Universidad Complutense de Madrid
Universidad Nacional de Educación a Distancia

Máster en Ingeniería de Sistemas y Control

Diseño y construcción de un sensor flotante para el cultivo de microalgas

Autora:
Ana Dorda Martín

Directores:
José Sánchez Moreno
Andrzej Pawlowski

Año lectivo 2017-2018 | Septiembre 2018

Máster en Ingeniería de Sistemas y Control

Diseño y construcción de un sensor flotante para el cultivo de microalgas

Proyecto Tipo A

Autora:
Ana Dorda Martín

Directores:
José Sánchez Moreno
Andrzej Pawlowski

Año lectivo 2017-2018 | Septiembre 2018

Autorización

Autorizamos a la Universidad Complutense y a la UNED a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la memoria de este Trabajo Fin de Máster, como el código, la documentación y/o el prototipo desarrollado.

Firmado:

Firma del alumno

Resumen

Una de las limitaciones actuales en el estudio y control de los fotobiorreactores *raceway* es el hecho de que las mediciones de temperatura o pH, siendo estas las variables a controlar, solo se pueden medir en un punto del fotobiorreactor. Esto limita el diseño de controladores, teniendo que adaptarse a dinámicas desconocidas del sistema.

Con el fin de ayudar a mejorar la comprensión de la evolución de las características del cultivo en todo el fotobiorreactor, en este trabajo se plantea la construcción de un sensor flotante que mida el pH, la temperatura y las asocie con su posición. Con esto se pretende conocer más a fondo las características del sistema a controlar y cómo evolucionan estos valores a lo largo del fotobiorreactor.

La construcción de este sensor se ha realizado desde el principio, partiendo de la elección de componentes, pasando por la programación y terminando en su construcción. Además, se programó una interfaz gráfica, la cual se encuentra en una estación base recibiendo los datos enviados por el sensor. Una vez construido el sistema, se realizó una prueba para comprobar su correcto funcionamiento. El sistema tomó múltiples mediciones en una piscina natural. Dichas mediciones fueron procesadas con MATLAB, representándose el pH y la temperatura en función de la posición, así como la representación de los puntos donde midió el sensor. Analizando los resultados, pudo verse una desviación entre la zona en la que se realizaron las mediciones y los puntos en los que se realizaron las mediciones según el GPS. Por otro lado, las representaciones del pH y temperatura en función de la posición dan una idea de la utilidad de este sistema en el estudio de los cultivos en fotobiorreactores.

En conclusión, tras los resultados obtenidos en las pruebas se pudo determinar que el prototipo funciona correctamente. Sin embargo, puesto que es un prototipo, este debe de seguir evolucionando. Algunas de las mejoras pueden ser añadir una antena al GPS, reemplazar el sensor de pH por uno de menor tamaño o imprimir una carcasa 3D que se adapte al circuito, entre otras.

Palabras clave

Microalgas, fotobiorreactor, raceway, sensor, pH, temperatura, GPS, Arduino, control.

Contenidos

Autorización 3

Resumen 4

Palabras clave 5

Lista de figuras 8

Lista de tablas 9

Nomenclatura 10

1. Introducción 11

2. Trabajo previo 13

2.1. Especificaciones del sistema 13

2.2. Elección de sensores y módulos de comunicación 14

Sensor de pH *14*

Sensor de temperatura *15*

Micro GPS *15*

Módulo de comunicación ZIGBEE *16*

Alimentación *17*

2.3. Layout del sistema 17

3. Diseño software 20

3.1. Diseño software del sensor flotante 20

Visión general del programa *20*

Función loop() *21*

Función readDS18B20() *22*

Función readGPS() *23*

Función readpH() *26*

Función sendXbee() *26*

3.2. Mecanismos software para la reducción del consumo 27

Desactivación de dispositivos externos *28*

Desactivación del ADC *28*

Desactivación del detector Brown-Out *29*

Desactivación de los módulos software *29*

Sleep Modes del ATmega328P *29*

Uso del WDT (Watchdog Timer) *30*

Función sleep() *31*
3.3. Diseño software de la estación base 32
Configuración del Arduino Uno y el módulo Xbee *32*
Visión general de la interfaz gráfica *33*
4. Montaje del prototipo y pruebas 36
4.1. Planteamiento del diseño 36
4.2. Montaje del sensor flotante 38
4.3. Pruebas finales 40
Metodología *40*
Procesamiento de las mediciones *41*
5. Conclusión 42
Bibliografía 44
Anexo 45
Anexo I: código transmisor (sensor flotante) *45*
Anexo II: código receptor *49*
Anexo III: código interfaz de la estación base *50*
Anexo IV: listado de componentes *56*

Lista de figuras

Figura 1-1. Ejemplo de control de pH en un fotobiorreactor raceway (4). 11
Figura 2-1. Esquemático de un fotobiorreactor de tipo raceway (3). 13
Figura 2-2. Layout del hardware en el sensor flotante. 17
Figura 2-3. Layout del receptor en la estación base. 18
Figura 2-4. Layout del sistema. 19
Figura 3-1. Diagrama de flujo del programa de la boya. 20
Figura 3-2. Diagrama de flujo de la función loop(). 22
Figura 3-3. Diagrama de flujo de la función readDS18B20(). 23
Figura 3-4. Diagrama de flujo de la función readGPS() 24
Figura 3-5. Diagrama de flujo de la función readpH(). 25
Figura 3-6. Diagrama de flujo de la función sendXbee(). 27
Figura 3-7. GUI para control de mediciones del sensor flotante. 33
Figura 3-8. Diagrama de flujo de la interfaz gráfica. 34
Figura 3-9. Ejemplo de fichero generado por la aplicación. 35
Figura 4-1. Idea inicial para la carcasa de la boya (13). 36
Figura 4-2. Planteamiento del diseño del sensor flotante para el cultivo de microalgas. 37
Figura 4-3. Caja estanca J160L de JSL con IP66. 38
Figura 4-4. Prensaestopas del sensor de pH (izquierda) y prensaestopas del sensor de temperatura (derecha). 39
Figura 4-5. Visión general del sensor flotante. 39
Figura 4-6. Piscina de Madrid Río donde se realizaron las mediciones. 40
Figura 4-7. Estación base (izquierda) y sensor flotante (derecha). 40
Figura 4-8. En rojo, puntos en los que se realizaron las mediciones según el GPS. En amarillo, zona en la que se realizaron las mediciones. 41
Figura 4-9. Izquierda, mediciones de temperatura. Derecha, mediciones de pH. 41

Lista de tablas

Tabla 2-1. Sensores de temperatura considerados y sus especificaciones. 15
Tabla 2-2. Módulos GPS considerados y sus especificaciones. 16
Tabla 2-3. Especificaciones del módulo de comunicación Zigbee. 16
Tabla 3-1. Sleep Modes del ATmega328p (11). 30
Tabla 3-2. Selección de prescaler del WDT. 30
Tabla Anexo-1. Lista con los componentes seleccionados para la boya y el receptor. 56

Nomenclatura

API	Interfaz de Programación de Aplicaciones
BNC	Bayonet Neill–Concelman
CAT5	Cable de Categoría 5
GND	Ground
GPGGA	Global Positioning System Fix Data
GPRMC	Recommended Minimum Specific GPS/Transit Data
GPS	Global Positioning System
GUI	Graphic User Interface
IP	Ingress Protection
ISFET	Ion Sensitive Field Effect Transistor
NMEA	National Marine Electronics Association
TTL	Transistor-Transistor Logic
UAL	Universidad de Almería
UART	Universal Asynchronous Receiver-Transmitter
USB	Universal Serial Bus
VTA	Vertical Tracking Angle
WDT	WatchDog Timer
WPAN	Red de Área Personal

1. Introducción

El cultivo de microalgas despierta un elevado interés ya que puede aplicarse en diferentes procesos como el tratamiento de aguas residuales, hacer frente a las emisiones de CO_2 o la producción de bioenergía. Esto se debe a sus características naturales. Sus procesos fotosintéticos son más eficientes que los de otras plantas terrestres, lo que incrementa su velocidad de crecimiento. Estas características hacen que tengan una elevada tasa de cultivo en comparación con el volumen que ocupan (1). Además, pueden ser cultivadas durante cualquier época del año, incluso en condiciones muy adversas como los climas semiáridos.

La biomasa obtenida tras los procesos de producción es utilizada en la fabricación de biocombustibles, productos farmacéuticos, cosmética, etc. El cultivo de microalgas se realiza en depósitos adaptados, denominados fotobiorreactores, en los que se procura que se den las condiciones necesarias para un crecimiento óptimo.

Estos son dispositivos que contienen metros cúbicos de biomasa cuya finalidad es la producción en masa de una microalga. Pueden estar diseñados para una especie de alga o para una aplicación específica y las condiciones de cultivo (temperatura, pH, concentración de O_2) deben ser muy estables. Además, deben recibir una gran cantidad de luz que sirve de nutriente para las algas. El rango de temperatura óptimo para el cultivo oscila entre 18 y 25 ºC mientras que el de pH entre 7 y 8 (2). El CO_2 afecta de manera directa al valor de pH del cultivo.

Los fotobiorreactores (3) pueden ser abiertos o cerrados. Los de tipo abierto pueden ser *open ponds* o *raceway*. Los de tipo *raceway* proveen de agitación y mezcla. Esta agitación se provee a través de una rueda de paletas que permite mantener las algas en suspensión con un escaso consumo energético.

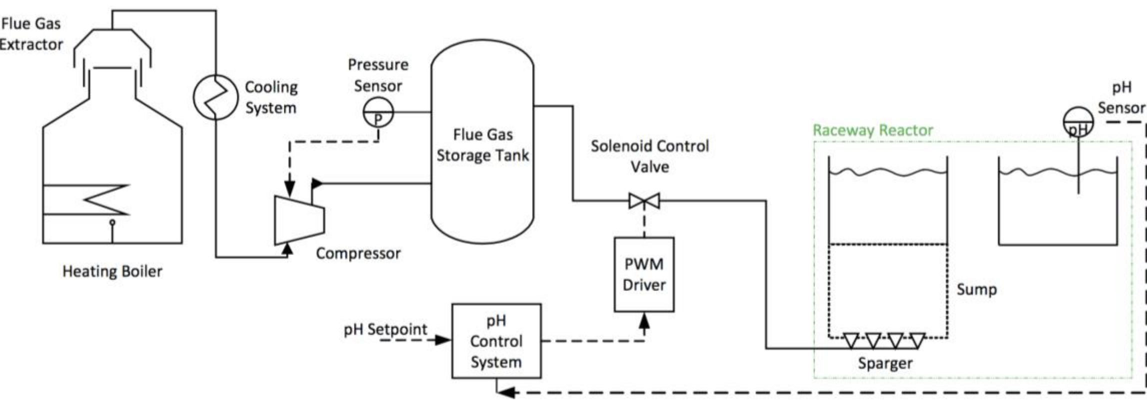

FIGURA 1-1. EJEMPLO DE CONTROL DE pH EN UN FOTOBIORREACTOR RACEWAY (4).

Desde hace años los fotobiorreactores *raceway* han sido estudiados como una opción para llevar a cabo el cultivo de microalgas a escala industrial. Esto es debido a la reducida inversión inicial que se requiere en comparación con otras estructuras (como los fotobiorreactores tubulares). Uno de los problemas a los que se hace frente es la optimización de los procesos de producción. Lo que se busca es situar los valores de temperatura y de pH del cultivo dentro de un rango óptimo. Para ello se utilizan estrategias de control automático, donde la temperatura y el pH son medidos en un punto del fotobiorrector y, en función de los valores medidos, se ajustan tanto la temperatura aplicada al cultivo como el caudal de CO_2 (5).

Sin embargo, esta metodología de control es limitada, ya que se basa en el desconocimiento de otras dinámicas del sistema (4). El hecho de medir los valores de pH y temperatura en un punto del fotobiorreactor supone una limitación, puesto que se desconoce como van a evolucionar estos valores a lo largo del dicho sistema de cultivo.

En este trabajo se pretende mostrar el diseño y construcción del prototipo de un sensor flotante que ayude en el cultivo de microalgas. El objetivo es que el sensor mida de manera periódica y autónoma los valores de pH y temperatura en todo el fotobiorreactor. Dichos valores serán recopilados para su posterior estudio. Con esto se pretende ayudar en la caracterización de los fotobiorreactores, así como reforzar la comprensión de la evolución de las características del cultivo.

2. Trabajo previo

2.1. Especificaciones del sistema

La idea de este trabajo parte de la posibilidad de diseñar un sensor flotante que sea autónomo y realice mediciones de las características del agua; ayudando de este modo al control del cultivo de microalgas. Este debe de ir metido en un envase hermético y estar flotando en el sistema de cultivo realizando mediciones. Dichas mediciones deben ser enviadas a un receptor situado en una estación base que se encargue de procesar los datos.

En un primer momento, la idea inicial era que este sensor también pudiera ser usado en fotobiorreactores tubulares. Pero debido al reducido tamaño que debería tener se optó por diseñar primero un prototipo para un fotobiorreactor *raceway*, lo que permite incrementar su tamaño. Dicho fotobiorreactor tiene las siguientes características:

- Un canal de 50 metros de diámetro (100 de total).
- Ancho del canal de 1 metro y una profundidad de 20 cm.
- Las palas tienen un ancho de 80 cm.
- La distancia entre el final de la pala y el fondo del reactor es de 9 cm.
- Rango de temperatura que soporta entre 17 ºC y 31 ºC, con una exposición constante al sol.

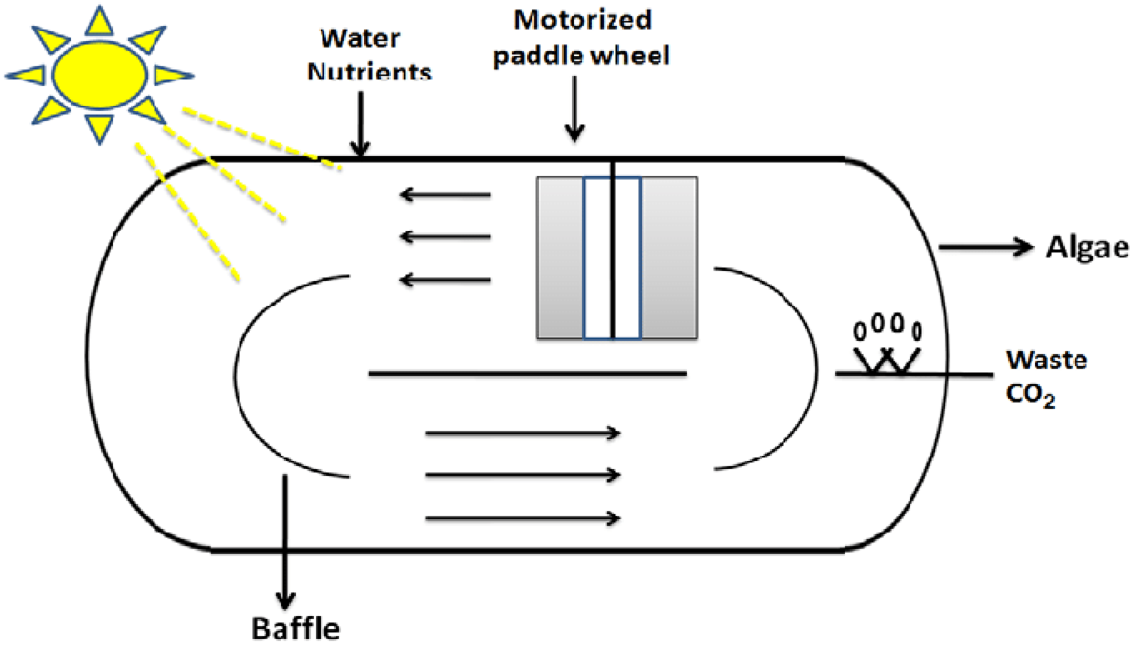

FIGURA 2-1. ESQUEMÁTICO DE UN FOTOBIORREACTOR DE TIPO RACEWAY (3).

La versión mas básica de este sensor debe llevar dos sensores integrados, uno de pH y otro de temperatura, y se deberá tener en cuenta la posibilidad de integrar otros sensores más adelante. También debe de contar con un módulo GPS para asociar la medida con un punto exacto del fotobiorreactor. Todo este sistema debe ir metido en una caja estanca que permita recibir y transmitir las mediciones realizadas. La comunicación se realizará a través de Zigbee. Zigbee es un estándar de comunicaciones inalámbricas diseñado por la Zigbee Alliance. Zigbee está basado en el estándar IEEE 802.15.4 de redes inalámbricas de área personal WPAN (Wireless Personal Area Network). Tiene como objetivo las aplicaciones que requieren comunicaciones seguras con baja tasa de envío de datos y maximización de la vida útil de sus baterías. Dicha transmisión deberá de ser periódica o por eventos.

