

# **GUÍA PARA LA CONSTRUCCIÓN DE UN MONITOREO ACUÍCOLA INTELIGENTE.**

**Autor: Olvin Joel Barrera Mendez.**

## **Funciones del sistema.**

Este dispositivo tiene diversas funciones, tales como la medición y envío de datos de oxígeno, pH, temperatura y ORP. Es capaz de controlar sus propios intervalos de medición, así como controlar esos intervalos de medición, así como el control de la ubicación de su sensor de oxígeno en función de un sensor de nivel de humedad, esto con el fin de proteger la membrana del sensor de oxígeno y extender su tiempo de vida. Este sistema tiene una App interfaz para los datos enviados por internet, así como el control de un Pin Output digital, dicha función pretende aplicarse en un futuro al control de, por ejemplo, un motor AC a través de un relevador, el control es realizado desde la App.

Todo el sistema se divide en dos secciones, la primera es la del monitoreo de pH, Temperatura y ORP, contenida en un Kit de Atlas Scientific, misma sección que llamaremos SECCIÓN-A, programada en un microcontrolador ESP8266 de Adafruit, al que llamaremos ESP32c.

La segunda sección SECCIÓN-B se subdivide en dos subsistemas, el primer subsistema al que llamaremos Control de censado, programado en la ESP32b, dirigido al control del monitoreo de oxígeno, es decir, el control del motor mediante PWM y un puente H, mismo motor que soporta al sensor de oxígeno, todo esto en función del sensor de nivel de humedad que envía la señal al ESP32a para que este decida si realizar o no el envío de datos, y que mantiene comunicación con el segundo subsistema.

El segundo subsistema ESP32a se encarga de la rutina para las mediciones y el envío de los datos obtenidos por el sensor de oxígeno, esto a través del sistema embebido de Atlas Scientific EZO™ Dissolved Oxygen Circuit (se explicará más a detalle adelante), así como también tiene la función de recibir la señal del sensor de humedad para saber si enviar o no los datos medidos.

Todas las variables se envían a la plataforma ThingSpeak, misma que comparte su API (Interfaz de programación de aplicaciones), usada en una App creada a partir de Appinventor, dicho procedimiento también se abordará.

Explicado todo lo anterior, la guía abordará las secciones, su programación y su diagrama de conexión, así como su funcionamiento a través del código, dicho código aún presenta retos y errores a resolver.

## SECCIÓN-A

La sección A se compone por el Wifi-Pool Kit, de Atlas Scientific, misma se compone de 3 sensores, uno de temperatura, uno de pH y uno de ORP, todos estos sensores son enviados a la plataforma ThingSpeak vía HTTP, por lo que se tiene procura una diferencia entre intervalos de al menos 17 segundos.

Cabe aclarar que en esta sección no hace falta calibrar los sensores, ya vienen pre-calibrados antes del envío, y esto tiene una duración de un año, para mayor información acerca de la calibración de cada sensor se debe consultar en la guía oficial de Atlas Scientific de cada sensor <https://files.atlas-scientific.com/Wi-Fi-Pool-kit-setup-guide.pdf>

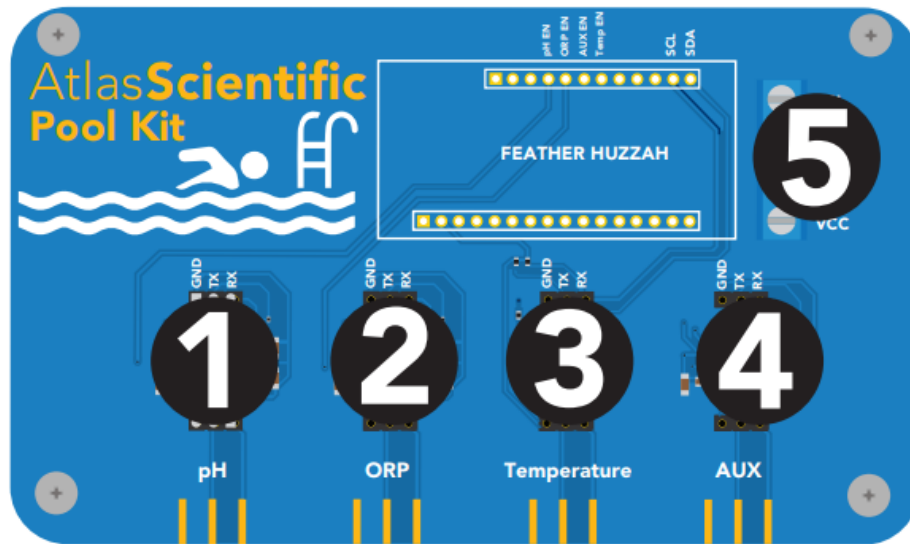
El Wi-Fi pool Kit fue diseñado para proveer una manera simple para el monitoreo remoto.

Para poder lograr esto, el Kit es controlado usando una ESP8266 Adafruit Feather HUZZAH como su CPU. El microcontrolador HUZZAH es programado usando el IDE de Arduino, si gusta leer el manual de este microcontrolador puede encontrarlo aquí <https://learn.adafruit.com/adafruit-feather-huzzah-esp8266>

### **Puertos de los sensores.**

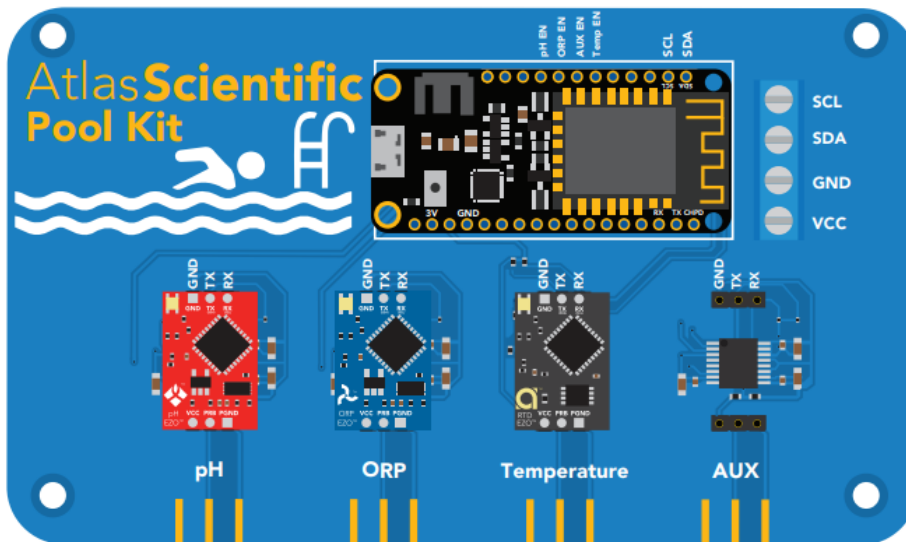
El PCB del Wi-Fi Pool Kit tiene 5 puertos. Tres de esos puertos están eléctricamente aislados. Los puertos aislados están marcados como pH, ORP y AUX. Los puertos aislados son necesitados para obtener lecturas electroquímicas libres de ruido. Debido a que el sensor de temperatura nunca está en directo contacto con el agua, el aislamiento eléctrico no es necesitado para su puerto asignado.

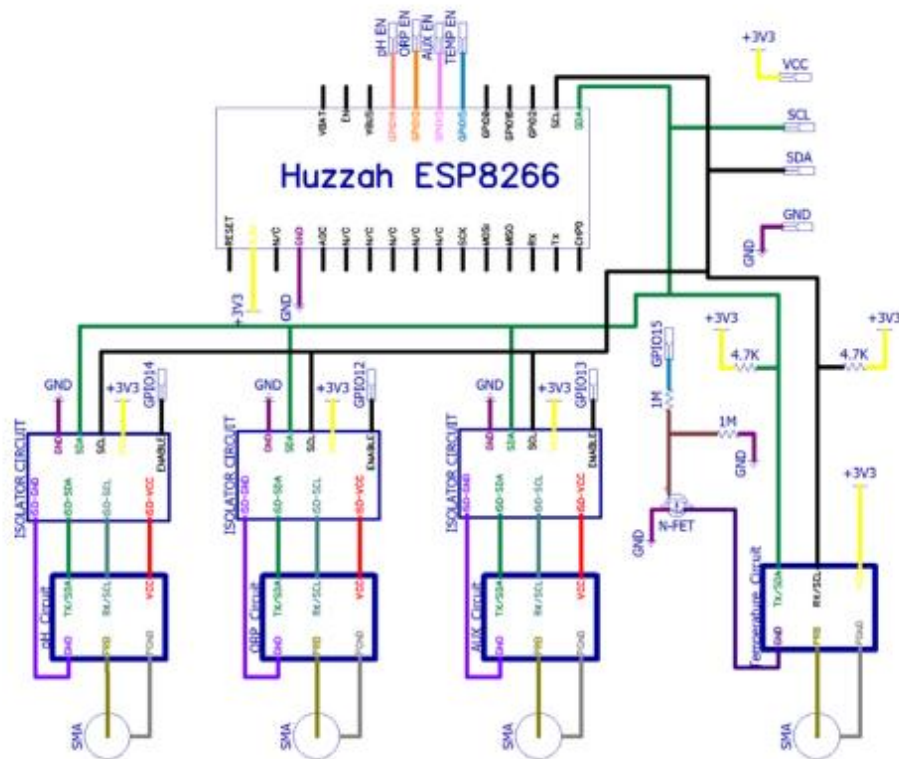
El puerto AUX puede ser usado para añadir un sensor adicional al gusto de cada diseño. La terminal marcada como puerto 5 fue diseñada para conectar una bomba para el dispositivo. Sin embargo, ese mismo puerto puede ser utilizado como un sensor de gas.



