COMILLAS

UNIVERSIDAD PONTIFICIA

ICAI

GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE GRADO

DISEÑO DE SOFTWARE Y CONTROL DE UN INSTRUMENTO POR CONTROL REMOTO

Autor: Joaquín Juliani Ramos
Director: Frédéric Boulanger

MADRID
29/08/2019

AUTORIZACIÓN PARA LA DIGITALIZACIÓN, DEPÓSITO Y DIVULGACIÓN EN RED DE PROYECTOS FIN DE GRADO, FIN DE MÁSTER, TESINAS O MEMORIAS DE BACHILLERATO

1º. Declaración de la autoría y acreditación de la misma.

El autor D. Joaquín Juliani Ramos

DECLARA ser el titular de los derechos de propiedad intelectual de la obra: Diseño de software y control de un instrumento por control remoto que ésta es una obra original, y que ostenta la condición de autor en el sentido que otorga la Ley de Propiedad Intelectual.

2º. Objeto y fines de la cesión.

Con el fin de dar la máxima difusión a la obra citada a través del Repositorio institucional de la Universidad, el autor **CEDE** a la Universidad Pontificia Comillas, de forma gratuita y no exclusiva, por el máximo plazo legal y con ámbito universal, los derechos de digitalización, de archivo, de reproducción, de distribución y de comunicación pública, incluido el derecho de puesta a disposición electrónica, tal y como se describen en la Ley de Propiedad Intelectual. El derecho de transformación se cede a los únicos efectos de lo dispuesto en la letra a) del apartado siguiente.

3º. Condiciones de la cesión y acceso

Sin perjuicio de la titularidad de la obra, que sigue correspondiendo a su autor, la cesión de derechos contemplada en esta licencia habilita para:

a) Transformarla con el fin de adaptarla a cualquier tecnología que permita incorporarla a internet y hacerla accesible; incorporar metadatos para realizar el registro de la obra e incorporar "marcas de agua" o cualquier otro sistema de seguridad o de protección.
b) Reproducirla en un soporte digital para su incorporación a una base de datos electrónica, incluyendo el derecho de reproducir y almacenar la obra en servidores, a los efectos de garantizar su seguridad, conservación y preservar el formato.
c) Comunicarla, por defecto, a través de un archivo institucional abierto, accesible de modo libre y gratuito a través de internet.
d) Cualquier otra forma de acceso (restringido, embargado, cerrado) deberá solicitarse expresamente y obedecer a causas justificadas.
e) Asignar por defecto a estos trabajos una licencia Creative Commons.
f) Asignar por defecto a estos trabajos un HANDLE (URL *persistente)*.

4º. Derechos del autor.

El autor, en tanto que titular de una obra tiene derecho a:

a) Que la Universidad identifique claramente su nombre como autor de la misma
b) Comunicar y dar publicidad a la obra en la versión que ceda y en otras posteriores a través de cualquier medio.
c) Solicitar la retirada de la obra del repositorio por causa justificada.
d) Recibir notificación fehaciente de cualquier reclamación que puedan formular terceras personas en relación con la obra y, en particular, de reclamaciones relativas a los derechos de propiedad intelectual sobre ella.

5º. Deberes del autor.

El autor se compromete a:

a) Garantizar que el compromiso que adquiere mediante el presente escrito no infringe ningún derecho de terceros, ya sean de propiedad industrial, intelectual o cualquier otro.
b) Garantizar que el contenido de las obras no atenta contra los derechos al honor, a la intimidad y a la imagen de terceros.
c) Asumir toda reclamación o responsabilidad, incluyendo las indemnizaciones por daños, que pudieran ejercitarse contra la Universidad por terceros que vieran infringidos sus derechos e

intereses a causa de la cesión.

d) Asumir la responsabilidad en el caso de que las instituciones fueran condenadas por infracción de derechos derivada de las obras objeto de la cesión.

6º. Fines y funcionamiento del Repositorio Institucional.

La obra se pondrá a disposición de los usuarios para que hagan de ella un uso justo y respetuoso con los derechos del autor, según lo permitido por la legislación aplicable, y con fines de estudio, investigación, o cualquier otro fin lícito. Con dicha finalidad, la Universidad asume los siguientes deberes y se reserva las siguientes facultades:

- La Universidad informará a los usuarios del archivo sobre los usos permitidos, y no garantiza ni asume responsabilidad alguna por otras formas en que los usuarios hagan un uso posterior de las obras no conforme con la legislación vigente. El uso posterior, más allá de la copia privada, requerirá que se cite la fuente y se reconozca la autoría, que no se obtenga beneficio comercial, y que no se realicen obras derivadas.
- La Universidad no revisará el contenido de las obras, que en todo caso permanecerá bajo la responsabilidad exclusive del autor y no estará obligada a ejercitar acciones legales en nombre del autor en el supuesto de infracciones a derechos de propiedad intelectual derivados del depósito y archivo de las obras. El autor renuncia a cualquier reclamación frente a la Universidad por las formas no ajustadas a la legislación vigente en que los usuarios hagan uso de las obras.
- La Universidad adoptará las medidas necesarias para la preservación de la obra en un futuro.
- La Universidad se reserva la facultad de retirar la obra, previa notificación al autor, en supuestos suficientemente justificados, o en caso de reclamaciones de terceros.

Madrid, a 29 de Agosto de 2019

ACEPTA

Fdo.....

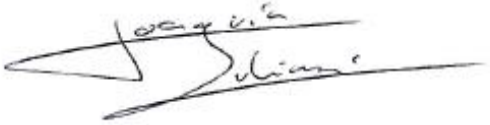
Motivos para solicitar el acceso restringido, cerrado o embargado del trabajo en el Repositorio Institucional:

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título:

Diseño de software y control de un instrumento musical por control remoto

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico 2018/2019 es de mi autoría, original e inédito y no ha sido presentado con anterioridad a otros efectos. El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada de otros documentos está debidamente referenciada.

Fdo.: Joaquín Juliani Ramos Fecha: 26/06/2019



Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Frédéric Boulanger Fecha: 24 /06 /2019

COMILLAS

UNIVERSIDAD PONTIFICIA

ICAI

GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE GRADO

DISEÑO DE SOFTWARE Y CONTROL DE UN INSTRUMENTO POR CONTROL REMOTO

Autor: Joaquín Juliani Ramos
Director: Frédéric Boulanger

MADRID
29/08/2019

DISEÑO DE SOFTWARE Y CONTROL DE UN INSTRUMENTO POR CONTROL REMOTO

Autor: Joaquín Juliani Ramos

Director: Frédéric Boulanger

Resumen del proyecto

Introducción:

En este proyecto, el objetivo era la automatización completa de un instrumento de música, haciendo posible el que fuese tocado vía un programa en el ordenador. En años anteriores, este proyecto ya se había propuesto, tratando de implementarlo en una flauta dulce y en un ukelele. Se eligió la guitarra ya que guardaba similitud con lo desarrollado para el ukelele y además había una disponible por lo que no era necesario comprar una guitarra.

Tras realizar un estudio inicial, estudiar costes de distintas opciones y repasar lo que se hizo en proyectos anteriores, se optó por desarrollar un sistema de cremalleras y engranajes; en vez de usar un sistema de pistones o rodamientos, como se había pensado inicialmente; que se desplazase sobre los trastes de la guitarra, y desarrollar de manera separada un sistema de rasgueo para las cuerdas utilizando servomotores.

Metodología:

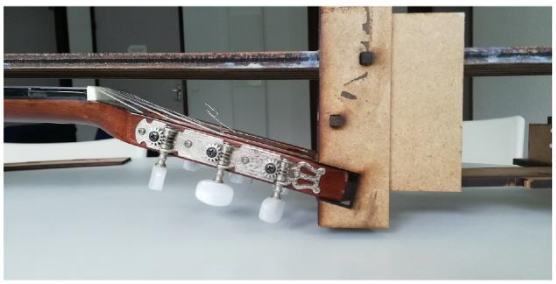
El proyecto constaba de tres partes bien distinguidas: la estructura que serviría para “tocar” la guitarra en sí, los motores que moverían las cremalleras y la aplicación que se comunicaría con los motores.

En lo que respecta a la estructura, inicialmente el diseño consistía en una larga plancha de madera que pasaría por encima de toda la guitarra, y serviría de guía para las cremalleras además de estructura de apoyo para el aparato de rasgueo. Rápidamente se apreció que esto no sería posible, ya que la plancha se combaba bajo su peso muy fácilmente, y esto conllevaba un rozamiento variable en función del traste, lo que complicaba mucho los cálculos para la función del movimiento.

Diseño original de la estructura

Por lo tanto, finalmente se optó por separar en dos partes la estructura: de un lado los trastes, y del otro el rasgueo. Para los trastes se usó el mismo concepto que el de la plancha larga, excepto que, en vez de ranuras, con la ayuda de una impresora láser se creó un surco de 5mm de profundidad para guiar la cremallera, a la cual se unía en la parte posterior, un espacio donde colocar los motores, con una guía para las ruedas dentadas y unas pequeñas planchas de madera para evitar que los motores girasen sobre si mismos.

De esta manera la plancha se abrazaba a la guitarra para evitar que se moviese o cayese sobre las cuerdas, a la vez que servía de guía hasta el último traste.



En esta imagen se observa como se sujetaba la guitarra por el extremo del mástil, dejando firme la plancha y sirviendo de apoyo para poner la plataforma para los motores.

Finalmente, esta es la plataforma donde estaban colocados los motores, siendo la plancha de abajo una guía para las ruedas dentadas, de forma que no se saliesen de los dientes de la cremallera.

La estructura de los servomotores para el rasgueo de las cuerdas fue mucho más sencilla de realizar, y no requirió tantas iteraciones y modificaciones como la otra. Consistió simplemente en sujetar una plancha sobre las cuerdas, a la cual se adherirían con *blue-tac* los servomotores. Calculando la altura, se desarrolló con un programa de diseño 3D una pieza que haría girar el servomotor para que actuase como una púa.

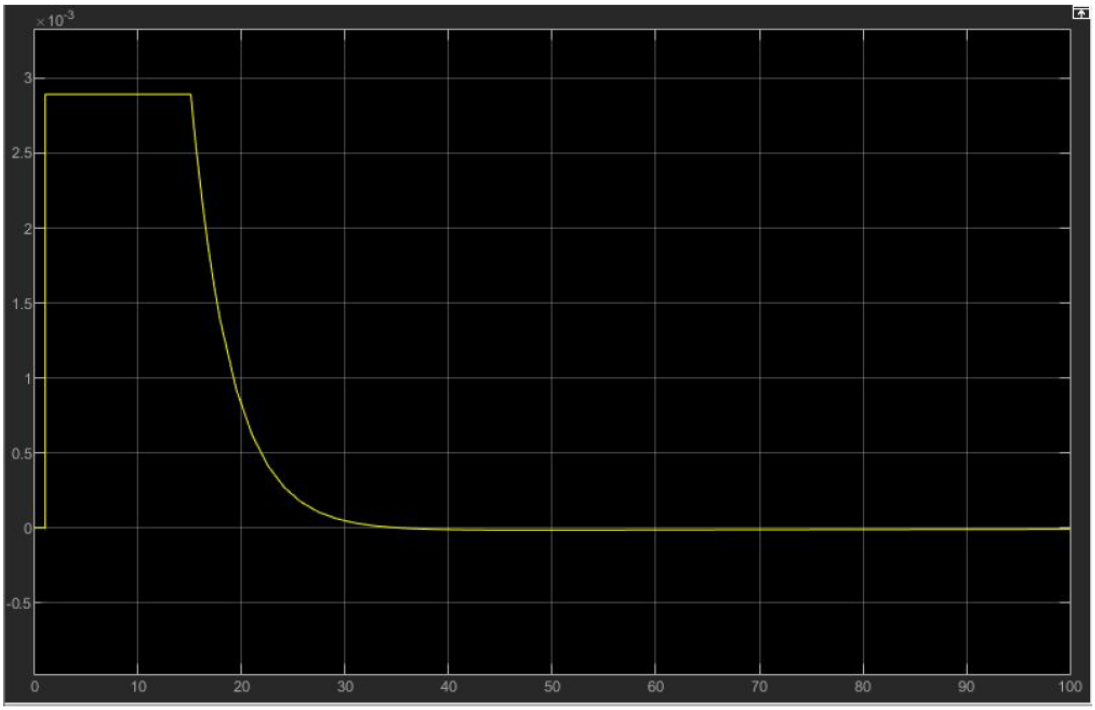
Para la parte de la cremallera y la rueda dentada, se utilizó un programa llamado *OpenScad*, empleando un código que estaba ya creado y disponible, que permitía regular la longitud de la cremallera, el radio de la rueda dentada y el número de dientes de esta.

El motor de los trastes debía ser lo suficientemente potente como para poder superar el rozamiento de los hierros de los trastes y de la madera y garantizar un tiempo de respuesta razonable a la hora de cambiar de nota. Esto último es un problema con el que se encontraron en el proyecto del ukelele al utilizar un tornillo sinfín, el cual no tiene mucha velocidad. Se eligió finalmente un motor de corriente continua con reductor y codificador rotatorio, para poder calcular la posición a través de este último.

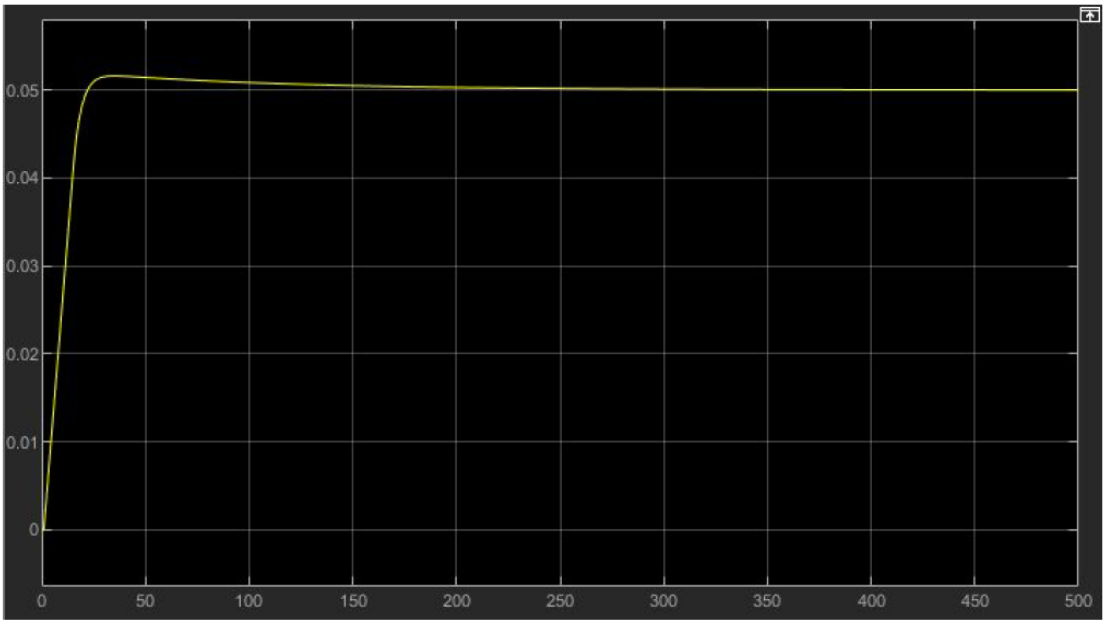
Sin embargo, el motor que se pidió no es el mismo que llegó, principalmente porque era de alimentación de 12V en vez de 6V. Por lo tanto, se volvieron a estudiar todos los parámetros para poder pilotarlo y poder llevar a cabo la regulación de este. Una vez que se probó físicamente, hubo problemas porque las soldaduras estaban muy sueltas, y debido a que resultaba difícil sujetarlo sin apretar en estas, debido a la torsión se soltaban muy fácilmente. Finalmente, se pidió otro motor (con otros distribuidores para garantizar que llegaba le deseado) y se repitieron los ensayos. Con el anterior se había observado que le faltaba fuerza, tenía una relación de reducción de 1:75, por lo que en el nuevo se buscó mejorarla, y se cogió uno con 1:34.

Una vez estudiado el comportamiento y calculados todos los parámetros necesarios para poder controlarlo, se diseñó un corrector PID para regularlo y garantizar una buena respuesta en posición y tiempo. Cuando se realizaron los ensayos en *Simulink*, se tuvieron en cuenta dos tipos de señal consigna: una simple constante, como haría un interruptor; y una que frenase al llegar al destino para que el motor no sufra y que fuese más preciso. Después de varias simulaciones, se obtuvieron los valores deseados para el PID, obteniendo las siguientes gráficas como respuestas a una consigna de 0,05m. Como se puede observar, no hay error estático debido a la acción integradora del PID.