A modo de resumen, las especificaciones que debe cumplir el sistema son:

- Soportar un rango de temperatura de 40ºC a 0ºC.
- Aprovechar la luz solar para aumentar su autonomía.
- Ser completamente hermético y estar preparado para soportar golpes.
- Tener un tamaño reducido para no quedarse atascado al pasar por las palas.
- Enviar datos por eventos o de manera periódica.
- Contar con un sensor de temperatura, uno de pH y un GPS.
- Contar con comunicación Zigbee.
- Los datos deben ser recibidos y procesados por una estación base.

2.2. Elección de sensores y módulos de comunicación

Sensor de pH

Este fue el sensor más complicado de encontrar. La medición del pH se realiza a través de una sonda conectada a un circuito que adapta la medida para transmitirla al micro. Debido a la electrónica interna de la sonda y a que cuentan con un sensor de temperatura integrado para compensar el calculo del valor de pH, los tamaños de este tipo de sondas son muy elevados. Esto es un problema, ya que la distancia desde el fondo del reactor hasta la pala es de 9 cm. Si el sensor flotante sobrepasa este tamaño, corre el riesgo de atascarse al pasar por la rueda de palas.

Una alternativa a la sonda de pH tradicional es la sonda de tipo ISFET. La tecnología de este tipo de sondas está basada en transistores de efecto de campo los cuales, al tratarse de tecnología microelectrónica, hacen que la sonda presente un menor tamaño. Sin embargo, las sondas de pH basadas en esta tecnología no están muy extendidas lo cual incrementa su coste. Finalmente se optó por el Industrial pH Probe de Atlas Scientific junto con el EZO™ pH Circuit de la misma marca.

Sensor de temperatura

TABLA 2-1. SENSORES DE TEMPERATURA CONSIDERADOS Y SUS ESPECIFICACIONES.

Specifications	*Adafruit DS18B20*	*Atlas Scientific ENV-TMP-D*	*Atlas Scientific PT-1000*
Temperature Range	-55℃ to 125℃	-20℃ to 133℃	-200℃ to 850℃
Accuracy	±0.5°C	±1°C	+/- (0.15 + (0.002*t))
Query Time	< 750ms	< 520ms	-
Output	Digital (1-Wire interface)	Digital (1-Wire interface)	Analógica (BNC)
Supply Voltage	3-5 VDC	3.3-5.5 VDC	3.3-5.5 VDC

Los diferentes sensores de temperatura que se tuvieron en consideración pueden verse en la Tabla 2.1.

El PT-1000 de Atlas Scientific asegura más exactitud en la medida, pero al ser analógico y tener una conexión BNC es necesario comprar a parte un conversor de BNC a TTL. Además, al ser analógico, habría que acondicionar la señal, añadiendo por ejemplo un amplificador de instrumentación que nos de todo el rango de 0 a 5V. Esto complica considerablemente la electrónica.

Finalmente, se optó por el sensor DS18B20 de Adafruit, ya que es el que aúna una mayor exactitud con la sencillez de su conexión al micro. Además, es sumergible.

Micro GPS

En la Tabla 2.2 puede verse una relación de los módulos GPS considerados para el sistema junto con sus especificaciones.

De todos ellos el escogido fue el Adafruit Ultime GPS Breakout. A pesar de tener una exactitud en la posición menor que los otros dos módulos y tardar más en encontrar enlace, su consumo se reduce a la mitad y cuenta con más satélites buscando. Además, todos los módulos devuelven la información siguiendo el estándar NMEA 0183.

TABLA 2-2. MÓDULOS GPS CONSIDERADOS Y SUS ESPECIFICACIONES.

Specifications	*Adafruit Ultimate Breakout GPS*	*NEO-6M GPS Module*	*GPS GP-20U7*
Satellites	66 searching	50 searching	56 searching
Position Accuracy	3 meters	2.5 meters	2.5 meters
Velocity Accuracy	0.1 meters/s	0.1 meters/s	0.1 meters/s
Warm/Cold Start	34 seconds	27 seconds	29 seconds
Maximum Velocity	515 m/s	500 m/s	515 m/s
Supply Voltage	3.0-5.5 VDC	3-5 VDC	3.3VDC
Consumption	22 mA @ 3.3 V	45 mA @ 3.3 V	40 mA @ 3.3 V
Output	NMEA 0183 9600 baud	NMEA 0183 9600 baud	NMEA 0183 9600 baud
Dimensions	25.5mm x 35 mm	25 mm x 35 mm	18 mm x 18 mm

Módulo de comunicación ZIGBEE

A continuación, pueden encontrarse las especificaciones del módulo de comunicación Zigbee escogido. Se necesitan dos módulos, uno para el transmisor (la boya o sensor flotante) y otro para el receptor (estación base). Cada uno irá acompañado de un *shield* de Libelium y otro de Gravi Tech para el Arduino UNO y Nano respectivamente.

TABLA 2-3. ESPECIFICACIONES DEL MÓDULO DE COMUNICACIÓN ZIGBEE.

Specifications	*2.4 GHz XBee-PRO*
Indoor/Urban Range	90 m
Outdoor/RF Line-Of-Sight Range	3200 m
Frequency Band	2.4 GHz
Supply Voltage	2.7 - 3.6VDC
Consumption	50 mA @ 3.3VDC

Alimentación

Puesto que el sistema va a estar expuesto a la luz solar durante largos periodos de tiempo, esto puede usarse a su favor para aumentar su autonomía. Para ello, se escogió una placa solar de Sundance Solar de 6V y 100mA. Esta placa solar debe de cargar una batería al mismo tiempo que esta alimenta el circuito. Para ello, se escogió una batería de polímero de litio de 2000 mAh y 3,7 V así como un cargador solar USB / DC / Solar Lithium Ion/Polymer charger de Adafruit.

2.3. Layout del sistema

En el Anexo IV puede encontrarse un listado de los componentes escogidos. Para el sistema de la boya se va a usar un Arduino Nano, el cual es una pequeña y completa placa basada en el procesador ATmega328. En la Figura 2.2 puede verse un esquemático de como van a ir conectados los distintos sensores. Este Arduino Nano irá conectado al módulo de comunicación Zigbee, 2.4 GHz XBee-PRO a través del *shield* de Gravi Tech. Según el esquemático de dicho *shield* (6) los pines serie UART en XBee (DIN y DOUT) se conectan a D1/TX y D0/RX en el Arduino Nano respectivamente. Esto puede resultar en limitaciones en la configuración y programación de la comunicación.

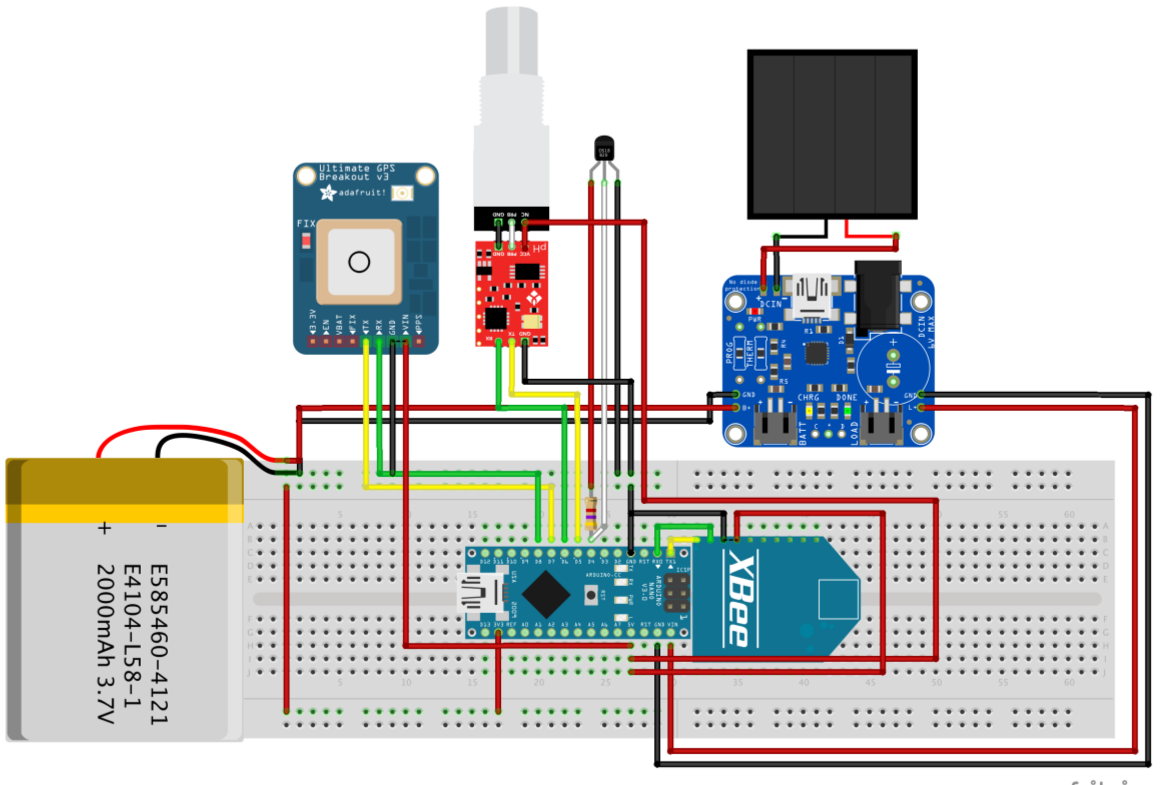

FIGURA 2-2. LAYOUT DEL HARDWARE EN EL SENSOR FLOTANTE.

Por otro lado, el sensor de temperatura DS18B20 cuenta con tres conexiones de salida. Una irá a la alimentación de 3.3V del Arduino, otra a GND y la salida de datos al pin D4 de Arduino (cable rojo, negro y blanco de la Figura 2.2 respectivamente). Además, entre el voltaje de alimentación y la salida de datos hay que colocar una resistencia de *pull up*. estas resistencias establecen un estado lógico en un pin cuando se encuentra en estado reposo. La resistencia de *pull up* establece un estado *HIGH*. Esto evita los falsos estados que se producen por el ruido generado por los circuitos electrónicos.

La sonda de pH, pH Industrial Probe, tiene una conexión de alimentación y otra de datos. Estos cables se conectan a un conversor de conexión coaxial CAT5 a BNC macho. Por otro lado, el circuito EZO™ pH Circuit va conectado a un conversor de TTL a BNC hembra. La conexión BNC se conecta con la sonda y el circuito al Arduino Nano. El circuito se alimenta con el pin de 5 V de Arduino y la conexión de tierra va conectada al GND. Los puertos TX y RX del circuito van conectados a los pines D5 y D6 de Arduino respectivamente (cables amarillo y verde de la Figura 2.2).

Por último, el módulo GPS se alimenta con 5V, y sus pines TX y RX se conectan a los pines D7 y D8 del Arduino Nano (cables amarillo y verde de la Figura 2.2). Además, todo el sistema es alimentado a través de una batería de polímero de litio de 3.7V y 2000mAh. Esta batería es cargada a través de la placa solar. Dichos elementos se interconectan entre si y con el sistema a través del cargador solar USB / DC / Solar Lithium Ion/Polymer charger de Adafruit.

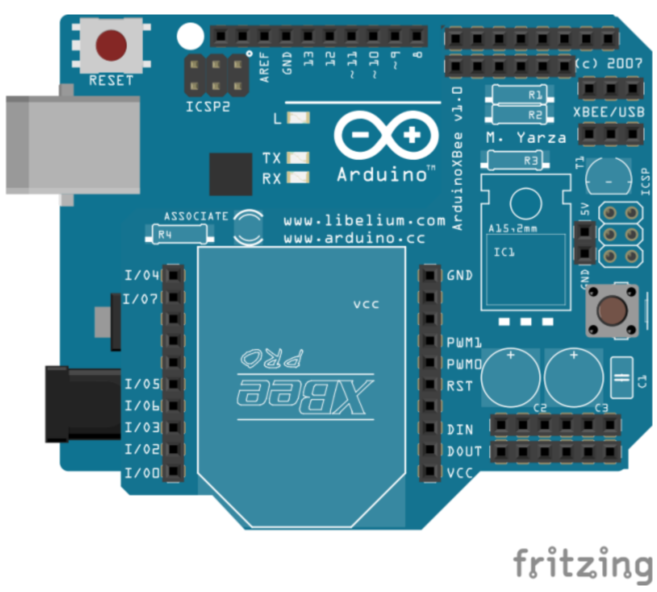

FIGURA 2-3. LAYOUT DEL RECEPTOR EN LA ESTACIÓN BASE.

En la Figura 2.3 puede verse el *layout* del receptor que irá incluido en la estación base. Este está compuesto de un Arduino UNO conectado a un módulo 2.4 GHz XBee-PRO a través del *shield* de Libelium. Al igual que el *shield* de Gravi Tech, los pines serie UART en XBee (DIN y DOUT) se conectan a D1/TX y D0/RX en el Arduino Nano respectivamente.

El sistema de medición estará compuesto por el sensor flotante, el cual se encontrará en el interior del fotobiorreactor realizando mediciones. Estas mediciones serán enviadas a través de Zigbee al receptor, el cual se encuentra conectado a un ordenador, componiendo así la estación base. Este ordenador contará con una interfaz gráfica en la que se presentarán por pantalla las mediciones realizadas. En el caso de que el usuario lo deseé, las mediciones podrán guardarse en un archivo de texto para su posterior procesamiento.

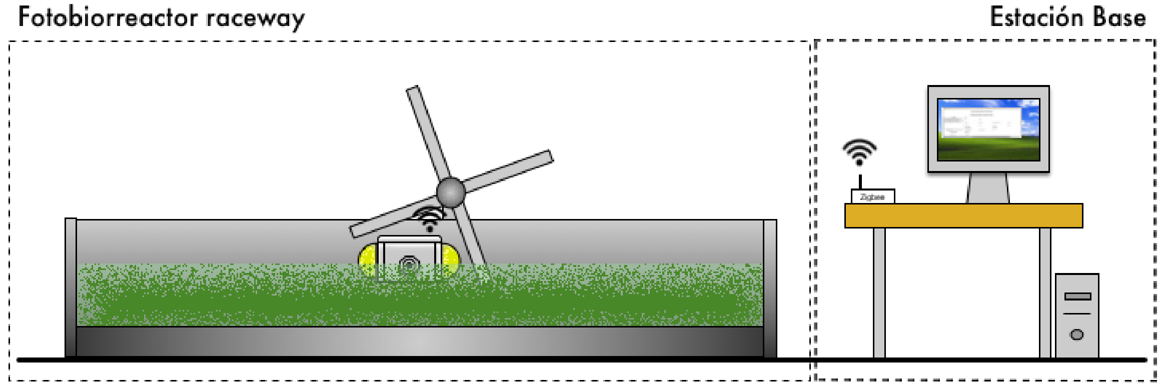

FIGURA 2-4. LAYOUT DEL SISTEMA.

En los siguientes apartados va a explicarse como se han programado las diferentes partes del sistema, tanto el sensor flotante como el receptor y la interfaz gráfica de la estación base. También, las diferentes técnicas utilizadas para reducir el consumo del sistema consiguiendo así aumentar su autonomía. Por último, se procederá a explicar el montaje del prototipo y los resultados de las pruebas realizadas con él.

3. Diseño software

3.1. Diseño software del sensor flotante

Visión general del programa

Ya en apartados anteriores se han dado algunas pinceladas sobre cómo va a ser el funcionamiento de la boya o sensor flotante. Más concretamente, una vez encendido el sistema, se abrirán los puertos de comunicación con cada uno de los sensores. Tras esto, se encuestan los sensores para obtener las mediciones pertinentes; para después proceder a su envío a la estación base. Una vez se han enviado las mediciones, para reducir el consumo, tanto el Arduino como alguno de los sensores entrarán en *sleep mode* durante cinco minutos. Transcurrido este tiempo, el sistema se despertará de nuevo para volver a medir y enviar las mediciones. En el caso de que se detecte un valor de temperatura o de pH fuera del rango deseado, se disparará un evento y el sistema comenzará a medir cada ocho segundos hasta que se vuelva a un rango normal. Con este método de control por eventos se consigue detectar con mayor precisión las zonas del fotobiorreactor que no se encuentran dentro de un rango óptimo de temperatura o pH pero disminuyendo el consumo del sistema.

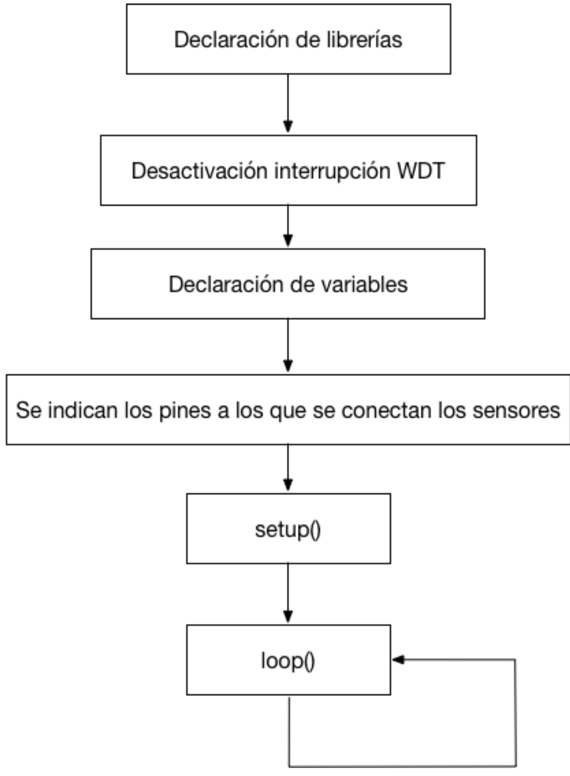

FIGURA 3-1. DIAGRAMA DE FLUJO DEL PROGRAMA DE LA BOYA.

