C1Tracker
Memoria final

Monitor técnico y tracker de actividad deportiva para piragüismo

Pablo Álvarez y Ángel Domínguez

17 de mayo de 2020

Resumen

En el presente documento se detalla el desarrollo e implementación de un sistema de monitorización deportiva para piragüistas. Este sistema estará formado por una unidad central, con acelerómetro, giroscopio, módulo GPS, transceptor de radio, transceptor Bluetooth y batería con carga inalámbrica; capaz de conectarse a una unidad secundaria inalámbrica con acelerómetro, giroscopio y transceptor de radio alimentada por un supercondensador.

Los datos recogidos serán enviados por Bluetooth a un smartphone Android, donde serán leídos por una aplicación de diseño propio. En ella, se les aplicará técnicas avanzadas de procesado de señal tales como eventanado, filtrado o Transformada Discreta de Fourier, con el fin de facilitar la interpretación de los mismos. Se realizará también un cálculo de estadísticos matemáticos que permitan al deportista conocer parámetros genéricos del transcurso del entrenamiento. Además, se incluye un sistema de Machine Learning (algoritmo K-means) capaz de identificar la intensidad de entrenamiento en cada instante. Estos resultados se mostrarán por pantalla en tiempo real, tanto de forma numérica como mediante una representación gráfica dinámica de los mismos, permitiendo informar al deportista de la calidad del entrenamiento en directo.

Finalmente, tanto los estadísticos calculados como los datos en bruto serán guardados en la memoria interna del dispositivo, de cara a un análisis *a posteriori* de la sesión. Los datos más relevantes se enviarán a un servidor Thingspeak a través de un canal MQTT, posibilitando de este modo el seguimiento de la evolución del deportista a lo largo de la temporada.

Índice

Índice de figuras

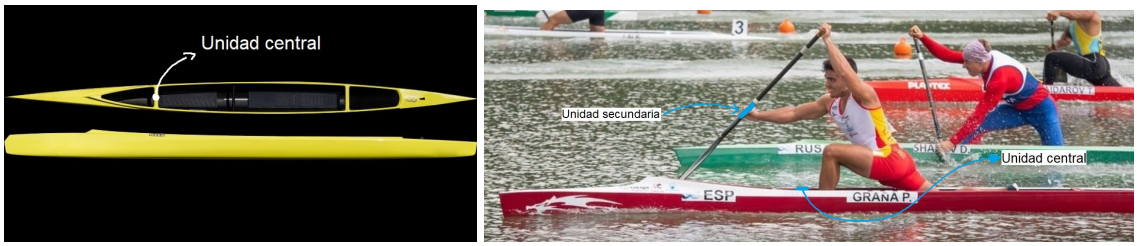
1. Introducción

En la actualidad existen en el mercado multitud de productos orientados a la monitorización de la actividad tanto deportiva como cotidiana de las personas. Desde pulseras o relojes hasta accesorios para los smartphones, todos tienen un elemento en común: tratan de recolectar determinar datos acerca del movimiento de las personas para posteriormente procesarlos e interpretarlos, devolviendo al usuario información útil que éste es capaz de interpretar.

Desde nuestro equipo identificamos que, si bien la oferta de este tipo de dispositivos en bastante amplia, todos ellos están orientados, bien a actividades deportivas no profesionales o bien, a deportes mayoritarios. Dada la ausencia de instrumentos de tracking para deportes pequeños (como el piragüismo en nuestro caso), decidimos apostar por el desarollo de uno, cuya etapa de procesado e interpretación de datos sea específico para el tipo de movimientos característicos de este deporte.

El hardware de nuestro sistema se divide en dos dispositivos. Por un lado, tenemos una unidad central, que irá situada fija en la embarcación. Esta unidad se basará en un microcontrolador Atmega328p, conectado a una Unidad de Medición Inercial (IMU en adelante), un receptor GPS, un receptor de radio para conectarse con la unidad secundaria y un transmisor de Bluetooth. Incluye batería de 1200mAh con opción de carga inalámbrica. Por otro lado, tenemos la unidad secundaria, basada en el mismo microcontrolador conectado a una IMU y un transmisor de radio. Esta segunda unidad se sitúa en una extremidad del deportista o en la pala, y permitirá recolectar datos no sólo del ritmo de entrenamiento sino de la técnica de paleo del atleta.

En lo referido al software, éste gira entorno a una aplicación para dispositivos Android. Ésta se conecta a la unidad central para recibir por Bluetooth los datos recolectados por los distintos sensores. Posteriormente, se aplican técnicas avanzadas de procesado de señal para facilitar el manejo e interpretación de los datos y realizar el cálculo de estadísticos. Estos datos se representan en tiempo real en la interfaz de usuario, tanto mediante numeración como mediante graficación en el dominio del tiempo. También son almacenados en formato CVS en un fichero dentro de la memoria interna del smartphone de cara a su posterior análisis. Finalmente, se incluye la opción de enviar los estadísticos más importantes de la serie o sesión a un servidor de Thingspeak (canal MQTT). Estos datos podrán ser procesados a posteriori y así obtener la progresión del deportista a lo largo de la temporada.



(a) Canoa (b) Deportista

Figura 1: Uso del proyecto

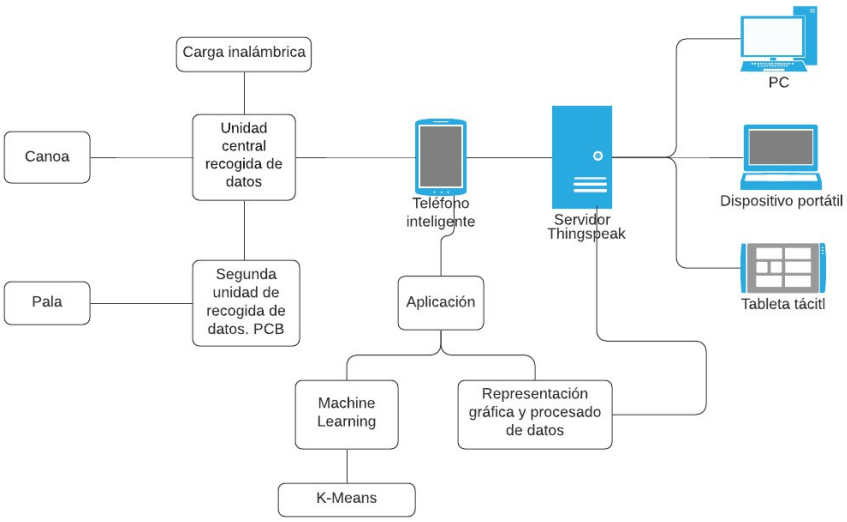


Figura 2: Sistema completo

2. Especificaciones del sistema

Unidad central

- Tensión de alimentación: 3V-4,2V
- Sistema basado en el microcontrolador Atmega328p.
- IMU de 16bits LSM9DS1.
- Unidad GPS Ublox M8N compatible con GPS, Galileo, Glonass y Beidou.
- Transceptor inalámbrico NRF24L01 (2.4GHz, consumo ultra bajo de 13mA en rx, hasta 2Mbps).
- Transceptor Bluetooth HC-06 con alcance de hasta 20m
- Batería de 1200mAh, capaz de ofrecer 6h de funcionamiento con GPS a máxima frecuencia.
- Carga inalámbrica, compatible con el estándar Qi, el más usado en la actualidad.

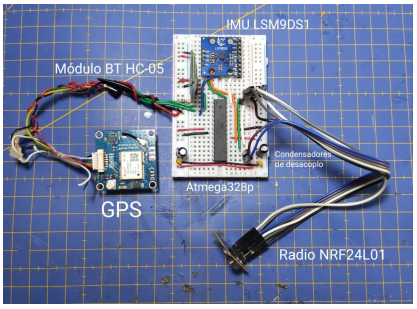
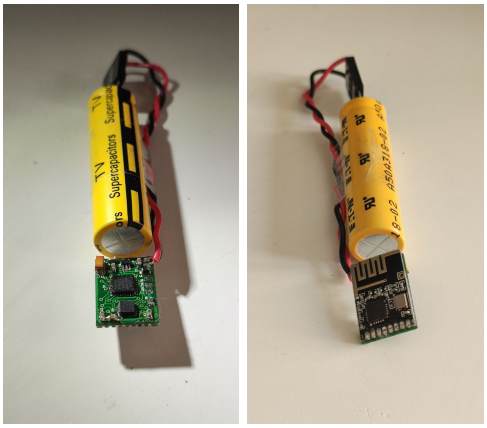


Figura 3: Unidad central

Unidad secundaria

- Tensión de alimentación: 0,8v- 3,5V
- Sistema basado en el microcontrolador Atmega328p.
- IMU de 16bits y consumo ultra bajo MPU9250.
- Regulador de tensión TLV61225 capaz de ofrecer 3,3V con tensiones de entrada de sólo 0,8V.
- Transceptor inalámbrico NRF24L01 (2.4GHz, consumo ultra bajo de 11,3mA en tx, hasta 2Mbps).
- Consumo ultra bajo (9mA durante el funcionamiento, ¡1mA en standby).
- Batería basada en supercondensador de 34F, capaz de ofrecer 2h de autonomía con tiempos de carga inferiores de unos pocos segundos.
- Tamaño de PCB de tan sólo 18x12mm.



(a) Parte superior (b) Parte inferior

Figura 4: Unidad secundaria

Aplicación Android

- Compatible con dispositivos Android 4.2 y superiores.
- Sólo se requiere de permisos de conectividad y escritura de ficheros.
- Graficación y representación de los datos obtenidos en tiempo real.
- Implementación de técnicas avanzadas de procesado de señal (eventanado, filtrado, DFT, etc.).
- Sistema de identificación y reconocimiento de características de paleo mediante machine learning (algoritmo k-means).
- Almacenado de resultados en archivo CSV para su posterior análisis.
- Conectividad a canales MQTT de Thingspeak, con el fin de visualizar los datos a posteriori y analizar el rendimiento y proyección del deportista en largos periodos de tiempo.
- Interacción para resetear los esadísticos acumulados de la sesión o guardarlos de forma definitiva en la nube Thingspeak, ofrecien mayor flexibilidad a la hora de elegir qué datos salvar.

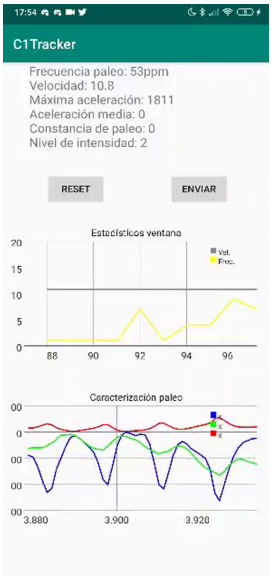


Figura 5: Aplicación Android

3. Funcionamiento

3.1. Hardware: Unidad central de recogida de datos

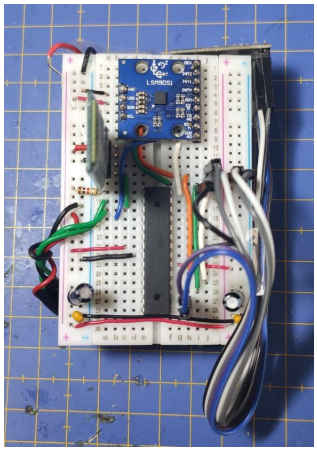


Figura 6: Forma final de la unidad central

La unidad central gira entorno a un microcontrolador **Atmega328p**. Este micro usa la tecnología AVR y cuenta con 28 pines, de los cuales 23 pueden ser destinados a E/S. Dado que esta unidad estará alimentada gracias a una batería, uno de los mayores retos es **optimizar el consumo al máximo** para obtener la mayor autonomía posible. Es por ello que el primer paso que se llevó a cabo antes de comenzar a programar el micro fue el flasheo de un nuevo cargador de arranque y la modificación de los fusibles (fuses) del mismo.

Por un lado, con el fin de simplificar esta etapa del diseño, sería conveniente ser capaces de programar el firmware en código Arduino. Sin embargo, el cargador de arranque de Arduino no está completamente optimizado, por lo que decidimos optar por una **alternativa compatible: Opti-**

boot. Flasheamos este bootloader conectando el bus SPI del Atmega328p junto con el pin de Reset a un programador ISP. Además, aprovechamos esta conexión para programar los fuses (registros propios del micro donde se almacena información crítica sobre el funcionamiento y arranque de este) según nuestras necesidades. Mediante la programación del LowFuse conseguimos configurar como fuente de reloj el oscilador interno del microcontrolador. Si bien su precisión es muy inferior a la de un cristal externo, ésta demostró ser suficientemente buena para el correcto funcionamiento de la comunicación serial, por lo que su bajo consumo lo convierte en la opción ideal para nuestro proyecto. Además, modificamos el ExtendedFuse, encargado de la Brown-out detection. La Brown-out detection es un mecanismo de protección encargado de apagar el microcontrolador en caso de que la tensión de entrada disminuya por debajo de cierto nivel. Como queremos que nuestro sistema trabaje en torno a los 3,3V, modificamos este registro para configurar un umbral inferior de 2,7V. Por último, cabe mencionar que, una vez programado el cargador de arranque OptiBoot, es posible enviar el código al microcontrolador mediante su interfaz UART, por lo que, haciendo uso de un adaptador FTDI-USB, ya no es necesario el programador ISP.

Uno de los sensores encargados de la **recolección de datos** es la Unidad de Medición Inercial **LSM9DS1**. Esta unidad cuenta con acelerómetro (3 ejes), giróscopo (3 ejes) y magnetómetro (3 ejes). En nuestro caso no haremos uso de este último, ya que no aporta información útil del entrenamiento y supone un gasto de consumo añadido. La interfaz utilizada para comunicarse con el Atmega es SPI (Serial Peripherial Interface). El chip ofrece también la posibilidad de conexión mediante I2C, pero nos decantamos por SPI de nuevo por temas de consumo.

Para configurar y leer los datos de la IMU utilizaremos una librería externa, ya que el hecho de que tenga definidos macros para cada registro facilita mucho el proceso y ahorra tiempo buscando información en el datasheet. A la hora de inicializar la IMU, configuraremos los registros pertinentes para seleccionar una frecuencia de muestreo de 14,9Hz y resolución baja. Para leer los datos, se utilizará una llamada a las funciones readAccel y readGyro de la librería, las cuales devuelven mediante paso por puntero los valores de los registros OUT_X_L_XL, ..,. OUT_Z_L_XL y OUT_X_L_G, ..., OUT_Z_L_G respectivamente.

Para trabajar con estos datos desde nuestro código se crea un objeto imu con variables ax, ay, az, gx, gy y gz.

Respecto al **GPS**, este se trata de una unidad basada en el chip M8N de la marca Ublox. Cuenta con frecuencia de actualización de hasta 1Hz, y se comunicará con el atmega mediante un puerto serie UART. El GPS envía mediante esta interfaz los distintos comandos NMEA y UBX con los datos recibidos (tiempo y localización). Dado que el procesado de estos comandos escapa a las competencias de esta asignatura, usamos la librería NeoGPS, capaz de calcular desplazamiento, velocidad, coordenadas y tiempo transcurrido a partir de ellos.

Para **recibir la información recolectada en la unidad secundaria** es necesario un transceptor de radio. En nuestro caso, optamos por módulos **NRF24L01**, ya que son altamente configurables, ofrecen tasas de transmisión elevadas a baja latencia, tienen buen alcance y ofrecen un consumo significativamente bajo. De nuevo, no entramos en detalle en el funcionamiento interno de estos módulos (empaquetado, corrección de errores, generación de ACK, etc.), sino que optamos por el uso de una librería externa.

El NRF estará conectado al Atmega por el mismo puerto SPI que la IMU salvo por el pin Chip Select, el cual, al estar conectado a otra salida digital, permitirá al microcontrolador elegir qué periférico utilizar en cada momento y evitar conflictos entre ambos. Este chip cuenta con modos dormido y apagado. Además, ofrece distintas potencias de transmisión a la hora de emitir información. De nuevo, con el fin de aumentar la autonomía del sistema global, el NRF se encontrará apagado la mayor parte del tiempo, encendiéndose únicamente cuando, por temporización, sea necesario conectarse a la unidad externa para recibir datos.

