinador y se proponen mejoras que se podrían hacer en un futuro y que no se han podido realizar por falta de tiempo o equipamiento.

6.1 Resultados

Para comprobar el funcionamiento del afinador se ha realizado la tarea de afinación en repetidas ocasiones partiendo de una guitarra previamente desafinada.

Se han observado ciertos comportamientos del afinador:

- Durante el primer medio segundo después de golpear las cuerdas los valores de afinación son bastante aleatorios. Esto era de esperar ya que el propio golpeo de las cuerdas provoca que tarden un tiempo en estabilizarse a la frecuencia de resonancia. Por tanto, al igual que en todos los demás afinadores del mercado, hay que esperar hasta que el valor se stabilice.
- Después, durante un tiempo que oscila entre 5 y 8 segundos dependiendo de la cuerda, los valores alcanzan su punto más estable. Si bien los indicadores de que la cuerda está desafinada se visualizan perfectamente, el indicador de que la cuerda está correctamente afinada no. Este indicador se mantiene demasiado poco tiempo fijo, sobre todo en las cuerdas más graves (5 y 6), variando continuamente entre los símbolos que indican que la cuerda está ligeramente desafinada tanto por arriba como por abajo. Si bien los afinadores comerciales tampoco mantienen ese nivel demasiado tiempo, lo mantienen durante bastante más. Esta falta de estabilidad puede deberse a varios factores que actúan conjuntamente:
 - Falta de resolución en la FFT inicial. El hecho de que esta falta de estabilidad sea mucho más notable en el afinador físico que en el prototipo Python hace pensar que la resolución de la FFT tenga un impacto sobre esto ya que la mayor diferencia entre ambos afinadores es que cuentan con FFTs de diferentes puntos.

Además, en las cuerdas agudas (1 y 2) la visualización de este símbolo es mejor, ya que el intervalo “afinado” tiene un ancho de banda mayor que en las cuerdas graves. Esto también lleva a pensar que la resolución de la FFT afecte a la visualización.

- La inexactitud en la vibración de las cuerdas. El hecho de que las cuerdas no vibren perfectamente y no se mantengan a una frecuencia concreta durante toda la vibración puede provocar que la FFT modifique su amplitud ligeramente. Esto, junto con la falta de resolución de la FFT puede causar que la interpolación se desvíe más de lo esperado.
- Despues de ese momento, el nivel de la señal empieza a decaer y los valores de afinación vuelven a ser aleatorios.

A pesar de los errores de visualización del afinador en las cuerdas más graves, las pruebas de afinación se pueden considerar positivas ya que, salvando el hecho de que se necesita ajustar la cuerda con mayor precisión para conseguir que se muestre el símbolo de “afinada” al menos durante medio segundo, la guitarra se afina correctamente. Comparando la guitarra afinada con otros afinadores comerciales como el Korg GA-30 y el Thomann CTM-700³², se puede observar cómo estos muestran que está correctamente afinada. Además, se ha comprobado también que suena bien al tocarse junto con otra guitarra afinada según otro sistema de referencia.

Estos resultados son satisfactorios debido a que se trata de un prototipo, aunque quizás un producto final necesite mejorar la estabilidad en la visualización para facilitar su uso y poder ser competitivo.

6.2 Trabajos futuros

Aunque se haya llegado a una solución funcional, el dispositivo se puede mejorar de muchas maneras con el suficiente tiempo y equipamiento, realizando pruebas más exhaustivas e implementando nuevo código.

Las principales mejoras que se podrían implementar son:

- **Añadir más afinaciones.** Si bien la afinación estándar es la más común, se varía de afinación bastante a menudo. Incorporando nuevas referencias se podría elegir la afinación que se quiere usar para hacerlo más polivalente.
- **Detector de afinaciones.** La mejora anterior permite la mejora de evitar tener que elegir la afinación manualmente y hacer que el dispositivo detecte la afinación más probable en función de las notas más cercanas a la frecuencia de cada cuerda. Un ejemplo de afinaciones muy comunes usadas por los guitarristas son las afinaciones en *drop*. En estas

³² https://www.thomann.de/es/thomann_ctm700.htm [Último acceso: 20/9/2017]

afinaciones se cambia la relación entre la sexta cuerda y las demás, bajándola un tono, mientras que las demás mantienen su relación habitual. Es decir, en *drop D*, por ejemplo, sólo se bajaría la sexta cuerda un tono y el resto continuarían en estándar, por tanto, la afinación sería DADGBE mientras en estándar es EADGBE.

- **Mejorar la indicación de afinación correcta.** Si bien pocos afinadores consiguen mantener la indicación de “bien afinado” al tocar una cuerda debido a las variaciones que introduce la propia cuerda, se podría incrementar el tiempo que se mantiene modificando las tasas de actualización de la pantalla o la frecuencia de muestreo original para aumentar la estabilidad de los resultados. En algunas soluciones comerciales monofónicas se puede observar cómo, al llegar la cuerda al valor afinado, la pantalla directamente deja de actualizarse para mantener ese valor. Esto se ha comprobado modificando rápidamente la afinación de la guitarra mientras el afinador marca ese valor y observando que tarda cierto tiempo en responder.
- **Comparar el funcionamiento en más guitarras.** A más guitarras se puedan analizar, mejor se podrá caracterizar la respuesta en frecuencia de estas y, por tanto, será más preciso el afinador en general. En este trabajo sólo se ha podido contar con una guitarra, lo cual ha limitado la información disponible al analizar el audio.
- **Diseñar una pantalla adaptada.** Un indicador adaptado al afinador en forma de LEDs individuales mejoraría la visibilidad en un espacio más reducido que el usado por una pantalla LCD genérica, ya que esta tiene mucho espacio no utilizado. Además, podría mejorar la estética del afinador.
- **Afinador software.** Tal y como están realizados los *scripts* en Python se deja el camino abierto al diseño de una versión software del afinador. Con las modificaciones pertinentes para permitir el uso en tiempo real y una interfaz gráfica se conseguiría un *software* completamente funcional.
- **Diseño de una PCB específica.** La implementación del afinador en una PCB adaptada conseguiría reducir el tamaño, incrementando su portabilidad. Además, haría el sistema más resistente a errores debidos a componentes mal conectados al facilitar el montaje.
- **Diseño de una carcasa/encapsulado para el sistema.** Esta mejora, junto con el diseño de la PCB y la pantalla conseguirían dotar al sistema de un uso práctico real, comparable al que puede tener cualquier otro afinador en formato pedal como sería el Polytune.

