Universidad de La Laguna

ESCUELA POLITÉCNICA SUPERIOR DE INGENIERÍA
Sección Náutica, Máquinas y Radioelectrónica Naval

Trabajo presentado para la obtención del título de:

GRADUADO EN TECNOLOGÍAS MARINAS

Diseño de un sensor para una boya oceanográfica de bajo coste

Presentado por

José Antonio Franco Galán

Dirigido por

Carlos Efrén Mora Luis

Presentado en Septiembre de 2015

ULL | Universidad de La Laguna

D. Carlos Efrén Mora Luis, Profesor perteneciente al área de Construcciones Navales del *Departamento de Ingeniería Agraria, Náutica, Civil y Marítima* de la Universidad de La Laguna certifica que:

D. José Antonio Franco Galán, ha realizado bajo mi dirección el trabajo de fin de grado titulado: *Diseño de un sensor para una boya oceanográfica de bajo coste.*

Revisado dicho trabajo, estimo que reúne los requisitos para ser juzgado por el tribunal que le sea designado.

Para que conste y surta los efectos oportunos, expido y firmo el presente Certificado en Santa Cruz de Tenerife, a 14, Septiembre de 2015.

Fdo. Carlos Efrén Mora Luis
Tutor del trabajo

Resumen

El presente trabajo describe el desarrollo de varios sensores de una boya oceanográfica de bajo coste con la que se pretende medir la elongación de las olas.

El dispositivo tiene como núcleo, una placa de Arduino Due con un microprocesador de 32bits, y dispone de un magnetómetro, un sensor de presión atmosférica, medición de la temperatura exterior y una unidad de medición inercial. Como soporte de almacenamiento utiliza una tarjeta SD donde se guardan los datos.

La elongación se obtiene mediante la doble integración y el estudio de la actidud del sistema.

Una vez que el algoritmo devuelve el valor de la altura de la ola —alrededor de 6.556 valores cada periodo de muestreo—, el sistema lo guarda en una tarjeta SD, en un muestreo cada hora.

Los resultados obtenidos hasta el momento muestran una correcta medición de los ángulos de giro en el espacio y una buena aproximación al cálculo del desplazamiento del sistema en el eje Z

Palabras Claves: IMU, Kalman, Arduino, MPU6050, Boya

Abstract

This work describes the development of the sensor of a low cost oceanographic buoy, pursued to measure the elongation of sea waves.

As a core, the device has an Arduino Due board, with a 32 bits microprocessor. It also has a magnetometer, an atmospheric pressure gaugage, an external temperature sensor, and an inertial unit. The data is stored SD card to keek all the information.

The elongation is measured by double integration and the study of attitude.

Once the algorithm gives back the height values of the wave —around 6.556 values every sampling period—, the system saves that information in the SD card hourly.

The results obtained show a proper measuring of the turning angles in the space and a good approximation to the calculus for the system displacement in the Z axis.

Key words: IMU, Kalman, Arduino, MPU6050, buoy

Índice general

Índice de figuras

Acrónimos y simbología

IMU	unidad de medición inercial
hPa	hecto Pascales
Ax	aceleración en el eje X
Ay	aceleración en el eje Y
Az	aceleración en el eje Z
Gx	velocidad angular sobre el eje X
Gy	velocidad angular sobre el eje Y
FPB	filtro paso bajo
FPA	filto paso alto
fc	frecuencia de corte
E/S	entrada y salida
PCB	placa de circuito impreso
CPU	Unidad Central de Proceso

1 Introducción

Este trabajo nace de la idea de diseñar un dispositivo capaz de calcular la elongación de la ola, guardando los datos en un soporte físico para posteriormente ser interpretarlos, usados en diferentes estudios.

El trabajo se divide en 7 capítulos.

Capítulo 1, Introducción.

Capítulo 2 que describe los fundamentos básicos que sustenta el trabajo, y explica la dinámica de la boya, la importancia del sistema de coordenadas y desarrolla una pequeña descripción los sensores montados.

Capítulo 3 que desarrolla el cálculo de la actitud del sistema y se examina su cinemática.

Capítulo 4 dedicado a la electrónica.

Capítulo 5 que desarrolla el código fuente.

Capítulo 6 dedicado a las conclusiones y a las líneas de trabajo para mejorar la plataforma dotándola de mayores prestaciones.

Lo que se plantea con este trabajo es poder diseñar una boya oceanográfica que mantenga un bajo coste de producción, favoreciendo así la integración de este tipos de dispositivos y la posibilidad de ser integrado en un sistema de redes de vigilancia de fenómenos costeros.

2 Fundamentos básicos

El principio de Arquímedes dice que *todo cuerpo sumergido en un líquido recibe un empuje, de abajo hacia arriba, igual al peso del líquido desalojado.*

Sobre un cuerpo sumergido actúan dos fuerzas; su peso ($\overline{P}$), que es vertical en sentido descendente y el empuje ($\overline{E}$) que tiene misma dirección, pero es de sentido contrario.

Para determinar la flotabilidad de un cuerpo es necesario conocer el comportamiento de ambas fuerzas.

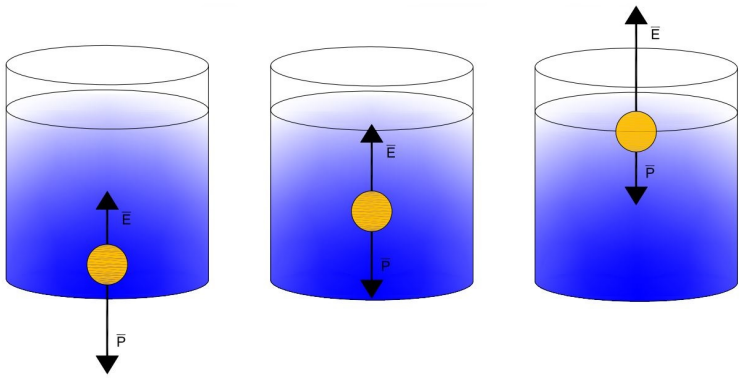

Figura 2.1: Principio de Arquímedes

Dado lo anterior, se pueden producir tres casos:

- Si el peso es mayor que el empuje ($\overline{P} > \overline{E}$), el cuerpo se hunde.
- Si el peso es igual que el empuje ($\overline{P} = \overline{E}$), el cuerpo no se hunde ni emerge.
- Si el peso es menor que el empuje ($\overline{P} < \overline{E}$), el cuerpo flota.

Si la boya está sumergida una distancia *h1*, (fig 2.2) y se cumple que ($\overline{P} = \overline{E}$), ésta permanecerá en equilibrio.

Si se aplica puntualmente un peso ($\overline{P'}$), o una fuerza en sentido vertical, la boya oscilará con un movimiento armónico simple, cuyas magnitudes físicas relevantes son [7]:

- **Elongación:** La separación instantánea de la boya respecto a su posición de equilibrio (X).
- **Periodo:** El tiempo necesario para que la boya recupere la misma posición y velocidad (T).
- **Frecuencia:** La inversa del periodo; indica cuántas oscilaciones se producen en la unidad de tiempo cuando sobre la boya no actúan fuerzas externas ($f = 1/T$) .
- **Amplitud:** La elongación máxima o la máxima separación de la boya respecto a la posición de equilibrio (A).

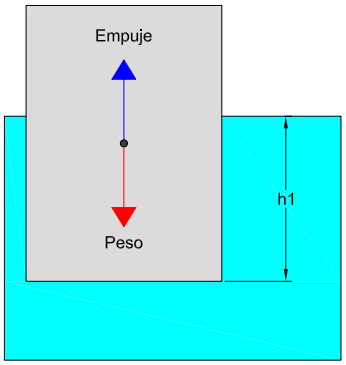

Figura 2.2: Diagrama del sistema en reposo
Fuente: http://www.sc.ehu.es/sbweb/fisica/fluidos/

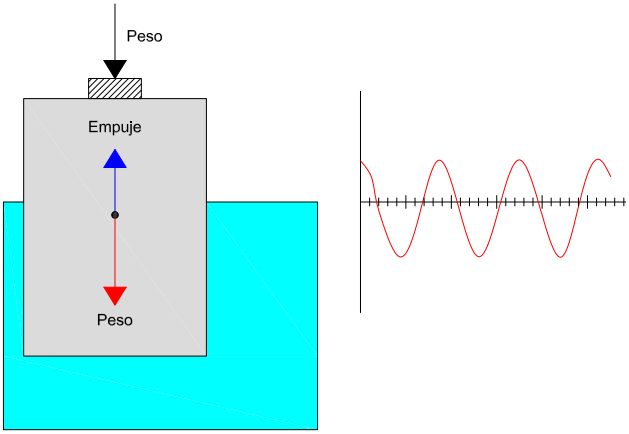

Figura 2.3: Diagrama del sistema en movimiento
Fuente: http://www.sc.ehu.es/sbweb/fisica/fluidos/

Parte del desarrollo matemático va orientado al cálculo de la elongación Z (Δz). Teniendo presente las ecuaciones cinemáticas, la velocidad v_z y la aceleración a_z en la dirección de la elongación se puede expresar matemáticamente como [11]:

$$v_z = \frac{dz}{dt} \tag{2.1}$$

$$a_z = \frac{dv_z}{dt} \tag{2.2}$$

De esta forma la elongación z puede hallarse a partir de la aceleración:

$$z = \iint a_z dt \tag{2.3}$$

Ésta es la ecuación fundamental para el el cálculo del desplazamiento de la boya. Implementar esta ecuación no resulta extremadamente complejo en Arduino, pero este proceso se detalla en el capítulo 3.

2.1 Sistema de coordenadas

Un sistema de coordenadas utiliza una o más variables para determinar la posición de un punto. Para que estos valores tengan cierta coherencia, es necesario relacionarlos con una referencia conocida.

Existen varios sistemas de coordenadas fundamentales. A no ser que se indique lo contrario, estos sistemas son ortogonales (los tres ejes fundamentales están separados entre sí 90°), son dextrógiros (cumplen con la regla de la mano izquierda) y cartesianos. Únicamente difieren en el origen, la orientación relativa de sus ejes y el movimiento relativo entre sus planos [11].

2.1.1 Ángulos de Euler

Un método común para determinar la orientación angular de un sistema de coordenadas respecto a otro es el uso de los tres ángulos de Euler.

Los ángulos de Euler constituyen un conjunto de tres coordenadas angulares que sirven para especificar la orientación de un sistema de referencia de ejes ortogonales, normalmente móvil, respecto a otro sistema de referencia de ejes ortogonales normalmente fijo [1].

Los ángulos de navegación son un tipo de ángulos de Euler usados para describir la actitud —la orientación de un objeto en un sistema de referencia— en tres dimensiones. Éstos son definidos como ángulo de balance (ϕ, o *roll*), ángulo de cabeceo (θ, o *pitch*) y ángulo de guiñada(Ψ, o *yaw*).

2.1.2 Matriz de giro

Un cuerpo flotando en el agua tiene un comportamiento aleatorio. Éste es inducido por los diversos parámetros que comprenden la dinámica de las olas. Debido a esto, se requiere una herramienta que permita trasladar las aceleraciones de un sistema de referencia a otro fijo. Esta herramienta es lo que se conoce como matriz de giro o de rotación.

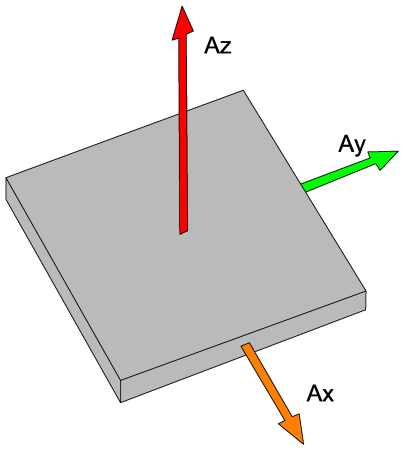

Figura 2.4: Fuerzas medida por un acelerómetro

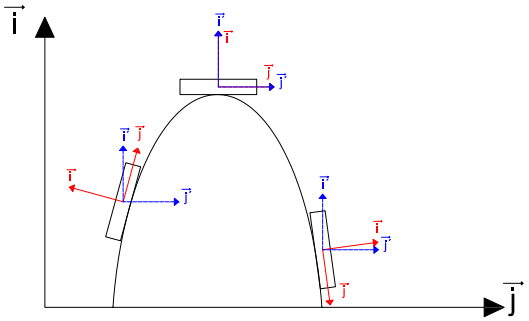

Figura 2.5: Diagrama de fuerzas sobre una ola
$\{i, j\}$ representa el sistema de referencia; $\{i^{'}, j^{'}\}$ el sistema fijo.

2.1.2.1 Uso de las matrices de giro

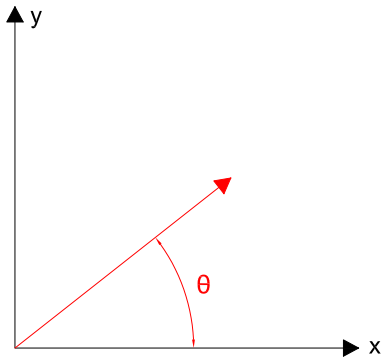

Figura 2.6: Representación de un vector en coordenadas polares

La matriz de rotación en dos dimensiones tiene la siguiente forma:

$$R(\theta) = \begin{bmatrix} cos(\theta) & -sen(\theta) \\ sen(\theta) & cos(\theta) \end{bmatrix} \tag{2.4}$$

Para rotar vectores columna, se multiplica el vector fijo por la matriz de rotación:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} cos(\theta) & -sen(\theta) \\ sen(\theta) & cos(\theta) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{2.5}$$

Así las coordenadas móviles (x', y') del punto (x, y) después de la rotación son:

$$x' = xcos(\theta) - ysen(\theta) \tag{2.6}$$

$$y' = xsen(\theta) + ycos(\theta) \tag{2.7}$$

La dirección del vector rotado es anti-horaria si θ es positiva y tiene sentido horario si θ es negativa. En el caso de que el giro sea tridimensional es necesaria una matriz para cada eje (ecuaciones 2.8, 2.9 y 2.10):

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\theta) & -sen(\theta) \\ 0 & sen(\theta) & cos(\theta) \end{bmatrix} \tag{2.8}$$

$$R_y(\theta) = \begin{bmatrix} cos(\theta) & 0 & sen(\theta) \\ 0 & 1 & 0 \\ -sen(\theta) & 0 & cos(\theta) \end{bmatrix} \tag{2.9}$$

$$R_z(\theta) = \begin{bmatrix} cos(\theta) & -sen(\theta) & 0 \\ sen(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.10}$$

2.2 Sistema de navegación inercial

Una unidad de medición inercial (IMU) es un dispositivo electrónico capaz de medir e informar de ciertos parámetros tales como *velocidad*, *orientación* y *fuerzas gravitacionales*. Estas unidades disponen de sensores inerciales que miden la variación de rotación —giróscopo— y la aceleración —acelerómetro— [6].