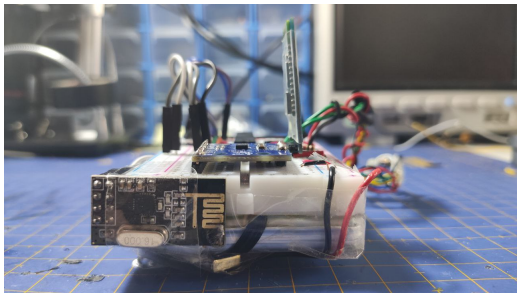


Figura 7: Módulo transceptor de RF con su antena

La conexión con el dispositivo Android se realizará mediante Bluetooth, para lo que es necesario el uso de un módulo externo. En nuestro caso utilizaremos un **HC-06**. Este módulo es extremadamente sencillo de utilizar, ya que sólo necesita conectarse a un puerto UART, enviando los datos que le llegan por pin RX y mandando por el pin TX los que recibe de la aplicación.

Una de las mayores limitaciones del microcontrolador con el que trabajamos es la existencia de un único puerto UART. En nuestro caso, esto supone un problema, ya que contamos con dos dispositivos que requieren hacer uso de él. Tras probar distintas estrategias que se detallarán en las próximas secciones, optamos por compartir el puerto, conectando el pin de TX del GPS al RX del micro y el RX del HC-06 al TX de éste.

Respecto a la alimentación del sistema, ésta se consigue gracias a una batería de polímero de Litio (Lipo) de una sola celda de tensión nominal 3,7V. Ya que tanto el Atmega328p como la IMU trabajan sin problema en este rango de tensiones y el resto de periféricos cuenta con sus propios reguladores de tensióon a 3,3V, el sistema funciona sin problemas.

A la hora de **cargar** la batería es necesario tener en cuenta que no basta con enchufar a una fuente de alimentación, sino que es necesario el uso de un circuito regulador de carga. Para ello, utilizamos un módulo basado en el **TP4056**, capaz de regular la corriente de carga y limitándola a 1A (suponemos que la batería es de al menos 1C aunque no tensamos sus especificaciones). Además, decidimos incluir la opción de **carga inalámbrica** para no sólo darle un aire de modernidad al proyecto sino añadir una funcionalidad bastante interesante de cara a encapsular la unidad central en una carcasa en el futuro. Para ello, se utiliza una antena de espira conectada a un circuito rectificador, que convierte la señal de AC recibida en una tensión continua. Esta señal se filtra, se convierte a 5V y se conecta a la entrada del TP4056.

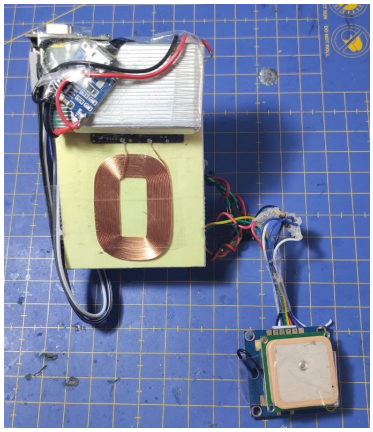


Figura 8: Sistema de carga inalámbrica, regulador de carga y batería

3.2. Hardware: Unidad secundaria de recogida de datos

El funcionamiento de esta segunda unidad es muy similar al de la unidad central, con la principal diferencia de que, en este caso, el único sensor disponible será una IMU. El microcontrolador elegido es de nuevo un Atmega328p, flasheado con Optiboot y funcionando con el reloj interno (al igual que en el caso de la unidad central). Este micro está conectado a una unidad de medición inercial. Ésta se trata de un modelo MPU9250, unidad de medición inercial con 9 grados de libertad caracterizada por su bajo consumo y alta precisión. De nuevo, se conecta por SPI al micro, y hacemos uso de una librería con todos los macros definidos para interactuar con sus registros. Se incluye también un transceptor NRF24L01 para comunicarse con la unidad central.

Si bien desde el principio tuvimos claro que la implementación de esta segunda unidad recolectora de datos era una mejora opcional, nos propusimos como reto el diseño de un dispositivo un tanto más profesional que la unidad central. Ya que ésta sólo necesita de dos periféricos y microcontrolador, surgió la idea de **diseñar una placa de circuito impreso** en la que incorporar todos los componentes. Ya teníamos experiencia en el diseño de este tipo de placas por la asignatura Circuitos Electrónicos (la realizamos como mejora voluntaria), por lo que decidimos intentar mejorar lo realizado en esta otra asignatura. Así, optamos por el uso de componentes SMD en lugar de THT, utilizar un diseño de doble capa, encargar la fabricación de ésta a una empresa especializada en China en lugar de quemar las pistas manualmente con ácido y ajustar el tamaño lo máximo posible. Fue así como terminamos con una **placa** de tan sólo 18x12mm, con circuitos integrados QFN y componentes pasivos de los tamaños 0805 y 0603. Fue soldada a mano por complicaciones con el soldador de aire caliente.

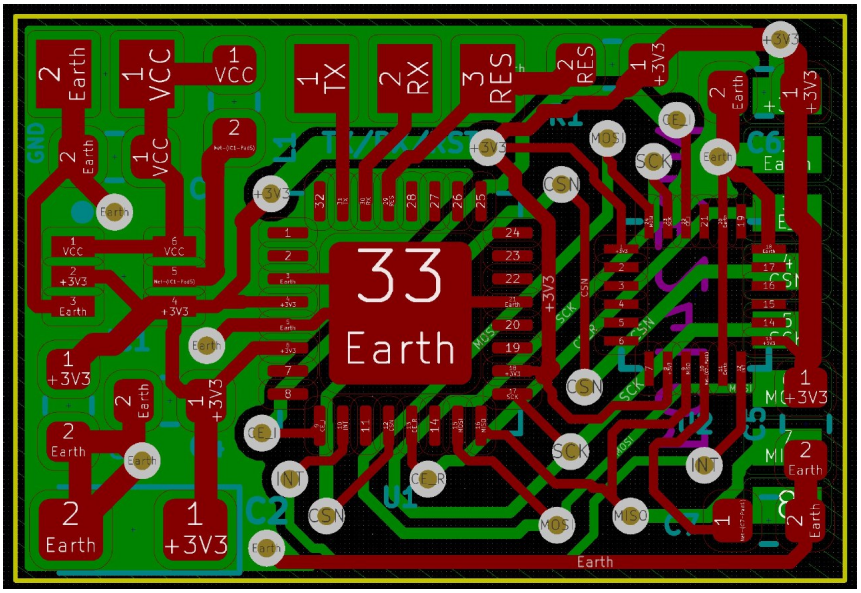


Figura 9: Diseño de PCB en KiCAD

Con respecto a la alimentación, dado el bajo consumo de este dispositivo (algo inferior a 9mA de media), decidimos evitar el uso de baterías tradicionales y probar opciones más innovadoras. Actualmente, la placa se **alimenta** gracias a un supercondensador de 34F con una tensión nominal de 3V. Todos los deportistas nos vimos alguna vez en la situación de ir entrenar y en el momento antes de comenzar darnos cuenta de que no podemos registrar nuestra actividad al no tener batería suficiente en el tracker. Los supercondensadores ofrecen una solución práctica y económica a este problema, ya que, con tiempos de carga muy reducidos (inferiores al minuto) pueden cargarse por completo y ofrecer suficiente autonomía para funcionar durante un entrenamiento entero. En nuestro caso, y según las pruebas realizadas, esta unidad secundaria de recolección de datos tiene una autonomía superior a las

dos horas y cuarto.

Según se reduce la carga almacenada en el condensador, se reducirá la tensión que este ofrece en sus bornas. Para compensar este efecto, y como todos los circuitos integrados requieren de una tensión de alimentación de entorno a 3,3V, se introduce un convertidor step-up **TLV61225**. Este chip recibe a su entrada una tensión continua de entre 0,7V y 3,3V y pone a la salida 3,3V constantes. Es capaz de ofrecer eficiencias cercanas al 94 por ciento. Dado que esta tensión de alimentación puede contener ruido, se incluyen condensadores de desacoplo situados lo más cerca posible de los pines de alimentación de los respectivos circuitos integrados.

Tanto el proyecto de Kicad con el esquemático, PCB y lista de materiales como los Gerbers y el código utilizado para la programación del microcontrolador se pueden encontrar en el repositorio de **GitHub**[**1**].

3.3. Firmware

El firmware que instalamos en ambas unidades es bastante similar; si bien la unidad central cuenta con un mayor número de sensores, ambas siguen una misma estructura.

En primer lugar se definen aquellas variables globales o de clase necesaria. A continuación, se inicializan en el método setup. Es aquí donde, además, se configuran las conexiones SPI o UART con los distintos dispositivos. Por último, se configura una interrupción por **WatchDog Timer**. Para ello, se ponen a uno los registros WDCE (activación del cambio del WD), WDE (prescaler del reloj del sistema) y WDTCSR (activacion de la interrupción por WD). Además, se escribe en el registro WDP el prescaler correspondiente en función de cuándo se quiere que salte la interrupción. Una vez completadas las correspondientes inicializaciones, se entra en el método loop.

Una vez se entra en el método loop, ocurren los siguiente sucesos por orden: se despiertan los sensores dormidos, se leen todos los datos necesarios, se vuelven a dormir los sensores, se envía por puerto serie los datos obtenidos y se duerme el microcontrolador. Una vez el micro entra en modo dormido, el método loop deja de ejecutarse.

Recordemos que nuestro sistema funciona a una frecuencia de trabajo de 10Hz. Para controlar la temporización, se hace uso de **interrupciones periódicas** mediante WatchDog Timer. El watchdog es una función del microcontrolador que, si está activada, genera una interrupción cuando transcurre un determinado tiempo sin que éste trabaje. Se puede utilizar para rescatar al micro en caso de fallos en la ejecución del programa o, como es el caso, para generar interrupciones periódicamente. Optamos por utilizar WD en lugar de interrupciones periódicas por timer ya que, con el fin de reducir el consumo global del sistema, se dormirá al microcontrolador entre mediciones y, en modo dormido, el reloj principal se detiene y los timers dejan de funcionar.

Se crea una variable volátil a modo de flag. Cuando salte la interrupción periódica, este flag se pondrá a 1. Una vez se ejecute el código de lectura y envío de datos, se volverá a poner a 0. Esto evita que en caso de errores se ejecute el código a destiempo.

Con respecto al modo dormido, el Atmega328p cuenta con distintos modos de bajo consumo. En nuestro caso utilizamos el modo **Power Save**; si bien existe un modo Power Down de consumo aún menor, el tiempo necesario para despertar al micro tras estar en él es demasiado elevado, lo que no nos permitiría cumplir con las especificaciones de frecuencia de muestreo previstas. Cuando se activa el modo Power Save, el microcontrolador deja de ejecutar el código. Una vez salta la interrupción por WDT, se atiende su ISR y, acto seguido, se vuelve a ejecutar el método lopp desde el comienzo.

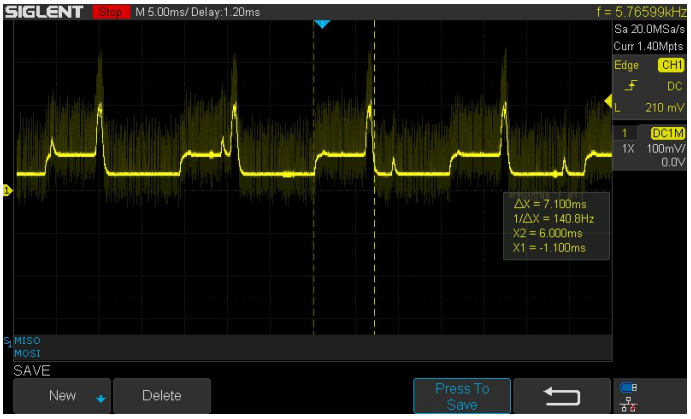


Figura 10: Análisis en osciloscopio del consumo de la unidad secundaria. Se pueden observar mínimos cuando el microcontrolador se encuentra dormido, niveles medios cuando se activa y picos en la transmisión de datos.

Se decidió que no sería necesario utilizar una máquina de estados finita, ya que al despertar al micro éste vuelve a ejecutar el loop desde el principio. Además, al no tener condiciones if en el código, siempre ocurriría la misma transición entre estados de forma periódica, por lo que carece de sentido su implementación.

3.4. Procesado de datos

Para el procesado de datos principalmente hemos utilizado el software de **Matlab**. Para ello, en primer lugar hemos pasado los datos extraídos de un entrenamiento en formato CSV al programa. Una vez extraído estos nos hemos centrado principalmente en la coordenada y de la aceleración y la coordenada x del giroscopio que recoge la unidad principal. El motivo de esto es que la barca está orientada en la dirección y por lo que consideramos esta como mas significativa respecto las otras para la aceleración. Además, la coordenada x del giroscopio corresponde con los cabeceos de la barca lo cual es muy interesante si lo juntamos con la frecuencia de paleo. En primer lugar, hemos diseñado un **filtro IIR** para poder extraer e interpretar mejor los datos recogidos.

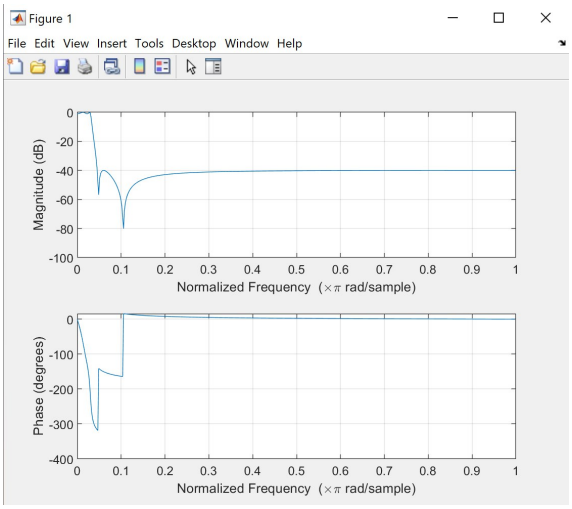


Figura 11: Filtro IIR paso bajo de orden 4

A continuación, hemos representado la **señal filtrada**:

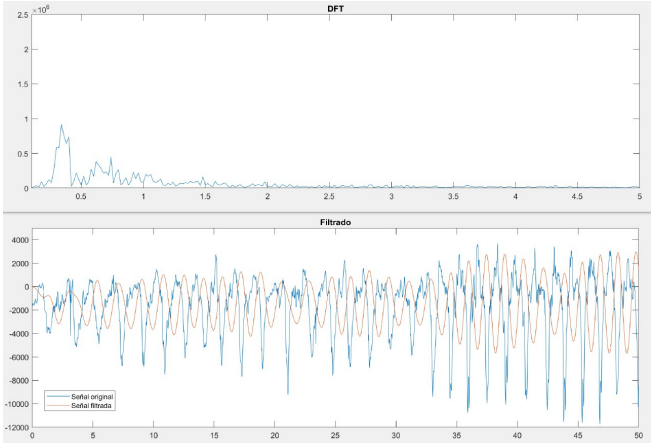


Figura 12: Señal filtrada

Una vez conseguido tanto la señal filtrada como su respectiva DFT hemos enventanado la señal, haciendo que la interpretación de nuestros datos sea más realista. Para ello, hemos diseñado **ventanas** de 400 muestras con un solapamiento de 200 muestras de la ventana anterior. Para cada una de las ventanas hemos realizado la extracción de:

- **Número de paladas:** para ellos hemos utilizado la función 'peaks' de Matlab, cada vez que la señal fitrada corresponda con un máximo esto corresponderá con una nueva palada.
- **Máxima frecuencia de la DFT:** coincide también con el número de paleos.
- **Ancho de banda de la máxima frecuencia de la DFT:** para estudiar la constancia del deportista.
- **Máxima aceleración.**
- **Mínima aceleración.**
- **Aceleración media.**
- **Máxima coordenada del eje y del giroscopio: cabeceo de la embarcación.**

Una vez extraídas las features o características hemos aplicado un algoritmo de **Machine Learning**. Este es un algoritmo no supervisado de clustering denominado **K-Means**. La idea de implementar dicho algoritmo es buscar distintas etapas del entrenamiento del deportista que corresponderán con los centroides del algoritmo. Para poder aplicar dicho algoritmo hemos elegido las features más importantes para nuestra tarea, aceleración máxima, número de paladas y cabeceo de la embarcación.