Anexos

1. Marco regulador

Debido a que el afinador diseñado es sólo un prototipo no se ha buscado que cumpla ninguna normativa. Además, dado el carácter DIY del dispositivo, no tendría por qué cumplirse ninguna normativa para la distribución gratuita de esquemáticos e instrucciones de montaje, ya que el montaje correría a cargo de quien lo construyera. En cambio, si se quisiera comercializar, se deberían tener en cuenta las normativas explicadas en este apartado, referentes a los componentes usados, a la calidad de la afinación y al aseguramiento de la seguridad del usuario.

1.1 RoHS

Para que el producto pueda ser apropiado para la venta debe cumplir la normativa RoHS^[12] relativa a los materiales por los que está formado el dispositivo. Concretamente el afinador se encontraría dentro de la categoría 11 de RoHS.

Es seguro que el prototipo no la va a cumplir por el hecho de usar soldadura con alto componente en plomo. En caso de buscar una comercialización del afinador debería usarse soldadura sin plomo y asegurar que todos los componentes utilizados cumplen esta normativa de forma individual y en conjunto.

1.2 ISO 16:1975

Esta normativa ISO especifica que la frecuencia para la nota A (La) central del piano debe ser de 440Hz. Además, especifica que la afinación a esa frecuencia de los instrumentos debe conseguirse en un rango de 0,5 Hz.

El prototipo de afinador realizado en este trabajo cumpliría la normativa, ya que, como se ha calculado durante esta memoria, se consiguen resoluciones mejores a frecuencias más bajas. Aunque la guitarra no se afine a 440Hz por ser más grave, esa resolución conseguida permite que, si fuera necesario afinar a esa frecuencia, se conseguiría con mayor precisión.

1.3 Real Decreto 110/2015

Según este Real Decreto^[13] sobre residuos de aparatos eléctricos y electrónicos, el productor del dispositivo debe cumplir ciertas exigencias con el fin de reducir los residuos producidos por estos sistemas. Al igual que las otras normativas, sólo sería necesario aplicarla para la venta comercial del afinador y no aplica sobre el prototipo ni sobre las unidades DIY.

Entre otras, estas exigencias, está la de informar sobre el número de unidades puestos en el mercado, lo cual está regulado por el Registro Integrado Industrial. Este registro asigna al productor un número identificatorio necesario para comercializar estos productos en España. Este número debe estar presente en todas las transacciones realizadas en la venta de estos productos o en la página web en caso de venta a distancia.

Además, se estipula que se deben diseñar los sistemas para prolongar lo máximo posible su vida útil con el fin de reducir los residuos generados. Además, al concluir su vida útil se debe facilitar la preparación para la reutilización. Tampoco se puede impedir la reutilización de productos usados previamente mediante sistemas de fabricación o diseño específicos, a no ser que supongan una mejora importante sobre la seguridad o para la protección del medio ambiente.

2. Presupuesto

Para la realización del prototipo se requirieron ciertos materiales y componentes. Si bien seguramente no sean exactamente los mismos usados en el producto final, un estudio del presupuesto utilizado para su realización puede ayudar a conocer de manera aproximada el precio del producto final.

Las variaciones que haya entre el prototipo y el producto final serán principalmente debido a los formatos usados para su construcción. Por ejemplo, para ahorrar tamaño, no se comprarían el microcontrolador y la placa de audio ensamblados en la PCB. Se comprarían los chips por separado, lo cual es más barato y se montarían en otra placa de una manera más apropiada al diseño específico que se está realizando. Lo mismo ocurriría con la pantalla LCD, ya que se podría usar otra más pequeña o con el conector JACK, que serviría uno mono en vez de estéreo, lo que a su vez causa que se necesiten la mitad de componentes de la etapa de entrada. Es decir, para el prototipo final se optimizarán los costes de producción.

Por tanto, el presupuesto utilizado para el prototipo, separando mano de obra y materiales, fue:

Tabla anexa 2-A Costes de los materiales del prototipo.

