uc3m | Universidad Carlos III de Madrid

Grado Universitario en Ingeniería Mecánica

2019-2020

Trabajo Fin de Grado

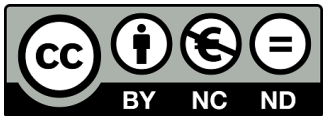
"Aplicación informática sobre Android para el análisis de vibraciones"

Jorge Barbolla Sánchez

Tutor:

Alejandro Bustos Caballero

Leganés, Noviembre 2020



RESUMEN

Este trabajo consiste en el desarrollo de una aplicación para dispositivos móviles en Android. Esta aplicación accede al acelerómetro que incorporan los móviles y toma mediciones con este sensor. Así las mediciones pueden ser guardadas en la memoria interna del dispositivo para su posterior análisis o exportación.

El análisis se realiza evaluando las señales medidas en el dominio de frecuencias de las mismas a través de la Transformada Rápida de Fourier o FFT, cuyos cálculos se realizan íntegramente en el dispositivo de forma que desde la toma de datos hasta los resultados finales ocurren en la misma aplicación de forma rápida.

Para comprobar la funcionalidad de la aplicación se realizan una serie de pruebas sobre su funcionamiento y en entornos reales donde se sabe los resultados que se deberían obtener para así realizar una comparación con los obtenidos en la aplicación.

Palabras clave: Android; Vibraciones; Transformada de Fourier; Aplicación móvil; FFT

DEDICATORIA

A todos los que han estado ahí cuando lo he necesitado y han colaborado de alguna forma en este proyecto.

Gracias por todo.

ÍNDICE GENERAL

1. INTRODUCCIÓN 1
1.1. Ámbito y motivación 1
1.2. Objetivos 2
1.3. Estructura del documento 3
2. COMPLEMENTOS TEÓRICOS 4
2.1. Vibraciones. 4
2.1.1. Clasificación de las vibraciones 4
2.2. Análisis de vibraciones 9
2.2.1. Dominio de la frecuencia 9
2.2.2. Transformada de Fourier 10
2.2.3. Muestreo de los datos 12
2.3. Android 14
2.3.1. Elementos de una aplicación 14
2.3.2. Interfaz del usuario 18
2.3.2.1. Views 18
2.3.2.2. Layouts 22
2.3.2.3. Eventos 24
2.3.2.4. Menús 25
2.3.2.5. Diálogos y notificaciones 25
2.3.3. Recursos 27
2.3.4. Sensores 27
2.3.4.1. Tipos de sensores 28
2.3.4.2. Identificar sensores y capacidades 28
2.3.4.3. Monitorización de eventos 30
2.3.5. Emulador. 32
2.3.6. Librerías Android 32
2.4. Acelerómetro. 34
2.4.1. Tipos de acelerómetro 34

3. DESCRIPCIÓN DE LAS HERRAMIENTAS EMPLEADAS 35
3.1. Programa para el desarrollo de aplicaciones 35
3.2. Android Studio 36
3.3. Librerías Android de interés 36
3.4. Otros programas 37
4. METODOLOGÍA 39
4.1. Etapas del proyecto. 39
4.2. Antecedentes. 40
4.2.1. Aplicaciones previas y documentación 40
4.3. Marco regulador. 43
4.3.1. Permisos y protección de datos 43
4.3.2. Play Store 44
4.4. Entorno socioeconómico 46
4.4.1. Android 46
4.5. Resultados esperados. 52
5. DESARROLLO DE LA APLICACIÓN 53
5.1. Estructura de la aplicación - MainActivity. 53
5.2. Medida de datos - *AccData.java* 57
5.2.1. Layout 58
5.2.2. Acceso al acelerómetro. 59
5.2.3. Tasa de muestreo 60
5.2.4. Gráfica de las mediciones 63
5.2.5. Propiedades del acelerómetro. 68
5.3. Guardado de los datos - GuardarDatos 70
5.3.1. Layout 70
5.3.2. Permisos de Guardado de Datos 72
5.3.3. Crear una nueva medición 73
5.3.4. Elección de número de medidas por medición 75
5.3.5. Iniciar o parar la medición 76
5.3.6. Guardar los datos en el archivo 78
5.3.7. Extra: Exportar datos 79

5.4. Análisis de los datos - FFT 82
5.4.1. Layout 82
5.4.2. Permisos y Selección de archivo 83
5.4.3. Importación al dispositivo del archivo 86
5.4.4. Análisis FFT 87
6. DESARROLLO DE LOS SISTEMAS DE PRUEBA 91
6.1. Construcción generador de vibraciones 91
6.2. Construcción banco de pruebas. 98
6.2.1. Diseño 98
6.2.2. Fabricación y Montaje 101
6.2.3. Equipos complementarios 106
6.2.4. Sujeción para móviles 108
7. RESULTADOS 119
7.1. Medidas acelerómetro activity AccData 119
7.2. Pruebas en casos reales 123
7.2.1. Mediciones en máquinas 123
7.2.2. Medidas generador de vibraciones 130
7.2.3. Medidas del banco de pruebas 137
8. PRESUPUESTO 145
9. CONCLUSIONES Y TRABAJOS FUTUROS 148
9.1. Trabajos futuros 150
BIBLIOGRAFÍA. 152
A. CÓDIGO DE LA ACTIVITY ACCDATA 155
B. CÓDIGO DE LA ACTIVITY GUARDARDATOS 167
C. CÓDIGO DE LA ACTIVITY FFT 177

ÍNDICE DE FIGURAS

2.1 Ejemplo sistemas de vibraciones 4
2.2 Ejemplo de sistema continuo 5
2.3 Ejemplo de sistema discretizado 5
2.4 Suma de ecuaciones armónicas 7
2.5 Representación de la transformada 11
2.6 Aliasing 12
2.7 Señales con muestreo de 100 Hz 13
2.8 Ciclo Vida Activity 15
2.9 Objetos descendientes de View 19
2.10 Ejemplo atributos padding y gravity 20
2.11 Uso de un Adapter 21
2.12 Tipos comunes de View 21
2.13 Tipos de Layout 23
2.14 Menú Barra con Opciones 25
2.15 AlertDialogo con opciones 26
2.16 Emulador en Android Studio 32

3.1 Logo de Android Studio 36
3.2 Proceso para la impresión de una pieza 38

4.1 Fases del proyecto 39
4.2 Capturas de Vibration Analyzer 41
4.3 Capturas de mVibe 42
4.4 Cuota de mercado sistema operativo mundial 46
4.5 Cuota de mercado sistema operativo España 47
4.6 Cuota de mercado sistema operativo tablets España 48
4.7 Cuota de mercado versión Android 49
4.8 Logo de Huawei 51

5.1 Diagrama de las *actvities* de la aplicación 53
5.2 Entorno Design del archivo xml 54
5.3 Entorno Text del archivo xml 55
5.4 Diagrama de flujo Accdata 57
5.5 Layout de AccData 58
5.6 Aceleraciones medidas 59
5.7 Tasa de muestreo 61
5.8 Spinner para seleccionar tasa 62
5.9 Avisos por pantalla 63
5.10 Ejemplo de gráfica con *GraphView* 64
5.11 Gráfica sin datos 65
5.12 Gráfica mediciones datos 67
5.13 Layout de la *activity* "propiedades" 68
5.14 Propiedades del acelerómetro (LG G6) 69
5.15 Layout de la *activity* "Guardar Datos" 71
5.16 Diagrama de flujo de la *activity* "Guardar Datos" 71
5.17 Aviso permisos para guardar datos 72
5.18 Estado inicial *activity* GuardarDatos 73
5.19 Recuadro selección nombre mediciones 74
5.20 Elección número de mediciones 75
5.21 Número exacto de mediciones 76
5.22 Mensaje de "Guardando datos..." 77
5.23 Mensaje al parar la medición 78
5.24 Mensajes al guardar las mediciones 79
5.25 Situación de archivos para exportarlos 80
5.26 Exportar los archivos 81
5.27 Diagrama de flujo la activity FFT 82
5.28 Layout de la activity FFT 83
5.29 Permisos para acceder a datos 84
5.30 Formulario para elegir el archivo a importar 84
5.31 Exportar archivos en csv 85

5.32 Importación del archivo al dispositivo 87
5.33 Selección tasa para el análisis 88
5.34 Gráficas con los resultados 89
5.35 Bloqueo para ampliar las gráficas 90

6.1 Despiece helicóptero 91
6.2 Estructura central helicóptero 92
6.3 Pata del generador en diseño 93
6.4 Disco perforado en Cura 94
6.5 Disco perforado en impresión 95
6.6 Pata del generador impresa 96
6.7 Disco para añadir desequilibrio impreso 96
6.8 Vista lateral de la base del generador 97
6.9 Estructura del generador completa 97
6.10 Cojinete para el eje 99
6.11 Cojinete para el eje 100
6.12 Plano del disco de desequilibrios 100
6.13 Banco de pruebas en Fusion 360 101
6.14 Base del banco de pruebas 102
6.15 Soporte del cojinete 102
6.16 Imágenes de la “L” y el disco 103
6.17 Banco de pruebas finalizado 105
6.18 Tacómetro digital 106
6.19 Pieza inferior de la sujeción 109
6.20 Pieza superior de la sujeción 110
6.21 Pieza de priete 110
6.22 Tuercas de la sujeción 111
6.23 Sujcompleta en Fusion 360 112
6.24 Renderizado de la sujeción en Fusion 360 113
6.25 Pieza inferior de la sujeción en Cura 114
6.26 Pieza superior de la sujeción en Cura 115
6.27 Pieza apriete en Cura 116

6.28 Pieza del apriete impresa . . . 117
6.29 Tuercas de la sujeción en Cura . . . 117
6.30 Tuercas Impresas . . . 118
6.31 Ejemplo de sujeción instalada . . . 118

7.1 Resultados motor coche . . . 124
7.2 Resultados motor coche en Matlab . . . 124
7.3 Torno Industrial . . . 126
7.4 Resultados Medida 1 del torno . . . 127
7.5 Resultados Medida 2 del torno . . . 128
7.6 Resultados Medida 3 del torno . . . 129
7.7 Disposición del generador para las mediciones . . . 130
7.8 Resultados Gen1 . . . 131
7.9 Resultados Gen2 . . . 132
7.10 Resultados Gen3 . . . 133
7.11 Resultados GenDisco1 . . . 134
7.12 Resultados GenDisco2 . . . 135
7.13 Resultados GenDisco3 . . . 136
7.14 Disposición del banco para las mediciones . . . 137
7.15 Resultados BancoA1o . . . 138
7.16 Resultados BancoA2o . . . 139
7.17 Resultados BancoA3o . . . 140
7.18 Resultados BancoA4o . . . 141
7.19 Resultados BancoAx . . . 142
7.20 Resultados BancoBo . . . 143
7.21 Resultados BancoBx . . . 144

ÍNDICE DE TABLAS

2.1 Algunos tipos de sensor 28

4.1 Ventas por fabricante 1Q19 (Miles de unidades) 50

6.1 Voltaje funcionamiento motor 370 92
6.2 Voltaje funcionamiento motor 777 99

7.1 Mediciones LG G6 Sin gráfica 119
7.2 Mediciones LG G6 Con gráfica 119
7.3 Estadísticas mediciones LG G6 120
7.4 Xiaomi Mi A2 Lite (Julio 2018) 121
7.5 Alcatel Pop3 5015D (Q4 2015) 121
7.6 Alcatel 1 5033D (Julio 2018) 121
7.7 Samsung A40 (Marzo 2019) 121
7.8 ZTE Nubia (2018) 122
7.9 Xiaomi mi 9 (Febrero 2019) 122
7.10 Samsung note 10 (Agosto 2019) 122
7.11 Xiaomi Redmi Note Plus 5 (Febrero 2018) 122
7.12 Mediciones del coche 125
7.13 Velocidades de las mediciones del torno 126
7.14 Resultados Medida1 127
7.15 Resultados Medida2 128
7.16 Resultados Medida3 129
7.17 Velocidades de las mediciones del generador 130
7.18 Resultados Gen1 131
7.19 Resultados Gen2 132
7.20 Resultados Gen3 133
7.21 Velocidades de las mediciones del generador con disco 134
7.22 Resultados GenDisco1 134

7.23 Resultados GenDisco2 135
7.24 Resultados GenDisco3 136
7.25 Velocidades mediciones banco 137
7.26 Resultados BancoA1o 138
7.27 Resultados BancoA2o 139
7.28 Resultados BancoA3o 140
7.29 Resultados BancoA4o 141
7.30 Resultados BancoAx 142
7.31 Resultados BancoBo 143
7.32 Resultados BancoBx 144

8.1 Costes de personal 145
8.2 Costes de equipos 146
8.3 Costes del banco de prueba 146
8.4 Costes totales 147

ÍNDICE DE CÓDIGOS

2.1 Código para generar las señales en Matlab 7
2.2 Ejemplo AndroidManifest.xml básico 16
2.3 View básico en xml 19
2.4 Layout básico en xml 22
2.5 Carga de Layout en activity (main) 22
2.6 Implementación Listener en un botón 24
2.7 Ejemplo AlertDialog 26
2.8 Ejemplo de Toast 27
2.9 Acceso al servicio sensor y lista 29
2.10 Ejemplo acceso a versión de sensor 29
2.11 Monitorización datos del sensor de luz 30
5.1 MainActivity.java 56
5.2 Acceso al acelerómetro 59
5.3 Ejemplo del cálculo de la tasa de muestreo 60
5.4 Implementación Librería GraphView 63
5.5 Gráfica de AccData en el archivo xml 64
5.6 Archivo java de test 68
5.7 Ejemplo de datos en archivo csv 81
5.8 Ejemplo de datos csv válidos 86
A.1 Archivo activity_acc_data.xml 155
A.2 Archivo accData.java 157
B.1 Archivo guardardatos.xml 167
B.2 Archivo guardardatos.java 168
C.1 Archivo guardardatos.xml 177
C.2 Archivo guardardatos.java 179

1. INTRODUCCIÓN

1.1. Ámbito y motivación

Tanto las vibraciones mecánicas como su posterior análisis son una parte imprescindible del mantenimiento industrial, en concreto del mantenimiento predictivo y preventivo. Los equipos usados por la industria para realizar dicho diagnostico son a menudo complejos y tienen un coste elevado sin tener en cuenta otras desventajas como la de adquisición de equipo de precisión o la de necesitar una formación específica en la materia para la utilización de dicha maquinaria.

La importancia de este proyecto radica en proveer al usuario de una forma sencilla y rápida de poder tomar mediciones y hacer uso de ellas para obtener datos del objeto analizado. Otro de los pilares en los que se basa el proyecto es la portabilidad y alcance que tiene una aplicación de un smartphone "dispositivo móvil". Todo esto unido al bajo coste, ignorando el coste del dispositivo, hace que esta sea una forma rápida e inmediata de analizar los datos necesarios.

Es innegable la rápida expansión que han tenido los smartphone en los últimos años multiplicándose exponencialmente el número de ellos que son fabricados y existen actualmente. Con su expansión se ha producido a su vez una mejora en la tecnología que utilizan y en lo relativo a este proyecto, sobre todo la mejora en los acelerómetros que llevan integrados y la mejora en la capacidad de procesamiento con móviles más potentes cada año. Esto se traduce en un procesado de los datos más rápido y preciso y por lo tanto el uso que se le puede dar a la aplicación a desarrollar es cada vez más amplio.

El análisis de vibraciones nunca ha sido tan accesible y portátil por lo que se quiere explorar en este trabajo las capacidades de los dispositivos móviles y descubrir las utilidades que nos brindan.

1.2. Objetivos

El **objetivo principal** de este proyecto es el *desarrollo de una aplicación móvil* basada en el sistema operativo Android que sea capaz de medir aceleraciones con el acelerómetro interno del dispositivo y después analizar dichas aceleraciones.

Para la consecución de el objetivo principal se deben cumplir y ejecutar varios **objetivos secundarios** necesarios para alcanzar el primario. También se generan objetivos a raíz del principal para complementarlo y ampliarlo. Estos objetivos secundarios son los siguientes:

- Para el desarrollo de la aplicación y alcanzar el objetivo primario:
  - Búsqueda de documentación sobre Android y los programas necesarios para el desarrollo de una aplicación.
  - Recopilación de aplicaciones similares para así realizar un estudio de mercado y ver en qué se puede diferenciar la aplicación a desarrollar.
  - Se establecen los requisitos de la aplicación con las necesidades.
  - Valorar la accesibilidad y las capacidades de los acelerómetros incorporados y su interacción con el sistema Android.
- Para completar el proyecto con base en la aplicación desarrollada:
  - Construcción de bancos de prueba para utilizar la aplicación en un entorno real.
  - Estudio de las propiedades de los acelerómetro incorporados en distintos dispositivos.
  - Valorar la utilidad de este tipo de mediciones con su análisis y la capacidad de estos acelerómetros incorporados en los dispositivos.

1.3. Estructura del documento

- *Complementos teóricos*: En este capítulo se profundizará y se hará una introducción a los conceptos necesarios para comprender el desarrollo de la aplicación. Se complementa la información sobre vibraciones, su análisis y la programación de la aplicación.

- *Herramientas utilizadas*: Aquí se detallarán las herramientas informáticas utilizadas para el desarrollo de la aplicación con una explicación de por qué se seleccionaron dichas herramientas.

- *Metodología*: En esta sección se establecen las fases del trabajo y se detalla alguno de los pasos previos al desarrollo del proyecto antes de entrar a su desarrollo. También se comentarán los resultados que se esperan obtener antes de su realización.

- *Desarrollo*: En este capítulo se detalla los pasos seguidos para la programación de la aplicación con enlaces al código utilizado en los anexos para de este modo encontrar una combinación entre explicación y código resultante. La explicación se centrará en la funcionalidad de este código para que cumpla con los requisitos establecidos.

  A su vez, se desarrollarán distintos sistemas de prueba que generan vibraciones para ensayar la aplicación y comprobar su funcionalidad.

- *Resultados*: En este apartado se muestran los resultados de las mediciones y las capacidades del acelerómetro en varios entornos de prueba.

- *Presupuesto*: Aquí se detallarán los distintos gastos asociados al desarrollo de este proyecto.

- *Conclusiones y trabajos futuros*: En este capítulo se comprobará la consecución de los objetivos iniciales del proyecto con el desarrollo de este y los resultados obtenidos. Además se nombrarán posibles extensiones en forma de proyectos a raíz de este para complementarlo.

2. COMPLEMENTOS TEÓRICOS

2.1. Vibraciones

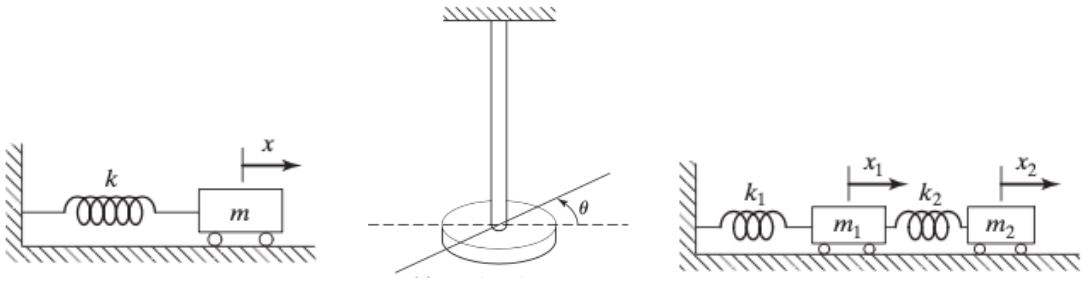
Se define vibración como cualquier movimiento que se repite en el tiempo. Es estudio de estos movimientos se basa en las oscilaciones del cuerpo en estudio y la interacción con fuerzas que le puedan afectar. Para entender estas vibraciones es necesario modelizarlas mediante sistemas vibratorios y determinar las ecuaciones que los definen.[1]

2.1.1. Clasificación de las vibraciones

2.1.1.1 Grados de libertad

Los grados de libertad de un sistema se definen como la mínima cantidad de coordenadas independientes que determinan completamente la posición de todos los elementos del sistema para un tiempo concreto.

Estos grados de libertad vienen dados por la configuración de los elementos del sistema y la interacción entre ellos. Así si se tiene con una barra empotrada no deformable por ejemplo que además de desplazarse puede rotar, este giro también se considerará otro grado de libertad más del sistema.



(a) 1 GDL (b) 1 GDL (c) 2 GDL

Fig. 2.1. Ejemplos de sistemas de vibraciones con distintos grados de libertad [1]

2.1.1.1.1 Sistemas continuos y discretos

Los sistemas como los que están en la figura 2.1 son relativamente sencillos y tienen un número concreto de grados de libertad y son los que se llaman sistemas discretos, mientras que los sistemas continuos son lo que tienen grados de libertad infinitos y otros en los que normalmente incluyen algún elemento que se puede

deformar elásticamente y tienen infinitos puntos de masa como la viga de la figura 2.2.

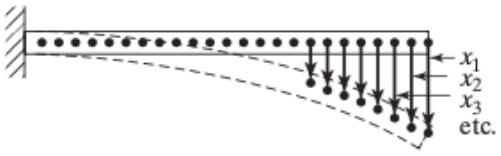


Fig. 2.2. Ejemplo de sistema continuo [1]

Para simplificar estos sistemas continuos el paso a seguir suele ser discretizar el sistema convirtiéndolo en un sistema discreto equivalente con un número finito de grados de libertad para que se pueda resolver de forma más fácil. Los sistemas continuos dan soluciones exactas pero solo se pueden utilizar para problemas simples como una barra o una chapa, para el resto se utilizan los sistemas discretizados con un número finito de elementos de tipo muelle, masas o amortiguadores. Incrementando el número de estos elementos y por lo tanto de grados más grados de libertad se obtienen resultados más exactos teniendo cálculos más complejos.

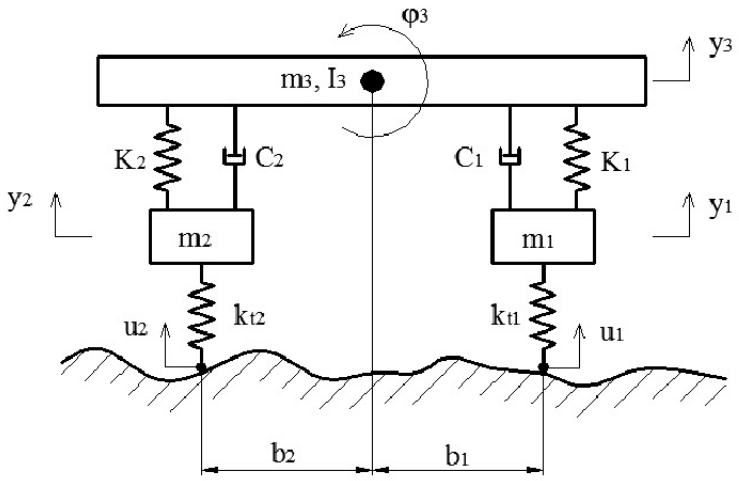


Fig. 2.3. Ejemplo de sistema discretizado [2]

En la figura 2.3 se puede ver como se discretiza la suspensión de una automóvil de un modelo continuo que tendría un número de grados de libertad muy altos, y se convierte en un modelo discreto más simple al que se puede llegar contando con un muelle, amortiguador y masa.

2.1.1.2 Elementos del sistema y modelización

Los sistemas suelen contar con medios para almacenar la energía elástica (muelles), la energía cinética (inercia) o para disipar energía (amortiguador). La energía

cinética se puede transformar en potencial y viceversa y si el sistema tiene amortiguamiento el sistema irá perdiendo energía a no ser que se le aplique una fuerza externa.

Así para modelizar un sistema mecánico para poder analizarlo se debe conocer las características del modelo que son: le rigidez, la masa y el amortiguamiento. Estos son los datos del sistema a analizar y se suponen constantes en el tiempo aunque en el uso real pueden variar.

2.1.1.3 Vibraciones libres/forzadas y deterministas/aleatorias

Las vibraciones **libres** son las que ocurren en un sistema que inicia fuera de su posición de equilibrio con energía potencial o elástica positiva y que en el tiempo debido a una pérdida más o menos rápida de energía llegará otra vez a la posición de equilibrio a un tiempo infinito. De modo que para que la vibración sea de este tipo deberá tener una posición distinta de la posición de equilibrio y/o una velocidad mayor que cero unido a que ninguna fuerza exterior actúa en el sistema.

Las vibraciones **forzadas** en cambio una fuerza externa actúa en el sistema y pueden ser de distintos tipos: **armónicas**, si siguen una forma sinusoidal; **choques**, si actúan durante un tiempo pequeño o **periódicas** que se repitan cada cierto periodo.

Con la acción de estas fuerzas el sistema puede tener una evolución de las deformaciones que sea periódica con un periodo "T" o que por el contrario tenga un comportamiento aleatorio y no tenga un periodo reconocible.

Con el comportamiento **aleatorio** ya se entra en las fuerzas aleatorias o ruido que no se pueden determinar exactamente pero se pueden estimar con valores estadísticos como la media, varianza o su composición en frecuencia entre otros métodos. Ejemplos de vibraciones aleatorias son las producidas en una suspensión por una carretera con baches o durante un terremoto.

Si se conoce el valor de la fuerza que actúa en el sistema a cada tiempo dado entonces esta fuerza causará una vibración **determinista**.

2.1.1.4 Movimiento armónico

El movimiento oscilatorio o vibraciones se pueden repetir en el tiempo y formar un movimiento periódico. El movimiento periódico más común y útil para su análisis es el movimiento armónico. En este movimiento el desplazamiento ocurren entre un valor máximo y mínimo y se repite con un periodo determinado.

En estos casos el movimiento se puede describir por una curva sinosuidal. La aceleración es directamente proporcional al desplazamiento y por tanto la vibración.

Un ejemplo de un movimiento simple armónico es dado por la ecuación $x = Acoswt$. Siendo "x" es la posición, "A" la amplitud del movimiento, "w" el desfase en su movimiento y "t" el tiempo transcurrido.

```
Fs=400; %Muestras por segundo (Hz)
dt=1/Fs; %Segundos por muestra
T=0.4; %Segundos muestreados
t=(0:dt:T-dt)';

%Ondas seno  Amplitud*sin(2*pi*Frecuencia*t + Fase)
x1=sin(2*pi*30*t);
x2=1.25*sin(2*pi*15*t);
x3=0.8*sin(2*pi*60*t);
x4=0.2*sin(2*pi*90*t);
suma=x1+x2+x3+x4;
```

CÓDIGO 2.1. Código para generar las señales en Matlab

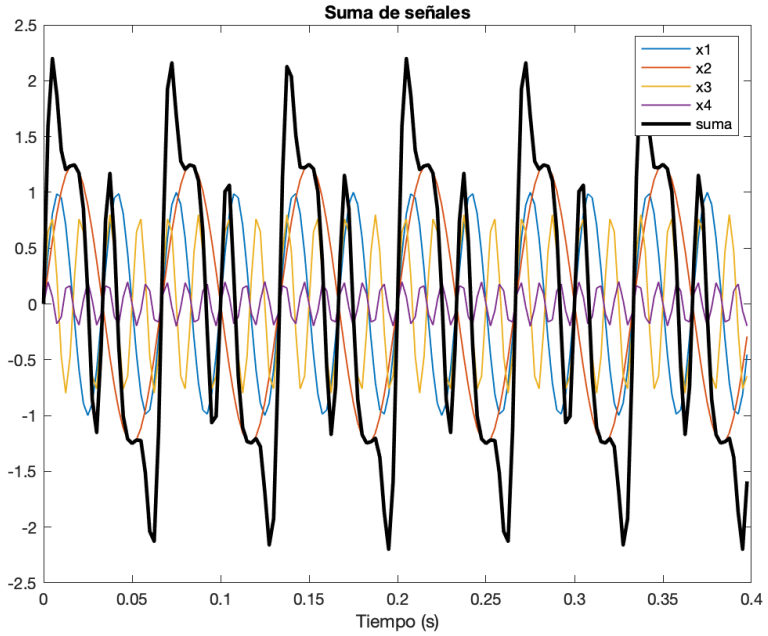


Fig. 2.4. Suma de ecuaciones armónicas

Como se puede ver en la figura 2.4 se ha obtenido una señal final (en negro) a partir de otras señales. De las señales iniciales se conoce su amplitud y su frecuencia pero de la señal final con cada frecuencia añadida es más difícil distinguir estas propiedades de la señal.

Estas ecuaciones armónicas se combinan sumándose formando una ecuación que se asimila más a una medición de un caso real. Estas mediciones las componen distintas vibraciones de distintas frecuencias y si el número es elevado de frecuencias diferentes, el análisis temporal de la señal es complejo de analizar. Para eso se recurre al análisis por frecuencias que se expondrá en el siguiente capítulo del proyecto.

Las ecuaciones ecuaciones de movimiento que expresan estas vibraciones se pueden expresar de forma vectorial, con números complejos o matricialmente pero como este proyecto no se requiere crear un modelo que describa las vibraciones, no se profundizará en estas ecuaciones.

2.2. Análisis de vibraciones

El objetivo mediante el análisis de vibraciones es que a partir de unos datos de entrada de por ejemplo vibraciones o fuerzas, obtener unas funciones o parámetros buscados. Así se puede diagnosticar el sistema vibratorio además de poder definirlo según su distintos criterios. Los

También se puede hacer un análisis conociendo tanto los datos de las excitaciones como la respuesta del sistema a estos. Estas excitaciones pueden ser controladas como en una prueba de una estructura para obtener las vibraciones naturales de esta. Este es el método con el que realizarán las pruebas de la aplicación para ver si su comportamiento es el esperado.

Los procesos más simples para el análisis son el filtrado y la descripción en el dominio de la frecuencia de los datos:

El **filtrado** se basa en modificar los datos de entrada según el objetivo que tenga el análisis. Puede que solo interesen las vibraciones lentas, entonces se podrían atenuar o eliminar las vibraciones rápidas.

Cuando la señal se pasa al **dominio de la frecuencia** se separa la señal en distintas componentes armónicas cuya suma resulta en la señal inicial.

2.2.1. Dominio de la frecuencia

El uso de un análisis por el dominio de la frecuencia es tan común que no se para a pensar la razón por la que se utiliza. Las principales razones por las que se recurre a este método son las siguientes:

1.) Como se puede ver en la figura 2.4, el análisis es más fácil en el dominio de la frecuencia respecto al dominio del tiempo. Cuando se representa la vibración con el tiempo, esta puede ser muy compleja y no se puede sacar ningún dato en claro. Mientras que con el dominio de la frecuencia se pueden distinguir frecuencia predominantes en la señal de forma clara como por ejemplo ocurren en las máquinas rotativas.

   Las pruebas para comprobar el funcionamiento de la aplicación se realizarán concretamente con máquinas rotativas para aprovechar la aparición de frecuencias esperadas derivadas de las vibraciones periódicas causadas por el giro de algún elemento de esa máquina.

2.) La aparición de patrones pequeños pueden ser indetectables dentro de una señal temporal mientras que en el dominio de la frecuencia pueden aparecer de forma aparente.

3.) La conversión al dominio de la frecuencia desde la señal temporal solo es posible debido a la existencia de algoritmos que los permiten como es la transformada rápida de Fourier o *FFT*.

4.) Por las propiedades ortogonales de la descomposición de Fourier de la señal, el producto cruzado de frecuencias diferentes no contribuyen a la suma de energía. Por esta propiedad es posible estudiar la contribución de una vibración de una frecuencia específica sin que el resto de vibraciones de la medición interfieran en el resultado.

2.2.2. Transformada de Fourier

La transformada de Fourier es una función matemática que descompone una función temporal o señal (a(t)) en las frecuencias que la componen. La transformada de una función temporal es una función en números complejos que depende de la frecuencia y el valor del valor absoluto del número complejo da la cantidad de cada frecuencia hay en la señal analizada.

$$A(f) = \int_0^T a(t) \cdot e^{-i2\pi ft} dt \tag{2.1}$$

$$a(t) = \int_0^T A(f) \cdot e^{i2\pi ft} dt \tag{2.2}$$

Para pasar al dominio de la frecuencia se utilizan las integrales de Fourier de las ecuaciones 2.1 y 2.2 pero como ha un número finito de muestras, estas integrales se discretizan en el periodo que se quiere analizar para dar las ecuaciones 2.3 y 2.4 para n=1,2...,N-1 y N=1,2,...,N-1.

$$A(k) = \frac{1}{N} \sum_{n=0}^{N-1} a(n) \cdot e^{-i\frac{2\pi kn}{N}} \tag{2.3}$$

$$a(n) = \sum_{k=0}^{N-1} A(k) \cdot e^{i\frac{2\pi kn}{N}} \tag{2.4}$$

Siendo N el numero de muestras de la señal en un intervalo de tiempo. a(n) da una solución en número reales y A(k) en complejos, siendo a(n) la transformación inversa. La combinación de ambas ecuaciones es lo que se conoce como *Transformada Discreta de Fourier*.

A efectos prácticos en el cálculo computacional se requerían de N^2 operaciones para N muestras. En 1965 en una revista matemática[1] aparece un nuevo algoritmo

[1] *Mathematics of Computation*, Nº 90, vol. 19

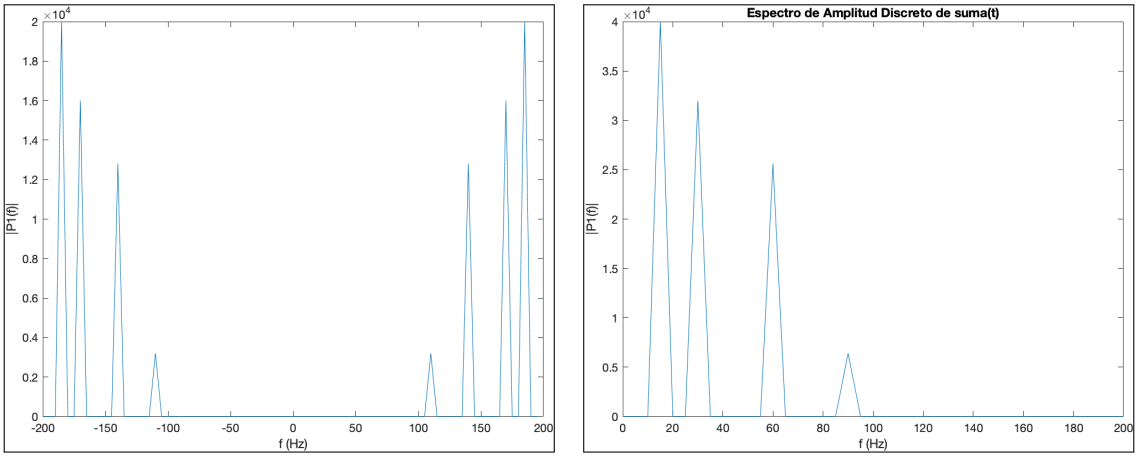
para el cálculo de la *Transformada Discreta de Fourier o DFT* en el que se reducen de forma sustancial el número de operaciones para el mismo cálculo ($N \log_2 N$), ahorrando en capacidad computacional y tiempo. Este nuevo algoritmo se denominó *Transformada Rápida de Fourier o FFT*. Ambos métodos producen los mismos resultados.

Representación de la transformada

De la ecuación 2.3 se obtiene una función compleja con su término real e imaginario que se expresa como $A(k) = Re(k) + iIm(k)$. $Re(k)$ expresa la parte real y $Im(k)$ la parte imagniaria. Este numero complejo de puede expresar por la definición de números complejos como un módulo y un ángulo de fase por lo que puede expresarse como $A(k) = |A(k)|e^{j\phi(k)}$ sonde el módulo se define como $|A(k)| = \sqrt{Re^2(k) + Im^2(k)}$ y el ángulo de desfase $\phi(k) = \tan^{-1} \frac{\mathrm{Im}(k)}{\mathrm{Re}(k)}$.

Para la representación es también importante número de muestras que tiene la medición. Este número influye directamente en la resolución de la gráfica resultante en el eje de la frecuencia. El número de puntos en el espectro corresponde a la mitad de los puntos de la medición ($N/2$). En la figura 2.5 a) aparecen lo N puntos pero como es simétrica se queda solo la mitad de ellos.

Por otra parte la resolución del eje horizontal que contiene la frecuencia viene dado por el cociente entre la tasa de muestreo y el número de muestras ($\Delta f = F_s/N$).



(a) Completo (b) Mitad

Fig. 2.5. Representación de la transformada

Como ejemplo de la representación del dominio de la frecuencia se va a mostrar la transformada aplicada a las señales de la figura 2.4. Esta señal ha sido muestreada a 400 Hz y en la figura 2.5 se puede ver las frecuencias que componen esta señal, que son a su vez otras señales armónicas de tipo seno con distintas amplitudes y las frecuencias que se muestran en esta gráfica. Esta muestra tiene un número

de medidas de $N = 160$ por lo que tiene una resolución en frecuencia de $\Delta f = 400/160 = 2,5$.

2.2.3. Muestreo de los datos

Teorema del muestreo

Cuando se toma una medición en analógico y se quiere convertir a digital se tienen que tomar puntos discretos de la señal continua. Para considerar que se ha realizado un muestreo correcto se debe poder volver a la señal continua desde los puntos discretos. Este teorema tiene especial interés en otros ámbitos como son las telecomunicaciones para la transmisión de datos son pérdida de información.

Este **teorema** dice que la frecuencia con la que se toman los datos o frecuencia de muestreo debe ser el doble de la frecuencia a medir. Esta frecuencia máxima que se puede medir se conoce como *frecuencia de Nyquist*.

$$f_N = \frac{1}{2 \cdot \Delta t} = \frac{fs}{2} = f_{\text{máx}} \tag{2.5}$$

Aliasing

Este efecto ocurre cuando se toman menos muestras por segundo que las que define la frecuencia de Nyquist. Este es el error principal en el tratamiento y procesado de señales ya que puede causar que una señal se confunda con otra y hacer que los datos sean inservibles.

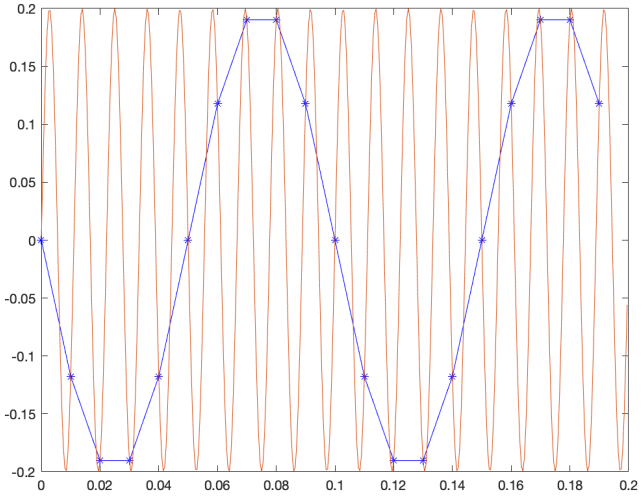
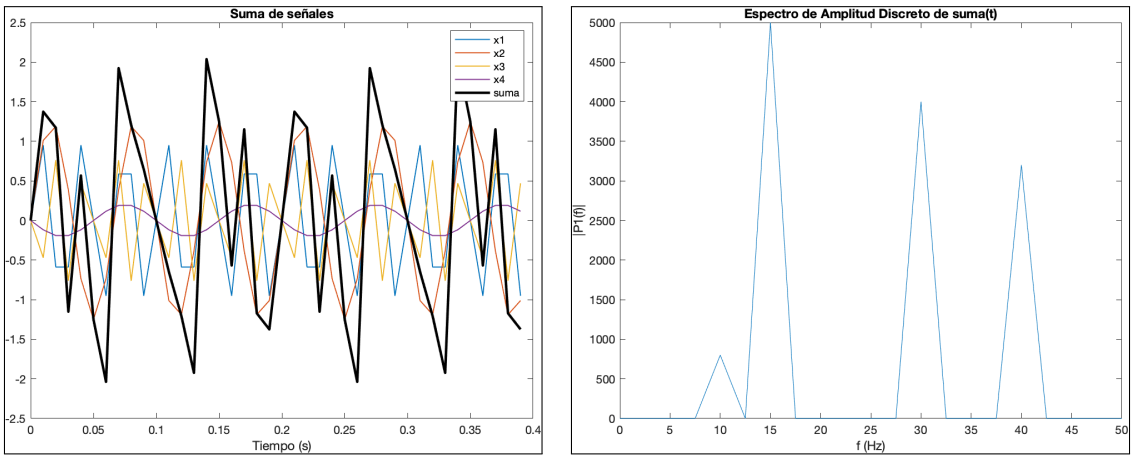


Fig. 2.6. Aliasing

Cuando esto ocurre, si la señal que ha sido muestreada a cierta frecuencia es de mayor frecuencia que el límite establecido por Nyquist, esta aparecerá con un

valor inferior de frecuencia del que realmente tiene. Afecta en mayor medida a las frecuencias altas ya que estas alcanzan el límite antes. Se tiene que dar la condición de que para un ciclo de señal senoidal, haya al menos dos muestras de esa señal para poder representarla correctamente.

Para ver una muestra gráfica de las consecuencias del *aliasing* se verá las señales de la figura 2.4 si se realizan mediciones con un valor de menor del la frecuencia de Nyquist.



(a) Dominio tiempo

(b) Dominio frecuencia

Fig. 2.7. Señales con muestreo de 100 Hz

La señal en la figura 2.7 se ha tomado con una tasa de muestreo de 100 Hz y la frecuencia máxima de las que componen la señal es de 90 Hz. La frecuencia de Nyquist para esta muestra de de $Fs/2 = 100/2 = 50Hz$ y toda frecuencia que sea mayor que esa incurrirá en el error producido por *aliasing*. La señal de 90 Hz aparecerá como una señal de 10 Hz (x4 en la figura 2.7 a).

Leakage

2.3. Android

2.3.1. Elementos de una aplicación

2.3.1.1. Activities y ciclo de vida

[3] Son de los elementos más importantes de los que forman una aplicación y su utilización hace que programar para *Android* se haga de una forma distinta. Son esencialmente las ventanas con las que el usuario interacciona con la aplicación y se utilizaría una distinta para cada interfaz gráfica que se use en la aplicación. Esta interfaz viene definida en el archivo .xml que se asocia a cada *activity*. Ahí se definirán la disposición de los elementos o *Layout*, texto, botones y demás elementos gráficos. También contará con el archivo *.java* para la funcionalidad de la *activity*.

Declarar activity

Con la creación de una nueva *activity* en la aplicación surge la necesidad de declararla para que la aplicación tenga constancia de que existe. Esto se hace en el archivo *AndroidManifest.xml* y el único atributo que es obligatorio introducir para declararlas es el nombre ->*android:name*.

Ciclo de vida

Un concepto importante que introducir es el ciclo de vida de las *activities* variando su estado desde se abren y se inician hasta que se cierran o destruyen. Por lo tanto tenemos tres estados básicos en los que se puede encontrar una activity:

- **Activo**: Este estado ocurre cuando está en ejecución y es la tarea principal de la aplicación.
- **Reposo**: La *activity* puede pasar a este estado en el que puede seguir realizando tareas en segundo plano pero sin ser la tarea principal. Hay que tener especial cuidado con esto y tener en cuanta que procesos se tienen activos en segundo plano para no sobrecargar la memoria y tener problemas con la aplicación.
- **Parado**: Finalmente en este estado los procesos han finalizado, no es visible para el usuario y ya no ocupa memoria. Si se quiere se puede reiniciar y pasar al primer estado.

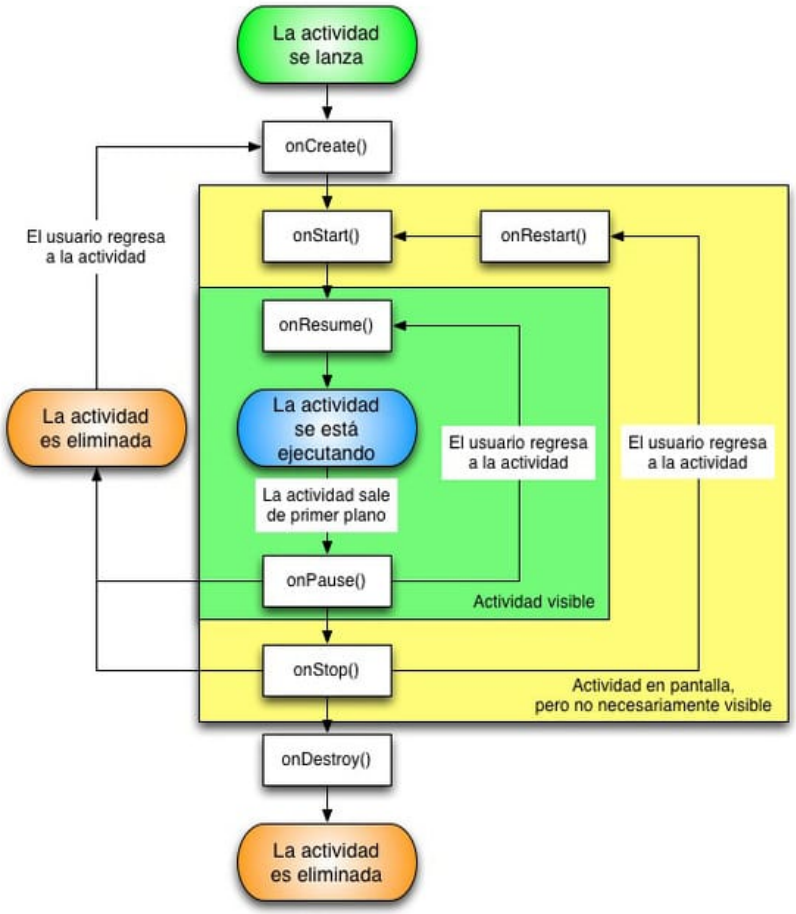


Fig. 2.8. Ciclo de vida de una Activity.[4]

Para pasar de un estado a otro existen una serie de métodos o comandos para pasar de un estado a otro. Estos son los más importantes:

- **onCreate**: Es el método que se lanza al crear una actividad y en que se deben incluir las variables que se tengan que inicializar así como los procesos que queramos que se inicien al mismo tiempo que se inicia la *activity*. Puede recibir un parémetro llamado *Bundle* en el que se almacenarían datos de la *activity* si esta has sido activada previamente, si no lo ha sido entonces este parámetro está vacío. Hay que señalar que al iniciarse los principales componentes se debe inicializar también en este método el método *setContentView()* para asignar el *layout* correspondiente.

- **onStart()**: Una vez utilizado *onCreate* se entra en un estado activo y se utiliza este método para activar una *activity* después de haber sido parada. Se utiliza después de un *onRestart* o después del *onCreate*.

- **onResume()**: Se invoca este método ya cuando la aplicación está en funcionamiento para permitir ya la interacción con el usuario después de que sea activa y pararía la *activity* a un primer plano.

- **onPause()**: Se llama a este método cuando la *activity* va dejar el primer plano en beneficio de otra. Se puede guardar su estado para ser introducido más adelante en el caso de reactivarla.
- **onStop()**: La *activity* pasa ya a un segundo plano pero de forma indefinida y la memoria que puede guardar tiene el riego de ser eliminada si la aplicación lo necesita.
- **onRestart()**: Se utiliza este método cuando una *activity* va a ser reinicidada al estado en que que estaba cuando se paró.
- **onDestroy()**: Este método marca el final de la vida de la *activity* y todos sus procesos. Se utiliza cuando ya no es de utilidad y se libera memoria. Es especialmente util para terminar procesos al ir a otra *activity* que ya no se necesiten y que si estuviesen activos consumirían gran parte de los recursos.

2.3.1.2. Intents

Un *intent* es una "intención" de hacer una acción, una comunicación entre distintos componentes de la aplicación y por lo tanto el medio de activación de dichos componentes. Se pueden declarar en el *Android Manifest* y pueden ser implícitos o explícitos, diferenciándose si se especifica al componente que van destinados o no respectivamente.

Según el componente al que vayan dirigidos se tratan de diferente manera, ya sea una *Activity*, *Service* u otros. Más específicamente en las *activities* que va a ser el tipo de *intent* que se va a usar en la aplicación, estos se lanzan con el método *startActivity(intent)* desde la *activity* desde la que lancemos el *intent* y con el método *getIntent()* se recibirá la información o la acción en la *activity* de destino.

2.3.1.3. Android Manifest

Es un fichero de tipo *.xml* en el que se declaran todos los elementos que va a tener la aplicación e interacciones entre ellos y externamente con otras aplicaciones. Este archivo es diferente a los *xml* que se asocian a cada *activity* en los que se detalla su apariencia. Este archivo se genera automáticamente al iniciar una nueva aplicación e incluye el paquete en el que se aloja la aplicación y todos sus componentes. Algunas de las tareas del *AndroidManifest* son proteger la aplicación y el acceso a sus distintos elementos mediante permisos que se deberán de aceptar para que se tenga acceso a ciertas partes de la aplicación. Esto incluye acceso a datos de la memoria del móvil o acceso a internet de la aplicación entre otras cosas.

```
1  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
2      package="com.javatpoint.hello"
3      android:versionCode="1"
```

```
4       android:versionName="1.0" >
5
6       <uses-sdk
7           android:minSdkVersion="8"
8           android:targetSdkVersion="15" />
9
10      <application
11          android:icon="@drawable/ic_launcher"
12          android:label="@string/app_name"
13          android:theme="@style/AppTheme" >
14          <activity
15              android:name=".MainActivity"
16              android:label="@string/title_activity_main" >
17              <intent-filter>
18                  <action android:name="android.intent.action.MAIN" />
19
20                  <category android:name="android.intent.category.LAUNCHER" />
21              </intent-filter>
22          </activity>
23          <activity android:name="OtraActivity"/>
24      </application>
25
26  </manifest>
```

CÓDIGO 2.2. Ejemplo AndroidManifest.xml básico

Los elementos principales que se incluyen en el *AndroidManifest* son:

- <**Manifest**>: Así se declara este archivo y dentro de este elemento se incluye el paquete al que pertenece la aplicación así como atributos suyos como es la versión de la aplicación. También incluye la version del SDK aunque este versión puede indicarse en otro lugar como ocurre en la aplicación que se va desarrollar, en el *build.gradle* de la aplicación.

- <**Application**>: Así es como se defina la aplicación y dentro contendrá todos los elementos. Es un subelemento del *Manifest* y se incluye también en este apartado parámetros que afecten a toda la aplicación como puede ser el nombre, estilo o *theme*, icono de la aplicación etc.

- <**Activity**>: Se incluyen dentro de la aplicación todos los <*Activity*> que contenga la aplicación. A su vez contarán con los mismos atributos de la aplicación y si no se definen nuevamente se utilizará el que usa la aplicación. De esta manera cada *activity* puede tener parámetros independientes como puede ser la orientación de pantalla. Se profundizará en el desarrollo de la aplicación en este tema.

- <**Intent-filter**>: Es un subelemento de la *activity* que señala el tipo de *intent* al que la *activity* puede tener respuesta. Como por ejemplo una llamada entrante o un enlace a otra aplicación.

- **Otros**: También pueden aparecer otros elementos en el *Manifest* como son los permisos de la aplicación.

Gradle

Gradle en un **constructor** de Android, es decir combina archivos de distinto tipo, librerías, recursos de la aplicación, archivos java y más elementos y los combina en un archivo **APK**. Este archivo es el con el que se puede instalar la aplicación en un dispositivo y contiene todos los datos necesarios para ello.

En el proceso de compilación y construcción de la aplicación se coge el código de los archivos *.xml* y *.java*, los recursos, las bibliotecas externas con todo su código y el *AndroidManifest.xml* que contienen información sobre la aplicación y lo junta todo en un archivo *.dex*. Luego este archivo con una extensión de Android Studio llamada "APK Manager" se combina con recursos que no están en el arcivo *.dex* y forma finalmente el *APK*.

Este *APK Manager* se utiliza también en la fase de desarrollo cuando se utiliza un *debug* o prueba de la aplicación. Se hace esto para comprobar que la aplicación funciona dentro del entrono de programación. Para realizar estar pruebas se conectará un dispositivo móvil por cable al ordenador con el que se desarrolla la aplicación el cual hará de emulador y probará el funcionamiento de la aplicación.

2.3.2. Interfaz del usuario

La interfaz de usuario es un elemento muy importante de la aplicación, define la interacción entre el usuario y el dispositivo. Esta interacción de hace a través de la pantalla y hay una serie de requerimientos que hay que tener para que sea una interacción fácil. Los elementos tienen que estar bien organizados y que sean fácilmente entendibles por el usuario.

2.3.2.1. Views

La interfaz se compone de distintos objetos descendientes de la clase **View**. Descendiente en Android se refiere a que hereda código y evita la necesidad de volver a escribir código innecesario. Esto es una de las bases de la programación en objetos y la simplifica notablemente, por eso los objetos que se van a utilizar son descendientes de la clase *View*.

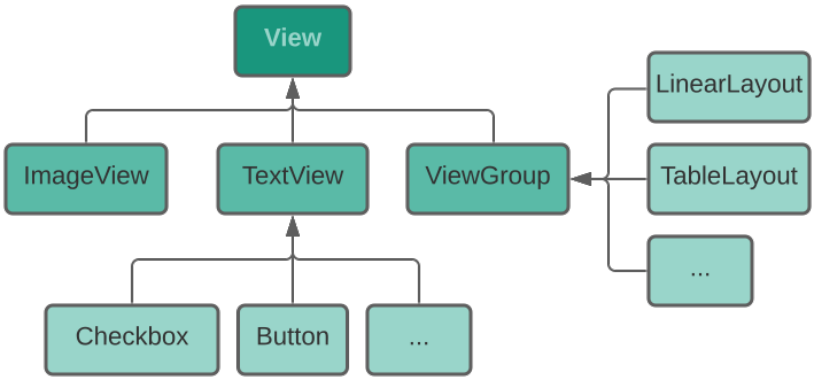


Fig. 2.9. Objetos descendientes de View.

De la clase *View* hay dos tipos principales de objetos, los de tipo **View** y los de tipo **ViewGroup**. Los de tipo *View* incluyen por ejemplo los botones o los cuadros de texto. Dentro de estos objetos los que son visibles por el usuario y tienen interacción con el usuario se llaman *Widgets*. Los objetos *ViewGroup* son una base para los elementos de tipo *layout*. Un elemento *ViewGroup* puede tener otros elementos *View* en su interior con los que usuario interacciona.

Para añadir un *View* a la interfaz hay que declararlo en el archivo xml. Incluyendo el tipo de *View* por ejemplo *button* y sus **atributos**. Cada atributo tendrá un valor dependiendo del tipo de atributo. Unos atributos tienen que ver con la forma, tamaño, estilo; y otros con su situación en la pantalla.

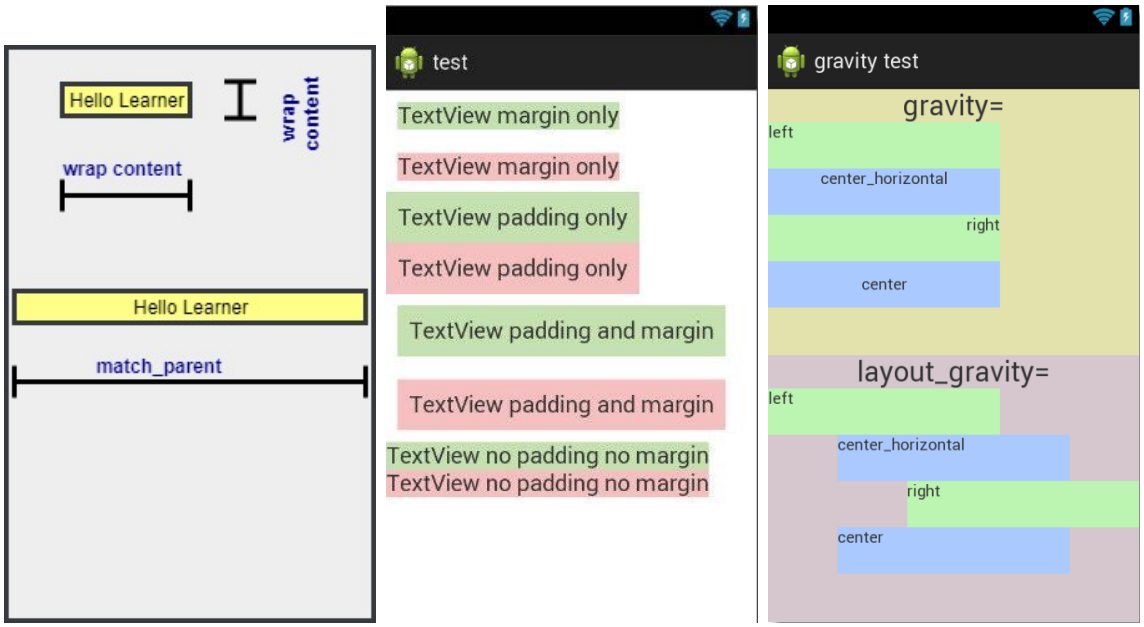
```
<ViewType
    Atributo1=Valor1
    Atributo2=Valor2
/>
```

CÓDIGO 2.3. View básico en xml

En la pantalla los *view* se pueden considerar como un rectángulo visible o invisible que es el elemento que se coloca en la pantalla. De esta manera no puede haber dos objetos en el mismo espacio. El tamaño de este rectángulo puede ser definido por el usuario en unidades exactas o se pueden usar unos valores predefinidos como son: *match_parent* y *wrap_content*.

Match_parent significa que el rectángulo ocupará el máximo tamaño disponible en la pantalla y *wrap_content* que el rectángulo se ajustará a su contenido.

Dos atributos que siempre tienen que estar son *android:layout_height* y *android:layout_width* que marcan la forma del rectángulo que ocupa cada *View*. Otros atributos que se usan bastante son *gravity* que marca la situación del *View* dentro del *layout* siendo por ejemplo centrado o *padding* que señala los márgenes con la pantalla.



(a) Tamaño [5] (b) Padding y Margin [6] (c) Gravity [7]

Fig. 2.10. Ejemplo de los atributos padding y gravity.

Cada componente de la interfaz puede tener los mismos atributos para diferenciarlos hay un atributo para identificarlo y darle un **ID** o nombre al componente. Este atributo se expresa de esta forma *android:id="@+id/nombre"* y de esa ese elemento tiene un ID propio.

Tipos de View

- **Button**: Los botones son la forma más sencilla de programar acciones cuando son seleccionados. Se pueden programar distintas acciones dependiendo de su se pulsan de forma prolongada o no. Como todos los elementos de tipo *View*, se puede modificar su apariencia mediante sus atributos.
- **TextView**: Este tipo de elementos se usa para mostrar información en forma de texto por pantalla. Tiene los mismo atributos que los botones pero son la capacidad de provocar ninguna acción.
- **TextEdit**: Son como los elementos *textView* pero editables. Se utilizan para introducir información en forma de texto o numérica que después se utilizará en la aplicación como por ejemplo en un formulario.
- **CheckBox**: Es un tipo de botón pero tiene dos estados de operación en vez de uno, activado o desactivado. En ambos estados se puede programar alguna acción o se puede utilizar para elegir entre dos opciones distintas.

- **Spinner**: Este *View* es una forma rápida y simple de elegir un elemento en una lista. El elemento seleccionado es el que queda reflejado en el rectángulo que ocupa el *spinner* y para seleccionar otro elemento se selecciona el *spinner* y aparecerá un menú desplegable con los elementos de este. Los pasos para la creación del menú desplegable o *spinner* para seleccionar las tasas son:

  1. Creación del *Spinner* (Menú desplegable).
  2. Creación de un *ArrayList* (Vector de elementos).
  3. Creación de un *Adapter* (Conexión entre los datos y la representación en pantalla).
  4. Uso del método *onItemSelected()* para cambiar de tasa al seleccionar una nueva.

  Para introducir elementos se utilizará un *SpinnerAdapter* y si los elementos vienen definidos en un *Array* o vector entonces se utilizará un *ArrayAdapter*. El *ArrayAdapter* es la conexión entre el *Array* que contiene los elementos y su representación visual por medio de una lista en el archivo *xml*.