Como se ha visto anteriormente el sistema está basado en Arduino, por lo que se utilizará tanto su entorno como su lenguaje de programación. La estructura básica de programación de Arduino divide la ejecución en dos partes: *setup* y *loop*. La función *setup()*

está destinada a la preparación del programa y *loop()* a la ejecución. *setup()* es la primera función que se ejecuta y se dedica a la declaración de variables, habilitación de pines y la inicialización de la comunicación serie. La función *loop()* incluye el código a ser ejecutado continuamente.

En la Figura 3-1 encontramos una visión genérica del código que se encuentra corriendo en el Arduino Nano. En primer lugar, es necesario declarar las librerías de donde se extraen las funciones necesarias para controlar los sensores o para dormir el Arduino. En este caso, se necesitan las librerías *OneWire.h* y *DallasTemperature.h* para poder establecer la comunicación y controlar el sensor de temperatura respectivamente. Para poder establecer la comunicación serie con el GPS y el sensor de pH es necesaria la librería *SoftwareSerial.h.* Para controlar el GPS, la librería *Adafruit_GPS.h.* Y para generar la interrupción con el WDT y dormir al Arduino, las librerías *avr/sleep.h* y *avr/wdt.h.*

Una vez declaradas las librerías, se desactiva la interrupción del WDT. Después se procede a declarar un *struct* denominado *sample*, donde se almacenan todas las mediciones hechas por los sensores y el GPS. A parte, también se declaran otras variables auxiliares necesarias en el código. Por último, se declaran los pines en los que se encuentran conectados los sensores.

En la función *setup()* se habilita la comunicación con los sensores. El sensor de temperatura tiene comunicación *OneWire*, mientras que el GPS, el sensor de pH y el módulo de comunicación Xbee se conecta con comunicación serie. Cabe comentar que por limitaciones del Xbee Shield de Gravi Tech, la comunicación serie se realiza directamente por los puertos 0 (Tx) y 1 (Rx) de Arduino, por lo que todo lo que se envíe por el puerto serie principal será enviado a través del módulo Xbee a la estación base. Es decir, no es necesario habilitar los pines, solo ajustar el *baudrate* del puerto serie principal de Arduino y abrir la comunicación.

Finalmente, se pasa a la función *loop(),* en la cual se encuestan los sensores y se duerme el Arduino repetidamente. Esta función se explicará con mayor detenimiento en el siguiente apartado.

Función loop()

En la Figura 3-2 se puede ver un esquemático del funcionamiento de la función *loop()*. Una vez abierta la comunicación con los sensores, el GPS y el módulo Xbee, se puede proceder a encuestar a los sensores y enviar las mediciones por el módulo Xbee a la estación base. Siguiendo el paradigma de programación modular, con el fin de que el programa sea más legible y manejable, se han creado sendas funciones destinadas a encuestar a cada sensor y enviar los datos a la estación base. Estas funciones son *readDS18B20()*, *readGPS()*, *readpH()* y *sendXbee()*, todas ellas serán explicadas más adelante. Después de enviar los datos se comprueban los eventos de temperatura y pH que nos indican si alguna de las

mediciones está fuera del rango deseado. Si es así, se ajusta el periodo de medición a ocho segundos. Si no, se deja en cinco minutos.

Tras esto se procede a introducir al Arduino Nano en *sleep mode*. Por limitaciones que se explicarán más adelante el Arduino solo puede dormir durante ocho segundos. Puesto que el objetivo es que duerma durante un máximo de cinco minutos, se ha generado una función *sleep()* que duerme al Arduino y lo despierta a los ocho segundos con una interrupción del WDT (*Watch Dog Timer*). Esta función es introducida dentro de un bucle *for* que ejecuta la función *sleep()* T veces, siendo T:

$$T = \frac{segundos}{8}$$

Con esto se consigue que el Arduino duerma el tiempo en segundos que se desee. Esta función *sleep()* también será explicada más adelante.

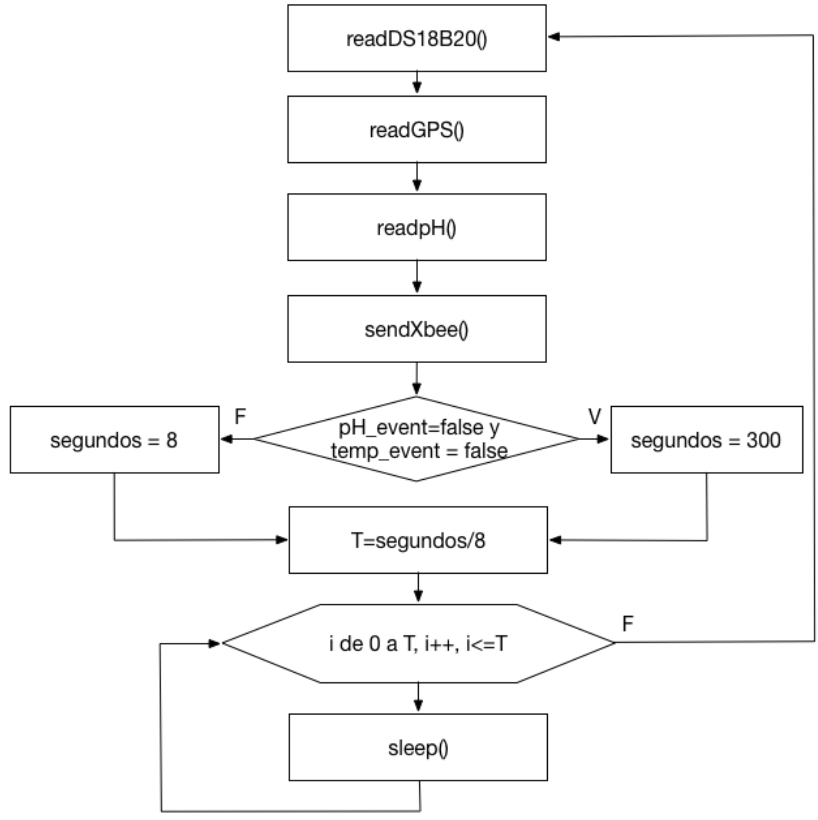

FIGURA 3-2. DIAGRAMA DE FLUJO DE LA FUNCIÓN LOOP().

Función readDS18B20()

Como se puede ver en la Figura 3-3, esta función es muy sencilla. Haciendo uso de las librerías antes mencionadas simplemente se pide al sensor que realice una medición. Después, se lee del sensor la medición, la cual estará en centígrados. Esta medición se almacena dentro del *struct* en su correspondiente variable. De la medida en centígrados

calculamos la medida en grados Fahrenheit y la guardamos en el s*truct* en su propia variable. En el caso de que el sensor se desconecte el valor de temperatura recibido es de -127 grados centígrados. Esto nos será de utilidad a la hora de programar el código de la estación base. Por último, si la temperatura no está en un rango aceptable salta el evento *temp_event*.

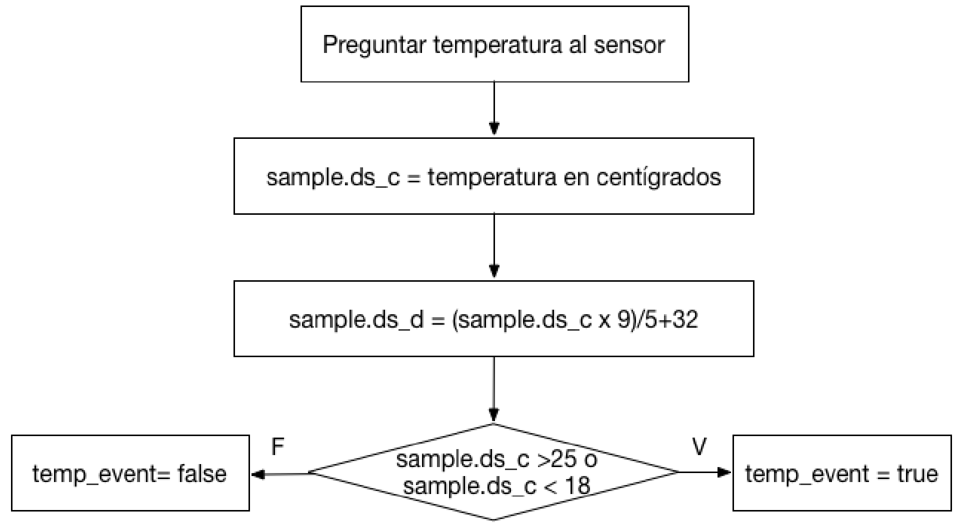

FIGURA 3-3. DIAGRAMA DE FLUJO DE LA FUNCIÓN READDS18B20().

Función readGPS()

Esta función se encarga de controlar y obtener datos del GPS Breakout 3 de Adafruit. Como se ha visto en las especificaciones de este GPS, la salida que devuelve son sentencias NMEA 0183 a 9600 baudrate.

La Asociación Nacional de Electrónica Marina (NMEA) (7) es una asociación compuesta por fabricantes sin ánimo de lucro, distribuidores, comerciantes, instituciones educativas y otros interesados en productos electrónicos marinos periféricos. El estándar NMEA 0183 define una interfaz eléctrica y un protocolo de datos para comunicaciones entre instrumentos marinos.

La sentencia NMEA (8) está compuesta por diferentes tipos de oraciones. Las más comunes son: $GPRMC (*Recommended Minimum Specific GPS/Transit Data*) y las oraciones $GPGGA (*Global Positioning System Fix Data*). Estas dos oraciones proporcionan la hora, la fecha, la latitud, la longitud, la altitud, *ground speed*, el enlace, la calidad del enlace y el VTA (*Vertical Tracking Angle*). El tipo de enlace indica si el GPS está enlazado con el satélite y ha recibido suficientes datos como para determinar la ubicación (enlace 2D) o la ubicación y la altitud (enlace 3D). En la función *setup()* se configuró el GPS para que envíe oraciones $GPRMC y $GPGGA.

Ahora bien, la función *readGPS()* comienza con un bucle *for,* el cual encuestará en repetidas ocasiones al GPS para asegurar que recibe correctamente la sentencia NMEA. Dentro de este bucle *for* lo primero que se hace es almacenar el instante de tiempo actual

en una variable auxiliar *current_time*. Esto será utilizado para detectar la desconexión del GPS, lo que produce un bucle infinito. Tras esto se entra en un bucle *while*, el cual leerá los caracteres recibidos por parte del GPS hasta obtener una sentencia NMEA correcta. Después de leer el carácter recibido se comprueba si ha transcurrido más de 1 segundo dentro del bucle *while*. Si es así, se pone la variable booleana *sensGPS* como *false* para indicar que el GPS está desconectado y se sale del bucle *while*. En el caso de que no haya transcurrido más de 1 segundo, se pone *sensGPS* como *true* y se sigue en el bucle.

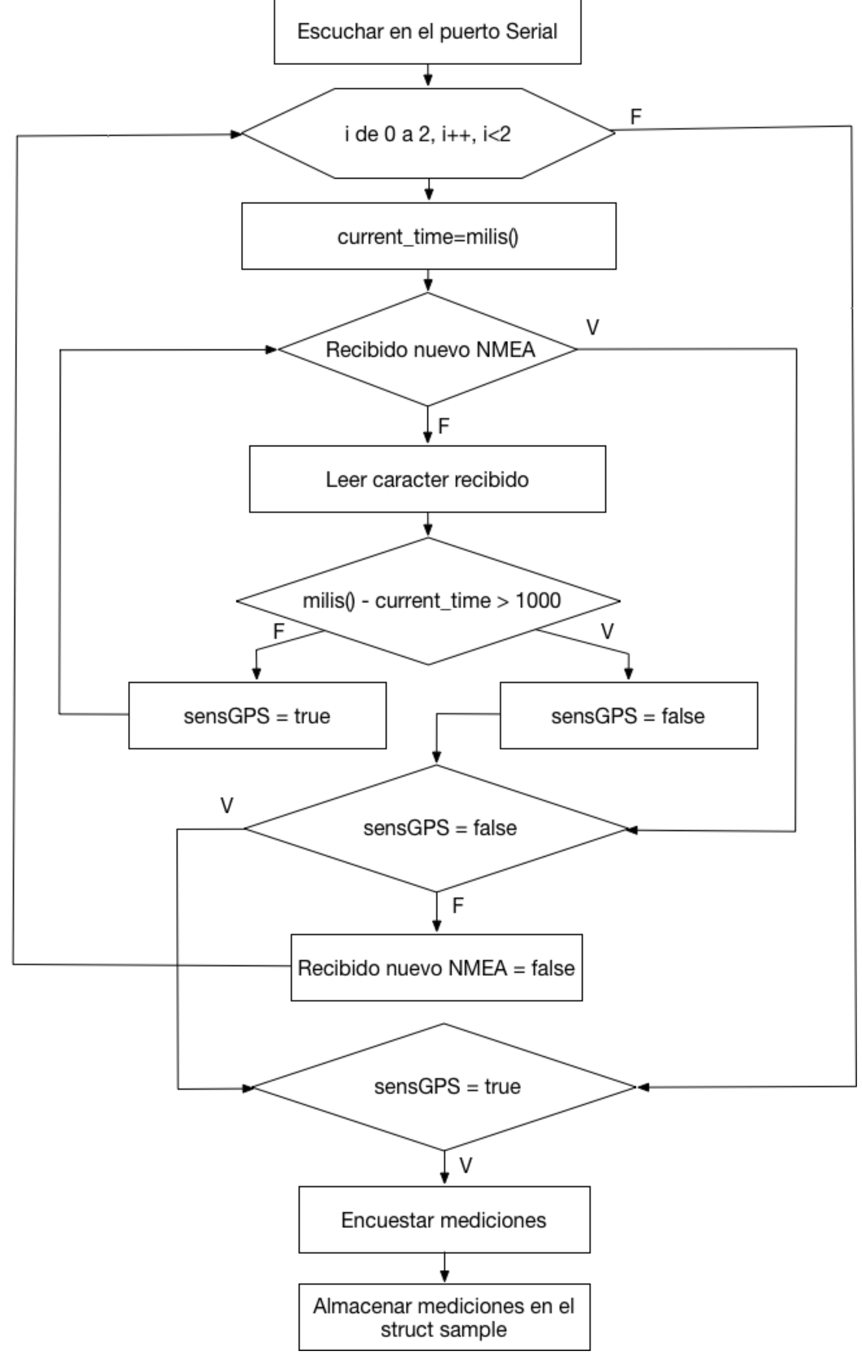

FIGURA 3-4. DIAGRAMA DE FLUJO DE LA FUNCIÓN READGPS()

Una vez fuera del *while*, se vuelve a comprobar si el GPS está desconectado viendo el valor de la variable *sensGPS*. Si su valor es *false*, se sale del bucle *for*. Finalmente, en el

caso de que el GPS esté conectado y se haya recibido correctamente la sentencia NMEA, se obtienen los valores de la hora, la fecha, la latitud, la longitud, la altitud, *ground speed*, el enlace, la calidad del enlace y el VTA; los cuales son posteriormente almacenados en el *struct* en sus correspondientes variables.