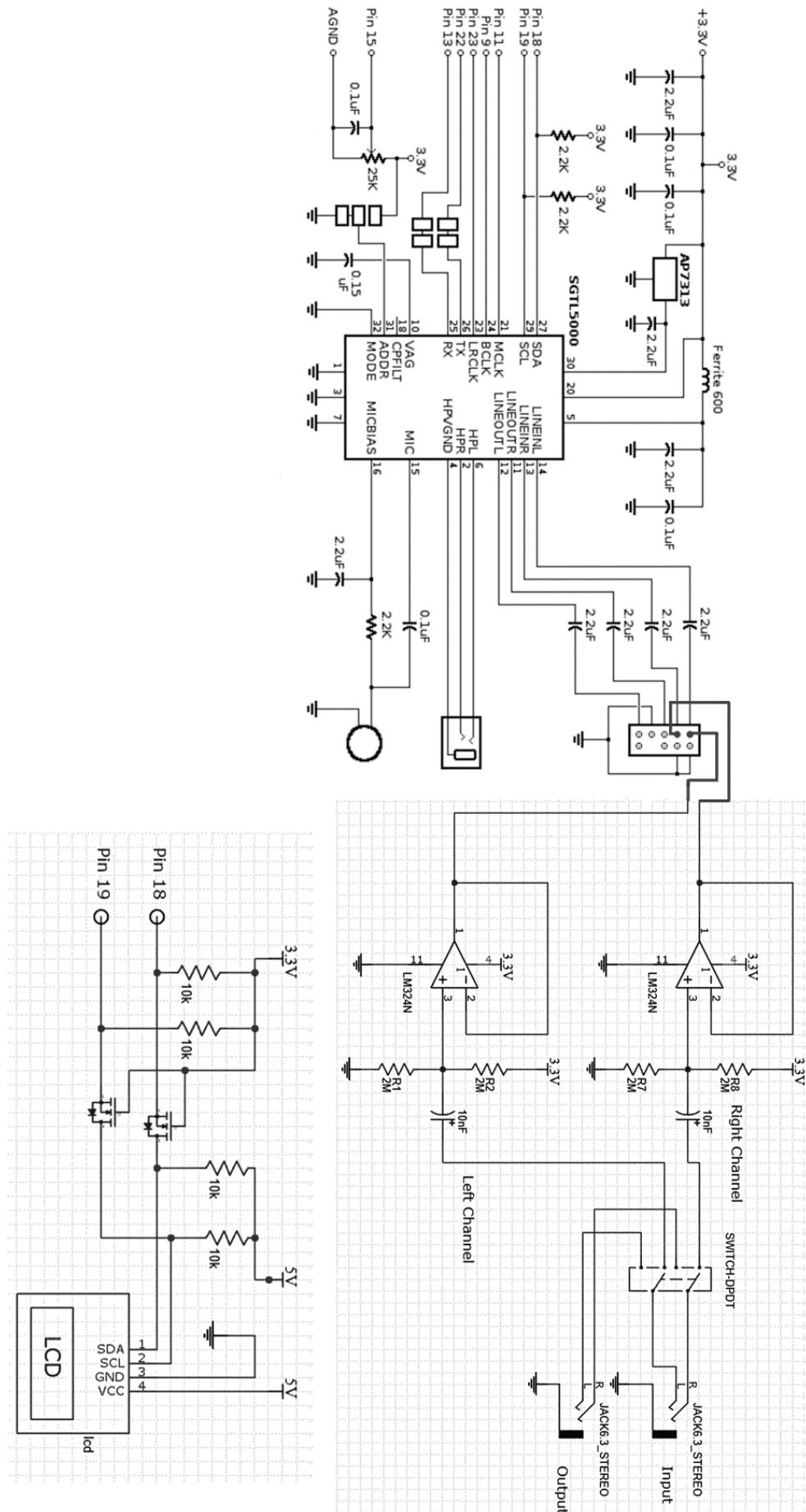
Nombre/modelo del producto	Concepto	Cantidad	Coste unitario (€)	Coste total (€)
Teensy 3.2	Microcontrolador	1	21,04	21,04
Teensy audio adaptor	Adaptador de audio	1	16,80	16,80
Neutrik Rean NYS 216 G	Conector JACK 6,3 hembra	1	0,59	0,59
Texas Instruments LM324N	Amplificador operacional	1	0,40	0,40

Resistencia 2MΩ 0,25W	Resistencia	4	0,04	0,16
Condensador 10nF	Condensador	2	0,15	0,30
Placa protoboard 165x55x10mm	Placa de montaje	1	5,00	5,00
LM2596	Regulador voltajes 9V-5V	1	2,56	2,56
I2C Logic Level Converter	Conversor lógico 3.3V-5V	1	1,89	1,89
IIC / I2C / TWI LCD2004 / 20x4	Pantalla LCD	1	10,99	10,99
Cables	Cables	2 (metros)	0,50	1,00
Licencia MATLAB 2017	Licencia de estudiante facilitada por la UC3M	1	0	0
TOTAL		60,73		

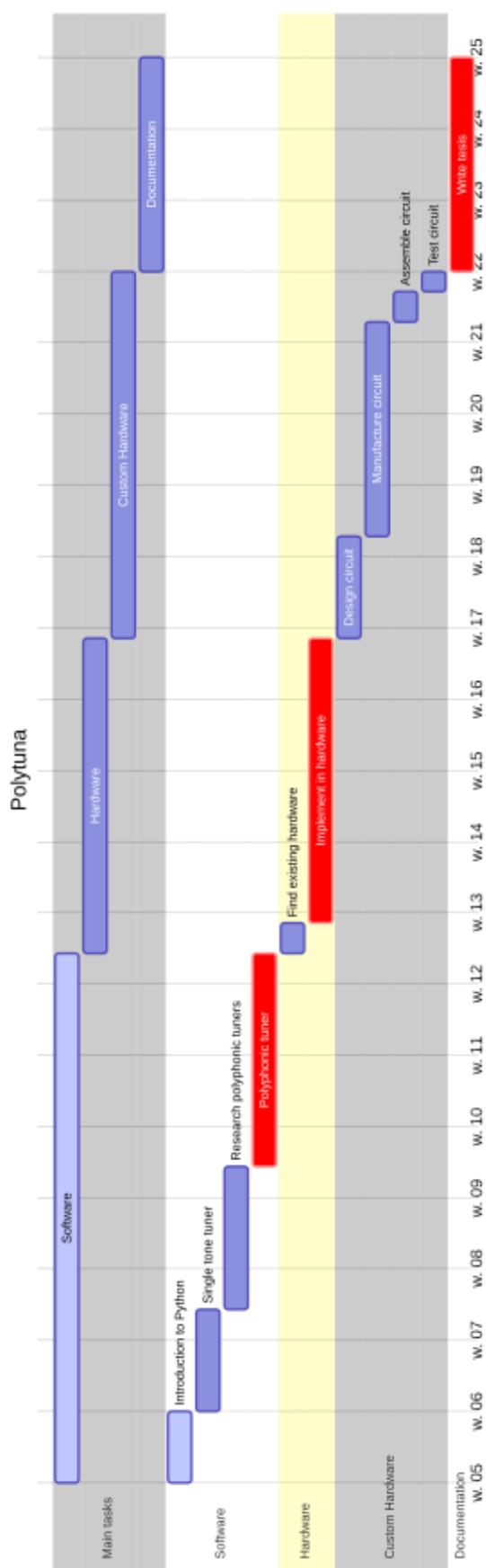
Tabla anexa 2-B Costes de la mano de obra.