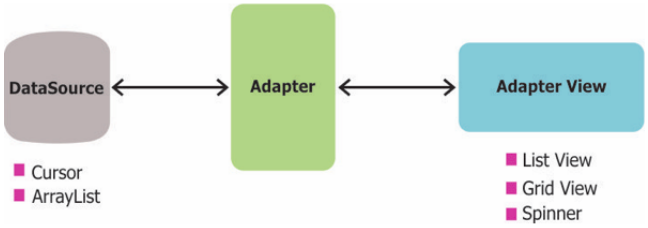


Fig. 2.11. Esquema del uso de un Adapter [8]

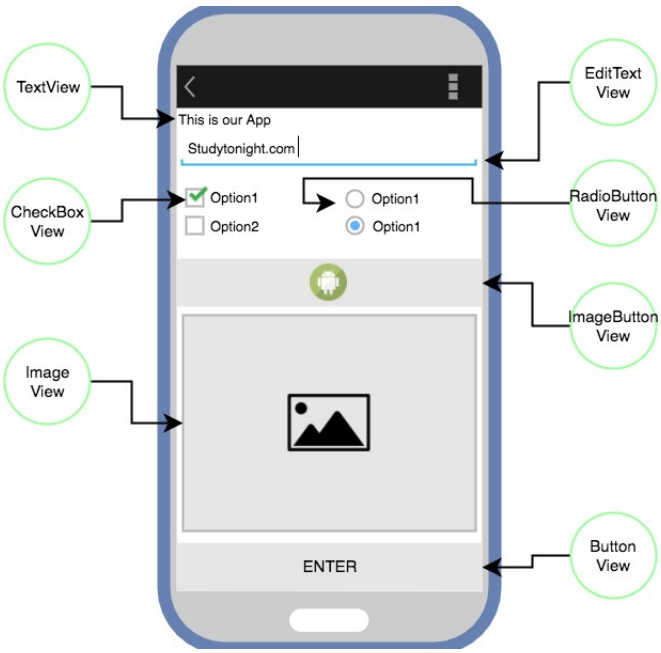


Fig. 2.12. Tipos más comunes de tipo View [5]

2.3.2.2. Layouts

Los *layouts* son un recurso de la aplicación que describen lo que se muestra por pantalla y la forma en la que se muestra. Al igual que los *View* siguen el mismo método de creación en el archivo xml que es el siguiente donde se especifican los atributos del *layout* y los elementos que contiene.

```
<TipodeLayout
    Atributo1=Valor1
    Atributo2=Valor2>

    Elemento1
    Elemento2
<TipodeLayout/>
```

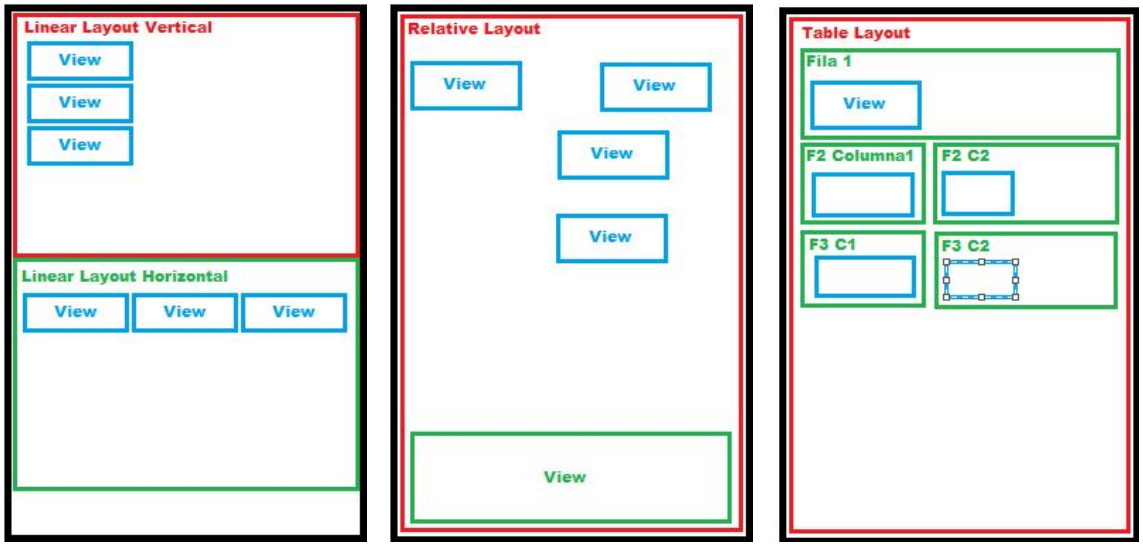
CÓDIGO 2.4. Layout básico en xml

Los atributos que son necesarios que tengan son igual que en los *view*, *layout_height* y *layout_width* para definir el ancho y la altura del *layout*. Y aunque se pueden definir numéricamente lo más recomendable es utilizar *match_parent* y *wrap_content* al igual que los *view*. Como ambos elementos descienden de la clase *View* tanto los *layout* como los elementos *view* comparten atributos y los que se utilizaban en los componentes también se pueden utilizar aquí. Se sigue considerando para su colocación el rectángulo que ocupa cada *layout* con la diferencia de que uno puede incluir elementos *view* en su interior u otros *layouts*.

El *layout* principal de cada *activity* se cargará en archivo java dentro del método *onCreate()* para asociarlo. Se hará de la siguiente forma:

```
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
```

CÓDIGO 2.5. Carga de Layout en activity (main)



(a) Linear Layout (b) Relative Layout (c) Table Layout

Fig. 2.13. Ejemplos de tipos de Layout

Linear Layout

Es el que se genera automáticamente al crear una nueva *activity* y es uno de los más simples. Los elementos se colocan verticalmente en una columna o horizontalmente en una fila. Si los elementos ocupan más que la pantalla estos no serán visibles.

Relative Layout

Este *layout* se basa en las posiciones relativas de sus elementos. Se puede definir una posición como la distancia a otro elemento o como la distancia a el *layout* que los contiene, por ejemplo se puede decir que un elemento vaya abajo de la pantalla. Este *layout* es uno de los más versátiles ya que cada elemento puede situarse exactamente dónde queramos.

Frame Layout

Es el más simple en cuanto a que todos los elementos que se introduzcan se colocarán arriba a la izquierda de forma que se bloquearían unos a otros. Es útil por ejemplo si la pantalla va atener un solo elemento, por ejemplo una imagen.

Table Layout

Los elementos *view* se dispondrán en filas y columnas. Se utiliza *<TableRow>* para iniciar una nueva fila de la columna. Cada fila funciona como un *Linear Layout*

de orientación horizontal. No se muestran las lineas que separan los elementos tanto en filas como columnas.

Scroll View

No es un *layout* como tal en el sentido de que no puede incluir elementos en su interior. Lo que sí que puede incluir son otros *layout* que su tamaño haga que no entren todos los elementos en la pantalla. Por ejemplo un *LinearLayout* que sea más alto que la pantalla se puede incluir dentro de un *ScrollView* y así se puede acceder a todos los elementos bajando en la pantalla.

2.3.2.3. Eventos

También conocidos como eventos de usuario se utilizan para detectar la interacción ente aplicación y usuario. Una de las formas más comunes de hacerlo es mediante un **Event Listener** que una forma de interactuar con la interfaz de la aplicación, por ejemplo pulsando un botón. Estos elementos *View* deben registrar el *Event Listener* correspondiente y activar una respuesta que se activará con la interacción del usuario.

Los *Event Listeners* más comunes son los siguientes:

- **onClick()**: Este método es llamado cuando el usuario hace una pulsación simple sobre un elemento *view* o también con las teclas de dirección.
- **onLongClick()**: Funciona exactamente igual que el método *onClick* pero con una pulsación larga.
- **onKey()**: Este método se inicia cuando el usuario selecciona una tecla determinada del teclado.
- **onTouch()**: Este método es llamado cuando el usuario hace alguna presión sobre la pantalla, retira la presión o hace algún movimiento.

Implementación de Event Listeners

Estos deben ser registrados en la aplicación de modo que al activarse se llame a un método específico en el que ya se se realiza lo que se haya programado. Un ejemplo de implementación de un *Listener* a la pulsación de un botón en el siguiente:

```
1  private Button BotonPrueba;
2
3  protected void onCreate(Bundle savedInstanceState) {
4          //Se inicializa el View(Boton)
5          BotonPrueba = (Button)findViewById(R.id.IDboton);
```

```
6           //Se registra el listener
7           button.setOnClickListener(btnListener);
8       }
9   private View.OnClickListener btnListener = new View.OnClickListener() {
10      public void onClick(View v) {
11          // Aquí va lo que se quiera realizar al hacer click
12      }
```

CÓDIGO 2.6. Implementación Listener en un botón

2.3.2.4. Menús

Los menús en las aplicaciones dotan al usuario de atajos, acciones y otras opciones al usuario son tener que salir de una *activity*. Los menús son por tanto un elemento de la interfaz de usuario. El menú más simple es un **menú de barra** que se sitúa en la parte superior de la pantalla y suele tener funciones que tienen un impacto global en la aplicación o *activity* como son por ejemplo realizar una búsqueda o "Ajustes".[9]

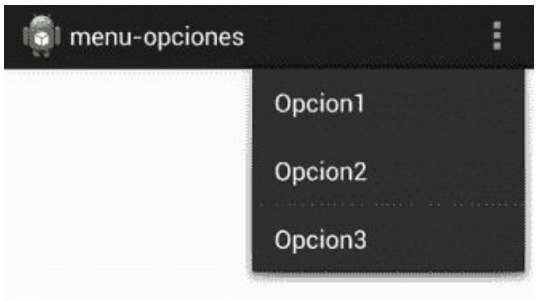


Fig. 2.14. Ejemplo de barra con opciones [10]

Los **menús contextuales** son los que aparecen cuando se hace una presión larga sobre un elemento.

2.3.2.5. Diálogos y notificaciones

Cuando se hacen ciertas tareas en la aplicación es útil realizar avisos y notificaciones al usuario. Para ello existen los diálogos y las notificaciones. La principal diferencia entre ambos es que los diálogos exigen una interacción con el usuario y se ejecutan en primer plano. Esta interacción puede definir una acción o recordar algo al usuario que la aplicación necesita su aceptación para realizar cierta tarea. Las notificaciones por otro lado solo aportan información al usuario y solo puede verlas y aceptarlas.

Los **AlertDialog** son un elemento emergente que muestra un título y hasta tres botones u otros elementos. El usuario entonces selecciona uno de ellos y se activará

ese botón y las acciones asociadas. El siguiente ejemplo sería un *AlertDialog* con dos botones que al querer salir de la aplicación se activa.

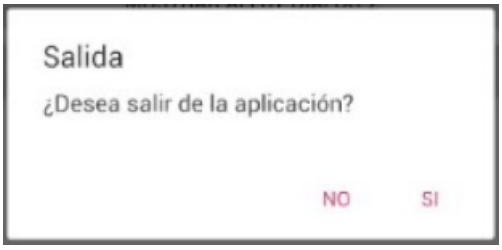


Fig. 2.15. AlertDialog con opciones para salir [11]

```
AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder.setMessage("Desea salir de la aplicación?")
    .setTitle("Salida")
    .setCancelable(false)
    .setPositiveButton("Si", new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int id) {
            MyActivity.this.finish();
        }
    })
    .setNegativeButton("No", new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int id) {
            dialog.cancel();
        }
    });
AlertDialog alert = builder.create();
```

CÓDIGO 2.7. Ejemplo AlertDialog

También existen otros tipos de *AlertDialog* en el sentido de que su contenido es diferente. Se siguen activando de la misma forma pero en su interior en vez de botones puede que haya una lista con distintas opciones entre las que elegir o entre distintas cifras y que al elegir uno se haga una acción.

Estos diálogos son personalizables en todos los aspectos posibles ya sea en tamaño, forma, color, disposición de los elementos. Para hacerlo se puede crear un *layout* del diálogo en xml y añadirlo al crearlo de una manera similar a la que se añade un *layout* a una *activity*.

Otro tipo de diálogo es el **ProgressDialog** que indica que indica que una tarea esta siendo ejecutada. Si se sabe lo que va a tardar se representa mediante una barra que se va llenando y si no mediante un círculo con movimiento. También puede incorporar otros elementos como botones en el caso por ejemplo de que se quiera cancelar la acción.

Notificaciones

Se cuenta con dos tipos de notificaciones, en primer lugar las **Toast** que aparecen en la pantalla en la que estemos. No afecta a lo que estemos realizando ni se requiere que el usuario haga nada, estas desaparecerán con el tiempo que tengan programado.

```
1  Toast.makeText(getActivity(), "Aquí va el mensaje que se mostrará",
2     Toast.LENGTH_LONG).show();
```

CÓDIGO 2.8. Ejemplo de Toast

Por otro lado están las notificaciones que aparecen en la barra de estado que se van aculando en el sistema Android y puede haber de distintas aplicaciones a la vez. Si se seleccionan, se abre la aplicación a la que pertenezca la notificación y al ser muy configurables se puede añadir vibración o sonido. Estas notificaciones siempre se realizan en segundo plano y no deben hacer que el usuario deje lo que está haciendo si no lo quiere.

2.3.3. Recursos

2.3.4. Sensores

Todos los sensores del dispositivo se encuentran en el paquete *Android.Hardware* como puede ser la cámara, acelerómetro y resto de sensores entre los que se suelen incluir sensores para la posición u orientación. Se puede acceder a los sensores del dispositivo mediante una interfaz en Android para realizar tareas relacionadas con los sensores como son: saber que sensores están disponibles; obtener los datos que salen de ellos y al ritmo en el que se toman; registrar esos datos con *eventlisteners* o determinar datos del propio sensor como son su fabricante, su rango de medidas o su capacidad.[12]

Sensor Framework

A través del *Sensor Framework* o interfaz de Android con los sensores se puede acceder a ellos y adquirir datos. Este *Sensor Framework* cuenta con clases e interfaces que son útiles para tratar con los sensores:

- **SensorManager**: Es una clase que da varios métodos para acceder a los sensores o comenzar o dejar de registrar datos del sensor a través de los *Event Listeners*. Cuenta con distintas constantes que definen la precisión del sensor o la tasa de adquisión de datos.
- **Sensor**: Esta clase se utiliza para crear una instancia del sensor, como una representación de él. Cuenta con métodos para saber la capacidad del sensor.

- **SensorEvent**: Esta clase se utiliza para crear un evento que incluyen la información de los datos tomados por el sensor, el sensor que los generó, la precisión de los datos y un *timestamp* que indica el momento que se tomaron.
- **SensorEventListener**: Esta interfaz se utiliza para llamar a métodos que se activan cuando los datos cambian o su precisión.

2.3.4.1. Tipos de sensores

Este *Sensor Framework* te deja acceder a muchos tipos de sensores entre los que se encuentran los basados en el hardware y los basados en software. Los basados en hardware dan sus datos midiendo directamente distintas propiedades como la aceleración, campo magnético o datos del giroscopio. Los sensores basados en software derivan las medidas de las medidas de uno o varios sensores basados en hardware como el sensor de gravedad que combina mediciones del acelerómetro y osciloscopio.

Hay sensores más comunes como el acelerómetro que lo incluye la mayoría de dispositivos pero otros como un un barómetro no son muy comunes. Un dispositivo puede contar también con varios sensores del mismo tipo.

TABLA 2.1. ALGUNOS TIPOS DE SENSOR

Sensor	Tipo	Descripción
TYPE_ACCELEROMETER	Hardware	Mide la fuerza de la aceleración en m/s^2 en tres ejes (x,y,z)
TYPE_GRAVITY	Hardware o Software	Mide la fuerza de la gravedad en los tres ejes en en m/s^2.
TYPE_GIROSCOPE	Hardware	Mide la velocidad de rotación alrededor de sus ejes en rad/s
TYPE_LIGHT	Hardware	Mide el nivel de luz ambiente en lx.
TYPE_LINEAR_ACCELERATION	Hardware o Software	Mide la aceleración en los ejes sin incluir la gravedad en m/s^2.
TYPE_ORIENTATION	Software	Mide la rotación del dispositivo en los tres ejes.
TYPE_PROXIMITY	Hardware	Mide la proximidad de un objeto en cm.
TYPE_PRESSURE	Hardware	Mide la presión en hPa o mbar.
Fuente: Developers Android Sensors (Abril 20) [12]		

2.3.4.2. Identificar sensores y capacidades

El *Sensor Framework* tiene varios métodos que hacen fácil saber que sensores tiene el dispositivo y si están disponibles. Para identificar los sensores o acceder a alguno se tienen que referencias el servicio de sensores y para hacerlo se crea una

instancia de la clase *SensorManager* llamando al método *getSystemService()* y utilizando el servicio *SENSOR_SERVICE*. Ahora para obtener una lista de los sensores del dispositivo se utilizará el método *getSensorList()* y la constante *TYPE_ALL* o de un tipo específico como por ejemplo *TYPE_GRAVITY*.

```
sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);

List<Sensor> deviceSensors = sensorManager.getSensorList(Sensor.TYPE_GRAVITY);

if (sensorManager.getDefaultSensor(Sensor.TYPE_GRAVITY) != null){
    //Hay sensor
} else{
    //No hay sensor
    }
}
```

CÓDIGO 2.9. Acceso al servicio sensor y lista

Para saber si un sensor existe en el dispositivo se puede usar el método *getDefaultSensor()* incluyendo la constante que indica el tipo de sensor. Si la respuesta a este método es *null* o nula es que el dispositivo no tiene ningún sensor de ese tipo.

Otros métodos para saber más cosas de los sensores y sus capacidades son *getMaximumRange()*, *getResolution()* y *getPower* obteniendo el rango máximo de medición del sensor, la resolución y la potencia que necesita el sensor respectivamente. Para conocer más datos de su fabricación están *getVendor()* y *getVersion()* para conocer el fabricante y la versión del sensor.

Todos estos datos sirven para conocer las limitaciones del sensor y saber que tareas podría realizar de forma que si se necesita por ejemplo una resolución mínima para realizar una tarea se puede aceptar o descartar el sensor en función de si lo cumple. También es útil si se sabe que solo los últimos sensores de cierto fabricante pueden hacer o si conociendo que el sensor que tienen el dispositivo no es suficiente, optimizar o cambiar el proceso para que un sensor menos capaz pueda funcionar.

```
sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);

if (sensorManager.getDefaultSensor(Sensor.TYPE_GRAVITY) != null){
    //Se crea una lista con todos los sensore tipo GRAVITY
    List<Sensor> sensorGrav = sensorManager.getSensorList(Sensor.TYPE_GRAVITY);
    //Para cada sensor de tipo GRAVITY se comprueba su versión
    for(int i=0; i<sensorGrab.size(); i++) {
        if (gravSensors.get(i).getVersion() == 2){
            // Si hay uno de versión 2 se usa.
            mSensor = gravSensors.get(i);
        }else if{
            //No hay de versión 2
        }
    }
}
```

CÓDIGO 2.10. Ejemplo acceso a versión de sensor

2.3.4.3. Monitorización de eventos

Para monitorizar los datos se utilizan dos métodos de *SensorEventListener* que se llaman cuando ocurre una de las siguientes cosas:

Si el sensor da un nuevo valor se llama al método **onSensorChanged()** dando un evento de tipo objeto. Este evento contienen toda la información del sensor y los datos registrados.

Si cambia la precisión del sensor se llama al método **onAccuracyChanged()** dando el sensor que ha cambiado su precisión y la nueva precisión. La precisión se da en la forma de cuatro constantes: *SENSOR_STATUS_ACCURACY_LOW*, *MEDIUM*, *HIGH* y *UNRELIABLE*.

```
1  public class SensorActivity extends Activity implements SensorEventListener {
2      //Se declaran las instancias
3      private SensorManager sensorManager;
4      private Sensor sesorLight;
5
6      @Override
7      public final void onCreate(Bundle savedInstanceState) {
8          super.onCreate(savedInstanceState);
9          setContentView(R.layout.main);
10
11         sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
12         sensorLight = sensorManager.getDefaultSensor(Sensor.TYPE_LIGHT);
13         sensorManager.registerListener(this, sensorLight, SensorManager.
               SENSOR_DELAY_NORMAL);
14     }
15
16     @Override
17     public final void onAccuracyChanged(Sensor sensor, int accuracy) {
18         // Si cambia la precisión se hace lo que haya aquí
19     }
20
21     @Override
22     public final void onSensorChanged(SensorEvent event) {
23         // Cuando cambian los datos se toma un valor en lux
24         float lux = event.values[0];
25         // Se hace lo que sea con el valor almacenado en un float (variable)
26     }
27     //Se retoma la monitorización de datos
28     @Override
29     protected void onResume() {
30         super.onResume();
31         sensorManager.registerListener(this, mLight, SensorManager.
               SENSOR_DELAY_NORMAL);
32     }
33     //Se para la monitorización de datos
34     @Override
35     protected void onPause() {
36         super.onPause();
37         sensorManager.unregisterListener(this);
38     }
39 }
```

CÓDIGO 2.11. Monitorización datos del sensor de luz

Como se puede ver en la línea 14 del código 2.11 se establece una tasa de toma de datos o *delay* que se específica cuando se usa el *listener*. Este *delay* especifica la tasa de refresco con la que se envían los eventos de datos mediante el método *onSensorChanged()*. Para la mayoría de acciones una tasa de refresco normal es válida pero existen otras tasas para otras situaciones más exigentes. El *delay* normal tiene una tasa de 200000 microsegundos o 5Hz mientras que si se expecifica *SENSOR_DELAY_GAME* tienes una tasa de 20000 μs o 50 Hz y con *SENSOR_DELAY_UI* son 60000 μs y 16,67 Hz. Por último se cuenta con *SENSOR_DELAY_FASTEST* sin restricción y toma datos tan rápido como el dispositivo pueda.

De puede seleccionar una cifra como valor del *delay* pero hay que tener en cuenta que el sistema tiene en cuenta este *delay* como una sugerencia y es probable que el que se utilice no sea el dado. Por esto los fabricantes recomiendan utilizar la tasa de refresco menor pero que siga cumpliendo las necesidades. No existe un método como tal en Android para sacar el *delay* de un sensor pero se podría utilizar los *timestamp* de cada medida para ello.

Por último, cabe destacar la importancia de los métodos *onResume()* y *onPause()* para registrar y anular el *listener* de los eventos. Si un sensor no se va a utilizar es siempre recomendable pausarlo o cancelarlo para que no gaste más recursos del dispositivo de los necesarios o batería.

Modos de funcionamiento

Los sensores tienen distintos modos de funcionamiento en función de la forma en la que se generan los eventos. Primero están los que funcionan de forma **continua**, tomando datos cada cierto tiempo definido. Ejemplos de funcionamiento en este modo son los acelerómetros.

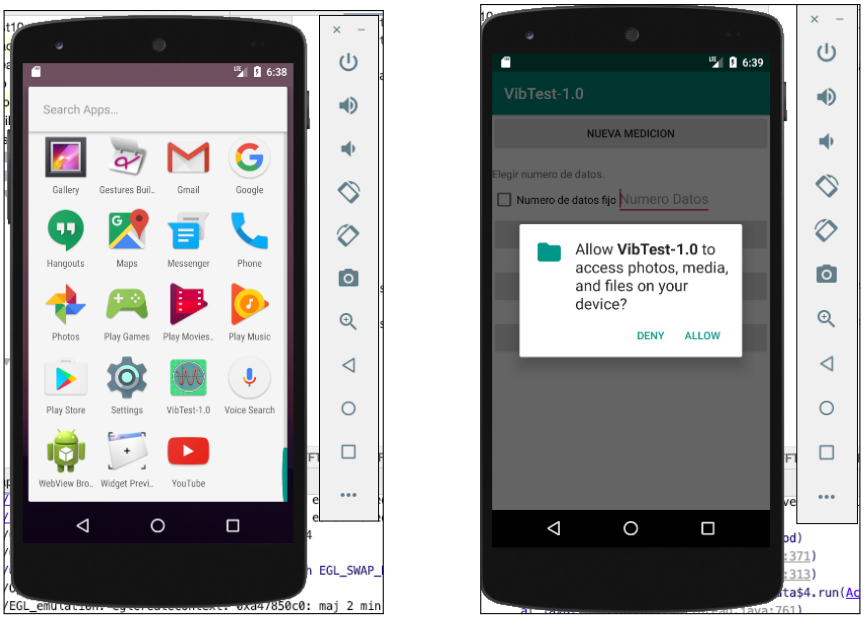
Mediante el modo de **cambio** u *on-Change* los eventos se registran solo cuando se ha producido un cambio. Esto se produce en sensores como un sensor cardiaco o de proximidad. Utilizan también una tasa de muestreo que marca cuando su puede volver a generar un evento y se registraría solo si ha cambiado respecto al estado anterior. Ejemplos de este funcionamiento pueden ser que un sensor cardiaco deje de tomar datos si no está en contacto con la piel o que el sensor de proximidad deje de tomar datos si el dispositivo lleva un periodo inmóvil.

Por último se el cuenta el modo **único** en el que el sensor envía un solo evento y se registran exactamente cuando ocurren. Un ejemplo es el sensor de caída.

2.3.5. Emulador

Existen una gran cantidad de emuladores de Android pero el programa *Android Studio* cuenta con uno propio. Este emulador simula el comportamiento de una gran cantidad de dispositivos para poder comprobar la funcionalidad de la aplicación que se esté desarrollando en estos sin la necesidad de tener el dispositivo físico. Se puede elegir entre dispositivos con distintas versiones de Android o distintos tamaños de pantalla.

A parte de las capacidades y características de cada dispositivo también se pueden simular acciones como recibir una llamadas, mensajes, comportamiento de los sensores o simular distintas velocidades de conexión a la red entre otras cosas.



(a) Aplicaciones (b) Ejemplo de aviso

Fig. 2.16. Emulador en Android Studio

2.3.6. Librerías Android

Una librería o biblioteca Android es similar a un módulo de app de Android en cuanto a su estructura, incluye todo lo necesario para hacer funcionar una app como son la carpeta de recursos, código y el manifiesto de Android todo incluido en un archivo AAR. A diferencia del módulo que es una pequeña parte de la aplicación que se puede compilar sin necesidad de compilar la aplicación entera, la librería se compila en un archivo Android Archive que se puede utilizar como dependencia para un módulo o la app posteriormente.

Definimos librería entonces como una recopilación o colección de código preprogramado que se utilizará para extender las funcionalidades de la app que se está desarrollando. Las librerías se utilizan para añadir funcionalidades sin necesidad de

programar código desde cero y se aprovechan de la compartimentación de Java para añadir objetos y métodos a la app de forma simple. Un ejemplo que se utilizará en la aplicación de este proyecto es la necesidad de añadir una gráfica que exprese datos. Android incluye ya unas gráficas programadas pero la visibilidad de los datos y su diseño no son las más adecuadas. De esta forma se añaden librerías al proyecto de Android Studio para su utilización.

Todo proyecto a desarrollar se desarrolla en base a un "Android SDK" siendo esto la versión de funcionalidades básicas y herramientas software necesarias para el desarrollo de una app. Un SDK es en base una colección de librerías y herramientas que se utilizarán en la aplicación. Toda funcionalidad que no se programe en la propia *activity* o en los archivos de una librería importada se encontrará en el SDK.[13]

2.4. Acelerómetro

Un acelerómetro es un instrumento de medida que cuantifica la aceleración propia de el objeto en el que esté situado. La aceleración medida corresponde a el cambio de velocidades de un objeto en un sistema de referencia del propio objeto. La función medir la aceleración es útil y básica en muchos ámbitos profesionales como industria o ciencia pero también están disponibles en muchos dispositivos de uso particular como puede ser en un mando de consola o un dispositivo móvil. Con el desarrollo de la industria estos son cada vez más baratos de producir, más pequeños y más precisos por lo que son más accesibles a un público tanto profesional como particular.

2.4.1. Tipos de acelerómetro

Acelerómetro capacitivo

Este tipo de acelerómetros obtienen los datos por el movimiento causado por la aceleración en las placas de un microprocesador. El movimientos entre placas del condensador hace cambie su capacidad al circular una corriente eléctrica por ellas.

Acelerómetro piezoeléctrico

Estos sensores se basan en las propiedades de los cristales piezo-eléctricos. Estos acelerómetros cuentan con un cristal en su interior que al ejercer una fuerza sobre este, produce una corriente eléctrica. Esta corriente se produce por la variación de la estructura cristalina de los cristales.

Acelerómetro piezorresistivo

Estos acelerómetros son similares a los piezoélectricos sustituyendo el cristal por un sustrato. Esta vez la corriente eléctrica se aplica externamente y el sustrato, dependiendo de la fuerza que tenga actuando en él, varía la resistencia a la corriente. La variación de la corriente con la variación de la resistencia es lo que se mide para obtener la aceleración.

3. DESCRIPCIÓN DE LAS HERRAMIENTAS EMPLEADAS

3.1. Programa para el desarrollo de aplicaciones

Existen una gran cantidad de entornos para el desarrollo de aplicaciones en Android. Estos entornos o IDE (Integrated Development Enviroment) se pueden clasificar en torno a su facilidad de uso y sus capacidades. Cada entorno ofrece ventajas frente al resto y la elección puede hacer que el desarrollador tenga una experiencia más sencilla o con mayores posibilidades. A continuación se verán algunos ejemplos de los entornos más comunes o recomendados.

Eclipse: Este era el IDE recomendado hasta la aparición de Android Studio. Es muy parecido a este y la programación se realiza de forma similar también pero tiene el problema de no ser un IDE específico de Android.

Este IDE es más versátil y puede desarrollar para distintas plataformas pero si se quiere utilizar únicamente para Android, al no estar especializado se producen bastantes errores y se recomienda que la aplicaciones desarrolladas anteriormente esta plataforma se miguen a Android Studio por seguridad y corrección de errores.

App Inventor: Este entorno ha sido desarrollado por el MIT (Massachusetts Institute of Technology) y en un entorno que facilita mucho la interacción con el usuario.

Xamarin: Este entorno está desarrollado por Microsoft y permite que a partir de un mismo código se pueda crear una aplicación para Android, iOS o Windows. No requiere conocimientos de Java ya que se utiliza el lenguaje C

API Multiplataforma: Una API (Application Programming Interface) es una interfaz con la que el usuario interaciona para la creación de as aplicaciones. Si se quiere desarrollar una para distintas plataformas puede interesar utilizar una de estas API para el ahorro de tiempo y por lo tanto dinero.

Muchos de estos API se desarrollan en la web y están basados en el lenguaje de programación Javascript. El diseño de las aplicaciones es similar a "App inventor" en el sentido de arrastrar distintos elementos gráficos para la programación. Ejemplos de estos API pueden ser: jQuery Movile, Titanium o PhoneGap entre muchos otros.

3.2. Android Studio

Este programa creado por Google es el oficial y por tanto hay mucha documentación al respecto y una de las razones por la que se elige. La posibilidad de poder encontrar respuesta de forma rápida es un punto muy positivo y esto de debe a la popularidad de este IDE y la gran comunidad que hay detrás.

Fig. 3.1. Logo de android studio

Las primeras versiones eran muy difíciles de trabajar con ellas y en las actuales iniciarse en este programa puede ser muy tedioso pero con cada versión, la programación se hace más fácil por ejemplo con las ayudas que te da el propio programa y consejos para resolver los errores que ocurren. La versión de Android Studio que se va a utilizar para el desarrollo de la aplicación será la 3.6.3.

Una de las principales razones para escoger un IDE nativo de Android es la mayor capacidad de personalización y acceso a las diferentes opciones disponibles. Inicialmente tiene un grado de mayor complejidad per se entienden y se pueden controlar mejor todos los procesos. Este mayor conocimiento se puede traducir en aplicaciones con mayor rendimiento y optimización.

3.3. Librerías Android de interés

A continuación se enumeran las librerías externas utilizadas en la aplicación para añadir los recursos necesarios al entorno de programación elegido (Android Studio).

3.3.1 GrapfView

[14] Esta librería externa de gráficas destaca por su simplicidad y por eso se eligió inicialmente para representar los datos del acelerómetro.

3.3.2 Storage Chooser

[15] Esta librería permite escoger directorios de forma simple al igual que escoger archivos. Sirve para navegar entre las carpetas y archivos sin salir de la aplicación de modo que se pueda seleccionar un archivo para procesarlo de alguna forma. Con la inclusión de esta librería se resuelve la selección del archivo a analizar de una forma que es sencilla para el usuario final.

La versión en la aplicación 2.0.4.4 bajo la licencia "Mozilla Public License v2" con la condición de notificar de su uso en el código.

3.3.3 Math FFT

Esta librería [16] se incluye para realizar el cálculo de la transformada rápida de Fourier (FFT). Incluye el algoritmo de la transformada y aparece para su programación como una función en la que se le introducen datos de entrada y devuelve el resultado de la transformada. Este resultado es el que se utilizará para sacarlo por pantalla a través de unas gráficas tras procesarlo.

3.3.4 HelloCharts

Se utiliza otra librería de gráficas (HelloCharts[17]) para los resultados de la transformada y las medidas temporales. Se opta por estas gráficas por su simplicidad y facilidad de ver los resultados. En ambas gráficas la utilidd de ampliar sobre los datos es muy útil y es estas gráficas está muy bien implementada.

Para la gráfica de la FFT conviene la utilidad de que al pulsar sobre uno de los puntos, aparezca su posición por pantalla en un mensaje para obtener la medida exacta. Mientras que la gráfica de las medidas temporales interesa la ampliación horizontal para observar los datos de las aceleraciones con mayor detalle y encontrar un patrón de las vibraciones.

3.4. Otros programas

3.4Fusion 360

Este es un programa de diseño asistido por ordenador basado en la nube del desarrollador *Autodesk*. Una de las ventajas que cuenta este programa es una licencia de uso gratuita para uso personal no comercial que será la que se utilizará aunque también se dispone de una licencia gratuita para estudiantes. Debido a esto y todas las capacidades del programa que tiene el soporte de *Autodesk* es el elegido para distintos diseños 3D que se vayan a necesitar en el transcurso del proyecto.

Este programa tiene la opción de realizar renderizados con iluminación real por lo que se utilizará este apartado par hacer comparativas del diseño con el resultado final. Para las piezas a imprimir se exportan los archivos en formato *stl* que es un formato que traduce la geometría de la pieza a una malla y dependiendo de la densidad de los puntos de esta malla, la pieza tendrá más o menos detalle. Como se puede ver en la figura

3.4Cura

Este programa es de código abierto y está desarrollado por *Ultimaker*, una reputada empresa en el campo de la fabricación aditiva, produciendo también impresoras. es gratuito e incluye una interfaz muy amigable para el usuario aunque tiene muchas opciones avanzadas si se requieren.

Este programa se utiliza para el procesado de los archivos *stl*, incluyendo la modificación de gran cantidad de parámetros. Estos parámetros varían para el material utilizado, la calidad buscada, la velocidad de impresión y variaciones en la resistencia de la propia pieza. Su motor hace el cálculo de cada capa de la impresión de forma rápida.

Fig. 3.2. Proceso para la impresión de una pieza

Una vez se elijan los parámetros de la impresión, todos estos se incluyen en un archivo *gcode* que contiene la posición de la punta por donde se expulsa el plástico, la temperatura y la cantidad de plástico que se expulsa durante toda la impresión que en el mayor de los casos duran horas. Este archivo es el que se manda a la impresora 3D y el procesador de la impresora lo interpretará para realizar la impresión.

3.4Matlab

Matlab es un sistema de cálculo que cuenta con su propio lenguaje de programación y entorno de programación. Su sistema se basa en el cálculo mediante matrices para solucionar cualquier problema computacional. Tiene una gran cantidad de herramientas para resolver problemas de ingeniería o científicos. Estas herramientas van desde la clasificación de datos, el análisis de señales a la expresión de resultados de forma gráfica para que sena más fáciles de comprender.

Para este proyecto se utilizará Matlab para la realización de gráficas para facilitar la explicación de varias cuestiones además de la realización de algún cálculo para comprobar unos resultados obtenidos fuera de Matlab.

4. METODOLOGÍA

4.1. Etapas del proyecto

Por las características de este proyecto no se puede hacer un plan con las etapas de este con tiempos de duración muy definidos. Debido a la necesidad de hacer un estudio muy profundo debido a la necesidad de aprender la programación y por desconocimiento de la duración se esta, se realizará una planificación únicamente por fases.

Estas fases serán diferenciales en cuanto a tareas a realizar en cada una de ellas. La **primera fase** es la documentación sobre *Android* y programación para iniciar el desarrollo de la aplicación. Se espera que esta fase no acabe al iniciar la programación y se extienda durante toda la programación de esta con nuevas dudas o cuestiones que deberán ser consultadas. Si el proyecto se extiende más allá de la aplicación, esta fase puede extenderse aún más.

La **segunda fase** es la programación de la propia aplicación que se podrá iniciar gracias a la documentación buscada previamente. Inicialmente esta fase ocupará un periodo de tiempo hasta que la aplicación cumpla los requerimientos establecidos y pueda dar un resultado en cuanto al análisis de datos. Al igual que la primera fase, esta fase puede extenderse con cambios de funcionalidades o al añadir nuevas a posteriori.

Solo si la segunda fase se ha completado con éxito y se ha desarrollado una aplicación funcional se pasará a una **tercera fase**. En esta fase se realizarán distintos tipos de pruebas para comprobar la utilidad de la aplicación y los acelerómetros que incorporan los dispositivos móviles. Idealmente se realizarán pruebas en entornos reales donde los resultados de las mediciones realizadas tengan una aplicación real.

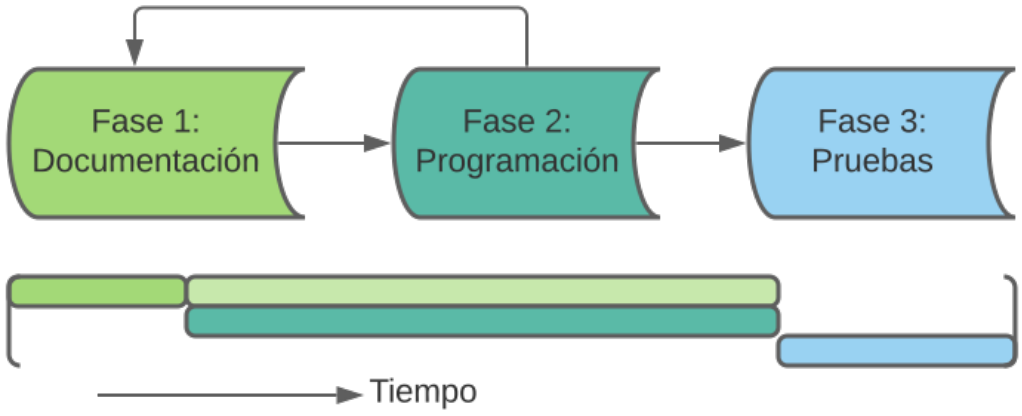


Fig. 4.1. Fases del proyecto

4.2. Antecedentes

Para encontrar antecedentes al tema del proyecto se buscaron formas de medir las vibraciones de manera fácil y sobre todo accesible. No se consideran los aparatos que aunque por su diseño sean cómodos de usar, por su elevado precio sean de difícil acceso a un gran número de personas.

Para encontrar antecedentes se buscaron aplicaciones con funciones similares a las que se quieren implementar en la que se va a desarrollar como son el acceso a los sensores del dispositivo móvil y en concreto al acelerómetro. Las funcionalidades que se buscaron concretamente fueron el acceso y muestra de los datos del acelerómetro, el guardado de dichos datos y su posterior análisis. Se valoraron otros aspectos como la claridad en la que se muestran dichos datos y facilidad de uso en general.

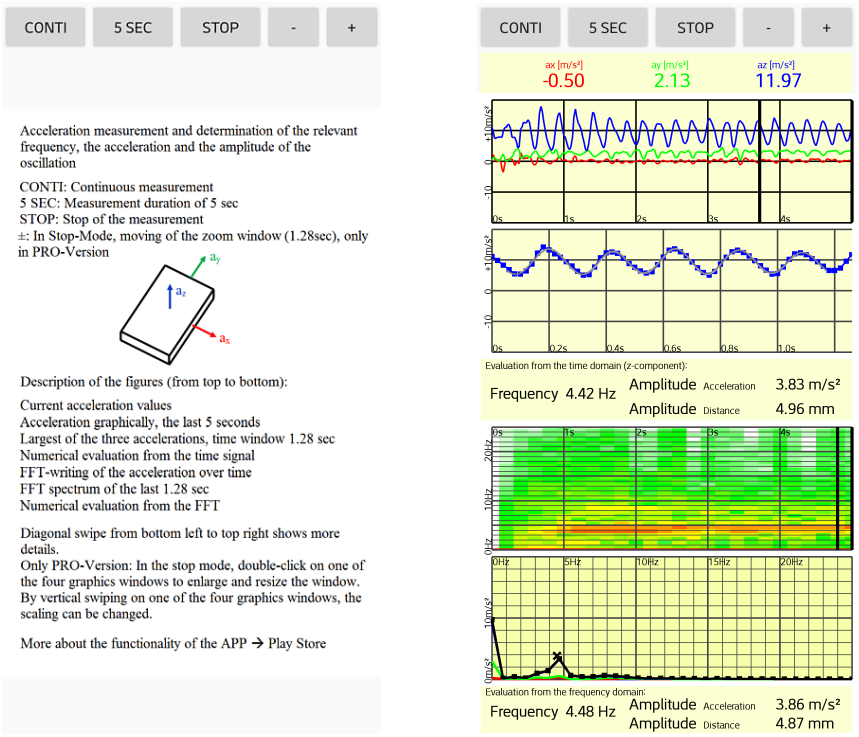
4.2.1. Aplicaciones previas y documentación

Previamente a iniciar el proyecto en Android Studio se realizaron otros proyectos que no tienen relación con el proyecto final pero fueron muy útiles para familiarizarse con el desarrollo de aplicaciones, Java y Android Studio. También se dedicó una cantidad importante de tiempo a documentarse y aprender que es Android y descubrir el potencial que tiene y lo que se puede hacer. La primera tarea es buscar información para situar que es lo que se quiere hacer y de que forma conseguirlo.

Se buscaron libros sobre Android y en muchos se referían a Android Studio para desarrollar las aplicaciones y fue uno de los motivos por los que se eligió tal programa para este proyecto. En el índice de estos libros aparece un resumen de los puntos a mirar si se quiere desarrollar una aplicación, no todos van a ser a ser utilizados pero es aconsejable saber de su existencia y de los que son capaces. Por ejemplo la aplicación en un principio no se va a conectar a internet y por lo tanto funcionalidades que requieren conexión a internet no se utilizarán y no se profundizará en ellas.

Vibration Analyzer

Esta aplicación se puede encontrar en la tienda de aplicaciones de Android de forma gratuita. Se compone solo de dos pantallas, en la primera hay una descripción de lo que la aplicación puede hacer y en la segunda las mediciones del acelerómetro de forma continua representándolo en distintas gráficas.



(a) Menú Principal (b) Medidas

Fig. 4.2. Capturas de la aplicación Vibration Analyzer

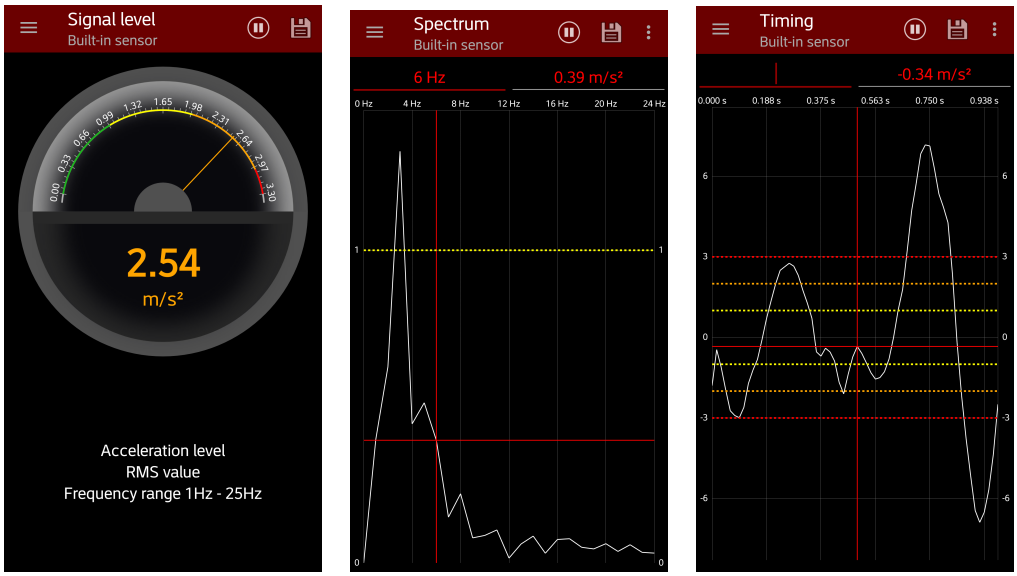
Se genera una vibración manualmente elevando el dispositivo a un ritmo constante con la pantalla hacia arriba para que uno de los ejes sea el predominante (en este caso el eje X) y se obtiene lo que se puede ver en la figura 4.2 apartado b. Es un buen ejemplo de un buen diseño en el que incorporar varios elementos siendo todos visibles. Se utilizan muestras de 5 segundos en los que se miden los datos del acelerómetro y se analizan pudiendo ver gráficas para la amplitud de la aceleración y la frecuencia de la vibración.

Se consiguen datos de la frecuencia de la vibración más significativa, tanto su frecuencia en hercios como su amplitud siendo valorados en función del tiempo y la frecuencia. Para la frecuencia se utilizaría una transformada de Fourier con un número de datos de muestra fijo con los nuevos datos de cada medición para de esa forma conseguir que las dos gráficas inferiores referidas a la frecuencia vayan actualizando sus datos.

Es una aplicación útil y bien diseñada si se sabe lo que se busca pero dicho esto no cumple con las funciones que se buscan. El objetivo es el análisis de las vibraciones para obtener frecuencias naturales y esta aplicación no lo permite. Tiene una sola frecuencia significativa, no se puede variar el tamaño de la muestra, no se pueden exportar ni los datos medidos ni los resultados de modo que si se utilizase para medir una vibración, no se podrían analizar los datos a posteriori. Estas necesidades serán las que intenten dar la aplicación a desarrollar.

mVibe

Esta aplicación es de la compañía Alitec, una empresa polaca que fabrica elementos para la supervisión y mantenimiento de distintos sistemas. También desarrollan *software* para el análisis de las medidas y han desarrollado esta aplicación para lo mismo. Esta aplicación es capaz de medir datos del acelerómetro y sacar el espectro tanto temporal como temporal con algunas restricciones. No se pueden sobrepasar los 25 Hz en el análisis de frecuencias que se actualiza cada medio segundo y no se tiene control sobre a qué frecuencia toma los datos el acelerómetro ni forma de verlo por pantalla. Las mediciones en la figura 4.3 (c) solo se muestra lo ocurrido en el último segundo y por lo tanto no se puede hacer una medida continuada.



(a) Nivel aceleración (b) Espectro frecuencia (c) Espectro temporal

Fig. 4.3. Capturas de la aplicación mVibe

A la hora de guardar las mediciones la aplicación te da una opción pero guarda lo que se ve por pantalla y en el caso temporal como frecuencial no es suficiente estos datos ya sea por tener una franja de tiempo (1 segundo) o frecuencia (25 Hz) muy limitada.

La misma compañía produce un producto para medir vibraciones que incluye un acelerómetro entre otras funciones y se conecta con el dispositivo de forma inalámbrica. En la propia aplicación no se recomienda usar el acelerómetro del dispositivo al no ser fiable o precios y te recomiendan hacerte con un sensor que siendo mejor el precio de este hace que el objetivo de que sea ampliamente accesible para un gran número de usuarios no se cumple.

4.3. Marco regulador

En cuanto a la normativa aplicable, se tendrá en cuenta las normas que tienen las aplicaciones en el caso de que se quisiesen vender o publicar. Inicialmente esta no es la intención con la aplicación que se va a desarrollar pero conviene conocer la normativa en caso de que se decida publicar.

4.3.1. Permisos y protección de datos

En el estado final de la aplicación que se desarrolla en este proyecto, no es necesario tener en cuenta la protección de datos ya que la aplicación sólo tiene acceso a los datos localmente y no utiliza la conexión a internet del dispositivo móvil para comunicación con el exterior. Dicho esto, en el futuro es posible que se implemente alguna actualización con alguna funcionalidad extra que necesite tener en cuenta la protección de los datos del usuario.

Los dispositivos móviles son una gran fuente de datos y estos son de muchos tipos. Existen los datos como tal que pueden ser imágenes, archivos, correos electrónicos, archivos de audio etc. o datos procedentes de sensores tanto internos como externos como pueden ser un sensor de actividad o de ritmo cardiaco. Las aplicaciones utilizan estos datos para los procesos internos y mejorar el servicio al usuario pero también pueden ser usadas de una forma no deseada por el usuario.

En el contexto actual de desarrollo tecnológico, la *AEPD* (Agencia Española de Protección de Datos)[18] y otros órganos legislativos han tenido que adaptarse a las nuevas tecnologías y los nuevos retos que se derivan de la generación de grandes cantidades de datos y la nueva gestión de la privacidad de los usuarios. Para ello entro en vigor en diciembre de 2018 la *Ley Orgánica de Protección de Datos y Garantía de Derechos Digitales o LOPDGDD*. Esta ley sustituye a la Ley de Protección de Datos de 1999 para adoptar criterios conjuntos con la ley de protección de datos europea también aprobada en 2018.

Obligaciones del desarrollador

Para cumplir con la ley aplicable garantizar las buenas prácticas se debe cumplir con la normativa general aplicable a cualquier tratamiento de datos además de normativa específica de las aplicaciones y *software*. Esta normativa se basa en la **transparencia** con el usuario, este debe conocer el uso que van a tener los datos con claridad.

Antes de cualquier acceso a los datos de forma externa el usuario debe ser informado de forma fácil y este debe dar su consentimiento. Este petición para el uso

de datos se debe hacer antes de instalar la aplicación en el caso de que se descargue de una fuente oficial dónde se debe detallar a qué tipo de datos se va acceder, pero también puede hacerse la petición dentro de la aplicación. La petición debe incluir una **política de privacidad** o uso de datos de forma clara e inteligible.

Para el caso específico de la aplicación que se va a desarrollar en este proyecto no será necesario crear ningún tipo de política de tratamiento de datos debido a que no se van a almacenar de forma externa y por tanto el desarrollador no va a tener acceso a ellos. Solo se requerirá permiso que se solicitará durante el uso de la aplicación para el acceso a datos (mediciones) o escribir las mediciones en la memoria interna del dispositivo móvil.

4.3.2. Play Store

Play Store es la plataforma o tienda oficial que tienen los dispositivos con sistema operativo Android para instalar nuevas aplicaciones. Las aplicaciones también se pueden instalar desde fuentes de terceros bajo la responsabilidad del usuario. El desarrollador de las aplicaciones debe cumplir una serie de normas para que las aplicaciones puedan formar parte del catálogo del Google Play y sean accesibles a todos los usuarios.

La normativa en este aspecto se organiza en "Las políticas del programa de desarrolladores" [19] y "El acuerdo de distribución para desarrolladores", cuyo objetivo es definir unos estándares de lo que puede causar algún daños y proveer de una plataforma segura para los usuarios. Algunas de las políticas que tiene la normativa son las siguientes:

- *Contenido inadecuado o restringido*:

  Este contenido engloba a contenido de tipo sexual, que puede incitar a la violencia o comportamientos peligrosos.

  No se permiten aplicaciones que puedan exponer a los usuarios a productos financieros sin regular que pueden resultar en fraude, incluyéndose también los tratos con criptomonedas como por ejemplo el BitCoin. Aunque se permitan las aplicaciones relacionadas con las apuestas, estas están muy reguladas.

  En aplicaciones en las que el contenido es generado por el usuario, este contenido debe de estar controlado y se deben aceptar unos términos de uso que garanticen que este contenido no sea malicioso.

- *Propiedad intelectual*:

  No se permiten aplicaciones que infrinjan los derechos de propiedad intelectual de otros usuarios. En cuanto al uso de de contenido con derechos de autor, se

deben de tener los permisos o utilizar contenido con un tipo de licencia que permita su uso. Es importante antes de incluir la aplicación para su distribución que todo su contenido es original o tiene una licencia que permite esta distribución.

- *Suplantación de identidad*:

  Quedan prohibidas las aplicaciones que se hacen pasar por otras para obtener algún beneficio. Esto también aplica a desarrolladores que se hacen pasar por otros. Se debe tener en cuenta que el icono de la aplicación, nombre y otros elementos identificativos no lleven a confusión con otra aplicación y otros servicios.

- *Software malicioso (malware)*:

  La normativa respecto a esto es simple, la aplicación no debe incluir ningún elemento en su programación que pueda causar algún daño. Entre las actividades de estos elementos destacan: el intento de ganar control sobre el dispositivo, acceso a datos sin consentimiento, provocar daños físicos al dispositivo o permitir un ataque externo tener acceso al dispositivo.

- *Privacidad*:

  Lo que se hace con los datos que la aplicación obtiene debe de ser explicado claramente al usuario y además de cumplir la normativa de protección de datos del lugar donde se desarrolle la aplicación, también debe de cumplir unos requerimientos extra.

- *Permisos*:

  Los permisos que pide la aplicación para su funcionamiento deben ser los mínimos necesarios para ello y el funcionamiento de la aplicación y permisos necesarios deben de estar detallados en la descripción de la aplicación. Los datos conseguidos con estos permisos solo deben usarse para lo que el usuario ha consentido y nunca para otras tareas o su venta a terceras partes.

  Hay algunos tipos de permisos que son considerados más peligrosos por el acceso que tienen a los datos dispositivo. Estos son el permiso de acceso a la localización, al registro de llamadas y los que permiten el acceso a los datos.

Con la normativa de esta tienda de aplicaciones, el desarrollador debe comprobar que cumple con todos los apartados de la normativa. En caso negativo, la aplicación puede ser eliminada de la tienda y dependiendo de la infracción puede acarrear problemas legales.

4.4. Entorno socioeconómico

4.4.1. Android

Actualmente el sistema operativo Android es el más extendido en el mundo en dispositivos móviles, por este motivo será el elegido para desarrollar la aplicación de modo que un mayor número de usuarios puedan tener acceso. Pero esto no siempre ha sido así, a continuación se va a analizar la evolución de la dispersión de los dispositivos móviles con sistema Android en el tiempo.

Los datos se van a obtener de un servicio web llamado *StatCounter* [20] que se encarga de analizar el trafico de visitas a una determinada página web. StatCounter después desvela al público los datos globales obtenidos de todos sus clientes de forma gratuita y de simple acceso. En general se contabilizan más de diez mil millones de visitas mensuales en las más de dos millones de páginas web. De este modo se obtiene una muestra muy grande y que se pueda adaptar más a la realidad.

Se ha elegido este método por su gran alcance en número de usuarios y aunque ahora sea popular con un gran número de usuarios no se puede garantizar que los datos más antiguos sean igualmente precisos, debido a que este servicio pudo tener un número más reducido de usuarios en sus inicios.

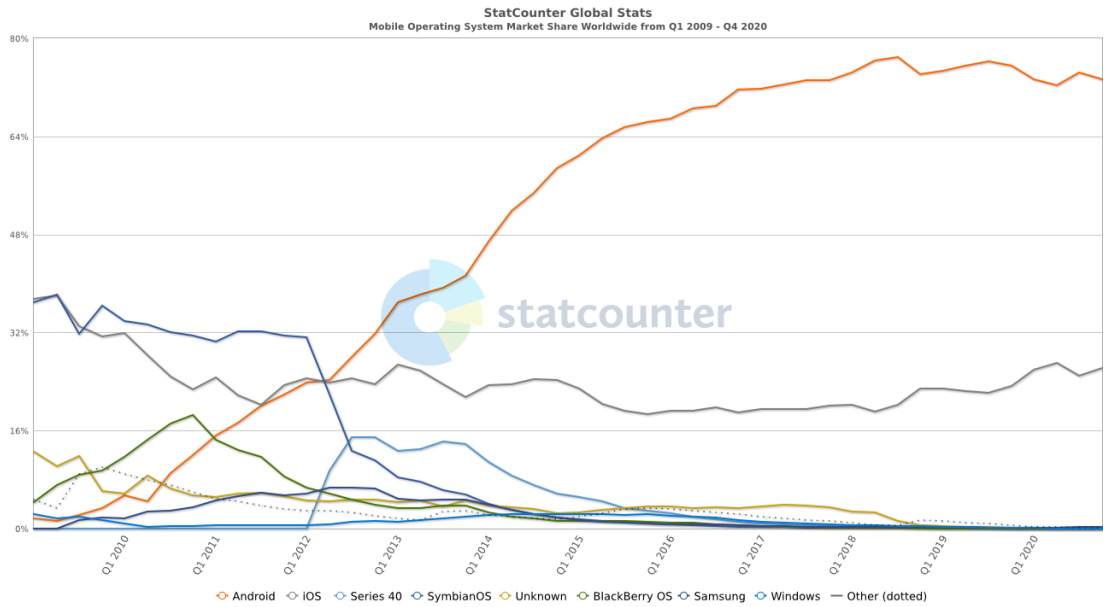


Fig. 4.4. Cuota de mercado sistema operativo mundial

En la figura 4.4 se puede apreciar la evolución de los sistemas operativos tanto de móviles y tabletas desde el primer trimestre de 2009 hasta el cuarto trimestre de 2020. Inicialmente el mercado se lo repartían los sistemas iOs y Symbian que juntos agrupaban casi el 70 % del mercado de dispositivos móviles. Destaca en naranja el ascenso continuado del sistema Android que ha ido superando en cuota de mercado

a todos sus competidores.

En verde se puede ver el auge de 2009 y 2010 de Blackberry OS y su rápida caída causada por las grandes ventas de Apple y el auge de Android lo que causó una gran competencia. Después de esto Android pasó tanto a iOS como a Symbian a principios de 2012 coincidiendo con el lanzamiento del sistema de Microsoft para dispositivos móviles. A Windows para dispositivos móviles le ocurrió algo parecido a lo que le ocurrió a Blackberry OS, la competencia y la falta de modernización lo convirtieron en un sistema operativo obsoleto y que a 2019 se encuentra casi en desuso.

La figura 4.4 demuestra que no esta garantizada la permanencia ni la dominancia de ningún sistema en el tiempo aunque Android está muy establecido actualmente y sería más difícil que decayese en el tiempo.

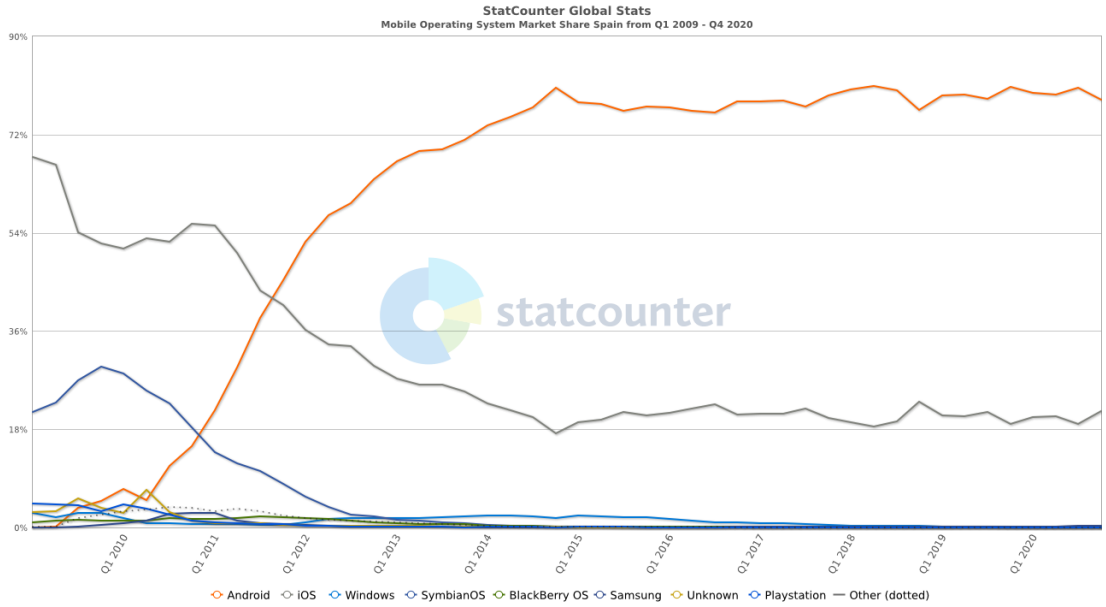


Fig. 4.5. Cuota de mercado del sistema operativo en España

Debido a que la aplicación a desarrollar inicialmente será en español y el alcance también será en España se analizará en concreto el mercado español. En este mercado se observan algunos cambios que destacan frente al mercado mundial de sistemas operativos. El sistema Android es aún más predominante en esta región geográfica, lo que es positivo ya que puede tener mayor alcance.