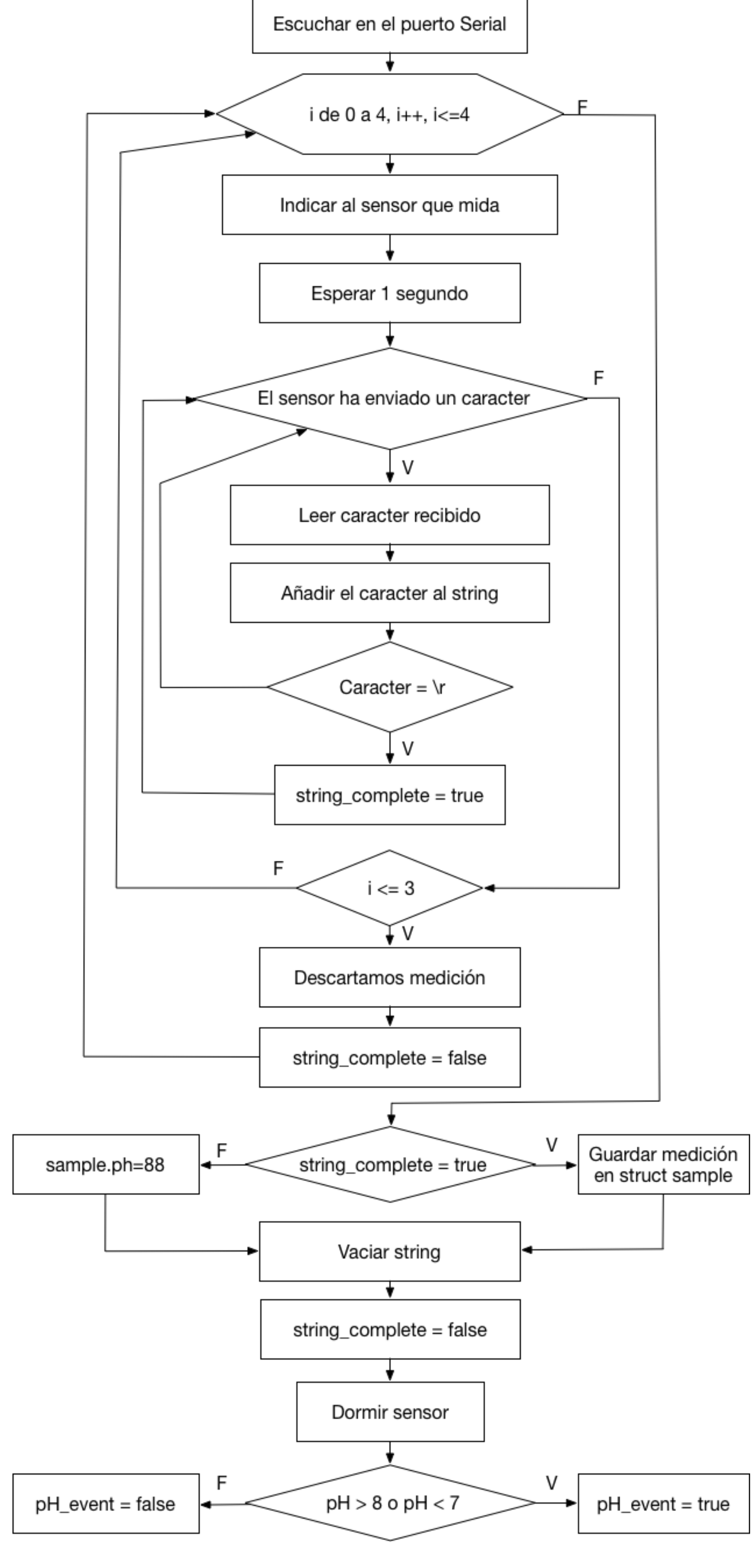

FIGURA 3-5. DIAGRAMA DE FLUJO DE LA FUNCIÓN READPH().

Función readpH()

Según el *datasheet* del pH Circuit EZO (9), una vez el sensor vuelve a funcionar después de haber estado en *sleep mode*, es necesario realizar cuatro mediciones consecutivas para que estas se puedan considerar validas. Por lo tanto, en la función *readpH()*, tras empezar a escuchar en el puerto serie correspondiente al sensor de pH, se entra en un bucle *for.* Dentro de este se mandará el carácter R para indicar al sensor que realice una medición. Después se esperará un segundo para que envíe la medición y se entrará en un bucle *while,* donde se leerá carácter a carácter el dato recibido. Este dato recibido se almacena en la variable auxiliar *sensorstring*. Una vez recibida toda la medición se pone el *flag sensor_stringcomplete* a *true* y se sale del bucle *while*.

Las cuatro primeras mediciones serán descartadas. Esto se hace vaciando la variable auxiliar y volviendo a poner el *flag* como *false*. Solo se conserva la última medición, la cual corresponde con el ultimo ciclo del bucle *for*. Para comprobar si el sensor está desconectado, en cuyo caso no se recibe ningún carácter, se comprueba el estado del *flag*. En caso de que su estado sea *true*, se almacena la medición en el *struct* en su correspondiente variable. Si es *false*, se almacena en el *struct* el valor 88, el cual indicará a la estación base que el sensor de pH no está funcionando correctamente.

Finalmente, se vacía la variable auxiliar, se vuelve a poner el valor del *flag* a *false* y se vuelve a dormir al sensor. Si el pH no está en un rango aceptable salta el evento *pH_event*. Todo este procedimiento puede verse representado en la Figura 3-5.

Función sendXbee()

La función *sendXbee()* es la encargada de enviar todas las mediciones, a través del módulo de comunicación Xbee, a la estación base.

En primer lugar, se limpia el valor de la variable auxiliar *xbeestring*, la cual contiene el *string* con los datos enviados a la estación base. Hecho esto, se comprueba si el GPS está desconectado (el valor de sensGPS es igual a false). Si es así, se concatena el carácter # a *xbeestring*. Con esto se indica a la estación base que el GPS está desconectado o está fallando. Si está conectado, se concatenan los valores de la hora, la fecha, el enlace, la calidad del enlace y el número de satélites. Además, también se comprueba si hay enlace. En caso de que lo haya, además de estos valores, se concatenan la latitud, la longitud, la altitud, el VTA y *ground speed*.

Finalmente, se concatenan los valores de temperatura y el pH, se envía el *string* a través del puerto serie y se espera a que la transmisión de datos por el puerto serie se haya completado.

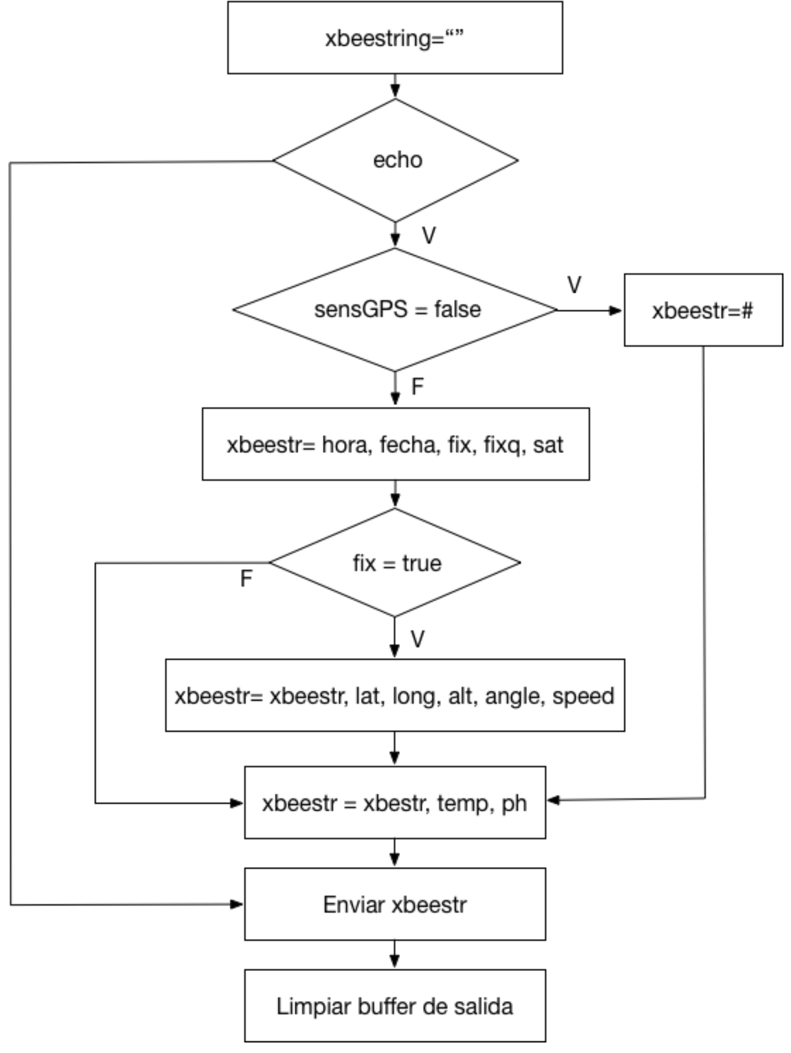

FIGURA 3-6. DIAGRAMA DE FLUJO DE LA FUNCIÓN SENDXBEE().

3.2. Mecanismos software para la reducción del consumo

Como ya se ha visto anteriormente, el funcionamiento de la boya consiste en medir y enviar los datos obtenidos cada cinco minutos. En tomar las mediciones y enviarlas, el sistema invierte cinco segundos. El resto del tiempo que transcurre hasta volver a medir el sistema no hace nada. En el tutorial de Gammon Forum (10) creado por Nick Gammon se investigan diferentes métodos para reducir el consumo del procesador Atmega328P. Algunas de estas técnicas son:

- Correr el procesador a baja frecuencia.
- Correr el procesador con bajo voltaje.
- Desconectar módulos software innecesarios.
- Desconectar la detección *brown-out*.
- Desconectar el ADC (Conversor Analógico Digital).

- Desconectar el WDT.
- Poner el procesador en *sleep mode*.
- Evitar usar reguladores de tensión que sean ineficientes.
- Despertar el procesador del *sleep mode* solo cuando sea necesario.
- Desactivar los dispositivos externos cuando no se utilicen.

Teniendo en cuenta como está configurada la boya y cual debe ser su funcionamiento se ha optado por aplicar las siguientes medidas: desconectar el ADC, desconectar la detección *brown-out*, desconectar módulos software innecesarios, poner el procesador en *sleep mode* despertándolo solo cuando es necesario y desactivar los dispositivos externos.

Desactivación de dispositivos externos

En primer lugar, como se ha explicado anteriormente, el circuito de medición del sensor de pH es puesto en *sleep mode* una vez se ha realizado la medición. Con esto se reduce su consumo de 18.3 mA a 1.16 mA (@5V). El sensor de temperatura se pone automáticamente en *standby* cuando no está en uso y no cuenta con la opción de ponerlo en *sleep mode* por software. Su consumo se reduce de 1 mA estando en estado activo a 750 nA en *standby* (@3.3V).

Sin embargo, a pesar de que el GPS de Adafruit cuenta con la opción de activar el *sleep mode* por *software*, este no es recomendable para este sistema en cuestión, ya que al entrar en *sleep mode* se pierde completamente el enlace con el satélite. Por lo que, cuando vuelve a activarse, necesita tiempos de hasta un minuto para volver a encontrar enlace y su consumo se incrementa de 20 mA a 25 mA (@5V).

Por otro lado, el módulo de comunicación Xbee y el Xbee Shield de Gravi Tech no pueden ser puestos en *sleep mode*.

Desactivación del ADC

Haciendo referencia al *datasheet* del ATmega328P (11), el microcontrolador de la placa Arduino Nano contiene un convertidor analógico a digital de 8 canales. Esto significa que mapeará voltajes de entrada entre 0 y 5 voltios en valores enteros entre 0 y 1023. Esto produce una resolución entre las lecturas de: 5 voltios / 1024 unidades o, 0.0049 voltios (4.9 mV) por unidad.

Según Gammon, desconectando el ADC el consumo disminuye de 335 µA a 0.355 µA. Puesto que todos los módulos y sensores conectados al Arduino Nano van a pines digitales, se puede prescindir del ADC durante el tiempo de espera hasta que se tome la siguiente medición.

Desactivación del detector Brown-Out

El Brown-Out Detector (BOD) se encarga de monitorizar el voltaje de alimentación del microprocesador durante el tiempo en el que se encuentre en *sleep mode*. Esto garantiza un funcionamiento seguro en caso de que el nivel de VCC disminuya durante el período de reposo.

Desde software el modo BOD puede ser desactivado durante los periodos de reposo, volviéndose a activar automáticamente una vez vuelva a despertar con el fin de garantizar que la BOD funcione correctamente antes de que la MCU continúe ejecutando el código.

Con el ATmega328P en SLEEP_MODE_PWR_DOWN y desactivando el BOD se consigue disminuir el consumo de 360 µA to 335 µA.

Desactivación de los módulos software

El Power Reduction Register (PRR) (11) proporciona un método para detener el reloj de los periféricos individuales para reducir el consumo de energía. Los diversos bits en este registro apagan los dispositivos internos de la siguiente manera:

- Bit 7 - PRTWI: Reducción de consumo del TWI
- Bit 6 - PRTIM2: Reducción de consumo del Timer 2
- Bit 5 - PRTIM0: Reducción de consumo del Timer 0
- Bit 4 - Res: bit reservado
- Bit 3 - PRTIM1: Reducción de consumo del Timer 1
- Bit 2 - PRSPI: Reducción de consumo de la interfaz de periféricos serial
- Bit 1 - PRUSART0: Reducción de consumo de USART0
- Bit 0 - PRADC: Reducción de consumo del ADC

Los cambios en este registro pueden hacerse a nivel de bit, sin embargo las macros *power_all_disable()* y *power_all_enable()* modifican este registro de la manera adecuada para cada procesador. Según el *datasheet* del ATmega328P el reloj de todos estos periféricos continua en funcionamiento en los modos Active e Idle, en los demás *sleep modes* estos relojes se desconectan automáticamente.

Los diferentes *sleep modes* del ATmega328P serán comentados a continuación.

Sleep Modes del ATmega328P

A continuación, puede verse una tabla con los diferentes Sleep Modes del ATmega328P y los diferentes métodos para despertarlo de cada uno:

TABLA 3-1. SLEEP MODES DEL ATMEGA328P (11).

	Active Clock Domains					Oscillators		Wake-up Sources							Software BOD Disable
Sleep Mode	clk_{CPU}	clk_{FLASH}	clk_{IO}	clk_{ADC}	clk_{ASY}	Main Clock Source Enabled	Timer Oscillator Enabled	INT and PCINT	TWI Address Match	Timer2	SPM/EEPROM Ready	ADC	WDT	Other I/O	
Idle			Yes	Yes	Yes	Yes	Yes[2]	Yes	Yes	Yes	Yes	Yes	Yes	Yes	
ADC Noise Reduction				Yes	Yes	Yes	Yes[2]	Yes[3]	Yes	Yes[2]	Yes	Yes	Yes		
Power-down								Yes[3]	Yes				Yes		Yes
Power-save					Yes		Yes[2]	Yes[3]	Yes	Yes			Yes		Yes
Standby[1]						Yes		Yes[3]	Yes				Yes		Yes
Extended Standby					Yes[2]	Yes	Yes[2]	Yes[3]	Yes	Yes			Yes		Yes

El consumo (10) para cada uno de estos modos es:

- SLEEP_MODE_IDLE: 15 mA
- SLEEP_MODE_ADC: 6.5 mA
- SLEEP_MODE_PWR_SAVE: 1.62 mA
- SLEEP_MODE_EXT_STANDBY: 1.62 mA
- SLEEP_MODE_STANDBY: 0.84 mA
- SLEEP_MODE_PWR_DOWN: 0.36 mA

El modo de menor consumo es el SLEEP_MODE_PWR-DOWN. Uno de los métodos para despertar el micro de este modo es el WDT.

Uso del WDT (Watchdog Timer)

TABLA 3-2. SELECCIÓN DE PRESCALER DEL WDT.

WDP3	WDP2	WDP1	WDP0	Number of WDT Oscillator Cycles	Typical Time-out at V_{CC} = 5.0V
0	0	0	0	2K (2048) cycles	16 ms
0	0	0	1	4K (4096) cycles	32 ms
0	0	1	0	8K (8192) cycles	64 ms
0	0	1	1	16K (16384) cycles	0.125 s
0	1	0	0	32K (32768) cycles	0.25 s
0	1	0	1	64K (65536) cycles	0.5 s
0	1	1	0	128K (131072) cycles	1.0 s
0	1	1	1	256K (262144) cycles	2.0 s
1	0	0	0	512K (524288) cycles	4.0 s
1	0	0	1	1024K (1048576) cycles	8.0 s
1	0	1	0	Reserved	
1	0	1	1		
1	1	0	0		
1	1	0	1		
1	1	1	0		
1	1	1	1		

El WDT es un temporizador que se encarga de contar los ciclos de un chip interno independiente con un oscilador de 128 kHz. El WDT lanza una interrupción o un reinicio del sistema cuando el contador alcanza un valor de tiempo de espera dado. En un modo de operación normal, es necesario que el programa use la instrucción de reinicio del WDT para que no salte dicha interrupción.

Este WDT puede ser utilizado para despertar al procesador de los distintos modos de reposo vistos en el apartado anterior. El modo de utilizarlo sería ajustando el WDT con uno de los periodos de reinicio que se pueden ver en la Tabla 3-2. El tiempo máximo es de ocho segundos y el mínimo de dieciséis milisegundos. Antes de entrar en el *sleep mode* deseado, se reinicia la cuenta de WDT, entonces se introduce al micro en modo reposo. Cuando haya transcurrido el tiempo indicado el WDT habrá terminado la cuenta, saltando la interrupción y despertando al procesador.

Función sleep()

Una vez vistos los diferentes métodos software para reducir el consumo del sistema es posible proceder a explicar la función *sleep()*. Como se vio en la función *loop()*, esta contiene un bucle *for* que llama un número T de veces a la función *sleep()*. Dicha función es la encargada de poner el procesador en reposo. En este caso, se ha escogido que el procesador entre en SLEEP_MODE_PWR_DOWN siendo despertado a los ocho segundos por el WDT. Además, en esta función también se desactiva el ADC, se desactivan los módulos software y se desactiva el BOD. Introducir a cada sensor y módulo externo en modo reposo se hace en cada una de sus funciones correspondientes.

