Facultad de Informática
UNIVERSIDADE DA CORUÑA

TRABAJO FIN DE GRADO
GRAO EN INGENIERÍA INFORMÁTICA
MENCIÓN EN TECNOLOGÍAS DE LA INFORMACIÓN

ArduImu: Unidad de medición inercial basada en arduino

Estudiante: José Luis Mosquera Lorenzo
Dirección: Juan Ramón Rabuñal Dopico
Alberto José Alvarellos González

A Coruña, septiembre de 2019.

A mi familia por apoyarme en esta etapa

Agradecimientos

A mis tutores Juan y Alberto por orientarme al realizar este proyecto y ayudarme a abordar las dificultades que han surgido.

Agradecer a mis amigos, Alba, Cris, David, Laura, Martín, Pablo, Sandra..., el haberme ayudado en los momentos que lo necesitaba y por hacerme pasar buenos momentos que recordar para siempre.

Resumen

El objetivo principal de este proyecto de fin de grado es realizar el diseño hardware y software, además de su posterior implementación de una unidad de medición inercial en arduino. Para cumplir ese objetivo principal establecimos algunos objetivos secundarios:

- Crear el software para el control de la placa de arduino.
- Interacción con un Imu a través de arduino.
- Registro de datos en una tarjeta sd en arduino.
- Crear una aplicación de escritorio para poder interactuar con arduino desde el pc.
- La posibilidad de ver en tiempo real los movimientos del imu desde el pc.
- Sincronización de un RTC con la hora del ordenador al que la placa arduino este conectada.
- Visualización en el pc de los datos guardados en la tarjeta SD.
- Establecer un modo sueño para que el sistema arduino consuma menos energía.

Para poder cumplir estos objetivos necesitamos algunos componentes hardware:

- Arduino MkrZero.
- Una unidad de medición inercial.
- Un RTC.
- Un pulsador.
- Tarjeta micro sd.

A mayores de los objetivos anteriores hemos alcanzado otros según íbamos avanzando en el desarrollo del proyecto, como evitar que la interfaz de usuario se congelase al recibir datos desde arduino por el puerto serie y mostrarlos en la pantalla. Poder guardar datos en la sd para poder cargarlos en el arranque de la placa de arduino. El realizar este proyecto me ha permitido adquirir una serie de conocimientos que antes no tenía o aumentar los conocimientos ya existentes. Algunos de ellos fueron: conseguir como el hardware y el software se unan para trabajar juntos a la perfección y en total sincronía, como a un sistema hardware se le pueden añadir más elementos para poder crear un sistema más complejo. A parte de aprender a conectar hardware y software, he conocido otras formas de crear interfaces de usuario en java basadas en xml. Al tener que actualizar esa interfaz de usuario con datos que provenientes del puerto serie he tenido que usar threads para evitar que se congelara dicha interfaz. El desarrollo del proyecto se realizó con una metodología basada en scrum con sprints o iteraciones. Cada sprint constaba de diversas fases: análisis, diseño, implementación y pruebas. Algunas fases podían estar divididas en subfases, por ejemplo, diseño comprendía tres categorías: diseño hardware, diseño software y diseño de la interfaz de usuario.

Destacar que este proyecto se realizó para poder observar los movimientos de un objeto que realiza en una superficie, aunque puede tener otros usos. Ya que actualmente las unidades de medición inerciales las encontramos en cualquier sitio, incluso en nuestros móviles, drones, etc. Para finalizar este resumen, decir que este proyecto solo araña una parte de lo que se puede hacer con una placa de arduino, porque a pesar de haberlo finalizado, aún se pueden realizar muchas mejoras, una de ellas sería conectar la placa a internet, para poder obtener los datos de manera remota sin tener que acercarnos al lugar donde este y poder guardar los datos en un servidor u otro dispositivo remoto, ya que la memoria de la tarjeta sd es más limitada.

Abstract

The main objective of this end-of-grade project is to carry out the hardware and software design, as well as the subsequent implementation of an arduino based inertial measurement unit. In order to achieve this main objective, we established some secondary objectives:

- Create the software for the control of the arduino board
- Interaction with an Imu through arduino.
- Data logging on an sd card in arduino.
- Create a desktop application to be able to interact with arduino from the pc
- The possibility to see in real time the movements of the imu from the pc.

- Synchronization of a RTC with the time of the computer to which arduino is connected.
- Visualization in the pc of the data saved in SD.
- Establish a sleep mode so that the arduino system consumes less energy.

In order to meet these objectives we need some hardware components:

- Arduino MkrZero
- A inertial measuring unit.
- A RTC.
- A pushbutton.
- Micro sd card.

In addition to the above objectives, we achieved others as we progressed in the development of the project, such as preventing the user interface from freezing when receiving data from arduino through the serial port and displaying it on the screen. To be able to save data in the sd to be able to load them in the start of the arduino board. This project has allowed me to acquire a series of knowledge that I did not have before or increase the existing knowledge. Some of them were: to get as the hardware and the software unite to work together to the perfection and in total synchrony, as to a hardware system more elements can be added to him to be able to create a more complex system. Apart from learning to connect hardware and software, I have known other ways to create user interfaces in java based on xml.Having to update that user interface with data from the serial port, I had to use threads to avoid freezing that interface. The development of the project was done with a scrum-based methodology with sprints or iterations. Each sprint consisted of several phases: analysis, design, implementation and testing. This phases could be divided into subphases, for example, design comprised two categories: hardware design and software design.

To end this summary, say that this project only scratches a part of what can be done with an arduino board, because despite having finished, many improvements can still be made, one of them would be to connect the board to the Internet and be able to obtain data remotely without having to go near the place where it is and be able to save the data on a server or other remote device because the memory of the sd card is more limited.

Palabras clave:

- Unidad de medición inercial.
- Giroscopio.
- Acelerómetro.
- Magnetómetro.
- Arduino.
- C++.
- Java.
- JavaFx.
- Hilos.
- Comunicación serie.

Keywords:

- Inertial measurement unit.
- Gyroscope.
- Accelerometer.
- Magnetometer.
- Arduino.
- C++.
- Java.
- JavaFx.
- Threads.
- Serial Comunication.

Índice general

Índice de figuras

Índice de tablas

Ejemplos de código

Capítulo 1

Introducción

En este capítulo de introducción abordaremos la motivación para realizar este proyecto, sus objetivos y lo más importante explicar que es una imu, algo de historia y sus aplicaciones.

1.1 Motivación

Este proyecto se ha realizado para crear una herramienta que permita capturar los movimientos de un objeto cuando esta sobre una superficie inestable y aunque pueda parecer que está parado, puede estar realizando algunos movimientos que no podemos percibir con nuestros ojos. Para explicar mejor esa frase pondremos un ejemplo, imaginémonos una barca en un lago, normalmente son aguas tranquilas y el movimiento capturado tendría poca variación, pero si esta misma barca la ponemos en el mar en un día que este picado, el registro de movimientos sería muy variable. Pues una vez que tenemos esos datos se los pasamos a otro sistema, esté sistema podría averiguar dónde estaba la barca analizando dichos datos. Entonces la funcionalidad principal del proyecto es crear un sistema que capture, registre los datos y nos proporcione una manera de visualizar los datos registrados y los movimientos en tiempo real. Además de utilizarse para capturar el movimiento de objetos podemos utilizarlo en las personas. Los smartwatches más modernos incluyen estos sensores como detector de caídas y en caso de que no realices ningún movimiento después de la caída avisara automáticamente a emergencias.

1.2 Objetivos

Como mencionamos en la sección anterior, el objetivo del proyecto es crear un sistema para capturar los movimientos que realiza un objeto, personas, animales, etc. Para poder alcanzar ese objetivo, tendremos que desarrollar e implementar un circuito electrónico y el software para poder capturar los datos proporcionados por ese circuito. Para hacer el proyecto más

sencillo dividimos ese gran objetivo en varios subobjetivos:

- Crear el circuito hardware necesario.
- Crear el software para controlar el circuito hardware y capturar los datos.
- Implementar el software para registrar los datos en un dispositivo de almacenamiento.
- Desarrollar el software para obtener los datos en tiempo real.
- Crear una aplicación de escritorio que nos permita copiar esos datos registrados y visualizarlos. Además en esa aplicación tendremos la posibilidad de ver el movimiento del sistema hardware en tiempo real.

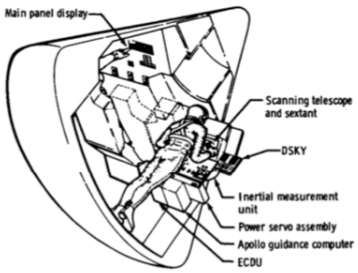
1.3 ¿Qué es una IMU?

En primer lugar definiremos lo que es una Imu o Inertial Measurement Unit o Unidad de Medición Inercial en español. Entonces una Imu es un conjunto de sensores que son capaces de medir parámetros físicos, como, la aceleración, velocidades, orientación, etc Una unidad de medición inercial suele estar formada por un acelerómetro, un giroscopio y un magnetómetro, aunque este puede variar dependiendo del modelo escogido como se mencionará más adelante.

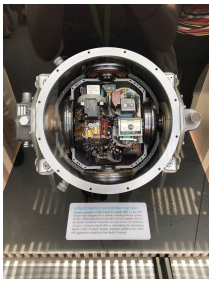
1.4 Un poco de historia

Antes de seguir profundizando en detalles técnicos sobre las Imu, hablaremos sobre su historia, la primera imu en el mundo real, imus en acontecimientos históricos importantes, etc. La primera imu en realidad estaba basada en un giroscopio. Esa ”imu” se utilizó en el primer avión con piloto automático combinado con un indicador de altitud, que podía imitar al acelerómetro y una brújula, que simula el magnetómetro [1].Este piloto añadió a su avión el estabilizador giroscópico que había inventado su padre, haciéndole algunas modificaciones, para que el avión pudiese mantener el rumbo y poder estabilizarse en caso de perder el equilibrio en alguna de sus alas [2].

Otro acontecimiento histórico tan importante como el primer vuelo con piloto automático fue la llegada a la luna con la misión Apollo 11. El módulo espacial de la Apollo 11, el cual se quedó orbitando alrededor de la luna, incluía una Imu para poder orientarse en el espacio. A continuación podemos unas figuras con la Imu del Apollo y su localización:



(a) Localización Imu[3]



(b) Imu Apollo 11[4]

Figura 1.1: Imu Apollo 11

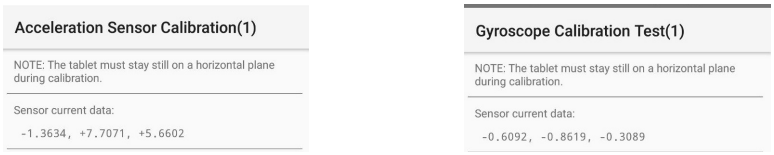
Este sistema para orientarse alrededor de la luna, se denominaba navegación inercial [5], el cual consistía en utilizar giroscopios y acelerómetros entre otros sistemas para orientarse. Hay que decir que este sistema no era perfecto porque se desviaba con el tiempo y cada cierto tiempo había que volver a alinear la nave[6].

1.5 Aplicaciones en el mundo real

Mencionados algunos detalles interesantes sobre la historia de las Imu, ahora nos centraremos en el panorama actual. Una de las principales diferencias y que más se nota es el tamaño, en la figura 1.1b observábamos que el imu era más grande que un balón de fútbol, en cambio las unidades de medición inerciales actuales las podemos esconder dentro de nuestra mano, como vemos a continuación:

Figura 1.2: Imu moderno

Incluso pueden llegar a ser tan pequeños que pueden ir integrados en otros dispositivos más grandes pero que caben en nuestro bolsillo o que podemos llevar en nuestras muñecas como pueden ser los smartphones, smartwatches, etc. En el siguiente grupo de figuras podemos observar datos de acelerómetros, giroscopios ejecutándose en un smartphone:



(a) Acelerómetro (b) Giroscopio

Figura 1.3: Sensores Smartphone

En estos dispositivos suelen utilizarse para detectar que los movimientos realiza, ya sea para jugar a un videojuego, donde el giroscopio y el acelerómetro actúan como controles como si estuvieras jugando con mando físico. A parte de los videojuegos se puede utilizar también para cuando sacas fotos, digamos que se utiliza de la siguiente manera: cuando el usuario se dispone a sacar una foto, el móvil puede detectar que le tiembla demasiado el pulso porque se está moviendo mucho, el movimiento seria registrado por el Imu, una vez detectado el software de la cámara mandaría una señal al estabilizador de imagen, para que este lo contrarreste y así evitar que la foto salta movida o borrosa, en la siguiente imagen podemos ver el efecto de una foto sin estabilizador y con él[7].

Figura 1.4: Comparativa sin/con estabilizador

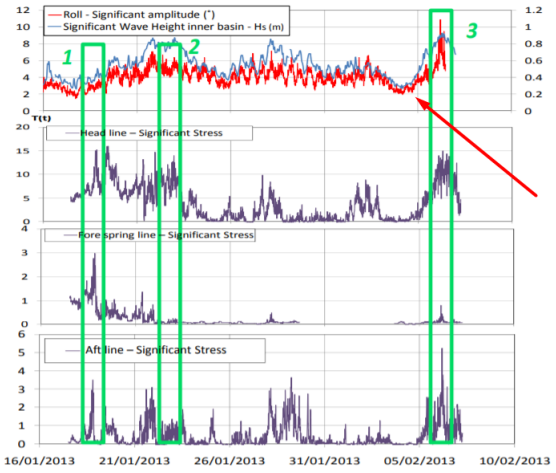
El siguiente elemento que también los incluye son las motos, en ellas actúa como un dispositivo de seguridad pasiva, puesto que no interviene directamente en la protección del usuario. La forma de actuar sería la siguiente:

Imaginemos que el motorista se acerca a una curva e inclina la moto para trazar mejor la curva, ese momento es crucial dado que cualquier problema mecánico puede hacer que el usuario pierda el equilibrio. En ese instante los frenos de la moto no pueden bloquearse, entonces el ABS tiene que estar funcionando y alerta para evitar el bloqueo y ahí es donde actúa el imu. Este sensor obtendría los datos y el software de la moto procesaría esos datos

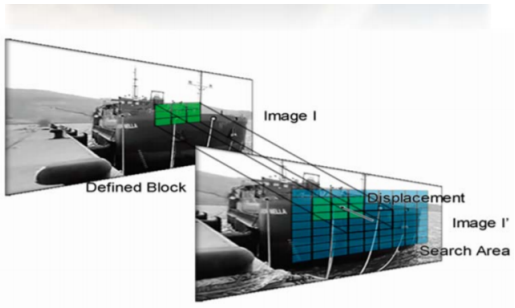
para saber que la moto esta inclinada y le mandaría una señal al ABS para que evite que los frenos se bloqueen[8].

Al principio mencionamos como se usó una Imu para crear el primer piloto automático 1.4, en la actualidad también se siguen utilizando en los aviones modernos a mayores de otros sensores, aparte de los aviones tripulados, también llevan este sensor los drones, los cuales están basados en el para funcionar.

El último ejemplo[9] que mencionaremos, también está un poco relacionado con los transportes, aunque esta vez el medio por el cual se desplazan es el agua. Este estudio fue realizado por la Universidad de la Coruña y consistía en analizar la tensión que producía un barco en sus amarres del puerto. Para poder realizar este experimento se utilizó una imu, un sistema GPS y un software de detección de imágenes para capturar los movimientos del barco. A continuación podemos ver algunas imágenes del estudio:



(a) Datos imu



(b) Detección de Imágenes

Figura 1.5: Medición de movimientos y tensiones en los amarres del barco Urania Mella en el Puerto Exterior de Punta Langosteira (A Coruña).

Capítulo 2

Estado del arte

En este capítulo profundizaremos en el funcionamiento de Imu, los elementos que la componen, diferentes tipos de Imu,etc. Incluiremos detalles técnicos sobre la plataforma arduino en la cual se implementó el software para la captura de los datos, el lenguaje C++ y el lenguaje Java el cual utilizamos para crear el software de escritorio.

2.1 Unidad de Medición Inercial

Como definimos en la introducción una Imu es un conjunto de sensores que son capaces de medir parámetros físicos, como, la aceleración, velocidades, orientación, etc. Una unidad de medición inercial suele estar formada por un acelerómetro, un giroscopio y un magnetómetro, aunque este último no tiene porque aparecer dependiendo del tipo de Imu que escojamos.

Las primeras Imu solían ser muy grandes y aparatosas como veíamos en la imagen 1.1b, después al ir avanzando la tecnología también avanzó el proceso de miniaturización, gracias a ese proceso las imus se fueron haciendo más pequeñas y manejables como mostrábamos en la imagen 1.2

Antes de entrar en el funcionamiento de la Imu explicaremos los distintos tipos que hay. Las diferentes Imu se diferencian unas de otras según los grados de libertad que tengan, y que son los grados de libertad.

La definición formal[10], expresada desde el punto de vista mecánico o físico sería la siguiente:

Definición Formal

”El grado de libertad (DOF) de un sistema mecánico es el número de parámetros independientes que definen su configuración”. Esos parámetros pueden determinar la posición del cuerpo, si está quieto o en movimiento, entre otros estados.

Una definición menos formal sería la siguiente:

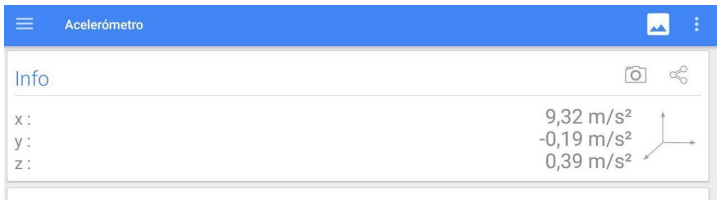
Definición de grados de libertad desde el punto vista de una Imu

Los grados de libertad de una Imu son todos los distintos movimientos que puede capturar.

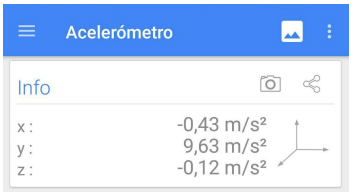
Con la segunda definición se quiere expresar que si una Imu consta de tres grados de libertad solo podría obtener datos de un sensor concretamente de un acelerómetro o un giroscopio, con lo cual el Imu de tres grados de libertad no se le considera una Imu, porque simplemente constaría de un sensor.

El primer tipo que explicaremos es el Imu de 6 grados de libertad (6 DOF). Este sensor esta formado por un giroscopio y un acelerómetro. Se dice que es de 6 grados de libertad porque puede capturar 6 movimientos, 3 para el giroscopio y 3 para el acelerómetro.

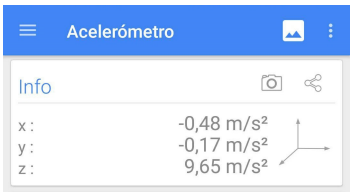
El acelerómetro captura los movimientos según la magnitud física de la aceleración. Decíamos en el párrafo anterior que el acelerómetro tenía tres movimientos uno por cada eje:



(a) Acelerómetro Eje X



(b) Acelerómetro Eje Y



(c) Acelerómetro Eje Z

Figura 2.1: Datos Acelerómetro

Observamos que dependiendo de la posición del sensor, este nos da resultados de la aceleración distintos. Por ejemplo, para el eje Z que es la imagen 2.1c, tenemos nuestro acelerómetro totalmente paralelo con una superficie plana y nos ofrece una aceleración de 9.65 m/s^2 que es prácticamente la gravedad de nuestro planeta, en cambio si levantamos el sensor hasta que es completamente paralelo al eje Y nos dará el valor de la gravedad para esté eje, para el eje X ocurriría lo mismo que para los otros dos casos. Estos movimientos que acabamos de realizar están registrados y cada uno tiene un nombre. El movimiento que se hace alrededor del eje X sería el movimiento de roll, respecto al eje Y pitch y cuando gira sobre el eje Z yaw.

Un uso del acelerómetro seria para averiguar si un cuerpo quiere ir hacia delante o no, es decir, si en nuestro smartphone tenemos un juego de carreras y el control del acelerador está basado en el acelerómetro, si inclinamos el smartphone hacia delante el sensor capturará los datos y el móvil los procesará y sabrá que queremos acelerar en cambio si lo inclinamos hacia atrás procederá a disminuir la velocidad del coche porque está detectando que queremos frenar.

El otro sensor que forma parte del Imu de 6 grados de libertad es el giroscopio, este sensor nos proporciona datos sobre la rotación, es decir, que si giramos hacia la derecha el sensor nos proporcionará un valor distinto, y si giramos hacia la izquierda nos proporcionará otro. Igual que el acelerómetro este también nos ofrece los datos en los tres ejes (X,Y,Z). En la siguiente figura vemos los datos que nos ofrece un giroscopio:

Figura 2.2: Datos Giroscopio

Siguiendo con el ejemplo del videojuego de coches, si inclinamos el teléfono hacia a la derecha, nuestro coche girará hacia ese lado, porque el giroscopio midió los datos que detectaban el giro hacia la derecha y el teléfono los procesó. Los datos de este sensor también se utilizan en dispositivos que utilizan realidad virtual para saber cuando el usuario está girando hacia un lado o hacia otro.

Con esto ya acabamos con el primer tipo, el siguiente tipo es una mejora de la Imu 6DOF, este tipo añade otro sensor y suma tres grados de libertad obteniendo una Imu 9DOF o 9 grados de libertad. El sensor que añadimos ahora es un magnetómetro y este sirve para obtener la orientación, podemos decir que el magnetómetro es como una brújula metida en la Imu. Este detecta los cambios electromagnéticos producidos por imanes, por el propio campo magnético de la tierra y otros dispositivos que generen un campo magnético. Una definición más formal, proporcionada por la RAE, ”es el aparato o sensor que nos proporciona información sobre un campo magnético”. Podemos apreciar en las imágenes que vemos a continuación, dos momentos distintos del estado de un magnetómetro.

En la imagen 2.3a se aprecian unos valores ligeramente bajos, dado que no tenemos ningún campo magnético de alta intensidad cerca, en cambio en la otra figura 2.3b, visualizamos unos valores mucho más altos que los anteriores, esto es debido a que acercamos un imán al sensor, por lo que aumentó la intensidad del campo magnético y como consecuencia los valores que registra el sensor aumentaron.

(a) Magnetómetro sin un Imán cerca

(b) Magnetómetro con un Imán cerca

Figura 2.3: Datos Magnetómetro

El uso principal de esté sensor sería el campo de la orientación para poder saber dónde está el polo norte magnético, aunque como comprobamos en las imágenes anteriores, se pueden corromper fácilmente los datos obtenidos si estamos cerca de un imán o cualquier dispositivo que genere un campo magnético o algún elemento metálico. En el caso de la orientación este sensor suele apoyarse en el GPS para obtener unos datos más fiables. Como mencionamos antes, si el magnetómetro tiene un cerca un metal puede variar los resultados, aunque gracias a eso, hemos descubierto otra utilidad de este, y es que podemos usar este sensor como detector de metales aunque no fuera diseñado para eso.

Dado que la tecnología sigue avanzando también existen Imus más modernos de 10[11] y 11[12] grados de libertad. Estos sensores incluyen un sensor barométrico para el de 10 DOF y el barómetro, sensor de temperatura y humedad para el de 11.

2.1.1 Elección del Imu

Dado que el mercado de Imus nos ofrece diversos tipos y dentro de cada tipo hay diferentes modelos de diferentes marcas, había que elegir uno. El primer filtro que aplicamos para decidirnos por un tipo u otro fue decidir que necesitábamos. En nuestro caso necesitamos obtener datos como mínimo del acelerómetro, giroscopio y magnetómetro. Entonces dimos por descartado las Imu de 6 grados de libertad. El siguiente paso fue preguntarnos si necesitábamos datos barométricos o de temperatura y humedad, para nuestro desarrollo no eran necesarios, por lo tanto nos decantamos por la Imu de 9 grados de libertad.