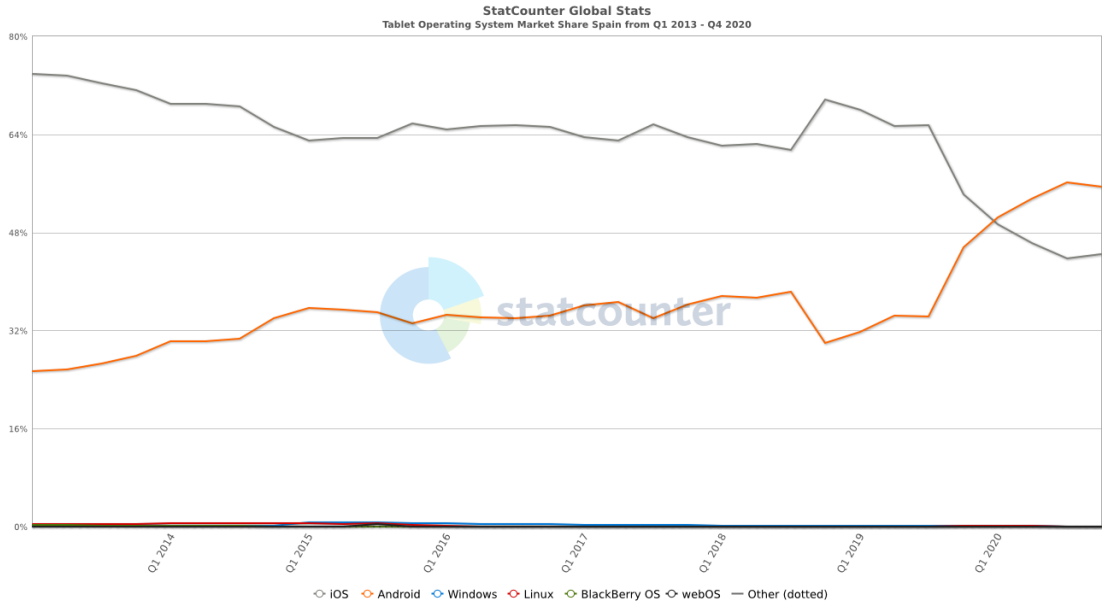


Fig. 4.6. Cuota de mercado del sistema operativo en tablets en España

La figura 4.6 indica la cuota de mercado del segmento de dispositivos móviles que son las tabletas o "tablets" en España. Desde que se tienen registros las tabletas del fabricante apple son las que lideran en ventas en España con un cambio de tendencia este último año. Para la utilidad que se va a requerir por lo general las tabletas tienen acelerómetros de menor calidad que un móvil de precio similar. Esto se debe a que no se llevan encima constantemente y las funciones de registro de la actividad no se utilizan.

Teniendo en cuenta esto y su mayor tamaño y peso hace que tomar una medición de aceleración sea más difícil. En cambio, para realizar los cálculos y procesamiento de las señales sí que son indicadas ya que el mayor tamaño ahora nos beneficia pudiendo ver las gráficas y resultados del análisis de señales con mayor detalle.

Versión de Android

La versión de Android que tenga el dispositivo también es importante a la hora de desarrollar la aplicación debido a que determinadas funciones que se incluyen con cada actualización, si son incluidas en la aplicación pueden hacer que la aplicación sea incompatible con el sistema que tenga el dispositivo. Por ejemplo si una función se incluyó en Android 6.0 y se quiere instalar en un dispositivo con Android 5.0 pueden surgir problemas.

De la misma forma aunque en menor medida se pueden producir errores si se instala una aplicación desarrollada para unan versión anterior que la que tiene el dispositivo. La aplicación se podrá instalar pero puede que no aproveche completamente el sistema al haberse añadido alguna característica nueva a Android posteriormente.

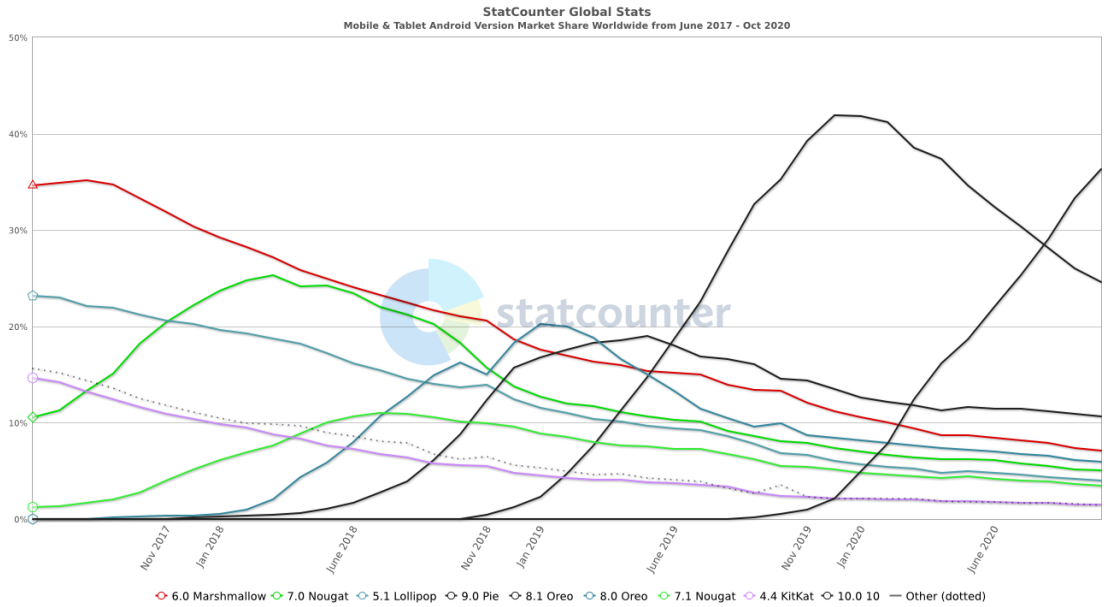


Fig. 4.7. Cuota de mercado de la versión de Android

Ventas por fabricante

TABLA 4.1. VENTAS POR FABRICANTE 1Q19 (MILES DE UNIDADES)

Fabricante	1Q19	1Q19 Cuota mercado (%)	1Q18	1Q18 Cuota mercado (%)
Samsung	71721	19.2	78562	20.5
Huawei	58436	15.7	40426	10.5
Apple	44568	11.9	54058	14.1
OPPO	29602	7.9	28173	7.3
Vivo	27368	7.3	23243	6.1
Otros	141405	37.9	159037	41.5
Total	**373001**	**100.0**	**383503**	**100.0**
Fuente: Gartner (Mayo 2019) [21]				

Con datos más recientes pertenecientes al primer cuatrimestre del año 2019 comparado con el mismo cuatrimestre del año anterior se puede notar una bajada del 2.1 % en ventas de dispositivos móviles quedándose en un número de ventas totales de 385 millones de unidades. Huawei mantiene la segunda posición como fabricante con un número mayor de ventas siendo este el fabricante con un mayor crecimiento.

Según este análisis por parte de Gartner la demanda de smartphones de gama alta es baja comparada con otros modelos más asequibles. Teniendo en cuenta que los modelos de gama alta son los que incorporan acelerómetros más avanzados tecnológicamente hace que la cuota de dispositivos de los que se puede sacar un mayor rendimiento al acelerómetro sea por el momento baja. Aún así, los avances en tecnología continúan y lo que se consideraba gama alta hace unos años actualmente ya no lo es de modo que la sensibilidad y tasa de muestreo máxima de los acelerómetros integrados en los smartphone aumenta con el desarrollo de los propios smartphones.

Este desarrollo de nuevas tecnologías va fomentado por la competencia entre distintos fabricantes y lo que interesa en cuento a este proyecto es la competencia entre fabricantes que incorporen el sistema Android en sus dispositivos.

Fig. 4.8. Logo del fabricante Huawei

El cambio más importante lo protagoniza el fabricante Huawei que consigue el mayor crecimiento de entre los cinco primeros fabricantes mundiales, con un 44.5 % y 58.4 millones de unidades vendidas. Creció sobre todo en la región de China en la donde tiene un 29.5 % de cuota de mercado. Pero este crecimiento se ve afectado por las sanciones de EE.UU. a la marca China prohibiendo a empresas estadounidenses hacer negocios con Huawei. Esto afecta al hardware como los procesadores de la empresa Quadcomm o Intel [22][23] entre otras pero lo que más afecta es el veto con Google.

La empresa Google es propietaria del sistema operativo Android y aunque este es de código abierto y cualquiera puede acceder a él, es prácticamente imprescindible contar con los servicios de Google para sacar el máximo rendimiento a Android. Las actualizaciones que recibe de Google dejará de recibirlas en el momento de su salida y por lo tanto Huawei puede esperar a que la nueva versión salga en código abierto o crear su propio sistema operativo que rivalice con Android. Esto puede generar una mayor competencia en un entorno dominado por Google y Apple pero también una mayor segregación teniendo en cuenta las buenas ventas de Huawei y que solo en China un tercio de todos los smartphones son de Huawei.

Como consecuencia directa al proyecto tiene que un porcentaje grande de smartphones dejen de ser compatibles al dejar de llevar el sistema Android. Cabe esperar que haya una forma relativamente fácil para exportar las aplicaciones al nuevo sistema operativo e incluso que sea más fácil de exportar de lo que sería actualmente a el sistema iOS propiedad de Apple. No cabe duda de que si este fuera el escenario final los dispositivos que podrían tener acceso a la aplicación disminuirían significativamente aunque la población actual de móviles Android es suficientemente amplia como para seguir siendo un porcentaje alto y aún no se sabe el desenlace que tendrá tanto Huawei como otros fabricantes Chinos.

4.5. Resultados esperados

El primer resultado del proyecto es una aplicación funcional. Sin importar todavía si los datos son útiles, la aplicación tiene que cumplir los requisitos esperados y obtener mediciones del acelerómetro.

Más tarde cuando se pruebe la aplicación en entornos reales se espera que el resultado del análisis de las vibraciones sea capaz de detectar la frecuencias que destacan en la medición.

Si las mediciones hechas no concuerdan con lo esperado, también es un resultado válido. Se esta utilizando un componente de los dispositivo móviles para un uso que no se ha diseñado por lo que sería normal que no se obtenga los mismos datos que proporcionaría un acelerómetro externo de grado profesional.

Como resumen, en un inicio los resultados son completamente desconocidos y cualquier resultado o información sobre el uso o capacidades de los acelerómetros de un dispositivo móvil para el análisis de vibraciones es un resultado correcto. Así que, o los resultados son coherentes comparándolo con un acelerómetro de mayor calidad o por el contrario los acelerómetros de los móviles están lejos todavía de estas capacidades y puede que en un futuro lo estén.

5. DESARROLLO DE LA APLICACIÓN

En este capítulo se detallará el desarrollo de la aplicación Android sobre el software Android Studio que brinda las funcionalidades buscadas. Este desarrollo se hará por bloques según las tres pantallas de la aplicación que son: entender el funcionamiento del acelerómetro en Android; guardado de las mediciones del acelerómetro y su exportación; y por último la importación de estas mediciones y su análisis.

5.1. Estructura de la aplicación - MainActivity

En esta *Activity* se encuentra la estructura básica de la aplicación y es lo primero que se ve al iniciar la aplicación. Esta *activity* es la principal y servirá como núcleo desde el que se accederá a el resto de *activities* mediante botones en un inicio. Una *activity* es una pantalla de la aplicación y se define por dos archivos, un archivo *xml* y un archivo *java*.

Esta pantalla principal cuenta con un botón para ir a cada una de las otras pantallas o *activities* y sirve para ver la estructura de la programación en el programa Android Studio y la interacción entre los dos archivos que la componen. En la figura 5.1 se puede ver la estructura de las pantallas que forman la aplicación y su funcionalidad básica. En los próximos apartados se desarrollan estas pantallas.

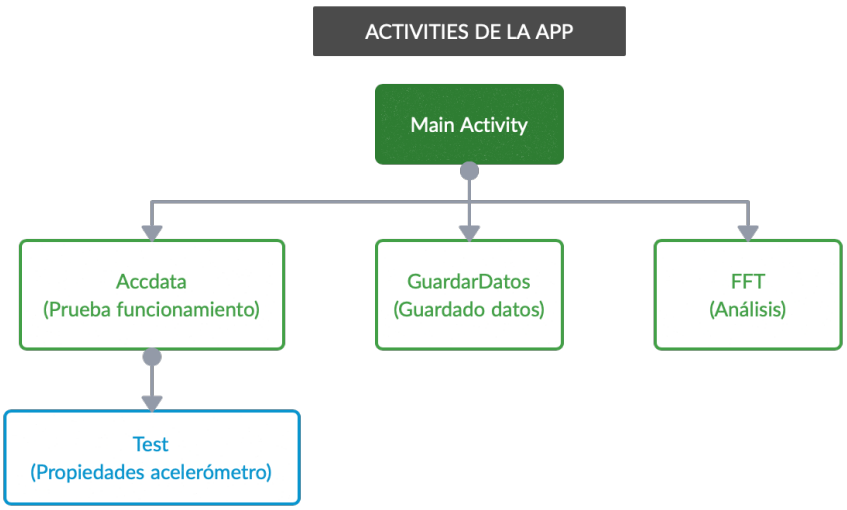


Fig. 5.1. Diagrama de las *activities* de la aplicación

Parte XML

La apariencia de esta *activity* viene dada por el archivo *xml* codificado en XML que incluye la organización de los distintos elementos, en este caso únicamente botones.

En el código *xml* viene toda la información referente a la representación visual en la pantalla del dispositivo en que se utilice la aplicación. Esta apariencia viene definida por el *Layout* elegido, en este caso se trata de un *Layout* tipo *Linear* en el que los elementos se muestras uno a continuación del otro y al elegir orientación vertical, uno debajo del otro.

Y como cada *activity* tiene la apariencia definida en el archivo *.xml*, las funcionalidades estarán en el archivo *.java*. Las funcionalidades que buscamos todas tienen que ver con desplazarnos a otra *activity* de modo que hacer *click* sobre el botón abandonemos el menú principal.

Como se puede ver al definir cada botón, estos se pueden definir modificando varios parámetros como son el estilo, tamaño o todo lo referente al mensaje como es el texto, el tipo de letra o el tamaño de esta. Para su funcionalidad, lo más importante es el parámetro *onClick* que determina la función de este botón al pulsarlo.

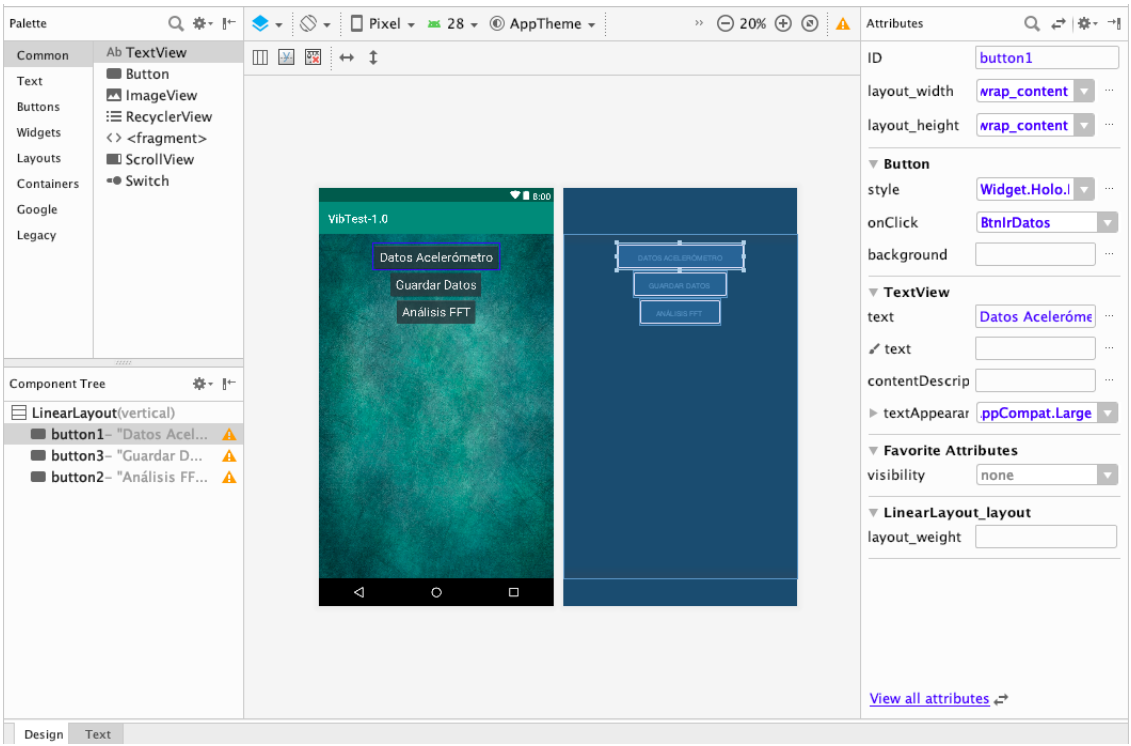


Fig. 5.2. Entorno Design del archivo xml.

A la hora de escribir el código en el archivo xml a través de Android Studio se puede hacer mediante dos entornos: el entorno *Design* y el entorno *Text*. Co-

mo se puede ver en la figura 5.2, mediante el entorno *Design* se puede hacer una previsualización a la vez que se colocan los elementos del diseño.

En este entorno se simplifica bastante la forma de añadir elementos o elementos gráficos al *layout*, como en esta caso botones. En la parte superior izquierda de la figura 5.2 hay un recuadro llamado "Palette" en el cual están los elementos gráficos principales separados por categorías y para añadirlos se puede hacer de forma tan simple como arrastrar un elemento a la parte en la que aparece la pantalla a la derecha. Estos componentes se añadirán al árbol de componentes que aparece abajo a la izquierda dentro de su correspondiente *layout*, en este caso uno de tipo lineal.

En el centro se puede ver una previsualización final y otra más simple en el que aparecen los elementos. En la previsualización simple se pueden mover los elementos arrastrándolos y esta acción modificaría el código automáticamente.

Por último en esta interfaz se puede de modificar los distintos atributos de los elementos de forma que al seleccionar uno te aparecen sus atributos. En el caso de la figura 5.2 los atributos del botón aparecen a la derecha y son modificables más fácilmente. Estos atributos son la apariencia, la posición y el más importante

Aunque esta forma de crear la interfaz sea más fácil genera un código no optimizado y hay que pasar a la interfaz *Text* (figura 5.3) para dejarlo bien ordenado y así sea fácil de encontrar los elementos y modificarlos.

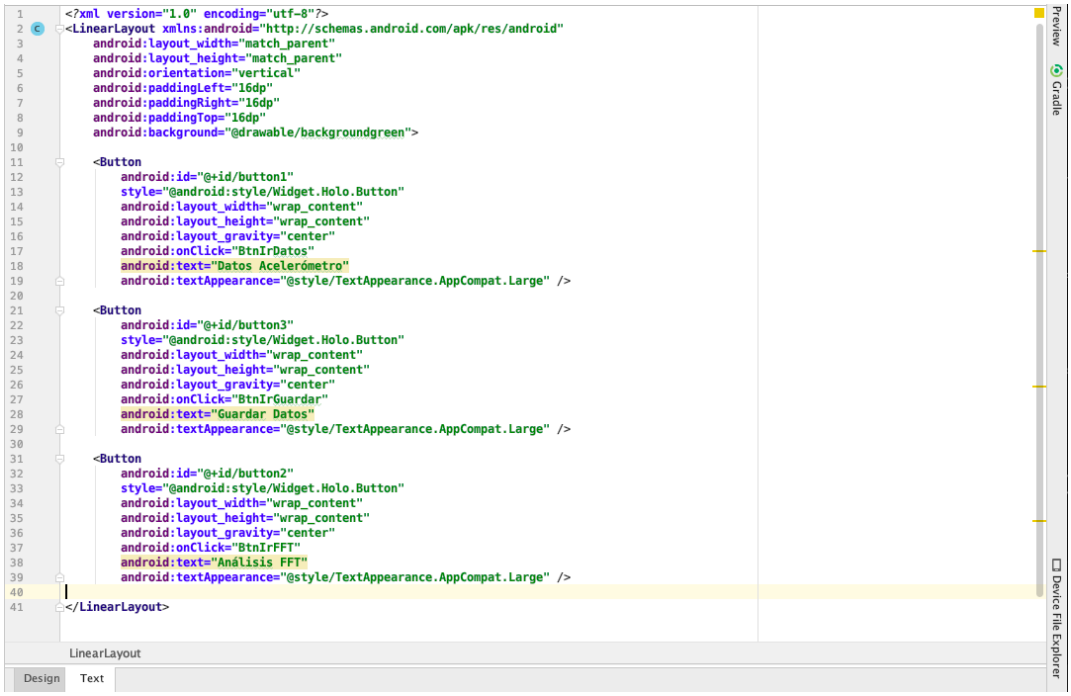


Fig. 5.3. Entorno Text del archivo xml.

Parte java

```
1  package com.example.jorge.vibtest_10;
2
3  import android.content.Intent;
4  import androidx.appcompat.app.AppCompatActivity;
5  import android.os.Bundle;
6  import android.view.View;
7
8  public class MainActivity extends AppCompatActivity {
9
10     @Override
11     protected void onCreate(Bundle savedInstanceState) {
12         super.onCreate(savedInstanceState);
13         setContentView(R.layout.activity_main);
14     }
15     public void BtnIrDatos (View vista){
16         Intent intent= new Intent (this, AccData.class);
17         startActivity(intent);
18     }
19
20     public void BtnIrFFT (View vista){
21         Intent intent= new Intent (this, FFT.class);
22         startActivity(intent);
23     }
24
25     public void BtnIrGuardar (View vista){
26         Intent intent= new Intent (this, GuardarDatos.class);
27         startActivity(intent);
28     }
29 }
```

CÓDIGO 5.1. MainActivity.java

Para definir el código del archivo *.java* correspondiente a cada actividad primero se definirá el paquete o *package* al que pertenece de manera que todas las actividades que se creen a partir de ahora puedan funcionar juntas relacionándose entre ellas. Al inicio también se importarán todas las librerías que se vayan a utilizar ya sean las que incluye el propio Android Studio como otras externas.

Se define la clase como pública de modo que otras puedan interactuar con lo que progrmado en ella. El método **onCreate** sirve para inicializar la actividad y aplicarle la apariencia que se ha definido previamente en el archivo *.xml* mediante el método *setContentView()*. En el método *onCreate()* se incluyen también los procesos que se quiera que inicien al crear la *activity*.

Por cada botón se define un método *void* o que no devuelve un resultado. Al seleccionar el botón elegido, este activa el método designado para cada botón y esta función se realizará mediante un *intent*. En este caso se realiza el *intent* para pasar de una pantalla a otra.

5.2. Medida de datos - *AccData.java*

La función de esta *activity* es entender cómo funciona su programación y acceder al acelerómetro del dispositivo y leer sus datos. Aunque no sea esencial para el funcionamiento principal que se busca (tomar, guardar y analizar vibraciones), se mantendrá en la aplicación para mostrar cómo se accede a los datos del acelerómetro y realizar una gráfica de las mediciones.

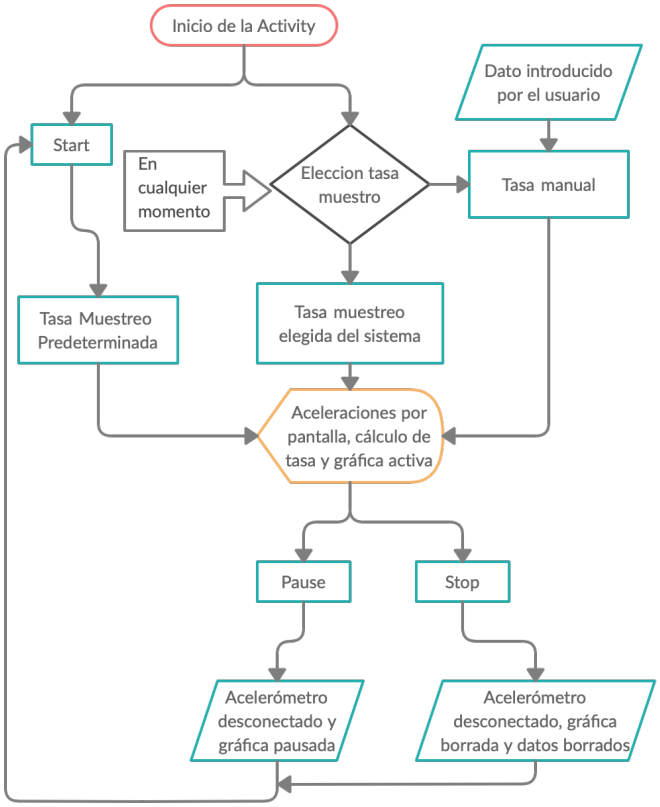


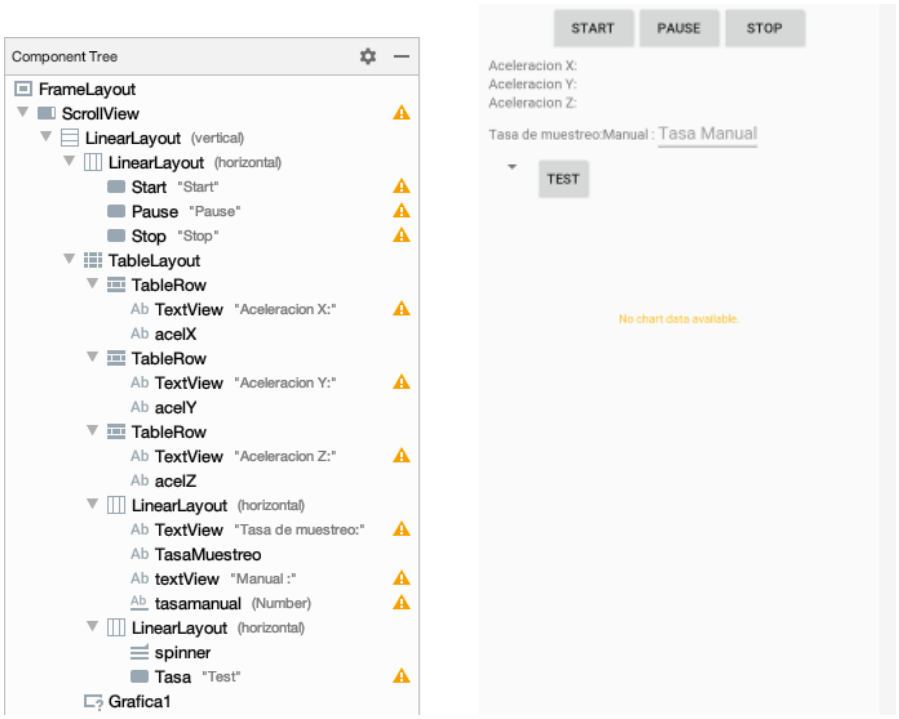
Fig. 5.4. Diagrama de flujo de la activity Accdata

En el diagrama de flujo de de esta *activity* explica su funcionamiento. En primer lugar se puede iniciar a tomar datos del acelerómetro con la tasa de muestreo predeterminada o elegir una propia. En el caso de ser una elección manual se debe tomar el dato escrito por el usuario. Una vez elegida una tasa: las medidas se sacan por pantalla, se hace un cálculo interno de la tasa de muestreo y se representan los datos en una gráfica activa.

En este punto se puede elegir otra medida o pausar o parar la toma de mediciones. Una vez paradas se pueden reiniciar en cualquier momento, ya sea eligiendo una tasa nueva o pulsando el botón "start" para seguir midiendo con la misma tasa que se estaba midiendo antes de la parada. En cualquier momento se puede salir de la pantalla y la toma de datos de pararía automáticamente.

5.2.1. Layout

En primer lugar hay que ver qué aspecto va a tener la *activity* por pantalla y que tenga todos los elementos que se necesitan para después en el archivo *.java* darle la función que necesiten. Su distribución final es la que se puede apreciar en la figura 5.5 b).



(a) Árbol del Layout (b) Layout

Fig. 5.5. Layout de la activity AccData

En el árbol de componentes de la figura 5.5 a) se puede apreciar de forma simplificada la estructura de la *activity*. Para su correcta visualización que los datos queden ordenados en pantalla se han incluido *layouts* dentro de otros *layouts* como una organización de elementos dentro de otra organización de elementos.

El código de este *layout* se puede encontrar en el anexo A.

5.2.2. Acceso al acelerómetro

En primer lugar hay que ver cómo se puede acceder al acelerómetro del dispositivo para luego ya poder coger los datos y mostrarlos por pantalla. El acelerómetro es uno de los varios sensores de los que dispone un dispositivo móvil. Para acceder a él se hará como se ha descrito en la sección 2.3.4.2 de los complementos teóricos como en el código 5.2

```
1          private SensorManager sensorManager;
2          Sensor acelerometro;
3          ...
4          sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
5          acelerometro = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
6          ...
```

CÓDIGO 5.2. Acceso al acelerómetro

Se crean dos instancias con dos variables para el servicio de Android y para el acelerómetro. Este "acelerómetro" será la representación del sensor en el código. Como el acelerómetro es uno de los sensores más comunes y es muy improbable que el dispositivo no cuente con uno, se omite la comprobación de si el dispositivo cuenta con uno. De todas formas si no contase con uno, la aplicación sería inútil ya que se basa en el funcionamiento del acelerómetro.

↓ Véase el código completo y comentado en la línea 84 del código A.2

Mediciones del acelerómetro

Se quiere sacar por pantalla los datos del acelerómetro y se hará en un cuadro de texto o *TextView* para cada eje. Se obtienen los datos del acelerómetro y se almacenan en una variable que se incluye dentro del cuadro de texto. Este proceso se hace simultáneamente para cada una de las tres coordenadas así se muestra el valor como aparece en la figura 5.6.

↓ Véase el código completo y comentado en la línea 416 del código A.2

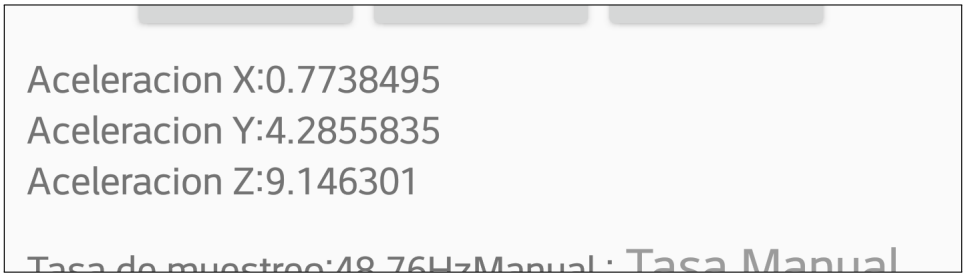


Fig. 5.6. TextView con las aceleraciones

5.2.3. Tasa de muestreo

El objetivo es saber cómo la define Android y cómo se puede calcular un valor en este entorno. Como ya se explica con más detalle en la sección 2.3.4.3, no existe un método en Android para calcularla por lo que hay que crear un método propio.

El método **onSensorChanged()** en uno de los más importantes de la *activity* ya se activa cada vez que se recibe una medición nueva. Parte de lo que se hará en este método es obtener las últimas mediciones del acelerómetro y una medida del tiempo actual. Se va a diferenciar si la tasa se ha definido manualmente y si es una de los distintos modos que tiene el dispositivo predeterminados, más información en la sección 2.3.4.3.

Para medir la tasa de frecuencia, se usa el tiempo del sistema debido a que no hay forma en Android de hacerlo. Para esto se usa el tiempo del sistema mediante el método **currentTimeMillis()** que te devuelve el tiempo del sistema en milisegundos desde 01/01/1970 a las 00:00 UTC.[2] Esta medida en milisegundos definirá la precisión de los datos ya que no se puede dar un dato del tiempo de una forma más exacta.

↓ Véase el código completo y comentado en la línea 365 del código A.2

Cálculo de la tasa de muestreo

La tasa de muestreo es el número de muestras en un intervalo de tiempo y como interesa medirlo siempre en las mismas unidades, se utilizarán los hercios o medidas por segundo. De este modo hay que obtener del sistema las dos variables para su cálculo.

$$Tasa\,muestreo = \frac{N^o\,Muestras}{Tiempo\,(s)}\ [Hz] \tag{5.1}$$

```
1           ...
2           if (tiempoactual >= tiempoanterior + 1000){
3               tasa = muestras / ((tiempoactual - tiempoanterior)/1000);
4               tiempoanterior = System.currentTimeMillis();
5               contador=0.0;
6               ...
7           }
```

CÓDIGO 5.3. Ejemplo del cálculo de la tasa de muestreo

Se produce un nuevo cálculo de la tasa cuando se cumple la condición de que hayan pasado al menos 1000 ms desde el cálculo anterior. El cálculo del tiempo se produce de la resta del tiempo actual y el tiempo en el que se hizo el cálculo de la

[2] A día de 23/04/2020 a las 19:30 UTC este número va por 1587670205339

tasa previo. Este tiempo del sistema está sujeto a los procesos de otras aplicaciones o el sistema por lo que puede variar y no es el método más exacto pero es el disponible.

Para obtener el número de muestras se utiliza un contador que suma uno cada vez que se obtiene un dato nuevo del acelerómetro y que se reinicia cuando se hace un nuevo cálculo de la tasa. Ya para hacer el calculo numérico de la tasa, se realiza la división de las variables que almacenan el número de muestras y el tiempo y se saca ese valor por pantalla en el recuadro de texto o *TextView* correspondiente. En la figura 5.7 en la parte superior izquierda se encuentra el *TextView* con la tasa como aparece en la aplicación.

$$Tasa\,muestreo = \frac{50}{1003\,ms/1000} = 49,85\ [Hz] \tag{5.2}$$

Como se puede ver en el ejemplo de la ecuación 5.2, la condición de que al menos pasen 1000 ms hace que en muchos casos el tiempo sea mayor a 1000 ms y se obtenga una tasa menor a la esperada. Una posible solución es calcular la tasa en un tiempo mayor para reducir el efecto de las variaciones del tiempo del sistema al máximo pero ya no se podría mostrar el cálculo casa segundo por pantalla.

↓ Véase el código completo y comentado en la línea 398 del código A.2

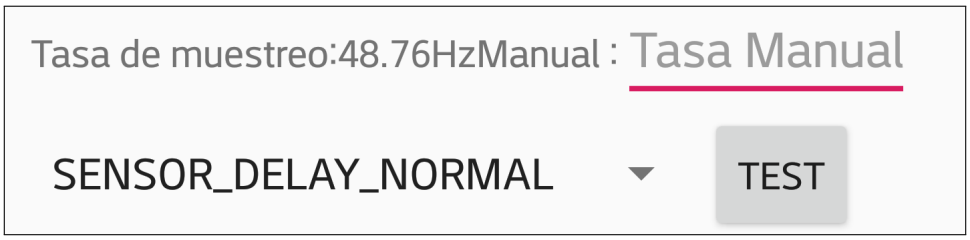


Fig. 5.7. Tasa de muestreo calculada y elección

Elección de la tasa de muestreo

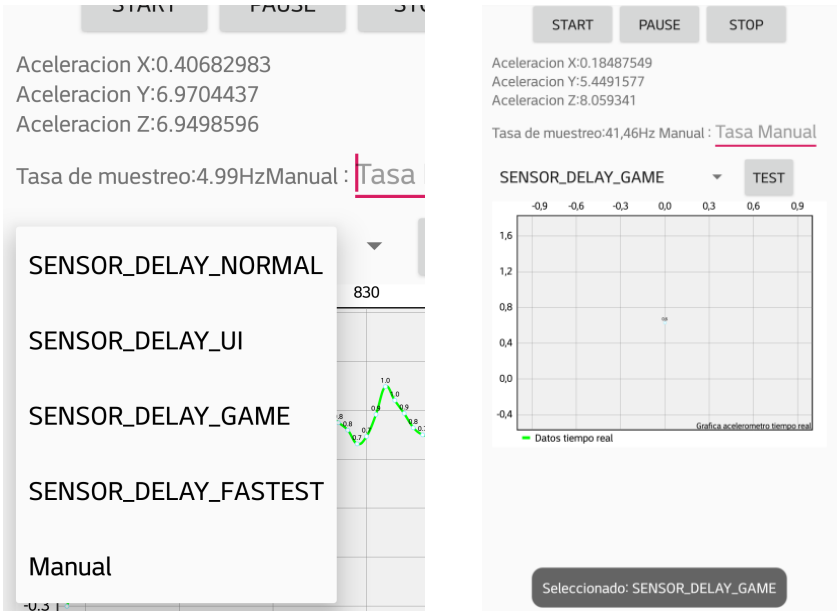
Visto cómo se calcula la tasa de muestreo, distintas tasas se seleccionan en un menú desplegable o *Spinner* entre los distintos modos predeterminados y en el caso de seleccionar el modo manual en el que habría que introducir el valor. Por tanto en el *spinner* que se creará tendrá cinco opciones, las cuatro predeterminadas por el sistema y una adicional para seleccionar manualmente la tasa.

Se crea una representación del *spinner* y se modificará para que no aparezca desplegado al inicio como aparece en la figura 5.8 (a y que se despliegue al seleccionarlo. Al seleccionar un elemento se quedará activado como aparece en la figura 5.8 (b).

Para definir los elementos que contendrá el *spinner*, se crea un **ArrayList** que es un vector de elementos con todas las tasas de muestreo elegibles. Como es un vector de elementos de pequeño tamaño (5 elementos) con saber la posición de cada uno

será suficiente para acceder a ellos. Por último, para la creación de un *spinner* en Android hace falta un *Adapter*. Este es el puente entre los elementos del *ListArray* y su representación visual por medio de un *View*.

↓ Véase completo y comentado en la línea 97 del código A.2



(a) Desplegado (b) Game seleccionado

Fig. 5.8. Spinner para seleccionar tasa

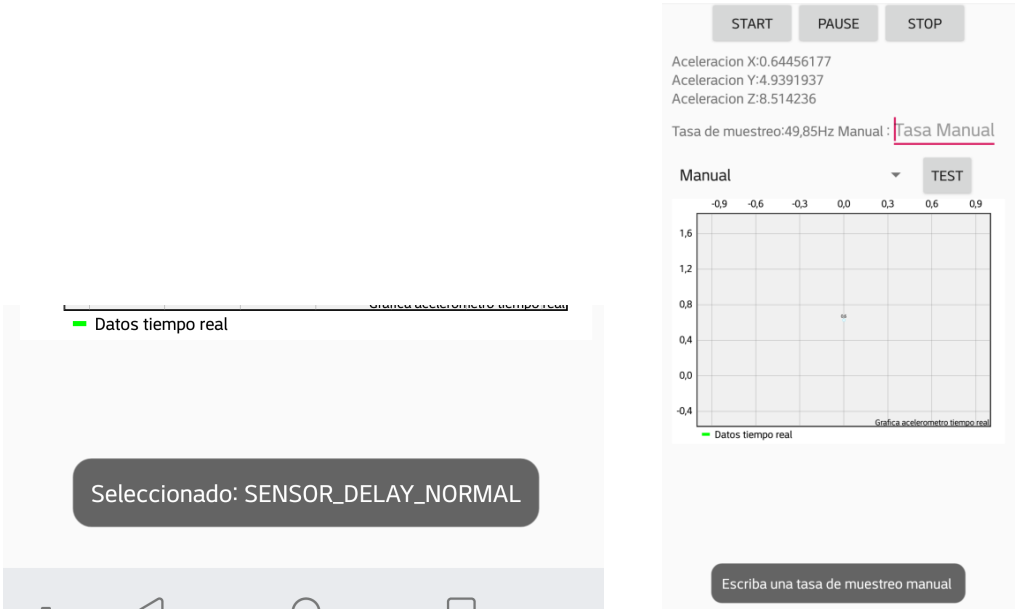
Método *onItemSelected()*. Funcionalidad

El *spinner* ya se ha creado pero le falta la funcionalidad, es decir, lo que la aplicación hace cuando se selecciona cada uno de sus elementos. El método *onItemSelected()* se activa cada vez que se selecciona cada uno de sus elementos y dependiendo de la tasa elegida se realiza una acción distinta.

Si se selecciona uno de las tasas predeterminadas por Android se vuelve a registrar el acelerómetro con la nueva tasa y si se elige el método manual primero se comprueba si se ha escrito un valor en el recuadro dedicado y se avisa al usuario en caso contrario como en la figura 5.9 (b). Para todos los casos se desregistra el acelerómetro con la tasa de muestreo seleccionada anterior y se registra con la tasa nueva.

Sea cual sea el caso elegido se ejecuta un *Toast* avisando al usuario de que tasa es la activa actualmente como aparece en la figura 5.9 (a).

↓ Véase el código completo y comentado en la línea 461 del código A.2



(a) Seleccionado (b) Manual

Fig. 5.9. Avisos por pantalla

5.2.4. Gráfica de las mediciones

Una vez que ya se ha accedido a los datos del acelerómetro se busca incluir una gráfica en la que se puedan representar estos datos respecto al tiempo. Android tiene algunas gráficas simples pero la gráfica que se quiere implementar debe de ser dinámica así que después de buscar entre distintas librería externas de gráficas creadas por otros desarrolladores se selecciona una teniendo en cuenta los requerimientos: que su acceso tiene que ser gratuito (con licencia de uso no comercial) y debe de poder expresar la medida del acelerómetro en el tiempo real. La librería finalmente seleccionada es *GraphView*[14] debido a su simplicidad para representar los datos en la gráfica.

```
1  implementation 'com.jjoe64:graphview:4.2.2'
```

CÓDIGO 5.4. Implementación Librería GraphView

Para la implementación de la gráfica de la librería externa se siguen los siguientes pasos:

- Implementar la gráfica como en el código 5.4 en el archivo *build.gradle*. Esta se descargará de forma automática.
- Incluir el *View* (código 5.5) en el archivo *xml*

- Definirla en el archivo *java*
  - Inicializar la gráfica y modificar su apariencia
  - Añadir un hilo o *thread* al ser dinámica
  - Creación de un conjunto de datos
  - Añadir datos al conjunto de datos

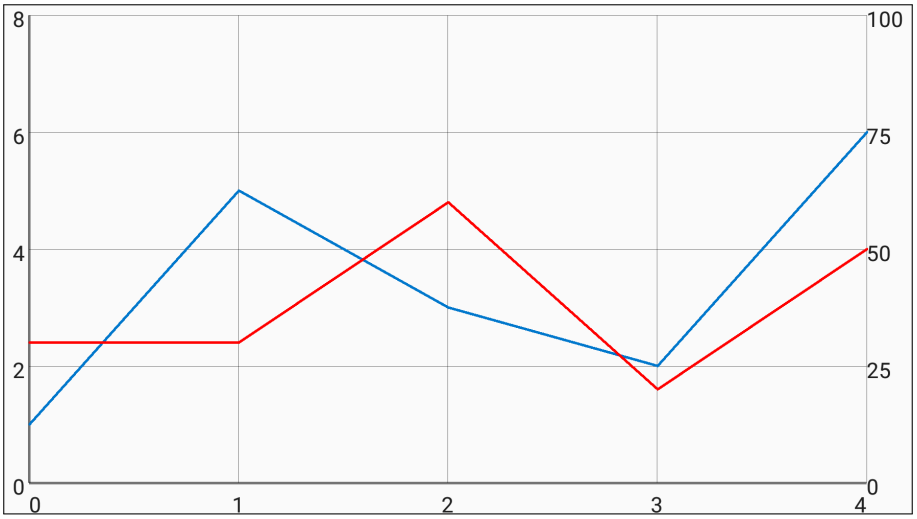


Fig. 5.10. Ejemplo de gráfica con *GraphView*[14]

```
<com.github.mikephil.charting.charts.LineChart
    android:id="@+id/Grafica1"
    android:layout_width="match_parent"
    android:layout_height="250sp">
</com.github.mikephil.charting.charts.LineChart>
```

CÓDIGO 5.5. Gráfica de AccData en el archivo xml

Una vez que se ha añadido la librería externa, ya se cuenta con todos los métodos y código de esta. En el código 5.5 se incluye un elemento de tipo *View* que pertenece a la librería externa y no Android por lo que es diferente de las gráficas nativas de Android.

Apariencia de la gráfica

Dentro del método *onCreate()* (al iniciar la *activity*) se crea, inicia y personaliza la gráfica. Se debe unir la instancia de la gráfica con su parte en el archivo *xml* por su ID. Para ello se crea una instancia de la gráfica que se llama "mChart", que se puede personalizar como por ejemplo permitiendo que se pueda escalar o el cambiando el color de fondo.

Se crea una línea al ser una gráfica de este tipo y se añaden después los distintos elementos de la gráfica como son la leyenda, el título y los dos ejes. Se personalizan estos elementos cambiándoles la forma o el color. A destacar el método *setAvoidFirstLastClipping()* que hará que el primer y último elementos de la gráfica no se salgan de esta, importante al ser una gráfica dinámica.

↓ Véase el código completo en la línea 130 del código A.2

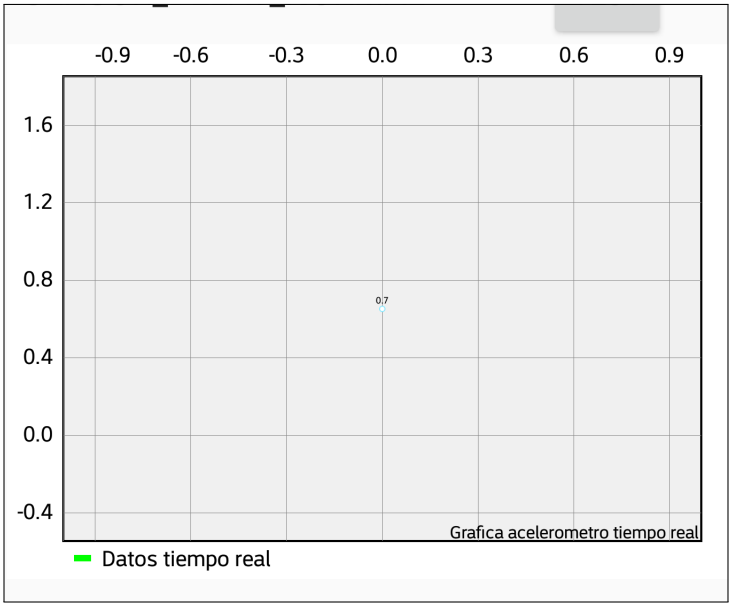


Fig. 5.11. Gráfica de mediciones sin datos

Hilo de la gráfica o *Thread* (Gráfica dinámica)

Para que los datos de la gráfica sean cambiantes se debe añadir un ***Thread*** o hilo de la gráfica. A diferencia de una gráfica de líneas en la que se conocen los datos desde el inicio y estos son fijos, esta gráfica va a mostrar los datos recibidos por el acelerómetro de forma dinámica por lo que necesita este *thread* para hacerlo. Por lo tanto cuando se llame al método *startPlot()* el *thread* comenzará a funcionar y se añadirá un elemento a la gráfica.

Como los acelerómetros pueden tomar muchas medidas por segundo si se sacasen todas por la gráfica sería muy difícil de entender por lo que este *thread* se le añade una pausa interna de 100 milisegundos para que en caso de tener tasas de muestreo altas la gráfica siga siendo legible.

↓ Véase el código completo en la línea 253 del código A.2

Creación de un conjunto de datos

Ya definida la gráfica y sus elementos hay que rellenarla. En gráficas de líneas hay que definir un *LineDataSet* o **conjunto de datos**, al que se llamará "set". En pantalla este conjunto se llamará "Datos tiempo real" como se ha definido también. Este conjunto se modificará con distintos métodos para personalizarlo para seleccionar en qué eje aparecerá, el color o tamaño de los puntos y en este caso la forma de la línea que será de función cúbica para hacer que se parezca más a una oscilación.

Este conjunto de datos es el que aparecerá en la leyenda de la gráfica de la figura 5.11 en la parte inferior fuera del recuadro que ocupa la gráfica. Esta es la situación de la leyenda donde sale el nombre de los conjuntos de datos que tenga la pantalla con una pequeña representación de como será la línea en cuanto a color y espesor para poder identificarla en la gráfica.

↓ Véase el códigocompleto en la línea 333 del código A.2

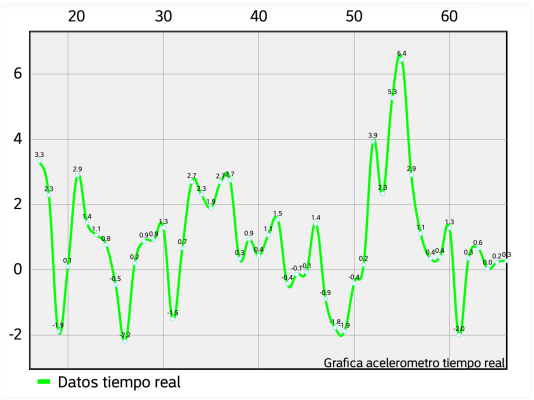
Añadir datos nuevos al conjunto de datos

Se añaden datos con el método *addEntry()* y se selecciona en el punto del *set* o conjunto de datos en el que se añade el dato y después el valor del dato. Se puede considerar el conjunto "set" como un vector con los datos de la gráfica y el punto en el que se van añadiendo va es cada vez mayor.

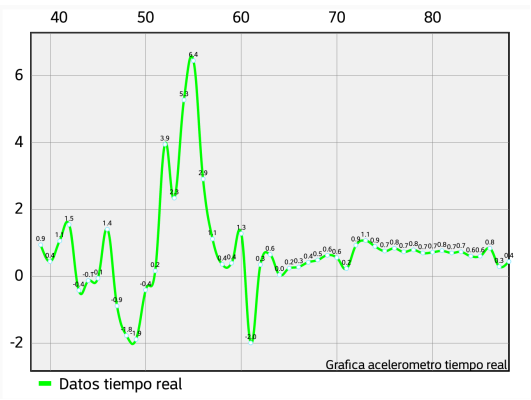
Como solo puede haber un número limitado de datos en la gráfica surge el problema de que los datos que no aparecen en la gráfica no son necesarios y solo ocupan memoria. No es mucha memoria la ocupada pero si la gráfica funciona durante un tiempo prolongado puede afectar al rendimiento del dispositivo.

Se incluye una variable nueva que actuará de contador para eliminar este exceso de datos. Así el vector "set" siempre tendrá 50 elementos, se le suma uno en la posición mayor de este vector y se elimina el primer dato que será el que acaba de desaparecer de la gráfica. De esta forma se optimizan los recursos del dispositivo y la aplicación funciona de una forma más fluida.

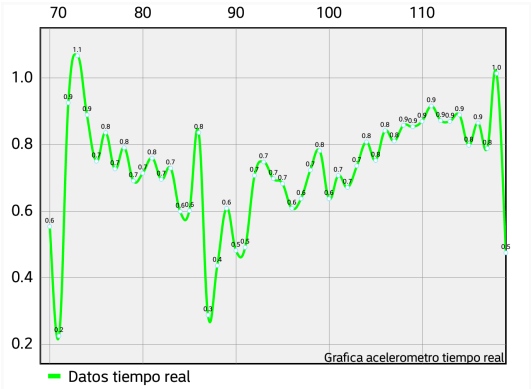
↓ Véase completo y comentado en la línea 293 del código A.2



(a) Momento 1



(b) Momento 2



(c) Momento 3

Fig. 5.12. Gráfica con las mediciones de datos en funcionamiento

En la figura 5.12 se muestra la evolución de la gráfica con el tiempo en tres momentos consecutivos. Entre el momento 1 y el 2 se ve algunas formas significativas que avanzan hacia la izquierda como se ha descrito anteriormente. También se puede ver el movimiento en los números del eje "Y" en la parte superior de la gráfica ya que cada medición tiene un número y si esta se desplaza los números también.

Del momento 2 al momento 3 también se ha producido un movimiento a la izquierda de los datos pero con la diferencia de que los nuevos datos tienen una amplitud baja y no se distinguen bien en la gráfica. Por esto la gráfica se escala automáticamente para acomodar los valores y que sean más fáciles de distinguir.

5.2.5. Propiedades del acelerómetro

Por último en la *activity* "AccData" se incluye otra *activity* de menor importancia en la que se pueden obtener las propiedades del acelerómetro del dispositivo.

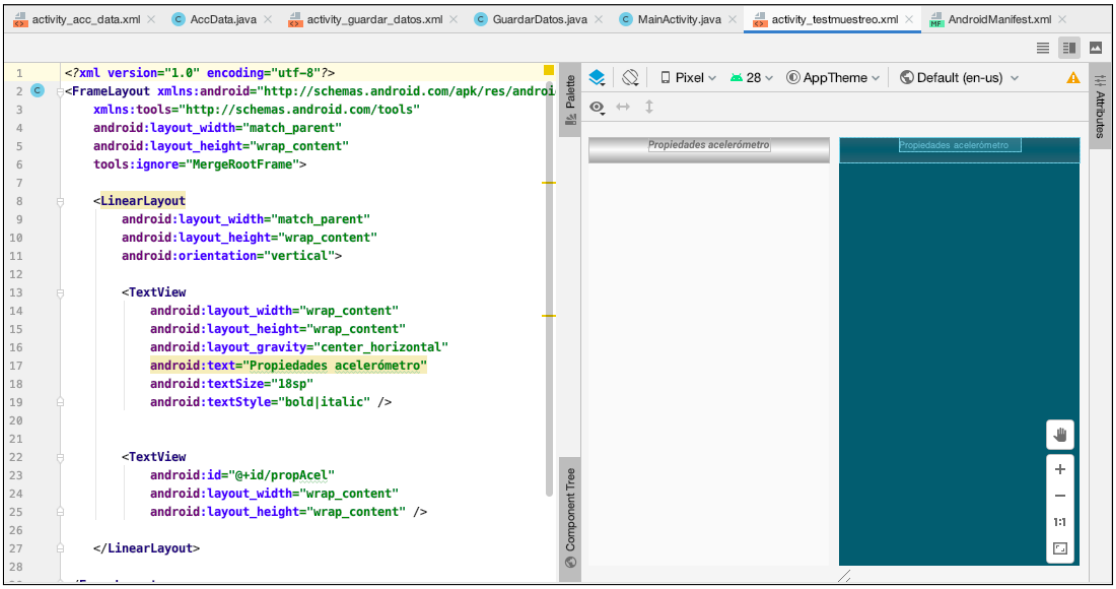


Fig. 5.13. Layout de la *activity* "propiedades"

En figura 5.13 se puede ver la disposición del ***Layout*** que será de tipo lineal vertical con dos elementos: el título en un *TextView* y otro *TextView* en el que se escribirán todas las propiedades relevantes del acelerómetro. El primer *TextView* utilizado para el título y personaliza para centrarlo, aumentar su tamaño y ponerlo en cursiva. El segundo *Textview* es el que se van a incluir todos los datos del acelerómetro y se definirá su contenido en el archivo *java*.

Parte *java* de "test".

```
public class Testmuestreo extends AppCompatActivity {

    private SensorManager sensorManager;
    Sensor acelerometro;
    private TextView nombreAcel;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_testmuestreo);

        sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
        acelerometro = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);

        nombreAcel = (TextView) findViewById(R.id.propAcel);

```

```
17          nombreAcel.setText("Nombre:" + this.acelerometro.getName() + "\nFabricante:"
                + this.acelerometro.getVendor() +
18                  "\nVersion:" + this.acelerometro.getVersion() + "\nResolucion:" +
                        this.acelerometro.getResolution() +
19                  "\nRango maximo:" + this.acelerometro.getMaximumRange() + "\nPotencia
                        :" + this.acelerometro.getPower() + "mA");
20      }
21  }
```

CÓDIGO 5.6. Archivo java de test

El código 5.6 que corresponde a la parte java y el acceso al acelerómetro se hace igual que en la *activity* anterior. Primero se crea una instancia del acelerómetro, se accede al servicio del Android referente a los sensores y finalmente se accede al acelerómetro. Falta rellenar el segundo *TextView* y para ello se utilizarán los métodos del siguiente párrafo.

getName() para obtener el nombre que el fabricante le ha dado al acelerómetro, *getVendor()* para obtener el fabricante, *getversion()* para obtener la versión del sensor, *getResolution()* para obtener la resolución máxima del sensor, *getMaximunRange()* para obtener el rango máximo de medidas del sensor y por último *getPower()* para obtener la potencia consumida por el sensor cuando esté en funcionamiento.

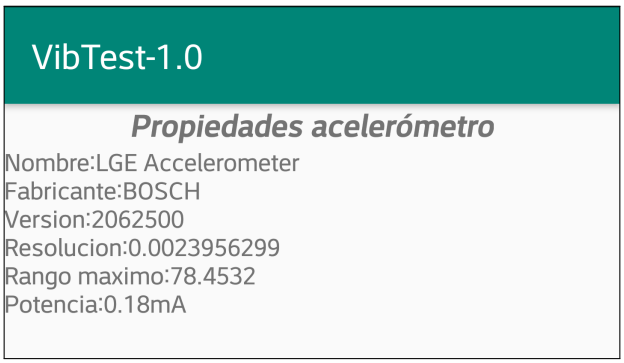


Fig. 5.14. Propiedades del acelerómetro (LG G6)

En la figura 5.14 se puede el resultado de esta *activity* y como saldría por pantalla. Los resultados se corresponden al acelerómetro dentro de un dispositivo del fabricante LG y modelo G6 del año 2017. Se puede apreciar los resultados obteniendo un acelerómetro del fabricante *Bosch* versión 2062500. Con una resolución de 0,00239 m/s^2 y un rango máximo de medidas de 78,45 m/s^2 o ≈8g.

5.3. Guardado de los datos - GuardarDatos

En esta nueva *activity* se desarrolla el guardado de los datos del acelerómetro en el dispositivo con distintos requerimientos. Inicialmente se barajó la idea de almacenar los datos en una base de datos donde cada elemento fuese una medición que incluía dentro todos los datos. Y aunque para otros usos es útil como una base de datos de personas con atributos como nombre y edad, para el uso que se le quiere dar no es el mejor método.

Esto de debe a que se debe guardar los datos obtenidos por el acelerómetro en los tres ejes además de su marca temporal del reloj del dispositivo de cada muestra. Así se guardarían cientos de datos por segundo y cada dato tiene la información de tres ejes por lo que la opción de utilizar una base de datos es finalmente descartada debido a que no es el sistema más adecuado por el volumen de datos y añadiría complejidad a la aplicación.

Con las bases de datos descartadas, los esfuerzos se centran en utilizar un sistema fácil de guardar los datos, acceder a ellos o exportarlos después. De este modo se llega al archivo ***CSV***, siendo este un archivo de texto que es fácil de generar pero con la peculiaridad de que se puede estructurar datos en formato tabla utilizando comas y espacios para separar columnas y filas. De este modo se puede guardar la cantidad que se quiera de datos añadiendo una nueva fila a la vez que es un sistema fácilmente exportable.

5.3.1. Layout

La disposición de elementos en esta *activity* se eligió para facilitar la navegación del usuario por la propia *activity* y que se siga un orden en las acciones como si fuese una lista de tareas pendientes. De modo que la estructura principal será de botones en una lista vertical y cada botón realzará una tarea fundamental para el guardado de los datos. El primer botón servirá para elegir el nombre de la medición, el segundo para iniciar la medición, el tercero para parar la medición y por el cuarto para guardarlo en el dispositivo.

También se cuenta con otros elementos de tipo *View* como son un *EditText* para introducir un número de mediciones o un *CheckBox* para seleccionar un número limitado o ilimitado de datos en la medición. Todos estos elementos y su estructura dentro de distintas organizaciones o *layouts* se puede ver en el árbol de componentes que aparece abajo a la izquierda en la figura 5.15. En esta figura se puede ver además la disposición de los elementos en la pantalla.