Cabe comentar que por limitaciones del viejo *bootloader* del Arduino Nano no era posible ajustar el WDT ni el *sleep mode*. Por lo tanto, fue necesario actualizar el *bootloader* del Arduino Nano siguiendo el siguiente tutorial (12). Para ello se utilizó una placa Arduino UNO para actualizar el Arduino Nano a través de la conexión SCPI. Una vez actualizado el *bootloader* es posible hacer uso del WDT. El código final puede verse a continuación

```
1. void sleep(){
2.         byte old_ADCSRA = ADCSRA;
3.         //Desactivacion del ADC
4.         ADCSRA = 0;
5.         // Preparacion del WDT
6.         MCUSR = 0;
7.         //Activar cambios y desactivar el reset
8.         WDTCSR = bit (WDCE) | bit (WDE);
9.         // Ajustar interrupcion e intervalo
10.        WDTCSR = bit (WDIE) | bit (WDP3) | bit (WDP0);
11.        // WDT Reset
12.        wdt_reset();
13.        //Ajuste del sleep mode
14.        set_sleep_mode (SLEEP_MODE_PWR_DOWN);
15.        //Desactivar modulos software
16.        power_all_disable();
```

```
17.         //No permitir interrupciones
18.         noInterrupts ();
19.         //Permitir sleep
20.         sleep_enable();
21.         // Desconexión del brown-out detector
22.         MCUCR = bit (BODS) | bit (BODSE);
23.         MCUCR = bit (BODS);
24.         //Permitir otras interrupciones
25.         interrupts ();
26.         //Dormir CPU
27.         sleep_cpu ();
28.         //No permitir sleep
29.         sleep_disable();
30.         //Activar modulos software
31.         power_all_enable();
32.         //Activar ADC
33.         ADCSRA = old_ADCSRA;
34.       }
```

Para comprobar cuánto se ha conseguido reducir el consumo gracias a este método, se procederá a medir la corriente que consume el sistema cuando se encuentra en activo y cuando entra en *sleep mode*. Para ello, se alimentará el sistema con una fuente ATX que dará 5V. En serie, se conectará un polímetro entre la alimentación de la fuente y el pin de V_{in} del Arduino Nano. Tras varias mediciones, se puede confirmar que el consumo del circuito se reduce de 147,18 mA a 88,55 mA (@5V).

3.3. Diseño software de la estación base

Configuración del Arduino Uno y el módulo Xbee

La estación base está compuesta por un Arduino UNO conectado a un módulo de comunicación Xbee a través de un *shield* de Libelium. Este sistema se conecta al ordenador a través del puerto USB.

Al igual que en el caso de la boya, el *shield* de Libelium para Arduino UNO limita la conexión serie del Arduino al módulo Xbee solo a través de los pines 0 y 1. Ambos pines son los correspondientes al puerto serie principal de Arduino. Por lo tanto, el código que se encuentra corriendo en el Arduino simplemente inicializa el puerto serie, abre la comunicación y espera a que lleguen datos en la función *serialEvent()*. Una vez llegan, los recopila en un *string* y los vuelve a escribir en el puerto serie, enviándolos de este modo al ordenador.

Los datos son recopilados por la interfaz gráfica, la cual los muestra por pantalla y, en el caso que se desee, los escribe en un fichero de texto.

Visión general de la interfaz gráfica

La aplicación que procesa los datos recibidos por el Arduino se ha desarrollado con el lenguaje de programación Python. Dicha aplicación consiste en una GUI (Interfaz Gráfica de Usuario) que abre la comunicación serie con el Arduino y muestra por pantalla los datos recibidos. En caso de que el usuario lo desee, estos datos pueden grabarse en un archivo *.txt* para su posterior procesamiento.

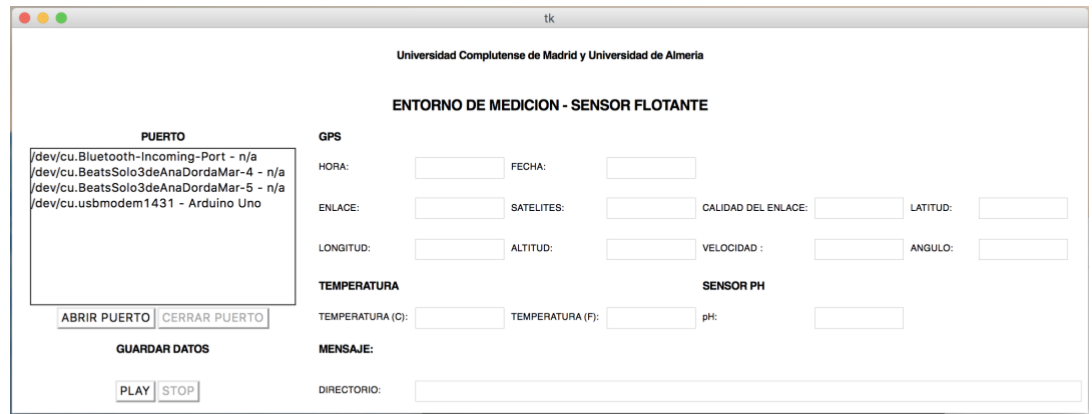

FIGURA 3-7. GUI PARA CONTROL DE MEDICIONES DEL SENSOR FLOTANTE.

En la Figura 3.7 puede verse el entorno de medición cuando se lanza la aplicación. Recuadrado en negro se encuentran los diferentes puertos serie del ordenador. Debajo de este recuadro, los botones para abrir el puerto serie seleccionado. Al lado de esto tenemos todos los campos donde aparecerán las mediciones en tiempo real. Por último, en la parte inferior de la aplicación, los botones para comenzar a guardar los datos y para parar de guardar datos, así como el campo donde aparecerá el directorio donde se encuentra el fichero con los datos.

Esta aplicación ha sido desarrollada haciendo uso de la librería Tkinter, la cual dota de las funcionalidades necesarias para programar GUI's a medida con Python. En la Figura 3.8 puede verse un diagrama de flujo del algoritmo que subyace bajo dicha aplicación. Partiendo del estado inicial donde los botones *Cerrar Puerto* y *Stop* están desactivados, se puede optar por pulsar el botón *Abrir Puerto* o *Play*. En el caso de que se pulse *Play*, se llamará a la función *save()*, la cual hará que salte un *pop-up* para que se escoja el directorio donde se quiere guardar el archivo donde se escribirán las mediciones. Una vez escogido el directorio, este aparecerá escrito en el campo *Directorio* de la aplicación y se activará el botón de *Stop*. Si se pulsa el botón de *Stop*, de nuevo se ejecutará la función *save()*, la cual indicará al proceso de lectura y escritura que deje de grabar los datos en el fichero de texto. Además, ambos botones volverán a sus estados iniciales.

Por otro lado, si se pulsa el botón de *Abrir Puerto* y no se ha seleccionado puerto, saldrá un mensaje en el campo *Mensaje* indicando que se seleccione un puerto. Una vez seleccionado un puerto de la lista, se puede proceder a abrir el puerto. Automáticamente se activará el botón *Cerrar Puerto* y saltará un mensaje indicando que se está estableciendo

la comunicación. Es en este punto donde se generará un hilo, el cual revisa constantemente el puerto serie para recoger datos, mostrarlos por pantalla y, dado el caso, generar el fichero con los datos medidos. La función *writeGUI()* es la encargada de escribir las mediciones en su correspondiente campo de la aplicación. En el caso de que se haya pulsado el botón *Play*, además de mostrar los datos por pantalla, también llamará a la función *writeFile()* la cual escribirá las mediciones en un fichero *.txt*.

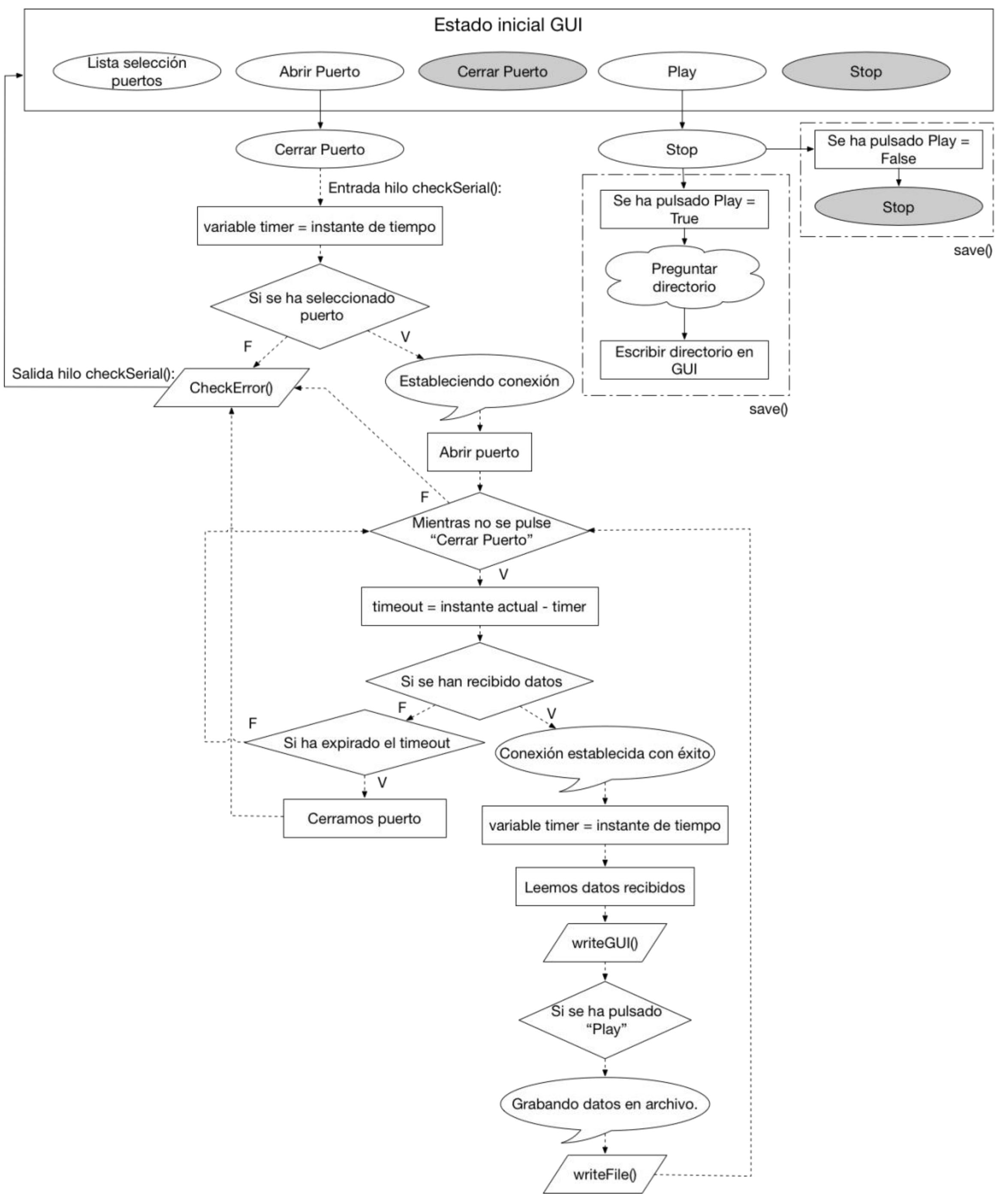

FIGURA 3-8. DIAGRAMA DE FLUJO DE LA INTERFAZ GRÁFICA.

Ahora bien, este bucle de lectura y escritura puede ser interrumpido de dos maneras. La primera es pulsando el botón *Cerrar Puerto*. Haciendo esto el programa sale del bucle *while*,

se cierra la comunicación serie, se imprime un mensaje indicando que la comunicación se ha cerrado con éxito y se mata el hilo. El otro método, orientado a detectar la perdida de comunicación con el sensor flotante, es generar un *timeout*. Antes de abrir el hilo se almacena ese instante de tiempo en la variable *timer*. Dentro del bucle *while* se comprueba en cada iteración el tiempo que ha transcurrido desde que se entró en el bucle. En caso de recibir datos, la variable *timer* se sobrescribe con un nuevo instante de tiempo. Si por el contrario no se reciben datos y se supera un valor de tiempo prefijado se sale del bucle, se cierra el puerto serie, salta un mensaje indicando que existe un error de conexión y se mata el hilo.

El hecho de que el bucle de escritura y lectura se encuentre en un hilo permite que al mismo tiempo se puedan ejecutar otros procesos en la aplicación, como pulsar los diferentes botones y ejecutar las funciones asociados a estos. Las funciones *writeGUI(), writeFile()* y *checkError(),* llamadas por el hilo, tienen las siguientes funcionalidades:

- **writeGUI():** se encarga de ordenar los datos recibidos e imprimirlos por pantalla. Para indicar al usuario si los sensores están funcionando correctamente se comprueban los valores recibidos. En el caso del GPS, si se recibe un # seguido de las medidas de temperatura y pH, por pantalla se imprimirán # en todos los campos relacionados con el GPS. Si por el contrario se reciben la fecha, la hora, el enlace, la calidad del enlace y los satélites, todos ellos seguidos de mediciones de temperatura y pH, en el resto de las mediciones del GPS no recibidas se imprimirá un #. Por otro lado, si la medida de pH es igual a 88 o si la medida de temperatura en grados centígrados es igual a -127, se imprimirá en uno o en otro un #.
- **writeFile():** esta función se encarga de generar el fichero de texto en el directorio escogido por el usuario y escribir en él las mediciones según se reciben. El nombre del fichero generado será la fecha indicada por el GPS. En la Figura 3.9 puede verse un ejemplo del fichero generado.
- **checkError():** a través de sendas variables booleanas esta función se encarga de determinar el motivo por el que se ha cerrado la comunicación, imprimir por pantalla el mensaje correspondiente y devolver a la interfaz gráfica a su estado inicial.

```
20-8-2018.txt
Hora, Fecha, Enlace, Calidad del enlace, Satelites, Latitud, Longitud, Altitud, Velocidad, Angulo, TempC, TempF, pH
20:21:27.0, 20-8-2018, 1, 1, 4, 40.403263, -3.668855, 214.90, 1.14, 252.71, 31.88, 89.38, 0.00

20:21:42.0, 20-8-2018, 1, 1, 4, 40.403297, -3.668885, 214.90, 1.29, 198.09, 31.88, 89.38, 0.00

20:21:56.984, 20-8-2018, 1, 1, 4, 40.403198, -3.668823, 216.80, 0.74, 334.70, 31.88, 89.38, 0.00

20:22:12.0, 20-8-2018, 1, 1, 4, 40.403152, -3.668839, 220.70, 1.39, 66.21, 31.88, 89.38, 0.00

20:22:27.0, 20-8-2018, 1, 1, 4, 40.403175, -3.668790, 225.00, 1.63, 9.37, 31.88, 89.38, 0.00

20:26:12.0, 20-8-2018, 1, 1, 4, 40.403023, -3.668782, 243.10, 0.86, 62.53, 31.88, 89.38, 0.00

20:27:40.984, 20-8-2018, 1, 1, 6, 40.401768, -3.668112, 270.20, 3.46, 213.33, 31.88, 89.38, 0.00
```

FIGURA 3-9. EJEMPLO DE FICHERO GENERADO POR LA APLICACIÓN.

4. Montaje del prototipo y pruebas

4.1. Planteamiento del diseño

Una vez programado el funcionamiento de la boya se puede proceder al montaje del prototipo. Para proteger el sistema, la carcasa que lo contiene debe ser completamente estanca. Además, debe de moverse con el curso del agua para tomar mediciones en todo el fotobiorreactor. Como ya se ha explicado anteriormente, en los fotobiorreactores de tipo *raceway* se produce agitación y mezcla. Esto es provocado por una rueda de palas que se encarga de remover el agua. El sensor flotante, debe ser capaz de resistir a la agitación provocada por dichas palas, siendo lo suficientemente estable como para no volcar.