Curva de velocidad:



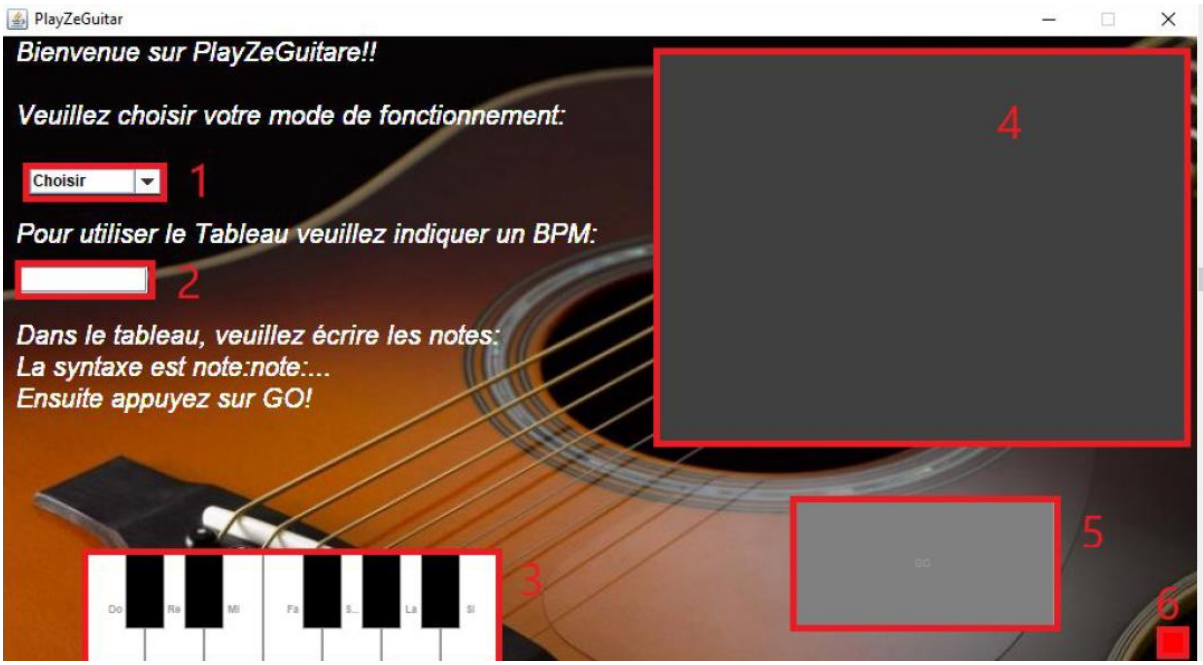
Curva de posición:



Para controlar el sistema, se utilizó un microcontrolador *PyBoard* junto con un *Adafruit Motorshield*. La *PyBoard* unida al *Motorshield* permite controlar la velocidad de los motores que mueven las cremalleras y los servomotores del sistema de rasgueo programando en *Python*. Se definieron todas las variables y funciones necesarias y se pasó a los ensayos. Sorprendentemente, de las dos señales consigna que se habían probado, funcionaba mejor la primera (estilo interruptor) que el código con el PID incorporado. Esto se debe al frotamiento que imponen los trastes; una vez que empieza a frenar no

tiene la potecia necesaria para superar el desnivel y se queda en el traste equivocado. Una vez que se aprecia el error y se elige entonces la función simple, se crea una función para tocar las notas; donde se guardan como valores las distancias a cada nota, e introduciendo una nota el motor desplaza la punta de la cremallera hasta el traste correspondiente tras lo que envía una señal al servomotor para tocarla.

Una vez que se puede controlar el aparato, faltaba crear una interfaz gráfica para facilitar su uso. Esta se llevo acabo utilizando *JavaScript*.



Con esta interfaz primero se elegiría el método de funcionamiento(1), ya sea piano (para utilizar el teclado (3)) o el tablero donde se escribirían las notas. Si se elige el piano, pulsando las notas del teclado se moverá la cremallera hasta la nota seleccionada y al llegar se accionará el servomotor para hacerla sonar una vez. De seleccionarse el tablero, se definirían los BPMs deseados (2) y en el tablero (4) se escribirá el orden de notas en formato: “nota:nota:nota”. Una vez se hayan rellenado los datos, pulsando el botón (5) se tocará la secuencia de notas. El pequeño cuadrado de la esquina inferior derecha (6) es un indicador de si la *PyBoard* está conectada al ordenador (verde) o desconectada (roja).

Conclusión:

Este proyecto fue evolucionando a lo largo de su desarrollo, desde los conceptos iniciales hasta el resultado final. Pese a que no se llegaron a implementar todos los motores al aparato, se diseñó una estructura apta para estos, por lo que unicamente requeriría comprarlos e instalarlos de la misma manera que el primero. Tras haber calculado los parámetros necesarios para controlar con un PID el motor, resultó mejor la utilización de un código sencillo, que simplemente ponga a su máxima potencia el motor y que lo pare al llegar al traste correspondiente. En este proyecto aún quedan opciones de mejora, por lo que resultaría interesante el retomarlo en un futuro, para implementar la lectura de ficheros MIDI por ejemplo, o modificar el mástil quitando los trastes para que

no haya tanto rozamiento, pudiendo utilizar la otra función y reduciendo los ruidos asociados.

Finalmente decir que en este proyecto se alcanzó a crear un producto incorporando el control remoto junto con el desarrollo de una interfaz gráfica, cumpliendo los objetivos que se habían planteado al comienzo de este.

SOFTWARE CONTROL OF A MUSICAL INSTRUMENT

Author: Joaquín Juliani Ramos

Director: Frédéric Boulanger

Summary of the project

Introduction:

In this project, the objective was the complete automatization of a musical instrument, making it possible for it to be played via a computer program. In previous years, this project had already been proposed, trying to control a flute and a ukulele. The guitar was chosen since it was similar to what had already been developed for the ukulele and there was one already available, therefore making unnecessary the acquisition of a new one.

After conducting an initial study, looking up costs of different options and reviewing what was done in previous projects, it was decided to develop a rack and gear system; instead of using a piston or bearing system, as initially thought; to move on the guitar's frets, and develop a strumming system for strings separately using servo motors.

Methodology:

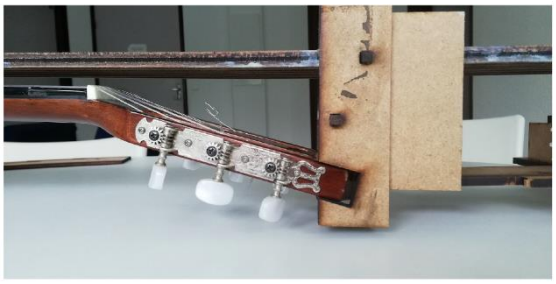
The project consisted of three distinct parts: the structure that would serve to "play" the guitar itself, the motors that would move the rack and gear system and the application that would communicate with the engines.

Regarding the structure, initially the design consisted of a long wooden plank that would pass over the entire guitar and would serve as a guide for the racks as well as supporting structure for the strumming apparatus. Quickly it was appreciated that this would not be possible, since the plank combated under its weight very easily, and this entailed a variable friction depending on the fret, which greatly complicated the calculations for the movement function.

Original structure design

Therefore, it was finally decided to separate the structure into two parts: on one side the frets, and on the other the strumming device. For the frets, the same concept was used as that of the long plank, except that, instead of holes, a 5mm deep groove was created to guide the rack, which connected to a space to place the engines, with a guide for the gears and with small wooden plates to prevent the motors from turning on themselves.

In this way the plank attached to the guitar to prevent it from moving or falling on top of the strings, while serving as a guide for the whole guitar neck.



This image shows how the guitar was held by the end of the neck, leaving the wooden plank firm while supporting the platform for the engines.

Lastly, this is the platform where the motors were placed, the bottom plank being a guide for the gears, so that they did not disengage from the rack.

The structure of the servo motors for strumming the strings was much easier to create and did not require as many iterations and modifications as the other one. It consisted simply on a small wooden board held over the strings, to which the servomotors would adhere with the help of blue tac. Calculating the height, a piece was developed with a 3D design program that the servo motor would rotate to act as a guitar pick.

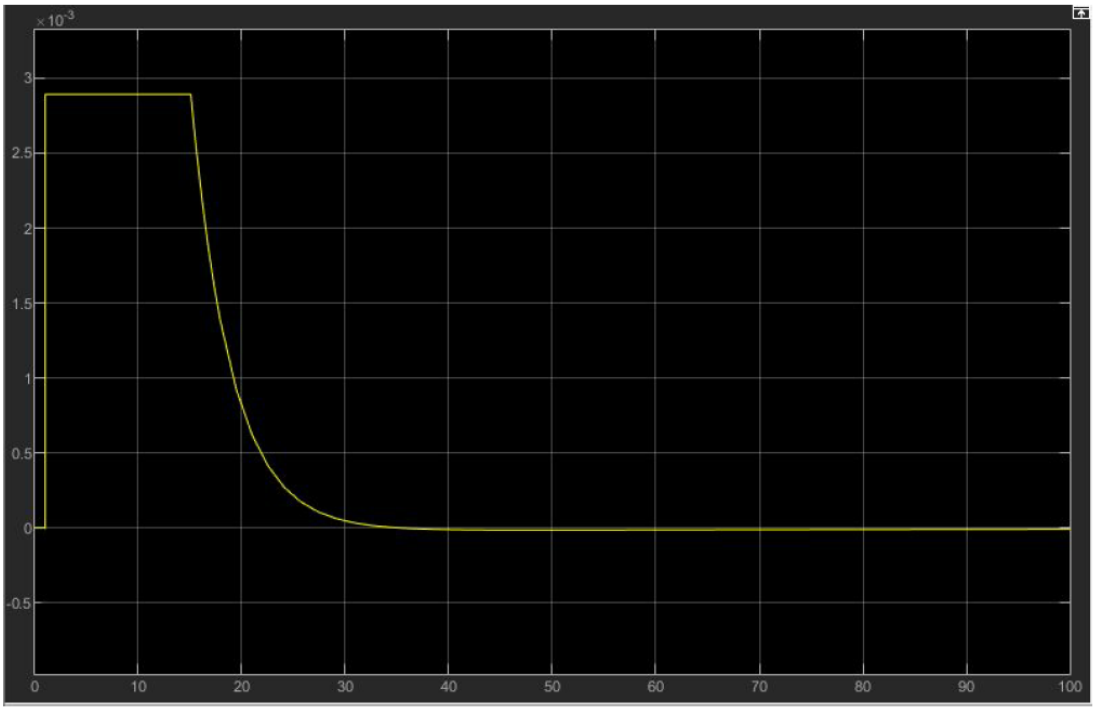
The rack and the gear were created with a program called *OpenScad*, using a code that was already available, which allowed to regulate the length of the rack, the radius of the gear and how many teeth it has.

The motor of the neck had to be powerful enough to be able to overcome the friction of the iron frets and that of the wood and guarantee a reasonable time response when changing notes. The latter is a problem they encountered in the ukulele project when using an endless screw, which does not have much speed. Finally, a direct current motor with a reduction and encoder was chosen to calculate the position with the help of the latter.

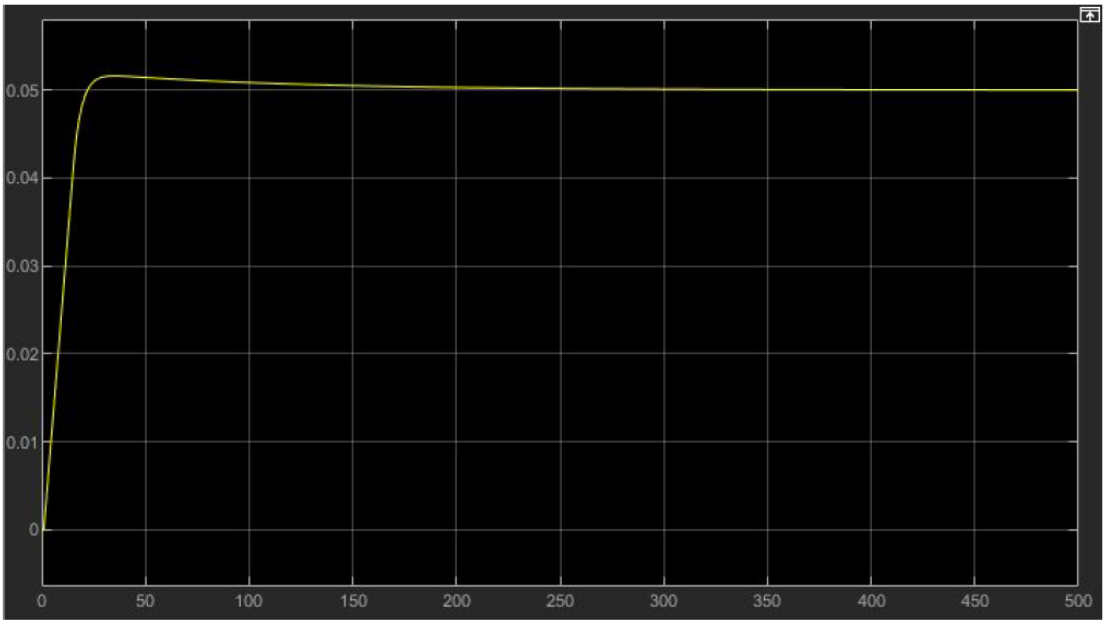
However, the motor that was ordered is not the same as the one that arrived, mainly because it was powered by 12V instead of 6V. Therefore, all the parameters were re-studied to be able to control and regulate it. When the physical tests were made, there were problems due to the welded parts being very loose. It was difficult to hold it without pulling on these and due to the torsion, they were released very easily. Finally, another motor was requested (with other distributors to ensure that the desired one arrived) and the tests were repeated. With the previous one, it had been observed that he lacked strength and it had a reduction ratio of 1:75, so in the new one in order to improve it, one with a 1:34 ratio was ordered.

Once the motor was studied and all the necessary parameters were calculated in order to control it, a PID corrector was designed to regulate it and guarantee a good response in position and time. When carrying out the *Simulink* tests, two types of order signal were taken into account: a simple constant, as a light switch would do; and one that slowed down when arriving at the destination in order to increase the precision and have a less rough control of the motor. After several simulations, the desired values for the PID were calculated, obtaining the following graphs as responses to a 0.05m signal. There is no static error due to the integrating action of the PID.

Motor speed curve:



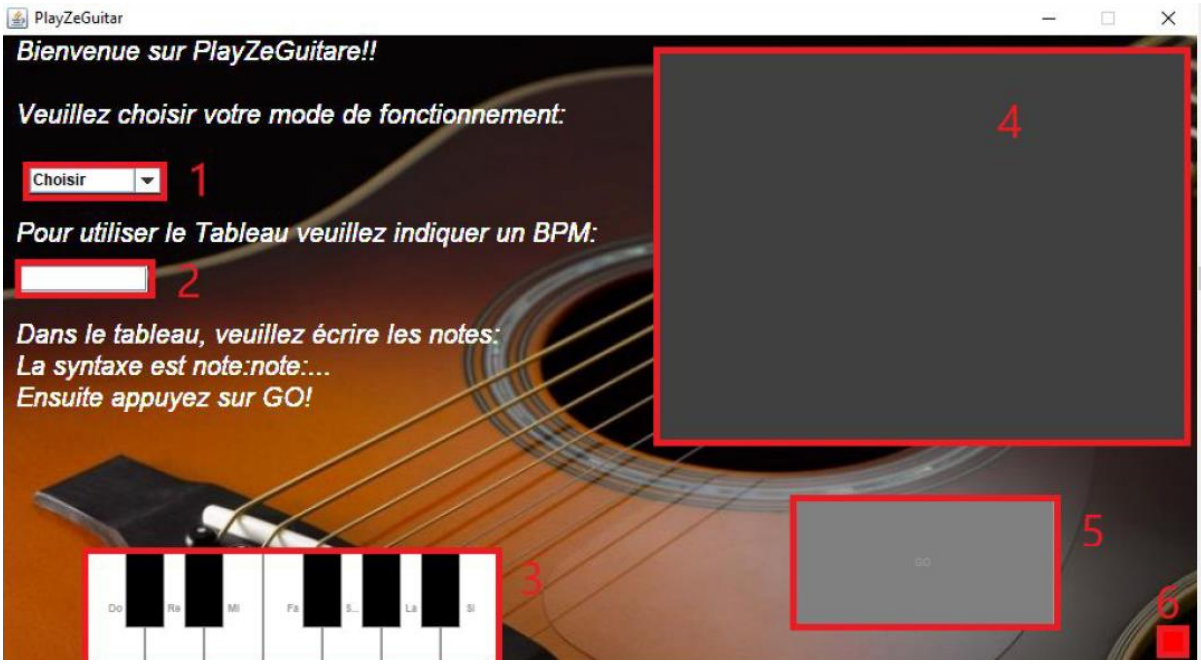
Position curve:



To control the system, a PyBoard microcontroller was used together with an Adafruit Motorshield. The PyBoard attached to the Motorshield allows to control the speed of the motors that move the rack and the servo motors of the strumming system programming in Python. All the necessary variables and functions were defined and the tests were runned. Surprisingly, of the two control signals that were tested, the first one (light switch style) worked better than the code with the built-in PID. This is due to the friction imposed by the frets; once it begins braking, it does not have the power to

overcome the unevenness and ends up in the wrong fret. Once the error is appreciated and the simple function is chosen, a function is created to play the notes: the distances to each note are saved as values, and by entering a note the motor moves the tip of the rack to the corresponding fret, then sends a signal to the servo motor to play it.