Los acelerómetros (fig. 2.4) están colocados de tal forma que sus ejes de medición son ortogonales entre sí. Éstos miden la aceleración inercial en unidades de fuerza G (fuerzas basadas en la aceleración producida por la gravedad).

Los tres giróscopos están colocados en un patrón ortogonal similar, midiendo la posición rotacional referida a un sistema de coordenadas seleccionado de forma arbitraria.

Mediante el seguimiento de la velocidad angular actual del sistema y de la aceleración lineal medida con el sistema de movimiento, es posible determinar su aceleración lineal en el sistema de referencia.

2.2.1 El Giróscopo

Los giroscopios MEMS —componentes electrónicos y mecánicos miniaturizados— miden la velocidad angular (ω) usando el efecto Coriolis. Cuando se hace girar el giroscopio, una pequeña masa se desplaza con los cambios de velocidad angular. Un sensor capacitivo mide esa velocidad angular que es proporcional al recorrido de dicha masa (fig. 2.7).

A partir de la variación del ángulo de giro es posible determinar el grado de inclinación del dispositivo en el que se encuentra montado. En el capítulo 3 se explica este concepto con más claridad.

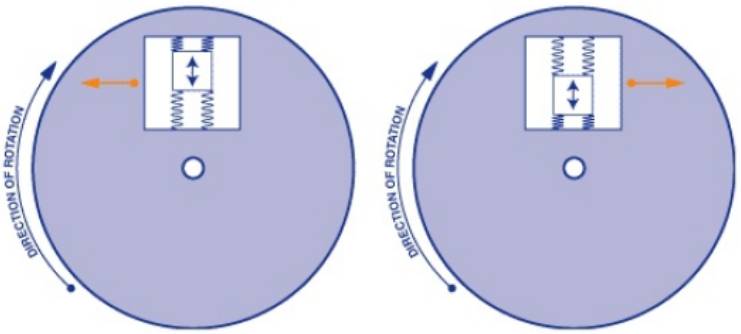

Figura 2.7: Funcionamiento interno de un sensor giroscópico MEMS.
Fuente: http://5hertz.com/

2.2.2 El Acelerómetro

El acelerómetro (fig. 2.8) es un dispositivo capaz de medir aceleraciones, es decir, la variación en la velocidad por unidad de tiempo, si se tiene en cuenta la segunda ley de Newton, que dice: $F = m * a$ y suponemos una masa (m) constante, resulta que la aceleración medida (a) será proporcional a la fuerza (F) que se le aplique al acelerómetro [11].

Esas fuerzas pueden ser estáticas o dinámicas, las estáticas incluyen la gravedad, mientras que las dinámicas pueden incluir vibraciones y/o movimiento, con el empleo de filtros paso bajo o paso alto se podrá discriminar la componente no deseada.

Generalmente, los acelerómetros contienen placas capacitivas internamente (fig. 2.9), algunas de estas placas son fijas, mientras que otras están unidas a resortes minúsculos que se mueven internamente conforme las fuerzas de aceleración que actúan sobre el sensor. Como estas placas se mueven en relación el uno al otro, la capacitancia entre ellos cambia. A partir de estos cambios en la capacitancia, se puede determinar la aceleración.

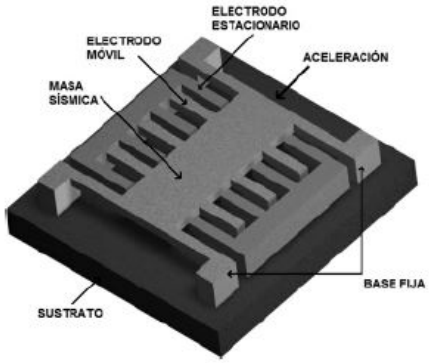

Figura 2.8: Sistema microeléctromecánico para la aceleración en un eje.
Fuente: http://5hertz.com

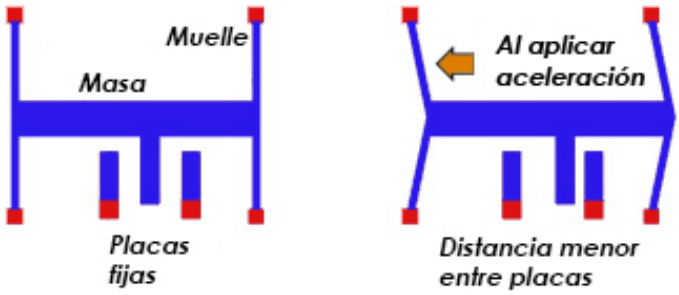

Figura 2.9: Fundamento físico de un acelerómetro.
Fuente: http://atomosybits.com/

2.2.3 Características del MPU6050

EL MPU6050 es una IMU que dispone de 6 grados de libertad, combinando 3 grados para el acelerómetro y 3 para el giróscopo.

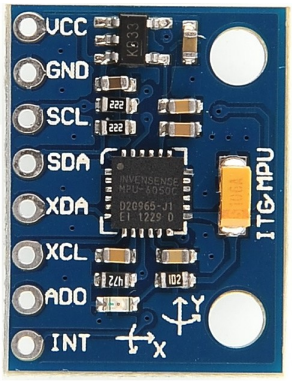

Figura 2.10: Sensor MPU6050 usado en el proyecto.
Fuente: http://www.alldatasheet.es/

Las principales características por las que se usó este sensor fueron:

Giróscopo.

- Fondo de escala programable por el usuario para las lecturas en los tres ejes en: ±250, ±500, ±1000 y ±2000°/*sec*.
- Conversor analógico-digital de 16bit.
- Ruido bajo a frecuencias bajas.
- Filtro paso bajo digital.
- Calibración por software.
- Corriente en funcionamiento $3,6mA$.
- Corriente en standby $5\mu A$.

Acelerómetro.

- Fondo de escala programable por el usuario para las lecturas en los tres ejes en: ±2, ±4, ±8 y $\pm 16G$.
- Conversor analógico-digital de 16bits.
- Corriente en funcionamiento $500\mu A$.

Otras características.

- Conexión por I2C.
- $3,9mA$ de consumo total.
- Rango de tensión de alimentación $2,375V - 3,46V$.

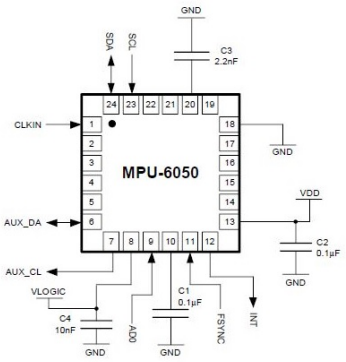

Figura 2.11: Diagrama de conexión MPU6050.
Fuente: http://www.alldatasheet.es/

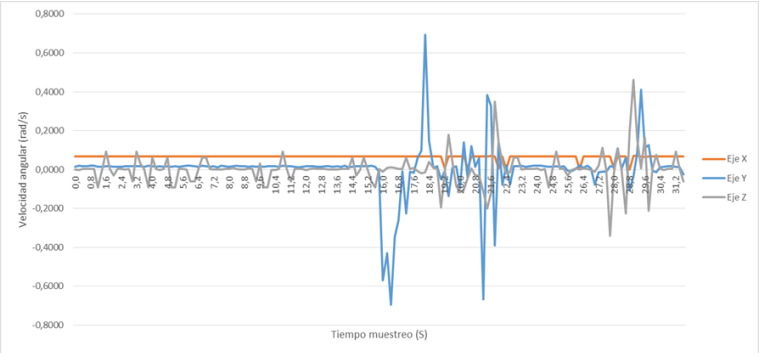

Figura 2.12: Datos obtenidos del giroscopio.

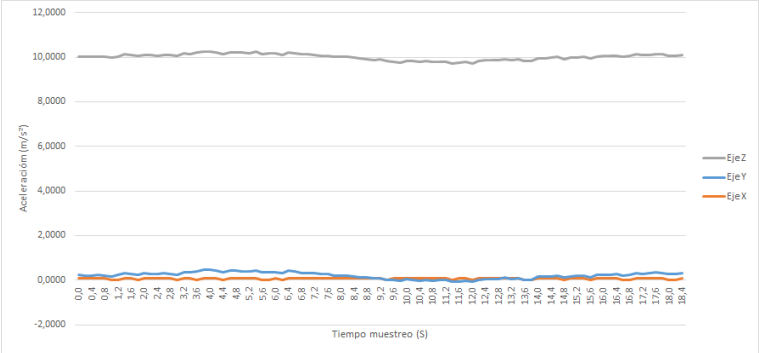

Figura 2.13: Datos obtenidos del acelerómetro.

2.2.4 El Magnetómetro

Además de la información de inclinación/aceleración y velocidades angulares, es necesario determinar la orientación a través de un magnetómetro —conocidos como brújulas digitales—.

Los magnetómetros son dispositivos que sirven para cuantificar en fuerza o dirección la señal magnética de una muestra. En función de como esté orientado el sistema, el magnetómetro medirá el campo terrestre en los tres ejes, pudiéndose calcular la orientación con respecto al norte magnético.

En el caso de la boya, se implementa el HMC5883 de *Honeywell*, cuyas características principales son:

- Magnetómetro triaxial.
- Conversor analógico-digital de 12bits con un bajo ruido en la conversión.

- $100\mu A$ de consumo total.
- Rango de medida ± 8 Gaus.

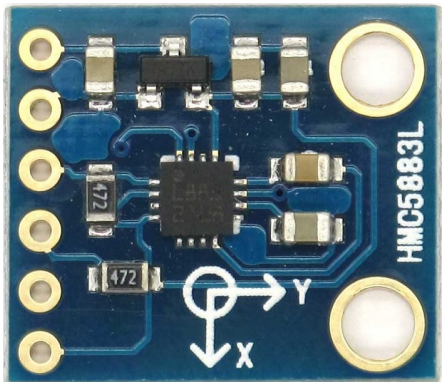

Figura 2.14: Sensor HMC5883 usado en el proyecto.
Fuente: http://www.alldatasheet.es/

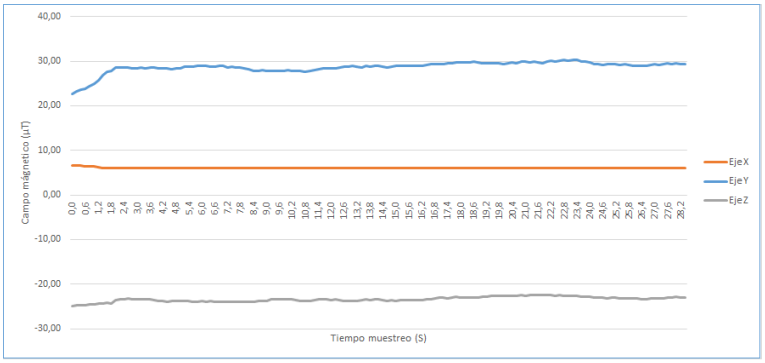

Figura 2.15: Datos obtenidos del magnetómetro.

2.3 Sensor de presión atmosférica

Para intentar incrementar la precisión del sistema, se ha añadido un sensor de presión atmosférica con la finalidad de medir la altura sobre el nivel del mar de la boya.

En el caso de pequeñas variaciones de presión el sensor no funciona como cabría esperar. Por este motivo, los datos que proporciona el sistema son el resultado de un algoritmo, que unifica los valores obtenidos por el medidor de presión y por la IMU.

El sensor es de tecnología piezo-resistiva, con un amplio rango de medida y con una precisión absoluta de hasta 0.03 hecto Pascales (hPa), las características principales son:

- Rango de presiones: 300 a 1,000 hPa ($+9000m... - 500m$ sobre el nivel del mar.)
- Tensión de alimentación: $1,8$ - $3,6V$.
- $5\mu A$ de consumo en el modo estándar (una muestra por segundo).
- Medición de la temperatura.
- Conexión por I2C.
- $0,25m$ de resolución mínima.

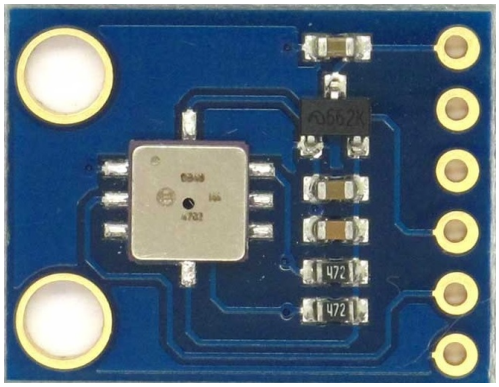

Figura 2.16: Sensor bmp085 usado en el proyecto.
Fuente: http://www.alldatasheet.es/

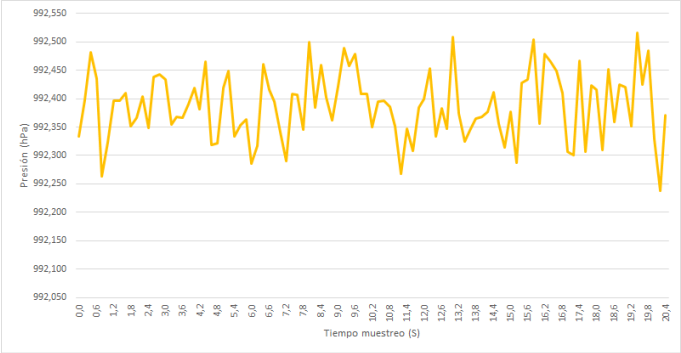

Figura 2.17: Datos obtenidos del sensor de presión.