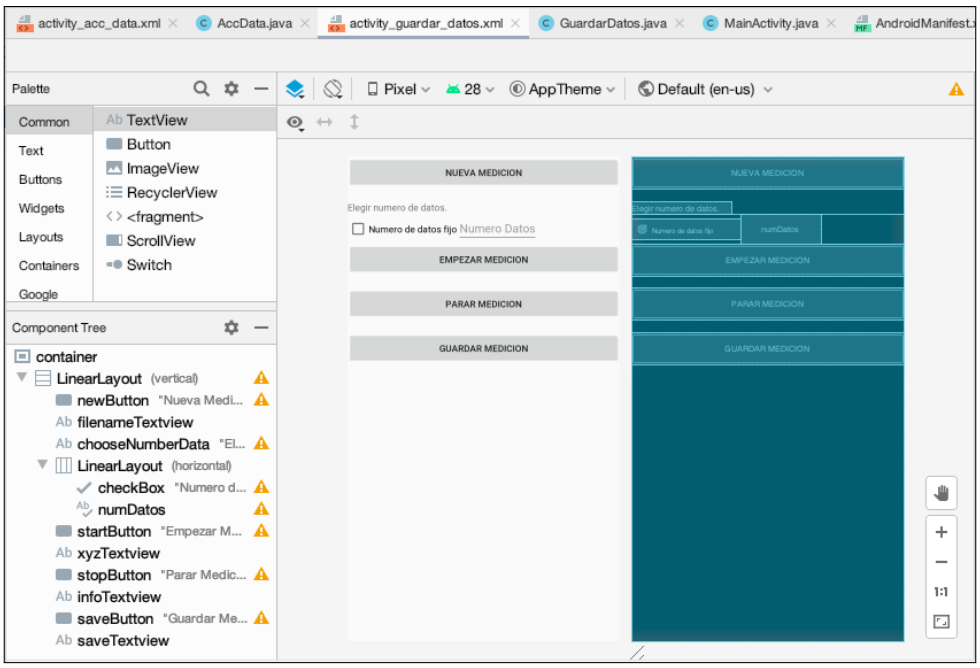


Fig. 5.15. Layout de la *activity* "Guardar Datos"

En el diagrama de flujo de la figura 5.16 se puede ver la secuencia de que puede hacer el usuario desde que se elige el nombre al archivo hasta que se guarda en la memoria del dispositivo. Habrá botones que aparecerán deshabilitados para el uso de usuario para que este siga el camino correcto para llegar a un archivo guardado.

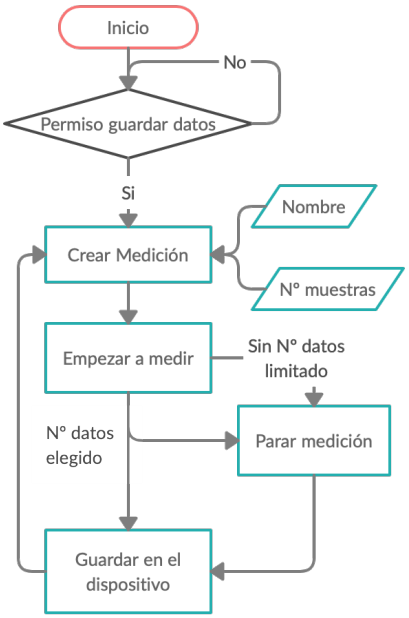


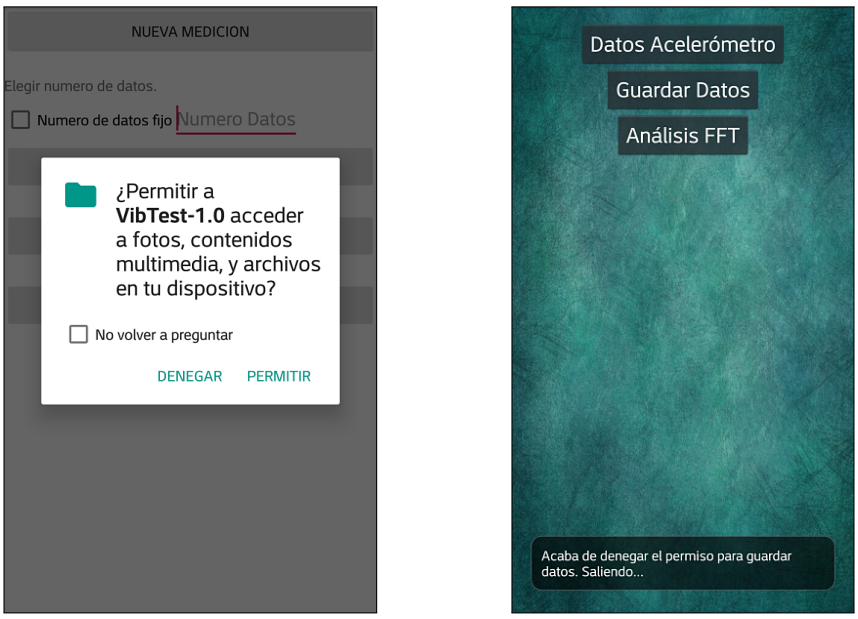
Fig. 5.16. Diagrama de flujo de la *activity* "Guardar Datos"

El código de este *layout* se puede encontrar en el anexo B.

5.3.2. Permisos de Guardado de Datos

Previamente al guardado de datos, se requieren unos pasos previos como son los permisos para poder guardar datos. Al iniciar la *activity* el primer paso en el guardado de datos será un mensaje que aparece en pantalla pidiendo permiso para poder guardar datos en el dispositivo como en la figura 5.17 (a). El usuario deberá aceptarlos si se quiere continuar utilizando esta utilidad de la aplicación. Si se deniegan dichos permisos, no se podrán guardar datos y se saldrá automáticamente de la *activity* al menú principal. Aparecerá además un mensaje informando al usuario de que se acaban de denegar los permisos necesarios para continuar, como en la figura 5.17 (b).

↓ Véase el código completo y comentado en la línea 44 del código B.2



(a) Pregunta permisos (b) Salir de la activity

Fig. 5.17. Aviso para permitir guardar datos en el dispositivo

5.3.3. Crear una nueva medición

Una vez aceptados los permisos para escribir datos en la memoria del dispositivo el usuario se encontrará con la pantalla de la figura 5.18, en la que la única acción permitida será el botón con el texto "nueva medición".

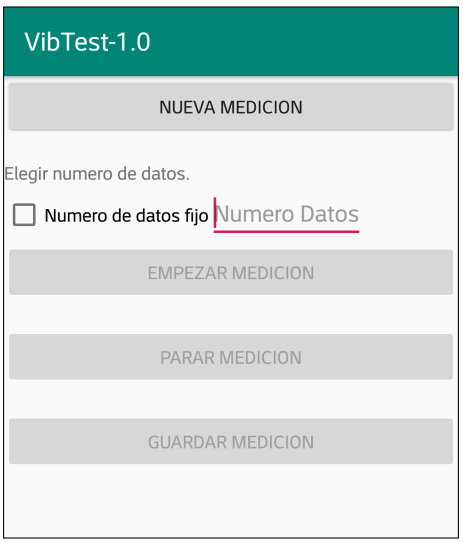


Fig. 5.18. Estado inicial de la *activity* GuardarDatos

Otro dato importante es que en el resto de botones de la figura 5.18 el texto está en gris, lo que significa que están desactivados. Esto se debe a que se han de seguir los pasos para la creación de la medición en un determinado orden, de forma que el usuario de la aplicación no pueda dar a un botón que no corresponde. Si esto sucediese la aplicación daría un error y se cerraría ya que esta intentaría acceder a una medición que todavía no se ha creado.

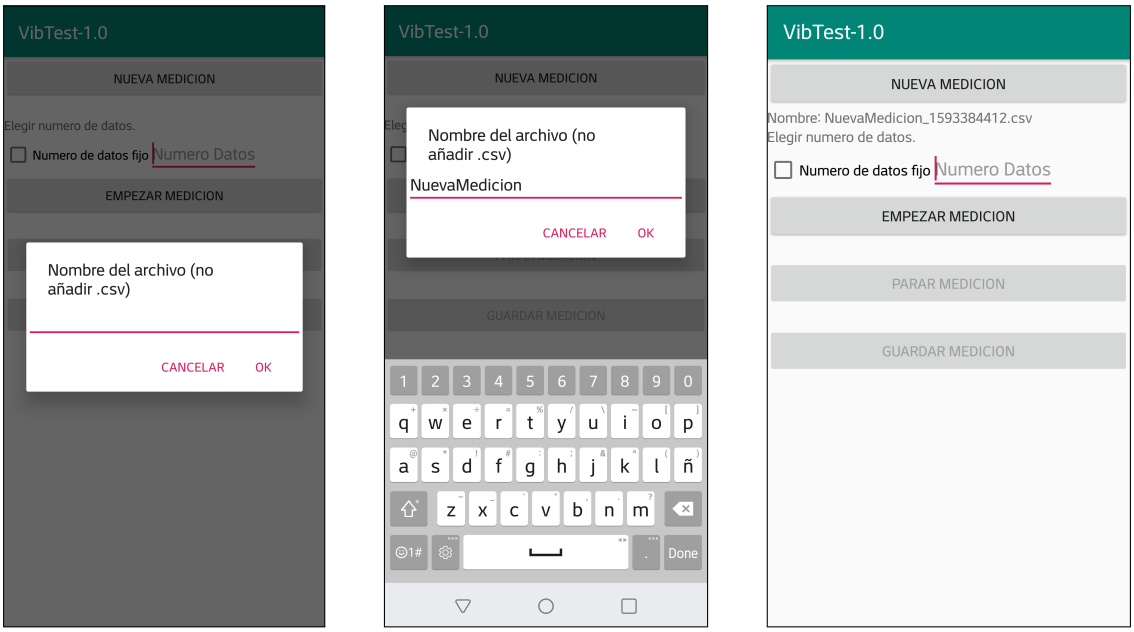
Al iniciar la *activity* tanto como al seleccionar el botón de guardar datos, el resto se deshabilitan. Adicionalmente cuando se selecciona el botón, se ejecutará el método *displayFileForm()* y aparecerá por pantalla un formulario para seleccionar el nombre de la medición.

↓ Véase el código completo y comentado en la línea 152 del código B.2

Método *displayFileForm()*. Formulario para el nombre de la medición

Cuando se ejecuta este método, aparece una alerta por pantalla o *AlertDialog* para introducir el nombre del archivo. Se utilizará esta alerta de modo de formulario y se utiliza el título como guía y pedir el nombre que se le quiera dar a la medición. Como se puede ver en la figura 5.19 hay dos formas de proseguir con el formulario, rellenarlo con un nombre y dar a "OK" o dar a "CANCELAR" y salir de formulario.

En el caso de seleccionar "OK" se guardará el nombre elegido que se utilizará para crear el nombre completo. Además se guardará la hora de creación de la medición en ms de la misma forma que se creaba para el cálculo de la tasa de muestreo en la sección 5.2.3. Por último al nombre se le añade la terminación ".csv" para que el archivo sea de este tipo.



(a) Introducir nombre (b) Nombre seleccionado (c) Nombre completo

Fig. 5.19. Recuadro para la selección del nombre de las mediciones

El resto de cuadros de texto que tiene la *activity* se ponen en blanco menos el que aparece justo debajo del botón para crear la medición. Este cuadro de texto mostrará el nombre completo del archivo que es nombre con el que se guardará en la memoria después de realizar la medición y aparecerá como en la figura 5.19 (c).

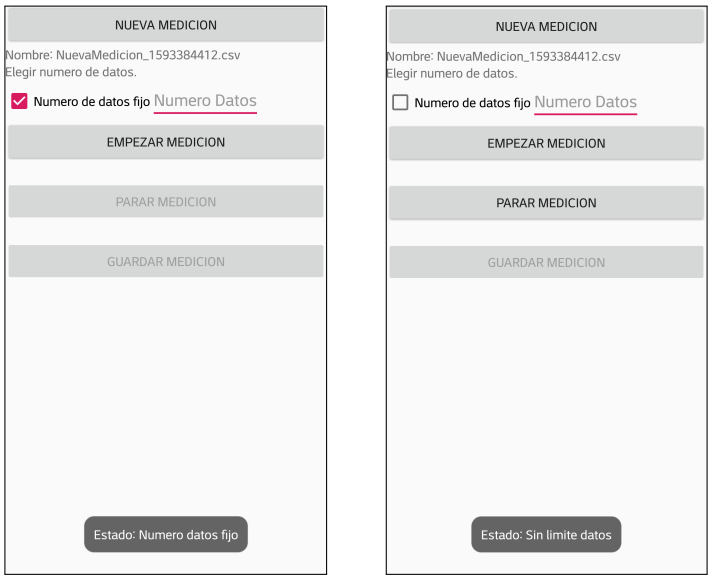
↓ Véase el código completo y comentado en la línea 346 del código B.2

5.3.4. Elección de número de medidas por medición

Antes de iniciar la medición se requiere que el usuario defina el número de medidas que se tomarán. Es importante saber que para luego su análisis, si se toma un mayor número de medidas la precisión de la gráfica de la Transformada Rápida de Fourier (FFT) será mayor por lo que se recomienda elegir un número que no sea bajo con la contrapartida de tiempos de procesamiento mayores.

Hay dos opciones a la hora de elegir el número de mediciones en cuanto a si este número es fijo y cuando este se alcance se para la medición automáticamente, o de si el usuario para la medición. Mediante un *CheckBox* se seleccionará entre ambas opciones y cuando se haga la elección aparecerá un *toast* o aviso por pantalla de la elección del usuario como se puede ver en la figura 5.20.

↓ Véase el código completo y comentado en la línea 247 del código B.2

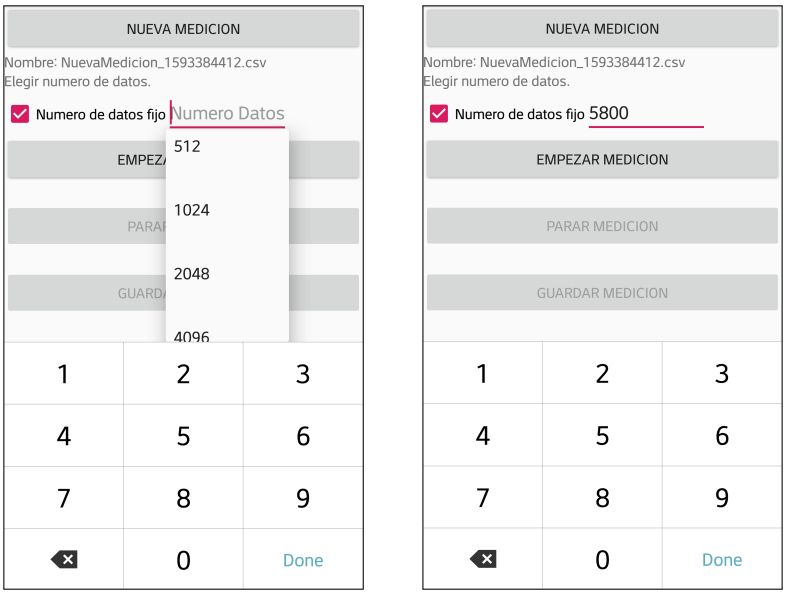


(a) Número datos fijo (b) Número datos ilimitado

Fig. 5.20. Elección del número de mediciones

En el caso de seleccionarse un número de datos fijos, para seleccionar el número se dispone de un cuadro de texto editable o *TextEdit* que se llama "Numero Datos". Al seleccionarlo, se despliega un teclado numérico para introducir el número que se desee. También se despliega una lista con varios ejemplos de números de mediciones. Este desplegable es similar al *Spinner* que se utiliza en la *activity* previa pero en este caso son solo opciones para rellenar.

Todos estas opciones de datos son potencias de dos ya que para realizar el análisis y la FFT se necesita que el número cumpla esta condición. En el caso de que no se cumpla se rellena con ceros hasta la próxima potencia de dos.



(a) Potencias de dos (b) Número introducido

Fig. 5.21. Número exacto de mediciones

5.3.5. Iniciar o parar la medición

Ya elegido el número de muestras a tomar, se inicia la medición pulsando el botón "**Empezar Medición**" y en el cuadro de texto justo debajo del botón aparecerá el mensaje "Guardando datos..." mientras continúe la medición como se puede ver en la figura 5.22. Lo que la aplicación empieza a hacer en este momento es guardar los datos como datos temporales para utilizar los mínimos recursos del dispositivo. En todo caso en la gran mayoría de los casos el usuario tendrá otras aplicaciones funcionando en segundo plano que consumirán recursos pero se minimizará el impacto en la aplicación.

↓ Véase el código completo y comentado en la línea 172 del código B.2

Fig. 5.22. Mensaje de "Guardando Datos..." al iniciar

Cuando se inicia una medición se hará uso del método *onSensorChanged()*, que se ejecuta cada vez que se recibe un dato nuevo. Dentro de este método es donde se guardan los datos que se reciben desde el acelerómetro en variables y estas variables se juntan en líneas con la forma de los archivos *csv* separando los datos por comas. Estas líneas correspondientes a cada nueva medición se guardan en la memoria interna del dispositivo y se repetirán los procesos descritos para nueva medición.

Ya que se están guardando los datos en la memoria más rápida pero volátil del dispositivo se esperará que para un número grande de mediciones sin que se guarden en la memoria externa permanente, el dispositivo baje su rendimiento. Estos datos ocupan poca memoria por lo que tardará en ocuparse completamente. Por este motivo no se recomienda usar la aplicación para medir vibraciones en un espacio grande de tiempo o para mantenimiento en el que se monitoricen datos en una gran franja de tiempo.

↓ Véase el código completo y comentado en la línea 275 del código B.2

Si por el contrario se ha elegido el camino de realizar una medición sin un número limitado de datos se debe presionar el botón de "**Parar Medición**" para que se dejen de tomar datos nuevos. Cuando la medición acabe ya sea porque se ha llegado al número elegido tanto como si se ha pulsado el botón "Parar Medición" debajo del botón de parar aparecerá un mensaje con el número de muestras medidas como líneas de datos guardadas. En la figura 5.24 (b) se puede ver como se puede para una medición antes de que llegue al número de datos seleccionado.

Aparecerá un mensaje por pantalla avisando al usuario de que se ha parado de tomar datos y de que el siguiente paso será guardar ya definitivamente la mediciones como en la figura 5.23. En este punto del proceso aparecerá el botón para guardar los datos habilitado para su uso.

↓ Véase el código completo y comentado en la línea 198 del código B.2

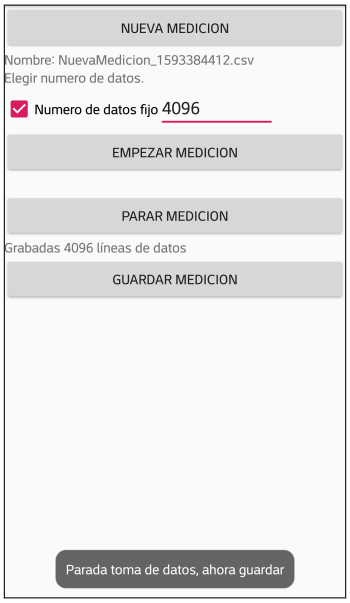


Fig. 5.23. Mensaje al parar la medición

5.3.6. Guardar los datos en el archivo

Para evitar que el dispositivo realice más tareas de las necesarias y así empeorar su rendimiento, no es hasta que ya se ha acabado de tomar datos que estos se guardan en el dispositivo de forma permanente. Cuando se selecciona el botón "Guardar Datos", se ejecuta el método *write()* y dentro se hará el guardado de los datos. Se deshabilita el botón que se acaba de pulsar ya que volver a guardar los mismos datos en una tarea improductiva.

↓ Véase el código completo y comentado en la línea 225 del código B.2

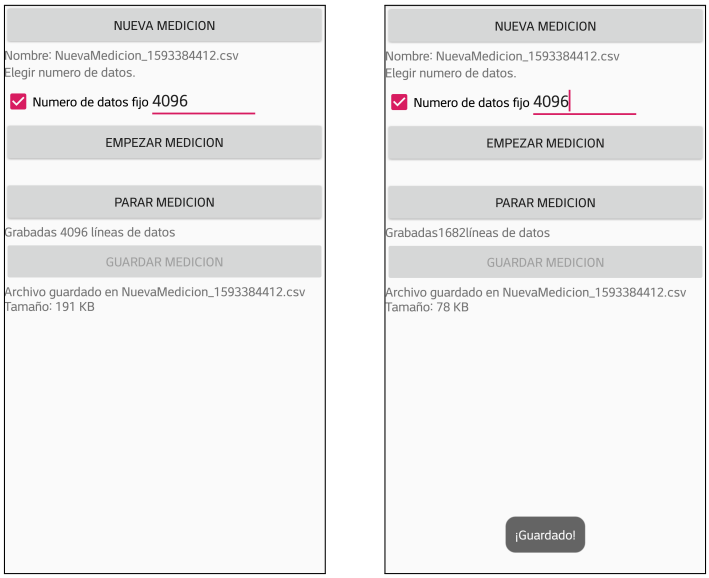
Dentro del método *write()* se guardará el archivo con el nombre elegido inicialmente y todos los datos en la memoria interna del dispositivo se pasarán al archivo final.

Para ello la aplicación creará un directorio en el que se guardarán todas las mediciones. Este directorio tiene el nombre de "AcelerometroDatos" y se situará en la memoria externa del dispositivo. Al añadirse la fecha en la que se hicieron no puede haber dos archivos con el mismo nombre así que se elimina el problema de tener algún archivo con nombre duplicado y que se pueda perder alguna medición.

Como se puede ver en la figura 5.24 (b finalmente aparece un aviso por pantalla que avisa de que ya se ha guardado el archivo y por tanto el proceso de toma de archivos a finalizado.

El siguiente es entrar en la *activity* encargada de analizar los datos.

↓ Véase el código completo y comentado en la línea 392 del código B.2



(a) Líneas de datos (b) Mensaje

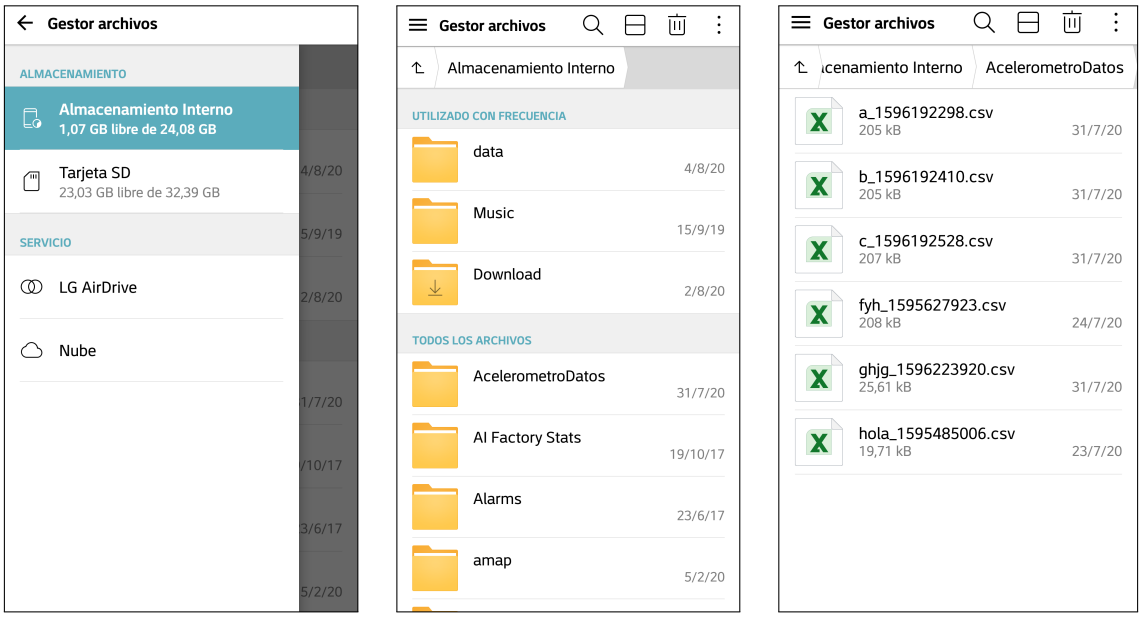
Fig. 5.24. Mensajes al guardar las mediciones

5.3.7. Extra: Exportar datos

Si se quiere exportar alguna medición, ya sea para guardarla de forma externa o analizarla con otros medios se ha programado la aplicación para que sea simple.

El primer paso es encontrar el archivo en la memoria del dispositivo. Si se accede desde el propio dispositivo se deberá acceder al archivo a través de alguna aplicación de gestión de archivos, todos los dispositivos suelen traer una instalada de fábrica. Desde la aplicación de archivos del dispositivo desde el que se tomarán las mediciones se siguen los pasos de la figura

Primero en la figura 5.25 (a se selecciona el almacenamiento interno y después se busca la carpeta “AcelerometroDatos” que aparecerá en las primeras posiciones por orden alfabético. En su interior como se ve en la figura 5.25 (c, estarán todas las mediciones que se hayan guardado con la aplicación.

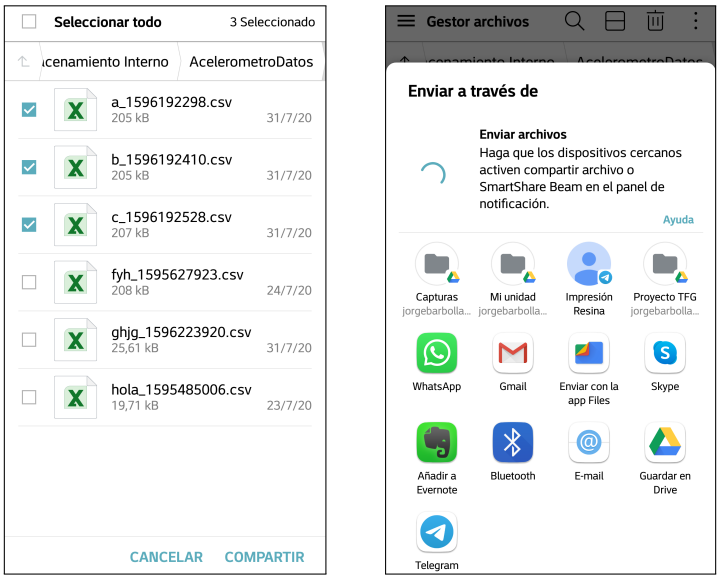


(a) Memoria interna (b) Carpeta AcelerometroDatos (c) Archivos mediciones

Fig. 5.25. Situación de los archivo para exportarlos en el dispositivo

Para exportar los archivos desde el propio dispositivo, una vez que se ha llegado a ellos desde la aplicación de archivos, estos se pueden seleccionar como se puede ver en la figura 5.26 (a). Una vez seleccionados los archivos se puede pulsar en "Compartir" y aparece la pestaña de la figura 5.26 (b).

En la figura 5.26 (b) se puede ver como enviar esos archivos de diferentes formas. Se pueden enviar por alguna aplicación de mensajería móvil, por correo electrónico o por algún almacenamiento en la nube como *Google Drive*. También se pueden sacar los archivos del dispositivo mediante cable y conectarlo directamente a un ordenador y copiar los archivos como si fuese una memoria de almacenamiento externo.



(a) Compartir (b) Alternativas

Fig. 5.26. Formas de exportar los archivos desde el dispositivo

El formato en el que se guardan los archivos hace que estos sean fáciles de leer en otros programas. Esta es la versatilidad de los archivos *.scv*, que a la vez de ocupar poca memoria se organizan de forma fácil las mediciones de los tres ejes y el tiempo en el que se han guardado.

La primera columna es la marca temporal de cada medición en milisegundos mientras que las otras tres columnas son las medidas del acelerómetro en los ejes "X", "Y" y "Z" respectivamente. Si se lee el archivo con un programa de texto se verán los datos en el orden anterior separados por comas, un ejemplo de como aparecerán es en el código 5.7.

```
1338988768815407, 0.55119324, 0.7996216, 9.548523
1338988771348366, 0.5368347, 0.8427124, 9.605988
1338988773850808, 0.21360779, 0.81877136, 9.807098
1338988776353249, -0.12875366, 0.7996216, 9.675415
1338988778886208, -0.07847595, 0.8618622, 9.421631
1338988781388650, 0.25671387, 0.7924347, 9.562897
```

CÓDIGO 5.7. Ejemplo de datos en archivo csv

Con un programa como *Excel* u otros de hojas de cálculo se puede importar el archivo *csv* y separar los datos en columnas. Para abrir este archivo en un programa de hojas de cálculo se debe importar de forma correcta ya que si solo se abre aparecerá cada fila separada por comas en la primera celda de la primera columna.

5.4. Análisis de los datos - FFT

Como última *activity*, está la encargada del análisis de los datos medidos anteriormente o importados en formato *CSV*. Aquí se importarán los datos del archivo elegido y se hará el análisis mediante la Transformada Rápida de Fourier o FFT para ver las frecuencias dominantes de la medición.

5.4.1. Layout

De modo similar al que se utilizó en la *activity* para guardar datos, en esta se utilizará una estructura en la que el usuario seguirá procesos de arriba a abajo en la pantalla. Y como todos los procesos no tienen espacio en la pantalla se incluye un *ScrollView* que envuelve al resto de elementos del *layout*, permitiendo al usuario bajar y ver los elementos no visibles.

Dentro del *ScrollView* se inserta una estructura de tipo tabla vertical donde irá el resto de botones con los que el usuario interactúa. La estructura de esta *activity* se puede ver en el árbol de componentes que aparece en la parte inferior izquierda de la figura 5.28. En esta misma figura se puede ver una representación como es la apariencia física de esta pantalla.

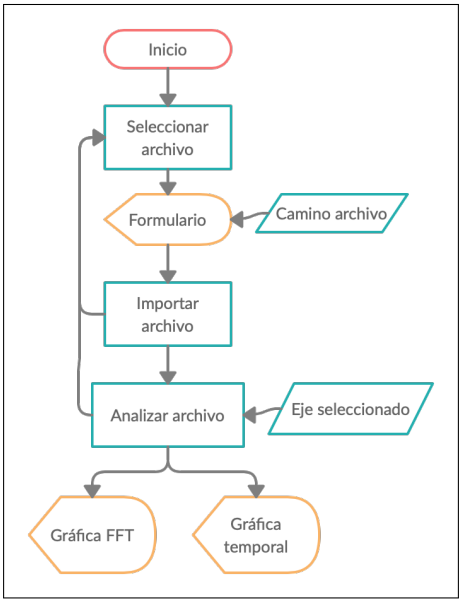


Fig. 5.27. Diagrama de flujo de la activity FFT

Al igual que la *activity* para guardar archivos, los distintos botones para realizar la tareas aparecerán habilitados o deshabilitados apareciendo el color gris. Esto se hace para que se siga la secuencia de pasos en el orden correcto. Este orden se puede ver en el diagrama de flujo de la figura 5.28.

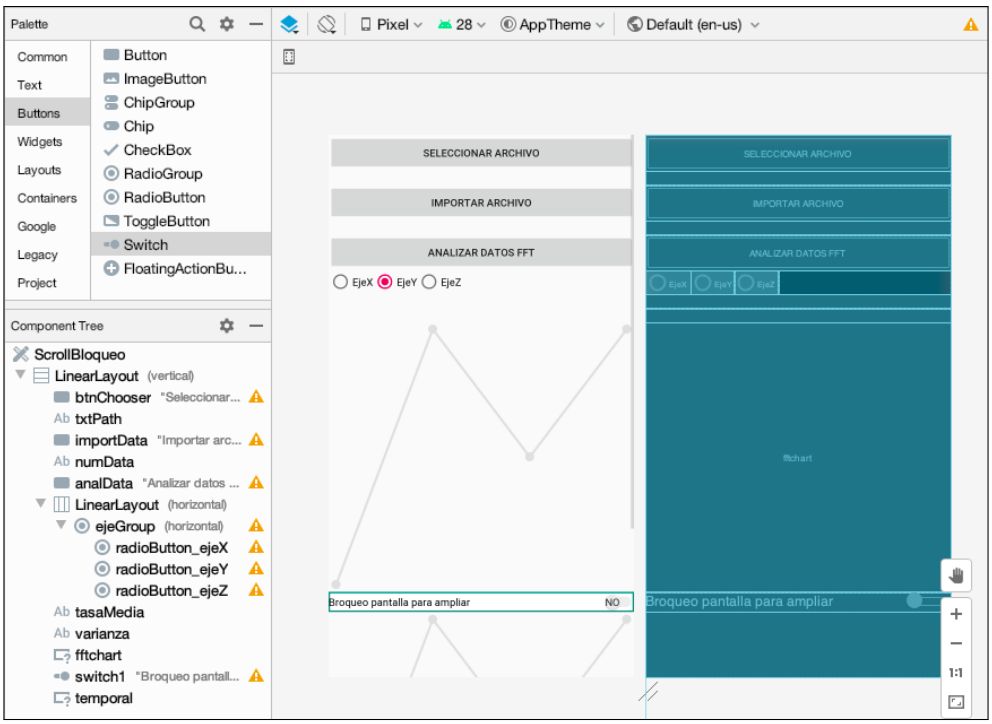


Fig. 5.28. Layout de la activity FFT

Las gráficas tendrán la funcionalidad de ampliar los datos si el usuario lo quiere para ver con mayor claridad los puntos importantes. Pero esta función interfiere con el desplazamiento hacia abajo del *ScrollView* por lo que se crea uno personalizado para que se pueda bloquear este desplazamiento y se pueda ampliar la gráfica de forma correcta. Se incluye un botón de tipo *switch* para bloquear y desbloquear este movimiento.

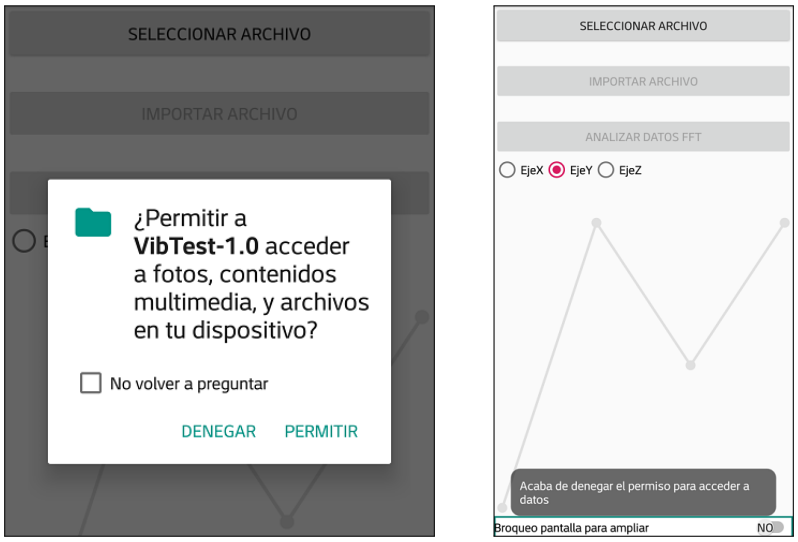
El código de este *layout* se puede encontrar en el anexo C.

5.4.2. Permisos y Selección de archivo

El primer paso para analizar un archivo será seleccionarlo entre los que estén en la memoria. Si se quiere analizar una medición de las que haya registrado el propio dispositivo, estas se encontrarán en la capeta "Acelerometrodatos" en la memoria de este. Véase la sección 5.3.7 para más detalle.

El proceso para elegir un archivo se inicia al pulsar en el botón "Seleccionar archivos". En primer lugar, si no se han dado ya los permisos para acceder a los datos de la memoria del dispositivo, aparece un aviso por pantalla para que el usuario acepte estos permisos como se puede ver en la figura 5.29 (a. Si no se aceptan no se puede pasar de este paso y la aplicación vuelve a la página principal de la *activity* con un mensaje por pantalla indicando que se han denegado los permisos como ve en la figura 5.29 (b.

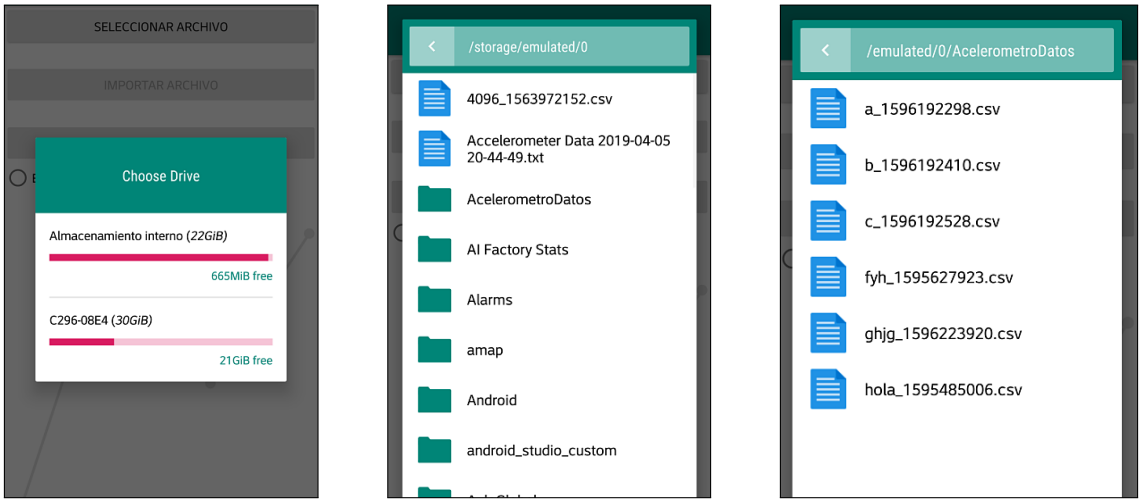
↓ Véase el código completo y comentado en la línea 364 del código C.2



(a) Aviso (b) Alternativas

Fig. 5.29. Permisos para acceder a datos de la memoria

Para realizar la selección del archivo *csv* que se quiere analizar se utiliza una biblioteca externa llamada *"Storage Chooser"* para facilitar el desarrollo de la aplicación y facilitar la tarea de elección de un archivo al usuario. Una vez se acepten los permisos aparecerá un formulario que actúa como una aplicación de archivos como la que se utiliza para encontrarlos en la memoria interna pero todo se produce dentro de la aplicación, haciendo el proceso más rápido y simple.



(a) Almacenamiento (b) Carpetas (c) Archivos

Fig. 5.30. Formulario para elegir el camino hasta el archivo a importar

Como se puede ver en la figura 5.30 el usuario sigue los pasos hasta llegar al archivo. Aparece una ventana emergente y se sigue la secuencia de carpetas hasta llegar al archivo que se quiere importar. Una vez seleccionado este archivo, el for-

mulario se cierra y debajo del botón de "Seleccionar Archivo" aparece en un cuadro de texto el camino hasta el archivo en la memoria del dispositivo como se puede ver en la figura .

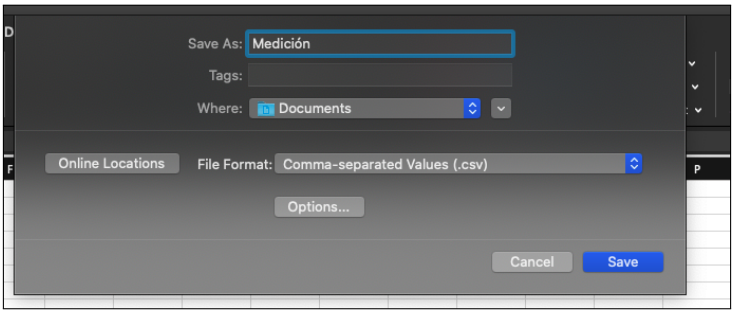
Este camino se almacena en una variable de texto llamada "path" que se guarda para su uso más adelante en la aplicación. También se muestra por pantalla en un cuadro de texto como se puede ver en la figura 5.32 (a, justo debajo del botón "Seleccionar archivo".

↓ Véase el código completo y comentado en la línea 406 del código C.2

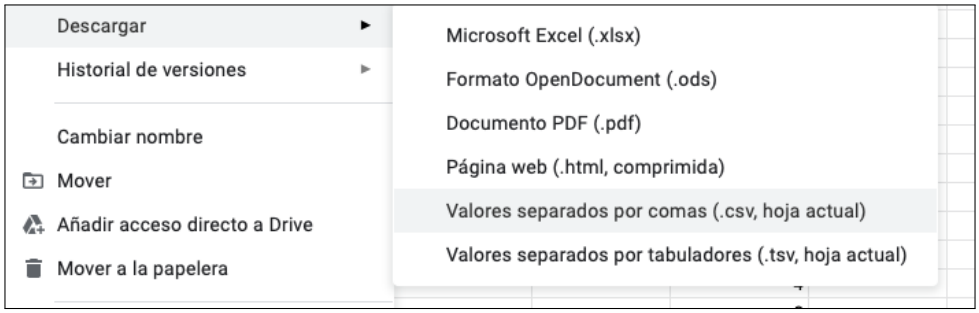
Importación de archivos externos

Si los datos que se quieren analizar se han obtenido con la aplicación, estos ya se encuentran en el formato correcto para que la aplicación los entienda. En caso contrario, si se quiere utilizar la aplicación para realizar el análisis pero los datos se han obtenido externamente se deben modificar hasta la forma correcta.

En primer lugar, los archivos tienen que estar en formato *csv*. Este es el formato que la aplicación entenderá y al que hay que pasar el archivo que se tuviera antes. Los distintos programas de hojas de cálculo tienen una interfaz más intuitiva en la que los datos se organizan en filas y columnas. Con estos programas se pueden guardas los datos en el formato de valores separados por comas o *csv* como se puede ver en la figura 5.31.



(a) Excel



(b) Docs

Fig. 5.31. Exportar archivos en csv

Las filas y columnas se pueden convertir es datos separados por comas de forma sencilla pero estos datos tienen que estar en el formato adecuado. La primera columna dedicada a los datos temporales de cada medición debe de ir en milisegundos. No importa si el primer dato empieza en cero milisegundos o no, lo importante es que la separación entre ellos vaya aumentando y sea en milisegundos.

Los datos del acelerómetro deben de ir en las siguientes tres columnas de forma que los datos para el eje "X" deben ocupar la segunda columna y así respectivamente con los otros dos ejes. Las unidades de estos datos deben de ser las del sistema internacional (m/s^2). Como los acelerómetros en estático la aceleración de la gravedad, los valores no oscilan respecto a cero pero eso no es ningún inconveniente porque solo importa la variación entre las mediciones. En el código 5.8 hay tres ejemplos de datos que la aplicación reconoce como válidos.

```
0 , 0    , 0 ,  0
4 , 0,12 , 0 , -0,34
9,  0,78 , 0 ,  0,2
-------------------------
1000 , 0    , 0 ,  0
1004 , 0,12 , 0 , -0,34
1009,  0,78 , 0 ,  0,2
-------------------------
0 , 0    , 0 ,  9
4 , 0,12 , 0 , -9,34
9,  0,78 , 0 ,  9,2
```

CÓDIGO 5.8. Ejemplo de datos csv válidos

5.4.3. Importación al dispositivo del archivo

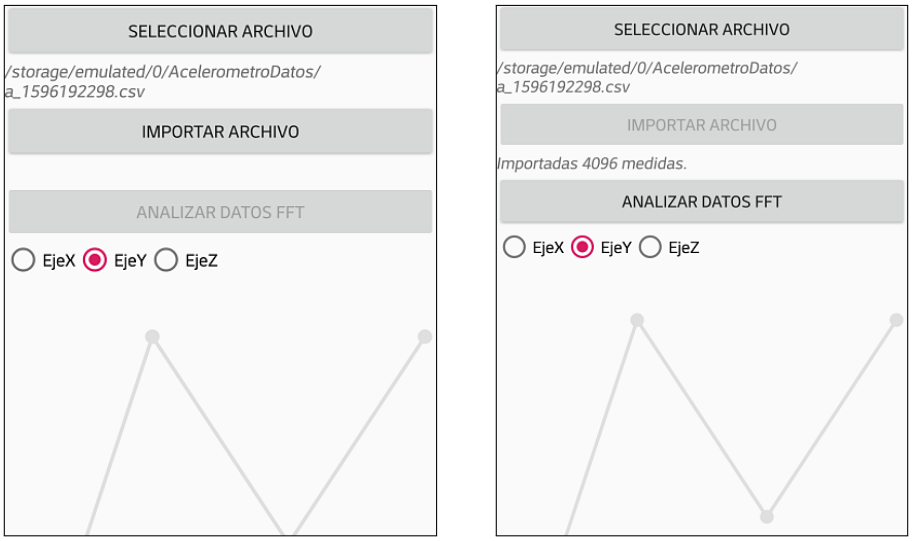
Una vez se haya seleccionado el archivo que se quiera analizar, el siguiente paso es importar todos los datos de ese archivo a la memoria temporal del dispositivo. Del paso anterior se sabe la localización del archivo y su nombre. Tras seleccionar el archivo la pantalla aparecerá como en la figura 5.32, de modo que el botón para analizar los datos aparece deshabilitado y el único paso a seguir es el botón para importar los datos a no ser que se quiera elegir un nuevo nombre para el archivo.

Para importar los datos, se crea un nuevo objeto que se llama "Mediciones". Aprovechando la sintaxis de Java, para almacenar los datos se creará un nuevo objeto que tiene distintos atributos. Los atributos de este objeto creado será el tiempo medida y las medidas de cada uno de los ejes por lo que serán cuatro atributos. Siguiendo una analogía, el objeto puede ser un coche y sus atributos pueden ser el color (azul, rojo, blanco...) o velocidad máxima (230,300...) y así en las mediciones cada color corresponde a una medida temporal y cada velocidad máxima a una medición del acelerómetro.

Cuando se hayan importado todos los datos, aparecerá un aviso por pantalla

en forma de cuadro de texto debajo del botón "Importar archivo" diciendo cuantas medidas se han importado. Este aviso se puede ver en la figura 5.32 (b. Se deshabilita el botón para importar ya que se acaba de utilizar y se habilita el botón para analizar que será el siguiente paso.

↓ Véase el código completo y comentado en la línea 136 del código C.2



(a) Camino (b) Importado

Fig. 5.32. Importación del archivo al dispositivo

5.4.4. Análisis FFT

El último paso y el más importante es pasar ya al análisis de los datos medidos. La pantalla se encuentra en este estado como aparece en la figura 5.32 (b, donde solo hay que elegir cuál de los tres ejes va a ser el analizado. Esta elección se realiza mediante un *RadioButton* de tres elementos.

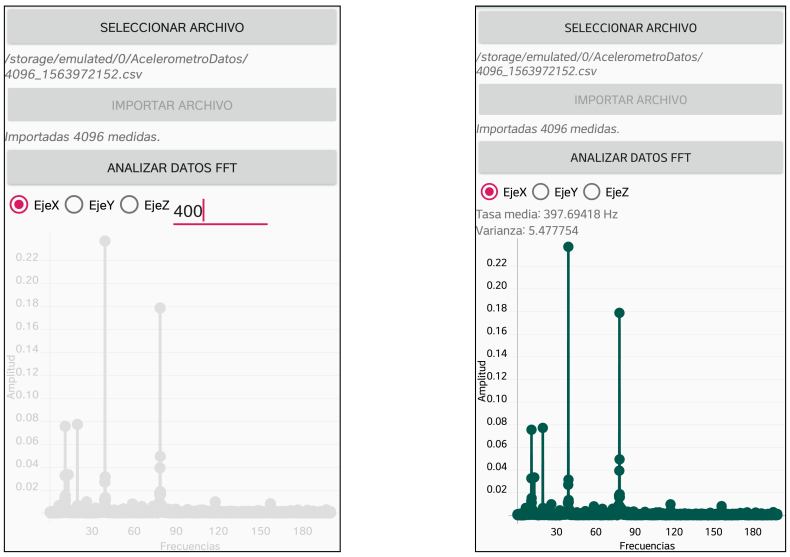
↓ Véase el código completo y comentado en la línea 556 del código C.2

Seleccionado el eje, se selecciona el botón de "Analizar Datos" e inmediatamente aparecerán las gráficas por debajo rellenas. La primera gráfica será la encargada de mostrar el resultado de las mediciones después de aplicarles la FFT y los resultados en el dominio de la frecuencia. La segunda gráfica muestra los datos del acelerómetro tal y como se tomaron respecto al tiempo. Para rellenar la primera gráfica habrá que realizar cálculos intermedios.

Primero se pasan los datos desde el objeto "Medición" a vectores para que sea más fácil trabajar con ellos. Como para realizar la FFT se debe tener un número de datos igual a una potencia de dos, el eje elegido si su tamaño es menor del necesario, se rellena con ceros hasta la siguiente potencia de dos. Se almacenan en variables el tamaño del vector antes y después de la corrección.

Se realiza el cálculo de la tasa de muestreo de las medidas y la varianza de estas. En cuanto a la tasa de muestreo se realiza la media de todas las separaciones temporales entre cada medida para obtener la tasa de refresco con la que se trabajará. No es el método más óptimo ya que la tasa de muestreo es variable con el tiempo pero es una buena aproximación. Se calcula la varianza de la muestra que muestra la variación de las medidas en amplitud respecto a una medida base. Ambos datos se mostrarán como texto como aparecen en la figura 5.33 (b).

La figura 5.33 muestra la forma actual y la forma anterior de obtener la tasa de muestreo. Anteriormente se introducía el valor a mano pero esto lleva a un error grande si no se sabe con exactitud dicho valor de la tasa de muestreo.



(a) Tasa seleccionada (b) Tasa calculada

Fig. 5.33. Elección de tasa de muestreo para el análisis

En el resultado de la FFT se compara las frecuencias predominantes. Para la frecuencia, los valores van desde cero hasta la mitad de la tasa de muestreo de la medición y su resolución viene dada por la cantidad de medidas, siendo mayor a mayor número de medidas. El eje de las frecuencias por lo tanto se calcula sabiendo los dos datos necesarios.

Para el cálculo de la FFT se utilizará una librería externa que tiene incluidos los cálculos necesarios de forma que se incluyen datos de entrada y se obtienen los resultados. Los datos de entrada son las propias mediciones, y el número de medidas. El resultado de la FFT es un número complejo que se almacena en dos vectores diferentes que son su parte real e imaginaria y de los que después se calcula el módulo. Y con el módulo y las frecuencias ya se tienen los datos para hacer la gráfica.

↓ Véase el código completo y comentado en la línea 200 del código C.2

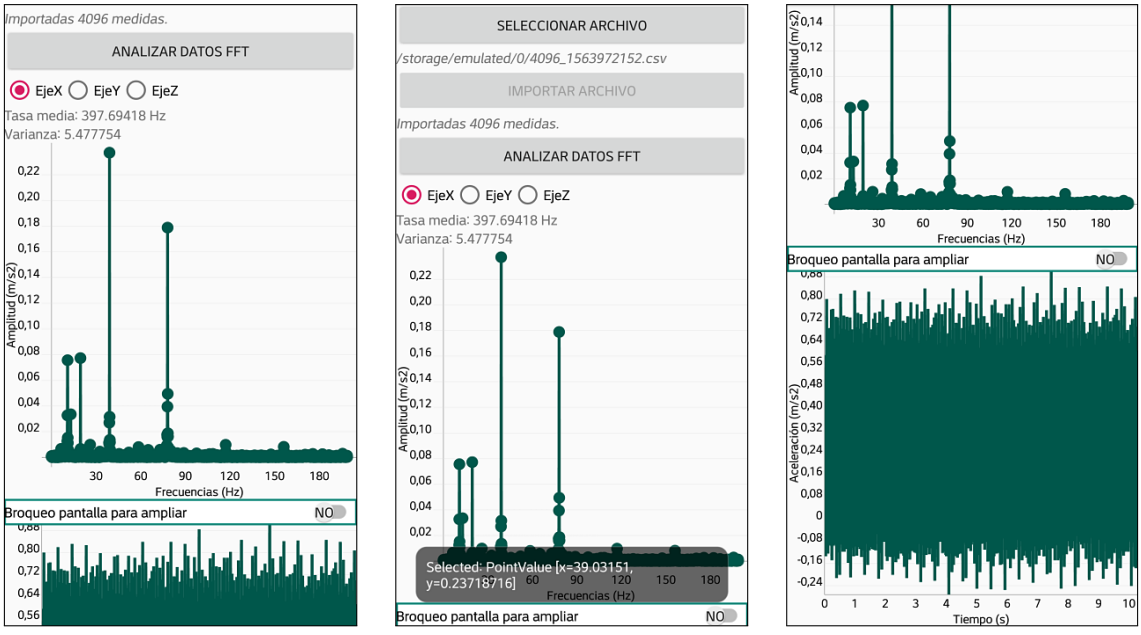
Gráficas

Para las gráficas se utiliza una librería externa de gráficas diferente a la utilizada en la primera *activity* de la aplicación [17]. Para estas gráficas se tienen otros requerimientos, como que se puedan ampliar de forma fácil además de que se pueda saber el valor de punto pulsando sobre él. Ya se tienen los datos calculados por lo que queda configurar las gráficas ye rellenarlas.

La **primera gráfica** va a ser la que muestre el resultado de la FFT y será una gráfica de puntos. Estos puntos serán algo más grande de tamaño para que destaquen las frecuencias importantes. Si estos puntos se seleccionan entonces se puede ver los valores de ese punto mediante un aviso por pantalla. Como solo importa los valores que destacan, no ocurre nada con que los que no, unque no sean fácil de seleccionar individualmente.

La **segunda gráfica** muestra los valores de las mediciones en el tiempo y se desactivan los puntos ya que harían que la gráfica fuese difícil de leer. La ampliación en esta gráfica solo se permite en el eje "X" que es el del tiempo para poder observar las mediciones con más detalle sin perder la referencia de la amplitud de las vibraciones. En esta gráfica se podría ver si la vibración está amortiguada ya que iría perdiendo amplitud con el tiempo. Con las pruebas que se realizarán en el apartado de resultados se obtendrán amplitudes constantes al ser una vibración forzada.

↓ Véase el código completo y comentado en la línea 454 del código C.2



(a) Primera gráfica (b) Punto seleccionado (c) Segunda gráfica

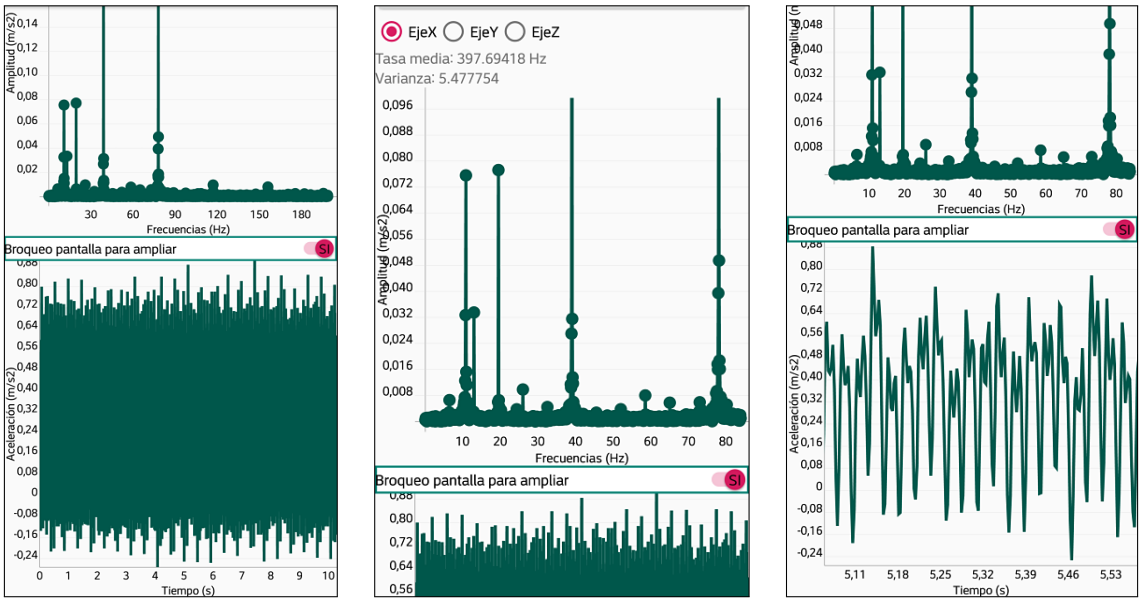
Fig. 5.34. Gráficas con los resultados

Ambas gráficas no entran en la pantalla por lo que se inserta el *layout* (*ScrollView*) que permite deslizar hacia abajo la pantalla. La aplicación confunde el ampliar la gráfica con desplazarse por la pantalla por lo que cuando se quiere ver algún detalle en las gráficas es mala.

Para solucionar este problema, se recurre a bloquear el deslizamiento de la página. El *ScrollView* que trae Android por defecto no tiene la opción de bloquear este deslizamiento vertical por lo que se crea un *laylout* personalizado que sí lo pueda hacer. Así se crea un nuevo *ScrollView* que se llamará "ScrollViewBloqueo" que sí se podrá bloquear cuando se necesite.

El bloqueo del deslizamiento vertical se debe hacer cuando la gráfica que se quiera ampliar esté completa situada en la pantalla. Para realizar el bloqueo se utilizará un nuevo elemento *View*, un *Switch*. Este elemento se comporta como un interruptor de la luz, teniendo dos posiciones que se utilizaran para el bloqueo de la pantalla. Además para hacerlo más fácil para usuario, se incluye dentro de este elemento las palabras "SI" y "NO" y así no haya confusión de saber si está bloqueado o no.

↓ Véase el código completo y comentado en la línea 573 del código C.2



(a) Switch activado (b) Gráfica ampliada (c) Gráfica ampliada

Fig. 5.35. Bloqueo para ampliar las gráficas

Como se puede ver en la figura 5.35, el interruptor está posicionado entre ambas gráficas para que se pueda bloquear teniendo en pantalla la gráfica que se quiera ampliar. En la primera gráfica están los resultados de la FFT de modo que se puede ampliar sobre una zona que no esté claro qué frecuencia es la más destacada. En la segunda gráfica se puede ver las medidas frente a el tiempo y en el caso de la figura 5.35 (c , se puede ver la gráfica ampliada.

6. DESARROLLO DE LOS SISTEMAS DE PRUEBA

6.1. Construcción generador de vibraciones

Siguiendo los principales objetivos del proyecto de conseguir una forma de medir y analizar vibraciones de una forma fácil y accesible, para probar las capacidades de la aplicación se desarrolló un generador de vibraciones improvisado con partes que iban a ser desechadas.