Nombre/modelo del producto	Cantidad	Coste unitario (€)	Coste total (€)
Mano de obra (desarrollo)	360	20	7200
Mano de obra (montaje unitario)	1	15	15
TOTAL		7215	

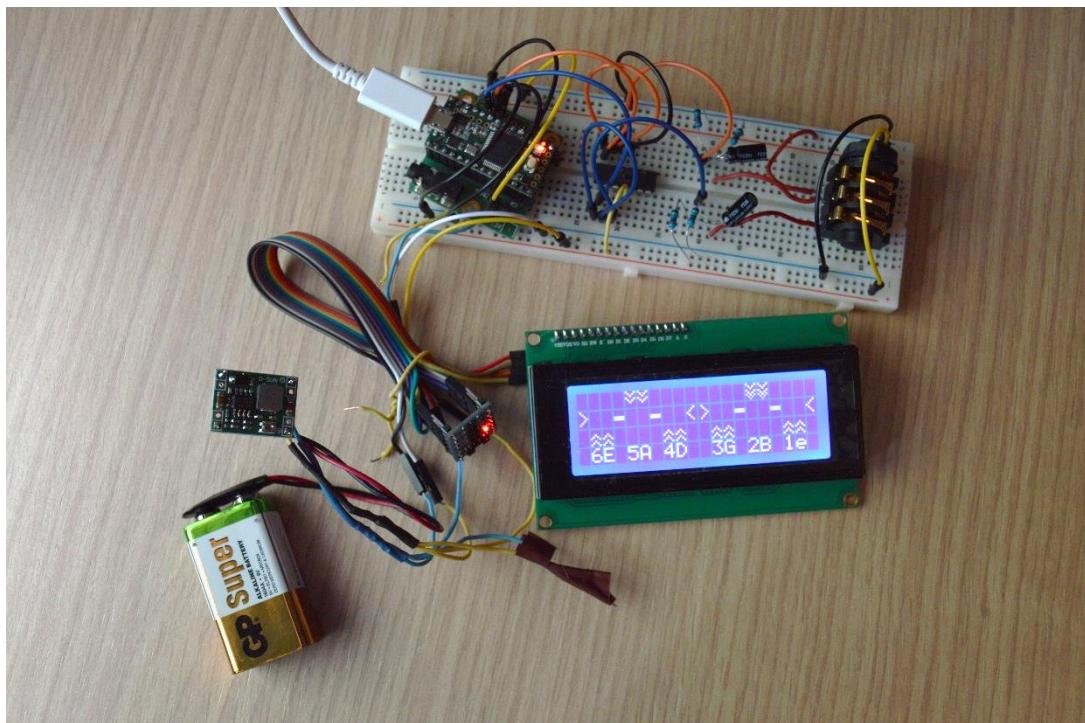
3. Esquemático del montaje físico



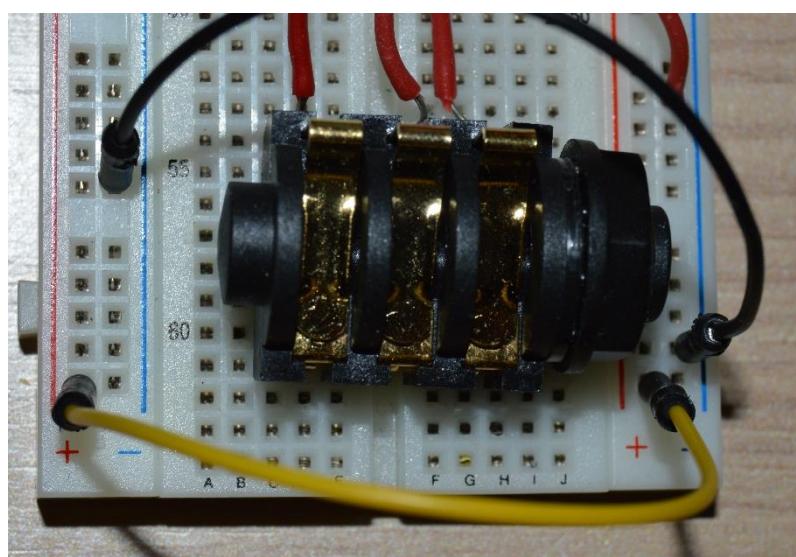
