Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

Grado en Ingeniería Mecánica

Diseño y prototipado de un dispositivo low-cost para la identificación de un sistema de 1GDL embarcado en una estructura.

Autor:

Cruz Carrillo, Moisés

Tutores:

Lorenzana Iban, Antolín

Magdaleno González, Álvaro

C.A.,I.T.,M.M.C y Teoría de Estructuras

Valladolid, Febrero 2020.

RESUMEN

El presente trabajo de fin de grado tiene como objetivo el realizar un prototipo de sensorización, adquisición y tratamiento de datos de bajo coste con el que poder sustituir un equipo profesional para identificar dos propiedades estructurales. Concretamente el equipo profesional será SIRIUS HD-STG, el prototipo se realizará con el microcontrolador Arduino y un dispositivo bluetooth conectado a un teléfono móvil. Las propiedades que se intentarán estimar en el prototipo serán la frecuencia propia y el amortiguamiento.

Palabras clave : frecuencia propia, amortiguamiento, Arduino, SIRIUS, SDOF.

ABSTRACT

The aim of this final degree project is to make a low-cost prototype of sensorization, acquisition and data processing to replace professional equipment in order to identify two structural properties. The specific equipment to be replaced will be SIRIUS HD-STG. The prototype will usa an Arduino microcontroller and a bluetooth device connected to a mobile phone. The properties that will be tried to reproduce in the prototype will be the natural frequency and the damping.

Keywords : natural frequency, damping, Arduino, SIRIUS,SDOF.

INDICE GENERAL

INDICE DE FIGURAS

INDICE DE TABLAS

Capítulo1.Introducción. Motivación. Aplicación a la ingeniería.

1.1. Introducción

El presente TFG responde a una doble intención, la primera es finalizar los estudios en el grado de Ingeniería Mecánica por la Universidad de Valladolid y por otro introducirse en el campo de la dinámica de estructuras.

Desde que apareció la primera estructura de la humanidad, esta ha ido evolucionando en complejidad, tanto que desde una sencilla choza hecha de barro se ha conseguido pasar a complejos rascacielos hechos de acero y hormigón armado.

Con estos avances, fueron aumentando los conocimientos acerca del cálculo estructural y su dificultad.

En la era actual la ingeniería está en un punto en el que el cálculo estructural no solamente se centra en la estática de las estructuras; sino que se le da importancia a la dinámica también.

La parte de la dinámica estructural se fundamenta como bien se intuye en la acción de movimientos de la estructura.

Una aproximación efectiva de una estructura en un comportamiento dinámico sería usar el sistema SDOF (“*Single Degree of Freedom*”), en castellano 1 grado de libertad (1gdl). Es interesante conocer las propiedades mck (masa, amortiguamiento y rigidez) de un sistema de 1 gdl embarcado en una estructura; para ello es necesario hacer estudios.

Normalmente para efectuar estudios experimentales sobre la dinámica de estructuras se necesitan herramientas modernas, tales como equipos de adquisición y procesamiento de datos como el siguiente:

Figura 1.1. SIRIUS HD-STG.

Estos equipos normalmente son de alta fiabilidad, pero responden a dos inconvenientes principales. El primero es que son costosos y el segundo es que son complejos.

Para responder a las necesidades de dichos equipos, el estudio lo realizará una persona con conocimientos en dicho campo. Que no abundan en la actualidad.

Con todos estos requerimientos (personal cualificado, material profesional, costes) surge la necesidad de optimizar los recursos usados.

Aquí es donde se centrará el presente documento, en desarrollar un dispositivo *low-cost* que pueda sustituir fiablemente a un equipo costoso y que además pueda ser manejado por una persona con unos conocimientos de menor nivel que un experto de la dinámica de estructuras. Para que el técnico que use este dispositivo *low-cost* sea capaz de interpretar los ensayos experimentales, también se incluirá en el dispositivo un módulo bluetooth que se conectará a su teléfono móvil.

Para ello se usará algo tan común y accesible a todo el público como lo es un Arduino.

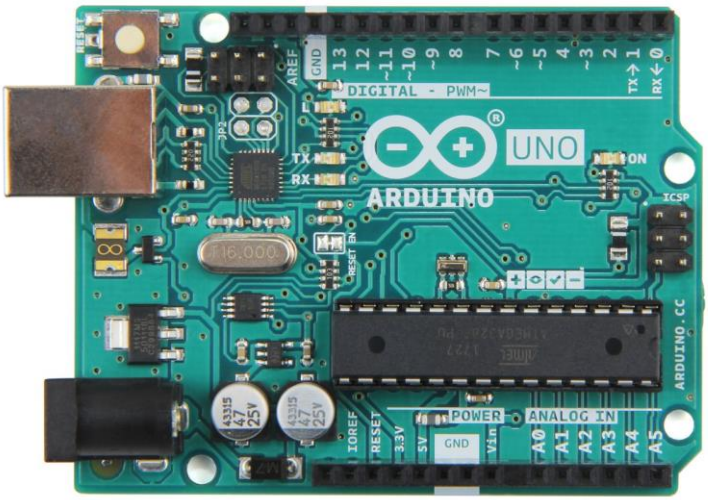


Figura 1.2. Microcontrolador Arduino, modelo UNO.

Por lo tanto, en este documento se recogerá toda la información necesaria para poder reproducir el funcionamiento del dispositivo profesional SIRIUS con un Arduino.

1.2. Objetivos

Los principales objetivos serán por tanto los recogidos a continuación:

- En primer lugar, ensayar una estructura con una tarjeta de adquisición de datos profesional (SIRIUS HD-STG) y con un prototipo de bajo coste (*low cost*).

- Conseguir de manera fiable extraer la propiedad de la frecuencia propia, evaluando la estructura como un sistema de 1 gdl-mck.

- Asimismo, evaluar la fiabilidad en el mismo sistema del coeficiente de amortiguamiento.

- Adquirir competencias adicionales a la titulación, tales como conocimientos electrónicos y de programación en C.

1.3. Metodología.

El presente trabajo de fin de grado está dividido en 5 capítulos:

- En este primer capítulo se recoge la introducción, los objetivos buscados y la metodología a seguir.

- El segundo capítulo será una breve introducción teórica en el estudio de los sistemas de 1 gdl.

- En el tercer capítulo se recogerá el montaje del prototipo de adquisición de datos que se ha diseñado, además de la programación del mismo.

- En cuarto capítulo se realiza la fase experimental en el que se intentará validar el prototipo.

- En el quinto capítulo se extraerán las conclusiones oportunas y se darán las aperturas hacia líneas futuras.

Para completar el trabajo se añadirán tres anexos en los que se explican los comandos básicos para manejar el software IDE de Arduino, el modo de calibración de los acelerómetros y por último un tercero en el que se incluirá el código del programa. Todos ellos con sus correspondientes comentarios para facilitar su entendimiento.

Capítulo 2. Teoría SDOF. Ecuación de movimiento, FRF y método de la transmisibilidad.

2.1. Introducción.

Una estructura no deja de ser un sistema mecánico y como tal tiene a una serie de propiedades características, tales como la masa, el amortiguamiento y la frecuencia propia.

Dicho sistema se puede ver sometidos a fuerzas, tanto internas como externas, produciendo así diferentes respuestas en el mismo. Si dichas fuerzas producen un movimiento relativo entre masas y además si este se repite en el tiempo se puede estar hablando de vibraciones.

Dentro del campo de las vibraciones, a la hora de hacer un análisis estructural este está delimitado por el número de elementos que posee la misma. Siendo objeto del análisis un número finito de grados de libertad determinado por un número finito de elementos.

Dentro de un estudio de este tipo, en el documento se estudiarán los sistemas de un grado de libertad, ya que estos son los más sencillos. Además de que muchas de las propiedades (que se presentan en los sistemas de 1 gdl) aparecen en otros sistemas más complejos.

2.2. Fundamentos mecánicos.

Primero a partir de un sistema de 1 grado de libertad con amortiguamiento como se observa en la figura 2.1:

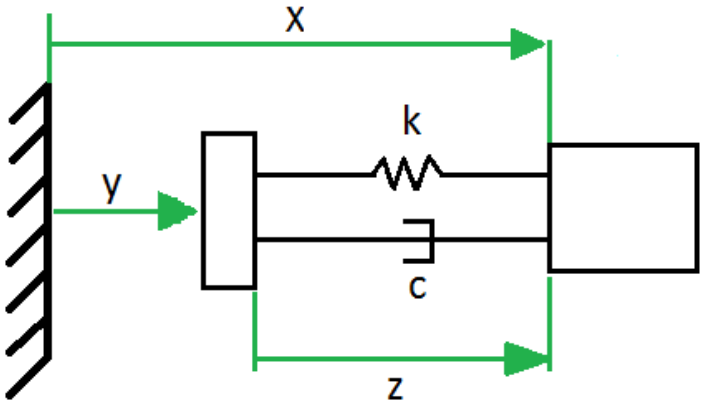


Figura 2.1. Sistema de 1 gdl [1].

El sistema de la *figura 1* está compuesto por una masa móvil **m**, a la cual se le asocia la coordenada **x**. Además de una base móvil cuya coordenada asociada

es **y**. Dos elementos más que son, un muelle con constante **k** y un amortiguador viscoso de constante **c**.

$$m\ddot{x} + 2\xi w_n \dot{x} + w_n^2 x = 2\xi w_n y + \dot{w}_n^2 y \quad (1)$$

Dicha ecuación [2] proviene de un sumatorio de fuerzas aplicado al sistema.

No se usarán directamente las coordenadas **x** e **y**, si no que se ha introducido una coordenada relativa la cual es la **z**.

$$z(t) = x - y \quad (2)$$

Para relacionar **k** y **c** en con (1) se tienen las siguientes ecuaciones:

$$w_n^2 = \frac{k}{m} \quad (3)$$

$$2\xi w_n = \frac{c}{m} \quad (4)$$

Donde $\boldsymbol{w_n}$ es la frecuencia natural del sistema y $\boldsymbol{\xi}$ es el factor de amortiguamiento del mismo.

Insertando (2), (3) y (4) en la ecuación (1) queda:

$$m\ddot{x} + c(\dot{x} - \dot{y}) + k(x - y) = 0 \quad (5)$$

Por lo que las masa ***m*** se puede eliminar de la ecuación (4) y se obtiene finalmente:

$$\ddot{z} + 2\xi w_n \dot{z} + w_n^2 z = -\ddot{y} \quad (6)$$

Pero si bien esta ecuación está en el dominio del tiempo, para procesar la información en tiempo real se necesitan procesos de cálculos mucho más potentes y ese no es el objetivo de este TFG, por lo que se optará al cálculo en el dominio de la frecuencia.

Para ello se necesita el uso de la transformada de Laplace [3],para la que se aplicará en la ecuación (6) :

$$s^2 z + 2\xi w_n s z + w_n^2 z = -s^2 y \quad (7)$$

Agrupando términos:

$$(s^2 + 2\xi w_n s + w_n^2) z = -s^2 y \quad (8)$$

Si se divide ***z*** entre *y* se obtiene lo siguiente.

$$\frac{z(s)}{y(s)} = \frac{-s^2}{s^2 + 2\xi w_n s + w_n^2} \quad (9)$$

Recapitulando en la ecuación (9), ya se tienen relacionadas las coordenadas relativas, que corresponden al método de la transmisibilidad. [2]

Donde **s** es la frecuencia angular compleja y tiene la siguiente forma:

$$s = jw \quad (10)$$

$$s^2 = -w^2 \quad (11)$$

Al introducir (10) y (11) en (9) deja la siguiente ecuación:

$$\frac{z(w)}{y(w)} = \frac{w^2}{w^2 + 2\xi w_n j w + w_n^2} \tag{12}$$

Esta ecuación se llama función de transferencia que tiene parte real e imaginaria. Por lo que se agruparán sus términos correspondientes a cada parte además de renombrar la ecuación como ***H(w)***:

$$H(w) = \frac{w^2}{(w_n^2 - w^2) + j2\xi w_n w} \tag{13}$$

Dicha ecuación se llama función de respuesta en frecuencia (FRF) [4] y será el centro del TFG, ya que a partir de la misma se obtiene una respuesta del sistema en el dominio de la frecuencia. De (13) se puede extrapolar tanto el coeficiente de amortiguamiento como la frecuencia propia del sistema.

Para extraer la frecuencia propia a partir de una FRF se necesitará identificar el máximo de dicha función.

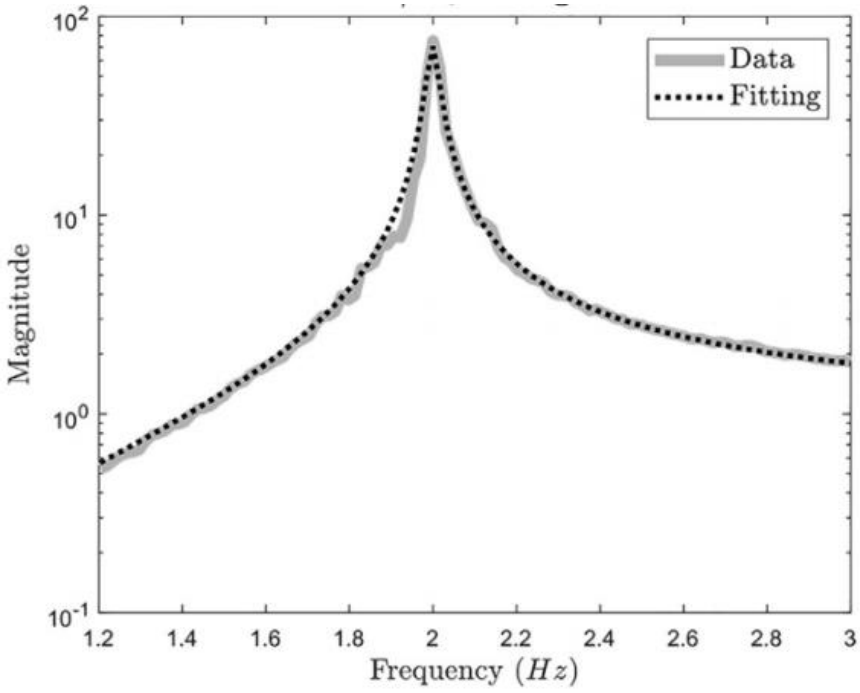


Figura 2.2.FRF extraída del documento [2].

En la figura 2.2 se puede ver una FRF que tiene un máximo en forma de pico en la frecuencia 2 Hz. El eje horizontal se representa la frecuencia y en el eje vertical la transmisibilidad. La resolución en el eje horizontal de la gráfica se extrae de la siguiente ecuación:

$$\Delta f = \frac{w_s}{N} \tag{14}$$

Siendo w_s la frecuencia de muestreo y N el número de muestras (mediciones) recogidas por los acelerómetros.

Entonces para identificar la frecuencia propia (w_n) del sistema, que ese pico representado en la figura 2.2 se halla de la ecuación siguiente:

$$w_n = p * \Delta f \tag{15}$$

Con ***p*** como la posición numérica en todo el vector de registro. Es decir, si el máximo se registra en la primera muestra de un total de 1000. ***p*** será igual a la unidad.

Por último, para hallar el coeficiente de amortiguamiento (ξ) partiendo de la ecuación (13) si la frecuencia ***w*** coincide con la frecuencia propia w_n queda de la siguiente manera:

$$H(w) = \frac{{w_n}^2}{(w_n^2 - {w_n}^2) + j2\xi w_n w_n} \tag{16}$$

Donde ***H(w)*** será la magnitud de dicho máximo, por lo que al final para determinar ξ, despejando de (16):

$$\xi = \frac{1}{2 * H(w)} \tag{17}$$

Ahora bien, para hacer estos cálculos de manera computacional se tiene que seguir un camino diferente. Por lo que se van a describir las operaciones matemáticas que ejecutará el código programado.

2.3. Fundamentos electrónicos.

2.3.1. Filtrado de señales.

En el siguiente apartado se va a comentar la importancia para el trabajo de un filtro de señales digitales.

Normalmente en una señal se producen interferencias debido a múltiples factores externos. Determinar el origen de estas interferencia o también llamado ruido es muy difícil.

La necesidad de reducir el ruido en trabajos de tipo electrónico es necesario en la mayoría de las veces si se trabaja con muestreos múltiples. Una de las soluciones más requeridas pasa por tomar varias muestras y combinarlas matemáticamente.

Lo primero será comentar la razón de elegir este tipo de filtrado y no uno físico. Esto viene dado porque un filtro físico (por ejemplo con la implementación de condensadores) elimina una serie de muestras ya sea por encima o por debajo de la medida de corte marcada por el propio condensador. Pero en este caso no se desea eliminar ningún tipo de muestra por que se podría perder alguna significativa. Lo que se hará será implementar un filtro digital.

Aquí es donde entran los filtros digitales. Que en definitiva son algoritmos matemáticos que permiten obtener unos valores de mayor significación de la muestra. Que los registrados directamente de la medición.

Hay una gran variedad de tipos de filtro digital, el que se va a usar en este trabajo es el denominado filtro de media móvil. Es un filtro bastante sencillo de implementar y calcular.

El filtro de media móvil se basa en la siguiente ecuación:

$$\bar{\chi} = \frac{\sum_{i=1}^{n} x_i}{n} \tag{18}$$

El filtro de media móvil toma los últimos "N" valores recibidos, a los que se llamarán ventana y se calcula su media aritmética. El resultado de esto es una señal más suavizada, eliminando así parte del ruido de la señal.

Como se puede observar este filtro es muy sencillo de implementar, puesto que para la operación matemática de la media aritmética sólo se suman los valores de "N" muestras y se dividen entre "N".

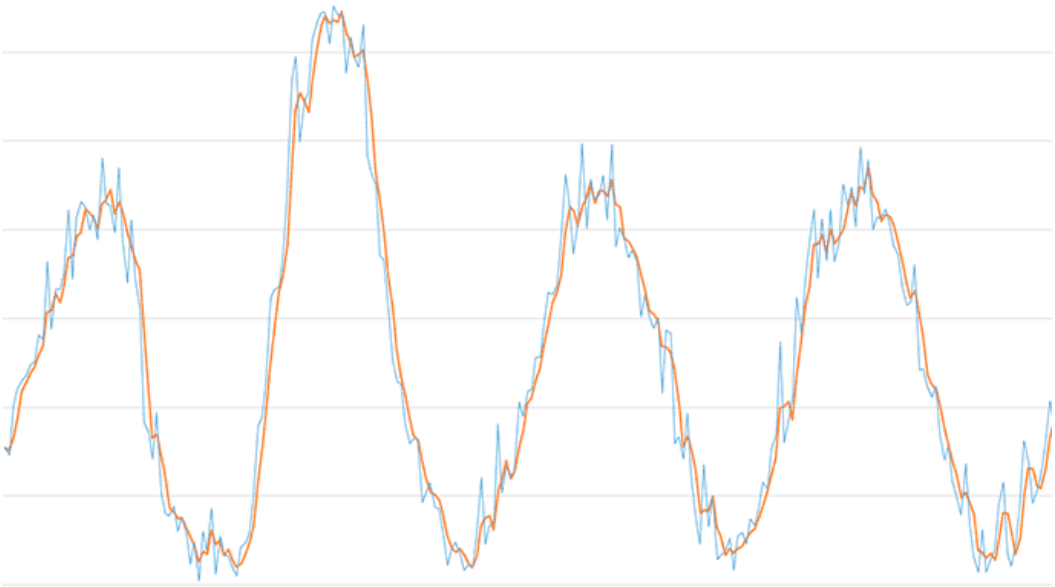


Figura 2.3. Aplicación de un filtro de media móvil [5].

En la figura 2.3 se puede observar en azul una serie de datos que simulan una señal senoidal con su correspondiente ruido. Y en naranja se puede observar el filtro de media móvil ya aplicado. Es bastante notorio percibir que la señal no recibe un gran cambio. Pero si el suficiente para suavizarla y conseguir eliminar el ruido. El tamaño de la ventana de dicho filtro en este caso es de 3.

El tamaño de la ventana es importante. Puesto que a medida que se amplía este rango el filtro será más fuerte y por tanto suavizará más la señal. Esto a veces no puede ser tan interesante puesto que se puede suavizar tanto la señal que al fin y al cabo se pierde la representatividad que se anda buscando [5].

Además, los filtros digitales tienen un pequeño inconveniente que es el desfase con respecto a la señal original, puesto que el primer valor que sale del filtrado depende de "N" valores de la señal original. En este trabajo se encontró una solución particular para este tipo de inconvenientes, que consistió en registrar un número de muestras adicionales del tamaño de la ventana.

2.3.2. Análisis FRF. Transformada rápida de Fourier.

Llegar a la ecuación (13) de manera computacional es algo más complicado, ya que tenemos que registrar señales temporales y traducirlas al dominio de la frecuencia. Para ello usaremos una herramienta ya conocida y habitualmente usada, la transformada de Fourier.

Esta transformada de Fourier es una función matemática que transforma una señal en el dominio del tiempo a una en el dominio de la frecuencia.

Una transformada de Fourier discreta (DFT), es la operación matemática usada cuando se aplica a una señal digital (ya que esta es discreta). Esta transformada sin embargo tiene un inconveniente, su computación. Ya que la DFT para el realizarla el número de operaciones que se realiza son N^2 (siendo N el número de muestras). Al tomar valores considerables de muestreo, el número de operaciones se dispara, lo que es perjudicial para el microcontrolador. Porque este está bastante más limitado en memoria computacional que un ordenador.

Para ello existe otra solución la cual es igual de potente, pero ahorra memoria computacional, se habla de la transformada rápida de Fourier (FFT), la cual en vez de tomar N^2 operaciones, usa $N \cdot \log_2(N)$ operaciones. Como se puede ver se ahorra significativamente más memoria computacional.

2.3.2.1. Transformada rápida de Fourier.

En este apartado se hablará de las características más importantes de esta herramienta en el TFG realizado.

La característica más importante de la FFT, es la necesidad de que el número de muestras registradas sea una potencia de 2. Por lo que a la hora de programar se usaran potencias de 2 para el registro de la señal.

Otra característica importante es que al finalizar el cálculo obtenemos una función simétrica. Por lo que a la hora de extrapolar las propiedades de estudio de dicha señal, solamente se necesita la mitad de la misma.

2.3.3. Lenguaje de comunicación.

La última parte del proceso de análisis y toma de datos es la comunicación con el módulo bluetooth, ya que no usan el mismo lenguaje entre dispositivos (tanto Arduino como el teléfono móvil).

Para ello se ideó un sistema muy sencillo de envío de información, el cual consiste en descomponer números en unidades sencillas (como bytes) en el Arduino y volver a componerlos en el teléfono móvil.

Se sabe que un byte puede representar valores en un rango de 256 valores [0-255], a la hora de mandar la información se usará este rango.

Se desea además una precisión de dos decimales a ser posible para ello, se plantea el siguiente sistema:

Siendo *Z* el número que se desea enviar. En el primer caso (19) se obtiene una precisión de dos decimales y en el segundo (20) de un decimal.

$$Z' = Z * 1000 \ (si \ z < 1000) \quad (19)$$

$$Z' = Z * 100 \ (si \ z > 1000) \quad (20)$$

La condición de la ecuación (19) ó (20) está puesta así ya que se puede tener valores muy elevados en la FRF y estos no se puede enviar con los mismos decimales.

Ahora que ya se tiene un número entero lo descompondremos:

$$C \ (cociente) = \frac{Z'}{256} \quad (21)$$

$$R \ (resto) = Z' - C * 256 \quad (22)$$

Teniendo así descompuesta ya la información en el rango de valores que se pueden transmitir.