A la hora de elegir el número de etapas o centroides tenemos que tener en cuenta que una mala elección de estos puede realizar grupos de datos muy heterogéneos, es decir con pocos centroides o agrupaciones de datos muy diferentes, muchos centroides. Finalmente el número de centroides óptimos para nuestro caso son **tres** para la explicación de dicho proceso de selección váyase al apartado 4 (Desarrollo del proyecto). Una vez elegidas las features y el número de centroides hemos aplicado dicho algoritmo en Matlab a nuestros datos, consiguiendo los siguientes resultados:

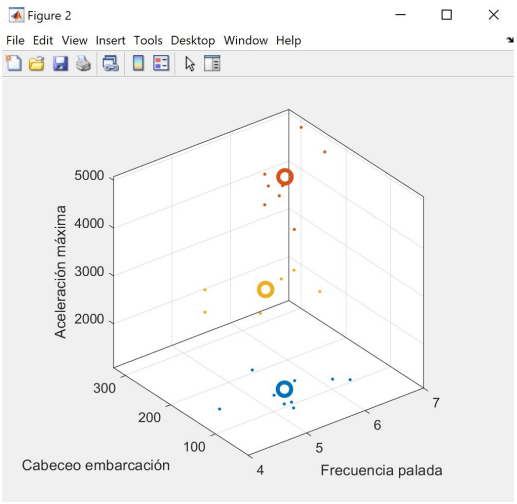


Figura 13: K-Means

(a) Aceleración máxima - Frecuencia paleo (b) Aceleración máxima - Cabeceo (c) Cabeceo - Frecuencia paleo

Figura 14: K-Means parámetros

A partir de las figuras podemos concluir **tres etapas bien diferenciadas del entrenamiento**. Alto rendimiento, medio rendimiento y bajo rendimiento. Cabe mencionar, que el estudio del procesado de datos se realizo con un frecuencia de muestreo de 50 Hz y un enventanado de 400 muestras con un solapamiento de 200 muestras ya que utilizamos unos datos recogidos de un entrenamiento antes de esta situación. En la aplicación de Android, finalmente se decidió cambiar dichos parámetros por una frecuencia de muestreo de 10 Hz y un enventanado de 80 muestras con solapamiento de 40 como se comenta en el siguiente apartado.

3.5. Aplicación Android

La aplicación Android con la que se debe acompañar el hardware para su correcto funcionamiento fue programada en Android Studio. Si bien tras la incorporación de las numerosas funciones que ofrece resultó ser una aplicación **bastante compleja**, su principio de funcionamiento es relativamente sencillo.

Toda la aplicación gira en torno a la comunicación Bluetooth, ya que depende de ésta para la

adquisición de datos y su posterior procesado y representación. Esta comunicación se consigue gracias a una **hebra** independiente de la actividad correspondiente a la interfaz de usuario. En ella, se hace uso del módulo Bluetooth del smartphone para la recepción de datos. Una vez se han recibido, se transmiten a la actividad principal mediante el uso de un **socket**. De este modo, los datos se pueden obtener en segundo plano sin afectar al rendimiento de la aplicación ni a la interfaz, y gracias al socket podemos trabajar con ellos en directo.

Una vez se han recibido los datos, se comprueba su validez. Para ello, aprovechando que están en formato CSV, se guarda cada uno de ellos en la posición de un array, se comprueba que efectivamente sean valores numéricos y se cuenta que el número de datos corresponda con la cantidad de información que se debería haber recibido.

—

El siguiente paso es el **cambio de sistema de referencia**, para asegurar que los resultados finales obtenidos tras el procesado son consistentes independientemente de la posición en que se encuentre colocado el dispositivo en la embarcación. Para ello, se filtra con un **filtro paso bajo elíptico de orden 4** de frecuencia de corte 0.1Hz la señal, a fin de obtener la componente constante. Este valor deberá corresponderse con la gravedad, ya que es extremadamente complicado para una canoa mantener una aceleración constante durante un tiempo prolongado, y el campo gravitatorio es la única otra causa de una aceleración de este tipo posible. Denotaremos, de ahora en adelante, $\vec{g}$ a la aceleración producida por el campo gravitatorio terrestre y

$$\vec{a_i}$$

a la i-ésima muestra de aceleración tomada ya filtrada por el filtro de frecuencia de paleo (sección anterior).

$$\vec{g} = \frac{1}{80}\sum_{i=0}^{79}\vec{a_i} \qquad (g_x, g_y, g_z) = \frac{1}{80}\sum_{i=0}^{79}(a_{xi} + a_{yi} + a_{zi}) \tag{1}$$

Una vez tenemos el valor de la aceleración gravitatoria según el sistema de referencia de la unidad central (recordemos que ésta no tiene por qué estar situada de forma plana y vertical sobre la embarcación, sino que puede colocarse en múltiples posiciones), podemos realizar el cambio de sistema de referencia a uno tal que el nuevo eje z coincida con la gravedad:

$$\vec{p_i} = \frac{\vec{a_i}\cdot\vec{g}}{||\vec{g}||^2}\vec{g} \qquad (p_{xi}, p_{yi}, p_{zi}) = \frac{(a_{xi}\cdot g_x + a_{yi}\cdot g_y + a_{zi}\cdot g_z)}{g_x^2 + g_y^2 + g_z^2}(g_x, g_y, g_z) \tag{2}$$

Donde $\vec{p_i}$ es la proyección ortogonal de la i-ésima medida de la aceleración sobre la gravedad.

Por otro lado, la componente de la **aceleración ortogonal** a la gravedad $\vec{n_i}$ vendrá dada por:

$$\vec{n_i} = \vec{a_i} - \vec{p_i} \qquad (n_{xi}, n_{xi}, n_{xi}) = (a_{xi} - p_{xi}, a_{yi} - p_{yi}, a_{zi} - p_{zi}) \tag{3}$$

De forma simultánea a este proceso, es necesaria la realización de un **eventanado** de los datos. Consideramos ventanas de 80 muestras, con un solapamiento de 40, para así ofrecer un periodo de actualización de 4s. Añadimos los valores que acabamos de recibir al final de la ventana y, en caso de que ésta se llene, se guardan las últimas 40 muestras para el solapamiento y se eliminan las anteriores. Cuando una ventana se llena, se procesan sus datos. En primer lugar, es necesario realizar el

Una vez caracterizados y calculados los 80 vectores $\vec{n_i}$, tenemos información real sobre la aceleración en la dirección de desplazamiento de la embarcación (n_{yi}) y sobre el cabeceo de ésta con las paladas (n_{zi}). El cambio de sistema de referencia efectuado no sólo nos permite obtener resultados fiables

independientemente de cómo se coloque el dispositivo en la embarcación, sino que, además, nos ayudará a evitar posibles fallos en las mediciones debidos a olas, cabeceo de la embarcación en ciabogas amplias u otros fenómenos que modifiquen la inclinación de la embarcación.

A continuación, debemos **filtrar los datos**. Esto se debe a que tanto el acelerómetro como el giroscopio pueden generar ruido de alta frecuencia (picos inusuales en el dominio del tiempo) que no sólo no aporta información sino que estropearía el cálculo de estadísticos. Para ello, se realiza una implementación del filtro FIR comentado anteriormente. Contamos con dos arrays, llamados *rax* y *fil*, de longitud igual al orden del filtro. Ambos actúan como registros de desplazamiento, de forma que al llegar una nueva muestra sin filtrar o filtrada respectivamente se desplazan las anteriores y se descarta la que más tiempo lleva. Para conseguir la nueva muestra filtrada, se realiza la siguiente operación:

$$\mathrm{Fil}[i] = \sum_{k=0}^{4} b_k \mathrm{Raw}[k] - a_k \mathrm{Fil}[k] \tag{4}$$

Ya disponemos de una señal filtrada en un sistema de referencia conocido, por lo que podemos comenzar a extraer información útil de ella. Por un lado, obtendremos ciertos estadísticos básicos de la ventana, como la aceleración máxima, mínima y media. Sin embargo, para conseguir los datos de mayor interés para el deportista durante un entrenamiento, serán necesarios cálculos más complejos.

Uno de los datos de mayor interés para el palista mientras entrena es su frecuencia de paleo. Para obtenerla a partir de la información de la que disponemos, podemos optar por dos vías: análisis en el dominio del tiempo y análisis en el dominio de la frecuencia.

En el caso del **análisis en el dominio del tiempo**, basta programar un comparador con histéresis, de tal forma que, mediante la elección de dos umbrales (uno positivo y otro negativo), se pueda detectar cuándo se produce una palada. Contando el número de paladas por ventana y aplicando una relación de proporcionalidad, podemos saber la frecuencia de paleo en paladas por minuto.

Para analizar el ritmo de paleo en el **dominio de la frecuencia**, basta, como se comentó anteriormente, encontrar el máximo de la DFT. Por desgracia Android no incorpora funciones dedicadas al proceso de la señal, por lo que será necesario programar una función **DFT a mano**. Esta función toma como entrada el array unidimensional de datos de una componente durante una ventana y devuelve un array bidimensional de la misma longitud en una de sus dimensiones y con la parte real e imaginaria por separado:

$$\mathrm{DFT}[0][i] = \sum_{k=0}^{79} n[k] \cos\left(-\frac{2\pi}{80} ki\right) \tag{5}$$

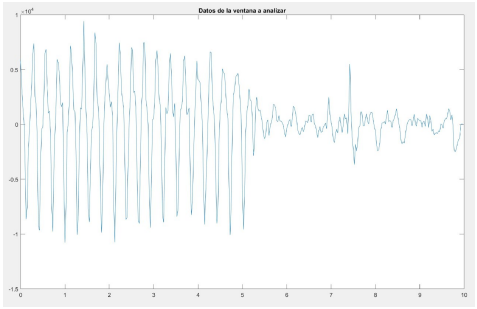
$$\mathrm{DFT}[1][i] = \sum_{k=0}^{79} n[k] \sin\left(-\frac{2\pi}{80} ki\right) \tag{6}$$

Para calcular el máximo de la DFT, primero deberemos hallar el módulo de cada coeficiente, y así obtener el espectro en amplitud de la señal:

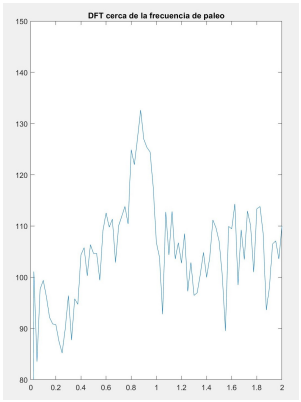
$$\mathrm{DFTamplitud}[i] = \sqrt{\mathrm{DFT}[0][i]^2 + \mathrm{DFT}[1][i]^2} \tag{7}$$

Finalmente, otro parámetro de bastante interés que podemos obtener gracias a la DFT es la constancia de paleo. Como se puede apreciar en la figura siguiente, existe un lóbulo principal en torno a la frecuencia de paleo de amplitud muy superior al resto del espectro. Como ya se mencionó, la anchura de este lóbulo se puede asociar a la constancia en el ritmo de paleo. Para su obtención, normalizaremos

la DFT dividiendo cada coeficiente entre el máximo, y buscaremos mediante un bucle que recorra cada valor los puntos donde la gráfica corta al valor 0,5. La distancia entre estos puntos se corresponderá con el ancho de banda buscado de la señal.



(a) Ventana en el dominio del tiempo sin filtrar



(b) DFT de la ventana en torno a la frecuencia de paleo

Figura 15: Como se puede apreciar, no es necesario que el paleo ocupe la ventana por completo para identificar el ritmo medio de paleo mediante DFT

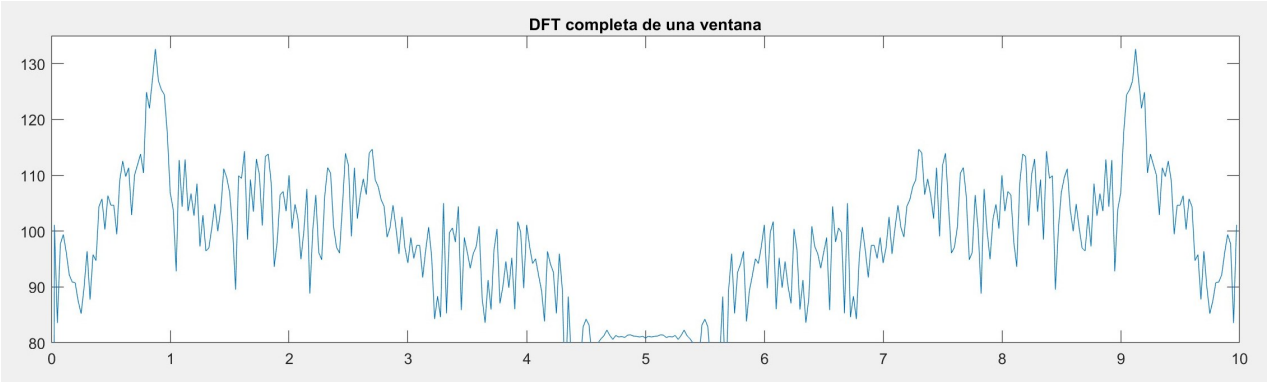


Figura 16: DFT completa de la ventana correspondiente a la Figura 13

Una vez se completa el cálculo de estadísticos, se procede a la **visualización y guardado de éstos**. Por un lado, la pantalla principal de la aplicación cuenta con dos gráficas que se actualizan con los datos recogidos **en tiempo real**. La primera muestra la aceleración recogida en la unidad secundaria (permite al deportista controlar su técnica en directo) y, la segunda, la progresión del ritmo de paladas y velocidad a lo largo del entrenamiento (ayuda al palista a mantener un ritmo constante rendir más en el entrenamiento). Además, se muestran otros parámetros de forma numérica como velocidad instantánea, distancia recorrida total, ritmo de paleo instantáneo o constancia del movimiento.

Todos los estadísticos calculados, junto con los datos en bruto y sin procesar, son almacenados en un fichero de texto en el almacenamiento interno del dispositivo en formato CSV. Esto permite volver a procesarlos o graficarlos una vez terminado el entrenamiento y llevar a cabo un análisis más detallado de la técnica del deportista.

Además, la aplicación cuenta con **dos botones**: comenzar serie y guardar datos. Una vez se pulsa comenzar serie, ciertas variables encargadas del cálculo de estadísticos (máximo, mínimo, media y desviación de distintas componentes) se resetearán a 0. Una vez se finalice la serie, pulsando el botón guardar datos se enviarán los resultados al **servidor de ThingSpeak**. Para ello, en la interfaz de usuario se introducen dos métodos onClick(). Uno se encarga de resetear las variables a 0. El otro, cuando se activa, se realiza una llamada a una AsyncTask (no se pueden ejecutar tareas de red en la misma hebra que la interfaz de usuario en Android) para la realización de una petición GET por HTTP al servidor MQTT de ThingSpeak. Esta petición incluye la clave de la API y el nombre de los canales a los que se deben enviar los nuevos datos.

3.6. Servidor Thingspeak

Como se comentó en el apartado anterior, hemos implementado en nuestro proyecto el uso de un servidor que recibe datos en **tiempo real** de la aplicación de Android.

Para ello, hemos utilizado la plataforma de IOT de Thingspeak, conocida por ser desarrollada por la misma empresa que Matlab. De aquí la importancia de haber procesado los datos en este software.

Para realizar esta tarea, hemos creado un **canal público** para nuestro proyecto [2], el cual recibe datos a través del protocolo MQTT. A continuación, estos datos son representados gráficamente para un post-estudio del entrenamiento. Cabe mencionar, que al no disponer de una versión premium de esta plataforma hemos tenido varias limitaciones. La principal de ellas, es que solo se permite enviar los datos cada 15 segundos lo cual no corresponde con el enventanado que hemos realizado. De todos modos, estos datos quedan archivados en un fichero como se comento anteriormente por lo que pueden ser procesados. Además de esta limitación, nos surgieron más al tener disponible solo la versión gratuita, por lo que decidimos que con una representación de los datos en tiempo real dábamos por concluida esta parte del proyecto para centrarnos más en otras.