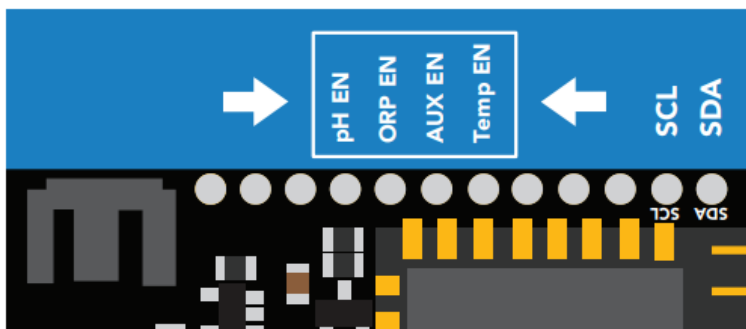
## PCB

El diseño general para el PCB es bastante simple. El CPU es alimentado y programado a través del conector USB. El regulador de voltaje del microcontrolador provee a la tarjeta un voltaje de 3.3V y un pico de corriente de 500mA. Todos los sensores conectados funcionan a 3.3V.





Cada uno de los cuatro puertos de los sensores tienen un Pin EN, el cual debe ser conectado correctamente para alimentar al sensor. Los pins EN se encuentran así:



Los primeros tres pines (pH, ORP y AUX) deben estar en un estado LOW para alimentar al sensor. El último pin (Temp) debe estar en un estado HIGH para alimentar al sensor.

Aquí está la tabla de la verdad:

Pin	State	Sensor Power
pH EN	LOW	ON
ORP EN	LOW	ON
Aux EN	LOW	ON
Temp EN	HIGH	ON

### Protocolo de datos.

El CPU se comunica con todos los sensores periféricos utilizando el protocolo I2C. Todos los datos están directamente conectados a los puertos I2C. Usando un protocolo distinto no es posible en este circuito.

Es importante mantener en mente que todos los componentes de Atlas Scientific están por defecto en UART mode. Cuando se añaden nuevos componentes de Atlas Scientific al kit, debe ser primero programado en I2C mode. Para poder realizar esto es necesario leer la datasheet de cada componente para proceder con sus respectivas instrucciones, todo esto se puede buscar fácilmente en el sitio oficial de Atlas Scientific <https://atlas-scientific.com/>

### Programación y configuración de ThingSpeak.

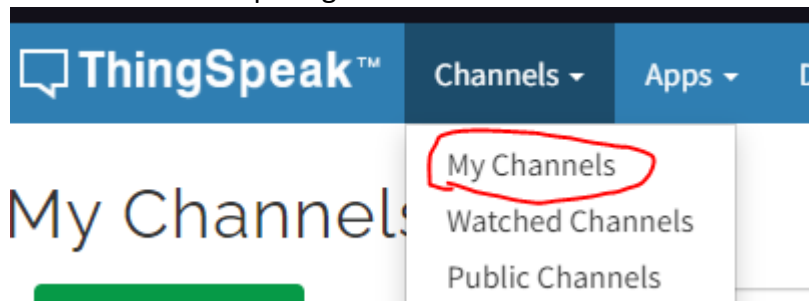
Para la programación es necesario entrar al link del código [https://github.com/Atlas-Scientific/Ezo\\_I2c\\_lib/blob/master/Examples/IOT\\_kits/pool\\_kit/pool\\_kit.ino](https://github.com/Atlas-Scientific/Ezo_I2c_lib/blob/master/Examples/IOT_kits/pool_kit/pool_kit.ino) y simplemente copiarlo, sin embargo, si se tienen más dudas se puede acceder a la carpeta raíz y leer el archivo README.md [https://github.com/Atlas-Scientific/Ezo\\_I2c\\_lib](https://github.com/Atlas-Scientific/Ezo_I2c_lib)

Una vez que se tiene el código listo para cargar en la ESP8266, simplemente se tienen que modificar algunos datos del código.

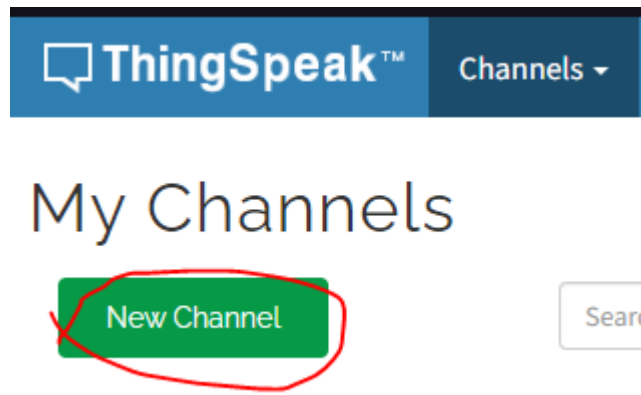
```
//-----Fill in your Wi-Fi / ThingSpeak Credentials---  
const String ssid = "Wifi Name";  
const String pass = "Wifi Password";  
const long myChannelNumber = 1234566;  
const char * myWriteAPIKey = "XXXXXXXXXXXXXXXXXX";  
//-----
```

Lo primero es cambiar el SSID que tiene por defecto como "Wifi Name" al SSID del modem que se desea conectar, así mismo con la "Wifi Password".

Sin embargo, para los dos siguientes pasos se tiene que estar familiarizado con la plataforma ThingSpeak, lo primero es crear un canal en ThingSpeak y antes de obtener esos datos se procederá a configurarlo, lo primero es acceder a <https://thingspeak.com>, una vez dentro lo que sigue es crear un canal



Una vez dentro se da click en New Channel



El formulario necesario para configurar la plataforma de ThingSpeak para este sistema debe ser llenado de la siguiente manera y se procede a guardar

Name	<input type="text" value="Monitoreo Atlas"/>	
Description	<input type="text" value="Sensado de oxígeno disuelto, temperatura, pH y ORP"/>	
Field 1	<input type="text" value="pH"/>	<input checked="" type="checkbox"/>
Field 2	<input type="text" value="ORP (mV)"/>	<input checked="" type="checkbox"/>
Field 3	<input type="text" value="Temp (°C)"/>	<input checked="" type="checkbox"/>
Field 4	<input type="text" value="Oxígeno Disuelto (mg/"/>	<input checked="" type="checkbox"/>
Field 5	<input type="text" value="Sensor_State"/>	<input checked="" type="checkbox"/>
Field 6	<input type="text" value="motor state"/>	<input checked="" type="checkbox"/>
<div>Save Channel</div>		

Lo siguiente es ir al apartado Public View

Private View	Public View	Chan
--------------	-------------	------

Una vez ahí, las gráficas se han configurado de la siguiente manera para que se presenten las unidades en el eje Y y el tiempo en el eje X, esto a través de la opción Add Visualizations

+ Add Visualizations

Field 1 Chart



Sensor de pH

pH

Date

ThingSpeak.com

Field 2 Chart



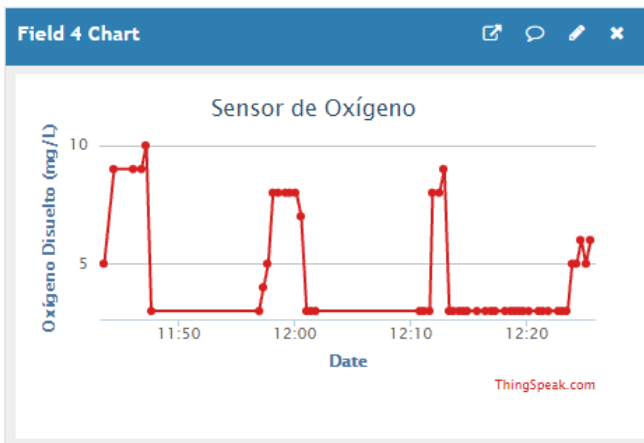
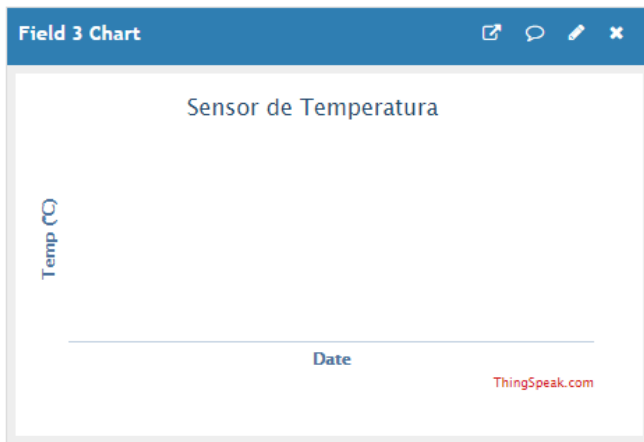
Sensor ORP

ORP (mV)

Date

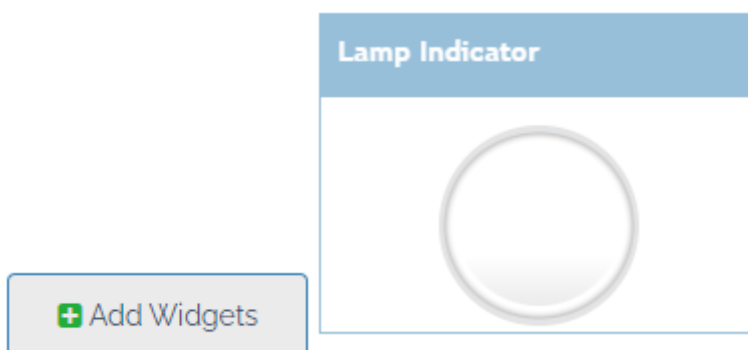
ThingSpeak.com





Sin embargo, para configurar los estados de sensor, es decir si el sensor está dentro o fuera del agua, así como un led de control que servirá para encender o apagar un motor (un aerador en este caso ya que fue dedicado a un proyecto acuícola), requiere un paso más.

