UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

TRABAJO FIN DE GRADO

"DESARROLLO E IMPLEMENTACIÓN DE UN SISTEMA DE MONITORIZACIÓN DE ACTIVIDAD FÍSICA BASADO EN UNA UNIDAD INERCIAL CON SOPORTE ANDROID"

Titulación: Grado en Ingeniería en Tecnología de la Telecomunicación

Mención: Sistemas Electrónicos

Autor: Heriberto J. Díaz Luis-Ravelo

Tutor: Dr. D. Alfonso Medina Escuela

Fecha: Junio de 2015

UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

TRABAJO FIN DE GRADO

"DESARROLLO E IMPLEMENTACIÓN DE UN SISTEMA DE MONITORIZACIÓN DE ACTIVIDAD FÍSICA BASADO EN UNA UNIDAD INERCIAL CON SOPORTE ANDROID"

HOJA DE FIRMAS

Tutor **Alumno**

Fdo.: Dr. D. Alfonso Medina Escuela **Fdo.: Heriberto J. Díaz Luis-Ravelo**

Fecha: Junio de 2015

UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

TRABAJO FIN DE GRADO

“DESARROLLO E IMPLEMENTACIÓN DE UN SISTEMA DE MONITORIZACIÓN DE ACTIVIDAD FÍSICA BASADO EN UNA UNIDAD INERCIAL CON SOPORTE ANDROID”

HOJA DE EVALUACIÓN

Calificación: _____

Presidente

Fdo.:

Vocal **Secretario/a**

Fdo.: **Fdo.:**

Fecha: Junio de 2015

Tabla de contenido

Lista de Figuras

Lista de Tablas

Capítulo 1: INTRODUCCIÓN

1.1. Antecedentes

Los avances en los sistemas de navegación en las últimas décadas han ido creciendo debido al gran avance de la tecnología, como la realización de dispositivos cada vez más pequeños, gracias a la microelectrónica. Se han desarrollado dispositivos de navegación más pequeños, fiables, robustos y precisos, con un coste de fabricación relativamente pequeño. Desde hace varias décadas ya existen estos sistemas de navegación en aviones, barcos y submarinos, también han sido utilizados en programas de balística de misiles y en la exploración del espacio.

El dispositivo más importante de los sistemas de navegación inerciales es la unidad de medida inercial (IMU – Inertial Measurement Unit). Una IMU es un dispositivo electrónico que se utiliza para la medición de la velocidad, orientación y fuerzas gravitatorias de un objeto. Todo este proceso lo realiza con la combinación de acelerómetros y giroscopios. Con los datos adquiridos se analizan en un sistema de computación. Dicho sistema deberá procesar los datos para estimar la posición del objeto.

Cada día se incorpora en más dispositivos comerciales las unidades inerciales, un buen ejemplo de ello pueden ser los dispositivos móviles, Smartphones, que utilizan una IMU tanto para su localización, como para su posición de la pantalla. Las IMU tienen una infinidad de aplicaciones, en el ámbito deportivo podría ayudar a conocer la evolución del saque de un jugador de voleibol, saber si un corredor de trail ha tenido un accidente o en el ámbito social, para saber si una persona mayor se ha caído.

1.2. Objetivos

El principal objetivo de este trabajo fin de grado es el monitoreo y análisis de los datos adquiridos de una unidad de medición inercial (IMU) para su posterior aplicación. Para la realización de este trabajo se propone la integración electrónica y el desarrollo del firmware para el análisis de las señales recibidas por una IMU. Para ello se utiliza el sensor de orientación UM7-LT y un sistema electrónico basado en un procesador ARM Cortex-M3.

Los objetivos que se cumplen son los siguientes:

- Estudio de la unidad inercial a utilizar.
- Estudio del sistema electrónico basado en ARM Cortex-M3.
- Estudio de las diferentes librerías proporcionadas.
- Conocer los posibles protocolos de comunicación entre la IMU y el sistema electrónico.
- Generar las librarías del firmware para el manejo y representación de los datos de la IMU.
- Proveer una salida de control visual a través de un dispositivo basado en Android.
- Implementación de una aplicación móvil basada en Android para la representación de los datos.

1.3. Unidades Inerciales

1.3.1. Introducción

Una unidad de medida inercial es un dispositivo electrónico cuyo objetivo es obtener mediciones de velocidad, rotación y fuerzas gravitacionales. Se utilizan como componentes fundamentales en los sistemas de navegación de barcos, aviones, misiles o cualquier móvil en que sea necesario estimar estas medidas, sin la posibilidad de utilizar referencias externas o medicines directas. Una IMU está formada por dos tipos de

sensores, acelerómetros y giroscopios, en algunos casos también se incluye un tercer sensor que sería un magnetómetro.

Los sensores que componen las IMU pueden proporcionar información de uno o más ejes ortogonales (XYZ), lo que permite determinar los grados de libertad. En el caso de este TFG, la IMU a utilizar obtendrá información de los ejes XYZ de cada uno de los sensores (acelerómetro, giroscopio y magnetómetro), por lo que se tiene nueve grados de libertad (9 DOF – Degrees Of Freedom) [1].

1.3.2. Acelerómetro

El acelerómetro es un sensor que permite medir la aceleración, que es la tasa de cambio de la velocidad de un objeto. Dicha medida se puede realizar en metros por segundos al cuadrado (m/s^2) o en fuerzas G (g), esta última es la utilizada por la unidad inercial de este trabajo. Esto supone que en las medidas aparecerá la fuerza de la gravedad ($9{,}8\ m/s^2 \approx 1g$), que irá variando con la altitud o no existirá si se realiza una caída libre.

Las unidades inerciales incorporan acelerómetros integrados en silicio, utilizando la tecnología MEMS (Sistemas MicroElectroMecánicos – Microelectromechanical Systems), debido a la necesidad de reducir cada vez más el tamaño de las IMUs. La mayoría de los acelerómetros son capacitivos, y calculan la aceleración mediante el voltaje obtenido entre dos placas, una de las cuáles varía su posición dependiendo del movimiento del acelerómetro. Éstos tipos de sensores se caracterizan por tener una alta precisión en situaciones estables, en cambio producen un gran error en situaciones vibratorias o movimientos muy inestables.

1.3.3. Giroscopio

El giroscopio es un sensor que mide la tasa de rotación de un objeto, permitiendo conocer la variación del ángulo respecto al tiempo (velocidad angular), como tal, dicho sensor es una variación de un acelerómetro y un actuador. La mayor ventaja de este tipo de sensor es que responden inmediatamente al cambio, en cambio sufren de un error constante y lineal llamado bias.

El giroscopio también están basados, como los acelerómetros, en la tecnología MEMS (Sistemas MicroElectroMecánicos – Microelectromechanical Systems),es decir, integrados y de tamaño reducido. La salida de estos tipos de sensores es una voltaje, la variación de este voltaje indica en grados por segundo la velocidad angular que ha sufrido el sensor.

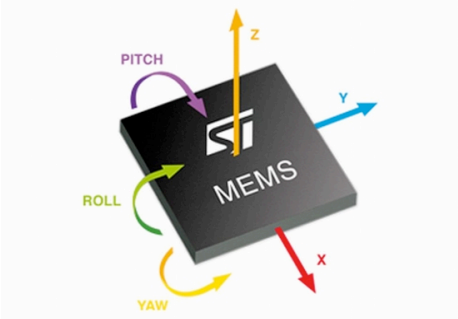

Figura 1. Movimientos fundamentales de un giroscopio y un acelerómetro.

1.3.4. Magnetómetro

El magnetómetro es una sensor capaz de medir la intensidad de campo magnético en 3 ejes. Con estas medidas se obtiene un vector de campo que da información del ángulo de azimut del dispositivo o de su posición. El campo se mide a través de unas magneto-resistencias que cambian su valor en función del campo que las atraviesa en su dirección. Se podría decir que es una brújula ampliada a 3 dimensiones, aunque éste puede distinguir giros en los ejes de roll (Alabeo) y pitch (Cabeceo). Se debe tener en cuenta que estos tipos de sensores se pueden ver influidos por variación de otros campos magnéticos de la zona.

Los magnetómetros se pueden dividir en dos tipos:

- Magnetómetros escalares: este tipo mide la intensidad total del campo magnético resultante al cuál están siendo sometidos en un punto, pero no aporta ningún dato sobre las componentes vectoriales del campo.
- Magnetómetros vectoriales: tienen la capacidad de medir la intensidad del campo magnético en una dirección particular, dependiendo de la colocación que le demos al dispositivo.

Con la información de las medidas proporcionadas por los tres tipos de sensores que componen una unidad inercial (acelerómetro, giroscopio y magnetómetro) descritos anteriormente, se puede representar la orientación de un objeto. Una de las formas principales de representar dicha orientación y la más intuitiva, son los Ángulos de Euler. Otra de las opciones para realizar la representación de un objeto, son los Cuaterniones, que son más complejos que los Ángulos de Euler pero solucionan los problemas que se producen con estos últimos. Estos dos tipos de orientación se describen a continuación.

1.3.5. Ángulos de Euler

Los Ángulos de Euler constituyen un conjunto de tres coordenadas angulares que se utilizan para representar la orientación espacial de cualquier sistema de referencia de ejes ortogonales como una composición de tres rotaciones elementales a partir de una rotación estándar conocida.

La orientación de un objeto se divide en tres rotaciones sucesivas: yaw (guiñada), pitch (cabeceo) y roll (alabeo). Estas rotaciones se realizan para transformar cualquier vector expresado en un sistema de coordenadas β, a otro sistema de coordenadas α.

La primera rotación, ψ, es el yaw. Esta rotación se realiza alrededor del eje z del sistema de coordenadas inicial que se quiere rotar, β. Tras estar rotación se obtiene un sistema de coordenadas girado cuyos ejes se designan con el superíndice ψ.

La segunda rotación, θ, el pitch. Esta rotación se realiza alrededor del eje y^{ψ} tras la anterior rotación. Tras esto, se obtiene un segundo sistema de coordenadas girado, cuyos ejes son designados con el superíndice θ.

La última rotación, ϕ, es el roll, que se realiza alrededor del eje x^{θ}. Con esto, se obtiene la rotación total existente entre los sistemas β y α. En la Figura 2 se muestran las tres rotaciones consecutivas para pasar de un sistema a otro.

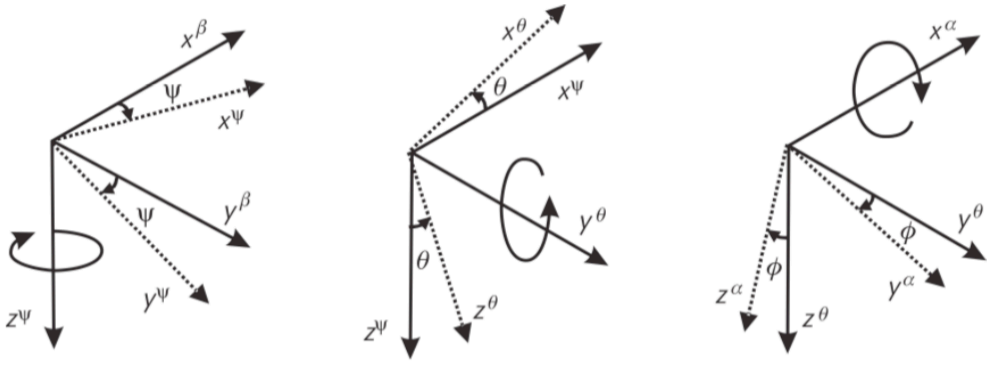

Figura 2. Ángulos de Euler.

Pero con los Ángulos de Euler se produce un problema llamado Gimbal Lock (Bloqueo de ejes), esto se produce cuando se pierde un grado de libertad. Es el caso cuando dos de los tres ejes necesarios para las rotaciones en tres dimensiones son llevados a la misma dirección. Este efecto se puede apreciar en la Figura 3.

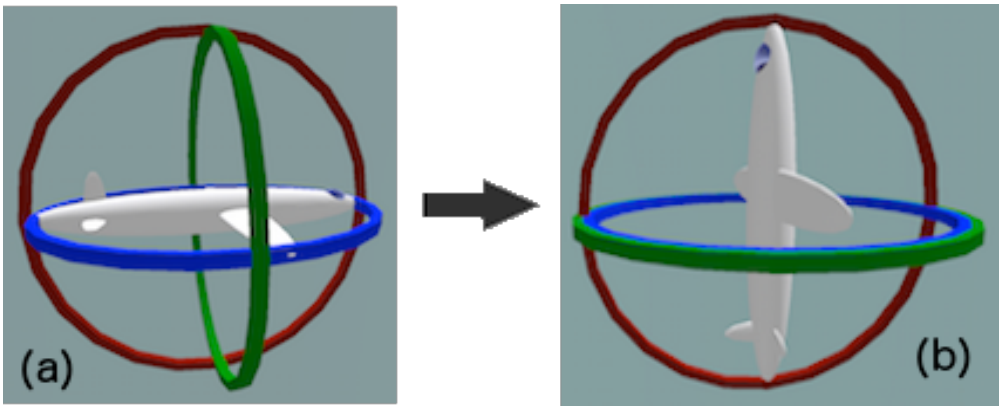

Figura 3. Gimbal Lock. (a) Todos los ángulos están perpendiculares entre sí. (b) El eje de color azul esta en paralelo con el eje de color verde y se ha perdido un grado de libertad.

1.3.6. Cuaterniones

Un cuaternión es un vector de cuatro elementos que se puede utilizar para codificar cualquier rotación en un sistema de coordenadas en tres dimensiones. Estos cuatro elementos son un elemento real y tres elementos complejos, y que pueden ser utilizados para mucho más que las rotaciones. Con los Cuaterniones se puede obtener la información necesaria para representar la posición de un sensor de orientación.

Una de las utilidades de los Cuaterniones son los gráficos por ordenador como coordenadas para las rotaciones y orientaciones. Su buen funcionamiento y facilidad de uso les permite competir con las coordenadas más habituales, como los ángulos de Euler.

1.4. Solución adoptada

En la Figura 4 se muestra el diagrama de bloques de la solución adoptada. En él se representan los diferentes componentes utilizados durante el transcurso y desarrollo del trabajo fin de grado.

Como unidad inercial se utiliza la UM7-LT, la cuál es la encargada de recoger la información de las medidas de los sensores (acelerómetro, giroscopio y magnetómetro) y enviarlas por el puerto serie. Para realizar el procesado de estas medidas, se utiliza un microprocesador basado en ARM Cortex-M3 que forma parte de la tarjeta de desarrollo HY Mini-STM32V. Dicha tarjeta ya incorpora una pantalla LCD táctil, en donde se muestra la señal procesada. Por último, para transmitir la señal de la unidad inercial hacia el dispositivo móvil Android se utiliza la placa FTDI Chip DS-UMFT311EV.

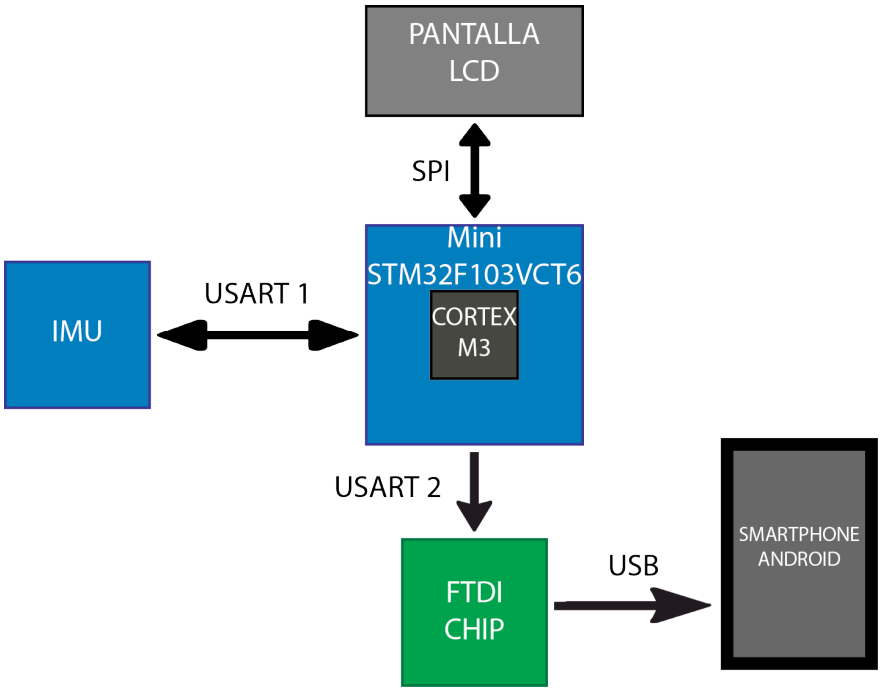

Figura 4. Diagrama de bloques de la solución.

INTRODUCCIÓN

Capítulo 2: DESCRIPCIÓN HARDWARE

2.1 Introducción

En este apartado se describen los diferentes componentes utilizados durante el transcurso y desarrollo del trabajo fin de grado. Además se realiza una breve descripción de sus principales características más relevantes para la realización del sistema a desarrollar.

2.2. UM7-LT

2.2.1. Introducción

En este apartado se describirá la unidad inercial que se utiliza para el desarrollo del trabajo fin de grado, se detalla sus especificaciones, la comunicación utilizada y los registros necesarios para desarrollar el sistema.

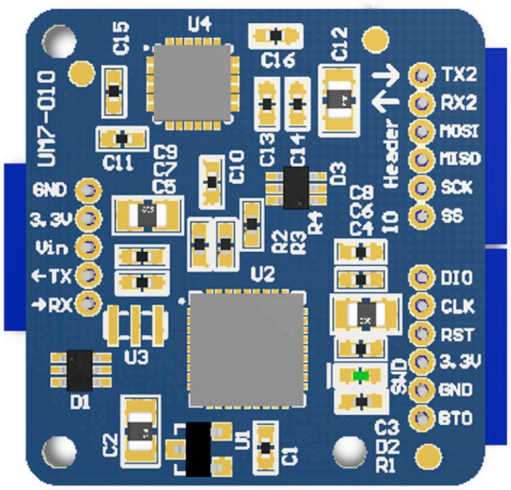

Figura 5. UM7-LT

2.2.2. Características

El UM7 [5] es un Sistema de Referencia de Posición y Orientación (AHRS – Attitude and Heading Reference System) de tercera generación que aprovecha el estado de la tecnología MEMS para mejorar el rendimiento y reducir los costes. El UM7 combina los sensores triaxiales, acelerómetro, giroscopio y magnetómetro, utilizando un sofisticado filtro Kalman extendido (EKF – Extended Kalman Filter) [6] para producir las estimaciones de posición y orientación.

Una de sus principales características es el alto rendimiento gracias a la estabilidad del giroscopio a altas temperaturas, la personalización del EKF, la posibilidad de añadir un GPS y su sincronización con los datos, y la posibilidad de una calibración personalizada para la compensación del acelerómetro, giroscopio y magnetómetro. El UM7 viene con una calibración de fábrica, que es por la cuál se ha optado. Se ha elegido esta opción, ya que a la hora de configurar cualquier parámetro de la unidad inercial se ha detectado un error en la misma , imposibilitando introducir cualquier tipo de configuración en la unidad inercial.

Otra de sus características es el bajo coste por el estado de los diseños de los dispositivos que utilizan la tecnología MEMS. También dentro de sus características hay que tener en cuenta la posibilidad de usar el tipo de transmisión de datos en formato NMEA, que son más fácil de entender , aunque en este TFG no se ha hecho uso de ello.

2.2.3. Especificaciones

Especificaciones de posición y orientación (Attitude and Heading).

Tasa de estimación EKF	500 Hz
Precisión estática para Pitch and Roll (Cabeceo y Alabeo)	+/- 1 grado
Precisión dinámica para Pitch and Roll	+/- 3 grados
Precisión estática para Yaw (Guiñada)	+/- 3 grados
Precisión dinámica para Yaw	+/- 5 grados
Repeatability*	0,5 grados
Resolución	< 0,01 grado

Tabla 1. Attitude and Heading Specifications

*Capacidad del sensor para entregar la misma salida para la misma entrada repetida, suponiendo que todas las demás condiciones son las mismas.

Especificaciones del giroscopio

Cambio de la sensibilidad frente a la temperatura	+/- 0,04% / grado C
Cambio de bias frente a la temperatura	+/- 20 grados/s de -40ºC a +85ºC
Rango dinámico	+/- 2000 grados/s
Eje X frecuencia mecánica	33 KHz nominal
Eje Y frecuencia mecánica	30 KHz nominal
Eje Z frecuencia mecánica	27 KHz nominal

Tabla 2. Gyro Specifications

Especificaciones del Acelerómetro

Cambio de la sensibilidad frente a la temperatura	+/- 0,02% / grados C
Cambio de bias frente a la temperatura (X, Y)	+/- 0,75 mg/grados C
Cambio de bias frente a la temperatura (Z)	+/- 1,50 mg/grados C
Rango dinámico	+/- 8 g

Tabla 3. Accelerometer Specifications

Especificaciones del magnetómetro

Rango dinámico	+/- 1200 μT
Facto de tolerancia de escala inicial	+/- 4%

Tabla 4. Magnetometer Specifications

Otras especificaciones

Tensión de entrada (Vin)	5 V nominal
Comunicación	3,3 V TTL UART, 3,3 V SPI bus
Velocidad de transmisión soportada	9600, 14400, 19200, 38400, 57600, 115200, 12800, 153600, 230400, 256000, 460800, 921600. Por defecto 115200
Consumo de potencia	50 mA a 5 V
Temperatura de operación	- 40ºC a + 85ºC
Dimensiones	27 mm x 26 mm x 6,5 mm
Peso	11 gramos
Tasa de salida de datos	1 Hz a 255 Hz, paquetes binarios 1 Hz a 100 Hz, paquetes NMEA
Datos de salida	Posición, Orientación (Ángulos de Euler) Posición (Cuaternión) Datos raw de acelerómetro, giroscopio y magnetómetro Datos calibrados de acelerómetro, giroscopio, magnetómetro y temperatura

Tabla 5. Other Specifications

2.2.4. Comunicación serie

El UM7 se comunica usando el puerto serie UART TTL a 3,3V, siendo configurable su velocidad de transmisión de 9600 a 921600 baudios. Por defecto viene configurada a 115200 baudios, la cuál es la velocidad de transmisión utilizada para realizar la comunicación con el microprocesador en este trabajo fin de grado.

Según como se configure el UM7-LT, se pueden transmitir paquetes binarios para mayor eficiencia con una tasa de hasta 255 Hz, paquetes NMEA para facilitar la lectura con una tasa de hasta 100 Hz o una combinación de ambos. En este sistema se ha optado por los paquetes binarios, ya que son más eficientes.

Figura 6. Diagrama de comunicación del UM7-LT.

La comunicación serie de este dispositivo funciona de la siguiente manera, cada uno de los sensores realizan sus medidas, estos son filtrados, almacenados cada uno en su registro correspondiente y enviados por el puerto serie según la frecuencia de actualización de la medida (Figura 6). Como la configuración utilizada en el UM7-LT es la que viene por defecto, éste envía cuatro tipos de tramas diferentes. Esta configuración realiza el envío de una trama del mismo tipo 5 veces por segundo (5 Hz), lo que significa que la frecuencia de envío entre una trama de un tipo y otra diferente, es de 20 Hz.

Estos cuatro tipos diferentes de tramas enviadas en paquetes binarios llevan información diferentes. Una de estas tramas lleva la información del estado de dispositivo, otra lleva las medidas realizadas de forma bruta (sin ser calibrada), la siguiente lleva las medidas pero habiendo sido procesadas y por último, lleva las información de los Ángulos de Euler. Todos estos tipos tramas son descritas más adelante, a continuación se describe que información llevan las tramas binarias en sus cabeceras.

2.2.5. Estructura de la trama

Como se ha mencionado el tipo de trama elegido para desarrollar este trabajo ha sido los paquetes binarios, la cuál utiliza la siguiente estructura:

1. La secuencia comienza con tres caracteres, ‘s’, ‘n’, ‘p’, para indicar el inicio de un nuevo paquete.

2. Un byte con el tipo de paquete (packet type – PT) describiendo la función y la longitud del paquete.

3. Un byte de dirección (Address) indicando la dirección del registro o comando.

4. Una secuencia de datos de tamaño byte (Data Bytes), la longitud es especificada en el byte PT.

5. Una suma de comprobación (checksum) de dos bytes para la detección de errores.

En la Figura 6 se puede apreciar el contenido de una trama binaria, esta trama lleva 7 bytes fijos, lo que significa que es el tamaño mínimo de envío. Estos 7 bytes son los que se indican en la Figura 7 con un número, con una X esta designado el paquete de datos ya que este valor puede variar entre 4 bytes y, para la configuración por defecto, hasta 48 bytes.

0	1	2	3	4	X	5	6
‘s’	‘n’	‘p’	Packet type (PT)	Address	Data Bytes (D0…DN-1)	Checksum 1	Checksum 0

Figura 7. Trama Binaria

El byte PT indica si el paquete es para una operación de lectura o escritura, si es una operación por lotes (batch operation) y la longitud de los datos enviados. La especificación de cada bit del byte PT se muestra a continuación en la Figura 7:

7	6	5	4	3	2	1	0
Has Data	Is Batch	BL3	BL2	BL1	BL0	Hidden	CF

Figura 8. Packet Type (PT) byte

BIT(S)	DESCRIPCIÓN
7	**Has Data:** Este bit se pone a uno si el paquete contiene datos, en caso contrario es un cero.
6	**Is Batch:** Si el paquete es una operación por lotes, este bit se pone a uno. En caso contrario en un cero.
5:2	**BL – Batch Length:** Cuatro bits que especifican la longitud de la operación por lotes.
1	**Hidden:** Si este bit esta a uno, la dirección del paquete especificada en el campo "Address" es una dirección oculta. Los registros ocultos se utilizan para almacenar la calibración de fábrica y de ajuste de los coeficientes del filtro que normalmente no necesitan ser vistos o modificados por el usuario. Este bit debe ser siempre cero para no alterar la configuración de fábrica.
0	**CF- Command Failed:** Utilizado para informar cuando un comando a fallado. Debe ponerse a cero.

Tabla 6. Descripción de los bit del Packet Type (PT)

El byte de dirección especifica que registros estará involucrado en la operación. Si se realiza una operación de lectura (Has Data = 0), la dirección especifica qué registro se leerá. Si se realiza una operación de escritura (Has Data = 1), la dirección especifica el registro en donde se escribirán los datos de la sección de datos del paquete. Para una operación de lectura/escritura por lotes (Is Batch = 1) la dirección especifica el primer registro de la operación.