4. Diagrama de Gantt del proyecto



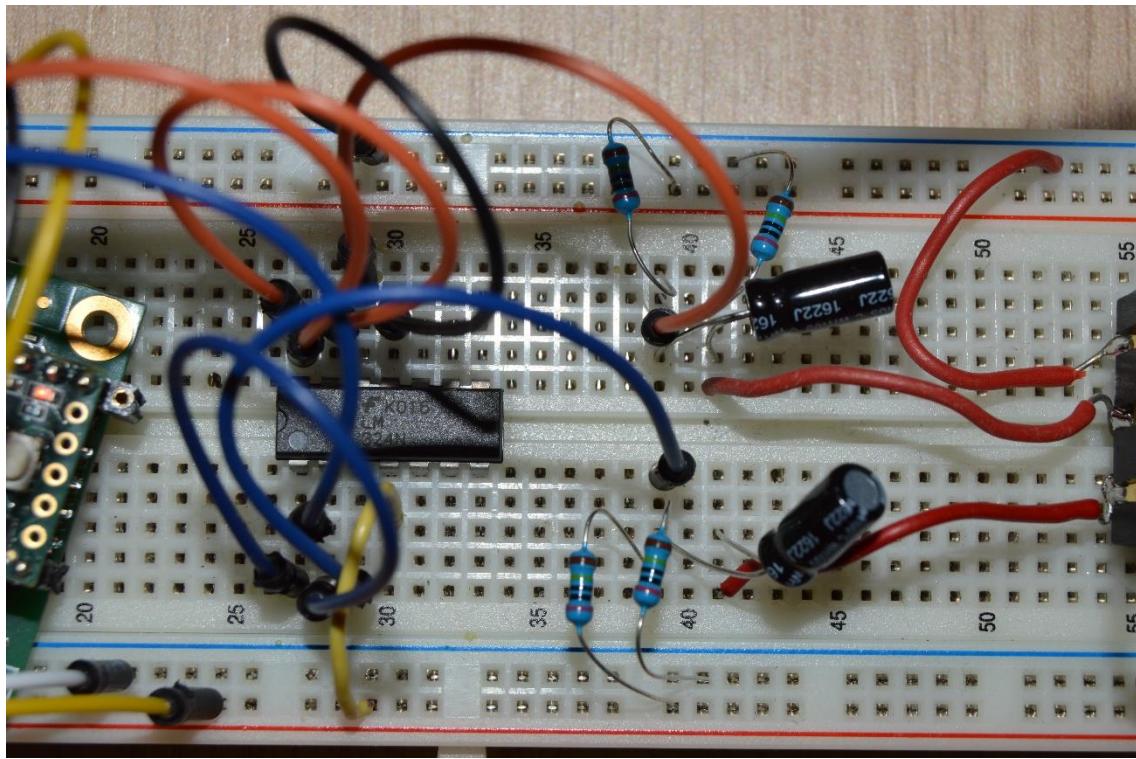
5. Fotografías del sistema físico



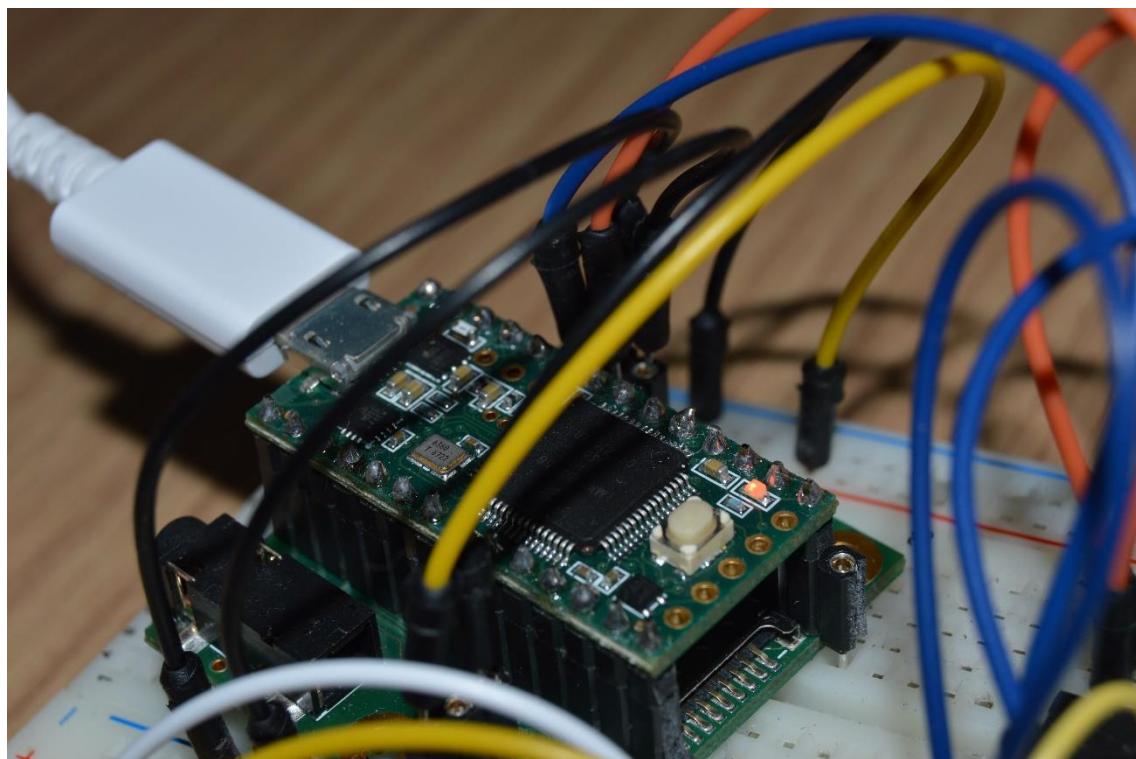
5-A Prototipo. Montaje completo.



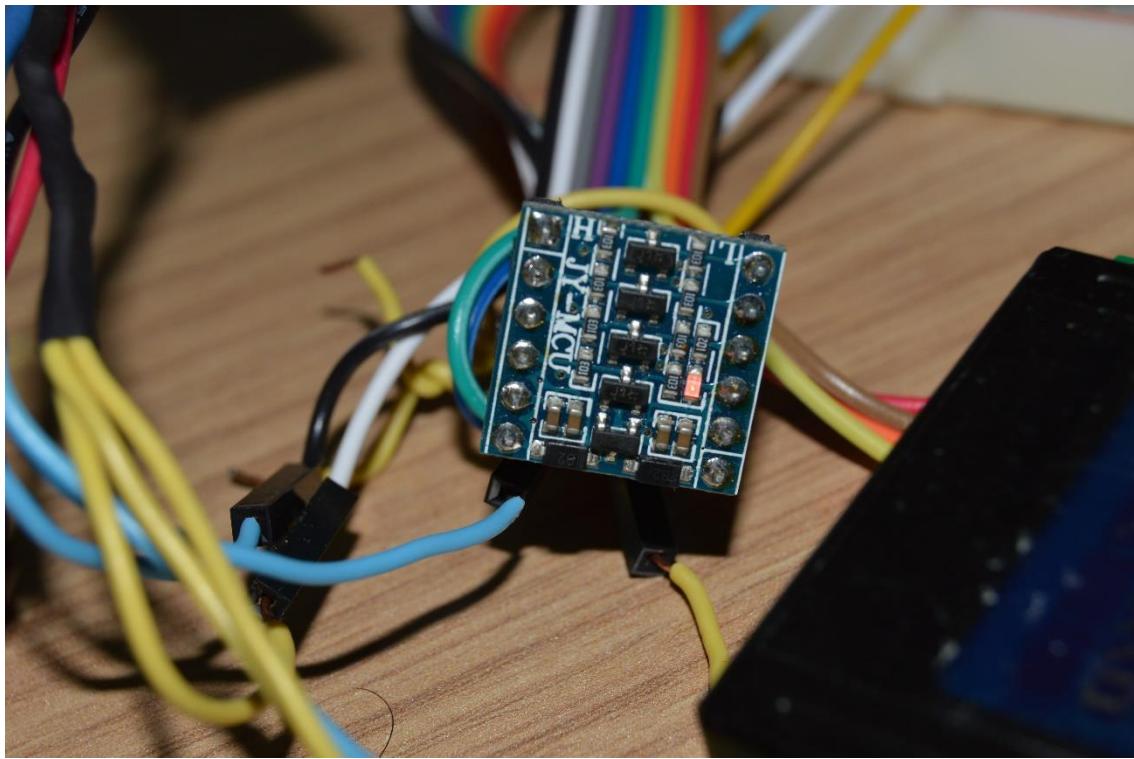
5-B Conector JACK 6,3mm estéreo de entrada.