Lo primero es que se debe seleccionar la opción Add Widgets y después el lamp indicator



Una vez hecho esto, en el caso del led del estado del sensor se deberá llenar el formulario de la siguiente manera.

Sensor\_state Options

?

x

Name

Sensor\_state

Condition

If

Field 5

is equal to

0

turn Lamp ON

Update Interval

15

second(s)

Color

Save

Cancel

De la misma manera, el campo número 6 debe ser llenado de la siguiente manera.

aereador state Options

?

x

Name

aereador state

Condition

If

Field 6

is greater than

0

turn Lamp ON

Update Interval

15

second(s)

Color

Save

Cancel

Una vez hecha la configuración total de ThingSpeak para este sistema, se procede entonces a obtener los datos que se editarán en los siguientes campos.

```
//-----Fill in your Wi-Fi / ThingSpeak Credentials-  
const String ssid = "Wifi Name";  
const String pass = "Wifi Password";  
const long myChannelNumber = 1234566;  
const char * myWriteAPIKey = "XXXXXXXXXXXXXXXXXX";  
//-----
```

Para esto se procede dentro de ThingSpeak a seleccionar la pestaña de API keys

ing API Keys Dat

Una vez dentro, se puede observar en la parte superior de todas las pestañas el número de tu canal, en este caso cubierto por el color azul por tema de protección de nuestro canal. Este número es el que se sustituye en myChannelNumber = 1234566;

## Monitoreo Atlas

Channel ID: [redacted]  
Author: mwa0000020340673  
Access: Public

Sensado de oxigeno disuelto  
ORP

Private View Public View Channel Settings Sharing API Keys [

Por otro lado, myWriteAPIkey = "XXXXXXXXXXXXXXXXXX"; se sustituye con el siguiente código cubierto por seguridad.

### Write API Key

Key

[redacted]

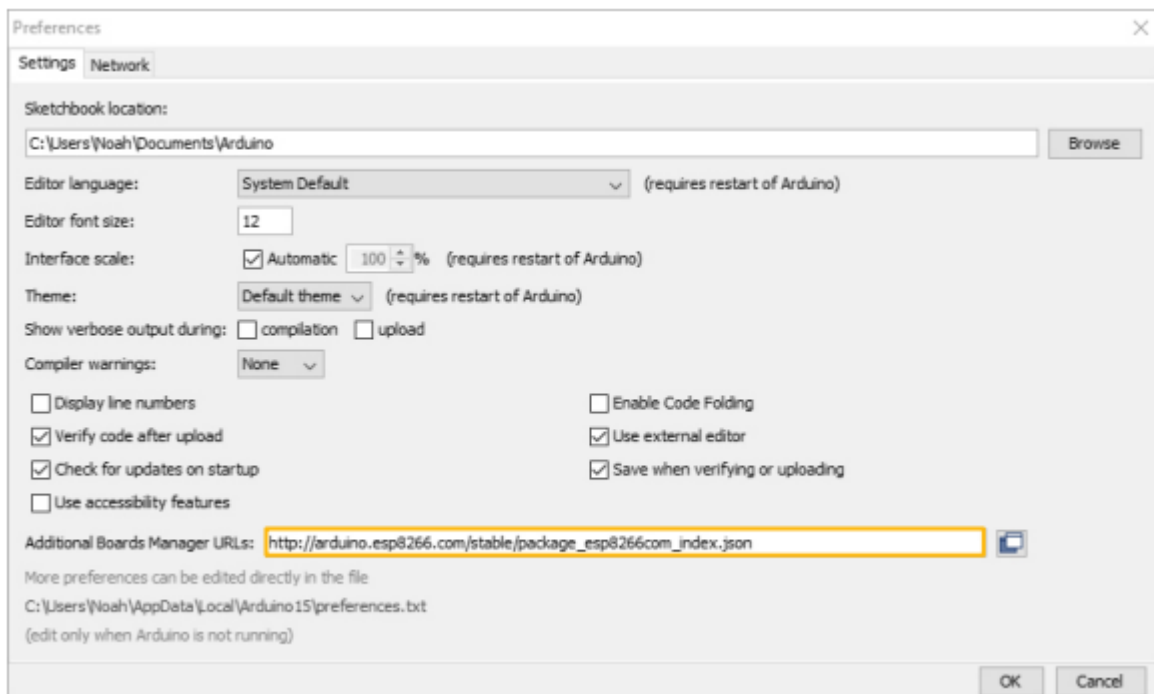
Generate New Write API Key

## Configuración del IDE.

El siguiente paso a realizar es configurar el IDE de Arduino correctamente para cargar el programa.

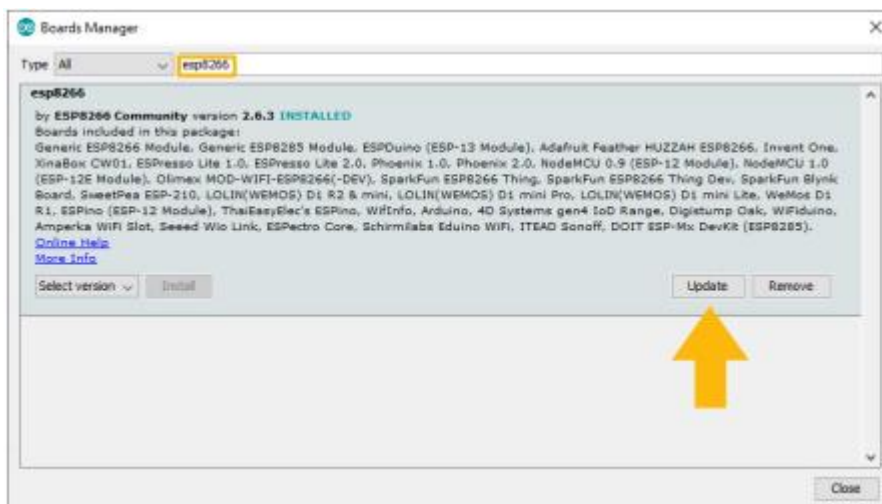
Dentro del IDE hay que acceder a FILE > PREFERENCES, posteriormente localizar la opción Additional Boards Manager URLs y pegar la URL

[http://arduino.esp8266.com/stable/package\\_esp8266com\\_index.json](http://arduino.esp8266.com/stable/package_esp8266com_index.json) y presionar OK.



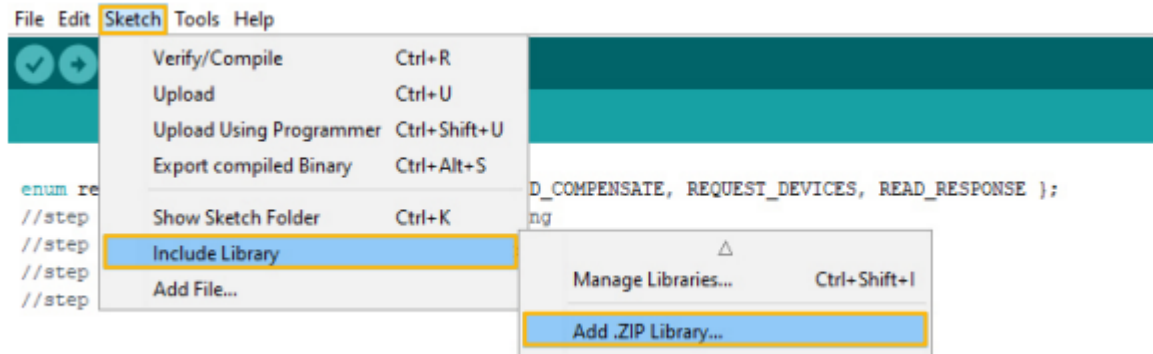
## Configuración de la ESP8266.

En el IDE ve a o Tools > Board > Boards Manager, en la barra de búsqueda escribe ESP8266 e instala la versión más actual.



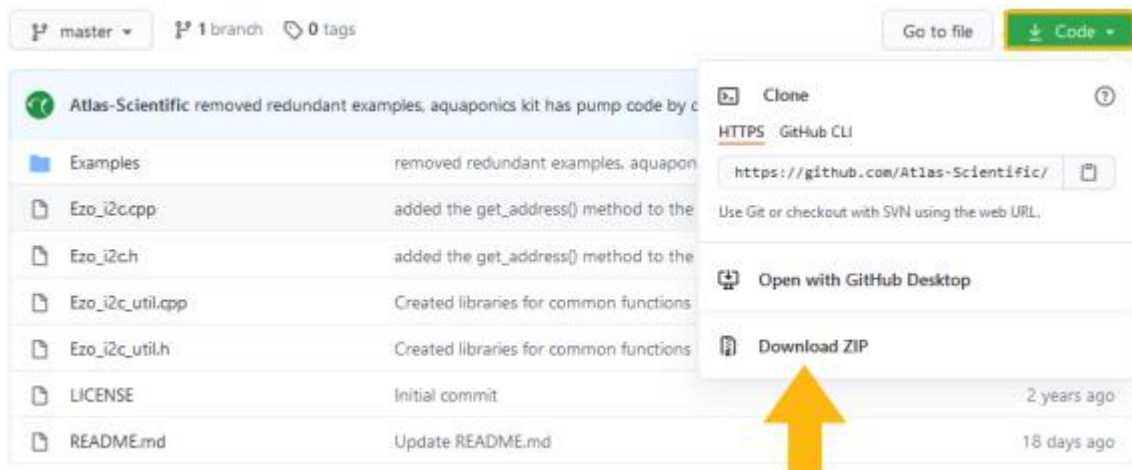
Descarga la librería de ThingSpeak para Arduino