A continuación, en el teléfono móvil se tiene que hacer la operación contraria:

$$S \ (salida) = C * 256 + R \quad (23)$$

Teniendo ya el número completo, pero multiplicado todavía por su correspondiente de las operaciones (19) o (20):

$$F \ (final) = \frac{S}{1000} \ (si \ z < 1000) \quad (24)$$

$$F(final) = \frac{S}{100} \ (si \ z < 100) \quad (25)$$

Con esto se cerraría la cadena de comunicación, obteniendo así el numero deseado por la pantalla que maneja el técnico.

Capítulo 3. Realización del dispositivo. Cableado del Arduino con los acelerómetros y módulo bluetooth. Filosofía de programación.

En el presente capítulo se van a describir los componentes electrónicos usados, así como sus características y el porqué de su selección.

3.1. Arduino DUE.

El microcontrolador que se usará en este TFG es de la marca Arduino, el modelo DUE.

Procesador	ARM Cortex M3 (32 bits a 84 MHz)
Puertos USB	2
Tensión de trabajo	3.3 V
Pines analógicos	12
Memoria Flash	512 KB
Puertos serie	4

Tabla 3.1.Características de Arduino DUE [6].

En la figura 3.1, se pueden observar las características de selección de este modelo.

Para empezar, se seleccionó Arduino por dos razones principales, la primera es su facilidad a la hora del aprendizaje en el entorno de programación ya que hay mucha información acerca de su uso (ya sea profesional o meramente para ocio). En segundo lugar, por su código abierto ya que existe una serie de librerías en el código que han sido desarrolladas por terceros y que son muy útiles en aplicaciones de este tipo.

Ahora se comenta el porqué de la selección de este modelo, el procesador de DUE no es el más potente de la gama pero si tiene una característica interesante. Su capacidad para trabajar con 12 bits en los pines analógicos y no con 10 como el resto de los modelos.

La posibilidad de tener dos puertos USB, uno que sirve como programador y comunicador y otro con posibilidad de incorporar gadgets externos que podrían ser útiles en un momento dado del TFG.

El número de pines analógicos no ha sido de gran relevancia, ya que para esta ocasión solamente se iban a registrar datos de dos sensores en una misma dirección. Pero si en un futuro se deseara implementar mejoras en el mismo cabe la posibilidad de colocar hasta 12.

La memoria *flash* del Arduino DUE si es la más alta de la gama de modelos nuevos, esto se traduce en una velocidad de arranque superior. Además de poder acceder a un volumen de datos de programación más grande.

Y por último hablaremos de los puertos serie, en este modelo hay 4 puertos serie. Lo que se puede conseguir con ellos es una comunicación con hasta 4 dispositivos que da una gran versatilidad.

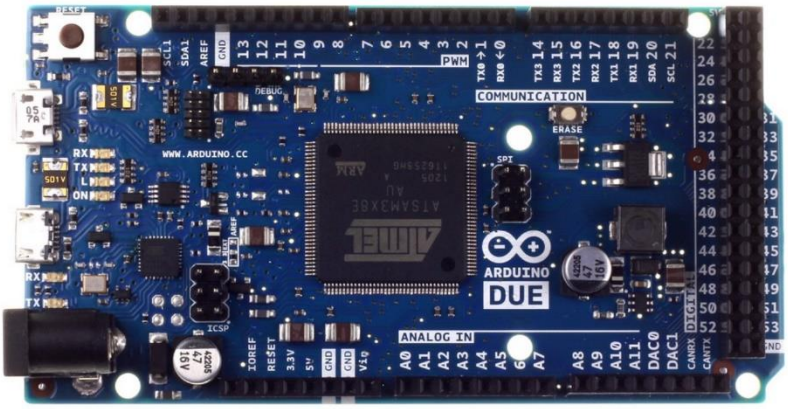


Figura 3.1. Placa Arduino DUE

3.2. Acelerómetros ADXL335.

Este acelerómetro comercial [7] es de los más usados. Hay mucha información acerca de su uso, por lo que se seleccionó dicho componente.

Figura 3.2. Acelerómetro ADXL 335, vista superior

Este acelerómetro dispone de 3 ejes para poder registrar datos, aunque se usará solamente uno de ellos para esta aplicación.

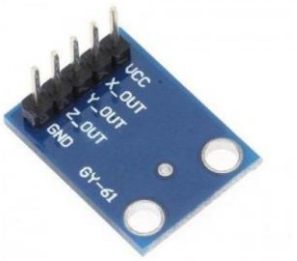


Figura 3.3. Acelerómetro ADXL, vista desde abajo.

En la vista desde abajo se pueden observar las conexiones que se usarán en el sensor, en este caso:

-VCC: es la patilla correspondiente a la tensión de alimentación del sensor, para este en específico es 3,3 V.

- X out, Y out, Z out: estas tres patillas son las correspondientes a los ejes x, y, z; del sensor. Se pueden observar sus orientaciones en la figura 3.2 vista superior.

-GND: es la patilla correspondiente a la toma de tierra del circuito, la cual va directa al Arduino.

3.3. Módulo bluetooth HC-08.

Este tipo de módulo bluetooth [8], son del tipo BLE ("*Bluetooth Low Energy*") que quiere decir de bajo consumo energético.

Figura 3.4. Módulo HC-08

Al igual que el sensor ADXL 335, tiene un patillaje similar:

-RXD y TDX: son ambas patillas con las que es posible conectar el puerto serie al Arduino, habilitando así un canal de comunicación. Es de mención que este patillaje se cruza con las homónimas en el Arduino.

-VCC: es la patilla correspondiente a la tensión de alimentación del módulo.

-GND: es la patilla correspondiente a la toma de tierra del circuito, la cual va directa al Arduino.

3.4. Elementos de conexión. Cableado. Protoboard.

Para conectar los elementos se dispone de 2 protoboard en las cuales van conectados los dos acelerómetros y el módulo bluetooth, realmente no sería necesario su uso, pero si es cierto que aporta una estabilidad a la medida al ser una superficie plana donde se pueden apoyar sendos acelerómetros y medir correctamente en el eje perpendicular al plano de apoyo de las protoboards.

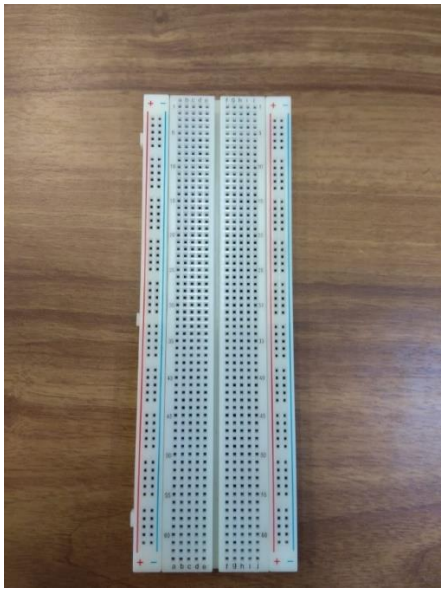


Figura 3.5. Protoboard

Además de las protoboard se usan unos cables de una longitud comprendida entre los 15 cm y los 70 cm, estos elementos no son comerciales, sino que son de elaboración propia a partir de unos cables en desuso, la modificación a dichos cables consistió en empalmar a cada extremo un cable con punta, los cuales son muy comunes en el uso de las protoboards.

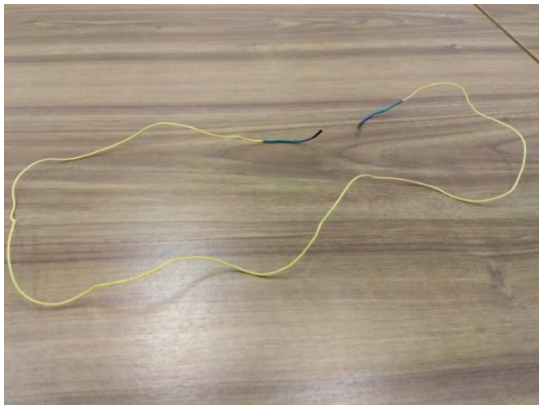


Figura 3.6. Cable modificado

3.5. Montaje de los elementos.

En este apartado se verá de una forma sencilla el montaje de dichos elementos, primero de una forma conceptual mediante la figura 3.7 y a continuación el montaje real del mismo.

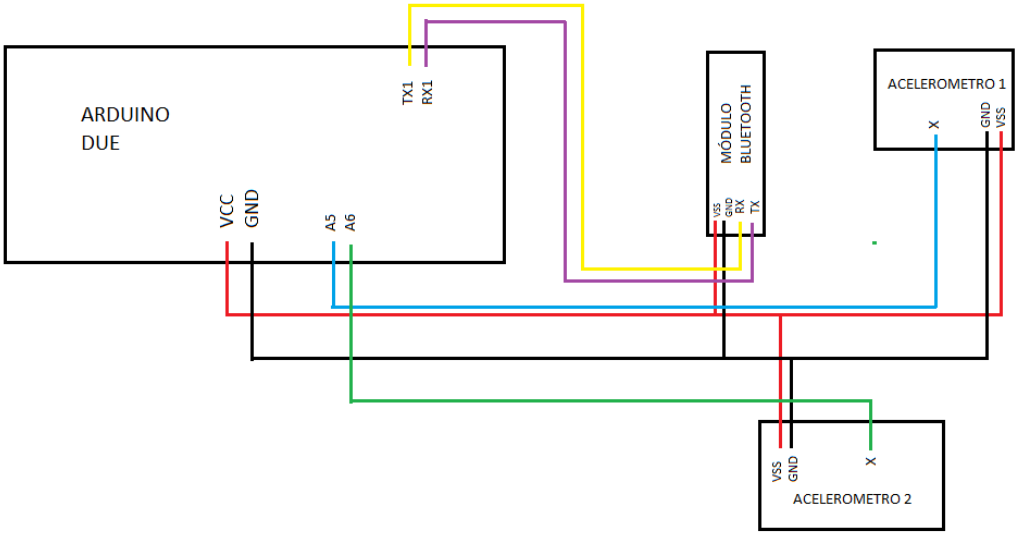


Figura 3.7. Diagrama del montaje

Primero se va a ver el diagrama representado en la figura 3.7:

- Rojo: se tiene el cableado correspondiente a la tensión de alimentación del circuito, en este caso de la placa Arduino sale una tensión de 3.3 Voltios.

- Negro: este es el cableado correspondiente a la toma de tierra del circuito, el cual es necesario para evitar un cortocircuito en la placa.

- Azul: este color corresponde a la conexión entre el eje x del acelerómetro 1 y el pin analógico A5.

- Verde: aquí se tiene la conexión correspondiente al eje x del acelerómetro 2 y el pin analógico A6.

- Amarillo: esta es la conexión correspondiente a la salida RX del módulo bluetooth, con la entrada TX1 de la placa Arduino.

- Morado: es la conexión a la cual corresponde la parte complementaria del amarillo, conecta la salida TX del módulo bluetooth con la entrada RX1 de la placa Arduino.

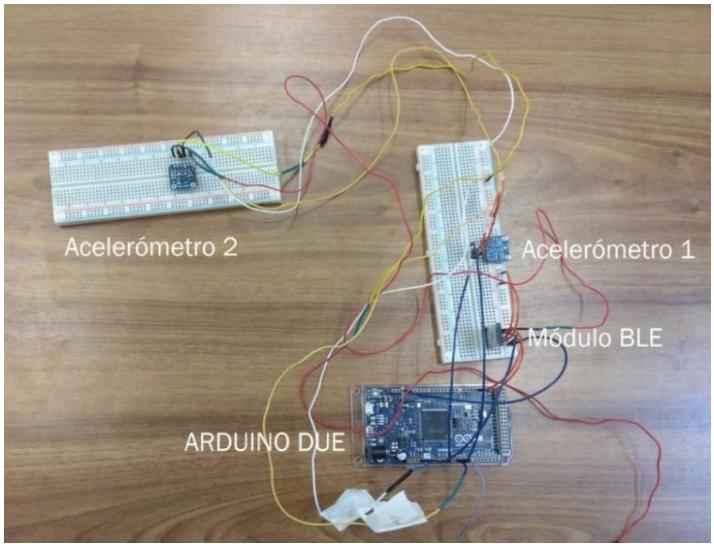


Figura 3.8. Imagen del montaje real.

En la figura 3.8 se puede observar el montaje real del prototipo con todos sus elementos juntos.

3.6. Filosofía de programación.

En este punto se describirá de una forma sencilla el funcionamiento del entorno programador de Arduino.

Este programa es de código abierto, donde los usuarios pueden modificar una serie de elementos tales como librerías propias, pero siempre sujetos a una serie de especificaciones especiales requeridas por los desarrolladores.

Arduino se caracteriza por utilizar un entorno de desarrollo propio el cual está escrito en lenguaje de programación Java, pero su uso es en lenguaje C y en C++ usando algunas reglas de escritura propias de la plataforma. Se utiliza para cargar dichos programas en tableros compatibles de Arduino, ya sean de la propia marca u otros similares.

3.6.1. Código del programa.

Lo primero que se tiene que iniciar en el programa son las librerías que se han usado, tanto la librería del filtro, cómo la librería de la transformada rápida de Fourier. Sendas librerías han sido programadas por otros usuarios de Arduino, pero ambas funcionan a la perfección. (Se ha de tener cuidado con las actualizaciones de ambas).

A continuación, vienen determinadas todas las variables que se necesitan inicializar para que el programa tenga un funcionamiento correcto. Se va enumerar mediante bloques la utilidad de cada una de las variables y después las funciones utilizadas con sus operaciones:

3.6.1.2. Bloque librerías.

En este caso se usarán dos librerías, una que servirá para que el programa realice correctamente un filtro de media móvil y otra que se usará para calcular la transformada rápida de Fourier.

```
1. #include <arduinoFFT.h>
2. #include <MeanFilterLib.h>
```

Figura 3.9. Bloque librerías en el código.

Para incluir las librerías en el programa es necesario poner de la siguiente forma; ***#include <>.***

Y dentro de los símbolos <>, se introducirá el nombre de la librería a la que se desea acceder.

3.6.1.3. Bloque registro.

En este bloque se comentarán las variables correspondientes a la parte de adquisición de datos.

```
1. /*Programa para leer datos de un acelerometro*/
2.
3. const int xpin = A5;
4. const int xpin2 = A6;
5.
6. /*Estas declaraciones son necesarias para nuestra fft*/
7. /*Estas variables son las que podemos ir cambiando para ajustar la
   fft*/
8. /*Las dejamos parametrizadas para que podamos cambiarlas a
   voluntad*/
9. const uint16_t samples = 1024;
10. int frecMuest = 200;
```

Figura 3.10. Bloque de adquisición de datos en el programa

Como se puede observar tenemos un tipo de variable, que son las de tipo enteras ("**int**"), estas sólo son capaces de almacenar datos de tipo entero y hasta un rango de valores determinado.

-Las dos primeras variables; ***cons int xpin*** y ***const int xpin***, corresponden a los pines analógicos a los que están conectados los sensores al Arduino y son los encargados de registrar la información.

-Las dos variables siguientes; ***const uint16_t*** y ***int frecMuest***, están asociadas al tamaño muestral del registro y a la velocidad de adquisición de datos de los acelerómetros.

3.6.1.4. Bloque filtrado.

En esta parte se comentarán las variables asociadas al filtro y una parte de almacenaje de los datos. Para ahorrar espacio en la memoria del Arduino Due

se ha usado la técnica de reescribir los vectores para no tener que ocupar innecesariamente dicha memoria.

```
1. /*Propiedades filtro*/
2. const short venFil = 3;
3. /*Las siguientes variables se declaran así por que necesitamos
   eliminar el pico del principio del acelerometro*/
4. float mean[samples+venFil+1];
5. float mean2[samples+venFil+1];
```

Figura 3.11. Bloque de filtro del programa

Aquí se tiene un tipo de variable que es capaz de almacenar parte decimal, son los de tipo coma flotante ("**float**"), además que se proceden a declarar también vectores.

-La primera variable ***const short ventFil*** , está asociada al número de muestras con las que el filtro hace su media, en este caso son 3.

-Las dos últimas variables ***float mean []*** y ***float mean2 []***, son las que están asociadas a todo el vector de registro de los acelerómetros, además de luego también ser las responsables de estar asociadas al filtro.

3.6.1.5. Bloque calibración.

Este es el bloque que está asociado a la calibración de los acelerómetros. Esta parte es importante porque durante el desarrollo del proyecto se observó un comportamiento anormal de la calibración que ofrecía el fabricante, por ello se incluyó este bloque. En el anexo B se detalla el método de calibración de este apartado.

```
1. /* Datos de calibración */
2. const float cal1 = 380.41/9.806; // mV/g / 9.806 m/s2/g
3. const float cal2 = 380.31/9.806; // mV/g / 9.806 m/s2/g
```

Figura 3.12. Bloque de calibración del programa

Estas variables serán también del tipo coma flotante, pero no serán vectores.

-Ambas variables son iguales ***conts float cal1*** y ***const float cal2***, la diferencia está en los números, que son casi idénticos. Esto es debido a que ambos acelerómetros tienen su propia tolerancia en la calibración.

3.6.1.6. Bloque almacenaje.

Para poder mejorar la eficacia y la precisión del prototipo se incluyó un método el cual consiste en realizar 3 mediciones diferentes y realizar la media de ellas en bloques de 2. Es decir, separar una medición completa en dos partes iguales y además añadir una tercera que corresponde a la mitad de ambas, haciendo así que el método resulte más preciso.

```
/*A cada paquete le corresponden los datos de cada acelerometro*/
float primerpaquete[samples/2];
float acel2primerpaquete[samples/2];
/*Primer paquete es el acelerometro de arriba y acel2 es el de
abajo*/
float segundopaquete[samples/2];
float acel2segundopaquete[samples/2];
float tercerpaquete[samples/2];
float acel2tercerpaquete[samples/2];
```

Figura 3.13. Bloque de almacenaje del programa.

El tipo de variables asociadas en esta parte también serán vectores, puesto que estos serán el resultado de seccionar el vector completo de registro.

-Todas las variables tienen la misma longitud, además de que cada acelerómetro tiene su propio vector de datos y están referenciados para facilitar así su programación. Siendo ***float primerpaquete [], segundopaquete [], tercerpaquete[];*** las correspondientes al acelerómetro uno y ***float acel2primerpaquete[], acel2segundopaquete[], acel2tercerpaquete[];*** las correspondientes al acelerómetro dos.

3.6.1.7. Bloque de la transformada rápida de Fourier.

Este bloque es el más largo, puesto que tiene una característica de la transformada de Fourier. Consiste en incluir en las operaciones parte real y parte imaginaria, por ello se tiene que “duplicar” el número de vectores.

```
/* Vamos a declarar ahora las variables necesarias para procesar
H1 lo dejamos en la mitad muestral por que es lo que nos
interesa*/
float NumR[samples/4];
float NumI[samples/4];
float DenR[samples/4];
float DenI[samples/4];

float NumprimeroR[samples/2];
float NumprimeroI[samples/2];
float DenprimeroR[samples/2];
float DenprimeroI[samples/2];
float NumsegundoR[samples/2];
float NumsegundoI[samples/2];
float DensegundoR[samples/2];
float DensegundoI[samples/2];
float NumterceroR[samples/2];
float NumterceroI[samples/2];
float DenterceroR[samples/2];
float DenterceroI[samples/2];

float Hreal[samples/4];
float Himag[samples/4];

float Hmagnitud[samples/4];
float Hfase[samples/4];

```

Figura 3.14. Bloque de la transformada rápida de Fourier.

En este bloque se declaran en total 20 variables que son todas vectoriales, se desglosarán a continuación para una fácil lectura:

-Los cuatro primeros, ***float NumR [], NumI[], DenR[], DenI[]***, corresponden al denominador y numerador, con su correspondiente parte real e imaginaria de las medias de las muestras realizadas. Estos se separan en numerador y denominador para facilitar la operabilidad de la parte matemática.

-Los doce siguientes, ***float NumprimeroR [], NumprimeroI[], DenprimeroR[], DenprimeroI[], NumsegundoR[], NumsegundoI[], DensegundoR[], DensegundoI[], NumterceroR[], NuterceroI[], DenterceroR[], DenterceroI[],*** se corresponden con los numeradores y denominadores de los vectores seccionados. Logrando así almacenar eficientemente los valores correspondientes a cada parte con una fácil lectura.

-Los cuatro últimos vectores son los correspondientes a la parte ya computada de la función de respuesta en frecuencia, teniendo así sus parte real e imaginaria, ***float Hreal [], Himag[];*** además de transformar estas a magnitud y fase, ***float Hmagnitud[], Hfase[],*** siendo estas últimas las que se mostrarán al usuario.

3.6.1.8. Bloque comunicación.

Este será el último bloque de variables que se incluirán en este código. Estas serán las correspondientes a la parte de comunicación entre el módulo bluetooth incorporado al Arduino y la aplicación (capítulo 3.6.2) móvil.

```
1.  /*Voy a declarar tambien dos valores para buscar un maximo en un
    vector y su posicion*/
2.  float maximo;
3.  int posicion;
4.  /*Para conseguir mandar la info por bluetooth tenemos que partir
    los numeros*/
5.  int b1;
6.  int b2;
7.  int c1;
8.  int c2;
9.  char valor;
10. long numerote;
11. long numerote2;
12. float fo;
13. float amortiguamiento;
```

Figura 3.15. Bloque de comunicación del programa

En este último bloque las variables declaradas serán de tipo entero y de tipo coma flotante.

-Las dos primeras variables corresponden al máximo valor de la función de respuesta en frecuencia y a su posición, ***float máximo, int posición,*** que con ambos valores se hallará la frecuencia propia del sistema.

-Las seis variables siguientes, ***int bq, b2, c1, c2***, ***numerote, numerote2***; se corresponden con los números descompuestos en su parte entera y su parte decimal, los cuales serán los que se envíen a través del bluetooth.

-Las dos últimas variables, ***float fo, amortiguamiento***; se corresponden con los valores que deseamos transmitir y están sujetos a la descomposición.

3.6.1.9. Bloque operaciones.

Este bloque contiene la mayor parte del programa. Para su comprensión será explicado mediante un diagrama de flujo y en el anexo C se incluirán todas las líneas de código del programa. También la asociación de cada bloque del flujograma (a las líneas de código correspondiente) estarán incluidos en el anexo C.

Primero se expondrá el flujograma con la metodología seguida y luego se asociarán las líneas de código correspondientes con los bloques del flujograma.

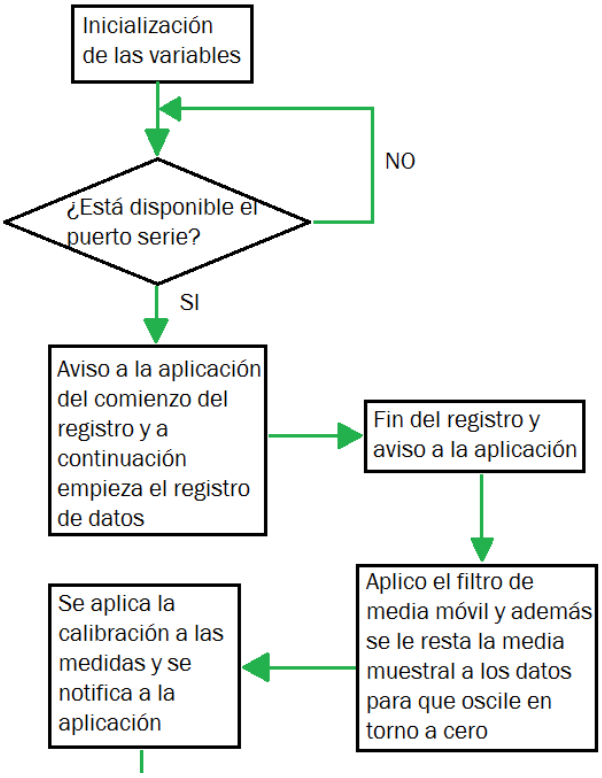


Figura 3.16. Diagrama (parte 1) de flujo del proceso para identificar la frecuencia y el coeficiente de amortiguamiento.