3 Actitud y Cinemática

Se considera *reposo* al estado en que la resultante de las fuerzas aplicadas a un cuerpo es cero ($\sum \overline{F} = 0$), pudiendo este encontrarse a velocidad constante o nula. Este estado sólo existe dentro de un sistema de referencia.

El *movimiento* es un cambio de posición de un cuerpo a lo largo del tiempo respecto a un sistema de referencia [11].

3.1 Error en las medidas

En este tipo de sensores —acelerómetro y giroscopio— aparecen dos aspectos bastantes problemáticas, el *ruido* y los *errores*. El ruido se debe a perturbaciones electromagnéticas que afectan a los dispositivos electrónicos. En el caso de un acelerómetro se puede medir cualquier ángulo, sin embargo estas lecturas son ruidosas y tienen un cierto margen de error.

Para poder entender la importancia de este fenómeno, se muestran los ángulos tomados por un acelerómetro (fig. 3.1).

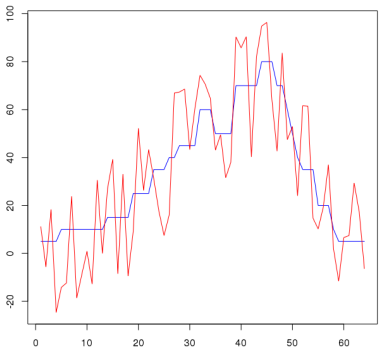

Figura 3.1: Relación entre el águlo real y medido de un acelerómetro
Fuente: https://robologs.files.wordpress.com

El color azul pertenece a las medidas ideales y en rojo las medidas reales. El diagrama pone de manifiesto la inexactitud de los valores en bruto del acelerómetro. Por si esto fuera poco, si se ve afectado por cualquier aceleración que no sea la de la gravedad, puede interpretarse como un cambio de rotación.

El giroscopio tiene unas medidas muchas más precisas que el acelerómetro. Sin embargo es inevitable que se produzca un pequeño error que, con el tiempo, va acumulándose, haciendo que la medida se aleje con cierta velocidad del valor real. A este fenómeno se le conoce como *drift o deriva*.

Para aumentar la precisión y corregir estos inconvenientes existen 2 formas de combinar ambos sensores.

- Filtro Kalman.
- Filtro Complementario.

3.2 Actitud mediante acelerómetros

En reposo un acelerómetro obtendrá un valor de $0G$ para la aceleración en el eje X (Ax) y para la aceleración en el eje Y (Ay), tomando el valor de $1G$ para la aceleración en el eje Z (Az) (fig. 2.4).

Con estos valores y mediante las ecuaciones (3.1) o (3.3) para ϕ y (3.2) o (3.4) para θ [9].

- **Método del arcoseno:**

$$\phi = arcseno\frac{Ax}{1} \tag{3.1}$$

$$\theta = arcseno\frac{Ay}{1} \tag{3.2}$$

  Ya que la aceleración de la gravedad siempre es 1, sólo se necesita la componente de las aceleración que se desea medir. Este método es un poco inexacto y sólo permite la obtención de ángulos de hasta 90°.

- **Método de la arcotangente:**

$$\phi = atan\frac{Ax}{Az} \tag{3.3}$$

$$\theta = atan\frac{Ay}{Az} \tag{3.4}$$

  Este método es más preciso que el anterior debido a que se tienen en cuenta dos medidas, el eje a medir y la inclinación respecto al eje Z. Esto es, la medición del ángulo se realiza respecto a lo que pasa en el cuadrante definido por los dos ejes. Con este método se obtienen ángulos de hasta 180° . Si el sistema sufre aceleraciones por desplazamiento el error cometido es menor, debido a que la aceleración en el eje Z ante un desplazamiento lateral es casi nulo.

Independientemente de las ecuaciones para ϕ y θ, lo más fácil para conocer Ψ sería el uso de un magnetómetro.

Todo esto sólo es aplicable si el sistema está en reposo; si se ve afectado por alguna aceleración diferente a las mencionadas con anterioridad el sistema induce errores en las medidas.

3.3 Actitud mediante giróscopo

Un giróscopo mide velocidad angular respecto a un eje de giro (fig. 3.2). Su influencia ante las perturbaciones electromagnéticas es baja, por lo que no se ve afectado por las vibraciones. El problema radica en que la medida que se busca es la inclinación del sistema y no su velocidad de giro. Por ello se calcula la integral de la ω respecto dt —tiempo que se tarda en tomar una muestra—, la expresión resultante sería:

$$\alpha = \alpha(t-1) + dt * \omega \tag{3.5}$$

Debido al término $\alpha(t-1)$, si no se dispone de valores en reposo correctos los resultados de la ecuación 3.5 son erróneos .

Particularizando la ecuación 3.5 para la velocidad angular sobre el eje X (Gx) y para la velocidad angular sobre el eje Y (Gy) queda que:

$$\phi = \phi(t-1) + dt * Gy \tag{3.6}$$

$$\theta = \theta(t-1) + dt * Gx \tag{3.7}$$

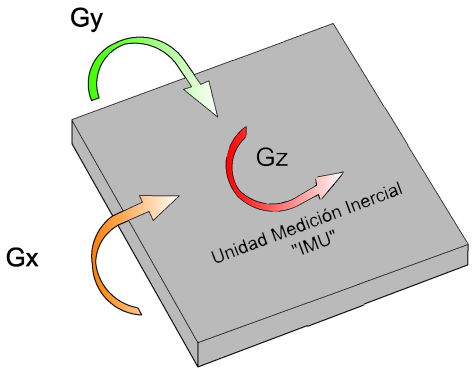

Figura 3.2: Velocidad angular medida por una IMU.

3.4 Filtro Complementario

La idea es muy simple, se debe conseguir eliminar el ruido, la deriva y conseguir que el acelerómetro no cambie de ángulo al detectar otra fuerza que no sea la gravedad. Hay distintos algoritmos, llamados filtros, que hacen esta tarea. Uno de los mejores es el *Filtro Kalman*.

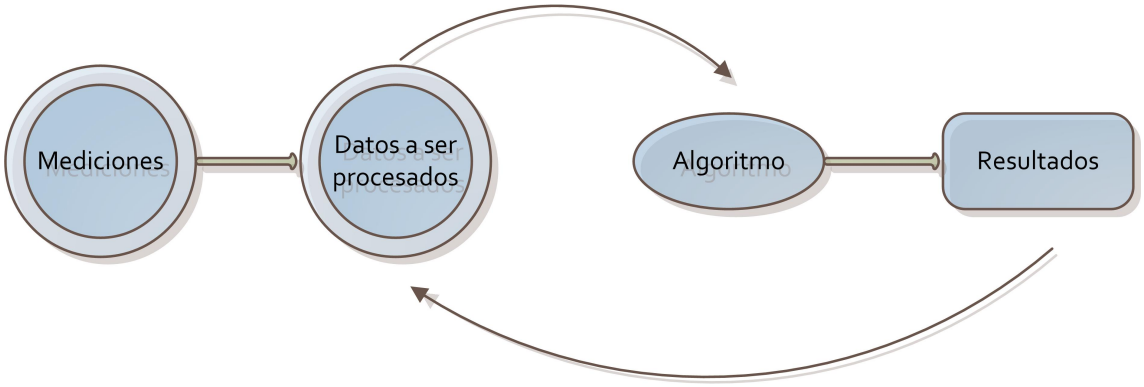

Figura 3.3: Recursividad en el filtro Kalman

El filtro Kalman es capaz de calcular el error de cada medida a partir de las medidas anteriores, eliminarlo y dar el valor real del ángulo. En cierto modo es un algoritmo que aprende en cada iteración, produciendo un consumo elevado de los recursos del microprocesador.

Aunque Arduino soporta este filtro, se descartó por el elevado consumo de recursos del sistema. La solución adoptada para resolver este inconveniente fue el uso del *filtro complementario*, con un menor coste de procesamiento y una buena precisión. El funcionamiento de este filtro consiste en la unión de dos filtros: un filto paso alto (FPA) para el giroscopio y un filtro paso bajo (FPB) para el acelerómetro. El primero deja pasar únicamente los valores por encima de un cierto límite, mientras que el FPB sólo permite a los que están por debajo.

La ecuación resultante de combinar los dos filtros es:

$$Angulo = \alpha * (Angulo(t-1) + AnguloGyro * dt) + (1-\alpha) * AnguloAccel \tag{3.8}$$

Donde:

- **Angulo:** Variable a calcular.
- α**:** Constante de los filtros.
- **Angulo(t-1):** Ángulo calculado en el instante de tiempo anterior.
- **AnguloGyro:** Ángulo de giro calculado únicamente con el giroscopio.
- **dt:** Tiempo transcurrido entre muestras.
- **AnguloAccel:** Ángulo de giro calculado únicamente con el acelerómetro.

Para calcular la ecuación 3.8 se debe determinar la constante α de los filtros.

$$\alpha = \frac{\tau}{\tau + dt} \tag{3.9}$$

La constante de tiempo τ depende de la frecuencia de corte (fc).

$$\tau = \frac{1}{2 * \pi * fc} \tag{3.10}$$

4 Diseño Electrónico

Un microcontrolador es un circuito integrado programable en cuya memoria sólo reside un programa que controla el funcionamiento de una tarea predefinida [10].

Está compuesto de varios bloques funcionales que cumplen una determinada tarea. Los tres principales son: Procesador o Unidad Central de Proceso (CPU), memoria y periféricos de entrada/salida.

4.1 Arduino

Arduino es una herramienta para el desarrollo de aplicaciones electrónicas de código abierto —Open Source—. Su programación se hace mediante un lenguaje propio, estando basado en el lenguaje de alto nivel Processing y en C, soportando muchas de las funciones de este [2].

4.1.1 Elección de la placa

Las necesidades del Arduino fueron incrementándose a la par que se desarrollaba el proyecto, de modo que la placa original fue sustituida por otra con mejor rendimiento.

Los requerimientos iniciales fueron:

- Tamaño: la placa debería tener un tamaño reducido, para poder montarlo en el prototipo.
- Entradas analógicas: el primer acelerómetro que se usó, daba las lecturas mediante una señal analógica. Esto se descartó cuando se incorporó una IMU a la boya.
- Posibilidad de conección I2C y SPI, para poder conectar todos los periféricos.

4.1.2 Arduino UNO

El Arduino Uno es una placa electrónica basada en el ATmega328. Cuenta con 14 pines entrada y salida (E/S) digitales (de los cuales 6 se pueden utilizar como salidas PWM), 6 entradas analógicas, un reloj de 16 Mhz, una conexión USB, un conector de alimentación y un puerto ICSP [2].

Caracteristicas principales:

- **Microcontrolador:** ATmega328
- **Tensión de alimentación:** 5V
- **Tensión de entrada (recomendado):** 7-12V
- **Tensión de entrada (límites):** 6-20V
- **E/S Digitales:** 14 (de las cuales 6 proporcionan salida PWM)
- **Entradas analógicas:** 6
- **Corriente DC por E/S:** 40 mA

- **Corriente DC de 3.3V:** 50 mA
- **Memoria:** Flash 32 KB
- **SRAM:** 2 KB
- **EEPROM:** 1 KB
- **Velocidad del reloj:** 16 MHz
- **Longitud:** 68,6 mm
- **Anchura:** 53,4 mm
- **Peso:** 25 g

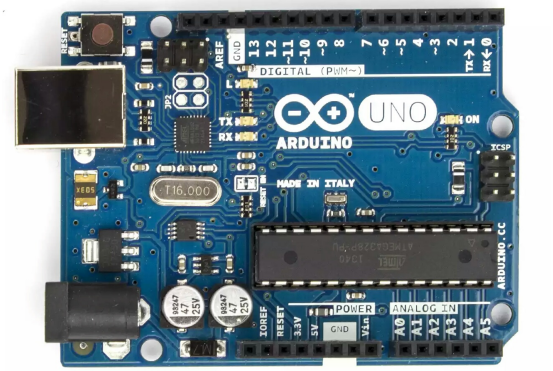

Figura 4.1: Placa Arduino Uno R3
Fuente: https://www.arduino.cc/

Con estas características y su reducido precio es una placa idónea para el desarrollo del proyecto. Sin embargo durante el desarrollo aparecieron limitaciones, principalmente por la velocidad y la memoria del microprocesador. Esto implicó un cambio a una placa con un microprocesador más potente.

4.1.3 Arduino Due

El Arduino Due es una placa electrónica basada en el Atmel SAM3X8E ARM Cortex-M3. Es la primera placa Arduino basado en un microcontrolador de 32 bits. Puede ejecutar 4 instrucciones de 8 bits en cada ciclo de reloj, incrementando sustancialmente la velocidad de procesamiento [2].

A diferencia de otras placas Arduino, la Due funciona a 3,3V. Para hacer totalmente compatible las conexiones de una placa a otra, se añadió un convertidor de niveles lógicos.