Para saber la longitud de los datos hay que evaluar el byte PT. Si Has Data = 0, no hay datos en el paquete por lo que el Checksum vendrá seguido de la dirección. Si Has Data = 1, entonces la longitud de los datos dependerá de Is Batch y Batch Length.

Para una operación por lotes (Is Batch = 1), la longitud de los datos será igual a 4*Batch Length. El valor de Batch Length se refiere al número de registros y no al número de bytes. Si no se realiza una operación por lotes (Is Batch = 0), la longitud de los datos es igual a 4 bytes, que es el tamaño de un registro. En la Tabla 7 se muestra un resumen de los posibles tamaños del paquete.

HAS DATA	IS BATCH	LONGITUD DE LA SECCIÓN DE DATOS (BYTES)	TOTAL DE LA LONGITUD DEL PAQUETE (BYTES)
0	NA	0	7
1	0	4	11
1	1	4*Batch Length	7 + 4*Batch Length

Tabla 7. Resumen longitud paquete

2.2.6. Registros

Los registros que se reciban de la unidad inercial se pueden elegir, simplemente configurando la unidad inercial. Para este trabajo se ha optado por la configuración por defecto, la cuál envía los siguientes 4 paquetes:

- Primero:

  Envía la dirección 0x55 sin operación por lotes, por lo que solo se envía un registro. El registro enviado se describe en el Anexo I Tabla 14.

- Segundo:

  Envía la dirección 0x56 con una operación por lotes, en donde indica que se envían 11 registros. Por lo que se puede calcular los registro que se envían, que son desde la dirección 0x56 a la 0x60.En el Anexo I Tabla 14 se describen.

- Tercero:

  Envía la dirección 0x61 con una operación por lotes, en donde indica que se envían 12 registros. Por lo que se puede calcular los registros que se envían, que son desde la

dirección 0x61 a la 0x6C. En el Anexo I Tabla 14 se describen todos los registros involucrados.

- Cuarto:

Envía la dirección 0x70 con una operación por lotes, en donde indica que se envían 5 registros. Por lo que se deduce que los registros que se envían van desde la dirección 0x70 a la 0x74. Estos registros se describen en el Anexo I Tabla 14.

Tras analizar las diferentes tramas recibidas y haber realizado el estudio de lo que se quiere mostrar como resultado, se ha optado por sólo procesar la tercera trama que envía los registros del 0x61 al 0x6C. A continuación se explican dichos registros para saber el contenido de cada uno de ellos:

- **0x61 - DREG_GYRO_PROC_X:**

Contiene la medida actual de la velocidad angular proporcionada por el eje X del giroscopio en grados/segundos después de aplicar la calibración. Dicho dato viene dado en IEEE Floating Point con una longitud de 4 bytes.

B3	B2	B1	B0
Gyro X (IEEE Floating Point)			

Figura 9. Contenido trama 0x61.

- **0x62 - DREG_GYRO_PROC_Y:**

Contiene la medida actual de la velocidad angular proporcionada por el eje Y del giroscopio en grados/segundos después de aplicar la calibración. Dicho dato viene dado en IEEE Floating Point con una longitud de 4 bytes.

B3	B2	B1	B0
Gyro Y (IEEE Floating Point)			

Figura 10. Contenido trama 0x62.

- **0x63 - DREG_GYRO_PROC_Z:**

Contiene la medida actual de la velocidad angular proporcionada por el eje Y del giroscopio en grados/segundos después de aplicar la calibración. Dicho dato viene dado en IEEE Floating Point con una longitud de 4 bytes.

B3	B2	B1	B0
Gyro Z (IEEE Floating Point)			

Figura 11. Contenido trama 0x63.

- **0x64 - DREG_GYRO_PROC_TIME:**

Contiene el tiempo en el que el último dato del giroscopio fue medido. Dicho dato viene dado en IEEE Floating Point con una longitud de 4 bytes.

B3	B2	B1	B0
Gyro Time (IEEE Floating Point)			

Figura 12. Contenido trama 0x64.

- **0x65 - DREG_ACCEL_PROC_X:**

Contiene la medida actual del eje X del acelerómetro en m/s^2 después de aplicar la calibración. Dicho dato viene dado en IEEE Floating Point con una longitud de 4 bytes.

B3	B2	B1	B0
Accel X (IEEE Floating Point)			

Figura 13. Contenido trama 0x65.

- **0x66 - DREG_ACCEL_PROC_Y:**

Contiene la medida actual del eje Y del acelerómetro en m/s^2 después de aplicar la calibración. Dicho dato viene dado en IEEE Floating Point con una longitud de 4 bytes.

B3	B2	B1	B0
Accel Y (IEEE Floating Point)			

Figura 14. Contenido trama 0x66.

- **0x67 - DREG_ACCEL_PROC_Z:**

Contiene la medida actual del eje Z del acelerómetro en m/s^2 después de aplicar la calibración. Dicho dato viene dado en IEEE Floating Point con una longitud de 4 bytes.

B3	B2	B1	B0
Accel Z (IEEE Floating Point)			

Figura 15. Contenido trama 0x67.

- **0x68 - DREG_ACCEL_PROC_TIME:**

Contiene el tiempo en el que el último dato del acelerómetro fue medido. Dicho dato viene dado en IEEE Floating Point con una longitud de 4 bytes.

B3	B2	B1	B0
Accel Time (IEEE Floating Point)			

Figura 16. Contenido trama 0x68.

- **0x69 – 0x6C:**

Estos registros contienen los datos del magnetómetro que para este trabajo fin de grado no son necesarios, por lo que no se procesan en ningún momento.

2.3. Microprocesador

2.3.1. Introducción

Para el desarrollo de este trabajo es importante seleccionar el componente sobre el que recae la mayor parte de carga computacional del sistema, y que permite integrar los distintos periféricos para que realicen una función conjunta. Para este TFG se ha optado por un microprocesador de ST Microelectronics de la familia STM32, su modelo es STM32F103VCT6 [2], basado en ARM Cortex-M3.

2.3.2. ARM Cortex-M3

ARM se basa en una arquitectura RISC [3] (Reduced Instruction Set Computer – Ordenador con Conjunto Reducido de Instrucciones) de 32 bits. La familia Cortex es, hasta el día de hoy, la más moderna de ARM y cuenta con tres tipos de núcleos:

- Cortex-A: diseñada para aplicaciones de usuario y complejos sistemas operativos.
- Cortex-R: diseñada para sistemas de tiempo real.
- Cortex-M: diseñada para aplicaciones de bajo coste. En la Figura 16 se puede apreciar la gama de los diferentes procesadores ARM Cortex-M.

Para este trabajo se ha optado por el Cortex-M3, dicho número al final del nombre se refiere al nivel de rendimiento, donde el 1 es el más bajo y el 8 el más alto, aunque actualmente el nivel de rendimiento 3 es el más alto disponible.

El ARM Cortex-M3 está basado en la arquitectura ARMv7M [4]. Contiene 18 registros de 32 bits con una eficiente gestión de interrupciones y del consumo de potencia. El núcleo que utiliza tiene un procesamiento de instrucciones del tipo Thumb-2, permitiendo utilizar instrucciones de 16 y 32 bits.

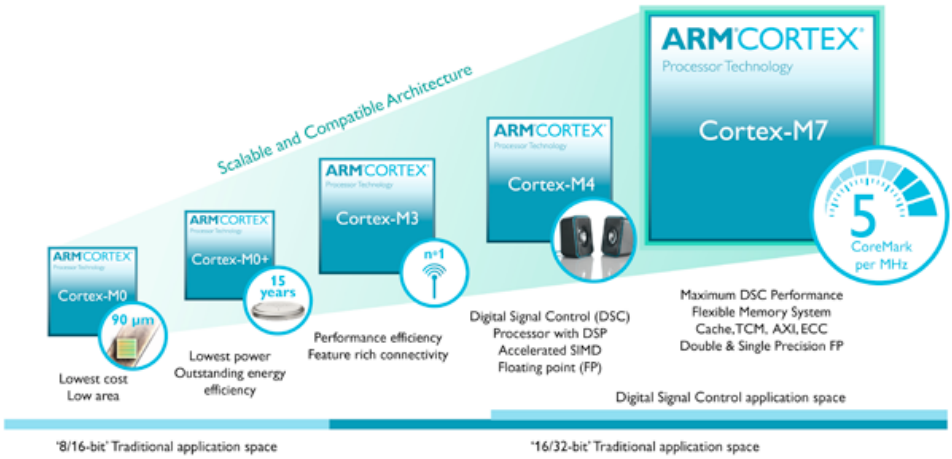

Figura 17. Gama de procesadores ARM Cortex –M.

2.3.3. Características del STM32F103VCT6

El microprocesador STM32F103VCT6 es uno de los modelos dentro de la familia STM32F103xC [2]. La diferencia entre los distintos modelos se basa en el tamaño de memoria, el número de canales del ADC (Analog to Digital Converter – Conversor Analógico a Digital), el número de puertos I/O (Input/Output – Entrada/Salida), el número de timers y el número de interfaces de comunicación.

A continuación se exponen los componentes más relevantes del STM32VCT6:

- Frecuencia CPU: 72 MHz.
- Memoria Flash: 256 Kbytes.
- SRAM: 46 Kbytes.
- 8 Timers: 4 de propósito general, 2 de control avanzado y 2 básicos.
- Interfaces de comunicación:
    - 3 SPI, dos de ellos se pueden utilizar como I^2S.
    - 2 I^2C.
    - 5 USART.
    - 1 USB.
    - 1 CAN.
    - 1 SDIO.
- 80 GPIOs.
- 3 ADC de 16 canales.
- 2 DAC (Digital to Analog Converter – Conversor Digital a Analógico) de 2 canales.

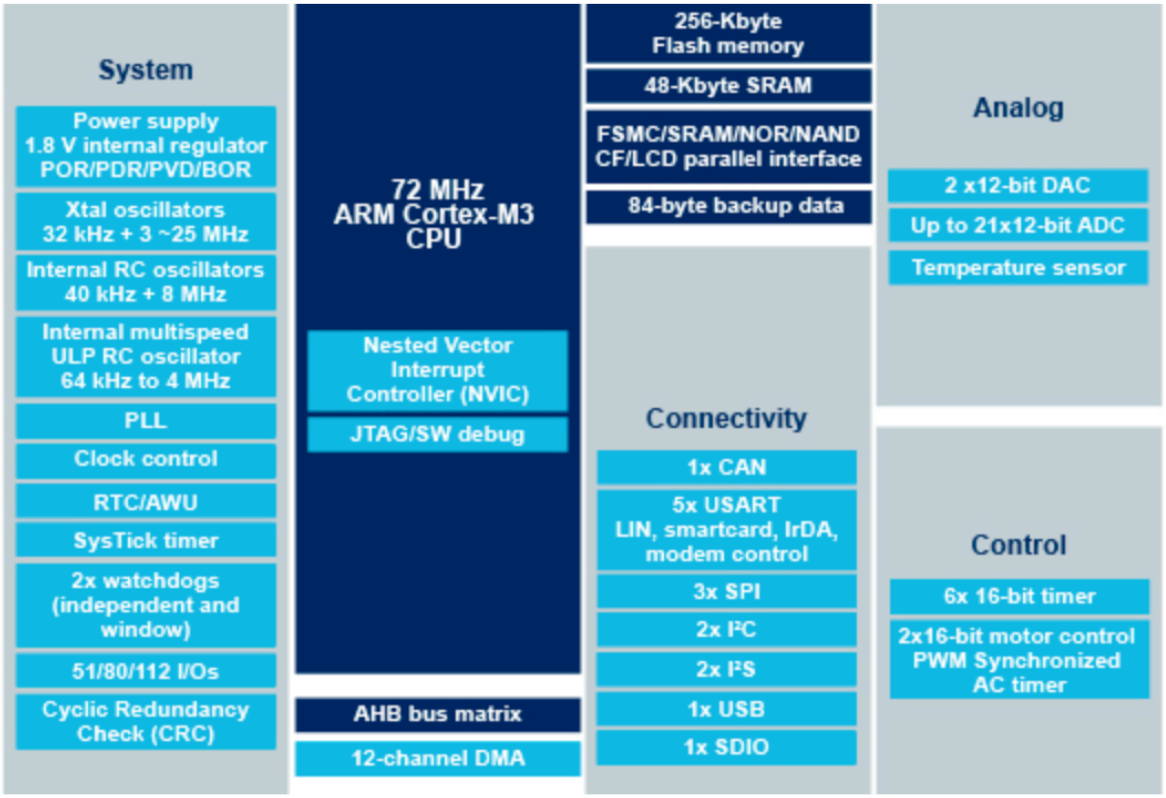

Figura 18. Diagrama de bloque del microprocesador SMT32F103xV.

Para facilitar al programador el desarrollo del software en los procesadores Cortex se ha definido un estándar llamado CMSIS (Cortex Microcontroller Software Interface Standard). En la Figura 18 se muestran las capas para el desarrollo software de microcontroladores Cortex.

CMSIS consiste en un pequeño número de funciones que proporcionan una interfaz de nivel de registro al microcontrolador subyacente. El CMSIS tiene tres grupos principales de abstracción: un grupo de funciones de periféricos del core para acceder a los periféricos de los procesadores Cortex-M3 (NVIC, SYSTICK timer, etc). Un grupo de funciones de periféricos de los dispositivos para acceder a los periféricos del microcontrolador (UART, I2C, etc.). Por último, un tercer grupo más complejo de funciones de acceso al middleware, como pilas TCP / IP.

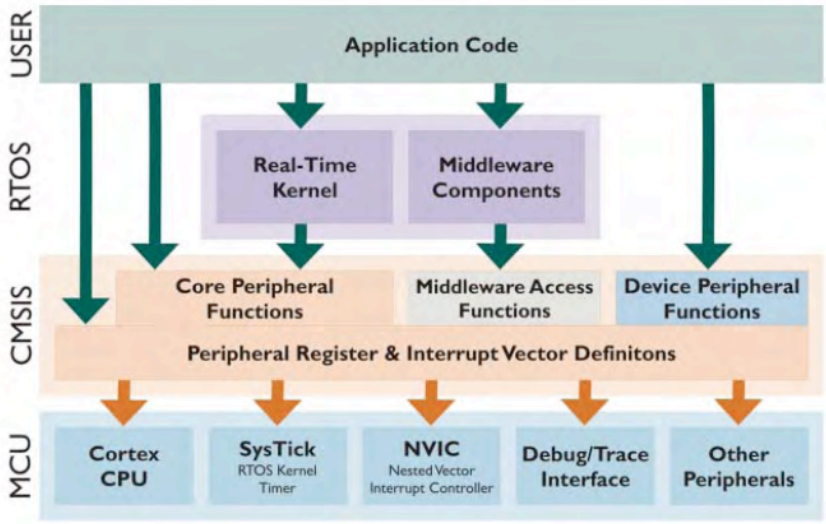

Figura 19. Capas en la programación de un microcontrolador Cortex.

2.4. Placa HY Mini-STM32V.

2.4.1. Introducción

En este apartado se describirán los diferentes componentes de la tarjeta de desarrollo utilizada en este trabajo fin de grado, la HY Mini-STM32V. Éste módulo de desarrollo es una placa de bajo coste y buenas prestaciones, basada en el microcontrolador STM32F103VCT6 descrito en el apartado anterior. A continuación de describen las diferentes características de esta tarjeta de desarrollo, resaltando los aspectos más relevantes para la realización de este trabajo fin de grado.

2.4.2. Características

La tarjeta de desarrollo HY Mini-STM32V cuenta con los siguientes componentes:

- Pantalla LCD TFT 3.2", 320 * 240 de resolución, 262.144 colores.
- Pantalla táctil resistiva, con el controlador RSM1843.
- 4 LED, 2 LED como indicadores.
- 8 MHz + 32,768 kHz clock crystal.
- Regulador lineal de5 V a 3,3 V.

- Dos botones GPIO, KEYA y KEYB.
- Un botón de RESET y un botón de Boot-loader.
- Mini-USB derecha – Interfaz de comunicación serial con PL2303.
- Mini-USB izquierda – Interfaz USB 2.0 máxima velocidad.
- Conector de tarjeta Micro SD.
- Interfaz de depuración JTAG / SWD (20 pines).
- RTC y registro de Backup de la batería (CR1220).
- Alimentación por USB.
- Tamaño de la tarjeta: 80 * 95 mm.
- Tamaño de la pantalla LCD: 62 * 95 mm.

En la Figura 20 aparece la tarjeta HY Mini-STM32V, de la cuál se pueden ver los bordes de la tarjeta y la pantalla LCD en primer término. En la parte alta hay dos bases mini-USB, pero sólo el de la izquierda es un USB 2.0 real, ya que el de la derecha es una entrada serie RS-232 que usa la tecnología USB para pasar las señales entre la placa y el ordenador.

Hoy en día ya no se venden ordenadores con salida RS-232, por este motivo, en la parte del ordenador se utiliza una solución de compromiso que consiste en convertir uno de sus USB en un puerto COM virtual, y por tanto generador de señales RS-232. Por la parte de recepción de la placa se utiliza también un USB pero que conduce la señal a un chip convertidor PL2303HX. Este chip se encarga de traducir de nuevo las señales de falso USB en las normales RS-232, las cuales ya son aprovechables directamente por el microcontrolador

En las cuatro esquinas de la tarjeta se pueden apreciar los 4 LEDs, serigrafiados de LD1 a LD4, y cuya función es la siguiente:

- LD1: pertenece al USB, se puede utilizar para pruebas, aunque su labor normal es indicar la transferencia de datos por el USB.
- LD2: LED de encendido. Se enciende siempre que la placa tenga tensión de 5 V, presente en uno de los pines del conector USB.

- LD3: LED de uso general, conectado a una salida del chip como indicador de lo que se desee.
- LD4: igual que el anterior, LED de uso general.

En la parte inferior se puede ver los 4 botones. Si se comienza por la derecha son:

- Reset: efectúa un reset al microcontrolador. Si estaba en modo de carga de programa, pasa a modo de ejecución.
- Boot: abreviatura de "Boot Loader". Apretado junto con el botón anterior, coloca la placa en modo de "carga de programa".
- KeyB: pulsador de uso general, perteneciente al puerto B, para introducir órdenes en el programa.
- KeyA: pulsador de uso general, perteneciente al puerto A, para introducir órdenes en el programa.

A parte de los elementos que descritos anteriormente, también son visibles las líneas E/S (entradas-salidas) del microcontrolador, donde de acuerdo al programa que corra en su interior, se debe conectar el resto del hardware que la placa pretenda controlar, como sensores, motores, etc.

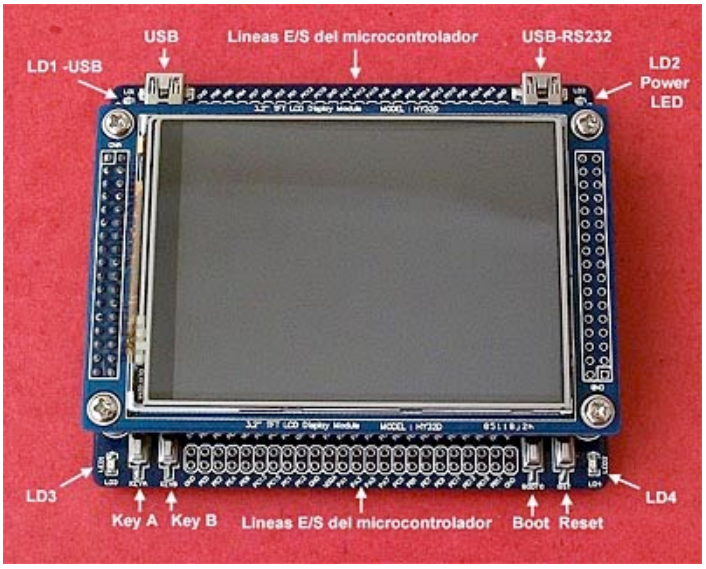

Figura 20. HY Mini-STM32V (1).

En la Figura 21 se puede apreciar la tarjeta sin la pantalla LCD colocada, en donde se distingue el microcontrolador. A la izquierda se ve el alojamiento para una pila de litio tipo DL1220, de 3V, destinada a mantener el funcionamiento del reloj interno en tiempo real. A la derecha esta el chip PL2303HX que "traduce" los datos USB en RS-232, y un conector denominado JTAG (Joint Test Action Group), y que puede servir desde la comprobación en fábrica de la integridad de la propia placa hasta como sistema de depuración de código.

La tensión de alimentación del microcontrolador, así como de la mayoría de elementos del circuito, es de 3,3 V, pero en las pruebas normales se obtiene a partir de los 5 V del USB del ordenador. La reducción la efectúa un integrado estabilizador, que tiene la forma de un pequeño transistor, situado al lado del conector JTAG.

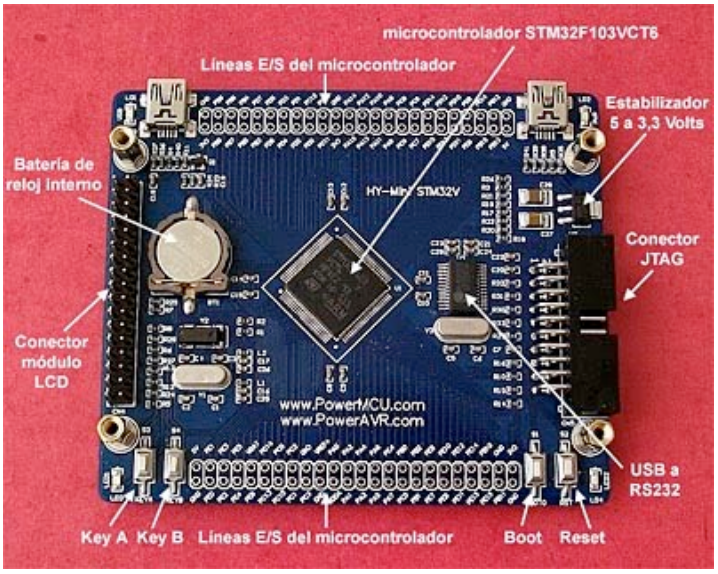

Figura 21. HY Mini STM32V (2).

Además de estos componentes, también cuenta con todas las interfaces de comunicación del microcontrolador, descritas en el apartado 2.3.3. Características del STM32F103VCT6. Para mayor detalle, en el Anexo II se encuentra el esquemático completo del módulo de desarrollo descrito en este apartado.

2.5. FTDI Chip DS-UMFT311EV

2.5.1. Introducción

El DS-UMFT311EV [7] proporciona un puente entre la tarjeta de desarrollo HY Mini-STM32V y una plataforma Android con un puerto USB, proporciona una interfaz para GPIO, UART, PWM, I2C Maestro, SPI Esclavo o SPI Maestro.

Para conectar un dispositivo a un Smartphone, Tablets, u otro dispositivo con el sistema operativo Android, a través de un puerto USB existen 2 maneras: a través de un USB OTG (On The Go) [11] o a través del modo llamado "Android Open Accessory" [8]. El USB OTG funciona muy bien con muchos dispositivos externos, pero este método tiene un requisito sustancial, el dispositivo Android debe reconocer un dispositivo externo, para ello debe tener un controlador adecuado instalado. El problema es que muchos dispositivos USB no cuentan con un controlador adecuado o el sistema está "bloqueado" (derechos de acceso limitado) por pérdida de la garantía en el dispositivo Android u otras muchas razones. Por todo esto se ha decido utilizar esta tarjeta, para evitar la pérdida de cualquier tipo de garantía.

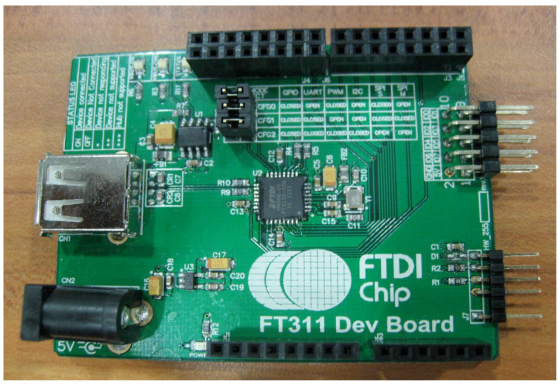

Figura 22. Tarjeta DS-UMFT311EV de FTDI Chip.

En la Figura 22 se puede apreciar la tarjeta que se está describiendo en este apartado, en ella se puede observar sus diferentes conectores y componentes. En la parte izquierda se tiene la alimentación de 5V en el conector CN2 y a su lado, el conector USB, que se utiliza para comunicarse con el dispositivo Android. En la parte inferior están los conectores J1 y

J2, que no tienen una función específica. Si se continua bordeando la placa, en la parte derecha se tiene el conector J7, que se puede usar para reprogramar el dispositivo, a su lado se encuentra el conector CON1, el cuál según la configuración seleccionada se utilizaría como puerto de comunicación. En la parte superior se encuentran los conectores J3, J4, J5 y J6, los cuales, como el puerto CON1, según la configuración se utilizan para un tipo de comunicación u otra. Para acabar se tiene en el centro de la tarjeta el microcontrolador FT311D-32Q1C y justo encima de él los tres jumpers de selección de interfaz de comunicación.

2.5.2. Características

La tarjeta DS-UMFT311EV está basada en un único chip USB Android Host FT311D IC, con seis tipos de interfaces de comunicación seleccionables manualmente a través de 3 jumpers, las cuales son:

- GPIO con siete líneas.
- UART básica.
- PWM con 4 canales.
- I2C Maestro.
- SPI Esclava soportando modos 0, 1, 2 y 3 con elección de MSB (Most Significant Bit – Bit Más Significativo) o LSB (Least Significant Bit – Bit Menos Significativo).
- SPI Maestra soportando modos 0, 1, 2 y 3 con elección de MSB (Most Significant Bit – Bit Más Significativo) o LSB (Least Significant Bit – Bit Menos Significativo).

La tarjeta es adecuada para cualquier tipo de plataforma Android que soporten Android Open Accessory Mode [8]. Normalmente este modo es soportado por dispositivos Android con versiones 3.1 en adelante, aunque algunas plataforma lo soportan desde la versión 2.3.4.