Dentro de las Imu de 9 grados de libertad, había diferentes modelos en el mercado, entonces para escoger una nos fijamos en los siguientes parámetros:

- Voltaje de Funcionamiento.
- Escala del acelerómetro, giroscopio y magnetómetro.
- Si incluía interfaz SPI o I2C.
- Intensidad del modo bajo consumo, modo eco, etc.
- Nº de bits proporcionados en la salida.
- Si disponía de interrupciones.
- Librerías disponibles y compatibles con arduino.
- Ejemplos de funcionamiento.
- Esquemas de conexión.
- Disponibilidad en el mercado.

En la siguiente tablas se mostrarán algunos de los Imus seleccionados para la comparativa incluyendo los parámetros mencionados arriba.

Modelo Sensor	Escalas		
	Acelerómetro (g)	Magnetómetro(gauss)	Giroscopio(dps)
Sparkfun LSM9DS1	$\pm 2, \pm 4, \pm 8, \pm 16$	$\pm 2, \pm 4, \pm 8, \pm 16$	$\pm 245, \pm 500, \pm 2000$
SparkFun MPU-9250	$\pm 2, \pm 4, \pm 8, \pm 16$	Rango máximo 48	$\pm 250, \pm 500, \pm 1000, \pm 2000$
Adafruit Precisión NXP 9-DOF	$\pm 2, \pm 4, \pm 8$	Rango máximo 12	$\pm 250, \pm 500, \pm 1000, \pm 2000, \pm 4000$
SparkFun 9DoF Sensor Stick	$\pm 2, \pm 4, \pm 8, \pm 16$	$\pm 2, \pm 4, \pm 8, \pm 16$	$\pm 245, \pm 500, \pm 2000$

Tabla 2.1: Tabla de escala de los tres dispositivos.

Modelo Sensor	Voltaje	SPI/SPC	Modo Bajo Consumo
Sparkfun LSM9DS1	1.9V a 3.6V	✓	3.1 mA
SparkFun MPU-9250	2.4V a 3.6V	✓	$315\mu A$
Adafruit Precisión NXP 9-DOF	1.95V a 3.6V	✓	$2.8\mu A + 35\mu Ao2.8\mu A + 240\mu A$
SparkFun 9DoF Sensor Stick	1.9V a 3.6V	✓	3.1 mA

Tabla 2.2: Tabla voltajes, modo bajo consumo y puertos SPI/SPC.

Modelo Sensor	Bits Salida	Interrupciones	librerías	Ejemplos	Esquemas
Sparkfun LSM9DS1	16	✓	✓	✓	✓
SparkFun MPU-9250	16	✓	✓	✓	✓
Adafruit Precisión NXP 9-DOF	16 y 14	✓	✓	✓	✓
SparkFun 9DoF Sensor Stick	16	✓	✓	✓	✓

Tabla 2.3: Tabla de bits de salida, Interrupciones, Librerías, Ejemplos y esquemas.

Tras buscar en internet varias imus, se escogieron 4. De esos 4 teníamos que quedarnos con uno solo. Para ellos fuimos analizando las características expuestas en las tablas anteriores para poder escogerlo.

El primer parámetro en el cual nos fijamos, fue en el rango de cada sensor, los cuales están recogidos en la tabla 2.1. Observamos que todos los sensores tenían más o menos el mismo rango, a veces un sensor era mucho mejor en un parámetro que los otros dos, pero en otro parámetro era superado por otro sensor, por ejemplo, el Imu LSM9DS1 tiene un rango para el acelerómetro entre 2 y 16 g pero en cambio para el giroscopio es mejor el sensor NXP 9-DOF porque tiene un rango entre 250 y 4000 dps, mientras que el LSM9DS1 solo tiene hasta 2000dps. Visto que con está comparativa no era suficiente, se compararon más parámetros. Observando todas las tablas veíamos que más o menos seguían teniendo características parecidas, todos incluían interrupciones, interfaces SPI e SPC, todos funcionaban a 3.3V, el nº de bits de salida eran prácticamente los mismos. El siguiente filtro era comprobar si todos eran compatibles con arduino, puesto que si uno no lo era, ya lo podíamos descartar sin comprobar nada más. También comprobamos que existieran ejemplos de funcionamiento y esquemas de conexión para aprender a utilizarlo. Con este último filtro seguíamos sin tener claro cual escoger, entonces el último parámetro en el cuál íbamos a escoger al Imu final, sería si tiene o no modo bajo consumo y analizar cuál era el que más ahorro energético nos ofrecía.

El modo bajo consumo de los tres Imus compartían una cosa, pues el magnetómetro no tenía este modo, por lo tanto no íbamos a fijarnos tanto en el valor de este sensor. Los otros dos sensores si incluían modo bajo consumo. Es importante decir que el consumo de los sensores varía con la frecuencia de trabajo, pues a mayor frecuencia mayor consumo. El primer sensor que descartamos fue el que tenía mayor consumo, este fue el LSM9DS1 pues tenía un consumo de 3.1mA, al descartar el LSM9DS1 también descartamos el 9DOF Sensor Stick porque tienen las mismas características, ya que el Sensor Stick usa un LSM9DS1 pero el chip en su conjunto es más pequeño[13]. Ahora la cosa quedaba entre los otros dos. El SparFun MPU-950 tenía un consumo combinado de $315\mu A$ mientras que el NXP 9DOF dependía del número de sensores activos. Por ejemplo, si este último tenía activados el giroscopio y el acelerómetro tenía un consumo de $37.8\mu A$ y cuando estaban activados los tres sensores, su consumo era de unos $248\mu A$. Por lo tanto tras analizar estos valores nos decantamos por esté último, puesto que nos ofrecía unas medidas más o menos parecidas que los otros dos con un menor consumo.

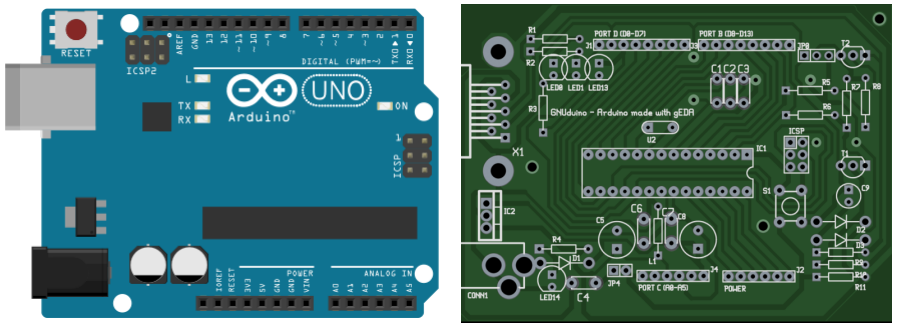
2.2 Plataforma Arduino

Primero explicaremos que es una placa de Arduino, y que definiremos de la siguiente manera:

Definición Formal

Una placa de arduino [14], podría ser una placa base en miniatura, ensamblada en una placa PCB, que incluye unos componentes básicos para las operaciones de E/S, un microcontrolador , pines para poder conectarles sensores o actuadores entre otros componentes.

A continuación, mostramos dos imágenes de una placa de arduino Uno, una de ellas es la imagen del circuito PCB.



(a) Arduino Uno

(b) PCB de Arduino Uno

Figura 2.4: Arduino Uno

El proyecto arduino[14] nació en el año 2005 de la mano del Instituto de diseño interactivo de Ivrea en Italia, este proyecto nació como material de apoyo para utilizar en las aulas para el aprendizaje de sus alumnos. Ese mismo año el instituto tuvo que cerrar y el equipo encargado de arduino por el temor a que se perdiera el proyecto tras el cierre decidió exponerlo al público.

Actualmente la plataforma arduino es de código y hardware abierto, operando bajos las licencias GPL, LGPL permitiendo que cualquier entidad pueda comercializar un diseño de arduino. Actualmente hay dos formas de conseguir una placa arduino, una de ellas es comprarla construida o adquirir un kit para montarla tú mismo.

2.2.1 Hardware genérico para Arduino

En esta sección vamos a mencionar el hardware básico de la placa de Arduino. Con lo de básico nos referimos a los componentes comunes a todas las placas de arduino dado que hay diferentes tipos y cada una tendrá sus peculiaridades.

El primer elemento más importe sería, obviando los circuitos conductores de electricidad y los conectores de alimentación ya que sin ellos no podría funcionar, sería el microprocesador, ya que sin este no podría ejecutar ningún programa, de la misma manera que los ordenadores sin el tampoco podrían funcionar.

A parte del microprocesador, otro elemento importante es la memoria, pues sin ella no se podrían almacenar los programas. Nos podemos encontrar tres tipos de memoria en una placa arduino[15]:

- **SRAM** → Memoria de acceso aleatorio y estático. Es donde el procesador guarda y manipula las variables del programa. Este tipo de memoria es volátil, es decir, cuando se le deja de aplicar un voltaje se borra.

- **EEPROM** → Es una memoria no volátil, la cual nos permite guardar datos y que no se borrarán aunque el sistema se quede sin energía. En esta memoria sería útil guardar el estado actual del sistema para en caso de corte energía y cuando se produzca el reinicio vuelva a recuperar dicho estado. En comparación con la SRAM es más lenta.

- **Memoria Flash** → Está memoria es no volátil como la anterior, es una similar a las memorias Usb, tarjetas sd o en mayor escala los discos SSD. En ella se guarda el programa a ejecutar en la placa de arduino entre otro software.

Estos componentes mencionados son comunes a todas las placas de arduino, aunque no todas los incluyen. Por ejemplo, el modelo MkrZero que utilizamos para nuestro proyecto no tiene EEPROM. A parte de algunas particularidades, lo que diferencia a una placa ya otra son los tamaños y las características de los componentes.

Otro elemento, aunque ya secundario, dado que no es necesariamente fundamental para su funcionamiento son los pines. Estos pines pueden ser de dos tipos analógicos o digitales. Los pines digitales funcionan a dos niveles de tensión, que son alto o bajo o 0 y 1. Aunque estos valores en digital no dependen del modelo de placa, si varía el voltaje. Dependiendo de la placa el valor alto, opera a distintos voltajes, 5V o 3.3V. Y para el valor bajo también es dependiente del voltaje, para una placa de 5V podríamos considerar un valor bajo cuando la placa detecte en un pin un valor menor que 3V y en el caso de las placas de 3,3V, el valor bajo se corresponde con un voltaje menor de 2V.

Respecto a los pines analógicos reciben señales de dispositivos analógicos, pero el microprocesador no sabe procesarlas, entonces los diseñadores le añadieron un conversor analógico/digital para poder procesarlas.

Las placas también incluyen diversos pines para la transmisión y recepción de datos, estos pines son los siguientes:

- **SPI**[16] → Interfaz periférica en serie, es un protocolo para el envió de datos en serie.
- **I2C**[17] También es un protocolo para el envió de datos de manera síncrona. Este tiene dos pines por los cuales se reciben o envían datos, uno de ellos es el pin SCL, para el reloj y el otro el SDA, para los datos. Existe un tercer pin GND, que es la toma de tierra.
- **TX/RX**[18] Son los pines encargados de la transmisión serie compartido con la transmisión serie por usb. El pin RX es de recepción y TX de transmisión, ambos son pines digitales.
- **VCC** Es el pin encargado de proporcionar tensión a los elementos conectados a la placa de arduino.
- **GND** Toma de tierra.

2.2.2 Dispositivos externos de ampliación

Como mencionamos en la definición a la placa le podemos conectar sensores, actuadores o otros elementos para ampliar la funcionalidad de esta. A continuación mostramos algunos de estos dispositivos:

2.2.2.1 Sensores

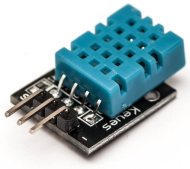
Estos dispositivos detectan cambios a su alrededor, pueden ser cambios físicos, químicos, relacionados con la luz, la presencia, etc.

(a) Sensor de Luz

(b) Sensor de Ultrasonidos

Figura 2.5: Sensores Luz y ultrasonidos

(a) Sensor de Movimiento



(b) Sensor de Temperatura

Figura 2.6: Sensores Movimiento y Temperatura

2.2.2.2 Actuadores

Este grupo, como indica su nombre. son los que actúan según la información que le llegan de los sensores. Por ejemplo, cuando se produce un incendio el detector de humo detecta que hay humo en el ambiente y este mandaría una señal para que los aspersores de agua se activen para apagarlo. Igual que en los sensores mostraremos alguno de ellos:

(a) Bomba de agua

(b) Motor eléctrico

Figura 2.7: Bomba de Agua y motor eléctrico

(a) Zumbador

(b) Led

Figura 2.8: Un zumbador y un Led

2.2.2.3 Shields

Por último tenemos los shields, estos dispositivos sirven para ampliar las capacidades de la placa de arduino, para poder ofrecer funcionalidades más complejas, por ejemplo:

(a) Módulo 4G

(b) Módulo wifi

Figura 2.9: Módulos 4 y wifi

(a) LCD

(b) Lector de tarjetas SD

Figura 2.10: Panel LCD y lector de tarjetas SD

2.2.2.4 Ejemplo Ilustrativo

En esta subsección os proporcionaremos un ejemplo, con todos los elementos anteriores. Imaginemos que queremos montar un sistema de alarma para nuestra casa, este sistema incluirá los siguientes tipos de alarma: alarma antincendios, detector de presencia, y también incluirá un detector de temperatura para averiguar la temperatura del ambiente a distancia. Con los cuatro sensores del principio los conectamos a la placa de arduino y cada uno de los sensores activará alguno de los actuadores, mencionados anteriormente, por ejemplo, en nuestra habitación empieza hacer mucho calor y sensor de temperatura lo detecta podemos hacer la placa de arduino envié una señal al aparato de aire a acondicionado a través de internet gracias al módulo wifi, además al tener conectado un módulo 4G incluso podríamos saber en tiempo real que temperatura hace en nuestra casa, o si alguno de los sensores de movimiento, ultrasonidos registraron algún movimiento. A mayores de ver en tiempo real la temperatura desde el móvil, gracias al panel LCD, la podemos visualizar si estamos en casa y llevar un registro de todos esos datos guardándolos en una tarjeta SD.

2.2.3 Tipos de placas arduino

A continuación mostraremos y resumiremos algunas características de las principales placas de arduino.

1. Arduino Mega[19] → Es el modelo más grande la marca, cuenta con las siguientes características:
    * Tiene un procesador ATmega2560 de 8 bits.
    * Voltaje operativo de 5V.
    * 54 pines digitales y 16 analógicos.
    * Memoria Flash de 256 KB, SRAM de 8kB y EEPROM de 4Kb.
    * Velocidad de reloj 16MHZ.
2. Arduino One[20] → Es el modelo más extendido de la marca y con el que recomiendan empezar, si es tu primera vez con Arduino, características:
    * Tiene un procesador ATmega328P de 8 bits.
    * Voltaje operativo de 5V.
    * 14 pines digitales y 6 analógicos.
    * Memoria Flash de 32KB, SRAM de 2kB y EEPROM de 1Kb.
    * Velocidad de reloj 16MHZ.
3. Arduino Ethernet[21] → Este modelo es prácticamente el mismo que el anterior, con dos diferencias, a mayores incluye conexión Ethernet y lector de tarjetas
    * Tiene un procesador ATmega328 de 8 bits.
    * Voltaje operativo de 5V.
    * 14 pines digitales y 6 analógicos.
    * Memoria Flash de 32KB, SRAM de 2kB y EEPROM de 1Kb.
    * Velocidad de reloj 16MHZ.
    * W5100 TCP/IP Embedded Ethernet Controller.
    * Lector de tarjetas MicroSd.

Estos tres modelos mencionados, estaban construidos con procesadores AVR del fabricante americano Atmel. Los procesadores AVR[22] están basados en la arquitectura RISC, la arquitectura RISC[23] está basada en conjunto de instrucciones más pequeñas y simples para que tarden menos tiempo en ejecutarse. Existen otros modelos de arduino basados en otros procesadores, ARM o intel. Estos modelos serían los siguientes:

- Arduino Due[24] → Este es el modelo basado en ARM y cuenta con las siguientes características:
  - Tiene un procesador AT91SAM3X8E basado en el ARM CORTEX-M3 de 32 bits.
  - Voltaje operativo de 3.3V.
  - 54 pines digitales y 14 analógicos.
  - Memoria Flash de 512KB, SRAM de 96kB .
  - Velocidad de reloj 84MHZ.
  - W5100 TCP/IP Embedded Ethernet Controller.
  - Lector de tarjetas MicroSd.
- Arduino Intel® Galileo Gen2[25] → Este es el modelo basado en Intel y cuenta con las siguientes características:
  - Tiene un procesador SoC Quark X1000 32 bits.
  - Voltaje operativo de 3.3V/5V.
  - 14 pines digitales y 6 analógicos.
  - Memoria Flash de 512KB, SRAM de 96kB, Ram 256MB DDR3, EPPROM 8Kb y almacenamiento Flash de 8MB.
  - Velocidad de reloj 84MHZ.
  - Conexión Ethernet.
  - Lector de tarjetas MicroSd.
  - Sistema Operativo Linux, distribución Yocto 1.4 Poky Linux.

En la placa Intel® Galileo Gen2 se puede ejecutar linux y los propios sketches de arduino con un mismo procesador, en cambio existen otras placas como la Arduino Yun[26], que incluye dos procesadores, uno para ejecutar los sketches de arduino y otro para ejecutar linux, las características de esta placa, están divididas en dos partes, una para el procesador de sketches:

- Tiene un procesador ATmega32U4 de 8 bits.
- Voltaje operativo de 5V.
- 20 pines digitales y 12 analógicos.
- Memoria Flash de 32KB, SRAM de 2.5kB y EEPROM de 1Kb.
- Velocidad de reloj 16MHZ.

y otra para el procesador que se encarga de ejecutar linux:

* Procesador Atheros AR9331 basado en MIPS.
* Voltaje operativo de 3.3V.
* Ethernet y conexión Wifi.
* RAM 64 MB DDR2 y memoria flash de 16MB.
* Velocidad de reloj 400MHZ.
* Lector de tarjetas micro-sd

La última placa que vamos a nombrar es una que pertenece a la familia MKR, concretamente, el modelo arduino MKRZero, en este modelo está basado nuestro proyecto de creación de un sistema de unidad de medición inercial con arduino. Está placa incluye un procesador SAMD21 Cortex-M0+ 32bit low power ARM MCU, funciona a un voltaje de 3.3 voltios. Incluye 22 pines digitales,de esos 22, 12 son pines digitales PWM, que sirven para situaciones en la que necesitemos variaciones de voltaje en un dispositivo final, por ejemplo, el control de intensidad de brillo de una bombilla. Incluye interfaces UART, SPI y IC2 para la transmisión de datos serie. Al contrario que otras placas no incluye memoria EEPROM, pero si dos memorias flash, una de ellas es exclusiva del bootloader. Una de las cualidades de esta placa es que fue diseñada para tener un bajo consumo de energía pero un alto rendimiento. En la siguiente figura podemos ver la distribución de todos sus pines.

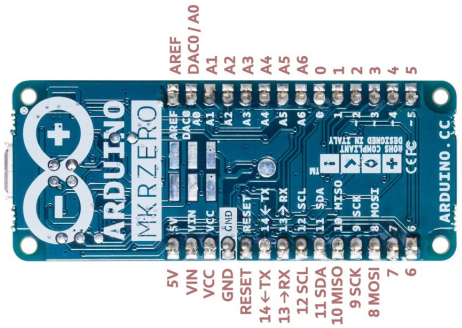


Figura 2.11: Arduino MkrZero

Para terminar con la sección de arduino, podemos decir que en la actualidad esta considerada un dispositivo para utilizar en el ámbito de las IoT (Internet of Things) puesto que es un dispositivo que puede hacer posible que elementos de uso cotidiano como aparatos de aire acondicionado, calentadores, calefacciones puedan conectarse a Internet a través de la

placa o también poder convertir ese termostato antiguo que tenemos en casa, que solo puede activarse manualmente, en un termostato inteligente y que a determinada temperatura pueda encenderse solo.

2.3 Lenguaje para programar en Arduino

Una vez visto los detalles hardware de arduino, tenemos realizar los programas para que pueda servirnos de alguna manera, ya que sin ningún programa ejecutándose en ella, podríamos decir que solo serviría como pisapapeles.

El lenguaje de programación de arduino es un subconjunto del lenguaje C++, es decir, tiene instrucciones de C++ y otras especificas para interactuar con arduino.

2.3.1 C++

El lenguaje C++[27] fue diseñado en 1979 por el científico de la computación Bjarne Stroustrup. Este lenguaje fue creado para ampliar la funcionalidad del lenguaje C, de ahí su nombre C++. En C no es posible aplicar el paradigma de orientación a objetos, que es el motivo por el cual nació el lenguaje C++. Aunque en el año de su creación no se llamó C++ hasta que fue utilizado fuera de un laboratorio donde otro científico le puso ese nombre.

2.3.1.1 Características del lenguaje C++

Algunas características[28] de este lenguaje serían las siguientes:

- Sintaxis igual que en c
- Existe un estándar ISO, ANSI-C++.
- Es compatible con el paradigma Orientado a objetos, incluyendo los siguientes mecanismos:
    - Abstracción
    - Encapsulado
    - Herencia
    - Polimorfismo
- Sobrecarga de operadores
- Soporta expresiones lambda o funciones anónimas
- Control de errores.

- Tipado de datos fuerte. Es decir, hay que declarar las variables antes de usarlas e indicar de tipo son. En otros lenguajes, como Python, esto no es necesario hacerlo.
- Es un lenguaje compilado.
- Permite usar punteros igual que C
- Es compatible con C, es decir un compilador de C++ puede compilar código C.

El paradigma orientado a objetos[29] es un paradigma de la programación que usa objetos y sus interacciones entre ellos para construir aplicaciones. Antes de definir algunas elementos de orientación a objetos, tenemos que definir que es un clase y que elementos la conforman. Una clase es donde definimos el comportamiento y el estado que tendrán los objetos creados a partir de esa clase. Una clase puede ser pública que puede ser accesible desde otros sitios, también puede ser virtual, esto quiere decir que no podemos crear un objeto directamente de esa clase. Una clase además está formada por atributos, que son las características, por ejemplo, si tenemos una clase cuadrado tendría un atributo lado y los métodos que son las operaciones que se pueden hacer sobre la clase que continuando con el ejemplo del cuadrado podría ser la operación de calcular el perímetro.

Una vez explicado lo que es una clase, explicaremos los mecanismos mencionados anteriormente:

- La abstracción en orientación a objetos se refiere a que podemos crear un objeto e utilizarlo sin saber cómo funciona por dentro. Un ejemplo de abstracción podría ser el siguiente, cuando conducimos un coche podemos acelerar pisando un pedal pero no tenemos porque saber como funciona todo lo que está asociado a la acción de pisar un pedal.
- El encapsulamiento podríamos definirlo como el nivel de acceso o de ocultación a un objeto. Hay diferentes niveles de encapsulamiento, aunque los dos más usados serían , público y privado. Un elemento público podría ser accedido desde cualquier sitio y incluso modificado y un elemento privado solo podría ser accedido desde dentro.
- La herencia es un mecanismo por el cual una clase puede obtener atributos y métodos de otra y poder usarlos el incluso sobre escribirlos. En el caso c++ puede usar herencia múltiple, es decir, poder heredar de distintas clases a la vez.
- Una Clase es polimórfica cuando se puede comportar de manera distinta. Por ejemplo, si tenemos una clase figura geométrica y tiene un método para calcular el área. Esa se puede calcular de manera distinta si la figura es un cuadrado o un triángulo.