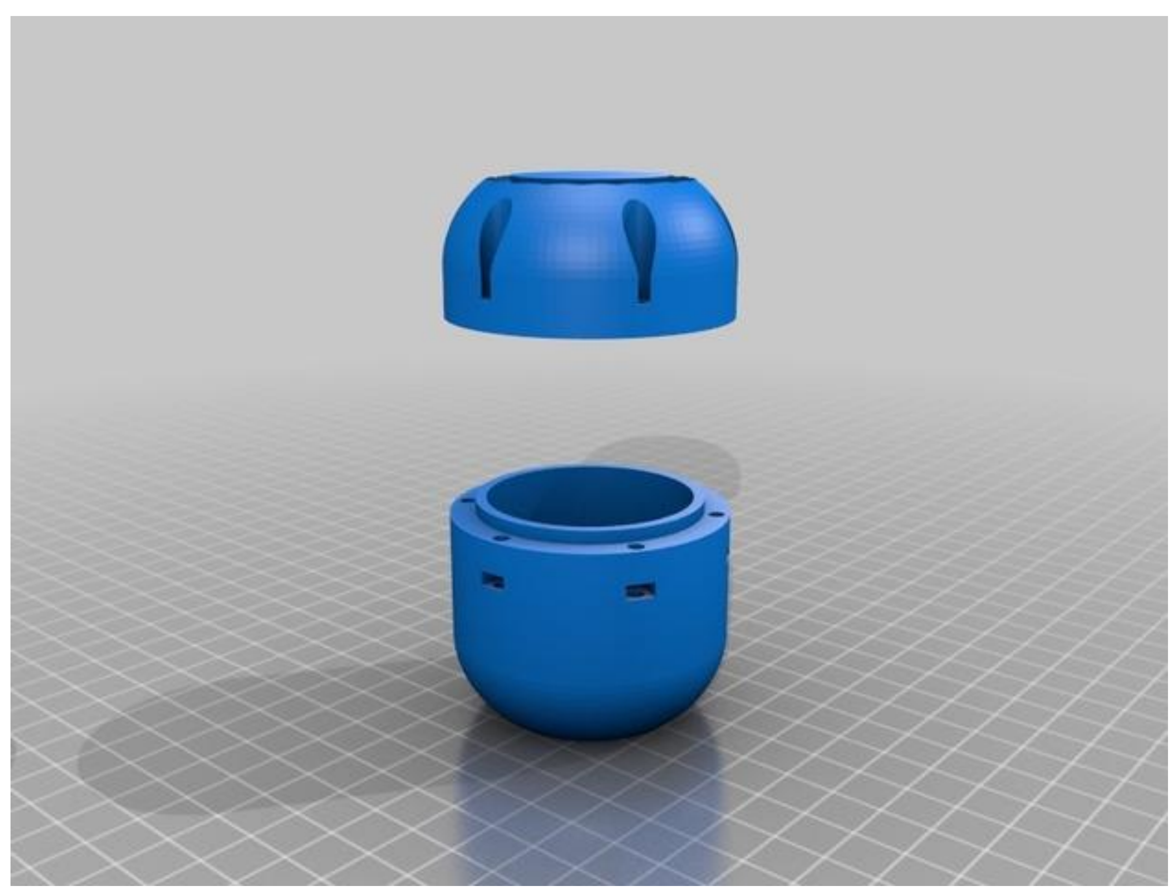

FIGURA 4-1. IDEA INICIAL PARA LA CARCASA DE LA BOYA (13).

Teniendo en cuenta todas estas especificaciones, en un primer momento para el diseño se partió de la idea de usar una impresora 3D para imprimir la carcasa que albergaría el circuito. Esta carcasa contendría el circuito con una disposición vertical. Sin embargo, el sensor de pH es excesivamente alargado como para pasar sin atascarse por las palas del fotobiorreactor. Por lo que, tanto por complejidad como por falta de tiempo se descartó esta primera idea.

La segunda idea, la cual se convirtió en el diseño escogido, se trata de una caja completamente estanca, de un tamaño lo suficientemente grande como para albergar toda la circuitería, pero permitiendo que pase sin problema por las palas. El circuito tendrá una disposición horizontal, en la cual el sensor de pH irá fijado en la parte inferior de la boya.

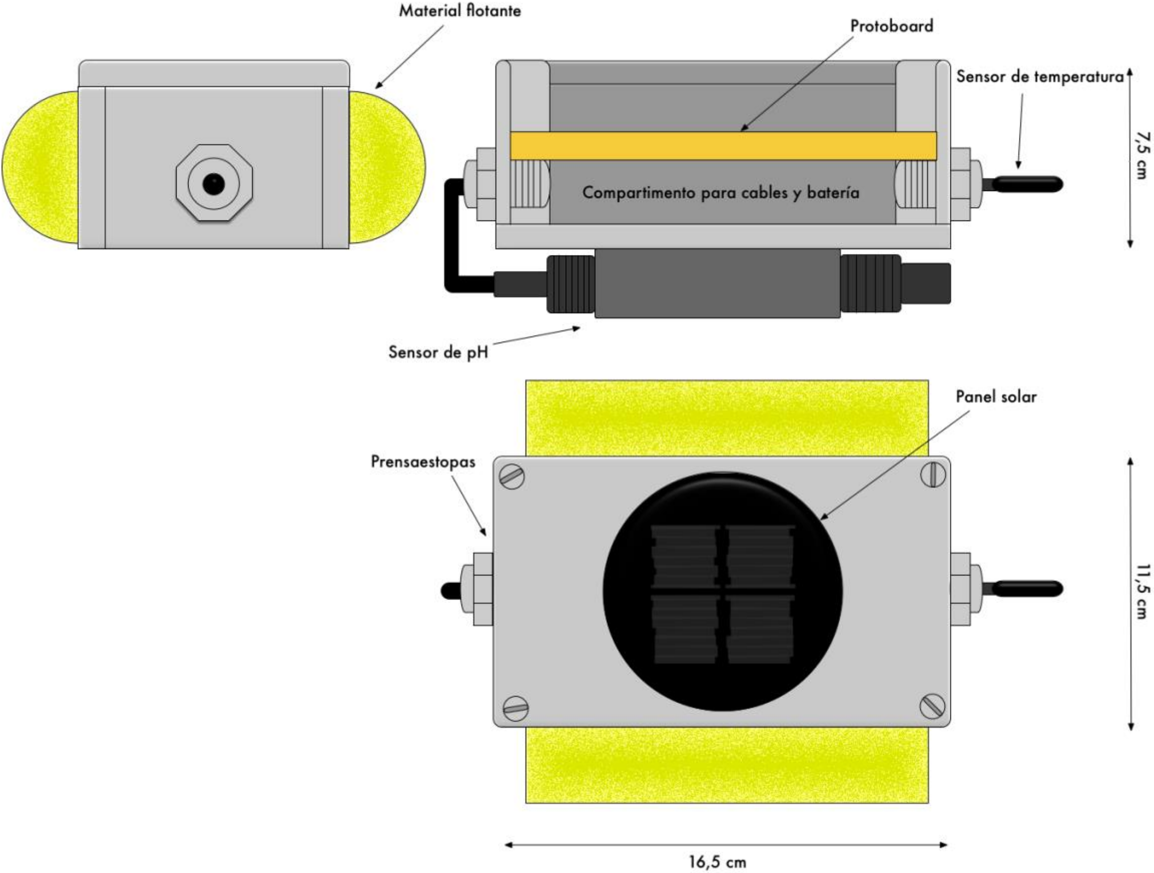

FIGURA 4-2. PLANTEAMIENTO DEL DISEÑO DEL SENSOR FLOTANTE PARA EL CULTIVO DE MICROALGAS.

El sensor de temperatura y el sensor de pH deben estar en contacto directo con el agua y conectados al circuito interno de la caja estanca. Para conseguir esto se utilizarán dos prensaestopas que se pondrán en los lados mas estrechos de la caja contenedora. Estos permiten pasar el cable de ambos sensores fuera de la caja estanca sin que entre el agua dentro. Por otro lado, en la tapa irá fijado con silicona el panel solar. Para incrementar la estabilidad de la boya en el fotobiorreactor, se fijarán dos piezas de un material con elevada flotabilidad en las paredes mas largas de la caja estanca.

En lo que corresponde al interior de la caja estanca, todo el circuito se conectará en una *protoboard.* Dicha *protoboard* se utilizará para dividir el interior en dos compartimentos superior e inferior. Dentro del inferior se encontrarán la batería y los cables de los sensores de temperatura y pH.

4.2. Montaje del sensor flotante

La caja estanca escogida para albergar el diseño es de la marca JSL modelo J160-L cuyas dimensiones son 16,5x11,5x7,5 cm con nivel de protección IP 66. El grado de protección IP (Ingress Protection) (14) hace referencia a la norma internacional CEI 60529 utilizado con mucha frecuencia en los datos técnicos de equipamiento eléctrico o electrónico. Se utiliza para clasificar los diferentes grados de protección aportados a los equipos eléctricos por los contenedores que resguardan los componentes que constituyen el equipo. En el caso de IP 66, el primer 6 hace referencia a la protección frente al polvo la cual es: "*El polvo no entra bajo ninguna circunstancia*". El segundo 6 hace referencia a la protección frente al agua la cual es: *"No debe entrar el agua arrojada a chorros (desde cualquier ángulo) por medio de una boquilla de 12,5 mm de diámetro, a un promedio de 100 litros por minuto y a una presión de 100 kN/m² durante no menos de 3 minutos y a una distancia que no sea menor de 3 metros"*.

FIGURA 4-3. CAJA ESTANCA J160L DE JSL CON IP66.

Puesto que este nivel de protección no garantiza la seguridad del equipo si se somete a una inmersión se hicieron modificaciones en la caja para aumentar su IP. Usando la silicona T-Rex Turbo de la marca SOUDAL se sellaron los agujeros por los que podía entrar el agua. Dicha silicona de secado rápido permite pegar todo tipo de materiales de construcción tanto fuera como dentro del agua, está indicada para uso exterior e interior y gracias a su flexibilidad es muy resistente.

Para pasar los cables de los sensores de temperatura y pH hacia el exterior se utilizaron dos prensaestopas de la marca Legrand con nivel de protección IP68. Es decir, protegen al equipo electrónico de una inmersión completa. Para poder incorporarlos a la caja se hicieron dos taladros en ambos lados de la caja y se fijaron los prensaestopas con la silicona antes mencionada. Para incrementar la seguridad, la parte de los cables que iba a ser presionada por los prensaestopas se cubrió con cinta para reparación de mangueras.

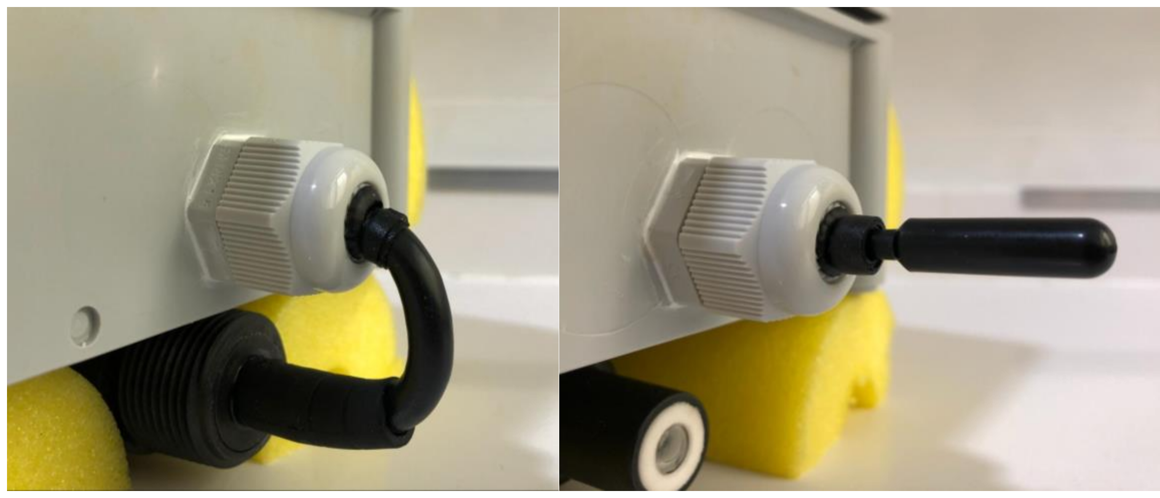

FIGURA 4-4. PRENSAESTOPAS DEL SENSOR DE pH (IZQUIERDA) Y PRENSAESTOPAS DEL SENSOR DE TEMPERATURA (DERECHA).

Una vez lista la caja que albergará el sistema se pasó a situar el circuito en su interior. Todo este va montado en una *protoboard* que va fijada en el interior sobre los prensaestopas. Debajo de la *protoboard* va pegada la batería de polímero de litio que alimenta el sistema. Los cables de los sensores de pH y temperatura fueron cortados y pelados para ajustar su longitud a las dimensiones de la caja. Una vez pasados por los prensaestopas los cables se situaron en el compartimento inferior, bajo la *protoboard*, para evitar que estorbaran al resto del sistema. En la tapa, fijada con silicona, se sitúa la placa solar. Debajo de la caja, fijado con cinta americana, se encuentra el sensor de pH.

La caja estanca se cierra a través de unos tornillos y se evita que el entre el agua gracias a una junta de protección. Tras realizar las primeras pruebas en el agua se optó por, con el fin de incrementar la estabilidad rotacional del sensor flotante, fijar a cada lado de la caja la mitad de un flotador para natación.

FIGURA 4-5. VISIÓN GENERAL DEL SENSOR FLOTANTE.

4.3. Pruebas finales

Metodología

Para probar la fiabilidad del sistema este se puso a prueba en una de las piscinas de Madrid Río. La idea de las pruebas es demostrar que el sensor es completamente estanco, pudiendo aguantar sumergido en el agua durante horas. Además, se pretende comprobar el correcto funcionamiento tanto del sensor como de la estación base y la interfaz gráfica, así como la fiabilidad del GPS. Para ello se realizarán mediciones durante una hora, las cuales serán recogidas en un fichero para su posterior procesamiento. Con esto se pretende demostrar la utilidad del sensor.

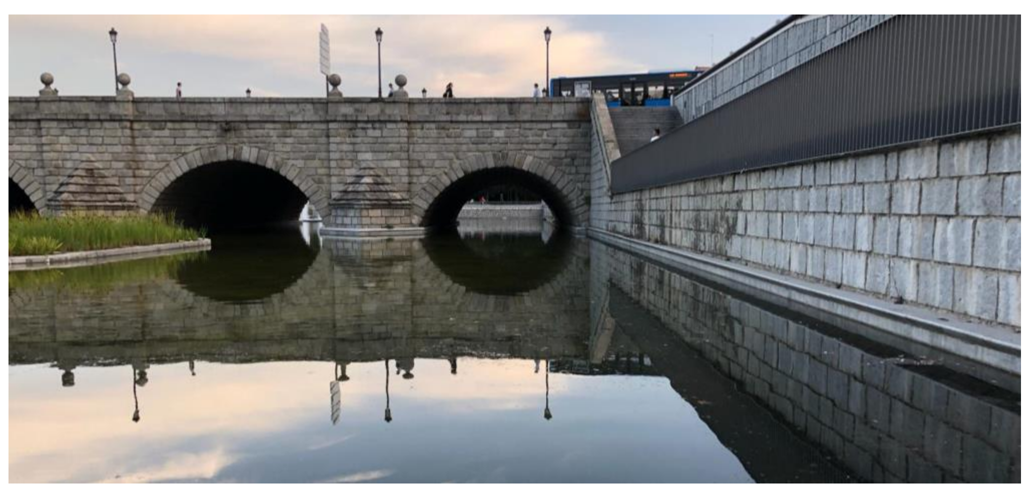

FIGURA 4-6. PISCINA DE MADRID RÍO DONDE SE REALIZARON LAS MEDICIONES.

La estación base se montó a un par de metros de distancia. Esta estaba constituida por un ordenador MacBook Pro, en el cual estaba corriendo la interfaz gráfica, conectado al Arduino Uno con comunicación Zigbee. Por otro lado, puesto que en estas piscinas no existen corrientes, fue necesario atar un cordel al sensor para poder moverlo por el agua y realizar mediciones en distintos puntos de la piscina.

FIGURA 4-7. ESTACIÓN BASE (IZQUIERDA) Y SENSOR FLOTANTE (DERECHA).

Procesamiento de las mediciones

Se realizaron un total de doscientas mediciones. Todas ellas fueron importadas del archivo .txt a MATLAB para su procesamiento. Lo primero que se comprobó fue la precisión del GPS. Para ello, haciendo uso de la API de Google Maps para MATLAB, se representaron la longitud y la latitud medidas por el GPS sobre el mapa. En la Figura 4-8, puede verse recuadrado en amarillo la zona en la que se realizaron las mediciones y en rojo los puntos en los que se realizaron las mediciones según el GPS. Como puede verse existe una clara desviación entre la zona en la que se estuvieron realizando las mediciones y donde el sensor GPS indica que se realizó cada una de ellas. Además, también existe una dispersión entre los puntos, estando más distanciados entre si de lo que en realidad estaban.

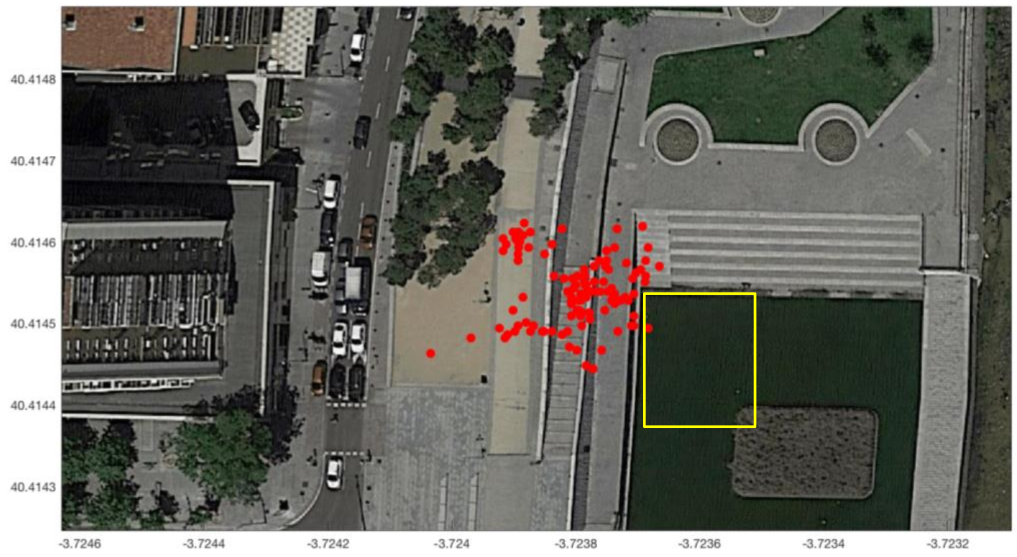

FIGURA 4-8. EN ROJO, PUNTOS EN LOS QUE SE REALIZARON LAS MEDICIONES SEGÚN EL GPS. EN AMARILLO, ZONA EN LA QUE SE REALIZARON LAS MEDICIONES.