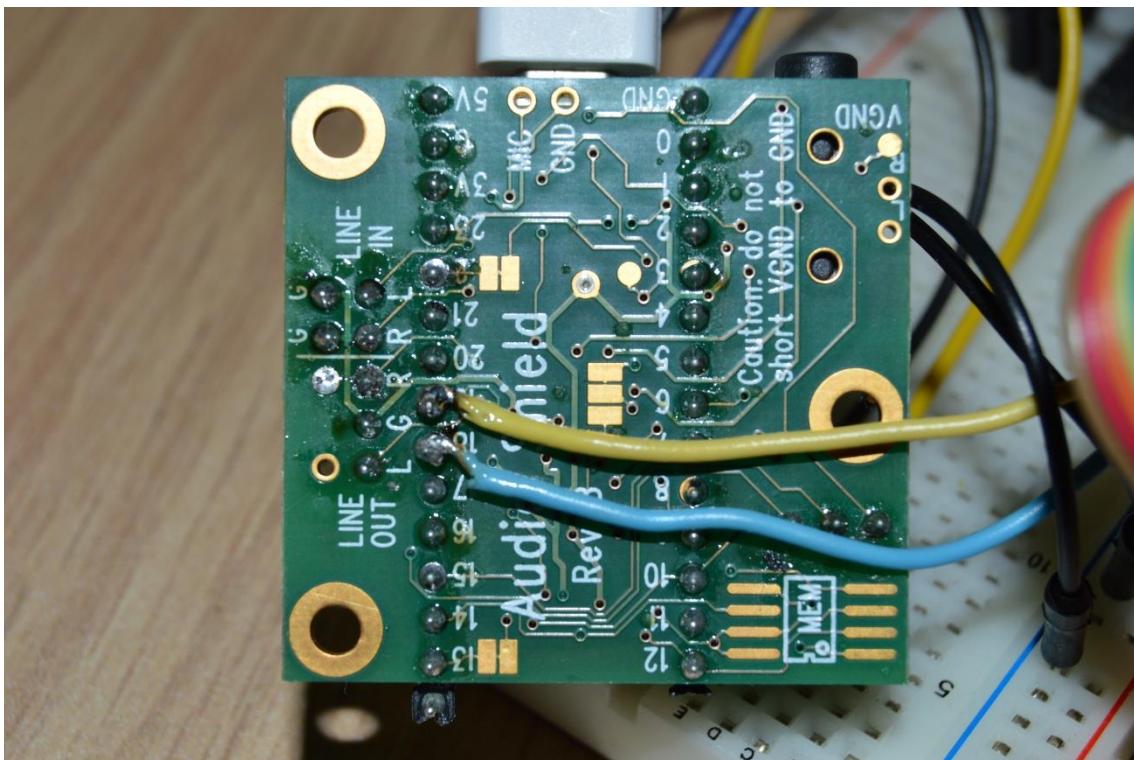
5-C Etapa de adaptación de la señal de la guitarra para su lectura en la placa Teensy.



5-D Placa Teensy 3.2 montada sobre la Teensy Audio Board.



5-E Conversor lógico I2C usado para conectar la pantalla LCD a la placaz Teensy.



5-F Conexionado del LCD a la placa Teensy. Cable amarillo: SCL. Cable Azul: SDA.



5-G Conversor de voltajes de 9V a 5V para el sistema de alimentación del LCD y la placa Teensy.

6. English summary

Musical instruments are tools that need to be perfectly tuned to sound well. In order to learn how to tune a guitar, first it's necessary to know how the musical notes relate to frequencies to be scientifically measurable.

The most common reference frequency used to establish the musical scale is defined in the ISO 16 standard as 440Hz for the note La₄. Starting from this note the *tempered scale* is defined logarithmically, defining an *octave* as an interval between two notes whose frequencies keep a 2:1 relation. More intervals between notes, called *semitones*, are defined dividing an octave in 12 logarithmic parts, creating the 12 notes of the musical scale. In this way, the relation between two consecutive notes goes like this:

$$F_{n+1} = F_n \cdot 2^{(1/12)}$$

Each semitone can also be divided in 100 units called *cents* with the relation seen in the following formula:

$$f_2 = f_1 \cdot 2^{\frac{n}{1200}}$$

This way all the frequency spectrum is divided into units that a human being will perceive as equally distributed due to ear's anatomy.

In the guitar, this theory is applied defining the note for each one of the six strings in standard tuning as EADBGE (from 6th to 1st, starting at the lowest string in terms of frequency) and using *frets* to change the note played by the string as the string is pressed against them. In electric or acoustic-electric instruments this sound is captured by an element called *pickup* which, whether it's piezo-electric or magnetic, converts the sound into an electrical signal and, through a circuit, is taking out to a 6.3mm JACK output.

Each string, due to its construction as a vibrant string, generates a reference frequency, called *fundamental*, and an unspecified number of frequencies called *harmonics*. These harmonics are at frequencies defined by $n \cdot f_0$, being f_0 the fundamental frequency and n each natural number between 2 and the number of harmonics.

References used to tune an instrument are so precise that tuners are required to. These tuners compare the notes played by an instrument to the reference frequencies defined to tell the musician if the instrument is at the correct frequency.

There are two types of guitar tuners, monophonic and polyphonic. Monophonic tuners are capable to detect the frequency of each string playing one string at a time. They are the most common type of tuners since the 20th century. Polyphonic tuners can detect the frequency of each string by playing all at the same time. This makes tuning easier because the musician is able to see which string is out of tune and just tune that one, making it faster, which is something important in live performances.

This project was directed to design an *open source* polyphonic tuner. This tuner had to capture the guitar sound via a 6.3mm JACK input, process the sound to obtain the frequency of each string represented in the captured sound and to show it to the musician using a LCD screen. All of this had to be done with 1 cent precision, to be competitive against some commercial tuners like Polytune by TC Electronic.

This tuner was also thought to be part of the *Do It Yourself* movement, so anyone with enough will would be able to have one. To ensure this DIY capability, it's expected for the tuner to be part of UC3Music's workshops at UC3M as soon as it's functional.