2.3.1.2 Estructura de un programa en C++

Explicadas algunas características de C++, ahora profundizaremos en la estructura de un programa en C++. Para poder ejecutar un programa en C++ siempre hay declarar una función principal que se ejecuta siempre que se inicia. Esa función se llama main. Antes de mostrar como es esa función, hay que explicar que es una función.

Una función en el ámbito de la programación es un conjunto de instrucciones que se puede llamar varias veces por su nombre sin tener que programarla entera otra vez. Por ejemplo, si tenemos una operación que se llama sumar dos enteros, que llamamos varias veces durante nuestro código, no tenemos que escribir de cada vez $entero3 = entero1 + entero2$, sino que le podemos asignar a esa operación un nombre y llamarla cada vez con ese nombre. A las funciones se le pueden indicar parámetros se ponen de lado de su nombre entre paréntesis. Un cambio respecto a c en las funciones es la sobrecarga, es decir, si tenemos una función a() que no recibe nada, podemos tener una función a(a1,a2) que reciba dos parámetros, aunque se llamen de la misma manera el compilador podrá diferenciarlas debido a que una tiene variables y otra no. Aplicando está solución podríamos hacer un solo programa haciendo que todas las funciones se llamen igual excepto el main.

Los parámetros que se le pasan a una función pueden ser de dos tipos, por valor o por referencia. Cuando pasamos un parámetro por valor a una función es como si estuviéramos haciendo una copia de este mismo, es decir, si dentro de la función modificamos ese valor, fuera de ella no va a variar. En cambio, si pasamos ese valor por referencia, en c++ se acompaña del carácter &, y modificamos su valor dentro de la función fuera también va a cambiar. Esto se debe a que cuando pasamos un parámetro por referencia, ambos están apuntando a la misma dirección de memoria, por eso siempre que cambie uno cambiará el otro.

Una última cosa respecto a las funciones, una función siempre tiene que devolver un resultado, puesto que si no devuelve nada sería un procedimiento, en c++ se declaran de la misma manera, pero en otros lenguajes puede variar, por ejemplo, Pascal.

Retomando el tema con el cual iniciábamos esta sección, en c++ siempre que iniciamos un programa se ejecuta la función main si existe, esta función tiene el siguiente formato:

```
1 int main(int argc, char *argv[])
2 {
3   return 0; % valor de retorno de la función
4 }
```

Listing 2.1: Main Example

Como vemos el formato es muy simple, le indicamos un nombre, en este caso es main, unos paréntesis pegados a este nombre, dentro de ellos vemos dos variables, estás básicamente sirven para indicarle al main que va a recibir unos argumentos, donde argc indica el número de argumentos que se reciben y argv son los propios argumentos, aunque no es obligatorio

recibir esos argumentos si el main no los va a utilizar. Después tenemos la palabra reservada int, esta palabra no tiene que ser siempre int, porque aquí se indica el valor devuelto por la función con la palabra return. Una vez tenemos lista la firma de la función abrimos una { que es donde indicamos que empieza el cuerpo de la función y cuando acabemos la cerramos con otra }.

2.3.1.3 Tipos de datos en C++

Dado que C++ es un lenguaje de tipado fuerte hay que saber cuáles son los tipos principales de datos que existen y como declararlos.

Para declarar un dato en C++ primero hay que especificar el tipo, seguido del nombre, asignarle un valor inicial y finalizando la línea con un ;. Por ejemplo si queremos declarar una variable de tipo entero lo haríamos de la siguiente manera:

```
1     int entero = 0;
```

Listing 2.2: Declaración de una variable

En la siguiente figura podemos ver algunos de los principales tipo de datos y su tamaño en la placa arduino mkrzero, porque dependiendo de la placa el tamaño de algunos datos puede variar, por ejemplo, en la arduino Uno[30] un entero ocupa 2 bytes:

```
--- Miniterm on COM7  9600,8,N,1 ---
--- Quit: Ctrl+C | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H ---
Tamaño Int -> 4 bytes
Tamaño long -> 4 bytes
Tamaño float -> 4 bytes
Tamaño double -> 8 bytes
Tamaño char -> 1 bytes
Tamaño bool -> 1 bytes
```

Figura 2.12: Tamaño de los diferentes tipos de datos

Estos datos que aparecen en la imagen de arriba son datos primitivos, es decir, que no se pueden descomponer en datos más pequeños. Un dato primitivo también se podría considerar el valor nulo(void) que significa la ausencia de datos.

Los siguientes tipos de datos, son los complejos, que básicamente son estructuras de datos que están compuestos a partir de los datos primitivos. Algunos de estos datos pueden ser:

- Arrays
- Punteros
- Clases

- Enumeraciones
- Conjuntos

Con los tipos complejos, se pueden crear otras estructuras más complejas aún, con los arrays, por ejemplo, se pueden crear matrices que viene siendo un array con dos dimensiones. Con los punteros también podemos crear más estructuras, pues haciendo que un puntero apunte a otro se pueden crear listas, colas .. todas ellas dinámicas, las estáticas se crean con arrays.

2.3.1.4 Librerías

A veces cuando hacemos un programa tenemos que usar código que realizó otra persona y que no está guardado en el mismo fichero que nuestro programa principal, o incluso nosotros mismos queremos desarrollar una funcionalidad que vamos a utilizar en múltiples programas. Para poder llamar a esa librería o a ese otro programa nuestro se usa la cláusula incluida al principio del fichero fuente, como mostramos en el siguiente ejemplo:

```
1
2 #include <iostream>
3
4 using namespace std;
5
6 int main() {
7     cout << "usando la librería iostream" << endl;
8     return 0;
9 }
```

Listing 2.3: Usando Include

También observamos que incluye una línea ”using namespace std”, poniendo esta línea podemos omitir poner la palabra std:: cada vez que llamamos a cout. En el caso que queramos guardar nuestro programa en otro fichero tendrás que generar dos ficheros, uno en el cual solo guardamos la firma de los métodos, este fichero tendrá una extensión .h, y en el otro fichero crearemos la implementación de la firmas incluidas den el .h, que posteriormente podremos incluir en nuestro programa principal usando include ”nombreSubprograma.h.”

2.3.2 Elementos propios de arduino en C++

En esta sección contaremos como funciona la programación en C++ al trabajar con arduino. La primera diferencia que nos encontramos es que en arduino el main no existe como función principal. La primera función que se ejecuta de un sketch de arduino es el setup. Está

función solo se ejecuta al principio del programa una vez y nada más, por lo que es el entorno ideal para inicializar elementos, por ejemplo, activación de sensores, inicializar variables, etc.

La siguiente función más importante es la función loop(), que hace lo mismo que un while(1), el loop se está ejecutando todo el tiempo hasta que la placa de arduino se quede sin energía, la única manera que no se esté ejecutando todo el tiempo es usar funciones para que pause un rato, como delay. Entonces la estructura principal del sketch de arduino sería la siguiente:

```
1    int inicial;
2    void setup(){
3        inicial = 0;
4    }
5    void loop(){
6        inicial = inicial  + 1;
7    }
```

Listing 2.4: Estructura sketch arduino

El programa inicializara la variable inicial a 0 y una vez esté dentro del loop, en cada iteración la incrementará en 1.

Otras funciones interesantes y propias de arduino sirven para interactuar con el hardware de la placa como:

- El conjunto de funciones Serial, nos permiten comunicarnos por el puerto serie, algunas funciones son:
    - Serial.read() → Para leer datos del puerto serie.
    - Serial.write() → Para escribir datos del puerto serie.
    - Serial.available() → Nos indica si el puerto serie esta disponible.
- Otras nos permiten interactuar con los pines de entrada y salida, tanto analógicos como digitales´. Estas serían digitalRead()/digitalWrite() o analogRead()/analogWrite para mandar señales a los sensores conectados. Otra función útil para los pines digitales es pinMode() que nos permite establecer el modo en que actuará el pin, concretamente si será un pin de entrada o salida.

Además de funciones propias, tenemos definidas algunas constantes que podemos utilizar, como:

- HIGH|LOW → Para indicar si queremos un pin este alto o bajo

- INPUT|OUTPUT|INPUT_PULLUP→ Podemos indicarle a un pin que sea solo de entrada, salida o conectar el pin a una resistencia pullup interna. Usar este constante para un pin significa que invertimos la lógica del pin, cuando esta Alto estaría bajo y viceversa.
- LED_BUILTIN → Nos indica el número del led interno de la placa.

Además de todos los elementos que vienen de serie con arduino podemos incluir otras muchas librerías para interactuar con otros muchos subsistemas, por ejemplo, tenemos las librerías:

- SD → Para operaciones con la tarjeta SD
- LiquidCrystal → Para interactuar con el LCD
- GSM → Para conexiones GSM/GPRS
- Wire → Para interactuar con la interfaz I2C

Y otras muchas, ocuparían varias páginas mencionarlas todas, además, muchos fabricantes de sensores crean sus propias librerías totalmente compatibles con arduino para que sus sensores puedan funcionar a la perfección.

2.3.3 Entornos de desarrollo para arduino

Para finalizar con la parte de Arduino comentaremos algunos entornos de desarrollo para arduino.

El primer Ide que mencionaremos será el entorno oficial de arduino.

Figura 2.13: Arduino IDE

Esta versión de escritorio nos incluye todas las placas de arduino disponibles, tanto propias como de terceros. Incluye diversos ejemplos con distintos sensores y la posibilidad de buscar librerías.

Otra IDE de arduino oficial, es su versión web Arduino Create, que nos permite tener actualizado la versión del IDE y tener nuestro sketch guardado en la nube.

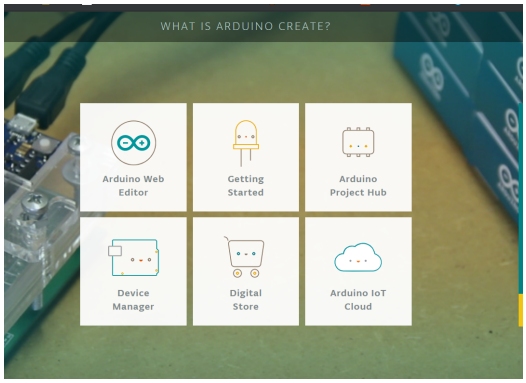


Figura 2.14: Arduino Create

Este IDE online nos proporciona el editor web, una guía de como empezar, un repositorio de proyectos entre otras cosas. Lo único que hay que hacer para utilizar es acceder a la página web registrarse e instalar un programa agente en el pc.

A parte de los entornos de desarrollo oficiales, existen añaden la compatibilidad con Arduino a través de plugins. Uno de ellos es Visual Studio Code de microsoft, para poder compilar y enviar los sketches a la placa de Arduino hace falta instalar la extensión de Platform.io

Figura 2.15: Arduino en visual studio

Esta extensión nos permite cargar proyectos de Arduino, elegir la placa que estamos utilizando y buscar las librerías necesarias para que nuestro proyecto funcione, todo ello sin salir del Visual Studio Code. Una ventaja respecto a los otros dos Ides es que nos permite la integración con repositorios remotos, si trabajamos en el proyecto de manera colaborativa.

El último ejemplo de entorno de desarrollo que vamos a mostrar es un entorno basado en elementos gráficos. ¿Y que es un entorno basado en elementos gráficos? Es un entorno basado en programación visual, como la que utilizan algunos lenguajes como Scratch. El modo de funcionamiento de programación visual es muy sencillo, es unir unas piezas con otras como si de un puzzle se tratará. Este tipo de programación está orientada a los niños que se adentran en

el mundo de programación para que les resulte más sencillo y ameno aprender a programar. A continuación enseñamos una captura del programa Ardublock, aunque existen otros muchos como Visualino, etc.

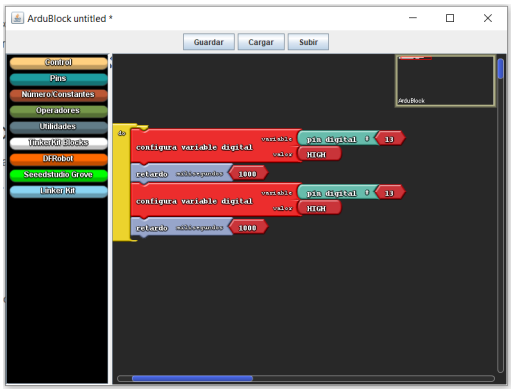


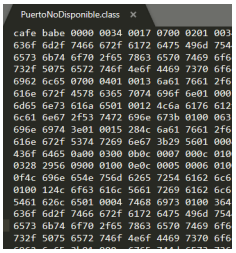
Figura 2.16: ArduBlock

Para concluir esta breve comparación distintos IDEs, nosotros hemos decidido utilizar el visual studio y el elemento diferenciador para elegirlo respecto a los otros IDEs es el que nos permite una integración con repositorios remotos, como git o svn, por lo que nos resulta más cómodo a la hora de subir nuestros cambios al repositorio.

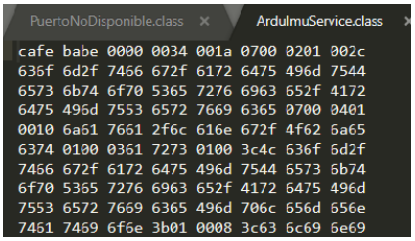
2.4 Java

En nuestro proyecto, aparte de C++, también utilizamos el lenguaje de programación Java para hacer una aplicación de escritorio, de la cual contaremos más detalles en el siguiente capítulo, en esta sección nos centraremos en contar algunas cosas sobre el lenguaje Java.

El lenguaje Java[31][32] fue creado alrededor de 1991 creado por la empresa Sun Microsystem por el equipo dirigido por James Gosling. Antes de llamarse Java tuvo otros nombres, que no prosperaron porque otras marcas ya lo utilizaban o creyeron que no era el adecuado, para finalmente llamarse Java. Sobre el origen de este nombre hay muchas historias distintas, como que son las iniciales de sus diseñadores o un acrónimo, ”Just Another Vague Acronym”, que básicamente quiere decir que no significa nada, aunque la teoría más fuerte, es la que su nombre es debido a un tipo de café, que se servía en una cafetería cercana y que su símbolo sea una taza de café. El origen de esta última teoría está fundamentada en los ficheros .class que se generan cuando compilas un programa en Java, pues estos ficheros están en hexadecimal y todos empiezan con los símbolos hexadecimales ”cafe”, como apreciamos en las siguientes imágenes:



(a) Símbolo CAFE



(b) Símbolo CAFE

Figura 2.17: Origen nombre JAVA

Vistos algunos detalles sobre el origen de Java[33], ahora veamos porque se creó. Esté lenguaje se creó con el objetivo de ser un lenguaje orientado a objetos y que fuera posible ejecutarlo en cualquier sistema sin que importara el sistema operativo, siempre y cuando estuviese instalada la máquina virtual de Java en el sistema.

Y a que nos referimos con la máquina virtual Java(JVM). Esta máquina virtual es la encargada de ejecutar los programas que se escriben en Java y es la encargada de conectar nuestras aplicaciones con el sistema operativo y el hardware de la propia máquina. Decíamos antes que un programa en Java se podría ejecutar sin importar el sistema operativo en el que esté, pero en cambio la JVM no es así, ya que cada sistema operativo cambia las librerías nativas del SO de las cuales Java puede utilizar y esa parte no es portable a cualquier sistema.

De la misma manera que C++, Java también posee las mismas características de ese lenguaje, es decir, tiene mecanismos de herencia, polimorfismo, encapsulación, abstracción, etc.

Hoy en día dado su popularidad es un lenguaje multiplataforma, multidispositivo(Pc, móviles, servidores, etc). También es muy versátil dado que puedes crear una aplicación de escritorio con una interfaz gráfica utilizando alguna de sus librerías gráficas, como SWT,SWING o JAVAFX. O crear un servidor web de diferentes tipos. Con lo de diferentes me refiero a los tipos principales de aplicaciones web, REST o SOAP.

Una aplicación web rest es un tipo de aplicación que expone sus operaciones a través de un API público. Esas operaciones expuestas se pueden invocar utilizando Http a usando comandos, los comandos http más conocidos son:

- POST → Normalmente se utiliza para la creación de recursos individuales, aunque también se puede usar para modificar un recurso existente.
- GET → Se utiliza para obtener un recurso del servidor web, también se puede obtener una colección de recursos.
- PUT → Se utiliza para actualizar un recurso.
- DELETE → Se utiliza para borrar un recurso

Estos recursos son enviados a través del protocolo HTTP en unos formatos de representación de recursos, los formatos más usados son XML o JSON. Una de las grandes ventajas de REST es que nos permite totalmente separar el servidor de las aplicaciones consumidoras de este. Esta ventaja es muy importante, porque el servidor puede cambiar su implementación interna y las aplicaciones que los usen no se den cuenta de ese cambio.

El otro tipo de servidor es SOAP, este tipo de servidor está basado en el modelo RPC, en este modelo se exponen una seria de operaciones para que se invoquen remotamente. Una de las ventajas de usar un servidor SOAP es que invocar una operación es igual de sencillo que llamar a una librería interna de Java. Esa facilidad de invocación de operaciones nos ofrece en cambio un inconveniente, porque cualquier cambio en el servidor nos obliga a realizar un cambio en el cliente, por lo que se dice que cliente y servidor están muy acoplados.

2.4.1 Estructura de un programa en Java

2.4.1.1 Tipos de datos en Java

Java igual que C++ es un lenguaje con tipado fuerte, dado que el compilador Java también necesita saber de qué tipo es una variable antes de compilarla. La manera de declarar variables es la misma que en C++ *tipoVariable + nombre = valor;*. En Java tenemos los siguientes tipos de datos:

- Primitivos → Estos son los tipos básicos que existen en todos los lenguajes int, char, double. float, etc.
- Estructurados → Estos tipos de datos contienen a los datos primitivos además estos tipos estructurados son clases de objetos. Algunos tipos más usados son los siguientes:
    - Clase String. Que viene siendo una cadena de caracteres.
    - Array. Pueden contener colecciones de tipos primitivos, también pueden tener colecciones de objetos. En Java hay clases como ArrayList que ya proporcionan las operaciones básicas de un array y que funcionan de manera eficiente.
    - Una clase envoltorio o wrapper es aquella que puede convertir un tipo primitivo en un objeto. Por ejemplo, si tenemos un int a y queremos convertirlo a un objeto solo tendríamos que hacer Integer a2 = new Integer(a). La clase Integer sería lo que consideraríamos un envoltorio.
    - En este tipo también están incluidas las clases propias que implemente el programador.

En la definición de wrapper mostramos una nueva palabra reservada, *new*, esta palabra se usa para crear un objeto, a esta cláusula le acompaña el nombre de la clase y dos paréntesis, que es a lo que llamamos el constructor del objeto.

Igual que C++, Java es un lenguaje orientado a objetos y se estructura en clases incluso su programa principal es un clase Java.

```
1 public class Principal {
2
3   public static void main(final String[] args) {
4         System.out.println("Clase principal")
5   }
6 }
```

Listing 2.5: Clase Principal Java

La clase principal también puede tener argumentos de entrada, aunque recibe directamente los argumentos y no el número de estos. Observando el fragmento de código de arriba vemos que tiene varias palabras reservadas antes de la palabra main. La primera de ellas es la *public*, que especifica el acceso del método. En Java existen 4 especificadores de acceso que son los siguientes:

- Public → Es el especificador más permisivo dado que permite al elemento desde cualquier sitio.
- Por defecto(Paquete) → Este se aplica cuando no se pone nada y permite el acceso al elemento dentro del mismo paquete que la clase principal.
- Protected → Este permite el acceso desde dentro del propio paquete, desde la clase y desde la subclase.
- Private → Este es el más restrictivo de todos porque solo permite acceder al elemento desde la propia clase.

Otro palabra reservada que vemos es static, esta palabra reservada como las anteriores se pueden aplicar a clases, métodos o atributos. La cláusula static hace posible acceder al elemento que la acompaña sin tener que instanciar la clase. Normalmente a la palabra static le suele acompañar *final* que indica que no se puede modificar el valor después de haberlo creado.

2.4.1.2 Estructura de una clase

Una vez vista la estructura del programa principal vamos a ver la estructura de una clase, como el programa principal también es una clase, viene a ser la misma estructura pero vamos a explicar algunos campos a mayores. Podríamos separar la clase en tres partes la primera donde se declaran los atributos, la segunda donde se declaran las operaciones relacionadas con la modificación y lectura de esos métodos y la tercera donde se hacen operaciones sobre

esos métodos, a continuación mostraremos un ejemplo de clase con un objeto de la vida real como un triángulo.

```
public class Triangulo {

    private int lado1;
    private int lado2;
    private int lado3;
    private int altura;
    private int base;

    public int getLado1(){
        return lado;}
    public void setLado1(int lado){
        this.lado = lado;}

    public int getBase1(){
        return base;}
    public void setBase1(int base){
        this.base = base;}
    public int area(){
        return (this.base*this.altura)/2;}
}
```

Listing 2.6: Ejemplo de clase

En este ejemplo tenemos dos atributos privados, a los cuales podemos acceder desde fuera de la propia clase con un get para leerlo y set para modificarlos, si quisiésemos que el atributo fuese de solo lectura no haría falta incluir el set. Cuando hacemos el set observamos que al atributo te acompaña la palabra this, esto lo hacemos cuando la variable que le pasamos al método se llama igual que el atributo para poder diferenciar una de otra.

Igual que en c++ en Java también existen mecanismos de herencia, aunque con alguna diferencia, en Java no se permite herencia múltiple mientras que en c++ sí. El mecanismo de herencia se utiliza cuando tenemos declarada una clase y queremos aprovechar alguna de sus características. A veces esa clase de la que heredamos no se puede instanciar directamente porque está declarada como una clase abstracta incluyendo la palabra clave abstract. Siguiendo con el ejemplo anterior del triángulo, ese triángulo representa un tipo de triángulo general, y sabemos que hay distintos tipos de triángulo dependiendo de sus lados. Pues podemos crear una clase TrianguloEquilatero que herede de esa clase añadiendo sus métodos específicos,esta subclase podría redefinir los métodos de la clase padre utilizando la palabra @override.

```
public class TrianguloEquilatero extends Triangulo {

    public boolean esEquilatero(){
        return (this.getLado1()==this.getLado2()) ?
    (this.getLado3()==this.getLado2()):false;
    }
}
```

Listing 2.7: Ejemplo de Herencia

Como vemos arriba solo declaramos el método esEquilatero, y si nos fijamos cuando llamamos a recuperar los lados, tenemos que recuperarlos a través de sus getters, porque sus atributos los declaramos privados.

En Java, a veces queremos declarar una clase con solo las firmas de los métodos pero sin implementación ninguna para que otra clase que lo utilice le dé una implementación específica. Por ejemplo, si queremos tener varios tipos de figuras que tengan todas las mismas firmas de métodos pero cada figura puede implementar la operación de una manera distinta, lo más cómodo sería usar una interfaz. Para crear esta clase hay que declararla usando la palabra *interface*. Entonces esta tendría la siguiente estructura:

```
public interface Figura{
    public int calcularArea();
}
```

Listing 2.8: Ejemplo de *interface*

Y una clase podría utilizarla usarla con la palabra *implements* y su nombre. En el momento de implementar esa *interface* tendremos que proporcionarle una implementación porque en la interfaz no lo tiene.

```
public class cuadrado implements Figura{

    private int lado;
    @Overrride
    public int calcularArea(){
        return lado*lado
    }
}
```

Listing 2.9: Ejemplo de implementación de un *interface*

A diferencia de la herencia una clase puede implementar múltiples interfaces. Normalmente cuando implementamos una interfaz, extendemos de una clase o usamos una librería externa tenemos que importarla porque no está en el mismo sitio que nuestra clase, ya sea que este

en otro paquete, en el caso de usar *implements o extends* o que usemos una librería, tenemos que usar la palabra clave *import* seguido del nombre del paquete y finalizando con el nombre del elemento a importar.