<https://www.arduino-libraries.info/libraries/thing-speak>, recuerda no descomprimirlo, importa el archivo .ZIP desde el Arduino IDE. Para importarlo, solamente ve a o Sketch > Include Library > Add .ZIP Library



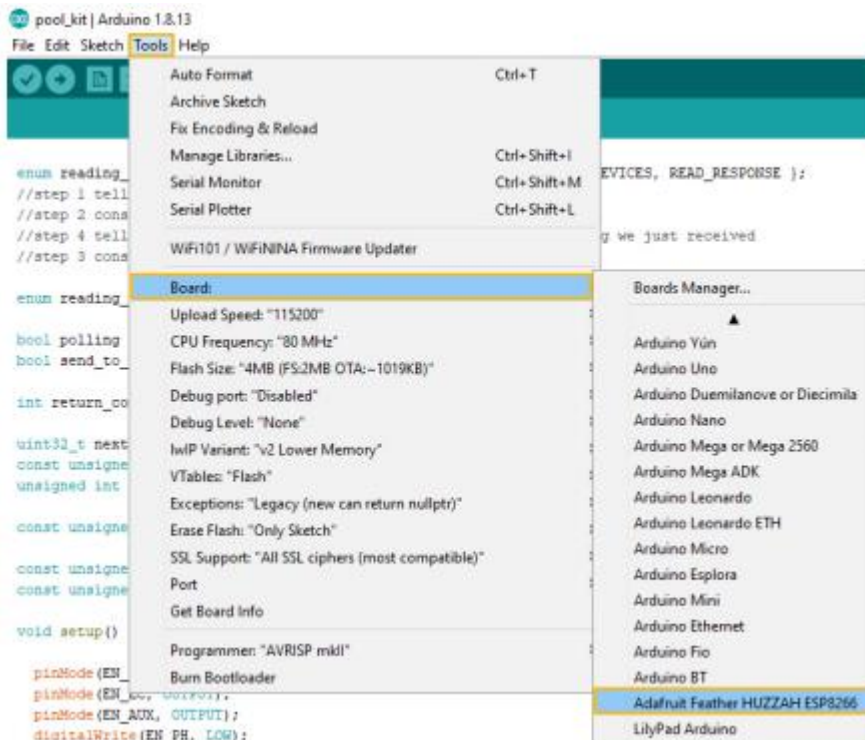
Del mismo modo, es necesario descargar la librería para EZO I2C.

Descarga la librería desde [https://github.com/Atlas-Scientific/Ezo\\_I2c\\_lib](https://github.com/Atlas-Scientific/Ezo_I2c_lib), y del mismo modo sin descomprimir importa el archivo desde el IDE con la ruta Sketch > Include Library > Add .ZIP Library

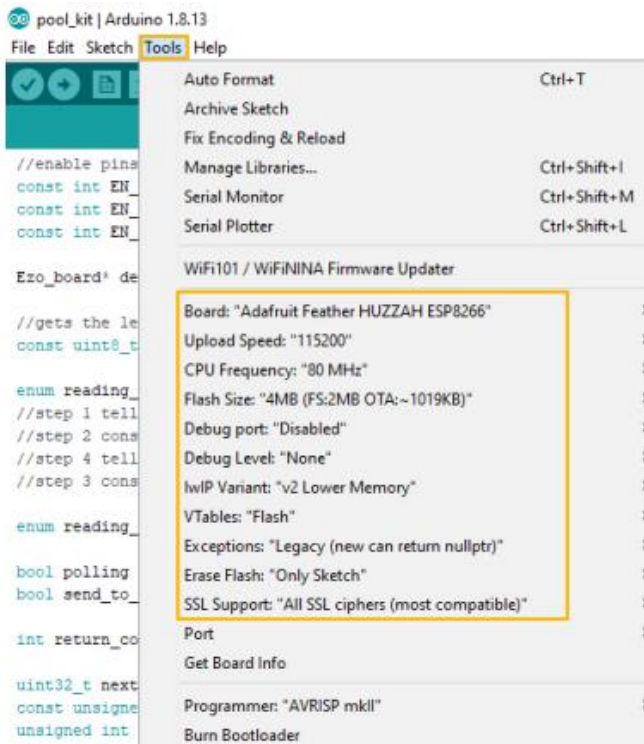


Por último, el código puede obtenerse como se mostró anteriormente, sin embargo, también se puede encontrar una vez descargadas las librerías en la ruta File > Examples > EZO\_I2C\_lib-master > Examples > IOT\_kits > pool\_kit. Recuerda editar las credenciales como se explicó con anterioridad.

Ahora bien, es necesario configurar la ESP8266, por lo que para flashear el código es necesario ir a la ruta Tools > Board > Adafruit Feather HUZZAH ESP8266. Tal como se muestra en la siguiente imagen.

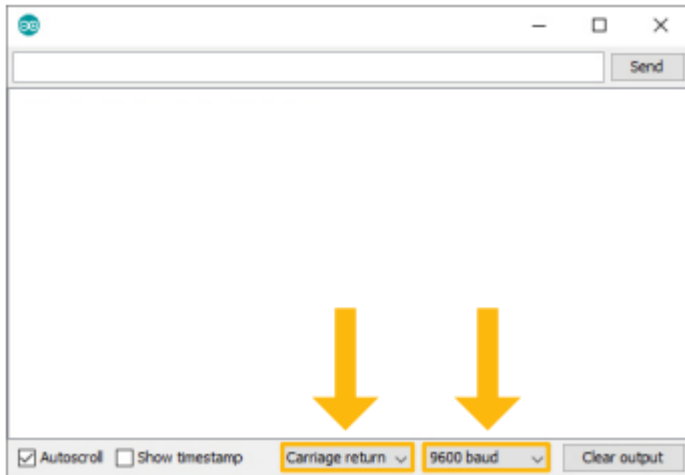


Una vez hecho esto, hay que asegurarse que la configuración del microcontrolador es la correcta.

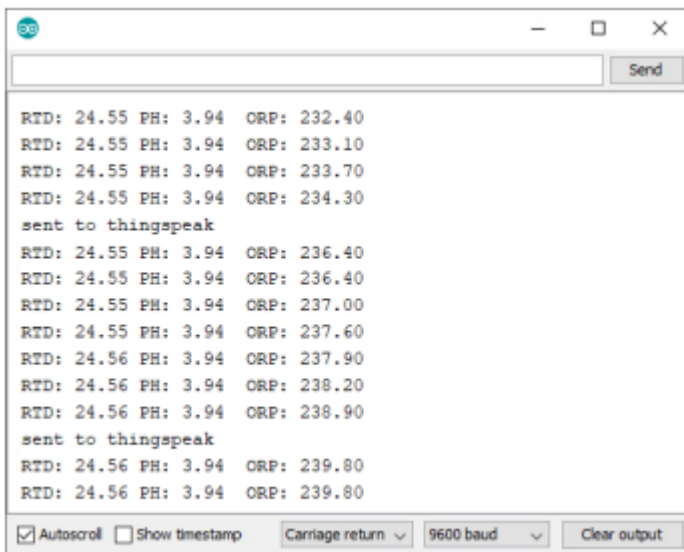


Y listo, una vez hecho eso se puede cargar el código y conectar cada sensor con su color correspondiente.

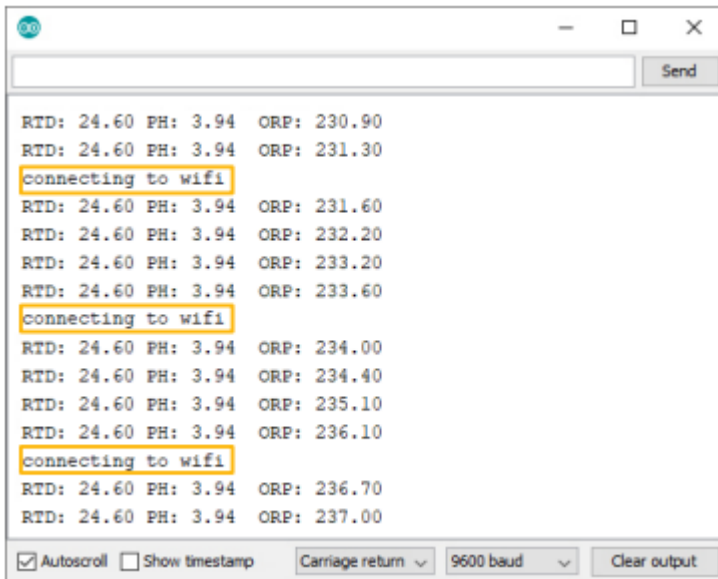
Para poder visualizar los datos desde el IDE de Arduino hay que entrar al monitor serial y configurarlo de la siguiente manera