The main reason to design this tuner is that there are no more options apart from commercial tuners like Polytune. Even in the market scope, not so many brands have released other polyphonic tuners, making the Polytune almost the only option at quite high price.

The design process for this tuner went through three phases: acoustic analysis, software prototyping and hardware development.

In the first phase, a test guitar was used to record some audios in order to analyse the signal using MATLAB 2017.

The first audio, containing all the strings, played one by one, revealed how each string created several harmonics apart from its fundamental frequency. At the lower strings (strings 6, 5, 4 and 3) this fact causes no problem because the fundamental frequency is easily recognizable. At the higher strings (strings 1 and 2) the harmonics created by the lowest ones generate higher amplitudes than each string itself. To avoid these harmonics, the system must take the information about the tuning at these strings from other points.

A deeper analysis of the spectrum showed that the same solution could be used for both strings. For each string, at its third harmonic, that string was the one that contributed the most to the amplitude in the FFT.

So, the detection for the four lowest strings (6, 5, 4 and 3) is done using the fundamental frequency of each one and for the highest strings it's done using the frequency of their third harmonic.

Due to the seeming easiness of locating each frequency in the spectrum, the FFT was chosen as the processing algorithm to detect the strings, just needing to define some searching interval for each string. Also, the FFT being natively implemented in some microprocessors helped to choose it as the most appropriate algorithm.

Once the processing algorithm was chosen the second phase, software prototyping, started, using Python 3.5 to develop the software to ensure the open source scope. This way, the software could be used to implement a full polyphonic and open source tuner in PC in future works.

In this phase, the minimum sampling frequency was calculated. Most microprocessors implement low resolution FFTs, with 1024 bins on the best case. These points are equally distributed between $-f_s/2$ and $f_s/2$ (being f_s the sampling frequency). Decreasing the sampling frequency from the 44100Hz used by default in audio allows the points to redistribute in a narrower bandwidth, increasing the resolution by forcing the bins to be closer to each other.

To ensure the system is capable of detecting all the strings and saving some frequency range for future implementations of capo tunings, the lowest sampling frequency allowed is 3200Hz. This way the tuner could be able to work in frequencies up to 1600 Hz.

Once that sampling frequency was tested, the actual software prototyping started.

First, real time audio lecture was implemented using a monophonic algorithm that detects the lower peak in the FFT (the fundamental frequency of the played string). Also, the console display was added, showing ">" when the string is tuned flat, "-" when it's tuned perfectly and "<" when it's higher. Also, time measurements regarding the execution time of the FFT and the detection of the peaks were done, in order to prove right the possibility of using this algorithm in real time processing.

After the real time functioning was assured, the polyphonic detector was successfully implemented. Though it worked fine, the resolution obtained made the system useless for tuning. To ensure enough resolution to make it usable in a real situation, an interpolation algorithm was implemented to generate new samples in order to locate the peak precisely.

This could be done due to the fact that, when the FFT processes a signal whose frequencies don't match exactly at a certain bin, the spectral energy created by

that signal scatters to the adjacent bins. This gives the interpolation some data to locate the peak precisely.

To keep the processing requirements low, second degree polynomic interpolation using the 3 highest samples in the searching interval was the chosen algorithm. This interpolation can be easily and efficiently implemented in a microprocessor using Lagrange polynomials. With this system, after testing with several amounts of samples calculated, the 1cent required resolution was achieved using 800 samples.

The method to test this was using two audios, one with the reference note and another with the reference note decreased by 1cent. Once the difference between both signals could be detected for the lowest strings (6 and 5), it could be considered that the tuner had enough resolution.

However, this interpolation causes certain deviation from the expected values. This deviation is not linear and depends on the distance to the “real” bin calculated by the FFT. This causes the tuner the need of being calibrated.

This calibration was done using an audio for each string containing the frequency to detect. Then, the frequency calculated by the tuner for each string was read and used as the new reference frequency. If the right frequency produces certain value, that value can be considered as the reference the tuner sees.

Considering the software prototyping complete, the third phase began. The first step was to choose the microcontroller. Though at first was thought that the microcontroller had to have an FPU, the Teensy 3.2 board seemed a good option due to factors as the price, the DSP capabilities that enhanced FFT performance, the 96MHz clock frequency and the specific audio board that makes the audio processing easier and more precise with the 16-bit resolution ADC.

To connect any guitar to the Teensy audio board it was necessary to design a circuit that worked as both impedance and voltage adapter.

Teensy audio board needs an input voltage with a 1.65V offset while the guitar offers a 0V offset signal. Connecting two $2\text{M}\Omega$ resistors, one to ground and one to 3.3V, that offset was added to the signal. To make sure the guitar signal has no offset or low frequency noise a 10nF capacitor was introduced before the resistors. This, combined with both resistors created a high-pass filter at 15Hz approximately.

To work as well as possible with any guitar, the impedance at which the guitar is connected should be as high as possible and the impedance seen by the audio board should be as low as possible. Also, if that impedance seen by the board is constant, the functioning with almost every guitar is assured. To do this, after the previously designed filter, an operational amplifier LM324-N, connected as a voltage follower, was used. This circuit introduces no gain in the signal, but has

a high input impedance to connect the guitar to and a low output impedance seen by the audio board.

Once the circuit was built, the board programming started in C++ with Arduino IDE. Using Teensy audio board GUI the code for the internal audio routing was generated. The audio enters using the Line In input of the board and then goes to the FFT 1024 points module.

Due to the low resolution FFT, the 44.1kHz frequency had to be decreased. Though there are no specific methods to change it since the designer made it constant, a few modifications to the libraries allowed the system to work at slower sampling rates.

The lowest sampling frequency allowed by the controller, the SGTL5000 is 8kHz, which is higher than the 3.2kHz calculated earlier as the minimum sampling frequency. Though lower sampling rates would be the best to achieve higher resolutions, the system uses 8kHz because of the limitation caused by the SGTL5000. This frequency gives the tuner a 7.792Hz resolution.

The next step was adapting the code to the microcontroller. This time the interpolation had to be written using directly the Lagrange formula instead of using a pre-existing method. This is because Python has specific libraries for mathematical calculations, while Teensy has not.

Also, the new searching intervals had to be calculated. In the Python script, the intervals matched frequency values, meaning that the 82nd bin corresponded to 82Hz, while at Teensy board, due to its resolution, the 82nd bin corresponds to 638.94Hz. So, the transformation needed to be done to search in the right intervals.