Características principales:

- **Microcontrolador:** AT91SAM3X8E
- **Tensión de alimentación:** 3,3V
- **Tensión de entrada (recomendado):** 7-12V
- **Tensión de entrada (límites):** 6-16V
- **E/S Digitales:** 54 (de las cuales 12 proporcionan salida PWM)
- **Entradas analógicas:** 12
- **Corriente DC por E/S:** 130 mA
- **Corriente DC de 3.3V:** 800 mA

- **Memoria:** Flash 512 KB
- **SRAM:** 96 KB
- **Velocidad del reloj:** 84 MHz
- **Longitud:** 101,52 mm
- **Anchura:** 53,3 mm
- **Peso:** 36 g

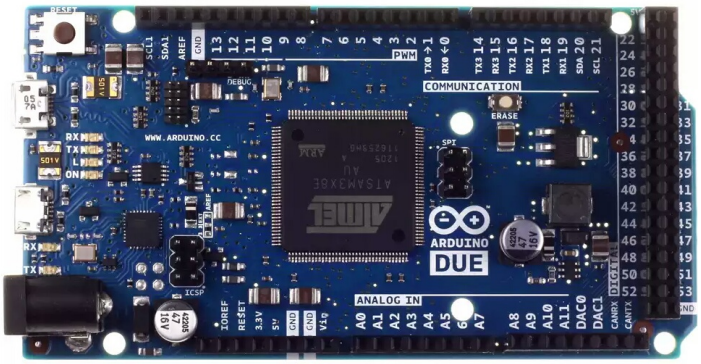

Figura 4.2: Placa Arduino DUE
Fuente: https://www.arduino.cc/

Debido al cambio de placa, se añadió al sistema una memoria EEPROM, ya que la Due no dispone de ella y es totalmente necesaria para alojar los valores de calibración de la boya.

4.2 Bus I^2C

El bus I^2C —Inter-Integrated Circuit— es un bus de comunicaciones en serie, su principal característica es que utiliza dos líneas para la comunicación (fig. 4.4).

- **SDA:** —System Clock— es la línea de los pulsos de reloj que sincronizan el sistema.
- **SCL:** —System Data— es la línea por la que se mueven los datos entre los dispositivos.

Los dispositivos en el bus I^2C pueden ser maestros o esclavos. El maestro siempre es que el inicia la comunicación, colocando la línea de reloj SCL en estado lógico bajo. Cuando los esclavos han respondido al maestro y termina la comunicación, el maestro vuelve a dejar la línea SCL en estado lógico alto, a la espera de una nueva comunicación.

Un esclavo no puede iniciar una transferencia, solamente un maestro tienen esta capacidad. Puede haber varios esclavos colgados del bus, aunque normalmente sólo hay un maestro [5].

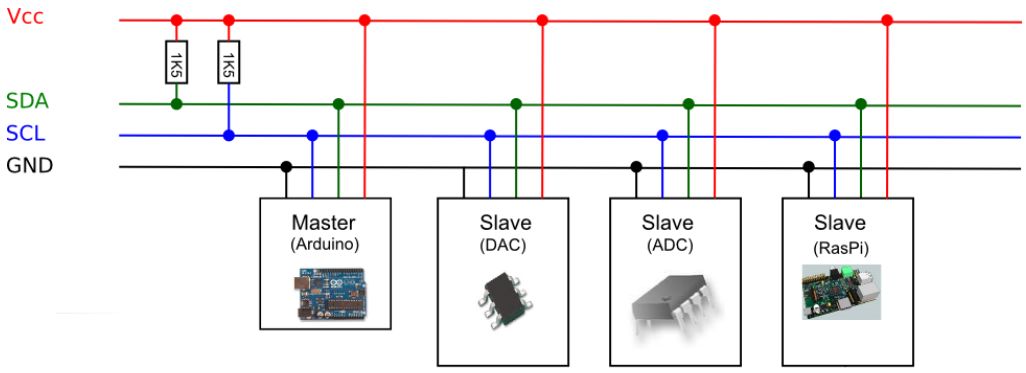

Figura 4.3: Esquema tipo de un bus I^2C con un maestro y tres esclavos
Fuente: http://openlabtools.eng.cam.ac.uk/

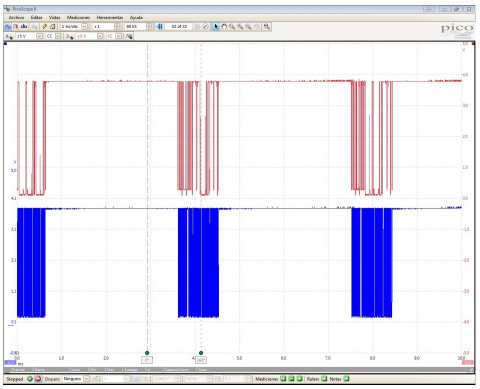

Figura 4.4: Señal del MPU6050 trasmitiendo por el bus I^2C

4.3 Diseño electrónico

4.3.1 Esquema en bloques

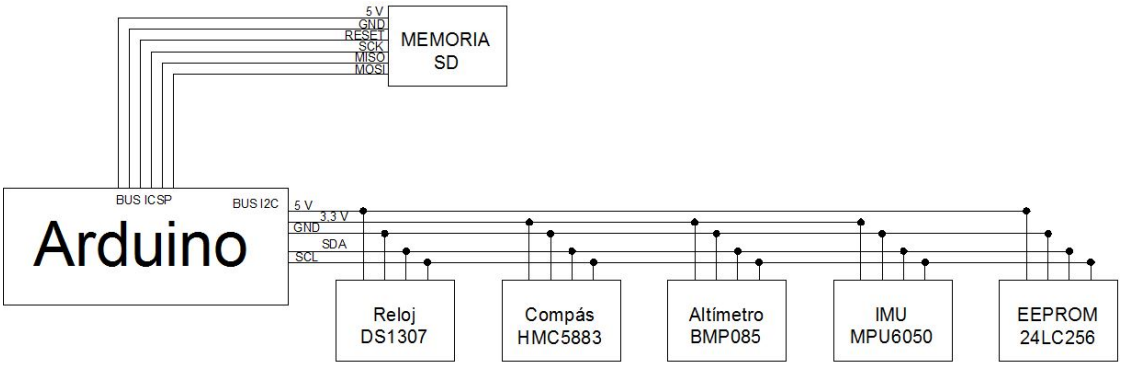

Figura 4.5: Esquema en bloques de los dispositivos usados en la boya

El esquema en bloques de la figura 4.5 es una representación resumida del esquema eléctrico. Cada bloque representa un componente. En el caso de la boya, estos bloques son:

- **Arduino:** Gestión de la comunicación entre los dispositivos, cálculo del desplazamiento producido por la ola.
- **Memoria SD:** Es el soporte físico donde se guardan la altura de la ola en cada toma de muestras.

- **Reloj:** Genera el nombre del archivo donde se guardan los datos y muestra la hora a la que fue tomada la muestra.
- **Compás:** Mide el campo magnético para el cálculo de la orientación respecto al norte.
- **Altímetro:** Mide la presión atmosférica para saber al altitud.
- **IMU:** Mide las aceleraciones y velocidades angulares de la boya.
- **EEPROM:** Almacenamiento de los valores de calibración y reposo.

4.3.2 Circuito impreso

Una placa de circuito impreso (PCB), es la superficie constituida por pistas de material conductor laminadas sobre una base no conductora. El circuito impreso se utiliza para conectar eléctricamente y sostener mecánicamente por medio de la base un conjunto de componentes electrónicos.

Dependiendo del material de que esté hecha la placa, podemos distinguir tres tipos fundamentales:

- Baquelita.
- Fibra de vidrio.
- Teflón.

Dependiendo del proceso de obtención de las pistas, se dividen en dos tipos más:

- Normal.
- Fotosensible.

La placa normal es aquella que se dibuja directamente la pista sobre el cobre. Pudiéndose dibujar con rotulador indeleble, o bien mediante pegatinas adecuadas.

La placa fotosensilble tiene un barniz que es sensible a la luz, que se impresiona mediante una insoladora o cualquier otro foco luminoso adecuado.

Para la exposición, se prepara una transparencia de las pistas. Tras la exposición, se introduce la placa en un líquido revelador que destruirá el barniz que no forma parte de las pistas, de forma que el restante actúa de protector contra la corrosión.

Una vez que se tiene la placa con las pistas impresionadas, el siguiente paso es el revelado (placas fotosensibles) o bien el atacado con los productos químicos adecuados.

Revelado: Consiste en eliminar el barniz fotosensible en las partes en que no es necesario, quedando el cobre al descubierto para que sea atacado por el ácido corrosivo en el siguiente proceso.

Atacado: Una vez revelada la placa, se sumerge en una disolución de cloruro férrico. Cuando el ácido elimina el cobre que no pertenece a las pistas, se pasa por agua para limpiar y taladrar (figuras 4.10 y 4.11).

4.3.2.1 Diseño y fabricación

Para el diseño de las placas se usó AutoCAD (figuras 4.6, 4.7, 4.8 y 4.9); debido a que las placas no planteaban una elevada dificultad no existió la necesidad de utilizar programas específicos.

El montaje de los sensores se hizo en forma de columna, este diseño favorece la posibilidad de colocarlo en el centro de la boya y evitar desplazar el centro de gravedad hacía los extremos, quedando el sistema en aguas iguales.

Debido a los diferente niveles lógicos ya mencionados anteriormente, los sensores están conectados a una placa intermediaria con el Arduino donde se encuentra el RTC y el conversor de nivel.