Si el WiFi es el correcto, entonces aparecerá algo así



De lo contrario aparecerá una imagen así



A screenshot of a terminal window with a title bar containing a green icon, a minus button, a maximize button, and a close button. The window has a 'Send' button in the top right corner. The main area displays a series of sensor readings: 'RTD: 24.60 PH: 3.94 ORP: 230.90', 'RTD: 24.60 PH: 3.94 ORP: 231.30', 'connecting to wifi', 'RTD: 24.60 PH: 3.94 ORP: 231.60', 'RTD: 24.60 PH: 3.94 ORP: 232.20', 'RTD: 24.60 PH: 3.94 ORP: 233.20', 'RTD: 24.60 PH: 3.94 ORP: 233.60', 'connecting to wifi', 'RTD: 24.60 PH: 3.94 ORP: 234.00', 'RTD: 24.60 PH: 3.94 ORP: 234.40', 'RTD: 24.60 PH: 3.94 ORP: 235.10', 'RTD: 24.60 PH: 3.94 ORP: 236.10', 'connecting to wifi', 'RTD: 24.60 PH: 3.94 ORP: 236.70', and 'RTD: 24.60 PH: 3.94 ORP: 237.00'. The 'connecting to wifi' lines are highlighted with yellow boxes. At the bottom, there are controls: a checked 'Autoscroll' checkbox, an unchecked 'Show timestamp' checkbox, a 'Carriage return' dropdown menu, a '9600 baud' dropdown menu, and a 'Clear output' button.

```
RTD: 24.60 PH: 3.94 ORP: 230.90
RTD: 24.60 PH: 3.94 ORP: 231.30
connecting to wifi
RTD: 24.60 PH: 3.94 ORP: 231.60
RTD: 24.60 PH: 3.94 ORP: 232.20
RTD: 24.60 PH: 3.94 ORP: 233.20
RTD: 24.60 PH: 3.94 ORP: 233.60
connecting to wifi
RTD: 24.60 PH: 3.94 ORP: 234.00
RTD: 24.60 PH: 3.94 ORP: 234.40
RTD: 24.60 PH: 3.94 ORP: 235.10
RTD: 24.60 PH: 3.94 ORP: 236.10
connecting to wifi
RTD: 24.60 PH: 3.94 ORP: 236.70
RTD: 24.60 PH: 3.94 ORP: 237.00
```

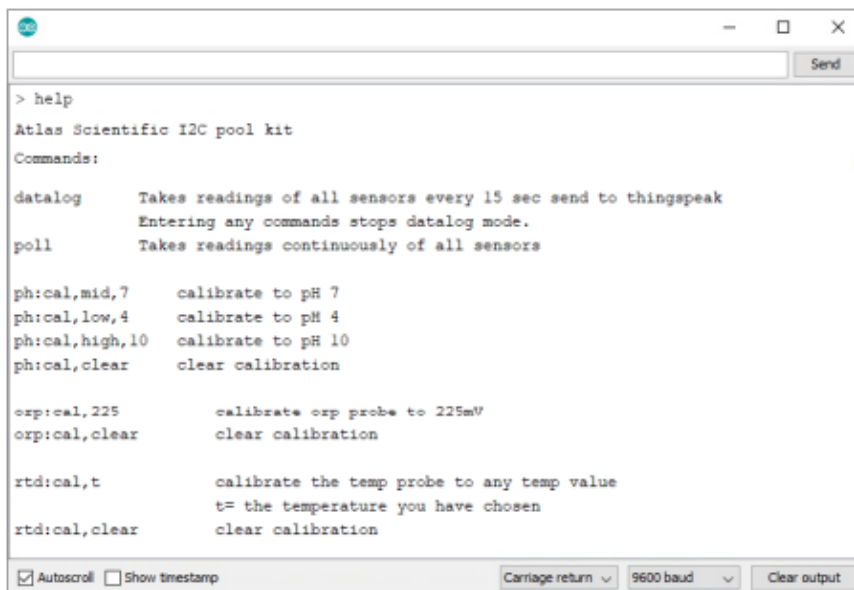
☒ Autoscroll ☐ Show timestamp Carriage return 9600 baud Clear output

Y listo, con esto finaliza la configuración de la SECCIÓN-A.

Cabe aclarar que cualquier duda, o interés a fondo de esta sección, se recomienda ir directamente a la guía de Atlas Scientific especificada para este Kit.

<https://files.atlas-scientific.com/Wi-Fi-Pool-kit-setup-guide.pdf>

Ahí encontrará los detalles para la calibración, por ejemplo.



A screenshot of a terminal window with a title bar containing a green icon, a minus button, a maximize button, and a close button. The window has a 'Send' button in the top right corner. The main area displays the help menu for the 'Atlas Scientific I2C pool kit'. It lists commands: 'datalog' (Takes readings of all sensors every 15 sec send to thingspeak), 'poll' (Takes readings continuously of all sensors), 'ph:cal,mid,7' (calibrate to pH 7), 'ph:cal,low,4' (calibrate to pH 4), 'ph:cal,high,10' (calibrate to pH 10), 'ph:cal,clear' (clear calibration), 'orp:cal,225' (calibrate orp probe to 225mV), 'orp:cal,clear' (clear calibration), 'rtd:cal,t' (calibrate the temp probe to any temp value), and 'rtd:cal,clear' (clear calibration). At the bottom, there are controls: a checked 'Autoscroll' checkbox, an unchecked 'Show timestamp' checkbox, a 'Carriage return' dropdown menu, a '9600 baud' dropdown menu, and a 'Clear output' button.

```
> help
Atlas Scientific I2C pool kit
Commands:

datalog    Takes readings of all sensors every 15 sec send to thingspeak
           Entering any commands stops datalog mode.
poll       Takes readings continuously of all sensors

ph:cal,mid,7    calibrate to pH 7
ph:cal,low,4    calibrate to pH 4
ph:cal,high,10  calibrate to pH 10
ph:cal,clear    clear calibration

orp:cal,225     calibrate orp probe to 225mV
orp:cal,clear   clear calibration

rtd:cal,t       calibrate the temp probe to any temp value
               t= the temperature you have chosen
rtd:cal,clear   clear calibration
```

☒ Autoscroll ☐ Show timestamp Carriage return 9600 baud Clear output



## SECCIÓN-B

Como se describió al principio, la SECCIÓN-B está dirigida al censado de oxígeno disuelto, sin embargo, debido a las características de la membrana de este sensor, fue necesario construir una rutina que sea capaz de mantener intervalos de medición pequeños, con intervalos de reposo, esto con el fin de alargar el mayor tiempo posible la vida útil de la membrana del sensor.

Es por eso que esta sección se divide en dos subsistemas, primero se explicará el subsistema ESP32b y después el subsistema ESP32a.

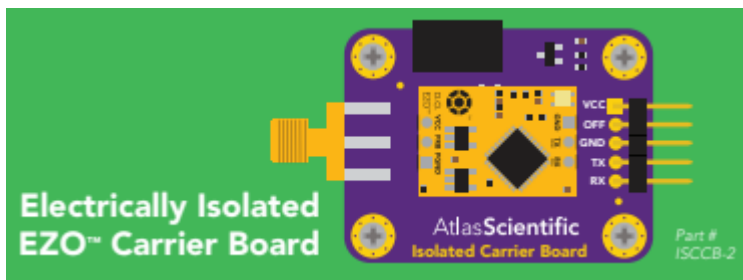
Además, en la SECCIÓN-A se explicó la configuración de la plataforma ThingSpeak para todo el sistema, por lo que se explicará solamente el diagrama electrónico, así como la programación de estos subsistemas.

Por supuesto, antes de hacer el diagrama de esta sección es muy necesario que se calibre el sensor de oxígeno, se hará un pequeño resumen al respecto, sin embargo, lo más recomendable es leer la guía directamente de Atlas Scientific <https://atlas-scientific.com/kits/dissolved-oxygen-kit/>

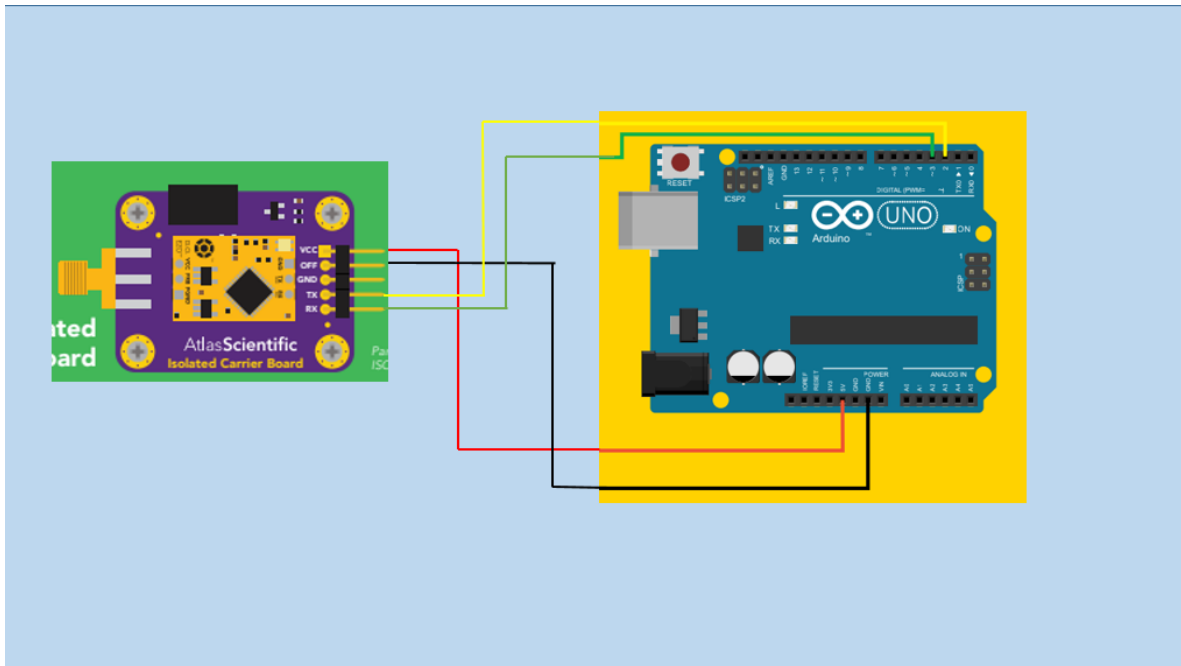
### Guía de calibración para el sensor de oxígeno disuelto.