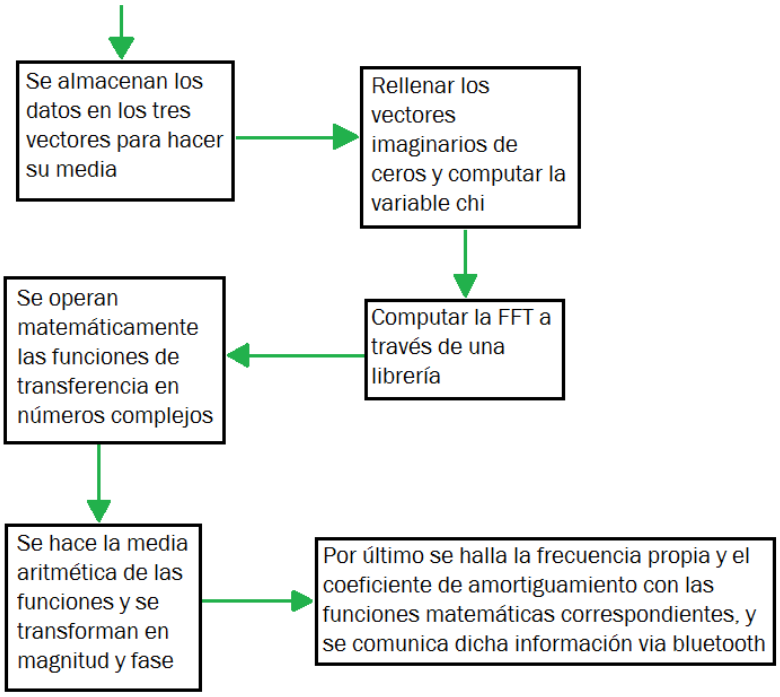


Figura 3.17. Diagrama (parte 2) de flujo del proceso para identificar la frecuencia y el coeficiente de amortiguamiento.

En la parte de inicialización de las variables están incluidos todos los bloques anteriores, ya que para poder operar primero se tiene que incluir todas las variables en las que van almacenados los datos del análisis computacional.

Una vez esto suceda, Arduino inicializará sus propias variables en un bucle llamado *void setup* (); esto es de necesaria obligación ya que estamos indicando al microcontrolador con que pines va a funcionar y de donde va a recibir las señales. Además de tener una curiosidad en este programa y es que Arduino trabajará en 12 bits, esto sólo es posible gracias a que el Arduino Due es capaz de trabajar con 12 bits en lugar de 10.

También se tiene la comunicación entre el puerto serie que es el que tiene la comunicación bluetooth, con el que trabajará a 9600 baudios. Los baudios indican una velocidad de transmisión de 9600 bits por segundo. Esta velocidad es más que suficiente para el programa.

La siguiente operación sería entrar en el bucle infinito *void loop* (), el cual permite ejecutar en bucle una serie de sentencias. La premisa que tendrá para entrar en dicho bucle será una condición del tipo *if*(), lo que indica que si el puerto serie del bluetooth está disponible se puede empezar con el bucle.

A continuación de este, se tiene puesta otra condición del tipo if (), que permite inicializar el bloque de registro cuando el usuario a través de la aplicación móvil lo requiera. Una vez que el Arduino recibe la orden de comenzar a registrar,

este la ejecuta inmediatamente y además envía información en la otra dirección indicando que está registrando datos.

Después de tener ya los datos almacenados en vectores, internamente el Arduino está aplicando un filtro de media móvil a los datos para eliminar así el ruido resultante del registro, mientras tanto en la vía de comunicación se envía la información de que el Arduino está ejecutando esta tarea pudiendo así el usuario seguir el proceso.

Una particularidad de este montaje es que al principio del registro los acelerómetros tienen una súbita subida de tensión, generando así mucho ruido y una serie de puntos anormales en el registro. Se optó por la solución de quitar siempre esos valores que estropearían la totalidad de la medición. Por lo que el Arduino reescribirá dichos vectores de datos quitando los 3 primeros valores.

Para una correcta lectura después de la función de respuesta en frecuencia se forzó a los valores que oscilen en torno a cero. La manera de proceder es la siguiente, primero se hace un promedio de cada vector y una vez hecho esto se restará a cada valor dicho promedio. Obteniendo así ya vectores que oscilen en torno a cero. Una vez hecho esto el Arduino enviará la información a la aplicación de dicha operación.

Después de obtener la medición completa, los vectores son troceados como se describe en el apartado 3.6.1.6.

A continuación, para poder trabajar con los dos acelerómetros como se describió en la ecuación (2). Se restan ambas magnitudes para relacionar la entrada y la salida del sistema.

Una vez hechas todas las operaciones anteriores se llama una de las librerías que se usan. Esta es la de la computación de la FFT. Una vez llamada la librería para cada función se trabaja ya en el dominio de la frecuencia. Esta librería se usará meramente como herramienta y no se desarrollará nada más acerca de la misma.

Cuando ya se obtiene la FFT, la próxima operación consiste ya en elaborar la FRF del sistema, esto se hace mediante el desarrollo matemático que se describe para llegar a la ecuación (13). Una vez computada ya la FRF, el objetivo es sacar dos parámetros característicos que serán los de interés del estudio, los cuales recordamos son la frecuencia propia y el coeficiente de amortiguamiento de un sistema, con las ecuaciones (15) y (17). Esto como viene siendo la tónica del programa se comunicará al usuario también.

Para que el usuario pueda controlar todo el proceso, existe un sistema de banderas que avisa del estado en el que se encuentra el Arduino. En el apartado 3.6.2.1 se explicará con más detalle.

El último paso es la comunicación final de los datos obtenidos a la pantalla del usuario, la parte matemática de esta parte abarca las ecuaciones (19) a la (25),

ya que esta al ser un volumen de información mayor no es tan sencillo de comunicar con la aplicación móvil.

3.6.1.10. Dificultades encontradas y soluciones.

A medida que se desarrollaba el prototipo surgieron una serie de complicaciones con el mismo, ya sea por las propias limitaciones del dispositivo o bien por la dificultad de cálculo de los algoritmos. Por ello en ese apartado se comentarán dichas dificultades y las soluciones aportadas.

Lo primero es hablar del filtro digital de media móvil, se ha elegido este filtro que es uno bastante fuerte porque a pesar de que el propio acelerómetro tiene filtros físicos internos, estos no son suficiente para obtener una señal limpia y libre de ruidos.

A continuación, se hablará de la restricción más importante encontrada y fue la memoria de procesamiento del Arduino. Para poder efectuar una buena FRF el tamaño muestral debe ser bastante grande y además ser una potencia de 2, pero el procesador de Arduino sólo puede computar datos tipo *long* hasta cierto punto. Tanto por el propio tamaño de estos datos como por el número de veces que se computan. Por lo que mediante ensayos heurísticos (prueba y error) se estipuló que en el código el tamaño máximo que se podía poner por muestras era de 512 datos. Mediante el mismo tipo de ensayos también se halló la máxima velocidad de muestreo en 200 muestras por segundo.

Traduciendo esto a la fase experimental se tiene pérdida significativa de precisión y sensibilidad en el registro. Por lo que en el registro si el prototipo no es capaz de identificar correctamente el pico del sistema [2], puede haber una ligera desviación en la frecuencia propia.

Por último, se comentará el porqué de elegir este sistema de comunicación y no otro. El sistema como se puede ver en el capítulo 2 consiste en enviar paquetes de un byte de información. En la fase de desarrollo en las primeras comunicaciones entre dispositivos, se pudo observar que la comunicación en tiempo real de ambos era defectuosa, por lo que se optó fraccionar la información y enviar estas fracciones de información. Posteriormente se componen dichas fracciones y el usuario puede leer dicha información.

3.6.2. Mitappinventor.

Para el desarrollo de la comunicación vía bluetooth se usará este desarrollador de aplicaciones creado por Google. La parte de programación se rige por un sistema muy sencillo. Consiste en bloques de sentencias de programación que se unen entre sí al igual que un puzle.

Figura 3.18. Vista general del entorno de diseño de una aplicación

A la hora de proyectar la pantalla de una aplicación, se tiene un menú desplegable a la izquierda con una serie de botones, cajas de texto, sensores, métodos de comunicación, etc.

Estos deben de ser arrastrados a la "pantalla" del teléfono genérico mostrado en la imagen. Para personalizar el diseño de dicha pantalla a la derecha de la imagen se pueden observar dos menús más. Se ubican a la derecha de la pantalla y corresponden a la selección de botones, actuadores, etc. El menú ubicado más a la derecha sirve para cambiar el tamaño de la fuente del texto, el color y elementos más visuales.

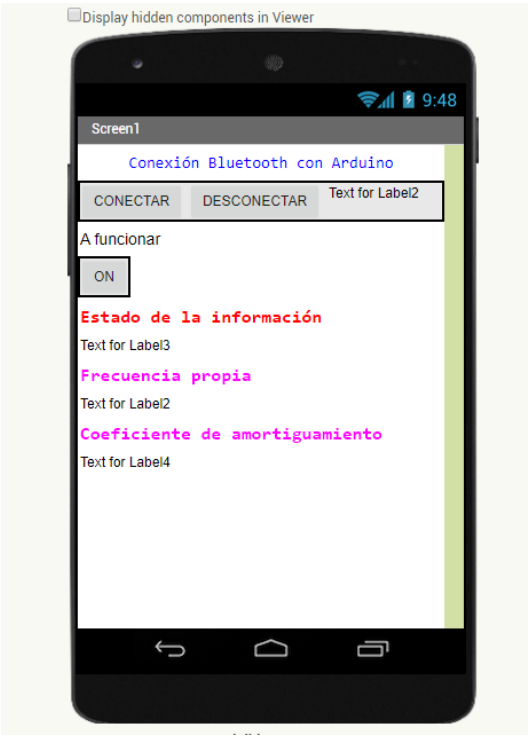


Figura 3.19. Pantalla de inicio de la aplicación

En la imagen 3.19 ya se puede observar la aplicación final desarrollada para el presente proyecto.

Se van a describir brevemente los elementos de la pantalla, los cuales como se pueden observar no son excesivamente complicados.

-El botón *CONECTAR*, al clicarlo abre un menú desplegable con todos los dispositivos bluetooth cercanos disponibles, al clicar sobre uno de ellos intentará conectarse a él si su puerto de comunicación está disponible.

-El botón *DESCONECTAR*, al hacer clic sobre él desconecta automáticamente el módulo bluetooth al que estaba conectado el dispositivo.

-La etiqueta 2 (*text for label* 2), muestra el estado de conexión del sistema, si está conectado al bluetooth aparece un mensaje de "conectado", y si por el contrario no está conectado, manda un aviso de "desconectado".

-*Estado de la información*, (*text for label* 3), muestra el estado de la información en el sistema mostrando un "registrando" cuando los acelerómetros están activados y recogiendo datos. "Filtrando señal" cuando el Arduino está efectuando los algoritmos de filtrado de señal. "Almacenando" cuando el Arduino está guardando el paquete de datos principal en 3 paquetes más pequeños. "Calculando" cuando el grueso del programa está trabajando en efectuar la parte matemática del sistema y para terminar "finalizado" para mostrar en la pantalla que finalmente ha terminado de trabajar.

- *Frecuencia propia*, debajo de la cual en la etiqueta correspondiente se muestra la frecuencia propia del elemento que es ensayado, con una precisión de dos decimales y con la unidad en Hz.

-*Coeficiente de amortiguamiento*, también debajo de la misma se muestra el coeficiente de amortiguamiento de la estructura expresada en porcentaje, con una precisión también de dos decimales.

A continuación, en la figura 3.20 se mostrará la parte de bloques de la herramienta, esta es visualmente muy agradable y sencilla de comprender.

Figura 3.20. Pantalla del entorno de programación de la aplicación.

En la parte central del entorno se puede observar una serie de bloques armados como un puzle. Estos bloques están prediseñados y programados para ser ensamblados. Así que la única tarea del usuario es armar los bloques evitando así los errores de lógica de programación. Ya que además estos bloques tienen restringido su uso con determinados bloques.

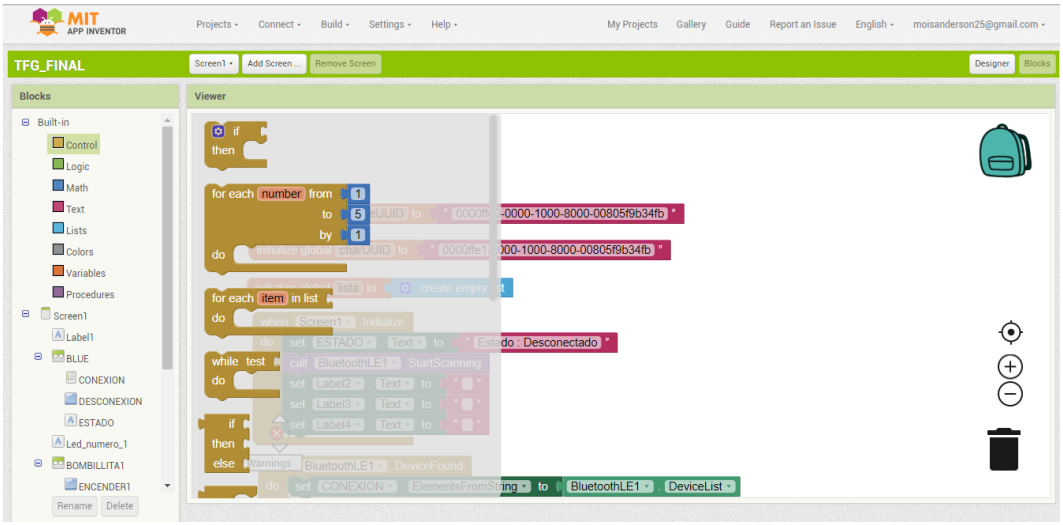


Figura 3.21. Desarrollo del menú de bloques

Para colocar dichas sentencias a la izquierda del menú principal de los bloques, hay un desplegable que almacena todas las sentencias que se pueden ejecutar. Además, existe un índice de ítems colocados en la pantalla de diseño.

Dentro del menú de desplegables existen 8 tipos de sentencias para trabajar, las cuales son del tipo control, lógicas, matemáticas, textuales, listado, colores, variables y procesales.

Al hacer clic sobre cada una cómo se puede ver en la figura 3.20, se abre un menú en paralelo con el primero, que muestra todas las sentencias de cada tipo.

3.6.2.1 Mitappinventor en el programa

En este apartado se comentarán los aspectos relacionados con el proyecto, en este caso en particular los bloques con sus sentencias y su funcionamiento específico en cada parte del sistema.

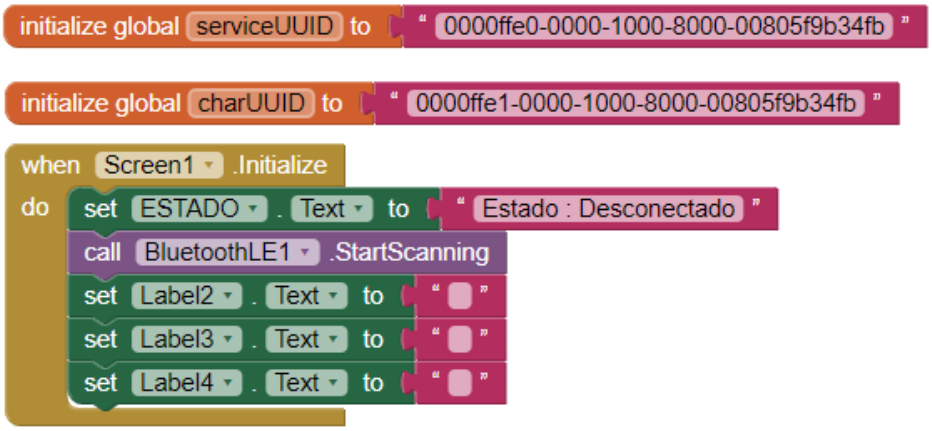


Figura 3.22. Bloque de sentencias de inicialización

En este pequeño grupo de bloques se comentarán todas las sentencias que se deben ejecutar al principio del programa para que este funcione.

Primero *initialize global serviceUUID to* y *initialize global charUUID to*, se necesitan para poder contactar con el módulo bluetooth que se usa en el Arduino. Estas dos hileras de números son los identificadores individuales de cada módulo BLE, en este caso dichos números se pueden obtener directamente del módulo.

Justo debajo tenemos una serie de bloques conectados a una sentencia lógica (color marrón), que indica cuando la pantalla principal se inicie ejecuta 5 subsentencias, 4 son del tipo texto, de las cuales 3 son dejar automáticamente sin texto y la restante indica el estado de conexión con el módulo bluetooth. La sentencia restante ejecuta la búsqueda de dispositivos bluetooth disponibles.

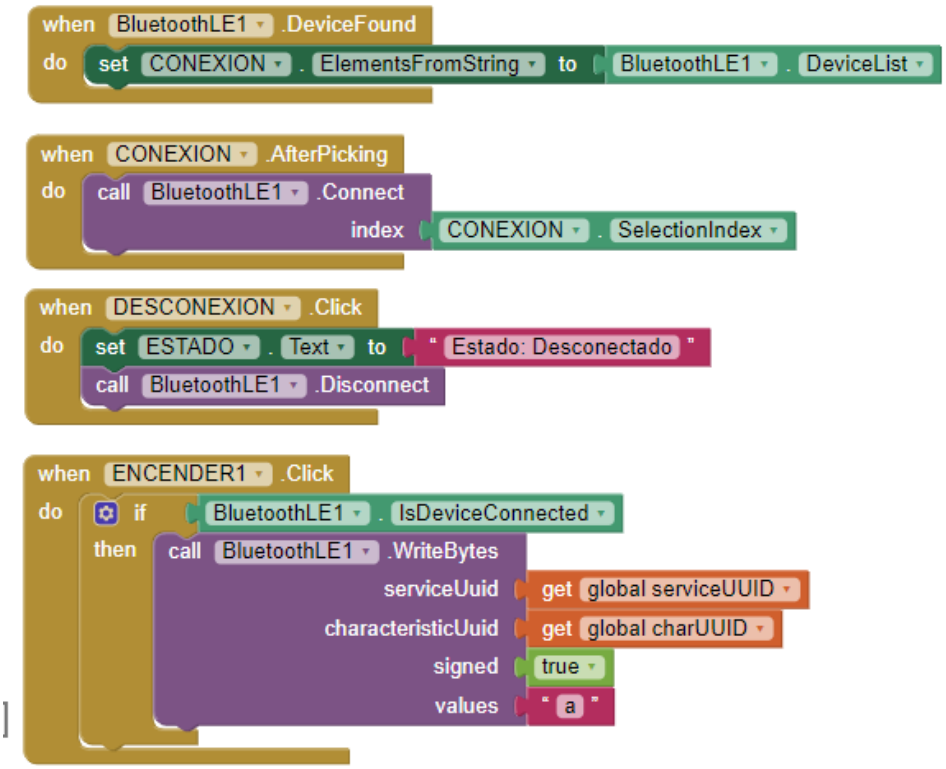


Figura 3.23. Bloque de sentencias de inicio de registro

El siguiente grupo de sentencias se ejecutarán inmediatamente después. Estas se centran principalmente en el establecimiento de la conexión.

Los 4 bloques son sentencias de control, el primero *when BuetoothLE1.Devicefound*, indica que cuando encuentre dispositivos bluetooth disponibles haga una lista de dichos dispositivos. Esto en conjunto con la segunda sentencia *when CONEXIÓN. Afterpicking*, que selecciona el dispositivo deseado de dicha lista, permite establecer la conexión al dispositivo.

La tercera sentencia *when DESCONEXIÓN. Click* ejecuta dos subsentencias. La primera es que en la etiqueta que indica el estado de la conexión escriba "Desconectado" y la segunda es que ejecute esa desconexión con el dispositivo y lo libere.

La cuarta y última sentencia *when ENCENDER. Click,* ejecuta una comunicación con el bluetooth, con una sentencia de condición la cual únicamente condiciona la disponibilidad del dispositivo. Esta manda una letra "a" al dispositivo, que interpreta el comienzo de las tareas de registro.

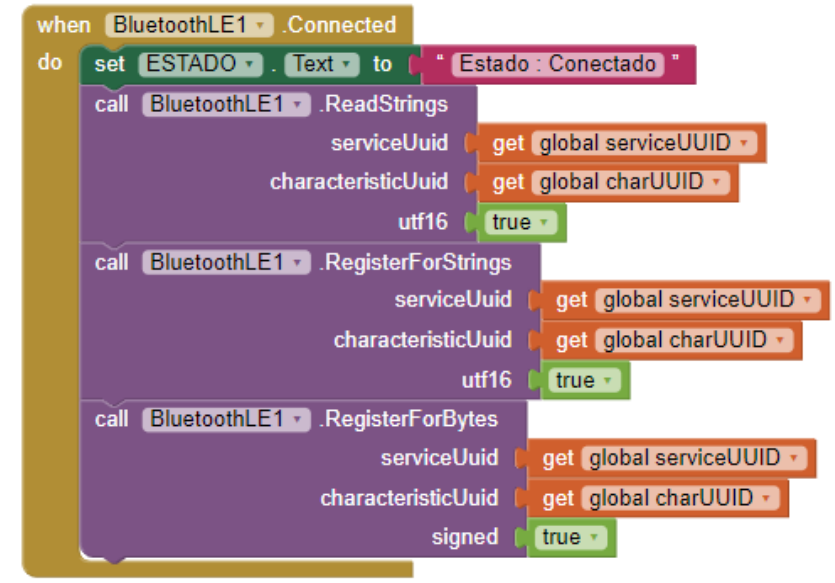


Figura 3.24. Registro del tipo de datos enviados desde Arduino.

Este es otro bloque de control ejecuta una escritura en la etiqueta estado: "conectado". Cuando el bluetooth seleccionado esté correctamente conectado. Además de habilitar el registro de datos del tipo byte.

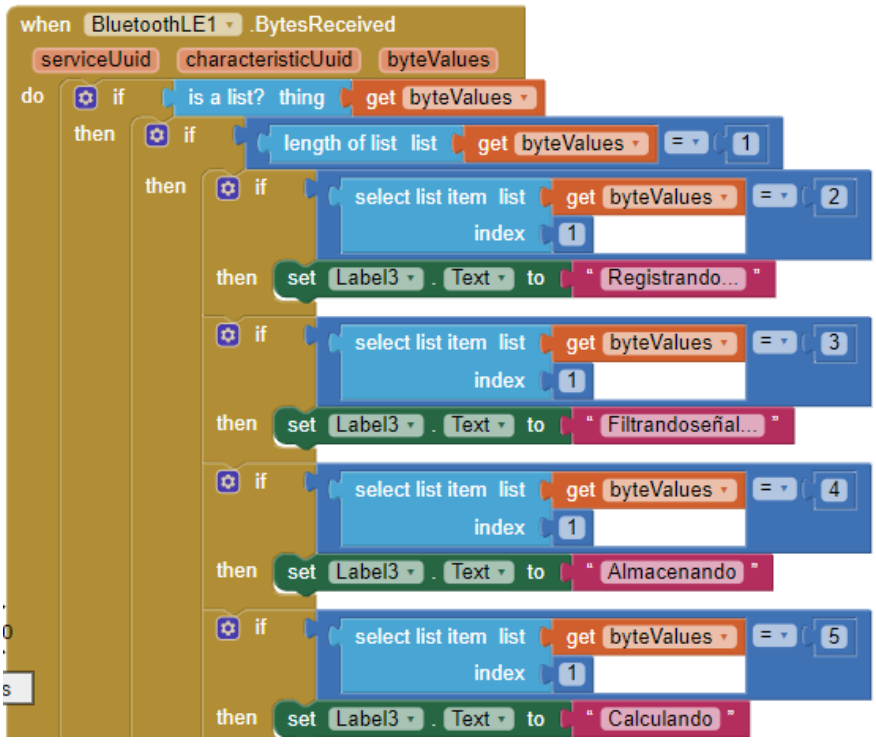


Figura 3.25. Funcionamiento de las banderas del programa.