2.4.1.3 Diseño de interfaces en Java

Nuestra aplicación en Java va a tener una interfaz gráfica, para realizar dicha interfaz existen diferentes frameworks (SWT,AWT,SWING, etc) que nos ofrecen componentes gráficos para poder crear interfaces más complejas. Nosotros escogimos el framework JavaFx concretamente una versión llamada jfoenix, que nos aporta el Material design de google usando componentes Java. Para diseñar la interfaz con Javafx, existen dos maneras. La primera es trabajar directamente sobre un fichero .fxml, que está basado en xml, donde vamos escribiendo los componentes que queremos crear.

```
<VBox fx:id="firstSubMenuList" styleClass="subVBox">
  <children>
    <Button />
    <JFXSlider fx:id="intervalo" styleClass="subMenu" />
    <Button fx:id="aceptarIntervalo"" onAction="#EnviarIntervalo"
    text="Aceptar" />
    <Label fx:id="etiquetaValorIntervalo"  />
  </children>
</VBox>
```

Listing 2.10: Elementos en fxml.

Está manera de crear interfaces también se utilizan en otros lenguajes como android, XAML que se utiliza en la plataforma Xamarín de Microsoft donde se pueden crear aplicaciones multiplataforma para android, Ios y windows, XUL desarrollado por Mozilla.

Otro fichero que tenemos que tener en cuenta, es el fichero de estilos en cascada(css). En este fichero podemos hacer referencia a los elementos de la vista para poder personalizar a nuestro gusto, como aparece en el siguiente recuadro:

```
.vBox{
-fx-border-style: solid inside;
-fx-border-width: 1;
-fx-border-color: #000;
-fx-background-color:#000;
}
```

Listing 2.11: Fichero de estilos.

La otra manera es usar un programa que nos permite insertar de manera gráfica los componentes y poder visualizar sin ejecutar como va a avanzado la creación de interfaz de usuario.

El programa que usamos se llama sceneBuider y se instala en el equipo:

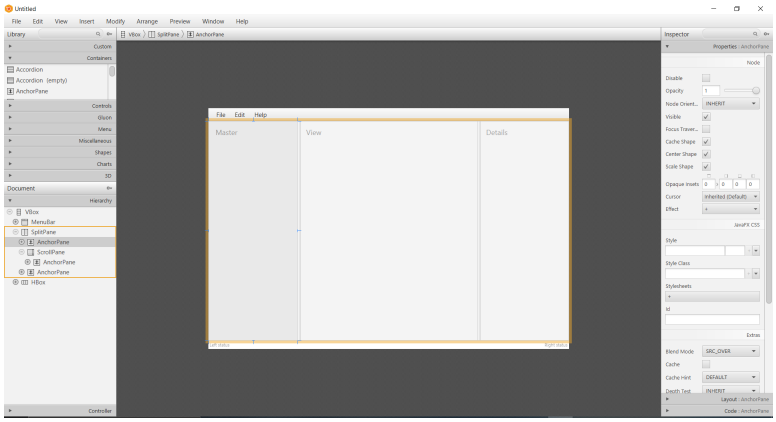
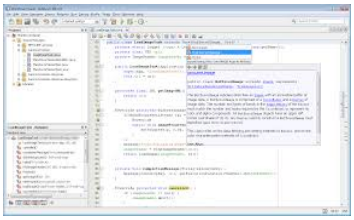


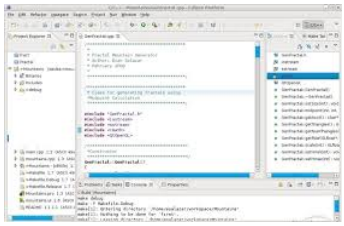
Figura 2.18: Scene Builder

2.4.2 Entornos de desarrollo para Java

Para desarrollar en Java existen múltiples entornos de desarrollo, algunos de ellos serían los siguientes:



(a) Netbeans



(b) Eclipse

Figura 2.19: IDEs desarrollados en Java

Otros como el visualStudio es totalmente compatible con el desarrollo de programas en Java, solo tenemos que instalarle un plugin y ya podemos empezar a desarrollar.

(a) Visual Studio

Para escoger el ide para desarrollar en Java nos fijamos en varias cosas, que tuviese inte-

gración con repositorios, integración con maven y con gestores de diseño de interfaces gráficas. Todos los mencionados cumplían esos requisitos entonces nos decantamos por el que nos resulta más cómodo para trabajar, que fue eclipse.

2.5 Otras tecnologías usadas

A mayores de las tecnologías indicadas en la sección anterior, como arduino y Java. Utilizaremos otras para apoyarnos en la realización de nuestro proyecto. En primer lugar tendremos que tener un sitio donde ir guardando nuestros avances de desarrollo del proyecto, para hacerlo utilizaremos un repositorio descentralizado como es git. Y qué es un repositorio descentralizado. En este tipo de repositorios existen dos versiones, uno local y otro remoto, el desarrollador trabaja contra el repositorio local y cuando acaba de hacer algunos cambios puede subirlos al repositorio remoto. Algunos de los comandos más utilizados son los siguiente:

- git add <archivos> → Permite añadir los nuevos ficheros al repositorio.
- git commit <opciones> <mensaje> → Permite confirmar los cambios en tu repositorio local
- git push → Permite subir los cambios confirmados al repositorio remoto.
- git pull → Permite traer cambios que existan en el repositorio remoto que no estén en el local.
- git merge → Permite unir una rama de desarrollo con otra

Normalmente a la hora de trabajar con git existen unas buenas prácticas que es bueno seguirlas, son las siguientes:

- Rama Master → Es la rama principal del proyecto, de ella cuelgan el resto de subramas. Nunca se debería desarrollar código nuevo sobre está rama.
- Development → El objetivo de esta rama es integrar todo los desarrollos nuevos realizados en otras ramas y comprobar que funcionan correctamente antes de llevarlos a master.
- Features\<nombreTarea> → En esta subrama se van realizando todas las tareas, una vez finalizadas se aplica un merge con develop
- Hotfix → Está la rama de corrección de errores que se detectan en la rama de Master cuando el desarrollo ya ha finalizado.

Otra software que hemos utilizado a sido maven, que es una herramienta de gestión de proyectos. Maven se basa en los ficheros POM, en este fichero se declaran todas las dependencias de librerías externas que necesita el proyecto, como se va a construir el ejecutable final, copia de recursos externos. Maven además tiene un ciclo de vida, que consta de varias fases:

- — Compile

  ◇ Es la fase encargada de la compilación de los ficheros fuente(.java), generando los ficheros .class.

- — Test

  ◇ Es la fase encargada de ejecutar los test de junit, si los hubiese.

- — Package

  ◇ Esta fase se encarga de empaquetar los ficheros .class en un fichero jar.

- — Install

  ◇ Copia el fichero .jar al directorio .m2 de nuestro ordenador para que pueda ser usado por otros programas. Si se ejecuta esta fase también se están ejecutando todas las fases anteriores evidentemente, pues para generar copiar el jar hace falta crearlo, compilar los class, etc.

- — Deploy

  ◇ Se encarga colocar el jar generado en un repositorio compartido por internet o en un red interna.

A parte de estas fases que están dentro del ciclo de maven, hay otras que no están incluidas pero que nos pueden ser de utilidad, son las siguientes:

- Clean

  — Elimina el directorio target.

- Assembly

  — Se encarga de crear un fichero comprimido con lo necesario para instalar nuestro programa.

- Site

  — Crea un fichero html con la info de nuestro proyecto.

- Site-deploy

  — Despliega nuestro servidor java, pagina web en lugar que le indicamos en la configuración.

Una ventaja de maven es que es integrable con la mayoría de los IDEs que existen en el mercado.

Capítulo 3

Metodología de desarrollo

La metodología de desarrollo escogida para nuestro proyecto será una basada en iteraciones o sprints fundamentada en la metodología ágil Scrum. Scrum normalmente implica un desarrollo incremental, que las tareas se puedan realizar de manera paralela e incluso solapar algunas fases del desarrollo. En scrum hay tres elementos importantes:

- Product BackLog, que es donde están indicados todos los requisitos que hay que cumplir y realizar en el proyecto.
- Sprint BackLog, aquí están indicadas todas las tareas que hay que hacer en un sprint.
- Sprint review, es una reunión que se hace después de cada sprint para revisar y mejorar lo realizado en el sprint.

A parte de estos tres elementos, el equipo scrum está formado por tres roles:

- El product owner, es quien se encarga de organizar el product backlog y es encargado principal del proyecto.
- Scrum manager, es el encargado de dirigir y asegurar que se cumpla el proceso scrum, además también está para apoyar al equipo para resolver cualquier problema durante el sprint.
- Por último, está el equipo de desarrollo que es el encargado de desarrollar las tareas indicadas en el sprint backlog y en el producto backlog.

Una vez comentadas algunas cosas de scrum vamos a seleccionar una herramienta para gestionar el proceso scrum, nosotros utilizaremos la herramienta trello, la cual nos permite gestionar, el producto backlog y el sprint backlog:

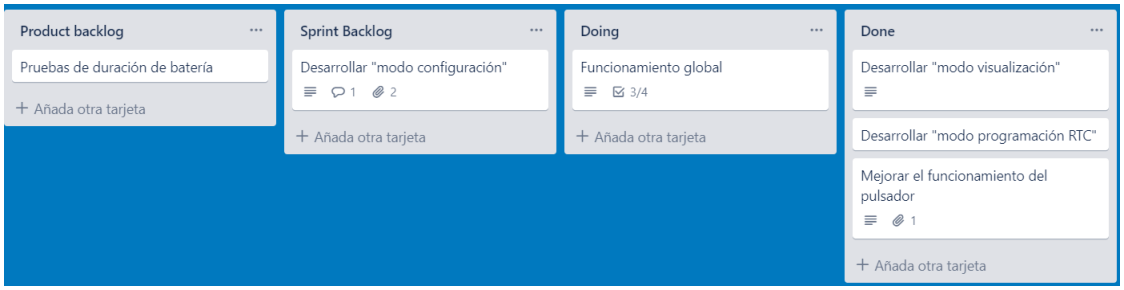


Figura 3.1: Elementos Scrum

Se ha escogido está metodología porque nuestro desarrollo tendrá varias fases y algunas de ellas se repetirán en todas las iteraciones, por ejemplo, en nuestro proyecto tenemos diseñar un circuito electrónico, entonces en una primera iteración se diseñará un circuito sencillo y funcional para la toma de contacto con el hardware, ver los componentes, como funcionan, etc. Y una vez concluido el desarrollo hardware empezaremos con el desarrollo software. Mencionar que una vez concluido el desarrollo hardware y si existe una iteración posterior ya no pasaremos por esa fase de diseño hardware porque ya no será necesario.

El ciclo de vida de nuestras iteraciones lo podemos ver en la siguiente figura:

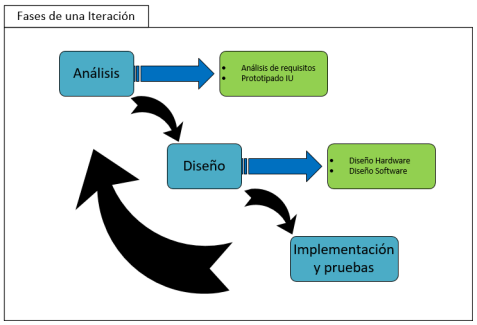


Figura 3.2: Fases de una iteración

Todas las fases principales de la iteración se harán de manera secuencial dado que solo una persona está involucrada en el proyecto, pero si fuese el caso de que hubiese más personas trabajan en el proyecto se podrían paralelizar. En la fase de diseño observamos que tenemos dos tipos de diseño uno hardware y otro software, pues ambas son perfectamente paralelizables puesto que un miembro del proyecto puede ir diseñando el circuito electrónico mientras que otro puede diseñar el software. Igualmente para la fase de implementación, se puede ir montando el hardware a la vez que otro miembro desarrolla el software. En lo relativo

al desarrollo software tenemos dos partes: una en c++ y otra en java. Esta parte también se puede hacer a la vez dado que un miembro puede implementar el código c++, mientras que otro desarrolla en java. Lo único que no podemos paralelizar en este proyecto son las pruebas ya que para verificar el funcionamiento del sistema se necesita que todas las partes estén unidas. Pero como decíamos al principio solo va a estar una persona trabajando en el proyecto entonces las tareas serán totalmente secuenciales.

Capítulo 4

Desarrollo Del Proyecto

En esté capitulo se explicará como fue el desarrollo del proyecto. Este proyecto de diseño de una Imu con arduino se podría dividir en varios subproyectos. Por un lado tenemos la parte hardware que es resultado de unir la Imu, el rtc y el botón a la placa de arduino, otra parte y directamente unida al desarrollo hardware, es la creación del software para la placa de arduino, el cual controlará el funcionamiento del botón, la imu y el RTC. Por último y no menos importante el desarrollo de la aplicación de escritorio en Java, la cual controlará algunos parámetros de configuración del sistema arduino, como la frecuencia de lectura del sensor o sincronización de la hora del RTC con el ordenador.

Antes de empezar con la fase de desarrollo del proyecto, haremos una estimación de cuanto tardaremos en realizar cada fase. Se estimarán las tres grandes fases de este proyecto (Análisis,Diseño e Implementación)

Analisis	**12,5 días**	**mié 13/03/19**	**vie 29/03/19**
Analisis Requisitos Funcionales	50 horas	mié 13/03/19	jue 21/03/19
Analisis Requisitos No Funcionales	50 horas	jue 21/03/19	vie 29/03/19
Diseño	**27,5 días**	**vie 29/03/19**	**mar 07/05/19**
Diseño Hardware	100 horas	vie 29/03/19	mar 16/04/19
Diseño Software	120 horas	mié 17/04/19	mar 07/05/19
Implementación y pruebas	**40,63 días**	**mié 08/05/19**	**mié 03/07/19**
Implementacion	200 horas	mié 08/05/19	mar 11/06/19
Pruebas	125 horas	mié 12/06/19	mié 03/07/19

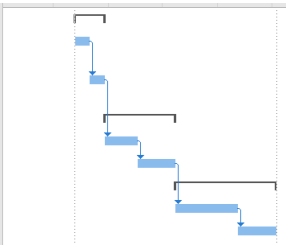


Figura 4.1: Diagrama de Gantt

Esta planificación se hizo contando que solo iba a existir una persona trabajando en el proyecto, si hubiese más personas involucradas, la planificación sería algo distinta pues algunas tareas se podrían paralelizar. Para la estimación de costes, diferenciamos dos tipos:

- Costes materiales (equipos electrónicos, hardware, software, etc).
- El coste de los trabajadores involucrados en el proyecto.

Entonces una estimación de los costes vendría resumida en la siguiente tabla:

Recurso	Coste Unitario	Horas	Coste Total
Ordenador Portátil	1000€	-	1000€
Pantalla Secundaria	121€	-	121€
Teclado externo	35€	-	35€
Ratón externo	27€	-	27€
Arduino MKRZero	20,90€	-	20,90€
RTC DS3231	7€	-	7€
IMU Adafruit Precision NXP 9DOF	20,31€	-	20,31€
Placa Protoboard	3,10€	-	3,10€
Cables de conexión	0,04€	-	0,04€
Tarjeta microsd	5,88€	-	5,88€
Programador Junior	10€	645 Horas	6450€
Estimación total	-	-	7656,23€

Tabla 4.1: Estimación de costes.

El coste total del proyecto serían alrededor de 7656€y con una duración total de 645 horas, sin contar los retrasos que se puedan producir cuando se vaya avanzando en el tiempo. En la tabla observamos que algunos recursos no tienen número de horas, puesto que al ser recursos materiales solo requieren una inversión inicial.

4.1 Primera Iteración: Trabajando con el Pulsador

4.1.1 Análisis

Está primera iteración tendrá como principal objetivo poner a funcionar el pulsador, además de crear un menú para cambiar entre los distintos modos de funcionamiento. En esta primera fase se definirán tres modos de funcionamiento. El primer modo será el de configuración de parámetros, donde podremos configurar la hora del sistema , la frecuencia de lectura, copiar los datos de la tarjeta sd al pc y ver el funcionamiento en tiempo real de la Imu..

4.1.1.1 Requisitos

Está fase tendrá tres requisitos fundamentales:

- **ArduImu-RF-1-A** → Conexión del pulsador a la placa de arduino. Aquí procederemos a crear las conexiones necesarias para unir el pulsador a la placa.
- **ArduImu-RF-1-B** → Crear la acción asociada a la pulsación. La acción de detectar un pulsación será implementada a través de interrupciones, es decir, cuando se produzca una interrupción se aplicará una acción.

- **ArduImu-RF-1-C** → Crear un menú para cambiar entre los distintos modos de funcionamiento. Cada vez que se pulse el pulsador se deberá avanzar a la siguiente opción del menú y cuando se llegue a la última se volverá a la primera opción.

4.1.2 Diseño

El apartado de diseño se dividirá en dos partes: diseño hardware y diseño software. En el diseño hardware crearemos el esquema de conexionado del circuito electrónico.

4.1.2.1 Diseño Hardware

Este primer diseño hardware es muy sencillo ya que solo se conectará el pulsador a la placa. A continuación se muestra el esquema de conexión:

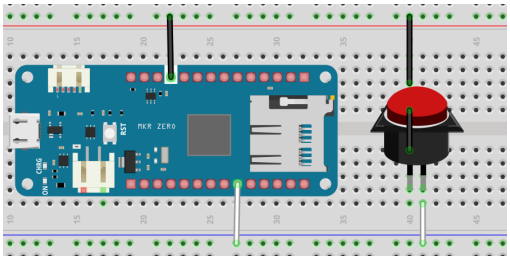


Figura 4.2: Conexión del botón con la placa

Como vemos es una conexión sencilla, pues solo conectamos uno de los pines del pulsador al pin de GND y el otro de los pines al pin digital 0.

4.1.2.2 Diseño Software

Con el esquema hardware finalizado, empezamos a diseñar el software de este apartado. El programa constará de las siguientes operaciones:

- Setup. En este procedimiento estableceremos el modo de funcionamiento del pin, uniremos la interrupción detectada en ese pin con la función encargada de aumentar el contador de menú.

- Loop. En este procedimiento se comprobará el valor del contador de menú y dependiendo de ese valor se entrará en una opción u otra.

4.1.3 Implementación y pruebas

Una vez finalizado con el diseño del software, procedimos a desarrollar el programa que iba a controlar el funcionamiento del botón, un primer desarrollo fue el siguiente:

```
void ()
{
    counter++;
}
void setup()
{
  // initialize the console
  Serial.begin(115200);
  // initialize LED digital pin as an output.
  pinMode(LED_BUILTIN, OUTPUT);
  pinMode(interruptPin, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(interruptPin),
    aumentarContador, CHANGE);
}
void loop()
{
      if (counter==1) {
        Serial.println("Modo Configuración")
    }
}
```

Listing 4.1: Primer desarrollo del funcionamiento del pulsador

Un primer desarrollo para probar como funcionaba el pulsador y el manejo de interrupciones fue el anterior. En la función setup del skecth de arduino, procedimos a configurar la velocidad del puerto serie, por el cual íbamos a enviar datos a la consola. En las líneas siguientes establecimos el modo de funcionamiento de los pin asociado al led de la placa y al pin 0 donde estaba conectado el pin del pulsador. El pin del led lo pusimos en modo Salida y el del pulsador en modo INPUT_PULLUP. Antes de unir la interrupción al pin que iba a lanzar la interrupción creamos la función, que lo que hacía era cambiar el estado del led y aumentar un contador cada vez que pulsábamos y enviarlo a la consola. Finalmente con attachInterrupt detectábamos la interrupción del pulsador y la uníamos con el procedimiento aumentarContador(), además a attachInterrupt le pasábamos el flag CHANGE que indicaba que cada vez que el pin cambiase de valor se activará la interrupción. Con el código creado lo compilamos y lo enviamos a la placa de arduino. En el procedimiento loop, mostramos un fragmento del menú, en este caso si contador es igual a 1 entraría en el modo configuración. Las otras opciones del menú serían si la variable counter es igual a 2(Modo grabar) o 3(Modo parar de grabar). De momento solo enviamos el comando por consola porque no tenemos nada que configurar.

Con el primer desarrollo finalizado procedemos a validar el funcionamiento del código. Esta implementación no tiene una batería de pruebas automatizadas, ya que requiere de una interacción externa para pulsar el botón. Para validar el funcionamiento mandamos por serial el valor del counter cada vez que lo pulsamos.

Al empezar a ejecutar el programa y visualizar la salida comprobamos que no estaba funcionando bien, porque cuando pulsamos la primera vez, en la salida nos debía mostrar el valor uno y eso no fue así, porque nos mostraba un valor más alto. Lo que pasaba era que el pulsador estaba produciendo ruido, conocido por efecto rebote, esto se debe a que cuando pulsamos en el dispositivo a parte de la señal de cambio de estado también genera señales de ruido, que si alcanzan cierto valor pueden ser interpretadas como un cambio de estado y que el sistema arduino lo procese como si fuese una pulsación.

Una vez nos dimos cuenta de este error tuvimos que reimplementar la función. Buscando documentación sobre el efecto rebote en Arduino encontramos una librería llamada Bounce2 la cual nos permitía corregir el efecto rebote y tras crear el código usando esa librería nos seguía sin funcionar del todo bien. Entonces decidimos crear nuestra propia solución sin usar ninguna librería externa. La solución fue la siguiente:

```
1 void aumentarContador()
2 {
3    if (millis() > contadorTiempo + umbral)
4    {
5       ISRCounter++;
6       contadorTiempo = millis();
7    }
8 }
```

Listing 4.2: Eliminando efecto rebote

Está función se ejecuta cada vez que la placa de arduino detecta una interrupción en un pin determinado, eso lo indicamos con la función attachInterrupt. La interrupción en este caso se produce al presionar el pulsador. Cuando el proceso entra en la función lo primero que hace es calcular el tiempo transcurrido desde el comienzo del programa. Entonces el código funciona de la siguiente manera, si el tiempo actual es mayor que el contadorTiempo, que es valor de la última vez que paso por aqui, más el umbral preestablecido por nosotros, en este caso de 250ms, que lo ajustamos con el método de prueba y error, entonces la pulsación fue correcta. En cambio si milis() es menor que contadorTiempo y umbral la pulsación no fue correcta. Imaginemos el siguiente caso, presionamos el pulsador y esta pulsación fue correcta, una vez que levantamos el dedo, se puede producir el siguiente caso, que mientras hacemos la acción de levantar el dedo exista ruido y la placa lo interprete como una interrupción pero en este caso el tiempo transcurrido va a ser muy pequeño y milis() va a ser mas pequeño que nuestro umbral más el tiempo de la última pulsación por lo que consideramos que no fue una pulsación real. Una vez compilado el código nuevo y ejecutado comprobamos que cada vez que presionábamos nos contaba una pulsación.