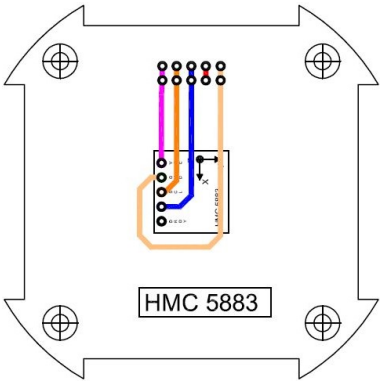

Figura 4.6: Diseño de la placa correspondiente al HMC5883

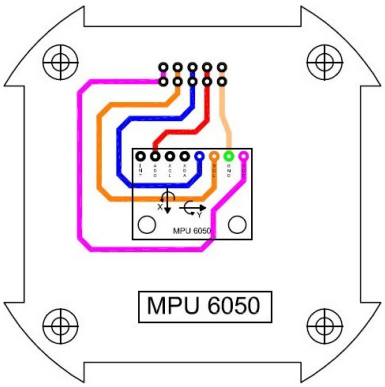

Figura 4.7: Diseño de la placa correspondiente al MPU6050

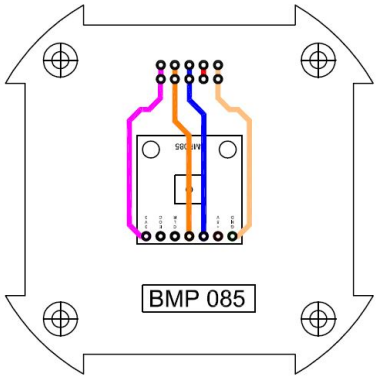

Figura 4.8: Diseño de la placa correspondiente al BMP085

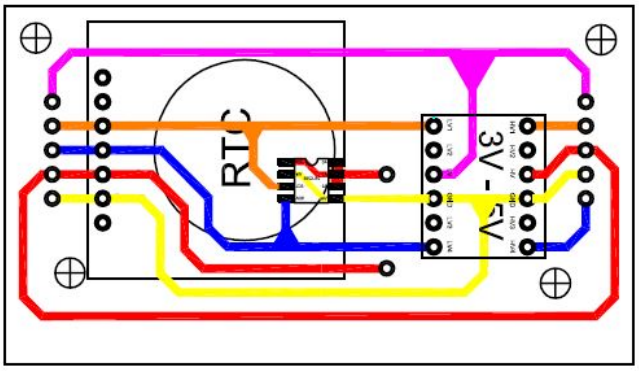

Figura 4.9: Diseño de la placa correspondiente al RTC y EEPROM

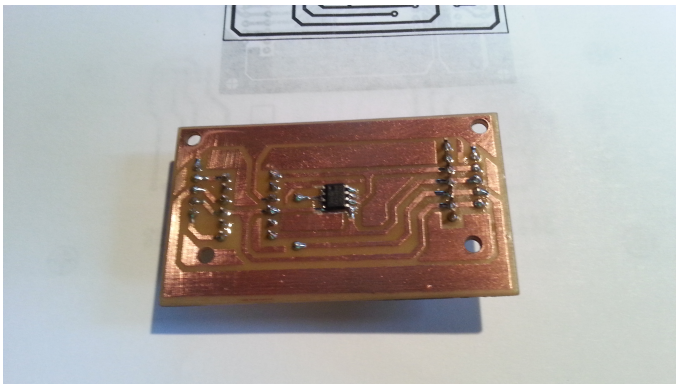

Figura 4.10: Vista de la memoria EEPROM SMD

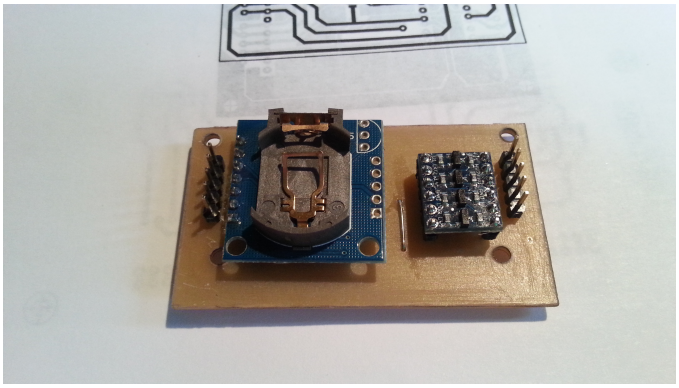

Figura 4.11: Vista del reloj y del convertidor de niveles

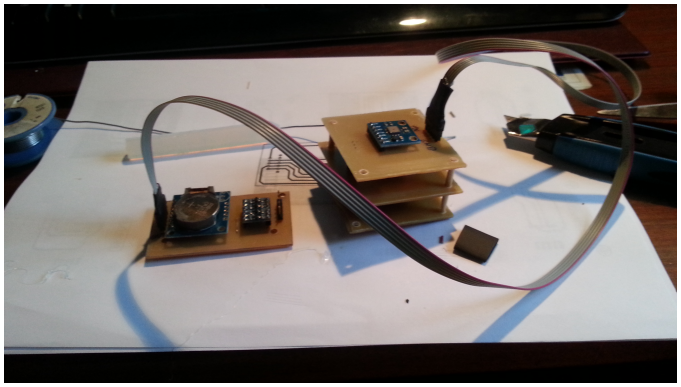

Figura 4.12: Interconexión entre las placas

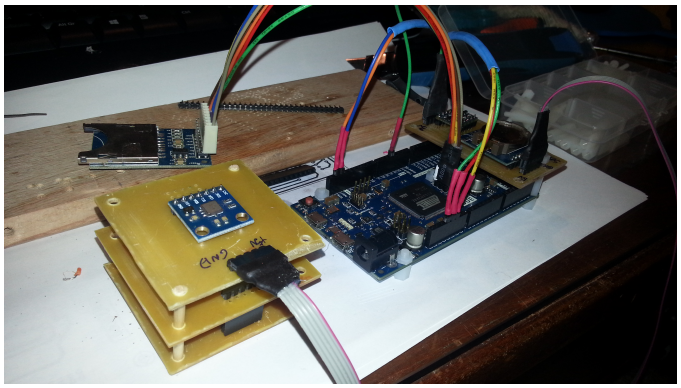

Figura 4.13: Vista de todo el conjunto montado y conectado

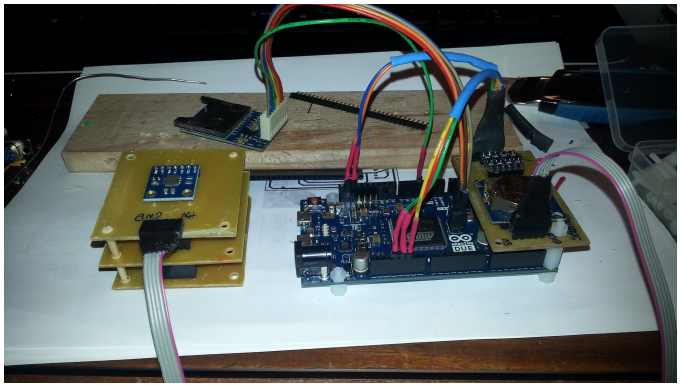

Figura 4.14: Vista de todo el conjunto montado y conectado

5 Software

Para la ejecución del software, se ha utilizado el entorno de programación Arduino IDE. Esta herramienta está disponible en la página web de Arduino. El Arduino IDE no dispone de simulador ni de depurador, por lo que se requiere del hardware para poder probar el código.

5.1 Software Arduino IDE

5.1.1 Instalación Software Arduino IDE

Para conectar Arduino DUE al ordenador se necesita de un cable micro-USB, tipo A-B, el cual proporcionará voltaje a la placa y permite su programación. La programación de Arduino DUE puede ser a través del puerto "Programming" o del puerto "Native", por recomendación del fabricante la programación se realizará por el puerto "Programming".

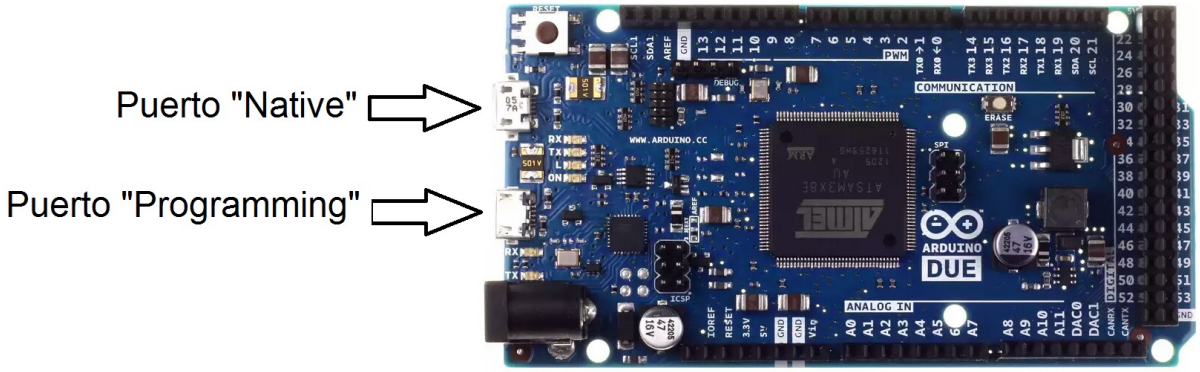

Figura 5.1: Puertos Arduino Due
Fuente: https://www.arduino.cc/

Desde *www.arduino.cc/en/Main/Software*, se puede descargar la última version del software de Arduino, una vez finalizada la descarga de los archivos, se realiza la conexión entre Arduino y el ordenador a través del cable micro USB.

La instalación del controlador de Arduino se realiza de manera manual con la finalidad de identificar el puerto asignado por el ordenador. El proceso bajo Windows es el que se indica a continuación:

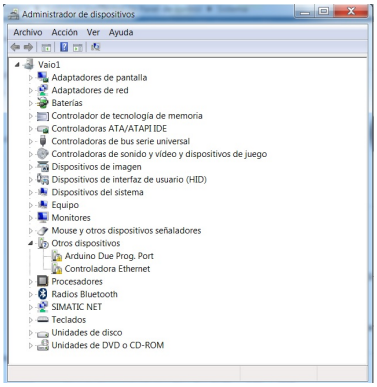

Figura 5.2: Administración de dispositivos

Seleccionar "Arduino Due Prog. Port" y con "click" derecho del "ratón" y seleccionar la opción "Actualizar software de controlador".

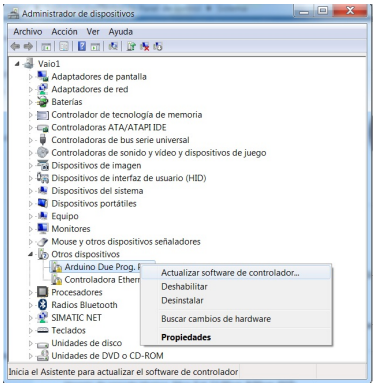

Figura 5.3: Actualización del software del controlador

Elegir la opción de "Instalar desde una ubicación específica", seleccionar la ubicación donde se guardaron los archivos de instalación de Arduino, seleccionar la carpeta con el nombre de "drivers" y continuar con la instalación del controlador.

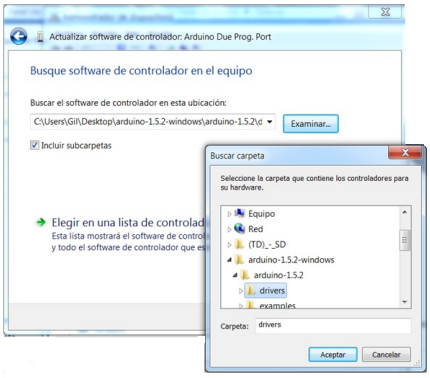

Figura 5.4: Búsqueda de drivers

Una vez concluida la instalación del controlador, se muestra la información actualizada del hardware, así como el puerto al que está conectado.

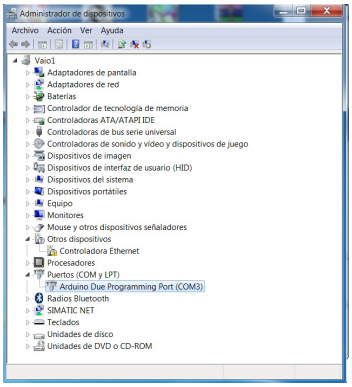

Figura 5.5: Divers actualizados

Para la configuración del software de Arduino, se ejecuta el ícono de aplicación "Arduino.exe", el cual está incluido en la carpeta de archivos descargados. Al ejecutarse la aplicación de Arduino se muestra la siguiente ventana:

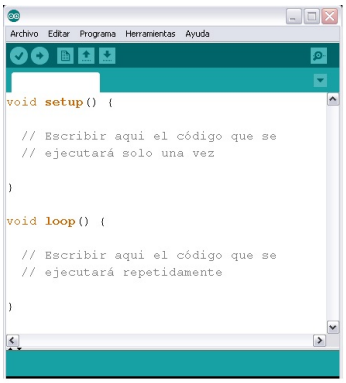

Figura 5.6: Sketch base

En el menú de Herramientas → Puerto Serie, especificar el puerto al cual está conectado Arduino.

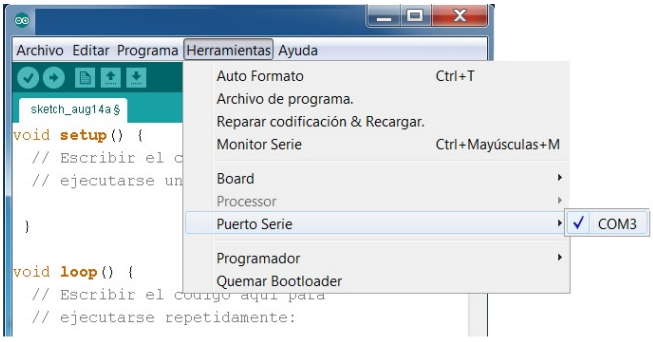

Figura 5.7: Selección puerto de la placa Arduino

Especificar el modelo de la plataforma Arduino en el menú de Herramientas → Placa, el cual corresponde a Arduino DUE y el puerto micro-USB mediante el cual se programará ("Arduino DUE Programming Port").

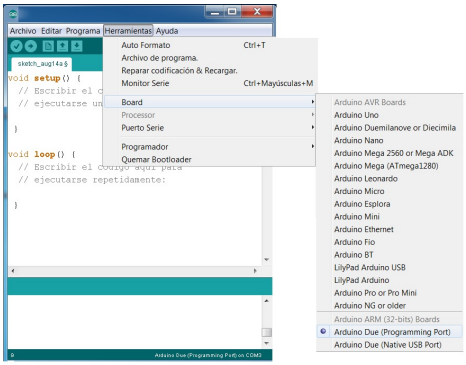

Figura 5.8: Selección de la placa conectada al pc

5.1.2 Estructura básica de un programa

Un programa en Arduino, se le conoce como "sketch" y se divide en tres partes fundamentales:

- Estructura.
- Valores (variables o constantes).
- Funciones

La estructura básica de sketch es bastante simple, divide la ejecución en dos partes: "setup" y "loop" (fig. 5.6). El código delimitado por el "setup" se ejecuta sólo una vez al inicio del programa. Aquí se definen constantes, variables y se configuran los diversos pines del Arduino. En cambio la sección "loop" se ejecuta ciclicamente hasta que se produzca un reset.

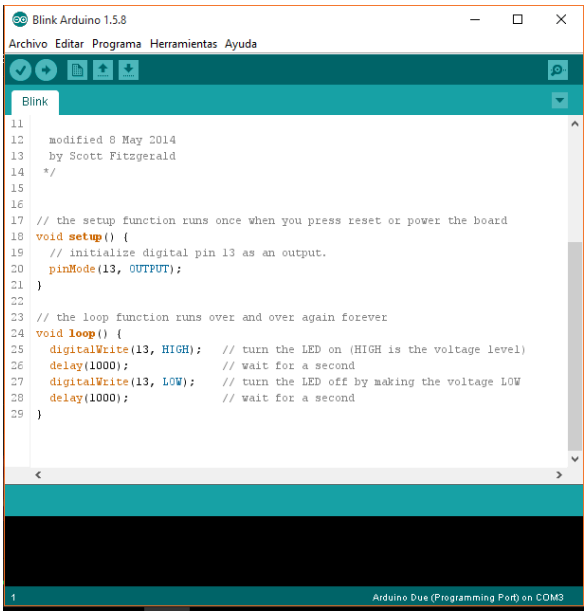

Figura 5.9: Ejemplo de sketch
Fuente: https://www.arduino.cc/

5.2 Código Fuente

El diagrama de flujo —representación gráfica del algoritmo— de la boya (fig. 5.10) representa los pasos que sigue el Arduino para poder calcular la elongación ola, el código fuente completo está disponible en el Anexo A.