En la figura 3.25, se observa como la aplicación móvil procesa los datos recibidos del Arduino. En este caso la sentencia de control es la recepción de datos en forma de byte, a su vez este tiene otra sentencia de control. Dicha sentencia indica que si recibe una lista de dichos datos ejecute un proceso. En este caso dos procesos paralelos, los cuales se diferencian en el tamaño de la lista que recibe.

Si es de tamaño 1 ejecuta dichas banderas, que indicaran el estado de ejecución del proceso.

Cada sentencia individual viene a ejecutar el siguiente proceso, si la lista es de longitud 1, entonces se procesará ese dato dependiendo de qué número sea, (rango 2-6), se escribirá texto en la etiqueta estado de la información.

Si es de tamaño 3 ejecutará unas operaciones matemáticas que se muestran a continuación.

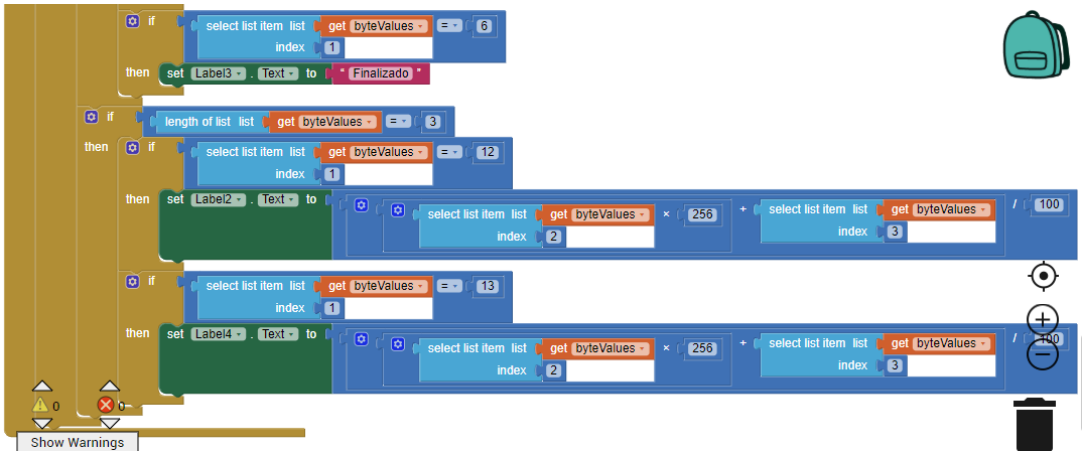


Figura 3.26. Ejecución de la parte matemática del sistema.

En la figura 3.26, también se puede ver que se sigue con las sentencias de control para poder proseguir con la tarea. Una vez que se cumple la condición de un tamaño de lista de 3 elementos, esta se vuelve a subdividir en dos tareas. Si se tiene la bandera igual al número 12, entonces se estará hablando de la frecuencia propia del sistema y esta se colocará en la etiqueta número 2.

Si es el número 13 entonces se estará procesando el coeficiente de amortiguamiento del sistema. Ambas dos operaciones se detallan en el capítulo dos, en la parte de descomposición y composición numérica.

Capítulo 4. Ensayos de prueba en edificios de dos plantas.

4.1. Sistema ubicado en maqueta de edificio.

Primero para asegurar el funcionamiento del prototipo se fue ejecutando paralelamente su desarrollo con pruebas en una maqueta de un edificio de dos plantas.

Al principio antes de intentar validar el prototipo con un sistema comercial y fiable (SIRIUS) se tuvo un problema conceptual y fue el siguiente.

Figura 4.1. Maqueta del edificio a ensayar

En la imagen 4.1, se puede observar la maqueta de un edificio de dos plantas. Como se desarrolló en el capítulo 2 para que el método de la transmisibilidad sea efectivo, es necesario que las señales de entrada y de salida tengan una diferencia apreciable. Es aquí donde surge el problema y es que en la disposición tal como se muestra en la imagen 4.1, el método no funciona. La razón es la sensibilidad de los acelerómetros usados. Ya que con estos la diferencia entre entrada y salida son prácticamente inexistentes.

Para seguir desarrollando el prototipo se optó por la siguiente vía:

Figura 4.2. Maqueta del edificio a ensayar invertida.

La figura 4.2 muestra dicha solución, sólo se le dio la vuelta a dicha maqueta del edificio para poder obtener así la diferencia entre entrada y salida que se buscaba.

En la figura 4.3 se muestra una gráfica comparativa. Primero con la colocación tal como se muestra en la figura 4.1 y después como se ve en la figura 4.2. Manteniendo el resto de los parámetros de ensayo idénticos.

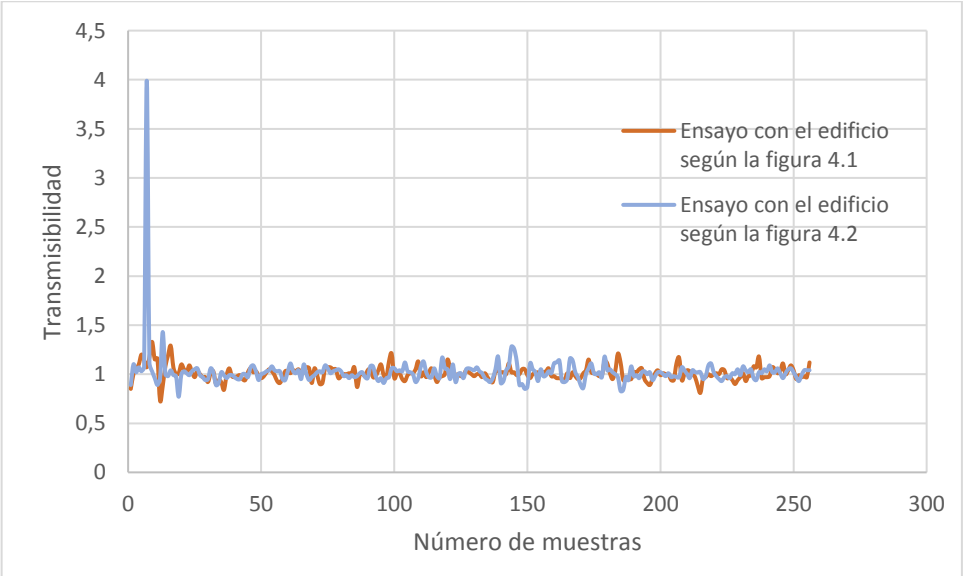


Figura 4.3. FRF obtenidas según la disposición de las figuras 4.1 y 4.2.

Como se puede observar en la figura 4.3, ambas FRF son diferentes. En azul se tiene el característico pico que se busca (figura 2.2) y en la gráfica de color naranja, claramente dicho pico no se observa.

Todos los ensayos realizados durante este documento siguieron un idéntico procedimiento. Las maquetas objeto de estudio fueron golpeadas con un martillo en su planta intermedia. A continuación, empezaron los registros de datos.

Primero se verán los resultados obtenidos por el equipo profesional, a continuación con el sistema *low-cost* y para finalizar una comparativa de ambos.

4.2 Ensayos con SIRIUS.

En la imagen 4.4 se puede observar el montaje de los acelerómetros en el edificio a ensayar. Ambos acelerómetros están colocados en cada una de las plantas del edificio y están adheridas a dichas plantas mediante imanes. Estos se acoplan con la parte metálica de dichas plantas.

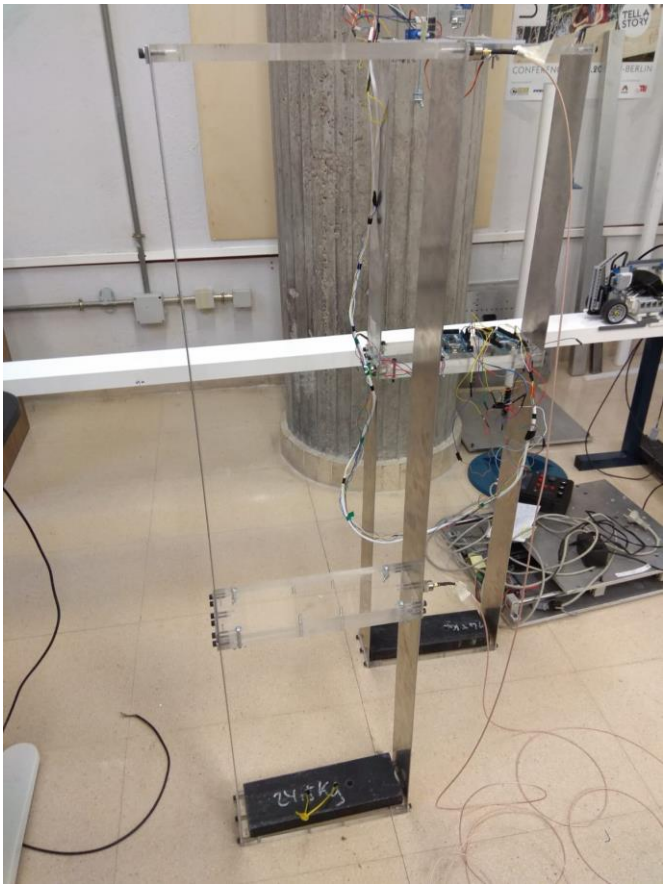


Figura 4.4. Edificio de dos plantas con los acelerómetros conectados.

Estos acelerómetros [9] se conectan mediante un cable cada uno a la tarjeta de adquisición de datos SIRIUS, que está montada como se muestra en la figura 4.5.

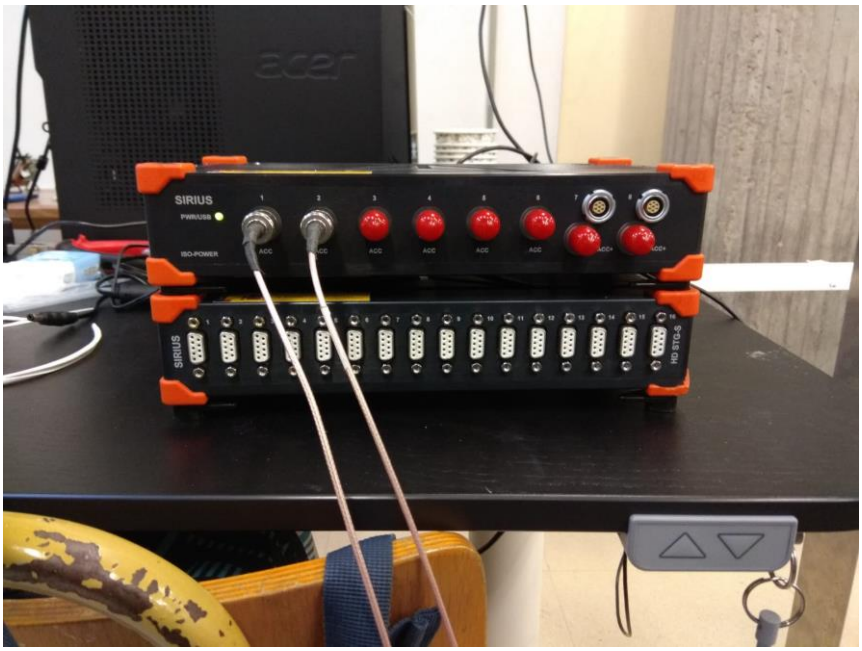


Figura 4.5. Tarjeta de adquisición de datos SIRIUS en funcionamiento vista de frente.

Figura 4.6. Tarjeta de adquisición de datos SIRIUS en funcionamiento vista desde arriba.

Con este montaje ya realizado se inició el proceso de registro. Para ello es necesario manejar un programa específicamente diseñado para la tarjeta de adquisición SIRIUS [10]. Dicho programa está desarrollado por el fabricante DEWSOFT.

Figura 4.7. Interfaz del programa DEWSOFT3x.

Este programa no es competencia de desarrollo del siguiente documento, por lo que solamente se comentarán los resultados obtenidos del mismo. Para ello se explicará brevemente que datos se muestran por pantalla de dicho programa.

En la parte izquierda de la ventana se pueden observar 3 gráficas consecutivas, estas se corresponden con las aceleraciones a las que está sometido el edificio. Las dos primeras corresponden a cada uno de los acelerómetros conectados (gráficas en amarillo) y la tercera (gráficas en azul) se corresponde con la resta de ambos, para así tener nuestras señales relacionadas (2).

En la parte derecha superior se pueden observar otras dos gráficas (de color verde), ambas corresponden a una función de respuesta en frecuencia FRF del sistema objeto de estudio. La gráfica situada en la parte superior se corresponde con la magnitud de dicha FRF y la gráfica de la parte inferior se corresponde con la fase de dicha función.

Por último, se puede observar en la parte inferior derecha otra gráfica diferente, que sigue el método circle-fit (apartado 44.2). De la cual se puede extrapolar el coeficiente de amortiguamiento del sistema objeto de estudio.

Bien una vez explicada esta interfaz, se procede a estudiar el caso particular que concierne al ensayo.

Para poder comparar fiablemente los resultados, lo primero que se hizo fue realizar el ensayo del edificio con el mayor número de muestras que puede registrar la tarjeta de adquisición de datos. Con una frecuencia de muestreo adecuada para este tipo de ensayos.

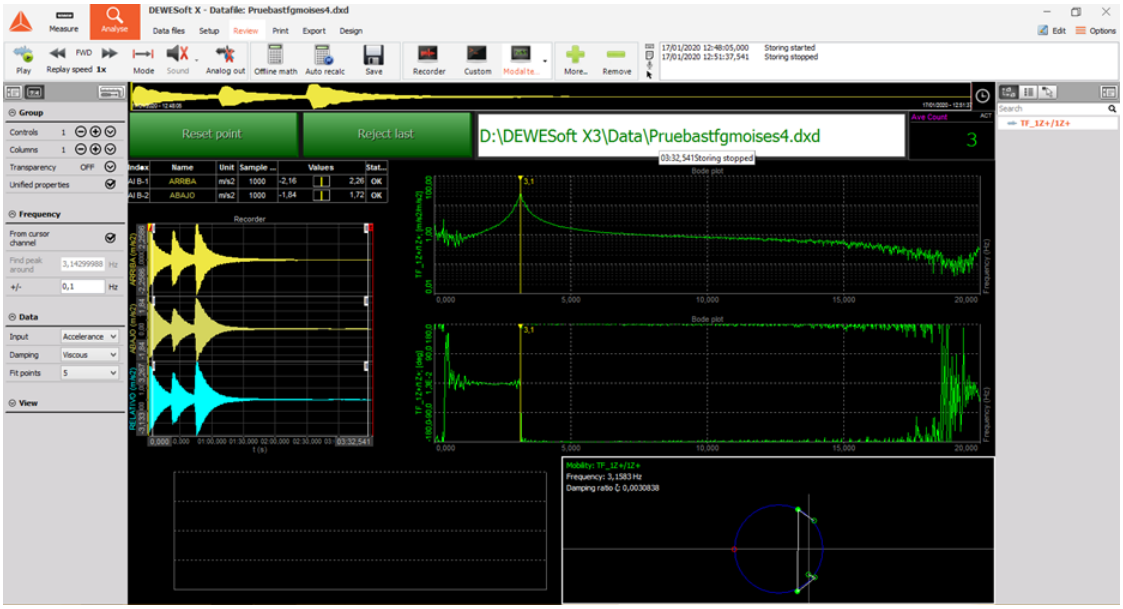


Figura 4.8. Imagen correspondiente al ensayo con más precisión.

El número de muestras son 32768 y la frecuencia de muestreo son 500 muestras por segundo, ambos valores son los marcados por el técnico de laboratorio que realiza este tipo de ensayos.

Los datos que interesan en este estudio son la frecuencia propia del sistema y el coeficiente de amortiguamiento. La frecuencia propia se obtiene directamente interpretando la FRF de la siguiente manera. El pico máximo el cual se puede observar en la gráfica de magnitud de la FRF se coloca el cursor encima y de ahí se puede leer directamente el valor en el programa, en este caso es 3,125 Hz.

Y de la gráfica inferior derecha se puede extrapolar directamente el dato que deseamos obtener el cual es el coeficiente de amortiguamiento, en este caso corresponde a un 0,00308, dicho en porcentaje un 0,3%. Dicho de otra manera, el edificio a ensayar apenas disipa energía al ser excitado.

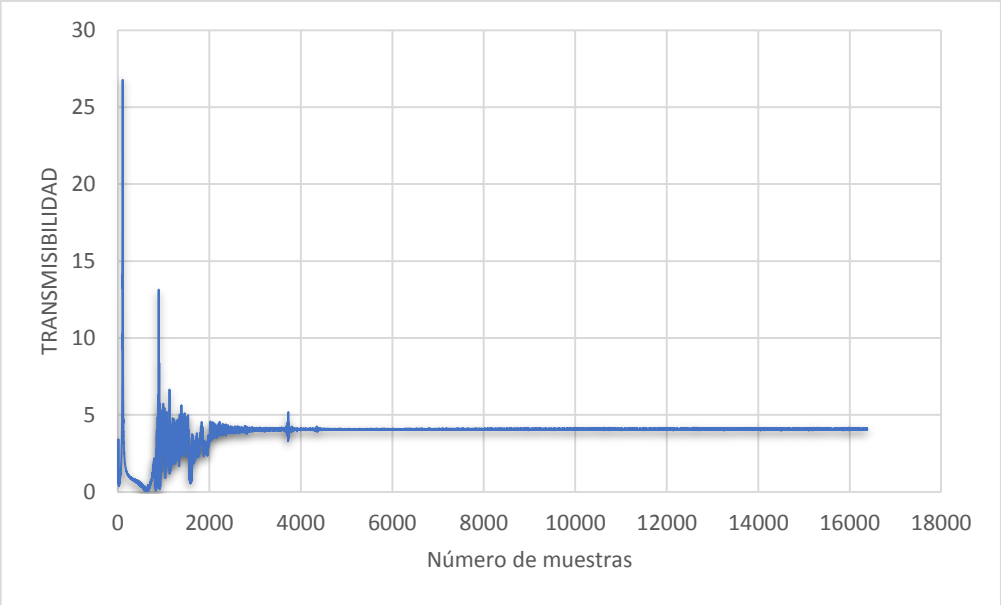


Figura 4.9. FRF en magnitud correspondiente al ensayo realizado a 500 muestras por segundo en un total de 32768 muestras.

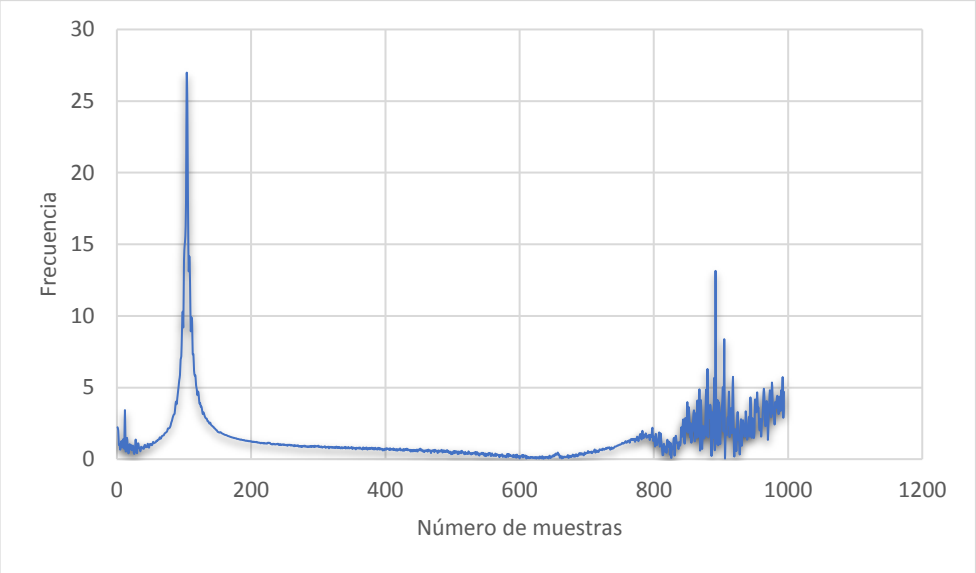


Figura 4.10. Zoom realizado a la figura 4.9

En la figura 4.9 se puede observar la FRF en magnitud correspondiente al ensayo. Para facilitar su lectura la figura 4.10 se corresponde con el mismo ensayo, pero sólo mostrando las 1000 primeras muestras. En la figura 4.10 se puede ver claramente marcado al principio de la gráfica.

Por último, para poder comparar con los mismos parámetros del Arduino, se realizó otro ensayo este con 512 muestras y a 200 muestras por segundo.

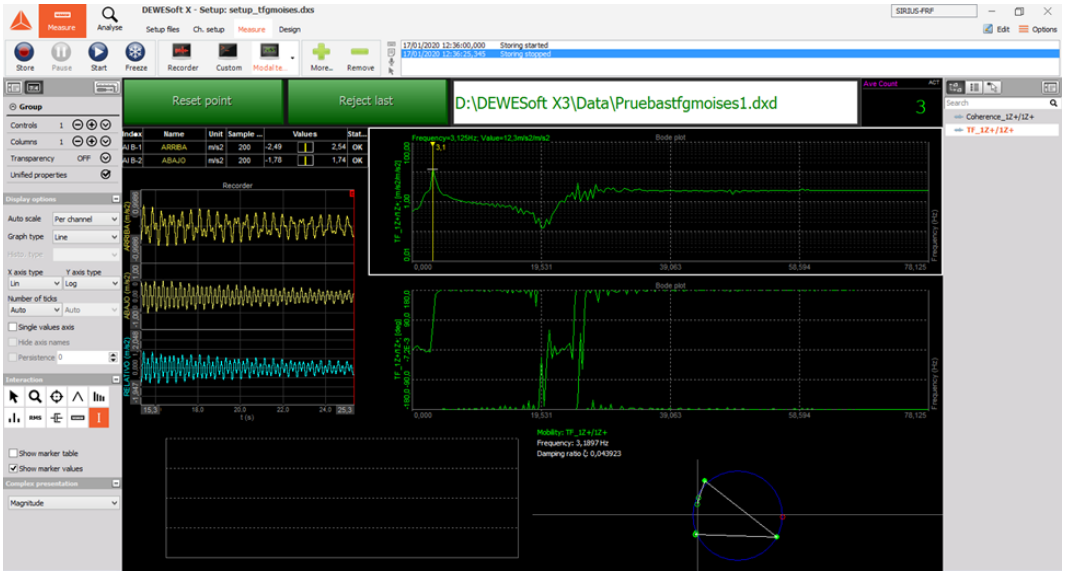


Figura 4.11. Imagen correspondiente al ensayo con los parámetros de Arduino

Como se puede observar en la imagen 4.11 el número de muestras, que es significativamente menor, afecta en la representación gráfica. Ya que se puede observar en todas las gráficas una tendencia mucho más filosa en sus cambios de pendiente y no tan suavizada como antes.

En este caso la frecuencia propia del sistema no se ha visto influenciada y la tenemos en 3,12 Hz, que es igual a la obtenida en el anterior ensayo.

Sin embargo, el coeficiente de amortiguamiento obtenido si ha cambiado radicalmente a un 0,043, es decir un 4,3%. En el apartado de conclusiones se comentará esta disparidad entre ensayos.