Once the device controllable, it was necessary to create a graphical interface to facilitate its use. This was done using *JavaScript*.



With this interface you would first choose the method of operation (1), either piano (to use the keyboard (3)) or the board, where the notes would be written. If the piano is chosen, pressing the notes of the keyboard will move the rack to the selected note and upon arrival, the servo motor will be activated to play the note. If the board is selected, the desired BPMs (2) would be determined, and the order of the notes will be written on the board (4) in the format: “note: note: note”. Once the data has been filled in, pressing the button (5) will play the sequence of notes. The small square in the lower right corner (6) is an indicator of whether the PyBoard is connected to the computer (green) or disconnected (red).

Conclusion:

This project has evolvied throughout its development, from the initial concepts to the final result. Although not all motors were implemented in the device, a structure was designed for them, so that the only requirement would be buying and installing them in the same way as the first one. Once the necessary parameters to control the motor with a PID were calculated, once tested, it was found out to be better to use a simple code that simply puts the motor to its maximum power and stops it when it reaches the corresponding fret. In this project there are still options for improvement, it would be interesting to implement the reading of MIDI files for example, or modify the neck by

removing the frets so that there is not as much friction, in order to use the other function and reduce the associated noises.

Finally, with the development of this project it was possible to create a product incorporating the remote control together with the development of a graphic interface, fulfilling the objectives that had been defined at the beginning of it.

Índice

Introducción

El proyecto realizado en École CentraleSupélec, fue propuesto por el profesor encargado de su supervisión en Francia. Durante varios años ha propuesto el mismo proyecto, estudiando cada vez distintas metodologías y técnicas con las que enfrentarse a la problemática.

En años anteriores, se desarrolló el mismo proyecto con un ukelele y con una flauta dulce; este año se decidió probarlo en una guitarra. La École exigía la incorporación de elementos tanto de innovación como de cierto desarrollo de software en el proyecto para darlo como válido. Por lo tanto, se planteó la problemática teniendo estas dos ideas en mente.

La parte de innovación consistió en la búsqueda de métodos y desarrollo de soporte físico para poder llevar a cabo el proyecto. Se utilizaron programas como *OpenSCAD*, *Fusion 360*, *CorelDRAW* e *Inkscape* para el desarrollo de la estructura; además de *Simulink* para validar las pruebas de los motores y crear el código de control de estos. El desarrollo software consistió en diseñar una interfaz gráfica con la que controlar el aparato que se había concebido, además de pilotarlo.

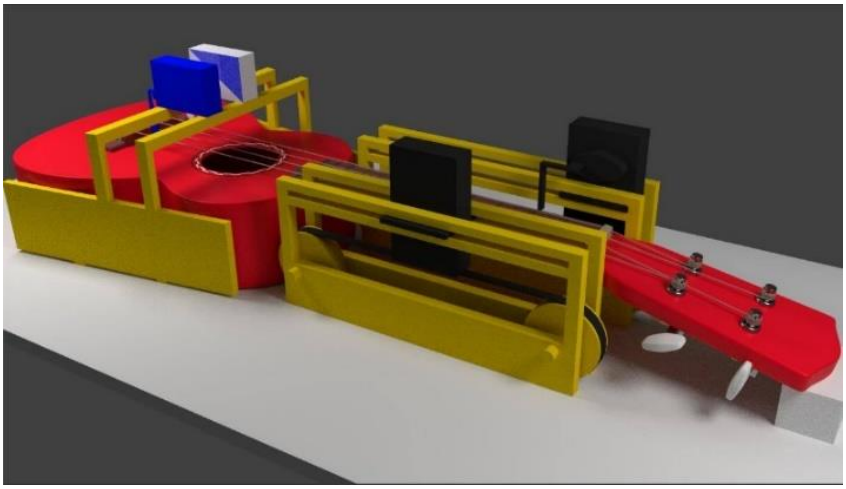
Teniendo claros los objetivos, se comenzó el proyecto mediante un estudio de lo que se había realizado los años anteriores.

1. CAPÍTULO 1: Estudio y planteamiento

1.1 Proyecto del ukelele

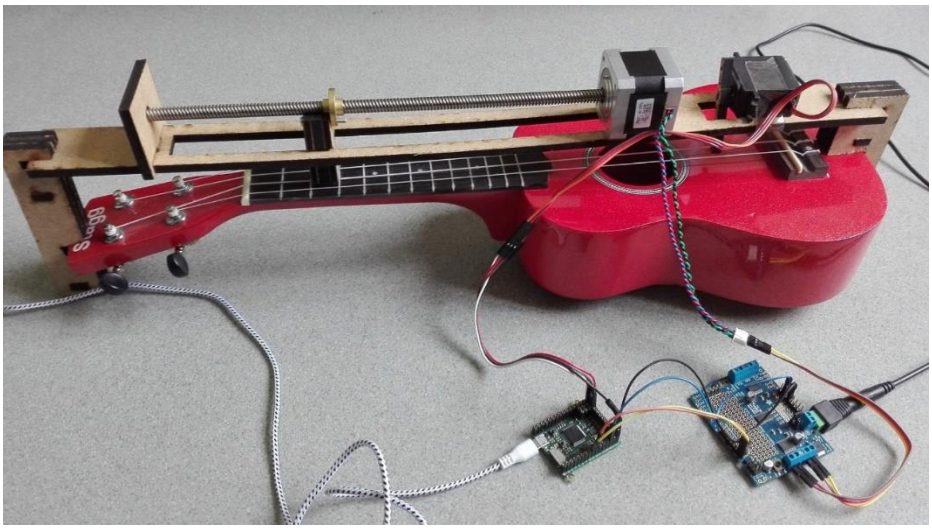
Para comenzar el proyecto, lo primero era analizar lo que habían hecho los alumnos de los años anteriores. Se había realizado este proyecto en dos ocasiones hasta la fecha, una adaptando un ukelele[1], y otra una flauta dulce[2]. Viendo el proyecto del ukelele, se vieron muchas opciones de mejora en el concepto, y ya había ciertos métodos que se había comprobado que no eran óptimos para este tipo de instrumentos. Debido a la similitud que guardaban la guitarra y el ukelele entre sí, que es un instrumento que ya conocía y, además, que había uno disponible, se decidió realizar el proyecto sobre una guitarra española.

Una vez que se eligió la guitarra como instrumento, el proyecto de la flauta perdió interés, y se estudió exclusivamente el del ukelele. Este instrumento, para el que no lo conozca, se asemeja a una guitarra pequeña, normalmente con cuatro cuerdas. El hecho de ser pequeño dificulta mucho la incorporación de elementos mecánicos que permitan operar en cuerdas distintas, por lo que se vieron obligados a trabajar con un único motor que apretaba en todas las cuerdas de un mismo traste simultáneamente. Además, este dispositivo operaba con un motor paso a paso, moviendo la plancha que apretaba sobre los trastes con la ayuda de un tornillo sinfín. Como se puede apreciar en la imagen del concepto inicial, y en la imagen del resultado final del proyecto del ukelele, este fue variando a lo largo de su desarrollo, inicialmente concebido para funcionar mediante correas, siendo las cuerdas independientes la una respecto de las otras.



El motor que se seleccionó resultó ser demasiado lento e ineficiente, tardando demasiado tiempo en moverse entre trastes, haciendo mucho ruido, por lo que impedía escuchar el sonido que hacía el ukelele una vez se tocaban las cuerdas, y haciendo imposible el tocar cualquier melodía ligeramente compleja (sin tener en cuenta, además, que no se podían generar acordes debido a que las cuatro cuerdas tocaban el mismo traste cada vez).

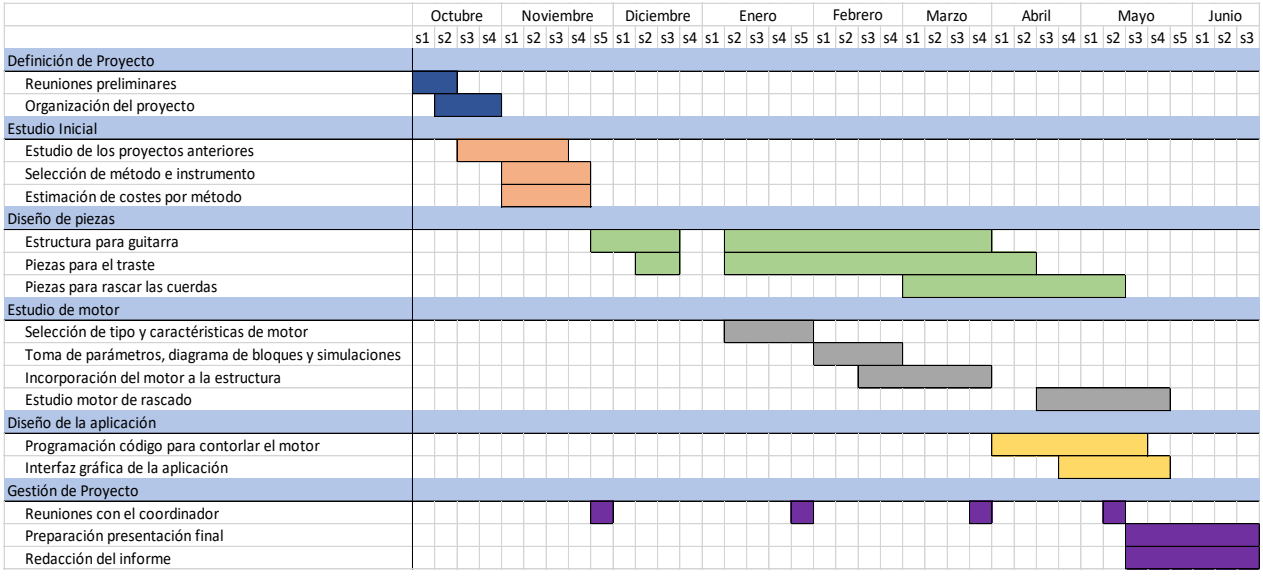
[1] Proyecto de Denis Tian, *Control de Ukelele por control remoto*
[2] Proyecto de Alexandre Broussadier, *Control de flauta dulce por control remoto*



Debido a estos problemas, se seleccionó la guitarra española en vez de una eléctrica o el propio ukelele debido a la gran diferencia de tamaño que hay entre cuerdas; pudiendo realizar así un sistema independiente entre cuerdas, para poder tocar acordes distintos.

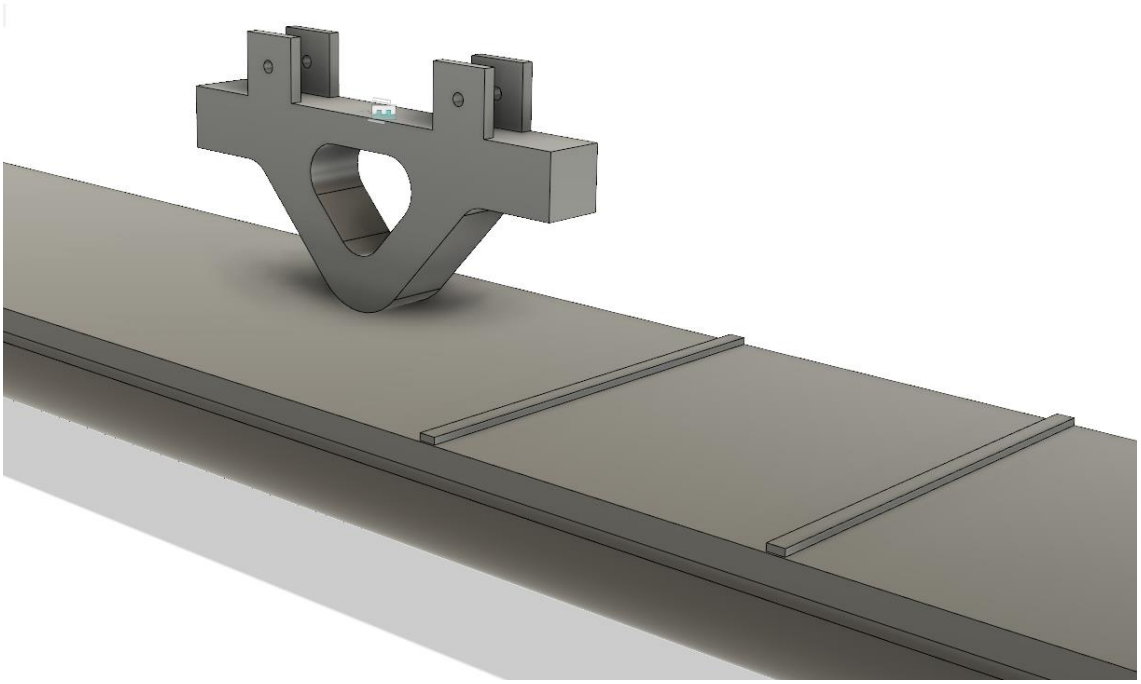
1.2 Definición del plan de trabajo

Una vez que se había seleccionado el proyecto, se definió un plan de trabajo con la ayuda de un diagrama de Gantt, para ponerse objetivos claros y tener cierto sentido de progreso y organización. La parte más complicada era la estructura para los trastes, ya que de esa parte dependía todo lo demás. La estructura de rasgueo no se planteaba especialmente complicada, inicialmente se pensó en poner unos servomotores que rotasen un cierto ángulo cada vez en un sentido horario distinto a la vez anterior, y es lo que finalmente se hizo y resultó práctico y funcional. Sin embargo, la parte del mástil había que diseñarla por completo y escalable a seis cuerdas.



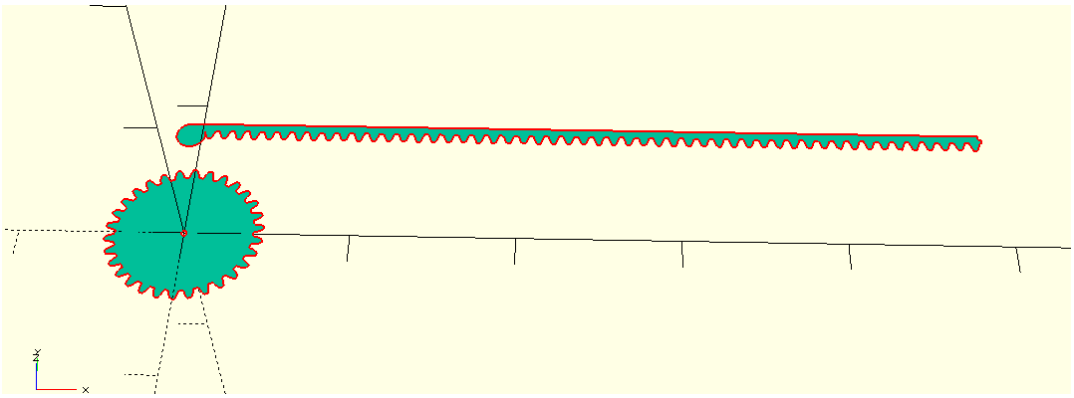
1.3 Elección de metodología

Inicialmente se pensó en varias opciones. Primero se planteó la idea de crear un sistema de motores con correas, similares a las de un tanque de combate, con una pieza adherida a la parte inferior de esta que se moviese haciendo presión sobre la cuerda. En la teoría funcionaba, pero una vez que se empezaron a buscar piezas del tamaño necesario para realizarlo en las seis cuerdas en paralelo se vio que no era posible, ya fuese porque no había piezas del tamaño necesario, o las que había tenían un precio demasiado elevado.



Posteriormente, se estudió la idea de crear un sistema de pistones, se descartó en seguida debido a la complejidad y costes, enfrentandose a los mismos problemas que la propuesta anterior; sumándole además a estos la complejidad de tener que pilotar un pistón por traste y cuerda o tener que mover todo el aparato con los pistones a lo largo del mátil.

Finalmente, se decidió llevar a cabo el proyecto utilizando un sistema de cremalleras y engranajes, de manera que hubiese que pilotar únicamente un motor por cuerda, desplazando la parte que ejerce la presión sobre la cuerda a lo largo de esta.



2 CAPÍTULO 2: Concepción de la estructura

2.1 Concepción inicial

Una vez definida la metodología por la que iba a operar el aparato, había que crear la estructura de soporte de la guitarra y los motores. Se decidió que toda la estructura se construiría a base de madera, como en el proyecto del ukelele, con la ayuda de la impresora láser del laboratorio Fablab Digiscope, una *Epilog Laser Fusion M2+*. Después de realizar un cursillo para poder operarla, se empezaron a diseñar las piezas.