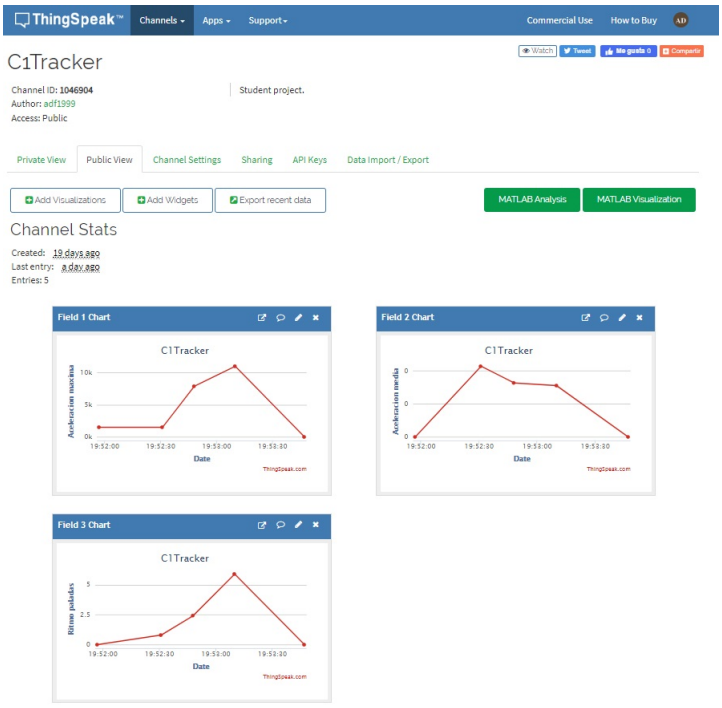


Figura 17: Servidor Thingspeak

4. Desarrollo del proyecto

Hardware y firmware

La unidad central pasó por **numerosas etapas** en lo que al hardware se refiere a lo largo del desarrollo de todo el proyecto.

Originalmente, trabajamos con un módulo **LOLIN32**. Esta placa se basa en el microcontrolador ESP32, microcontrolador pensado para aplicaciones IOT con conectividad Bluetooth 4.0 y Wifi incorporadas, además de 36 pines de E/S. A él conectamos únicamente la IMU LSM9DS1, y programamos el envío de datos por Bluetooth a Android. Fue con este primer prototipo con el que nos fue posible hacer la primera recolección de datos necesaria para el diseño de filtros en Matlab. Cabe destacar que este primer prototipo utilizaba una frecuencia de muestreo de 50Hz, superior a la que implementó en el sistema final. Además, aún no incluía batería, por lo que hacíamos uso de la entrada de alimentación del LOLIN conectándole una batería externa para móviles.

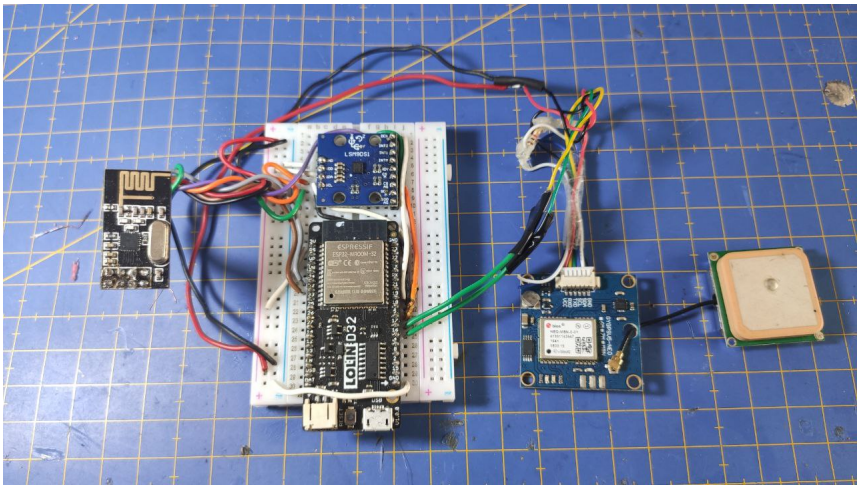


Figura 18: Prototipo con ESP32

Tan pronto como conseguimos resultados satisfactorios en las primeras pruebas, comenzamos el desarrollo de la unidad secundaria. Primero buscamos componentes y analizamos **datasheets**, con el fin de que elegir aquellos que mejor se ajustaran a nuestras necesidades. Un factor determinante en esta elección fue el empaquetado de los chips. Evitamos el uso de componentes BGA o LGA por su difícil soldadura a mano. Con respecto al microcontrolador, optamos por un Atmega328p-AU. Se trata de un microcontrolador famoso para el que existe multitud de documentación en internet, por lo que no debería darnos muchos problemas. Además, la versión AU viene en formato QFN, ideal para este proyecto.

Para el regulador de tensión, desde el primer momento consideramos que un **switching converter** sería la mejor opción, dado que necesitaríamos elevar el voltaje a 3V y este tipo de dispositivos se caracterizan por su alta eficiencia en esta función. Finalmente optamos por el TLV61225, ya que sus especificaciones son suficientemente buenas para lo que necesitamos (capaz de funcionar con tensiones de hasta 0,7V) y su empaquetado aparentaba sencillo de soldar.

Con respecto a la unidad de medición inercial, nos surgieron bastantes dudas sobre cuál elegir. Nuestras principales opciones eran los chips **ADXL345, LIS3DH y MPU9250**. Si bien el que mejores especificaciones ofrecía y el que menor consumo tenía con diferencia era el LIS3DH, lo descartamos debido a que sólo se comercializa en formato LGA. De los otros dos candidatos, nos terminamos decantando por la MPU9250 dado su menor consumo (menor a 4mA a 3,3V).

Finalmente, debíamos elegir un módulo de radio. Una vez más, el factor más determinante fue el **consumo energético**. Es por ello que descartamos adoptar estándares como Bluetooth o Wifi (a frecuencias de envío superiores a unos pocos hercios consumen demasiado para nuestras necesidades). Nuestras dos últimas opciones fueron módulos genéricos de transmisión de datos a 433MHz o el NRF24L01. Ambos ofrecían prácticamente el mismo consumo según datasheets. Decidimos que el NRF24L01 sería la opción más conveniente debido a su versatilidad, superiores especificaciones, reducido tamaño y la posibilidad de elegir tanto la potencia de transmisión como el modo de dormido que se desee.

Respecto a los **componentes pasivos** (resistencias de pull-up, condensadores de desacoplo, etc) optamos por usar las tallas 0805 y 0603, al ser estas suficientemente pequeñas para mantener una buena gestión del espacio pero no tanto como para suponer un problema a la hopra de soldarlas. La PCB se encargó de color verde, doble capa y grosor de 0,8mm. Fue diseñada en KiCAD (proyecto en Github).

Una vez se encargaron los componentes, previendo un tiempo de envío superior a un mes, continuamos avanzando en el desarrollo del firmware de la unidad principal. Con la intención de mejorar el consumo, decidimos que la mejor opción era dormir al microcontrolador entre mediciones y despertarlo mediante una interrupción periódica. Fue entonces cuando nos encontramos con uno de los **mayores problemas** que nos surgieron durante el desarrollo.

El **ESP32** cuenta con varios modos dormido. Por un lado tiene el modo de módem dormido, donde simplemente apaga los periféricos relacionados con comunicaciones inalámbricas. Nosotros buscábamos un ahorro mayor (en este modo el consumo es de unos 20mA), por lo que no nos conformamos con esta opción. Otro de los modos de bajo consumo es el de dormido profundo. En esta caso, el microcontrolador apaga no sólo los periféricos sino si núcleo principal y la memoria, perdiéndose el contenido de las variables. Esto deriva en la necesidad de comenzar el programa desde el inicio (incluyendo la función setup) cada vez que se despierta, lo que puede llevar un tiempo superior a 500ms. Esto nos privaría de conseguir la frecuencia de muestreo que buscamos. En este modo existe un segundo procesador de ultra bajo consumo que sí que funcionaría junto al reloj RTC, pero éste no es capaz de realizar comunicaciones por SPI. Es por estos motivos que este modo de funcionamiento queda descartado. Por último, existe un modo apagado, con los mismos problemas que la opción anterior.

Llegamos a la **conclusión** de que el ESP32 es un microcontrolador pensado para aplicaciones de IoT, donde se realizan mediciones con una frecuencia extremadamente inferior a la que nosotros buscamos. Quizás en este ámbito sí que nos encontraríamos ante un dispositivo muy optimizado y capaz de ofrecer un consumo extremadamente reducido, pero para nuestra aplicación concreta no se da el caso. Es por ello que decidimos cambiar de microcontrolador a un Atmega328p. Este otro micro permite una configuración muy detallada de qué periféricos están encendidos en cada momento. Además, cuenta con interrupciones por WatchDog y Timers, ideales para controlar la temporización del sistema.

Como ya se comentó en secciones anteriores, fueron necesarios ciertos preparativos previos antes de programar el Atmega328p. Se flasheó un nuevo cargador de arranque, Optiboor, a través de su puerto SPI mediante un ISP. Además, se cambió el valor de los fuses para reducir la tensión de detección de brown-out y utilizar el reloj interno a 8MHz. Además, al no contar este micro con conectividad inalámbrica integrada, fue necesario el uso de un módulo HC-06 para enviar los datos a la aplicación móvil.

Más tarde, decidimos añadir los sensores de GPS y Radio. Si bien el módulo de radio NRF24L01 no dio ningún problema (comparte puerto SPI con la IMU y utiliza dos pines digitales para sus Chip Select), con el GPS tuvimos una importante complicación.

El Atmega328p sólo cuenta con un puerto UART, y, en nuestro caso, tenemos dos periféricos que lo utilizan (Bluetooth y GPS). Es por ello que necesitábamos buscar una alternativa que nos premitiera

conectar ambos. La primera opción en la que pensamos fue el uso de la librería SoftWareSerial, la cual permite emular un puerto serie con dos pines de E/S digitales cualesquiera. Sin embargo, esta librería requiere del uso de interrupciones periódicas. El hecho de dormir el microcontrolador y despertarlo imposibilita el uso de esta librería sin modificaciones (en modo dormido se deshabilitan los Timers y sería necesario reconfigurar ciertos registros una vez se despierta). Además, la baja eficiencia de la emulación de Serial por software fue otro de los motivos que hiciera que nos decantáramos por otras opciones.

La siguiente **alternativa** que probamos fue el **diseño de un multiplexor** por hardware. Este constaba de dos transistores BJT para cada canal (2 para RX y 2 para TX) con resistencias en el colector y la base. Somos conscientes de que no era una solución óptima al problema (habría sido mejor alternativa usar FET), pero dado el periodo de cuarentena en que nos encontrábamos, el ser capaces de solucionarlo con componentes sueltos de casa era nuestra única opción. Ambos transistores tenían las salidas conectadas juntas y al puerto serie del microcontrolador. Cada colector iba al puerto serie de uno de los dos periféricos, y mediante una salida digital en la base de los transistores se seleccionaba cuál de ellos se utilizaba en cada momento.

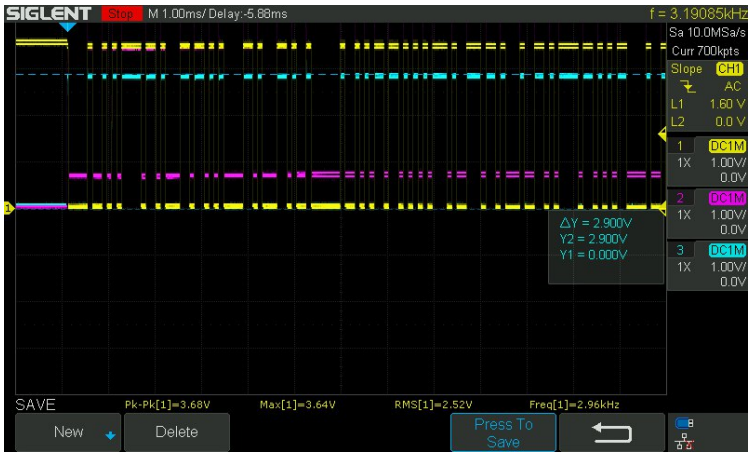


Figura 19: Análisis del multiplexor diseñado con dos BJT en osciloscopio. Amarillo: señal del colector. Magenta: señal en la base. Azul: señal en el emisor.

Esta opción funcionó sin problema ninguno y, en principio, iba a ser definitiva. Sin embargo, nos acabamos dando cuenta que el GPS sólo hacía uso del RX del micro y, el Bluetooth, del TX. Es por ello que conectando un cable a cada uno se puede compartir el puerto serie entre ambos periféricos sin problema alguno.

Por último, quedaba resolver el **problema de la alimentación**. El LOLIN32 tiene incorporado un circuito de carga de baterías, por lo que, de haber seguido con esta placa, no habríamos tenido mayo prroblema. Sin embargo, al cambiar al Atmega fue necesario incluir un circuito de carga externo para que el proceso de carga de la Lipo se hiciera a una corriente constante y adecuada a las características de la batería. fue así como decidimos usar un módulo basado en el TP4056 que, aunque esté pensado para baterías de ion-Litio, funciona también para la nuestra. Añadimos el módulo de carga inalámbrica de cara a, en un futuro, empaquetar todo el sistema en una carcasa sumergible y poder suministrarle carga sin necesidad de clavijas o conectores.

Aplicación

El desarrollo de la aplicación se produjo de forma bastante **lineal:** a medida que iban avanzando el resto de componentes del proyecto (hardware y procesado de datos), se iban implementando en la

aplicación las mejoras logradas.

Cabe destacar un contratiempo que nos surgió durante el desarrollo, cuando aún nos hallábamos probando el prototipo basado en ESP32. En su momento decidimos que no era mala opción probar a utilizar el protocolo **Bluetooth Low Energy (BLE)** para la comunicación entre la unidad central y el smartphone. Implementamos esta alternativa, donde el funcionamiento fue el esperado. Para ello, creamos una característica para cada uno de los parámetros que recogíamos. A continuación, le asignamos la propiedad de 'READ' y 'NOTIFY' a cada uno de ellos para que en el momento que se recibía un dato se notificase al dispositivo que recogía la información. Por último, en el bucle loop del lenguaje Arduino (En esta etapa del proyecto no habíamos desarrollado interrupciones) recogíamos la información del IMU, la convertíamos a string y la enviábamos a cada una de las características. Cabe mencionar que en el código que desarrollamos se incluía toda la arquitectura del BLE, esta se basa en un servidor que contiene servicios y estos últimos características.

Sin embargo, utilizar BLE nos supuso **renunciar a las altas frecuencias de envío de datos** a las que trabajábamos anteriormente. Si se decidiera realizar un Notify para cada valor actualizado, apenas podríamos enviar más de 3/4 datos por segundo. Si, por el contrario, decidiéramos enviar todos los datos en un solo Notify, apenas habría diferencia con el Bluetooth tradicional, ya que, midiendo el consumo y comparando ambas soluciones, apenas encontramos diferencia alguna. Dadas las complicaciones que acarreaba el uso de esta nueva tecnología en nuestro sistema y las pocas ventajas del mismo, decidimos deshacer lo programado y continuar utilizando **Bluetooth classic.**

Procesado de datos

Por parte del desarrollo del procesado de datos el proyecto ha pasado por muchas etapas donde la siguiente etapa mejoraba siempre las prestaciones de la anterior. En primer lugar, comenzamos por buscar un filtro para nuestra señal (para la coordenada y de la aceleración ya que el barco se mueve en esta dirección y es la más representativa), probamos tanto con filtros FIR como IIR de distintos parámetros, esta etapa fue básicamente prueba y error e ir ajustando los parámetros para nuestra señal hasta conseguir un filtrado limpio.

Finalmente lo conseguimos con un filtro IIR con los siguientes parámetros:

- **Orden del filtro:** 4
- **Rizado:** 1 dB
- **Rechazo en la banda no deseada:** 40 dB
- **Frecuencia de corte normalizada:** 0.03

Una vez conseguida la señal filtrada calculamos distintos parámetros de toda la señal como la aceleración máxima, aceleración mínima o aceleración media. Al mismo tiempo calculamos la DFT de dicha señal filtrada para poder sacar más información de esta.