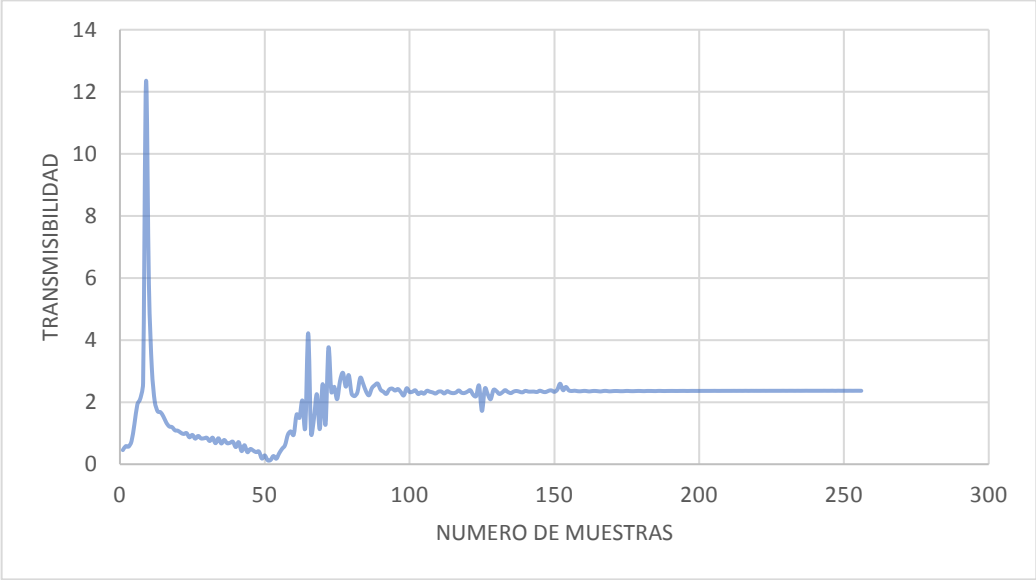


Figura 4.12.FRF en magnitud correspondiente al ensayo realizado a 200 muestras por segundo en un total de 512 muestras.

Como se puede comparar con el anterior ensayo en la forma de la FRF apenas se tiene diferencias apreciables, sin embargo, donde existe una diferencia notable es en la magnitud del máximo. La magnitud es la mitad en este ensayo en comparación al primer ensayo. Por lo que afecta a la propiedad del coeficiente de amortiguamiento, esto se comentará más exhaustivamente en el apartado de conclusiones.

4.3 Ensayos con Arduino.

La manera de proceder en la ejecución de este ensayo es exactamente igual que la descrita en el apartado 4.1, con la salvedad de la colocación de los elementos. En la imagen 4.13 se podrá observar con más detalle:

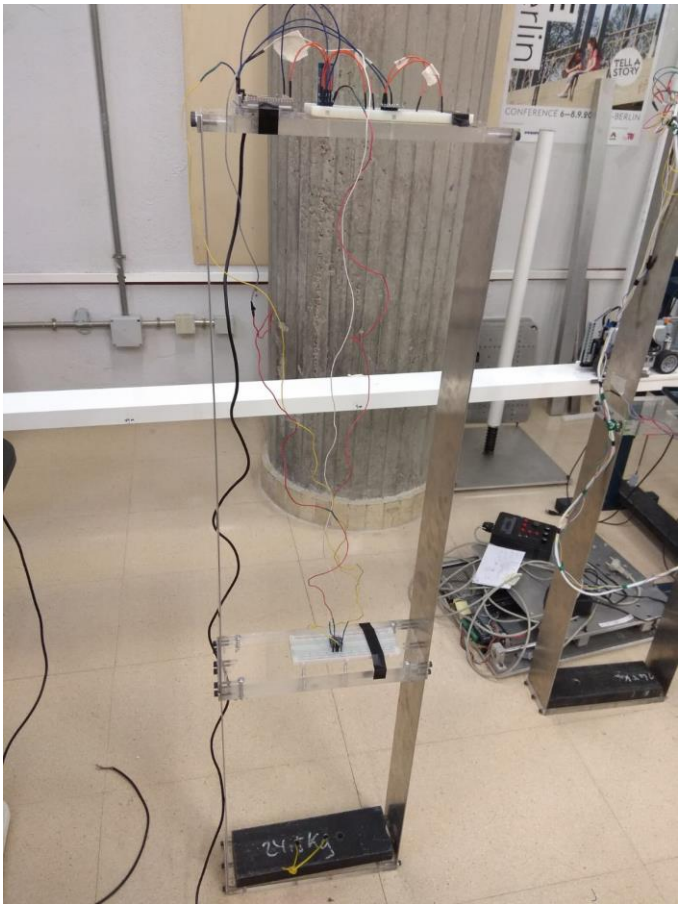


Figura 4.13. Montaje del ensayo correspondiente con Arduino.

En este ensayo como se puede observar, sendos acelerómetros están colocados encima de los pisos del edificio y el sistema de adquisición de datos en este caso está colocado sobre el piso superior.

Para este ensayo analizaremos el máximo número de muestras que puede registrar el Arduino y con dos frecuencias de muestreo diferentes, la primera

será de 100 muestras por segundo (ensayo 1) y la segunda de 200 muestras por segundo (ensayo 2).

Se realizaron múltiples medidas por lo que se mostrará una media de ambos casos.

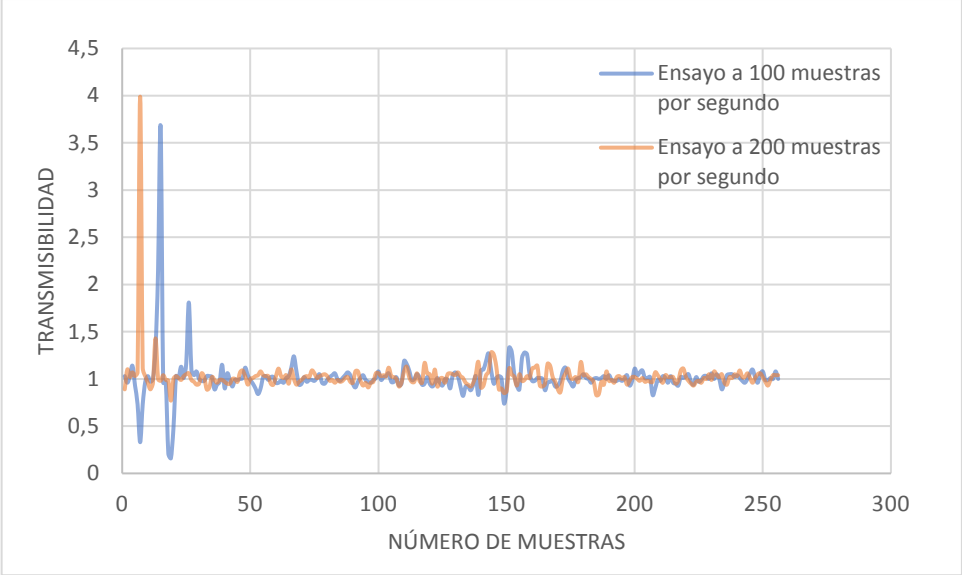


Figura 4.14. Comparativa de 2 FRF con frecuencias de muestreo diferentes.

Como se puede observar en ambas FRF tenemos una forma similar a la vista en los ensayos realizados con la tarjeta de adquisición de datos SIRIUS. Con la salvedad que en Arduino las gráficas son mucho más acentuada en los picos. Además de tener una ligera disparidad al final de la gráfica, esto nos indica que tenemos una cantidad de ruido mayor en los registros.

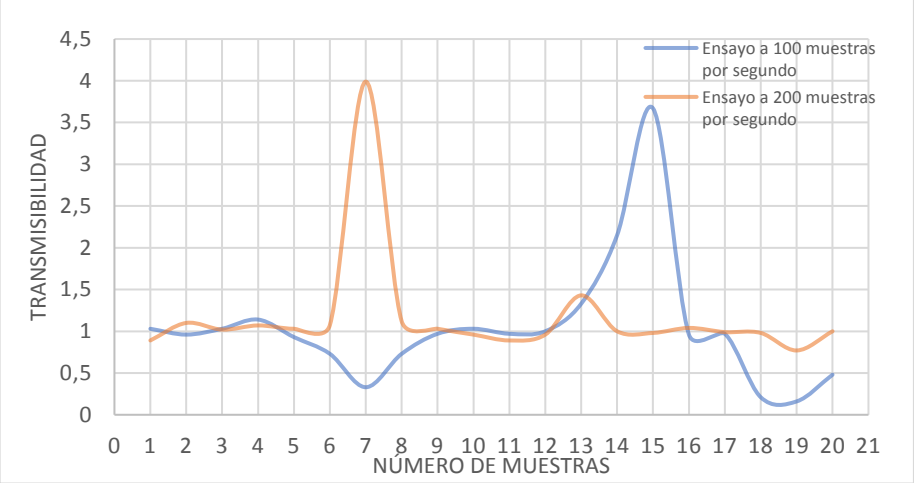


Figura 4.15. Zoom realizado a la figura 4.14.

En este caso de registro del ensayo 1 (azul figura 4.15) la frecuencia propia del sistema es 2,93 Hz, ya que la muestra con la frecuencia más alta es la número

15. Para hallar dicho dato se usó la ecuación (15). Pero a la hora de analizar el coeficiente de amortiguamiento perdemos un importante rango de valores y este se nos queda en 0,13, muy lejos de lo obtenido con el sistema profesional. Para obtener dicho coeficiente se usó la ecuación (17). También es de nombrar que los métodos usados para obtener este coeficiente son mucho menos preciso que con el sistema profesional, pero aquí los cálculos están limitados por el número de muestras registradas.

La FRF del ensayo 2 (naranja figura 4.15) también tiene la misma forma que las anteriores. Con un pico claramente marcado el cual indica la frecuencia propia del sistema.

En este caso el pico más alto está ubicado en la muestra número 7, usando de nuevo la ecuación (15), se obtiene una frecuencia propia 2,73 Hz. Que se separa aún más de la frecuencia propia obtenida con el sistema profesional y el coeficiente de amortiguamiento vuelve también a estar lejos de lo esperado 0,125. Obtenido aplicando la ecuación (17).

	Velocidad de muestreo	Nº de muestras	Δf	Valor de referencia	Rango de referencia
Ensayo 1	100	512	0,195 Hz	3,12 Hz	[2,925 Hz- 3.315 Hz]
Ensayo 2	200	512	0,390 Hz	3,12 Hz	[2,73 Hz – 3,51 Hz)

Tabla 4.1. Rango de error aceptable.

En la tabla 4.1 como valor de referencia se usó el obtenido en los ensayos del apartado 4.2, concretamente figura 4.8. Para hallar Δf ecuación (14).

El rango de referencia indica los posibles valores en los que se puede cometer error, ya que la variación de posición del máximo en un solo número puede cambiar drásticamente la medida. Tal y como se puede observar en la tabla 1.

Como conclusión de estos ensayos se puede decir que el prototipo no halla exactamente el valor real, pero si es capaz de aportar una aproximación fiable.

Estos resultados obtenidos son una media de múltiples ensayos, por lo que se puede hablar qué de algún ensayo sí que fue exitoso y consiguió acercarse aún más a los resultados. Para ello se mostrarán unas capturas obtenidas directamente del dispositivo móvil.

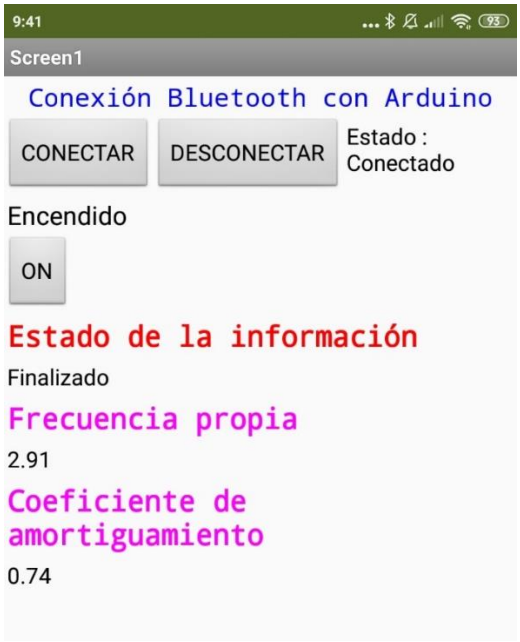


Figura 4.16. Captura de pantalla del dispositivo móvil.

Como se puede observar en la figura 4.16, en el parámetro de la frecuencia propia, se tiene un valor aceptable y dentro de los márgenes de error del mismo (valor de referencia Tabla 4.1).

En cambio, el coeficiente de amortiguamiento está bastante lejos de la realidad, teniendo así un fallo bastante considerable.

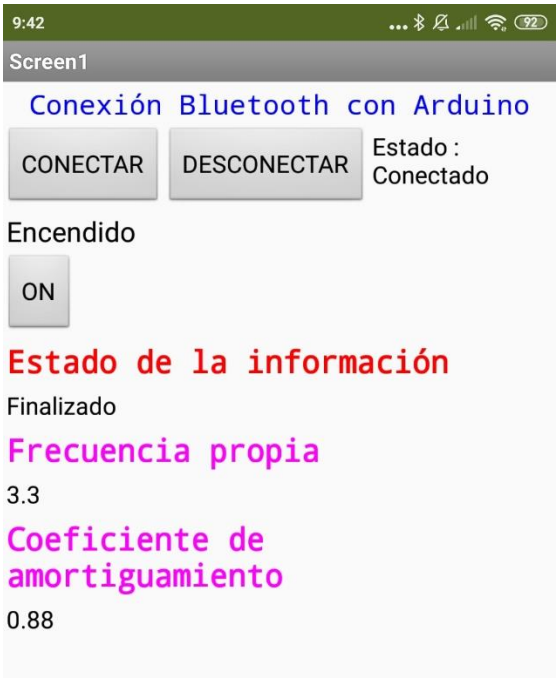


Figura 4.17. Captura de pantalla del dispositivo móvil.

Aquí se puede ver también otra captura de otro ensayo, la frecuencia propia sigue oscilando en el valor real, en este caso por encima del mismo.

Y el coeficiente de amortiguamiento se mantiene en la misma tónica que antes, por lo que será comentado en el capítulo 5.

4.4 Comparativa entre ambos ensayos.

En este apartado se procederá al análisis entre ambos resultados, sus disparidades y sus posibles causas.

En primer lugar, se van a separar las dos propiedades que conciernen al estudio para poder hablar mejor de ellas.

4.4.1. Frecuencia propia.

Esta propiedad se obtiene como ya se mencionó en el capítulo 2 ecuación (15), directamente de la función de respuesta en frecuencia (FRF). Para hallar correctamente este parámetro dependemos fundamentalmente del tamaño muestral y del muestreo por segundo. Ya que siguiendo la ecuación (15), al dividir el muestreo por segundo entre el tamaño de la muestra, se tiene la resolución entre la cual se puede mover la FRF.

Por ejemplo, si se tiene un tamaño de muestra 512 y una velocidad de muestreo de 200 muestras por segundo, se tiene una resolución de 0,39 Hz. Con esta premisa se puede incluir dicho resultado dentro del error.

Para mejorar el entendimiento se va a mostrar una comparativa de dos funciones de respuesta en frecuencia, con idéntica velocidad de muestreo pero diferente tamaño muestral.

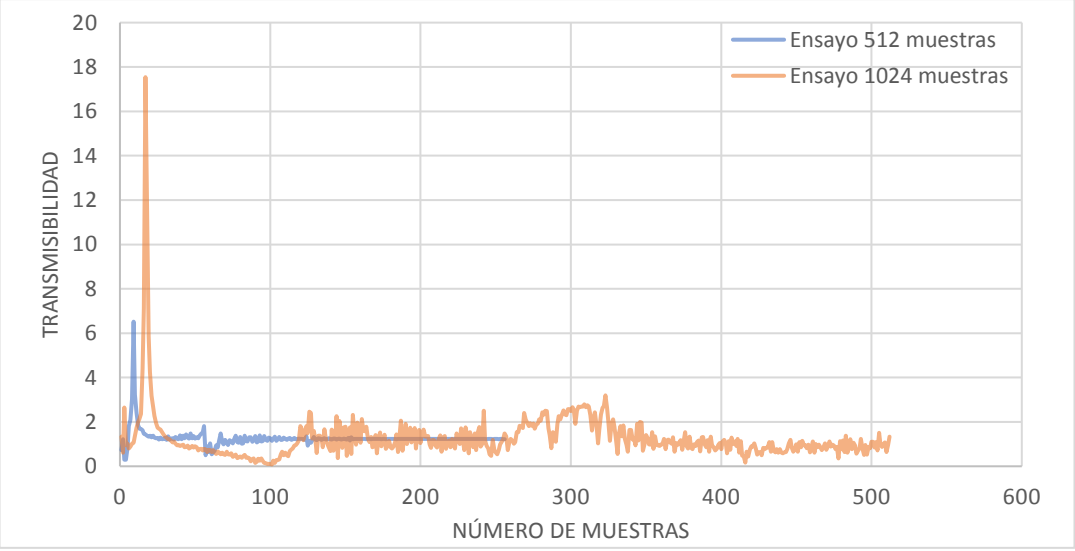


Figura 4.18. Comparación de dos FRF con una velocidad de 200 muestras por segundo.

Estas dos FRF son del ensayo con el sistema profesional SIRIUS, en azul se tiene una FRF con un tamaño muestral de 512 y en naranja otra con un tamaño

de 1024 muestras. Se puede ver una clara diferencia entre ambos máximos, ya que la función de un tamaño muestral mayor muestra una frecuencia mayor. En la obtención de la frecuencia propia no afectará ya que la posición de dicho máximo coincide con una frecuencia propia de 3,125 Hz en ambos casos.

Ahora se procederá a comparar los resultados obtenidos entre ensayos realizados con el sistema profesional y con el sistema desarrollado en el presente documento.

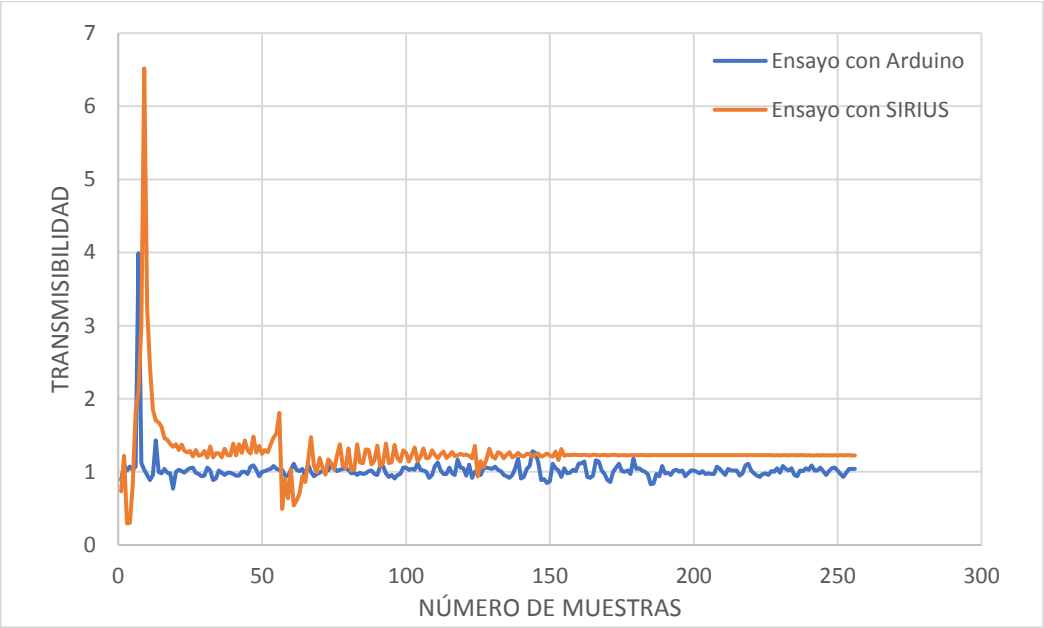


Figura 4.19. Comparación de dos FRF de idénticos parámetros entre un sistema profesional y uno de desarrollo propio.

En la figura 4.19 de color naranja la función de respuesta en frecuencia obtenida mediante SIRIUS y en azul tenemos la función de respuesta en frecuencia obtenida mediante Arduino. Ambos ensayos no se ejecutaron simultáneamente.

Como se puede ver una sutil diferencia entre ambas en la posición de la máxima frecuencia obtenida. Mientras que en la gráfica de color naranja el pico se corresponde con una frecuencia propia del sistema de ensayo de 3,125 Hz, de color azul se obtiene una frecuencia propia un poco menor concretamente 2,734 Hz.

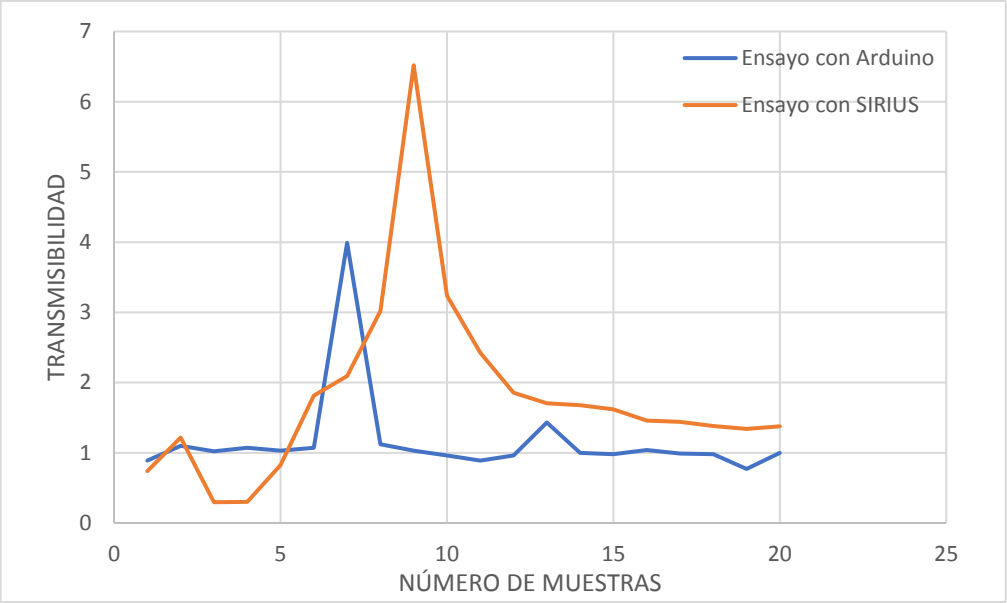


Figura 4.20. Zoom figura 4.19.

En la figura 4.20 se puede observar mejor la desviación mencionada anteriormente.

Se puede notar claramente que no es el mismo valor, sin embargo; está dentro del rango de aceptación. Ya que con esta frecuencia de muestreo y este tamaño muestral a la hora de identificar la frecuencia propia se puede tener un margen de error de 0,39 Hz. Tanto por arriba como por abajo respecto de la frecuencia propia del sistema.

Por lo que un rango aceptable de valores sería el comprendido entre 2,734 Hz - 3,515 Hz (valor de referencia tabla 4.1) y el valor extraído del ensayo con Arduino está en el límite de dicho rango.

4.4.2. Amortiguamiento.

Esta otra propiedad se puede obtener de varias maneras diferentes, en el capítulo 2 ecuación (17) se hace mención respecto una de ellas. Para el prototipo con Arduino se seleccionó (17) dada su sencillez matemática, que se traduce también en una programación mucho más fácil.

Pero en el sistema profesional esto no es un problema, por lo que en este apartado se podrá observar la diferencia entre ambos procedimientos y su fiabilidad.

El procedimiento usado con SIRIUS será el *circle-fit* [11] ,una técnica que, en lugar de trabajar con la FRF en forma de magnitud y fase, utiliza la parte real y la parte imaginaria de la misma.