Cuenta con un puerto USB Host estándar para conectar con dispositivos Android USB esclavos, este puerto USB es compatible con USB 2.0. Además cuenta con un pin para indicar si a surgido un error en el puerto USB.

A continuación de describirán las funciones de los componentes del módulo de desarrollo FT311D, los cuáles se muestran en la Figura 23.

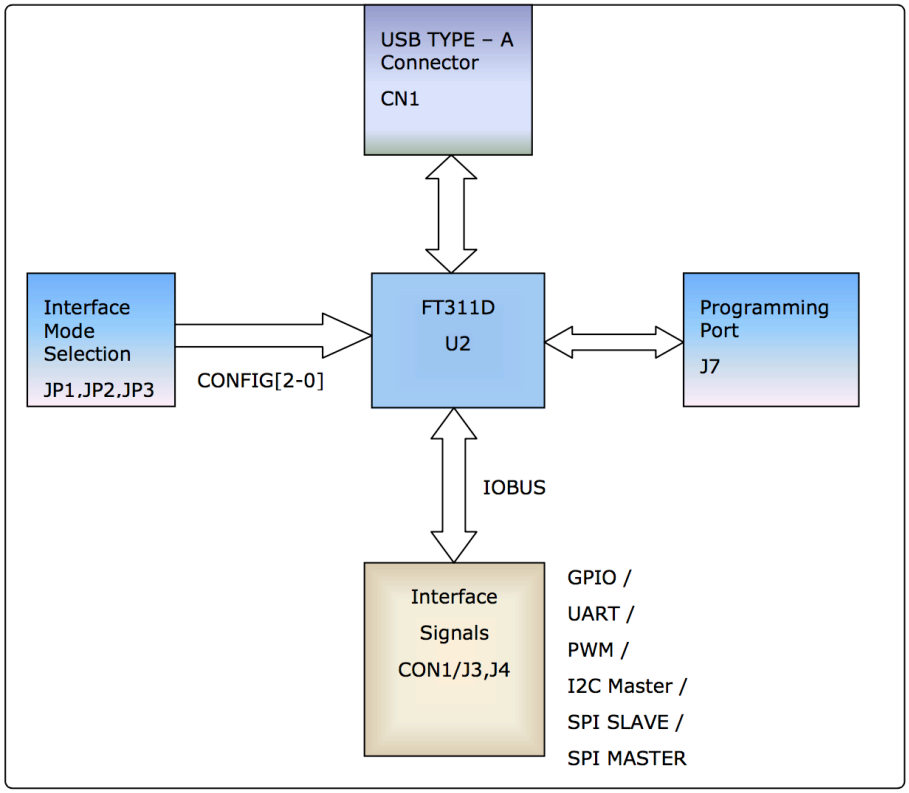

Figura 23. Diagrama de bloques del módulo FT311D.

- Modo de selección de interface (Interface Mode Selection): ésta selección se realiza manualmente con tres jumpers (JP1, JP2, JP3), en la Tabla 8 se muestra las diferentes configuraciones posibles.

- USB Host: puerto USB Host CN1 que se utiliza para conectar el dispositivo Android Open Accessory. El puerto USB Host no admite otras clases de dispositivos USB.

- Señales de Interfaz (Interface Signals): las señales de interfaz CON1 se basan en el modo seleccionado por el modo de selección de la interfaz.

- Puerto de programación (Programming Port): el puerto de programación J7 se utiliza para reprogramar el dispositivo FT311D con un nuevo archivo ROM. El dispositivo no se suele reprogramar ya que viene listo para usar.

Modos de Interfaz	Selección del Modo de Interfaz		
	JP1	JP2	JP3
GPIO	0	0	0
UART	0	0	1
PWM	0	1	0
I2C Master	0	1	1
SPI Slave	1	0	0
SPI Master	1	0	1

Tabla 8. Modos de selección de Interfaz.

Para el desarrollo de este trabajo fin de grado se utiliza el puerto UART para la transmisión con la tarjeta HY Mini-STM32V. Tras la selección de este puerto, siguiendo los pasos mencionados, las señales de la UART que se utilizarán en este sistema estarán disponible en el conector CON1 y J3, como se muestra en la Tabla 9.

NOMBRE SEÑAL	CONECTOR CON1	CONECTOR J3	TIPO E/S	DESCRIPCIÓN
UART_TXD	CON1-10	J3-3	Salida	Transmisión de datos
UART_RXD	CON1-9	J3-4	Entrada	Recepción de datos

Tabla 9. Señales de la UART.

Para mayor detalle de la estructura del módulo de desarrollo FT311D, en el Anexo III se encuentra el esquemático completo de dicha tarjeta.

Capítulo 3: INTEGRACIÓN ELECTRÓNICA

3.1. Introducción

En este capítulo se describe como se han realizado las diferentes conexión entre los distintos dispositivos necesarios para el desarrollo del sistema y las interfaces utilizadas para la comunicación entre dispositivos.

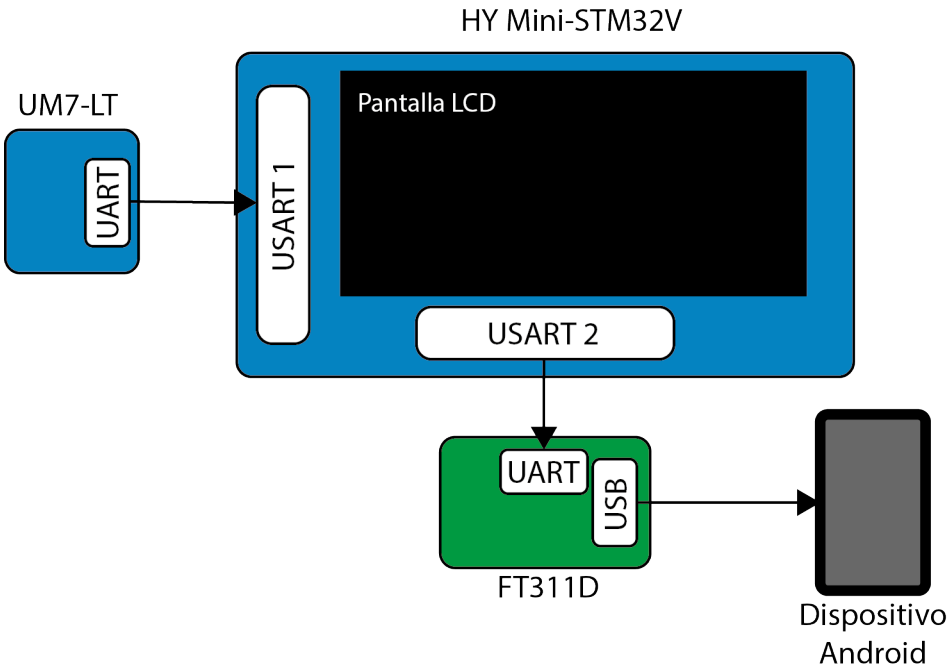

Figura 24. Diagrama de conexión.

3.2. Conexión entre HY Mini-STM32V y UM7-LT

Para la comunicación entre la tarjeta de desarrollo HY Mini-STM32V y la unidad inercial UM7-LT se ha optado por una comunicación serie a través de la UART. De la tarjeta HY Mini-STM32V se ha escogido la USART1 que se encuentra en los pines del GPIOA PA9 (TXD) y PA10 (RXD). El esquemático de la tarjeta HY Mini-STM32V se

puede ver en [9]. Por parte de la unidad inercial se ha optado por la única salida UART, la cual se conecta a través de un conector JST ZHR-5 [10].

Para una mejor maniobrabilidad con la unidad inercial, se ha colocado en medio de la conexión IMU-STM32 un conector multipolar de 6 pines, para poder colocar una extensión de esta conexión en cualquier momento sin tener que realizar mayores cambios que no sea el de conectar y desconectar.

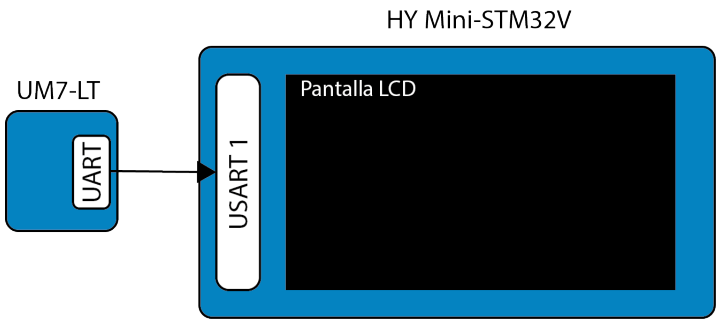

Figura 25. Diagrama conexión IMU-STM32.

3.2.1. Componentes

Para realizar este sistema se han utilizado los componente existentes en la división de Equipos y Sistemas de Comunicación del IUMA. Dichos componentes se nombran a continuación:

- Tarjeta HY Mini-STM32V. Esta tarjeta se muestra en la Figura 26 y es la utilizada para el envío de las señales, la cuál ha sido descrita en el apartado 2.4. Placa HY Mini-STM32V.

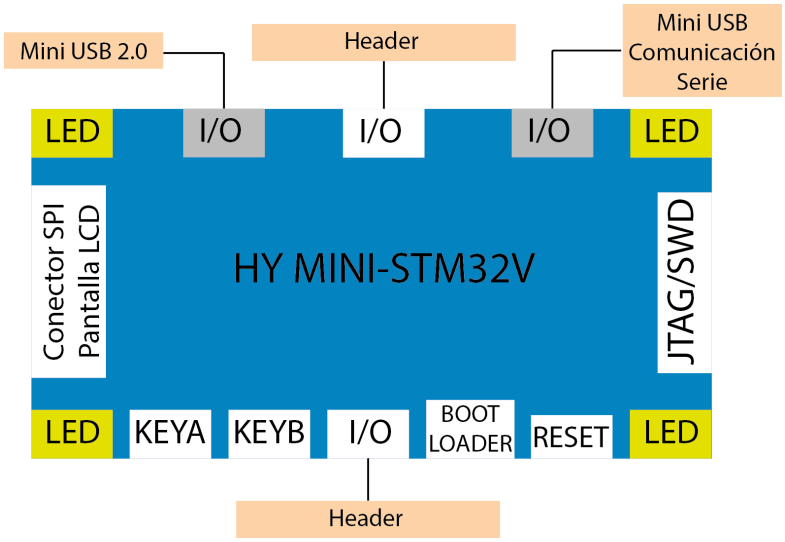

Figura 26. Diagrama de la tarjeta HY Mini-STM32V.

- Unidad inercial UM7-LT. En la Figura 27 se muestra el diagrama de esta unidad inercial, cuyo conector utilizado para la comunicación con la tarjeta HY Mini-STM32V es el situado a la izquierda. Como destacado esta unidad inercial necesita un conector especial para realizar esta comunicación.

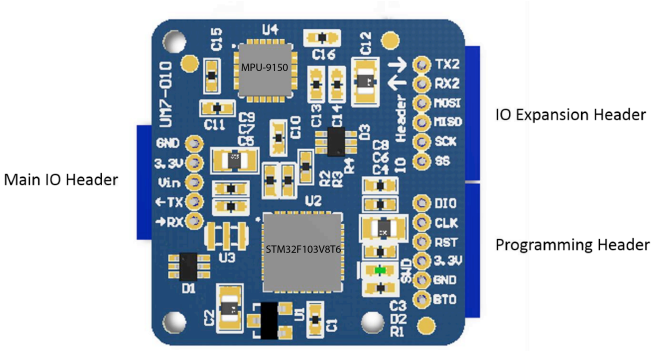

Figura 27. UM7-LT.

3.3. Conexión entre HY Mini-STM32V y FTDI Chip DS-UMFT311EV

Para la comunicación entre la tarjeta de desarrollo HY Mini-STM32V y la tarjeta FTDI Chip DS-UMFT311EV se ha optado por una comunicación serie a través de la UART. De la tarjeta HY Mini-STM32V se ha escogido la USART2 que se encuentra en los pines del GPIOA PA2 (TXD) y PA3 (RXD). El esquemático de la tarjeta HY Mini-STM32V se

puede ver en [8]. Por parte de la tarjeta FTDI Chip DS-UMFT311EV se ha optado por la única salida UART, la cual se conecta a través del conector J3 o también se puede realizar a través del CON1, para ver estos conectores se puede ver en [7]. Para mayor comodidad se crearon unos cables con conectores hembras para conectarlos a ambas UART.

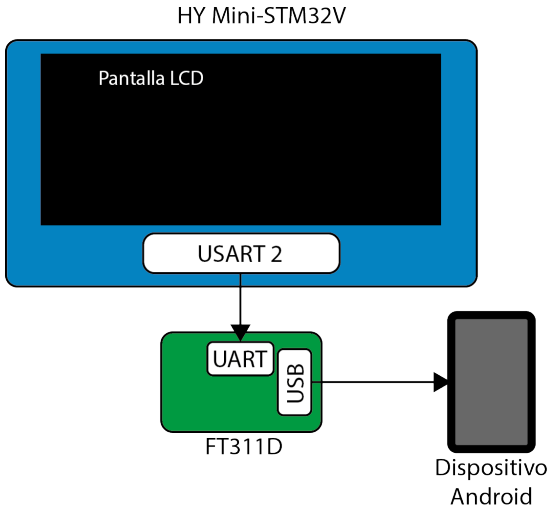

Figura 28. Diagrama conexión STM32-FT311D.

3.3.1. Componentes

Para realizar este sistema se han utilizado los componente existentes en la división de Equipos y Sistemas de Comunicación del IUMA. Dichos componentes se nombran a continuación:

- Tarjeta HY Mini-STM32V. Esta tarjeta se muestra en la Figura 29 y es la utilizada para el envío de las señales, la cuál ha sido descrita en el apartado 2.4. Placa HY Mini-STM32V.

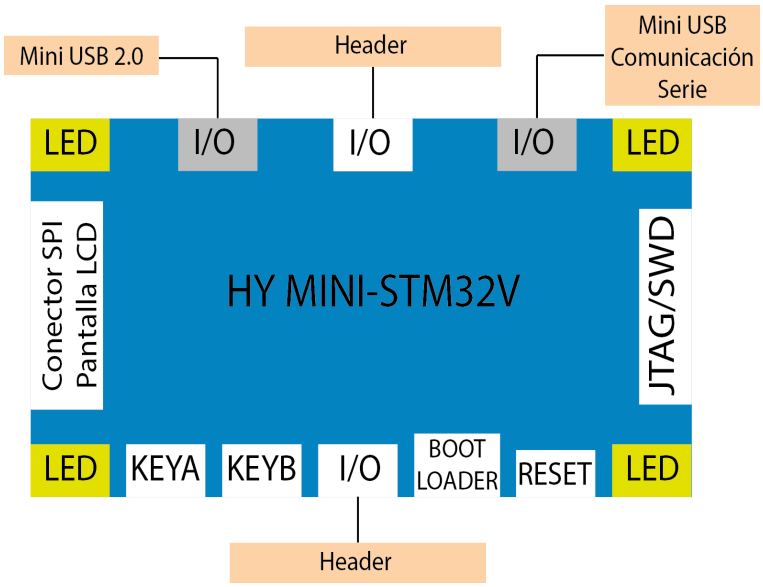

Figura 29. Diagrama de la tarjeta HY Mini-STM32V.

- Tarjeta FTDI Chip FT311D. Esta tarjeta, como se muestra en la Figura 30, recibe las señales de la tarjeta HY Mini-STM32V y las envía al dispositivo Android por su puerto USB para que éste las represente.

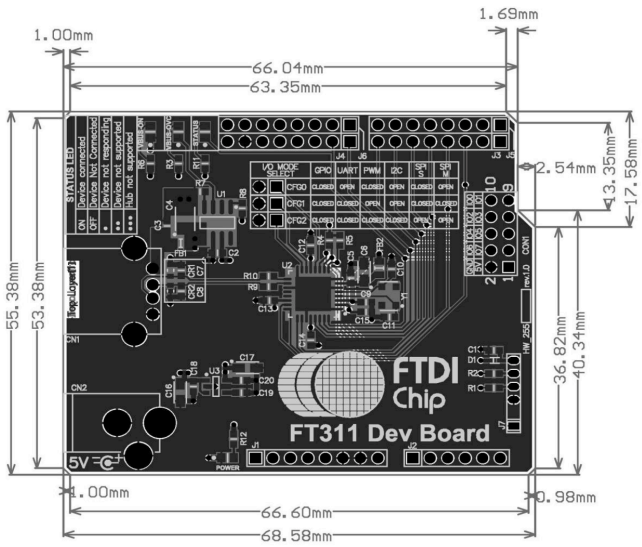

Figura 30. Diagrama de la tarjeta FTDI Chip FT311D.

3.4. Montaje prototipo final

A continuación en la Figura 31 se muestra el montaje de todas la conexiones necesarias para el desarrollo de este sistema. Se puede apreciar el extensor creado para la conexión entre la unidad inercial y la tarjeta HY Mini-STM32, además de la conexión de las interfaces UART tanto del FT311D como del HY Mini-STM32. También destacar la conexión para la alimentación de la palca HY Mini-STM32V a través del puerto mini-USB y de la tarjeta FT311D por su puerto de alimentación.

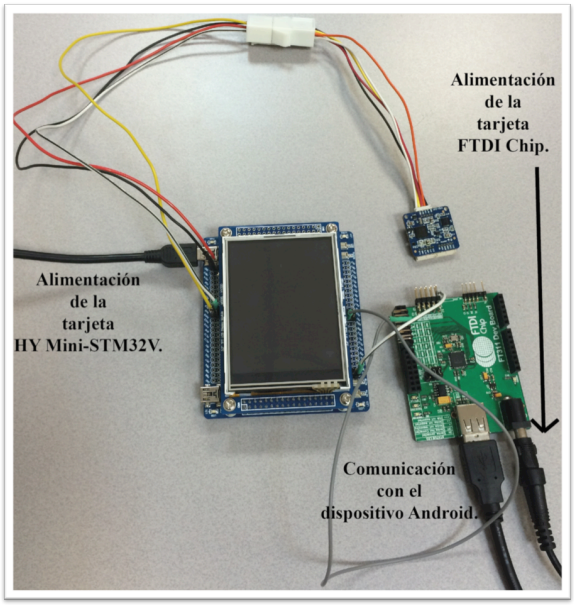

Figura 31. Conexión de todo el sistema.

Capítulo 4: DESARROLLO SOFTWARE

4.1. Introducción

En este capítulo se explica el proceso de diseño de un firmware capaz de cumplir con los requisitos funcionales mencionados en el apartado 1.4. Solución adoptada. Dado que el código del este firmware se incluye en los anexos, el propósito de esta sección es describir el trabajo realizado y comentar sus particularidades.

4.2. Firmware de Mini-STM32F103VCT6

4.2.1. Estructura del código

Para el desarrollo del firmware se ha elegido como lenguaje de programación el lenguaje C, ya que se considera que es perfecto entre las características de alto y bajo nivel. Además es el lenguaje más usado para la programación en sistemas empotrados y está soportado por la mayoría de los fabricantes de microcontroladores.

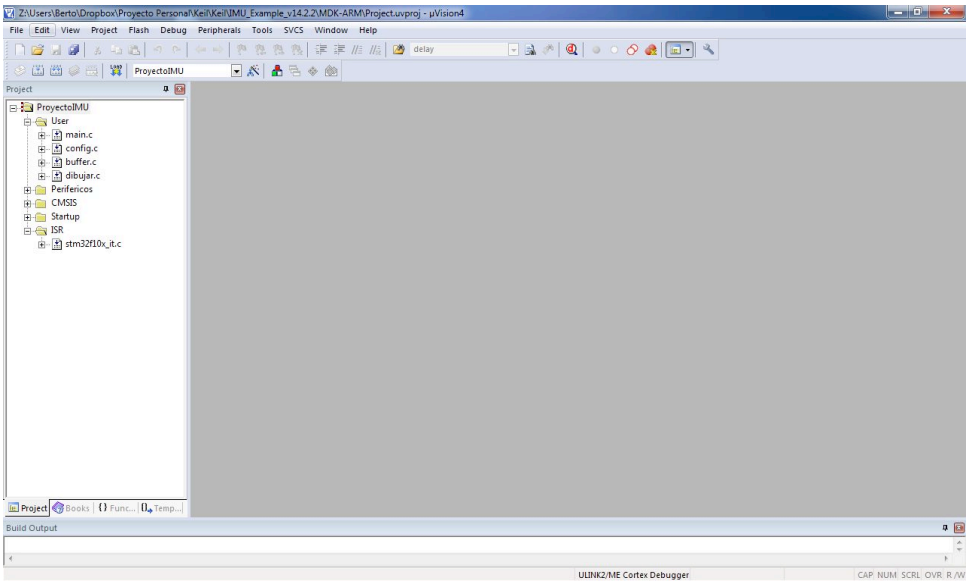

Figura 32. Compilador Keil.

Para entender mejor la estructura del firmware se han separado las librerías por categorías, de esta forma se simplifica el acceso y manejo de los distintos archivos y sus funciones. Dicha estructura puede verse reflejada en la Figura 33 donde se muestra la estructura de ficheros del Keil (Figura 32), que se organizan en las siguientes secciones:

- **User**

En esta sección se encuentran todos los archivos principales del desarrollo del firmware, entre ellos está el archivo *main.c*, que contiene el programa principal del firmware.

- **Periféricos**

Esta segunda sección contiene todas las librerías del fabricante, que controlan los distintos periféricos del sistema.

- **Startup**

En esta sección, se incluyen los archivos necesarios para la correcta inicialización del microprocesador.

- **ISR**

Aquí se incluye la librería encargada de gestionar las rutinas de servicio de las posibles fuentes de interrupción del sistema. Estas interrupciones son necesarias para permitir la recepción de tramas a través de la USART1.

En los siguientes apartados, se explicarán las librerías relativas a la funcionalidad propia del firmware, es decir, las que contienen las funciones de comunicación y de procesado de señales.

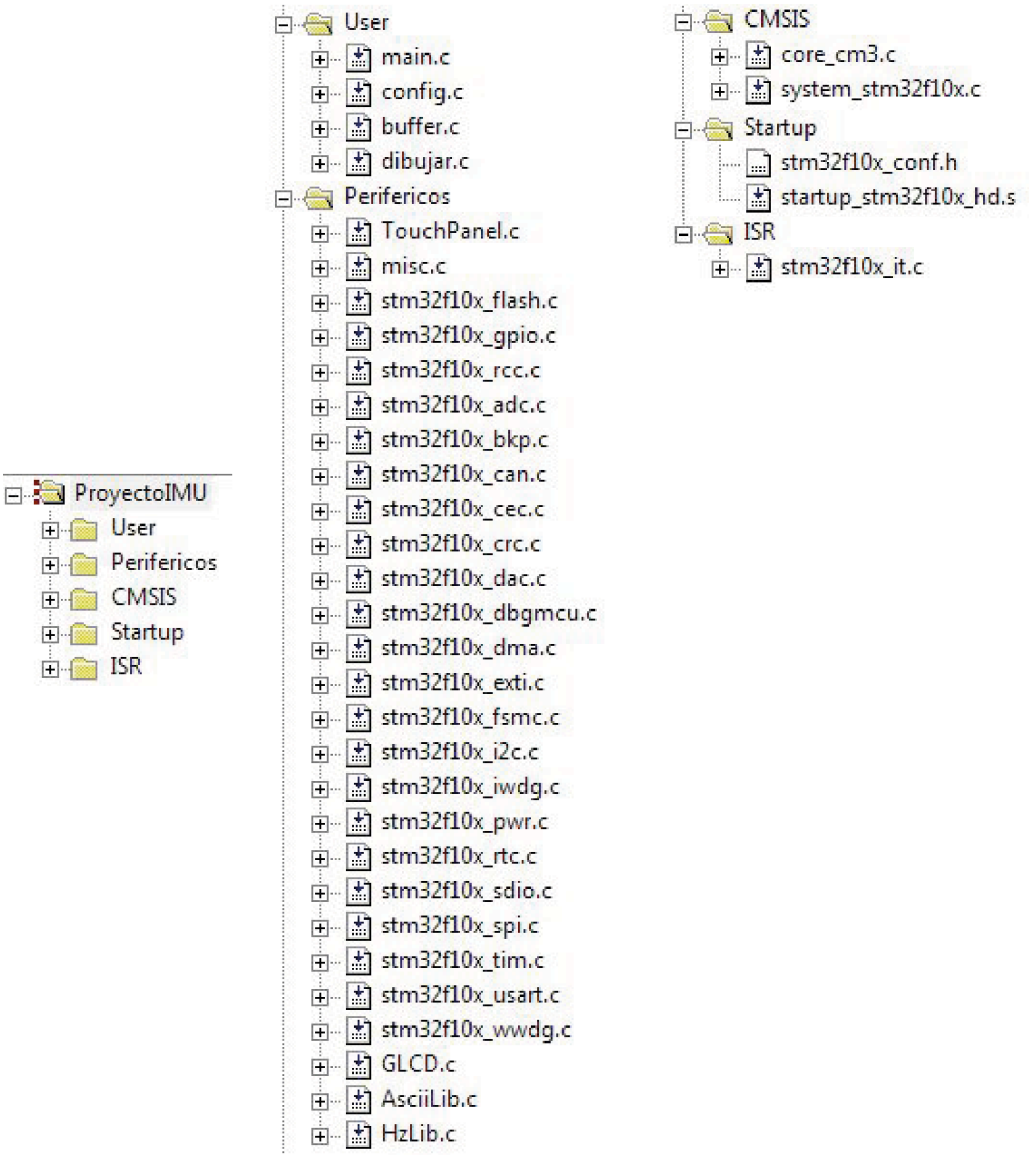

Figura 33. Estructura de ficheros en Keil.

4.2.2. Programa principal

En el diagrama de actividad de la Figura 28 se muestra el flujo de ejecución que sigue el programa principal, la función *int main(void){}*. Durante la cuál se configura el microcontrolador, sus periféricos y se realizan todos los cálculos necesarios para el programa principal, y se ejecuta de forma cíclica y permanente la parte principal del programa. En donde se recogen los datos y se representan o se envían por la USART.

En la Figura 34 se puede apreciar que el diagrama se ha dividido en tres bloques: Configuración del microcontrolador, configuración del sistema y bucle cíclico.

Configuración del microcontrolador:

La primera tarea de este bloque consiste en habilitar el reloj principal del sistema y configurar su frecuencia. Como el resto de relojes y frecuencias del sistema derivan del principal, esta configuración es bastante relevante en el rendimiento general del sistema. La configuración del reloj se realiza a 72 MHz, la cuál es la velocidad máxima permitida por este microcontrolador, con el objetivo de maximizar el rendimiento.