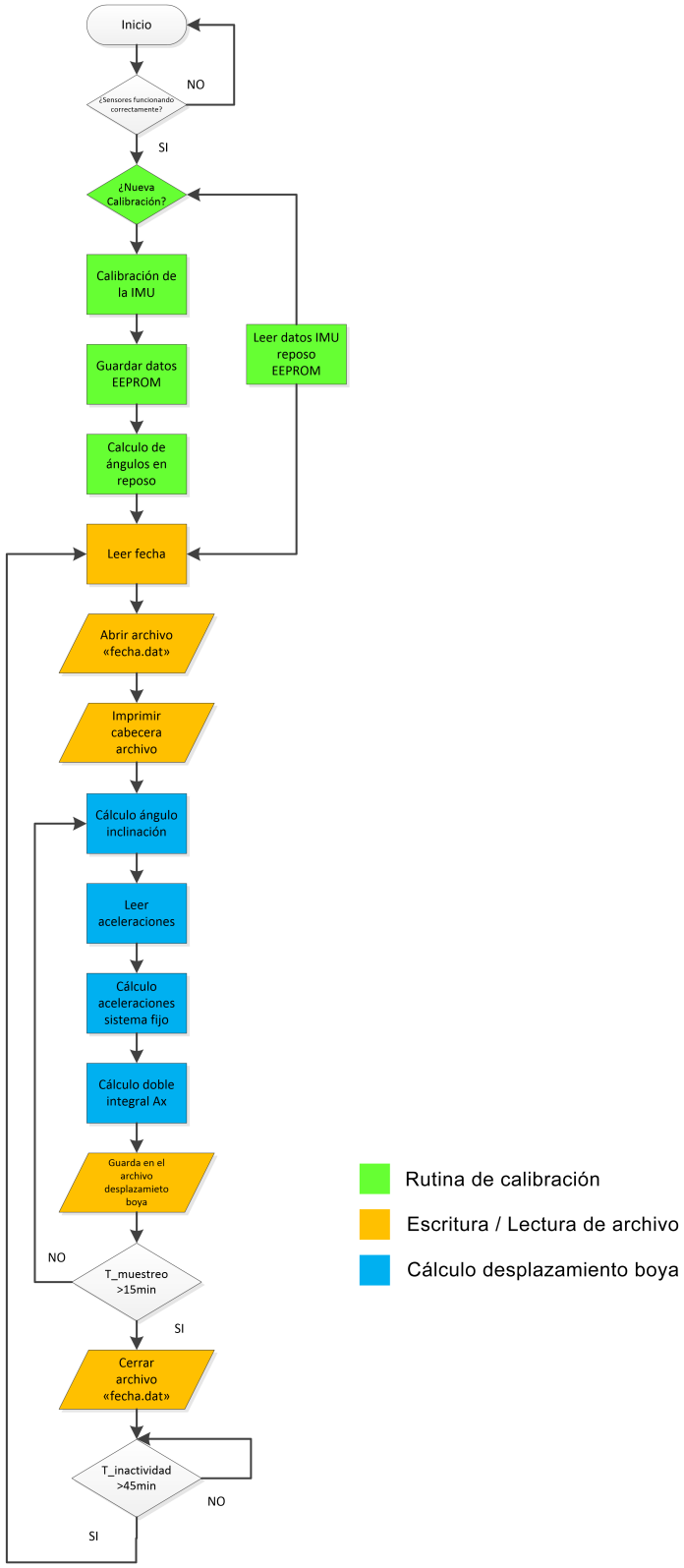

Figura 5.10: Diagrama de Flujo del funcionamiento de la boya

5.2.1 Rutinas relevantes

5.2.1.1 Rutina de Calibración y uso de la EEPROM

Cuando arranca el sistema, se comprueba el correcto funcionamiento de los sensores (fig. 5.11), en caso de que aparezca un error, la ejecución del programa se para y la boya no guarda ningún dato.

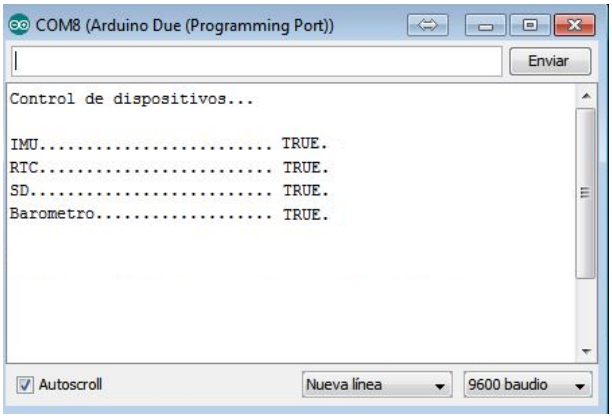

Figura 5.11: Comprobación de los sensores

Lo siguiente en ejecutarse es el menú de calibración, como se puede ver en la figura 5.12 tiene dos opciones:

- **Realizar una nueva calibración**: La rutina ejecuta un bucle anidado 2.000 veces (fig. 5.13) por eje, así se consigue tener un valor de offset bajo, calculando los ángulos en reposo y el valor de la gravedad.
- **Tomar los datos de calibración de la memoria EEPROM**: Ésta opción es la predefinida del sistema, en caso de que la boya se reinicie en el agua, tendrá una referencia correcta para calcular los datos del reposo con los valores de la última calibración.

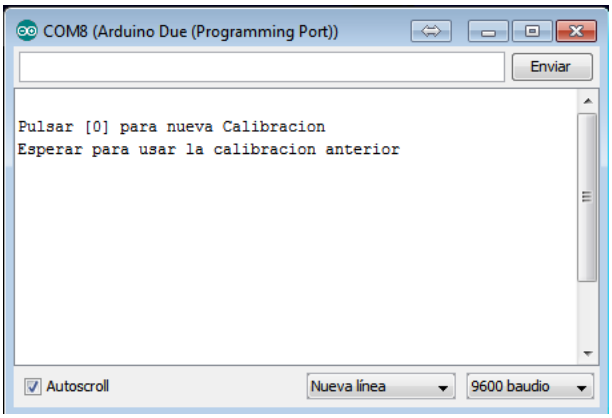

Figura 5.12: Menú de calibración mostrado en el arranque del sistema

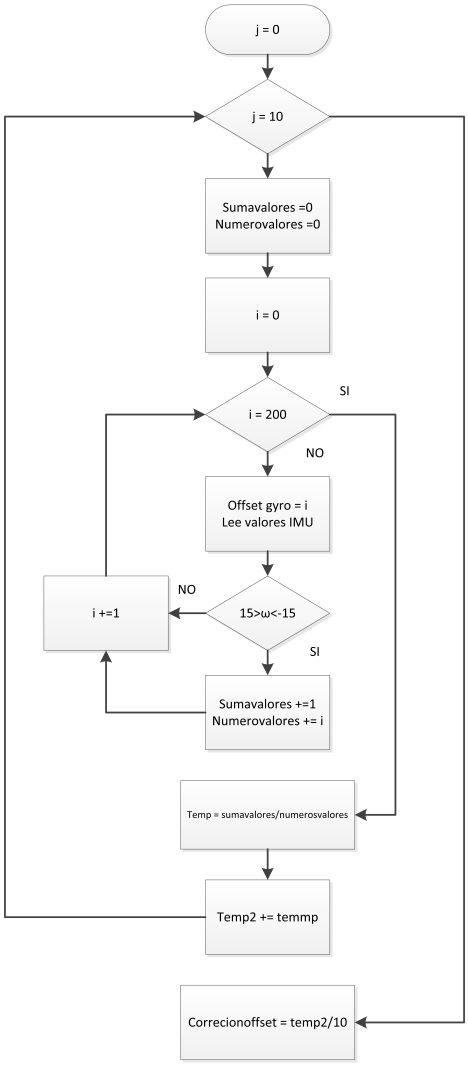

Figura 5.13: Diagrama de flujo de la rutina de calibración

5.2.2 Filtrado de las aceleraciones

Un *filtro digital* es un sistema que, dependiendo de las variaciones de las señales de entrada en el tiempo y amplitud, se realiza un procesamiento matemático, generalmente mediante el uso de la transformada rápida de Fourier obteniéndose en la salida el resultado [4].

Un *filtro paso bajo* corresponde a un filtro caracterizado por permitir el paso de frecuencias bajas y atenuar el paso de las altas. La función de transferencia de un filtro pasa bajo de primer orden corresponde a:

$$H(s) = k\frac{1}{1+\frac{s}{wc}} \tag{5.1}$$

Con la función de transferencia de la figura 5.1 y según la frecuencia de corte obtendremos el comportamiento esperado del filtro. El diagrama de Bode de un filtro paso bajo de primer orden es:

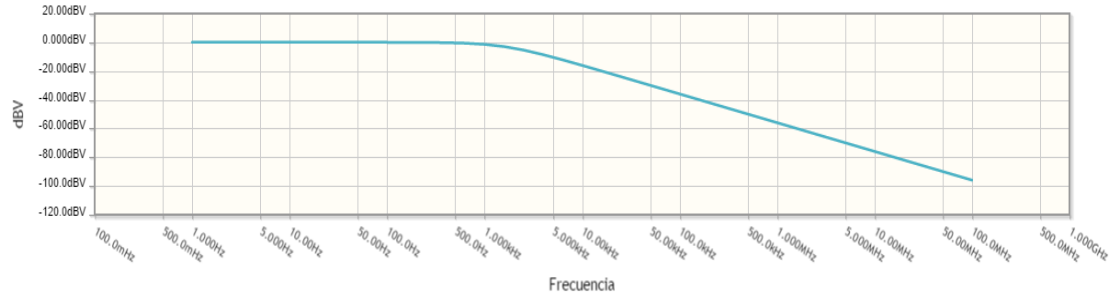

Figura 5.14: Representación de la ganancia en un filtro paso bajo

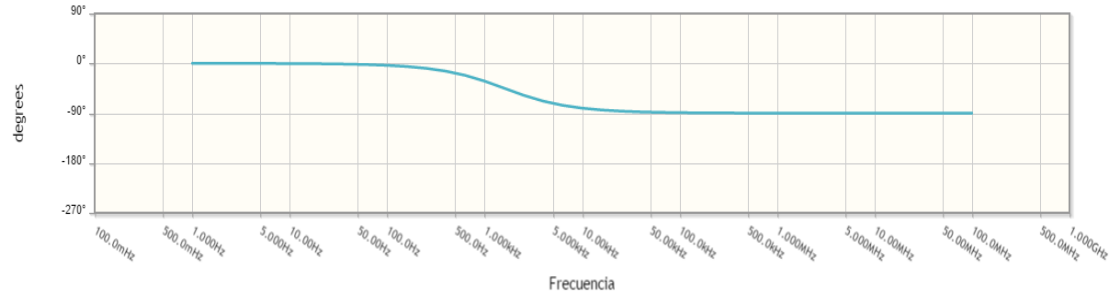

Figura 5.15: Representación del desfase en un filtro paso bajo

El uso de estos filtros es necesario debido que la frecuencia de las olas son bajas (fig. 5.16). Ésto hace imprescindible eliminar toda señal producida por encima de estos valores, evitando errores de cálculos [8].

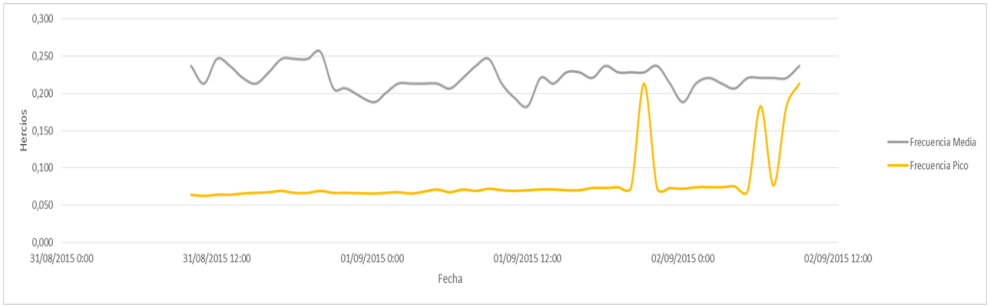

Figura 5.16: Frecuencia del oleaje
Transmitido por la red de boyas de Puertos del Estado en el sur de la isla de Tenerife desde el 31/08/2015 al 02/09/2015
Fuente: http//portus.puertos.es

5.2.2.1 Pseudocódigo filtro digital

Para poder implementar el filtro en Arduino se requiere una función matemática que pueda ser programada en el microcontrolador. El pseudocódigo de la función es el siguiente [3].

function pasobajo(real[0..n] x, real dt, real RC)
var real [0..n]
var real α:= RC/(RC + dt)
y[0] := x[0]
for i **from** 1 **to** n
    y[i] := α * x[i] + (1-α) * y[i-1]
return y

5.2.3 Cálculo de la elongación

Este apartado aunque técnicamente no sea un rutina si que la parte fundamental del funcionamiento de la boya. Se toma como punto base que la boya esta calibrada y ya esta en el agua tomando valores de las aceleraciones, en la figura 5.17 se puede ver el diagrama de flujo.

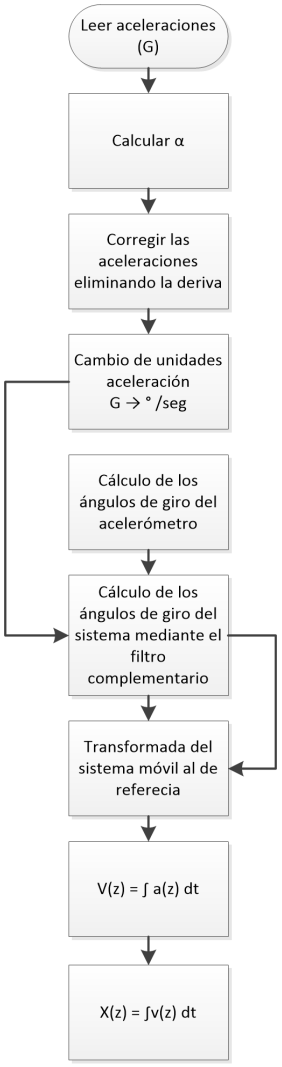

Figura 5.17: Diagrama de flujo del cálculo de la elongación

En síntesis el cálculo de la elongación es relativamente fácil, una vez que se tiene los valores de las aceleraciones, se calculan los ángulos de giro del sistema. Las aceleraciones están en el sistema en movimiento por lo que mediante la matriz de rotación las trasladamos al sistema de referencia, apuntando en todo momento a_x en sentido del movimiento del sistema. Ya por último queda calcular la doble integración para llegar a la elongación que es de lo que se trata.

5.2.4 Fichero resultante

El sistema realiza una toma de muestras cada hora. La duración de la adquisición es de 15 minutos y los datos junto con los cálculos resultantes son volcados en una tarjeta de memoria SD. Para evitar que se reescriban archivos, el nombre de cada archivo se corresponde con la fecha y hora de cuando se realiza la muestra.

En la figura 5.18 se puede ver que aparte de todos los datos que arrojan la IMU y el magnetómetro, también se guarda la temperatura ambiente, la orientación de la boya respecto al norte y la localización de la misma, que en el caso de tener mas de una boya nos sirve para identificar cada fichero.