Antes de cualquier calibración, lo primero es leer la guía para ensamblar correctamente el embebido del sensor de oxígeno disuelto, donde se recomienda utilizar el módulo de aislamiento eléctrico para evitar ruido y tener mediciones más precisas, así como extender la vida útil del embebido del sensor. [https://files.atlas-scientific.com/DO\\_EZO\\_Datasheet.pdf](https://files.atlas-scientific.com/DO_EZO_Datasheet.pdf)

Lo primero entonces es conectar todo correctamente:



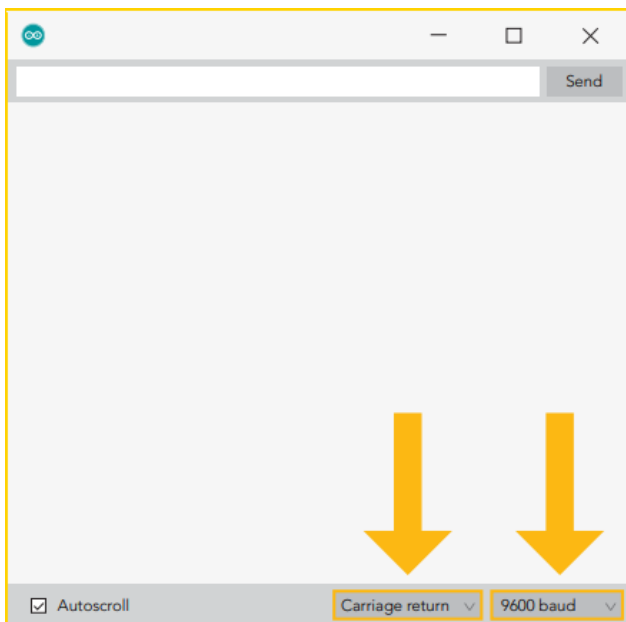
El dispositivo debe estar conectado de esta manera, una vez realizado esto, el siguiente paso es conectarlo a un Arduino UNO, en esta guía se calibró desde un Arduino UNO, ya que al intentar hacerlo con un Arduino Mega siguiendo la guía se obtuvieron errores y bugs, además de que no pudo permanecer configurado para cuando se conectó en la ESP32.



Con esta conexión realizada, entonces se procede a la programación, la cual es bastante simple, solamente es cuestión de cargar el código desde la IDE de Arduino, seleccionar la placa Arduino UNO y configurar en 9600 Baud.

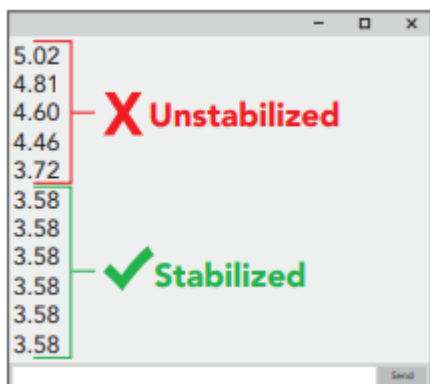
El código se puede descargar desde el siguiente link [https://www.atlas-scientific.com/files/code/ino\\_files/Arduino\\_UNO\\_DO\\_sample\\_code.zip](https://www.atlas-scientific.com/files/code/ino_files/Arduino_UNO_DO_sample_code.zip)

Una vez cargado el código a la placa de Arduino, lo siguiente es abrir el Monitor serial y asegurarse de que esté configurado de la siguiente manera



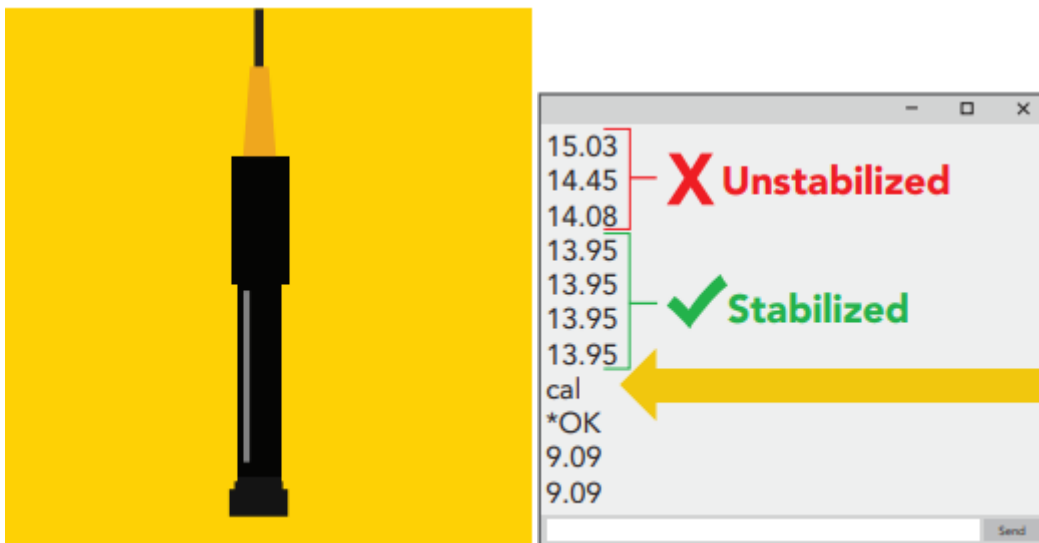
Con el circuito y el programa listo, el monitor serial debería mostrar entonces las medidas de oxígeno escribiendo el comando C en la barra de comandos del monitor serial, activando así las lecturas continuas de oxígeno, una por segundo por defecto, para mayor información se recomienda leer la guía [https://files.atlas-scientific.com/DO\\_EZO\\_Datasheet.pdf](https://files.atlas-scientific.com/DO_EZO_Datasheet.pdf).

Una vez que se estén obteniendo lecturas continuas, el monitor debería presentar una lectura como la siguiente ya que se haya estabilizado.



Para comenzar entonces con la calibración, se debe mantener al sensor entre 5 a 30 segundos en el aire hasta que las lecturas se estabilicen.

Una vez que las lecturas se estabilicen, es necesario escribir el comando “cal” en la barra de comandos del monitor serial, aparecerá entonces un mensaje como el que se muestra a continuación.



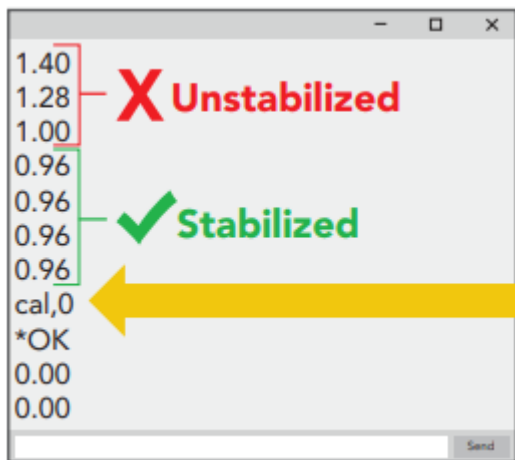
Una vez que la calibración está completa, las lecturas deberían estar estables entre 9.09 – 9.1X mg/L. Solamente si la compensación de temperatura, salinidad y presión están en sus valores de defecto.

Ahora bien, si se desean lecturas con una precisión por debajo de 1.0 mg/L entonces es necesario realizar un paso más.

Después de haber calibrado el sensor con el comando “Cal”, se debe remover la tapa de la Zero Dissolved Oxygen calibration solution, e insertar la probeta del sensor dentro, para remover el aire atrapado hay que batir la solución en pequeños círculos (esto dará lecturas altas e inestables). Lo siguiente es dejar en reposo el sensor dentro de la solución hasta que las lecturas se estabilicen.



Una vez que las lecturas se hayan estabilizado por debajo de 1.0 mg/L, se procede a escribir el comando “Cal,0”



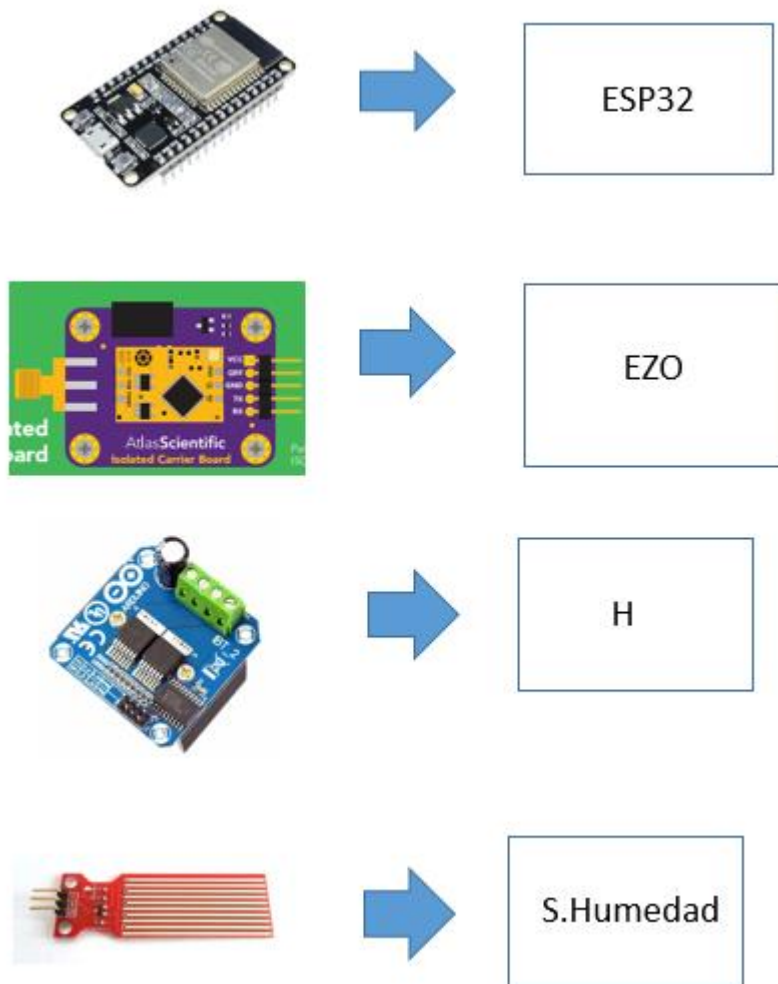
Una vez hecho esto, entonces el embebido de el sensor de oxígeno está listo y calibrado para poder conectarse a la ESP32, sin embargo, para preservación de esta calibración así como de la solución de calibración, en esta guía se insiste leer la guía oficial de atlas scientific [https://files.atlas-scientific.com/DO\\_EZO\\_Datasheet.pdf](https://files.atlas-scientific.com/DO_EZO_Datasheet.pdf).