La siguiente tarea que se realiza es la de configuración de las interrupciones tanto para la USART1 como para el KEYA (botón A) y KEYB (botón B). Para la interrupción de la USART1 se configura como la interrupción con mayor prioridad, seguida por la de KEYA y siendo la interrupción de KEYB la de menor prioridad.

Lo siguiente a realizar es la configuración de los puertos de entrada/salida a los que se conectan la unidad inercial en la USART1 y la tarjeta FTDI Chip en la USART 2. Estas configuraciones son estándar, en donde se indica el pin, el modo de funcionamiento y la velocidad.

A continuación, se configuran los periféricos de comunicación que se utilizan para la comunicación con la unidad inercial y con la tarjeta FTDI Chip a través del puerto USART. La configuración de este puerto es básica, en donde se selecciona los parámetros de la comunicación serie como la tasa de transferencia de datos, la longitud de los datos, la paridad, bit de stop, el control de flujo y el modo de funcionamiento.

La siguiente tarea que se realiza es la de configuración de las interrupciones externas para el KEYA y KEYB. Estas configuraciones son básicas también, se seleccionan los parámetros como la línea que se va a utilizar, el modo y el tipo de disparo. Estos botones se utilizarán para volver a los menús de selección en la pantalla LCD.

Este bloque se termina con la configuración de la pantalla LCD, la cuál se realiza con la llamada a las funciones de inicialización de la librería del fabricante. También se realiza la llamada a la función inicialización del TouchPanel.

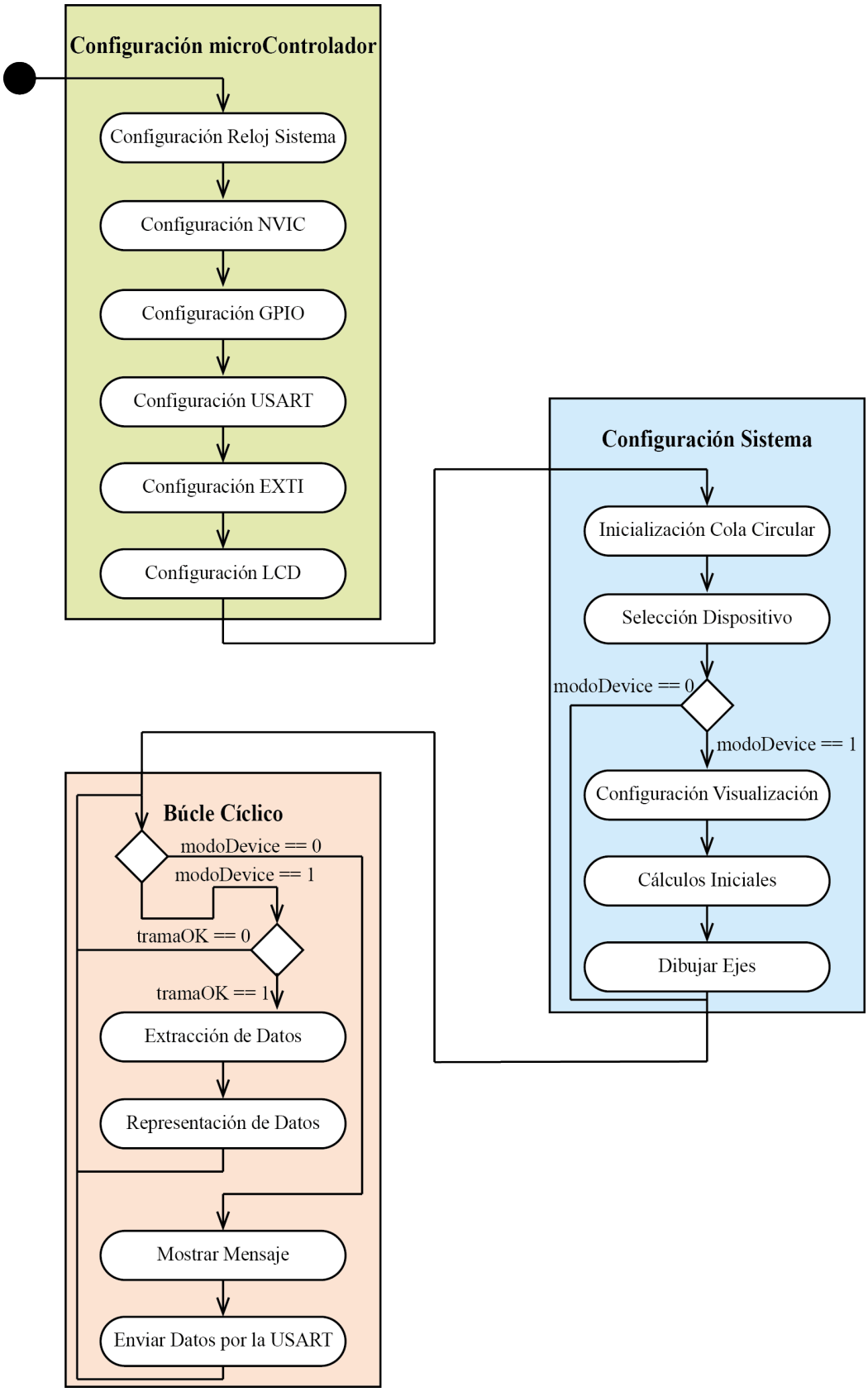

Figura 34. Diagrama de actividad del programa principal.

Configuración del sistema:

En primer lugar, se inicializa la cola circular en donde se guardaran los datos recibidos por la unidad inercial y de donde se extraerán para representarlos. Tras iniciar la cola circular se pasa a crear el menú en la pantalla LCD, en donde se tiene que seleccionar qué dispositivo se quiere utilizar para representar la señal de la unidad inercial. Estos dispositivos son el dispositivo Android externo o la pantalla LCD integrada.

Dependiendo esta última selección se pasará directamente a la siguiente sección o previamente se mostrará en la pantalla LCD otro menú en el cuál se tiene que seleccionar el sensor que se quiere visualizar, el acelerómetro o el giroscopio. También se tiene que elegir el tiempo de pantalla que se quiere ver y, en el caso de seleccionar el acelerómetro, la sensibilidad de éste. Tras ello, se realiza el cálculo de *incremento_x*, el cuál indica cada cuanto se tiene que incrementar el eje X para la representación de la señal. Y para terminar con esta sección se dibujan en la pantalla LCD los ejes de coordenadas, en el caso de seleccionar el giroscopio, o, aparte de los ejes, también una pequeña leyenda y tres avisadores junto con sus umbrales. Estos tres indicadores se iluminan si la señal sobrepasa alguno de los umbrales representados.

Bucle Cíclico:

Por último, el bucle cíclico ejecutará una parte del código u otro dependiendo del dispositivo seleccionado. Si se ha seleccionado el dispositivo Android se extraen los datos de la cola circular y se envían por la USART2, mientras a su vez se representa un mensaje indicando que sensor se esta representando. En el caso de haber seleccionado la pantalla LCD para representar la señal de la unidad inercial, se extraen los datos, se almacenan en un buffer según el eje (XYZ) y se representan. Cuando la señal llega al límite de la pantalla se comienza a representar desde el principio mientras se va borrando la señal que ya se había representado. También se ha generado una parte de código en la que se calcula si se ha producido una caída de la unidad inercial, y en tal caso se muestra un mensaje de aviso. Ya que si el dispositivo lo lleva incorporado una persona se puede saber si esta persona se ha caído.

4.2.3. Comunicaciones

En este apartado se describirán las comunicaciones entre la unidad inercial y la tarjeta HY Mini-STM32V, y entre la tarjeta HY Mini-STM32V y la tarjeta FTDI Chip.

Comunicación entre la unidad inercial y la tarjeta HY Mini-STM32V:

Esta comunicación se realiza por interrupciones y en un solo sentido, ya que como se ha mencionado en el apartado 2.4.2. Características no se pueden enviar tramas a la unidad inercial porque no se obtiene respuesta alguna.

El diagrama de actividad de la Figura 35 se muestra el flujo de ejecución de la interrupción de la USART1, la cuál se usa para la recepción de las tramas de la unidad inercial.

La interrupción se inicia comprobando el flag de *tramaEntrante*, en el caso de que sea uno, esto significa que se a recibido la cabecera 'snp' explicada en el apartado **2.4.5.** Estructura de la trama. En el caso de ser cero, se pasa a esperar a recibir la cabecera completa, cuando se haya recibido se pone el flag *tramaEntrante* a uno y se sale de la interrupción para volver a entrar cuando se reciba otra trama por la USART1.

En el caso de que ya se haya recibido la cabecera 'snp', se pasa a comprobar con el siguiente byte recibido, si en la trama vienen datos. En el caso de que este bit (Has Data) indique que si vienen datos, se comprueba el bit Is Batch. Este bit indicaba si es una operación por lotes, lo que significaba que en la trama venían varios registros de datos de las diferentes medidas de la unidad inercial. Tras saber que vienen datos en la trama, se calcula la longitud de los datos.

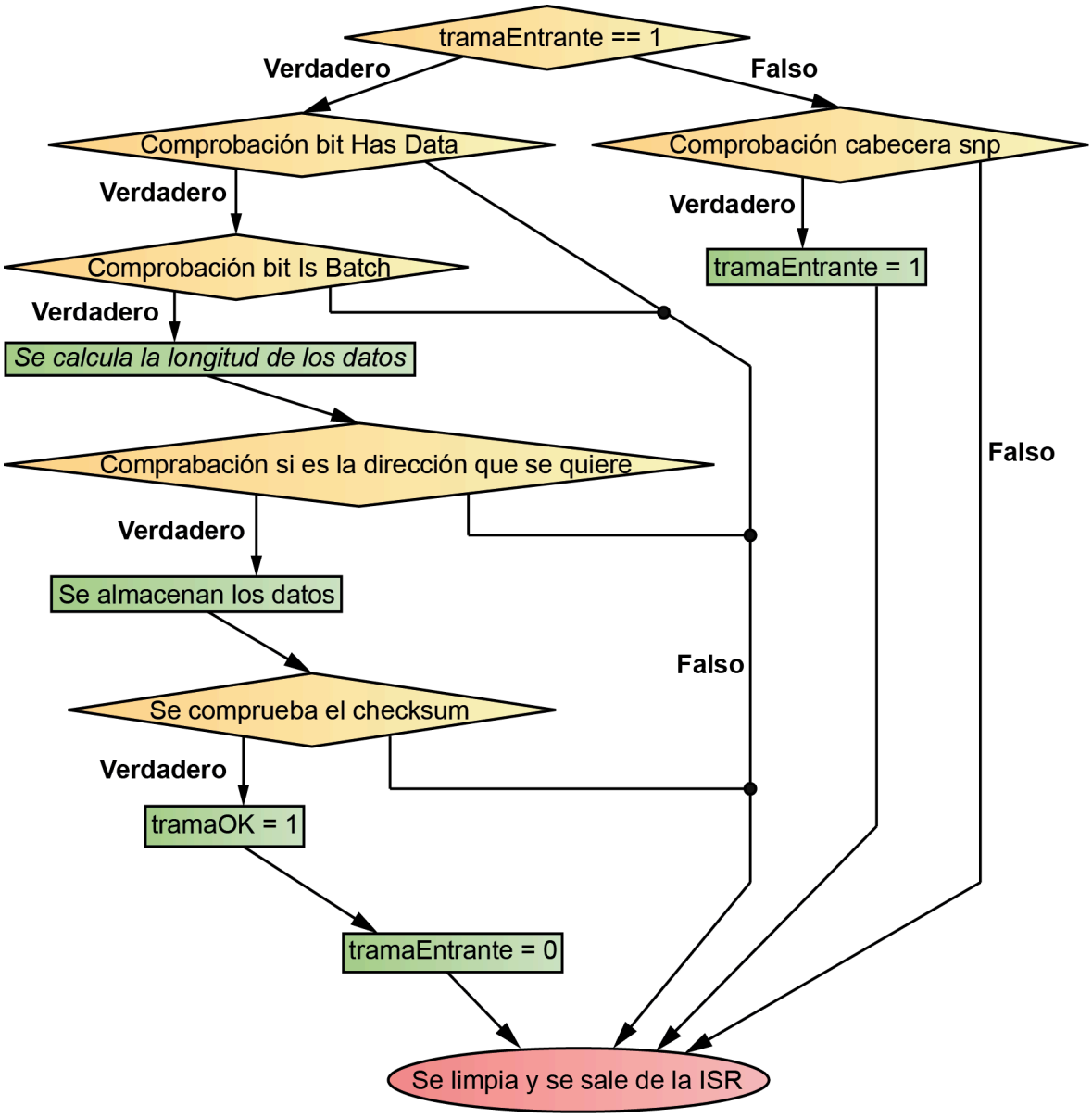

Figura 35. Diagrama de flujo de la interrupción de la USART1.

En el siguiente byte que se recibe si indica la dirección del primer registro que se recibirá, el cuál se comprueba para verificar que es la dirección que se necesita. Para este TFG en donde se quiere representar el giroscopio y el acelerómetro, con las medidas ya calibradas, necesitamos que la dirección sea la 0x61. Después de esta comprobación se pasa a recibir todos los datos y a almacenarlos en la cola circular creada. Una vez se almacenen todos los datos, se recibirán en los próximos dos bytes el checksum, el cuál se comparará con una variable llamada checksum en donde se hay ido calculando el checksum. De esta manera se sabe si la trama recibida es correcta.

Si la comprobación del checksum es correcta, se pone a uno un flag llamado *tramaOK,* el cuál indicará en el programa principal que se a recibido una trama correctamente.

Comunicación entre la tarjeta HY Mini-STM32V y la tarjeta FTDI Chip:

Esta comunicación se realiza en el programa principal, como se ha mencionado en el apartado anterior. Cuando se ha seleccionado que la señal se visualice por el dispositivo Android y se haya recibido una trama, indicado con el flag *tramaOK*, se envían los datos del acelerómetro por la USART2. Esto se realiza con las librerías del fabricante de la tarjeta HY Mini-STM32V, en donde existe una función para enviar datos de un byte de tamaño por la USART.

Como los datos a enviar son del tipo float, lo que significa que tienen un tamaño de 32 bit (4 bytes) y se tienen que dividir para poder enviarlo por la USART. En este caso se ha decidido por dividir en 4 bit (8 tramas) y utilizar los otros 4 bit del byte que se puede enviar por la USART para colocar la cabecera e indicar que trama se esta enviando. Para ello se ha seguido el siguiente criterio:

Formato del primer byte de datos, dividido en dos tramas:

0	1	2	3	4	5	6	7
0	0	0	0	X	X	X	X

Tabla 10. Primer byte, primera trama.

0	1	2	3	4	5	6	7
0	0	0	1	X	X	X	X

Tabla 11. Primer byte, segunda trama.

Formato del segundo byte de datos, dividido en dos tramas:

0	1	2	3	4	5	6	7
0	0	1	0	X	X	X	X

Tabla 12. Segundo byte, primera trama.

0	1	2	3	4	5	6	7
0	0	1	1	X	X	X	X

Tabla 13. Segundo byte, segunda trama.

Formato del tercer byte de datos, dividido en dos tramas:

0	1	2	3	4	5	6	7
0	1	0	0	X	X	X	X

Tabla 14. Tercer byte, primera trama.

0	1	2	3	4	5	6	7
0	1	0	1	X	X	X	X

Tabla 15. Tercer byte, segunda trama.

Formato del cuarto byte de datos, dividido en dos tramas:

0	1	2	3	4	5	6	7
0	1	1	0	X	X	X	X

Tabla 16. Cuarto byte, primera trama.

0	1	2	3	4	5	6	7
0	1	1	1	X	X	X	X

Tabla 17. Cuarto byte, segunda trama.

4.2.4. Manejo de señales

Como se ha mencionado en el apartado anterior se necesita que la dirección del primer registro de la trama recibida, sea la 0x61. Como se ha explicado en el apartado 2.4.5. Estructura de la trama, la trama que se necesita contiene 4 registros del acelerómetro, 4 del giroscopio y 4 del magnetómetro, este último no se utiliza. Estos 4 registros de cada sensor son de 32 bits de tamaño, por ser un dato tipo float, que para poder enviarlos por la USART la unidad inercial los divide en 4 a su vez. Esto es debido a que por la USART sólo se envía 8 bits de datos.

Como se explico en el apartado anterior, cuando se reciben los datos se van almacenando en la cola circular de 8 en 8 bits. En el programa principal se van extrayendo

estos datos de dicha cola, se unen de cuatro en cuatro para formar los datos tipo float y se almacenan en un array tipo float para cada sensor y eje, además se crea otro array más, también tipo float, en donde se almacenarán los datos del tiempo en el que fueron tomadas las medidas. Este último array no es necesario para este trabajo fin de grado pero sí puede ser interesante para líneas futuras.

A los datos almacenados no es necesario realizarle ningún tipo de tratado especial, ni ningún filtrado, ya que los datos ya vienen filtrados por la unidad inercial. La cuál lleva incorporada un filtro Kalman [6], que es el filtro ideal para este tipo de señales.

4.2.5. Representación LCD

La representación de la señales en la pantalla LCD de la tarjeta HY Mini-STM32V, se realiza en el programa principal tras haber recibido las tramas, unirlas y almacenarlas cada una en su array correspondiente. En la Figura 36 se puede ver su diagrama de flujo.

A continuación según el sensor que se haya seleccionado en el menú (giroscopio o acelerómetro) se extrae el dato del eje X. Este dato se escala para poder representarlo en la pantalla LCD, ya que dicha pantalla cuenta con 320x240 pixeles, y los valores de los datos van de 0 hasta ±8 para el acelerómetro y de 0 a ±2000 para el giroscopio. El escalado se realiza de la siguiente manera, el dato se multiplica por el número de pixeles que tenemos para representar en el eje vertical, 100 pixeles para la parte positiva y 100 pixeles para la parte negativa, y se divide por el valor máximo que se puede recibir. Para el giroscopio este valor máximo es 2000 y en el caso del acelerómetro va a depender del valor seleccionado en el menú, que pueden ser 2, 4 u 8. Este cálculo nos proporciona el número de pixeles en vertical que es equivalente al dato que se quiere representar.

Una vez obtenido el número de pixeles se comprueba que no haya superado el valor máximo de pixeles, que en tal caso se le asigna este máximo que es 100 pixeles. Ahora se comprueba si el valor es positivo o negativo, para saber si se debe representar por encima o por debajo del eje central que divide la parte positiva de la parte negativa, y se almacena en un vector en donde estarán todos los datos que se representen.

A continuación se para a representar la señal en la pantalla, la cuál se realiza con una función de la librería del fabricante. A esta función se le tiene que pasar por parámetro el valor actual a representar y el valor anterior (que esta almacenado en el vector), también se le tiene que pasar la posición del eje horizontal en donde se tiene que representar tanto el valor actual como el anterior. Este valor del eje horizontal se va incrementando por cada vez que se represente un dato, y cuánto se tiene que incrementa se calcula en los cálculos iniciales del programa principal.

Seguido a la representación se realiza una comprobación de si el valor representado a alcanzado alguno de los umbrales representados para que en tal caso se ilumine un rectángulo en la pantalla según el umbral alcanzado. Además se comprueba si se ha alcanzado el límite de la pantalla, que en el caso de ser así se pone a uno un flag, *limpiar*, que indicará que se tiene que realizar el scroll. Todo este proceso que se ha desarrollado se realiza para cada uno de los ejes (XYZ).

Para realizar el scroll se ha de forma que la señal vuelva a empezar a representar desde la izquierda, pero esta vez llevará una pequeña ventana delante que irá borrando la señal que ya se había representado.

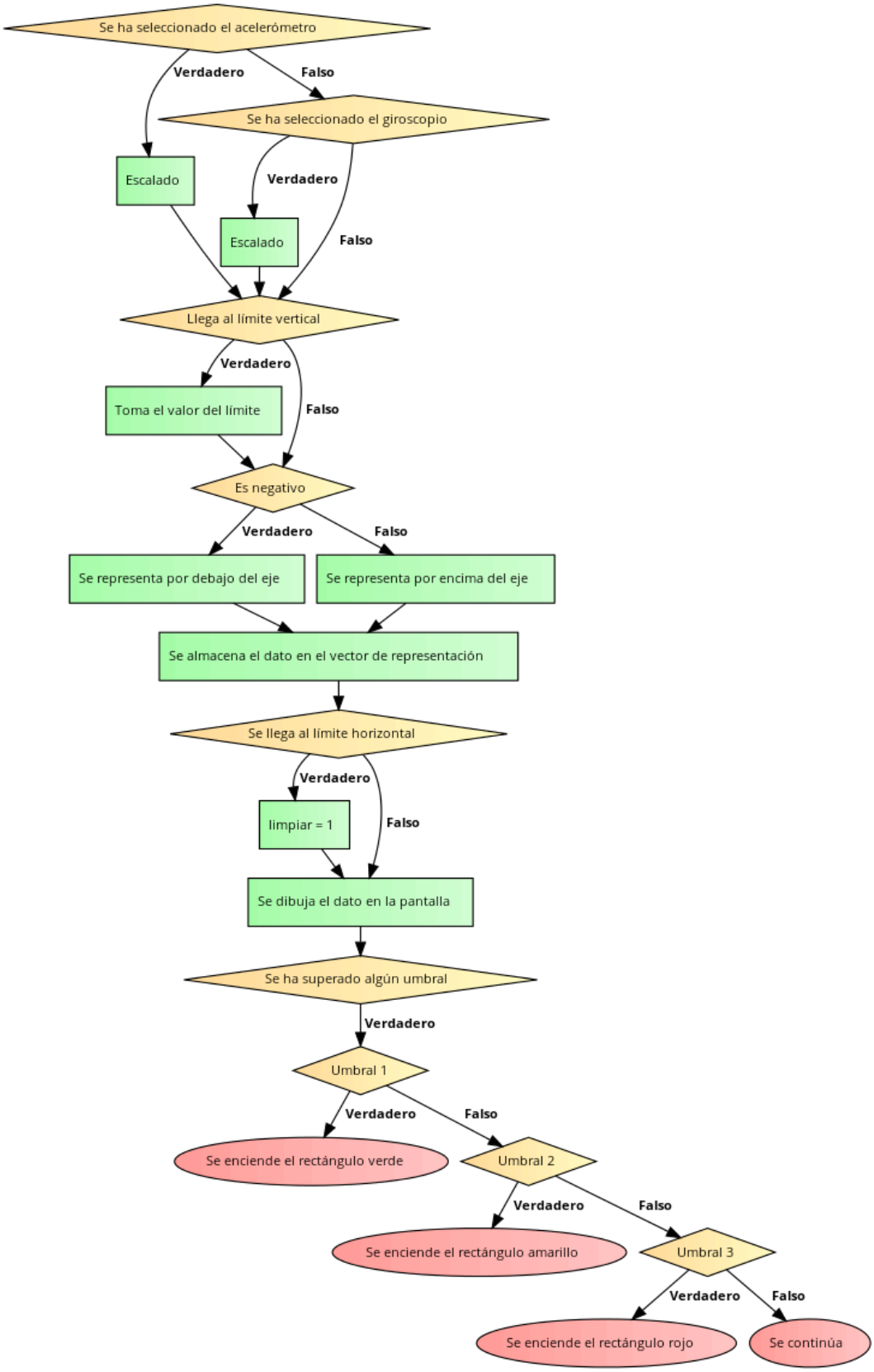

Figura 36. Diagrama de flujo de representación en la pantalla LCD.

4.2.6. Librerías

Para el desarrollo del firmware se han utilizado las librerías facilitadas por el fabricante, en este caso las librerías son las descritas a continuación:

- `TouchPanel.c`
- `GLCD.c`
- `stm32f10x_gpio.c`
- `stm32f10x_rcc.c`
- `stm32f10x_exti.c`
- `stm32f10x_usart.c`

4.3. Aplicación Android

4.3.1. Introducción

Para la representación de la señal en el dispositivo móvil se ha utilizado la tarjeta FTDI Chip DS-UMFT311EV descrita en al apartado 2.5. FTDI Chip DS-UMFT311EV. Esta tarjeta sirve como puente entre la tarjeta HY Mini-STM32V y el dispositivo móvil. Se ha optado por el uso de esta tarjeta porque no todos los dispositivos móviles Android están provisto del USB OTG [11], lo que implica que en caso de los dispositivos que no cuenten con este puerto, se les tendrían que realizar un Rooting [12] para poder comunicarse con él. Y como la realización de este sistema supondría al usuario la perdida de la garantía del dispositivo y que no todos los usuarios son capaces de realizar el Rooting.

Por otra parte para a realización de la aplicación web se ha optado por trabajar en el entorno oficial del sistema operativo Android, llamado Android Studio [15]. Dicho entorno de trabajo a sido desarrollado por Google exclusivamente dedicado para la programación de aplicaciones para dispositivos Android (Figura 37).

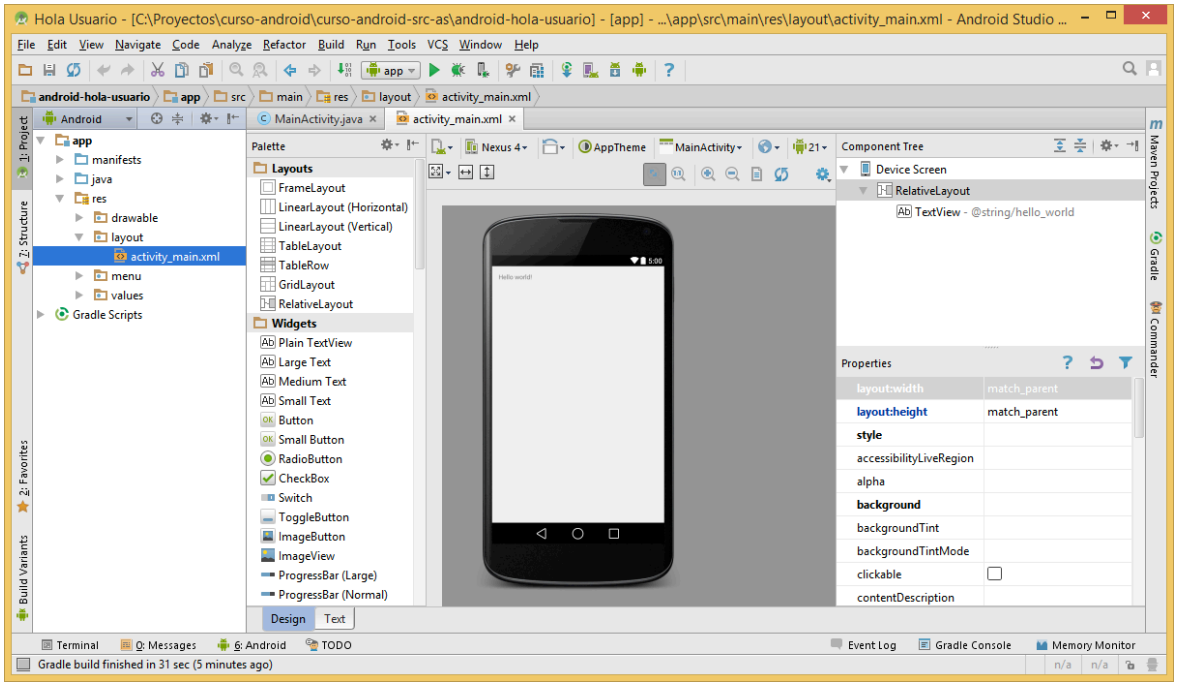

Figura 37. Entorno de trabajo Android Studio.