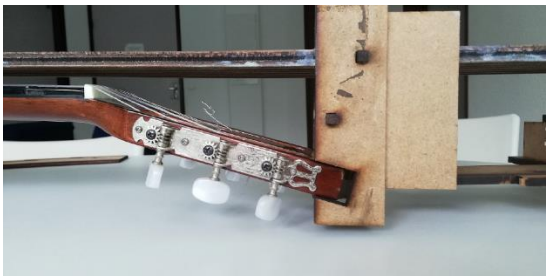
El concepto inicial partía de una plancha de madera que cubriría toda la parte superior de la guitarra. Como se observa en la siguiente imagen, la cual es uno de los conceptos iniciales para la estructura, los motores irían en un principio en la parte superior, teniendo la cremallera los dientes en el lado opuesto a la punta que presiona las cuerdas. La ranura de la izquierda serviría para que la punta de las cremalleras pudiese salir cuando se tocasen notas más graves, y sobre el reborde de la derecha, además de servir de abrazadera para darle estabilidad a la guitarra, serviría para colocar los servomotores encargados del rasgueo.

Este concepto no era práctico, debido a la fragilidad de la plancha superior. En la siguiente iteración se decidió dejar la plancha superior completa, con unas ranuras que no la travesasen del todo y sirviesen como guía a las cremalleras. Los motores esta vez estarían colocados más allá del mango de la guitarra, teniendo por consiguiente los dientes y la punta que ejerce la presión en el mismo lado. Este concepto también planteaba problemas, esta vez porque, al ser tan larga la plancha, se combaba con facilidad, variando la presión que ejercía sobre la cremallera en función de la distancia, llegando hasta el punto en el que en ciertos sitios se salía la cremallera de los "raíles" impresos para ella.

Por lo tanto, finalmente se decidió separar la parte de las cremalleras y motores de la parte de los servomotores.

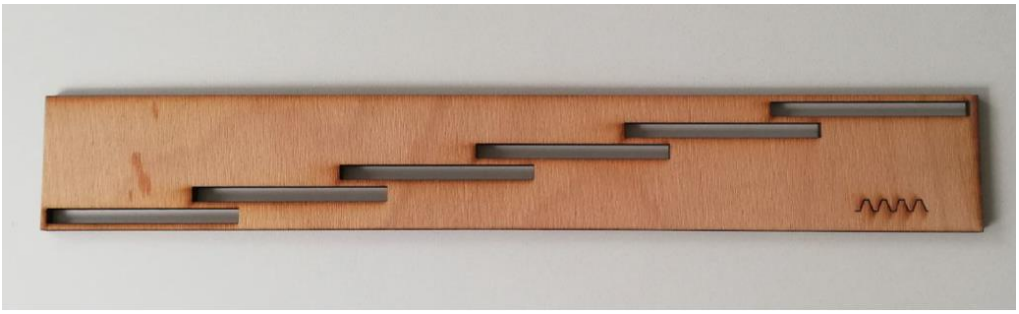
2.2 Estructura del mástil

Una vez separadas las estructuras, el diseño quedó más simplificado. Ya no era necesario tener una plancha tan grande, por lo que el problema de combado ya no era tan grave. Se hizo un soporte para abrazar la plancha a la parte estrecha del cuerpo de la guitarra, de manera a garantizar que estuviese siempre alineada con el mástil, y en la otra punta se agarraba al clavijero con otras piezas de madera, que a su vez conectaban con la parte de los motores.



Esta estructura era desmontable, ya que había que poner aceite en la plancha para que deslizase mejor, limar, colocar las cremalleras cuando se salían y lo más importante, guardarlo, por lo que no todas las piezas estaban pegadas.

Por lo raíles que se habían construido en la plancha debían circular las cremalleras, por lo que los motores con los engranajes debían estar bien colocados para que no estuviesen dobladas. Tras medir las alturas correspondientes, se diseñó una plancha en la que se colocarían los motores, con sus ranuras para que pudiesen pasar los engranajes.



No valía además con simplemente apoyar el motor sobre esta plancha. Debido al par ejercido sobre la rueda dentada, de no sujetarlo rotaría sobre sí mismo en vez de hacer avanzar la cremallera por el mástil. Por ello, se ideó un sistema de sujeción que a su vez permitiese el cambio de motor, en el caso de que este no sirviese, o se rompiese en cierto momento. Este sistema consistía en crear unas piezas cuadradas con un agujero, que se pegarían al motor. Estas encajarían en unas “paredes” que se construirían sobre la pieza de la imagen superior. De esta manera, se evitaba el problema del par sobre el motor. En la imagen inferior se puede ver el motor encajado en su posición.

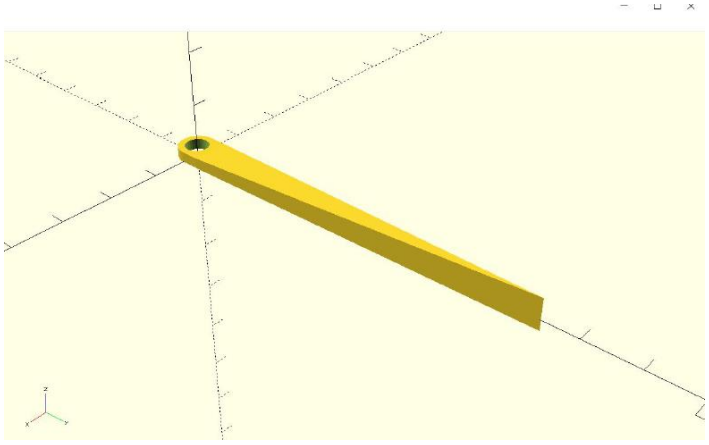
Pese a que se previó la estructura para poder incorporar un motor para cada cuerda, por motivos prácticos se pidió un único motor para hacer las pruebas y probar toda la estructura y metodología escogida, debido a que, si finalmente era necesario cambiar, sería mejor haber comprado sólo uno en vez de seis. Por ello, no se acabaron de comprar los motores y se dejó el aparato funcionando únicamente sobre la tercera cuerda, pero con una estructura perfectamente adaptada a las seis, requiriendo únicamente la compra de los motores y reimpresión de las piezas asociadas a estos. La estructura del mástil con el motor, rueda dentada y cremallera instalados quedaría así.

2.3 Estructura de rasgueo

Hecha la distinción entre las partes, la estructura de rasgueo se hizo de una manera muy rápida, ya que lo único que se necesitaba era suspender sobre las cuerdas los servomotores de manera que pudiesen rascar la cuerda cuando la cremallera hubiese llegado a su destino. Para ello, se diseñó una estructura similar a la parte que abrazaba la guitarra. De la misma manera que queríamos tener la opción de cambiar los motores si se averiaban, nos pasaba lo mismo con los servomotores; por ello los sujetábamos utilizando blue tac en vez de pegamento convencional, ya que si no nos veríamos obligado a reimprimir toda la estructura asociada a esta parte en el caso de que ocurriese algún error. Se tomaron las medidas correspondientes para garantizar que los servomotores elegidos cabían los seis uno al lado del otro, cada uno sobre su cuerda correspondiente.

Cuando se tomaron las fotos aún no estaba instalado el servomotor correspondiente a la tercera cuerda, debido a que, al estar las dos partes controladas por la misma *PyBoard* y ser muy cortos los cables, en los ensayos iniciales no se podían operar los dos de manera simultánea, hasta que finalmente llegó una extensión de cable que se encargó, para poder tener sendos motores instalados y funcionando. En el capítulo de motores hay una foto de los servomotores, donde se aprecia cómo eran.

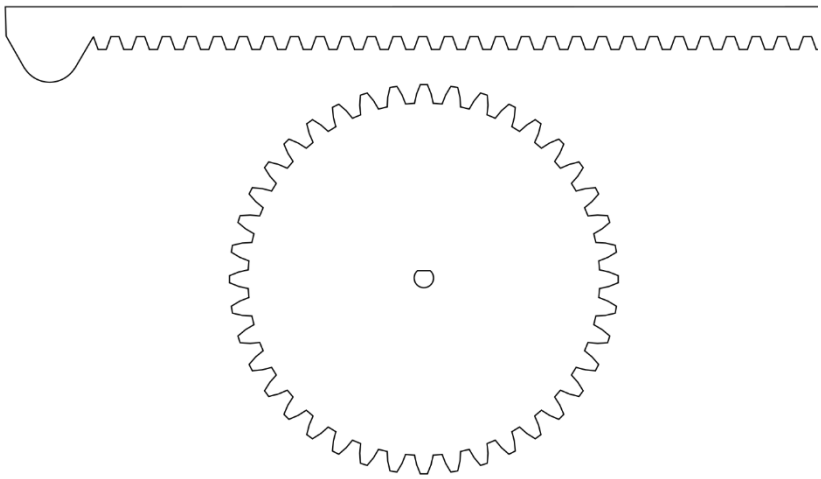
Cuando se comenzaron los ensayos, se utilizaba una de las piezas del proyecto de la flauta, que son los que habían utilizado estos servomotores, para rasguear las cuerdas. En este proyecto se habían diseñado estas piezas para que sujetasen un tapón de oídos sobre los agujeros de la flauta dulce, para poder tocarla. Después de algunos ensayos, se decidió diseñar una pieza más adecuada al rasgueo de las cuerdas, por lo que se creó una pieza en *OpenSCAD* que se pudiese acoplar al servomotor y que tuviese al final forma similar a la de una púa de guitarra. La pieza finalmente quedó así:



2.4 Cremallera y rueda dentada

Habiendo decidido como metodología el mecanismo de cremallera y rueda dentada, había que crear las piezas para realizar los ensayos. En un primer momento, todas las piezas se construyeron con la ayuda del programa *Inkscape*, debido a que traía incorporado un mecanismo generador de ruedas dentadas y cremalleras, los cuáles engranaban perfectamente entre sí. Con este programa pudimos hacer varias pruebas, probando distintos radios de ruedas para ver cuál era el mejor tamaño en lo relativo a velocidad y viabilidad de incorporación a la estructura.

Imprimiendo las piezas en madera con la ayuda de la cortadora láser, la cual operaba con el programa de *CorelDRAW*, apareció un problema que no se había detectado con anterioridad. Los dibujos de *Inkscape* son dibujos vectoriales, por lo que la escala no debería afectar a la calidad del diseño. Sin embargo, al pasar el fichero de este programa al de la cortadora láser, estos dibujos se renderizaban mal, dejando de engranar entre ellos. Los dientes dejaban de ser uniformes, perdiendo calidad y volviéndose desiguales. Además, pese a que se definía el mismo paso entre la rueda y la cremallera, y en el programa se veía que efectivamente engranaban bien, una vez impresos desengranaban tras un giro de aproximadamente 180º. Por lo tanto, se decidió buscar otro programa para crear estas piezas.



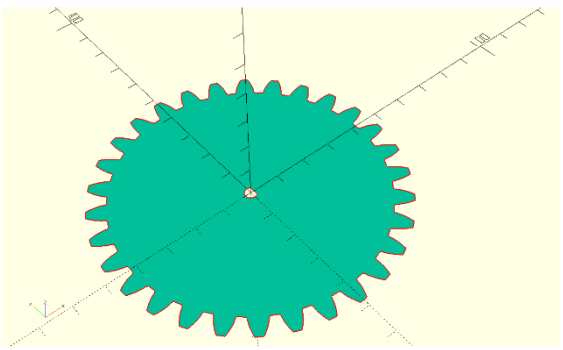
La siguiente opción que se probó y la que finalmente se utilizó, fue el programa *OpenSCAD*, el mismo que se utilizó para diseñar la pieza a cargo del rasgueo. Este programa es un compilador 3D basado en un lenguaje de descripción textual. Aunque está orientado a la creación de objetos 3D, también se pueden exportar ficheros en 2D con la extensión de fichero .svg, la cual posteriormente se pasaba a *CorelDRAW*, esta vez sin plantear problemas. El código necesario para generar las cremalleras y las ruedas dentadas se sacó de internet[3], de una página de libre acceso. Este código permitía elegir el número de dientes, la longitud de la cremallera, el diámetro de la rueda, etc. El proceso de generación de la cremallera consistía en rotarla alrededor de la rueda dentada, garantizando así un engrane perfecto. Posteriormente, con código propio se modificaba la rueda para que pudiese encajar en el motor del que se disponía y se añadía una punta a la cremallera que era la encargada de ejercer la presión sobre las cuerdas.

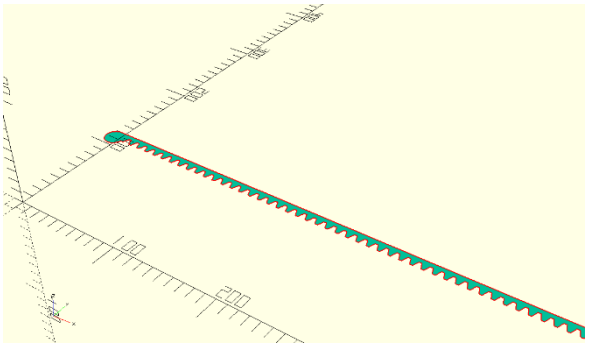
[3] Thingiverse: Cut your own gears with profile shift - OpenSCAD library, by “Parkinbot”.

Finalmente, como la cortadora láser corta únicamente en un plano, para que la punta estuviese más adaptada a la forma de la cuerda e impedir que se salga de esta, se limó esta para crear una pequeña ranura a forma de guía por la que iría la cuerda, como se observa en la siguiente imagen.

Tras haber hecho algunas pruebas en madera, se decidió que sería mejor imprimirlo en metacrilato. Aunque resultaba en una pieza más frágil, deslizaba con un rozamiento mucho menor por las ranuras de la plancha de madera. Las piezas que se habían impreso en madera se quedaban enganchadas y requerían un par mayor para desplazarlas a lo largo del mástil. Aun así, cuando se imprimió la última cremallera, siendo esta última más larga para poder legar a notas más lejanas, se observó que se curvaba mucho la cremallera, no llegaban a tocar los dientes con la cuerda, pero en el caso de la sexta cuerda, en la cual el motor está en la posición más lejana, esto podría conllevar problemas más serios.

Estos son los modelos finales de las piezas en *OpenSCAD*:





3 CAPÍTULO 3: Estudio de los motores

3.1 Elección de los motores

Una vez que la metodología de trabajo estaba decidida, era necesario obtener un motor para poder medir los parámetros, realizar las pruebas pertinentes y diseñar las piezas asociadas a este. Se necesitaba un motor lo suficientemente potente para poder superar la fricción asociada a los trastes, garantizando a su vez un tiempo de respuesta razonable cuando se le pidiese que cambiase de posición. Se decidió por lo tanto utilizar un motor de corriente continua con reductor y codificador rotatorio, para poder calcular la posición a través de este último.

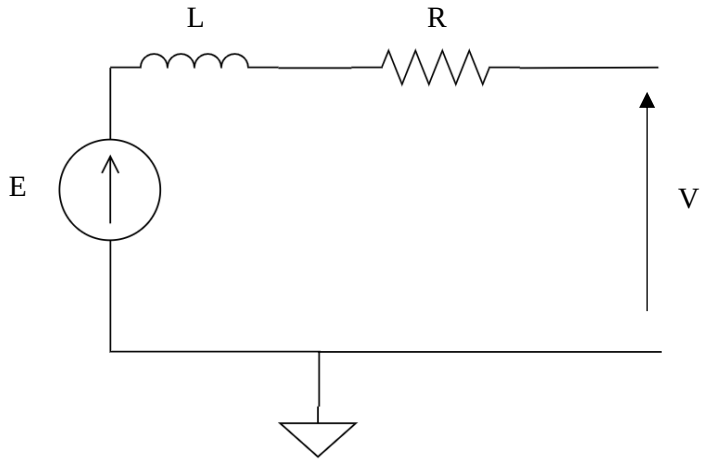
Se encargó el motor de la imagen superior, el "*Motoréducteur 75:1 + encodeur RS003*[4]", con un reductor de 75:1, con una alimentación de 6V. El codificador contaba con una precisión de 1200 impulsiones por vuelta sin tener en cuenta el reductor. Sin embargo, este no es el motor que llegó, dado que el que manejamos se alimentaba a 12V. Como no era el que se había encargado, hubo que volver a tomar las medidas y calcular los parámetros asociados a este. Este motor, además de los problemas asociados a no ser el que se había encargado, era muy frágil por la parte del codificador. Las soldaduras estaban colocadas en una posición muy incómoda, por lo que era complicado fijarlo a la estructura. Realizando las pruebas se soltaban con facilidad, por lo que finalmente se encargó otro motor, esta vez con otro proveedor para recibir el deseado.

Se observó durante los ensayos que la relación de reducción (75:1) era excesiva, por lo que el segundo motor se encargó con una relación menor (34:1). El motor que se encargó esta vez fue el "*Motoréducteur + encodeur FIT0521*[5]". Este motor venía también con los cables ya instalados contrariamente al primero, y con un codificador. Esta vez se recibió el deseado, y ser realizaron los cálculos pertinentes.