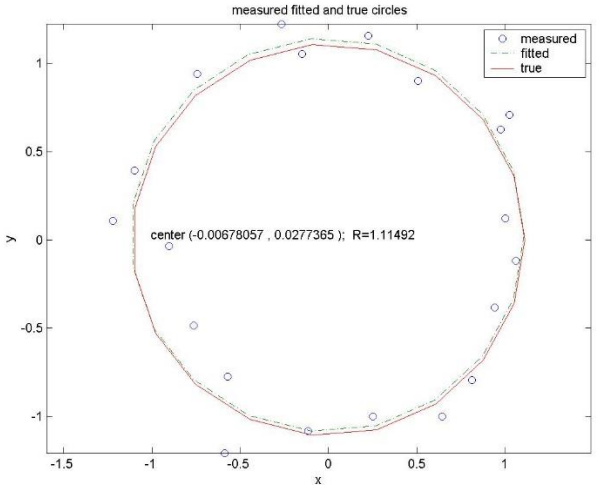


Figura 4.21. Gráfica tipo circle-fit extraída del directorio de Mathworks [11] .

En la figura 4.21 se representa la parte real en el eje horizontal y la parte imaginaria en la parte vertical.

Llevando esta técnica al terreno de la amortiguación, es necesario colocar una parte de la función de respuesta en frecuencia y no todo el espectro de dicha función. En concreto se representarán el punto de la frecuencia propia y los más cercanos al mismo.

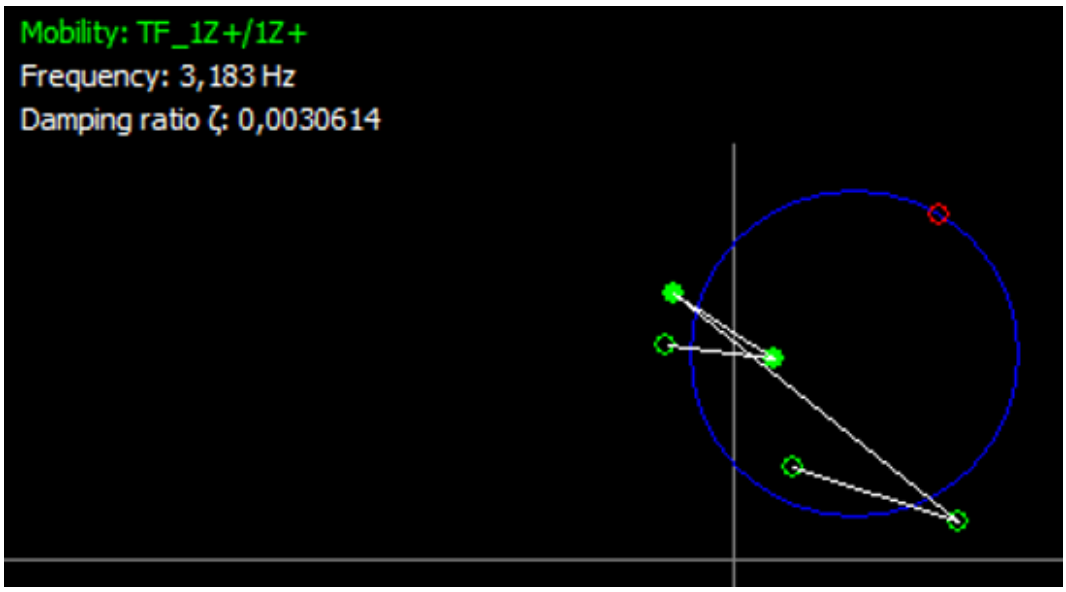


Figura 4.21. Gráfica circle-fit extraída de dewsoftx3.

En el caso de la figura 4.21 se puede ver como el programa grafica los datos y además enseña estos por pantalla de manera numérica.

Se procederá entonces a analizar dos de estas gráficas, la primera obtenida de un ensayo con un tamaño de 32768 muestras a una velocidad de 500

muestras por segundo. Que se tomará como referencia para el resto de los ensayos.

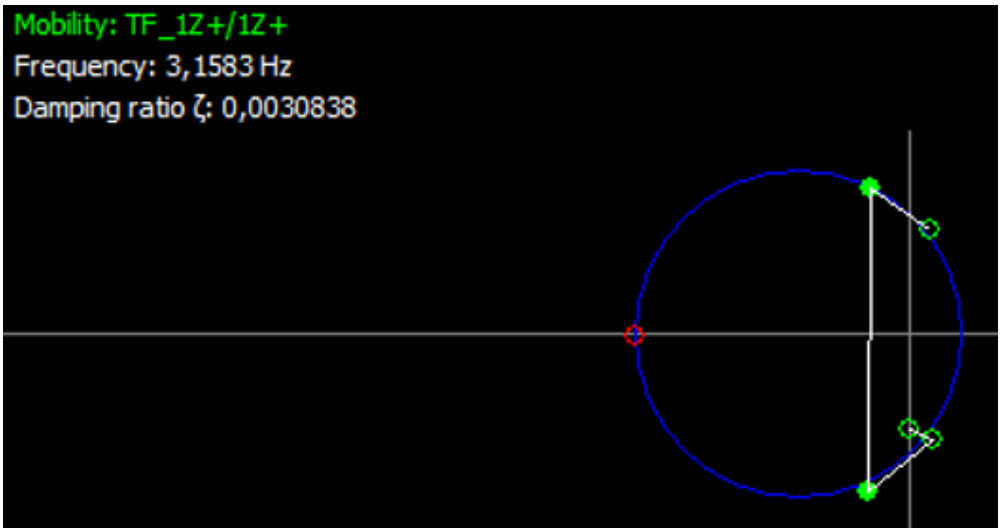


Figura 4.22. Coeficiente de amortiguamiento con el ensayo de referencia.

Este ensayo 4.22 es el tomado como referencia ya que tiene un tamaño muestral bastante considerable, además de una velocidad de muestreo muy alta. A partir de lo cual dichos valores se consideran ya lo suficientemente fiables.

En la figura 4.22 se puede ver que el coeficiente de amortiguamiento se corresponde con un valor de 0.0030838, lo que es un 0,308%.

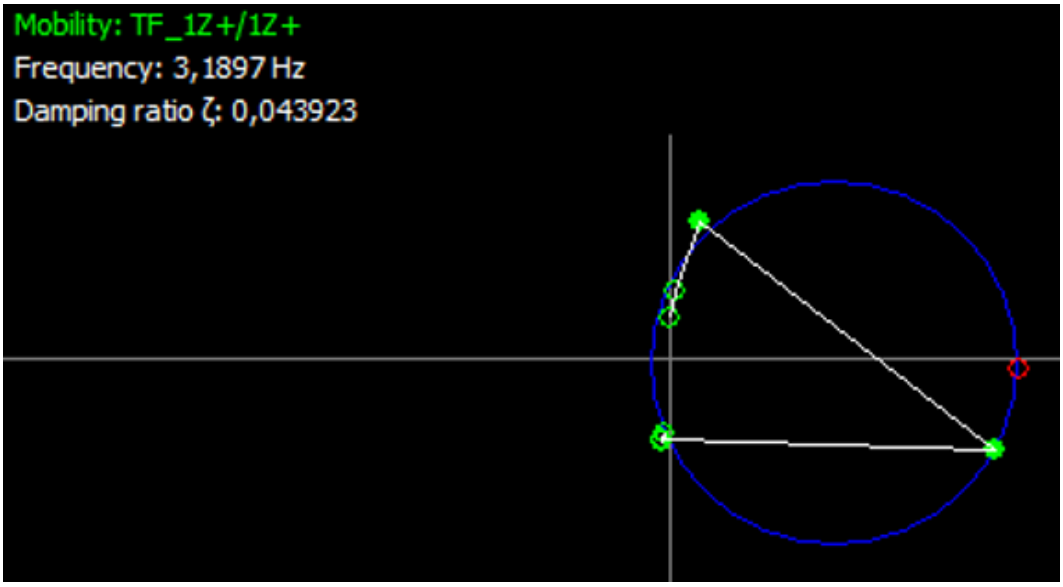


Figura 4. 23. Coeficiente de amortiguamiento ensayo con los mismos parámetros que Arduino.

En la gráfica 4.23 se representa el coeficiente de amortiguamiento extraído del ensayo que se corresponde con los mismos parámetros que Arduino. Es decir 512 muestras registradas a una velocidad de 200 muestras por segundo.

Se puede ver claramente que el coeficiente de amortiguamiento ha cambiado radicalmente, con un valor de 0.0439 es decir un 4,3%. Lo que comparando con el valor obtenido en el ensayo de referencia es aproximadamente 14 veces mayor.

Esto indica que esta técnica es buena para un tamaño muestral bastante elevado, pero para valores reducidos como es el caso de Arduino no resulta una técnica efectiva.

Se optó entonces por seguir la ecuación (17) del capítulo 2, para la cual se necesita únicamente el valor del máximo obtenido de la FRF.

En la siguiente tabla se podrá ver la diferencia con este método entre los ensayos, tanto los obtenidos con Arduino como los obtenidos con SIRIUS.

Los valores de los datos ensayados que se muestran en la tabla 4.1 corresponden a las figuras 4.9,4.10,4.14 y 4.15 respectivamente.

Valor máximo (transmisibilidad)	Coeficiente de amortiguamiento	N.º de muestras	Velocidad de muestreo	Ensayado con
26,77	0,01867	32768	500	SIRIUS
12,32	0,04058	512	200	SIRIUS
3,99	0,1253	512	200	ARDUINO
3,67	0,1362	512	100	ARDUINO

Tabla 4.2. Tabla resumen de los coeficientes de amortiguamiento obtenidos con diferentes métodos de ensayo.

Como se puede observar en la tabla 4.1 los resultados no podían ser más dispares entre ambos métodos de ensayo.

Sin embargo, se puede observar una similitud entre la pérdida de precisión del método circle-fit del ensayo número 2 de la tabla, ya que con el método circle-fit se obtenía un coeficiente de amortiguamiento de 0,043.

En cambio, en el primer elemento de la tabla 4.1 sí podemos hablar de una disparidad total entre ambos métodos. Ya que no se asemeja al valor obtenido en la figura 4.22.

Capítulo 5. Conclusiones. Líneas futuras. Valoración económica.

Respondiendo a los objetivos buscados descritos en el capítulo 1, en este capítulo se ha extraído una serie de conclusiones, las cuales se mencionarán a continuación, además de plantear posibles líneas futuras en las que pueda evolucionar la propuesta.

5.1 Conclusiones.

Uno de los objetivos propuestos en el capítulo 1 era conseguir reproducir fielmente los resultados obtenidos con un equipo profesional. Como se puede observar en la figura 4.19 los resultados obtenidos en cuanto a la obtención de la frecuencia propia son bastante fiables.

A la hora de realizar los ensayos con Arduino se tiene en cuenta un posible margen de error, en la ecuación (14) se puede ver cómo obtener la resolución de cada ensayo.

Por lo que en muestras tan pequeñas una desviación de solamente una muestra puede influir en la medida 0,39 Hz.

Si comparamos este margen de 0,39 Hz en la medida, con una frecuencia propia de 3,125 Hz (obtenida de los ensayos). Al dividir la primera entre la segunda se obtiene una posible variación de 0,125 (un 12,5%). Esta variación es bastante elevada.

En otro orden de cosas a la hora de obtener el coeficiente de amortiguamiento, el cual era otro de los objetivos de este trabajo. Se puede hablar con total rotundidad que el método empleado es ineficaz. Esto se puede corroborar en la tabla 4.2, la desviación que sufre Arduino respecto de Sirius es muy elevada.

Como conclusión general se puede afirmar que el método de bajo coste solamente funciona en una de las propiedades buscadas, (la frecuencia propia) con rango aceptable de error. Pero no se puede sustituir el sistema profesional con ello.

5.2. Líneas futuras.

Visto que el principal problema del prototipo es su capacidad de procesamiento de muestras, la solución más viable es una mejora en los equipos. Para seguir con la premisa de un equipo de bajo coste se ha seleccionado un dispositivo de desarrollo para estudiantes del fabricante National Intruments.

Figura 5.1. MyRio 1900.

Procesador	ARM Cortex-A9 MPCore (32 bits 2 GHz)
Puertos USB	2
Tensión de trabajo	5 V
Pines analógicos	68
Memoria Flash	512 MB
Puertos serie	1

Tabla 5.1. Resumen características MyRio 1900 [12].

El nombre del instrumento de mejora es MyRio 1900, dicho dispositivo tiene un coste bastante mayor que Arduino, aproximadamente 18 veces superior.

Pero sus características también son notablemente superiores, los elementos principales que son mejores son su memoria flash, la cual supera en 1024 veces a la de Arduino Due y además de su reloj interno que funciona a 2 GHz en su rango máximo de trabajo. Mientras que Arduino funcionaba a 84 MHz.

Con estas características se podría mejorar en el prototipo, tanto en velocidad de muestreo gracias a su reloj interno, como en el número de muestras adquiridas debido a su memoria flash.

Otro aspecto mejorable de dicho prototipo es los acelerómetros usados, dichos elementos funcionan bien en Arduino ya que su resolución es única y no ocupa parte de la memoria del procesador.

Esto quiere decir que los acelerómetros solamente pueden registrar en un rango, el predeterminado por el fabricante.

Pero con MyRio se pueden usar acelerómetros de resolución variable como el que se muestra a continuación:

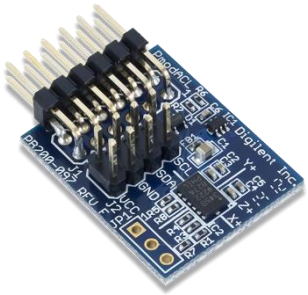


Figura 5.2. Acelerometro P-Mod ACL.

Fabricante	DIGILENT
Número de ejes	3
Resolución	Variable (0-13 bits)
Rango	± 16 g
Tensión de trabajo	3.3 V

Tabla 5.2. Características acelerómetro P-Mod ACL.

La ventaja principal de este acelerómetro respecto del usado en la fase de prototipado es la capacidad resolutiva del mismo, ya que esta la podemos ajustar.

Lo que se traduciría en una sensible mejora en los registros.

5.3. Aspecto económico.

Una de las principales premisas de este trabajo es una mejora económica respecto de un sistema profesional en este caso SIRIUS HD-STG.

Para hacer una comparación efectiva se tendrán en cuenta dos aspectos fundamentales, el precio de los instrumentos de medida y el salario medio de un ingeniero mecánico computado por sus horas de trabajo.

	Sistema profesional	Prototipo desarrollado
Precio tarjeta adquisición de datos	8.427,00 €	35,00€
Precio de los acelerometros	650,00 €	4,99 €
Precio/Horas de trabajo	16,67 €	11,20 €
Horas de trabajo	2 horas	300 horas
TOTAL	9110,34 €	3399,99 €

Tabla 5.3. Tabla resumen comparativa de ambos sistemas.

El precio de las horas de trabajo se ha computado mediante el sueldo medio de un ingeniero mecánico. En el caso del sistema profesional con la media del

sueldo de estos profesionales y en el caso del prototipo desarrollado con el de un ingeniero junior en formación.

En este caso no se incluirá el aspecto de la seguridad laboral ya que no incluye un trabajo con grandes pesos elevados, ni con peligros acerca de la red eléctrica para la cual se necesite algún tipo de formación y equipamiento especial.

Bibliografía. Webgrafía.

La bibliografía sigue la normativa ISO 690.

[1] Enrique de la fuente Tremps, Fundamentos de Dinámica Estructural, Garceta,2015.

[2] Álvaro Magdaleno, Antolin Lorenzana, et Al, A transmissibility-based procedure to estimate the modal properties of an on-board tuned mass damper, 2019.

[3] Murray R. Spiegel, Transformada de Laplace, McGraw-Hill,1996.

[4] B. Balachandran, Vibraciones. Thomson-Paraninfo, 2005.

[5] "Implementar un filtro de media móvil rápido en Arduino". [Online] https://www.luisllamas.es/arduino-filtro-media-movil/

[6] "Arduino Due". [Online]. Available: https://store.arduino.cc/arduino-due-without-headers

[7] "Acelerometro ADXL335" [Online]. Available : https://es.farnell.com/analog-devices/adxl335bcpz/ic-aceler-metro-3-ejes-3g-16lfcsp/dp/2068032?scope=partnumberlookahead&ost=ADXL335BCPZ&searchref=searchlookahead&exaMfpn=true&ddkey=https%3Aes-ES%2FElement14_Spain%2Fw%2Fsearch

[8] "Módulo bluetooth HC-08" [Online]. Available: https://es.aliexpress.com/item/32696967947.html

[9] "Universal Accelerometers", Mmf.de, 2020. [Online]. Available: https://www.mmf.de/pdf/1-1.pdf.

[10] "SIRIUS, EtherCat data acquisition system". [Online] Available: https://dewesoft.com/products/daq-systems/sirius`

[11] "Circle–fit". [Online] https://www.mathworks.com/matlabcentral/fileexchange/5557-circle-fit

[12] "MyRio 1900 National Instrument "[Online] Available: https://www.ni.com/es-es/support/model.myrio-1900.html

Anexo A. Guía básica IDE de Arduino.

Para conseguir que el proyecto se lleve a cabo, se utiliza la herramienta Arduino, la cual es un microcontrolador electrónico que permitirá cargar en un programa ya automático el programa que servirá para el propósito planteado en el capítulo 2.

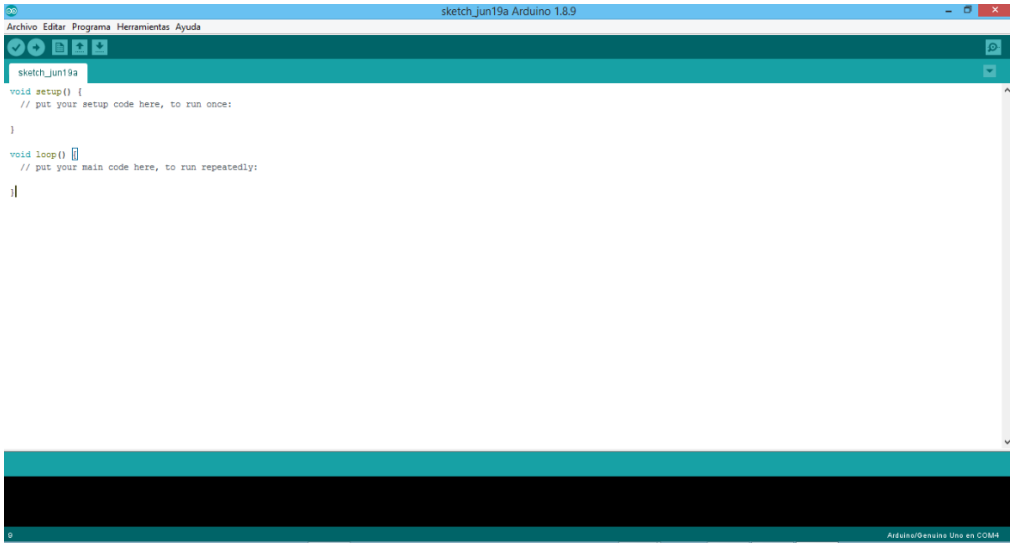


Figura A.1. Pantalla de inicio de la IDE de Arduino

El manejo del programa es bastante intuitivo desde el inicio puesto que se compone de 5 menús que al hacer clic se desarrollan en cascada los submenús correspondientes.

A continuación, se describirán de forma breve cada uno de los 5 menús y su utilidad para el proyecto:

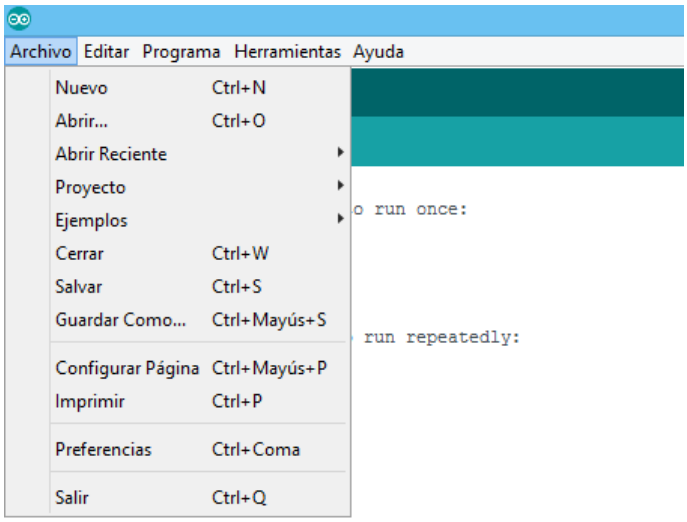


Figura A.2 Desarrollo en cascada del menú archivo

El desarrollo del menú archivo es el clásico de cualquier programa, el cual incluye funciones básicas como *nuevo, abrir, cerrar, guardar, configurar página, imprimir, preferencias y salir.*

Estas funciones son las básicas de cualquier programa ya que permiten un entorno de comunicación con el ordenador para poder almacenar los programas y acceder a ellos, además de poder exportar la información.

Además, tenemos unas funciones propias como son *abrir reciente*, la cual permite acceder a los últimos proyectos abiertos dentro de la IDE. También tenemos *Proyecto* que permite abrir un conjunto de programas de Arduino que forman un proyecto de Arduino. Por último, está *ejemplos*, el submenú el cual despliega en cascada una serie de programas para el aprendizaje de Arduino, a la hora de empezar con este tipo de entornos de desarrollo es útil puesto que enseña la construcción de funciones básicas.

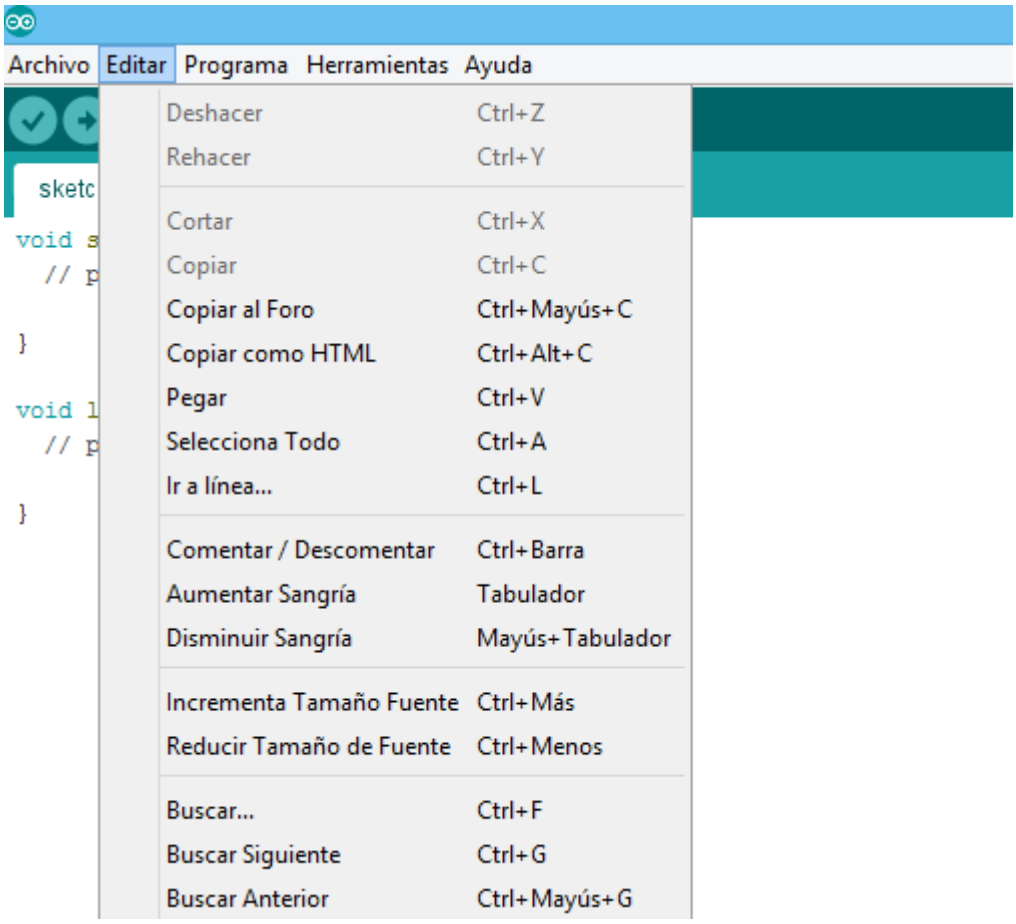


Figura A.3. Desarrollo en cascada del menú editar

Como se puede observar este menú sigue la misma dinámica del menú anterior teniendo sus funciones básicas de cualquier programa como son: *Deshacer, Rehacer,Cortar,Copiar,Pegar,Seleccionar Todo y Buscar.*

Luego tiene unos comandos más particulares como son: *Copiar al Foro*, el cual conecta directamente el código utilizado con el foro de ayuda para los internautas de Arduino, en el que se puede formular dudas y responderlas. También se puede *Copiar como HTML*, que como el propio nombre del comando indica copia en ese formato.