4.3.2. Configuración de comunicaciones

Para realizar la comunicación entre la tarjeta HY Mini-STM32V es necesario realizar unas configuraciones previas. Dicha configuración se realiza de dos formas, una se tiene que realizar manualmente en el hardware en donde se selecciona el tipo de comunicación que se realizará con la tarjeta HY Mini-STM32V. Como para este caso la comunicación se realiza por el puerto UART, en la tarjeta FTDI Chip hay que quitar el primer jumper, todas las posibles configuraciones están explicadas en el datasheet [7]. En la Figura 38 se puede apreciar la configuración ya realizada.

Figura 38. Configuración jumpers de la tarjeta FTDI Chip DS-UMFT311EV.

La otra parte de la configuración se realiza por software, en donde se utiliza la librería del fabricante para indicar las configuraciones básicas de una UART, como puede ser la

velocidad de transmisión, el número de bits, etc. Todas las posibles configuraciones se explican en la guía de programación de FTDI [13].

4.3.3. Comunicaciones

La recepción de las tramas enviadas por la tarjeta HY Mini-STM32V hacia el dispositivo Android se realiza de forma parecida a la que se produce entre la unidad inercial y la tarjeta de desarrollo. Cuando se recibe una trama se comprueba la cabecera, según se explicó en el apartado 4.2.2. Comunicaciones, una vez recibidas las 8 tramas de un dato se unen y se envían a representar en la pantalla.

4.3.4. Representación

Tanto la parte de recepción de datos como la parte de representación de los mismos se han realizado a partir de las librerías creadas por Jonas Gehring, las cuáles se encuentran en la siguiente referencia [14].

En el Anexo XIII se adjunta un ejemplo de la clase desarrollada para poder graficar en el dispositivo Android la señal de la unidad inercial recibida de la tarjeta HY Mini-STM32V. En este ejemplo solo se representa el eje X de las medidas enviadas por el acelerómetro.

Capítulo 5: RESULTADOS

En este capítulo se mostraran los resultados obtenidos tras la finalización del sistema, se puede ver en la Figura 39 el diagrama de bloques del sistema final y en la Figura 40 el prototipo creado. En él se aprecia la unidad inercial UM7-LT, que es la encargada de la recogida de los datos, junto con la tarjeta de desarrollo HY Mini-STM32V, que es la encargada de recoger los datos, procesarlos y representarlos en la pantalla LCD. Por último, se puede observar la tarjeta FTDI Chip, que realiza la función de puente entre la tarjeta de desarrollo y el dispositivo Android.

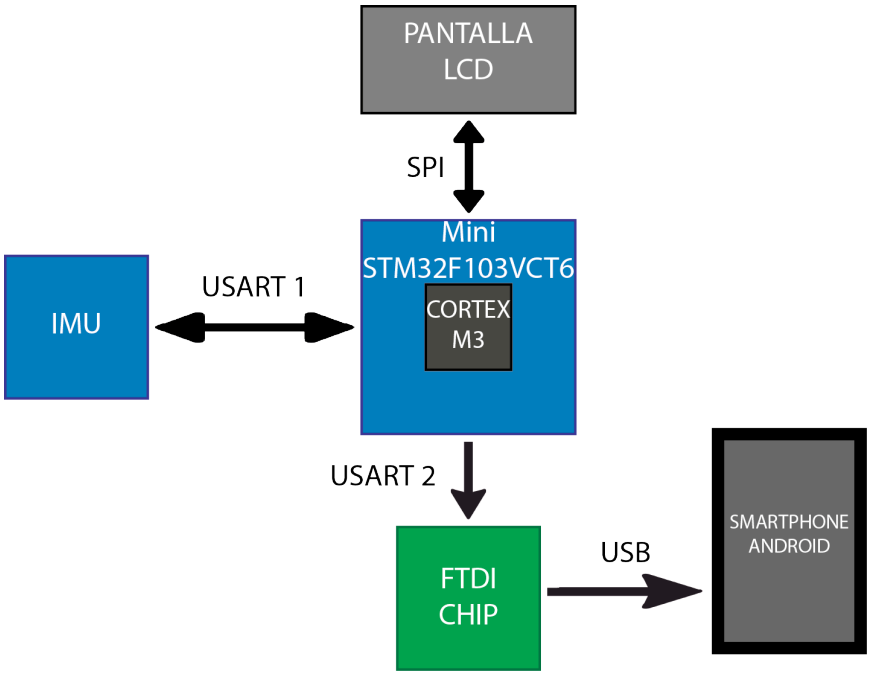

Figura 39. Diagrama de bloques de la solución.

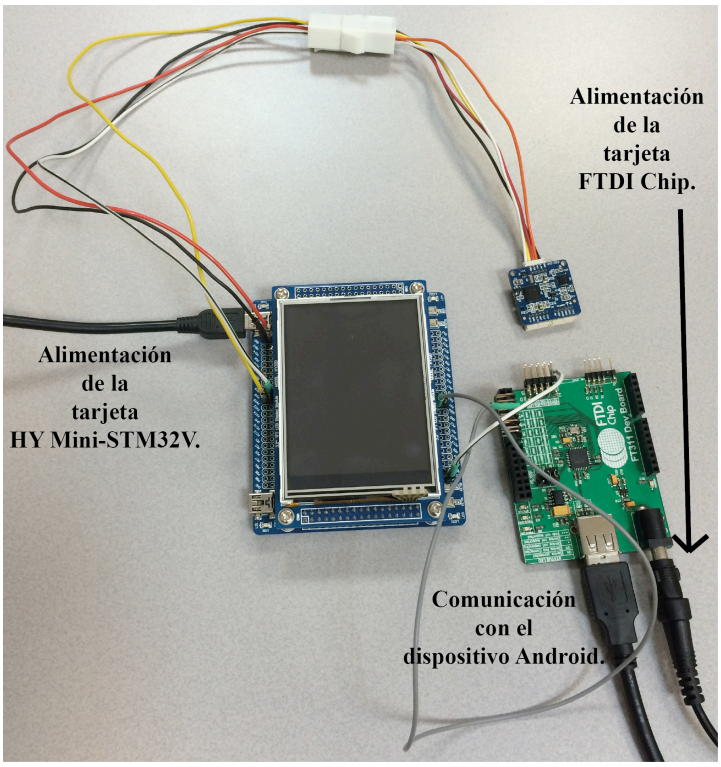

Figura 40. Conexión de todo el sistema.

Como ya se ha comentado anteriormente, se trata de un prototipo funcional que captura la señal de la IMU y los represente. En la Figura 41 se puede apreciar la señal representada en la pantalla LCD, cuya representación corresponde a una caída.

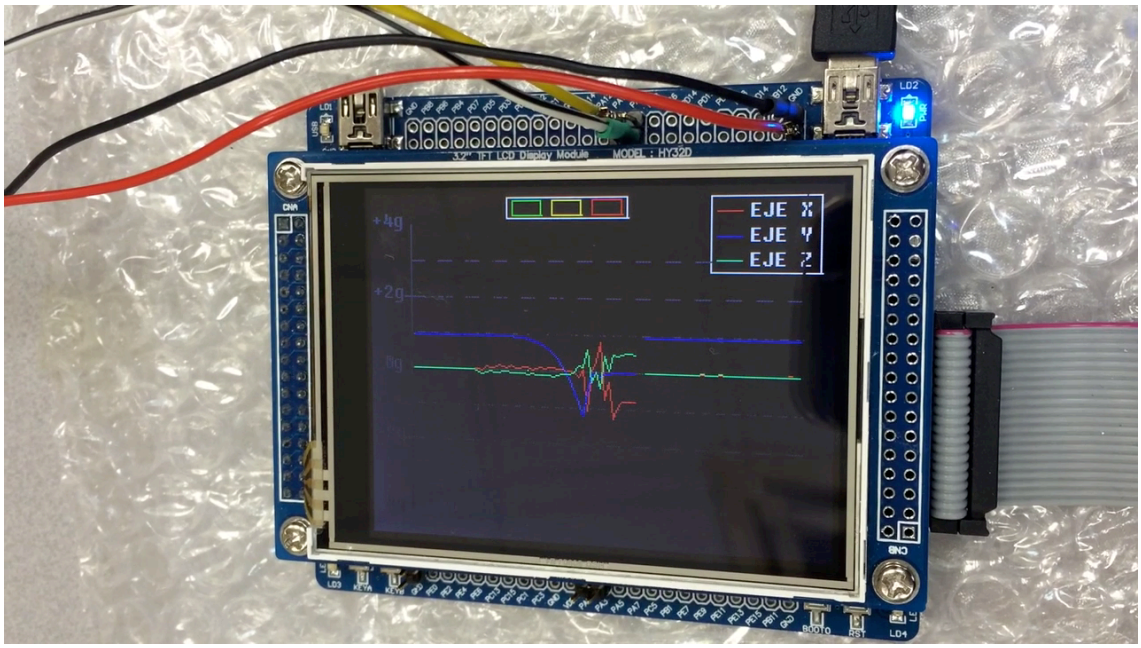

Figura 41. Señal del acelerómetro en pantalla LCD.

Para poder mostrar el funcionamiento del sistema se ha desarrollado un programa de detección de caídas como ejemplo. Se ha comenzado con la realización de mediciones de varias caídas para poder obtener un patrón lo más ajustado posible de una caída.

Para realizar dicho experimento se ha creado un montaje practico en donde se ha colocado la unidad inercial en un soporte vertical reproduciendo a una persona de estatura media, 170 cm, este montaje se puede apreciar en la Figura 42.

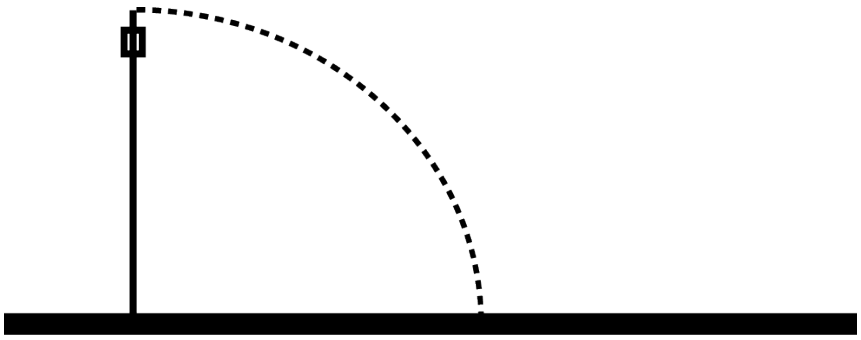

Figura 42. Situación inicial de la simulación.

Cuando se procede ha realizar el experimento, se deja caer el soporte y cuando esté en el punto final de la caída el vector de aceleración se quedaría de la forma representada en la Figura 43.

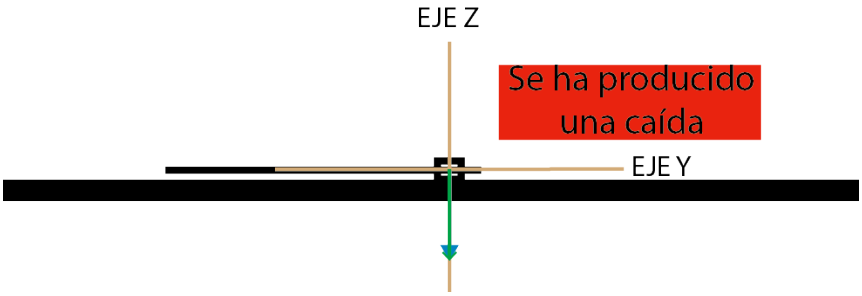

Figura 43. Situación final de la simulación.

Es este punto (Figura 43) es donde se realiza el cálculo de la detección de una caída, ya que, aunque en teoría solo habrá valor en una componente (Eje Z), cunado el objeto llega a impactar contra el suelo se produce rebotes. Por este motivo surgirán valores en las tres componentes.

El cálculo para la detección de una caída se ha realizado con las medidas de las tres componentes XYZ del acelerómetro al impactar contra el suelo. Con estos datos se ha calculado el módulo de la aceleración de la unidad inercial al chocar contra el suelo. En la Figura 44 se puede observar la señal del acelerómetro cuando éste impacta contra el suelo y debajo se muestra el mensaje que se ha creado tras detectar una caída.

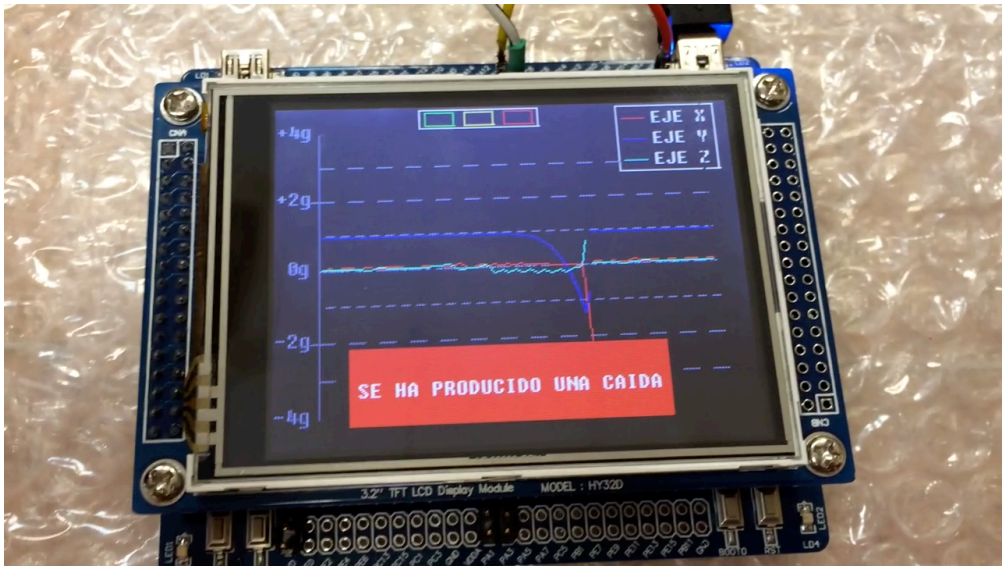

Figura 44. Representación de una detección de una caída.

A continuación en la Figura 45 se muestra una imagen del montaje práctico creado para realizar esta simulación de detección de caídas.

Figura 45. Montaje práctico.

Como continuación de este TFG se puede desarrollar un sistema de detección de caída mejorado, en donde entren en juego otros parámetros también facilitados por la unidad inercial.

RESULTADOS

Capítulo 6: CONCLUSIONES

El objetivo de este Trabajo Fin de Grado era monitorizar una actividad física basado en una unidad inercial con soporte en Android, a través de la creación de un prototipo capaz de representar la señal de la unidad inercial.

Una unidad inercial es un dispositivo electrónico que mide e informa acerca de la velocidad, orientación y fuerzas gravitacionales de un objeto, usando una combinación de sensores como son el acelerómetro y el giroscopio. Con estas medidas se es capaz de saber la velocidad angular que ha tomado una objeto al inclinarse o su aceleración a la hora de desplazarse, expresado en fuerzas g.

Existe una gran cantidad de fabricantes de dispositivos electrónicos de los que ofrecen diferentes modelos de unidades inerciales. Para el desarrollo de este sistema se ha contado de antemano con dichos dispositivos al ser los existentes en la división de Equipos y Sistemas de Comunicación del IUMA.

CONCLUSIONES

Capítulo 7: LÍNEAS FUTURAS

Como futura línea de continuación de este TFG podría ser el desarrollo de una aplicación la cuál pueda monitorizar la evolución del saque de un jugador de voleibol o la detección del movimiento de un corredor de trail.

El prototipo se ha desarrollado mediante conexión USB por su facilidad de integración, como desarrollo para un futuro de este sistema y para una mayor flexibilidad se incorporaría una conexión BLTE (Bluetooth Low Energy) para la comunicación con el dispositivo Android, y junto con la disminución del tamaño de los componentes se podría desarrollar un prototipo en donde queden todos los componentes ensamblados en un solo dispositivo, de tipo wearable, portable y autónomo.

El desarrollo de nuevas aplicaciones accediendo directamente a la unidad inercial procesando la información con ángulos de Euler o bien, con Cuaterniones, pues ofrece soluciones más específicas.

LINEAS FUTURAS

Capítulo 8: PRESUPUESTO

Presupuesto detallado

Siguiendo las recomendaciones del COIT, el presupuesto se ha desglosado en varias secciones. En estas secciones se han separado los distintos costes orientativos asociados al desarrollo del proyecto. Estos costes se dividen en:

1. Recursos materiales.
2. Trabajo tarifado por tiempo empleado.
3. Costes de redacción del Trabajo Fin de Grado.
4. Material fungible.
5. Derechos de visado del COIT.
6. Costes de tramitación y envío.
7. Aplicación de impuestos.

Recursos Materiales

Para la ejecución de este Trabajo Fin de Grado han sido necesarios las herramientas software para la realización de los estudios, el paquete office para la redacción de la memoria y los equipos hardware usados para dar soporte a esas herramientas.

Recursos Hardware

Para la ejecución de este estudio las herramientas hardware que se han utilizado son las siguientes:

- Placa de desarrollo HY Mini-STM32V Dev. Board + 3.2" TFT LCD Module.
- Módulo de desarrollo Host para Android UMFT311EV.
- Unidad inercial UM7-LT.
- Osciloscopio PicoScope 3224.

- 2 sondas CP-260 60 MHz,
- Ordenador de mesa Sun Microsystems Intel Core 2.
- Pantalla Sun Microsystems 22" LCD.

Indicar que el período estimado de amortización de un equipo electrónico es de 24 meses. Por lo tanto, para el desarrollo de este TFG se emplean 4 meses, de esta forma, el importe corresponde a una sexta parte del precio de mercado. Señalar que otros recursos como las placas o dispositivos comprados, no presentan período de amortización.

Hardware	Precio mercado	Período de amortización (meses)	Tiempo de uso (meses)	Importe
HY Mini-STM32V	28,69 €	-	-	28,69 €
UMFT311EV	28,65 €	-	-	28,65 €
UM7-LT	115,59 €	-	-	115,59 €
Osciloscopio PicoScope 3224	590,66 €	24	4	98,44 €
Sondas CP-260 60 MHz	38.99 €	24	4	6,50 €
Ordenador	854,85 €	24	4	142,48 €
Pantalla Sun Microsystems 22" LCD	284,25 €	24	4	47,38 €
TOTAL	467,73 €			

Tabla 18. Costes de Recursos Hardware.

Recursos Software

Las herramientas software utilizadas para el desarrollo de este TFG fueron:

- Keil uVision 4.73
- Microsoft Office 2013

Como en el caso del hardware el período estimado de amortización de un software electrónico es de 24 meses. Por lo tanto, para el desarrollo de este Trabajo Fin de Grado se emplean 4 meses, de esta forma, el importe corresponde a una sexta parte del precio de mercado.

Software	Coste	Período de amortización (meses)	Tiempo de uso (meses)	Importe
Keil uVision 4.73	8.538,10 €	24	4	1.423,01 €
Microsoft Office 2013	269 €	24	4	44,83 €
TOTAL	1.467,84 €			

Tabla 19. Costes de Recursos Software.

Trabajo tarifado por tiempo empleado

En este Trabajo Fin de Grado se han invertido 300 horas en las tareas de preparación, desarrollo y documentación necesarias para la elaboración del mismo. El importe de las horas de trabajo utilizadas para la realización del proyecto se calcula siguiendo las recomendaciones del COIT:

$$H = C_t \cdot 74{,}88 \cdot H_n + C_t * 96{,}72 \cdot H_e \quad (1.1.)$$

Donde:

- H son los honorarios totales por el tiempo dedicado.
- H_n son las horas normales trabajadas (dentro de la jornada laboral).
- H_e son las horas especiales.
- C_t es un factor de corrección función del número de horas trabajadas.

Como en la elaboración de este Trabajo Fin de Grado han sido necesarias 300 horas (5h/día * 60 días), todas ellas dentro del horario normal. Según el COIT, el coeficiente C_t tiene un valor variable en función del número de horas empleadas de acuerdo con la siguiente tabla:

Horas empleadas	Factor de corrección
Hasta 36 horas	1
Desde 36 horas a 72 horas	0,9
Desde 72 horas a 108 horas	0,8
Desde 108 horas a 144 horas	0,7
Desde 144 horas a 180 horas	0,65
Desde 180 horas a 360 horas	0,6
Desde 360 horas a 540 horas	0.55

Tabla 20. Factor de corrección según el número de horas trabajadas.

Como se puede observar el número de horas está comprendido entre 180 y 360 horas, por lo que según la Tabla 20 el factor de corrección es de C_t = 0,60. Con ello, la ecuación del importe de horas de trabajo resulta de la siguiente forma:

$$H = 0{,}60 * 74{,}88 * 300 + 0{,}60 * 96{,}72 * 0 = 13478.40€ \quad (1.2)$$

Los honorarios totales por tiempo dedicado, libres de impuestos, ascienden a *trece mil cuatrocientos setenta y ocho euros con cuarenta céntimos* (**13478.40 €**).

Costes de redacción del Trabajo Fin de Grado

El importe de la redacción del proyecto se calcula de acuerdo a la siguiente expresión:

$$R = 0{,}07 * P * C_n \quad (1.3)$$

Donde:

- P es el presupuesto del proyecto.
- C_n es el coeficiente de ponderación en función del presupuesto.

En la siguiente tabla se muestra el presupuesto calculado hasta el momento:

RECURSOS	COSTES
Recursos hardware	467,73 €
Recursos software	1.467,84 €
Trabajo tarifado por tiempo empleado	13.478,40 €
TOTAL	15.413,97 €

Tabla 21. Presupuesto.

El presupuesto calculado hasta el momento asciende a 15.413,97 €. Como el coeficiente de ponderación para presupuestos inferiores a 30.050 € viene definido por el COIT, con un valor de 1.00, el coste derivado de la redacción del Trabajo Fin de Grado es de:

$$R = 0{,}07 * 15.413{,}97 * 1{,}00 = 1.078{,}98 \text{ €} \tag{1.4}$$

Material fungible

Además de los recursos hardware y software, en este proyecto, se han utilizado otros materiales como son los folios, impresión y encuadernación.

MATERIALES	COSTES
Folios	12 €
Impresión	40 €
Encuadernación	9 €
TOTAL	65 €

Tabla 22. Costes Material Fungible.

Derechos de visado del COIT

Los gastos de visado del COIT, precios orientativos, se tarifan mediante la siguiente expresión.

$$V = 0{,}006 * P * C_V$$

Donde:

- P es el presupuesto del proyecto.
- C_V es el coeficiente reductor en función del presupuesto del proyecto.

El presupuesto P, calculado hasta el momento corresponde a la suma de los costes de ejecución material, de redacción y de material fungible.

$$P = 1.078{,}98 + 65 + 15.413{,}97 = 16.557{,}94\ € \tag{1.6}$$

Como el coeficiente de ponderación para presupuestos menores de 30.050 € viene definido por el COIT con un valor de 1.00, el coste de los derechos de visado del proyecto asciende a la cantidad de:

$$V = 0{,}006 * 16.557{,}94 * 1.00 = 99{,}35\ € \tag{1.7}$$

Por tanto el coste de los derechos de visado del proyecto asciende a *noventa y nueve euros con treinta y cinco céntimos* **(99,35 €).**

Gastos de tramitación y envío

Los gastos de tramitación y envío están fijados en 6.01 €.

Aplicación de impuestos

El coste total del proyecto, antes de aplicarle los correspondientes impuestos, asciende 18.523,72 €, a lo que hay que sumarle el 7% de IGIC, con lo que el coste definitivo del Trabajo Fin de Grado es:

RECURSOS MATERIALES	1.935,57 €
TRABAJO TARIFADO POR TIEMPO EMPLEADO	13.478,40 €
REDACCIÓN DEL TFG	1.078,98 €
MATERIAL FUNGIBLE	65 €
DERECHOS DE VISADO	99,35 €
GASTOS DE TRAMITACIÓN Y ENVÍO	6,01 €
SUBTOTAL	16.663,31 €
APLICACIÓN DE IMPUESTOS (IGIC 7%)	1.166,43 €
TOTAL PRESUPESTO	17.829,74 €

Tabla 23. Costes Totales del TFG.

El presupuesto total asciende a la cantidad de *diecisiete mil ochocientos veintinueve euros con setenta y cuatro céntimos* (**17.827,74 €**).

Las Palmas de Gran Canaria a 8 de Junio del 2015.

Fdo.: Heriberto J. Díaz Luis-Ravelo.