[4] Ver referencias y anexo para la ficha técnica
[5] Ver referencias y anexo para la ficha técnica

3.2 Cálculo de los parámetros del motor

El motor elegido era de corriente continua, por lo que se modeliza su comportamiento utilizando el siguiente esquema (en convención receptor):



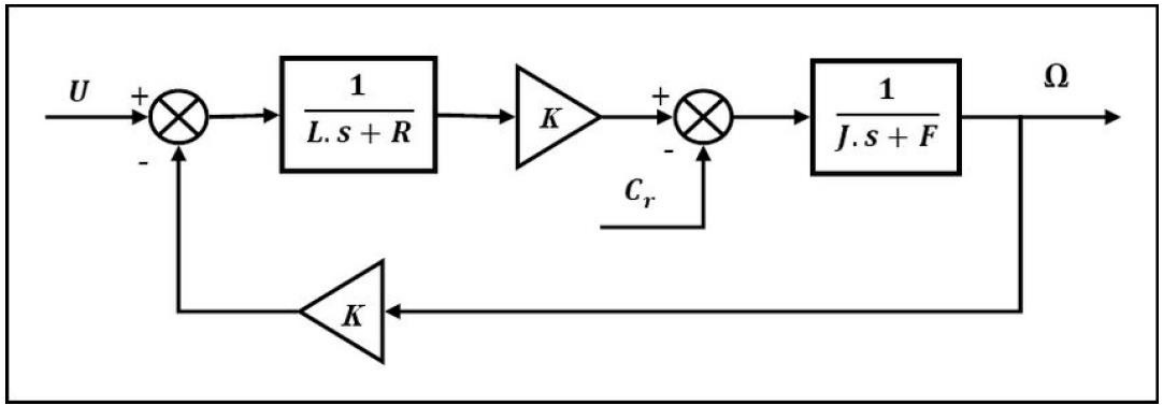
Con este esquema, y conociendo el funcionamiento de un motor de corriente continua se obtienen las ecuaciones $E = K * \omega(t)$, $Cm = K * i(t)$ y $Cf = f * \omega(t)$, siendo ω la velocidad de rotación del motor e i la corriente en el motor y f el coeficiente de rozamiento del motor.

Se conocen además las igualdades $V(t) = E + L\frac{di}{dt}$ y $J\frac{d\omega}{dt} = Cm - Cr - Cf$, siendo J la inercia del motor, Cm el par del motor, Cr el par de rozamiento de la cremallera con la plancha de madera y Cf el par de rozamiento del motor.

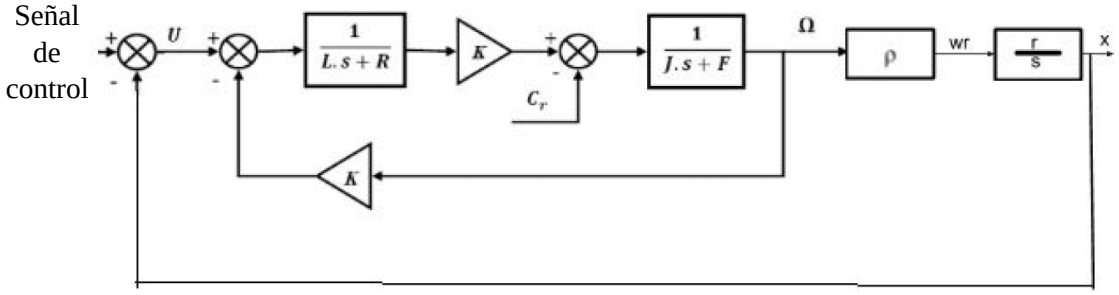
Aplicando la transformada de Laplace y manipulando las ecuaciones se llega a la relación:

$$\Omega(s) = \frac{K * I(s) - Cr}{J * s + f}$$

Gracias a todo esto se construye el esquema de bloques en *Simulink* para nuestro motor de corriente continua:



Ahora, se busca a representar el sistema completo, con el reductor, y el desplazamiento de la cremallera. Por lo tanto, incorporamos al esquema el bloque correspondiente a la transformación de las velocidades por el reductor, siendo ρ la relación de reducción y r el radio de la rueda dentada. En el siguiente esquema no se tiene en cuenta la calidad del codificador, considerando que nos envía en todo instante la posición x de la cremallera. Esto puede justificarse teniendo en cuenta la escala de la rueda dentada y del codificador, tenemos un error del 0.8mm, lo cual es bien inferior a la distancia entre trastes, por lo que no tendrá repercusiones en la práctica.



Con este esquema de bloques se puede simular el comportamiento del sistema, y permite determinar un corrector PID para poder controlarlo. Sin embargo, antes de poder calcularlo, es necesario determinar todas las constantes que intervienen en esta regulación, es decir, L, R, K, J y f, las cuales no todas aparecen en la ficha técnica del fabricante. Todos los cálculos y resultados que se muestran corresponden al segundo motor.

Para simplificar los cálculos, se supone que el motor es un cilindro homogéneo. Teniendo una masa de 96g y un diámetro de 24,4mm; el momento de inercia resultaba en:

$$J = \frac{1}{2} * m * r^2 = \frac{1}{2} * 0.096 * \frac{0.0244^2}{4} = 7{,}14 * 10^{-6} kg * m^2$$

Para calcular la resistencia, se bloquea el motor y se impone una tensión continua en las bornas, para a continuación medir la intensidad. En la teoría, tendríamos que $E = 0, L$ se comportaría como un cable, por lo que tendríamos únicamente $U = R * I$. Cuando se trató de calcular la resistencia de esta manera, se apreciaba mucho ruido, y había un error muy grande, pero los valores entre los que oscilaba se asemejaban al valor del fabricante, por lo que se utilizó este, resultando en:

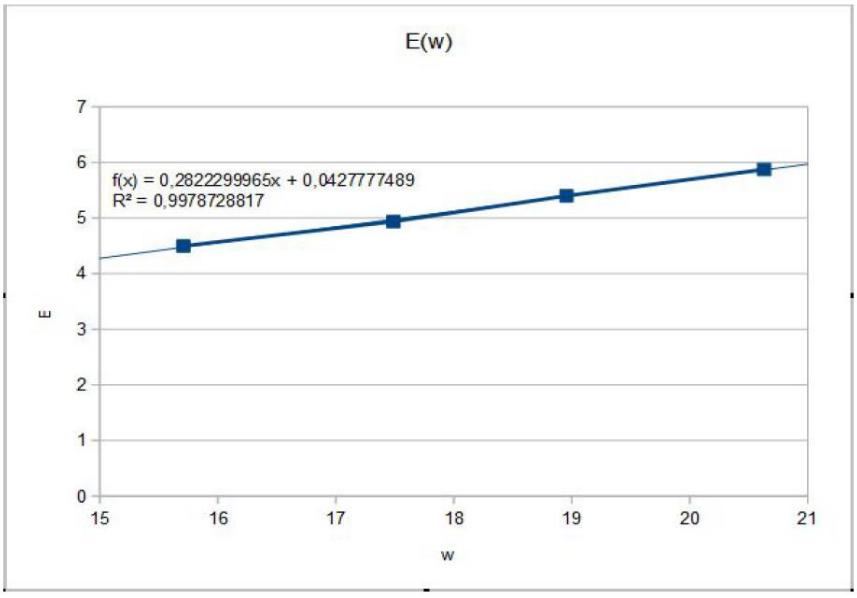
$$R = \frac{U}{I} = \frac{6V}{3{,}2A} = 1{,}875\Omega$$

El coeficiente K se puede calcular imponiéndole al motor un régimen estático, y midiendo los valores de U, I y ω. Teniendo las dos ecuaciones en las que aparece la constante K y despejando, se llega a la siguiente ecuación y resultado:

$$K = \frac{U - R * I}{\omega} = 0{,}285 \frac{V * s}{rad}$$

Para calcular L, uno debe situarse en régimen transitorio. Se mantiene bloqueado el motor con la mano, y se suelta para observar la curva de corriente, utilizando esta para calcular L, ya que $V = L * \frac{di}{dt}$. Sin embargo, tras varios ensayos no fue posible calcularla de esta forma debido al ruido, y con los valores que aparecen en la ficha técnica tampoco es posible calcularla. Por lo tanto, tras algunos ensayos se decidió elegir como valor 1mH ya que parecía estar en el buen orden de magnitud.

Finalmente, sólo quedaba por determinar el valor de la constante de rozamiento f. Para calcular este valor se utiliza la relación estática $E = f * \omega + Cs$, siendo Cs el par de frotamiento seco. Tomando valores para distintas velocidades, se traza la siguiente curva:



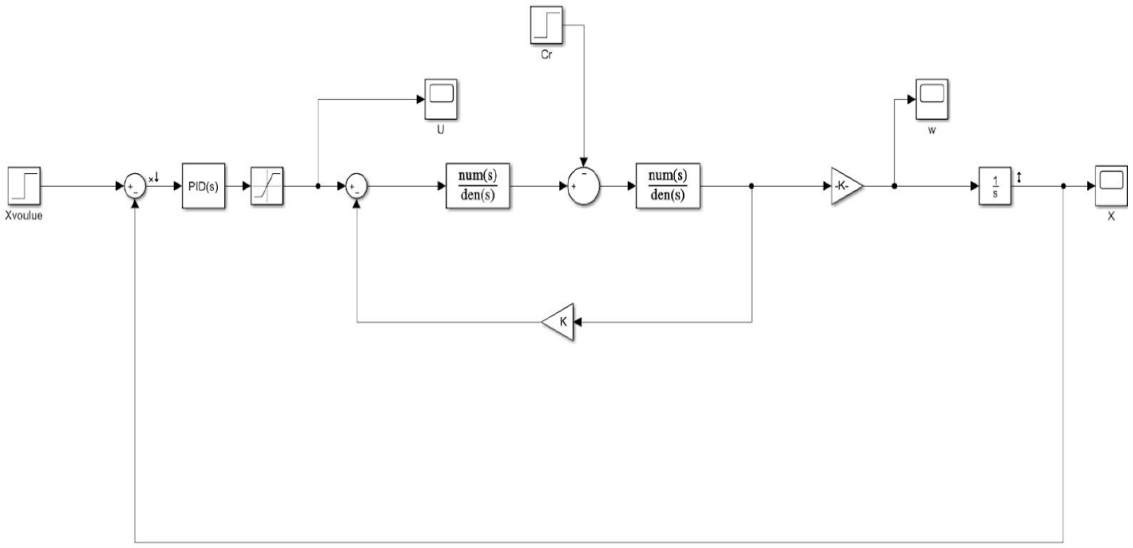
Se obtiene, por lo tanto, un valor de $f = 0{,}282 \frac{N*m*s}{rad}$ con un coeficiente de correlación muy próximo a uno.

3.3 Simulación

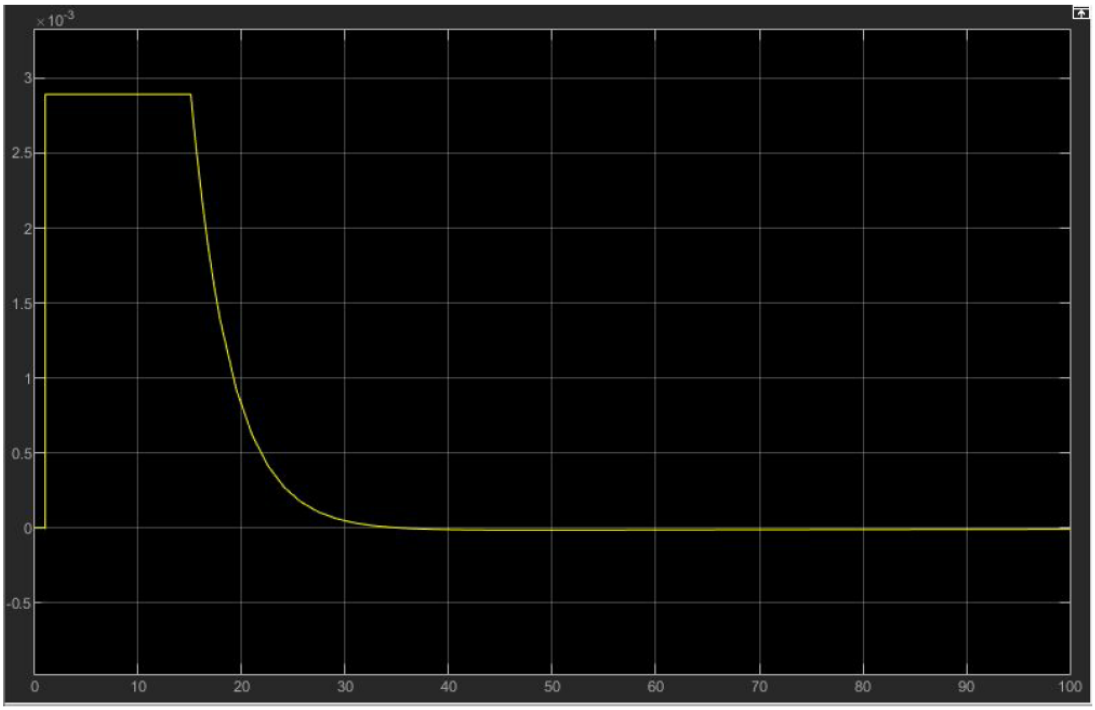
Una vez se han obtenido los valores de las constantes del motor, se pueden comenzar las simulaciones en *Simulink*, un entorno de programación visual, que funciona sobre el entorno de programación *Matlab*. Con estas simulaciones, se busca a definir las constantes de un corrector PID.

Se consideran dos tipos de señales de control. La primera consiste en hacer funcionar a su máxima capacidad el motor, y apagarlo una vez llegue a la posición deseada. Para pilotar de esta manera el motor, no es necesario incorporar un corrector PID. La otra señal, buscaría a ir frenando poco a poco a medida que la punta de la cremallera se fuese aproximando a la posición deseada. Para esta señal de control, es necesario incorporar como mínimo un corrector PI para evitar un error estático en la salida.

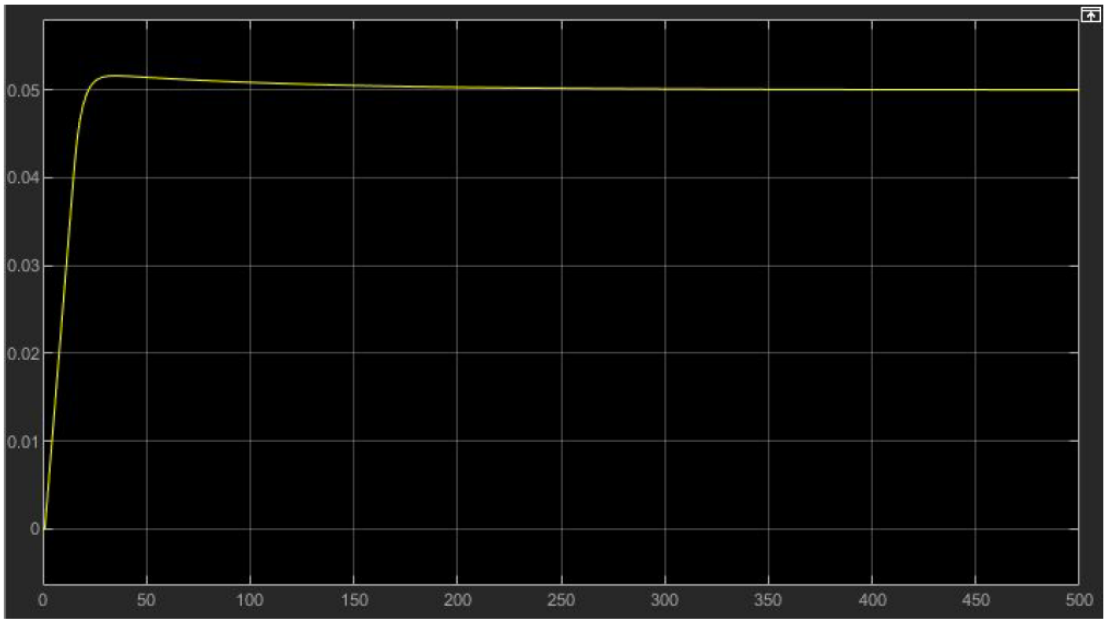
Se añade, por lo tanto, para las simulaciones, un bloque PID del que se calcularán los valores, además de un saturador de señal a la entrada para tener en cuenta el que no se superen los 6V en la entrada del motor.



Después de varias simulaciones, se obtienen como los valores más adecuados para el corrector PID los siguientes: P=500, I=0,01 y D=0,5; obteniendo con ellos las siguientes curvas para una señal de desplazamiento de 0,05m.



Curva de velocidad



Curva de desplazamiento

Como se puede apreciar en las curvas, no aparece ningún error estático gracias a la acción integral del corrector, y tenemos un tiempo de respuesta razonable, del orden de 50ms para este desplazamiento. Conservamos, por lo tanto, estos valores para el PID.

3.4 Los servomotores

Los servomotores que se utilizaron en el proyecto se heredaron del de la flauta dulce. El modelo era un *Servo motor SG90*, más adelante en la parte del código se mostrará cómo se pilotaba. Lo único que debía hacer este motor era girar un ángulo de 15º, que es lo que se determinó, para rasguear la cuerda una vez la cremallera hubiese llegado a la posición que se había indicado. La imagen de a continuación muestra como era el motor, siendo distinta solamente la punta encargada del rasgueo.