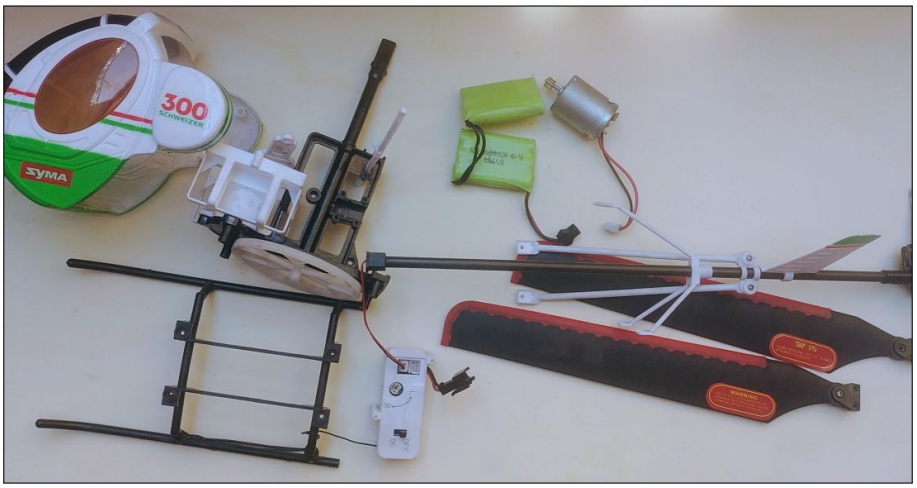
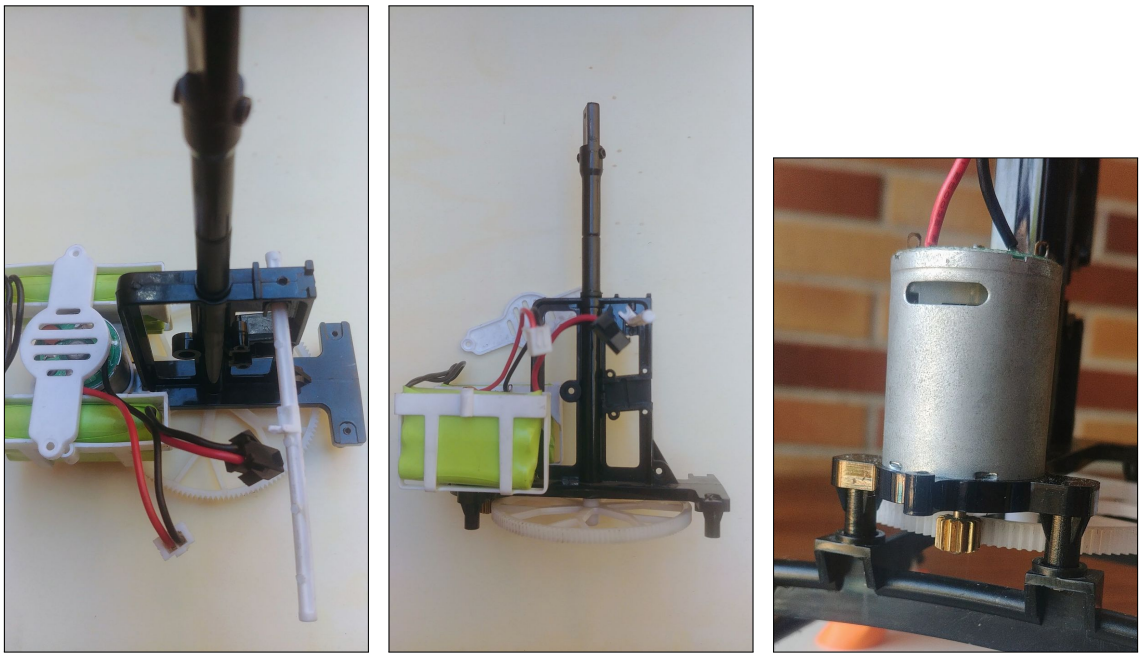


Fig. 6.1. Despiece del helicóptero

El objeto a desechar era a ser un helicóptero de radiocontrol del fabricante *Syma* modelo "300". Este tenía parte de las hélices descompensadas por una reparación anterior e iba a ser desechado. Se quiso aprovechar las partes como motor y otros componentes para realizar un montaje que de alguna forma generase vibraciones pero se vio que la estructura interior principal era utilizable. Tras eliminar la carcasa y otras partes no esenciales como la placa de control, la cola del helicóptero y las hélices se queda una estructura con el motor y el eje que movía las hélices así que será ese eje el que genere las vibraciones junto con las que genere el motor.



(a) Vista superior (b) Vista lateral (c) Motor 370

Fig. 6.2. Estructura central del helicóptero

Eje y motor girarán a distintas velocidades debido a que están conectados mediante una conexión de piñón y rueda siendo la velocidad de rotación del motor superior a la del eje por el número de dientes menor en el piñón. Según especificaciones de este modelo, el motor es un motor 370 que se suele utilizarse en radiocontrol. Según las especificaciones de modelos de repuesto este motor tiene un rango de funcionamiento de 3 a 7,4 V que se corresponde con el voltaje de funcionamiento escrito en la batería que tenía este modelo a 7,2 V y 650mAh.

TABLA 6.1. VOLTAJE FUNCIONAMIENTO MOTOR 370

Voltaje sin carga (V)	Corriente (mA)	Velocidad (RPM)
3,0	620	13200
4,5	770	18800
6,0	900	25200
7,2	1100	31000

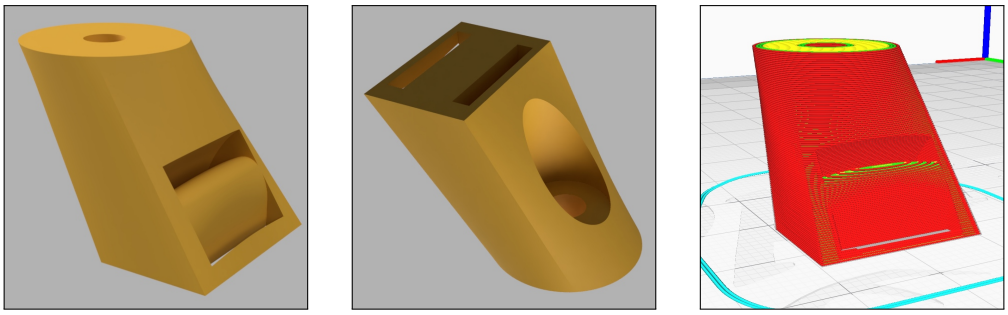
Diseño

Con la parte de la estructura que se ha querido conservar y añadiendo las patas originales sobre las que aterrizaría el helicóptero para sujetarlo, debido a que estas patas están hechas ya para encajar con la estructura central. Ahora hay que sujetar toda la estructura de alguna forma y que se quede fija para poder situar el dispositivo móvil y realizar la medida de las vibraciones.

Las patas tienen una forma cilíndrica y el plástico no es demasiado duro para poder atornillarlas directamente en una superficie por lo que se optó por agarrarla con bridas alrededor del cilindro. Nuevamente hay que encontrar una forma de sujetar estas bridas de plástico con la base. Además hay que tener en cuenta que se va a querer poner unos muelles que separen el montaje de donde apoye para que las vibraciones de la estructura no sean amortiguadas por la superficie a la que esté anclada.

Con la posibilidad de utilizar una impresora 3D para realizar estas piezas, se hizo un diseño que solucionase los problemas descritos anteriormente. Se utilizó el programa *Fusion 360* para su diseño. Los requerimientos de la pieza es que pueda coger con bridas de plástico en su parte superior a la base de la estructura y en la parte inferior tenga una zona plana para que pueda apoyar en una chapa o tabla. La pieza se debe poder sujetarse a la parte inferior y el tornillo que se utilice para ello debe incorporar el muelle y actuar el tornillo como soporte de la estructura entera y así utilizar el número menor posible de piezas.

En la imagen se puede ver el diseño final de la pieza para el anclaje de la estructura.



(a) Frontal (b) Trasera (c) Cura

Fig. 6.3. Vistas del render de la pata y en el programa Cura

También se diseñará un disco que irá acoplado a el extremo superior de la estructura. Este se deberá conectar a esta con la forma que tiene este eje que es de un rectángulo con los lados de forma circular. Se pondrá un pin para que no pueda salirse y además se diseñará para que tenga un buen ajuste. Este disco contará con varios agujeros para poder añadir pesos y añadir desequilibrios a la estructura.

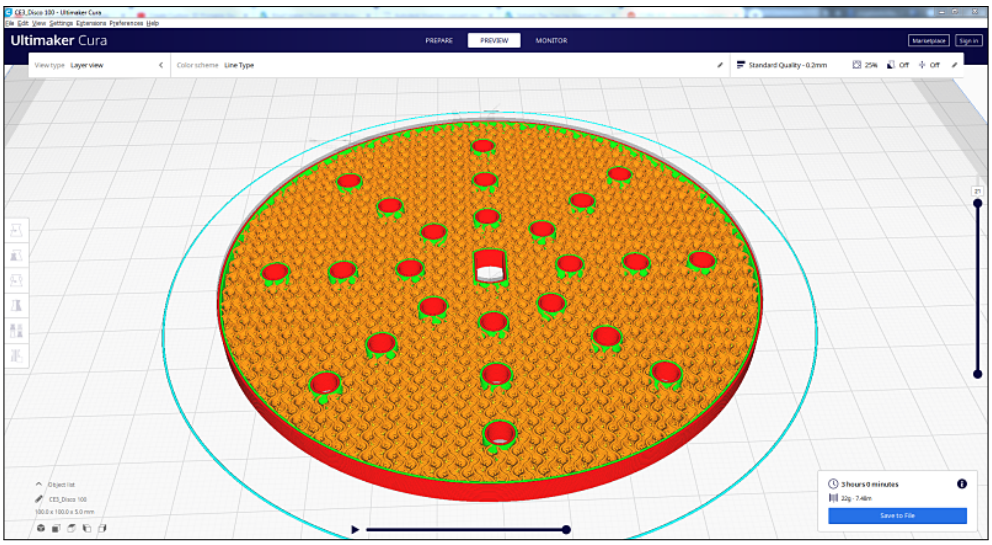


Fig. 6.4. Disco perforado laminado en Cura

El disco se procesará con el programa *Cura* para proceder a su impresión. Este programa separa los perímetros que son el espesor de las capas externas de el relleno interior para ahorrar en material. En este caso se le dio un relleno de tipo "giroide" para que el peso quede repartido y quede equilibrado además de aumentar la densidad de este relleno para darle más rigidez al estar girando a altas velocidades.

Finalmente su peso será de 22 gramos y toma un tiempo total de 3 horas de impresión. El material tanto para el disco como para las patas será PLA del fabricante BQ. Este material clasificado como biodegradable tiene una densidad de 1,24 g/cm^3 además de tener un módulo elástico a flexión de 3600 Mpa y un módulo elástico de tracción de 1230 o 1120 Mpa dependiendo de la dirección de la impresión. Estos y más datos se pueden ver en la ficha técnica del material en [24].

Por regla general las piezas impresas tienen el fallo a tracción con la carga perpendicular a la dirección de las capas. Esto se produce por la falta de adhesión entre las capas del material y ocurre antes de que en material se produzca una rotura en estado plástico. Ninguna de las piezas impresas del generador soportarán este tipo de cargas (las patas funcionarán a compresión y el disco a tracción en el sentido de las capas debido al giro) y además las cargas soportadas son muy pequeñas en comparación de los límites elásticos del material. Más información sobre la variación de la resistencia del material con la orientación de las capas en [25] y [26].

Fabricación y Montaje final

Se procede a imprimir las piezas restantes en este material; las patas y el disco perforado como se puede ver en la figura 6.5. Para su impresión se utilizó el modelo de impresora 3D de deposición de plástico fundido (FDM) "Ender 3" del fabricante "Creality". Es una impresora de las consideradas económicas pero bien calibrada y ajustada produce muy buenos resultados. Se utilizó una base de vidrio de borosilicato con buenas propiedades térmicas con una capa de laca para que la primera capa de la impresión tenga buena adherencia. Esta impresora cuenta con cama calefactable y para la impresión en este material se calienta hasta unos 50-55 ºC. Este material puede imprimirse en una base sin calentar pero los resultados son mejores calentándola y se evita que la pieza pueda despegarse durante la impresión. Se puede ver el disco durante su impresión en la figura 6.5.

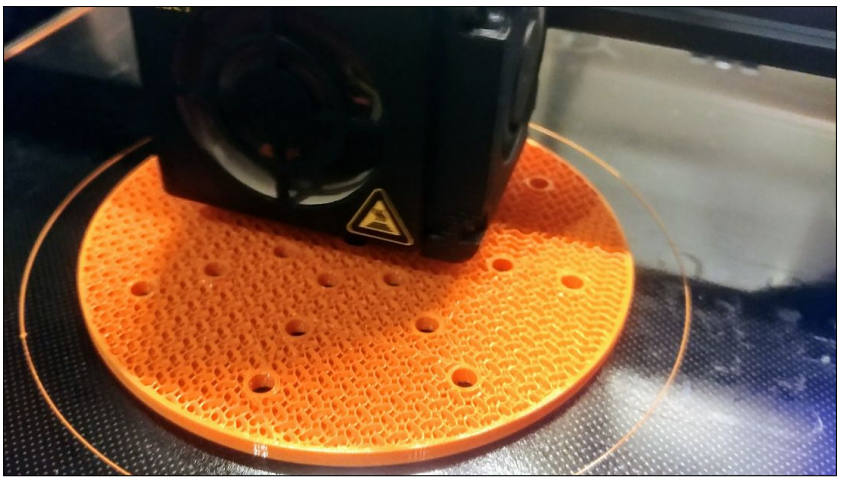


Fig. 6.5. Disco perforado durante su impresión

Se sigue el mismo procedimiento para las patas de la estructura llevando aproximadamente 39 minutos en la impresión de cada una de ellas. Para separar las piezas de la base estas se meten con la base de cristal en un refrigerador y la propia compresión del plástico con la temperatura hace que se despeguen sin esfuerzo. Las patas tienen a su vez un postprocesado debido a los soportes de material necesarios para las partes que se imprimen en voladizo. Estos soportes se retiran con unos alicates de forma fácil. La pieza resultante es la que se puede ver en la figura 6.6 (a y (b.

Como se comentó en el apartado de diseño, se utilizarán bridas para unir la estructura central a las patas. Con los canales interiores que tiene la pieza impresa se puede pasar un extremo de la brida y cerrarla de forma que abarque la pata y el cilindro de la estructura. Se colocan dos bridas para que el apoyo sea uniforme y se aprietan. Se repite este proceso para cada una de las cuatro patas y el resultado es el que se puede ver en la figura 6.6.

(a) Frontal (b) Trasera (c) Brida

Fig. 6.6. Vistas de la pata impresa

En la figura 6.7 se puede ver el disco perforado ya impreso. En la fabricación de elementos en impresión 3D si se tienen agujeros interiores, estos deben ser ligeramente de mayor tamaño ya que por la expansión con el cambio de temperatura, los agujeros finales son de menor tamaño.

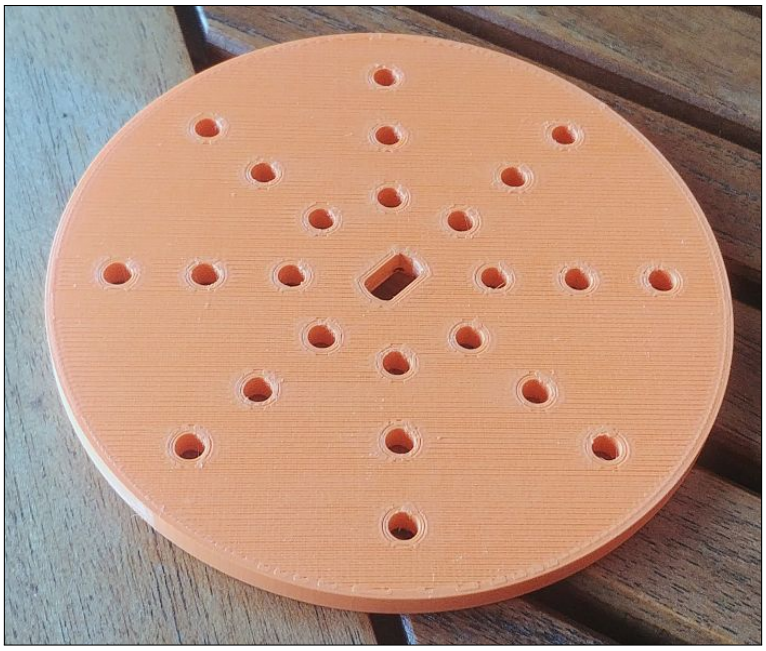


Fig. 6.7. Disco para añadir desequilibrio impreso

Se corta un rectángulo de madera que actuará como base de la estructura donde apoyarán sus patas. Teniendo en cuenta el efecto final en las mediciones, el material y espesor no son relevantes así que con que tenga buena rigidez para que absorba el mínimo de vibraciones es suficiente. Se aprovecha la sujeción en la parte inferior de la pata para incluir in tornillo más largo e incorporar un muelle para separar la estructura de la superficie donde se vaya a apoyar. En la figura 6.8 se puede ver como el tornillo que a su vez soporta la estructura es lo suficientemente largo para tener la base, el muelle y la tuerca a lo largo de su longitud. Estos cuatro tornillos

son los que harán de soporte de la estructura completa.

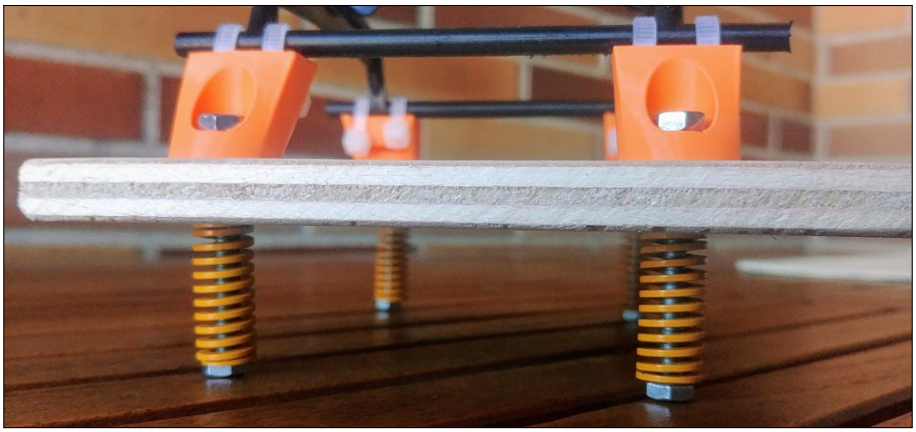
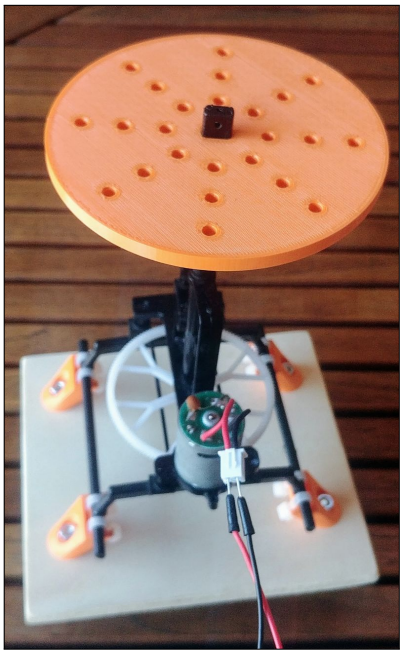
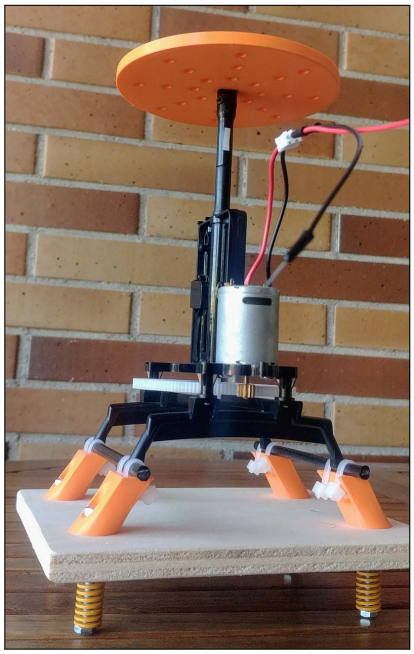


Fig. 6.8. Vista lateral de la base del generador

Finalmente se tiene la estructura completa como se puede ver en la figura 6.9. Variando la velocidad del motor y colocando el dispositivo en el espacio que queda entre la estructura y la base se realizarán las mediciones.



(a) Vista 1



(b) Vista 2

Fig. 6.9. Estructura del generador completa

6.2. Construcción banco de pruebas

Consideraciones iniciales

Buscando una forma de comprobar el funcionamiento de la aplicación de una forma más precisa, se decide la construcción de un banco de pruebas en el que se puedan realizar mediciones de una manera más controlada. La imposibilidad de probar la aplicación en el banco de pruebas de la universidad por motivos de seguridad por el Covid-19 además de realizar las prácticas en un taller mecánico propiciarán la construcción de uno.

Se debe tener en cuenta varias consideraciones iniciales y limitaciones a la hora de su construcción que serán las siguientes:

- Las principales variables que son la velocidad de rotación así como la amplitud de la vibración generada tienen que ser controlables por el usuario.
- El límite de velocidad de rotación lo da la capacidad máxima del dispositivo de medición: La máxima tasa de refresco del dispositivo son 400 hercios.
- Los materiales deben de resistir el giro a altas velocidades.
- El banco de pruebas debe de ser sencillo en su fabricación. Se tiene en cuenta la maquinaria disponible en un taller mecánico.
- Tiene que tener posibilidad de variación en la geometría del banco. Tanto la separación entre los soportes del eje como de los desequilibrios se tienen que poder mover y asegurar sin dejarlos fijos.

6.2.1. Diseño

Las primeras directrices en el diseño es que fuese un eje movido por motor alineado con este. Se necesitarían rodamientos para sujetar el eje, aguantar la rotación a altas velocidades y reducir el rozamiento.

Por las necesidades de velocidad de rotación y torque se utiliza un motor 777 del fabricante EsportsMJJ. Este motor esta destinado a electrodomésticos o máquinas eléctricas. Tiene un voltaje nominal de 24 V al que funcionaría a 7000 rpm. Ajustando el voltaje se podrán obtener distintas velocidades de rotación. En la tabla 6.2 se puede ver los datos de voltaje que da el fabricante para el motor sin carga. La potencia se calcula como la multiplicación del voltaje por la corriente y el par del motor como el cociente entre potencia y velocidad en rad/s.

TABLA 6.2. VOLTAJE FUNCIONAMIENTO MOTOR 777

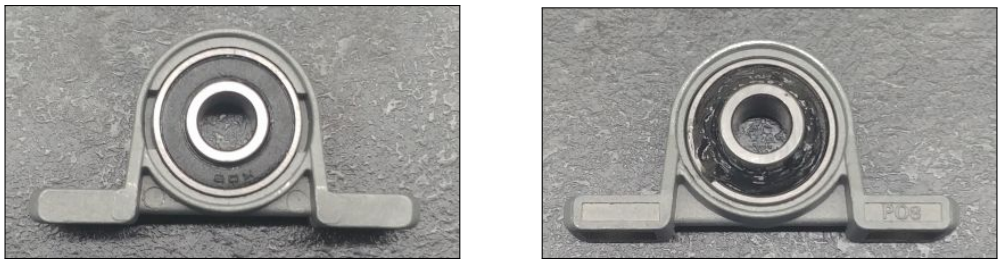
Voltaje(V)	Corriente (A)	Velocidad (RPM)	Potencia(W)	Par(N/m)·10^{-3}
12	0,14	3500	1,68	4,58
18	0,15	4500	2,7	5,72
24	0,16	7000	3,84	5,23
30	0,17	8100	5,1	6,01
36	0,20	9000	7,2	7,63

(a) Motor (b) Acople

Fig. 6.10. Motor 777 y acople flexible

Este motor tiene dos taladros roscados de métrica 4 que se utilizarán para sujetarlo y un eje de salida de diámetro 5 mm que será el que haya que conectar con el eje principal del banco de pruebas. Para conectar el eje del motor con el eje del banco se utiliza un acople flexible para eliminar un posible desalineamiento entre ambos ejes. Se podría hacer que los ejes estuviesen desalineados en su union pero por simplicidad se decide que las vibraciones solo se causen por desequilibrios.

Para sujetar el eje se utilizarán cojinetes para tener el mínimo rozamiento en el giro. Los cojinetes que se utilizan tienen que llevar carcasa para poder sujetarlos a un soporte que es el que tendrá que subir la altura del eje la altura necesaria. Este soporte será la conexión entre el cojinete y la base del banco. Con la intención de añadir discos perforados para añadir vibraciones estos ejes, estos cojinetes deben de estar a una altura suficiente para que los discos montados en el eje puedan girar y tengan espacio.



(a) Vista1 (b) Vista 2

Fig. 6.11. Cojinete para el apoyo del eje

Estos soportes deben ser una pieza sólida a la que se pueda acoplar en la parte superior el cojinete y en la parte inferior a la base. Además, la carcasa de los cojinetes tiene unos agujeros para poder atornillarlo así que se utilizará un material para estos soportes que pueda ser mecanizado y realizar una rosca. Se decide utilizar un bloque de aluminio que tendrá dos agujeros roscados en la parte superior y dos agujeros roscados en la parte inferior. Es un material que se puede mecanizar de forma más fácil que otros metales, es rígido y ligero.

Los dos agujeros en la parte inferior serán los que conectan el soporte con la base. Como se quiere que los soportes sean capaces de desplazarse a lo largo del eje, la base tendrá que tener algún tipo de rasgado para poder moverlos lateralmente.

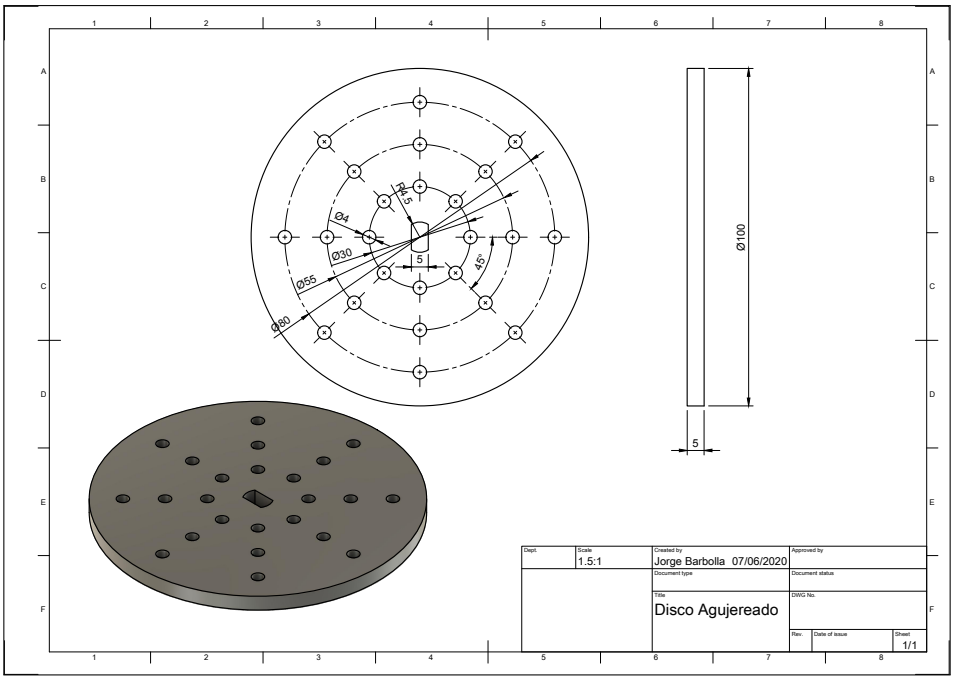


Fig. 6.12. Plano del disco para añadir desequilibrios

Por último, faltan los discos para añadir desequilibrios. Estos serán unos discos perforados para poder añadir pesos en distintos puntos de estos en forma de tornillos sujetos con una tuerca. Dependiendo donde se coloque estos pesos, tienen distintos

efectos en las vibraciones que es lo que se quiere medir. Para poder mover los ejes de posición dentro del eje y poder sacarlos se tienen que poder fijar y asegurarlos al eje con un tornillo. Para esto se coloca un cilindro con agujero roscado, y el tornillo que se añada ahí será el que fije el disco al eje de forma segura.

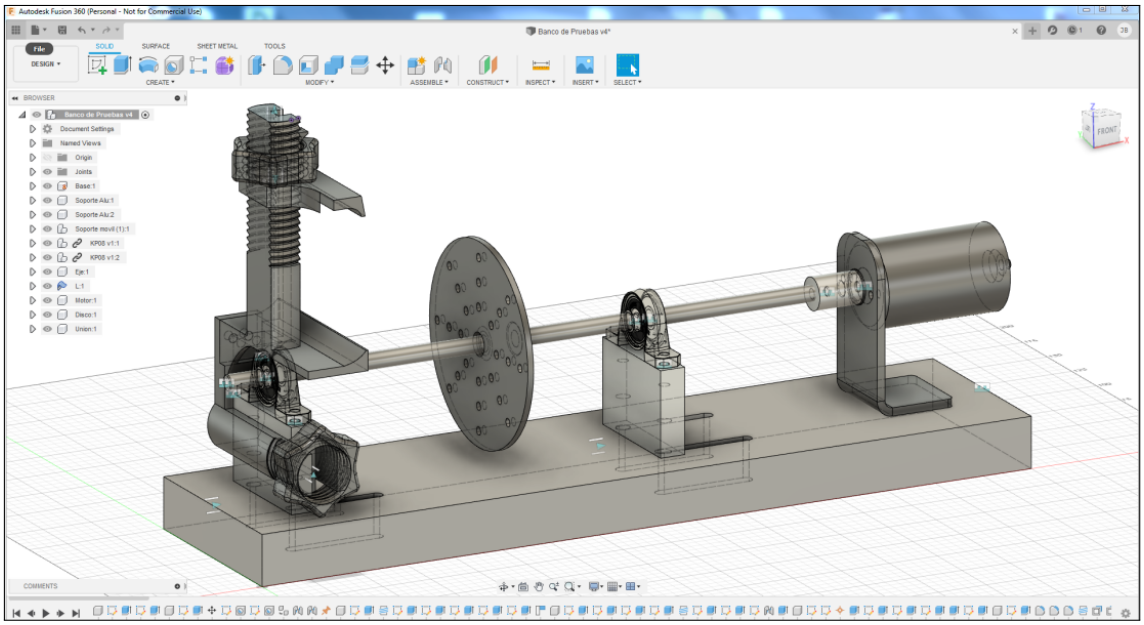


Fig. 6.13. Banco de pruebas en Fusion 360

Una vez definidas las piezas se realiza un modelo del conjunto completo con el programa Fusion 360 como se puede ver en la figura 6.13. De esta forma se puede asegurar de que las medidas son correctas antes de fabricar las piezas y que todas las piezas encajarán en su lugar. Este modelo servirá de base para diseñar el soporte para colocar el dispositivo móvil.

6.2.2. Fabricación y Montaje

Para la fabricación de la base se utilizará una base de polietileno de alta densidad. Su bajo peso y su facilidad para ser mecanizado además de sus propiedades mecánicas hace que sea un buen material para mantener la estructura del banco de pruebas. Se cortará a la medida con un disco, y las ranuras para poder mover los soportes de posición se realizan con una fresadora. Así estos soportes pueden moverse 50 mm cada uno que es la longitud de la ranura que tiene una ancho suficiente para que pase el tornillo que sujeta los soportes con la base.

Los soportes están fabricados de aluminio y tienen agujeros que se roscaron con la fresadora a la distancia de los agujeros de los cojinetes. Se necesitaba un material que se pudiese roscar, buena rigidez y el mínimo peso posible así que el aluminio es una buena opción.

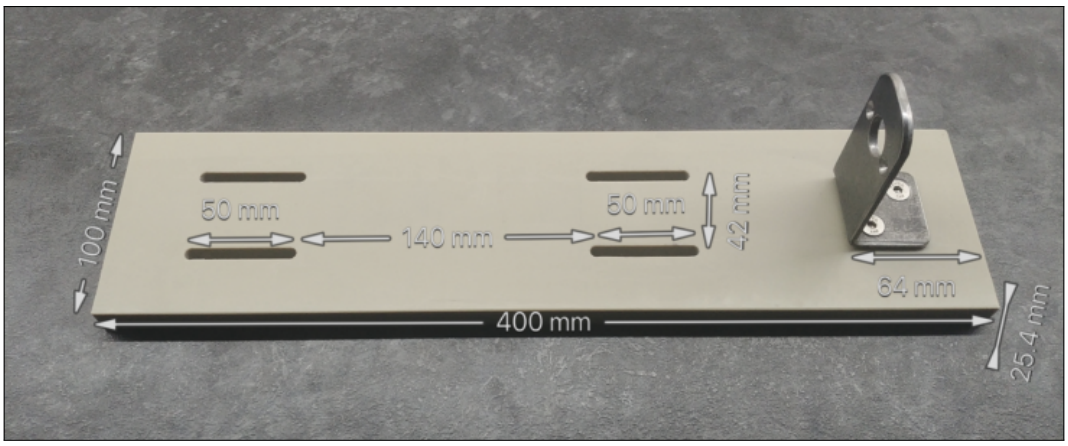
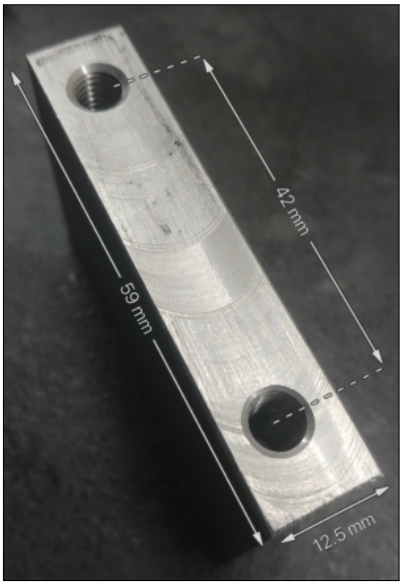
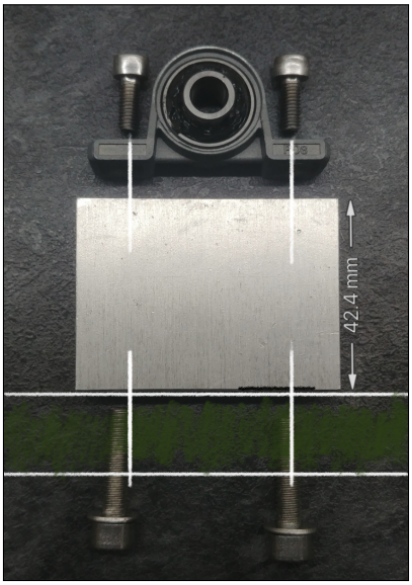


Fig. 6.14. Base del banco de pruebas



(a) Agujeros roscados



(b) Conjunto

Fig. 6.15. Soporte del cojinete

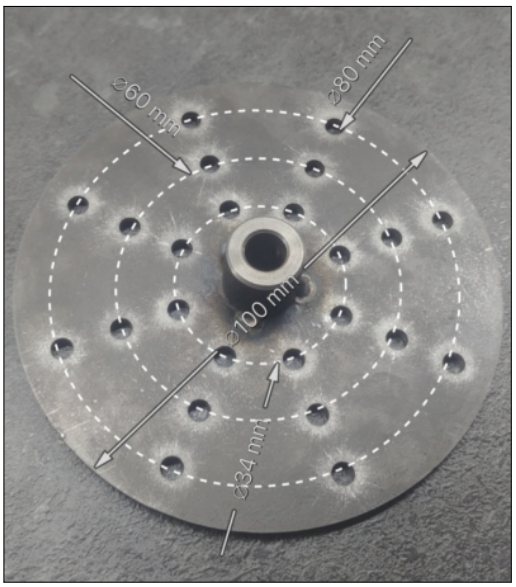
El eje principal es de acero inoxidable 304 de 8 mm de diámetro y este se corta a la longitud elegida. Es una medida de diámetro común y su fácil acceso además de que la medida también sea común para los acoples de 5 a 8 mm llevaron a la decisión de coger este diámetro para el eje.

Para sujetar el motor a la base se utilizará una "L" y para darle forma se utiliza una plegadora y se deja la tira de acero a $90^{\underline{o}}$. Se le harán distintos agujeros a la "L", dos para los tornillos que sujetan el motor a la "L", otro para un círculo que sobresale en el eje que puede ser utilizado para su montaje en otras máquinas. Se añaden otros dos agujeros para sujetar la "L" a la base y como esta unión se quiere que sea permanente se atornillan directamente los tornillos en la base de polietileno.

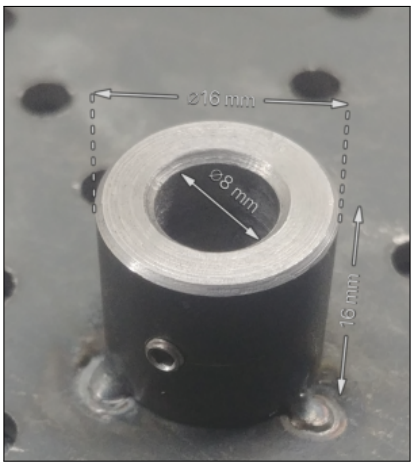
Para fabricar los discos se utiliza el método de corte por láser. La necesidad de

que los discos sean de acero añadido a la necesidad de tener mucha precisión para que el sistema permanezca balanceado hace que este sea el sistema más adecuado. A estos discos se les suelda un cilindro con un tornillo para poder asegurarlos al eje.

(a) L



(b) Disco

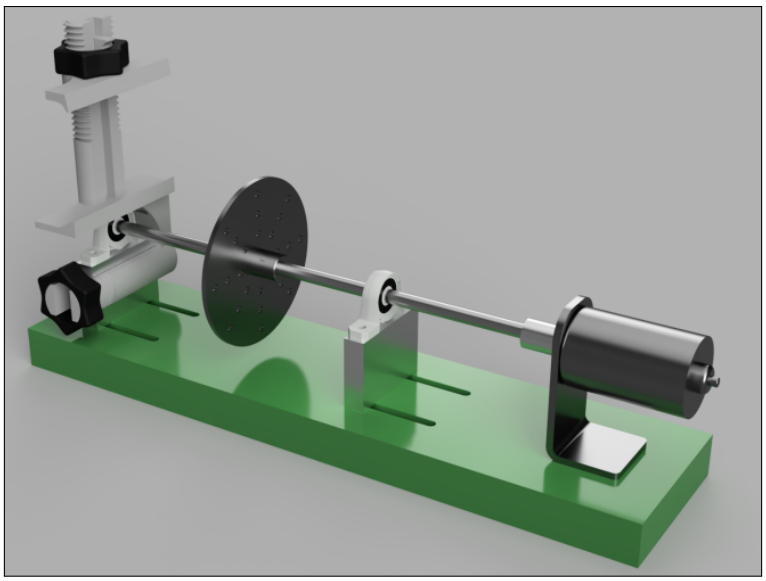


(c) Cilindro

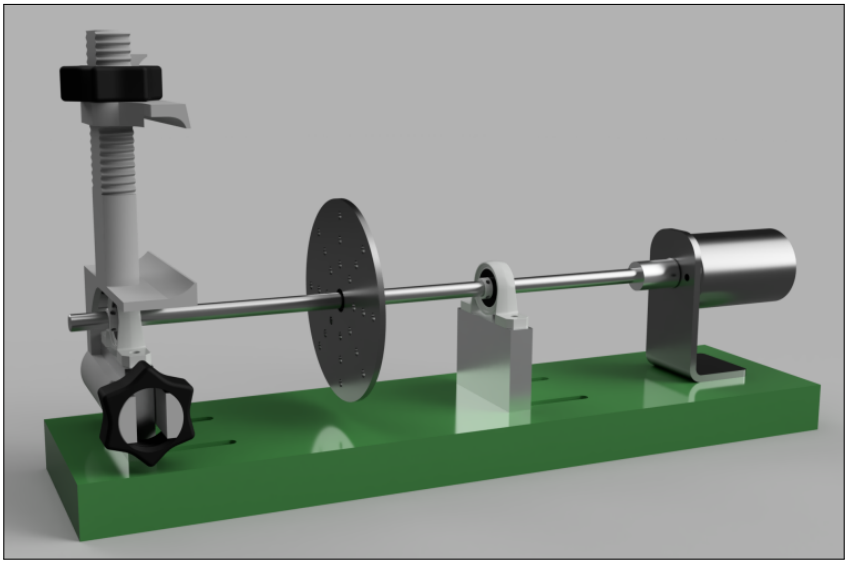
Fig. 6.16. Imágenes de la "L" y el disco

Para el montaje final, se atornillan los elementos de sujeción a la base como son los dos soportes de los cojinetes y la "L". Se añaden los cojinetes y el motor atornillándolos. Los soportes no se deben apretar del todo ya que al introducir el eje alinea los rodamientos y ya es cuando se aprieta todo definitivamente. Al introducir el eje en los dos rodamientos y se encuentre en su posición final se aseguran con dos tornillos que tiene cada rodamiento para para unirlo con el eje. Si se quiere introducir los discos entre los dos ejes se debe introducir el disco en el eje antes de que este entre en el segundo cojinete.

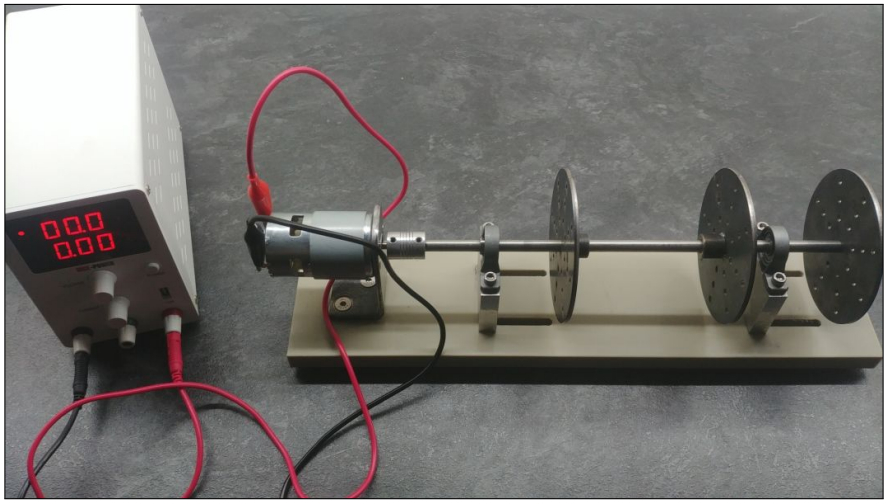
Variando la distancia entre los soportes, el número de discos, las posiciones de los discos y los pesos añadidos a los discos se pueden generar vibraciones de distinta frecuencia y amplitud. Además de estas variaciones se puede variar la velocidad de rotación del eje por lo que se puede comprobar la aplicación en una gran variedad de casos.



(a) Render 1



(b) Render 2



(c) Real

Fig. 6.17. Banco de pruebas finalizado

6.2.3. Equipos complementarios

Tacómetro

Este será el instrumento encargado de medir la velocidad de rotación del eje. Se utiliza el modelo DT-2234C de la marca WINGONEER. Su funcionamiento se basa en utilizar una cinta reflectante sobre la que incide un láser y mediante la rotación, se impide que el láser vuelva al tacómetro si no está actuando sobre la cinta. Cada segundo se realiza el cálculo de cuántas veces se ha cortado la conexión del láser y saca por pantalla la velocidad de rotación en revoluciones por minuto.

Fig. 6.18. Tacómetro digital

Estos tacómetros son más precisos en cuanto mayor sea la velocidad y mayor sea el número de mediciones por lo que para velocidades de rotación bajas se coloca dos cintas y después se divide el resultado de la velocidad de rotación por dos.

Teniendo en cuenta la variación de las medidas con cada cálculo cada segundo, se escogerá la medida que sea más estable y se repita más. Esto lleva a un error que es mayor a velocidades más bajas que hace que exista una incertidumbre asociada al método de medida y la propia herramienta. Por el uso que se va a dar a estos datos, esta incertidumbre no tiene importancia ya que no se va a hacer un control exhaustivo de las frecuencias y no se necesita un dato muy preciso de la velocidad de rotación, tan solo tiene que ser orientativo.

Fuente de alimentación

La fuente de alimentación utilizada es de tipo variable para poder variar la velocidad de rotación del motor. Esta fuente tiene un rango de voltaje de salida de 0-60 V, de corriente de 0-5 A por lo que puede dar una potencia máxima de 300W. La resolución por la pantalla del voltaje es de 100 mV y de corriente de 10 mA. Estas mediciones se complementarán con un multímetro para tener datos del voltaje más precisos. Se conectarán las dos salidas de la fuente con las dos conexiones del motor.

La regulación de línea es <5 mV siendo este valor la capacidad que tiene la fuente de alimentación de mantener el valor de voltaje de salida con un voltaje de entrada fluctuante. Así como la regulación de carga <5 mV siendo esta la capacidad de mantener el voltaje con valores de carga variables, es decir, si en este caso el motor funciona libre o de si tiene un peso que mover. Se puede ver la fuente en la figura 6.17 (c) en la parte izquierda.

6.2.4. Sujeción para móviles

Cuando ya se tiene la estructura es necesario sujetar el dispositivo de medida de alguna forma. Hay ciertos requerimientos para esta sujeción:

- Debe sujetar firmemente el dispositivo para transmitir bien las vibraciones y que no se amortigüen.
- La sujeción debe de estar cerca de los rodamientos ya que es ahí donde apoya el eje y la vibración es más fuerte.
- Se debe poder anclar y desanclar de una forma relativamente fácil.
- Su fabricación debe de ser sencilla.
- Se debe poder acceder a la pantalla del dispositivo para poder iniciar y guardar nuevas mediciones.

Con estos requerimientos se descarta un soporte que sujete el dispositivo por debajo de la base ya que tiene un acceso complicado además de no poder darle al botón de "Iniciar Medición" desde esa posición. El eje apoya en los dos rodamientos por lo que el soporte debe de estar cerca o justo en esa posición para maximizar la fuerza de las vibraciones que este vayan a recibir.

Se utilizará de nuevo la impresión 3D para la fabricación, se utiliza este método debido a que en poco tiempo se puede pasar del diseño por ordenador a la pieza lista además de esta pieza será personalizada para sujetar dispositivos móviles en este banco de pruebas especifico. El diseño va condicionado por tanto por el método de fabricación y por la dirección de las capas de la impresión.

Para sujetar el soporte se barajó la idea de sujetarlo aprovechando los tornillos que sujetan los rodamientos pero no hay mucho sitio donde se pueda apoyar la pieza de forma firma además de que tendría que desmontar el rodamiento cada vez que se quisiese mover el soporte. Finalmente se opta por la opción de dos brazos que abrazan al soporte de aluminio y el dispositivo móvil. Estos brazos acaban en una parte roscada para sujetar la pieza o el móvil. Aprovechando que el material para imprimir es algo flexible, se puede hacer un buen apriete y dejar la pieza fija con el soporte de aluminio.

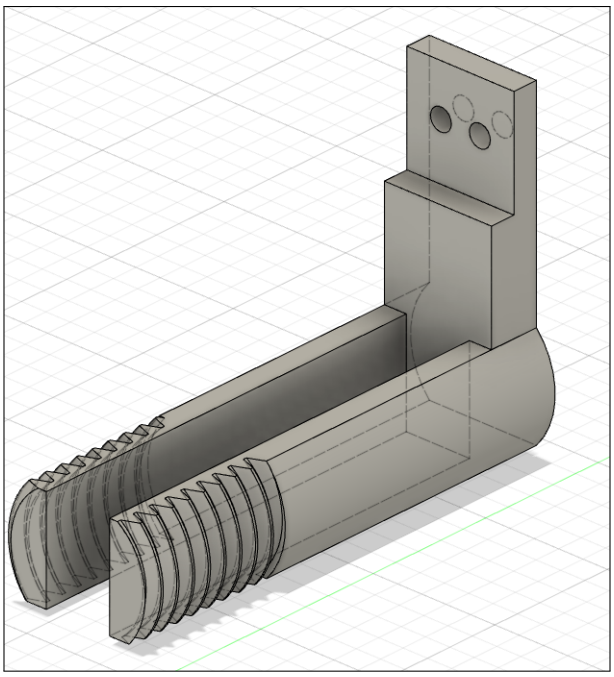
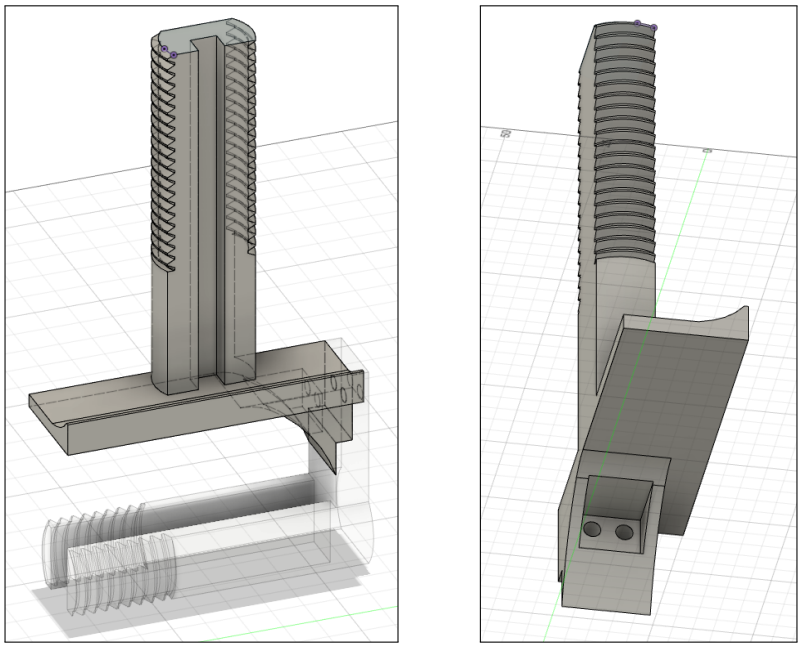


Fig. 6.19. Pieza inferior de la sujeción

La rosca tiene que estar abierta y no puede estar completa al tener que introducir esta pieza en el soporte como se puede ver en la figura 6.19. Para ahorro de material y cuestiones de diseño, se corta los brazos roscados por la parte superior e inferior. Así tiene una buena superficie en la parte inferior para que la pieza tenga buena adherencia en la impresión además de que la rosca, que es la parte de la impresión más crítica, tiene una mejor dirección de impresión para que salga con la mayor precisión geométrica posible.

Para la sujeción del dispositivo móvil se pretende usar el mismo método de una rosca impresa que pueda sujetar el dispositivo firmemente. Como el móvil tiene que estar situado perpendicular al eje para que no pueda chocar con ningún elemento de este. Para que la posición de la rosca impresa sea la ideal se debe separar la estructura principal de la sujeción en dos partes ya que no coincide con la orientación de la rosca inferior.

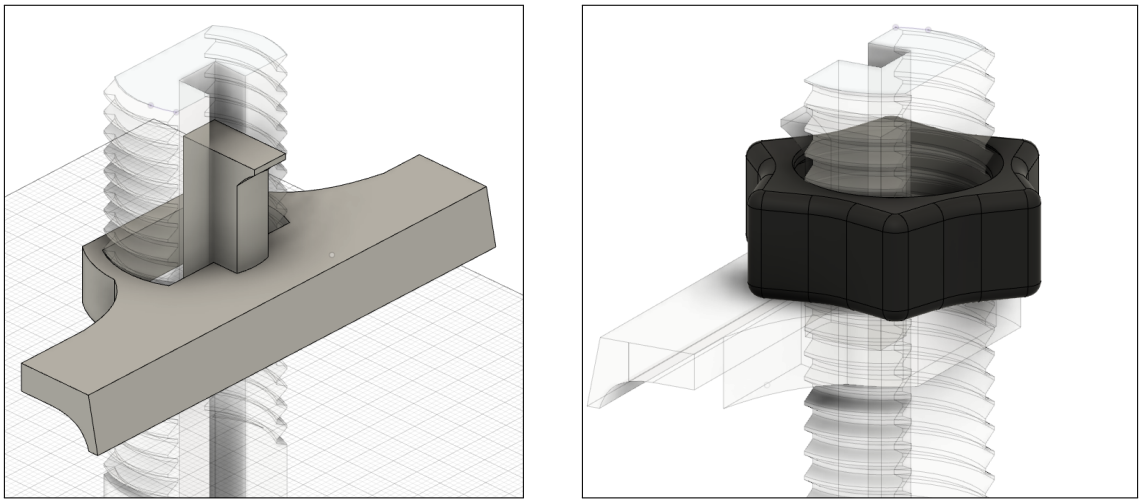
La **parte superior** que se puede ver en la figura 6.20 se unirá con la **parte inferior** con dos tornillos. En la parte de unión entre las dos piezas, se realiza un escalonado para que apoyen bien la parte superior en la inferior. La parte superior apoyará en el cojinete así teniendo otro punto de apoyo y de esa forma transmitir las vibraciones de forma más directa al dispositivo.



(a) Unión (b) Vista inferior

Fig. 6.20. Pieza superior de la sujeción

Para poder sujetar el dispositivo móvil será necesario que la tuerca de la rosca de la parte superior apriete sobre una superficie ancha para que se pueda colocar el móvil sujetado por los lados. Esta pieza, que se denominará **apriete**, tendrá un carril en la parte superior de la estructura para que pueda deslizarse sin perder la posición. Además para no poder salirse de este carril esta pieza tiene un aro que envuelve la rosca. Por ultimo esta pieza integra una parte que sujeta la rosca para se mueva en conjunto con esta y así que el subir o bajar esta pieza al girar la rosca.

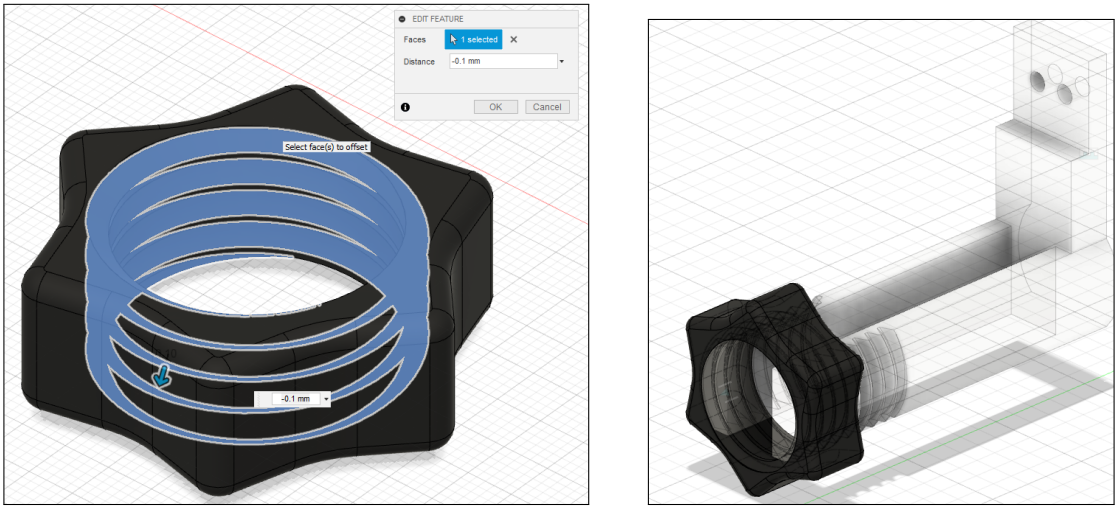


(a) Canal del apriete (b) Tuerca apriete

Fig. 6.21. Pieza de apriete

Por último falta el diseño de las **roscas**, en el que se tiene que están normalizadas. El programa Fusion 360 permite que las roscas estén modeladas así que se pueden utilizar para una rosa impresa. En la parte roscada de la estructura la rosca es de métrica 30 o M30 así que las tuercas tendrán este mismo roscado para que encajen. Para que se pueda roscar a mano, estar tuercas tienen forma de pentágono en su parte exterior para conseguir más agarre.

El PLA tras ser extruído a alta temperatura y enfriarse se produce una reducción en volumen por lo que aunque la rosca de la tuerca sea M30, tiene que haber una holgura para que ambas piezas encajen por lo que se añade un desplazamiento de la rosca interior de 0,1 mm.



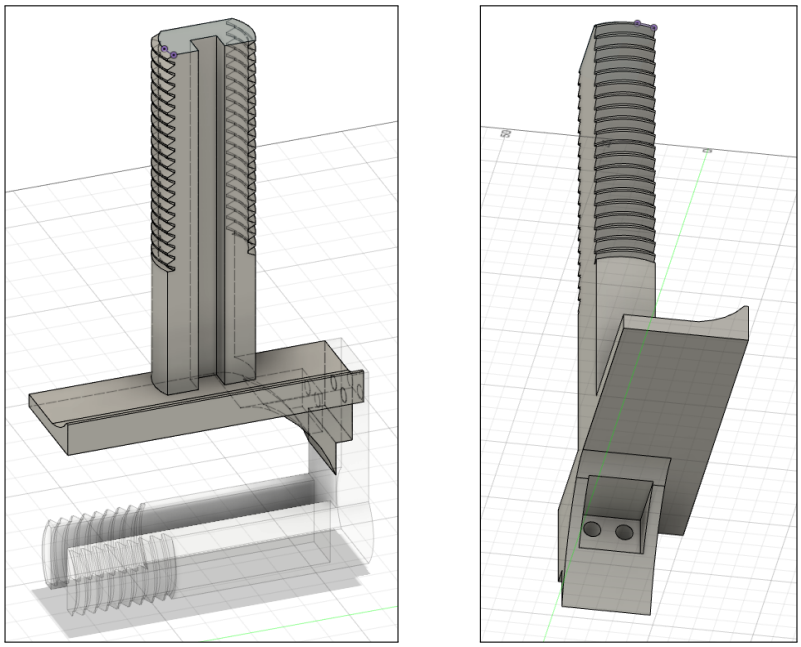
(a) Holgura

(b) Parte inferior

Fig. 6.22. Tuercas de la sujeción

El programa de modelado permite que el diseño de cada pieza se realice dentro del mismo proyecto en el que están incluidas el resto de piezas del banco de pruebas como se puede ver en la figura 6.23. Esto especialmente útil porque se puede diseñar una pieza teniendo en cuenta que sus dimensiones serán las correctas y que todo encajará. Este es otro de los motivos por los que se utiliza este programa.

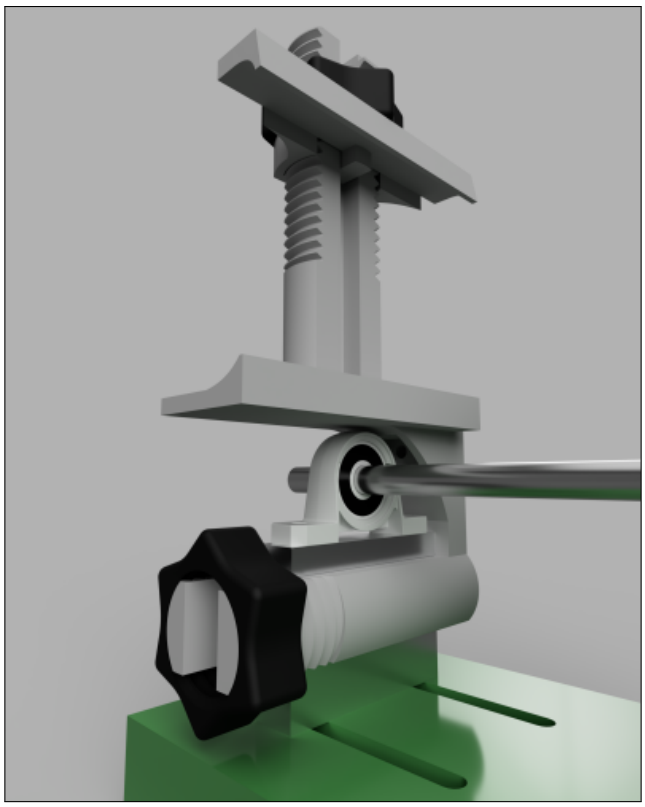
En la figura 6.23 (b se puede comprobar que la pieza superior de la sujeción va a apoyar en la parte superior del rodamiento como se buscaba. Y además este apoyo de hace a la altura justa para que la rosca inferior no quede demasiado abajo ni arriba y tenga espacio para girar.



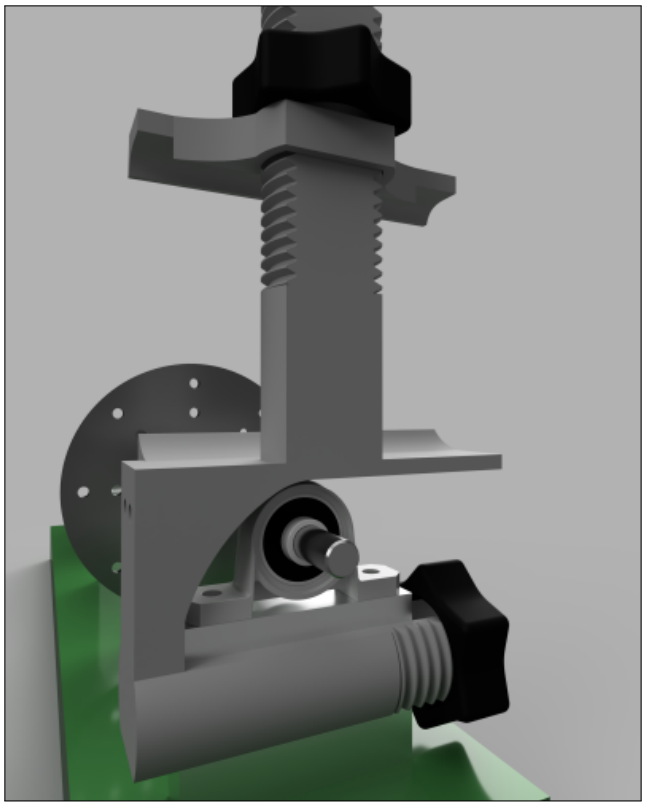
(a) Completo (b) Vista lateral

Fig. 6.23. Sujeción completa en Fusion 360

En la figura 6.24 se puede apreciar como quedaría la pieza en su posición final mediante un renderizado. El mismo programa, Fusion 360, tiene un módulo para realizar estas imágenes simulando la iluminación para obtener una imagen realista.