Llegado a este punto queríamos conseguir más información de los datos que teníamos y se nos ocurrió buscar información sobre el algoritmo de **Machine Learning K-Means** que se introduce en la asignatura IEIN. Después de varios días buscando información acerca de Machine Learning, llegamos a la conclusión de que el software más utilizado para este tipo de tareas es **Python y R**. Al final nos decantamos por el primero ya que requiere de un aprendizaje más simple que el segundo.

El algoritmo K-Means se basa en clasificar los datos en N centroides basándose en M features o características. En este punto del trabajo probamos el algoritmo en Python utilizando como features

la aceleración del eje x y la aceleración del eje y sin filtrar, descartando la coordenada z por la poca información que nos aporta.

Aquí surgió el **primer problema**, no sabíamos cuantos centroides eran adecuados para nuestros datos. Investigamos acerca de esto y concluimos que no había ningún algoritmo que se utilizase de manera convencional para dicha tarea pero si que había distintos métodos para optimizar esta búsqueda. En nuestro caso nos decantamos por **'Elbow Method'**. Este método se basa en estudiar la suma de las distancias al cuadrado de cada punto con su centroide para distintos casos con diferentes números de centroides:

$$W_{css} = \sum_{r=1}^{k} \frac{1}{n_r} \cdot D_r \tag{8}$$

Siendo k el número de centroides.

$$D_r = \sum_{i=1}^{n_r - 1} \sum_{j=1}^{n_r} ||d_i - d_j||^2 \tag{9}$$

Siendo D_r la suma de todas las distancias entre las muestras y su centroide y n_r el número de muestras en el centroide r.

Una vez calculada dicha distancia con distintos números de centroides, representamos la siguiente función lineal:

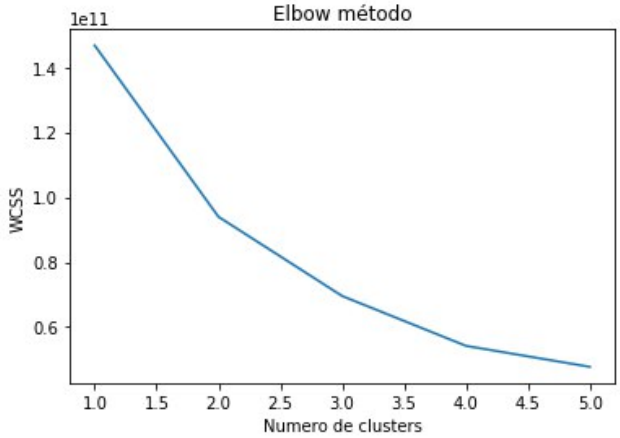


Figura 20: Elbow method

La manera de interpretar dicha gráfica es buscar el punto donde la función se asemeja a consante con el incremento del número de centroides. En un primer lugar, podríamos pensar que el número óptimo es 4 pero realmente si hacemos la gráfica para un número mayor de centroides vemos que realmente donde empieza a ser lineal es con 3. Por lo que en este punto, habíamos encontrado nuestro número óptimo de centroides. Cabe mencionar, que para optimizar dicho método iniciamos la posición de los centroides con el algoritmo **K-means++** incluido en una librería de Python.

Llegado aquí, aplicamos el algoritmo para 3 centroides y dos features (aceleración eje x e y) con ayuda de las librerías pandas, numpy y matplot de Python obteniendo el siguiente resultado:

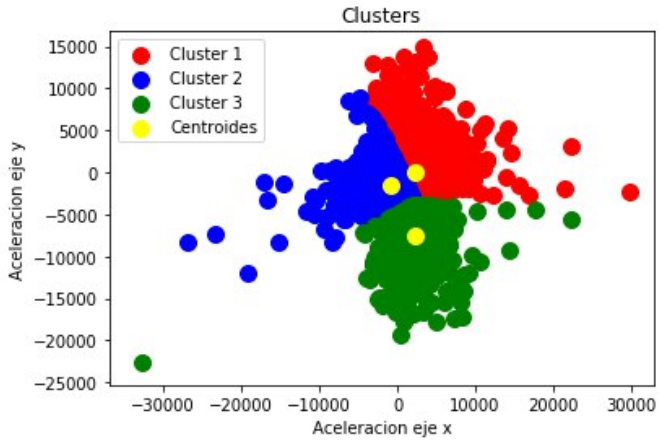
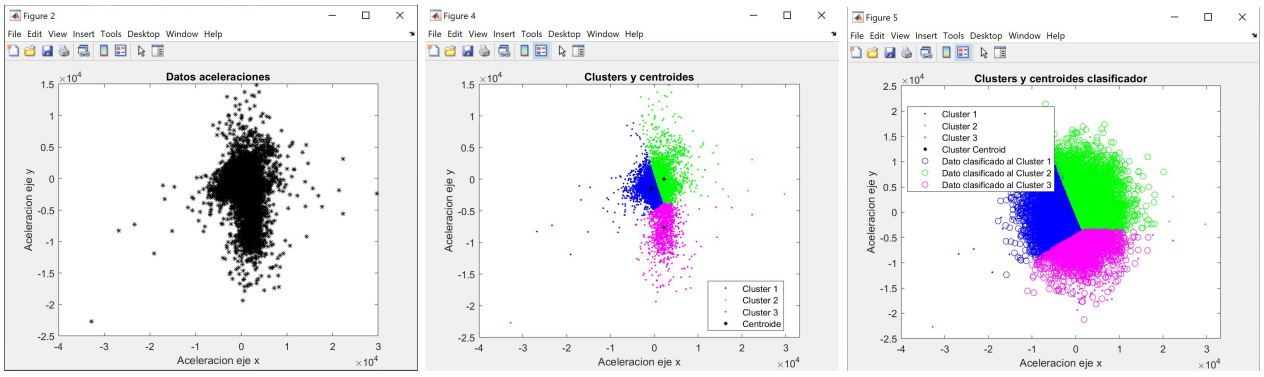


Figura 21: K-Means: Aceleración máxima x e y

En este punto, habíamos conseguido aplicar nuestro primer algoritmo de Machine Learning pero realmente nos surgieron **dos problemas adicionales**. El primero, es que la interpretación de los datos a partir de dos features como la aceleración en el eje x acompañada con la aceleración del eje eran muy complicada. El segundo, es que necesitábamos implementar dicho código en Matlab para seguir orientadondolo en nuestro proyecto.

Nuestra conclusión, fue que para solventar el primer problema era necesario primero solventar el segundo por lo que nos pusimos a investigar como implementar el código que habíamos desarrollado en Python en el software de Matlab. Realmente, esto no fue un problema díficil de solventar ya que Matlab también cuenta con distintas librerías que aceleran el proceso. Tras implementar dicho algoritmo en Matlab, conseguimos los mismos resultados que habíamos obtenido en Python. Además, generamos valores aleatorios para probar un clasificador de muestras en los distintos centroides. Estos fueron los resultados:



(a) Datos (b) K-Means (c) Clasificador

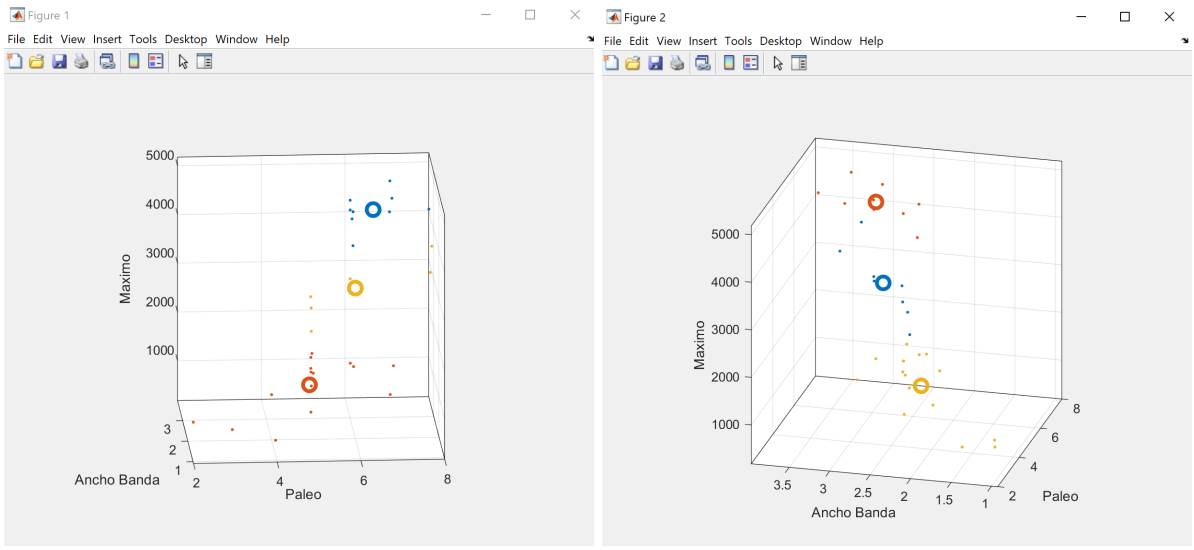
Figura 22: K-Means aceleración x e y en Matlab

Llegados a este punto, habíamos solucionado el problema de adaptarnos al software que necesitábamos pero nuestros datos seguían sin tener una interpretación sencilla. Después de analizar el problema, concluimos que la solución era analizar distintos parámetros de la señal y volver a realizar el estudio con algunos de ellos. Llegados aquí, conseguimos más parámetros que los mencionados anteriormente (aceleración máxima, mínima y media):

- **La frecuencia de paleo con la interpretación de que cada máximo de la señal era un paleo.**
- **Frecuencia máxima de la DFT.**
- **Ancho de banda la frecuencia máxima de la DFT:** constancia del deportista.

Volvimos a realizar el estudio de K-Means eligiendo tres features distintas: máxima aceleración, frecuencia de paleo y ancho de banda de la máxima frecuencia de la DFT y los resultados fueron buenos. Aquí, de nuevo, nos surgió **otro problema**, había puntos donde la aceleración máxima era negativa. Esto pronto lo solventamos eliminando la componente continua.

Llegados aquí, quisimos ir un paso más allá en nuestro proyecto, en lugar de realizar el estudio comentado anteriormente para todas las muestras de la señal, realizarlo para distintas **ventanas de la señal**. A la hora de realizar el enventanado de la señal, consideramos que el hecho de sobreponer muestras de una ventana a otra hacía nuestro trabajo aún mas profesional. Para realizar dicha labor, hemos empleado ventanas de 400 muestras con un solapamiento de 200 muestras. Asimismo, sacamos los parámetros comentados anteriormente para cada uno de las ventanas, consiguiendo así una matriz de 6x28 (Las columnas son las ventanas y las filas los parámetros). Volvimos a realizar el estudio con K-Means y los resultados fueron los siguientes:



(a) Vista 1 (b) Vista 2

Figura 23: K-Means: aceleración máxima, frecuencia de paleo y ancho de banda

Llegado a este punto del trabajo consideramos que deberíamos haber buscado un parámetro **más representativo** que el ancho de banda de la máxima frecuencia de la DFT. Es decir, un parámetro que tuviese una relación lineal con el máximo de la aceleración y el número de paleos.

Después de un intenso estudio, llegamos a la conclusión que podíamos utilizar la coordenada x del giroscopio. Esta, coincide con el cabeceo del barco y es un parámetro muy interesante para nuestra tarea. Repetimos el proceso comentado anteriormente y conseguimos una matriz 7*28, volvimos realizar el estudio K-Means sustituyendo el ancho de banda por dicho parámetro y obtuvimos un resultado mucho mejor. De esta manera, dimos por concluida esta parte del proyecto con los resultados comentados en el apartado 3.4 (Procesado de datos).

5. Futuras implementaciones

En caso de continuar mejorando el proyecto en un futuro próximo, los principales frentes abiertos donde es posible incluir nuevas funcionalidades o mejorar las ya existentes son:

- Mejora del sistema de Machine Learning e inclusión de redes neuronales
- Diseño de PCB para la unidad central
- Reemplazo del módulo HC-06 por uno de menor consumo
- Reemplazo de la batería Lipo por una de tecnología Li-ion
- Mejora del servicio en la nube e Thingspeak con nuevas funcionalidades
- Diseño de una carcasa e impresión 3D de la misma

6. Conclusión

Este proyecto nos ha servido como resumen de todo lo estudiado en la carrera hasta el momento. En él, se juntan las ramas de la electrónica (etapas de hardware, programación a bajo nivel y control de registros, interrupciones y consumo e implementación de Machine Learning), telemática (empaquetado de datos y envío de ellos mediante protocolo HTTP para la subida a un servidor MQTT, conectando el dispositivo a Internet y formando parte del IoT), sistemas audiovisuales (filtrado de la señal y cálculo de sus parámetros más característicos) y sistemas de telecomunicación (trabajo en el dominio de la frecuencia). El hecho de realizar un trabajo práctico en el que se junten todas las especialidades nos sirvió para afianzar los conocimientos teóricos aprendidos a lo largo del grado y asimilarlos de forma definitiva, logrando así desarrollar una aplicación práctica de los mismos.

7. Bibliografía

1. https://github.com/pepassaco/pIMU
2. https://thingspeak.com/channels/1046904
3. A. V. Oppenheim, R.W. Schafer: Discrete-Time Signal Processing. Prentice-Hall, 3rd Ed, 2010. ("Libro de texto")
4. https://es.mathworks.com/help/matlab/index.html
5. https://docs.python.org/3/reference/
6. "Signals and Systems", segunda edición, de A.V. Oppenheim, A.S. Willsky y S.H. Nawab, editorial Prentice Hall, 1997
7. "MATLAB for Engineers", Holly Moore. Editorial Pearson Education 2009
8. Multimedia Big Data Computing for IoT Applications. Concepts, Paradigms and Solutions. Sudeep TanwarSudhanshu TyagiNeeraj Kumar

8. Anexo

8.1. Código de la unidad secundaria

```
/*
 * portable Inertial Measurement Unit
 *
 * Instructions and more can be found at:
 *
 * https://github.com/pepassaco/pIMU
 *
 * by Pablo Alvarez Dominguez.
 */

#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
#include "MPU9250.h"
#include <avr/sleep.h>
#include <avr/power.h>
#include <avr/wdt.h>


MPU9250 IMU(SPI,5);   //IMU MPU9250 por SPI, con CS en pin 5
int status;           //Estado de la IMU
double msg[3];        //Array donde se almacenan los valores del acelerometro

RF24 radio(9, 8);     // CE, CSN el emisor de radio NRF24L01
const byte address[6] = "00001";

volatile int f_wdt=1; //Flag para el WDT


ISR(WDT_vect)         //Rutina de atencion a la interrupcion del watchdog
{
  if(f_wdt == 0)      //se limpia el flag
  {
    f_wdt=1;
  }
}

void imuConf(){              //Configuracion inicial de la imu
  status = IMU.begin();
  if (status < 0) {          //Si no se inicia, no hagas nada

    while(1) {
    }

  }
}

void nrfConf(){                          //Configuracion de la radio
  radio.begin();                         //Inicio
  radio.openWritingPipe(address);        //Se abre una tuberia de datos en la direccion
      especificada
  radio.setPALevel(RF24_PA_MIN);         //Nivel de potencia de tx al minimo
  radio.stopListening();                 //Desactiva la escucha (solo tx)
```

```
}

void wdtConf(){                             //Configuracion del WDT (registros del
    microcontrolador)

  /* Clear the reset flag. */
  MCUSR &= ~(1<<WDRF);

  /* In order to change WDE or the prescaler, we need to
   * set WDCE (This will allow updates for 4 clock cycles).
   */
  WDTCSR |= (1<<WDCE) | (1<<WDE);

  /* set new watchdog timeout prescaler value */
  WDTCSR = 0<<WDP0 | 0<<WDP1 | 0<<WDP2 | 0<<WDP3; /* 16 miliseconds */

  /* Enable the WD interrupt (note no reset). */
  WDTCSR |= _BV(WDIE);
}

void enterSleep(void){                  //Metodo que duerme al micro

  set_sleep_mode(SLEEP_MODE_PWR_DOWN); //Selecciona como modo el apagado del micro

  sleep_mode();                         //Entra en el modo seleccionado

  ADCSRA = 0;                           //Desactiva el ADC
  power_all_disable();                  //Desactiva todos los modulos

  power_timer0_enable();                //Reactiva el timer0
  power_spi_enable();                   //Reactiva el SPI
}

void manda() {

  IMU.readSensor();

  msg[0] = IMU.getAccelX_mss();
  msg[1] = IMU.getAccelY_mss();
  msg[2] = IMU.getAccelZ_mss();

  radio.powerUp();
  radio.write(&msg, sizeof(msg));
  radio.powerDown();

}

void setup() {

  ADCSRA = 0;
  power_all_disable();

  power_timer0_enable();
  power_spi_enable();

  imuConf();
  nrfConf();
  wdtConf();
```

```
}

void loop(){

  if(f_wdt == 1){
    manda();
    f_wdt = 0;     //limpiamos el flag
    enterSleep();
  }

}
```