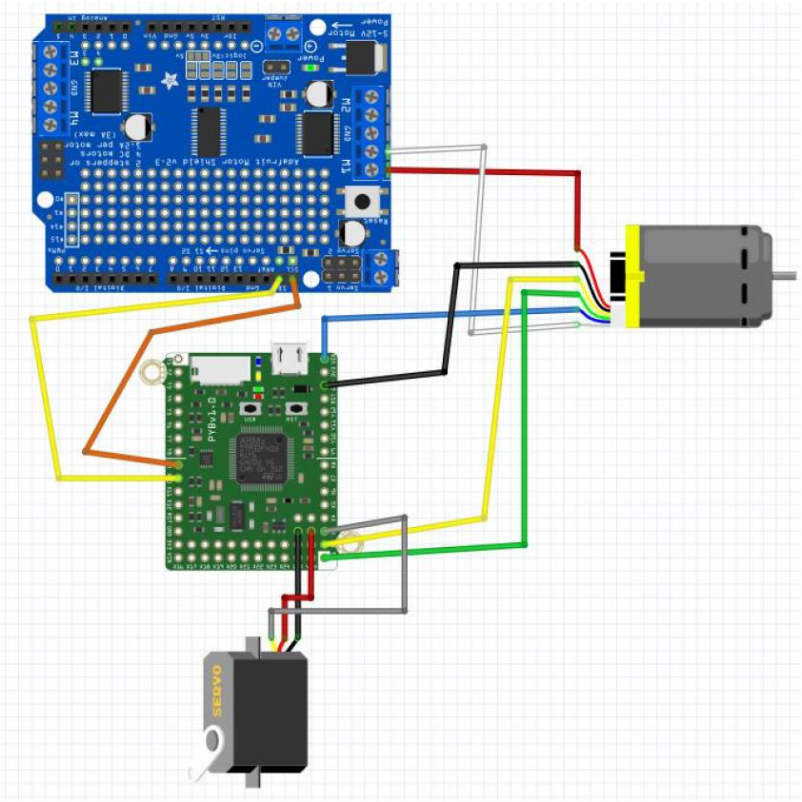
Imagen del servomotor utilizado[6]

[6] Imagen obtenida de la página web “Colombianízate”

4 CAPÍTULO 4: Software de control

4.1 Control vía *Python*

Para controlar el sistema se utilizó tanto un microcontrolador *PyBoard*, como un *Adafruit Motorshied*. Estos dos en conjunto permiten controlar la velocidad del motor, a la vez que los servomotores.



La *PyBoard* permite controlar todo implementando código *Python*. Se creó un fichero llamado *asservissement.py* (en español sería control.py), en el que se definían todas las variables y funciones necesarias del sistema. A continuación, basta con importar el fichero en el *main* del microcontrolador.

```
1   import asservissement as ass
```

A continuación, se presentarán las principales funciones del código, pero no se mostrará al completo. Se añadirá en anexo todo el código para los interesados.

La primera función que había que definir era la de inicialización del sistema (función ini()). Con esta función, se aseguraba que la cremallera estuviese en la posición inicial (primer traste), y a continuación los valores de todas las variables de posición se ponían a cero. Esto se conseguía haciendo girar el motor en el sentido contrario, y cuando el sistema detectase que la variable del codificador ya no está cambiando, utilizando un *time.sleep* de medio segundo, se supone que ya ha llegado al final del recorrido, por lo que se para el motor y se ponen a cero las variables.

```
#fonction d'initialisation, pour qu'au départ le moteur se retrouve en butée
def ini():
    global a_count
    global dist
    global b_count
    m.throttle(-3500) #on fait tourner le moteur en sens inverses jusqu'en buté
    b0=b_count        # on lit la valeur de l'impulsion à l'instant présent
    time.sleep(0.5)
    b=b_count         # on lit la valeur de l'impulsion 0.5 secondes après
    while b0 != b:      # tant que les valeurs d'impulsions ne sont pas identiques on n'est pas en butée
        b0=b_count
        time.sleep(0.5)
        b=b_count
    m.brake()
    a_count,b_count,dist=0,0,0   # on initialise les compteurs d'impulsions et la variable dist qui représente la position de la tige
```

Como se probaron dos señales distintas, se crearon dos funcionas para pilotar por los dos métodos distintos: la función de *bourrin*(x), que en español se podría traducir por “a la fuerza”, y la función *deplacer*(x), en español desplazar. La función *bourrin*(x) funciona como su nombre indica, toma como argumento la distancia a la que debe desplazarse, hace girar el motor hasta que se llega a la posición deseada, donde lo para en seco.

```
#Méthode bourrine, le moteur tourne à fond jusqu'à ce que l'on atteigne la position souhaitée
def bourrin (x):
    global a_count
    global d
    global dist
    z=0
    eps=x-a_count*d
    while abs(eps)>0.001:
        if signe(eps):
            m.throttle(4096)
        else:
            z=a_count
            a_count=z-(a_count-z)
        dist=a_count
        eps=x-a_count*d

    m.brake()
```

La función correspondiente a la segunda señal tiene un código más complejo como cabe a esperar. En este se incorporan los elementos calculados del PID.

```
def deplacer(x):     #x correpond à la distance à parcourir en mètre
    global a_count
    global I
    global Kc
    global Td
    global r
    global s
    global d
    maxSomme=1000
    z=0                 # z est utile lorsque l'on se déplace vers l'arrière
    micros = pyb.Timer(2, prescaler=83, period=0x3fffffff)  # on créé un timer en microseconde (pour le dérivateur)
    micros.counter(0)   # on l'initialise à 0
    eps0=0              #erreur0 en metre
    t0=micros.counter()
    eps1=(x-a_count*d)   # erreur 1
    print(eps1)
    t1=micros.counter()     # t1- t0 correspond à la durée séparant 2 mesures d'erreur (c'est la période d'échantillonage)
    if eps1*(t1-t0)>maxSomme:
        somme=maxSomme
    else:
        somme=eps1*(t1-t0)
```

```
    else:
        somme=eps1*(t1-t0)
    while abs(eps1)>0.0008:
        Vcons=Kc*(eps1+I*somme+Td*(eps1-eps0)/(t1-t0))
        print("V =", Vcons)     # on génère la consigne en fonction de l'erreur de son intégrale et de sa dérivée
        if abs(Vcons)>6:    #si consigne en tension trop élevée alors on sature la consigne pour ne pas abîmer le moteur
            if signe(Vcons):     #(le moteur tourne alors à sa vitesse max)
                m.throttle(4096)
                eps0=eps1
                t0=micros.counter()
                eps1=(x-a_count*d)
                print(eps1)
                t1=micros.counter()
                if eps1*(t1-t0)>maxSomme:
                    somme=maxSomme
                else:
                    somme=eps1*(t1-t0)
            else:
                z=a_count
                m.throttle(-4096)
                eps0=eps1
                t0=micros.counter()
                a_count=z-(a_count-z)
                eps1=(x-a_count*d)
                print(eps1)
                t1=micros.counter()
                if eps1*(t1-t0)>maxSomme:
                    somme=maxSomme
                else:
                    somme=eps1*(t1-t0)
        else:
            z=a_count
            m.throttle(int(Vcons*s))
            if not signe(Vcons):
                eps0=eps1
                t0=micros.counter()
                a_count=z-(a_count-z)
                eps1=(x-a_count*d)
                print(eps1)
                t1=micros.counter()
                if eps1*(t1-t0)>maxSomme:
                    somme=maxSomme
                else:
                    somme=eps1*(t1-t0)

            else:
                eps0=eps1
                t0=micros.counter()
                eps1=(x-a_count*d)
                print(eps1)
                t1=micros.counter()
                if eps1*(t1-t0)>maxSomme:
                    somme=maxSomme
                else:
                    somme=eps1*(t1-t0)
    m.brake()
    print(eps1)
    print(a_count*d)

```

Después de los primeros ensayos, se observó que el código que regulaba el motor con el PID era menos eficiente que el que empleaba el método más simple, debido a que, al frenar antes de pasar el hierro de los trastes, se quedaba bloqueado en la posición errónea. Por lo tanto, se decidió utilizar el método más simple para pilotar el aparato.

Se observó también, que cuando el movimiento era en el sentido contrario, yendo de una nota más aguda a una más grave, había un ligero desfase entre las posiciones. Los contadores *a_count* y *b_count*, encargados de contar las impulsiones enviadas por el motor, crecen, aunque el motor gire en el sentido negativo. En la función *bourrin(x)* se puede ver que para hacer decrecer los contadores se almacena en una variable z uno de los valores, por ejemplo, *a_count*, en el momento en el que el motor se pone a girar en el sentido negativo (y cada vez que se entra en el bucle). A continuación, justo antes de pedir al motor que se detenga, se da a *a_count* el valor z-(*a_count*-z) para simular el decrecimiento de *a_count*.

Teóricamente, si todas las líneas de código se ejecutasen simultáneamente y el motor se detuviese de una forma perfecta cuando llegase a la posición deseada, entonces el valor almacenado en *a_count* debería corresponderse con l aposición real de la punta de la cremallera. Sin embargo, en la práctica puede haber ligeros desfases entre el momento en el que se le da un nuevo valor a *a_count* y el momento en el que se detiene. Este problema conlleva el tener un valor de *a_count* muy grande respecto a la posición real de la cremallera. Este desfase entre el contador y la posición real puede provocar, por lo tanto, un error de posición que puede resultar en que la posición real este en un traste distinto al deseado.

Para corregir este problema, se definió una nueva función, del mismo tipo que la primera *bourrin(x)*, pero que razone en términos de desplazamiento relativo en vez de posición absoluta. A esta función se le llama *bourrin2(x)*.

```
87  def bourrin2 (x):
88      global a_count
89      global d
90      global dist
91      dist+=x
92      a_count=0
93      eps=x
94      while abs(eps)>0.001:
95          if x>=0:
96              m.throttle(4096)
97              eps=x-a_count*d
98          else:
99              m.throttle(-4096)
100             eps=x+a_count*d
101     m.brake()
102
103
```

En esta, *bourrin2(x)* toma como argumento el desplazamiento deseado con respecto a la posición actual en vez de la posición final. Por ello, cada vez que se llama a la función *bourrin2(x)* se reinicializa a cero el contador *a_count* y se almacena el valor de la nueva posición actual en una variable global llamada *dist*. Tras unos ensayos se comprobó que el problema había sido resuelto, y funcionaba sin ningún tipo de desfases, y por ello es la función que se utilizó para el control final. Una vez se tenía la función encargada del desplazamiento, se creó una función que uniese el concepto de nota con el desplazamiento hasta esta.

Para crear esta función, primero se creó una lista de valores, donde se almacenaron las distancias reales en metros de los trastes. Como finalmente sólo se implementó en una cuerda, los valores que se registrando son únicamente los de esa cuerda, la tercera. A continuación, la función *jouer_note(note)*, en español “tocar nota”, toma como argumento una de las notas, y desplaza la cremallera la distancia necesaria.

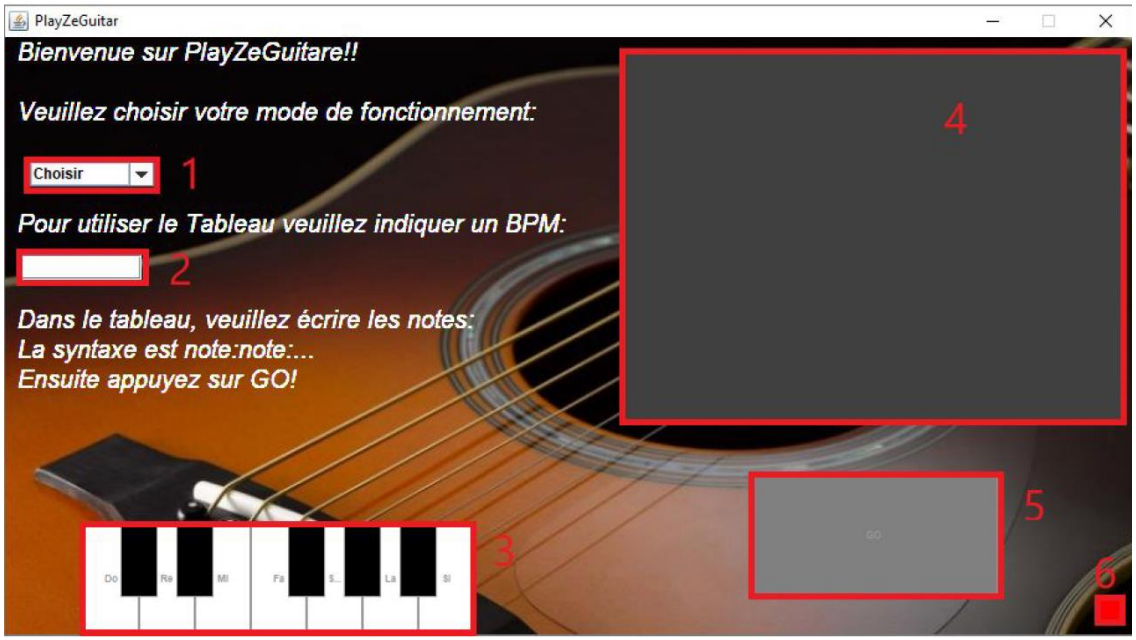
```
199  def jouer_note(note):
200      global angle
201      global dist
202      k=0
203      while note!=Liste_note[k][0]:
204          k+=1
205      bourrin2(Liste_note[k][1]-dist)
206      angle=-angle
207      ser.angle(angle)
208
```

En esta función se llama a la función *bourrin2(x)* para que realice el desplazamiento, y una vez se ha parado el motor, mueve el servomotor, ubicado encima de la cuerda, para que rasguee y suene la nota.

Con estas funciones se puede por lo tanto tocar la guitarra utilizando la *PyBoard*, siendo esta controlada con *Putty*. Lo siguiente sería crear una interfaz gráfica para un usuario cualquiera pueda utilizar el aparato, sin necesidad de conocer el funcionamiento de la *PyBoard* vía un *Shell*.

4.2 Interfaz Gráfica

Para desarrollar la interfaz gráfica, se decidió de desarrollarla con Java, en particular con la biblioteca *Swing*. Al igual que con el código *Python*, no se entrará en los detalles del código, pero estará en el anexo para los interesados.



En la interfaz se puede leer: “Bienvenidos a PlayZeGuitar!! Por favor, seleccionen un modo de funcionamiento:”, a continuación en el elemento 1 se seleccionaría el método, “para utilizar el tablero indique los BPM”, entre el elemento 1 y el 2, se pide precisar los BPM para utilizar la función de tablero; y finalmente “En el tablero, escriba las notas: La sintaxis es nota:nota:… ¡A continuación, apriete sobre GO!”. La ventana gráfica es un JFrame, y en el mismo fichero se incluye un documento de texto llamado *readme.txt* para explicar el funcionamiento del software. En esta interfaz hay seis elementos principales, se explicarán en el orden en el que están numerados en la imagen.

En primer lugar, hay un menú desplegable, en el cual se pueden elegir dos opciones, obviando el estado por defecto de elección. Las opciones que hay son piano y tablero. Seleccionando piano, se habilitará el teclado virtual, señalado con el número 3 en la imagen; y seleccionando la opción de tablero, se deberán definir los BPM y escribir las notas en el tablero. Este menú se creó en Java utilizando un *JComboBox*.

El segundo elemento de la interfaz es un *JTextField*, donde se deberá completar los BPMs (pulsos por minuto en español) con los que se desea que se toquen las notas. Para garantizar un buen funcionamiento, y que la cremallera pueda llegar a una nota antes de tener que moverse a la segunda se escoge un valor entre 1 y 60.

El tercer elemento, es un teclado, formado a partir de varios *JButton*, cada tecla siendo uno distinto. Este piano sólo funciona si se ha seleccionado previamente la opción “piano” en el primer elemento. Una vez se presione una tecla, el aparato tocará la nota correspondiente.

Una vez se seleccione la opción de tablero y se completen los BPMs, en el tablero de la derecha se escriben las notas que se desean tocar, siguiendo el formato “nota:nota:nota…”. Este tablero es un *JTextArea*. Después, apoyando en el botón señalado con un cinco, se tocarán las notas elegidas en acorde con los pulsos por minutos que se han definido en el elemento 2.

El elemento seis es simplemente, un pequeño cuadrado que estará de color rojo si no se detecta la *PyBoard* y de color verde si está bien conectada. En el caso de que este rojo, hará falta reiniciar el programa una vez se haya conectado correctamente.

4.3 Conexión a la *PyBoard*

La comunicación entre la interfaz gráfica y la *PyBoard*, se utilizó la biblioteca *JSerialComm* junto con la clase *PybSerialComm*[7] desarrollada por el director del proyecto, Frédéric Boulanger. Una vez que la *PyBoard* está conectada al puerto USB del ordenador y el driver asociado está instalado, se puede comenzar a desarrollar la comunicación.