4.2 Segunda iteración: Montaje y Ajuste del RTC

4.2.1 Análisis

En esta segunda iteración ampliaremos las funcionalidades de la aplicación añadiendo un rtc para cuando registremos los datos del sensor, también registraremos la hora en la que se tomó la medida. Además empezaremos a crear la aplicación de escritorio en java, dado que en esta fase la necesitaremos para ajustar la hora del rtc. Antes de exponer la lista de requisitos explicaremos que es un RTC o Real Time Clock. Este dispositivo nos permite saber que la fecha y hora en cualquier instante, además no depende de un fuente de alimentación externa ya que cuenta con una pila interna. Un rtc suele usarse para registrar eventos y así poderlos guardar con una marca de tiempo. Nosotros utilizaremos el modelo DS3231[34], que cuenta con las siguientes características:

- Compensación de año bisiesto hasta el año 2100.
- Tiene incluido un oscilador de cristal.
- Tiene incluido un sensor digital de temperatura, con una precisión de 3ºC.
- Nos proporciona segundos, minutos, horas, día del mes, mes, día de la semana y el año.
- Una precisión de 2ppm para rangos de temperatura en 0º y 40ºC.
- Interfaz I2C para transmisión de datos.
- Voltaje operativo de 3.3V.

En este sprint vamos a tener que poder comunicarnos desde nuestra aplicación en Java con el programa que se está ejecutando en arduino a través del puerto serie. Para poder comunicarnos usaremos la librería jSerialComm. Con ella podremos detectar si existe algún puerto activo y obtener datos recibidos por el puerto serie y enviarlos. Para poder obtener y enviar esos datos nos apoyaremos con elementos del paquete java.io como OutputStream para enviar datos por serial y InputStream para recibirlos. Aunque usamos realmente elementos de java.io para recibir y enviar, en realidad esos streams de entrada y salida nos lo proporciona la librería jSerialComm a través de los métodos getOutputStream(escritura) y getInputStream(lectura); Una vez comentado estos dos aspectos necesarios para la realización de esté sprint, definiremos los requisitos.

4.2.1.1 Requisitos funcionales

- **ArduImu-RF-2-A** → Se ampliará el requisito **ArduImu-RF-1-A**. Añadiendo al circuito un rtc.

- **ArduImu-RF-2-B** → Implementación de la funcionalidad para ajustar del rtc con la del ordenador.

- **ArduImu-RF-2-C** → Implementación del envió hora del rtc por puerto serie.

- **ArduImu-RF-2-D** → Diseño e implementación de la función de sincronizar rtc desde la aplicación de escritorio.

- **ArduImu-RF-2-E** → Diseño e implementación de la funcionalidad que permite ver la hora del rtc desde la aplicación de escritorio.

- **ArduImu-RF-2-F** → La interfaz de usuario, a partir de aquí IU, constará de un menú lateral a la izquierda de la ventana. Será un menú de apertura en acordeón, donde cada elemento principal del menú tendrá diversas opciones dentro. En la siguiente sección podremos ver ese menú.

- **ArduImu-RF-2-G** → En esta iteración solo existirá un apartado en el menú, modo configuración, el cual tendrá dos botones, uno para sincronizar la fecha y la hora y otro para mostrarla.

- **ArduImu-RF-2-H** → Cuando se pulse el botón de visualizar la fecha y la hora, se mostrarán en una etiqueta en el centro del panel derecho arriba de todo.

4.2.1.2 Requisitos no funcionales

- **ArduImu-RNF-2-A** → No se podrá entrar en el modo Configuración desde la IU si la placa de arduino no está conectada al equipo.

- **ArduImu-RNF-2-B** → La IU no deberá congelarse cuando se estén recibiendo datos del puerto serie y estos deban pintarse en la IU.

4.2.2 Diseño

4.2.2.1 Diseño Hardware

En el diseño hardware, vamos a ampliar el circuito anterior conectando el RTC a la placa de arduino, este dispositivo va a guardar la hora actual. El circuito montado sería el siguiente:

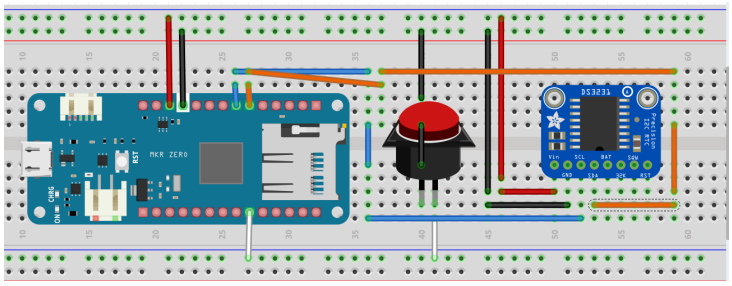


Figura 4.3: Conexión del RTC con la placa

Como podemos observar en la figura ya a aumentado un poco la complejidad del circuito y con ello el número de cables. El esquema conexión sería el siguiente:

- El pin *Vin* del RTC la conectamos al pin Vcc de la placa de arduino(cable rojo)
- EL pin del GND del RTC, igual que el pin GND del pulsador, al GND de Arduino arduino(**cable negro**)
- El pin SCL del RTC, para la señal de reloj del RTC, lo conectamos al pin SCL de la placa, el cable azul) y por último el pin SCA, para enviar los datos de la hora, al pin SCA de la placa de arduino, cable naranja)

4.2.2.2 Diseño Software

En la parte de software de esta iteración vamos a subdividirla en dos partes, una primera parte que sería crear la funcionalidad de ajustar el rtc en c++ recibiendo la hora actual por el puerto serie y enviar la hora del rtc, para saber si está bien ajustada por el puerto serie también. Y la segunda parte comenzar con el diseño y desarrollo de aplicación de escritorio en Java.

En primer lugar vamos a empezar a diseñar la parte de c++. Esta funcionalidad la vamos a implementar en un fichero aparte del fichero principal, para hacerlo crearemos dos ficheros, un timeUtils.h y timeUtils.cpp, en el primero irán las firmas de los métodos y en el segundo le proporcionaremos una implementación a esas firmas. En el fichero timeUtils.h declararemos 4 métodos:

- ajustarFechayHora
- obtenerFechayHora
- imprimirFechayHora
- sinBatería

Una vez que tenemos listo el diseño en arduino para poder ajustar el rtc por puerto serie necesitamos alguna manera de mandárselo por el pc. Para hacerlo empezaremos a diseñar nuestra aplicación de escritorio en java.

Para el diseño de la aplicación en java nos vamos a basar en el modelo-vista-controlador, aunque lo vamos a variar un poco , ya que nuestra capa modelo, la subdividiremos en dos partes, una capa para manejar las operaciones de entrada y salida con arduino, que nos referiremos a ella como la capa de lógica de negocio y otra capa si implementará la capa de persistencia de los datos obtenidos de la capa de lógica de negocio. Ambas capas serán independientes y no se llamarán directamente una a la otra, sino que se comunicarán a través de la capa controladora entre ellas y la vista.

En este diagrama también mostramos como se relacionan la capa lógica de negocio con la capa controladora y está con la vista.

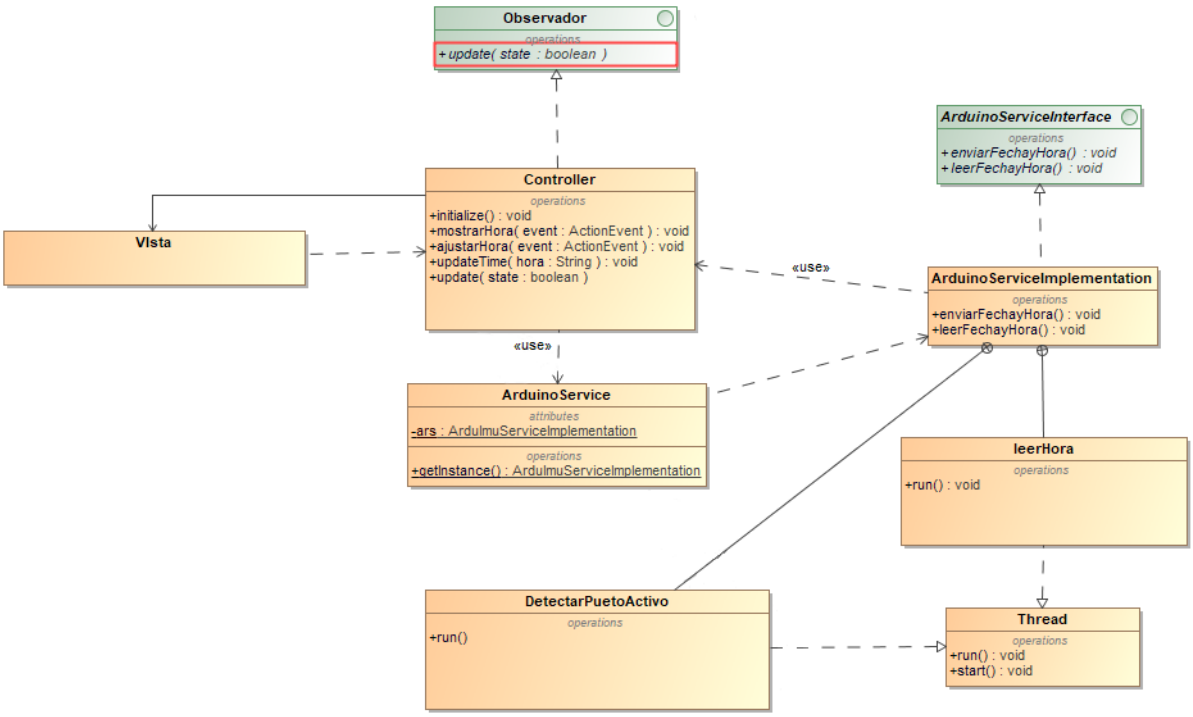


Figura 4.4: Diagrama UML Capa lógica de negocio y controladora

Vamos a comentar un caso de ejemplo para detallar el funcionamiento de este diseño. Imaginemos que tenemos la IU con el botón de ajustar hora, al pulsarlo la vista se conectará

con el controlador que ejecutará la función encargada del controlador de llamar a la capa lógica de negocio, a partir de ahora ArduinoService, la función de ejecutar hora y mandar una señal a arduino indicándole que vamos a proceder a ajustar la hora del rtc y posteriormente enviamos la hora del sistema donde se está ejecutando la aplicación. En el caso que pulsemos el botón de recibir la hora, está se recibirá en otro hilo al principal para que la IU no se congele, además en otro hilo también se estará ejecutando una función que detectará la conexión y desconexión de la placa y actuar deshabilitando o habilitando algunos botones.

4.2.2.3 Interfaz de usuario

Como en está iteración vamos a comenzar a crear la aplicación de escritorio y esta va a tener una interfaz gráfica, en esta sección mostraremos todas las pantallas de manera incremental, empezando con una ventana sencilla, que según avancemos por las distintas iteraciones, ira evolucionando hasta convertirse en la ventana final de la aplicación de escritorio. La siguiente vista refleja lo expuesto en los requisitos **ArduImu-2-F**, **ArduImu-2-G**, **ArduImu-2-H**

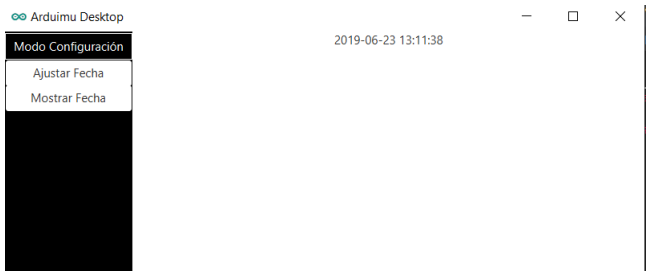


Figura 4.5

El funcionamiento de esta ventana seria el siguiente, al pulsar en el botón menú configuración aparecerán justo debajo los otros dos botones. Si se pulsa el botón de ajustar hora se enviará la hora del sistema por serial al RTC y si se pulsa el botón mostrar hora, se leerá la hora del rtc y se mostrará en la etiqueta central.

4.2.3 Implementación y pruebas

Aquí vamos a empezar la fase de implementación de está iteración, de la misma manera que en la anterior fase, empezaremos desarrollando en c++. El primer método de los definidos anteriormente, ajustarFechayHora, lo que hace es recibir en forma de String la fecha, por el puerto serie, con la siguiente línea[1]:

[1]Esto solo es un pequeño fragmento de ejemplo del código de real, ya que solo se quiere ilustrar la forma de capturar la hora por el puerto serie

```
String fechaHora = Serial.readString();
```

Listing 4.3: Recibiendo la hora

Una vez tenemos capturada la hora procedemos a ajustar el RTC con dicha hora[2]:

```
strcpy(fecha, hora.c_str());
sscanf(fecha, "%d/%d/%d %d:%d:%d", &yy, &month, &dd, &hh, &mm, &ss);
rtc.adjust(DateTime(yy, month, dd, hh, mm, ss));
```

Listing 4.4: Ajustando el RTC

Lo primero que hacemos es copiar el String hora a un array de char, porque sscanf no admite directamente el uso de Strings, una vez hecho eso obtenemos cada elemento de la fecha y hora aplicando un descriptor de formato que indicando el número de parámetros a separar y que son todos números. Por último a la función *adjust* del rtc le pasamos la fecha en forma de Datetime. El resto de funciones nos permiten recuperar la hora del rtc, con la función *rtc.now()*, enviar por puerto serie (*serial.println()*) esa hora y enviar un mensaje por puerto serie también si el rtc se quedó sin batería.

Finalizado el código en c++, vamos a empezar a desarrollar la aplicación en java, lo primero que haremos será crear nuestra IU, para hacerlo utilizaremos el framework de creación de interfaces de usuario, javafx. El papel de la vista en javaFx, es un fichero xml, en el cual se declaran los elementos que formarán parte de ella. En el siguiente fragmento de código, se muestra un boton en el fichero fxml.

```
<Button fx:id="abrir" onAction="#crearAbrirArchivo"
    styleClass="subMenu" text="Abrir">
```

Listing 4.5: Botón en fxml

Donde la palabra Button nos indica el tipo de componente, en este caso es un botón, pero si pusiésemos otra palabra sería otro tipo de componente, el siguiente elemento es el fx:id, que es el id que llamaremos desde java para realizar acciones con ese elemento. El atributo onAction nos indica el método a ejecutar cuando se haga click en el botón abrir. Por último nos queda los atributos styleClass, que busca en un fichero css si existe algún estilo que se llama *subMenu* y la cláusula *text* que indica el nombre a mostrar en el elemento.

Una vez finalizado la creación de la IU, procedemos a codificar las funciones del controlador,en la declaración del botón indicamos el nombre de este método, para que cada vez que se pulse llamaremos a una función del controlador, como la siguiente:

[2]Los ejemplos anteriores o siguientes omitirán la declaración de las variables para no mostrar ejemplos de código demasiado extensos

```
@FXML
private void AjustarHora(final ActionEvent event) {
  try {
    ars.enviarFechayHora();
  } catch (final PuertoNoDisponible e) {
    etiquetaInfo.setText(e.getMessage());
  }
}
```

Listing 4.6: Acción asociada al botón ajustar fecha y hora

Una vez entra en la función, llamaría a **ars** que es una instancia de ArduinoService, creada en la fase de inicialización del controlador. Entonces se llamará a la función enviarFechayHora. Este método se encargará de enviar la fecha y la hora con un señal indicando que es la fecha lo que se envía para que el software desarrollado para arduino sepa lo que es y lo procese correctamente.

```
public void enviarFechayHora() throws PuertoNoDisponible {
  outstream = serialPort.getOutputStream();
  try {
    outstream.write(("fecha: " +
    obtenerFechayHoraActual()).getBytes());
    outstream.close();
  } catch (final IOException e) {
    throw new PuertoNoDisponible(
    Constantes.PUERTONODISPONIBLE);
  } catch (final NullPointerException e) {
    throw new PuertoNoDisponible(
    Constantes.PUERTONODISPONIBLE);
  }
}
```

Listing 4.7: Método que envía la fecha y la hora a arduino

Si al ejecutarse esta función pueden producirse dos errores importantes, que no pueda conseguir el puerto, por lo tanto outstream tendría el valor nulo, o que se produzca error en el proceso de envió de los datos entonces saltaría un excepción propia **PuertoNoDisponible**, si se lanza incluiría el mensaje ”puerto no disponible”. El proceso de recibir la hora del rtc para insertarla en la IU, es parecido al proceso de enviarla. En primer lugar enviamos un mensaje al sistema arduino indicando que queremos que nos envie la hora actual, la leemos del puerto serie y avisamos al controlador enviándole los datos para que los actualiza en pantalla. Este último proceso de enviar leer los datos de la fecha y la hora se realiza en segundo plano para no bloquear la IU.

```
  @Override
public void leerFechaYhora() throws PuertoNoDisponible {
  try {
    final OutputStream outstream = serialPort.getOutputStream();
    outstream.write("Imprimir Hora".getBytes());
    outstream.close();
    Thread.sleep(1500);
    final byte[] readBuffer = new byte[20];
    final InputStream Input = serialPort.getInputStream();
    while (Input.available() > 0) {
      final int numBytes = Input.read(readBuffer);
  }
  str2 = new String(readBuffer);
  observadores.updateTime(str2);
}
```

Listing 4.8: Acción de obtener la fecha y la hora

Después de enviar el String con la orden de imprimirHora, esperamos unos segundos para que el programa en arduino procese la orden y nos envié la respuesta. Después de ese tiempo leemos los bytes enviados y se lo pasamos al controlador(”observadores”). Todo este método como comentamos arriba se realiza en segundo plano, para poder realizar cuando se pulsa el botón guarda en un comando la cadena ”leerFechaYhora” y el thread encargado de leer esos datos lo está comprobando constantemente y solo cuando ese comando tiene ese valor llama realmente al método anterior de leerFechaYhora.

```
class leerHora extends Thread {
  @Override
  public void run() {
      while (true) {
        logger.info("leerHora");
        if (comando.equals("leerFechaYhora")) {
          leerFechaYhora();
        }
        }
  }
}
```

Listing 4.9: Thread encargado de leer la fecha y la hora

Para finalizar con la implementación de esta iteración, nos queda crear la funcionalidad que detecte si arduino está conectado al pc o no, igual que el método anterior está comprobando si se pulsa o no el botón de la mostrar la fechar y hora, esté está comprobando si el puerto serie está disponible o no.

```
class detectarPuertoActivo extends Thread {
  @Override
  public void run() {
    while (true) {
      if (SerialPort.getCommPorts().length != 0) {
        abierto = true;
        observadores.update(!abierto);
      } else {
        abierto = false;
        observadores.update(!abierto);
}}}}
```

Listing 4.10: Thread encargado de detectar el puerto activo

En este caso, comprueba el número de puertos disponibles, si hay alguno le indica al controlador que los puertos están abiertos, como podemos ver si el puerto está abierto, a *true*, se está enviando *false*, porque lo hace el controlador es habilitar o deshabilitar elementos entonces si queremos que un elemento de la vista esté deshabilitado lo tenemos que poner a falso.

Igual que en la iteración uno tenemos que comprobar el funcionamiento del sistema, y de la misma manera no tenemos implementadas unas pruebas automatizadas porque se requiere interacción con un usuario para realizar las acciones necesarias para validar su funcionamiento. La forma de comprobar su correcto funcionamiento fue la siguiente:

1. Primero presionamos el pulsador para poder el sistema arduino en el modo configuración.

2. Una vez puesto en ese modo, arrancamos el programa de escritorio sin conectar la placa y observamos que el botón del modo configuración esta apagado.

3. Ahora conectamos el cable usb de arduino, se habilitan los botones y procedemos a ajustar la hora.

4. Para comprobar que la hora se ajustó bien, ejecutamos la acción de obtener la hora y observamos que nos la devolvía correctamente.

5. Desconectamos la placa de arduino y verificamos que los botones se deshabilitaron.

Después de realizar todas estas pruebas, realizamos una última comprobación. Esta comprobación fue si la hora se estaba guardando después de quitar la alimentación. Volvimos a conectar la placa, arrancar la aplicación y a recuperar la hora. Cuando nos recuperó la hora, estaba mal porque tenía unos valores erróneos. Primero verificamos la conexión del circuito, por si había algún pin mal conectado, todo era correcto, después verificamos si ajustaba bien la hora, todo correcto. Entonces llegamos a una conclusión, la posibilidad que la pila del RTC estuviese

agotada, se procedió a cambiar la pila y volvimos a realizar todos los pasos anteriores. Esta vez después de desconectar arduino y volver a conectar si nos devolvió la hora correcta y podemos dar por finalizada está iteración.

4.3 Tercera Iteración: Conectar Imu y registrar datos

4.3.1 Análisis

Seguimos mejorando nuestro desarrollo, volveremos a ampliar el circuito electrónico, añadiendo el ultimo componente que nos faltaba, la imu, además crearemos la funcionalidad de registrar los datos medidos y la manera de verlos en el pc. En la siguiente sección enumeraremos los requisitos para esta fase.

4.3.1.1 Requisitos funcionales

- **ArduImu-RF-3-A** → Se ampliarán los requisitos **ArduImu-RF-1-A** y **ArduImu-RF-2-A**. Añadiendo el imu al circuito actual.
- **ArduImu-RF-3-B** → Creación de las funciones para inicializar, preparar los sensores y capturar sus datos
- **ArduiImu-RF-3-C** → Crear la funcionalidad para leer y escribir datos en la tarjeta sd.
- **ArduImu-RF-3-D** → Crear la funcionalidad para poder copiar los datos de la tarjeta sd a un fichero csv en el ordenador personal.
- **ArduImu-RF-3-E** → Añadir al menú de la aplicación de escritorio un apartado *Modo histórico* que tenga los siguientes botones en su interior. Copiar fichero, ver fichero, mostrar datos, filtrar por fecha y hora y un botón para aplicar los filtros.
- **ArduImu-RF-3-F** → Un cuadro de texto donde se visualizarán los datos del fichero csv.
- **ArduImu-RF-3-G** → Se añadirá un botón ver histórico para habilitar el panel del modo histórico.

4.3.1.2 Requisitos no funcionales

- **ArduImu-RNF-3-A** Si la placa de arduino no está conectada al pc, el botón de copiar historio desde la sd no estará habilitada, mientras que los otros si ya que no necesita que la placa esté conectada al pc para visualizar el histórico.

- **ArduImu-RNF-3-B** El fichero csv no tendrá una localización definida, sino que podrá abrir el fichero a través de una ventana de dialogo que permita buscar y seleccionar el archivo. Eso si solo se podrán abrir ficheros con extensión .csv desde la aplicación.