PRESUPUESTO

Capítulo 9: BIBLIOGRAFÍA

[1] *Definición de DOF(Degrees of freedom).* http://en.wikipedia.org/wiki/Degrees_of_freedom

[2] *Hoja de características del microprocesador STM32F103VCT6.* http://www.st.com/web/en/resource/technical/document/datasheet/CD00191185.pdf

[3] *Manual de referencia de la arquitectura ARM.* https://www.scss.tcd.ie/John.Waldron/3d1/arm_arm.pdf

[4] *The Insider's Guide to the STM32 ARM Based Microcontroller.* http://www.hitex.com/fileadmin/pdf/insiders-guides/stm32/isg-stm32-v18d-scr.pdf

[5] *Hoja de características de la unidad inercial UM7-LT.* http://www.chrobotics.com/docs/UM7_Datasheet.pdf

[6] *Grewal, M. S., & Andrews, A. P. (2011). Kalman filtering: theory and practice using MATLAB. John Wiley & Sons.*

[7] *Hoja de características del FTDI Chip DS-UMFT311EV.* http://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT311D.pdf

[8] *Explicación de Android Open Accessory Mode.* https://source.android.com/accessories/protocol.html

[9] *Esquemático Mini-STM32F103VCT6.* http://www.haoyuelectronics.com//Attachment/HY-MiniSTM32V/HY-MiniSTM32V_SCH.pdf

[10] *Conector JST ZHR-5.* http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Prototyping/ZH%20Connector.pdf

[11] *Explicación de OTG (On The Go).* http://www.usb.org/developers/onthego/

[12] *Explicación de Rooting.* http://es.wikipedia.org/wiki/Android_rooting

[13] *FT31xD Android Programmers Guide.* http://www.ftdichip.com/Support/Documents/ProgramGuides/FT31XD_Android_programmer_guide(FT_000532).pdf

[14] *Página de las librerías de Jonas Gehring.* http://www.android-graphview.org

[15] *Página oficial de Android Studio. https://developer.android.com/sdk/*

[16] *Noureldin, A., Karamat, T. B., & Georgy, J. (2012). Fundamentals of inertial navigation, satellite-based positioning and their integration. Springer Science & Business Media.*

[17] *Hanson, A. J. (2005, July). Visualizing quaternions. In ACM SIGGRAPH 2005 Courses (p. 1). ACM.*

[18] *Acar, C., & Shkel, A. (2008). MEMS vibratory gyroscopes: structural approaches to improve robustness. Springer Science & Business Media.*

[19] *Sazonov, E., & Neuman, M. R. (Eds.). (2014). Wearable Sensors: Fundamentals, implementation and applications. Elsevier.*

[20] *Millor, N., Lecumberri, P., Gomez, M., Martinez-Ramirez, A., & Izquierdo, M. (2014). Kinematic parameters to evaluate functional performance of sit-to-stand and stand-to-sit transitions using motion sensor devices: a systematic review.*

[21] *Mannini, A., & Sabatini, A. M. (2014). Walking speed estimation using foot-mounted inertial sensors: Comparing machine learning and strap-down integration methods. Medical engineering & physics, 36(10), 1312-1321.*

[22] *Rodríguez, J.G. (2010). Programación de un Registrador de Datos Sísmicos Autónomo de Bajo Consumo. https://upcommons.upc.edu/pfc/bitstream/2099.1/9257/1/Memòria.pdf*

[23] *Acevedo, M. (2012). Sistema de control de accesos basado en microprocesadores ARM32 Cortex. http://e-archivo.uc3m.es/bitstream/handle/10016/16869/TFG_Mario_Acevedo_Aguilar.pdf?sequence=1*

[24] *Aguado, D. (2014). Sistema de Logging Geoposicional en Tiempo Real para Sistemas Empotrados. https://ruidera.uclm.es/xmlui/bitstream/handle/10578/3953/TFG_Aguado%20Araujo.pdf?sequence=1*

[25] *Vazquez, A. (2014). INTEGRACIÓN MEDIANTE FILTRO DE KALMAN DE SENSORES INERCIALES Y GPS PARA LA ESTIMACIÓN DE LA POSICIÓN DE UN VEHÍCULO. http://ruc.udc.es/bitstream/2183/13190/2/VazquezFraga_Alejandro_TFG_2014.pdf*

[26] *Alcalá, E. (2014). DESARROLLO E IMPLEMENTACIÓN DE UN CUADRICÓPTERO.* *https://zaguan.unizar.es/record/14367/files/TAZ-TFG-2014-538.pdf*

[27] *Giménez, A.R. (2010). Desarrollo de un Sistema Integrado de Navegación Inercial: Interfície IMU + FPGA.* *http://ddd.uab.cat/pub/trerecpro/2010/hdl_2072_151848/PFC_AlfredRaulGimenezBonastre.pdf*

BIBLIOGRAFÍA

ANEXOS

ANEXOS

Anexo I. Registros de la unidad inercial.

En la Tabla 24 se encuentran los registros de la unidad inercial UM7-LT en donde se almacenan las diferentes medidas. En ella se nombra su dirección en hexadecimal, su nombre y una breve descripción de la información que almacena.

DIRECCIÓN	NOMBRE DEL REGISTRO	DESCRIPCIÓN
0x55	DREG_HEALTH	Información sobre el estado de la unida inercial.
0x56	DREG_GYRO_RAW_XY	Datos raw del giroscopio del eje X e Y.
0x57	DREG_GYRO_RAW_Z	Datos raw del giroscopio del eje Z.
0x58	DREG_GYRO_TIME	Tiempo en el que los datos del giroscopio fueron adquiridos.
0x59	DREG_ACCEL_RAW_XY	Datos raw del acelerómetro del eje X e Y.
0x5A	DREG_ACCEL_RAW_Z	Datos raw del acelerómetro del eje Z.
0x5B	DREG_ACCEL_TIME	Tiempo en el que los datos del acelerómetro fueron adquiridos.
0x5C	DREG_MAG_RAW_XY	Datos raw del magnetómetro del eje X e Y.
0x5D	DREG_MAG_RAW_Z	Datos raw del magnetómetro del eje Z.
0x5E	DREG_MAG_RAW_TIME	Tiempo en el que los datos del magnetómetro fueron adquiridos.
0x5F	DREG_TEMPERATURE	Datos de temperatura.

0x60	DREG_TEMPERATURE_TIME	Tiempo en el que los datos de la temperatura fueron adquiridos.
0x61	DREG_GYRO_PROC_X	Datos procesados del giroscopio del eje X.
0x62	DREG_GYRO_PROC_Y	Datos procesados del giroscopio del eje Y.
0x63	DREG_GYRO_PROC_Z	Datos procesados del giroscopio del eje Z.
0x64	DREG_GYRO_PROC_TIME	Tiempo en el que los datos del giroscopio fueron adquiridos.
0x65	DREG_ACCEL_PROC_X	Datos procesados del acelerómetro del eje X.
0x66	DREG_ACCEL_PROC_Y	Datos procesados del acelerómetro del eje Y.
0x67	DREG_ACCEL_PROC_Z	Datos procesados del acelerómetro del eje Z.
0x68	DREG_ACCEL_PROC_TIME	Tiempo en el que los datos del acelerómetro fueron adquiridos.
0x69	DREG_MAG_PROC_X	Datos procesados del magnetómetro del eje X.
0x6A	DREG_MAG_PROC_Y	Datos procesados del magnetómetro del eje Y.
0x6B	DREG_MAG_PROC_Z	Datos procesados del magnetómetro del eje Z.
0x6C	DREG_MAG_PROC_TIME	Tiempo en el que los datos del magnetómetro fueron adquiridos.
0x70	DREG_EULER_PHI_THE	Ángulo del Roll y Pitch.

	TA	
0x71	DREG_EULER_PSI	Ángulo del Yaw.
0x72	DREG_EULER_PHI_THETA_DOT	Velocidad angular de Roll y Pitch.
0x73	DREG_EULER_PSI_DOT	Velocidad angular de Yaw.
0x74	DREG_EULER_TIME	Tiempo en que se calculó la posición y la velocidad de Euler.

Tabla 24. Registros enviados por la IMU.

ANEXO I

Anexo II. Esquemático HY Mini-STM32V.

En la Figura 46 y Figura 47 se muestra el esquemático completo de la tarjeta HY Mini-STM32V, en donde se puede apreciar la asignación de cada uno de sus pines, como de los diferentes componentes que forman la placa.

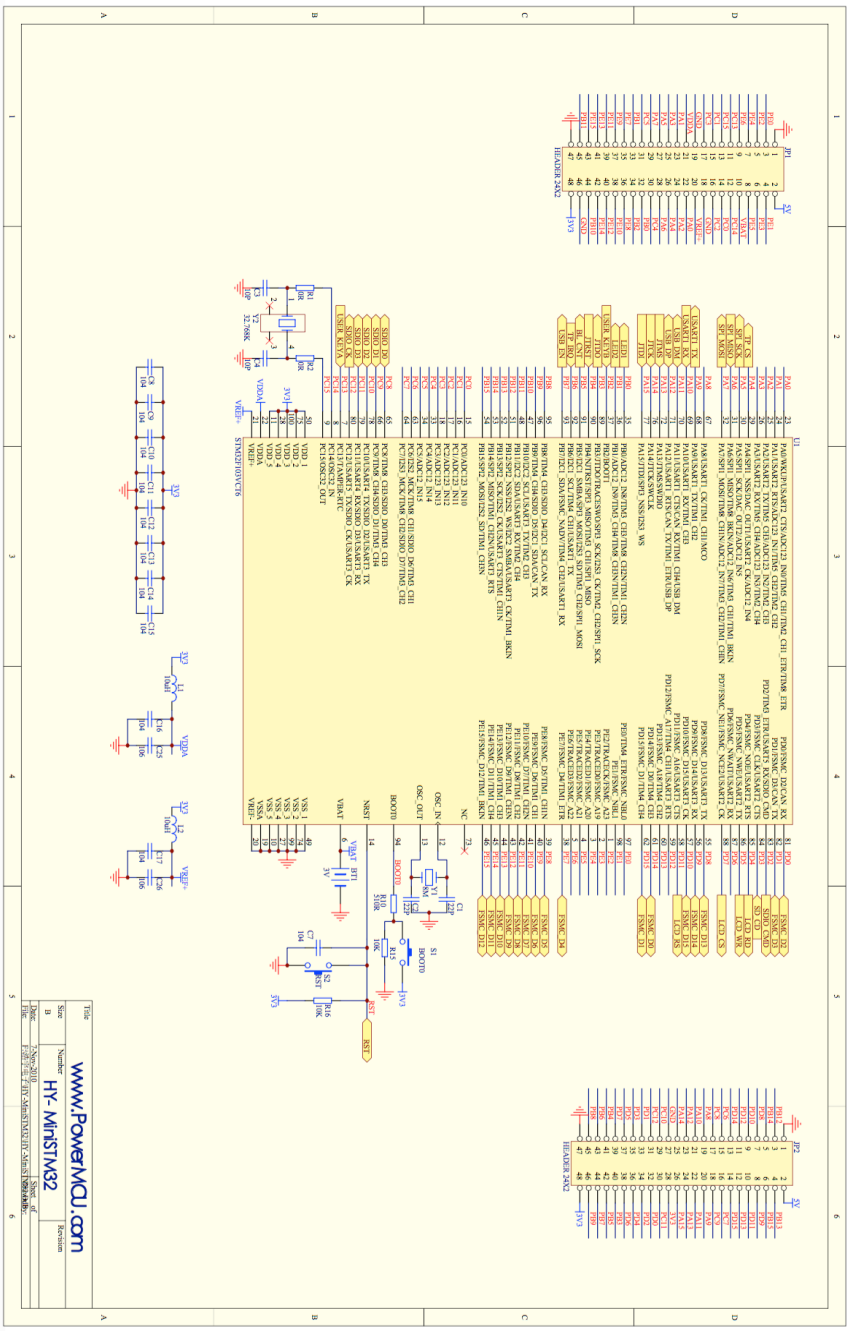

Figura 46. Esquemático HY Mini-STM32V (1).

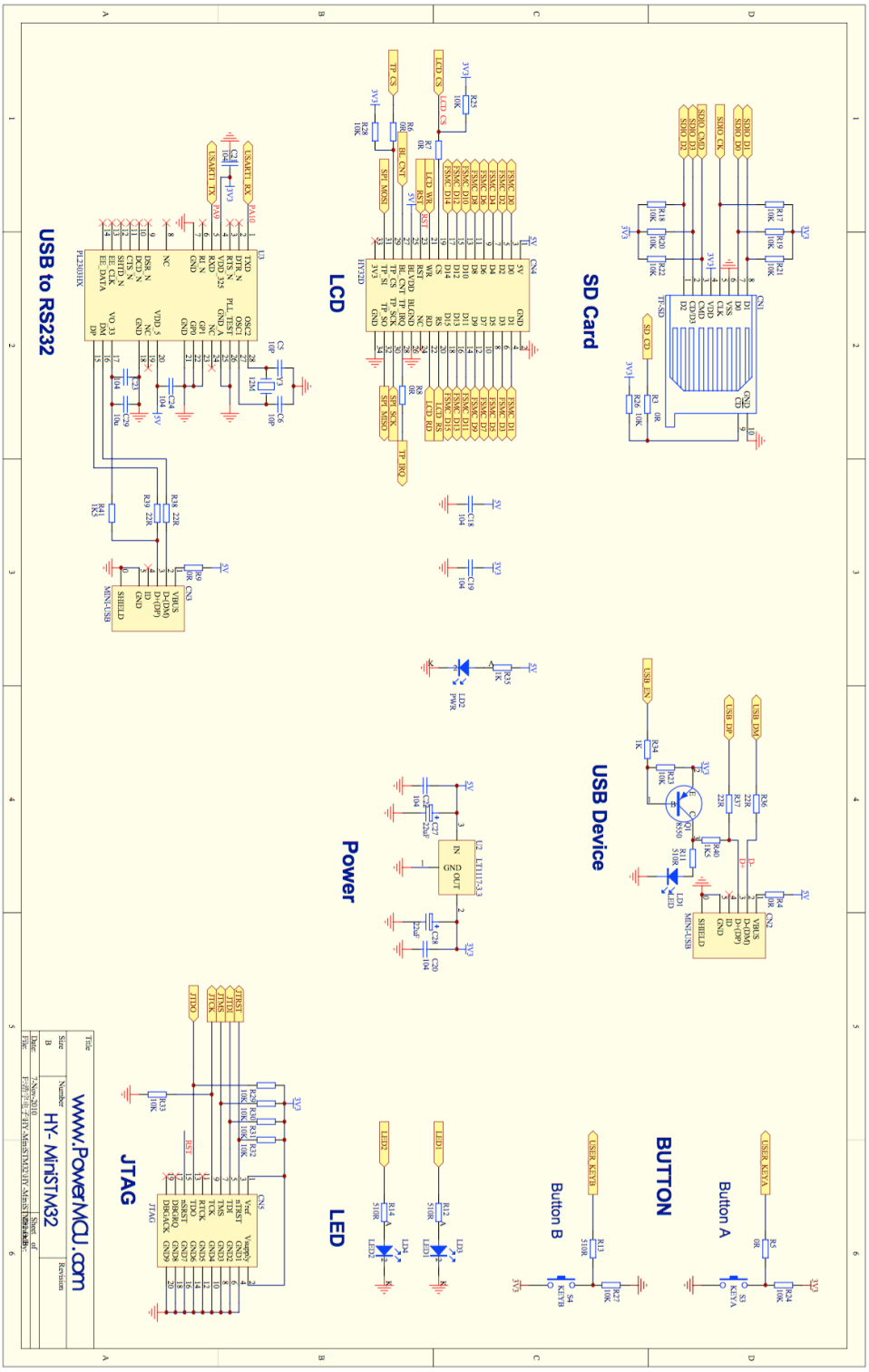

Figura 47. Esquemático HY Mini-STM32V (2).

Anexo III. Esquemático FTDI Chip.

En la Figura 48 se muestra el esquemático de la tarjeta FTDI Chip FT311D, en ella se puede apreciar la asignación de cada de sus pines, como de cada conector necesario para su buena utilización.

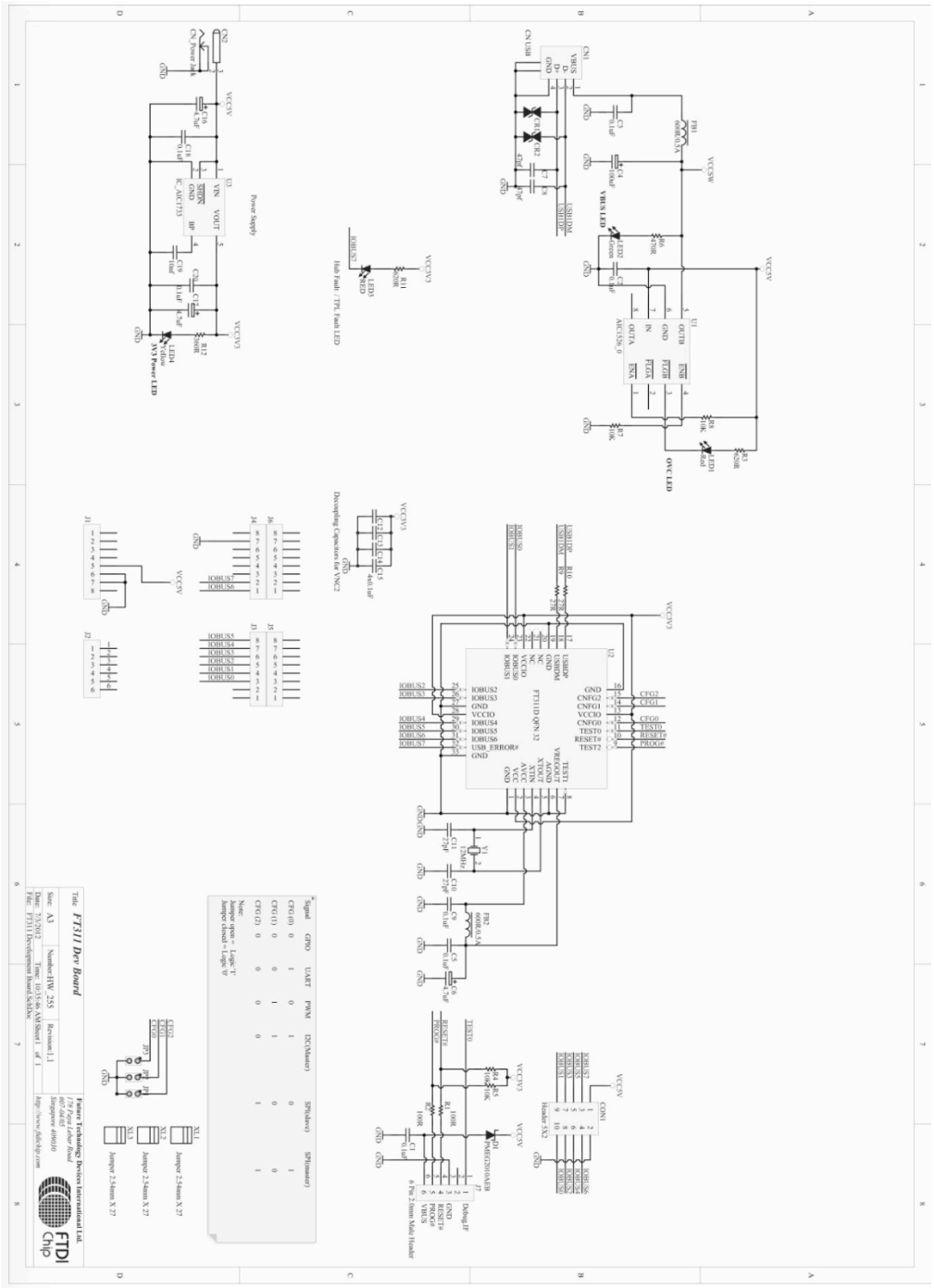

Figura 48. Esquemático FTDI Chip.