Se asignó a cada elemento un *ActionListener*, que utilizando como intermediarias a las funciones de la clase *PybSerialComm*, envía una o varias órdenes a efectuar a la

[7] Clase de *Python* creada por el director del proyecto, Frédéric Boulanger

PyBoard. Por ejemplo, aquí tenemos la implementación que va asociada a la tecla del piano “Do”, la cual cada vez que es tocada, le pide a la *PyBoard*, gracias a la función *startCommand* de la clase *PybSerialComm*, de lanzar la orden *ass.jouer_note(do)*, es decir, lanzar la función a cargo de tocar las notas dando como argumento la distancia a la nota “Do”.

```
1  package Logiciel;
2
3  import java.awt.event.*;
4
5
6
7
8
9  public class ActDo implements ActionListener  {
10
11
12
13     public void actionPerformed (ActionEvent e)  {
14         //System.out.println("On joue la note Do");
15         String cmd="ass.jouer_note('do')";
16         Main.pyb.kill();
17         try{
18             Main.pyb.startCommand(cmd);
19         }
20         catch(PybSerialComm.PyboardException ev) {
21             System.exit(1);
22         }
23     }
24 }
25
```

Este es el principio que se siguió para el resto de los elementos de la interfaz gráfica, se pueden encontrar en anexo los demás.

Conclusión

Este proyecto se comenzó con unas ideas que fueron evolucionando y modificándose a lo largo de todo su desarrollo, como suele pasar con cualquier tipo de proyecto. Siendo un trabajo que se ofrece todos los años, se terminó el curso con un producto funcional y terminado, requiriendo únicamente la compra de los motores adicionales para su funcionamiento; resultando en un proyecto que se podrá continuar fácilmente en años posteriores, mejorándolo cada año con nuevas funciones y retoques.

El objetivo de este proyecto era llegar a controlar vía remoto un instrumento, y se puede afirmar que se ha llegado a ese objetivo. La estructura, la cual estuvo en constante evolución, y fue objeto de numerosas iteraciones, quedó finalmente perfectamente adaptada para incorporar todos los motores necesarios. El aparato a cargo del rasgueo, si bien puede parecer rudimentario, es efectivo y tiene un muy buen tiempo de respuesta. El control del motor es mejorable, debido a que finalmente se utilizó la orden de control más simple, pero se podría mejorar modificando la guitarra, y finalmente la interfaz gráfica es fácil e intuitiva, y con el fichero que se incluye con las instrucciones, permite a cualquier usuario utilizarlo.

A medida que se iba progresando en el desarrollo del aparato, se iban descubriendo nuevas opciones de mejora, muchas de las cuales no daba tiempo a desarrollarlas en un año. Si se retoma este proyecto en el futuro, estos son distintos caminos por lo que se puede enfocar la continuación de este.

Una de las mejoras que resultaría más interesantes, obviando la instalación de los últimos motores, sería la implementación de la lectura de ficheros MIDI al software del equipo. Con ello, si se tuviese cualquier canción o partitura en este formato, cargándola en el programa, el aparato podría tocarla automáticamente, sin tener que cargar manualmente las notas. Otra idea que iría de la mano de la anterior sería la programación de acordes en vez de notas. Una vez estuviesen instalados los motores sobre todas las cuerdas, en lugar de pedir la nota “Do”, o cargar a mano las seis ordenes para las cuerdas correspondientes a este acorde; con una simple orden se podrían colocar las seis cremalleras en su posición correspondiente.

Otra opción de mejora, esta más orientada a la parte mecánica del aparato, sería el eliminar por completo los herrajes de los trastes. De esta manera se reduciría mucho el ruido que hacen las cremalleras al pasar sobre estos, se reduciría también el rozamiento, por lo que disminuiría el tiempo de respuesta, y, sobre todo, se podría utilizar la función calculada con el PID, la cual es una orden de control mucho más adecuada para el control de un motor.

Finalmente, en una nota más personal, este proyecto me motivó mucho y me obligó a investigar y aprender a utilizar programas que no habría utilizado en otras situaciones. Gracias a él pude experimentar el cómo sería desarrollar un producto o llevar a cabo un trabajo de investigación en un entorno laborable, teniendo que buscar nuevas soluciones al encontrar obstáculos y analizar bien las distintas opciones antes de tomar una decisión importante, para no perder tiempo ni dinero teniendo que dar marcha atrás. Este proyecto ha sido muy gratificante para mí, el verlo crecer y evolucionar hasta

convertirse en un producto final. He disfrutado haciéndolo y el poder aplicar lo que he estado aprendiendo durante esto años a algo concreto más allá de un examen ha sido una experiencia única para dar por finalizado el estudio del grado de ingeniería.

Referencias

[1] Denis Tian, *Control de Ukelele por control remoto*, *Projet long* (equivalente a proyecto de fin de grado en España), *École CentraleSupélec.* Documento privado de la universidad CentraleSupélec, curso 2017-2018.

[2] Alexandre Broussadier, *Control de flauta dulce por control remoto*. *Projet long* (equivalente a proyecto de fin de grado en España), *École CentraleSupélec.* Documento privado de la universidad CentraleSupélec, curso 2017-2018.

[3] Thingiverse: Cut your own gears with profile shift - OpenSCAD library, by "Parkinbot": https://www.thingiverse.com/thing:636119. Consultada por última vez el 23/01/2019.

[4] GOTRONIC, dirección de compra del primer motor: https://www.gotronic.fr/art-motoreducteur-75-1-encodeur-rs003-21384.htm. Consultada por última vez el 07/12/2018.

[5] GOTRONIC, dirección de compra del segundo motor: https://www.gotronic.fr/art-motoreducteur-encodeur-fit0521-27897.htm. Consultada por última vez el 12/02/2019.

[6] Colombianízate, Servo motor SG90 – Arduino – Raspberry – Robótica, se utiliza para la obtener la imagen del servomotor utilizado: https://www.colombianizate.com.co/tienda/modulos-arduino/servo-motor-sg90-arduino-raspberry-robotica/. Consultada por última vez el 20/08/2019.

Anexos

A. Ficha técnica del primer motor:

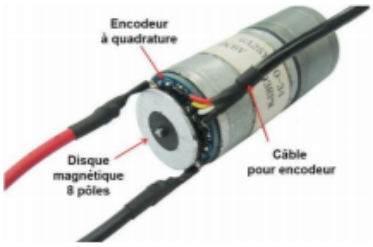
Motor comprado vía la página GOTRONIC el 07/12/2018, ver referencias.

Encodeur RS003

Les motoréducteurs WT341 et WT751 sont équipés d'un encodeur à quadrature.

Ces encodeurs peuvent être utilisés en 3,3 V et 5 V et sont protégés contre les inversions de polarité.

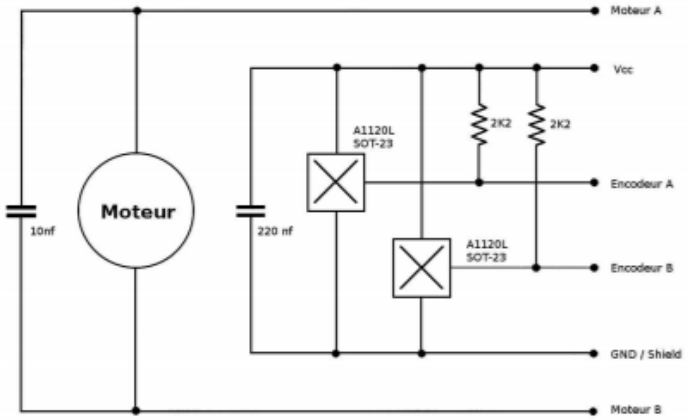
Les encodeurs utilisent des disques magnétiques 8 pôles et 2 capteurs à effet hall pour obtenir deux signaux carrés déphasés de 90° avec un total de 16 changements d'état par tour du moteur.



La résolution du signal est donc de 544 impulsions par tour pour les motoréducteurs 34 :1 (WT341) et 1200 impulsions par tour pour les motoréducteurs 75 :1 (WT751).

Spécifications :

Motoréducteur	**WT341**	**WT751**
Alimentation moteur	6 Vcc (7,5 Vcc maxi)	
Vitesse à vide	295 tours/min	133 tours/min
Consommation à vide	0,35 A	
Consommation moteur bloqué	5,5 A	
Couple moteur bloqué	4,0 kg.cm	8,8 kg.cm
Rapport de réduction	34:1	75:1
Axe	Ø4 x 10 mm avec méplat	
Dimensions	Ø25 x 77 mm (axes inclus)	
Résolution encodeur	16 impulsions/tour moteur 544 imp./tour sortie d'axe	16 impulsions/tour moteur 1200 imp./tour sortie d'axe
Alimentation encodeur Vcc	3 – 24 V	
Sortie A	Signal carré GND-Vcc	
Sortie B	Signal carré GND-Vcc	

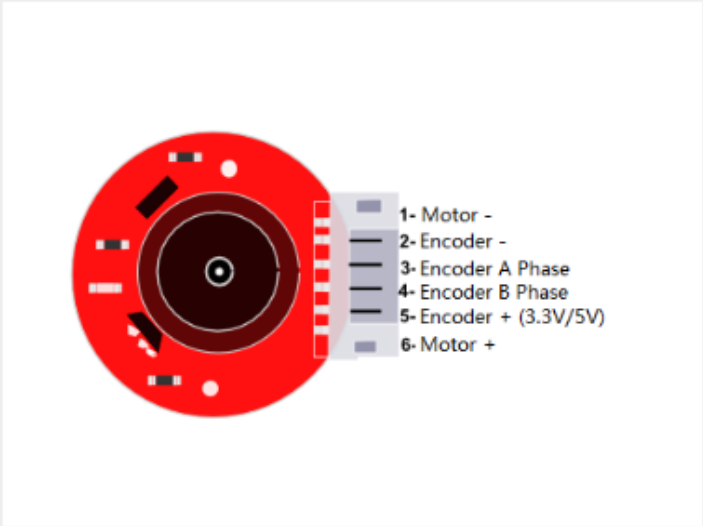


B. Ficha técnica del segundo motor:

Motor comprado vía la página GOTRONIC el 12/02/2019, ver referencias.

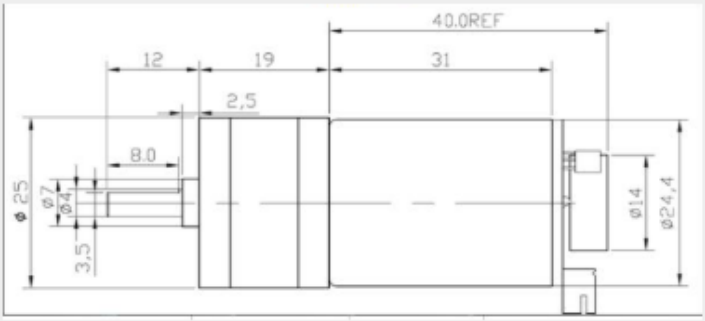
INTRODUCTION

This is a metal geared DC motor w/Encoder. It is a 6V motor with a 34:1 metal gearbox and an integrated quadrature encoder that provides a resolution of 11 counts per revolution of the motor shaft, which corresponds to 341.2 counts per revolution of the gearbox's output shaft.



Pinmap

These units have an 8 mm length, 4 mm-diameter D-shaped output shaft. This motor is intended for use at 6V, though the motor can begin rotating at voltages as low as 1V. This motor is an ideal option for your mobile robot project.



Dimension

SPECIFICATION

Motor Rated Voltage: 6V
Encoder Rated Voltage: 3.3 / 5V
Reducer Reduction Ratio: 1: 34
No load Speed: 210RPM@0.13A
Maximum Efficiency Point: load 2.0kg·cm/170RPM/2.0W/0.6A
Maximum Power point: Load 5.2kg·cm/110RPM/3.1W/1.1A
Stall Torque: 10kg·cm
Stall Current: 3.2A
Hall Resolution: Hall Resolution 11x Precision Reduction Ratio 34.02 = 341.2PPR
Dimension: 52 * Φ24.4 mm / 2.05 * Φ0.96inches
Weight: 96g

SHIPPING LIST

Metal DC Geared Motor w/Encoder - 6V 210RPM 10Kg.cm x1
JST 6-Pin cable x1

C. Código de Python

```
import math
import pyb
import pca9685fb
import adamotshv2fb
import time

# compteur d'impulsions de la phase A de l'encodeur
a_count = 0
# compteur d'impulsions de la phase B de l'encodeur
b_count = 0

# Fonctions de traitement des interruptions
def phaseA(line):
    global a_count
    a_count += 1

def phaseB(line):
    global b_count
    b_count += 1

# Interruptions générées sur les fronts descendants des signaux de
l'encodeur
pyb.ExtInt(pyb.Pin("X1"), pyb.ExtInt.IRQ_FALLING, pyb.Pin.PULL_UP,
phaseA)
pyb.ExtInt(pyb.Pin("X2"), pyb.ExtInt.IRQ_FALLING, pyb.Pin.PULL_UP,
phaseB)

# Initialisation de gestionnaire de PWM
pca = pca9685fb.PCA9685(1)    # SCL/SDA = X9/X10
pca.start()

# Le moyteur est branché sur le port M1 su shield
m = adamotshv2fb.DCMotor(pca,1)     # Motor M1
# Le servomoteur est branché sur X3 de la pyboard
ser=pyb.Servo(3)
angle=-15 #on initialise l'angle du servomoteur
ser.angle(angle)

dist=0
##Partie Asservissement
#on définie les constantes
I=0.01  #Kc, Ti et Td sont les constantes du correcteur PID I=1/Ti
Kc=500
Td=0.5
r=0.045*9/10  # rayon de l'engrenage en mètre
```

```
s=4096/6  # le rapport 4096 (vitesse maximale du moteur) divisée par la
tension maximale du moteur
d=2*math.pi*r/341.2  # distance parcourue entre deux impulsions de
l'encodeur si r=4.5 cm alors précision de 0.8 mm

 #fonction d'initialisation, pour qu'au départ le moteur se retrouve en
butée
def ini():
    global a_count
    global dist
    global b_count
    m.throttle(-3500) #on fait tourner le moteur en sens inverses
jusqu'en buté
    b0=b_count        # on lit la valeur de l'impulsion à l'instant
présent
    time.sleep(0.5)
    b=b_count         # on lit la valeur de l'impulsion 0.5 secondes
après
    while b0 != b:      # tant que les valeurs d'impulsions ne sont pas
identiques on n'est pas en butée
        b0=b_count
        time.sleep(0.5)
        b=b_count
    m.brake()
    a_count,b_count,dist=0,0,0   # on initialise les compteurs
d'impulsions et la variable dist qui représente la position de la tige

##fonction qui donne le signe du nombre (utile pour l'asservissement)
def signe(a):
    if a<0:
        return False
    else:
        return True

#Méthode bourrine, le moteur tourne à fond jusqu'à ce que l'on atteigne
la position souhaitée
def bourrin (x):
    global a_count
    global d
    global dist
    z=0
    eps=x-a_count*d
    while abs(eps)>0.001:
        if signe(eps):
            m.throttle(4096)
        else:
            z=a_count
            a_count=z-(a_count-z)
        dist=a_count
```

```
        eps=x-a_count*d

    m.brake()
#Methode bourrine 2 qui permet de régler le problème de la 1ere
def bourrin2 (x):
    global a_count
    global d
    global dist
    dist+=x
    a_count=0
    eps=x
    while abs(eps)>0.001:
        if x>=0:
            m.throttle(4096)
            eps=x-a_count*d
        else:
            m.throttle(-4096)
            eps=x+a_count*d
    m.brake()









#Fonction PID

def deplacer(x):     #x correpond à la distance à parcourir en mètre
    global a_count
    global I
    global Kc
    global Td
    global r
    global s
    global d
    maxSomme=1000
    z=0                 # z est utile lorsque l'on se déplace vers
l'arrière
    micros = pyb.Timer(2, prescaler=83, period=0x3fffffff)  # on créé un
timer en microseconde (pour le dérivateur)
    micros.counter(0)   # on l'initialise à 0
    eps0=0              #erreur0 en metre
    t0=micros.counter()
    eps1=(x-a_count*d)   # erreur 1
```

```
    print(eps1)
    t1=micros.counter()     # t1- t0 correspond à la durée séparant 2
mesures d'erreur (c'est la période d'échantillonage)
    if eps1*(t1-t0)>maxSomme:
        somme=maxSomme
    else:
        somme=eps1*(t1-t0)
    while abs(eps1)>0.0008:
        Vcons=Kc*(eps1+I*somme+Td*(eps1-eps0)/(t1-t0))
        print("V =", Vcons)     # on génère la consigne en fonction de
l'erreur de son intégrale et de sa dérivée
        if abs(Vcons)>6:    #si consigne en tension trop élevée alors on
sature la consigne pour ne pas abîmer le moteur
            if signe(Vcons):     #(le moteur tourne alors à sa vitesse
max)
                m.throttle(4096)
                eps0=eps1
                t0=micros.counter()
                eps1=(x-a_count*d)
                print(eps1)
                t1=micros.counter()
                if eps1*(t1-t0)>maxSomme:
                    somme=maxSomme
                else:
                    somme=eps1*(t1-t0)
            else:
                z=a_count
                m.throttle(-4096)
                eps0=eps1
                t0=micros.counter()
                a_count=z-(a_count-z)
                eps1=(x-a_count*d)
                print(eps1)
                t1=micros.counter()
                if eps1*(t1-t0)>maxSomme:
                    somme=maxSomme
                else:
                    somme=eps1*(t1-t0)
        else:
            z=a_count
            m.throttle(int(Vcons*s))
            if not signe(Vcons):
                eps0=eps1
                t0=micros.counter()
                a_count=z-(a_count-z)
                eps1=(x-a_count*d)
                print(eps1)
                t1=micros.counter()
                if eps1*(t1-t0)>maxSomme:
```

```
                    somme=maxSomme
                else:
                    somme=eps1*(t1-t0)

            else:
                eps0=eps1
                t0=micros.counter()
                eps1=(x-a_count*d)
                print(eps1)
                t1=micros.counter()
                if eps1*(t1-t0)>maxSomme:
                    somme=maxSomme
                else:
                    somme=eps1*(t1-t0)
    m.brake()
    print(eps1)
    print(a_count*d)


#Liste reliant les notes et leurs positions sur la corde

Liste_note=[['do',0.17],['dobis',0.197],['re',0.23],['rebis',0.255],['mi'
,0.277],['fa',0.3],['fabis',0.318],['sol',0.01],['solbis',0.04],['la',0.0
85],['labis',0.115],['si',0.145]]

#Fonction qui permet de donner directement la note à jouer
def jouer_note(note):
    global angle
    global dist
    k=0
    while note!=Liste_note[k][0]:
        k+=1
    bourrin2(Liste_note[k][1]-dist)
    angle=-angle
    ser.angle(angle)
```