4.3.2 Diseño

4.3.2.1 Diseño Hardware

El diseño hardware será ampliar el circuito existente, en este caso conectando el imu a la placa de arduino. A continuación mostramos el circuito final resultante:

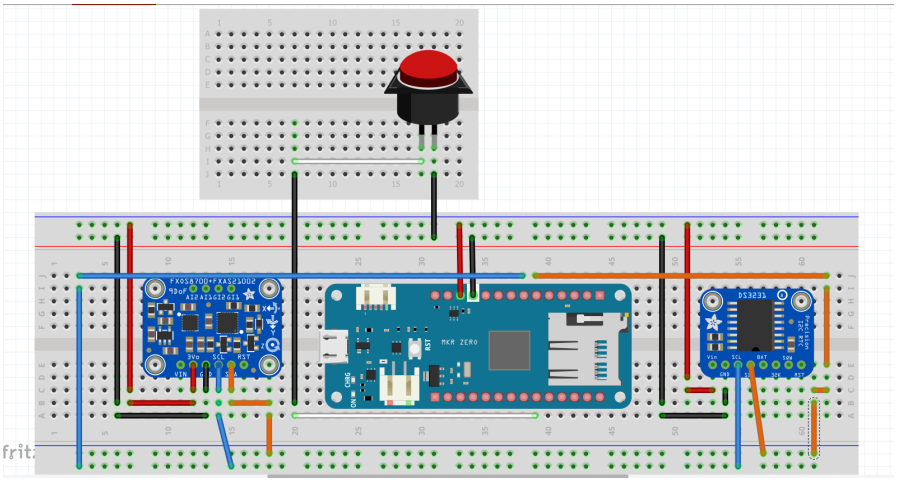


Figura 4.6: Conexión del Imu con la placa

En la imagen de arriba podemos ver el esquema completo, la Imu es el sensor que está a la izquierda de la placa de arduino. Como podemos observar en la figura ya aumentó un poco la complejidad del circuito y con ello el número de cables. El esquema conexión sería el siguiente:

- El pin *Vin* del RTC la conectamos al pin Vcc de la placa de arduino(cable rojo)

- EL pin del GND del Imu, igual que el pin GND del pulsador, al GND de Arduino arduino(**cable negro**)

- El pin SCL del Imu, lo conectamos al pin SCL de la placa, el cable azul) y por último el pin SCA, para enviar los datos de la de los sensores, al pin SCA de la placa de arduino, cable naranja)

4.3.2.2 Diseño Software

El diseño software de esta etapa viene siendo el mismo que el de la etapa anterior solo que hemos ampliado algunos elementos, como la capa de guardar los datos importados de arduino en el fichero csv o el conjunto de funciones para importar esos datos. Siguiendo la mecánica de las dos iteraciones anteriores vamos a empezar a diseñar el software de arduino. Tendremos dos ficheros donde declararemos las cabeceras las funciones a implementar, uno para las funciones relacionadas con la Imu y otro con la operaciones de escritura y lectura de la tarjeta sd. Las operaciones de la parte del sensor serán las siguientes:

- initSensors
- prepareSensors
- Y tres funciones que nos darán los valores de los movimientos capturados por el imu, que serán getRoll, getYaw y getPicth.

Para las operaciones relacionadas con la sd se implementarán 5 operaciones:

- abrirSd → Como su propio nombre indica nos permitirá abrir la tarjeta SD.
- abrir Fichero → Nos permitirá abrir un fichero guardado en la sd
- abrirFicheroLectura → Nos permitirá abrir el fichero pero solo en modo lectura.
- escribirFicheroSD → Esta función nos permitirá guardar los datos recibidos por parámetro en la tarjeta sd.
- getFichero → Nos permitirá recuperar el fichero abierto.

Para la parte de diseño de java expandiremos el diagrama anterior añadiendo algunas operaciones a la *interface* ArduinoServiceInterface y creamos la capa que se encargará de guardar los datos en el fichero csv. El diagrama UML[3] que mostraremos a continuación solo refleja la relación entre la capa controladora y la capa modelo para facilitar la lectura, ya que el resto de relaciones expuestas en la iteración anterior no varían.

[3] Aquí solo se muestra un parte del diagrama UML, para que se puedan apreciar mejor los detalles, el diagrama completo estará incluido como un anexo en la sección de apéndices.

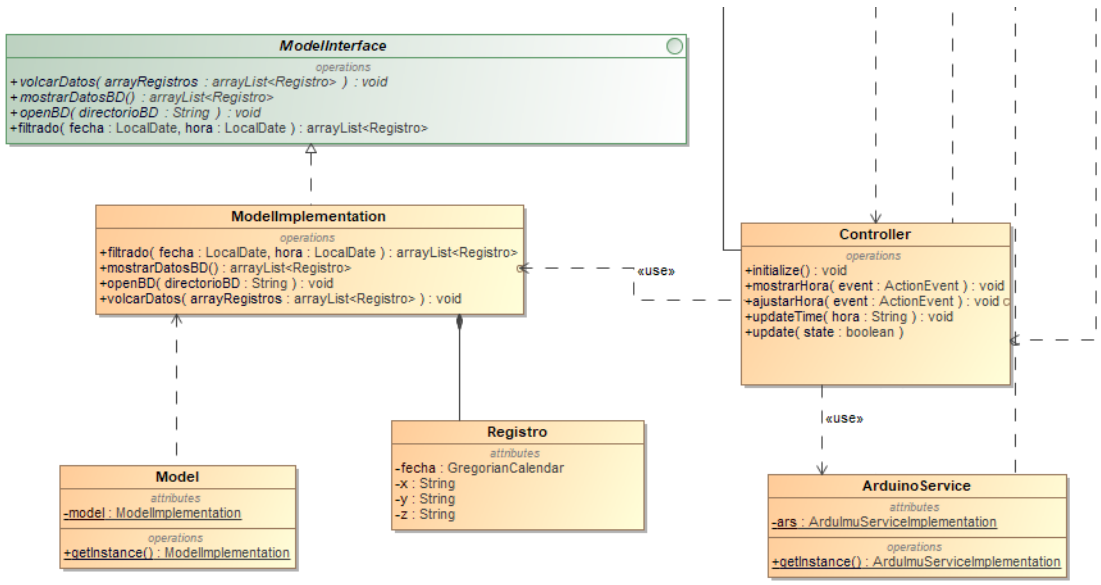


Figura 4.7: Diagrama UML Capa lógica de negocio y controladora

La estructura de la capa modelo viene a ser la misma que la capa ArduinoService, pues está consta de una *interface* donde declaramos las cabeceras de las funciones expuestas, una implementación de esas funciones y la clase que aplica el patrón instancia una de la implementación del modelo. Por último, decir que existe una entidad Registro que tendrá cuatro atributos: fecha, valor x, valor *y* y el valor z, que corresponden con los datos enviados del imu. El funcionamiento de esta parte sería el siguiente:

1. Si se pulsa el botón de copiar datos, el controlador llamará en primer lugar al método de ArduinoServiceImplemention copiarFichero, una vez a recibido los datos, volverá a tomar el mando el controlador e invocando al modelo para guardar los datos en el fichero csv.

2. En el caso de que queramos ver los datos almacenados en el fichero csv, el controlador invocará al modelo para recuperar dichos datos y llamará a la vista para que los muestre en pantalla.

3. Por último, si ejecutamos la opción de filtrar datos el controlador volverá a llamar al modelo, pero esté ultimo no volverá a acceder al fichero csv ya que tiene los datos guardados en memoria y una vez los valores han sido filtrados el controlador le indica a la vista que se refresque para mostrar esos valores.

4.3.2.3 Interfaz de usuario

Vamos a ampliar la pantalla diseñada en la fase anterior, en este caso añadiremos más botones al menú lateral, un botón en el panel derecho para mostrar el subpanel donde mostraremos los datos recogidos de la tarjeta sd en un cuadro de texto.

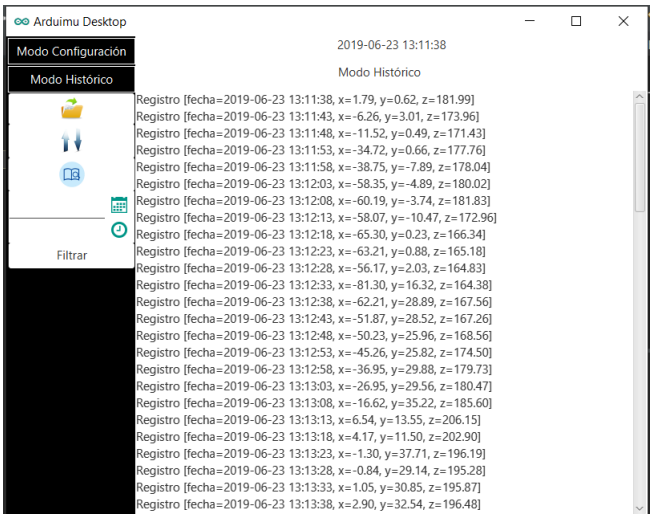


Figura 4.8: Segunda Pantalla

El funcionamiento del menú es mismo que en la primera pantalla, si pulsas en el botón del menú lateral se te mostrarán las opciones de ese apartado. Si pulsamos sobre el botón mostrar datos, se visualizarán en el panel derecho, a esos datos mostrados los podremos filtrar, usando los filtros que se ven en el menú lateral. En esta fase el botón Modo Histórico del panel derecho solo mostraría u ocultaría el cuadro de texto.

4.3.3 Implementación y pruebas

Siguiendo con el patrón las iteraciones anteriores, comenzaremos implementando el código de arduino, empezando con la preparación y captura de los datos de los sensores. Para poder utilizar los sensores tenemos que iniciarlos, que se hace de la siguiente manera:

```
1  void initSensors(){
2    if(!gyro.begin()){
3      Serial.println("Giroscopio no detectado");}
4    if(!accelmag.begin(ACCEL_RANGE_4G)){
5      Serial.println("Acelerometro y magnetómetro no conectados");}
6    filter.begin(10);// Esto indica cuantas muestras por segundo se
     toman}
```

Listing 4.11: Inicialización Imu

La inicialización es muy simple, se utiliza la instrucción begin y si no pudiese iniciarse lo más seguro es que el sensor no está conectado. El siguiente paso fue preparar los sensores para la lectura, en este pasó tenemos que enlazar las variables anteriores que le aplicamos el begin con los eventos de los sensores, además a mayores tenemos que calibrar el magnetómetro, indicándole unos valores que obtenemos a partir de un programa para arduino, proporcionado por el fabricante. Ademas de hacer el begin tambien unimos el evento de los tres sensorres a nuestro sensor para poder capturar los datos.

```
sensors_event_t gyro_event;
sensors_event_t accel_event;
sensors_event_t mag_event;
//Sincronización eventos con los sensores
gyro.getEvent(&gyro_event);
accelmag.getEvent(&accel_event, &mag_event);
//Aplicación de los valores de compensación para X
//se aplican las mismas operaciones para Y y Z
float x = mag_event.magnetic.x - mag_offsets[0];
float mx = x * mag_softiron_matrix[0][0] + y *
  mag_softiron_matrix[0][1] + z * mag_softiron_matrix[0][2];
float gx = gyro_event.gyro.x + gyro_zero_offsets[0];
//conversion de de rad/s a °/s
 gx *= 57.2958F;
 //Actualización del filtro con los valores capturados
filter.update(gx, gy, gz,
accel_event.acceleration.x, accel_event.acceleration.y,
  accel_event.acceleration.z,
mx, my, mz);
```

Listing 4.12: Preparación Imu

En esta parte lo más importante es la calibración del magnetómetro, dado que de los tres sensores es más sensible a los cambios. Y es el más sensible porque es más probable que le alcance una onda electromagnética al magnetómetro, que descompensar el acelerómetro por un cambio de la gravedad, o que acaso que salgamos de la tierra no pasará. Entonces con el offset generado por la aplicación del fabricante se aplica la compensación multiplicando el valor del magnetómetro por el offset obtenido. A mayores el filtro usado para obtener los valores de los sensores solo admite grados por segundo para el giroscopio y el sensor nos lo devuelve en rad/s por lo que hay que convertirlos usando aplicando la siguiente transformación $1rad/s\, es\, equivalente\, a\, 57.2958/s$ entonces el valor obtenido por el sensor en radianes segundo hay que multiplicarlo por 57.2958, la F que aparece a continuación es de Float. Y una vez hecho le pasamos al filtro los valores actualizados. El filtro es un elemento que nos devuelve todos los valores de los sensores de manera unificada, es decir, en forma de roll, yaw y pitch. Por último implementamos tres funciones que nos devuelven el valor los valores

llamando la función filter.getRoll(), filter.getPitch() y filter.getYaw().

Ahora vamos a crear la parte correspondiente a la lectura y escritura en la tarjeta sd. Estas funciones son muy simples, ya que prácticamente se escriben en muy pocas líneas de código. Para operar con la tarjeta SD usaremos una librería que proporciona arduino. Para abrir la tarjeta sd usaremos *SD.begin()*, para poder abrir el fichero usaremos *SD.open(Fichero,ModoApertura)* donde le indicaremos el nombre del fichero a abrir y el modo, en este caso lectura o escritura y la última función será la de escribir los datos en el fichero, que utilizaremos el fichero abierto anteriormente, que se llamará file por ejemplo, y procederemos a hacer file.println(datos), entonces nos guardara los datos al final del fichero. Ya está listo el código en C++, vamos con el código java.

En primer lugar crearemos la entidad Registro, formada por los cuatro atributos que mencionamos antes más los getters, los setters y el toString(). Lo siguiente fue crear las funciones para operar con el fichero csv. La primera ellas fue *openBD*, cuyo código mostramos a continuación:

```
1 public void openBD(final String directorioBD) {
2   archivoCSV = directorioBD;}
```

Listing 4.13: Abrir fichero

Esta función lo que hace es recibir la ruta del fichero nada más que la recibe del controlador cuando desde la vista seleccionamos el fichero csv. Una vez que tenemos el fichero abierto podemos darle a copiar fichero desde arduino para hacerlo el controlador llamará al método copiar fichero de ArduinoServiceImplementation, el cual nos devolverá un array de registros. Y una vez tenga ese array de registros se lo pasará a la capar modelo para que lo escriba en el fichero csv como aparece en el siguiente fragmento de código:

```
1   @Override
2 public void volcarDatos(final ArrayList<Registro> arrayRegistros) {
3     writer = Files.newBufferedWriter(Paths.get(archivoCSV),
4     StandardOpenOption.APPEND);
5     csvPrinter = new CSVPrinter(writer,
6     CSVFormat.DEFAULT.withDelimiter(separador));
7     for (final Registro registro : arrayRegistros) {
8         csvPrinter.printRecord(Arrays.asList(
9         Utils.calendarToString(registro.getFecha()),
10        registro.getX(), registro.getY(), registro.getZ()));
11        csvPrinter.flush();}
12 }
```

Listing 4.14: Volcar datos fichero

La forma de volcarlos en el fichero es ir recorriendo el array de registros y utilizar un csvPrinter para que los escriba en el archivo, el método Utils.calendarToString convierte un fecha en

formato calendar a string. La función mostrarDatosBd es para recuperar los datos del fichero y crear un array de registros, que lo recibe el controlador y se lo pasa a la vista. Por último la función de filtrado lo que hace es ir comparando la fecha pasada por parámetro con la fecha de cada registro del array y las que no coincidan se eliminan del array y una vez recorrido se envía al controlador para que refresque la vista con los nuevos datos. Más adelante esta funcionalidad se podría cambiar haciendo que el controlador se encargue de borrar esos datos porque ya tiene el array de registros creado y no hace falta acceder a la capa modelo.

En está iteración para verificar el funcionamiento de la capa modelo creamos alguna clases de prueba con junit. Realizamos las siguientes pruebas:

- Prueba de la función de volcado de datos a un fichero
- Prueba de recuperar los datos del fichero
- Prueba de filtrado de los datos recuperados del fichero

Además también se realizaron pruebas manuales de poner a ejecutar el programa de arduino y que se grabaran datos en la tarjeta sd y después posteriormente copiarlos al ordenador a través de puerto serie, mostrar los datos copiados y filtrarlos. Además también se implementó se muestre un aviso cuando intentas ver el histórico sin tener el fichero abierto.

4.4 Cuarta Iteración: Ver funcionamiento Imu en tiempo real

4.4.1 Análisis

Esta iteración va a ser un poco distinta a las anteriores ya que no tenemos un diseño hardware que crear porque el circuito ya está completo. Una vez creado la acción de registrar datos en la tarjeta sd, vamos a crear la funcionalidad de ver los movimientos de la imu en tiempo real, es decir, si yo muevo el sensor de una forma poder visualizar como una figura se mueve de la misma forma en la aplicación de escritorio.

4.4.1.1 Requisitos funcionales

- **ArduImu-RF-4-A** → Añadir al menú del código de arduino una entrada para entrar en el modo tiempo real.
- **ArduImu-RF-4-B** → Añadir un nueva entrada al menú de IU de la aplicación escritorio.
- **ArduImu-RF-4-C** → En esa nueva entrada se añadirá un botón que implementará la acción de apagar o encender el modo tiempo real.

- **ArduImu-RF-4-D** → También insertaremos un nuevo botón, al lado del botón de modo histórico, el cual nos permita acceder al panel del modo tiempo real para visualizar los movimiento.

4.4.1.2 Requisitos no funcionales

- **ArduImu-RNF-4-A** → De la misma que en el requisito **ArduImu-RNF-2-B**, la IU no podrá congelarse mientras se estén recibiendo los datos del modo tiempo real.

4.4.2 Diseño

4.4.2.1 Diseño Software

El diseño de la aplicación en cuanto a su estructura no va a variar mucho, dado que solo añadiremos una clase interna más para capturar los movimientos del imu en segundo plano para evitar la congelación de la IU, se añadirán nuevos métodos la capa controlador y la capa ArduinoService. En cuanto al código de arduino solo añadiremos lo necesario en el menú para enviar los datos en tiempo real. Como se puede apreciar en el siguiente diagrama, la clase interna leerEnTiempoReal, se encargará de leer los datos recibidos por el puerto serie y a su vez el controlador la observará para cuando se hayan recibido los datos dar la orden a la vista para que los pinte en pantalla. En la clase controlador también existen los métodos cambioReal, que iniciará el modo tiempo real, y cambioTreal que detectará si el estado del botón que controla el apagado o encendido y actuará en consecuencia.

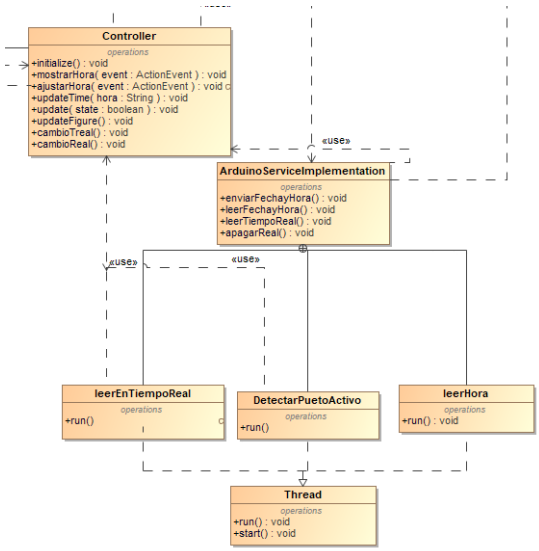


Figura 4.9: Diagrama extendido con el modo tiempo real

4.4.2.2 Interfaz de usuario

En esta sección volvemos a ampliar el diseño de la interfaz de usuario, está vez a añadiendo los elementos para el control de modo tiempo real. Como mencionábamos en los requisitos tendremos un nuevo apartado en el menú con un botón estilo que funcione como un conmutador entre las posiciones on y off. Estando en esta ventana podríamos acceder al modo histórico para poder ver los datos guardados en el csv y el modo tiempo real se seguiría ejecutándose por si volviésemos a ese estado. Una cosa importante es que si pulsamos el botón de copiar los datos de la sd mientras funciona el modo real, esté se apagará.

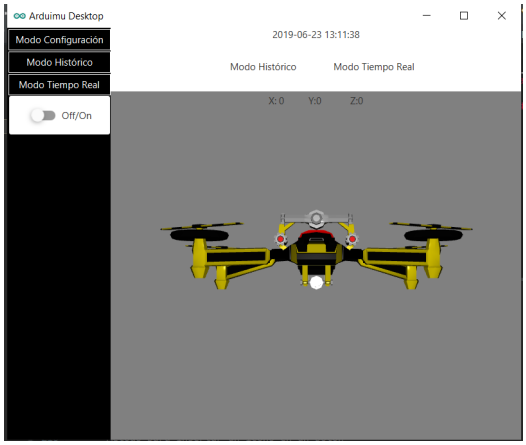


Figura 4.10: Tercera Pantalla

En el panel derecho insertaremos otro botón para mostrar el panel del modo tiempo real y se añadirá una imagen 3d de un dron para representar los movimientos de Imu y en la parte superior del subpanel mostraremos unas etiquetas con los valores numéricos en los tres ejes.

4.4.3 Implementación y pruebas

Como mencionamos arriba, en el código de arduino solo añadimos la entrada en la función del menú que permite él envió de los datos continuamente.

```
1 if (option.compareTo("tiempo real start")==0 ||
      option.compareTo("tiempo real stop")){
2 while( (ISRCounter==1) &&
      (option.substring(12,option.length()).compareTo("start")==0)){
3     prepareSensors();
4     Serial.print(getYaw());Serial.print(" ");
5     Serial.print(getPitch());Serial.print(" ");
6     Serial.print(getRoll());Serial.print("\n");
7     delay(100);
```

Listing 4.15: Envio datos en tiempo real

```
1  if ( Serial.available() > 0 &&
     Serial.readString().compareTo("tiempo real start")!=0){
2    break;
3  }}
```

Listing 4.16: Envio datos en tiempo real continuación

El funcionamiento es simple, mientras el contador esté en modo 1 y la opción indicada sea ”start”, se enviarán los datos por el puerto serie, con un retardo de 100ms. Esté tiempo se puso para que lleguen los datos completos entre cada envío, porque si era menor o no tenía a veces podrían llegar cortados. Y para salir de este modo se puede hacer de varias maneras, presionando el pulsador conectado a la placa de arduino, que hará que cambie de estado el contador de menú y termine el bucle while o poniendo en off desde la IU que mandará la señal ”tiempo real stop” que hará que se interrumpa el bucle while con la sentencia break. En cuanto a la implementación de la parte de java, solo mostraremos el código de la función que captura los datos en tiempo real, ya que es la más importante de esta iteración.

```
1  class leerEnTiempoReal extends Thread {
2    @Override
3    public void run() {
4    while (!apagado) {
5      final BufferedReader reader = new BufferedReader(
6      new InputStreamReader(serialPort.getInputStream()));
7    try {
8      final String line = reader.readLine();
9      final String[] res = line.split(" ");
10     try {
11       observadores.updateFigure(Double.valueOf(res[2]),
12       Double.valueOf(res[1]), Double.valueOf(res[0]));
13     } catch (final NumberFormatException e) {
14       continue;}}
```

Listing 4.17: Método encargado de recibir los datos en tiempo real

Los datos recibidos por el puerto serie vienen en forma de líneas, entonces lo que hacemos es ir leyendo todas las líneas, estas ya nos traen los tres valores que necesitamos, separados por espacios. Entonces a esa línea que se hemos leído con readLine(), le aplicamos un split para obtener cada elemento separado. Una vez que los datos han sido separados, procedemos a convertirlos a double, que los podíamos a ver dejado en forma de String, pero haciendo está conversión nos aseguramos que los datos recibidos sean numéricos y que de repente que la vista recibiese cualquier carácter que no fuese un número y podría producirse un error. Si por cualquier motivo no puede hacer la transformación a doble, saltaría la excepción NumberFor-

matException y lo que hacemos es desechar la línea y pasar a la siguiente. Si todo fue bien el controlador observa que los datos están listos y se los pasa a la vista para que pueda actualizarse. En este apartado vamos a explicar una cosa que no explicamos en los anteriores. Cuando el controlador recibe los datos del puerto serie, los está recibiendo en un thread distinto al thread en el que se está ejecutando la vista, por lo que el controlador no se los puede pasar a la vista directamente. Para solucionar ese problema usamos la función Platform.runLater, que inicializamos con un objeto runnable, donde en su método run hacemos la operación de actualizar la vista.

```
@Override
public void updateFigure(final double roll, final double pitch,
final double yaw) {
  Platform.runLater(new Runnable() {
  @Override
  public void run() {
    rotateX.setAngle(roll);valueX.setText(String.valueOf(roll));
    rotateY.setAngle(yaw);valueY.setText(String.valueOf(yaw));
    rotateZ.setAngle(pitch);valueZ.setText(String.valueOf(pitch));
    }
  });}
```

Listing 4.18: Actualizar figura

El método Platform.runLater, hace que podamos ejecutar el runnable que le indicamos en un subproceso de JavaFX y ahí ya podemos actualizar los datos de la IU. Hay que apuntar que ahora mismo solo tenemos un método runnable funcionando, pero si el futuro tuviésemos varios métodos runnables a la vez, quizás está solución aplicada no fuese la óptima, porque tendremos varios objetos runnables en una cola de eventos pudiendo ocasionar fallos de funcionamiento de la aplicación. Los otros métodos creados en en ArduinoService realizan las funciones de mandar mensajes por el puerto serie a arduino, por ejemplo, la función apagarReal, le envía el mensaje off para que pare de enviarnos datos del sensor por el puerto serie. Para terminar con esta sección vamos a proceder a probar si funciona todo el desarrollo y solo se puede probar de una manera, que es ejecutando todos los programas en su conjunto, como si todo el sistema fuese una caja negra y no supiésemos que hay dentro. En primer lugar conectamos la placa al pc y presionamos el pulsador para ponerlo en el modo 1, posteriormente arrancamos la aplicación de escritorio y hacemos click en el botón del panel derecho ModoTiempoReal, este le manda un mensaje a arduino para que empiece a enviar datos por el puerto serie, una vez que se empiezan a recibir datos, visualizamos que la figura del dron se empieza a mover a la vez que movemos el sensor, por último ponemos el botón de apagar en off y al instante el imu se deja de mover porque ya no estamos recibiendo datos.