El comando *Ir a línea...* es bastante útil para solucionar fallos de programación puesto que nos envía directamente a la línea de código a la que queremos acceder.

Las siguientes funciones son de utilidad para el propio programador, ya que por ejemplo *Comentar/Descomentar* permite inhabilitar/habilitar líneas de código en el programa, que sirven por ejemplo para realizar anotaciones en lenguaje común, de tal manera que el programador se sienta más cómodo en el entorno.

Los comandos *Aumentar Sangría* y *Disminuir Sangría*, tienen el mismo propósito que el comando anterior, ya que estos comandos permiten iniciar la escritura de código con sangría o disminuir la misma, esto no afecta al lenguaje de programación; pero si puede ser de utilidad para esclarecer más el programa. Al igual que los dos comandos anteriores *Incrementar Tamaño Fuente* y *Disminuir Tamaño Fuente*, tienen el mismo objetivo, al poder aumentar o disminuir el tamaño de la fuente utilizada en el código.

Por último, *Buscar Siguiente* y *Buscar Anterior*, se complementan con el comando Buscar, al poder realizar una búsqueda múltiple por el código si se diera la situación, ya que permite ir al término siguiente y anterior de la búsqueda en el cual nos encontramos.

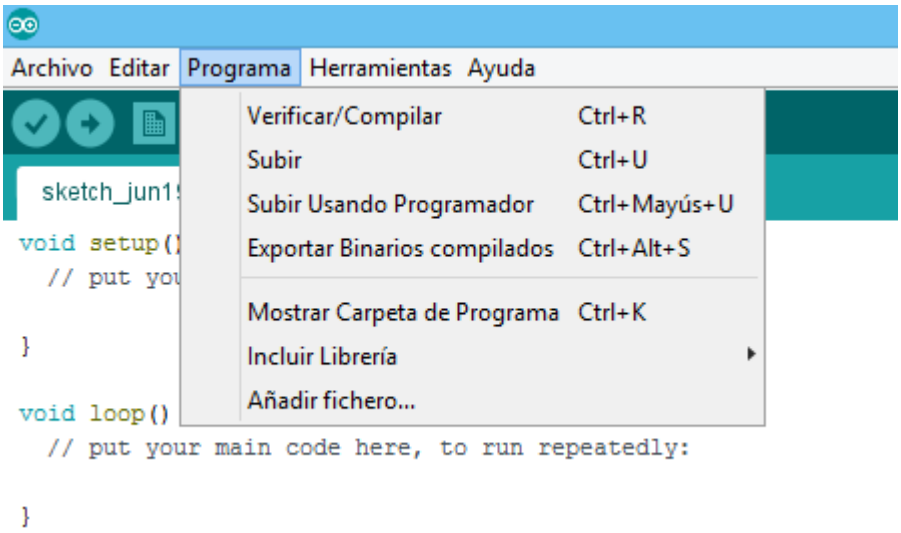


Figura A.4 Desarrollo en cascada del menú programa

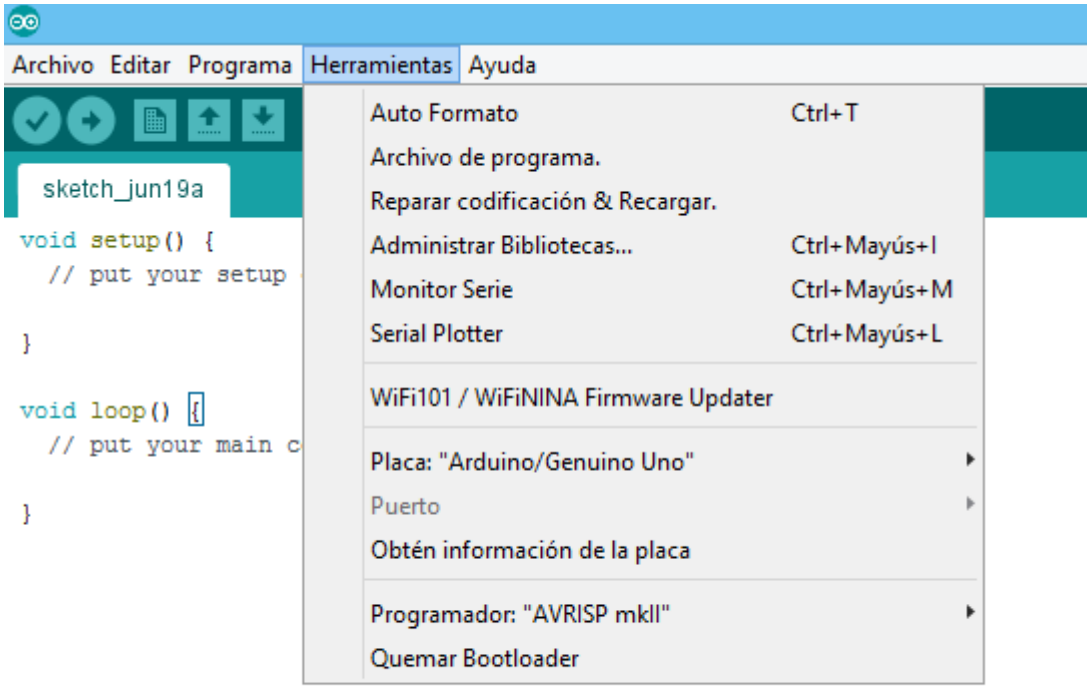
Este menú ya es más particular del propio entorno al tener comandos más propios de una herramienta de desarrolladores.

El comando *Verificar/Compilar* tiene la utilidad de que el programa revisa todo el texto escrito y notifica si hubiera algún error del tipo sintaxis de programación. Este es la parte inicial del comando *Subir*, ya que este ejecuta primero *Verificar/Compilar* antes de cargar el programa a la placa de Arduino usada.

Los comandos *Subir Usando Programador*, *Exportar Binarios compilados, Mostrar Carpeta de programa y Añadir fichero* no se han usado en el presente proyecto, por tanto, no se va a detallar su uso.

Uno de bastante utilidad e importancia para el presente proyecto es *Incluir Librería,* este comando es muy útil para realizar programas de ligera complicación, puesto que nos permite acceder a un tipo de archivos llamados librerías los cuales tienen unas funciones muy específicas, que de ser programadas a mano serían una tarea muy tediosa; pero al haberlas realizado otro programador previamente nos podemos aprovechar de su trabajo y ahorrar tiempo y esfuerzo para centrarnos en otros aspectos del programa.

Este tipo de archivos es de obligada necesidad que estén almacenados en el mismo ordenador con el que se programa, puesto que la IDE necesita acceder a los mismos en cuanto una librería es llamada en la programación.



FiguraA.5 Desarrollo en cascada del menú herramientas

El comando *Auto Formato* es muy útil para el programador puesto que formatea el código para que sea legible, para programas de muchas líneas de código es muy útil.

Archivos de programa, permite comprimir todos los archivos del proyecto en formato zip.

El comando *Monitor Serie* simplemente es un monitor que nos permite comunicarnos con el Arduino, este tiene que estar programado previamente para esta comunicación, pero es muy útil en ese aspecto, además de mostrarnos por pantalla en el monitor los datos que están programados en el mismo que se desean mostrar.

Serial Plotter al igual que el *Monitor Serie* nos muestra los datos por pantalla del programa, pero este la muestra de forma gráfica, lo cual es bastante útil en muchas aplicaciones, ya que la información visual es mucho más sencilla de procesar que la numérica.

Por último, el comando *Placa:""*, dicho comando permite seleccionar la placa de trabajo del proyecto, y establecer la comunicación con la misma, este paso es importante ya que no todas las placas tienen los mismos protocolos de comunicación. En el caso de este proyecto la placa Arduino Due no aparece directamente en el listado de placas disponibles, sino que es necesario descargar del repositorio de Arduino una serie de protocolos especiales de comunicación.

Anexo B. Calibración de los acelerómetros.

La necesidad de calibrar sendos acelerómetros es importante, ya que estos al registrar los datos lo hacen en un espectro totalmente diferente al sistema internacional o similar; es decir registra datos entre dos valores, un máximo y un mínimo, que en el caso del Arduino son las resoluciones digitales del mismo (12 bits en Arduino Due).

Lo que lleva a la necesidad de trasladar unos registros comprendidos entre 0 y 4095, a valores con los que se puedan trabajar, en este caso unidades del sistema internacional de aceleración [m/s^2].

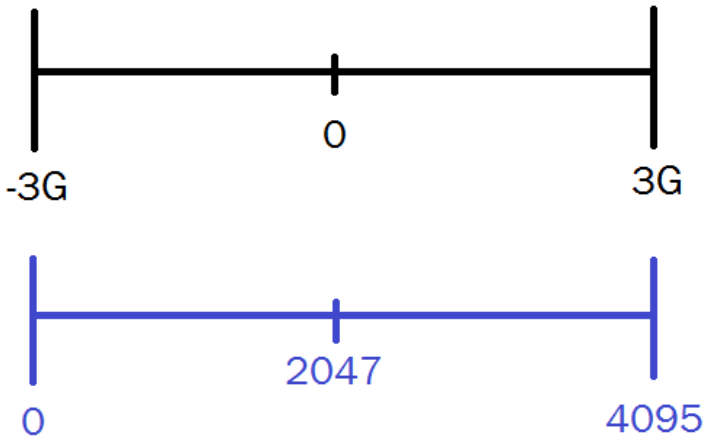


Figura B.1 Diagrama visual de como registra muestras Arduino

Como se puede observar en la imagen B1, el rango del acelerómetro según el fabricante es desde -3G hasta 3G, pero como se ha mencionado antes el Arduino recibe la información de dicho rango en valores 0-4095.

Ahora bien, sabiendo realmente como es la lectura de dichos valores, si se hace una división de ambas gráficas, se puede extraer que 1G equivale a 682.66. Lo que habría que confirmar con un sencillo experimento.

Para ello se experimentó con ambos acelerómetros y en los 3 ejes que tiene cada uno.

Para ejecutar la calibración se usó una medida ya conocida, la cual es la aceleración de la gravedad, con esta medida lo único que se hizo es colocar cada uno de los ejes (alternando su sentido) en la misma perpendicular al suelo, se ejecutaron varias medidas y se hizo la media. A continuación, una tabla con dichas medidas:

Eje	Acelerómetro 1	valor
x	2399,95 mV	1G
y	2453,79 mV	1G
z	1664,51 mV	-1G

Tabla B1. Medidas calibración del acelerómetro 1.

Eje	Acelerómetro 2	valor
x	2400,91 mV	1G
y	1610,12 mV	-1G
z	2447,14 mV	1G

Tabla B2. Medidas calibración del acelerómetro 2.

Con estas medidas, lo que se hizo fue restar en valor absoluto la medida correspondiente a una aceleración nula, que son 2047 mV, y como último paso realizar la media muestral de las mismas.

Como se puede ver en la fase experimental, la primera conclusión que se extrajo es falsa.

Con la calibración se pudo cuantificar realmente cual era el valor de 1G que lee el Arduino, y para cada acelerómetro es ligeramente diferente.

-Acelerómetro 1: 381.41 mV/G.

-Acelerómetro 2: 381.31 mV/G.

A posteriori sólo hubo que incluir dos líneas de código descritas en el capítulo 3 del presente documento.

Anexo C. Código del programa.

En este anexo se asociarán los bloques del flujograma del capítulo 3.6.9 a las líneas de código correspondientes y además se adjuntarán todas las líneas de código generadas en el programa.

Anexo C.1. Asociación de los bloques del flujograma con sus líneas.

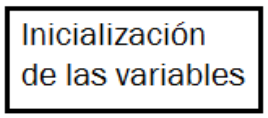


Figura C.1.Bloque 1.

```
11. #include <arduinoFFT.h>
12. #include <MeanFilterLib.h>
13.
14.
15. /*Programa para leer datos de un acelerometro*/
16.
17. const int xpin = A5;
18. const int xpin2 = A6;
19.
20. /*Estas declaraciones son necesarias para nuestra fft*/
21. /*Estas variables son las que podemos ir cambiando para ajustar la
    fft*/
22. /*Las dejamos parametrizadas para que podamos cambiarlas a
    voluntad*/
23. const uint16_t samples = 1024;
24. int frecMuest = 200;
25.
26.
27. /*Estas variables son importantes para diferenciar entre la parte
    real del numero y la parte imaginaria*/
28.
29. /* Datos de calibración */
30. const float cal1 = 380.41/9.806; // mV/g / 9.806 m/s2/g
31. const float cal2 = 380.31/9.806; // mV/g / 9.806 m/s2/g
32.
33. /*A cada paquete le corresponden los datos de cada acelerometro*/
34. float primerpaquete[samples/2];
35. float acel2primerpaquete[samples/2];
36. /*Primer paquete es el acelerometro de arriba y acel2 es el de
    abajo*/
37. float segundopaquete[samples/2];
38. float acel2segundopaquete[samples/2];
39. float tercerpaquete[samples/2];
40. float acel2tercerpaquete[samples/2];
41. /*Ponemos otro parametro chi que va a ser la coordenada relativa,
    tambien tiene que llevar su parte imaginaria*/
42. /*Estas se quedan declaradas con los los valores que nos
    interesan*/
43.
44. float chiprimero[samples/2];
45. float chiprimeroimag[samples/2];
46. float chisegundo[samples/2];
47. float chisegundoimag[samples/2];
48. float chitercero[samples/2];
49. float chiterceroimag[samples/2];
50. /* Esta variable es la del filtro de media movil*/
```

```
/*La media del filtro tambien queda declarada con mas datos ya que necesito procesarlos todos*/
/*Propiedades filtro*/
const short venFil = 3;
/*Las siguientes variables se declaran así por que necesitamos eliminar el pico del principio del acelerometro*/
float mean[samples+venFil+1];
float mean2[samples+venFil+1];


float primerpaqueteimag[samples/2];
float acel2primerpaqueteimag[samples/2];
float segundopaqueteimag[samples/2];
float acel2segundopaqueteimag[samples/2];
float tercerpaqueteimag[samples/2];
float acel2tercerpaqueteimag[samples/2];

/* Vamos a declarar ahora las variables necesarias para procesar H1 lo dejamos en la mitad muestral por que es lo que nos interesa*/
float NumR[samples/4];
float NumI[samples/4];
float DenR[samples/4];
float DenI[samples/4];

float NumprimeroR[samples/2];
float NumprimeroI[samples/2];
float DenprimeroR[samples/2];
float DenprimeroI[samples/2];
float NumsegundoR[samples/2];
float NumsegundoI[samples/2];
float DensegundoR[samples/2];
float DensegundoI[samples/2];
float NumterceroR[samples/2];
float NumterceroI[samples/2];
float DenterceroR[samples/2];
float DenterceroI[samples/2];

float Hreal[samples/4];
float Himag[samples/4];

float Hmagnitud[samples/4];
float Hfase[samples/4];

/*Voy a declarar tambien dos valores para buscar un maximo en un vector y su posicion*/
float maximo;
int posicion;
/*Para conseguir mandar la info por bluetooth tenemos que partir los numeros*/
int b1;
int b2;
int c1;
int c2;
char valor;
      long numerote;
      long numerote2;
      float fo;
      float amortiguamiento;

      float promedio;
      float suma;
```

```
107.      float suma2;
108.      float promedio2;
109.      MeanFilter<float> meanFilter(venFil);
110.      MeanFilter<float> meanFilter2(venFil);
111.
112.      /*Definicion de las variables del segundo acelerometro
113.      /*Definicion del tiempo de muestreo */
114.      void setup() {
115.          Serial.begin(115200);
116.          Serial1.begin(9600);//Probar a configurar con otras
   frecuencias
117.          /*Estoy funcionando ahora en 115200 lo que me da una
   comunicacion rápida con el puerto serie */
118.          pinMode(xpin,INPUT);
119.          pinMode(xpin2,INPUT);
120.          /*Se define ahora la inicialización de los pines del
   acelerometro*/
121.          analogReadResolution(12);
122.          /*Colocamos esta variable para poder leer valores en un
   mayor rango*/
123.      }
```

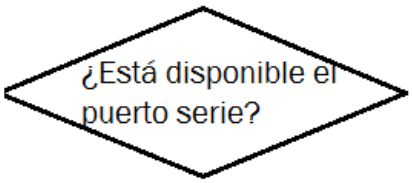


Figura C.2.Bloque 2.

```
1.
2.  void loop() {
3.
4.      if(Serial1.available ())//Si el puerto del bluetooth esta
   disponible
5.      {
```

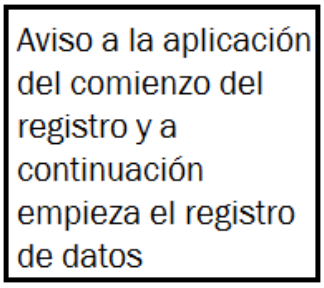


Figura C.3.Bloque 3.

```
1.          valor = Serial1.read ();
2.          if(valor== 'a') {
3.              Serial1.write(2);
4.              delay(100);
5.              /* Registro el paquete de datos de todo el samples*/
6.
7.              for (uint16_t i = 0; i < (samples+venFil+1); i++) {
8.                  mean[i] = analogRead(xpin);
9.                  mean2[i] = analogRead(xpin2);
```

```
                delay(1000/frecMuest);
            }
```

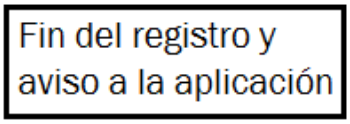


Figura C.4.Bloque 4.

```
            Serial1.write(3);
            delay(100);
```

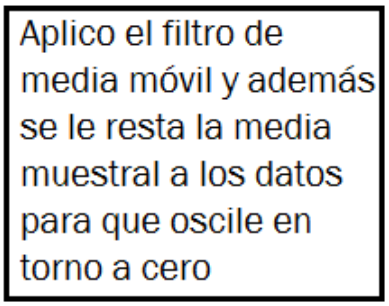


Figura C.5.Bloque 5.

```
            /*Aqui se aplica el filtro de media móvil de dichos
   datos registrados*/

            for (uint16_t i = 0; i < samples+venFil+1; i++) {
                mean[i] = meanFilter.AddValue(mean[i]);
                mean2[i] = meanFilter2.AddValue(mean2[i]);
            }

            /*Normalizamos todos los vectores, quitando los
   primeros venFil+1 valores*/
            for(uint16_t
   i = venFil+1; i < samples+venFil+1; i++) {
                mean[i-venFil-1] = mean[i];
                mean2[i-venFil-1] = mean2[i];
            }
             suma = 0.0;
             suma2 = 0.0;
            /*Voy a realizar el promedio de la medicion para luego
   restarlo y eliminar los datos no representativos*/
            for (uint16_t i = 0; i < samples; i++) {
                suma = suma + mean[i];
                suma2 = suma2 + mean2[i];
            }

            promedio = suma/samples;
            promedio2 = suma2/samples;

            /*A continuacion restituyo los valores de mean
   quitandole la media para evitar que oscile en torno a 0*/
            for (uint16_t i = 0; i < samples; i++) {
                mean[i] = mean[i] - promedio;
                mean2[i] = mean2[i] - promedio2;
            }
```

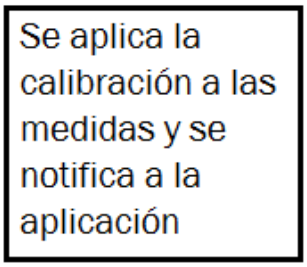


Figura C.6.Bloque 6.

```
            /* CALIBRACIÓN */
            for (uint16_t i = 0; i < samples; i++) {
                mean[i] = mean[i]/cal1;
                mean2[i] = mean2[i]/cal2;
            }

            Serial1.write(4);
            delay(100);

```

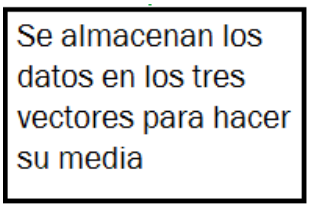


Figura C.7.Bloque 7.

```
            /*primerpaquete pertenece al valor del acelerometro
   del piso superior medido*/
            /*2primerpaquete pertenece al valor del acelerometro
   del piso inferior medido*/
            for (uint16_t i = 0 ; i < samples/2; i++){
                primerpaquete[i] = mean[i];
                acel2primerpaquete[i] = mean2[i];
            }

            /*segundopaquete pertenece al valor del acelerometro
   del piso superior medido*/
            /*2segundopaquete pertenece al valor del acelerometro
   del piso inferior medido*/
            for (uint16_t i = samples/2 ; i < samples; i++){
                segundopaquete[i-samples/2] = mean[i];
                acel2segundopaquete[i-samples/2] = mean2[i];
            }

            /*tercerpaquete pertenece al valor del acelerometro
   del piso superior medido*/
            /*2tercerpaquete pertenece al valor del acelerometro
   del piso inferior medido*/
            for (uint16_t
   i = samples/4 ; i < 3*(samples/4) ; i++){
                tercerpaquete[i-samples/4] = mean[i];
                acel2tercerpaquete[i-
   samples/4] = mean2[i];
```

```
20.             }
21.             Serial1.write(5);
22.             delay(100);
```

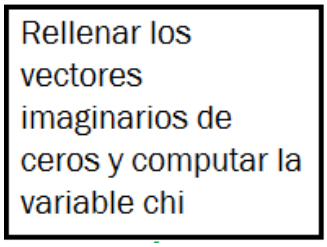


Figura C.8.Bloque 8.

```
1.              /*Necesito llenar de 0 mi vector por que ahora no
    tendre parte imaginaria en el registro*/
2.              for (uint16_t i = 0; i < samples/2; i++) {
3.                  chiprimeroimag[i] = 0.0;
4.                  chisegundoimag[i] = 0.0;
5.                  chiterceroimag[i] = 0.0;
6.                  primerpaqueteimag[i] = 0.0;
7.                  acel2primerpaqueteimag[i] = 0.0;
8.                  segundopaqueteimag[i] = 0.0;
9.                  acel2segundopaqueteimag[i] = 0.0;
10.                 tercerpaqueteimag[i] = 0.0;
11.                 acel2tercerpaqueteimag[i] = 0.0;
12.             }
13.
14.             /*Aqui se calcula el chi de los datos registrados*/
15.             for (uint16_t i = 0; i < samples/2; i++) {
16.                 chiprimero[i] = primerpaquete[i] - acel2primerpaqu
    ete[i];
17.                 chisegundo[i] = segundopaquete[i] - acel2segundopa
    quete[i];
18.                 chitercero[i] = tercerpaquete[i] - acel2tercerpaqu
    ete[i];
19.             }
```