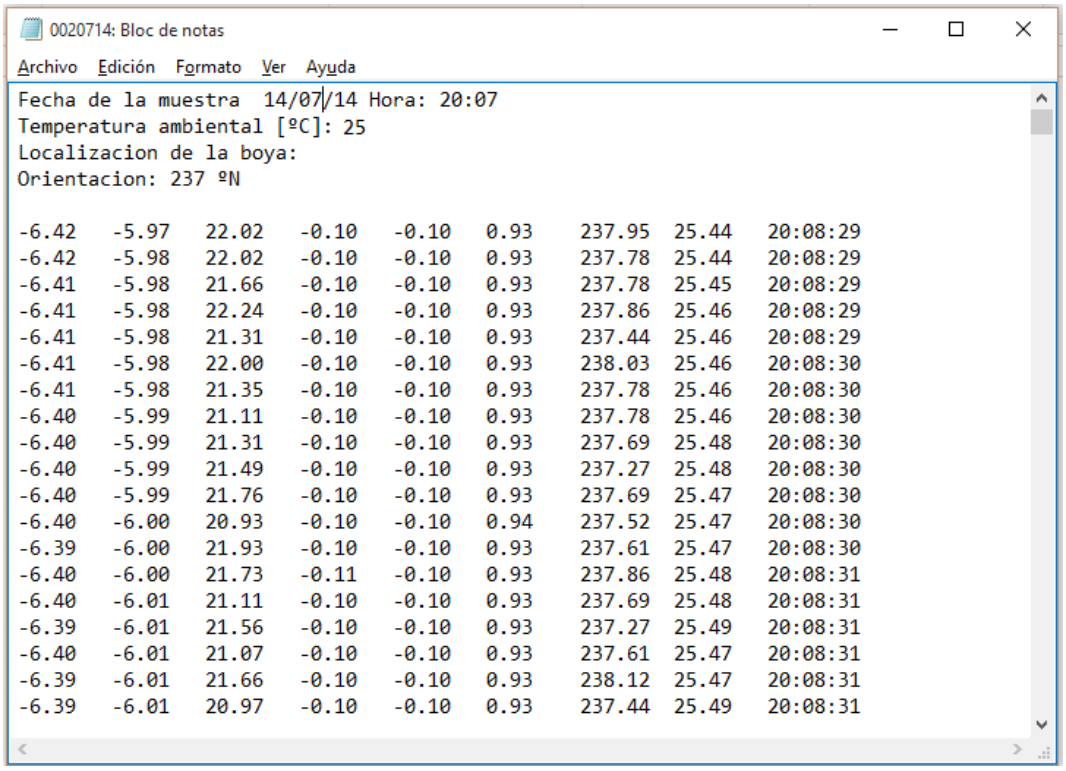

Figura 5.18: Archivo tipo generado por la boya

6 Conclusiones

El objetivo del trabajo era el desarrollo e implementación de una boya oceanografía de bajo coste, la boya debía tener un sistema de adquisición de datos. Una vez recuperado estos datos, el cálculo y análisis se haría en tierra.

Esta línea original cambio un poco gracias a la capacidad de la placa Arduino utilizada: la boya es capaz de hacer todos los cálculos y solo devolver el valor de la elongación, suprimiendo la necesidad de un tratamiento externo de los datos.

Por motivos de plazos no se pudo fabricar un modelo para probar todo el conjunto en el agua. Debido a esto no se pudo concluir el ajuste de la electrónica debido que su comportamiento en agua es diferente a tierra.

Este tipo de boyas, permiten la creación de sistemas de vigilancia que pueden comprender desde la simple motorización del oleaje en puntos determinados, como la vigilancia de espacios donde entran en conflicto la navegación recreativa con el disfrute de espacios dedicados al baño mediante el análisis varios puntos estratégicos, y determinado si las perturbaciones del agua son producidas por el oleaje natural o por el paso cercano de una embarcación deportiva.

El sistema aporta las siguientes ventajas:

- Menor impacto ambiental por su reducido tamaño.
- Proceso de detección y corrección de errores mas efectivo, debido a que tanto empresas como particulares puedan aportar soluciones a nuevas características.
- Es totalmente moldeable, cada usuario puede adaptarlo a sus necesidades.
- Tiene la posibilidad de extrapolarse a otros ambientes, pudiendo ser el caso de las mediciones atmosféricas en tierra.
- Menos coste de mantenimiento.

Por contra tiene las siguientes inconvenientes:

- Su reducido tamaño la hace vulnerable frente a los daños producidos por agentes externos.
- La dificultad para la recopilación de los datos obtenidos.
- La reducción de prestaciones con las existentes en el mercado.

Por otra parte durante el desarrollo del proyecto, se generaron ideas nuevas y se plantearon mejoras.

- Desarrollo de un modelo a escala real donde se pueda integrar todos los dispositivos y hacer pruebas en ambientes reales.
- A parte del cálculo de la elongación de la ola mediante unidades inerciales, sería factible la incorporación de una unidad GPS —Global Position System— para poder calcular el movimiento de la boya mediante DGPS —Differential Global Positioning System—, lo que incrementaría la resolución del cálculo y el futuro control de una red de boyas.

- Incorporación de un sistema de trasmisión boya-tierra. La idea era montar un enlace RF —Radio Frecuencia— en 2,4Ghz para poder hacer captura de los datos en tiempo real, liberando al Arduino de todo el cálculo, pudiendo tomar más muestras por segundo incrementando así la resolución del perfil de la ola.
- Sustitución de la plataforma Arduino por una plataforma del tipo *Raspberry Pi*, *Intel Galileo*, *pcduino* etc., incrementando significativamente su capacidad de cálculo y control sobre la boya.