8.2. Código de la unidad principal

```
#include <NeoGPS_cfg.h>
#include <ublox/ubxGPS.h>
#include <GPSport.h>
#include <avr/sleep.h>
#include <avr/power.h>
#include <avr/wdt.h>

#include <Streamers.h>
#include <NeoTeeStream.h>

#include <Wire.h>
#include <SPI.h>
#include <SparkFunLSM9DS1.h>

#include <nRF24L01.h>
#include <RF24.h>

#define LSM9DS1_M_CS 10
#define LSM9DS1_AG_CS 2
#define NRF_CE 10
#define NRF_CSN 9

#define PRINT_SPEED 20


uint8_t LastSentenceInInterval = 0xFF;
static gps_fix fix;
const unsigned char dormir[] = {0xB5, 0x62, 0x02, 0x41, 0x08, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x02, 0x00, 0x00, 0x00, 0x4D, 0x3B};
float dist =0;
NeoGPS::Location_t empieza;
bool primera;

float olddist, oldlat, oldlong;
float oldvel;

volatile int f_wdt=1; //VOLATILE PORQUE SE COMPARTE OCN LA ISR

LSM9DS1 imu;
```

```
RF24 radio(NRF_CE, NRF_CSN);
const byte address[6] = "00001";
float msg[3] = {0,0,0};
int t;

class MyGPS : public ubloxGPS {
public:

    enum
      {
        GETTING_STATUS,
        GETTING_LEAP_SECONDS,
        GETTING_UTC,
        RUNNING
      }
        state NEOGPS_BF(8);

    MyGPS( Stream *device ) : ubloxGPS( device )
    {
      state = GETTING_STATUS;
    }

    //-------------------------

    void get_status()
    {
      static bool acquiring = false;

      if (fix().status == gps_fix::STATUS_NONE) {
        static uint32_t dotPrint;
        bool           requestNavStatus = false;

        if (!acquiring) {
          acquiring = true;
          dotPrint = millis();
          DEBUG_PORT.print( F("Acquiring...") );
          requestNavStatus = true;

        } else if (millis() - dotPrint > 1000UL) {
          dotPrint = millis();
          DEBUG_PORT << '.';

          static uint8_t requestPeriod;
          if ((++requestPeriod & 0x07) == 0)
            requestNavStatus = true;
        }

        if (requestNavStatus)
          // Turn on the UBX status message
          enable_msg( ublox::UBX_NAV, ublox::UBX_NAV_STATUS );

      } else {
        if (acquiring)
          DEBUG_PORT << '\n';
        DEBUG_PORT << F("Acquired status: ") << (uint8_t) fix().status << '\n';

        #if defined(GPS_FIX_TIME) & defined(GPS_FIX_DATE) & \
```

```
        defined(UBLOX_PARSE_TIMEGPS)

      if (!enable_msg( ublox::UBX_NAV, ublox::UBX_NAV_TIMEGPS ))
        DEBUG_PORT.println( F("enable TIMEGPS failed!") );

      state = GETTING_LEAP_SECONDS;
    #else
      start_running();
      state = RUNNING;
    #endif
  }
} // get_status

//--------------------------

void get_leap_seconds()
{
  #if defined(GPS_FIX_TIME) & defined(GPS_FIX_DATE) & \
      defined(UBLOX_PARSE_TIMEGPS)

    if (GPSTime::leap_seconds != 0) {
      DEBUG_PORT << F("Acquired leap seconds: ") << GPSTime::leap_seconds << '\n';

      if (!disable_msg( ublox::UBX_NAV, ublox::UBX_NAV_TIMEGPS ))
        DEBUG_PORT.println( F("disable TIMEGPS failed!") );

      #if defined(UBLOX_PARSE_TIMEUTC)
        if (!enable_msg( ublox::UBX_NAV, ublox::UBX_NAV_TIMEUTC ))
          DEBUG_PORT.println( F("enable TIMEUTC failed!") );
        state = GETTING_UTC;
      #else
        start_running();
      #endif
    }
  #endif

} // get_leap_seconds

//--------------------------

void get_utc()
{
  #if defined(GPS_FIX_TIME) & defined(GPS_FIX_DATE) & \
      defined(UBLOX_PARSE_TIMEUTC)

    lock();
      bool           safe = is_safe();
      NeoGPS::clock_t sow = GPSTime::start_of_week();
      NeoGPS::time_t utc = fix().dateTime;
    unlock();

    if (safe && (sow != 0)) {
      DEBUG_PORT << F("Acquired UTC: ") << utc << '\n';
      DEBUG_PORT << F("Acquired Start-of-Week: ") << sow << '\n';

      start_running();
    }
  #endif
```

```
} // get_utc

//--------------------------

void start_running()
{
  bool enabled_msg_with_time = false;

  #if defined(UBLOX_PARSE_POSLLH)
    if (!enable_msg( ublox::UBX_NAV, ublox::UBX_NAV_POSLLH ))
      DEBUG_PORT.println( F("enable POSLLH failed!") );

    enabled_msg_with_time = true;
  #endif

  #if defined(UBLOX_PARSE_PVT)
    if (!enable_msg( ublox::UBX_NAV, ublox::UBX_NAV_PVT ))
      DEBUG_PORT.println( F("enable PVT failed!") );

    enabled_msg_with_time = true;
  #endif

  #if defined(UBLOX_PARSE_VELNED)
    if (!enable_msg( ublox::UBX_NAV, ublox::UBX_NAV_VELNED ))
      DEBUG_PORT.println( F("enable VELNED failed!") );

    enabled_msg_with_time = true;
  #endif

  #if defined(UBLOX_PARSE_DOP)
    if (!enable_msg( ublox::UBX_NAV, ublox::UBX_NAV_DOP ))
      DEBUG_PORT.println( F("enable DOP failed!") );
    else
      DEBUG_PORT.println( F("enabled DOP.") );

    enabled_msg_with_time = true;
  #endif

  #if defined(UBLOX_PARSE_SVINFO)
    if (!enable_msg( ublox::UBX_NAV, ublox::UBX_NAV_SVINFO ))
      DEBUG_PORT.println( F("enable SVINFO failed!") );

    enabled_msg_with_time = true;
  #endif

  #if defined(UBLOX_PARSE_TIMEUTC)

    #if defined(GPS_FIX_TIME) & defined(GPS_FIX_DATE)
      if (enabled_msg_with_time &&
          !disable_msg( ublox::UBX_NAV, ublox::UBX_NAV_TIMEUTC ))
        DEBUG_PORT.println( F("disable TIMEUTC failed!") );

    #elif defined(GPS_FIX_TIME) | defined(GPS_FIX_DATE)
      // If both aren't defined, we can't convert TOW to UTC,
      // so ask for the separate UTC message.
      if (!enable_msg( ublox::UBX_NAV, ublox::UBX_NAV_TIMEUTC ))
        DEBUG_PORT.println( F("enable TIMEUTC failed!") );
```

```
        #endif

      #endif

      state = RUNNING;

    } // start_running

    //------------------------

    bool running()
    {
      switch (state) {
        case GETTING_STATUS    : get_status    (); break;
        case GETTING_LEAP_SECONDS: get_leap_seconds(); break;
        case GETTING_UTC       : get_utc       (); break;
      }

      return (state == RUNNING);

    } // running

} NEOGPS_PACKED;


ISR(WDT_vect)
{
  t++;
  if(f_wdt == 0)
  {
    f_wdt=1;
  }
}

void enterSleep(void)
{
  set_sleep_mode(SLEEP_MODE_PWR_SAVE); /* EDIT: could also use SLEEP_MODE_PWR_DOWN for lowest
      power consumption. */
  sleep_enable();

  /* Now enter sleep mode. */
  sleep_mode();

  /* The program will continue from here after the WDT timeout*/
  sleep_disable(); /* First thing to do is disable sleep. */

  /* Re-enable the peripherals. */
  power_all_enable();
}


// Construct the GPS object and hook it to the appropriate serial device
static MyGPS gps( &gpsPort );

#ifdef NMEAGPS_INTERRUPT_PROCESSING
  static void GPSisr( uint8_t c )
  {
    gps.handle( c );
```

```
  }
#endif

//NeoGPS::Location_t club( 422585645L, -87838015L );

//--------------------------

static void configNMEA( uint8_t rate )
{
  for (uint8_t i=NMEAGPS::NMEA_FIRST_MSG; i<=NMEAGPS::NMEA_LAST_MSG; i++) {
    ublox::configNMEA( gps, (NMEAGPS::nmea_msg_t) i, rate );
  }
}

//--------------------------

static void disableUBX()
{
  gps.disable_msg( ublox::UBX_NAV, ublox::UBX_NAV_TIMEGPS );
  gps.disable_msg( ublox::UBX_NAV, ublox::UBX_NAV_TIMEUTC );
  gps.disable_msg( ublox::UBX_NAV, ublox::UBX_NAV_VELNED );
  gps.disable_msg( ublox::UBX_NAV, ublox::UBX_NAV_POSLLH );
  gps.disable_msg( ublox::UBX_NAV, ublox::UBX_NAV_DOP );
}

//--------------------------

void setup()
{
  t = 0;

  // Start the normal trace output
  DEBUG_PORT.begin(9600);
  while (!DEBUG_PORT)
    ;

  DEBUG_PORT.print( F("ublox binary protocol example started.\n") );
  DEBUG_PORT << F("fix object size = ") << sizeof(gps.fix()) << '\n';
  DEBUG_PORT << F("ubloxGPS object size = ") << sizeof(ubloxGPS) << '\n';
  DEBUG_PORT << F("MyGPS object size = ") << sizeof(gps) << '\n';
  DEBUG_PORT.println( F("Looking for GPS device on " GPS_PORT_NAME) );
  DEBUG_PORT.flush();

  // Start the UART for the GPS device
  #ifdef NMEAGPS_INTERRUPT_PROCESSING
    //gpsPort.attachInterrupt( GPSisr );
  #endif
  gpsPort.begin(9600);

  // Turn off the preconfigured NMEA standard messages
  configNMEA( 0 );

  // Turn off things that may be left on by a previous build
  disableUBX();



  while (!gps.running())
```

```
    if (gps.available( gpsPort ))
      gps.read();



  if (imu.beginSPI(LSM9DS1_AG_CS, LSM9DS1_M_CS) == false) // note, we need to sent this our
      CS pins (defined above)
  {

    while (1)
      ;
  }

  radio.begin();
  radio.openReadingPipe(0, address); //Setting the address at which we will receive the data
  radio.setPALevel(RF24_PA_MAX);    //You can set this as minimum or maximum depending on the
      distance between the transmitter and receiver.
  radio.startListening();          //This sets the module as receiver

  primera = true;


  /*** Setup the WDT ***/

  /* Clear the reset flag. */
  MCUSR &= ~(1<<WDRF);

  /* In order to change WDE or the prescaler, we need to
   * set WDCE (This will allow updates for 4 clock cycles).
   */
  WDTCSR |= (1<<WDCE) | (1<<WDE);

  /* set new watchdog timeout prescaler value */
  WDTCSR = 0<<WDP0 | 0<<WDP1 | 0<<WDP2 | 0<<WDP3; /* 16 miliseconds */

  /* Enable the WD interrupt (note no reset). */
  WDTCSR |= _BV(WDIE);

}


void sendUBX( const unsigned char *progmemBytes, size_t len )
{
  gpsPort.write( 0xB5 ); // SYNC1
  gpsPort.write( 0x62 ); // SYNC2

  uint8_t a = 0, b = 0;
  while (len-- > 0) {
    uint8_t c = pgm_read_byte( progmemBytes++ );
    a += c;
    b += a;
    gpsPort.write( c );
  }

  gpsPort.write( a ); // CHECKSUM A
  gpsPort.write( b ); // CHECKSUM B
```

```
} // sendUBX


void loop(){

    if(f_wdt == 1) {

      actualizaGPS();
      actualizaIMU();
      actualizaRadio();
      printDatos();
      Serial.flush();

      f_wdt = 0;
      enterSleep();

    }

}

static void actualizaGPS() {

  while (gps.available( gpsPort )) {
    fix = gps.read();
    if(primera){
      empieza = fix.location;
      olddist = 0;
      oldlat = fix.latitudeL();
      oldlong = fix.longitudeL();
      oldvel = fix.speed_kph();
      primera = false;

    }
    dist = fix.location.EquirectDistanceKm(empieza);


    if(dist != olddist){
      olddist = dist;
    }


    if(fix.latitudeL() != oldlat){
      oldlat = fix.latitudeL();
    }

    if(fix.longitudeL() != oldlong){
      oldlong = fix.longitudeL();
    }

     if(fix.speed_kph() != oldvel){
      oldvel = fix.speed_kph();
    }

  }


}
```

```
void actualizaIMU(){

  imu.readAccel();
  imu.readGyro();

}

void actualizaRadio(){
  if (radio.available()) {

      radio.read(&msg, sizeof(msg)); //Reading the data

  }
}

void printDatos(){

  Serial.print(imu.ax);
  Serial.print(",");
  Serial.print(imu.ay);
  Serial.print(",");
  Serial.print(imu.az);
  Serial.print(",");

  Serial.print(imu.gx);
  Serial.print(",");
  Serial.print(imu.gy);
  Serial.print(",");
  Serial.print(imu.gz);
  Serial.print(",");

  Serial.print(msg[0]);
  Serial.print(", ");
  Serial.print(msg[1]);
  Serial.print(", ");
  Serial.print(msg[2]);
  Serial.print(", ");

  Serial.print(olddist);
  Serial.print(",");
  Serial.print(oldlat);
  Serial.print(",");
  Serial.print(oldlong);
  Serial.print(",");
  Serial.println(oldvel);

}
```