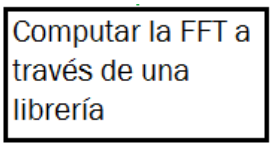


Figura C.9. Bloque 9.

```
1.
2.              /*Para que funcione bien la FFT tenemos que declarar
    aqui la libreria*/
3.              arduinoFFT
    FFTprimerpaquete = arduinoFFT(acel2primerpaquete,acel2primerpaquet
    eimag,samples/2,frecMuest);
```

```
4.              arduinoFFT
    FFTchiprimero = arduinoFFT(chiprimero,chiprimeroimag,samples/2,fre
    cMuest);
5.
6.              arduinoFFT
    FFTsegundopaquete = arduinoFFT(acel2segundopaquete,acel2segundopaq
    ueteimag,samples/2,frecMuest);
7.              arduinoFFT
    FFTchisegundo = arduinoFFT(chisegundo,chisegundoimag,samples/2,fre
    cMuest);
8.
9.              arduinoFFT
    FFTtercerpaquete = arduinoFFT(acel2tercerpaquete,acel2tercerpaquet
    eimag,samples/2,frecMuest);
10.             arduinoFFT
    FFTchitercero = arduinoFFT(chitercero,chiterceroimag,samples/2,fre
    cMuest);
11.
12.             FFTprimerpaquete.Windowing(FFT_WIN_TYP_HANN, FFT_FORWA
    RD);
13.             FFTchiprimero.Windowing(FFT_WIN_TYP_HANN, FFT_FORWARD)
    ;
14.
15.             FFTsegundopaquete.Windowing(FFT_WIN_TYP_HANN, FFT_FORW
    ARD);
16.             FFTchisegundo.Windowing(FFT_WIN_TYP_HANN, FFT_FORWARD)
    ;
17.
18.             FFTtercerpaquete.Windowing(FFT_WIN_TYP_HANN, FFT_FORWA
    RD);
19.             FFTchitercero.Windowing(FFT_WIN_TYP_HANN, FFT_FORWARD)
    ;
20.
21.             FFTprimerpaquete.Compute(FFT_FORWARD);
22.             FFTchiprimero.Compute(FFT_FORWARD);
23.
24.             FFTsegundopaquete.Compute(FFT_FORWARD);
25.             FFTchisegundo.Compute(FFT_FORWARD);
26.
27.             FFTtercerpaquete.Compute(FFT_FORWARD);
28.             FFTchitercero.Compute(FFT_FORWARD);
```

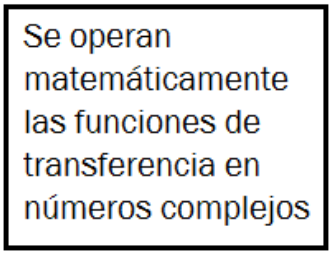


Figura C.10. Bloque 10.

```
1.              for (uint16_t i =0; i < samples/2; i++) {
2.                NumprimeroR[i] = (chiprimero[i]*acel2primerpaquete[i
    ] + chiprimeroimag[i]*acel2primerpaqueteimag[i]);
3.                NumprimeroI[i] = (chiprimeroimag[i]*acel2primerpaque
    te[i] - chiprimero[i]*acel2primerpaqueteimag[i]);
```

```
4.                DenprimeroR[i] = (acel2primerpaquete[i]*acel2primerp
   aquete[i] + acel2primerpaqueteimag[i]*acel2primerpaqueteimag[i]);
5.                DenprimeroI[i] = 0.0;
6.
7.                NumsegundoR[i] = (chisegundo[i]*acel2segundopaquete[
   i] + chisegundoimag[i]*acel2segundopaqueteimag[i]);
8.                NumsegundoI[i] = (chisegundoimag[i]*acel2segundopaqu
   ete[i] - chisegundo[i]*acel2segundopaqueteimag[i]);
9.                DensegundoR[i] = (acel2segundopaquete[i]*acel2segund
   opaquete[i] + acel2segundopaqueteimag[i]*acel2segundopaqueteimag[i
   ]);
10.               DensegundoI[i] = 0.0;
11.
12.               NumterceroR[i] = (chitercero[i]*acel2tercerpaquete[i
   ] + chiterceroimag[i]*acel2tercerpaqueteimag[i]);
13.               NumterceroI[i] = (chiterceroimag[i]*acel2tercerpaque
   te[i] - chitercero[i]*acel2tercerpaqueteimag[i]);
14.               DenterceroR[i] = (acel2tercerpaquete[i]*acel2tercerp
   aquete[i] + acel2tercerpaqueteimag[i]*acel2tercerpaqueteimag[i]);
15.               DenterceroI[i] = 0.0;
16.             }
17.
```

Se hace la media aritmética de las funciones y se transforman en magnitud y fase

Figura C.11. Bloque 11.

```
1.              /*Ahora hago la media aritmetica de los numeros
   complejos en la forma binomial*/
2.              for (uint16_t i =0; i < samples/4; i++) {
3.                NumR[i] = (NumprimeroR[i]+NumsegundoR[i]+NumterceroR
   [i])/3.0;
4.                NumI[i] = (NumprimeroI[i]+NumsegundoI[i]+NumterceroI
   [i])/3.0;
5.                DenR[i] = (DenprimeroR[i]+DensegundoR[i]+DenterceroR
   [i])/3.0;
6.                DenI[i] = (DenprimeroI[i]+DensegundoI[i]+DenterceroI
   [i])/3.0;
7.              }
8.
9.
10.
11.             /*Aqui por ultimo calculo la H en forma binomial
   tambien*/
12.             for (uint16_t i =0; i < samples/4; i++) {
13.               Hreal[i] =(NumR[i]*DenR[i]+DenI[i]*NumI[i])/(DenR[i]
   *DenR[i]+DenI[i]*DenI[i]);
14.               Himag[i] =(NumI[i]*DenR[i]-
   NumR[i]*DenI[i])/(DenR[i]*DenR[i]+DenI[i]*DenI[i]);
15.             }
16.
```

```
17.
18.             /*Con la siguiente operacion convierto la H en
    magnitud*/
19.             for (uint16_t i =0; i < samples/4; i++) {
20.                Hmagnitud[i] = sqrt(sq(Hreal[i]) + sq(Himag[i]));
21.             }
22.
23.
24.             for (uint16_t i = 0; i < samples/4; i++) {
25.                 Serial.println(Hmagnitud[i]);
26.                 /*Serial.print(",");
27.                 Serial.println(mean2[i]);*/
28.             }
```

Por último se halla la frecuencia propia y el coeficiente de amortiguamiento con las funciones matemáticas correspondientes, y se comunica dicha información via bluetooth

Figura C.12. Bloque 12.

```
1.              /*Este bucle for lo que me hara es recorrerme el
    vector y almacenarme el máximo valor y su posicion dentro del
    mismo vector*/
2.              maximo = 0.0;
3.              for (uint16_t i = 0; i < 256; i++) {
4.                  if (Hmagnitud[i] > maximo) {
5.                      maximo = Hmagnitud[i];
6.                      posicion = i;
7.                  }
8.              }
9.
10.           /*Con esto hago las cuentas finales para que pueda
    mandar a la aplicación móvil mis datos*/
11.           fo = posicion*(2*frecMuest/samples);
12.           amortiguamiento = 1/(2*maximo);
13.
14.           if (fo < 1000) {
15.               numerote = fo*100;
16.
17.               b1= numerote/256;
18.               b2= numerote - b1*256;
19.           } else {
20.               numerote = fo*10;
21.               b1=numerote/256;
22.               b2= numerote - b1*256;
23.           }
24.
25.           if (amortiguamiento < 1000) {
26.               numerote2 = amortiguamiento*100;
27.
28.               c1 = numerote2/256;
29.               c2 = numerote2 - b1*256;
30.           } else {
31.               numerote2 = amortiguamiento*10;
32.               c1 = numerote2/256;
33.               c2 = numerote2 - b1*256;
34.           }
35.
```

```
            Serial1.write(12);
            Serial1.write(1);
            Serial1.write(330);
            delay(100);
            Serial1.write(13);
            Serial1.write(c1);
            Serial1.write(88);
            delay(100);
            Serial1.write(6);

            }
      }
}
```

Anexo C.2. Código generado en Arduino.

```
#include <arduinoFFT.h>
#include <MeanFilterLib.h>


/*Programa para leer datos de un acelerometro*/

const int xpin = A5;
const int xpin2 = A6;

/*Estas declaraciones son necesarias para nuestra fft*/
/*Estas variables son las que podemos ir cambiando para ajustar la fft*/
/*Las dejamos parametrizadas para que podamos cambiarlas a voluntad*/
const uint16_t samples = 1024;
int frecMuest = 200;


/*Estas variables son importantes para diferenciar entre la parte real del numero y la parte imaginaria*/

/* Datos de calibración */
const float cal1 = 380.41/9.806; // mV/g / 9.806 m/s2/g
const float cal2 = 380.31/9.806; // mV/g / 9.806 m/s2/g

/*A cada paquete le corresponden los datos de cada acelerometro*/
float primerpaquete[samples/2];
float acel2primerpaquete[samples/2];
/*Primer paquete es el acelerometro de arriba y acel2 es el de abajo*/
float segundopaquete[samples/2];
float acel2segundopaquete[samples/2];
float tercerpaquete[samples/2];
float acel2tercerpaquete[samples/2];
/*Ponemos otro parametro chi que va a ser la coordenada relativa, tambien tiene que llevar su parte imaginaria*/
/*Estas se quedan declaradas con los los valores que nos interesan*/

float chiprimero[samples/2];
float chiprimeroimag[samples/2];
float chisegundo[samples/2];
float chisegundoimag[samples/2];
float chitercero[samples/2];
float chiterceroimag[samples/2];
```

```
/* Esta variable es la del filtro de media movil*/
/*La media del filtro tambien queda declarada con mas datos ya que necesito procesarlos todos*/
/*Propiedades filtro*/
const short venFil = 3;
/*Las siguientes variables se declaran así por que necesitamos eliminar el pico del principio del acelerometro*/
float mean[samples+venFil+1];
float mean2[samples+venFil+1];


float primerpaqueteimag[samples/2];
float acel2primerpaqueteimag[samples/2];
float segundopaqueteimag[samples/2];
      float acel2segundopaqueteimag[samples/2];
      float tercerpaqueteimag[samples/2];
      float acel2tercerpaqueteimag[samples/2];

      /* Vamos a declarar ahora las variables necesarias para procesar H1 lo dejamos en la mitad muestral por que es lo que nos interesa*/
      float NumR[samples/4];
      float NumI[samples/4];
      float DenR[samples/4];
      float DenI[samples/4];

      float NumprimeroR[samples/2];
      float NumprimeroI[samples/2];
      float DenprimeroR[samples/2];
      float DenprimeroI[samples/2];
      float NumsegundoR[samples/2];
      float NumsegundoI[samples/2];
      float DensegundoR[samples/2];
      float DensegundoI[samples/2];
      float NumterceroR[samples/2];
      float NumterceroI[samples/2];
      float DenterceroR[samples/2];
      float DenterceroI[samples/2];

      float Hreal[samples/4];
      float Himag[samples/4];

      float Hmagnitud[samples/4];
      float Hfase[samples/4];

      /*Voy a declarar tambien dos valores para buscar un maximo en un vector y su posicion*/
      float maximo;
      int posicion;
      /*Para conseguir mandar la info por bluetooth tenemos que partir los numeros*/
      int b1;
      int b2;
      int c1;
      int c2;
      char valor;
      long numerote;
      long numerote2;
      float fo;
      float amortiguamiento;

      float promedio;
      float suma;
```

```
145.      float suma2;
146.      float promedio2;
147.      MeanFilter<float> meanFilter(venFil);
148.      MeanFilter<float> meanFilter2(venFil);
149.
150.      /*Definicion de las variables del segundo acelerometro
151.      /*Definicion del tiempo de muestreo */
152.      void setup() {
153.          Serial.begin(115200);
154.          Serial1.begin(9600);//Probar a configurar con otras
   frecuencias
155.          /*Estoy funcionando ahora en 115200 lo que me da una
   comunicacion rápida con el puerto serie */
156.          pinMode(xpin,INPUT);
157.          pinMode(xpin2,INPUT);
158.          /*Se define ahora la inicialización de los pines del
   acelerometro*/
159.          analogReadResolution(12);
160.          /*Colocamos esta variable para poder leer valores en un
   mayor rango*/
161.      }
162.
163.      void loop() {
164.
165.          if(Serial1.available ())//Si el puerto del bluetooth
   esta disponible
166.          {
167.              valor = Serial1.read ();
168.              if(valor== 'a') {
169.                  Serial1.write(2);
170.                  delay(100);
171.                  /* Registro el paquete de datos de todo el
   samples*/
172.
173.                  for (uint16_t
   i = 0; i < (samples+venFil+1); i++) {
174.                      mean[i] = analogRead(xpin);
175.                      mean2[i] = analogRead(xpin2);
176.                      delay(1000/frecMuest);
177.                  }
178.                  Serial1.write(3);
179.                  delay(100);
180.                  /*Aqui se aplica el filtro de media móvil de
   dichos datos registrados*/
181.
182.                  for (uint16_t
   i = 0; i < samples+venFil+1; i++) {
183.                      mean[i] = meanFilter.AddValue(mean[i]);
184.                      mean2[i] = meanFilter2.AddValue(mean2[i]);
185.                  }
186.
187.                  /*Normalizamos todos los vectores, quitando los
   primeros venFil+1 valores*/
188.                  for(uint16_t
   i = venFil+1; i < samples+venFil+1; i++) {
189.                      mean[i-venFil-1] = mean[i];
190.                      mean2[i-venFil-1] = mean2[i];
191.                  }
192.                   suma = 0.0;
193.                   suma2 = 0.0;
194.                  /*Voy a realizar el promedio de la medicion para
   luego restarlo y eliminar los datos no representativos*/
```

```
195.                    for (uint16_t i = 0; i < samples; i++) {
196.                        suma = suma + mean[i];
197.                        suma2 = suma2 + mean2[i];
198.                    }
199.
200.                    promedio = suma/samples;
201.                    promedio2 = suma2/samples;
202.
203.                    /*A continuacion restituyo los valores de mean
    quitandole la media para evitar que oscile en torno a 0*/
204.                    for (uint16_t i = 0; i < samples; i++) {
205.                        mean[i] = mean[i] - promedio;
206.                        mean2[i] = mean2[i] - promedio2;
207.                    }
208.
209.                    /* CALIBRACIÓN */
210.                    for (uint16_t i = 0; i < samples; i++) {
211.                        mean[i] = mean[i]/cal1;
212.                        mean2[i] = mean2[i]/cal2;
213.                    }
214.
215.                    Serial1.write(4);
216.                    delay(100);
217.
218.                    /*primerpaquete pertenece al valor del
    acelerometro del piso superior medido*/
219.                    /*2primerpaquete pertenece al valor del
    acelerometro del piso inferior medido*/
220.                    for (uint16_t i = 0 ; i < samples/2; i++){
221.                        primerpaquete[i] = mean[i];
222.                        acel2primerpaquete[i] = mean2[i];
223.                    }
224.
225.                    /*segundopaquete pertenece al valor del
    acelerometro del piso superior medido*/
226.                    /*2segundopaquete pertenece al valor del
    acelerometro del piso inferior medido*/
227.                    for (uint16_t i = samples/2 ; i < samples; i++){
228.                        segundopaquete[i-samples/2] = mean[i];
229.                        acel2segundopaquete[i-samples/2] = mean2[i];
230.                    }
231.
232.                    /*tercerpaquete pertenece al valor del
    acelerometro del piso superior medido*/
233.                    /*2tercerpaquete pertenece al valor del
    acelerometro del piso inferior medido*/
234.                    for (uint16_t
    i = samples/4 ; i < 3*(samples/4) ; i++){
235.                        tercerpaquete[i-
    samples/4] = mean[i];
236.                        acel2tercerpaquete[i-
    samples/4] = mean2[i];
237.                    }
238.                    Serial1.write(5);
239.                    delay(100);
240.
241.                    /*Hasta aqui tenemos todo finalizado y correcto.
    Faltaría mirar si se puede hacer lo mismo con menos vectores.*/
242.
243.                    /*Necesito llenar de 0 mi vector por que ahora
    no tendre parte imaginaria en el registro*/
244.                    for (uint16_t i = 0; i < samples/2; i++) {
```

```
                    chiprimeroimag[i] = 0.0;
                    chisegundoimag[i] = 0.0;
                    chiterceroimag[i] = 0.0;
                    primerpaqueteimag[i] = 0.0;
                    acel2primerpaqueteimag[i] = 0.0;
                    segundopaqueteimag[i] = 0.0;
                    acel2segundopaqueteimag[i] = 0.0;
                    tercerpaqueteimag[i] = 0.0;
                    acel2tercerpaqueteimag[i] = 0.0;
                }

                /*Aqui se calcula el chi de los datos registrados*/
                for (uint16_t i = 0; i < samples/2; i++) {
                    chiprimero[i] = primerpaquete[i] - acel2primerpaquete[i];
                    chisegundo[i] = segundopaquete[i] - acel2segundopaquete[i];
                    chitercero[i] = tercerpaquete[i] - acel2tercerpaquete[i];
                }


                /*Para que funcione bien la FFT tenemos que declarar aqui la libreria*/
                arduinoFFT FFTprimerpaquete = arduinoFFT(acel2primerpaquete,acel2primerpaqueteimag,samples/2,frecMuest);
                arduinoFFT FFTchiprimero = arduinoFFT(chiprimero,chiprimeroimag,samples/2,frecMuest);

                arduinoFFT FFTsegundopaquete = arduinoFFT(acel2segundopaquete,acel2segundopaqueteimag,samples/2,frecMuest);
                arduinoFFT FFTchisegundo = arduinoFFT(chisegundo,chisegundoimag,samples/2,frecMuest);

                arduinoFFT FFTtercerpaquete = arduinoFFT(acel2tercerpaquete,acel2tercerpaqueteimag,samples/2,frecMuest);
                arduinoFFT FFTchitercero = arduinoFFT(chitercero,chiterceroimag,samples/2,frecMuest);

                FFTprimerpaquete.Windowing(FFT_WIN_TYP_HANN, FFT_FORWARD);
                FFTchiprimero.Windowing(FFT_WIN_TYP_HANN, FFT_FORWARD);

                FFTsegundopaquete.Windowing(FFT_WIN_TYP_HANN, FFT_FORWARD);
                FFTchisegundo.Windowing(FFT_WIN_TYP_HANN, FFT_FORWARD);

                FFTtercerpaquete.Windowing(FFT_WIN_TYP_HANN, FFT_FORWARD);
                FFTchitercero.Windowing(FFT_WIN_TYP_HANN, FFT_FORWARD);
```

```
283.
284.                   FFTprimerpaquete.Compute(FFT_FORWARD);
285.                   FFTchiprimero.Compute(FFT_FORWARD);
286.
287.                   FFTsegundopaquete.Compute(FFT_FORWARD);
288.                   FFTchisegundo.Compute(FFT_FORWARD);
289.
290.                   FFTtercerpaquete.Compute(FFT_FORWARD);
291.                   FFTchitercero.Compute(FFT_FORWARD);
292.
293.                   /*HASTA AQUI DA COSAS LOGICAS*/
294.
295.                   for (uint16_t i =0; i < samples/2; i++) {
296.                     NumprimeroR[i] = (chiprimero[i]*acel2primerpaq
    uete[i] + chiprimeroimag[i]*acel2primerpaqueteimag[i]);
297.                     NumprimeroI[i] = (chiprimeroimag[i]*acel2prime
    rpaquete[i] - chiprimero[i]*acel2primerpaqueteimag[i]);
298.                     DenprimeroR[i] = (acel2primerpaquete[i]*acel2p
    rimerpaquete[i] + acel2primerpaqueteimag[i]*acel2primerpaqueteimag
    [i]);
299.                     DenprimeroI[i] = 0.0;
300.
301.                     NumsegundoR[i] = (chisegundo[i]*acel2segundopa
    quete[i] + chisegundoimag[i]*acel2segundopaqueteimag[i]);
302.                     NumsegundoI[i] = (chisegundoimag[i]*acel2segun
    dopaquete[i] - chisegundo[i]*acel2segundopaqueteimag[i]);
303.                     DensegundoR[i] = (acel2segundopaquete[i]*acel2
    segundopaquete[i] + acel2segundopaqueteimag[i]*acel2segundopaquete
    imag[i]);
304.                     DensegundoI[i] = 0.0;
305.
306.                     NumterceroR[i] = (chitercero[i]*acel2tercerpaq
    uete[i] + chiterceroimag[i]*acel2tercerpaqueteimag[i]);
307.                     NumterceroI[i] = (chiterceroimag[i]*acel2terce
    rpaquete[i] - chitercero[i]*acel2tercerpaqueteimag[i]);
308.                     DenterceroR[i] = (acel2tercerpaquete[i]*acel2t
    ercerpaquete[i] + acel2tercerpaqueteimag[i]*acel2tercerpaqueteimag
    [i]);
309.                     DenterceroI[i] = 0.0;
310.                   }
311.
312.                   /*Ahora hago la media aritmetica de los numeros
    complejos en la forma binomial*/
313.                   for (uint16_t i =0; i < samples/4; i++) {
314.                     NumR[i] = (NumprimeroR[i]+NumsegundoR[i]+Numte
    rceroR[i])/3.0;
315.                     NumI[i] = (NumprimeroI[i]+NumsegundoI[i]+Numte
    rceroI[i])/3.0;
316.                     DenR[i] = (DenprimeroR[i]+DensegundoR[i]+Dente
    rceroR[i])/3.0;
317.                     DenI[i] = (DenprimeroI[i]+DensegundoI[i]+Dente
    rceroI[i])/3.0;
318.                   }
319.
320.
321.
322.                   /*Aqui por ultimo calculo la H en forma binomial
    tambien*/
323.                   for (uint16_t i =0; i < samples/4; i++) {
324.                     Hreal[i] =(NumR[i]*DenR[i]+DenI[i]*NumI[i])/(D
    enR[i]*DenR[i]+DenI[i]*DenI[i]);
```

```
                    Himag[i] =(NumI[i]*DenR[i]-NumR[i]*DenI[i])/(DenR[i]*DenR[i]+DenI[i]*DenI[i]);
                }


                /*Con la siguiente operacion convierto la H en magnitud*/
                for (uint16_t i =0; i < samples/4; i++) {
                   Hmagnitud[i] = sqrt(sq(Hreal[i]) + sq(Himag[i]));
                }


                for (uint16_t i = 0; i < samples/4; i++) {
                    Serial.println(Hmagnitud[i]);
                    /*Serial.print(",");
                    Serial.println(mean2[i]);*/
                }

                /*Este bucle for lo que me hara es recorrerme el vector y almacenarme el máximo valor y su posicion dentro del mismo vector*/
                maximo = 0.0;
                for (uint16_t i = 0; i < 256; i++) {
                    if (Hmagnitud[i] > maximo) {
                        maximo = Hmagnitud[i];
                        posicion = i;
                    }
                }

              /*Con esto hago las cuentas finales para que pueda mandar a la aplicación móvil mis datos*/
                fo = posicion*(2*frecMuest/samples);
                amortiguamiento = 1/(2*maximo);

                if (fo < 1000) {
                    numerote = fo*100;

                    b1= numerote/256;
                    b2= numerote - b1*256;
                } else {
                    numerote = fo*10;
                    b1=numerote/256;
                    b2= numerote - b1*256;
                }

                if (amortiguamiento < 1000) {
                    numerote2 = amortiguamiento*100;

                    c1 = numerote2/256;
                    c2 = numerote2 - b1*256;
                } else {
                    numerote2 = amortiguamiento*10;
                    c1 = numerote2/256;
                    c2 = numerote2 - b1*256;
                }

                Serial1.write(12);
                Serial1.write(1);
                Serial1.write(330);
                delay(100);
                Serial1.write(13);
```

```
381.                 Serial1.write(c1);
382.                 Serial1.write(88);
383.                 delay(100);
384.                 Serial1.write(6);
385.
386.                 }
387.           }
388.      }
```