D. Código de la interfaz gráfica

a. Clase main:

```java
package Logiciel;
import javax.swing.*;

public class Main {
     public static int a=0;
     public static PybSerialComm pyb;


    public static void main(String[] args) throws PybSerialComm.PyboardException {
        // Get a list of available pyboards
        List<PybSerialComm> pyboards = PybSerialComm.findPyboards();
        if (pyboards.size() > 0) {// Use the first one
            pyb = pyboards.get(0);

            // Open the serial port to the REPL
            pyb.open();
            // Switch to the raw REPL mode
            if (pyb.enter_raw_repl()) {
                a=1;
                String ini="ass.ini()";
                Main.pyb.kill();
                try{
                    Main.pyb.startCommand(ini);
                }
                catch(PybSerialComm.PyboardException ev) {
                    System.exit(1);
                }
        }
            }
            JFrame cadre = new javax.swing.JFrame("PlayZeGuitar");
            Interface panneau = new Interface();
            cadre.setContentPane(panneau);
            cadre.setLocation(400, 300);
            cadre.pack();
            cadre.setResizable(false);
            cadre.setVisible(true);
            cadre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            }
}
```

b. Clase interface:

```
1  package Logiciel;
2  import javax.swing.*;
11
12
13
14 public class Interface extends JLayeredPane {
15     static final long serialVersionUID = 1;
16     public static Font f =new Font ("Arial",Font.BOLD,8);
17     public static Font t= new Font("Arial",Font.ITALIC,20);
18     public static Font y= new Font("Arial",Font.ITALIC,15);
19     String nom_fichier_image = "guitare.jpg";
20     public static JButton Do;
21     public static JButton Re;
22     public static JButton Mi;
23     public static JButton Fa;
24     public static JButton Sol;
25     public static JButton La;
26     public static JButton Si;
27     public static JButton Dobis;
28     public static JButton Rebis;
29     public static JButton Fabis;
30     public static JButton Solbis;
31     public static JButton Labis;
32     public static JComboBox<String> menu;
33     public static JTextField BPM;
34     public static JTextArea Note;
35     public static JButton GO;
36     public static JButton OK;
37
38
39
40     public Interface() {
41     setBackground(Color.GRAY);
42     setPreferredSize(new Dimension (900,470));
43
44
45
46
47 }
```

```
53 public void dessiner_Piano(Graphics g) {
54     Do = new JButton("Do");
55     Do.setBounds(63, 390, 45, 90);
56     Do.setFont(f);
57     Do.setBackground(Color.WHITE);
58     Do.setForeground(Color.BLACK);
59     Do.setMnemonic('A');
60     add(Do,DEFAULT_LAYER);
61     Re = new JButton("Re");
62     Re.setBounds(108, 390, 45, 90);
63     Re.setFont(f);
64     Re.setBackground(Color.WHITE);
65     Re.setForeground(Color.BLACK);
66     Re.setMnemonic('Z');
67     add(Re,DEFAULT_LAYER);
68     Mi = new JButton("Mi");
69     Mi.setBounds(153, 390, 45, 90);
70     Mi.setFont(f);
71     Mi.setBackground(Color.WHITE);
72     Mi.setForeground(Color.BLACK);
73     Mi.setMnemonic('E');
74     add(Mi,DEFAULT_LAYER);
75     Fa = new JButton("Fa");
76     Fa.setBounds(198, 390, 45, 90);
77     Fa.setFont(f);
78     Fa.setBackground(Color.WHITE);
79     Fa.setForeground(Color.BLACK);
80     Fa.setMnemonic('R');
81     add(Fa,DEFAULT_LAYER);
82     Sol = new JButton("Sol");
83     Sol.setBounds(243, 390, 45, 90);
84     Sol.setFont(f);
85     Sol.setBackground(Color.WHITE);
86     Sol.setForeground(Color.BLACK);
87     Sol.setMnemonic('T');
88     add(Sol,DEFAULT_LAYER);
89     La = new JButton("La");
90     La.setBounds(288, 390, 45, 90);
```

```
    La.setBounds(288, 390, 45, 90);
    La.setFont(f);
    La.setBackground(Color.WHITE);
    La.setForeground(Color.BLACK);
    La.setMnemonic('Y');
    add(La,DEFAULT_LAYER);
    Si = new JButton("Si");
    Si.setBounds(333, 390, 45, 90);
    Si.setFont(f);
    Si.setBackground(Color.WHITE);
    Si.setForeground(Color.BLACK);
    Si.setMnemonic('U');
    add(Si,DEFAULT_LAYER);
    Dobis=new JButton();
    Dobis.setBounds(93, 390, 30, 60);
    Dobis.setFont(f);
    Dobis.setBackground(Color.BLACK);
    Dobis.setForeground(Color.WHITE);
    Dobis.setMnemonic(KeyEvent.VK_2);
    add (Dobis,POPUP_LAYER);
    Rebis=new JButton();
    Rebis.setBounds(138, 390, 30, 60);
    Rebis.setFont(f);
    Rebis.setBackground(Color.BLACK);
    Rebis.setForeground(Color.WHITE);
    Rebis.setMnemonic(KeyEvent.VK_3);
    add (Rebis,POPUP_LAYER);
    Fabis=new JButton();
    Fabis.setBounds(228, 390, 30, 60);
    Fabis.setFont(f);
    Fabis.setBackground(Color.BLACK);
    Fabis.setForeground(Color.WHITE);
    Fabis.setMnemonic(KeyEvent.VK_5);
    add (Fabis,POPUP_LAYER);
    Solbis=new JButton();
    Solbis.setBounds(273, 390, 30, 60);
    Solbis.setFont(f);

    Solbis.setBackground(Color.BLACK);
    Solbis.setForeground(Color.WHITE);
    Solbis.setMnemonic(KeyEvent.VK_6);
    add (Solbis,POPUP_LAYER);
    Labis=new JButton();
    Labis.setBounds(318, 390, 30, 60);
    Labis.setFont(f);
    Labis.setBackground(Color.BLACK);
    Labis.setForeground(Color.WHITE);
    Labis.setMnemonic(KeyEvent.VK_7);
    add (Labis,POPUP_LAYER);
    Do.setEnabled(false);
    Dobis.setEnabled(false);
    Re.setEnabled(false);
    Rebis.setEnabled(false);
    Mi.setEnabled(false);
    Fa.setEnabled(false);
    Fabis.setEnabled(false);
    Sol.setEnabled(false);
    Solbis.setEnabled(false);
    La.setEnabled(false);
    Labis.setEnabled(false);
    Si.setEnabled(false);
```

```
153 }
154
155 public void dessiner_OK( Graphics g) {
156     OK=new JButton();
157     OK.setBounds(880, 450, 20, 20);
158     if(Main.a==0) {
159         OK.setBackground(Color.RED);
160         }
161     else {
162         OK.setBackground(Color.GREEN);
163     }
164
165     OK.setEnabled(false);
166     add(OK);
167
168
169
170 }
171
172
173 public void dessiner_Menu(Graphics g) {
174     menu = new JComboBox<String>();
175     menu.setBounds(20, 100, 100, 20);
176     menu.setBackground(Color.WHITE);
177     menu.setForeground(Color.BLACK);
178     menu.addItem("Choisir");
179     menu.addItem("Piano");
180     menu.addItem("Tableau");
181     add(menu);
182
183 }
184 public void dessiner_Tableau(Graphics g) {
185     BPM=new JTextField();
186     BPM.setBounds(10, 175, 100,20);
187     add(BPM);
188     Note=new JTextArea();
189     Note.setBounds(500,10,400,300);
190     Note.setBackground(Color.DARK_GRAY);
191     Note.setForeground(Color.WHITE);
192     add(Note);
193     GO=new JButton("GO");
194     GO.setFont(f);
195     GO.setBounds(600,350,200,100);
196     GO.setBackground(Color.GRAY);
197     GO.setForeground(Color.BLACK);
198     add(GO);
199     GO.setEnabled(false);
200
201
202
203
204
205 }
206
207 public void dessiner_Texte(Graphics g) {
208     JLabel Accueil=new JLabel("<html>Bienvenue sur PlayZeGuitare!!<br><br>Veuillez choisir votre mode de fonc
209     Accueil.setBounds(10, -15, 500, 100);
210     Accueil.setFont(t);
211     Accueil.setForeground(Color.WHITE);
212     add(Accueil);
213     JLabel Tableau1=new JLabel("Pour utiliser le Tableau veuillez indiquer un BPM:");
214     Tableau1.setBounds(10,100,500,100);
215     Tableau1.setFont(t);
216     Tableau1.setForeground(Color.WHITE);
217     add(Tableau1);
218     JLabel Tableau2=new JLabel("<html>Dans le tableau, veuillez écrire les notes:<br>La syntaxe est note:note
219     Tableau2.setBounds(10,200,500,100);
220     Tableau2.setFont(t);
221     Tableau2.setForeground(Color.WHITE);
222     add(Tableau2);
```

```
}


    public void paintComponent(Graphics g) {
    super.paintComponent(g);
    dessiner_OK(g);
    dessiner_Piano(g);
    dessiner_Menu(g);
    dessiner_Tableau(g);
    dessiner_Texte(g);
    ImageIcon m = new ImageIcon(nom_fichier_image);
    Image monImage = m.getImage();
    g.drawImage(monImage, 0, 0,this);
    (new Menu()).start();
    (new Piano()).start();
    (new Tableau()).start();


}
```

c. Clase ActDo:

```
package Logiciel;

import java.awt.event.*;




public class ActDo implements ActionListener  {



    public void actionPerformed (ActionEvent e)  {
        //System.out.println("On joue la note Do");
        String cmd="ass.jouer_note('do')";
        Main.pyb.kill();
        try{
            Main.pyb.startCommand(cmd);
        }
        catch(PybSerialComm.PyboardException ev) {
            System.exit(1);
        }
    }
}

```

Como todas las clases de las distintas notas son iguales, sólo se mostrará esta.

d. Clase ActGo:

```
1  package Logiciel;
2  import java.awt.event.ActionEvent;
6
7
8  public class ActGo implements ActionListener {
9      private JTextField BPM;
10     private JTextArea Note;
11
12     public ActGo(JTextField BPM,JTextArea Note) {
13         this.BPM=BPM;
14         this.Note=Note;
15
16     }
17
18     public void actionPerformed (ActionEvent e) {
19         //System.out.println("GO");
20         Integer Rythme=Integer.parseInt(BPM.getText());
21         //System.out.println(Rythme);
22         String Part =Note.getText();
23         String[] Partition = Part.split(":");
24         Timer timer=new Timer();
25         Tim task=new Tim(Partition);
26         timer.schedule(task, 0, (60/Rythme)*1000);
27
28
29
30     }
31 }
```

e. Clase ActionBox:

```
1  package Logiciel;
2  import java.awt.event.*;
4
5
6  public class ActionBox implements ActionListener {
7
8      private JButton Do;
9      private JButton Dobis;
10     private JButton Re;
11     private JButton Rebis;
12     private JButton Mi;
13     private JButton Fa;
14     private JButton Fabis;
15     private JButton Sol;
16     private JButton Solbis;
17     private JButton La;
18     private JButton Labis;
19     private JButton Si;
20     private JButton GO;
21
22     public  ActionBox(JButton Do,JButton Dobis,JButton Re,JButton Rebis,JButton Mi,JButton Fa,JButton Fabis,J
23         this.Do=Do;
24         this.Dobis=Dobis;
25         this.Re=Re;
26         this.Rebis=Rebis;
27         this.Mi=Mi;
28         this.Fa=Fa;
29         this.Fabis=Fabis;
30         this.Sol=Sol;
31         this.Solbis=Solbis;
32         this.La=La;
33         this.Labis=Labis;
34         this.Si=Si;
35         this.GO=GO;
36
37
38
39     }
```

```
    public void actionPerformed(ActionEvent e) {
        JComboBox<String> menu= (JComboBox<String>) e.getSource();
        String selected=(String) menu.getSelectedItem();
        if(Main.a==1) {
            if(selected.equals("Piano")) {
            //System.out.println("Vous avez choisi le Piano");
            Do.setEnabled(true);
            Re.setEnabled(true);
            Mi.setEnabled(true);
            Fa.setEnabled(true);
            Sol.setEnabled(true);
            La.setEnabled(true);
            Si.setEnabled(true);
            Dobis.setEnabled(true);
            Rebis.setEnabled(true);
            Fabis.setEnabled(true);
            Solbis.setEnabled(true);
            Labis.setEnabled(true);
            GO.setEnabled(false);
            menu.setEnabled(false);
            menu.setEnabled(true);

        }
        if(selected.equals("Tableau")) {
            //System.out.println("Vous avez choisi le Tableau");
            Do.setEnabled(false);
            Re.setEnabled(false);
            Mi.setEnabled(false);
            Fa.setEnabled(false);
            Sol.setEnabled(false);
            La.setEnabled(false);
            Si.setEnabled(false);
            Dobis.setEnabled(false);
            Rebis.setEnabled(false);
            Fabis.setEnabled(false);
            Solbis.setEnabled(false);
            Labis.setEnabled(false);
            GO.setEnabled(true);
            menu.setEnabled(false);
            menu.setEnabled(true);

        }
        if(selected.equals("Choisir")) {
            //System.out.println("Il faut choisir");
            Do.setEnabled(false);
            Re.setEnabled(false);
            Mi.setEnabled(false);
            Fa.setEnabled(false);
            Sol.setEnabled(false);
            La.setEnabled(false);
            Si.setEnabled(false);
            Dobis.setEnabled(false);
            Rebis.setEnabled(false);
            Fabis.setEnabled(false);
            Solbis.setEnabled(false);
            Labis.setEnabled(false);
            GO.setEnabled(false);
            menu.setEnabled(false);
            menu.setEnabled(true);
        }

        }


    }

}
```

f. Clase Menu:

```
package Logiciel;



public class Menu extends Thread {

    public void run() {
        Interface.menu.addActionListener(new ActionBox(Interface.Do,Interface.Dobis,Interface.Re,
                Interface.Rebis,Interface.Mi,Interface.Fa,Interface.Fabis,Interface.Sol,
                Interface.Solbis,Interface.La,Interface.Labis,Interface.Si,Interface.GO));

    }

}
```

g. Clase Piano:

```
package Logiciel;



public class Piano extends Thread {
    public void run() {
        Interface.Do.addActionListener(new ActDo());
        Interface.Dobis.addActionListener(new ActDobis());
        Interface.Re.addActionListener(new ActRe());
        Interface.Rebis.addActionListener(new ActRebis());
        Interface.Mi.addActionListener(new ActMi());
        Interface.Fa.addActionListener(new ActFa());
        Interface.Fabis.addActionListener(new ActFabis());
        Interface.Sol.addActionListener(new ActSol());
        Interface.Solbis.addActionListener(new ActSolbis());
        Interface.La.addActionListener(new ActLa());
        Interface.Labis.addActionListener(new ActLabis());
        Interface.Si.addActionListener(new ActSi());

    }

}
```

h. Clase Tableau (tablero):

```
package Logiciel;

public class Tableau extends Thread {

    public void run() {
        Interface.GO.addActionListener(new ActGo(Interface.BPM,Interface.Note));
    }

}
```

i. Clase Tim:

```
package Logiciel;

import java.util.TimerTask;

public class Tim extends TimerTask {
     private  int i=0;
     private  String[] Partition;

     public Tim(String[] Partition) {
         this.Partition=Partition;


     }
     public void run() {
                //System.out.println(Partition[i]);
                String cmd="ass.jouer_note('"+ Partition[i]+"')";
                Main.pyb.kill();
                try{
                    Main.pyb.startCommand(cmd);
                }
                catch(PybSerialComm.PyboardException ev) {
                    System.exit(1);
                }
                i+=1;
                if (i==Partition.length) {
                cancel();
                }



         }

}
```