Anexos

A Anexo código fuente

```
#include "DpI2CEEPROM2_Wire1.h" // libreria EEPROM
#include "HMC58X3-Wire1.h"
#include "helper_3dmath.h"
#include "MPU6050-Wire1.h"
#include "DS1307-Wire1.h"
#include "I2Cdev-Wire1.h"
#include "Adafruit_BMP085_Wire1.h"
#include "SPI.h"
#include "Wire.h"
#include "math.h" // libreria para el calculo matematico
#include "String.h" //Libreria para convertir String en char
#include "SD.h"
//#include "Kalman.h" // Source: https://github.com/TKJElectronics/KalmanFilter

// ******************************** CONSTANTES DEL PROGRAMA ********************************
const int chipSelect = 4; // cs del bus SPI donde se comunica la SD
const unsigned long Inactividad = 2700000; // Pausa de 45min x 60seg x 1000ms
const unsigned long Actividad = 900000; // tiempo de 15 min x 60 seg x 1000ms
const float gyro_full_scale = 16; //(65536/4000)-> 2^16 bits  /  (+- 2000 grados/s)   Valor
del fondo de escala  del gyro
const float accel_full_scale = 8192; // (65536/8)-> 2^16 bits  /  (+- 4g)   Valor del fondo
de escala  del acelerometro
const int sizeDatos = 22096;
const float fc = 1; //Frecuencia de corte en Hz de los filtros
const float pi = 3.14159265359;
// ******************************** VARIABLES DEL PROGRAMA ********************************
int tmp = 0; // variable temporal para diversos calculos en el programa.
float tmp_float =0;
int16_t ax, ay, az; // Definicion de las variables de la aceleracion
int16_t axf, ayf, azf; // Definicion de las variables de la aceleracion despues de pasar por
el filtro paso bajo
int16_t gx, gy, gz; // Definicion de las variables del gyro
String tempArchivo;
char Archivo[12];
float Altitud;
unsigned long tiempo_muestreo;
String Cadena;
unsigned int temporal_tiempo;
unsigned int tiempo_ejecucion;
double dt = 0;
double  temp_dt = 0;
long Pressure = 0, Altitude = 0, Temperature = 0;
uint16_t year;
uint8_t month, day, dow, hours, minutes, seconds;
float Pitch = 0; // variable de giro en el eje X
float Roll = 0; //variable de giro en el eje Y
float Yaw = 0; //variable de giro en el eje Z
float H_barometro; // variable que almacena la variable la altura del barometro
float axc, ayc, azc; // Definición de las variables de las aceleraciones corregidas
float gxc, gyc, gzc; // Definición de las variables del gyro corregidas
float gravedad;
boolean menu = false; // controla si ha entrado en el menu de calibracion
float RC, AlfaLp, AlfaHp; // Variables para la constante de tiempo para los filtros paso
alto y bajo
int DerivaX, DerivaY; // Variable para el calculo de la deriva
double compAngleX, compAngleY, compAngleZ; // Calculated angle using a complementary filter
boolean TarjetaSD =0;

//******************** DEFINICION DE OBJETOS ****************************************
```

```
MPU6050 accelgyro(0x69); // <-- use for AD0 high
HMC58X3 magn;  // crea un objeto HMC58x3
Adafruit_BMP085 bmp; // crea un objeto  BMP085
File ArchivoSD;  // crea el objeto ArchivoSD
DS1307 rtc;
//Kalman kalmanX, kalmanY, kalmanZ; // Create the Kalman instances
DpI2CEEPROM2 EEPROM(0x57);

void setup() {
    //rtc.setDateTime24(2014, 11, 25, 22, 21, 0); //SOLO DESCOMENTAR PARA CONFIGURAR LA HORA
    DEL RTC           NO TOCAR
    Wire1.begin();
    Serial.begin(9600);
    magn.init(false);
    magn.calibrate(1, 32); // calibración de la brújula
    magn.setMode(3);
    bmp.begin(); //  inicialización del altímetro
    SPI.setClockDivider(chipSelect,1); // aumenta la velocidad del bus
    spi
    rtc.initialize();
// ************************ CONFIGURACIÓN DE LA IMU ********************************
    accelgyro.initialize();
    accelgyro.setFullScaleAccelRange(MPU6050_ACCEL_FS_4); // configura el fondo de escala
    del acelerómetro.
    accelgyro.setFullScaleGyroRange(MPU6050_GYRO_FS_2000); // configura  el fondo de escala
    del gyro
    //accelgyro.setDLPFMode(MPU6050_DLPF_BW_98); //configura el filtro paso bajo
// ************* NO INICIALIZA SI NO ESTAN TODOS LOS SENSORES CONECTADOS
**********************
  Serial.println("Control de dispositivos...");
  Serial.println("");
  delay(600);
  Serial.print("IMU.......................... ");
   if(accelgyro.testConnection() == 0)
   {
     Serial.println("FALSE.");
   }
   else
   {
     Serial.println("TRUE.");
     }
  delay(600);
  Serial.print("RTC.......................... ");
   if(rtc.testConnection() == 0)
   {
     Serial.println("FALSE.");
        }
   else
   {
     Serial.println("TRUE.");
          }
  delay(600);
  Serial.print("SD........................... ");
   if(SD.begin(chipSelect) == 0)
   {
     Serial.println("FALSE.");
     TarjetaSD = 0;
   }
   else
```

```
   {
     Serial.println("TRUE.");
     TarjetaSD = 1;
     }
  delay(600);
  Serial.print("Barometro.................. ");
   if(bmp.begin() == 0)
   {
     Serial.println("FALSE.");
   }
   else
   {
     Serial.println("TRUE.");
     }
  Serial.println("");
  delay(600);
  if (Sensores_conectados() == 0)
   {
     Serial.println("Comprobar los dispositivos y reiniciar el sistema.");
    }
   else
   {
     Serial.println("Dispositivos funcionando correctamente.");
     }
   Serial.println(" ");
  while (!Sensores_conectados());{
           }
//*************************** CALIBRACIÓN DE LA IMU ***********************************

    Serial.println("");
    Serial.println("Pulsar [0] para nueva Calibracion");
    Serial.println("Esperar para usar la calibracion anterior");
    Serial.flush(); //vacia la serial
    for (unsigned long i = 0; i <= 10000000; i++)
    {
      if ((Serial.available() > 0)&&(Serial.read() == 48))// Entra si el bufer esta lleno y
      se lee el 0 por el puerto serial.
        {
          menu = true;
          Serial.println("Calibrando.....................................");
          delay(100);
          tmp = calibracionXGyro();
          EEPROM.writeInt(0, tmp); // calibracion gx
          tmp = calibracionYGyro();
          EEPROM.writeInt(1000, tmp);//Calibracion gy
          tmp = calibracionZGyro();
          EEPROM.writeInt(2000, tmp);//Calibracion gz
          tmp = 833;
          EEPROM.writeInt(3000, tmp); //Calibracion az
            accelgyro.setXGyroOffset(EEPROM.readInt(0)); // valores eje x
            accelgyro.setYGyroOffset(EEPROM.readInt(1000)); // valores eje y
            accelgyro.setZGyroOffset(EEPROM.readInt(2000));  // valores eje z
            accelgyro.setZAccelOffset(EEPROM.readInt(3000));  // valores eje z
          accelgyro.getAcceleration(&ax, &ay, &az);
          EEPROM.writeInt(4000, -ax); // DerivaX
          EEPROM.writeInt(5000, -ay); // DerivaY
          tmp_float= (az/accel_full_scale);
          EEPROM.writeFloat(6000,tmp_float); // Gravedad
```

```
          break; // sale del bucle.
        }
      }
    if ( menu ==false)
    {
      accelgyro.setXGyroOffset(EEPROM.readInt(0)); // valores eje x
      accelgyro.setYGyroOffset(EEPROM.readInt(1000)); // valores eje y
      accelgyro.setZGyroOffset(EEPROM.readInt(2000));  // valores eje z
      accelgyro.setZAccelOffset(EEPROM.readInt(3000));  // valores eje z
    }
    accelgyro.getAcceleration(&ax, &ay, &az);
    DerivaX = (EEPROM.readInt(4000));
    DerivaY = (EEPROM.readInt(5000));
    gravedad = (EEPROM.readFloat(6000));
    temp_dt = micros();
    compAngleX = (atan2((ax+DerivaX),az))* RAD_TO_DEG;
    compAngleY =(atan2((ay+DerivaY),az))* RAD_TO_DEG; //  Calculo de los angulos en la fase de
    calibración

    Serial.print(ax);Serial.print(" "); Serial.println(ay);
    Serial.print(DerivaX);Serial.print(" "); Serial.print(DerivaY);Serial.print(" ");
    Serial.print(gravedad);Serial.print(" "); Serial.print(compAngleX );Serial.print(" ");
    Serial.println(compAngleY);
    Serial.println(" ");
    Serial.println("Fin de la calibracion");
    //*********************************************************************************
}

void loop() {
 //**************** IMPRESIÓN DE LA CABEZERA DEL ARCHIVO   ***********************************
    tempArchivo = Nombre_Fecha(); // Carga en la variable tempArchivo la fecha en forma de
    String
    tempArchivo += ".bo";
    tempArchivo.toCharArray(Archivo,12);
    abrir_archivo();
    ArchivoSD.println(Fecha_muestra()); // Imprime en el archivo la cabezera
    ArchivoSD.print("Temperatura ambiental [ºC]: "); ArchivoSD.println(bmp.readTemperature());
    ArchivoSD.println("Localizacion de la boya: ");
    ArchivoSD.print("Orientacion: "); ArchivoSD.print(Orientacion()); ArchivoSD.print("ºN");
    ArchivoSD.println();
    ArchivoSD.println();
    //ArchivoSD.close(); // cerramos el archivo donde se almacenan los datos*/
//*********************************************************************************
    //ArchivoSD = SD.open( "temp.txt", FILE_WRITE);
    tiempo_muestreo = 0;
    float VelZ_girada = 0;
    float AlturaZ_girado = 0;
    temporal_tiempo = hora_segundos();
    while(tiempo_muestreo <= sizeDatos){
    // *********** LECTURA DE LOS VALORES DEL GYRO Y DEL ACELEROMETRO ***********
       accelgyro.getMotion6(&ax, &ay, &az, &gx, &gy, &gz); // lectura de los valores de las
       aceleraciones y velocidades angulares
       dt = (double)(micros() - temp_dt) / 1000000; // Calcula delta time
       temp_dt = micros();
//****************** CALCULO CONSTANTE DE TIEMPO DE LOS FILTROS *************
       RC = 1/(2*pi*fc);
       AlfaLp = dt/(RC+dt);
       AlfaHp = RC/(RC+dt);
// ************* CALCULO DEL FILTRO PASO BAJO PARA LAS ACELERACIONES**********
```

```
     axf = axf + (AlfaLp*((ax+DerivaX)-axf)); // Aceleracion X filtrada
     ayf = ayf + (AlfaLp*((ay+DerivaY)-ayf)); // Aceleracion Y filtrada
     azf = azf + (AlfaLp*(az-azf)); // Aceleracion Z filtrada
//************************ CORRECCION LECTURA DEL GYRO *********************
     gxc = (gx/gyro_full_scale); // conversion de la aceleración en el eje X en º/seg
     gyc = (gy/gyro_full_scale); // conversion de la aceleración en el eje Y en º/seg
     gzc = (gz/gyro_full_scale); // conversion de la aceleración en el eje Z en º/seg
 // ****************** CORRECIÓN LECTURA DEL ACELEROMETRO ******************
    axc = ((ax+DerivaX)/accel_full_scale); // conversion de la aceleración en el eje X en G
    ayc = ((ay+DerivaY)/accel_full_scale); // conversion de la aceleración en el eje Y en G
    azc = (az/accel_full_scale); // conversion de la aceleración en el eje Z en G
   // **** CALCULO DEL ANGULO DE GIRO SEGUN EL ACELEROMETRO ******************
    Roll = (atan2(ayc,azc))* RAD_TO_DEG;
    Pitch = (atan2(-axc,sqrt(sq(ayc)+sq(azc))))* RAD_TO_DEG;
    //Serial.print(Roll); Serial.print(" "); Serial.println(Pitch);
    compAngleX = AlfaHp * (compAngleX + gxc * dt) + (1-AlfaHp) * Roll; // Calculo del angulo
    usando filtro complementario
    compAngleX = AlfaHp * (compAngleY + gyc * dt) + (1-AlfaHp) * Pitch;
    float Pitch_RAD = Pitch * DEG_TO_RAD;
    float Roll_RAD = Roll * DEG_TO_RAD;

    float AcelZ_girada
    =((((-sin(Pitch_RAD)*axf)+((cos(Pitch_RAD)*sin(Roll_RAD))*ayf)+((cos(Pitch_RAD)*cos(Roll_R
    AD))*azf)) - gravedad)*9.80665); // Vector de giro solo para el eje Z
    VelZ_girada += (AcelZ_girada  * dt);
    AlturaZ_girado += (VelZ_girada * dt);
    //Serial.print(compAngleX); Serial.print(" "); Serial.println(compAngleX);
    //ArchivoSD.print(AlturaZ_girado); ArchivoSD.print(" "); ArchivoSD.println(dt,3);
    tiempo_muestreo++; // incrementamos la cuenta del contador de las muestras
 }
  tiempo_ejecucion = ((hora_segundos())-(temporal_tiempo));
  ArchivoSD.println();
  ArchivoSD.print("Tiempo muestreado:  "); ArchivoSD.print(tiempo_ejecucion);
  ArchivoSD.println(" Segundos");
  ArchivoSD.close(); // cerramos el archivo donde se almacenan los datos*/

  delay(Inactividad); // retrasa 45 min la nueva ejecucion


}
//*************************************************************************************************
//*************************************************************************************
//*                                RUTINAS                                            *
//*************************************************************************************

//*************************************************************************************
//*                          RUTINAS  CALIBRACION GYRO                                *
//*************************************************************************************
//********************************* EJE X *********************************************
int calibracionXGyro (){
int temp2 = 0;

for (int j = 0; j < 10; j++){
  int sumadevalores = 0;
  int numerodevalores = 0;
  int temp = 0;
  for (int i = 0; i <= 200; i++)
  {
    accelgyro.setXGyroOffset(i);
```

```
    accelgyro.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);
    if ((gx >-15) && (gx <15))
        {
            numerodevalores += 1;
            sumadevalores += i;
        }
  }
temp = sumadevalores/numerodevalores;
temp2 += temp;
}
int correccionoffset = temp2/10;
return correccionoffset;
}
//******************************** EJE Y ****************************************
int calibracionYGyro (){
int temp2 = 0;

for (int j = 0; j < 10; j++){
  int sumadevalores = 0;
  int numerodevalores = 0;
  int temp = 0;
  for (int i = 0; i <= 200; i++)
  {
    accelgyro.setYGyroOffset(i);
    accelgyro.getMotion6(&ax, &ay, &az, &gx, &gy, &gz); // lectura de los valores de las
    aceleraciones y velocidades angulares
    if ((gy >-15) && (gy <15))
        {
            numerodevalores += 1;
            sumadevalores += i;
        }
  }
temp = sumadevalores/numerodevalores;
temp2 += temp;
}
int correccionoffset = temp2/10;
return correccionoffset;
}
//******************************** EJE Z ****************************************
int calibracionZGyro (){
int temp2 = 0;

for (int j = 0; j < 10; j++){
  int sumadevalores = 0;
  int numerodevalores = 0;
  int temp = 0;
  for (int i = 0; i <= 200; i++)
  {
    accelgyro.setZGyroOffset(i);
    accelgyro.getMotion6(&ax, &ay, &az, &gx, &gy, &gz); // lectura de los valores de las
    aceleraciones y velocidades angulares
    if ((gz >-15) && (gz <15))
        {
            numerodevalores += 1;
            sumadevalores += i;
        }
  }
temp = sumadevalores/numerodevalores;
temp2 += temp;
```

```
}
int correccionoffset = temp2/10;
return correccionoffset;
}

//*****************************************************************************
//*                  RUTINA PARA CONCATENAR LOS VALORES DEL ^                  *
//*                RELOJ PARA USARLOS COMO NOMMBRE DEL ARCHIVO                 *
//*****************************************************************************
String Nombre_Fecha()
{
     String Fecha;
    rtc.getDateTime24(&year, &month, &day, &hours, &minutes, &seconds);

 if (month < 10)                    // correct date if necessary
  {
    Fecha = "0";
    Fecha += month;
  }
  else
  {
    Fecha = "1";
    Fecha += (month - 10);
  }
 if (day < 10)                   // correct month if necessary
  {
    Fecha += "0";
    Fecha += day;
  }
  else
  {
    Fecha += day;
  }
  if (hours < 10)                    // Correción de la hora si es necesarioo
  {
    Fecha += "0";
    Fecha += hours;
  }
  else
  {
    Fecha += hours;
  }
  if (minutes < 10)                   // corrección de los minutos si es necesario
  {
   Fecha += "0";
   Fecha += minutes;
  }
  else
  {
   Fecha += minutes;
  }
  return Fecha;
}
//*****************************************************************************
//*                  RUTINA PARA IMPRMIR LA FECHA DE LA MUESTRA                *
//*****************************************************************************
String Fecha_muestra()
{
    rtc.getDateTime24(&year, &month, &day, &hours, &minutes, &seconds);
```

```
    String Fecha ="Fecha de la muestra: " ;

 if (day < 10)                    // correct date if necessary
  {
    Fecha += "0";
    Fecha += day;
  }
  else
  {
    Fecha += day;
  }
 Fecha += "/";
 if (month < 10)                   // correct month if necessary
  {
    Fecha += "0";
    Fecha += month;
  }
  else
  {
    Fecha += month;
  }
  Fecha += "/";
  Fecha += year ;
  Fecha +="    Hora:";
  if (hours < 10)                    // Correción de la hora si es necesarioo
  {
    Fecha += "0";
    Fecha += hours;
  }
  else
  {
    Fecha += hours;
  }
  Fecha += ":";
  if (minutes < 10)                  // corrección de los minutos si es necesario
  {
   Fecha += "0";
   Fecha += minutes;
  }
  else
  {
   Fecha += minutes;
  }
  return Fecha;
}


//*****************************************************************************
//*                    RUTINA  PARA ABRIR UN   ARCHIVO                        *
//*****************************************************************************
void abrir_archivo()
{
  tempArchivo.toCharArray(Archivo,12); // COnversión de String en Char para pasarlo a la
  funcion SD.Open
  ArchivoSD = SD.open( Archivo, FILE_WRITE); // abrimos un archivo en la tarjeta SD
}

//*****************************************************************************
//*                 RUTINA  PARA CALCULAR LA ORIENTACION                      *
```

```
//*****************************************************************************
int Orientacion()
{
  float fx,fy,fz; // toma los valores de la brujula
  magn.getValues(&fx,&fy,&fz);
  float heading = atan2(fy, fx);
  if(heading < 0)
    {
      heading += 2 * M_PI;
    }
   return (heading * 180/M_PI);
}
//*****************************************************************************
//*                 RUTINA  PARA COMBERTIR LA HORA EN SEGUNDOS                 *
//*****************************************************************************
unsigned int hora_segundos()
{
  rtc.getDateTime24(&year, &month, &day, &hours, &minutes, &seconds);
  return ((hours*3600)+(minutes*60)+(seconds));
}
//*****************************************************************************
//*          RUTINA  PARA COMPROBAR LAS CONEXION DE LOS SENSORE                *
//*****************************************************************************
boolean Sensores_conectados()
{
   return(accelgyro.testConnection()&& rtc.testConnection() &&  TarjetaSD);
  //return ((accelgyro.testConnection())&&(rtc.testConnection())&&(SD.begin(chipSelect)));
  //&&(bmp.begin())
}
```

B Presupuesto

Como una de las premisas era que se mantuviera un bajo costo de fabricación, se presenta un presupuesto de lo invertido hasta el momento en la ejecución del proyecto.

Aún sin saber el coste de fabricación del soporte físico, se podría decir que el objetivo del coste esta cumplido, aunque a esto habría que añadir el importe de las mejoras planteadas, haciendo más eficientes el funcionamiento de la boya.

Artículos	Cantidad	Costo total (€)
DS1307		
Módulo reloj para Arduino con pila incluida	1	3,2
HMC5883L		
Módulo Brujula Digital 3 Ejes Para Arduino	1	3,4
BMP085		
Módulo Presión Atmosférica, Altímetro y Barómetro	1	8,9
IMU		
Modulo Giroscopio 3 Ejes + Acelerómetro Arduino	1	6,2
EEPROM		
Eeprom Serial 256 Kb, Smd,	1	8,5
SD		
Modulo lector tarjeta SD	1	3,2
ARDUINO DUE		
Placa Arduino DUE	1	22
Placa Baquelita		
Placa de baquelita y p/p de clable, estaño y cloruro férrico	1	15
Total		70,40 €

Bibliografía

[1] "Ángulos de Euler - Wikipedia, la enciclopedia libre". URL https://es.wikipedia.org/wiki/%C3%81ngulos_de_Euler.

[2] "Arduino - Home". URL https://www.arduino.cc/.

[3] "Cálculo de filtros digitales". URL http://www.schwietering.com/jayduino/filtuino/.

[4] "Concepto Generales, Clasificación de los filtros digitales". URL http://www2.dis.ulpgc.es/~li-pso/tema5/tema5.htm.

[5] "Especificacionde del Bus I^2C". URL http://www.i2c.info/i2c-bus-specification.

[6] "A Guide To using IMU".

[7] "Oscilaciones de una boya en el agua". URL http://www.sc.ehu.es/sbweb/fisica/fluidos/estatica/boya/boya.htm.

[8] "Puertos del Estado". URL http://www.puertos.es/es-es/oceanografia/Paginas/portus.aspx.

[9] Mínguez, G. F., *Integración Kalman de sensores inerciales INS con GPS en un UAV*. Tesis doctoral, UPC, 2009. URL http://upcommons.upc.edu/bitstream/handle/2099.1/6930/memoriadef.pdf?sequence=1.

[10] Stallings, W., *Organización y arquitectura de computadores*. 7 ed., ISBN 978-84-89660-82-3.

[11] Tipler, P. A., *Física para la ciencia y la técnología*. Barcelona: Ed. Reverté, ISBN 84-291-43823.