ANEXO III

Anexo IV. PROGRAMA PRINCIPAL

En el programa principal se realiza la parte más importante del firmware, en donde se procesan los datos, se representan, se analizan lo datos, se calcula la detección de caída y se envía hacia el dispositivo Android.

```
int main(void)
{

  /* System clocks configuration ---------------------------------------*/
  RCC_Configuration();

  /* NVIC configuration ------------------------------------------------*/
  NVIC_Configuration();

  /* GPIO configuration ------------------------------------------------*/
  GPIO_Configuration();

  /* USART configuration -----------------------------------------------*/
  USART_Configuration();

  /* EXTI configuration ------------------------------------------------*/
  EXTI_Configuration();

  /* LCD Inicializacion ------------------------------------------------*/
  LCD_Initializtion();
  LCD_Clear(Black);

  /* TouchPanel Inicializacion -----------------------------------------*/
  TP_Init();
  TouchPanel_Calibrate();

  /* Inicializacion Cola Circular --------------------------------------*/
  init_Buffer(&colaCirc, 100);

  /* Seleccion dispositivo para visualizar -----------------------------*/
  select_Device();

  /*
  Si modoDevice == 1 se muestra por la pantalla LCD, y para ello se
necesitan algunos calculos como el tiempo de pantalla a mostrar, el
sensor a mostar y en el caso del acelerómetro la sensibilidad.
  Si modoDevice == 0 no hace falta ningun cálculo se elige el sensor a
enviar por la UART hacia el dispositivo movil y se muestra un mensaje.
  */
  if(modoDevice == 1){

    config_visual();

      if(accelOK == 1){
            modo = 1;
```

```
            eje_x = 31;
        }else if(gyroOK == 1){
            modo = 0;
            eje_x = 21;
        }

        /* Calculo incremento_x ----------------------------------------*/
        calculo_Inicial(modo, tempo);

        /* Calculo del valor para escalar ------------------------------*/
        cociente = (((float)100)/((float)sensibilidad));

        /* Dibujar Ejes ------------------------------------------------*/
        dibujar_Ejes(modo, sensibilidad);

    }else if (modoDevice == 0){

        gyroOK = 0;
        accelOK = 1;

        if(accelOK == 1){

            modo = 1;
            eje_x = 31;
            tempo = 4;
            sensibilidad = 4;
            LCD_Clear(Black);
            GUI_Text(30, 100, "REPRESENTANDO EN EL DISPOSITIVO",Grey,
Black);
            GUI_Text(80, 120, "MOVIL EL ACELEROMETRO.",White, Black);

        }else if(gyroOK == 1){

            modo = 0;
            eje_x = 21;

        }/* End if accelOK y gyroOK */

    }/* End if modoDevice */

    // Habilita la interrupción para registro de recepción no vacío.
    USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);

    /* Infinite loop */
    while (1){

    /*
    En el caso de tocar KEYA volver al menu de seleccion de dispositvo
para mostrar el sensor elegido y se realiza los mismo que en el inicio de
la configuracion.
    */
    if(flagBoton2 == 1){

      /* Seleccion dispositivo para visualizar --------------------------*/
      select_Device();
```

```
    /*
    Si modoDevice == 1 se muestra por la pantalla LCD, y para ello se
necesitan algunos calculos como el tiempo de pantalla a mostrar, el
sensor a mostar y en el caso del acelerómetro la sensibilidad.
    Si modoDevice == 0 no hace falta ningun cálculo se elige el sensor a
enviar por la UART hacia el dispositivo movil y se muestra un mensaje.
    */
    if(modoDevice == 1){

      config_visual();

      if(accelOK == 1){
            modo = 1;
            eje_x = 31;
      }else if(gyroOK == 1){
            modo = 0;
            eje_x = 21;
      }

      /* Calculo incremento_x ----------------------------------------*/
      calculo_Inicial(modo, tempo);

      /* Calculo del valor para escalar ------------------------------*/
      cociente = (((float)100)/((float)sensibilidad));

      /* Dibujar Ejes ------------------------------------------------*/
      dibujar_Ejes(modo, sensibilidad);

}else if (modoDevice == 0){

      gyroOK = 0;
      accelOK = 1;

      if(accelOK == 1){

            modo = 1;
            eje_x = 31;
            tempo = 4;
            sensibilidad = 4;
            LCD_Clear(Black);
            GUI_Text(30, 100, "REPRESENTANDO EN EL DISPOSITIVO",Grey,
Black);
            GUI_Text(80, 120, "MOVIL EL ACELEROMETRO.",White, Black);

      }else if(gyroOK == 1){

            modo = 0;
            eje_x = 21;

      }/* End if accelOK y gyroOK */

    }/* End if modoDevice */

    flagBoton2 = 0;

}/* End if flagBoton2 */

if(modoDevice == 1){
```

```
if(flagBoton == 1){

   config_visual();

   if(accelOK == 1){
         modo = 1;
         eje_x = 31;

   }else if(gyroOK == 1){
         modo = 0;
         eje_x = 21;

   }

   /* Calculo incremento_x ----------------------------------------*/
   calculo_Inicial(modo, tempo);

         /* Calculo del valor para escalar -------------------------*/
         cociente = (((float)100)/((float)sensibilidad));

         /* Dibujar Ejes -------------------------------------------*/
         dibujar_Ejes(modo, sensibilidad);

         flagBoton = 0;
         flagAvisoCaida = 0;
         g = 0;
         d = 0;
         k = 0;

   }/* End if flagBoton */

   /* Si trama recibida correctamente */
   if(tramaOK == 1){

         /* Puntero de lectura al inicio de la cola */
         colaCirc.read = colaCirc.start;

         /* Se extrae el numero de registros de la cola */
         cola = pop(&colaCirc);
         registros = *cola;

         /* Se extrae la direccion del primer registro de la cola */
         cola = pop(&colaCirc);
         direccion = *cola;

         /* Se extraen los datos de la cola */
         l = 0;
         for(l=0; l<length; l++){

               cola = pop(&colaCirc);
               datos[l] = (uint8_t)*cola;

         }/* end for datos */

         m = 0;
         l = 0;
         for(l=0; l<length; l=l+4){
```

```
                valor = datos[l] << 24;
                valor |= datos[l+1] << 16;
                valor |= datos[l+2] << 8;
                valor |= datos[l+3];
                valorDatosProc = *(float*)&valor;
                datosProcFloat[m] = valorDatosProc;
                m++;

        }/* end for */

        /* Acelerometro */
        if(modo == 1){
                m = 4;
                accelX[j] = datosProcFloat[m++];
                accelY[j] = datosProcFloat[m++];
                accelZ[j] = datosProcFloat[m++];
   accelTime[j] = (uint16_t)datosProcFloat[m++];
}

/* Giroscopio */
if(modo == 0){
   m = 0;
   gyroX[j] = datosProcFloat[m++];
   gyroY[j] = datosProcFloat[m++];
   gyroZ[j] = datosProcFloat[m++];
   gyroTime[j] = (uint16_t)datosProcFloat[m++];
}

//------ X -----------------------------------------------------------/
if(accelOK == 1){
   //Se calcula el pixel a pintar
   result = ((accelX[j])*(cociente));
   //en esta variable se guarda el pixel a pintar
   valor_pixel = (int16_t)result;
}else if(gyroOK == 1){
   //Se calcula el pixel a pintar
   result = ((gyroX[j])*(cociente));
   //en esta variable se guarda el pixel a pintar
   valor_pixel = (int16_t)result;
}

if(result >= 100){
   result = 100;
}else if (result <= -100){
   result = -100;
}

auxX = result;
valorMaxX = accelX[j];

if((valor_pixel & 0xF000) == 0xF000){
   almacen_DatosX[k] = 120 + auxX;
   actual = almacen_DatosX[k];
}else{
   almacen_DatosX[k] = 120 - auxX;
   actual = almacen_DatosX[k];
}
```

```
    k++;
    if (k > 1){

       anterior = almacen_DatosX[k-2];

       if(k == (muestraCog+1)){
             limpiar = 1;
             k = 0;
       }

       colorLinea = 0xF000;    //Rojo
       dibujar_Linea2(eje_x, actual, (eje_x - incremento_x), anterior,
colorLinea);
    }/* End if k > 1 */



    if(modo == 1){
       switch(sensibilidad){

             case 8:
                   if(((actual <= 107) && (actual > 95)) || ((actual >=
132) && (actual < 145))){
                         LCD_RectangleFill(104, 4, 20, 10, Green);
                         LCD_Delay(50);
                         LCD_RectangleFill(105, 5, 18, 8, Black);

                   }else if(((actual <= 95) && (actual > 82)) || ((actual
>= 145) && (actual < 157))){
                         LCD_RectangleFill(132, 4, 20, 10, Yellow);
                         LCD_Delay(50);
                         LCD_RectangleFill(133, 5, 18, 8, Black);

                   }else if((actual <= 82) || (actual >= 157) ){
                         LCD_RectangleFill(160, 4, 20, 10, Red);
                         LCD_Delay(50);
                         LCD_RectangleFill(161, 5, 18, 8, Black);
                   }
                   break;

             case 4:
                   if(((actual <= 95) && (actual > 70)) || ((actual >=
145) && (actual < 170))){
                         LCD_RectangleFill(104, 4, 20, 10, Green);
                         LCD_Delay(50);
                         LCD_RectangleFill(105, 5, 18, 8, Black);

                   }else if(((actual <= 70) && (actual > 45)) || ((actual
>= 170) && (actual < 195))){
                         LCD_RectangleFill(132, 4, 20, 10, Yellow);
                         LCD_Delay(50);
                         LCD_RectangleFill(133, 5, 18, 8, Black);

                   }else if((actual <= 45) || (actual >= 195) ){
                         LCD_RectangleFill(160, 4, 20, 10, Red);
                         LCD_Delay(50);
```

```
                        LCD_RectangleFill(161, 5, 18, 8, Black);
                    }
                    break;

            case 2:
                    if(((actual <= 70) && (actual > 45)) || ((actual >=
170) && (actual < 195))){
                        LCD_RectangleFill(104, 4, 20, 10, Green);
                        LCD_Delay(50);
                        LCD_RectangleFill(105, 5, 18, 8, Black);

                    }else if(((actual <= 45) && (actual > 20)) || ((actual
>= 195) && (actual < 220))){
                        LCD_RectangleFill(132, 4, 20, 10, Yellow);
                        LCD_Delay(50);
                        LCD_RectangleFill(133, 5, 18, 8, Black);

                    }else if((actual <= 20) || (actual >= 220) ){
                        LCD_RectangleFill(160, 4, 20, 10, Red);
                        LCD_Delay(50);
                        LCD_RectangleFill(161, 5, 18, 8, Black);
                    }
                    break;
      }/* End switch sensibilidad */

   }/* End if modo */

   //------ X ---------------------------------------------------------/

   //------ Y ---------------------------------------------------------/
   if(accelOK == 1){
     //Se calcula el pixel a pintar
     result = ((accelY[j])*(cociente));
     //en esta variable se guarda el pixel a pintar
     valor_pixel = (int16_t) result;
   }else if(gyroOK == 1){
     //Se calcula el pixel a pintar
     result = ((gyroY[j])*(cociente));
     //en esta variable se guarda el pixel a pintar
     valor_pixel = (int16_t) result;
   }

   if(result >= 100){
     result = 100;
   }else if (result <= -100){
     result = -100;
   }

   auxY = result;
   valorMaxY = accelY[j];

   if((valor_pixel & 0xF000) == 0xF000){
     almacen_DatosY[d] = 120 + auxY;
     actual = almacen_DatosY[d];
   }else{
     almacen_DatosY[d] = 120 - auxY;
     actual = almacen_DatosY[d];
   }
```

```
    d++;
    if (d > 1){
      anterior = almacen_DatosY[d-2];

      if(d == (muestraCog+1)){
            limpiar = 1;
            d = 0;
      }

      colorLinea = 0x00FF;    // Azul
      dibujar_Linea2(eje_x, actual, (eje_x - incremento_x), anterior,
colorLinea);
    }/* End if d > 1 */

    if(modo == 1){
      switch(sensibilidad){

            case 8:
                  if(((actual <= 107) && (actual > 95)) || ((actual >=
132) && (actual < 145))){
                        LCD_RectangleFill(104, 4, 20, 10, Green);
                        LCD_Delay(50);
                        LCD_RectangleFill(105, 5, 18, 8, Black);

                  }else if(((actual <= 95) && (actual > 82)) || ((actual
>= 145) && (actual < 157))){
                        LCD_RectangleFill(132, 4, 20, 10, Yellow);
                        LCD_Delay(50);
                        LCD_RectangleFill(133, 5, 18, 8, Black);

                  }else if((actual <= 82) || (actual >= 157) ){
                        LCD_RectangleFill(160, 4, 20, 10, Red);
                        LCD_Delay(50);
                        LCD_RectangleFill(161, 5, 18, 8, Black);
                  }
                  break;

            case 4:
                  if(((actual <= 95) && (actual > 70)) || ((actual >=
145) && (actual < 170))){
                        LCD_RectangleFill(104, 4, 20, 10, Green);
                        LCD_Delay(50);
                        LCD_RectangleFill(105, 5, 18, 8, Black);

                  }else if(((actual <= 70) && (actual > 45)) || ((actual
>= 170) && (actual < 195))){
                        LCD_RectangleFill(132, 4, 20, 10, Yellow);
                        LCD_Delay(50);
                        LCD_RectangleFill(133, 5, 18, 8, Black);

                  }else if((actual <= 45) || (actual >= 195) ){
                        LCD_RectangleFill(160, 4, 20, 10, Red);
                        LCD_Delay(50);
                        LCD_RectangleFill(161, 5, 18, 8, Black);
                  }
                  break;
```

```
            case 2:
                  if(((actual <= 70) && (actual > 45)) || ((actual >=
170) && (actual < 195))){
                        LCD_RectangleFill(104, 4, 20, 10, Green);
                        LCD_Delay(50);
                        LCD_RectangleFill(105, 5, 18, 8, Black);

                  }else if(((actual <= 45) && (actual > 20)) || ((actual
>= 195) && (actual < 220))){
                        LCD_RectangleFill(132, 4, 20, 10, Yellow);
                        LCD_Delay(50);
                        LCD_RectangleFill(133, 5, 18, 8, Black);

                  }else if((actual <= 20) || (actual >= 220) ){
                        LCD_RectangleFill(160, 4, 20, 10, Red);
                        LCD_Delay(50);
                        LCD_RectangleFill(161, 5, 18, 8, Black);
                  }
                  break;
      }/* End switch sensibilidad */

   }/* End if modo */

   //------ Y ---------------------------------------------------------/

   //------ Z ---------------------------------------------------------/
   if(accelOK == 1){
     //Se calcula el pixel a pintar
     result = ((accelZ[j])*(cociente));
     //en esta variable se guarda el pixel a pintar
     valor_pixel = (int16_t) result;
   }else if(gyroOK == 1){
     //Se calcula el pixel a pintar
     result = ((gyroZ[j])*(cociente));
     //en esta variable se guarda el pixel a pintar
     valor_pixel = (int16_t) result;
   }

   if(result >= 100){
     result = 100;
   }else if (result <= -100){
     result = -100;
   }

   auxZ = result;
   valorMaxZ = accelZ[j];

   if((valor_pixel & 0xF000) == 0xF000){
     almacen_DatosZ[g] = 120 - auxZ;
     actual = almacen_DatosZ[g];
   }else{
     almacen_DatosZ[g] = 120 + auxZ;
     actual = almacen_DatosZ[g];
   }

   g++;
   if (g > 1){
```

```
      anterior = almacen_DatosZ[g-2];

      if(g == (muestraCog+1)){
            limpiar = 1;
            g = 0;
      }

      colorLinea = 0x0FF0;          //Verde
      dibujar_Linea2(eje_x, actual, (eje_x - incremento_x), anterior,
colorLinea);
    }/* Enf if g > 1 */
    if(modo == 1){
      switch(sensibilidad){

            case 8:
                  if(((actual <= 107) && (actual > 95)) || ((actual >=
132) && (actual < 145))){
                        LCD_RectangleFill(104, 4, 20, 10, Green);
                        LCD_Delay(50);
                        LCD_RectangleFill(105, 5, 18, 8, Black);

                  }else if(((actual <= 95) && (actual > 82)) || ((actual
>= 145) && (actual < 157))){
                        LCD_RectangleFill(132, 4, 20, 10, Yellow);
                        LCD_Delay(50);
                        LCD_RectangleFill(133, 5, 18, 8, Black);

                  }else if((actual <= 82) || (actual >= 157) ){
                        LCD_RectangleFill(160, 4, 20, 10, Red);
                        LCD_Delay(50);
                        LCD_RectangleFill(161, 5, 18, 8, Black);
                  }
                  break;

            case 4:
                  if(((actual <= 95) && (actual > 70)) || ((actual >=
145) && (actual < 170))){
                        LCD_RectangleFill(104, 4, 20, 10, Green);
                        LCD_Delay(50);
                        LCD_RectangleFill(105, 5, 18, 8, Black);

                  }else if(((actual <= 70) && (actual > 45)) || ((actual
>= 170) && (actual < 195))){
                        LCD_RectangleFill(132, 4, 20, 10, Yellow);
                        LCD_Delay(50);
                        LCD_RectangleFill(133, 5, 18, 8, Black);

                  }else if((actual <= 45) || (actual >= 195) ){
                        LCD_RectangleFill(160, 4, 20, 10, Red);
                        LCD_Delay(50);
                        LCD_RectangleFill(161, 5, 18, 8, Black);
                  }
                  break;

            case 2:
                  if(((actual <= 70) && (actual > 45)) || ((actual >=
170) && (actual < 195))){
                        LCD_RectangleFill(104, 4, 20, 10, Green);
```

```
                        LCD_Delay(50);
                        LCD_RectangleFill(105, 5, 18, 8, Black);

                    }else if(((actual <= 45) && (actual > 20)) || ((actual
>= 195) && (actual < 220))){
                        LCD_RectangleFill(132, 4, 20, 10, Yellow);
                        LCD_Delay(50);
                        LCD_RectangleFill(133, 5, 18, 8, Black);

                    }else if((actual <= 20) || (actual >= 220) ){
                        LCD_RectangleFill(160, 4, 20, 10, Red);
                        LCD_Delay(50);
                        LCD_RectangleFill(161, 5, 18, 8, Black);
                    }
                    break;
      }/* End switch sensibilidad */

   }/* End if modo */

   //------ Z ---------------------------------------------------------/

   //------ Calculo de caída ------------------------------------------/

   valorMaxX = valorMaxX*valorMaxX;
   valorMaxY = valorMaxY*valorMaxY;
   valorMaxZ = valorMaxZ*valorMaxZ;

   total = valorMaxX + valorMaxY + valorMaxZ;
   total = sqrt(total);

   if(total >= limite){
      flagAvisoCaida = 1;

      LCD_RectangleFill(50, 175, 210, 50, Red);
      GUI_Text(55,195,"SE HA PRODUCIDO UNA CAIDA",White, Red);

      /* Espera a que se toque boton */
      while(flagBotonCaida !=1);

      LCD_Clear(Black);
      /* Dibujar Ejes ----------------------------------------------*/
      dibujar_Ejes(modo, sensibilidad);

      eje_x = 31;
      flagBotonCaida = 0;
      flagAvisoCaida = 0;
      g = 0;
      d = 0;
      k = 0;
   }/* End if total */

   //------ Calculo de caída ------------------------------------------/

   j++;
   if(j >= 100){
      j = 0;
   }
```

```
    /* Incremento del eje X */
    eje_x = eje_x + incremento_x;

    /*
    Si se llega al limite de la pantalla se inicia del principio y
dependiendo del sensor se empieza mas adelante o un poco mas atras.
    */
    if (eje_x >= 300){
      if(modo == 1){
            eje_x = 31;
      }else{
            eje_x = 21;
      }/* End if modo */
    }/* End if eje_x */

    /*
    Si se llega al final de la pantalla se necesita limpiar la señal, ya
que empezamos a representar desde el principio de nuevo.
    */
    if(limpiar == 1){
      limpiarY1 = almacen_DatosX[k];
      limpiarY2 = almacen_DatosX[k+1];

      if((limpiarY1 == 120) && (limpiarY2 == 120)){
            dibujar_Linea2((eje_x), limpiarY1,(eje_x + incremento_x),
limpiarY2, colorLimpiar);
      }else{
            dibujar_Linea2((eje_x + incremento_x), limpiarY2,(eje_x),
limpiarY1, Black);
      }

      limpiarY1 = almacen_DatosY[d];
      limpiarY2 = almacen_DatosY[d+1];

      if((limpiarY1 == 120) && (limpiarY2 == 120)){
            dibujar_Linea2((eje_x), limpiarY1,(eje_x + incremento_x),
limpiarY2, colorLimpiar);
      }else{
            dibujar_Linea2((eje_x + incremento_x), limpiarY2,(eje_x),
limpiarY1, Black);
      }

      limpiarY1 = almacen_DatosZ[g];
      limpiarY2 = almacen_DatosZ[g+1];

      if((limpiarY1 == 120) && (limpiarY2 == 120)){
            dibujar_Linea2((eje_x), limpiarY1,(eje_x + incremento_x),
limpiarY2, colorLimpiar);
      }else{
            dibujar_Linea2((eje_x + incremento_x), limpiarY2,(eje_x),
limpiarY1, Black);
      }

    }/* End if limpiar */

    /* Una vez leida la trama se inicia la recogida de otra */
    tramaOK = 0;
  }/* end if tramaOK */
```

```
}else if (modoDevice == 0){

    /* Si trama recibida correctamente */
    if(tramaOK == 1){

      /* Puntero de lectura al inicio de la cola */
      colaCirc.read = colaCirc.start;

      /* Se extrae el numero de registros de la cola */
      cola = pop(&colaCirc);
      registros = *cola;

      /* Se extrae la direccion del primer registro de la cola */
      cola = pop(&colaCirc);
      direccion = *cola;

      /*
      Se extraen los datos de la cola en 8 bits, se separan en 4 bits y
se le añade la cabecera para enviarlos por la UART hacia el dispositivo
movil.
      */
      l = 0;
      cab = 0;
      cab02 = 0x80;
      send3 = 0x00;

      for(l=0; l<length; l++){
            cola = pop(&colaCirc);
            datos[l] = (uint8_t)*cola;


            /* Eje X */
            if((l > 15) && (l < 20)){
                  send2 = (uint8_t)*cola;

                  while(USART_GetFlagStatus(USART2, USART_FLAG_TXE) ==
RESET);

                  send3 = (send2 & 0xF0) >> 4;
                  send3 = send3 | cab;

                  USART_SendData(USART2, send3);

                  send3= send3 + 1;
                  cab = cab + 16;

                  LCD_Delay(10);

                  while(USART_GetFlagStatus(USART2, USART_FLAG_TXE) ==
RESET);

                  send3 = (send2 & 0x0F);
                  send3 = send3 | cab;

                  USART_SendData(USART2, send3);

                  send3= send3 + 1;
```

```
                cab = cab + 16;
                LCD_Delay(10);
            }
        }/* end for datos */
      }/* End if tramaOK */
    }/* End if modoDevice */
  }/* end while(1) */
}/* end main */
```

Anexo V. Rutina de Servicio

En esta rutina de servicio se reciben los datos de la unidad inercial, en donde se comprueba si la cabecera se recibe y tras comprobar el checksum, se almacenan los datos en la cola circular.

```
void USART1_IRQHandler(void)
{
    uint16_t dato;
    uint16_t check = 0;
    int v = 0;

    /* Comprobacion de si la interrupcion es por recepcion */
    if(USART_GetITStatus(USART1, USART_IT_RXNE) != RESET){

      /* Comprobacion si recepcion de cabecera snp */
      if(tramaEntrante == 1){
            checksum = 's' + 'n' + 'p';

            /* Espera a recepcion del siguiente byte */
            while(USART_GetFlagStatus(USART1, USART_FLAG_RXNE) == RESET);
            dato = USART_ReceiveData(USART1);
            checksum = checksum + dato;

            /* Calculo de la longitud */
            if(((dato & 0x80) >> 7) == 1){
                  length = 4;
                  if((dato & 0x40) >> 6){
                        length = 4*((dato >> 2) & 0x0F);
                        numeroRegistro = ((dato >> 2) & 0x0F);
                  }
            }else{
                  length = 0;
                  if((dato & 0x01) == 1){
                        errorCF = 1;
                  }else{
                        errorCF = 0;
                  }
            }

            colaCirc.write = colaCirc.start;

            /* Espera a recepcion del siguiente byte */
            while(USART_GetFlagStatus(USART1, USART_FLAG_RXNE) == RESET);
            dato = USART_ReceiveData(USART1);

            /* Recepcion de los datos si la direccion es la requerida */
            if(dato == 0x61){

                  if(length != 0){
                        push(&colaCirc, numeroRegistro);
```

```
                        checksum = checksum + dato;
                        push(&colaCirc, dato);

                        for(v=0; v<length; v++){
                              /* Espera a recepcion del siguiente byte */
                              while(USART_GetFlagStatus(USART1,
USART_FLAG_RXNE) == RESET);
                              dato = USART_ReceiveData(USART1);
                              checksum = checksum + dato;
                              push(&colaCirc, dato);
                        }

                        }else{
                              /* Espera a recepcion del siguiente byte */
                              while(USART_GetFlagStatus(USART1,
USART_FLAG_RXNE) == RESET);
                              dato = USART_ReceiveData(USART1);
                              checksum = checksum + dato;
                              push(&colaCirc, dato);
                        }/* end if comprobacion longitud */

                        /* Espera a recepcion del siguiente byte */
                        while(USART_GetFlagStatus(USART1,
USART_FLAG_RXNE) == RESET);
                        dato = USART_ReceiveData(USART1);
                        check = (dato << 8);

                        /* Espera a recepcion del siguiente byte */
                        while(USART_GetFlagStatus(USART1,
USART_FLAG_RXNE) == RESET);
                        dato = USART_ReceiveData(USART1);
                        check = check | dato;

                        /* Comprobacion del checksum */
                        if(check == checksum){
                              tramaOK = 1;
                        }
                  }/* end if comprobacion de direccion */

                  tramaEntrante = 0;

            /* Recepcion de cabecera snp */
            }else{            /*(tramaEntrante == 0)*/
                  dato = USART_ReceiveData(USART1);

                  if(dato == cab1){
                        while(USART_GetFlagStatus(USART1,
USART_FLAG_RXNE) == RESET);
                        dato = USART_ReceiveData(USART1);

                        if(dato == cab2){
                              while(USART_GetFlagStatus(USART1,
USART_FLAG_RXNE) == RESET);
                              dato = USART_ReceiveData(USART1);

                              if(dato == cab3){
                                    tramaEntrante = 1;
                              }
```

```
                        }
                    }
                }

                USART_ClearITPendingBit(USART1, USART_IT_RXNE);
            }
        }
```

ANEXO V

Anexo VI. Funciones de la cola circular

Estas funciones son las que generan y manejan a la cola circular en donde se guardan los datos recibidos por la unidad inercial. También se muestra la estructura de dicha cola, que esta compuesta por 4 punteros, una variable entera y una array.

```
/* Estructura de la cola circular */
struct buffer_Circ{
	int size;
	uint16_t *start;
	uint16_t *end;
	uint16_t *read;
	uint16_t *write;
	uint16_t buffer[100];
};


/*
*	Función para crear buffer circular, se le pasa una estructura, un
*	array y el tamaño de la misma
*/
void init_Buffer(circular *buffer, int size2){
	buffer->size = size2;
	buffer->start = buffer->buffer;
	buffer->end = &buffer->buffer[size2-1];
	buffer->read = buffer->buffer;
	buffer->write = buffer->buffer;
}

/* Comprobar si el buffer esta lleno */
int buffer_Full(circular *buffer){
	if(buffer->write == buffer->end){
		return 1;
	}else{
		return 0;
	}
}

/* Comprobar si el buffer esta vacio */
int buffer_Empty(circular *buffer){
	if(buffer->write == buffer->start){
		return 1;
	}else{
		return 0;
	}
}

/*
*	Insertamos dato en el buffer y en el caso de estar lleno se empieza
*	a insertar desde el principio.
*/
```

```
void push(circular *buffer, uint16_t data) {
      if (buffer_Full(buffer)) {
            if(buffer->read > buffer->start){
                  buffer->write = buffer->start;
            }else if(buffer->write == buffer->read){
            }
      } else {
            if (buffer->write >= buffer->end & buffer->read > buffer-
>start) {
                  buffer->write = buffer->start;
            }
            *buffer->write = data;
            buffer->write++;
      }
}


/*
*     Sacamos dato en el buffer y en el caso de leer la ultima posicion
se *  inicia desde el principio, y otro caso es que se iguale a la
*     escritura
*/
uint16_t * pop(circular *buffer) {
      if (buffer_Empty(buffer)) {
            error = 1;
            return &error;
      } else {
            if (buffer->read >= buffer->end) {
                  buffer->read = buffer->start;
            }else if (buffer->read == buffer->write){
                  error = 2;
                  return &error;
            }
            return buffer->read++;
      }
}
```