El siguiente paso en esta guía entonces es el diagrama electrónico de la SECCIÓN-B del sistema.

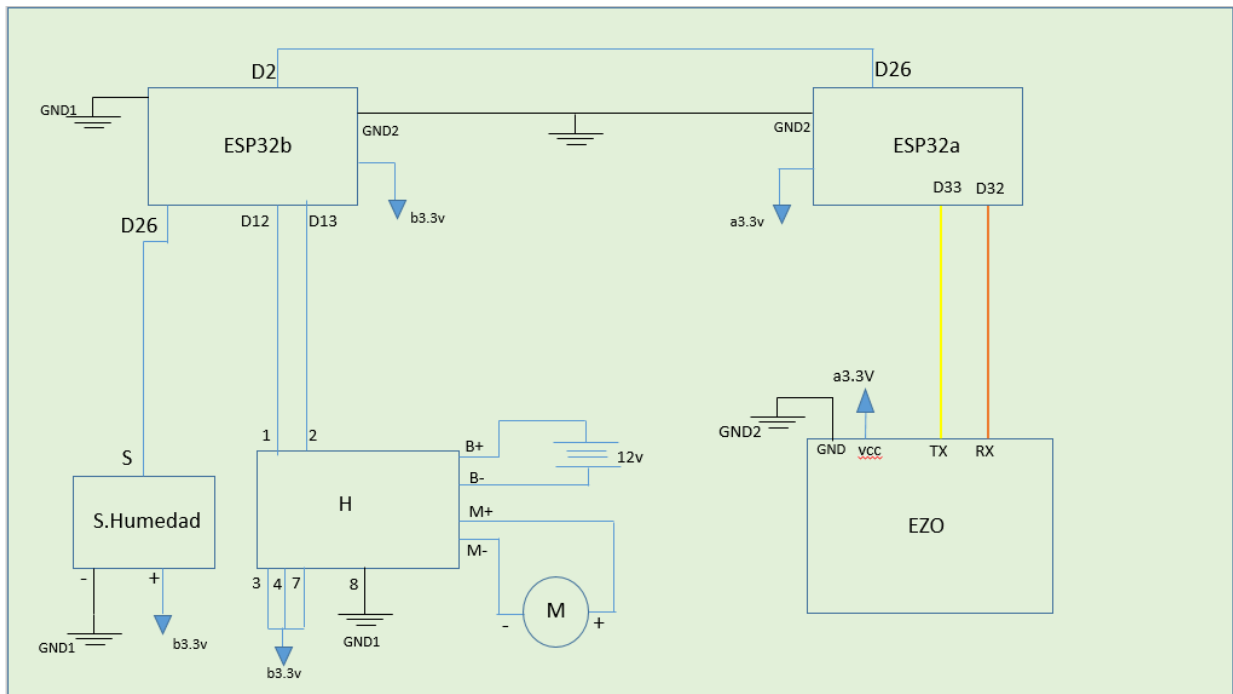
## Diagrama Electrónico SECCIÓN-B

Para tener un mayor entendimiento del funcionamiento de este sistema, es necesario comprender los bloques que serán plasmados en el diagrama, y saber relacionarlos con sus contrapartes físicas.

En este sistema se utilizó la ESP32-Wroom-32, el puente H BTN7960B, el ya conocido circuito eléctricamente aislado de Atlas Scientific, y el sensor de humedad T1592.



Una vez comprendido esto se procede a construir el diagrama electrónico de la siguiente manera.



El común del b3.3v es GND1, así como respectivamente el común de a3.3 es GND2.

Con esto concluye el diagrama electrónico, es obligatorio respetar las conexiones tal cual lo indica el diagrama, esto para el correcto funcionamiento del sistema.

## Rutina de programación.

Como se había explicado antes, la SECCIÓN-B se divide en dos subsistemas, por lo que es una rutina de programación por cada ESP32.

Para poder comenzar a programar ESP32 desde la IDE de Arduino, es necesario primero configurar la IDE, por lo que es necesario consultar una guía para poder lograr esto, la que se recomienda aquí son las siguientes:

<https://www.youtube.com/watch?v=4UfklYUMP1E> GUÍA EN ESPAÑOL.

<https://youtu.be/--Fj8QDIguQ> GUÍA EN INGLES.

La ESP32b se dedica al control de la posición del sensor de oxígeno en función del sensor de nivel de humedad, así como de comunicarle a la ESP32a cuando enviar los datos y cuando no. La ESP32a se dedica exclusivamente a realizar la medición de oxígeno disuelto y a enviar dichos datos a la plataforma ThingSpeak. Cabe destacar que ambos códigos se pueden consultar en [https://github.com/Barrendero/Monitoreo\\_DO\\_CtrlMotor\\_SendDataHTTP](https://github.com/Barrendero/Monitoreo_DO_CtrlMotor_SendDataHTTP). Por lo que de ser necesario inspeccionarlo a detalle, o simplemente copiarlo y pegarlo se puede ingresar al link del repositorio público.

## ESP32b.

```
// Print level sensor value to give a condition
void printValues(void *parameters) {

    // Loop forever, wait for semaphore, and print value
    while (1) {
        val = analogRead(adc_pin);
        if (val <= 100){
            Serial.println("La medida es \t" + String(val));
            digitalWrite(CON, HIGH); //
            vTaskDelay(2000 / portTICK_PERIOD_MS);
            TheState = 1;
        }if (val >= 101){
            Serial.println("La medida es \t" + String(val));
            digitalWrite(CON, LOW); //
            vTaskDelay(2000 / portTICK_PERIOD_MS);
            TheState = 2;
        }
    }
```

En esta parte se realiza la primera tarea, que es obtener los datos del sensor de nivel de líquido, este sensor da un valor entre 0 y 4500 donde 4500 es el máximo del sensor cubierto por líquido, en este caso agua.

Se abre una función while debido a que se están utilizando dos núcleos, y es obligatorio señalarle al núcleo secundario (que es el que controla la parte del sensado y el envío de datos via I2C a la ESP32a) que debe realizar un bucle infinito.

Después se obtiene el valor analógico, en el que se señala una condición, si el valor es menor a 100, entonces se enciende un pin seleccionado para enviar la data, en este caso es el D2, así mismo una variable llamada "TheState" cambiará su valor inicial de "TheState = 0;" a "TheState = 1;" Esto es necesario saberlo ya que en un futuro se mostrarán los casos condicionales en que el motor va a funcionar y esperar.

Por otro lado, cuando "val >= 101" entonces el sensor de nivel y el sensor de oxígeno ya están dentro del agua, por lo que será necesario avisarle a la ESP32a que el sensor ya está dentro, por lo que el D2 pasa a dejar de emitir la señal, así mismo cambia "TheState = 2;".

```
// Control de motor
void MotorCtrl(void *parameters){
    while(1){
        switch(TheState){

            case 0:
                for(int w = 0; w<5; w++){

                    Serial.println("\tIniciando motor...");
                    vTaskDelay(5000 / portTICK_PERIOD_MS);
                }
                Serial.println("\tEl motor está listo");
                break;
```

La segunda tarea es realizada por el núcleo principal, de la misma manera mediante la función "while (1)" conforme al método de programación RTOS se le indica que debe realizar un bucle infinito. Se inicia con la función "switch" para poder utilizar los distintos casos conforme a la variable cambiante "TheState" que ya se vió con anterioridad.