(a) Vista frontal



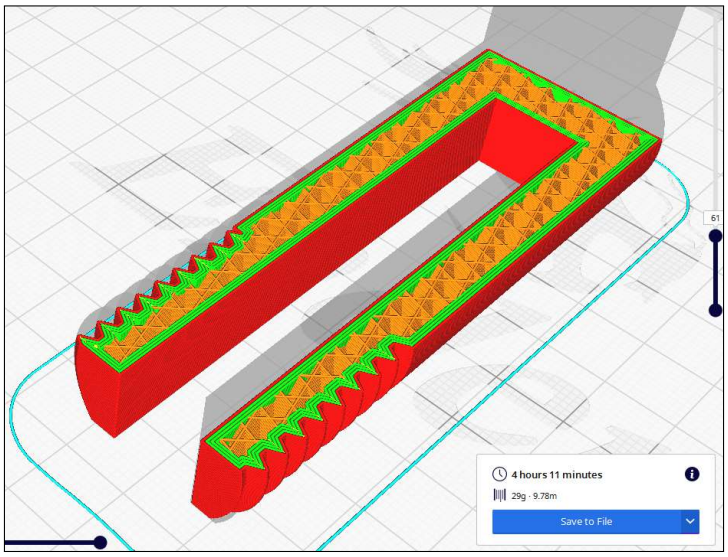
(b) Vista trasera

Fig. 6.24. Renderizado de la sujeción en Fusion 360

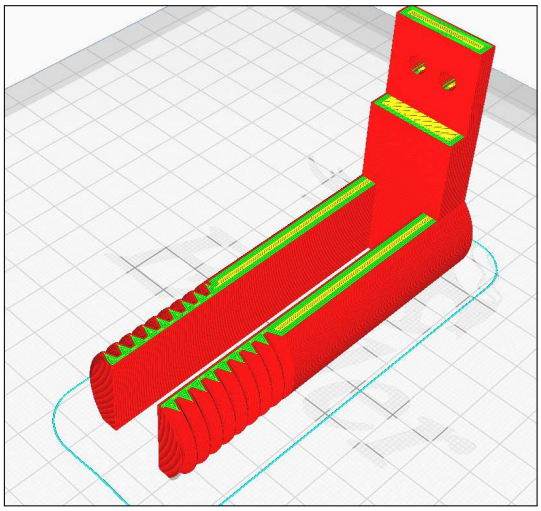
Impresión

Para la impresión de todas las piezas menos las roscas se utiliza PLA del fabricante Sakata, siendo las propiedades de este material son muy similares a el resto de PLA. Este fabricante español utiliza virutas de plástico 850 Ingeo que funden ellos y fabrican el filamento de plástico al espesor requerido. La ficha técnica de este material se puede encontrar para descargar en su tienda en línea [27].

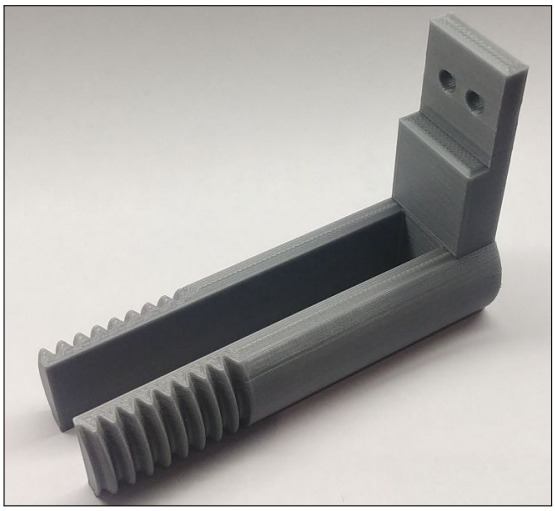
Este material tiene mejores propiedades que otros PLA aunque en este caso no es necesario que sea muy resistente ya que las piezas no soportarán mas fuerzas que el peso del dispositivo.



(a) Capa horizontal



(b) Completo



(c) Impreso

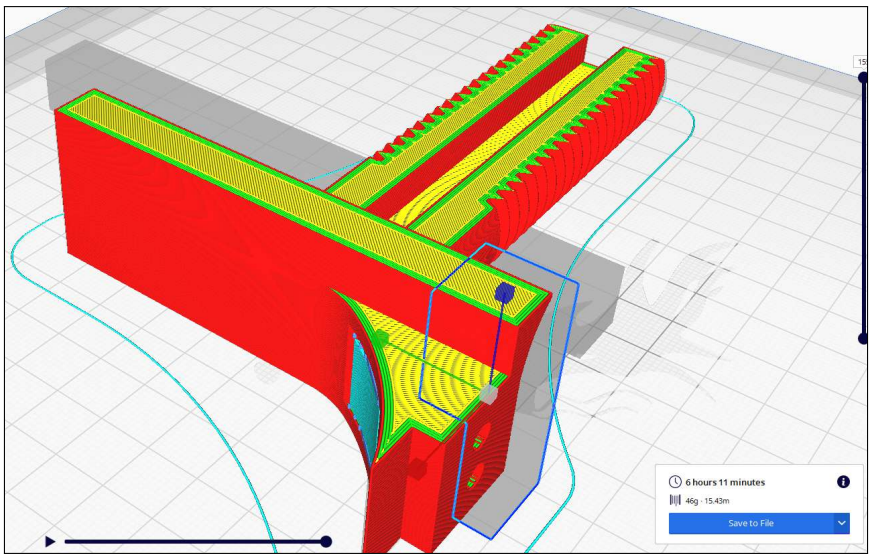
Fig. 6.25. Pieza inferior de la sujeción en Cura

El programa para la impresión de todas las piezas será nuevamente Cura. Aunque estas piezas no estén sujetas a esfuerzos, sí que deben ser rígidas para transmitir

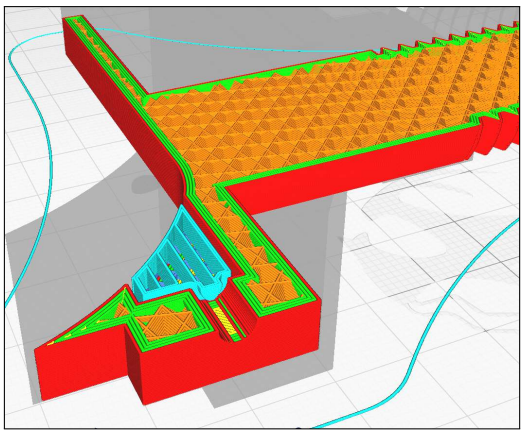
bien las vibraciones por lo que utilizan cuatro perímetros para darle más rigidez. Un perímetro es cada vuelta que da la cabeza de la impresora 3D a el borde de la pieza por lo que con cuatro perímetros y la punta por donde sale el plástico siendo de 0,4 mm, se obtiene un espesor de 1,6 mm.

En el diseño de la **pieza inferior** ya se tiene en cuenta la orientación para que la rosca salga de la forma más correcta ya que es la parte más sensible a una impresión no precisa. La precisión dimensional de la impresora es mayor en las direcciones del plano de impresión que verticalmente, ya que tiene una altura de capa de 0,2 mm. De este modo interesa que el dibujo de la rosca se haga en esta dirección.

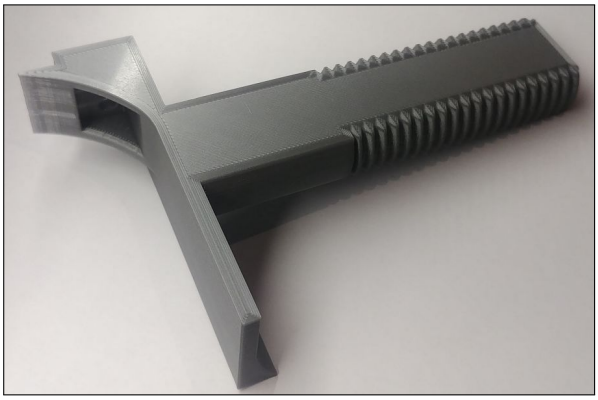
Por la posición en la que está colocada la pieza esta además no necesita ningún soporte y los agujeros tampoco necesitan de soportes. Esta pieza tarda en su impresión 4 horas y utiliza 29 gramos de material.



(a) Completo



(b) Sección soporte



(c) Impreso

Fig. 6.26. Pieza superior de la sujeción en Cura

La orientación de la **pieza superior** también viene dado en el diseño para que

rosca tenga la mejor dirección de capas. Además esta colocada para utilizar el mínimo número de soportes, salvando la zona de la figura 6.26 (b donde es necesario realizar esa entrada en la pieza para colocar los tornillos que unen las dos piezas principales de la estructura. Esta pieza tarda 6 horas en su impresión y se utilizan 46 gramos de material.

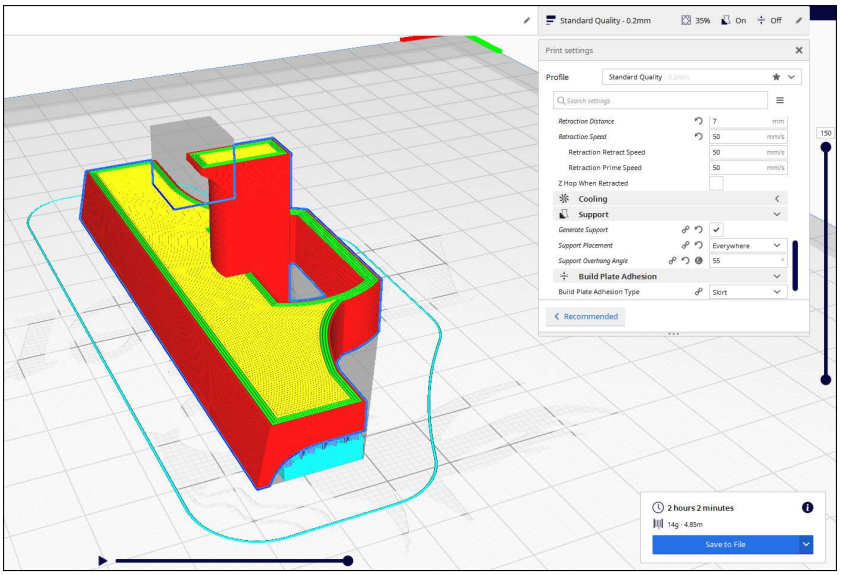
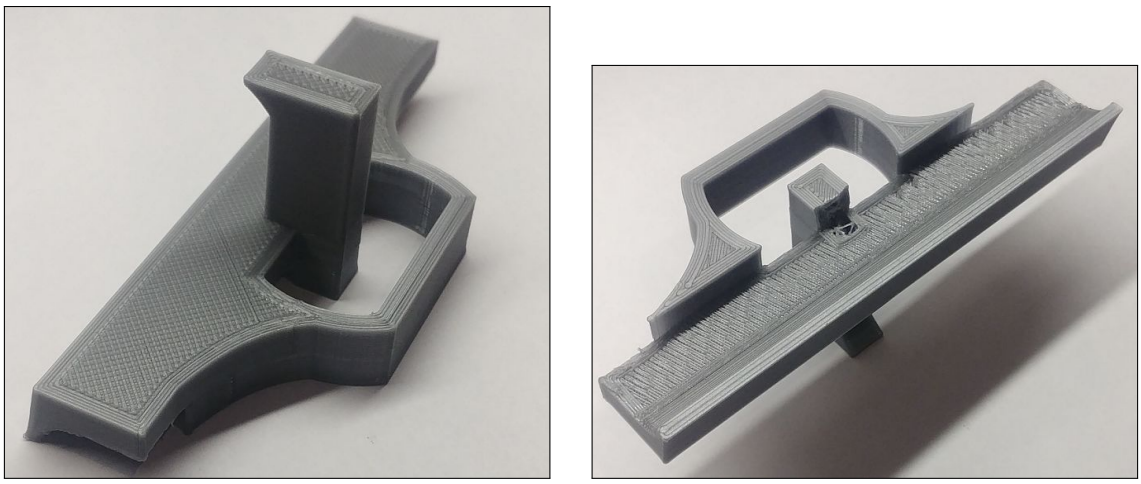


Fig. 6.27. Pieza de apriete de la sujeción en Cura

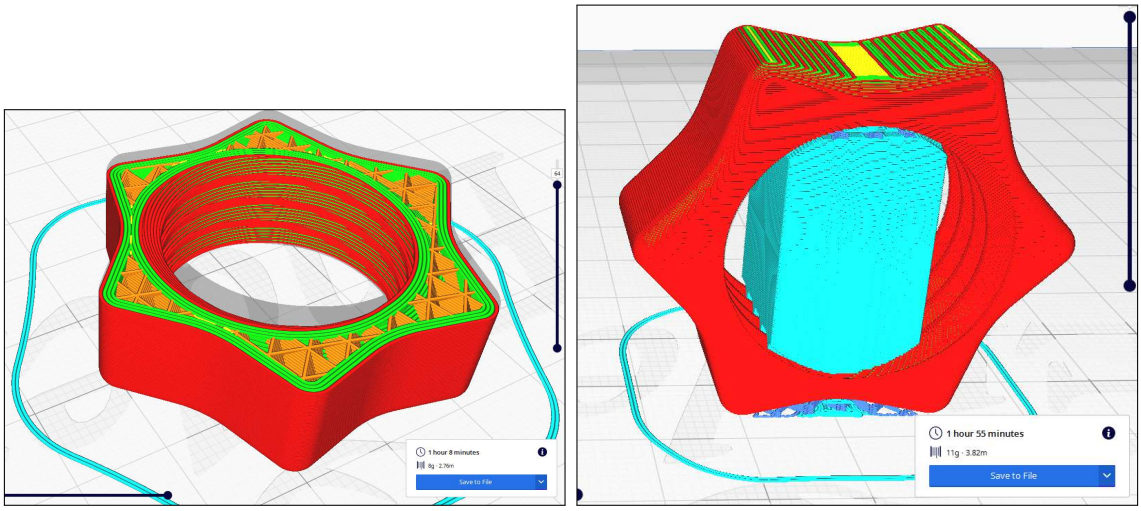
La pieza de **apriete** para sujetar el dispositivo por arriba sí que necesita soportes al haber una superficie plana grande sin apoyos. El resto de pendientes de la pieza están hechas con una pendiente que no necesita soportes y es suave para sujetar el dispositivo en la parte plana y arriba para sujetar la tuerca que se moverán en conjunto con esta pieza de apriete.

En la figura 6.28 se puede ver las dos partes de esta pieza y en la figura (b cómo queda la parte en la apoyará uno de los lados del dispositivo móvil tras quitar el soporte.



(a) Horizontal (b) Vertical

Fig. 6.28. Pieza del apriete impresa



(a) Horizontal (b) Vertical

Fig. 6.29. Tuercas de la sujeción en Cura

Para comprobar si la orientación horizontal o vertical es la más idónea para imprimir las tuerca se imprime una de cada tipo. La que se imprime en vertical deberá llevar soportes en su interior lo que puede afectar en la forma final de la rosca. Al ser una rosca entera, en teoría la posición con la que saldría mejor definida la rosca interior es la horizontal. Pero puede haber fallos si en la rosca hay pendientes muy pronunciadas y se expulse plástico en una zona con poco apoyo. La impresión final resultó correcta como se puede ver en la rosca izquierda de la figura 6.30.

Inicialmente encajan ambas en las otras piezas con muy poca holgura pero tras moverlas un poco funcionan de forma perfecta. Si no se hubiese hecho en la parte de diseño que la tuerca tuviese algo de más holgura en el roscado, estas no entrarían en

su ubicación final. Así que ambos diseños funcionan y se utilizarán pero si se tuviese que repetir se imprimiría el diseño horizontal por ahorrar material.

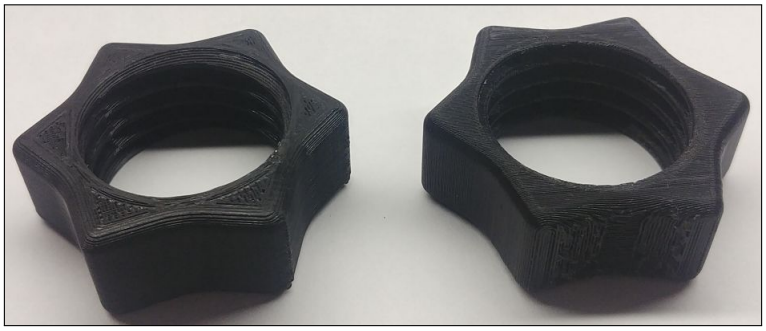
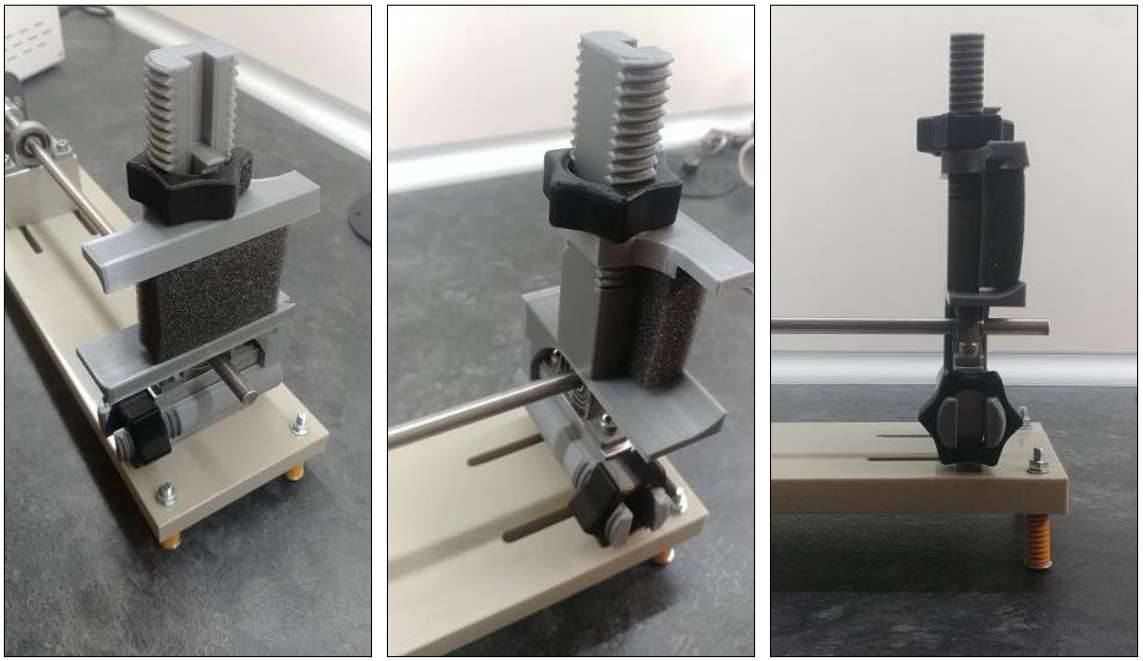


Fig. 6.30. Tuercas impresas



(a) Completo (b) Vista lateral (c) Vista lateral

Fig. 6.31. Ejemplo de sujeción instalada

En la figura 6.31 se puede ver cómo iría colocado un dispositivo en este soporte, sustituyendo el dispositivo por una esponja. La parte más estrecha de la pantalla es la que se introduce en el hueco antes de apretar la tuerca superior.

Tal y como se podía ver en el programa de diseño, la sujeción apoya en el soporte justo encima del rodamiento y la rosca inferior queda a la altura diseñada para que la rosca pueda girar y la base no impida el giro y pueda fijarse la sujeción al soporte de aluminio.

7. RESULTADOS

7.1. Medidas acelerómetro activity AccData

Una vez realizada la programación que hace que podamos acceder a los datos del acelerómetro se realizaron pruebas para saber si funciona y si la tasa de refresco calculada tiene un valor esperado. En primer lugar se prueba la tasa manual en el que al registrar el acelerómetro se le incluye el *delay* de las mediciones como un número cuando se registra.

En las pruebas que se realizaron en un dispositivo móvil *LG G6* tanto si se marcaba por ejemplo 80 Hz, 100 Hz o 150 Hz el dispositivo acababa devolviendo un valor aproximado de 100 Hz midiéndolo de la forma que se hace en el código 5.3. De esta forma se confirma la teoría de que el valor introducido es un valor recomendado y el sistema no necesariamente da los valores con de tasa introducida. Por este motivo se descartó la forma manual como método de elección de la tasa de muestreo.

Este problema con gran probabilidad es una cuestión de *software* debido a que los acelerómetros en los dispositivos móviles suelen tener otros usos menos exigentes y por lo tanto el sistema no está preparado para exigirle una tasa manual aunque el sensor pudiese realizarlo. Cuando se elige un método de los predeterminados por el sistema sí que se obtiene una medida de la tasa de una forma sostenida. Por lo tanto queda fuera del alcance del desarrollador modificar los valores predeterminados.

TABLA 7.1. MEDICIONES LG G6 SIN GRÁFICA

Tasa Muestreo [Hz] LG G6 Sin gráfica activa									
Normal	5,00	4,92	5,01	4,99	4,88	5,02	4,58	4,98	4,99
UI	24,80	24,25	24,78	24,80	24,83	24,78	24,93	24,78	24,85
Game	49,55	50,10	49,70	49,55	49,85	49,46	49,55	50,25	49,80
Fastest	396,60	397,60	396,21	396,70	398,00	399,00	397,60	395,60	396,21

TABLA 7.2. MEDICIONES LG G6 CON GRÁFICA

Tasa Muestreo [Hz] LG G6 Con gráfica activa									
Normal	4,41	4,95	4,00	4,91	5,00	4,98	4,99	4,05	5,03
UI	24,40	24,85	25,8	24,22	24,81	25,47	24,78	24,20	24,75
Game	50,00	49,51	49,10	50,10	47,71	51,54	49,65	48,71	50,60
Fastest	387,20	403,00	396,05	390,30	403,00	396,60	403,00	395,60	390,08

En las tablas 7.1 y 7.2 se muestran los primeros 9 valores de valores de una muestra de 30 valores en total del resultado de tomar los valores calculados como en el código 5.3 al seleccionar cada una de las tasas de muestreo que tiene el sistema Android, con la gráfica activa y sin ella.

De esta forma se puede comprobar si la gráfica, como otro proceso del dispositivo que consume recursos, tiene un efecto en estas medidas. También se hace esto para saber si los datos son lo suficientemente consistentes para poder considerarlos de modo que se puedan utilizar para su análisis ya haya el menor error posible.

A simple vista se puede ver que la mayoría de los valores dan resultados menores al esperado, esperando 5, 25, 50 y 400 Hz respectivamente.

TABLA 7.3. ESTADÍSTICAS MEDICIONES LG G6

	Muestras LG G6 Con gráfica [Hz]				Muestras LG G6 Con gráfica [Hz]			
	Normal	**UI**	**Game**	**Fastest**	**Normal**	**UI**	**Game**	**Fastest**
Media	4,967	24,82	49,749	397,57	4,89	24,816	49,562	396,41
Varianza	0,019	0,032	0,039	0,943	0,0754	0,257	0,704	36,585
Desviación	0,091	0,178	0,174	0,971	0,275	0,507	0,839	6,163

En la tabla 7.3 se pueden ver datos estadísticos de las mediciones de tasa de refresco realizadas. El objetivo es comprobar su consistencia y los valores obtenidos para ver si esas mediciones son consistentes y por lo tanto válidas. En primer lugar se quería comparar los resultados con la gráfica y sin la gráfica para saber si tenía efecto y se puede afirmar que sí que lo tiene.

Con la desviación típica calculada que tiene en cuenta cuánto se desvían los datos de la media para poder afirmar que unos datos son más consistentes que otros, se puede ver que cuando la gráfica está funcionando los datos son más dispersos y por tanto menos útiles. Esto se puede explicar por el consumo de mayores recursos del dispositivo al tener el código que calcular los datos al mismo tiempo que escribe los datos en la pantalla. Puede que sea un tiempo pequeño pero tiene un efecto al ser todas las desviaciones mayores en el caso de la gráfica en funcionamiento.

7.1.1. Otros dispositivos

Con el objetivo de conocer cómo se distribuyen las capacidades de los acelerómetros de distintos dispositivos se hace un estudio con dispositivos de distintos fabricantes y gamas de precio para ver sus capacidades. Los datos de las gráficas inferiores se obtienen mediante la pantalla "*Test*" de la aplicación y los resultados son los siguientes:

TABLA 7.4. XIAOMI MI A2 LITE (JULIO 2018)

Ver. Android	NORMAL(Hz)	UI(Hz)	GAME(Hz)	FASTEST(Hz)
8.1 Actual 10	≈ 17,34	≈ 17,31	≈ 51,9	≈ 208
Propiedades Acelerómetro. Fabricante: STMicroelectronics				
Nombre	Versión	Resolución	Rango Máximo	Consumo
LSM6DS3	1	0,0023956 m/s^2	78,4532 m/s^2	0,15 mA

TABLA 7.5. ALCATEL POP3 5015D (Q4 2015)

Ver. Android	NORMAL(Hz)	UI(Hz)	GAME(Hz)	FASTEST(Hz)
5.1	≈ 16,7	≈ 16,7	≈ 75	≈ 150
Propiedades Acelerómetro. Fabricante: MTK				
Nombre	Versión	Resolución	Rango Máximo	Consumo
ACCELER	3	0,003906 m/s^2	32,0 m/s^2	0,13 mA

TABLA 7.6. ALCATEL 1 5033D (JULIO 2018)

Ver. Android	NORMAL(Hz)	UI(Hz)	GAME(Hz)	FASTEST(Hz)
8.1	≈ 15,18	≈ 15,2	≈ 50	≈ 100
Propiedades Acelerómetro. Fabricante: MTK				
Nombre	Versión	Resolución	Rango Máximo	Consumo
ACCELER	1	0,0012 m/s^2	39,2266 m/s^2	0,001 mA

TABLA 7.7. SAMSUNG A40 (MARZO 2019)

Ver. Android	NORMAL(Hz)	UI(Hz)	GAME(Hz)	FASTEST(Hz)
9.0	≈ 50	≈ 50	≈ 50	≈ 100
Propiedades Acelerómetro. Fabricante: STM				
Nombre	Versión	Resolución	Rango Máximo	Consumo
LSDM6DSL	1	0,001197 m/s^2	39,2266 m/s^2	0,13 mA

TABLA 7.8. ZTE NUBIA (2018)

Ver. Android	NORMAL(Hz)	UI(Hz)	GAME(Hz)	FASTEST(Hz)
9.0	≈ 5	≈ 25	≈ 50	≈ 100
Propiedades Acelerómetro. Fabricante: MTK				
Nombre	Versión	Resolución	Rango Máximo	Consumo
ACCELER	3	0,0039 m/s^2	32 m/s^2	0,13 mA

TABLA 7.9. XIAOMI MI 9 (FEBRERO 2019)

Ver. Android	NORMAL(Hz)	UI(Hz)	GAME(Hz)	FASTEST(Hz)
9.0	≈ 5	≈ 15	≈ 50	≈ 400
Propiedades Acelerómetro. Fabricante: Bosch				
Nombre	Versión	Resolución	Rango Máximo	Consumo
BMI160	50528522	0,0023928 m/s^2	78,45 m/s^2	0,18 mA

TABLA 7.10. SAMSUNG NOTE 10 (AGOSTO 2019)

Ver. Android	NORMAL(Hz)	UI(Hz)	GAME(Hz)	FASTEST(Hz)
10	≈ 5	≈ 15	≈ 50	≈ 500
Propiedades Acelerómetro. Fabricante: STM				
Nombre	Versión	Resolución	Rango Máximo	Consumo
LSM6DSO	15932	0,0023942 m/s^2	78,45 m/s^2	0,25 mA

TABLA 7.11. XIAOMI REDMI NOTE PLUS 5 (FEBRERO 2018)

Ver. Android	NORMAL(Hz)	UI(Hz)	GAME(Hz)	FASTEST(Hz)
7.1.4 Actual 9	≈ 50	≈ 50	≈ 50	≈ 400
Propiedades Acelerómetro. Fabricante: Bosch				
Nombre	Versión	Resolución	Rango Máximo	Consumo
BMI120	2062701	0,0023956 m/s^2	156,9 m/s^2	0,18 mA

Para móviles más antiguos, con menos capacidad y de un precio menor de salida se ha observado que la menos potencia del dispositivo afecta a la precisión con la que se toman los datos y sobre todo el tiempo del sistema. Mediciones menos precisas causan que los resultados analizados no se correspondan a la realidad y si a eso se añade que estos dispositivos de gama media-baja montan acelerómetros de menor calidad, los resultados son aún menos precisos y con una tasa máxima de muestreo menos. Esta tasa máxima suele ser de 100 Hz en estos dispositivos.

Por el contrario en móviles de los últimos años (2018-2020) y sobre todo de gama alta montan un acelerómetro mejor, con más precisión y con una tasa de muestreo

máxima mayor. La tasa de muestreo máxima en estos dispositivos suele ser de 400 Hz y así pueden tomar mediciones en un mayor rango de casos.

Así también se pueden observar variaciones entre las mediciones de distintas *Apps*, haciéndose estas mediciones desde el mismo dispositivo y mismo acelerómetro. En este artículo [28] realizan un análisis más en profundidad de las diferencias que puede haber al escoger una aplicación u otra.

7.2. Pruebas en casos reales

7.2.1. Mediciones en máquinas

Muchas máquinas tienen partes rotativas entre sus componentes por lo que generan vibraciones a una determinada frecuencia de modo que son un buen sujeto de prueba para ver si desde un dispositivo móvil se pueden captar dichas vibraciones. A continuación se probará la aplicación en distintas máquinas de las que se conoce de forma aproximada la velocidad de rotación de alguno de sus elementos.

Todas las mediciones en casos reales se realizaron con un dispositivo "LG G6", capaz de obtener una tasa de muestreo máxima de 400 Hz y por la frecuencia de Nyquist, el rango para el que se puede realizar mediciones es de 0 a 200 Hz. Para comparar las frecuencias obtenidas se utiliza un tacómetro que da el resultado en revoluciones por minuto que se convertirán a hercios para mostrar en las distintas gráficas la frecuencia que se espera encontrar en los datos analizados.

7.2.1.1 Motor Audi A6

Para tomar medidas del motor de este coche, el dispositivo móvil se colocó encima de una tapa que cubre el motor con la pantalla hacia arriba y la parte inferior apuntando hacia un lateral. De esta forma la dirección "Y" sería la que toman el eje de las ruedas, la dirección "X" tomaría la dirección del coche y el eje "Z" hacia arriba. Como un motor dispone de varios elementos giratorios diferentes se producirá una combinación de todos ellos y solo se conoce la velocidad de ralentí del coche que es alrededor de 700 rpm según el tacómetro del coche.

Esos 700 RPM equivalen a 11,67 Hz por lo que se espera que esa sea una de las frecuencia principales al igual que múltiplos de esta frecuencia como 23,34 o 35 Hz pero también se esperan frecuencia de otros elementos. También estas vibraciones deben tener una amplitud suficiente para que el dispositivo sea capaz de detectarlas.

Importante es también valorar la diferencia entre las mediciones teniendo en cuenta la dirección del árbol de levas del motor se tendrán dos ejes perpendiculares a este y que debería tener mediciones similares.

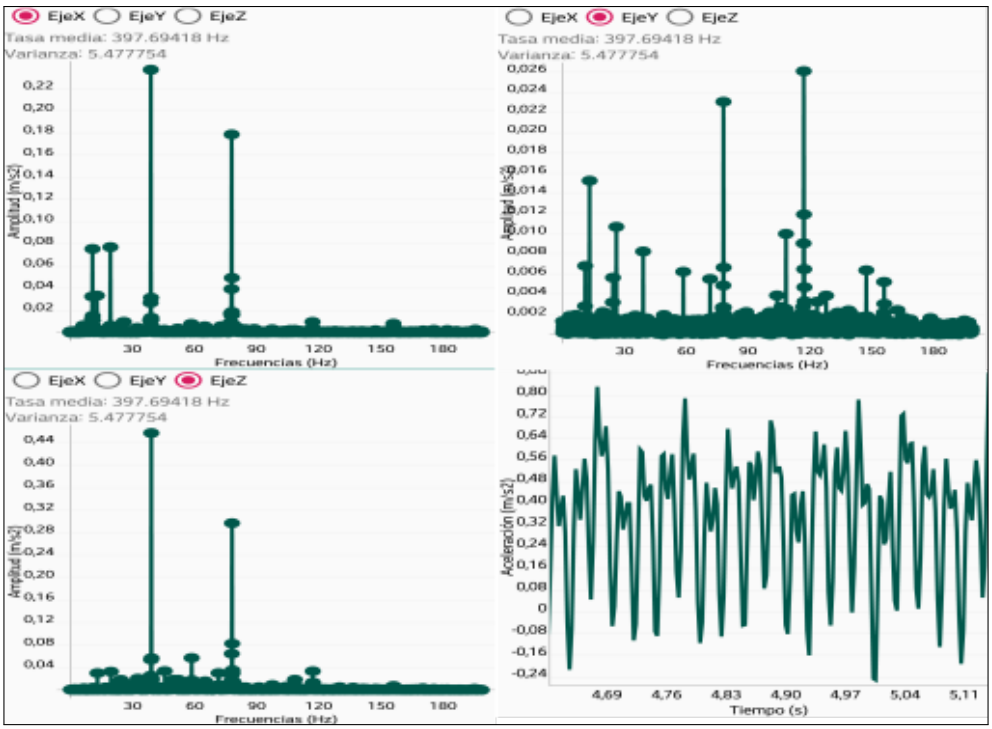


Fig. 7.1. Resultados del motor del coche

Se comprueba en la primera medida que se analiza dentro de la aplicación que la Transformada Rápida de Fourier se ha aplicado de forma correcta y señala las frecuencias más significativas. Para ello se realizará el mismo análisis en Matlab y como se puede ver en la figura 7.2, los resultados coinciden con los obtenidos en la aplicación. Matlab utiliza algoritmo para la transformada que está incluido en el propio entorno de programación de Matlab.

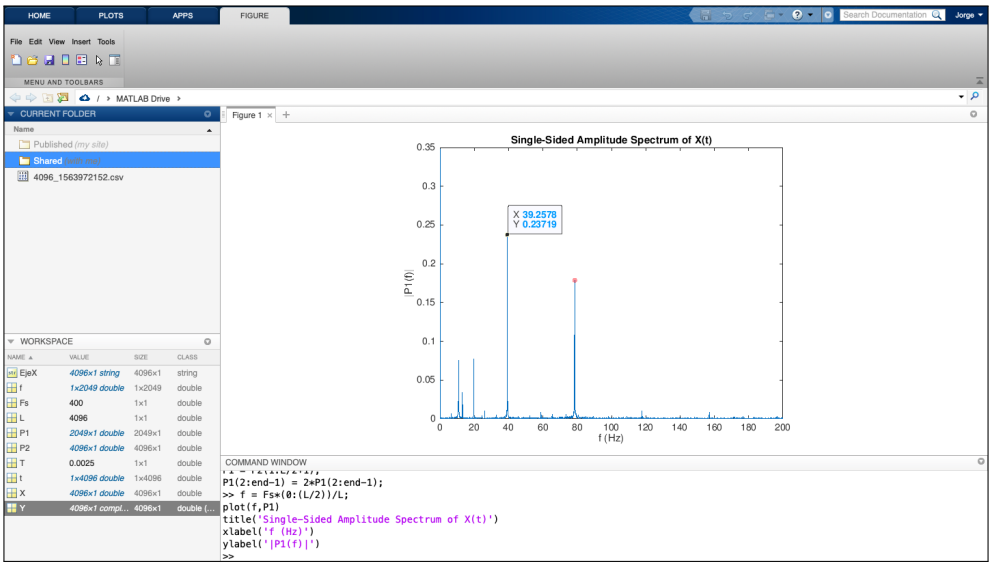


Fig. 7.2. Resultados del motor del coche en Matlab

TABLA 7.12. MEDICIONES DEL COCHE

Coche	Nº Muestras = 4096	Frec. esperada (Hz) = 11,67
Frecuencias Destacadas (Hz)		
Eje X	10,87 / 19,51 / 39,03 / 78,06	
Eje Y	13,01 / 78,06 / 117,09	
Eje Z	39,03 / 78,06	

Como en las muestras que tienen un elemento rotativo principal, se espera que dos de sus ejes que van perpendiculares al eje muestren resultados similares como se puede ver en los ejes "X" y "Z" en la figura 7.1. Por lo que se puede comprobar en la dirección en la que gira el cigüeñal del motor (eje "Y") la amplitud de las vibraciones es menor y por lo tanto los resultados son menos significativos ya que las frecuencias que destacan son muy débiles.

Las frecuencias destacadas en "X" y "Z" se pueden atribuir a todos los elementos con movimiento del motor. Estas vibraciones pueden ser provocadas por la geometría del soporte del motor, del propio motor o del recubrimiento de plástico del motor sobre el que se coloca el dispositivo para realizar las mediciones entre otros elementos.

Para un análisis más detallado hay que realizar un estudio más profundo con más pruebas para determinar qué parte del motor causa cada vibración, pero los resultados obtenidos verifican que el acelerómetro del dispositivo móvil es capaz de obtener algunas frecuencias aunque no se sepa a que elemento corresponden.

7.2.1.1 Torno Industrial

Se hacen varias mediciones en un torno modelo S 90/250-105 del fabricante *Pinacho*. Se coloca el dispositivo móvil sobre el protector de la pinza que sujeta las piezas. En la figura 7.3 este protector es la pieza blanca con asa que se sitúa a la derecha de la caja que contiene el motor y la caja de cambios para variar la velocidad de giro. Se seleccionan tres velocidades con la caja de cambios que se comprobarán con el tacómetro y se hará una medición para cada velocidad.

Fig. 7.3. Torno industrial

Se coloca el dispositivo con la pantalla hacia arriba para poder iniciar las mediciones y así la dirección del eje "X" va en la misma dirección que el eje del motor; el eje "Y" va perpendicular al eje del motor en el plano de la pantalla del dispositivo y el "Z" va hacia arriba. Se puede esperar que al ser una máquina rotativa los ejes "Y" y "Z" tengan mediciones similares.

Como nota: Las vibraciones producidas por el torno en el punto en el que va a medir son muy débiles al tacto, así que se comprobará la sensibilidad del acelerómetro del dispositivo si detecta esas vibraciones.

TABLA 7.13. VELOCIDADES DE LAS MEDICIONES DEL TORNO

	Rpm torno	Rpm Tacometro	Rpm Tacometro en Hz
Medida 1	510	541	9
Medida 2	960	986	16,43
Medida 3	1200	1233	20,55

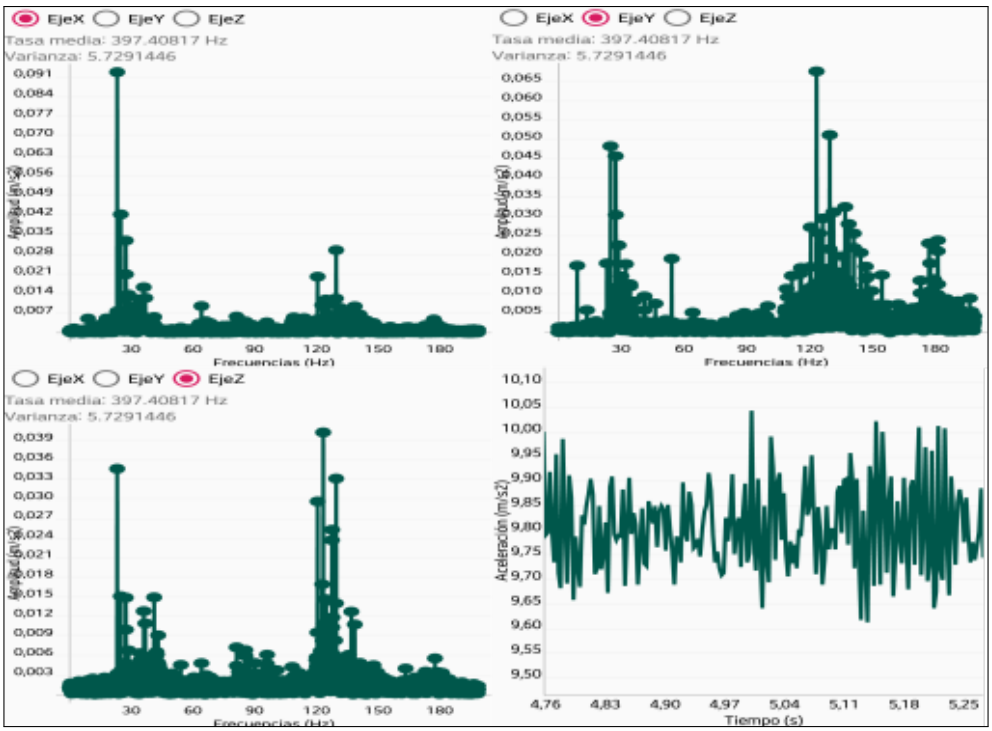


Fig. 7.4. Resultados Medida 1 del torno

TABLA 7.14. RESULTADOS MEDIDA1

Medida1	Nº Muestras = 4096	Frec. esperada (Hz) = 9
Frecuencias Destacadas (Hz)		
Eje X	23,19 / 129,62	
Eje Y	9,02 / 24,93 / 123,31	
Eje Z	23,18 / 123,31	

Con la primera medición con el motor girando a 541 rpm no se aprecian diferencias entre los tres ejes, ni una frecuencia destacada a la frecuencia prevista. Las frecuencias encontradas se pueden explicar por la falta de vibraciones del motor a estas velocidades, debido a un buen balanceo de este.

La repetición de unos resultados similares en los tres ejes pueden ser causados por las frecuencias propias del protector causadas por su geometría. Las vibraciones del motor no son lo suficiente potentes para ser captadas por el acelerómetro pero sí capta las vibraciones que produce el propio protector al no ser una pieza que esté sujeta firmemente al cuerpo del torno.

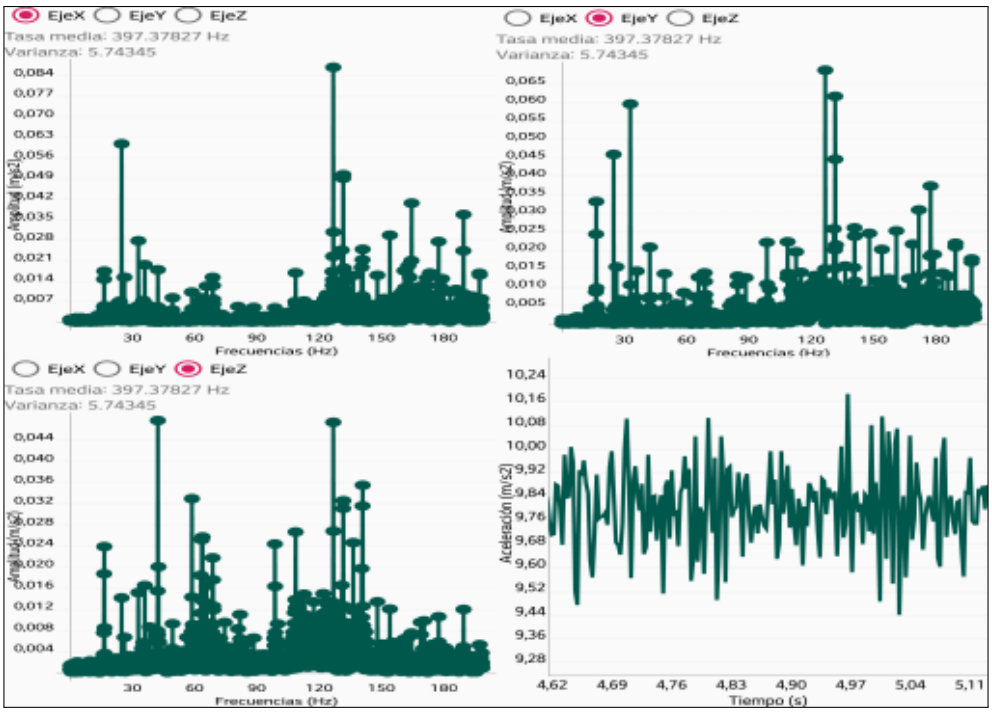


Fig. 7.5. Resultados Medida 2 del torno

TABLA 7.15. RESULTADOS MEDIDA2

Medida2	Nº Muestras = 4096	Frec. esperada (Hz) = 16,43
Frecuencias Destacadas (Hz)		
Eje X	24,83 / 126,8	
Eje Y	24,83 / 32,88 / 126,80	
Eje Z	42,29 / 126,80	

Subiendo la velocidad del motor en esta "Medida2" se obtienen resultados de frecuencia similares a la "Medida 1". La forma de las frecuencias significativas de las gráficas son muy similares solo que ligeramente en una frecuencia superior. Por ejemplo de 23,18 se pasa a 24,83 Hz, lo que indica que las vibraciones que capta el acelerómetro provienen de la protección sobre la que apoya el dispositivo y no el propio motor.

Se obtiene un pico a la frecuencia esperada de 16 Hz pero en amplitud está a en torno un cuarto de las frecuencias más destacadas. Aunque esta frecuencia sigue siendo importante ya que a estas velocidades de giro del motor, el acelerómetro ya es capaz de captar las vibraciones del motor.

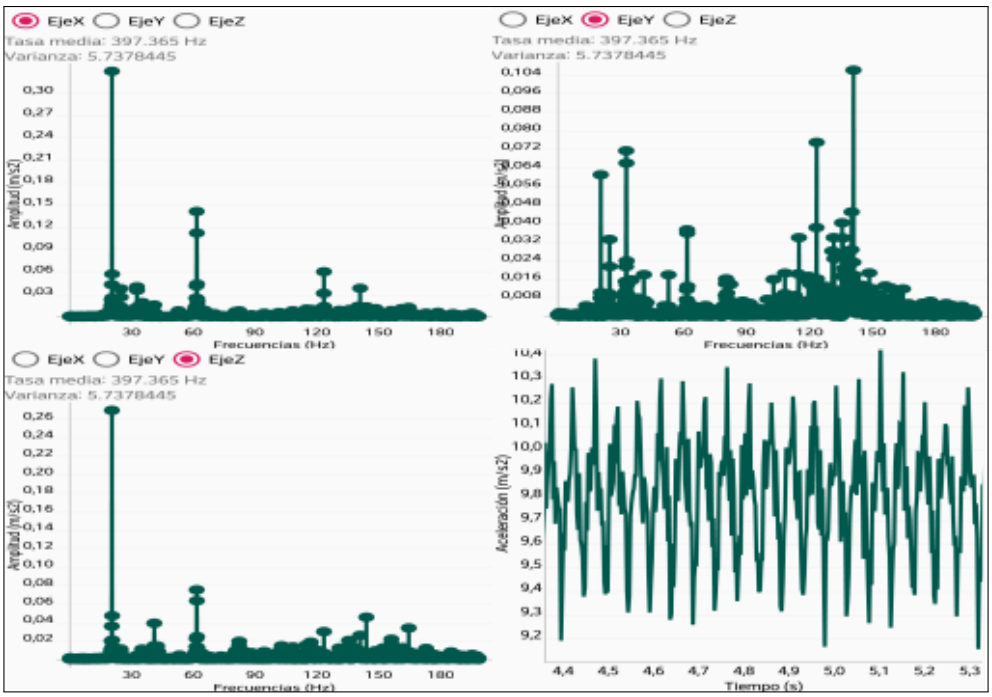


Fig. 7.6. Resultados Medida 3 del torno

TABLA 7.16. RESULTADOS MEDIDA3

Medida3	Nº Muestras = 4096	Frec. esperada (Hz) = 20,55
Frecuencias Destacadas (Hz)		
Eje X	20,566 / 61,70 / 123,49	
Eje Y	20,566 / 32,69 / 123,49 / 141,05	
Eje Z	20,566 / 61,70	

Con la velocidad más alta que permite el torno se ve una clara diferencia con las medidas anteriores a menos velocidad. En primer lugar, en la gráfica que muestra la aceleración frente al tiempo es mucho más regular por lo que se intuye que ahora las vibraciones del motor son las más destacadas por ser más fuertes. Así en los ejes "X" y "Z" se puede ver una frecuencia que destaca frente al resto que corresponde con la frecuencia esperada de giro (20,56 Hz).

7.2.2. Medidas generador de vibraciones

Con la estructura lista como aparece en la figura 7.7 se situará el dispositivo encima de la plataforma de madera que queda por debajo de la estructura. El eje "Z" es el que va paralelo al eje en movimiento. A parte del eje principal, está el eje del motor que puede causar vibraciones a una frecuencia mayor que la medida al tener un par de engranajes que reducen la velocidad de giro.

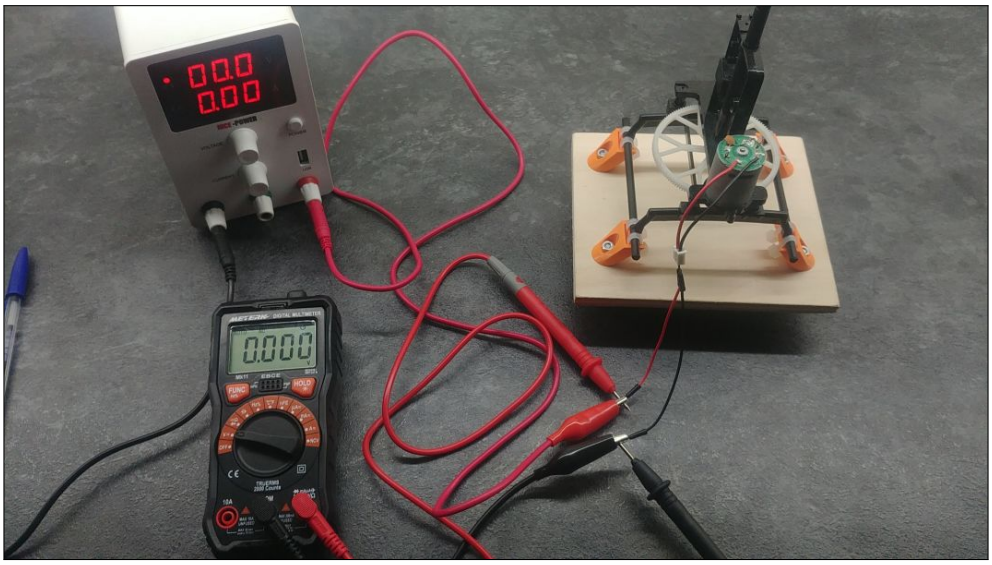


Fig. 7.7. Disposición del generador para las mediciones

Se realizan una series de medidas a diferentes velocidades de giro del motor en dos configuraciones: con el disco incorporado en la parte superior y sin este.

TABLA 7.17. VELOCIDADES DE LAS MEDICIONES DEL GENERADOR

	Velocidad (Rpm)	Velocidad (Hz)	Voltaje(V)	Corriente(A)
Gen1	1140	19	3,5	0,64
Gen2	1640	27,33	5,61	0,85
Gen3	2040	34	6,07	0,87

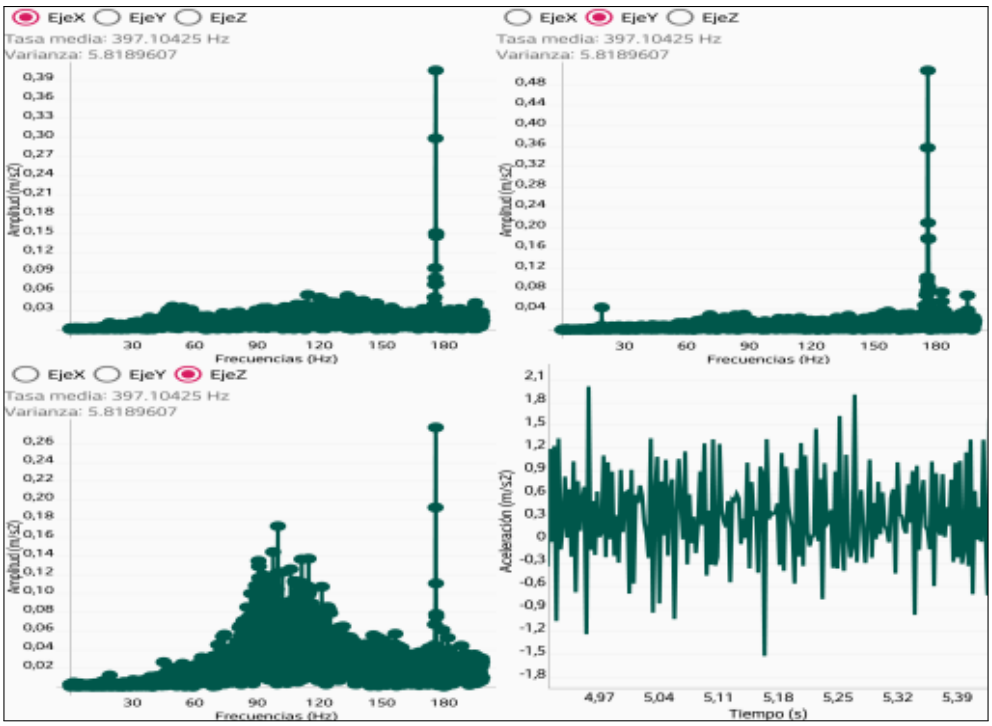


Fig. 7.8. Resultados Gen1

TABLA 7.18. RESULTADOS GEN1

Gen1	Nº Muestras = 4096	Frec. esperada (Hz) = 19
Frecuencias Destacadas (Hz)		
Eje X	176,05	
Eje Y	19,09 / 176,06	
Eje Z	100,05 / 176,06	

Se obtiene un pico en la frecuencia esperada en el eje "Y" pero de amplitud baja por lo que en estas condiciones la amplitud de las vibraciones generadas no llegan a la sensibilidad del acelerómetro. Sí que se capta una vibración de una frecuencia mayor y esto puede ser causado por la diferencia de velocidades entre el eje principal y el eje del motor debido a la rueda y el piñon que tiene la estructura, siendo el motor de giro más rápido.

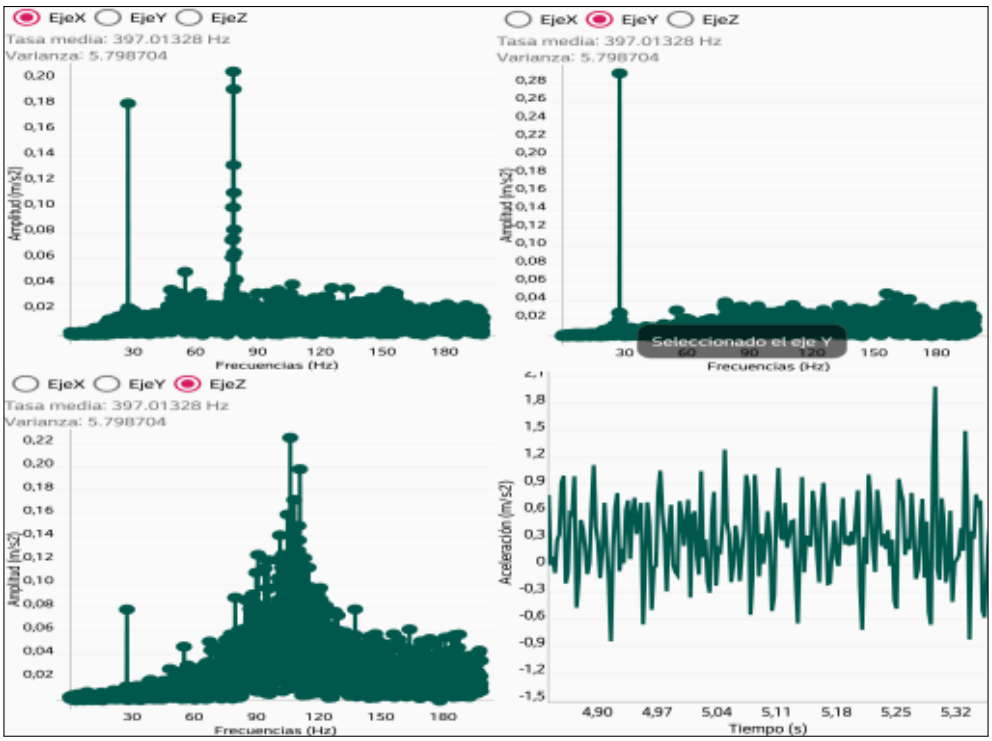


Fig. 7.9. Resultados Gen2

TABLA 7.19. RESULTADOS GEN2

Gen2	Nº Muestras = 4096	Frec. esperada (Hz) = 27,33
Frecuencias Destacadas (Hz)		
Eje X	27,527 / 78,22	
Eje Y	27,527	
Eje Z	27,527 / 106,32	

En esta muestra con una velocidad mayor ya se puede distinguir la velocidad de rotación del eje principal ya que este está muy desequilibrado y no necesita de unas velocidades de rotación muy altas para generar una vibración que sea capaz de detectar el acelerómetro.

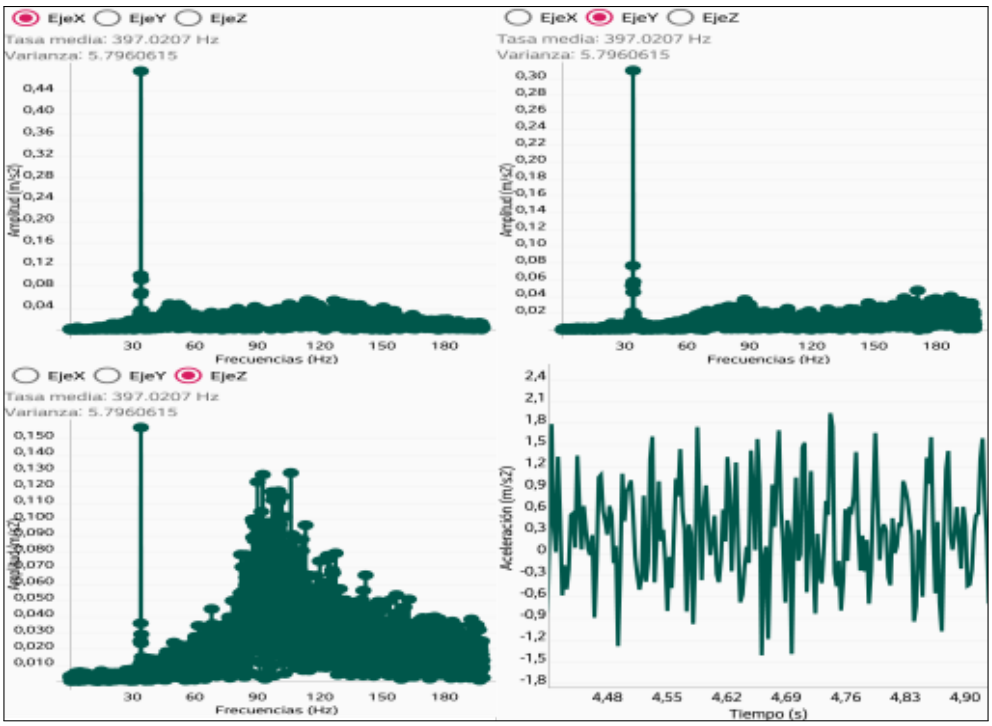


Fig. 7.10. Resultados Gen3

TABLA 7.20. RESULTADOS GEN3

Gen3	Nº Muestras = 4096	Frec. esperada (Hz) = 34
Frecuencias Destacadas (Hz)		
Eje X	34,02	
Eje Y	34,02	
Eje Z	34,02 / 106,13	

Para los resultados de "Gen3" se puede ver, que una mayor velocidad de giro hace que se acentúe la frecuencia de giro esperada y destaque entre el resto. Al no ser una estructura muy equilibrada inicialmente es normal que aparezcan otras vibraciones que no concuerden con lo esperado.

Resultados con el disco montado

Los siguientes resultados pertenecen a los medidos con la inserción del disco perforado en la ranura superior del eje principal. Por la baja tolerancia de diseño necesaria para el funcionamiento previo causa la aparición de una holgura en el eje. Con esta holgura, la inserción de pesos en el disco perforado se estima no necesaria ya que no se vería resultados claros dada las vibraciones fuertes causadas por la propia holgura del eje.