8.3. Actividad principal de la aplicación Android

```
package com.SDG2.tracker;

/*
    Aplicacin  Android por Pablo lvarez y ngel  Domnguez
 */
```

```
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothSocket;
import android.content.Context;
import android.content.Intent;
import android.graphics.Color;
import android.os.Bundle;
import android.os.Environment;
import android.os.Handler;
import android.os.Looper;
import android.os.Message;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

import okhttp3.Call;
import okhttp3.Callback;
import okhttp3.OkHttpClient;
import okhttp3.Request;
import okhttp3.Response;


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

import com.example.sdg2.R;


import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStreamWriter;
import java.util.Random;
import java.util.UUID;

public class Ardcon extends AppCompatActivity {

    public final static String MODULE_MAC = "00:21:13:02:C1:29";
    //public final static String MODULE_MAC = "24:6F:28:A4:AD:82"; //MAC LOLIN
    public final static int REQUEST_ENABLE_BT = 1;
    private static final UUID MY_UUID =
        UUID.fromString("00001101-0000-1000-8000-00805f9b34fb");

    private final static String url =
        "https://api.thingspeak.com/update?api_key=G0V3XXZR1DW0K0XQ&field";


    public final static double[] coefBG = {0.0299852500336256, -0.0260754799058467,
        0.0498393194648968, -0.0260754799058467, 0.0299852500336256};
    public final static double[] coefAG = {1, -2.77517371712076, 3.20104067333350,
        -1.74952498817971, 0.381984549816238};
```

```
public final static double[] coefB = {0.0147104653027411, -0.0260303388144790,
    0.0357726297842435, -0.0260303388144790, 0.0147104653027411};
public final static double[] coefA = {1, -3.29637927902407, 4.27367254011965,
    -2.55937709441142, 0.595994888302495};


public final static double umbralSup = 500;
public final static double umbralInf = -500;

public final static int ncentroides = 3;

public final static double[][] centroides = {{5.09090909090909, 87.9728341680708,
    1578.23937472080},{5.57142857142857, 192.511543549617, 2798.24538895399},
    {6.30000000000000, 244.461224980252, 4493.27787026733}};

private static final int orden = coefB.length;
private static final int nPuntos = 50;

private static final int fMuestreo = 10;

private File file;

private OkHttpClient client;
private Request request;

private LineGraphSeries<DataPoint> seriesax;
private LineGraphSeries<DataPoint> seriesay;
private LineGraphSeries<DataPoint> seriesaz;
private LineGraphSeries<DataPoint> seriesgx;
private LineGraphSeries<DataPoint> seriesgy;
private LineGraphSeries<DataPoint> seriesgz;

int t,t2;
int nel = 13;
int nvent = 10;
int ordenDFT = 80;
int tamventana = 80;
int solapamiento = 40;
int cuentamVentana;

boolean pasaSup;
boolean pasaInf;

double[] max = new double[nel];
double[] min = new double[nel];
double[] avr = new double[nel];
double[] std = new double[nel];
double pal, maxDFT, dftBW;
int centroideAsociado;

double maxAcum, avrAcum, ritmoAcum;


double[][] ventana = new double[nel][tamventana];

double[][] raw = new double[nel][orden];
double[][] rawG = new double[nel][orden];
double[] proy = new double[nel];
```

```
double[][] gravedad1 = new double[3][orden];
double[][] gravedad2 = new double[3][orden];
double[][] fil = new double[nel][orden];
//double[][] dftcoef = new double[2][ordenDFT];  // 2 por real imaginaria
double[] valorNuevo = new double[nel];

double debug;

BluetoothAdapter bta;
BluetoothSocket mmSocket;
BluetoothDevice mmDevice;
ConnectedThread btt = null;
Button switchLight, switchRelay;
TextView response;
GraphView graph1;
GraphView graph2;
boolean lightflag = false;
boolean relayFlag = true;
public Handler mHandler;

private Random rand;


@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_ardcon);
    Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);
    client = new OkHttpClient();
    rand = new Random();
    createFile();
    Log.i("[BLUETOOTH]", "Creating listeners");
    response = (TextView) findViewById(R.id.response);
    switchRelay = (Button) findViewById(R.id.relay);
    switchLight = (Button) findViewById(R.id.switchlight);
    graph1 = (GraphView) findViewById(R.id.graph);
    graph2 = (GraphView) findViewById(R.id.graph2);
    seriesax = new LineGraphSeries<>();
    seriesay = new LineGraphSeries<>();
    seriesaz = new LineGraphSeries<>();
    seriesgx = new LineGraphSeries<>();
    seriesgy = new LineGraphSeries<>();
    //seriesgz = new LineGraphSeries<>();
    seriesax.setTitle("x");
    seriesay.setTitle("x");
    seriesaz.setTitle("x");
    seriesgx.setTitle("Vel.");
    seriesgy.setTitle("Frec.");
    //seriesgz.setTitle("Inten.");
    seriesax.setColor(Color.BLUE);
    seriesay.setColor(Color.GREEN);
    seriesaz.setColor(Color.RED);
    seriesgx.setColor(Color.GRAY);
    seriesgy.setColor(Color.YELLOW);
    //seriesgz.setColor(Color.BLACK);
    graph1.setTitle("Caracterizacin paleo");
    graph2.setTitle("Estadsticos ventana");
```

```
graph1.getLegendRenderer().setVisible(true);
graph1.getLegendRenderer().setTextSize(30);
graph1.getLegendRenderer().setBackgroundColor(Color.TRANSPARENT);
graph1.getLegendRenderer().setFixedPosition(725,0);
graph2.getLegendRenderer().setVisible(true);
graph2.getLegendRenderer().setTextSize(30);
graph2.getLegendRenderer().setBackgroundColor(Color.TRANSPARENT);
graph2.getLegendRenderer().setFixedPosition(725,0);
graph1.addSeries(seriesax);
graph1.addSeries(seriesay);
graph1.addSeries(seriesaz);
graph2.addSeries(seriesgx);
graph2.addSeries(seriesgy);
//graph2.addSeries(seriesgz);
graph1.getViewport().setXAxisBoundsManual(true);
graph1.getViewport().setMinX(0);
graph1.getViewport().setMaxX(nvent);
graph2.getViewport().setXAxisBoundsManual(true);
graph2.getViewport().setMinX(0);
graph2.getViewport().setMaxX(nPuntos);
//graph1.getViewport().setYAxisBoundsManual(true);
//graph1.getViewport().setMinY(-35000);
//graph1.getViewport().setMaxY(35000);
//graph2.getViewport().setYAxisBoundsManual(true);
//graph2.getViewport().setMinY(-35000);
//graph2.getViewport().setMaxY(35000);

cuentamVentana = 0;
maxDFT = 0;
dftBW = 0;
centroideAsociado = 0;
pasaSup = false;
pasaInf = false;
maxAcum = 0;
avrAcum = 0;
ritmoAcum = 0;

t = 0;
t2 = 0;
rellenaCeros();


switchLight.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

        maxAcum = 0;
        avrAcum = 0;
        ritmoAcum = 0;

        Log.i("[BLUETOOTH]", "Attempting to send data");
        if (mmSocket.isConnected() && btt != null) { //if we have connection to the
            bluetoothmodule
            if (!lightflag) {
                String sendtxt = "LY";
                btt.write(sendtxt.getBytes());
                lightflag = true;
            } else {
```

```
                String sendtxt = "LN";
                btt.write(sendtxt.getBytes());
                lightflag = false;
            }
        } else {
            Toast.makeText(Ardcon.this, "Something went wrong",
                Toast.LENGTH_LONG).show();
        }
    }
});
switchRelay.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {


        /*
        request = new Request.Builder()
                .url(url + "1=" + maxAcum + "&field2=" + avrAcum + "&field3=" +
                    ritmoAcum)
                .build();

         */

        request = new Request.Builder()
                .url(url + "1=" + maxAcum + "&field2=" + avrAcum + "&field3=" +
                    ritmoAcum)
                .build();

        client.newCall(request).enqueue(new Callback() {
                                        @Override
                                        public void onFailure(Call call, IOException
                                            e) {
                                            e.printStackTrace();
                                        }

                                        @Override
                                        public void onResponse(Call call, Response
                                            response) throws IOException {
                                            if (response.isSuccessful()) {
                                                final String myResponse =
                                                    response.body().string();
                                            }
                                        }
                                    });

    }
});

bta = BluetoothAdapter.getDefaultAdapter();

//if bluetooth is not enabled then create Intent for user to turn it on
if(!bta.isEnabled()){
    Intent enableBTIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
    startActivityForResult(enableBTIntent, REQUEST_ENABLE_BT);
}else{
    initiateBluetoothProcess();
}
}
```

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);

    if(resultCode == RESULT_OK && requestCode == REQUEST_ENABLE_BT){
        initiateBluetoothProcess();
    }
}

public void initiateBluetoothProcess(){

    if(bta.isEnabled()){

        //attempt to connect to bluetooth module
        BluetoothSocket tmp = null;
        mmDevice = bta.getRemoteDevice(MODULE_MAC);

        //create socket
        try {
            tmp = mmDevice.createRfcommSocketToServiceRecord(MY_UUID);
            mmSocket = tmp;
            mmSocket.connect();
            Log.i("[BLUETOOTH]","Connected to: "+mmDevice.getName());
        }catch(IOException e){
            try{mmSocket.close();}catch(IOException c){return;}
        }

        Log.i("[BLUETOOTH]", "Creating handler");
        mHandler = new Handler(Looper.getMainLooper()){
            @Override
            public void handleMessage(Message msg) {
                //super.handleMessage(msg);
                if(msg.what == ConnectedThread.RESPONSE_MESSAGE){
                    String txt = (String)msg.obj;
                    String[] separado = txt.split(",");

                    if(separado.length == nel){
                        if(isNumeric(separado[0]) && isNumeric(separado[1]) &&
                             isNumeric(separado[2]) && isNumeric(separado[3]) &&
                             isNumeric(separado[4]) && isNumeric(separado[5]) &&
                             isNumeric(separado[6]) && isNumeric(separado[7]) &&
                             isNumeric(separado[8])){
                            for(int i = 0; i < nel; i++){
                                valorNuevo[i] = Double.parseDouble(separado[i]);
                            }
                            debug = valorNuevo[0];
                           filtra(valorNuevo);
                            proyecta(valorNuevo);
                            hazventana(valorNuevo);
                            plotea1();

                        }
                    }


                    Context context = getApplicationContext();
```

```
                    writeToFile(txt+"\n", context);
/*
                    if(response.getText().toString().length() >= 20){
                        response.setText("");
                        response.append(txt);
                        //response.append("\n" + avr[0] + avr[1] + avr[2]);
                    }else{
                        response.append("\n" + txt);
                        //response.append("\n" + avr[0] +" "+ avr[1]+" " + avr[2]);
                    }

 */
                }
            }
        };

        Log.i("[BLUETOOTH]", "Creating and running Thread");
        btt = new ConnectedThread(mmSocket,mHandler);
        btt.start();

    }
}

private void writeToFile(String data,Context context) {

    try{
        FileOutputStream fOut = new FileOutputStream(file, true);
        OutputStreamWriter myOutWriter = new OutputStreamWriter(fOut);
        myOutWriter.append(data);

        myOutWriter.close();

        fOut.flush();
        fOut.close();
    }catch(IOException e){
        Toast.makeText(context,"Petou o IO",Toast.LENGTH_SHORT).show();
    }

}

private void createFile(){

    Context context = getApplicationContext();

    File path = new File(Environment.getExternalStorageDirectory().getAbsolutePath() +
        File.separator);

    file = new File(path, "datos.txt");
    try {
        file.createNewFile();

    } catch (IOException e) {
        Toast.makeText(context,"Arquivo NON creado",Toast.LENGTH_SHORT).show();
    }

}

private static boolean isNumeric(String strNum) {
```

```
    if (strNum == null) {
        return false;
    }
    try {
        double d = Double.parseDouble(strNum);
    } catch (NumberFormatException nfe) {
        return false;
    }
    return true;
}

public void rellenaCeros(){
    for(int i = 0; i < nel; i++){
        for(int j = 0; j < orden; j++){
            raw[i][j] = 0;
            fil[i][j] = 0;
            rawG[i][j] = 0;
            if(i<3){
                gravedad1[i][j] = 0;
                gravedad2[i][j] = 0;
            }

        }
        for(int j=0; j < tamventana; j++){
            ventana[i][j] = 0;
        }
        proy[i] = 0;
        valorNuevo[i]=0;
        max[i] = 0;
        min[i] = 0;
        avr[i] = 1;
        std[i] = 0;

    }
}

private void plotea1(){

    seriesax.appendData(new DataPoint(t, proy[6]), true, nPuntos);
    seriesay.appendData(new DataPoint(t, proy[7]), true, nPuntos);
    seriesaz.appendData(new DataPoint(t, proy[8]), true, nPuntos);


    if(t > nPuntos){
        graph1.getViewport().setMinX(t-nPuntos);

        graph1.getViewport().setMaxX(t);

    }else{
        graph1.getViewport().setMinX(0);

        graph1.getViewport().setMaxX(nPuntos);
    }

    t++;
}

private void plotea2(){
```

```
    seriesgx.appendData(new DataPoint(t2, ventana[12][ventana[12].length-1]), true,
        nvent);
    seriesgy.appendData(new DataPoint(t2, maxDFT), true, nvent);
    //seriesgz.appendData(new DataPoint(t2, ), true, nvent);



    if(t2 > nvent){
        graph2.getViewport().setMinX(t2-nvent);
        graph2.getViewport().setMaxX(t2);

    }else{
        graph2.getViewport().setMinX(0);
        graph2.getViewport().setMaxX(nvent);
    }

    t2++;

}


private void filtradumb(){
    for(int i = 0; i < nel-4; i++){     //4 por el GPS, que no se filtra
        for(int j = 0; j < orden-1; j++){
            raw[i][j+1] = raw[i][j];
            fil[i][j+1] = fil[i][j];
        }

        raw[i][0] = proy[i];
        fil[i][0] = raw[i][0];

        for(int j = 1; j < orden; j++){
            //fil[i][0] += raw[i][j]*coefB[j]-fil[i][j]*coefA[j];
        }

    }


}

private void filtraIIR(){
    for(int i = 0; i < 3; i++){     //4 por el GPS, que no se filtra
        for(int j = 0; j < orden-1; j++){
            raw[i][j+1] = raw[i][j];
            fil[i][j+1] = fil[i][j];
        }

        raw[i][0] = proy[i];
        fil[i][0] = raw[i][0]*coefBG[0];

        for(int j = 1; j < orden; j++){
            fil[i][0] += raw[i][j]*coefBG[j]-fil[i][j]*coefAG[j];
        }

    }
```

```
}

private void proyecta(double[] n){

    double[] g1 = {gravedad1[0][0], gravedad1[1][0], gravedad1[2][0]};
    //double[] g2 = {gravedad2[0][0], gravedad2[1][0], gravedad2[2][0]};
    double pr = 0;

    for(int i = 0; i < 3; i++){
        pr += fil[i][0]*g1[i];
    }

    pr /= norm3(g1)*norm3(g1);
    for(int i = 0; i < nel; i++){

        if(i < 3){
            proy[i] = fil[i][0]-pr*g1[i];

        /*
        }else if(i > 5 && i < 9){
            proy[i] = n[i]-((n[i]*g2[i-6])/norm3(g2));

         */
        }else{
            proy[i] = n[i];
        }
    }
}


private void proyectadumb(double[] n){
    for(int i = 0; i < nel; i++){
            proy[i] = n[i];
    }
}

private void filtra(double[] n){
    for(int i = 0; i < 3; i++){     //4 por el GPS, que no se filtra
        for(int j = 0; j < orden-1; j++){
            rawG[i][j+1] = rawG[i][j];
            gravedad1[i][j+1] = gravedad1[i][j];
            fil[i][j+1] = fil[i][j];
        }

        rawG[i][0] = n[i];
        gravedad1[i][0] = rawG[i][0]*coefBG[0];
        fil[i][0] = rawG[i][0]*coefB[0];

        for(int j = 1; j < orden; j++){
            gravedad1[i][0] += rawG[i][j]*coefBG[j]-gravedad1[i][j]*coefAG[j];
            fil[i][0] += rawG[i][j]*coefB[j]-fil[i][j]*coefA[j];
        }


    }


}
```

```
public double norm3(double[] v){
    return(Math.sqrt(v[0]*v[0] + v[1]*v[1] + v[2]*v[2]));
}

public double norm2(double[] v){
    return(Math.sqrt(v[0]*v[0] + v[1]*v[1]));
}


public double[][] dft(double v[]){

    double[][]dftcoef = new double [2][ordenDFT];

    for(int i = 0; i < ordenDFT; i++){

        dftcoef[0][i] = 0;
        dftcoef[1][i] = 0;
    }

    for(int i = 0; i < ordenDFT; i++){
        for(int j = 0; j < v.length; j++){
            dftcoef[0][i] += v[j]*Math.cos(-2*Math.PI*i*j/ordenDFT);                    //
                Parte real
            dftcoef[1][i] += v[j]*Math.sin(-2*Math.PI*i*j/ordenDFT);                    //
                Parte imaginaria
        }
    }

    return dftcoef;
}


private void hazventana(double[] v){
    for(int i = 0; i < nel; i++){
        ventana[i][cuentamVentana+solapamiento] = v[i];
    }

    cuentamVentana++;

    if (cuentamVentana >= (tamventana-solapamiento)) {

        procesaVentana(ventana);

        for(int i = 0; i < nel; i++){
            for(int j = 0; j < tamventana-solapamiento; j++){
                ventana[i][j] = ventana[i][j+solapamiento];