Una vez analizados los datos recopilados por el GPS, se representaron las mediciones tanto de pH como de temperatura en función de la posición. Para ello, se hace uso de la función *scatter3()* de MATLAB. La representación de las mediciones realizadas puede verse en la Figura 4-9.

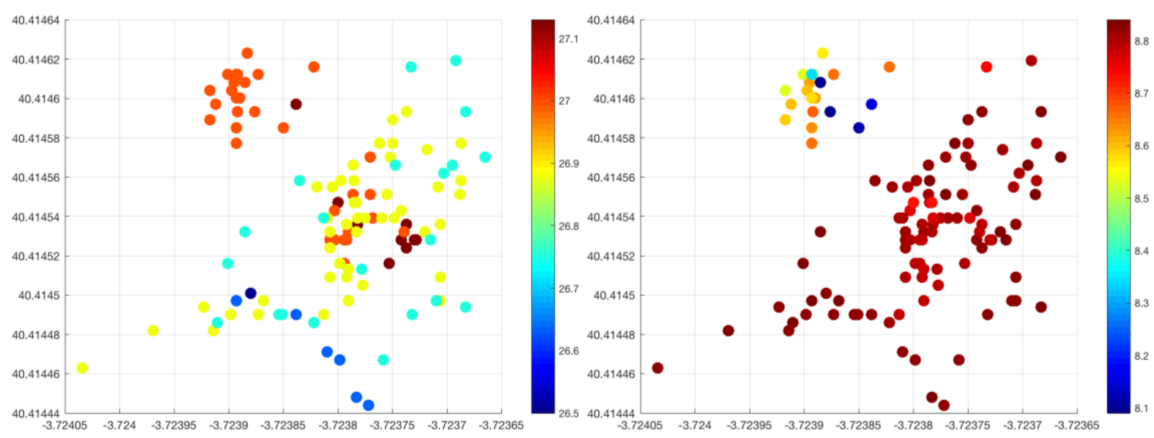

FIGURA 4-9. IZQUIERDA, MEDICIONES DE TEMPERATURA. DERECHA, MEDICIONES DE PH.

5. Conclusión

A lo largo de este trabajo se ha seguido el proceso de diseño, programación y construcción de un prototipo de sensor flotante que ayude en el cultivo de microalgas. Como en la creación de cualquier prototipo del que no existen una gran cantidad de referencias previas, se ha tenido que hacer frente a ciertas limitaciones tanto en la elección de componentes, como en la programación o su construcción.

No obstante, se ha obtenido un prototipo satisfactorio. Su tamaño es compacto y la caja estanca protege completamente al circuito. Además, en las pruebas realizadas, se comprueba que el funcionamiento tanto del sensor como de la estación base es el esperado. El sensor realiza mediciones de manera periódica, disminuyendo dicho periodo cuando detecta rangos de pH o temperatura fuera de lo óptimo para el cultivo de las microalgas. Durante el tiempo en el que el sensor no mide, este entra en *sleep mode,* lo que disminuye su consumo a un 60% del consumo en activo. Añadido a esto, gracias a la placa USB / DC / Solar Lithium Ion/Polymer charger y al panel solar, la batería de polímero de litio se carga con la luz solar, lo que aumenta su autonomía.

Dichas mediciones son enviadas a la estación base. Gracias a las mediciones de temperatura y pH en función de la posición se pueden realizar representaciones como las de las Figuras 4-9 y 4-10. Por ejemplo, en esas figuras, en zonas cercanas a la orilla se puede ver como la temperatura aumenta y el pH disminuye. Sin embargo, según nos alejamos a la orilla, la temperatura comienza a disminuir y el pH a aumentar.

Este tipo de gráficas pueden ser de gran utilidad en el estudio de los fotobiorreactores para detectar con mayor precisión las zonas donde el pH o la temperatura queda fuera de los rangos óptimos. Además, realizando mediciones durante largos periodos de tiempo, a distintas horas del día o distintas épocas del año, se pueden obtener una gran cantidad de datos que podrán ser posteriormente procesados haciendo uso de métodos relacionados con la minería de datos.

Ahora bien, en las pruebas también se detectaron ciertos defectos. Las posiciones medidas por el GPS presentan desviación, esto es normal ya que como se puede ver en la Tabla 2-2 la exactitud del módulo GPS de Adafruit es de 3 metros. Sin embargo, se podría mejorar comprando una antena que amplifique la señal. Además, también sería conveniente probar el sensor en un fotobiorreactor ya que, al tener una forma definida, se podría comprobar si se mantiene la trayectoria a pesar de la desviación en la medición.

Por otro lado, este sensor puede ser mejorado con el fin de optimizar y reducir su diseño. El sensor de pH, como ya se ha visto, es el componente más complicado de ubicar en el sensor debido a su tamaño. Sin embargo, recientemente Atlas Scientific ha sacado al mercado sus nuevos microsensores de pH de tamaño similar al de una moneda; a parte de que sigue existiendo la opción de usar un sensor de pH basado en tecnología ISFET. Por

otro lado, una vez comprobado el correcto funcionamiento del prototipo se podría considerar sustituir el Arduino Nano y el módulo Zigbee por un micro con la comunicación Zigbee integrada como el Atmel ATZB-24-A2/B0. Con respecto a la carcasa, sería conveniente rediseñarla e imprimirla con una impresora 3D.

Haciendo estos cambios se podría conseguir un diseño más compacto del sensor el cual, a largo plazo, podría incluso usarse para medir dentro de fotobiorreactores cerrados de tipo tubular.

Bibliografía

1. **Fernandez Berenguel, Maria Dolores.** Modelado y Control de Fotobiorreactores Industriales. *Trabajo Fin de Grado.* Almería : s.n., 2014, págs. 11-13.

2. **Abalde Alonso, Julio.** *Microalgas: cultivo y aplicaciones.* s.l. : Monografías, 1995. ISBN 84-88301-84-7.

3. **UAL.** Ingenieria de Procesos aplicada a la Biotecnología de Microalgas. *1.7 - Fotobiorreactores para el cultivo masivo de microalgas.* [En línea]

4. **Carreño, J.J, Guzmán, J.L, Moreno, J.C y Villamizar, R.** *Control Robusto del pH en Fotobiorreactores mediante Rechazo Activo de Perturbaciones.* Jornadas de Automática : ResearchGate, 2017. págs. 1-6.

5. **Pawlowski, A., Mendoza, J., Guzman, J., Berenguel, M., Acien, F., y Dormido, S.** Effective utilization of flue gases in raceway reactor with event-based ph control for microalgae culture. *Bioresource Technology.* 2014, Vol. 170, págs. 1-9.

6. **GRAVITECH.** XBee add-on for Arduino Nano. *SCH-XBEE-4NANO V1.0.* [En línea]

7. **DePriest, Dale.** GPS Information. *NMEA data.* [En línea]

8. **Lady Ada.** Adafruit Ultimate GPS Datasheet. *pag 13.* [En línea]

9. **Atlas Scientific.** pH Circuit EZO Datasheet. *pag 37.* [En línea]

10. **Gammon, Nick.** Gammon Forum. *Power saving techniques for microprocessors.* [En línea]

11. **ATMEL.** ATmega328/P Datasheet. *Datasheet Complete.* [En línea]

12. **Alex74.** Domotica-Arduino. *Actualizar bootloader Arduino NANO.* [En línea]

13. **Denko.** Thingverse. *thing:516797.* [En línea]

14. **Ministerio de Tecnología y Ciencia.** Guía Técnica de Aplicación-Anexos. *Significado y Explicación de los Códigos IP, IK.* 2003, pág. 2.

15. **UAL.** Ingenieria de Procesos aplicada a la Biotecnología de Microalgas. *1.2 - Cultivo de microalgas: laboratorio y gran escala.* [En línea]

Anexo

Anexo I: código transmisor (sensor flotante)

```
1.  // Declaración de las librerias
2.  #include <OneWire.h>
3.  #include <DallasTemperature.h>
4.  #include <Adafruit_GPS.h>
5.  #include <SoftwareSerial.h>
6.  #include <avr/sleep.h>
7.  #include <avr/wdt.h>
8.  #include <avr/power.h>
9.
10. // Interrupción con el watchdog timer
11. ISR (WDT_vect)
12. {
13.    wdt_disable();
14. }
15.
16.
17. //Struct con las mediciones
18. struct DATA
19. {
20.   int gps_h, gps_m, gps_s, gps_ms;
21.   int gps_day, gps_mo, gps_y;
22.   int gps_sat, gps_fix, gps_fixq;
23.   char latitud[10];
24.   char longitud[10];
25.   float gps_lat, gps_lon, gps_alt, gps_speed, gps_angle;
26.   float ds_c, ds_f;
27.   String sensorstring = "";
28.   boolean sensor_stringcomplete = false;
29.   float ph;
30.   boolean pH_event;
31.   boolean temp_event;
32. };
33.
34. DATA sample;
35.
36. //Declaracion de otras variables auxiliares
37. boolean echo = true;
38. int debug_rate = 9600;
39. String xbeestring = "";
40. unsigned long current_time;
41. boolean sensGPS = true;
42. float segundos=8;
43. int T=round(segundos/8);
44.
45.
46. // Indicamos los pines a los que se encuentran conectados los
    sensores
47. #define ONE_WIRE_BUS 4
48. OneWire oneWire(ONE_WIRE_BUS);
49. DallasTemperature DS18B20(&oneWire);
50. DeviceAddress tempDeviceAddress;
51.
```

```
52. SoftwareSerial
    pHsensor(5, 6);

53. Adafruit_GPS GPS(&AdaGPS);
54.
55.
56. void setup()
57. {
58.   pinMode(LED_BUILTIN, OUTPUT);
59.
60.
61.   Serial.begin(debug_rate);
62.
63.   Serial.println("Start");
64.
65.   // DS18B20
66.   DS18B20.begin();
67.   DS18B20.getAddress(tempDeviceAddress, 0);
68.   DS18B20.setResolution(tempDeviceAddress, 11);
69.
70.   // GPS
71.   GPS.begin(9600);
72.   GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCGGA);
73.   GPS.sendCommand(PMTK_SET_NMEA_UPDATE_1HZ);
74.
75.   // pH
76.   sample.sensorstring.reserve(30);
77. }
78.
79.
80. void loop()
81. {
82.     readDS18B20();
83.     readGPS();
84.     readpH();
85.     sendXbee();
86.     digitalWrite(LED_BUILTIN, LOW);
87.     if(!sample.pH_event & !sample.temp_event){
88.       segundos = 300;
89.     }
90.     else{
91.       segundos = 8;
92.     }
93.     T=round(segundos/8);
94.     for (int i=0; i < T; i++)
95.     {
96.       sleep();
97.     }
98.     digitalWrite(LED_BUILTIN, HIGH);
99. }
100.
101.
102.   void readDS18B20()
103.   {
104.       DS18B20.requestTemperatures();
105.       sample.ds_c = DS18B20.getTempCByIndex(0);
106.       sample.ds_f = (sample.ds_c * 9.0) / 5.0 + 32;
107.       if ((sample.ds_c > 25) | (sample.ds_c < 18)){
108.         sample.temp_event=true;
109.       }
110.       else{
```

```
111.        sample.temp_event=false;
112.      }
113.  }
114.
115.
116.  void readGPS()
117.  {
118.    AdaGPS.listen();
119.    for ( int i = 0; i < 2 ; i++){
120.      current_time=millis();
121.      while(!GPS.newNMEAreceived()){
122.        char c = GPS.read();
123.        if (millis()-current_time > 1000){
124.          sensGPS = false;
125.          break;
126.        }
127.        else {
128.          sensGPS = true;
129.        }
130.      }
131.
132.      if (!sensGPS){
133.        break;
134.      }
135.      GPS.parse(GPS.lastNMEA());
136.    }
137.
138.    if (sensGPS){
139.      sample.gps_h = GPS.hour+2;
140.      sample.gps_m = GPS.minute;
141.      sample.gps_s = GPS.seconds;
142.      sample.gps_ms = GPS.milliseconds;
143.      sample.gps_day = GPS.day;
144.      sample.gps_mo = GPS.month;
145.      sample.gps_y = GPS.year;
146.      sample.gps_sat = (int)GPS.satellites;
147.      sample.gps_fix = (int)GPS.fix;
148.      sample.gps_fixq = (int)GPS.fixquality;
149.      sample.gps_lat = GPS.latitudeDegrees;
150.      sample.gps_lon = GPS.longitudeDegrees;
151.      sample.gps_alt = GPS.altitude;
152.      sample.gps_angle = GPS.angle;
153.      sample.gps_speed = GPS.speed;
154.      sample.gps_speed *=   1.15;
155.
156.      dtostrf(sample.gps_lat,2,6,sample.latitud);
157.      dtostrf(sample.gps_lon,2,6,sample.longitud);
158.    }
159.  }
160.
161.
162.  void readpH()
163.  {
164.    pHsensor.listen();
165.
166.    for (int i=0; i <= 4; i++){
167.      pHsensor.println('R');
168.      delay(1000);
169.      while (pHsensor.available() > 0) {
170.        char inchar = (char)pHsensor.read();
171.        sample.sensorstring += inchar;
```

```
172.         if (inchar == '\r') {
173.           sample.sensor_stringcomplete = true;
174.         }
175.       }
176.       if(i<=3){
177.         sample.sensorstring ="";
178.         sample.sensor_stringcomplete = false;
179.       }
180.     }
181. 
182.     if (sample.sensor_stringcomplete) {
183.       sample.ph = sample.sensorstring.toFloat();
184.     }
185.     else{
186.       sample.ph = 88;
187.     }
188. 
189.     sample.sensorstring ="";
190.     sample.sensor_stringcomplete = false;
191.     pHsensor.println("Sleep");
192.     if ((sample.ph > 8) | (sample.ph < 7)){
193.        sample.pH_event=true;
194.      }
195.      else{
196.        sample.pH_event=false;
197.      }
198.   }
199. 
200. 
201.   void sendXbee()
202.   {
203.      xbeestring = String();
204.      if (echo){
205.       if (!sensGPS){
206.         xbeestring = xbeestring + "#";
207.       }
208.       else{
209.         xbeestring = xbeestring+sample.gps_h+':'+
210.         sample.gps_m+':'+sample.gps_s+'.'+
211.         sample.gps_ms+", "+sample.gps_day+
212.         '-'+sample.gps_mo+"-20"+sample.gps_y+
213.         ", "+sample.gps_fix+", "+sample.gps_fixq+
214.         ", "+sample.gps_sat;
215.         if (sample.gps_fix) {
216.           xbeestring = xbeestring+", "+sample.latitud
217.           +", "+sample.longitud+", "+sample.gps_alt+
218.           ", "+sample.gps_speed+", "+sample.gps_angle;
219.         }
220.       }
221.       xbeestring = xbeestring+", "+sample.ds_c+", "+
222.       sample.ds_f+", "+sample.ph;
223.      }
224. 
225.      Serial.println(xbeestring);
226.      Serial.flush();
227. 
228.   }
229. 
230. 
231.   void sleep(){
232.     byte old_ADCSRA = ADCSRA;
```

```
233.     //Desactivacion del ADC
234.     ADCSRA = 0;
235.     // Preparacion del WDT
236.     MCUSR = 0;
237.     //Activar cambios y desactivar el reset
238.     WDTCSR = bit (WDCE) | bit (WDE);
239.     // Ajustar modo de interrupcion y el intervalo (8
    segundo)
240.     WDTCSR = bit (WDIE) | bit (WDP3) | bit (WDP0);
241.     // WDT Reset
242.     wdt_reset();
243.     //Ajuste del sleep mode
244.     set_sleep_mode (SLEEP_MODE_PWR_DOWN);
245.     //Desactivar modulos software
246.     power_all_disable();
247.     //No permitir interrupciones
248.     noInterrupts ();
249.     //Permitir sleep
250.     sleep_enable();
251.     // Desconexión del brown-out detector
252.     MCUCR = bit (BODS) | bit (BODSE);
253.     MCUCR = bit (BODS);
254.     //Permitir otras interrupciones
255.     interrupts ();
256.     //Dormir CPU
257.     sleep_cpu ();
258.     //No permitir sleep
259.     sleep_disable();
260.     //Activar modulos software
261.     power_all_enable();
262.     //Activar ADC
263.     ADCSRA = old_ADCSRA;
264.   }
```