As well as in the Python code, the interpolation was implemented and the precision needed to be assured. This process was realized the same way as in Python, using 2 tones separated by 1 cent to test how many samples were necessary to differentiate between them.

This time, the algorithm caused the read frequency to not stay at a fixed value all the time even though the input signal was a constant sine wave. Because of that, the precision was assured using the mean and the variance of the frequencies calculated by the algorithm. Once the values for both tones were different enough (meaning that less than 5% of the samples had the same values for each tone), the 1 cent resolution was considered to be enough. This was achieved at 200 samples.

That instability in the calculated frequencies causes the calibration in this system to be different from the calibration in the Python code. Python code was more consistent with the calculated values, so comparing the calculated frequency with

a fixed number worked good enough. In this case the best option was to define an interval of values in which the string will be considered in tune.

To calculate this interval for each string while assuring the 1cent resolution, the best process is to use 2 input signals per string. The first signal was that string's theoretical reference frequency down by 1cent and the second was the same theoretical reference up by 1 cent. The mean of both outputs defined the "in tune" interval for each string. This way, if the string is out of tune by 1 cent, the LCD will vary between 2 symbols because some of the values will be out of the interval and some will be inside, so the musician will have a reference about the tuning with 1 cent resolution.

To represent the result of the algorithm, a 20x4 LCD screen was used, connected via i2c due to the low number of available pins in the 3.2 board, caused by the connections used by the audio board.

In this screen, a static display will show the number and note of each string and some references to know where the indicators should be to consider the string tuned. Also, depending on the calculation results, there are six characters for each string that show five different symbols. This way the LCD will tell if the string is more than 20 cents flat, between 20 and 1 cent flat, in tune, between 1 cent and 20 cents high or more than 20 cents high.

Due to the LCD using 5V bidirectional buses and the Teensy 3.2 using 3.3V buses, a logic converter between them was installed.

Also, to be able to run the tuner using a 9V battery (which is the standard in most guitar effects), a switching voltage converter was used, to turn the 9V given by the battery to the 5V used to power the board and the LCD.

In order to test if the tuner was appropriate for tuning, some tuning tests were done. Though the representation of the notes is a bit confusing due to the instability of the calculated frequencies, causing the "in tune" symbol to disappear too quickly from the screen, the results can be considered a success. The tunings done using the prototype were completely accurate according to some other tuning devices used, so the device worked well enough.

However, some future work can be done to this prototype. Adapting some components, like designing a PCB, a smaller display with higher image contrast or pedal-format box could be a way to continue the work.

Other options are to include more possible tunings and tuning detection, to use more guitars as examples to test if the algorithm still works with them (only one guitar could be used for the acoustic analysis), and to improve the display to make it easier to use, making the "in tune" symbol appear more time once the guitar is completely tuned. Also, the software programmed in Python could be used as a start point for a polyphonic tuning app.

Bibliografía

- [1] *Acoustics -- Standard tuning frequency (Standard musical pitch)*, 1^a Edición, Comité técnico ISO/TC 43 Acoustics, ISO 16:1975, [Última consulta: 21/9/2017], disponible en: <https://www.iso.org/standard/3601.html>
- [2] TC Electronic, Polytune 3, 2017, disponible en <http://www.tcelectronic.com/polytune-3/> [Última consulta: 21/9/2017]
- [3] Korg, TM_50tr, 2017, disponible en: http://www.korg.com/es/products/tuners/tm_50tr/specifications.php [Última consulta: 21/9/2017]
- [4] BOSS, Tu-3, 2017, disponible en <https://www.boss.info/us/products/tu-3/specifications/> [Última consulta: 21/9/2017]
- [5] D'ADDARIO, James; CUNNINGHAM, Robert J. Music tuner display screen with graphical user interface. U.S. Patent No D786,287, 9 mayo 2017.
- [6] MURPHY, Jim W., et al. Robot: Tune Yourself! Automatic Tuning for Musical Robotics. En *NIME*. 2014. p. 565-568.
- [7] Gibson Guitar Corporation, 2007, disponible en <http://archive.gibson.com/RobotGuitar/guitar.html> [Último acceso: 23/9/2017]
- [8] Gibson Guitar Corporation, 2014, disponible en http://images.gibson.com/Products/Min-ETune/Documents/Gibson_MinETune_Manual-2012_v1.pdf [Ultimo acceso: 23/9/2017]
- [9] Liao, Yizheng. *Phase and Frequency Estimation: High-Accuracy and Low-Complexity Techniques*, Worcester Polytechnic Institute, mayo 2011, disponible en: https://web.stanford.edu/~yzliao/pub/master_thesis.pdf [Última consulta: 21/9/2017]
- [10] S.Adrián-Martínez, M.Ardid, M.Bou-Cabo, I.Felis, C.Llorens, J.A.Martínez-Mora, M.Saldaña. *Acoustic Signal Detection Through the cross-correlation method in experiments with different signal to noise ratio and reverberation conditions*, Universitat Politècnica de València, Institut d'Investigació a la Gestió Integrada de Zones Costaneres (IGIC), Gandia, España, disponible en: <https://arxiv.org/ftp/arxiv/papers/1502/1502.05038.pdf> [Última consulta: 21/9/2017]
- [11] Cooley, James W., Tukey, John W. *An algorithm for the Machine Calculation of Complex Fourier Series*. Nueva York, 1965. Disponible en: <http://www.ams.org/journals/mcom/1965-19-090/S0025-5718-1965-0178586-1/S0025-5718-1965-0178586-1.pdf> [Última consulta: 21/9/2017]

[12] Directiva 2002/95/CE, *Official Journal L 037*, 13 febrero 2003, P. 0019 – 0023, disponible en <http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:32002L0095:EN:HTML> [Último acceso: 23/9/2017]

[13] Real Decreto 110/2015, sobre *residuos de aparatos eléctricos y electrónicos*, BOE núm. 45, de 21 de febrero de 2015, páginas 14211 a 14312. Disponible en: https://www.boe.es/diario_boe/txt.php?id=BOE-A-2015-1762 [Último acceso: 23/9/2017]