4.5 Quinta Iteración: Añadir el modo sueño

4.5.1 Análisis

En esta última iteración crearemos la funcionalidad de poner en modo sueño la placa de arduino, este modo deshabilita muchos de los componentes hardware de arduino para gastar menos energía y que el procesador pueda entrar en modo bajo consumo y que no este trabajando todo el tiempo lo que nos proporcionara cierto ahorro de energía. Para implementar el modo bajo consumo en la placa arduino mkrzero utilizaremos la librería ArduinoLowPower. Está librería nos ofrece distintos modos de bajo consumo:

- Modo Idle. Este modo tiene el tiempo de activación más rápido y además la cpu está en un estado de reposo.
- Modo sleep. Este tiene un tiempo de reactivación más lento, solo desconecta algunos dispositivos externos como, el conector usb.
- Modo deep sleep. Este modo tiene el tiempo de reactivación más lento de los tres y detiene todos los periféricos, excepto el rtc.

A los tres métodos se le puede indicar el tiempo que tiene que estar en ese estado de reposo, pasándole el número de milisegundos por parámetro, y una vez pasó ese tiempo la placa se despierta. Se puede dar el caso que pongamos a dormir la placa durante un tiempo infinito y que solo queramos que se despierte cuando ocurra algún evento externo, ese suceso vendrá en forma de interrupción. Para se despierte cuando se produzca una interrupción utilizaremos la función attachInterruptWakeup() para unir un pin de interrupción a una función igual que la función de attachinterrutption de arduino.

4.5.1.1 Requisitos funcionales

- **ArduImu-RF-5-A** → Poner en modo sueño la placa de arduino cada cierto tiempo.
- **ArduImu-RF-5-B** → Se debe poder despertar a placa mediante el pulsador
- **ArduImu-RF-5-C** → El tiempo que estará dormido se le indicará a través da la aplicación de escritorio.

4.5.1.2 Requisitos no funcionales

- **ArduImu-RNF-5-A** → Cuando se reciba el valor del modo sueño se guardará en la tarjeta sd, para que en caso de corte de energía pueda recuperarse cuando se vuelva a encender la placa de arduino.

4.5.2 Diseño

4.5.2.1 Interfaz de usuario

A la interfaz de usuario se le aplicarán muy pocas modificaciones pues solo añadiremos un deslizador para poder elegir el tiempo que estará en el modo sueño, un botón aceptar para enviar el valor y una etiqueta para saber el valor actual. A continuación os mostramos la pantalla:

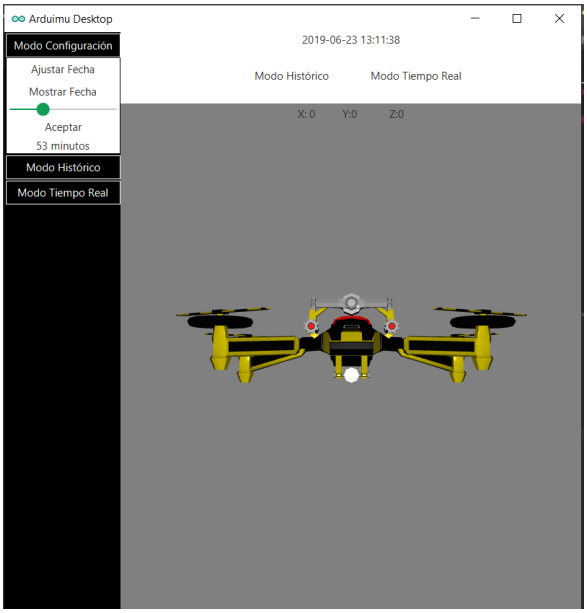


Figura 4.11: Controles para configurar el modo sueño

4.5.3 Implementación y pruebas

En está iteración ya no realizaremos el apartado de diseño software porque se va a respectar el diseño del sprint anterior y solo se añadirán algunos método extras. Comenzando por el software de arduino configuraremos el modo sueño, para ello al modo grabar le añadiremos algunas lineas:

```
Serial.flush();Serial.end();
sscanf(intervalo.c_str(), "%d", &intervaloN);
while (1 && ISRCounter==2){
  LowPower.sleep(intervaloN);
  prepareSensors();abrirFichero();
  escribirFicheroSD(obtenerFechayHora(rtc)+";"+getRoll()+";"
  +getPitch()+";"+getYaw());}
```

Listing 4.19: Entrando en modo sueño

Antes de llamar a la función sleep de LowPower tenemos que asegurarnos que no hay datos salientes del serial con la función Serial.flush() y procedemos a cerrarlo. Una vez que el puerto serie esté cerrado convertirmos el dato intervalo, que esta en forma de string, a un entero con la función sscanf y se lo pasamos a la función sleep. Una vez transcurrido el tiempo indicado, el sistema se despertará obtendrá una lectura del sensor con la función prepareSensors y los almacenará en la sd y volverá ponerse en modo bajo consumo. A mayores en la función setup usamos la función attachInterruptWakeup() para que cuando se produzca una interrupción se despierte. La interrupción esta unida a aumentarContador, para que cuando presionemos el pulsador, avance el contador y salga del bucle.

Una vez implementada el modo sueño en arduino tenemos que enviárselo desde la aplicación de escritorio, para poder hacerlo hemos añadido los siguientes componentes:

```
1 <JFXSlider fx:id="intervalo"/>
2 <Button fx:id="aceptarIntervalo" onAction="#EnviarIntervalo"
    text="Aceptar"/>
3 <Label fx:id="etiquetaValorIntervalo"/>
```

Listing 4.20: Elementos para configurar el modo sueño

Como mostrábamos en el la figura 4.11. En la etiqueta Button, en el atributo onAction le asignamos el valor ”#EnviarIntervalo” que es el método encargado de enviar el valor seleccionado a arduino. La forma de enviarlo es la misma que aplicamos en el resto de datos enviados, primero obtenemos el Stream de salida con *outstream = serialPort.getOutputStream()* y posteriormente enviamos el dato con un marca para señalar que estamos enviando el dato correspondiente al modo sueño *outstream.write((”Intervalo” + intervalo).getBytes());*. Para probar este desarrollo conectamos la placa y arrancamos la aplicación de escritorio, nos pusimos en el modo configuración y procedimos a enviar el dato. Con el dato recibido en arduino, 2 minutos para realizar la prueba, y esperamos un par de minutos. Después volvimos al modo configuración y desde la aplicación del PC importamos los datos registrados y los mostrarnos en el histórico, observando 3 medidas con una separación de dos minutos.

Figura 4.12: Datos cada 2 minutos

Con esto damos por finalizado la iteración y la primera versión de la aplicación, tanto de

escritorio como del programa de arduino.

Como análisis final del desarrollo decir que los plazos marcados antes de iniciar el desarrollo no los hemos cumplido, ya que no habíamos incluido que íbamos a tener que parar de trabajar debido a los exámenes. Al final la distribución temporal del proyecto quedo de la siguiente manera:

Analisis	**12,5 días**	**mié 13/03/19**	**vie 29/03/19**
Analisis Requisitos Funcionales	50 horas	mié 13/03/19	jue 21/03/19
Analisis Requisitos No Funcionales	50 horas	jue 21/03/19	vie 29/03/19
Diseño	**39,5 días**	**vie 29/03/19**	**jue 23/05/19**
Diseño Hardware	100 horas	vie 29/03/19	mar 16/04/19
Diseño Software	120 horas	mié 17/04/19	jue 23/05/19
Implementación y pruebas	**62,63 días**	**vie 24/05/19**	**mar 20/08/19**
Implementacion	200 horas	vie 24/05/19	lun 29/07/19
Pruebas	125 horas	mar 30/07/19	mar 20/08/19

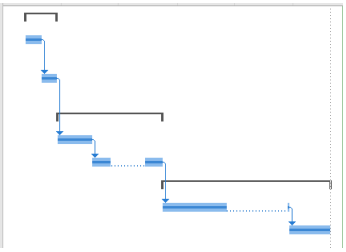


Figura 4.13: Diagrama de Gantt

Como observamos en la imagen en dos están divididas y separadas por unos .... Esto significa que en esos instantes no estuvimos trabajando en el proyecto, ya que tuvimos que hacer una parada para estudiar para los exámenes.

Capítulo 5

Conclusiones

Cuando empecé a trabajar en el proyecto de creación de la unidad de medición inercial no sabía muy bien por dónde empezar. Antes de entrar en el grado si había trabajado un poco con arduino, pero en la actualidad dominaba con mayor soltura el desarrollo en java. Después de mirar documentación por Internet para recordar cómo funcionaba arduino, decidí empezar por ahí, y una vez hechas algunas partes arduino, empezar a realizar la aplicación en java. Ahora una vez finalizado todo el proceso, me doy cuenta que acerté eligiendo empezar por la parte de arduino, porque si hubiese empezado por la parte de java, hubiese tenido la aplicación de escritorio hecha pero sin saber si iba a funcionar, porque básicamente no lo iba a poder probar hasta que acabara la otra parte. Para la parte de arduino, como mencione al principio, he tenido recordar algunas cosas respecto a su funcionamiento. Esas cosas fueron a que cuando se inicia la placa de arduino puedes configurarla algunas configuraciones propias de arduino o de tu programa en la función setup. Una vez se acabó de ejecutar el setup, llama a la función loop, que se esta ejecutando siempre como si fuera un bucle infinito. Una vez recordado como funcionaba arduino aprendí a arreglar el efecto rebote que produce un pulsador. Para poder implementar el programa de arduino antes he tenido que investigar como funcionaban los dispositivos que iba a utilizar. Esos dispositivos fueron el rtc, para el que tuve que averiguar como conectarlo, como ajustar la hora y como poder obtenerla. En cuanto al imu, primero tuve que documentarme para saber que tipos de movimientos podía capturar(roll, pitch, yaw) y como tenía que capturarlos. A parte de de como obtener los datos del imu, descubrí que a veces el mangnetómetro del imu se puede descalibrar y para eso hay que ejecutar un programa que proporciona el fabricante para volver a calibrarlo. Estos datos obtenidos tanto del Imu como del rtc los tenia que enviar por puerto serie al ordenador y cuado recibía esos datos en la aplicación que se desarrolló para el pc, observé la primera vez que interfaz de usuario se congelaba, para solucionar este problema tuve que implementar threads en java, que nunca había usado. Al usar otro plano para recibir los datos del puerto serie, me encontré con otro problema, pues no se podían enviar los datos desde un thread

distinto al thread en el que se ejecutaba la interfaz gráfica. Para poder pintar esos datos tuve que utilizar la función Platform.runlater para recibir los datos en un subproceso de javaFx y así poder enviárselos a la vista. Para finalizar tuvimos que tratar con la gestión de energía en arduino. Es decir, poder ponerla en un modo sueño o de bajo consumo para que no requiera tanto consumo de energía. Está parte me costó un poco más, puesto que cada vez que entraba en este modo se deshabilitaba el puerto serie y no podía saber que estaba pasando por detrás. Otro problema que me sucedió fue que si el puerto serie esta abierto o tiene datos pendientes por enviar, no entra en dicho modo y antes hay que asegurarse que el puerto serie no tiene datos pendientes y que esta cerrado.

Está aplicación que he desarrollado puede tener muchas utilidades en el mundo real, cualquier aplicación que requiera detección de cambios en el movimiento ya lo va utilizar, ya que como contamos en la introducción, todos los smartPhones cuentas con este conjunto de sensores y podemos asegurar que todos los días en cada segundo del día se están ejecutando en millones de teléfonos esos sensores, ya que según las últimas encuestas sobre el smartphones por persona alcanzan un total de 5000 millones de usuarios [1] que viene a ser un 66% de los habitantes de la tierra. En otros muchos sectores también se usan las Imus, por ejemplo, en la aeronáutica, en todo tipo de transportes, robótica, etc.

El realizar este proyecto me ha mostrado que a pesar de que parezca que te has olvidado de muchas cosas que has aprendido durante la carrera, no es verdad. Porque al ir avanzando en el proyecto empiezas a recordad como se hacen algunas cosas que has utilizado en otras asignaturas. Por ejemplo, a utilizar patrones de diseño, a diseñar interfaces gráficas de la mejor manera posible y cómoda para el usuario, a integrar diferentes tecnologías y que todas ellas trabajen en común sin saber nada una de las otras.

Aunque damos por finalizado este proyecto, podemos decir que esto solo es una primera versión de la aplicación, ya que podemos desarrollar otras muchas nuevas funcionalidades que no fueron implementadas aún en este desarrollo. Algunas de estas ideas son una extensión de la aplicación existente, por ejemplo, cuando estamos observando el histórico de medidas podemos querer ver a la vez los movimientos en tiempo real, nosotros u otra persona. Pues podíamos habilitar un botón para que el panel de tiempo real se nos abriese en otra ventana. Otra de las mejoras que se podía crear, es que a veces tener que ir con un ordenador a todos lados para poder copiar, ver los datos de la imu o configurarla es un poco incómodo. Pero podríamos realizar una app en android para conectar la imu y transmitir esos datos por usb, evidente el smartphone android tendría que contar con OTG, además al ser android compatible con java podríamos reutilizar algunas partes del código desarrollado perfectamente y así contar con un sistema de captura de datos más manejable. Bueno y aparte de mejorar y crear nuevo software, porque no avanzar más y liberarnos de tener que conectarnos por

[1] https://elpais.com/tecnologia/2018/02/27/actualidad/1519725291_071783.html

usb a arduino y poder hacerlo por wifi. Como mencionamos en el apartado de arduino existen módulos de ampliación para las placas de arduino, como el de wifi, entonces podríamos crear la funcionalidad que la placa pueda enviar los datos registrados en la tarjeta sd por wifi usando protocolos de transporte por internet(http) a un servidor web, al cual nosotros nos conectaríamos para ver dichos datos. Y actualmente como está de moda el internet de las cosas, podríamos montarnos todo el servidor web con una raspberry pi y poder acceder a ella de modo remoto, por si no podemos ir a coger los datos de manera presencial o está en un lugar muy complicado de acceder. Ya que mencionamos de acceder a un raspberry de manera remota, hay que añadir realizar un tarea de vital importancia, proteger el servidor de accesos no deseados, para hacerlo tendríamos que añadir un sistema de seguridad. Como podemos observar se puede ampliar de muchas maneras el proyecto pero en el actual procedemos a finalizar aquí, aunque podíamos seguir contando muchas más cosas sobre el mundo de arduino, IoT, etc.

Apéndices

Apéndice A

Manual De usuario

En esté apartado incluiremos un pequeño manual de uso toda la aplicación desarrollada en este proyecto.

Empezaremos explicando el como funciona el programa en arduino. Su uso es muy sencillo, actualmente hay tres modos de menú, cada opción del menú tiene una posición. Es decir, el modo configuración tiene la posición 1, el modo grabar tiene la posición 2, y el modo parar de grabar tiene la posición 3. Entonces para cambiar de posición en el menú hay que presionar el pulsador, si es la primera vez que lo ejecutas el contador de menú se pondrá a 1 y estarás en el modo configuración, si lo vuelves a pulsar entrarás en el modo 2. Cuando se entra en el modo 2, se inicia el modo sueño, para despertar a la placa de arduino hay presionar otra vez el pulsador, una vez pulsado se despertará y no se volverá a entrar en esté modo hasta que regresemos al modo 2. Decir que despertando al sistema, aun estamos en el modo 2, no avanzamos hacia el tres. Presionando una vez más, entraremos en el modo 3 que lo único que hará es reiniciar los contadores a cero para volver a empezar la cuenta. En el modo configuración puedes hacer lo siguiente:

- Ajustar la fecha y la hora.
- Consultar la fecha y la hora.
- Establecer la frecuencia de lectura del sensor.
- Transferir los datos del fichero csv de la tarjeta microsd al pc.
- Entrar el modo tiempo real.

En el modo 2, modo de grabación de datos en la microsd, lo único que hace es registrar los valores en la tarjeta sd con la fecha y hora de la lectura. Una vez contado los detalles de la parte de arduino, procedemos a detallar el funcionamiento de la aplicación en java. Dado que la aplicación en java tiene una interfaz gráfica vamos a apoyar la explicación sobre ella. Está a

aplicación tiene un menú vertical en el lateral izquierdo, aquí nos encontramos los principales botones:

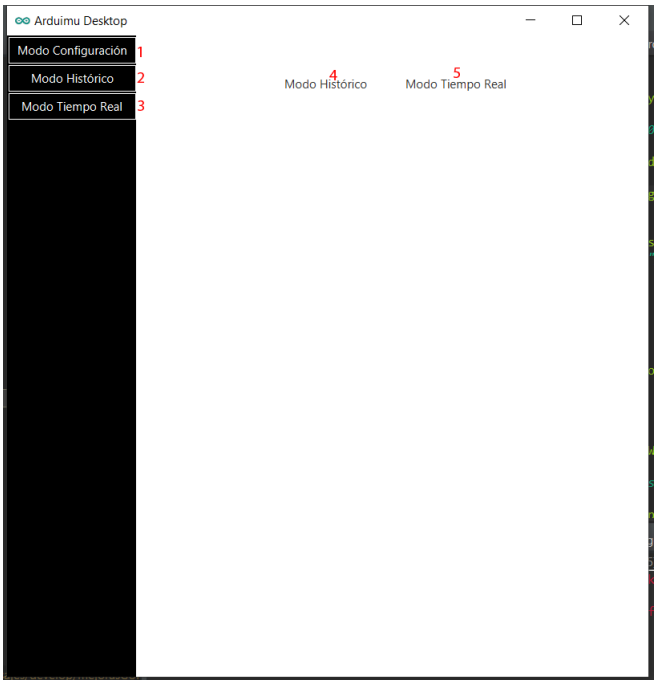


Figura A.1: Vista General

Los botones 1,2,3 son los que despliegan las distintas opciones del menú, mientras que los botones 4 y 5 son los que intercambian entre los distintos paneles, mostrar el panel del registro de datos y el panel de la vista en tiempo real. Cada uno de los botones del menú tiene en su interior otras opciones, en el modo configuración existen las opciones de: ajustar hora, mostrar hora y un deslizador que permite ajustar la frecuencia de lectura y un botón de aceptar para mandar ese valor a arduino.

Figura A.2

En este modo podemos presionar en mostrar fecha y hora, que nos mostrará la hora del rtc en una etiqueta arriba de los botones centrales:

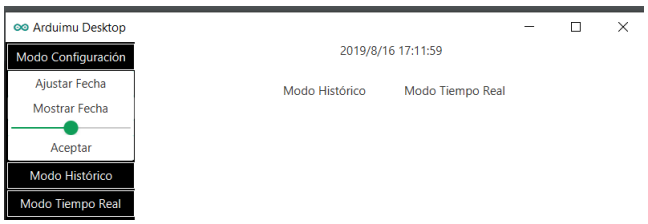


Figura A.3: Mostrando fecha y hora del rtc

El menú del modo histórico agrupa los botones para poder copiar, ver y filtrar el histórico de datos

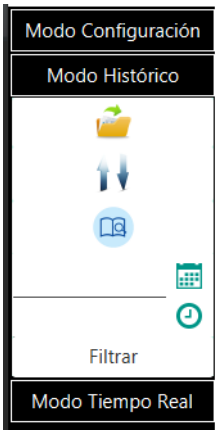


Figura A.4: Menú Histórico

Donde el símbolo de la carpeta permite abrir un fichero:

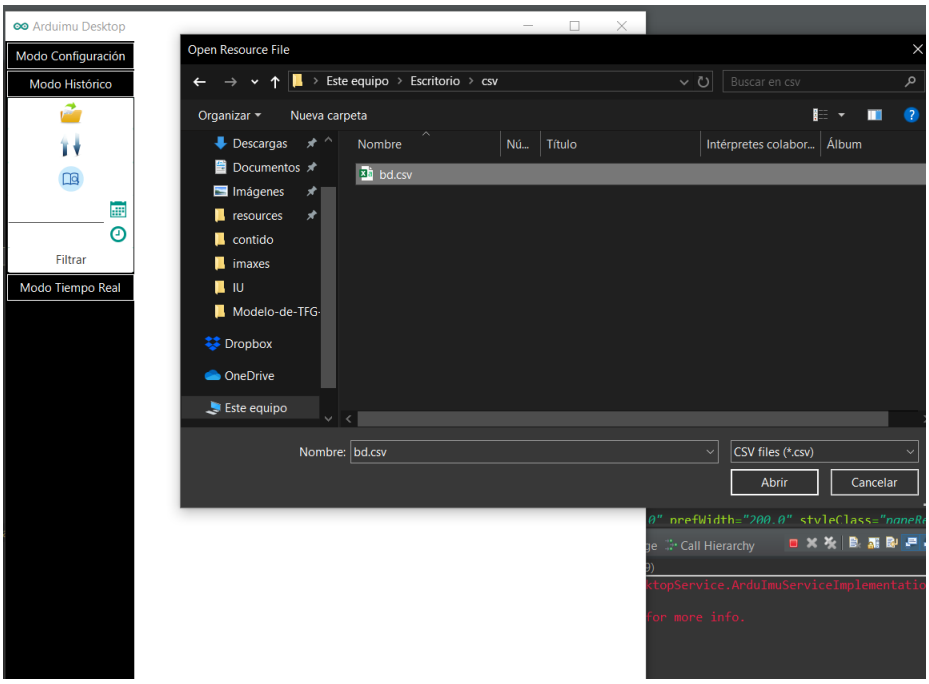


Figura A.5: Abrir Fichero

La imagen de las dos flechas, representa la acción de transferir los datos de la tarjeta sd al pc. El libro con la lupa permite visualizar el banco de datos:

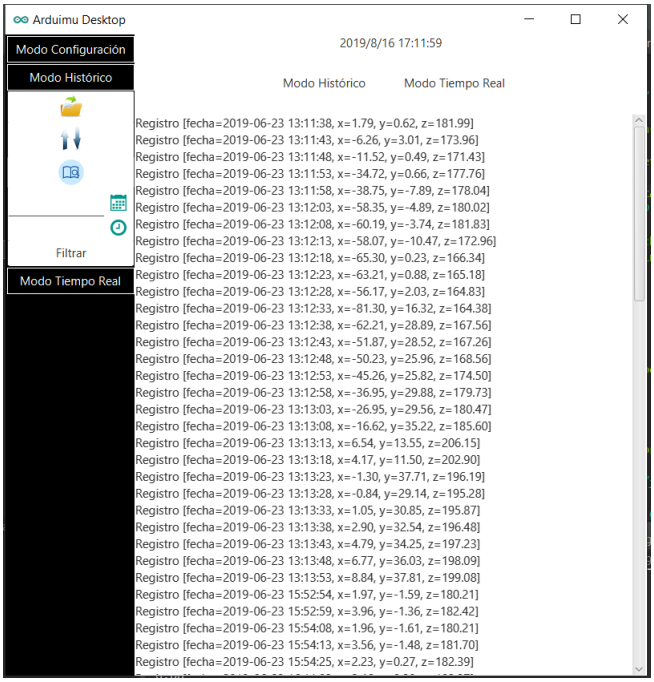
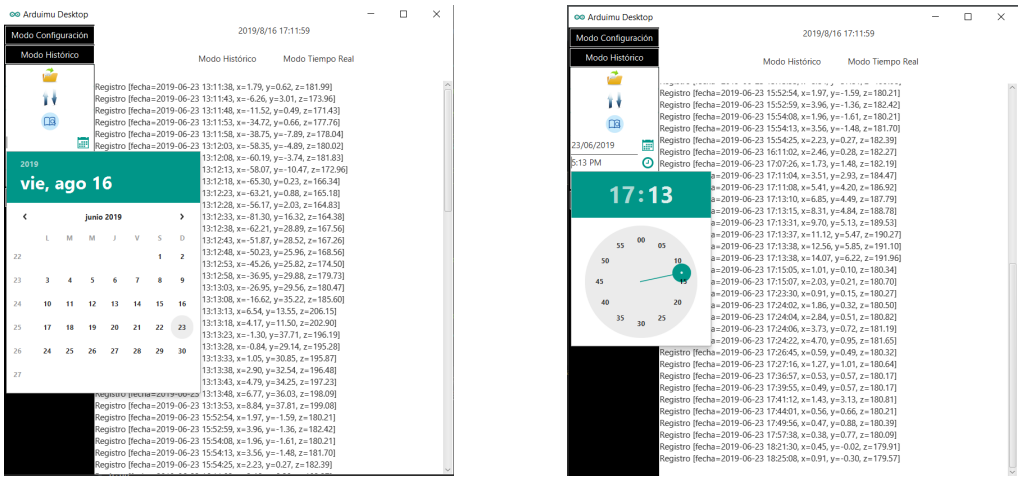


Figura A.6: Registro de datos

Mientras que los botones del calendario, y el reloj permiten insertar la franja horaria a filtrar, posteriormente teniendo que pulsar el botón filtrar para que se apliquen. Los filtros se pueden aplicar de manera conjunta (fecha y hora) o aplicar solo la fecha o solo la hora., incluso si el usuario quiere aplicar primero el filtro de la fecha, pulsar en filtrar y después aplicar el filtro de la hora:



(a) Filtro Fecha (b) Filtro hora

Figura A.7: Filtros

Como se aprecia en la figura A.7b en el recuadro de texto solo aparecen el día seleccionado en el la imagen A.7a, obteniendo un resultado final, con las horas filtradas, de:

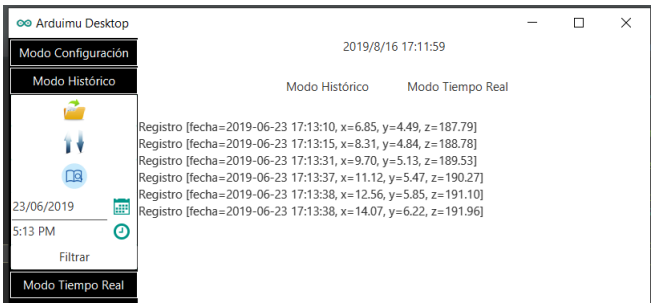


Figura A.8: Filtro final

Visto el modo histórico vamos con el modo tiempo real, ponerlo a funcionar es sencillo, solo tenemos que pulsar sobre el botón Modo tiempo real que está en el panel derecho de la pantalla, y una vez pulsado nos aparecerá la figura que representa el movimiento del Imu:

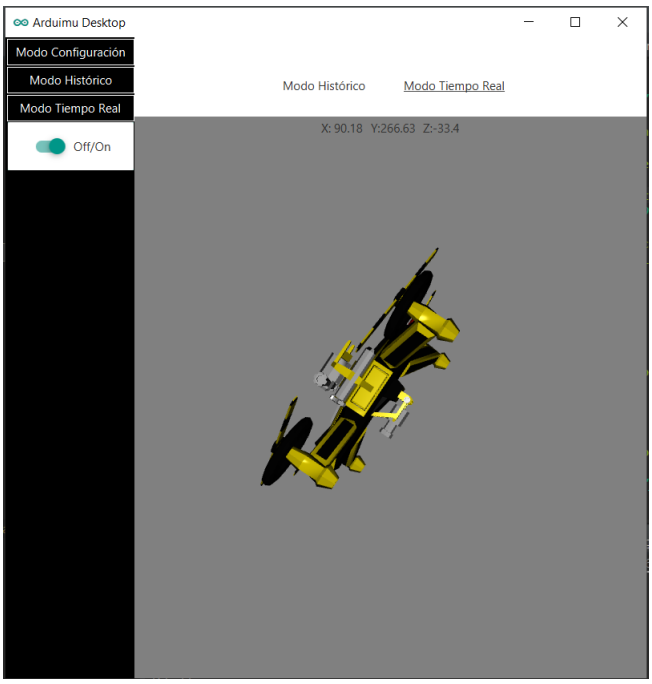


Figura A.9: Modo tiempo real

Justo encima de la figura podemos ver los valores numéricos actualizándose a la par que se mueve la figura. A veces cuando el imu está parado podemos observar que los números siguen cambiando eso es porque el imu sigue detectando variación de los movimientos aunque nosotros no los llegamos a apreciar. Para finalizar con este modo si desplegamos el menú modo tiempo real, no aparece un botón de on/off, que nos permite apagar o encender este modo:

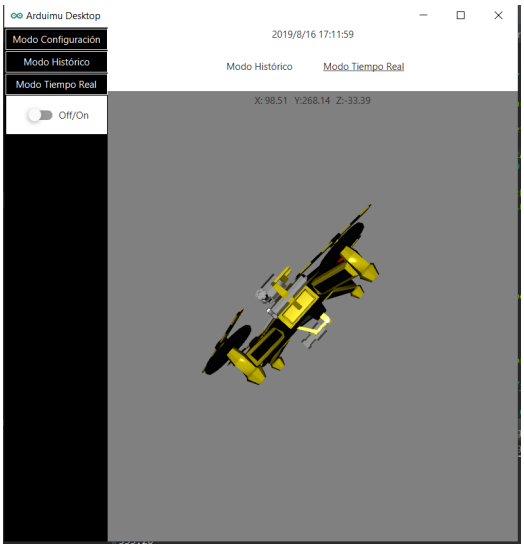


Figura A.10: Tiempo real off

Aunque en la imagen no se puede apreciar, aunque moviésemos el sensor la figura no se movería, destacar que si está en off tampoco se están enviando datos por el puerto serie, otra forma de salir del modo en tiempo real es presionado el pulsador conectado a arduino. Para finalizar con este manual, decir que si la placa de arduino no está conectada al ordenador se deshabilitan algunos botones ya que no podrían funcionar con ella desconectada. Decir también que estos botones se actualizan solos al desconectar/conectar la placa:

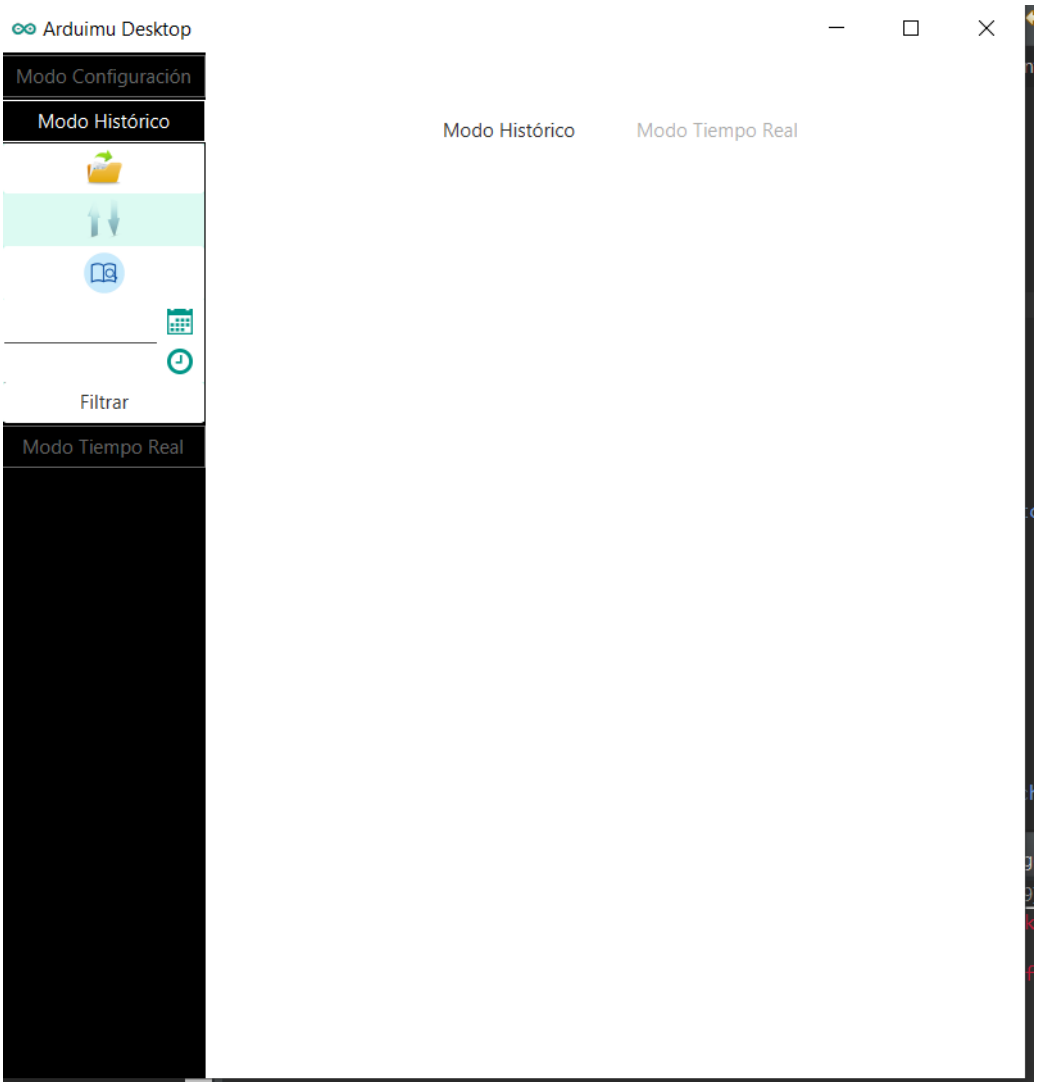
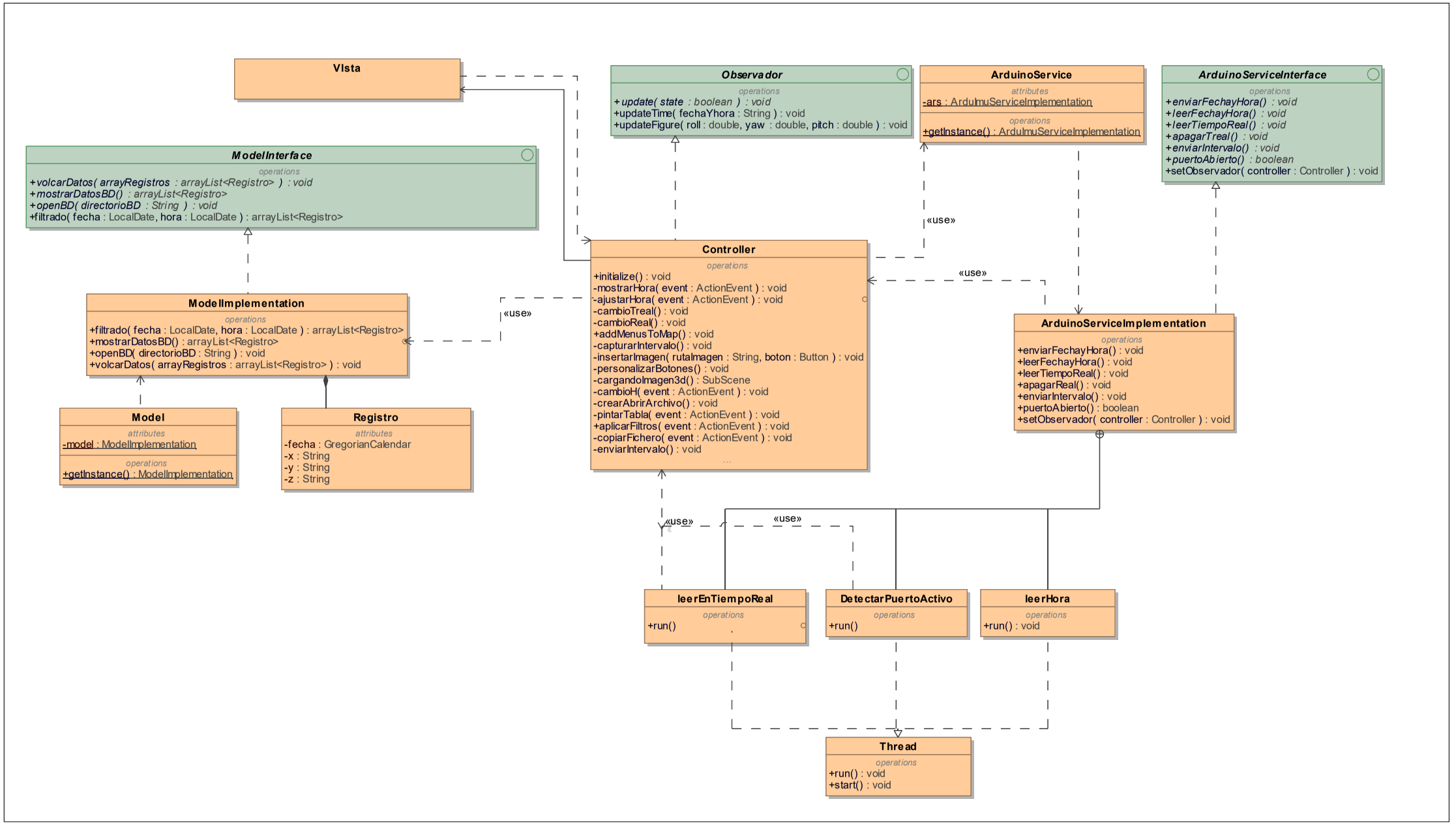


Figura A.11: Botones deshabilitado

En el modo histórico, solo está deshabilitado el botón de transferir datos ya que es el único que requiere la conexión con arduino.

Apéndice B

Material adicional


Vista
Observador
operations
+update(state : boolean) : void
+updateTime(fechaYhora : String) : void
+updateFigure(roll : double, yaw : double, pitch : double) : void
ArduinoService
attributes
-ars : ArduImuServiceImplementation
operations
+getInstance() : ArduImuServiceImplementation
ArduinoServiceInterface
operations
+enviarFechayHora() : void
+leerFechayHora() : void
+leerTiempoReal() : void
+apagarTreal() : void
+enviarIntervalo() : void
+puertoAbierto() : boolean
+setObservador(controller : Controller) : void
ModelInterface
operations
+volcarDatos(arrayRegistros : arrayList<Registro>) : void
+mostrarDatosBD() : arrayList<Registro>
+openBD(directorioBD : String) : void
+filtrado(fecha : LocalDate, hora : LocalDate) : arrayList<Registro>
«use»
Controller
operations
+initialize() : void
-mostrarHora(event : ActionEvent) : void
-ajustarHora(event : ActionEvent) : void
-cambioTreal() : void
-cambioReal() : void
+addMenusToMap() : void
-capturarIntervalo() : void
-insertarImagen(rutaImagen : String, boton : Button) : void
-personalizarBotones() : void
-cargandoImagen3d() : SubScene
-cambioH(event : ActionEvent) : void
-crearAbrirArchivo() : void
-pintarTabla(event : ActionEvent) : void
+aplicarFiltros(event : ActionEvent) : void
-copiarFichero(event : ActionEvent) : void
-enviarIntervalo() : void
...
ModelImplementation
operations
+filtrado(fecha : LocalDate, hora : LocalDate) : arrayList<Registro>
+mostrarDatosBD() : arrayList<Registro>
+openBD(directorioBD : String) : void
+volcarDatos(arrayRegistros : arrayList<Registro>) : void
ArduinoServiceImplementation
operations
+enviarFechayHora() : void
+leerFechayHora() : void
+leerTiempoReal() : void
+apagarReal() : void
+enviarIntervalo() : void
+puertoAbierto() : boolean
+setObservador(controller : Controller) : void
Model
attributes
-model : ModelImplementation
operations
+getInstance() : ModelImplementation
Registro
attributes
-fecha : GregorianCalendar
-x : String
-y : String
-z : String
leerEnTiempoReal
operations
+run()
DetectarPuertoActivo
operations
+run()
leerHora
operations
+run() : void
Thread
operations
+run() : void
+start() : void

Lista de acrónimos

IMU → *Inertial Measure Unit/Unidad de Medición Inercial.*

dps → *Degrees per second/Grados por segundo*

g → *Medida equivalenta a 1 veces la fuerza de la gravedad*

SDA → *Linea de datos*

SCL → *Señal de reloj*

GND → *Ground o toma de tierra*

SRAM → *Static Random Access Memory*

EEPROM → *Electrically Erasable Programmable Read-Only Memory/Memoria de solo lectura borrable electricamente*

PCB → *Printed Circuit Board/Placa de circuito Impreso*

GPL → *General Public License/Licencia General Pública*

LGPL → *Lesser General Public License/Licencia General Pública Reducida*

MIPS → *Microprocessor without Interlocked Pipeline Stages/Microprocesador sin bloqueos en la etapa de segmentación*

ANSI → *American National Standards Institute/Instituto Nacional Americano de Estándares*

RTC → *Real Time Clock/Reloj en Tiempo Real*

RF → *Requisitos Funcionales*

RNF → *Requisitos No Funcionales*

OTG → *USB On-The-Go*

ABS → *AntiBlockierSystem/Sistema Anti Bloqueo*

POM → *Project Object Model*

REST → *Representational State Transfer*

SOAP → *Simple Object Access Protocol*

HTTP → *Hypertext Transfer Protocol*

SWT → *Standard Widget Toolkit*

Glosario

MIPS Es una familia de procesadores que utilizan la arquitectura RISC

Bibliografía

[1] wikipedia, "Piloto automático," 2019. [Online]. Available: https://es.wikipedia.org/wiki/Piloto_autom%C3%A1tico

[2] Jaolpnik, "Primer imu," 2019. [Online]. Available: https://jalopnik.com/lawrence-sperry-inventor-of-the-autopilot-and-the-m-1592623110

[3] grupo alava, "Imu en barcos," 2019. [Online]. Available: https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19760016247.pdf

[4] H. the Moon, "Imu apollo," 2019. [Online]. Available: https://twitter.com/pretavoyager/status/1152620925508096000/photo/

[5] roboticsqut, "Apolo xi navegacion inercial," 2017. [Online]. Available: https://www.youtube.com/watch?time_continue=183&v=RE8HrHrQQWE

[6] wikipedia, "Imu apollo," 2019. [Online]. Available: https://en.wikipedia.org/wiki/Apollo_PGNCS

[7] kubestudio, "Foto sin/con estabilizador," 2019. [Online]. Available: https://images.app.goo.gl/q41nmQBujqBn4vJu6

[8] motorPasionMoto, "Imu en las motos," 2018. [Online]. Available: https://www.motorpasionmoto.com/tecnologia/plataforma-medicion-inercial-que-como-funciona-mayor-avance-seguridad-motos

[9] F. S.-T. D.-P. U. L. P. M. U. J. F. R. U. E. Peña González (UDC), J. Sande González-Cela (UDC), "Udc," 2013. [Online]. Available: https://www.udc.es/citeec/images/proyectos/puertos_y_costas/puertos_costas_10.pdf

[10] WIKIPEDIA, "Grados de libertad," 2019. [Online]. Available: https://en.wikipedia.org/wiki/Degrees_of_freedom_(mechanics)

[11] Adafruit, "10dof," 2019. [Online]. Available: https://www.adafruit.com/product/1604

[12] rs online, "11dof," 2019. [Online]. Available: https://es.rs-online.com/web/p/kits-de-desarrollo-de-sensores/8949310/

[13] Sparkfun, "Sparkfun 9dof sensor stick." [Online]. Available: https://learn.sparkfun.com/tutorials/9dof-sensor-stick-hookup-guide/all

[14] arduino.cl, "arduino." [Online]. Available: http://arduino.cl/que-es-arduino/

[15] aprendiendoarduino.wordpress, "Memoria arduino." [Online]. Available: https://aprendiendoarduino.wordpress.com/2017/06/21/memoria-flash-sram-y-eeprom-3/

[16] arduino, "Interfaz spi." [Online]. Available: https://www.arduino.cc/en/reference/SPI

[17] ——, "Interfaz i2c." [Online]. Available: https://www.arduino.cc/en/Reference/Wire

[18] ——, "Rx/tx." [Online]. Available: https://www.arduino.cc/reference/en/language/functions/communication/serial/

[19] arduino.cc, "Arduino mega." [Online]. Available: https://store.arduino.cc/mega-2560-r3

[20] ——, "Arduino uno." [Online]. Available: https://store.arduino.cc/arduino-uno-rev3

[21] ——, "Arduino ethernet." [Online]. Available: https://store.arduino.cc/arduino-ethernet-rev3-without-poe

[22] wikipedia, "Procesador avr." [Online]. Available: https://es.wikipedia.org/wiki/AVR

[23] rcmcomputointegrado, "arquitectura risc." [Online]. Available: http://rcmcomputointegrado.blogspot.com/2012/03/arquitectura-risc-y-cisc.html

[24] arduino.cc, "Arduino due." [Online]. Available: https://store.arduino.cc/due

[25] ——, "Intel galileo gen2." [Online]. Available: https://www.arduino.cc/en/ArduinoCertified/IntelGalileoGen2

[26] ——, "Arduino yun." [Online]. Available: https://store.arduino.cc/arduino-yun-rev-2

[27] wikipedia, "C++." [Online]. Available: https://es.wikipedia.org/wiki/C%2B%2B

[28] L. de programación.net, "Características c++." [Online]. Available: https://lenguajesdeprogramacion.net/cpp/

[29] M. M. M. y RAUL ROMERO MAÑUS, "Paradigma orientado a objetos. fundamentos y origen de java." [Online]. Available: http://dis.um.es/~lopezquesada/documentos/IES_1415/IAW/curso/UT3/ActividadesAlumnos/java7/paginas/pag1.html

[30] arduino, “Tipo int en arduino.” [Online]. Available: https://www.arduino.cc/reference/en/language/variables/data-types/int/

[31] tuprogramacion.com, “origen java.” [Online]. Available: http://www.tuprogramacion.com/programacion/historia-de-java/

[32] wikipedia, “origen java.” [Online]. Available: https://es.wikipedia.org/wiki/Java_(lenguaje_de_programaci%C3%B3n)#Historia

[33] J. D. Cero, “Carácteristicas.” [Online]. Available: https://javadesdecero.es/fundamentos/breve-historia-caracteristicas-y-aplicaciones/

[34] maximintegrated, “Rtc ds3231.” [Online]. Available: https://datasheets.maximintegrated.com/en/ds/DS3231.pdf