TABLA 7.21. VELOCIDADES DE LAS MEDICIONES DEL GENERADOR CON DISCO

	Velocidad (Rpm)	Velocidad (Hz)	Voltaje(V)	Corriente(A)
GenDisco1	1004	16,73	3,03	0,54
GenDisco2	1606	26,77	4,82	0,82
GenDisco3	2514	41,9	7,38	0,88

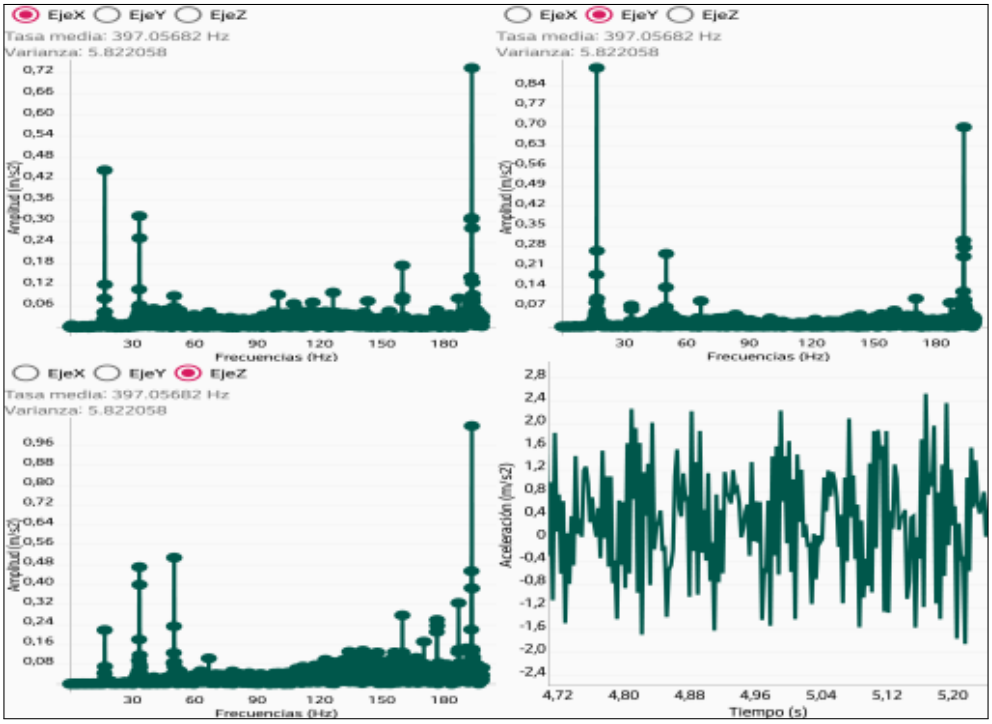


Fig. 7.11. Resultados GenDisco1

TABLA 7.22. RESULTADOS GENDISCO1

GenDisco1	Nº Muestras = 4096	Frec. esperada (Hz) = 16,73
Frecuencias Destacadas (Hz)		
Eje X	16,67 / 33.34 / 193,39	
Eje Y	16,67 / 50,11 / 193,39	
Eje Z	16,67 / 33,34 / 50,11 / 193,39	

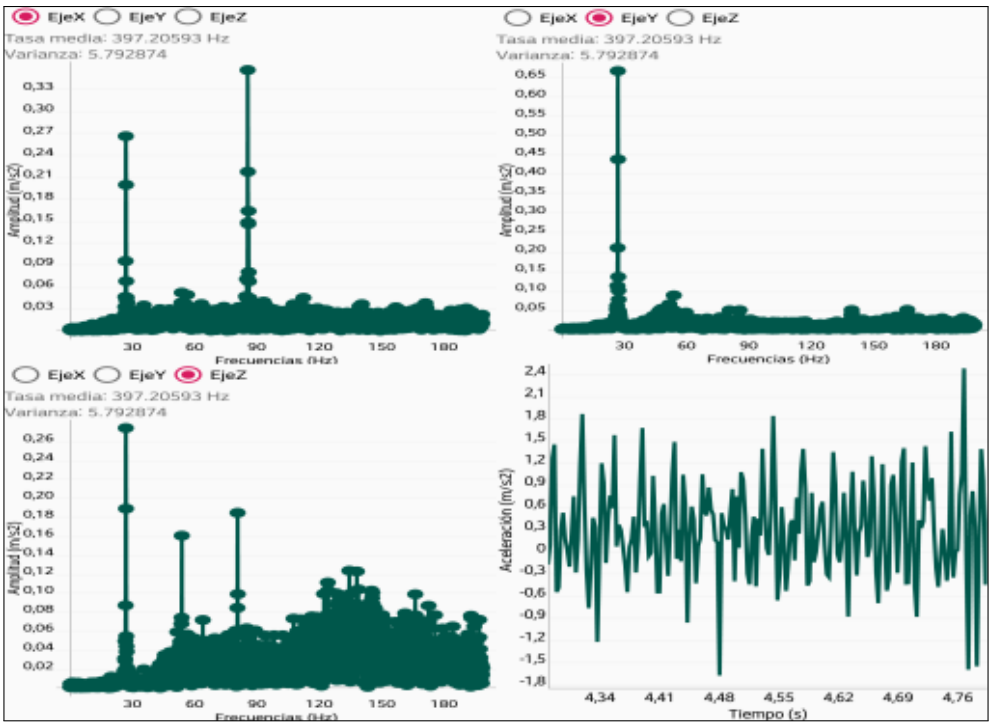


Fig. 7.12. Resultados GenDisco2

TABLA 7.23. RESULTADOS GENDISCO2

GenDisco2	Nº Muestras = 4096	Frec. esperada (Hz) = 26,77
Frecuencias Destacadas (Hz)		
Eje X	26,86 / 85,53	
Eje Y	26,86	
Eje Z	26,86 / 53,82 / 80,68	

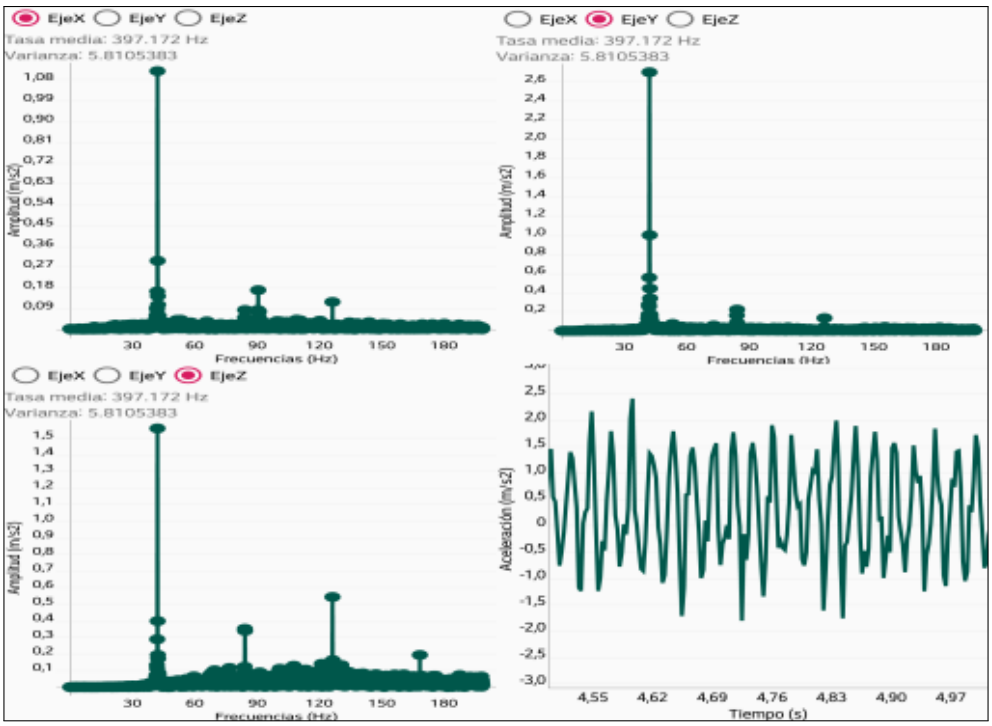


Fig. 7.13. Resultados GenDisco3

TABLA 7.24. RESULTADOS GENDISCO3

GenDisco3	Nº Muestras = 4096	Frec. esperada (Hz) = 41,9
Frecuencias Destacadas (Hz)		
Eje X	42,08 / 90,56 / 126,34	
Eje Y	42,08 / 84,26 / 126,34	
Eje Z	42,08 / 84,26 / 126,34	

Con el peso extra en la parte superior de eje principal, se produce un mayor desequilibrio que se traduce en una vibración mayor. Por este motivo, las pruebas realizadas con el disco insertado producen unos resultados en los que se puede distinguir la frecuencia de este eje a una velocidad de rotación menor.

Esta vibración, al ser mayor, se puede detectar con facilidad incluso en el eje "Z" que va paralelo a el eje principal de esta estructura. Realmente las máquinas que se puedan analizar en uso real no van a tener este tipo de desequilibrios ya que son grandes pero se puede ver la capacidad del acelerómetro del dispositivo de capturar estas vibraciones.

7.2.3. Medidas del banco de pruebas

Para realizar las mediciones en el banco de pruebas se utilizará la configuración que aparece en la figura 7.14 y a través de variar el voltaje de entrada a el motor se varía la velocidad de rotación de este. Esta velocidad se mide con el tacómetro y servirá para compararla con los resultados del análisis de frecuencias.

El soporte para el dispositivo se coloca en el soporte del eje más alejado del motor y la separación de los cojinetes es la configuración máxima del banco de pruebas.

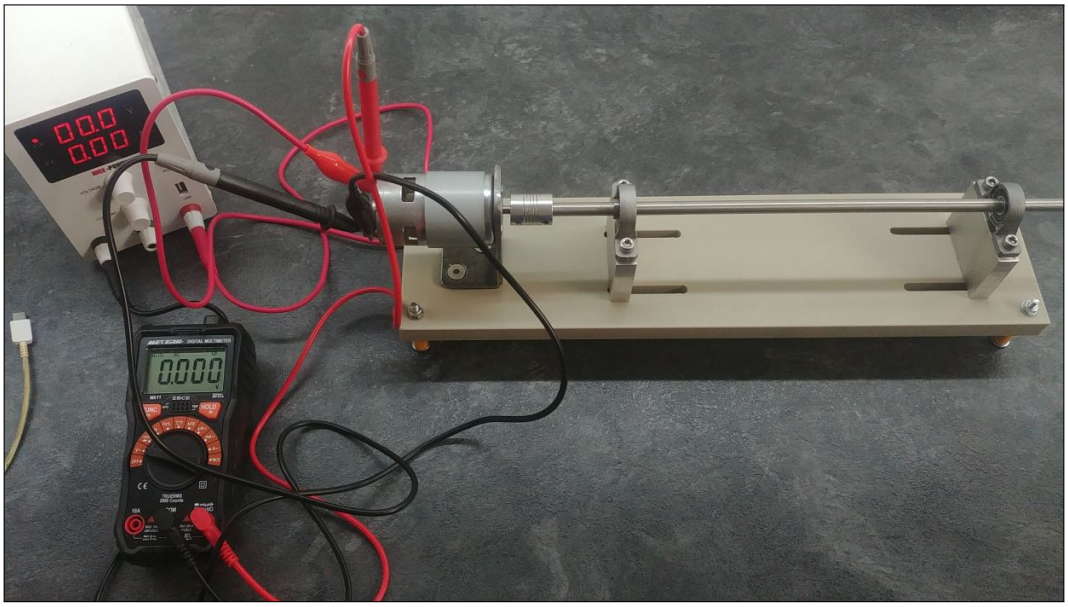


Fig. 7.14. Disposición del banco para las mediciones

TABLA 7.25. VELOCIDADES MEDICIONES BANCO

	Velocidad (Rpm)	Velocidad (Hz)	Voltaje(V)	Corriente(A)
BancoA1o	4196	69,93	12,67	0,76
BancoA2o	5748	95,8	17,16	0,83
BancoA3o	7526	125,43	22,1	0,86
BancoA4o	9587	159,78	28,2	0,88

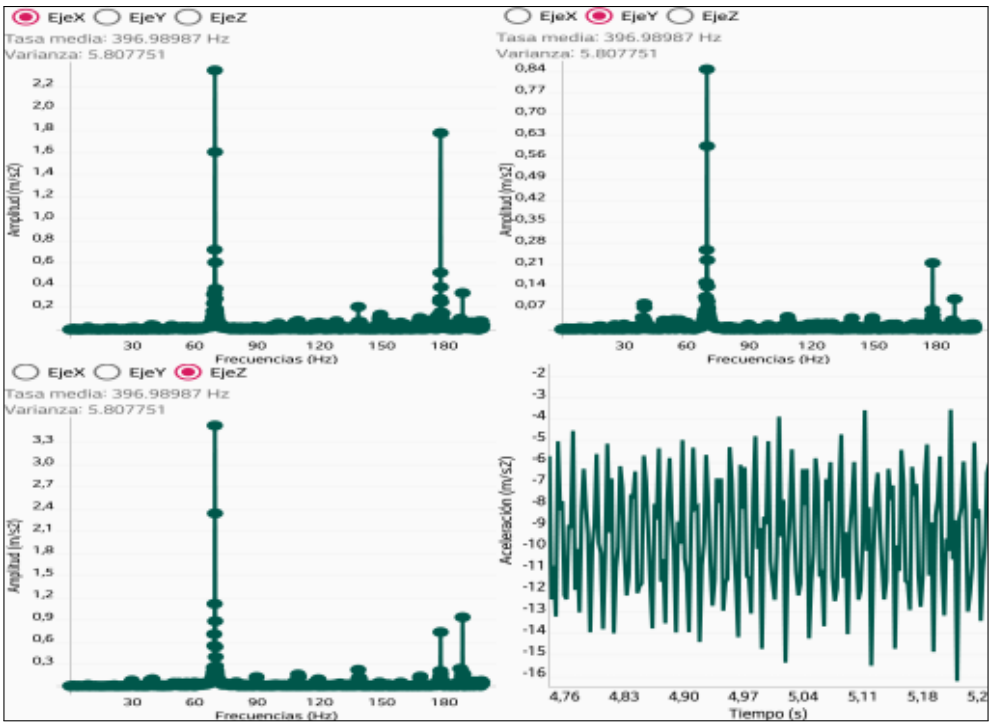


Fig. 7.15. Resultados BancoA1o

TABLA 7.26. RESULTADOS BANCOA1O

BancoA1o	Nº Muestras = 4096	Frec. esperada (Hz) = 69,93
Frecuencias Destacadas (Hz)		
Eje X	69,68 / 178,33	
Eje Y	69,68	
Eje Z	69,68	

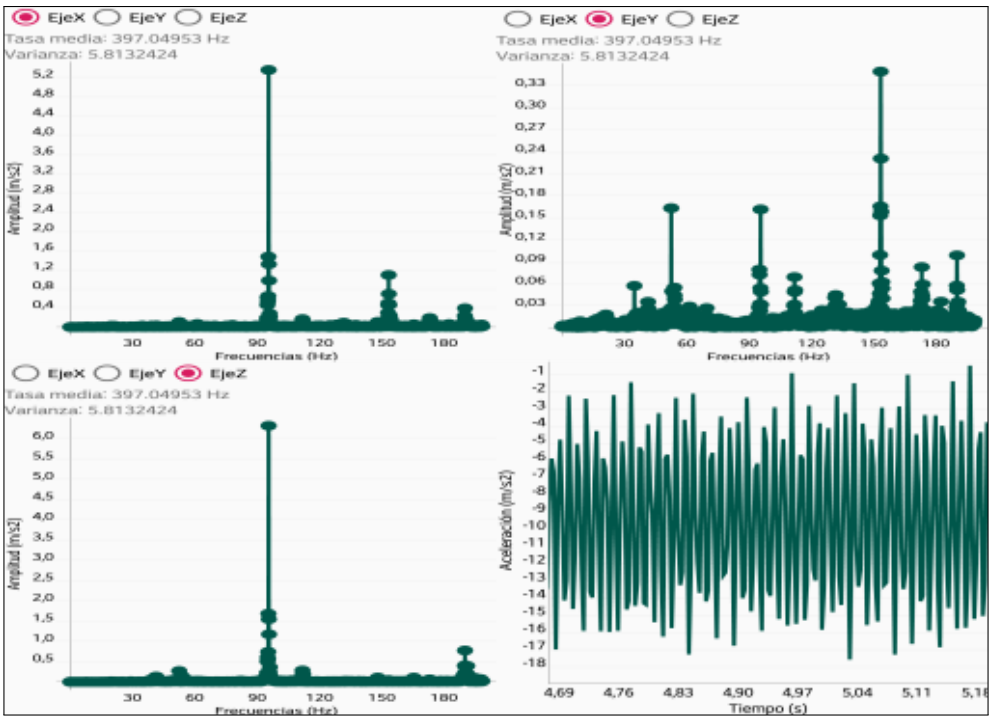


Fig. 7.16. Resultados BancoA2o

TABLA 7.27. RESULTADOS BANCOA2O

BancoA2o	Nº Muestras = 4096	Frec. esperada (Hz) = 95,8
Frecuencias Destacadas (Hz)		
Eje X	95,57 / 153,35	
Eje Y	52,53 / 95,57 / 153,35	
Eje Z	95,57	

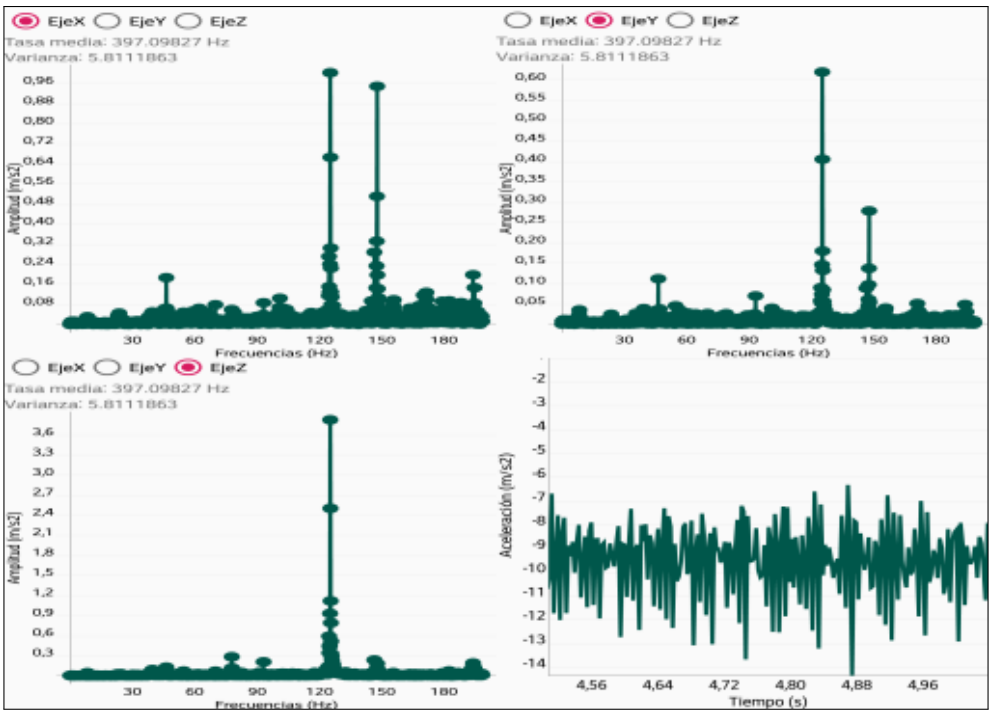


Fig. 7.17. Resultados BancoA3o

TABLA 7.28. RESULTADOS BANCOA3O

BancoA3o	Nº Muestras = 4096	Frec. esperada (Hz) = 125,43
Frecuencias Destacadas (Hz)		
Eje X	125,25 / 147,84	
Eje Y	125,25 / 147,84	
Eje Z	125,25	

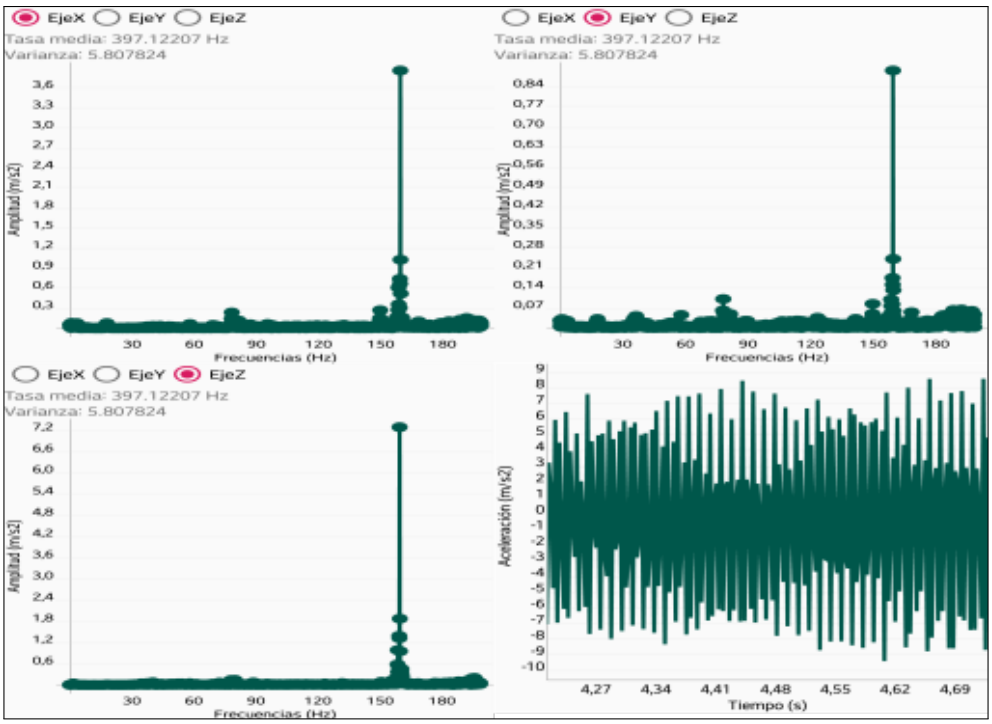


Fig. 7.18. Resultados BancoA3o

TABLA 7.29. RESULTADOS BANCOA4O

BancoA4o	Nº Muestras = 4096	Frec. esperada (Hz) = 159,78
Frecuencias Destacadas (Hz)		
Eje X	159,48	
Eje Y	159,48	
Eje Z	159,48	

Como se anticipó, con la construcción de un banco de pruebas más preciso y calibrado las vibraciones correspondientes a otros elementos que no sean el eje principal han sido muy minimizadas. En todos los casos la vibración principal es la esperada por la velocidad de rotación.

En los casos con velocidad de rotación menor aparecen otras vibraciones que pueden ser causadas por otros elementos como por ejemplo la propia sujeción para el dispositivo móvil, la geometría del banco de pruebas o los rodamientos. También pueden ser vibraciones de mayor frecuencia que las del máximo medible (200 Hz) y por *aliasing* aparecen en la gráfica.

Otras posiciones de medida

Todas las mediciones realizadas hasta el momento en el banco de pruebas se han realizado con los dos soportes del eje en su posición más cercana el uno al otro y el soporte del dispositivo móvil se ha situado en el soporte más lejano al motor. Aprovechando que el soporte del dispositivo se diseño para que se pueda mover entre soportes fácilmente y que estos soportes se pueden desplazar a lo largo del dispositivo móvil se realizarán cambios en las posiciones de estos.

Estos cambios de posiciones pueden dar mediciones distintas debido por ejemplo a que las mediciones mas cercanas al motor puedan captar en mayor medida del desalineamiento de la conexión de los ejes del motor y el eje principal o también por que el eje principal no tiene una alineación correcta y puede causar otras vibraciones por ese motivo. Solo se cogerá las mediciones del acelerómetro en la dirección "X" para los nuevos casos.

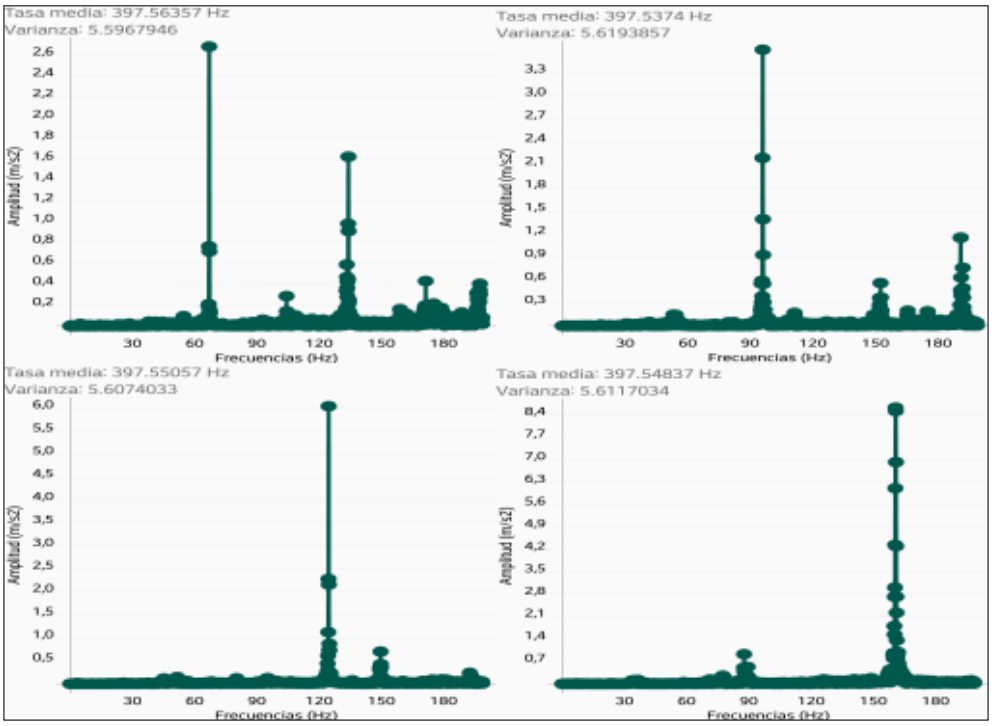


Fig. 7.19. Resultados BancoAx

TABLA 7.30. RESULTADOS BANCOAX

BancoAx	Nº Muestras = 4096	Soportes Juntos - Cerca Motor
Frecuencias Destacadas (Hz)		
A1x	67,06 / 134,13	Esperada: 66.88
A2x	95,98	Esperada: 96,1
A3x	124,62	Esperada: 125,13
A4x	160,72	Esperada: 160,56

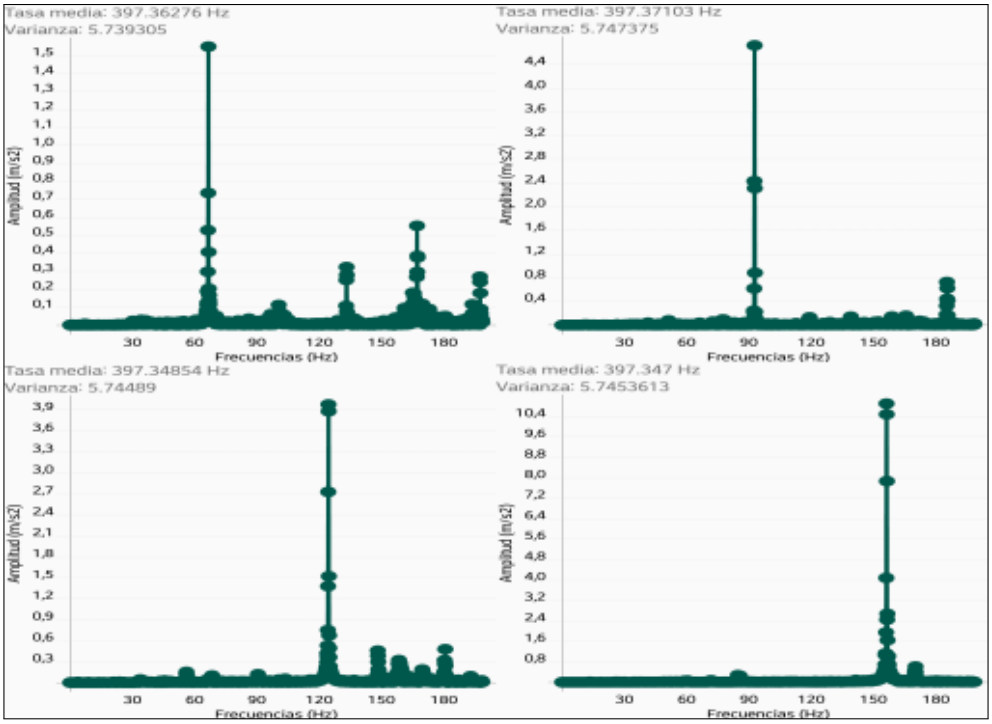


Fig. 7.20. Resultados BancoBo

TABLA 7.31. RESULTADOS BANCOBO

BancoBo	Nº Muestras = 4096	Soportes Separados - Lejos Motor
Frecuencias Destacadas (Hz)		
B1o	66,64 / 133,19 / 167,15	Esperada: 66,78
B2o	92,74	Esperada: 92,71
B3o	124,46	Esperada: 123,3
B4o	156,28	Esperada: 156,33

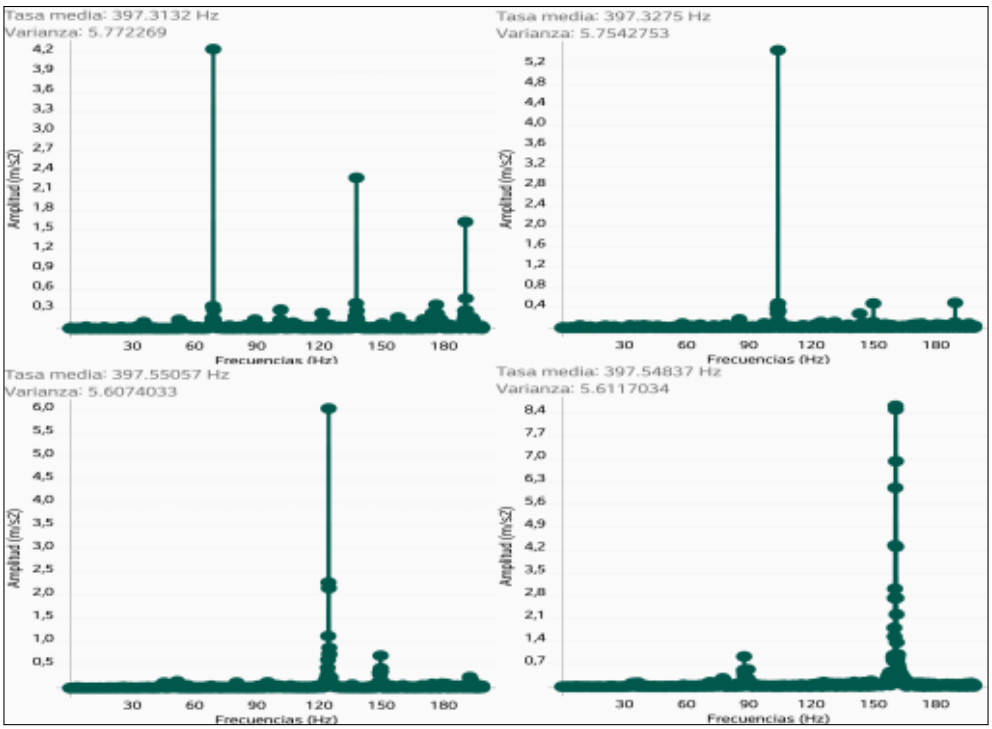


Fig. 7.21. Resultados BancoBx

TABLA 7.32. RESULTADOS BANCOBX

BancoBx	Nº Muestras = 4096	Soportes Separados - Cerca Motor
Frecuencias Destacadas (Hz)		
B1x	68,96 / 137,93	Esperada: 68,95
B2x	103,98	Esperada: 104
B3x	126,59	Esperada: 126,66
B4x	160,73	Esperada: 160,43

Las medidas realizadas en distintos soportes y distinta separación entre los soportes son muy similares entre sí por lo que la influencia del cambio de geometría y la posición de medida es nula si es que existe. Aparecen algunas frecuencias que hacen que los casos se diferencien pero dado que las medidas se hicieron a distintas velocidades con las pruebas realizadas no es posible decir si los cambios tienen un efecto en los resultados.

Lo que sí es útil de estos datos es que aportan más mediciones que corroboran que los resultados obtenidos son buenos para obtener una frecuencia predominante. Como se puede ver en las tablas, la frecuencia esperada esta cerca de las obtenidas.

8. PRESUPUESTO

En este apartado se desarrollarán todos los costes derivados de la realización del proyecto. En primer lugar se desarrollan los costes relacionados con las horas dedicadas al proyecto como **gastos de personal**. Estas horas pueden ser en distintos aspectos del proyecto pero se contabilizarán de la misma forma por hora trabajada.

Las divisiones hechas del tiempo incluyen lo siguiente: *Documentación*: Documentación e información buscada antes de iniciar la programación y durante esta. Después en otros aspectos como el diseño de los bancos de prueba; *Programación*: Tiempo invertido con el programa Android Studio para la programación de la aplicación; *Diseño 3D*: Diseño de las piezas que fueron impresas y bocetos y diseño de los bancos de prueba; *Pruebas*: Realización de las distintas pruebas con la aplicación en los bancos de prueba y sin ellos; *Redacción y Latex*: Programación en Latex, búsqueda de cohesión de la información en un archivo y su redacción, realización de distintas gráficas y exposición de resultados.

TABLA 8.1. COSTES DE PERSONAL

	Horas
Documentación	50
Programación	120
Diseño 3D	22
Construcción	18
Pruebas	10
Redacción y Latex	95
Horas totales	315
Precio hora (€/h)	20
Total (€)	6300

En la tabla 8.1 se puede ver un desglose de las horas utilizadas en distintas tareas. Para el coste por hora se ha considerado un valor no muy alto debido a que el trabajo es realizado por un ingeniero que aún no se ha graduado y en algunos aspectos como el desarrollo de la aplicación tomaron más tiempo del usual debido a tener que aprender mucha información. Un desarrollador de aplicaciones con experiencia en ello cobraría un precio muy elevado en comparación aunque le costaría menos horas en llegar al mismo resultado.

TABLA 8.2. COSTES DE EQUIPOS

Tipo	Modelo	Años amortización	Tiempo uso (años)	Precio (€)	Precio con amort. (€)
Ordenador	MacBook Air 2015	4	1,5	1100	375
Móvil	LG G6	4	1	700	175
Impresora 3D	Ender 3	2	0,1	250	12,5
Sensor	Tacómetro	-	-	15	15
Generador	-	-	-	40	40
Coste total				617.5€	

Para valorar el **coste por los equipos utilizados** con otros propósitos se tiene en cuenta el tiempo en el que tiene que ser amortizado un equipo y el tiempo que se ha dedicado a su uso como se puede ver en la tabla 8.2. Para equipos electrónicos con un alto valor de adquisición se considerará un periodo de amortización de cuatro años por los avances en tecnología para no quedarse obsoleto y por la degradación de elementos como la batería.

Para la impresora 3D se acorta este periodo de amortización ya que se están realizando mayores avances en este campo y se esta utilizando la impresora para otros usos por lo que es más fácilmente amortizable.

Todos las **herramientas informáticas** utilizadas durante este proyecto no suman al presupuesto ya que se eligieron en parte a su coste cero. Este coste se debe a que los programas utilizados son gratuitos, de código libre o disponen de algún tipo de licencia académica o de uso no comercial.

TABLA 8.3. COSTES DEL BANCO DE PRUEBA

Materiales	
Tipo	Coste (€)
Placa	15
Motor	12
Rodamientos	20
Corte láser discos	35
Impresiones 3D	50
Otros	40
Personal	340
Total	512 €

La construcción del **banco de pruebas** se realizó en un taller y el trabajo que requirió utilizar máquinas de mecanizado u otras herramienta se realizó por personal

cualificado lo que repercute en el coste por personal, siendo este coste mayor. Se incluye el coste de algunas de las piezas que conformaron el banco además del coste de las impresiones 3D utilizadas como medio de apoyo a la toma de mediciones en el banco.

TABLA 8.4. COSTES TOTALES

Tipo	Coste (€)
Personal	6300
Equipos	617,5
Herramientas informáticas	0
Banco de pruebas	512
Total	7429,5 €

A los costes totales de la tabla 8.4 se deben añadir los costes derivados de los pagos de impuestos de un trabajador o una cuota de autónomo si el trabajo se realizase a cuenta propia. El resto de equipos y partes ya se abonó el IVA correspondiente en el momento de su compra.

9. CONCLUSIONES Y TRABAJOS FUTUROS

Se ha cumplido el **objetivo principal** de programar una aplicación para el análisis de vibraciones. Se ah conseguido desarrollar una aplicación que cumple los requisitos que se le pedían y es completamente funcional. La aplicación es capaz de acceder a los datos del acelerómetro incorporado en el dispositivo y se ha identificado la forma de hacerlo en el sistema Android.

Estos datos pueden ser guardados en la memoria del dispositivo para su exportación o análisis posterior en otra pantalla de la aplicación. Este análisis consiste en aplicarle la transformada rápida Fourier o FFT y mostrar los resultados en un gráfica para ver que frecuencias son las más significativas de la señal medida.

A parte del objetivo principal del proyecto también se han alcanzado otros objetivos que pueden ser previos al desarrollo de la aplicación o a consecuencia de ella. Estos objetivos cumplidos se desarrollan a continuación:

- *Documentación y estudio de mercado*:
  - *Aplicaciones previas*: Se llevó a cabo un estudio de las aplicaciones existentes que utilizan y acceden a los datos del acelerómetro de algún modo. Se encontró que ninguna cumplía los requisitos que se quería para la aplicación. Cumplían algunos pero no todos y dos de las aplicaciones más interesantes se exponen con profundidad en el proyecto.
  - *Android y programación*: Se realizó una documentación muy amplia sobre Android y como programar para este sistema que amplió su duración a toda la programación de la aplicación. Se consultaron una gran cantidad de fuentes bibliográficas pero la mejor forma de adquirir conocimientos de programación es realizando la programación en sí.
  - *Herramientas informáticas*: Se consultó la documentación disponible para las distintas herramientas utilizadas y así poder utilizar un número mayor de recursos. Así se realiza un proyecto más complejo pero sobre todo más completo.
- *Requisitos de la aplicación*: Se definen unos requisitos claros que debe cumplir la aplicación y sobre todo se definen los requisitos que diferencian la aplicación a desarrollar de otras. Estos requisitos son por ejemplo la capacidad de importar y exportar los datos de las mediciones de una forma sencilla o la representación del análisis frecuencial de la señal para obtener la frecuencias principales.

- *Diagrama de flujo de la aplicación*: La aplicación se divide en pantallas o *activities*. Desde el menú principal se accede a estas pantallas que actúan como pestañas y cada una con una función diferenciada. Estas funciones se describen a continuación:

    - *Activity - Medida de datos*: En esta pantalla se hace una pruebas de los distintos modos de funcionamiento que tiene Android para el acelerómetro y el cálculo de la tasa de muestreo con el reloj interno del dispositivo. Los datos del acelerómetro se muestran a través de una gráfica dinámica. Desde esta pantalla se accede a través de un botón a otra donde se muestran las propiedades del acelerómetro.
    - *Activity - Guardado de datos:* Aquí se elije los parámetros de la medición como el número de muestras y estas se guardan en un directorio dentro de la memoria interna
    - *Activity - Análisis de datos*: En esta pantalla se cogen la medición que se ha hecho previamente y se realiza la FFT obteniendo una gráfica. Se puede elegir de qué eje se quiere hacer el análisis.

- *Desarrollo de bancos de ensayo*: Se consiguió desarrollar dos bancos de ensayo para comprobar la funcionalidad de la aplicación en casos reales. Uno de los bancos de una forma más precisa que el otro. Ambos fueron útiles para generar vibraciones de una forma controlada y así poder medirlas con la aplicación.

- *Verificación de la aplicación*: Con los resultados de las distintas pruebas realizadas se ha comprobado que el acelerómetro incorporado es capaz de detectar vibraciones de frecuencia específicas al comparar los resultados con la frecuencia esperada debido a la rotación mediante un tacómetro.

  Aunque se puedan distinguir distintas frecuencias, la desviación de las mediciones por el resto de procesos funcionando en el dispositivo, la intencionalidad de uso del acelerómetro que se puede ver en la programación hace ver que los acelerómetros montados en estos dispositivos nunca fueron pensados para el propósito que se le ha dado en este proyecto. Las tareas para las que han sido diseñados son tareas en las que no se exige una tasa de muestreo muy alta como puede ser contabilizar los pasos.

  Además la identificación de frecuencias principales no aplica a todos los dispositivos (se han realizado pruebas las pruebas principales con un solo dispositivo) que puedan llevar instalada esta aplicación. Como se ha comprobado en los resultados, hay dispositivos con acelerómetros con capacidades inferiores (principalmente por su antigüedad y su precio) y no se puede asegurar la funcionalidad de la aplicación en dichos dispositivos.

Con el rápido avance de la tecnología los acelerómetros que incorporan los dispositivos móviles son cada vez más precisos y pueden producir datos con una tasa de muestreo mayor por lo que la capacidad en cuanto al análisis de vibraciones solo puede mejorar con los nuevos dispositivos que salgan al mercado. El acelerómetro es un componente que suele ser olvidado en la compra de un nuevo móvil por lo que no hay información al respecto y no se sabe las capacidades del acelerómetro antes de probarlo.

Impacto del proyecto

Como se ha podido ver en los resultados de trabajo, estos están limitados por las capacidades del acelerómetro del dispositivo y por tanto ocurre lo mismo con el impacto que puede tener la aplicación. Aunque no todos los dispositivos sean capaces de obtener resultado, lo que sí se puede afirmar actualmente es el alcance de la aplicación. Toda persona que tenga un dispositivo Android con una versión de este compatible puede utilizar la aplicación sin ningún coste añadido.

La aplicación podría servir para analizar vibraciones en entornos industriales de forma rápida. En medio minuto se pueden ver fallos en maquinaria o motores que de utilizar otras herramientas seria mucho más costoso. Si los acelerómetros de los dispositivos móviles mejoran sustancialmente, este tipo de aplicaciones puede ser una alternativa competitiva.

También puede tener la aplicación utilidad didáctica para comprender las vibraciones, su análisis y la información que se puede obtener de este.

9.1. Trabajos futuros

Los dispositivos móviles están en constante innovación por lo que las capacidades de estos hoy en día pueden ser muy diferentes de las capacidades de aquí a diez años o incluso mucho menos. Este mismo proyecto en una fecha futura puede dar datos muy diferentes. Las versiones de Android permiten más manipulación del acceso al acelerómetro o simplemente los acelerómetros son mejores y más capaces.

Ya se ha visto que los móviles más modernos y de las gamas altas de los fabricantes ya consiguen tasas de muestreo y sensibilidades mucho mejores que el resto por lo que no sería extraño que con el tiempo el coste de estos acelerómetros más potentes baje y se conviertan en un estándar en los dispositivos.

En cuento a los proyectos que se pueden realizar que complementan o profundizan en aspectos de este proyecto son los siguientes:

- **Aplicación mejorada**:

  Apariencia: Con los conocimientos sobre Android se centra el esfuerzo en la parte funcional sin comprometer en exceso las indicaciones visuales y la apariencia de la aplicación. Se puede pulir la apariencia haciéndola más atractiva al usuario además de más intuitiva, en ocasiones puede hacerse un poco difícil de seguir.

  Funcionalidades: Se pueden añadir funcionalidades extra a modo de actualización por ejemplo una función que detecte automáticamente los picos de las gráficas FFT.

- **Bluetooth**: Añadir la opción de obtener las mediciones de acelerómetros externos por bluetooth de forma inalámbrica. Estos acelerómetros pueden tener mayores capacidades que el que que monta el dispositivo y por tanto obtener mejores mediciones.

- **Mejora mediciones**:

  Filtros: Se pueden añadir algún tipo de filtro para quedarse con las medidas que más interesen o mejorar las existentes por ejemplo ante la variación de rendimiento del dispositivo.

  Optimización: Se puede optimizar la aplicación para que un mayor rendimiento se traduzca en mediciones más precisas.

- **Ampliación de la muestra de dispositivos probados**: Las capacidades y las especificaciones de los acelerómetros no son fáciles de encontrar probablemente causado porque no es un punto determinante para la elección de dispositivo en la actualidad. Un estudio más amplio de los modelos de distintas marcas puede determinar que dispositivos son más capaces en cuanto a la toma y análisis de mediciones.

- **Desarrollo en iOS**: Como se pudo ver en el apartado de entorno socioeconómico, gran parte de los dispositivos móviles mundiales son del fabricante Apple y funcionan bajo el sistema "iOS". Conocer cómo funciona el acelerómetro en estos dispositivos y su capacidad sería interesante ya que se podría llegar a más usuarios.

BIBLIOGRAFÍA

[1] S. S. Rao, *Mechanical vibrations*, 5.ª ed. Upper Saddle River, NJ, USA: Pearson Education, 2011.

[2] V. Goga y M. Kľúčik, "Optimization of Vehicle Suspension Parameters with use of Evolutionary Computation," *Procedia Engineering*, vol. 48, 174–179, 12-2012. DOI: 10.1016/j.proeng.2012.09.502.

[3] *"Introduction to Activities"*, 27-12-2019. [En línea]. Disponible en: https://developer.android.com/guide/components/activities/intro-activities (Acceso: 04-04-2020).

[4] *"El ciclo de vida de una aplicación de Android"*. [En línea]. Disponible en: https://www.androidsis.com/el-ciclo-de-vida-de-una-aplicacion-de-android/ (Acceso: 28-04-2020).

[5] *"Introduction to Android Views and ViewGroups"*. [En línea]. Disponible en: https://www.studytonight.com/android/introduction-to-views (Acceso: 15-03-2020).

[6] *"Difference between a Views Padding and Margin"*, 2014. [En línea]. Disponible en: https://stackoverflow.com/questions/4619899/difference-between-a-views-padding-and-margin (Acceso: 21-04-2020).

[7] *"What is the difference between gravity and layout gravity in Android? "* 2014. [En línea]. Disponible en: https://bit.ly/2FZZ1Pw (Acceso: 23-04-2020).

[8] *"Using an ArrayAdapter with ListView"*. [En línea]. Disponible en: https://bit.ly/3cPMyZg (Acceso: 25-04-2020).

[9] *"Menus"*, 24-02-2020. [En línea]. Disponible en: https://developer.android.com/guide/topics/ui/menus (Acceso: 15-04-2020).

[10] *"Como crear menús de opciones en Android"*. [En línea]. Disponible en: https://bit.ly/2HBEhxJ (Acceso: 23-04-2020).

[11] *"Cómo crear un AlertDialog en Android"*. [En línea]. Disponible en: https://bit.ly/3oqIsxg (Acceso: 03-05-2020).

[12] *"Sensores"*, 27-12-2019. [En línea]. Disponible en: https://developer.android.com/guide/topics/sensors (Acceso: 17-04-2020).

[13] *"Crear una Biblioteca en Android"*. [En línea]. Disponible en: https://developer.android.com/studio/projects/android-library?hl=es-419 (Acceso: 24-08-2019).

[14] *"GraphView"*, 12-04-2019. [En línea]. Disponible en: https://github.com/jjoe64/GraphView (Acceso: 27-04-2019).

[15] *"StorageChooser"*, 12-04-2020. [En línea]. Disponible en: https://github.com/codekidX/storage-chooser (Acceso: 12-08-2020).

[16] *"Paquete apache.commons.math3"*. [En línea]. Disponible en: https://commons.apache.org/proper/commons-math/javadocs/api-3.4/org/apache/commons/math3/transform/package-summary.html (Acceso: 03-03-2020).

[17] *"HelloCharts"*, 18-03-2018. [En línea]. Disponible en: https://github.com/lecho/hellocharts-android (Acceso: 14-08-2020).

[18] *Ficha protección de datos AEPD*. [En línea]. Disponible en: https://www.aepd.es/sites/default/files/2019-11/nota-tecnica-apps-moviles.pdf (Acceso: 09-08-2020).

[19] *"Centro de políticas de desarrolladores"*. [En línea]. Disponible en: https://play.google.com/intl/es/about/developer-content-policy/ (Acceso: 24-06-2020).

[20] *"StatCounter Global Stats"*, 2020. [En línea]. Disponible en: https://gs.statcounter.com (Acceso: 02-10-2020).

[21] Gartner. (11-06-2019). ""Gartner Says Global Smartphone Sales Declined 2.7 % in First Quarter of 2019",” [En línea]. Disponible en: https://www.gartner.com/en/newsroom/press-releases/2019-06-11-gartner-survey-finds-more-australian-workers-ready-to0 (Acceso: 08-08-2019).

[22] F. Bajak, *"US adds new sanction on Chinese tech giant Huawei"*, 15-05-2020. [En línea]. Disponible en: https://www.seattletimes.com/business/u-s-ramps-up-sanctions-on-chinese-tech-giant-huawei/.

[23] S. Kiran, *"US tightens restrictions on suppliers to Huawei"*, 17-08-2020. [En línea]. Disponible en: https://www.ft.com/content/b6e8cbd3-88dd-4212-b98d-58c7bf0334ce.

[24] *Ficha técnica PLA BQ*. [En línea]. Disponible en: https://www.bq.com/es/support/pla-premium/support-sheet (Acceso: 17-07-2020).

[25] T. Yao, Z. Deng, K. Zhang y S. Li, “A method to predict the ultimate tensile strength of 3D printing polylactic acid (PLA) materials with different printing orientations,” *Composites Part B: Engineering*, vol. 163, pp. 393 -402, 2019. DOI: 10.1016/j.compositesb.2019.01.025.

[26] T. Yao et al., “Tensile failure strength and separation angle of FDM 3D printing PLA material: Experimental and theoretical analyses,” *Composites Part B: Engineering*, vol. 188, 2020. DOI: 10.1016/j.compositesb.2020.107894.

[27] *Ficha técnica PLA 850 Sataka*. [En línea]. Disponible en: https://sakata3d.com/es/pla-850/24-pla-850-blanco.html (Acceso: 12-08-2020).

[28] P. Cahill, L. Quirk, P. Dewan y V. Pakrashi, "Comparison of smartphone accelerometer applications for structural vibration monitoring," *Advances in Computational Design*, vol. 4, pp. 1-13, 01-2019. DOI: 10.12989/acd.2019.4.1.001.

[29] T. L. Schmitz y K. S. Smith, *Mechanical Vibrations: Modeling and Measurement*, 1.ª ed. New York, NY: Springer New York, 2011.

[30] M. Baez et al., *Introducción a Android*, 1.ª ed. Madrid: Grupo Tecnología UCM, 2013.

[31] M. L. Murphy, *The Busy Coder's Guide to Advanced Android Development*. CommonsWare, LLC, 2009.

[32] M. Gargenta, *Learning Android*, 1.ª ed. 2011.

[33] J. Tomás Gironés, *El gran libro de Android*. Barcelona: Marcombo, 2011.

[34] P. Barberán Molina, *Aspectos jurídicos de las aplicaciones móviles (APPS)*, Acta, 2016. [En línea]. Disponible en: https://www.acta.es/medios/informes/2016002.pdf.

[35] N. Aliheidari, R. Tripuraneni, A. Ameli y S. Nadimpalli, "Fracture resistance measurement of fused deposition modeling 3D printed polymers," *Polymer Testing*, vol. 60, pp. 94 -101, 2017. DOI: 10.1016/j.polymertesting.2017.03.016.

[36] J. Ramírez, F. Palma, A. Montellano y M. Martínez, "Montaje de un Banco de Pruebas para Medir Esfuerzos de Contacto y Vibración en Pares de Engranes," *Cultura Científica y Tecnológica*, vol. 16, pp. 12-17, 01-2019. DOI: 10.20983/culcyt.2019.1.2.2.

[37] C. Soto, J. Mera, J. D. Cano-Moreno y J. Garcia-Bernardo, "Low-Cost, High-Frequency, Data Acquisition System for Condition Monitoring of Rotating Machinery through Vibration Analysis-Case Study," *Sensors*, vol. 20, p. 3493, 06-2020. DOI: 10.3390/s20123493.

[38] S. L. Lau y K. David, "Movement recognition using the accelerometer in smartphones," 07-2010, pp. 1 -9.

[39] J. Peláez Barrajón y A. F. San Juan, "Validity and Reliability of a Smartphone Accelerometer for Measuring Lift Velocity in Bench-Press Exercises," *Sustainability*, vol. 12, n.º 6, p. 2312, 2020. DOI: 10.3390/su12062312.

[40] P. Lehman, P. Kime, M. Wemheuer, A. Boruvka y J. Wright, *The biblatex Package*, 30-10-2018. [En línea]. Disponible en: http://linorg.usp.br/CTAN/macros/latex/contrib/biblatex/doc/biblatex.pdf.

[41] M. Ruiz de Luzuriaga, *Guía para citar y referenciar. IEEE Style*, Biblioteca de la Universidad Pública de Navarra. Oficina de Referencia, 15-07-2016. [En línea]. Disponible en: http://www2.unavarra.es/gesadj/servicioBiblioteca/tutoriales/Citar_referenciar_(IEEE).pdf.