Anexo II: código receptor

```
1.  #include <SoftwareSerial.h>
2.  #include <stdio.h>
3.
4.  String inputString = "";
5.  boolean stringComplete = false;
6.
7.
8.  void setup()
9.  {
10.   Serial.begin(9600);
11.   inputString.reserve(200);
12. }
13.
14.
15. void loop() {
16.   if (stringComplete) {
17.     Serial.println(inputString);
18.     inputString = "";
19.     stringComplete = false;
20.   }
21. }
22.
23. void serialEvent() {
```

```
24.   while (Serial.available()) {
25.     char inChar = (char)Serial.read();
26.     inputString += inChar;
27.     if (inChar == '\n') {
28.       stringComplete = true;
29.     }
30.   }
31. }
32.
```

Anexo III: código interfaz de la estación base

```
1.  import serial
2.  from Tkinter import *
3.  import numpy
4.  import time
5.  import serial.tools.list_ports
6.  import tkFileDialog
7.  import threading
8.  import os
9.
10.
11. #Global values
12. master = Tk()
13. comm = IntVar()
14. comm.set(0)
15. nocomm = IntVar()
16. nocomm.set(0)
17. record = IntVar()
18. record.set(0)
19. stoprecord = IntVar()
20. stoprecord.set(0)
21. foldvar=StringVar()
22. gps_hvar=StringVar()
23. gps_datvar=StringVar()
24. gps_satvar=StringVar()
25. gps_fixvar=StringVar()
26. gps_fixqvar=StringVar()
27. gps_latvar=StringVar()
28. gps_lonvar=StringVar()
29. gps_altvar=StringVar()
30. gps_speedvar=StringVar()
31. gps_anglevar=StringVar()
32. ds_cvar=StringVar()
33. ds_fvar=StringVar()
34. phvar=StringVar()
35. errclose = False
36. errconnection = False
37. errconnectport = False
38.
39.
40. def callback():
41.         off.config(state='normal')
42.         hilo1 = threading.Thread(target=checkSerial)
43.         hilo1.start()
44.
45. def checkSerial():
46.         global errclose
47.         global errconnection
48.         global errconnectport
```

```
49.         timer = time.time()
50.         item = listbox.curselection()
51.         if item:
52.                 error.config(text='MENSAJE: Estableciendo
    conexion...', foreground="blue")
53.                 port= str(ports[item[0]])
54.                 port = port.split(' ')
55.                 port = str(port[0])
56.                 s = serial.Serial(port=port, baudrate=9600)
57.                 while (errclose!=True):
58.                         timeout = time.time()-timer
59.                         if(s.in_waiting>0):
60.                                 error.config(text='MENSAJE: Conexion
    establecida con exito!', foreground="green")
61.                                 timer=time.time()
62.                                 read=s.readline()
63.                                 measures = read.split(', ')
64.                                 writeGUI(measures)
65.                                 if (record.get() == 1) & (len(measur
    es) > 1):
66.                                         error.config(text='MENSAJE:
    Grabando datos en archivo...', foreground="blue")
67.                                         writeFile(measures)
68.                         elif( timeout > 600):
69.                                 errconnection = True
70.                                 break
71.         else:
72.                 errconnectport = True
73.         if (errclose == True)|(errconnection == True):
74.                 s.close()
75.         CheckError()
76.
77.
78. def writeGUI(data):
79.                 if len(data) > 1:
80.                         if (data[2]=='0') & (len(data)==8) :
81.                                 gps_hvar.set(data[0])
82.                                 gps_datvar.set(data[1])
83.                                 gps_fixvar.set(data[2])
84.                                 gps_fixqvar.set(data[3])
85.                                 gps_satvar.set(data[4])
86.                                 gps_latvar.set('#')
87.                                 gps_lonvar.set('#')
88.                                 gps_altvar.set('#')
89.                                 gps_speedvar.set('#')
90.                                 gps_anglevar.set('#')
91.                                 if data[5] == '-127.00':
92.                                         ds_cvar.set('#')
93.                                         ds_fvar.set('#')
94.                                 else:
95.                                         ds_cvar.set(data[5])
96.                                         ds_fvar.set(data[6])
97.                                 if data[7] == '88.00\r\n':
98.                                         phvar.set('#')
99.                                 else:
100.                                          phvar.set(data[7])
101.                          elif (data[0] == '#') & (len(data) == 4):
102.                                  gps_hvar.set('#')
103.                                  gps_datvar.set('#')
104.                                  gps_fixvar.set('#')
105.                                  gps_fixqvar.set('#')
```

```
106.                                    gps_satvar.set('#')
107.                                    gps_latvar.set('#')
108.                                    gps_lonvar.set('#')
109.                                    gps_altvar.set('#')
110.                                    gps_speedvar.set('#')
111.                                    gps_anglevar.set('#')
112.                                    if data[1] == '-127.00':
113.                                            ds_cvar.set('#')
114.                                            ds_fvar.set('#')
115.                                    else:
116.                                            phvar.set(data[1])
117.                                            ds_fvar.set(data[2])
118.                                    if data[3] == '88.00\r\n':
119.                                            phvar.set('#')
120.                                    else:
121.                                            phvar.set(data[3])
122.                            elif (len(data) == 13):
123.                                    gps_hvar.set(data[0])
124.                                    gps_datvar.set(data[1])
125.                                    gps_fixvar.set(data[2])
126.                                    gps_fixqvar.set(data[3])
127.                                    gps_satvar.set(data[4])
128.                                    gps_latvar.set(data[5])
129.                                    gps_lonvar.set(data[6])
130.                                    gps_altvar.set(data[7])
131.                                    gps_speedvar.set(data[8])
132.                                    gps_anglevar.set(data[9])
133.                                    if data[10] == '-127.00':
134.                                            ds_cvar.set('#')
135.                                            ds_fvar.set('#')
136.                                    else:
137.                                            ds_cvar.set(data[10])
138.                                            ds_fvar.set(data[11])
139.                                    if data[12] == '88.00\r\n':
140.                                            phvar.set('#')
141.                                    else:
142.                                            phvar.set(data[12])
143.
144.
145.    def writeFile(data):
146.            file_path = os.path.join(foldvar.get(),data[1]+'.txt')
147.            print(file_path)
148.            if os.path.exists(file_path):
149.                    with open(file_path, 'a') as myFile:
150.                            myFile.write(gps_hvar.get()+',
    '+gps_datvar.get()+', '+gps_fixvar.get()+', '+gps_fixqvar.get()+',
    '+gps_satvar.get()+', '+gps_latvar.get()
151.                            +', '+gps_lonvar.get()+',
    '+gps_altvar.get()+', '+gps_speedvar.get()+',
    '+gps_anglevar.get()+', '+ds_cvar.get()+', '+ds_fvar.get()+',
    '+phvar.get()+'\n')
152.            else:
153.                    with open(file_path, 'w+') as myFile:
154.                            myFile.write('Hora, Fecha, Enlace, Calidad
    del enlace, Satelites, Latitud, Longitud, Altitud, Velocidad,
    Angulo, TempC, TempF, pH\n')
155.                            myFile.write(gps_hvar.get()+',
    '+gps_datvar.get()+', '+gps_fixvar.get()+', '+gps_fixqvar.get()+',
    '+gps_satvar.get()+', '+gps_latvar.get()
156.                            +', '+gps_lonvar.get()+',
    '+gps_altvar.get()+', '+gps_speedvar.get()+',
```

```
    '+gps_anglevar.get()+', '+ds_cvar.get()+', '+ds_fvar.get()+',
    '+phvar.get()+'\n')
157.
158.
159.   def save():
160.           if (record.get() == 1) & (stoprecord.get() == 0):
161.                   stop.config(state='normal')
162.                   folder_path = tkFileDialog.askdirectory()
163.                   foldvar.set(folder_path)
164.           elif (stoprecord.get() == 2) & (record.get() == 1):
165.                   record.set(0)
166.                   stoprecord.set(0)
167.                   stop.config(state='disabled')
168.
169.
170.   def closeGUI():
171.           global errclose
172.           errclose = True
173.
174.
175.   def CheckError():
176.           global errconnection
177.           global errclose
178.           global errconnectport
179.           if (errconnection == True):
180.                   error.config(text='MENSAJE: Error de conexion.
    Revise la boya o el puerto.', foreground="red")
181.                   errconnection = False
182.           elif (errclose == True):
183.                   error.config(text='MENSAJE: Conexion cerrada con
    exito!', foreground="green")
184.                   errclose = False
185.           elif (errconnectport == True):
186.                   error.config(text='MENSAJE: Error de conexion.
    Seleccione puerto.', foreground="red")
187.                   errconnectport = False
188.           comm.set(0)
189.           nocomm.set(0)
190.           record.set(0)
191.           stoprecord.set(0)
192.           stop.config(state = 'disabled')
193.           off.config(state = 'disabled')
194.
195.
196.   #Titulos
197.   title = Label(master, text="Universidad Complutense de Madrid y
    Universidad de Almeria", font=("Helvetica", 12, "bold"))
198.   title.grid(row=0, column=0,columnspan=10, pady=20, sticky=W+E)
199.   subtitle = Label(master, text="ENTORNO DE MEDICION - SENSOR
    FLOTANTE", font=("Helvetica", 16, "bold"))
200.   subtitle.grid(row=1, column=0, columnspan=10, pady=10, sticky=W+E)
201.   subtitle1 = Label(master, text="PUERTO", font=("Helvetica", 12, "b
    old"))
202.   subtitle1.grid(row=2, column=0, columnspan=2, sticky= W+E)
203.   subtitle1 = Label(master, text="GUARDAR
    DATOS", font=("Helvetica", 12, "bold"))
204.   subtitle1.grid(row=9, column=0, columnspan=2, pady=10, sticky= W+E
    )
205.   subtitle2 = Label(master, text="GPS", font=("Helvetica", 12, "bold
    "))
206.   subtitle2.grid(row=2, column=2, sticky=W)
```

```
207.  subtitle3 = Label(master, text="TEMPERATURA", font=("Helvetica", 1
    2, "bold"))
208.  subtitle3.grid(row=6, column=2, sticky=W)
209.  subtitle4 = Label(master, text="SENSOR
    PH", font=("Helvetica", 12, "bold"))
210.  subtitle4.grid(row=6, column=6, sticky=W)
211.  subtitle5 = Label(text="DIRECTORIO: ", font=("Helvetica", 10))
212.  subtitle5.grid(row=10, column=2, sticky=W)
213.
214.  #Mensaje de error
215.  error=Label(master,  text="MENSAJE:", font=("Helvetica", 12, "bold
    "))
216.  error.grid(row=9, column=2, columnspan=4, sticky=W)
217.
218.
219.  #Lista seleccion de puerto
220.  listbox = Listbox(master,width=32)
221.  listbox.grid(row=3, column=0, columnspan=2, rowspan=4, padx=20)
222.  ports = list(serial.tools.list_ports.comports())
223.  for item in ports:
224.      listbox.insert(END, item)
225.
226.  #Botones
227.  play = Radiobutton(master, text='PLAY', variable=record, value=1,
    indicatoron=0, command=save, state='normal')
228.  play.grid(row=10, column=0, sticky=E)
229.  stop = Radiobutton(master, text='STOP', variable=stoprecord, value
    =2, indicatoron=0, command=save, state='disabled')
230.  stop.grid(row=10, column=1, sticky=W)
231.  on=Radiobutton(master, text='ABRIR
    PUERTO', variable=comm, value= 3, indicatoron=0, command=callback, s
    tate='normal' )
232.  on.grid(row=7, column=0, sticky=E)
233.  off=Radiobutton(master, text='CERRAR
    PUERTO', variable=nocomm, value= 4, indicatoron=0, command=closeGUI,
     state = 'disabled')
234.  off.grid(row=7, column=1, sticky=W)
235.
236.  #Folder
237.  fold=Entry(master, width=80, textvariable=foldvar)
238.  fold.grid(row=10, column=3, columnspan=8, pady= 10, sticky=W)
239.
240.  #Todos los datos de los sensores
241.  gps_h_text=Label(master,     text="HORA:",font=("Helvetica", 10))
242.  gps_dat_text=Label(master,   text="FECHA:", font=("Helvetica", 10)
    )
243.  gps_sat_text=Label(master,   text="SATELITES:", font=("Helvetica",
     10))
244.  gps_fix_text=Label(master,   text="ENLACE:", font=("Helvetica", 10
    ))
245.  gps_fixq_text=Label(master,  text="CALIDAD DEL
    ENLACE:", font=("Helvetica", 10))
246.  gps_lat_text=Label(master,   text="LATITUD:", font=("Helvetica", 1
    0))
247.  gps_lon_text=Label(master,   text="LONGITUD:", font=("Helvetica",
    10))
248.  gps_alt_text=Label(master,   text="ALTITUD:", font=("Helvetica", 1
    0))
249.  gps_speed_text=Label(master, text="VELOCIDAD
    :", font=("Helvetica", 10))
```

```
250.  gps_angle_text=Label(master, text="ANGULO:", font=("Helvetica", 10
   ))
251.  ds_c_text = Label(master,    text="TEMPERATURA
   (C):", font=("Helvetica", 10))
252.  ds_f_text = Label(master,    text="TEMPERATURA
   (F):", font=("Helvetica", 10))
253.  ph_text = Label(master,      text="pH:", font=("Helvetica", 10))
254.
255.  gps_h=Entry(master, width=10, textvariable=gps_hvar)
256.  gps_dat=Entry(master, width=10, textvariable=gps_datvar)
257.  gps_sat=Entry(master, width=10, textvariable=gps_satvar)
258.  gps_fix=Entry(master, width=10, textvariable=gps_fixvar)
259.  gps_fixq=Entry(master, width=10, textvariable=gps_fixqvar)
260.  gps_lat=Entry(master, width=10, textvariable=gps_latvar)
261.  gps_lon=Entry(master, width=10, textvariable=gps_lonvar)
262.  gps_alt=Entry(master, width=10, textvariable=gps_altvar)
263.  gps_speed=Entry(master, width=10, textvariable=gps_speedvar)
264.  gps_angle=Entry(master, width=10, textvariable=gps_anglevar)
265.  ds_c = Entry(master, width=10, textvariable=ds_cvar)
266.  ds_f = Entry(master, width=10, textvariable=ds_fvar)
267.  ph = Entry(master, width=10, textvariable=phvar)
268.
269.  gps_h_text.grid(row=3, column=2, sticky=W)
270.  gps_h.grid(row=3, column=3, sticky=W)
271.  gps_dat_text.grid(row=3, column=4, sticky=W)
272.  gps_dat.grid(row=3,column=5, sticky=W)
273.  gps_fix_text.grid(row=4, column=2, sticky=W)
274.  gps_fix.grid(row=4,column=3, sticky=W)
275.  gps_sat_text.grid(row=4, column=4, sticky=W)
276.  gps_sat.grid(row=4, column=5, sticky=W)
277.  gps_fixq_text.grid(row=4, column=6, sticky=W)
278.  gps_fixq.grid(row=4, column=7, sticky=W)
279.  gps_lat_text.grid(row=4, column=8, sticky=W)
280.  gps_lat.grid(row=4, column=9, padx=20, sticky=W)
281.  gps_lon_text.grid(row=5, column=2, sticky=W)
282.  gps_lon.grid(row=5, column=3, sticky=W)
283.  gps_alt_text.grid(row=5, column=4, sticky=W)
284.  gps_alt.grid(row=5, column=5, sticky=W)
285.  gps_speed_text.grid(row=5, column=6, sticky=W)
286.  gps_speed.grid(row=5, column=7, sticky=W)
287.  gps_angle_text.grid(row=5, column=8, sticky=W)
288.  gps_angle.grid(row=5, column=9, padx=20, sticky=W)
289.  ds_c_text.grid(row=7, column=2, sticky=W)
290.  ds_c.grid(row=7, column=3, sticky=W)
291.  ds_f_text.grid(row=7, column=4, sticky=W)
292.  ds_f.grid(row=7, column=5, sticky=W)
293.  ph_text.grid(row=7, column=6, sticky=W)
294.  ph.grid(row=7, column=7, sticky=W)
295.
296.
297.  mainloop()
```

Anexo IV: listado de componentes

TABLA ANEXO-1. LISTA CON LOS COMPONENTES SELECCIONADOS PARA LA BOYA Y EL RECEPTOR.

Ubicación	***Fabricante***	***Modelo***	
Sensor Flotante	*Adafruit*	DS18B20	
	Atlas Scientific	EZO™ pH Circuit	
	Atlas Scientific	pH Industrial Probe	
	Adafruit	Ultimate Breakout GPS	
	Arduino	Nano	
	Gravi Tech	Arduino Nano – Xbee shield	
	Desconocido	Batería Polímero Litio 3,7V / 2000MA	
	Adafruit	USB / DC / Solar Lithium Ion/Polymer charger - v2	
	Sundance Solar	Solar Panel 6v 100mA	

Receptor y sensor flotante	*Xbee*	*2.4 GHz XBee-PRO*	
Receptor	*Libelium*	Arduino UNO – Xbee shield	
	Arduino	UNO	