Anexo VII. Funciones de los menús de inicio

Estas funciones representan el menú inicial del programa y como captar los parámetros seleccionados para configurar el programa.

```
/*
*     Funcion del primer menu, en donde se selecciona el dispositivo
*     para mostrar la señal de la unidad inercial.
*/
void select_Device(){
      uint16_t x, y, p;
      uint16_t oper = 0;
      uint8_t eb1 = 0, eb2 = 0;

      LCD_Clear(Blue);

      LCD_Button(50,100,100,50,0," Movil ",Red,Grey);
      LCD_Button(175,100,100,50,0," LCD ",Black,Grey);

      while(oper == 0){
            p = LCD_TouchRead(&display);  //Lee la posición del puntero
sobre la pantalla
            x = display.x;                //Extrae coordenada x
            y = display.y;                //Extrae coordenada y

            if(p == 1) {            //Detecta pulsación sobre la pantalla

                  /* Se comprueba que boton fue pulsado */

      if((eb1==0)&&(x>50)&&(x<150)&&(y>100)&&(y<150)){eb1=1,oper=1;}

      if((eb2==0)&&(x>175)&&(x<275)&&(y>100)&&(y<150)){eb2=1,oper=1;}

                  if(oper==1){            // Efecto de boton pulsado

                        if(eb1==1){
                              LCD_Button(50,100,100,50,1," Movil
",Red,Grey);
                              modoDevice = 0;
                        }

                        if(eb2==1){
                              LCD_Button(175,100,100,50,1," LCD
",Black,Grey);
                              modoDevice = 1;
                        }
                  }
            }
      }

      LCD_Delay(500);
```

```
}

/*
*     Funcion del segundo menu, en donde se selecciona el sensor a
*     visualizar, el tiempo
*     de pantalla, y en el caso de ser el acelerometro, la sensibilidad.
*/
void config_visual(){
      uint16_t x, y, p;
      uint16_t oper = 0, oper1 = 0;
      uint8_t em1 = 0, em2 = 0;
      uint8_t et1 = 0, et2 = 0, et3 = 0, et4 = 0, et5 = 0, et6 = 0;
      uint8_t eg1 = 0, eg2 = 0, eg3 = 0;

      LCD_Clear(Blue);

      //Se dibuja los dos CheckBox de seleccion de sensor
      GUI_Text(23,2,"Elija el modo:",White,Blue);
      LCD_CheckBox(20,20,0,Red);
      GUI_Text(45,20," Giroscopio ",Blue,White);
      LCD_CheckBox(20,40,0,Red);
      GUI_Text(45,40," Acelerometro ",Blue,White);

      //Se dibuja los CheckBox de seleccion del tiempo de pantalla
      GUI_Text(23,70,"Tiempo a mostrar:",White,Blue);
      LCD_CheckBox(20,90,0,Red);
      GUI_Text(45,90," 6 segundos ",Blue,White);
      LCD_CheckBox(20,110,0,Red);
      GUI_Text(45,110," 5 segundos ",Blue,White);
      LCD_CheckBox(20,130,0,Red);
      GUI_Text(45,130," 4 segundos ",Blue,White);
      LCD_CheckBox(20,150,0,Red);
      GUI_Text(45,150," 3 segundos ",Blue,White);
      LCD_CheckBox(20,170,0,Red);
      GUI_Text(45,170," 2 segundos ",Blue,White);
      LCD_CheckBox(20,190,0,Red);
      GUI_Text(45,190," 1 segundos ",Blue,White);

      while(oper == 0){
            p = LCD_TouchRead(&display);  //Lee la posición del puntero
sobre la pantalla
            x = display.x;                //Extrae coordenada x
            y = display.y;                //Extrae coordenada y

            if(p == 1) {      //Detecta pulsación sobre la pantalla

                  /* Detecta alguno de los CheckBox pulsado */
                  /* Sensores */

      if((em1==0)&&(x>20)&&(x<36)&&(y>20)&&(y<36)){em1=1,oper=1;}

      if((em2==0)&&(x>20)&&(x<36)&&(y>40)&&(y<56)){em2=1,oper=1;}

                  /* Tiempo */

      if((et1==0)&&(x>20)&&(x<36)&&(y>190)&&(y<206)){et1=1,oper=1;}

      if((et2==0)&&(x>20)&&(x<36)&&(y>170)&&(y<186)){et2=1,oper=1;}
```

```
        if((et3==0)&&(x>20)&&(x<36)&&(y>150)&&(y<176)){et3=1,oper=1;}

        if((et4==0)&&(x>20)&&(x<36)&&(y>130)&&(y<156)){et4=1,oper=1;}

        if((et5==0)&&(x>20)&&(x<36)&&(y>110)&&(y<126)){et5=1,oper=1;}

        if((et6==0)&&(x>20)&&(x<36)&&(y>90)&&(y<106)){et6=1,oper=1;}

                    if(oper==1)
                    {
                    //Indicación de "Activado" en etiquetas de los CheckBox
                          if(em1==1){
                                LCD_CheckBox(20,20,1,Red);
                                GUI_Text(45,20," Giroscopio ",Red,White);
                                LCD_CheckBox(20,40,0,Red);
                                GUI_Text(45,40," Acelerometro
",Blue,White);
                                gyroOK = 1;
                                accelOK = 0;
                                sensibilidad = 2000;
                                oper=0;
                                p = 0;
                          }

                          if(em2==1){
                                LCD_CheckBox(20,20,0,Red);
                                GUI_Text(45,20," Giroscopio ",Blue,White);
                                LCD_CheckBox(20,40,1,Red);
                                GUI_Text(45,40," Acelerometro ",Red,White);
                                gyroOK = 0;
                                accelOK = 1;
                                oper=0;
                                p = 0;
                          }

                          if(et1==1){
                                LCD_CheckBox(20,190,1,Red);
                                GUI_Text(45,190," 1 segundos ",Blue,White);
                                tempo = 1;
                                oper = 1;
                                p = 0;
                          }

                          if(et2==1){
                                LCD_CheckBox(20,170,1,Red);
                                GUI_Text(45,170," 2 segundos ",Blue,White);
                                tempo = 2;
                                oper = 1;
                                p = 0;
                          }

                          if(et3==1){
                                LCD_CheckBox(20,150,1,Red);
                                GUI_Text(45,150," 3 segundos ",Blue,White);
                                tempo = 3;
                                oper = 1;
                                p = 0;
```

```
                        }

                        if(et4==1){
                              LCD_CheckBox(20,130,1,Red);
                              GUI_Text(45,130," 4 segundos ",Blue,White);
                              tempo = 4;
                              oper = 1;
                              p = 0;
                        }

                        if(et5==1){
                              LCD_CheckBox(20,110,1,Red);
                              GUI_Text(45,110," 5 segundos ",Blue,White);
                              tempo = 5;
                              oper = 1;
                              p = 0;
                        }

                        if(et6==1){
                              LCD_CheckBox(20,90,1,Red);
                              GUI_Text(45,90," 6 segundos ",Blue,White);
                              tempo = 6;
                              oper = 1;
                              p = 0;
                        }

                        LCD_Delay(500);

                  }
            }
      }

      /* Si se pulso el acelerometro se muestra la seleccion de
sensibilidad */
      if(accelOK == 1){
            GUI_Text(200,55,"Sensibilidad",White,Blue);
            GUI_Text(200,70,"acelerometro:",White,Blue);
            LCD_CheckBox(200,90,0,Red);
            GUI_Text(225,90," 8g ",Blue,White);
            LCD_CheckBox(200,110,0,Red);
            GUI_Text(225,110," 4g ",Blue,White);
            LCD_CheckBox(200,130,0,Red);
            GUI_Text(225,130," 2g ",Blue,White);

            while(oper1 == 0){
                  p = LCD_TouchRead(&display);  //Lee la posición del
puntero sobre la pantalla
                  x = display.x;                //Extrae coordenada x
                  y = display.y;                //Extrae coordenada y

                  if(p == 1) {      //Detecta pulsación sobre la pantalla

                        /* Detecta alguno de los CheckBox pulsado */

      if((eg1==0)&&(x>200)&&(x<216)&&(y>90)&&(y<106)){eg1=1,oper1=1;}

      if((eg2==0)&&(x>200)&&(x<216)&&(y>110)&&(y<126)){eg2=1,oper1=1;}
```

```
        if((eg3==0)&&(x>200)&&(x<216)&&(y>130)&&(y<146)){eg3=1,oper1=1;}

                        if(oper1==1){

                              if(eg1==1){
                                    LCD_CheckBox(200,90,1,Red);
                                    GUI_Text(225,90," 8g ",Blue,White);
                                    sensibilidad = 8;
                              }

                              if(eg2==1){
                                    LCD_CheckBox(200,110,1,Red);
                                    GUI_Text(225,110," 4g ",Blue,White);
                                    sensibilidad = 4;
                              }

                              if(eg3==1){
                                    LCD_CheckBox(200,130,1,Red);
                                    GUI_Text(225,130," 2g ",Blue,White);
                                    sensibilidad = 2;
                              }

                              LCD_Delay(500);
                        }

                  }
            }
      }

      LCD_Clear(Black);
}
```

ANEXO VII

Anexo VIII. Listado de funciones de la librería del GPIO

En este anexo se muestran todas las funciones que componen la librería de la GPIO, facilitada por el fabricante de la tarjeta HY Mini-STM32V. Únicamente las funciones utilizadas de esta librería son las de configuración de la GPIO.

```
void GPIO_DeInit(GPIO_TypeDef* GPIOx);
void GPIO_AFIODeInit(void);
void GPIO_Init(GPIO_TypeDef* GPIOx, GPIO_InitTypeDef* GPIO_InitStruct);
void GPIO_StructInit(GPIO_InitTypeDef* GPIO_InitStruct);
uint8_t GPIO_ReadInputDataBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
uint16_t GPIO_ReadInputData(GPIO_TypeDef* GPIOx);
uint8_t GPIO_ReadOutputDataBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
uint16_t GPIO_ReadOutputData(GPIO_TypeDef* GPIOx);
void GPIO_SetBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
void GPIO_ResetBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
void GPIO_WriteBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, BitAction
BitVal);
void GPIO_Write(GPIO_TypeDef* GPIOx, uint16_t PortVal);
void GPIO_PinLockConfig(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
void GPIO_EventOutputConfig(uint8_t GPIO_PortSource, uint8_t
GPIO_PinSource);
void GPIO_EventOutputCmd(FunctionalState NewState);
void GPIO_PinRemapConfig(uint32_t GPIO_Remap, FunctionalState NewState);
void GPIO_EXTILineConfig(uint8_t GPIO_PortSource, uint8_t
GPIO_PinSource);
void GPIO_ETH_MediaInterfaceConfig(uint32_t GPIO_ETH_MediaInterface);
```

ANEXO VIII

Anexo IX. Listado de funciones de la librería de la USART

En este anexo se muestran todas las funciones que componen la librería de la USART, facilitada por el fabricante de la tarjeta HY Mini-STM32V. Cabe destacar que las funciones más utilizadas de esta librería, son las funciones de envío y recepción de datos, `USART_SendData()` y `USART_ReceiveData()`, respectivamente.

```
void USART_DeInit(USART_TypeDef* USARTx);
void USART_Init(USART_TypeDef* USARTx, USART_InitTypeDef*
USART_InitStruct);
void USART_StructInit(USART_InitTypeDef* USART_InitStruct);
void USART_ClockInit(USART_TypeDef* USARTx, USART_ClockInitTypeDef*
USART_ClockInitStruct);
void USART_ClockStructInit(USART_ClockInitTypeDef*
USART_ClockInitStruct);
void USART_Cmd(USART_TypeDef* USARTx, FunctionalState NewState);
void USART_ITConfig(USART_TypeDef* USARTx, uint16_t USART_IT,
FunctionalState NewState);
void USART_DMACmd(USART_TypeDef* USARTx, uint16_t USART_DMAReq,
FunctionalState NewState);
void USART_SetAddress(USART_TypeDef* USARTx, uint8_t USART_Address);
void USART_WakeUpConfig(USART_TypeDef* USARTx, uint16_t USART_WakeUp);
void USART_ReceiverWakeUpCmd(USART_TypeDef* USARTx, FunctionalState
NewState);
void USART_LINBreakDetectLengthConfig(USART_TypeDef* USARTx, uint16_t
USART_LINBreakDetectLength);
void USART_LINCmd(USART_TypeDef* USARTx, FunctionalState NewState);
void USART_SendData(USART_TypeDef* USARTx, uint16_t Data);
uint16_t USART_ReceiveData(USART_TypeDef* USARTx);
void USART_SendBreak(USART_TypeDef* USARTx);
void USART_SetGuardTime(USART_TypeDef* USARTx, uint8_t USART_GuardTime);
void USART_SetPrescaler(USART_TypeDef* USARTx, uint8_t USART_Prescaler);
void USART_SmartCardCmd(USART_TypeDef* USARTx, FunctionalState NewState);
void USART_SmartCardNACKCmd(USART_TypeDef* USARTx, FunctionalState
NewState);
void USART_HalfDuplexCmd(USART_TypeDef* USARTx, FunctionalState
NewState);
void USART_OverSampling8Cmd(USART_TypeDef* USARTx, FunctionalState
NewState);
void USART_OneBitMethodCmd(USART_TypeDef* USARTx, FunctionalState
NewState);
void USART_IrDAConfig(USART_TypeDef* USARTx, uint16_t USART_IrDAMode);
void USART_IrDACmd(USART_TypeDef* USARTx, FunctionalState NewState);
FlagStatus USART_GetFlagStatus(USART_TypeDef* USARTx, uint16_t
USART_FLAG);
void USART_ClearFlag(USART_TypeDef* USARTx, uint16_t USART_FLAG);
ITStatus USART_GetITStatus(USART_TypeDef* USARTx, uint16_t USART_IT);
void USART_ClearITPendingBit(USART_TypeDef* USARTx, uint16_t USART_IT);
```

ANEXO IX

Anexo X. Listado de funciones de la librería del EXTI

En este anexo se muestran todas las funciones que componen la librería de la EXTI, facilitada por el fabricante de la tarjeta HY Mini-STM32V. Únicamente las funciones utilizadas de esta librería son las de configuración de la EXTI y las de limpiar interrupción.

```
void EXTI_DeInit(void);
void EXTI_Init(EXTI_InitTypeDef* EXTI_InitStruct);
void EXTI_StructInit(EXTI_InitTypeDef* EXTI_InitStruct);
void EXTI_GenerateSWInterrupt(uint32_t EXTI_Line);
FlagStatus EXTI_GetFlagStatus(uint32_t EXTI_Line);
void EXTI_ClearFlag(uint32_t EXTI_Line);
ITStatus EXTI_GetITStatus(uint32_t EXTI_Line);
void EXTI_ClearITPendingBit(uint32_t EXTI_Line);
```

ANEXO X

Anexo XI. Listado de funciones de la librería del GLCD

En este anexo se muestran todas las funciones que componen la librería del GLCD, facilitada por el fabricante de la tarjeta HY Mini-STM32V. Las funciones mñs utilizadas de esta librería son las funciones de dibujar línea, `LCD_DrawLine()`, y mostrar mensajes en pantalla, `GUI_Text()`.

```
void LCD_Delay(uint16_t nCount);
void LCD_Initializtion(void);
void LCD_Clear(uint16_t Color);
uint16_t LCD_GetPoint(uint16_t Xpos,uint16_t Ypos);
void LCD_SetPoint(uint16_t Xpos,uint16_t Ypos,uint16_t point);
void LCD_DrawLine( uint16_t x0, uint16_t y0, uint16_t x1, uint16_t y1 ,
uint16_t color );
void PutChar( uint16_t Xpos, uint16_t Ypos, uint8_t ASCI, uint16_t
charColor, uint16_t bkColor );
void GUI_Text(uint16_t Xpos, uint16_t Ypos, uint8_t *str,uint16_t Color,
uint16_t bkColor);
void PutChinese(uint16_t Xpos,uint16_t Ypos,uint8_t *str,uint16_t
Color,uint16_t bkColor);
void GUI_Chinese(uint16_t Xpos, uint16_t Ypos, uint8_t *str,uint16_t
Color, uint16_t bkColor);
void LCD_Rectangle(uint16_t x, uint16_t y, uint16_t longx, uint16_t
longy, uint8_t grosor, uint16_t color);
void LCD_RectangleFill(uint16_t x, uint16_t y, uint16_t longx, uint16_t
longy, uint16_t color);
void LCD_CheckBox(uint16_t x, uint16_t y, uint8_t p, uint16_t ink);
uint8_t LCD_FormatRead(void);
void LCD_Button(uint16_t x, uint16_t y, uint16_t longx, uint16_t longy,
uint8_t p, uint8_t* text, uint16_t ink, uint16_t bkcolor);
```

ANEXO XI

Anexo XII. Listado de funciones de la librería del TouchPanel.

En este anexo se muestran todas las funciones que componen la librería del TouchPanel, facilitada por el fabricante de la tarjeta HY Mini-STM32V. La función mas utilizada de esta librería es la de facilitar el punto en donde se ha tocado la pantalla, `getDisplayPoint()`.

```
void TP_Init(void);
Coordinate *Read_Ads7846(void);
void TouchPanel_Calibrate(void);
void DrawCross(uint16_t Xpos, uint16_t Ypos);
void TP_DrawPoint(uint16_t Xpos, uint16_t Ypos);
FunctionalState setCalibrationMatrix( Coordinate * displayPtr, Coordinate
* screenPtr, Matrix * matrixPtr);
FunctionalState getDisplayPoint(Coordinate * displayPtr, Coordinate *
screenPtr, Matrix * matrixPtr );
uint8_t LCD_TouchRead(Coordinate * displayPtr);
void GetFormat(void);
```

ANEXO XII

Anexo XIII. Ejemplo de representación en Android

En este anexo se adjunta una clase de ejemplo de representación de la señal enviada por la tarjeta HY Mini-STM32 en un dispositivo Android.

```
package es.amsr.healthanalyzer;

import android.app.Activity;
import android.content.Intent;
import android.graphics.Color;
import android.os.Handler;
import android.os.Message;
import android.os.Bundle;
import android.view.View;
import android.content.Context;
import android.widget.TextView;
import android.widget.ToggleButton;

import com.jjoe64.graphview.GraphView;
import com.jjoe64.graphview.series.DataPoint;
import com.jjoe64.graphview.series.LineGraphSeries;

public class MainActivityAC extends Activity {

    private final Handler mHandler = new Handler();
    private LineGraphSeries<DataPoint> mSeriesX;
    private LineGraphSeries<DataPoint> mSeriesZ;
    private double graph2LastXValue = 0d;

    /* thread to read the data*/
    public handler_thread handlerThread;

    /* declare a FT311 UART interface variable */
    public FT311UARTInterface uartInterface;

    /* local variables */
    byte[] writeBuffer;
    byte[] readBuffer;
    int[] actualNumBytes;

    int numBytes;
    byte status;

    /*UART Configuration variables*/
    int baudRate;
    byte stopBit;
    byte dataBit;
    byte parity;
    byte flowControl;
    public Context global_context;
    public boolean bConfiged = false;


    /*Play and Stop*/
    boolean running = true;
```

```
    /*Data received from UART*/
    int[] dataReceived = new int[4096];
    int read = 0;
    int read2=0;
    int write = 0;

    String configuration;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        Intent intent = getIntent();
        configuration =
intent.getStringExtra(Menu_activity.EXTRA_MESSAGE);

        setContentView(R.layout.activity_main_ac);
        global_context = this;

        /* allocate buffer */
        writeBuffer = new byte[64];
        readBuffer = new byte[4096];
        actualNumBytes = new int[1];

        init_UART();
        init_Graph();

        handlerThread = new handler_thread(handler);
        handlerThread.start();
    }

    @Override
    protected void onResume() {
        // Ideally should implement onResume() and onPause()
        // to take appropriate action when the activity looses focus
        super.onResume();
        if (2 == uartInterface.ResumeAccessory()) {
            baudRate = 115200;
            stopBit = 1;
            dataBit = 8;
            parity = 0;
            flowControl = 0;
        }
        Runnable mTimer = new Runnable() {
            @Override
            public void run() {

                if (running) {
                    for(int i=0; i<5;i++){
                        graph2LastXValue += 1d;
                        mSeriesX.appendData(new
DataPoint(graph2LastXValue, generateDataX()), true, 160);
                        mSeriesZ.appendData(new
DataPoint(graph2LastXValue, generateDataZ()), true, 160);
                    }
                }
                mHandler.postDelayed(this, 1000);
```

```
            }
        };
        mHandler.postDelayed(mTimer, 500); //tiempo que tarda en empezar
a dibujar
    }

    @Override
    protected void onPause() {
        super.onPause();
    }

    @Override
    protected void onStop() {
        super.onStop();
    }

    @Override
    protected void onDestroy() {
        uartInterface.DestroyAccessory(bConfiged);
        super.onDestroy();
    }

    final Handler handler = new Handler() {
        @Override
        public void handleMessage(Message msg) {

            for (int i = 0; i < actualNumBytes[0]; i++) {
                dataReceived[write] = Integer.valueOf((readBuffer[i] &
0xff));
                write++;
                if (write == 4095) {
                    write = 0;
                }
            }
        }
    };

    /* usb input data handler */
    private class handler_thread extends Thread {
        Handler mHandler;

        /* constructor */
        handler_thread(Handler h) {
            mHandler = h;
        }

        public void run() {
            Message msg;

            while (true) {

                try {
                    Thread.sleep(200);

                } catch (InterruptedException e) {}

                status = uartInterface.ReadData(4096, readBuffer,
actualNumBytes);
```

```
                if (status == 0x00 && actualNumBytes[0] > 0) {
                    msg = mHandler.obtainMessage();
                    mHandler.sendMessage(msg);
                }
            }
        }
    }

    private double generateDataX() {
        double result = 0;
        int dataFirstFrame = dataReceived[read];
        int dataSecondFrame = dataReceived[read + 1];
        int dataThirdFrame= dataReceived[read + 2];
        int dataFourthFrame= dataReceived[read + 3];
        int dataFifthFrame=dataReceived[read + 4];
        int dataSixthFrame=dataReceived[read + 5];
        int dataSeventhFrame=dataReceived[read + 6];
        int dataEighthFrame=dataReceived[read + 7];
        int firstFrameMask = 0x00;
        int secondFrameMask = 0x10;
        int thirdMask=0x20;
        int fourthMask=0x30;
        int fifthMask=0x40;
        int sixthMask=0x50;
        int seventhMask=0x60;
        int eighthMask=0x70;
        int completedFrame=0;
        float frame=0;
        boolean frameOK=false;
        TextView trama= (TextView)findViewById(R.id.trama);


        if ((dataFirstFrame & firstFrameMask) != firstFrameMask) {
            read++;
        } else if ((dataFirstFrame & firstFrameMask) == firstFrameMask) {
            dataFirstFrame = ((dataFirstFrame & 0x0F) << 28);
            read++;
            if (read == 4095) {
                read = 0;
            }
            if ((dataSecondFrame & secondFrameMask) == secondFrameMask) {
                dataSecondFrame = ((dataSecondFrame & 0x0F)<<24);
                read++;
                if (read == 4095) {
                    read = 0;
                }

                if ((dataThirdFrame & thirdMask) == thirdMask) {
                    dataThirdFrame = ((dataThirdFrame & 0x0F)<<20);
                    read++;
                    if (read == 4095) {
                        read = 0;
                    }

                    if ((dataFourthFrame & fourthMask) == fourthMask) {
                        dataFourthFrame = ((dataFourthFrame & 0x0F) <<
16);
```

```
                        read++;
                        if (read == 4095) {
                            read = 0;
                        }

                        if ((dataFifthFrame & fifthMask) == fifthMask) {
                            dataFifthFrame = ((dataFifthFrame & 0x0F) <<
12);
                            read++;
                            if (read == 4095) {
                                read = 0;
                            }

                            if ((dataSixthFrame & sixthMask) ==
sixthMask) {
                                dataSixthFrame = ((dataSixthFrame & 0x0F)
<< 8);
                                read++;
                                if (read == 4095) {
                                    read = 0;
                                }

                                if ((dataSeventhFrame & seventhMask) ==
seventhMask) {
                                    dataSeventhFrame = ((dataSeventhFrame
& 0x0F) << 4);
                                    read++;
                                    if (read == 4095) {
                                        read = 0;
                                    }

                                    if ((dataEighthFrame & eighthMask) ==
eighthMask) {
                                        dataEighthFrame =
((dataEighthFrame & 0x0F));
                                        frameOK=true;
                                        read++;
                                        if (read == 4095) {
                                            read = 0;
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
            if(frameOK==true) {
                completedFrame = (dataFirstFrame | dataSecondFrame |
dataThirdFrame | dataFourthFrame
                        | dataFifthFrame | dataSixthFrame |
dataSeventhFrame | dataEighthFrame);
                frame=Float.intBitsToFloat(completedFrame);
                result=(double)frame;
                trama.setText(trama.getText()+" "+result);

            }else{
                result=0;
```

```
            }
        }

    return result;
    }

    private double generateDataZ() {
        double result2 = 0;
        int dataFirstFrame2 = dataReceived[read2+8];
        int dataSecondFrame2 = dataReceived[read2 + 9];
        int dataThirdFrame2= dataReceived[read2 + 10];
        int dataFourthFrame2= dataReceived[read2 + 11];
        int dataFifthFrame2=dataReceived[read2 + 12];
        int dataSixthFrame2=dataReceived[read2 + 13];
        int dataSeventhFrame2=dataReceived[read2 + 14];
        int dataEighthFrame2=dataReceived[read2 + 15];
        int firstFrameMask2 = 0x80;
        int secondFrameMask2 = 0x90;
        int thirdMask2=0xA0;
        int fourthMask2=0xB0;
        int fifthMask2=0xC0;
        int sixthMask2=0xD0;
        int seventhMask2=0xE0;
        int eighthMask2=0xF0;
        int completedFrame2=0;
        float frame2=0;
        boolean frameOK2=false;

        if ((dataFirstFrame2 & firstFrameMask2) != firstFrameMask2) {
            read2++;
            if (read2 == 4095) {
                read2 = 0;
            }
        } else if ((dataFirstFrame2 & firstFrameMask2) ==
firstFrameMask2) {
            dataFirstFrame2 = ((dataFirstFrame2 & 0x0F) << 28);
            read2++;
            if (read2 == 4095) {
                read2 = 0;
            }

            if ((dataSecondFrame2 & secondFrameMask2) ==
secondFrameMask2) {
                dataSecondFrame2 = ((dataSecondFrame2 & 0x0F)<<24);
                read2++;
                if (read2 == 4095) {
                    read2 = 0;
                }

                if ((dataThirdFrame2 & thirdMask2) == thirdMask2) {
                    dataThirdFrame2 = ((dataThirdFrame2 & 0x0F)<<20);
                    read2++;
                    if (read2 == 4095) {
                        read2 = 0;
                    }

                    if ((dataFourthFrame2 & fourthMask2) == fourthMask2)
{
```

```
                        dataFourthFrame2 = ((dataFourthFrame2 & 0x0F) <<
16);
                        read2++;
                        if (read2 == 4095) {
                            read2 = 0;
                        }

                        if ((dataFifthFrame2 & fifthMask2) == fifthMask2)
{
                            dataFifthFrame2 = ((dataFifthFrame2 & 0x0F)
<< 12);
                            read2++;
                            if (read2 == 4095) {
                                read2 = 0;
                            }

                            if ((dataSixthFrame2 & sixthMask2) ==
sixthMask2) {
                                dataSixthFrame2 = ((dataSixthFrame2 &
0x0F) << 8);
                                read2++;
                                if (read2 == 4095) {
                                    read2 = 0;
                                }

                                if ((dataSeventhFrame2 & seventhMask2) ==
seventhMask2) {
                                    dataSeventhFrame2 =
((dataSeventhFrame2 & 0x0F) << 4);
                                    read2++;
                                    if (read2 == 4095) {
                                        read2 = 0;
                                    }

                                    if ((dataEighthFrame2 & eighthMask2)
== eighthMask2) {
                                        dataEighthFrame2 =
((dataEighthFrame2 & 0x0F));
                                        frameOK2=true;
                                        read2++;
                                        if (read2 == 4095) {
                                            read2 = 0;
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }

            if(frameOK2==true) {
                completedFrame2 = (dataFirstFrame2 | dataSecondFrame2 |
dataThirdFrame2 | dataFourthFrame2
                        | dataFifthFrame2 | dataSixthFrame2 |
dataSeventhFrame2 | dataEighthFrame2);
                frame2=Float.intBitsToFloat(completedFrame2);
                result2=(double)frame2;
```

```
            }else{
                result2=0;
            }
        }
        return result2;
    }


    public void init_UART() {
        baudRate = 115200; /* baud rate */
        stopBit = 1; /* 1:1stop bits, 2:2 stop bits */
        dataBit = 8;/* 8:8bit, 7: 7bit */
        parity = 0; /* 0: none, 1: odd, 2: even, 3: mark, 4: space */
        flowControl = 0; /* 0:none, 1: flow control(CTS,RTS) */

        uartInterface = new FT311UARTInterface(this);
        uartInterface.SetConfig(baudRate, stopBit, dataBit, parity,
flowControl);

    }

    public void init_Graph() {
        GraphView graph = (GraphView) findViewById(R.id.graph);
        mSeriesX = new LineGraphSeries<DataPoint>();
        mSeriesZ = new LineGraphSeries<DataPoint>();
        graph.addSeries(mSeriesX);
        graph.addSeries(mSeriesZ);
        mSeriesX.setColor(Color.RED);
        mSeriesZ.setColor(Color.GREEN);
        graph.getViewport().setScrollable(true);
        graph.getViewport().setScalable(true);
        graph.getViewport().setYAxisBoundsManual(true);
        graph.getViewport().setXAxisBoundsManual(true);
        graph.getViewport().setMinY(-8);
        graph.getViewport().setMaxY(8);
        graph.getViewport().setMinX(0);
        graph.getViewport().setMaxX(160);

        graph.setBackgroundColor(0xFF0F3B05);
        graph.getGridLabelRenderer().setGridColor(0x4490EE90);

graph.getGridLabelRenderer().setHorizontalLabelsColor(0xFF0F3B05);
        graph.getGridLabelRenderer().setVerticalLabelsColor(0xFF0F3B05);
        graph.getGridLabelRenderer().setNumHorizontalLabels(27);
        graph.getGridLabelRenderer().setNumVerticalLabels(15);
        graph.getGridLabelRenderer().setHighlightZeroLines(false);

    }

    public void playAndStop(View view) {
        GraphView graph = (GraphView) findViewById(R.id.graph);
        ToggleButton button_playStop = (ToggleButton)
findViewById(R.id.button_playStop);
        boolean on = button_playStop.isChecked();
        if (!on) {
            running = false;
            graph.getViewport().setScalable(true);
```

```
            } else {
                running = true;
            }
        }
    }
```