En el caso inicial que es "case 0:" y que corresponde a "TheState = 0;" se procede a esperar durante 5 periodos de 5 segundos, es decir, 25 segundos de inicialización del sistema, esto con el fin de que se pueda tener una lectura certera de la ubicación actual del sensor, así como preparación de las demás ESP32 para que puedan conectarse y enviar datos. En este primer "case 0:" se procede una vez cumplida la espera a pasar al siguiente estado, sea "TheState = 1;" o "TheState = 2;" dependiendo de la lectura del sensor de nivel de líquido.



```

case 1:
//Serial.println("La medida es \t" + String(val));
//digitalWrite(LED1, HIGH);
//delay(3000);
//digitalWrite(LED1, LOW);
for (int motorValue = 0; motorValue <= 60; motorValue += 60){
    ledcWrite(canalPWM, motorValue);
    vTaskDelay(500 / portTICK_PERIOD_MS);
}

for (int motorValue = 60; motorValue >= 0; motorValue -= 60){
    ledcWrite(canalPWM, motorValue);
    vTaskDelay(500 / portTICK_PERIOD_MS);
}

Serial.println("\tEl sensor está dentro!");
vTaskDelay(1000 / portTICK_PERIOD_MS);
break;

```

En caso de que el sensor marque "TheState = 1;" entonces el "case 1:" salta a efectuarse, inicializando con una función "for" que moverá el motor en dirección horaria en un movimiento corto, esto con el fin de no tener errores en cuanto al control del motor y de la ubicación de los sensores, una vez que culminan los ciclos "for" de movimiento del motor se procede a romper el estado actual y a esperar la siguiente lectura del sensor, si el sensor sigue obteniendo un "val <= 100" entonces procederá de nuevo a ejecutar el ciclo "case 1:" hasta que deje de cumplirse.

```

case 2:
    for(int r = 0; r<300; r++){

        Serial.println("\tEl sensor está tomando las medidas");
        vTaskDelay(1000 / portTICK_PERIOD_MS);
    }
    Serial.println("\tEl sensor está listo para subir!");
    TheState = 3;
break;

```

Si, por consiguiente, después de ejecutar el estado anterior se obtiene un valor de "val >= 101" entonces se procede al "case 2:", mismo case que tiene la orden de esperar 300 periodos de un segundo, es decir, 5 minutos de pausa en el motor, mismos que darán el tiempo suficiente para obtener lecturas del sensor de oxígeno, recordando que al ejecutarse este ciclo ya se está avisando a la ESP32a que el oxígeno está en el agua y puede proceder a tomar lecturas, una vez finalizados los 5 minutos se cambia el estado de "TheState" y se convierte en "TheState = 3;" para proceder con el siguiente ciclo.

```

case 3:
//Serial.println("La medida es \t" + String(val));
//digitalWrite(LED2, HIGH);
//delay(3000);
//digitalWrite(LED2, LOW);
for (int motorValue = 0; motorValue <= 100; motorValue += 50){
    ledcWrite(canalPWM1, motorValue);
    vTaskDelay(500 / portTICK_PERIOD_MS);
}

for (int motorValue = 100; motorValue >=0; motorValue -= 50){
    ledcWrite(canalPWM1, motorValue);
    vTaskDelay(500 / portTICK_PERIOD_MS);
}
for(int y = 0; y<10; y++){

    Serial.println("\tEl sensor está saliendo");
    vTaskDelay(1000 / portTICK_PERIOD_MS);
}
Serial.println("\tEl sensor está afuera!");
TheState = 4;
break;

```

Una vez transcurridos los minutos y convertido el “TheState = 3;” comienza el ciclo “case 3:” mismo que procede con la activación del motor en dirección contraria a la que se ejecutó con anterioridad, a una velocidad y una duración mayor, suficiente para poder sacar el motor del agua, así mismo existe una pausa de 10 segundos y se procede al siguiente estado “TheState = 4;”.

```

case 4:
    for(int z = 0; z<900; z++){

        Serial.println("\tEl sensor está en pausa");
        vTaskDelay(1000 / portTICK_PERIOD_MS);
    }
    Serial.println("\tEl sensor está listo para moverse!");
break;
}

```

Una vez que el estado “TheState = 4;” se cumple, se comienza el ciclo “case 4:” mismo que es un ciclo for de 900 periodos de 1 segundo, siendo igual a 15 minutos de espera para que el sensor de nivel de líquido pueda secarse, en este estado con el otro núcleo se siguen obteniendo los datos del sensor de líquido, por lo que una vez fuera del agua puede avisar a la ESP32a que el sensor de oxígeno ya no está enviando datos.

## ESP32a.

Esta segunda rutina se divide en dos partes, el núcleo principal se dedica a la obtención de los datos enviados por la ESP32b con respecto a la ubicación del sensor de nivel de líquido, por lo que también es capaz de cambiar los estados del flag "TheState" aplicado en este microcontrolador ESP32a, esto con el fin de avisarle al sensor de oxígeno disuelto que sense los datos y decirle al núcleo secundario que envíe o no los datos vía internet, tanto de oxígeno disuelto como del estado del sensor.

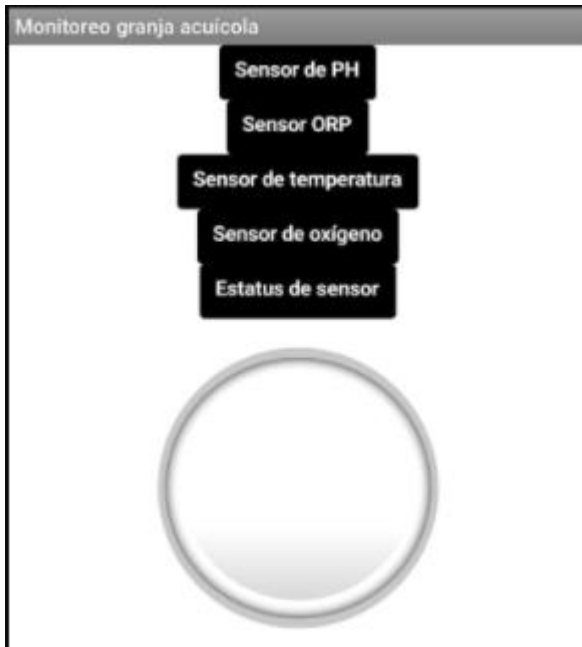
```
void loop(){
    val = digitalRead(dc_pin);
    if(val == 0){
        TheState = 1;
        Serial.println("\t\t\tEl valor es: " + String(val));
        Serial.println("\t\t\tEl sensor está dentro!");
        vTaskDelay(2000 / portTICK_PERIOD_MS);
    }if(val == 1){
        TheState = 2;
        Serial.println("\t\t\tEl valor es: " + String(val));
        Serial.println("\t\t\tEl sensor está afuera!");
        vTaskDelay(2000 / portTICK_PERIOD_MS);
    }
}
```

---

Esta parte se dedica al núcleo principal, donde el valor "val = digitalRead(dc\_pin);" corresponde al valor enviado por la ESP32b, dicho esto, si "val == 0" entonces el estado "TheState" corresponde a "TheState = 1;", lo que indica que el sensor está dentro del agua y nuestro sensor de oxígeno disuelto puede sense los datos, así mismo, se puede enviar al servidor que el sensor está tomando datos tal como se observa en la siguiente imagen:



Por otro lado, si el valor “val == 1” entonces el estado “TheState” corresponde a “TheState = 2;” por lo que se procede a dejar de sensar y enviar datos, así mismo el estado del sensor enviado al servidor indica que no se están sensando datos, como lo indica la siguiente imagen.



Una vez comprendido el núcleo principal, el segundo núcleo está realizando las tareas correspondientes al estado que el núcleo principal le indique.

```
// Task GETOX
void GetOx(void *parameters) {
    while(1){
        // GetOx
        if (TheState == 1){
            ud = 0;

            if (input_string_complete == true) {
                myserial.print(inputstring);
                myserial.print('\r');
                inputstring = "";
                input_string_complete = false;
            }

            if (myserial.available() > 0) {
                char inchar = (char)myserial.read();
                sensorstring += inchar;
                if (inchar == '\r') {
                    sensor_string_complete = true;
                }
            }
        }
    }
}
```

Si “TheState == 1;”, entonces se procede a decir que la variable “ud = 0;” así como a comenzar la rutina de obtención de datos del embebido de Oxígeno Disuelto de Atlas Scientific (esta rutina se puede entender mejor directamente en la guía de atlas Scientific “Dissolved Oxygen Kit”).

```

if (sensor_string_complete == true) {                                //if a
    //Serial.println(sensorstring);                                // Pri
    ox = sensorstring.toInt();
    OxCopia = ox;
    if(OxCopia > 2){
        Serial.println("El oxígeno es: " + String(OxCopia));
        ThingSpeak.setField (5,ud);
        ThingSpeak.setField (4,OxCopia);
        //sensor_string_complete = false;+
        vTaskDelay(1000 / portTICK_PERIOD_MS);
        ThingSpeak.writeFields(channelID,WriteAPIKey);
        Serial.println("Datos enviados a ThingSpeak!");
    }

    //listo_para_enviar = true;
    sensorstring = "";                                              //clea
    sensor_string_complete = false;                                //rese
}
}

```

Una vez que se cumple la rutina del embebido del oxígeno disuelto, se procede a obtener la data del sensor de oxígeno disuelto, se realiza una copia de los datos obtenidos en String y se convierten en un Entero tal como se observa en “Ox = sensorstring.toInt();”.

Una vez realizada la copia y transformación de los datos, se plantea una condicional, en la que si el sensor de oxígeno disuelto es mayor a 2 entonces se procede a enviar la data “ud = 0;” valor que indica que el sensor está trabajando, así como la variable donde se envió la copia de los datos “OxCopia = ox;”, una vez enviados los datos se le avisa al embebido que la obtención de datos ha sido completada para una vez más comenzar con el ciclo de medición “sensor\_string\_complete = false;”, obviamente solo si se cumple con el estado de medición.

```

else{
    ud = 1;
    ThingSpeak.setField (5,ud);
    vTaskDelay(1000 / portTICK_PERIOD_MS);
    ThingSpeak.writeFields(channelID,WriteAPIKey);
    Serial.println("Datos enviados a ThingSpeak!");
}
}
}

```

Por otro lado, si se obtiene un "TheState == 0;" entonces se procede con la variable "ud = 1;" indicando al servidor que el estado del sensor es No operativo, en este ciclo jamás se procede el ciclo en el que se activa el ciclo de trabajo del embebido.

Con esto se concluyen ambas rutinas de programación.

### **Conclusión.**

Aún existen fallas y mejoras, un error que se ha visto en plena operación es que, debido a que el sensor de humedad sigue húmedo al salir del agua, se envían mediciones con el sensor fuera del agua.

Otra son los intervalos de medición del sensor de oxígeno, mismos que interfieren con las mediciones de los sensores de la SECCIÓN-A, esto por el límite del ancho de banda de ThingSpeak.