A. CÓDIGO DE LA ACTIVITY ACCDATA

CÓDIGO A.1. Archivo activity_acc_data.xml

```
1
2  <?xml version="1.0" encoding="utf-8"?>
3  <FrameLayout xmlns:tools="http://schemas.android.com/tools"
4      android:layout_width="match_parent"
5      android:layout_height="match_parent"
6      xmlns:android="http://schemas.android.com/apk/res/android">
7
8      <ScrollView xmlns:tools="http://schemas.android.com/tools"
9          android:layout_width="match_parent"
10         android:layout_height="wrap_content"
11         android:paddingLeft="10dp"
12         android:paddingRight="10dp"
13         tools:ignore="ScrollViewCount">
14
15         <LinearLayout
16             android:layout_width="match_parent"
17             android:layout_height="wrap_content"
18             android:orientation="vertical">
19
20             <LinearLayout
21                 android:layout_width="match_parent"
22                 android:layout_height="wrap_content"
23                 android:gravity="center_horizontal"
24                 android:orientation="horizontal">
25
26                 <Button
27                     android:id="@+id/Start"
28                     android:layout_width="wrap_content"
29                     android:layout_height="wrap_content"
30                     android:gravity="center"
31                     android:text="Start" />
32
33                 <Button
34                     android:id="@+id/Pause"
35                     android:layout_width="wrap_content"
36                     android:layout_height="wrap_content"
37                     android:text="Pause" />
38
39                 <Button
40                     android:id="@+id/Stop"
41                     android:layout_width="wrap_content"
42                     android:layout_height="wrap_content"
43                     android:text="Stop" />
44             </LinearLayout>
45
46             <TableLayout
47                 android:layout_width="match_parent"
48                 android:layout_height="wrap_content"
49                 android:layout_marginTop="5dp"
50                 android:orientation="horizontal">
51
52                 <TableRow>
53
54                     <TextView
55                         android:layout_width="wrap_content"
```

```
56                          android:layout_height="wrap_content"
57                          android:text="Aceleracion X:" />
58
59                      <TextView
60                          android:id="@+id/acelX"
61                          android:layout_width="match_parent"
62                          android:layout_height="wrap_content" />
63                  </TableRow>
64
65                  <TableRow>
66
67                      <TextView
68                          android:layout_width="wrap_content"
69                          android:layout_height="wrap_content"
70                          android:text="Aceleracion Y:" />
71
72                      <TextView
73                          android:id="@+id/acelY"
74                          android:layout_width="match_parent"
75                          android:layout_height="wrap_content" />
76                  </TableRow>
77
78                  <TableRow>
79
80                      <TextView
81                          android:layout_width="wrap_content"
82                          android:layout_height="wrap_content"
83                          android:text="Aceleracion Z:" />
84
85                      <TextView
86                          android:id="@+id/acelZ"
87                          android:layout_width="match_parent"
88                          android:layout_height="wrap_content" />
89                  </TableRow>
90
91                  <LinearLayout
92                      android:layout_width="match_parent"
93                      android:layout_height="wrap_content"
94                      android:orientation="horizontal">
95
96                      <TextView
97                          android:layout_width="wrap_content"
98                          android:layout_height="wrap_content"
99                          android:text="Tasa de muestreo:" />
100
101                     <TextView
102                         android:id="@+id/TasaMuestreo"
103                         android:layout_width="wrap_content"
104                         android:layout_height="wrap_content" />
105
106                     <TextView
107                         android:id="@+id/textView"
108                         android:layout_width="wrap_content"
109                         android:layout_height="wrap_content"
110                         android:text="Manual :" />
111
112                     <EditText
113                         android:id="@+id/tasamanual"
114                         android:layout_width="wrap_content"
115                         android:layout_height="wrap_content"
116                         android:hint="Tasa Manual"
117                         android:inputType="number"
118                         android:importantForAutofill="no" />
119                 </LinearLayout>
```

```
                <LinearLayout
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:orientation="horizontal">

                    <Spinner
                        android:id="@+id/spinner"
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content" />

                    <Button
                        android:id="@+id/Tasa"
                        style="@style/Widget.AppCompat.Button.Small"
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:gravity="center"
                        android:onClick="BtnIrTest"
                        android:text="Test" />

                </LinearLayout>
            </TableLayout>

            <com.github.mikephil.charting.charts.LineChart
                android:id="@+id/Grafica1"
                android:layout_width="match_parent"
                android:layout_height="250sp">
            </com.github.mikephil.charting.charts.LineChart>

        </LinearLayout>
    </ScrollView>
```

CÓDIGO A.2. Archivo accData.java

```
package com.example.jorge.vibtest_10;

import android.content.Context;
import android.content.Intent;
import android.content.res.Configuration;
import android.graphics.Color;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.icu.util.TimeUnit;
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Spinner;
import android.widget.TextView;
import android.widget.Toast;

import com.github.mikephil.charting.charts.LineChart;
import com.github.mikephil.charting.components.Legend;
import com.github.mikephil.charting.components.XAxis;
import com.github.mikephil.charting.components.YAxis;
import com.github.mikephil.charting.data.Entry;
```

```
29  import com.github.mikephil.charting.data.LineData;
30  import com.github.mikephil.charting.data.LineDataSet;
31  import com.github.mikephil.charting.interfaces.datasets.ILineDataSet;
32
33  import java.text.DecimalFormat;
34  import java.text.NumberFormat;
35  import java.util.ArrayList;
36  import java.util.List;
37
38
39  public class AccData extends AppCompatActivity implements SensorEventListener,
        AdapterView.OnItemSelectedListener {
40
41      private SensorManager sensorManager;
42      Sensor acelerometro;
43
44      private Button start;
45      private Button pause;
46      private Button stop;
47
48      private LineChart mChart;
49      private Thread thread;
50      private boolean plotData = true;
51
52      TextView acelX, acelY, acelZ;
53      TextView TasaMuestreo;
54
55      private EditText tasaManual;
56
57      private float lastX, lastY, lastZ;
58
59      private int i;
60
61      private boolean manual=false;
62
63
64      @Override
65      protected void onCreate(Bundle savedInstanceState) {
66          super.onCreate(savedInstanceState);
67          setContentView(R.layout.activity_acc_data);
68
69          acelX = (TextView) findViewById(R.id.acelX);
70          acelY = (TextView) findViewById(R.id.acelY);
71          acelZ = (TextView) findViewById(R.id.acelZ);
72          TasaMuestreo = (TextView) findViewById(R.id.TasaMuestreo);
73
74          start = (Button) findViewById(R.id.Start);
75          pause = (Button) findViewById(R.id.Pause);
76          stop = (Button) findViewById(R.id.Stop);
77
78          tasaManual = (EditText) findViewById(R.id.tasamanual);
79
80
81          ↑ Véase Acceso al Acelerómetro - Página 59
82
83          // Se accede al servicio de Android encargado de los sensores creando una
                instancia "sensorManager"
84          sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
85          // Se accede a un sensor en específico con \textit{getDefaultSensor()} y se
                une a la instancia "acelerometro"
86          acelerometro = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
87
88          // Se omite esta comprobación para saber si el dispositivo cuenta con aceleró
                metro
```

```
89          if(acelerometro != null) {
90              sensorManager.registerListener(AccData.this, acelerometro, SensorManager.
                    SENSOR_DELAY_GAME);
91          }
92
93
94          ↑ Véase Creación del Spinner - Página 61
95
96          // Creación del spinner para la elección de la tasa de muestreo. "final"
                porque será contante y contendrá siempre los mismos elementos
97          final Spinner spinner = (Spinner) findViewById(R.id.spinner);
98          // Evitar que el spinner salga desplegado al inicio
99          spinner.setSelected(false);
100         // Se adjudica el elemento cero como la elección por defecto
101         spinner.setSelection(0,true);
102         // Se añade un listener para realizar una acción (cambiar de tasa) al elegir
                un elemento nuevo
103         spinner.setOnItemSelectedListener(this);
104
105         // Se crea un ArrayList(vector) para meter los elementos
106         List<String> categories = new ArrayList<String>();
107         // Se añade el elemento "0" del ArrayList
108         categories.add("SENSOR_DELAY_NORMAL");
109         // Elemento "1"
110         categories.add("SENSOR_DELAY_UI");
111         // Elemento "2"
112         categories.add("SENSOR_DELAY_GAME");
113         // Elemento "3"
114         categories.add("SENSOR_DELAY_FASTEST");
115         // Elemento "4"
116         categories.add("Manual");
117
118         // Se crea el adaptador para el spinner. Se define donde va a ir el spinner,
                su apariencia y de que ArrayList se sacarán los elementos
119         ArrayAdapter<String> dataAdapter = new ArrayAdapter<String>(this, android.R.
                layout.simple_spinner_item, categories);
120         // Estilo del spinner. Se despliegan los elementos hacia abajo
121         dataAdapter.setDropDownViewResource(android.R.layout.
                simple_spinner_dropdown_item);
122
123         // Se une el adaptador al spinner
124         spinner.setAdapter(dataAdapter);
125
126
127         ↑ Véase Inicialización de la Gráfica - Página 64
128
129         // Creación de la gráfica y se une con su parte en el archivo xml con su ID
130         mChart = (LineChart) findViewById(R.id.Grafica1);
131
132         // Se añade título a la gráfica
133         mChart.getDescription().setEnabled(true);
134         mChart.getDescription().setText("Grafica acelerometro tiempo real");
135
136         // Se restringe el poder tocar la gráfica y moverla
137         mChart.setTouchEnabled(false);
138         mChart.setDragEnabled(true);
139         // Se permite el auto-escalado de la gráfica
140         mChart.setScaleEnabled(true);
141         // Se incluye una cuadrícula de fondo
142         mChart.setDrawGridBackground(true);
143         // Se define el color de fondo de la gráfica (blanco)
144         mChart.setBackgroundColor(Color.WHITE);
145
146         // Se crea una línea, en la librería externa llamada LineData
```

```
147             LineData data = new LineData();
148             data.setValueTextColor(Color.WHITE);
149             // De une la línea a la gráfica
150             mChart.setData(data);
151 
152             // Se crea una leyenda para la gráfica
153             Legend l =mChart.getLegend();
154             // Se modifica la leyenda: forma y color
155             l.setForm(Legend.LegendForm.LINE);
156             l.setTextColor(Color.BLACK);
157 
158             // Creación de el eje X y se define su color
159             XAxis xl = mChart.getXAxis();
160             xl.setTextColor(Color.BLACK);
161             // La gráfica tendrá líneas de fondo en ese eje
162             xl.setDrawGridLines(true);
163             xl.setAvoidFirstLastClipping(true);
164             xl.setEnabled(true);
165 
166             // Creación del eje Y en la parte izquierda de la gráfica
167             YAxis izqEje = mChart.getAxisLeft();
168             izqEje.setDrawGridLines(true);
169 
170             // Se deshabilita el eje Y en la parte derecha de la gráfica
171             YAxis derEje = mChart.getAxisRight();
172             derEje.setEnabled(false);
173 
174             // Se añaden los ejes a la gráfica
175             mChart.getAxisLeft().setDrawGridLines(true);
176             mChart.getXAxis().setDrawGridLines(true);
177             // Se añade un bode a la gráfica
178             mChart.setDrawBorders(true);
179 
180 
181 
182             start.setOnClickListener(new View.OnClickListener() {
183                 @Override
184                 public void onClick(View v) {
185 
186 
187                     int spinnerpos = spinner.getSelectedItemPosition();
188                     if (spinnerpos==0){
189                         sensorManager.unregisterListener(AccData.this, acelerometro);
190                         sensorManager.registerListener(AccData.this, acelerometro,
                                SensorManager.SENSOR_DELAY_NORMAL);
191                         manual = false;
192                     } else if (spinnerpos==1) {
193                         sensorManager.unregisterListener(AccData.this, acelerometro);
194                         sensorManager.registerListener(AccData.this, acelerometro,
                                SensorManager.SENSOR_DELAY_UI);
195                         manual = false;
196                     } else if (spinnerpos==2) {
197                         sensorManager.unregisterListener(AccData.this, acelerometro);
198                         sensorManager.registerListener(AccData.this, acelerometro,
                                SensorManager.SENSOR_DELAY_GAME);
199                         manual = false;
200                     } else if (spinnerpos==3) {
201                         sensorManager.unregisterListener(AccData.this, acelerometro);
202                         sensorManager.registerListener(AccData.this, acelerometro,
                                SensorManager.SENSOR_DELAY_FASTEST);
203                     } else if (spinnerpos==4) {
204                         if (tasaManual.getText().toString().length()==0){
205                             Toast.makeText(getApplicationContext(), "Escriba una tasa de
                                    muestreo manual", Toast.LENGTH_LONG).show();
```

```
206                     }
207                     sensorManager.unregisterListener(AccData.this, acelerometro);
208                     sensorManager.registerListener(AccData.this, acelerometro,
                            SensorManager.SENSOR_DELAY_FASTEST);
209                     manual=true;
210                 }
211
212                 startPlot();
213
214             }
215         });
216
217         pause.setOnClickListener(new View.OnClickListener() {
218             @Override
219             public void onClick(View v) {
220                 sensorManager.unregisterListener(AccData.this, acelerometro);
221
222                 if(thread != null){
223                     thread.interrupt();
224
225
226                 }
227             }
228         });
229
230
231         stop.setOnClickListener(new View.OnClickListener() {
232             @Override
233             public void onClick(View v) {
234                 sensorManager.unregisterListener(AccData.this, acelerometro);
235
236                 if(thread==null) {
237                     Toast.makeText(getApplicationContext(), "Primero inicia la
                            ejecucion", Toast.LENGTH_LONG).show();
238                 } else {
239                 thread.interrupt();}
240
241                 acelX.setText("");
242                 acelY.setText("");
243                 acelZ.setText("");
244
245                 TasaMuestreo.setText("0.00Hz ");
246             }
247         });
248     }
249
250
251     ↑ Véase Hilo de la Gráfica - Página 65
252
253     private void startPlot(){
254         //Comprobación de si existe un hilo
255         if(thread !=null){
256             thread.interrupt();
257         }
258         //Creación del hilo
259         thread = new Thread(new Runnable() {
260             @Override
261             public void run() {
262                 while (true){
263                     plotData = true;
264                     try{
265                         // Parada de 100 milisegundos
266                         Thread.sleep(100);
267                     }catch (InterruptedException e){
```

```
                        e.printStackTrace();
                    }
                }
            }
        });
        // Inicio del  hilo
        thread.start();
    }


    //Evitar el reinicio de la activity con el giro de la pantalla
    @Override
    public void onConfigurationChanged(Configuration newConfig) {
        super.onConfigurationChanged(newConfig);
    }


    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy) {
    }


    ↑ Véase Añadir Datos - Página 66

    // Variable contador para eliminar datos
    private int remouvalcounter = 0;
    // Variable con el número de datos que tendrá la gráfica
    private int numdatosvisibles = 50;

    private void addEntry (SensorEvent event){
        // Se añade la línea a la gráfica
        LineData data = mChart.getData();
        // Comprobación si la línea tiene ya algún dato
        if(data !=null){
            // Si tiene algún dato se vacía y se empieza a añadir datos en la posició
                n 0
            ILineDataSet set = data.getDataSetByIndex(0);

            // Si no hay ningún dato se crea un conjunto (set) al que se añadirán los
                 datos
            if(set ==null){
                set = createSet();
                data.addDataSet(set);
            }

            // Añadir un dato el \textit{set}. \textit{addEntry((posición en el \
                textit{set}, dato añadido, 0}. La posición la define el tamaño del
                conjunto al que se le suma el contador "remouvalcounter" y el valor
                introducido la celeración en el eje X
            data.addEntry(new Entry(set.getEntryCount() + remouvalcounter, event.
                values[0]),0);
            // Se actualiza el \textit{set} con el dato nuevo
            data.notifyDataChanged();

            // Se actualiza la gráfica con el dato nuevo
            mChart.notifyDataSetChanged();
            // Se selecciona el máximo de datos en el eje X (50 en este caso)
            mChart.setVisibleXRangeMaximum(numdatosvisibles);
            // Se mueve la vista al último dato introducido
            mChart.moveViewToX(data.getEntryCount());
            // Condicional para eliminar datos de la gráfica que ya no se ven para
                ahorrar memoria. Cuando la gráfica llega al número de datos máximo
                este contador empieza a subir de valor
            if (set.getEntryCount() > numdatosvisibles){
```

```
                data.removeEntry(remouvalcounter, 0);
                remouvalcounter++;
            }
        }
    }


    ↑ Véase Conjunto de Datos - Página 66

    private LineDataSet createSet(){
        // Creación del conjunto de datos
        LineDataSet set = new LineDataSet(null, "Datos tiempo real");
        // Este conjunto se representará en el eje Y izquierdo
        set.setAxisDependency(YAxis.AxisDependency.LEFT);
        // Cambio de ancho de la línea
        set.setLineWidth(1.5f);
        // Elección del color
        set.setColor(Color.GREEN);
        // Elección de la función de la línea en este caso cúbica
        set.setMode(LineDataSet.Mode.CUBIC_BEZIER);
        // Modificar la forma de la función cúbica
        set.setCubicIntensity(0.2f);
        set.setCircleRadius(1.5f);
        // Devuelve el conjunto de datos personalizado
        return set;
    }


    ↑ Véase Tasa de Muestreo - Página 60

    // Inicialización de variables importantes para el cálculo de la tasa
        // Variable para guardar la tasa calculada. Inicializada en "0.00"
        private double samplingrate = 0.00;
        // variable para el contador de muestras
        private double numsamples;
        // Variable para el tiempo
        private double startTime;
        private long startTime2;

    @Override
    // Método que se activa cada vez que se obtiene un datos nuevo del acelerómetro
    public void onSensorChanged(SensorEvent event) {
        // Variable que almacena cada vez el tiempo del sistema en milisegundos
        long now = System.currentTimeMillis();

        // Si la tasa seleccionada NO es manual se sigue este camino. Variable
            booleana "manual"= false
        if(!manual) {
            // Se suma 1 al contador de número de mediciones
            numsamples++;
            // Se llama al método para obtener los datos del acelerómetro. Línea 293
            getData(event);

        }

        // Si la tasa seleccionada es manual se sigue este camino
        if ((manual) && (now >= startTime2 + (820/i))) {
            // Se suma 1 al contador de muestras
            numsamples++;
            // Nueva variable temporal para el cálculo de la tasa en modo manual
            startTime2=System.currentTimeMillis();
            // Se llama al método para obtener los datos del acelerómetro. Línea 293
            getData(event);
        }
```

```
387
388             // Si la gráfica está activa. Variable booleana "plotData"=true
389             if(plotData){
390                 addEntry(event);
391                 plotData = false;
392             }
393
394
395             ↑ Véase Cálculo de Tasa de Muestreo - Página 60
396
397             //  Tasa de muestreo calculada cada segundo (1000 milisegundos)
398             if (now >= startTime + 1000){
399                 // Tasa de muestreo = número mediciones / tiempo desde la última medición
                        en segundos
400                 samplingrate = numsamples / ((now - startTime)/1000);
401                 // Tiempo de inicio de la nueva medición
402                 startTime = System.currentTimeMillis();
403                 // Se formatea el número para que incluya dos decimales
404                 NumberFormat formatter = new DecimalFormat("#0.00");
405                 // Se saca por pantalla la tasa formateada y se le añaden las unidades "
                        Hz"
406                 TasaMuestreo.setText(formatter.format(samplingrate)+ "Hz ");
407                 // Se reinicia el contador de muestras
408                 numsamples=0.0;
409             }
410
411         }
412
413
414         ↑ Véase Mediciones del Acelerómetro - Página 59
415
416         private void getData(SensorEvent event) {
417             // Se obtiene el valor del sensor y se almacena en una variable. Values[0]
                    para el eje "X". 1 para el "Y" y 2 para el "Z"
418             lastX = event.values[0];
419             lastY = event.values[1];
420             lastZ = event.values[2];
421
422             // Se incluyen los datos medidos en el \textit{TextView}. Como el dato de
                    salida esta en formato \textit{float} (un formato numérico) se pasa a
                    texto con \textit{toString()}
423             acelX.setText(Float.toString(lastX));
424             acelY.setText(Float.toString(lastY));
425             acelZ.setText(Float.toString(lastZ));
426         }
427
428
429         // Método para reanudar la actividad de la \textit{activity} al volver a esta
430         protected void onResume() {
431             super.onResume();
432         }
433         // Método para que si se sale de esta \textit{activity} y se pausa al ir a otra
                activity, la gráfica y el acelerómetro dejen de tomar datos
434         @Override
435         protected void onPause() {
436             super.onPause();
437             if(thread != null){
438                 thread.interrupt();
439             }
440             sensorManager.unregisterListener(this,acelerometro);
441         }
442         // Método para que si se sale de la aplicación esta deje de registrar datos del
                acelerómetro
443         @Override
```

```
444         protected void onDestroy() {
445             sensorManager.unregisterListener(this,acelerometro);
446             super.onDestroy();
447         }
448
449
450         // Botón para ir a la \textit{activity} de propiedades del acelerómetro
451         public void BtnIrTest (View vista){
452             Intent intent= new Intent (this, Testmuestreo.class);
453             startActivity(intent);
454         }
455
456
457         ↑ Véase Método onItemSelected() - Página 62
458
459         private int tasamuestreo;
460         @Override
461         public void onItemSelected(AdapterView<?> parent, View view, int position, long
                id) {
462
463             // Cada vez que se selecciona un elemento del spinner "item" guarda el texto
                    de ese elemento
464             String item = parent.getItemAtPosition(position).toString();
465
466             // El switch compara el texto de "item" con los que hay para cada opción o "
                    case".
467             switch (item) {
468                 // En los modos de tasa de muestreo predeterminados de Android se
                        desregistra el acelerómetro y se vuelve a registrar con la tasa
                        seleccionada. La varible booleana "manual" se da como falsa
469                 case "SENSOR_DELAY_NORMAL":
470                     sensorManager.unregisterListener(this, acelerometro);
471                     sensorManager.registerListener(this, acelerometro, SensorManager.
                            SENSOR_DELAY_NORMAL);
472                     manual = false;
473                     break;
474                 case "SENSOR_DELAY_UI":
475                     sensorManager.unregisterListener(this, acelerometro);
476                     sensorManager.registerListener(this, acelerometro, SensorManager.
                            SENSOR_DELAY_UI);
477                     manual = false;
478                     break;
479                 case "SENSOR_DELAY_GAME":
480                     sensorManager.unregisterListener(this, acelerometro);
481                     sensorManager.registerListener(this, acelerometro, SensorManager.
                            SENSOR_DELAY_GAME);
482                     manual = false;
483                     break;
484                 case "SENSOR_DELAY_FASTEST":
485                     sensorManager.unregisterListener(this, acelerometro);
486                     sensorManager.registerListener(this, acelerometro, SensorManager.
                            SENSOR_DELAY_FASTEST);
487                     manual = false;
488                     break;
489                 // En el caso de elegir la tasa de muestreo manual se procede de forma
                        diferente
490                 case "Manual":
491                     if (tasaManual.getText().toString().length() == 0) {
492                         // Si no se ha introducido ningún número en la casilla
                                correspondiente a ello salta un aviso por pantalla para
                                seleccionar uno
493                         Toast.makeText(parent.getContext(), "Escriba una tasa de muestreo
                                manual ", Toast.LENGTH_LONG).show();
494                     } else {
```

```
495                         manual = true;
496                         // Se almacena en la variable "i" el valor introducido por
                                pantalla de valor de tasa manual
497                         i = Integer.parseInt(tasaManual.getText().toString());
498                         // Se necesita introducir el dato en ms así que se hace una
                                conversión
499                         tasamuestreo = (int) ((1 * Math.pow(10, 6) / i));
500                         // Se desregistra el acelerómetro
501                         sensorManager.unregisterListener(this, acelerometro);
502                         // Se registra el acelerómetro con la tasa manual
503                         sensorManager.registerListener(this, acelerometro, tasamuestreo)
                                ;}
504
505                         break;
506
507                     }default:
508             }
509             Toast.makeText(parent.getContext(), "Seleccionado: " + item, Toast.
                    LENGTH_LONG).show();
510
511
512         }
513
514         @Override
515         public void onNothingSelected(AdapterView<?> parent) {
516
517         }
518
519
520     }
```

B. CÓDIGO DE LA ACTIVITY GUARDARDATOS

CÓDIGO B.1. Archivo guardardatos.xml

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/container"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:ignore="MergeRootFrame" >

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical" >

        <Button
            android:id="@+id/newButton"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Nueva Medicion" />

        <TextView
            android:id="@+id/filenameTextview"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            />

        <TextView
            android:id="@+id/chooseNumberData"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Elegir numero de datos."
            />

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content">

            <CheckBox
                android:id="@+id/checkBox"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="Numero de datos fijo" />

            <AutoCompleteTextView
                android:id="@+id/numDatos"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:hint="Numero Datos"
                android:inputType="number" />

        </LinearLayout>

        <Button
            android:id="@+id/startButton"
```

```
54              android:layout_width="match_parent"
55              android:layout_height="wrap_content"
56              android:text="Empezar Medicion" />
57
58          <TextView
59              android:id="@+id/xyzTextview"
60              android:layout_width="wrap_content"
61              android:layout_height="wrap_content"
62              />
63
64
65          <Button
66              android:id="@+id/stopButton"
67              android:layout_width="match_parent"
68              android:layout_height="wrap_content"
69              android:text="Parar Medicion" />
70
71          <TextView
72              android:id="@+id/infoTextview"
73              android:layout_width="wrap_content"
74              android:layout_height="wrap_content"
75              />
76
77
78          <Button
79              android:id="@+id/saveButton"
80              android:layout_width="match_parent"
81              android:layout_height="wrap_content"
82              android:text="Guardar Medicion" />
83
84          <TextView
85              android:id="@+id/saveTextview"
86              android:layout_width="wrap_content"
87              android:layout_height="wrap_content"
88              />
89
90      </LinearLayout>
91  </FrameLayout>
```

CÓDIGO B.2. Archivo guardardatos.java

```
1
2   package com.example.jorge.vibtest_10;
3
4   import android.Manifest;
5   import android.content.Context;
6   import android.content.DialogInterface;
7   import android.content.pm.PackageManager;
8   import android.hardware.Sensor;
9   import android.hardware.SensorEvent;
10  import android.hardware.SensorEventListener;
11  import android.hardware.SensorManager;
12  import android.os.Environment;
13  import androidx.annotation.NonNull;
14  import androidx.core.app.ActivityCompat;
15  import androidx.core.content.ContextCompat;
16  import androidx.appcompat.app.AlertDialog;
17  import androidx.appcompat.app.AppCompatActivity;
18  import android.os.Bundle;
19  import android.text.InputType;
20  import android.util.Log;
21  import android.view.View;
```

```
22  import android.widget.ArrayAdapter;
23  import android.widget.AutoCompleteTextView;
24  import android.widget.Button;
25  import android.widget.CheckBox;
26  import android.widget.EditText;
27  import android.widget.TextView;
28  import android.widget.Toast;
29
30  import java.io.File;
31  import java.io.FileNotFoundException;
32  import java.io.FileOutputStream;
33  import java.io.FileWriter;
34  import java.io.IOException;
35  import java.util.ArrayList;
36  import java.util.Arrays;
37  import java.util.List;
38
39  public class GuardarDatos extends AppCompatActivity implements SensorEventListener,
        View.OnClickListener {
40
41
42      ↑ Véase Permisos para Guardar Datos - Página 72
43
44      private static final int REQUEST_CODE_ASK_PERMISSIONS = 1;
45      private static final String[] REQUIRED_SDK_PERMISSIONS = new String[] {Manifest.
            permission.WRITE_EXTERNAL_STORAGE };
46
47      protected void checkPermissions() {
48          final List<String> missingPermissions = new ArrayList<String>();
49          // Comprobar los permisos necesarios
50          for (final String permission : REQUIRED_SDK_PERMISSIONS) {
51              final int result = ContextCompat.checkSelfPermission(this, permission);
52              if (result != PackageManager.PERMISSION_GRANTED) {
53                  missingPermissions.add(permission);
54              }
55          }
56          if (!missingPermissions.isEmpty()) {
57              // Pedir los permisos necesarios
58              final String[] permissions = missingPermissions.toArray(new String[
                    missingPermissions.size()]);
59              ActivityCompat.requestPermissions(this, permissions,
                    REQUEST_CODE_ASK_PERMISSIONS);
60          } else {
61              final int[] grantResults = new int[REQUIRED_SDK_PERMISSIONS.length];
62              Arrays.fill(grantResults, PackageManager.PERMISSION_GRANTED);
63              onRequestPermissionsResult(REQUEST_CODE_ASK_PERMISSIONS,
                    REQUIRED_SDK_PERMISSIONS,
64                      grantResults);
65          }
66      }
67
68      @Override
69      public void onRequestPermissionsResult(int requestCode, @NonNull String
            permissions[], @NonNull int[] grantResults) {
70          switch (requestCode) {
71              case REQUEST_CODE_ASK_PERMISSIONS:
72                  for (int index = permissions.length - 1; index >= 0; --index) {
73                      if (grantResults[index] != PackageManager.PERMISSION_GRANTED) {
74                          // Salir si el permiso no se ha concedido y comunicarlo por
                                pantalla con un mensaje
75                          Toast.makeText(this, "Acaba de denegar el permiso para
                                guardar datos. Saliendo... ", Toast.LENGTH_LONG).show();
76                          finish();
77                          return;
```

```
78                      }
79                  }
80                  // Todos los permisos necesarios concedidos
81
82                  break;
83          }
84      }
85      private SensorManager sensorManager;
86      Sensor acelerometro;
87
88      private String nombrearchivo;
89      private StringBuilder dataBuffer;
90      private int lineCnt;
91
92      private int estado;
93
94      private static final int REC_inicio = 1;
95      private static final int REC_parada = 2;
96
97      //private int tasamuestro = SensorManager.SENSOR_DELAY_GAME;
98
99      private Button newButton;
100     private Button startButton;
101     private Button stopButton;
102     private Button saveButton;
103
104     private TextView nameTv;
105     private TextView xyzTv;
106     private TextView infoTv;
107     private TextView saveTv;
108
109     private CheckBox seleccionDatos;
110     //private EditText numeroDatos;
111     private AutoCompleteTextView numeroDatos;
112
113     boolean datosnumero=false;
114
115     private static final String savedDir = "/AcelerometroDatos";
116
117     @Override
118     protected void onCreate(Bundle savedInstanceState) {
119         Log.w("Jorge", "here");
120         checkPermissions();
121         super.onCreate(savedInstanceState);
122         setContentView(R.layout.activity_guardar_datos);
123
124         sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
125         acelerometro = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
126         if(acelerometro != null) {
127             sensorManager.registerListener(this, acelerometro, SensorManager.
                    SENSOR_DELAY_FASTEST);
128         }
129
130         newButton = (Button) findViewById(R.id.newButton);
131         startButton = (Button) findViewById(R.id.startButton);
132         stopButton = (Button) findViewById(R.id.stopButton);
133         saveButton = (Button) findViewById(R.id.saveButton);
134
135         nameTv = (TextView) findViewById(R.id.filenameTextview);
136         xyzTv = (TextView) findViewById(R.id.xyzTextview);
137         infoTv = (TextView) findViewById(R.id.infoTextview);
138         saveTv = (TextView) findViewById(R.id.saveTextview);
139
140         seleccionDatos = (CheckBox) findViewById(R.id.checkBox);
```

```
141             seleccionDatos.setOnClickListener(this);
142             //numeroDatos = (EditText) findViewById(R.id.numDatos);
143 
144             numeroDatos = (AutoCompleteTextView) findViewById(R.id.numDatos);
145             ArrayAdapter<String> adaptador = new ArrayAdapter<>(this, android.R.layout.
                    simple_dropdown_item_1line, NUM_MEDIDAS);
146             numeroDatos.setAdapter(adaptador);
147 
148 
149             ↑ Véase Crear una Medición - Página 73
150 
151             // Los botones aparecen inhabilitados al iniciar la activity de guardar datos
152             startButton.setEnabled(false);
153             stopButton.setEnabled(false);
154             saveButton.setEnabled(false);
155 
156             // Acciones a realizar cuando se selecciona el botón "Nueva Medición"
157             newButton.setOnClickListener(new View.OnClickListener() {
158                 @Override
159                 public void onClick(View v) {
160                     // Abre el formulario para el nombre del archivo con las mediciones
161                     displayFilenameForm(); // Línea 346
162                     // Se deshabilitan el resto de botones
163                     startButton.setEnabled(true);
164                     stopButton.setEnabled(false);
165                     saveButton.setEnabled(false);
166                 }
167             });
168 
169 
170             ↑ Véase Botón Inicio de Mediciones - Página 76
171 
172             startButton.setOnClickListener(new View.OnClickListener() {
173                 @Override
174                 public void onClick(View v) {
175                     //vacía DataBuffer y empieza a grabar datos
176                     estado = REC_inicio;
177                     dataBuffer = new StringBuilder();
178                     // Se inicializa el contador de líneas
179                     lineCnt = 0;
180                     // Se saca por pantalla en el recuadro destinado el mensaje avisando
                            de que se estan guadando datos
181                     xyzTv.setText("Guardando datos...");
182                     // Comprobación de si se ha elegido un número determinado de
                            mediciones que el usuario ha escrito algo
183                     if (numeroDatos.getText().toString().length() == 0 && datosnumero==
                            true) {
184                         // Avisa de que no se ha escrito ningún número de datos
185                         Toast.makeText(getApplicationContext(), "Escriba un numero de
                                datos", Toast.LENGTH_LONG).show();
186                     }
187 
188                     // Se activa el botón para parar mientras que el de guardar los datos
                            seguirá deshabilitado
189                     stopButton.setEnabled(true);
190                     saveButton.setEnabled(false);
191                     }
192             });
193 
194 
195             ↑ Véase Botón Parada de Mediciones - Página 77
196 
197             // Se ejecutará este método cuando se presione el botón de para medición
198             stopButton.setOnClickListener(new View.OnClickListener() {
```

```
199                 @Override
200                 public void onClick(View v) {
201                     // Comprobación de se presiona le botón antes de empezar a tomar
                            datos
202                     if (estado != REC_inicio){
203                         String msg = "Empiza a grabar datos primero";
204                         // Aviso por pantalla de que se empiece a tomar datos antes
205                         Toast.makeText(getApplicationContext(), msg, Toast.LENGTH_SHORT).
                                show();
206                         return;
207                     }
208                     // Se cambia el estado a parada
209                     estado = REC_parada;
210 
211                     // Se muestran las líneas grabadas en un cuador de texto
212                     String msg = "Grabadas" + lineCnt + "líneas de datos";
213                     infoTv.setText(msg);
214                     // Se vacía en cuadro de texto que mostraba antes "Guardando datos
                            ..."
215                     xyzTv.setText("");
216 
217                     // Se habilita el botón para guardar los datos
218                     saveButton.setEnabled(true);
219                 }
220             });
221 
222 
223             ↑ Véase Botón Guardado de Datos - Página 78
224 
225             saveButton.setOnClickListener(new View.OnClickListener() {
226                 @Override
227                 public void onClick(View v) {
228                     //Escribe el contenido de DataBuffer al archivo
229                     writeFile(); // Línea 392
230                     saveButton.setEnabled(false);
231                 }
232             });
233 
234             numeroDatos.setOnClickListener(new View.OnClickListener() {
235                 @Override
236                 public void onClick(View v) {
237                     numeroDatos.showDropDown();
238                 }
239             });
240         }
241 
242 
243         ↑ Véase Elección Número de Medidas - Página 75
244 
245         @Override
246         // Método que se activa con CheckBox al seleccionarlo
247         public void onClick(View view){
248 
249             // Si se selecciona un número fijo de mediciones
250             if (seleccionDatos.isChecked()){
251                 datosnumero=true;
252                 loguearCheckbox();
253                 // Se deshabilita el botón de parar la medición
254                 stopButton.setEnabled(false);
255             // Si el número de mediciones no tiene límite
256             } else {
257                 datosnumero=false;
258                 loguearCheckbox();
259                 stopButton.setEnabled(true);
```

```
260             }
261         }
262 
263         // Aviso de la opción seleccionada al hacer click en el recuadro
264         private void loguearCheckbox() {
265             // Texto que cambia en función de la opción seleccionada habiendo dos avisos
                    posibles
266             String s = "Estado: " + (seleccionDatos.isChecked() ? "Numero datos fijo" : "
                    Sin limite datos");
267             // Aviso por pantalla
268             Toast.makeText(this, s, Toast.LENGTH_LONG).show();
269         }
270 
271 
272         ↑ Véase onSensorChanged tomando Medidas - Página 77
273 
274         // Método que se activa cada vez que se registra un dato nuevo del sensor
275         public void onSensorChanged (SensorEvent event) {
276 
277             // Solo si se ha dado al botón de iniciar la medición y la variable booleana
                    "REC_inicio" coincide con el "estado" (ambos en verdadero) se guardan los
                     valores del acelerómetro. En este caso se añade la condición de que no
                    haya número limitado de mediciones
278             if (estado == REC_inicio && numeroDatos.getText().toString().length()==0) {
279                 // Se guardan los valores de los tres ejes en variables
280                 float x = event.values[0];
281                 float y = event.values[1];
282                 float z = event.values[2];
283                 // Delimitador de las columnas en un archivo .csv, una coma
284                 String delim = ", ";
285                 // Se crea una línea con el tiempo y los datos del acelerómetro separados
                         por comas. Se añade "\n" que se interpreta como final de la línea
286                 String line = event.timestamp + delim + x + delim + y + delim + z + "\n";
287                 // Se añade la línea al dataBuffer o memoria interna
288                 dataBuffer.append(line);
289                 // Se suma uno al contador de líneas (mediciones)
290                 lineCnt++;
291 
292             // Caso para un número definido de mediciones
293             } else if (estado == REC_inicio) {
294                 // Se guardan los daros de la misma forma que el caso anterior
295                 float x = event.values[0];
296                 float y = event.values[1];
297                 float z = event.values[2];
298                 String delim = ", ";
299                 String line = event.timestamp + delim + x + delim + y + delim + z + "\n";
300                 dataBuffer.append(line);
301                 lineCnt++;
302 
303                 // Cuando se llega al número de mediciones elegido se activa esta función
304                 if (Integer.parseInt(numeroDatos.getText().toString())==lineCnt &&
                        datosnumero==true){
305                     // El estado pasa a parado
306                     estado=REC_parada;
307                     // Se muestra un mensaje por pantalla de que se ha parado de tomar
                            datos
308                     Toast.makeText(getApplicationContext(), "Parada toma de datos, ahora
                            guardar", Toast.LENGTH_LONG).show();
309                     // El recuadro de texto debajo del boton de parar la medicioón
                            muestra cuantas líneas de datos o mediciones se han grabado en la
                             memoria interna
310                     String msg = "Grabadas " + lineCnt + " líneas de datos";
311                     // Se habilita el botón para guardar los datos
312                     saveButton.setEnabled(true);
```

```
313                     infoTv.setText(msg);
314                     // Se vacía el recuadro que texto que mostraba "guardando datos..."
315                     xyzTv.setText("");
316                 }
317 
318             }
319         }
320 
321 
322         public void onAccuracyChanged (Sensor sensor, int accuracy){
323         }
324 
325 
326         // Métodos onResume(), onPause() y onDestroy() necesarios por si se sale de la
                activity para no dejar procesos funcionando que ya no se necesitan
327         protected void onResume () {
328             super.onResume();
329             sensorManager.registerListener(this, acelerometro, SensorManager.
                    SENSOR_DELAY_FASTEST);
330 
331         }
332         @Override
333         protected void onPause() {
334             super.onPause();
335             sensorManager.unregisterListener(this,acelerometro);
336         }
337         @Override
338         protected void onDestroy() {
339             sensorManager.unregisterListener(this,acelerometro);
340             super.onDestroy();
341         }
342 
343 
344         ↑ Véase Formulario Nombre de Medición - Página 74
345 
346         private void displayFilenameForm(){
347             //AlertDialog o alerta por pantalla para introducir el nombre.
348             AlertDialog.Builder builder = new AlertDialog.Builder(this);
349             // Título de la alerta
350             builder.setTitle("Nombre del archivo (no añadir .csv)");
351 
352 
353             // Inicializar la entrada de texto
354             final EditText input = new EditText(this);
355             // Especificar el tipo de entrada
356             input.setInputType(InputType.TYPE_CLASS_TEXT);
357             builder.setView(input);
358 
359             // Inicializar los botones con una nueva medición y nombre de esta
360             builder.setPositiveButton("OK", new DialogInterface.OnClickListener() {
361                 @Override
362                 public void onClick(DialogInterface dialog, int which) {
363                     // Fecha de la medición en ms
364                     Long tslong = System.currentTimeMillis()/1000;
365                     String ts = tslong.toString();
366                     // Se define el nombre del archivo como el nombre dado por el usuario
                            más la fecha y el formato de archivo (.csv)
367                     nombrearchivo = input.getText().toString() + "_" + ts + ".csv";
368                     // Se incluye el nombre elegido por pantalla
369                     nameTv.setText("Nombre: " + nombrearchivo);
370                     // El resto de cuadros de texto de la activity se vacían
371                     xyzTv.setText("");
372                     infoTv.setText("");
373                     saveTv.setText("");
```

```
            }
        });

        // Se define lo que pasa si se cancela la elección del nombre
        builder.setNegativeButton("Cancelar", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                dialog.cancel();
            }
        });
        builder.show();
    }



    ↑ Véase Guardado de los Datos - Página 79

    // Método para guardar datos en el archivo .csv final
    private void writeFile(){

        File root = Environment.getExternalStorageDirectory();
        // Se crea el directorio en la dirección principal del almacenamiento con el
            nombre "AcelerometroDatos"
        File dir = new File(root.getAbsolutePath(), "AcelerometroDatos");

        // Comprobación de si existe el directorio
        if (!dir.exists()) {
            dir.mkdirs();
        }

        // Se crea un archivo dentro del directorio con el nombre que se eligió
            inicialmente
        File file = new File(dir, nombrearchivo);

        try {
            // Instancia de donde se guardan los datos previamente
            FileOutputStream fos;
            // Se saca el peso en una variable
            byte[] data = dataBuffer.toString().getBytes();
            fos = new FileOutputStream(file);
            // Se escriben los datos en el archivo
            fos.write(data);
            fos.flush();
            fos.close();
            Toast.makeText(this, "¡Guardado!", Toast.LENGTH_SHORT).show();

            // Tamaño del archivo
            String size;
            // Tamaño del archivo en bytes o kilobytes si es de más de 1024 bytes
            if (data.length < 1024)
                size = data.length + " B";
            else
                size = (data.length / 1024) + " KB";
            // Mensaje de texto con el tamaño del archivo
            String msg = "Archivo guardado en " + nombrearchivo + "\nTamaño: " + size
                ;
            saveTv.setText(msg);
        }catch (FileNotFoundException e) { e.printStackTrace();
        }catch (IOException e) { e.printStackTrace();}
    }

    public static String[] NUM_MEDIDAS ={
            "256",
            "512",
```

```
435                 "1024",
436                 "2048",
437                 "4096"
438         };
439     }
```

C. CÓDIGO DE LA ACTIVITY FFT

CÓDIGO C.1. Archivo guardardatos.xml

```
<com.example.jorge.vibtest_10.ScrollViewBloqueo
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/ScrollBloqueo"
    android:layout_height="wrap_content"
    android:layout_width="match_parent">
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".FFT"
    android:orientation="vertical">

    <Button
        android:id="@+id/btnChooser"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Seleccionar archivo" />

    <TextView
        android:id="@+id/txtPath"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textStyle="italic"/>

    <Button
        android:id="@+id/importData"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Importar archivo" />

    <TextView
        android:id="@+id/numData"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textStyle="italic"/>

    <Button
        android:id="@+id/analData"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Analizar datos FFT" />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">

        <RadioGroup
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:id="@+id/ejeGroup"
            android:orientation="horizontal">
            <RadioButton
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
```

```
                android:text="EjeX"
                android:id="@+id/radioButton_ejeX"
                android:checked="false" />

            <RadioButton
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="EjeY"
                android:id="@+id/radioButton_ejeY"
                android:checked="true" />

            <RadioButton
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="EjeZ"
                android:id="@+id/radioButton_ejeZ"
                android:checked="false" />
        </RadioGroup>


    </LinearLayout>

    <TextView
        android:id="@+id/tasaMedia"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"/>

    <TextView
        android:id="@+id/varianza"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"/>

    <lecho.lib.hellocharts.view.LineChartView
        android:id="@+id/fftchart"
        android:layout_width="match_parent"
        android:layout_height="364dp" />

    <Switch
        android:id="@+id/switch1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:checked="false"
        android:text="Broqueo pantalla para ampliar"
        android:showText="true"
        android:textOff="NO"
        android:textOn="SI"
        android:background="@drawable/back"
        />

    <lecho.lib.hellocharts.view.LineChartView
        android:id="@+id/temporal"
        android:layout_width="match_parent"
        android:layout_height="364dp" />

</LinearLayout>
</com.example.jorge.vibtest_10.ScrollViewBloqueo>
```

CÓDIGO C.2. Archivo guardardatos.java

```
1  package com.example.jorge.vibtest_10;
2
3  import androidx.annotation.NonNull;
4  import androidx.appcompat.app.AppCompatActivity;
5  import androidx.core.app.ActivityCompat;
6  import androidx.core.content.ContextCompat;
7  import androidx.fragment.app.Fragment;
8  import lecho.lib.hellocharts.gesture.ZoomType;
9  import lecho.lib.hellocharts.listener.LineChartOnValueSelectListener;
10 import lecho.lib.hellocharts.model.Axis;
11 import lecho.lib.hellocharts.model.Line;
12 import lecho.lib.hellocharts.model.LineChartData;
13 import lecho.lib.hellocharts.model.PointValue;
14 import lecho.lib.hellocharts.view.LineChartView;
15
16 import android.Manifest;
17 import android.content.pm.PackageManager;
18 import android.graphics.Color;
19 import android.os.Bundle;
20 import android.os.Environment;
21 import android.util.Log;
22 import android.view.LayoutInflater;
23 import android.view.View;
24 import android.view.ViewGroup;
25 import android.widget.Button;
26 import android.widget.CompoundButton;
27 import android.widget.EditText;
28 import android.widget.RadioButton;
29 import android.widget.RadioGroup;
30 import android.widget.Switch;
31 import android.widget.TextView;
32 import android.widget.Toast;
33
34 import com.codekidlabs.storagechooser.Content;
35 import com.codekidlabs.storagechooser.StorageChooser;
36 import com.github.mikephil.charting.charts.LineChart;
37 import com.github.mikephil.charting.data.Entry;
38 import com.github.mikephil.charting.data.LineData;
39 import com.github.mikephil.charting.data.LineDataSet;
40 import com.github.mikephil.charting.interfaces.datasets.ILineDataSet;
41
42 import org.apache.commons.math3.complex.Complex;
43 import org.apache.commons.math3.transform.DftNormalization;
44 import org.apache.commons.math3.transform.FastFourierTransformer;
45 import org.apache.commons.math3.transform.TransformType;
46
47 import java.io.File;
48 import java.util.ArrayList;
49 import java.util.Arrays;
50 import java.util.List;
51 import java.util.Scanner;
52
53 public class FFT extends AppCompatActivity implements CompoundButton.
       OnCheckedChangeListener{
54
55     TextView txtPath;
56     TextView numData;
57     TextView tasamediaHz;
58     TextView variance;
59
60     private Button btnChooser;
61     private Button importData;
```

```
62      private Button analData;
63
64      private int STORAGE_PERMISSION_CODE = 23;
65
66
67      private LineChartView lineChartView;
68      private LineChartView lineChartView2;
69      int medidasSize;
70      float[] modulo;
71      float[] frecuencias;
72      float[] tiempos;
73      double[] ejefinal;
74
75
76      Complex[] complx;
77      float valid_length;
78
79      private RadioGroup ejeGroup;
80      private RadioButton ejeX;
81      private RadioButton ejeY;
82      private RadioButton ejeZ;
83
84      Switch miSwitch;
85
86      @Override
87      protected void onCreate(Bundle savedInstanceState) {
88          super.onCreate(savedInstanceState);
89          setContentView(R.layout.activity_fft);
90
91          ScrollViewBloqueo miScrollViewBloqueo = (ScrollViewBloqueo) findViewById(R.id
                .ScrollBloqueo);
92
93          txtPath = (TextView) findViewById(R.id.txtPath);
94          numData = (TextView) findViewById(R.id.numData);
95          tasamediaHz = (TextView) findViewById(R.id.tasaMedia);
96          variance = (TextView) findViewById(R.id.varianza);
97
98          btnChooser = (Button) findViewById(R.id.btnChooser);
99          importData = (Button) findViewById(R.id.importData);
100         analData = (Button) findViewById(R.id.analData);
101
102         ejeGroup = (RadioGroup) findViewById(R.id.ejeGroup);
103         ejeX = (RadioButton) findViewById(R.id.radioButton_ejeX);
104         ejeY = (RadioButton) findViewById(R.id.radioButton_ejeY);
105         ejeZ = (RadioButton) findViewById(R.id.radioButton_ejeZ);
106
107
108         importData.setEnabled(false);
109         analData.setEnabled(false);
110
111         final ArrayList<Mediciones> medidas = new ArrayList<>();
112
113         this.ejeGroup.setOnCheckedChangeListener(new RadioGroup.
                OnCheckedChangeListener() {
114             @Override
115             public void onCheckedChanged(RadioGroup group, int checkedId) {
116                 doOnEjeChanged(group, checkedId);
117             }
118         });
119
120
121         // Método que se ejecuta cuando se presiona el botón de seleccionar el
                archivo
122         btnChooser.setOnClickListener(new View.OnClickListener() {
```

```
123                 @Override
124                 public void onClick(View v) {
125                     // Se llama al método para iniciar el formulario
126                     openChooser();
127 
128                     // Deja todos los campos de texto de la pantalla vacíos
129                     medidas.clear();
130                 }
131             });
132 
133 
134              ↑ Véase Importación del archivo al dispositivo- Página 86
135 
136             importData.setOnClickListener(new View.OnClickListener() {
137                 @Override
138                 public void onClick(View v) {
139 
140                     // Camino hasta el archivo
141                     String Path = txtPath.getText().toString();
142                     // Instancia del archivo en el camino elegido
143                     File archivo = new File(Path);
144                     Scanner s = null;
145 
146                     try{
147                         // Se lee lo que contiene el archivo
148                         s = new Scanner(archivo);
149                         //Obtener datos de las mediciones mientras exista una línea
                                siguiente
150                         while (s.hasNextLine()){
151                             // Obtengo una linea del archivo
152                             String linea = s.nextLine();
153                             // Se separa cada línea en partes entre las comas
154                             String[] cortarString = linea.split(",");
155 
156                             // Nuevo objeto de la clase Mediciones
157                             Mediciones medida = new Mediciones();
158 
159                             // Se añaden los atributos del objeto medida
160                             // El tiempo es el primer corte que se hace en una coma.
                                    Atrubuto1=Tiempo
161                             medida.setTiempo(Double.parseDouble(cortarString[0]));
162                             // Atributo2=ejeX
163                             medida.setAcelX(Double.parseDouble(cortarString[1]));
164                             // Atributo3=ejeY
165                             medida.setAcelY(Double.parseDouble(cortarString[2]));
166                             // Atributo4=ejeZ
167                             medida.setAcelZ(Double.parseDouble(cortarString[3]));
168 
169                             //Añadir el objeto medida a medidas(ArrayList)
170                             medidas.add(medida);
171                         }
172 
173                         // Se rellena el cuadro de texto con el número de medidas
                                importadas
174                         numData.setText("Importadas " + medidas.size() + " medidas.");
175                         setMedidasSize(medidas.size());
176                         // Se deshabilita el botón de importar
177                         importData.setEnabled(false);
178                         // Se habilita el botón de analizar
179                         analData.setEnabled(true);
180                         System.out.println(medidasSize);
181 
182                     } catch (Exception e) {
183                         e.printStackTrace();
```

```
                } finally{
                    try {
                        if (s != null)
                            s.close();
                    } catch (Exception e2) {
                        e2.printStackTrace();
                    }
                }

            }
        });


        ↑ Véase Análisis de los datos- Página 87

        // Método que se ejecuta al pulsar el botón de ``Analizar datos FFT''
        analData.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {

                getMedidasSize();

                // Se almacena el número de medidas
                float medSize = medidasSize;

                // Se crean un vector para almacenar los datos de cada uno de los
                    atributos de mediciones del tamaño necesario vacío
                double[] acelX = new double[medidasSize];
                double[] acelY = new double[medidasSize];
                double[] acelZ = new double[medidasSize];
                double[] nanosec = new double[medidasSize];

                // Se rellena los vectores creado anteriormente
                for (int i=0; i<medidasSize; i++){

                    acelX[i]=medidas.get(i).getAcelX();
                    acelY[i]=medidas.get(i).getAcelY();
                    acelZ[i]=medidas.get(i).getAcelZ();
                    nanosec[i]=medidas.get(i).getTiempo();

                }

                // Se crea un vector que será el eje que se elija
                ejefinal = new double[medidasSize];

                // Dependiendo de el eje elegido se copia el eje relleno de datos a "
                    ejefinal"
                if(ejeX.isChecked()){
                    ejefinal = acelX.clone();
                }else if(ejeY.isChecked()){
                    ejefinal = acelY.clone();
                }else if(ejeZ.isChecked()){
                    ejefinal = acelZ.clone();
                }

                // Variable con el número de datos - Longitud actual
                int actual_length = ejefinal.length;

                // Variable con la próxima potencia de dos - Longitud válida
                setValid_length(nextPowerOf2(actual_length));  // Línea 340

                // Eje on la longitud válida (Potencia de dos)
                double[] ejepotencia2 = new double[(int) valid_length];

```

```
246                     // Se crean vectores para llenarlos después y que aparezcan en las gr
                            áficas
247                     // Módulo de la FFT. Longitud mitad a la válida por Nyquist
248                     modulo = new float[(int) (valid_length/2)];
249                     // Frecuencias de la FFT. Longitud mitad a la válida por Nyquist
250                     frecuencias = new float[(int) (valid_length/2)];
251                     // Tiempo de las medidas
252                     tiempos = new float[medidasSize];
253
254                     // Se rellena el eje con el número de datos válidos
255                     for (int i = 0; i<ejepotencia2.length; i++) {
256                         // Si la el número de medidas es menor de una potencia de dos se
                                rellena con ceros
257                         if (i < actual_length) {
258                             ejepotencia2[i] = ejefinal[i];
259                         } else {
260                             ejepotencia2[i] = 0;
261                         }
262                     }
263
264                     // Calculo de la tasa de muestreo de la medición. Media de todas
                            mediciones
265                     float puntoHz = 0;
266                     float puntoHz2 = 0;
267                     float mediaHz;
268
269                     float[] tasapunto = new float[medidasSize-1];
270
271                     for (int j=0; j<medidasSize; j++){
272                         if (j==0){
273                             tiempos[j]=0;
274                         } else {
275                             tiempos[j] = (float) ((nanosec[j] - nanosec[0]) / 1000000000)
                                    ;
276                             puntoHz = (1 / (tiempos[j] - tiempos[j - 1]));
277                             tasapunto[j-1] = puntoHz;
278                             puntoHz2 = puntoHz2 + puntoHz;
279                         }
280                     }
281                     // Se hace la media. Se divide la suma entre el número de mediciones
282                     mediaHz = puntoHz2/medidasSize;
283                     // Se muestra por pantalla la tasa de muestreo media
284                     tasamediaHz.setText("Tasa media: " + mediaHz + " Hz");
285
286
287                     // Se calcula la varianza de la tasa de refresco
288                     float varianza;
289                     float puntoVar = 0;
290                     float puntoVar2 = 0;
291                     for (int s=0; s<(medidasSize-1); s++) {
292                         puntoVar = (float) Math.pow((tasapunto[s]-mediaHz),2);
293                         puntoVar2 = puntoVar2 + puntoVar;
294                     }
295                     varianza = puntoVar2/(medidasSize-1);
296                     // Se muestra por pantalla el resultado
297                     variance.setText("Varianza: " + varianza);
298
299
300                     // Frecuencias de la gráfica de la FFT. De cero a la mitad de la
                            longitud válida por Nyquist
301                     for (int w=0; w<(valid_length/2); w++){
302                         frecuencias[w] = (mediaHz/valid_length)*w;
303                     }
304
```

```
305
306                 // Se instancia la transformada desde la librería externa
307                 FastFourierTransformer transformer = new FastFourierTransformer(
                        DftNormalization.STANDARD);
308                 try{
309                     // Se aplica la FFT y se obtiene un número complejo
310                     Complex[] complx = transformer.transform(ejepotencia2,
                            TransformType.FORWARD);
311
312                    // Se rellena los vectores de la parte real e imaginaria del
                           resultado de la FFT
313                     for (int j=0; j<complx.length/2; j++) {
314                         // Parte real del número complejo
315                         double rr = (complx[j].getReal());
316                         // Parte imaginaria del número complejo
317                         double ri = (complx[j].getImaginary());
318                         // Se calcula el módulo del número complejo
319                         modulo[j] = (float) (Math.sqrt((rr*rr) + (ri*ri)))/(
                                valid_length/2);
320                     }
321
322                 } catch (IllegalArgumentException e) {
323                     System.out.println(e);
324                 }
325
326                 setModulo(modulo);
327                 setTiempos(tiempos);
328                 setEjefinal(ejefinal);
329
330                 // Se llama al método para dibujar las gráficas
331                 drawFFTChart();
332                 }
333         });
334
335         miSwitch = (Switch) findViewById(R.id.switch1);
336         miSwitch.setOnCheckedChangeListener(this);
337     }
338
339     // Método que devuelve la potencia de dos inmediatamente superior
340     private static float nextPowerOf2(int actual_length) {
341         float nextPower=1;
342         while (actual_length > nextPower){
343             nextPower=nextPower*2;
344             System.out.println(nextPower);
345         }
346         return nextPower;
347     }
348
349     // Método para abrir el formulario tras comprobar que si se puede leer la memoria
350     public void openChooser() {
351         if (!isStorageReadable()){
352             // Se comprueba si se tienen los permisos
353             requestStoragePermission();
354         }
355         else {
356             chooser(); // Línea 406
357         }
358     }
359
360
361      ↑ Véase Permisos para acceder a la memoria- Página 83
362
363     // Comprobación para saber si ya se tiene el permiso para acceder a la memoria
364     private boolean isStorageReadable(){
```

```
365             //Obtener el estado de permisos
366             int result = ContextCompat.checkSelfPermission(this, Manifest.permission.
                    READ_EXTERNAL_STORAGE);
367 
368             //Si se obtiene el permiso devuelve un true
369             if (result == PackageManager.PERMISSION_GRANTED) {
370                 return true;
371             }
372             //Si se deniega el permiso devolver falso
373             return false;
374         }
375 
376         //Pedir permiso
377         private void requestStoragePermission() {
378 
379             if (ActivityCompat.shouldShowRequestPermissionRationale(this, Manifest.
                    permission.READ_EXTERNAL_STORAGE)) {
380                 Toast.makeText(getApplicationContext(), "Se necesita permiso para acceder
                         al almacenamiento", Toast.LENGTH_LONG).show();
381             }
382 
383             //Finalmente se pide permiso
384             ActivityCompat.requestPermissions(this, new String[]{Manifest.permission.
                    READ_EXTERNAL_STORAGE}, STORAGE_PERMISSION_CODE);
385         }
386 
387         @Override
388         public void onRequestPermissionsResult(int requestCode, @NonNull String[]
                permissions, @NonNull int[] grantResults) {
389 
390             //Comprobar el codigo
391             if (requestCode == STORAGE_PERMISSION_CODE) {
392 
393                 //Si se concede el permiso
394                 if (grantResults.length > 0 && grantResults[0] == PackageManager.
                        PERMISSION_GRANTED) {
395                     chooser();
396                 } else {
397                     Toast.makeText(this, "Acaba de denegar el permiso para acceder a
                            datos", Toast.LENGTH_SHORT).show();
398                 }
399             }
400         }
401 
402 
403         ↑ Véase Formulario para elegir archivo - Página 84
404 
405         // Se ejecuta el método para el formulario
406         private void chooser() {
407 
408             // Instncia del contenido del formulario
409             Content c = new Content();
410             // Título de la memoria interna
411             c.setInternalStorageText("Almacenamiento interno");
412             // Texto en botón de cancelar
413             c.setCancelLabel("Cancelar");
414             // Texto en botón de elegir
415             c.setSelectLabel("Elegir");
416             c.setOverviewHeading("Elija el amacenamiento");
417 
418             // Personalizacion del formulario
419             StorageChooser chooser = new StorageChooser.Builder()
420                     .withActivity(FFT.this)
421                     .withFragmentManager(getFragmentManager())
```

```
                .withMemoryBar(true)
                // Tipo de formulario - Selección de archivos
                .setType(StorageChooser.FILE_PICKER)
                .showHidden(false)
                .allowCustomPath(true)
                .withPredefinedPath(Environment.getExternalStorageDirectory().
                    toString())
                .setDialogTitle("Elige archivo .csv")
                .withContent(c)
                .build();

        chooser.show();

        // Acciones en la activity cuando se selecciona un archivo
        chooser.setOnSelectListener(new StorageChooser.OnSelectListener() {
            @Override
            public void onSelect(String path) {
                // Se vacían todos los cuadros de texto de la activity
                txtPath.setText(path);
                numData.setText("");
                tasamediaHz.setText("");
                variance.setText("");
                // Se habilita el botón para importar los datos
                importData.setEnabled(true);

            }
        });
    }


    ↑ Véase Resultado de la FFT - Página 89

    // Método para crear y rellenar las gráficas
    private void drawFFTChart (){
        //Primera gráfica

        //Se une con la representación visual en el archivo xml
        lineChartView = (LineChartView) findViewById(R.id.fftchart);

        // Cuando se selecciona un punto hay una acción
        lineChartView.setOnValueTouchListener(new ValueTouchListener());

        // Se crea una matriz para añadir los datos de la gráfica
        List<PointValue> values = new ArrayList<PointValue>();

        // Se añaden los valores de la frecuencia y módulo de la FFT
        for (int k=1; k<(valid_length/2); k++){
            values.add(new PointValue(frecuencias[k], modulo[k]));
        }

        // Se crea una línea
        Line line = new Line(values);
        // SI se toca un punto se muestra el valor del punto
        line.setHasLabelsOnlyForSelected(true);
        List<Line> lines = new ArrayList<Line>();
        line.setColor(getResources().getColor(R.color.colorPrimaryDark));
        lines.add(line);

        LineChartData data = new LineChartData();
        Axis axisX = new Axis();
        Axis axisY = new Axis().setHasLines(true);
        // Nombres de los ejes
        axisX.setName("Frecuencias");
        axisY.setName("Amplitud");
```

```
485             // Color de los ejes
486             axisX.setTextColor(Color.BLACK);
487             axisY.setTextColor(Color.BLACK);
488             // Posición de los ejes
489             data.setAxisXBottom(axisX);
490             data.setAxisYLeft(axisY);
491
492             data.setLines(lines);
493
494             lineChartView.setLineChartData(data);
495
496             //Segunda gráfica
497
498             // Se une con la parte de la gráfica en el archivo xml
499             lineChartView2 = (LineChartView) findViewById(R.id.temporal);
500
501             // Solo se puede ampliar esta gráfica en horizontal
502             lineChartView2.setZoomType(ZoomType.HORIZONTAL);
503             lineChartView2.setOnValueTouchListener(new ValueTouchListener());
504
505             // Igual que en la primera gráfica
506
507             List<PointValue> values2 = new ArrayList<>();
508
509             for (int z=0; z<medidasSize; z++){
510                 values2.add(new PointValue(tiempos[z], (float) ejefinal[z]));
511             }
512
513             Line line2 = new Line(values2);
514             line2.setHasLabelsOnlyForSelected(true);
515             line2.setHasPoints(false);
516             line2.setColor(getResources().getColor(R.color.colorPrimaryDark));
517             //line2.setStrokeWidth(Integer.parseInt("0.1dp"));
518             List<Line> lines2 = new ArrayList<>();
519             lines2.add(line2);
520
521             LineChartData data2 = new LineChartData();
522             Axis axisX2 = new Axis();
523             Axis axisY2 = new Axis().setHasLines(true);
524             axisX2.setName("Tiempo");
525             axisY2.setName("Aceleración");
526             axisX2.setTextColor(Color.BLACK);
527             axisY2.setTextColor(Color.BLACK);
528             data2.setAxisXBottom(axisX2);
529             data2.setAxisYLeft(axisY2);
530
531             data2.setLines(lines2);
532
533             lineChartView2.setLineChartData(data2);
534
535         }
536
537         private class ValueTouchListener implements LineChartOnValueSelectListener {
538
539             @Override
540             public void onValueSelected(int lineIndex, int pointIndex, PointValue value)
                     {
541                 Toast.makeText(getApplicationContext(), "Selected: " + value, Toast.
                         LENGTH_SHORT).show();
542             }
543
544             @Override
545             public void onValueDeselected() {
546                 // TODO Auto-generated method stub
```

```
547
548             }
549
550         }
551
552
553         ↑ Véase Selección Eje a Analizar- Página 87
554
555         // Método para cambiar de eje a analizar
556         private void doOnEjeChanged(RadioGroup group, int checkedId) {
557             int checkedRadioId = group.getCheckedRadioButtonId();
558             // Se marca como seleccionado el eje seleccionado
559             if(checkedRadioId== R.id.radioButton_ejeX) {
560                 Toast.makeText(this,"Seleccionado el eje X",Toast.LENGTH_SHORT).show();
561             } else if(checkedRadioId== R.id.radioButton_ejeY) {
562                 Toast.makeText(this,"Seleccionado el eje Y",Toast.LENGTH_SHORT).show();
563             } else if(checkedRadioId== R.id.radioButton_ejeZ) {
564                 Toast.makeText(this,"Seleccionado el eje Z",Toast.LENGTH_SHORT).show();
565             }
566         }
567
568
569         ↑ Véase Interruptor para bloquear la pantalla- Página 90
570
571         // Método para bloquear la pantalla
572         @Override
573         public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
574             if (isChecked) {
575                 // Movimiento desbloqueado
576                 ((ScrollViewBloqueo)findViewById(R.id.ScrollBloqueo)).setEnableScrolling(
                        false);
577             } else {
578                 // Movimiento bloqueado
579                 ((ScrollViewBloqueo)findViewById(R.id.ScrollBloqueo)).setEnableScrolling(
                        true);
580             }
581         }
582
583
584         public void setMedidasSize(int medidasSize) {
585             this.medidasSize = medidasSize;
586         }
587
588         public int getMedidasSize(){
589             return medidasSize;
590         }
591
592         public void setModulo(float[] modulo) {
593             this.modulo = modulo;
594         }
595
596         public float[] getModulo(){
597             return modulo;
598         }
599
600         public void setComplex(Complex[] complx){
601             this.complx = complx;
602         }
603
604         public void setValid_length (float valid_length){
605             this.valid_length = valid_length;
606         }
607
608         public float getValid_length(){
```

```
609             return valid_length;
610         }
611 
612         public void setTiempos(float[] tiempos){
613             this.tiempos = tiempos;
614         }
615 
616         public void setEjefinal(double[] ejefinal){
617             this.ejefinal=ejefinal;
618         }
619     }
```