            }
        }

        cuentamVentana = 0;
        plotea2();

    }
}
```

```
private void procesaVentana(double[][] v){

    for(int i = 0; i < nel; i++) {
        max[i] = 0;
        min[i] = 0;
    }

    double med;

    for(int i = 0; i < 9; i++) {            //OJO NUMERO MAGICO, CORREGIR

        med = media(v[i]);

        for (int j = 0; j < tamventana; j++) {
            v[i][j] = v[i][j] - med;
        }
    }


    for(int i = 0; i < nel; i++){
            for(int j = 0; j < tamventana; j++) {

                if(v[i][j] > max[i]){
                    max[i] = v[i][j];
                }
                if(v[i][j] < min[i]){
                    min[i] = v[i][j];
                }

            }

    }


    pal = cuentaPaladas(v[0]);

    double[][] c = dft(v[0]);
    double[] cAbs = abs(c);
    maxDFT = 0;
    double m = 0;

    for(int i = 0; i < cAbs.length/4; i++){

        if(cAbs[i] > m){
            m = cAbs[i];
            maxDFT = i;
        }
    }

    if(maxDFT > 90){
        maxDFT = 0;
    }



    dftBW = bw(cAbs);

    if(dftBW<0){
```

```
            dftBW = 0;
        }

        //Toast.makeText(Ardcon.this, "Paladas: " + pal +" maxDFT: " + maxDFT + " BW: " +
            dftBW, Toast.LENGTH_LONG).show();

        centroideAsociado = kmeans(new double[]{maxDFT, max[3], max[0]});
        if(max[0] > maxAcum){
            maxAcum = max[0];
        }

        avrAcum = (avrAcum*(t2+1) + media(v[0]))/(t2+2);
        ritmoAcum = (ritmoAcum*(t2+1) + pal)/(t2+2);

        //Toast.makeText(Ardcon.this, "maxAcum: " + maxAcum +" avrAcum: " + avrAcum + "
            ritmoAcum: " + ritmoAcum, Toast.LENGTH_LONG).show();
        response.setText("Frecuencia paleo: " + Math.round(maxDFT/ordenDFT*fMuestreo*60) +
            "ppm\nVelocidad: " + v[12][v[12].length-1] + "\nMxima aceleracin: " +
            Math.round(max[1]) + "\nAceleracin media: " + Math.round(media(ventana[1]))+
            "\nConstancia de paleo: " + Math.round(dftBW) + "\nNivel de intensidad: " +
            centroideAsociado);

    }

    private int cuentaPaladas(double[] v){
        int p = 0;

        for(int i = 0; i < tamventana; i++){
            if(!pasaSup){
                if(v[i] > umbralSup){
                    pasaSup = true;
                }
            }else{
                if(!pasaInf && (v[i] < umbralInf)){
                    pasaInf = true;
                }
            }

            if (pasaSup && pasaInf){
                p++;
                pasaSup = false;
                pasaInf = false;
            }
        }

        return p;
    }


    public double media(double[] v){
        double r = 0;
        for(int i = 0; i < v.length; i++){
            r+=v[i];
        }
        return (r/v.length);
    }
```

```
    public int kmeans(double[] v){
       double min = norm3(new double[]{v[0] - centroides[0][0], v[1] - centroides[0][1], v[2]
           - centroides[0][2]});
       int c = 0;

        for(int i = 1; i < ncentroides; i++){
            if( norm3(new double[]{v[0] - centroides[i][0], v[1] - centroides[i][1], v[2] -
                centroides[i][2]}) < min){
               min = norm3(new double[]{v[0] - centroides[i][0], v[1] - centroides[i][1],
                   v[2] - centroides[i][2]});
               c = i;
            }
        }
        return c;
    }


    public double[] abs(double[][] c){
        double[] s = new double[c[0].length];
        for(int i = 0; i < c[0].length; i++){
            s[i] = norm2(new double[]{c[0][i],c[1][i]});
        }
        return s;
    }


    public double bw (double[] c){
        double[] cnorm = new double[c.length];
        double m = 0;

        for(int i = 0; i < c.length; i++){
            if(c[i] > m){
                m = c[i];
            }
        }

        for(int i = 0; i < c.length; i++){
            cnorm[i] = c[i]/m;
        }

        double puntoIz = 0;
        double puntoDer = 0;

        for(int i = 0; i < cnorm.length-1; i++){
            if(cnorm[i] < 0.25 && cnorm[i+1] >= 0.25){
                puntoIz = i;
            } else if(cnorm[i] > 0.25 && cnorm[i+1] <= 0.25){
                puntoDer = i;
            }
        }

        return(puntoDer - puntoIz);
    }

}
```

8.4. Procesado Python

```
# K-Means

# Librerias
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Datos de un entrenamiento
dataset = pd.read_csv('C1_Raul.csv')
X = dataset.iloc[:, [0, 1]].values

# Elbow mtodo para buscar el nmero ptimo de Clusters
from sklearn.cluster import KMeans
wcss = []
for i in range(1, 6):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', max_iter = 300, n_init = 10,
        random_state = 0)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 6), wcss)
plt.title('Elbow mtodo')
plt.xlabel('Numero de clusters')
plt.ylabel('WCSS')
plt.show()

# Entrenando K-Means
kmeans = KMeans(n_clusters = 3, init = 'k-means++', max_iter = 300, n_init = 10,
    random_state = 0)
y_kmeans = kmeans.fit_predict(X)

# Representacion
plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s = 100, c = 'red', label = 'Cluster
    1')
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s = 100, c = 'blue', label = 'Cluster
    2')
plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s = 100, c = 'green', label = 'Cluster
    3')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 100, c =
    'yellow', label = 'Centroides')
plt.title('Clusters')
plt.xlabel('Aceleracion eje x')
plt.ylabel('Aceleracion eje y')
plt.legend()
plt.show()
```

8.5. Procesado Matlab

```
% Filtrado
[b,a] = ellip(4,1,40,0.03); freqz(b,a) % Filtro IIR
Fs = 50; % Frecuencia de muestreo
T = table2array(readtable('C1_Raul.csv'));
t=0:(length(T(:,1))-1); % Todas las componentes
t = t/Fs; % Eje y
```

```
Trecortado = T(1:30*Fs,2); % FFT recortada para R0 hasta 30. El 2 porque es bidimensional.
DFT= abs(fft(Trecortado)); % FFT de la recortada
trecortado = t(1:30*Fs); % t ajustado
yIIR = filter(b,a,T(:,2)); % Salida del filtro IIR con entrada aceleracion en y
yrecortadafiltrada= yIIR(1:30*Fs); % Seal filtrada recortada

% REPRESENTACION DFT
figure
plot(trecortado,DFT)
title('DFT')
axis([0.1 5 0 -50])

% REPRESENTACION FILTRADO
figure
plot(t,T(:,2))
title('Filtrado')
hold on
plot(t,yIIR)
axis([0 50 -12000 5000])

% K-Means aceleraciones
[b,a] = ellip(4,1,40,0.03); freqz(b,a) % Filtro IIR
Fs = 50; % Frecuencia de muestreo
T = table2array(readtable('C1_Raul.csv'));
t=0:(length(T(:,1))-1); % Todas las componentes
t = t/Fs; % Eje y
DFT= abs(fft(T)); % FFT de la recortada
yIIR = filter(b,a,T(:,2)); % Salida del filtro IIR con entrada aceleracion en y

%Datos iniciales
T = table2array(readtable('C1_Raul.csv'));
X = T (:,1:2);
figure; plot(X(:,1),X(:,2),'k*','MarkerSize',5);
title 'Datos aceleraciones';
xlabel 'Aceleracion eje x';
ylabel 'Aceleracion eje y';

%Clasificamos
[idx,C] = kmeans(X,3);

%Representamos
figure;
plot(X(idx==1,1),X(idx==1,2),'r.','MarkerSize',12)
hold on
plot(X(idx==2,1),X(idx==2,2),'b.','MarkerSize',12)
hold on
plot(X(idx==3,1),X(idx==3,2),'g.','MarkerSize',12);
xlabel 'Aceleracion eje x';
ylabel 'Aceleracion eje y';
plot(C(:,1),C(:,2),'kx','MarkerSize',15,'LineWidth',3);
legend('Cluster 1','Cluster 2','Cluster 3','Centroids','Location','NW');
title 'Clusters y centroides';
xlabel 'Aceleracion eje x';
ylabel 'Aceleracion eje y';
hold off

%Representamos con scatter
figure
```

```
gscatter(X(:,1),X(:,2),idx,'bgm')
hold on
plot(C(:,1),C(:,2),'kx','MarkerSize',3,'LineWidth',3);
legend('Cluster 1','Cluster 2','Cluster 3','Centroide')
title 'Clusters y centroides';
xlabel 'Aceleracion eje x';
ylabel 'Aceleracion eje y';

%Probamos nuevos datos
figure
gscatter(X(:,1),X(:,2),idx,'bgm')
hold on
plot(C(:,1),C(:,2),'kx','MarkerSize',3,'LineWidth',3);
Xtest =
    [randn(11041,2)*1000+ones(11041,2);randn(11041,2)*10000*0.5-ones(11041,2);randn(11041,2)*1000];
[~,idx_test] = pdist2(C,Xtest,'euclidean','Smallest',1);
gscatter(Xtest(:,1),Xtest(:,2),idx_test,'bgm','ooo')
legend('Cluster 1','Cluster 2','Cluster 3','Cluster Centroid','Dato clasificado al Cluster
    1','Dato clasificado al Cluster 2','Dato clasificado al Cluster 3')
title 'Clusters y centroides clasificador';
xlabel 'Aceleracion eje x';
ylabel 'Aceleracion eje y';

% Proyecciones

%Fp = 50;        % 3 Hz fp
Fs  = 50;       % 50 hz fs
[b,a] = ellip(4,1,40,0.03);


T = table2array(readtable('C1_Raul.csv'));
t=0:(length(T(:,1))-1);
t = t/Fs;

fitlrada = zeros(length(T(:,1)),6);
proyectada = zeros(length(T(:,1)),3);

gravedad = [mean(T(:,1)) mean(T(:,2)) mean(T(:,3))];


for i = 1:length(T(:,1))
      proyectada(i,:) = T(i,1:3)-((T(i,1:3).*gravedad)/norm(gravedad));
end

filtradaP(:,1) = filter(b,a,proyectada(:,1))-mean(proyectada(:,1));
filtradaP(:,2) = filter(b,a,proyectada(:,2))-mean(proyectada(:,2));
filtradaP(:,3) = filter(b,a,proyectada(:,3))-mean(proyectada(:,3));
filtrada(:,1) = filter(b,a,T(:,1))-mean(T(:,1));
filtrada(:,2) = filter(b,a,T(:,2))-mean(T(:,2));
filtrada(:,3) = filter(b,a,T(:,3))-mean(T(:,3));
filtrada(:,4) = filter(b,a,T(:,4))-mean(T(:,4));
filtrada(:,5) = filter(b,a,T(:,5))-mean(T(:,5));
filtrada(:,6) = filter(b,a,T(:,6))-mean(T(:,6));

figure
subplot(2,3,1)
plot(t,T(:,1))
hold on
```

```
plot(t,T(:,2),'r')
hold on
plot(t,T(:,3),'g')
title("Seal original")
axis([180 200 -2e4 2e4])

subplot(2,3,2)
plot(t,filtrada(:,1))
hold on
plot(t,filtrada(:,2),'r')
hold on
plot(t,filtrada(:,3),'g')
title("Seal original filtrada")
axis([180 200 -5e3 5e3])

subplot(2,3,3)
plot(t,filtradaP(:,1))
hold on
plot(t,filtradaP(:,2),'r')
hold on
plot(t,filtradaP(:,3),'g')
title("Componente ortogonal filtrada")
axis([180 200 -5e3 5e3])

subplot(2,3,4)
plot(t,T(:,1))
hold on
plot(t,T(:,2),'r')
hold on
plot(t,T(:,3),'g')
axis([40 60 -2e4 2e4])

subplot(2,3,5)
plot(t,filtrada(:,1))
hold on
plot(t,filtrada(:,2),'r')
hold on
plot(t,filtrada(:,3),'g')
axis([40 60 -5e3 5e3])

subplot(2,3,6)
plot(t,filtradaP(:,1))
hold on
plot(t,filtradaP(:,2),'r')
hold on
plot(t,filtradaP(:,3),'g')
axis([40 60 -5e3 5e3])



% figure
% subplot(1,2,1)
% plot(t,T(:,4))
% hold on
% plot(t,T(:,5),'r')
% hold on
% plot(t,T(:,6),'g')
```

```
% axis([0 220 -5e3 5e3])
%
% subplot(1,2,2)
% plot(t,filtrada(:,4))
% hold on
% plot(t,filtrada(:,5),'r')
% hold on
% plot(t,filtrada(:,6),'g')
% axis([0 220 -5e3 5e3])


% Procesado final

[b,a] = ellip(4,1,40,0.03); % Filtro IIR
Fs = 50; % Frecuencia de muestreo
T = table2array(readtable('C1_Raul.csv'));
t=0:(length(T(:,1))-1); % Todas las componentes
t = t/Fs; % Eje y
DFT= abs(fft(T)); % FFT de la recortada
yIIR = filter(b,a,T(:,2)) - mean(T(:,2)); % Salida comp 'y' filtrada y sin continua
zIIR = filter(b,a,T(:,4)) - mean(T(:,4)); % Salida comp filtrada y sin continua

% Enventenado
nMuestras = 400;
solapamiento = 200;
nVentanas = ceil(length(yIIR)/nMuestras);
nDatos = 7;

y = buffer(yIIR,nMuestras,solapamiento); % Enventanado
z = buffer(zIIR,nMuestras,solapamiento); % Enventanado

final = zeros (nDatos,nVentanas); % Matriz con ventanas y parmetros

for i=1:nMuestras
    for j=1:nVentanas
        prueba = y(:,j);
        prueba2 = z(:,j);
        [pks,locs] = findpeaks(prueba,Fs,'MinPeakDistance',0.5);
        meanCycle=mean(locs);
        DFTvent = abs(fft(prueba));
        [pk,MaxFreq] = findpeaks(DFTvent,'NPeaks',1,'SortStr','descend');
        final(1,j) = length(pks); % Numero de paladas
        final(2,j) = MaxFreq; % Max frecuencia DFT
        final(3,j) = obw(DFTvent, MaxFreq); % Ancho de banda max frec
        final(4,j) = max(prueba); % Maxima acel ventana
        final(5,j) = min(prueba); % Minima acel ventana
        final(6,j) = mean(prueba); % Media acel ventana
        final(7,j) = max(prueba2);
        clear meanCycle
        clear pk
        clear MaxFreq
        clear DFTvent
        clear pks
        clear locs
    end
end

% Centroides, ploteado 3D
```

```
X = final ([2,7,4],:);
XT = X.';
clr = lines(3);
[idx,C] = kmeans(XT,3);
figure
scatter3(XT(:,1), XT(:,2), XT(:,3), 50, clr(idx,:), 'Marker','.');
hold on
scatter3(C(:,1), C(:,2), C(:,3),100, clr, 'Marker','o', 'LineWidth',3)
view(3), axis vis3d, box on, rotate3d on
xlabel('Frecuencia palada'), ylabel('Cabeceo embarcacin'), zlabel('Aceleracin mxima')

% DFT

T = table2array(readtable('C1_Raul.csv'));
diez = decimate(T(:,2),5);


% Enventenado
nMuestras = 400;
solapamiento = 200;
nVentanas = ceil(length(diez)/nMuestras);
nDatos = 8;

y = buffer(diez,nMuestras,solapamiento); % Enventanado



t=0:(nMuestras-1);
t = t/400*10;
y2 = y(:,8)-mean(y(:,8));
dft= abs(fft(y2)); % FFT de la recortada

figure
plot(t,y2)
title("Datos de la ventana a analizar")

figure
subplot(2,1,1)
plot(t, 10*log(dft))
axis([0 length(t)/40 80 135])
title("DFT completa de una ventana")

subplot(2,1,2)
plot(t, 10*log(dft))
axis([0 2 80 150])
title("DFT cerca de la frecuencia de paleo")
```