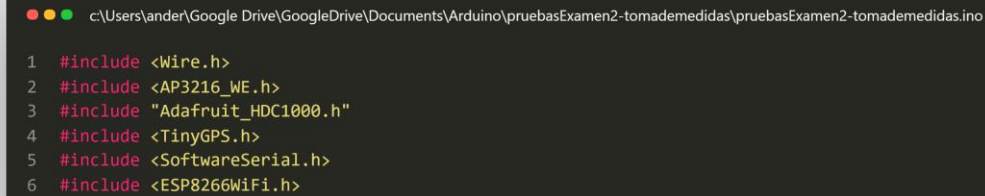


## DESCRIPCIÓN DEL CÓDIGO.

El Objetivo del programa es poder capturar la medida de tres sensores: sensor de temperatura y humedad, sensor de luz y GPS. Para lograr esto, es necesario agregar las librerías que nos permiten la correcta manipulación de estos sensores:

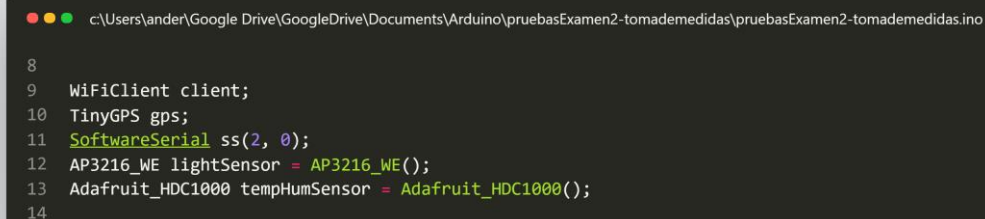


```
c:\Users\ander\Google Drive\GoogleDrive\Documents\Arduino\pruebasExamen2-tomademedidas\pruebasExamen2-tomademedidas.ino

1  #include <Wire.h>
2  #include <AP3216_WE.h>
3  #include "Adafruit_HDC1000.h"
4  #include <TinyGPS.h>
5  #include <SoftwareSerial.h>
6  #include <ESP8266WiFi.h>
```

- **Wire.h:** esta librería nos permite realizar las comunicaciones con los sensores mediante el protocolo de comunicación I2C.
- **AP3216\_WE.h:** es la encargada de controlar y manipular las funciones de nuestro sensor de luz.
- **Adafruit\_HDC1000.h:** con ella podemos acceder a las funcionalidades del sensor HDC1080 (sensor de temperatura y humedad)
- **TinyGPS.h:** Cuando integramos la librería TinyGPS, podemos manipular las funciones relacionadas con el sistema de GPS.
- **SoftwareSerial.h:** esta librería nos ayuda a realizar la comunicación con el GPS, ya que usa un modo de comunicación diferente a los otros dos sensores.
- **ESP8266WiFi.h:** Con esta librería podemos manipular las funciones de conexión a internet de nuestro cerebro ESP.

Ahora el siguiente paso es definir las variables globales que estarán involucradas en el código



```
8
9  WiFiClient client;
10 TinyGPS gps;
11 SoftwareSerial ss(2, 0);
12 AP3216_WE lightSensor = AP3216_WE();
13 Adafruit_HDC1000 tempHumSensor = Adafruit_HDC1000();
14
```

Primero procedemos a crear los objetos correspondientes a cada sensor para controlar sus funcionalidades, en el caso del objeto SoftwareSerial, definimos los pines de transmisión y recepción que en nuestra placa ESP8216 corresponden a los pines D4 y D3 correspondientemente. También se crea un cliente WiFi para tener la posibilidad de enviar datos a nuestro servidor remoto

El siguiente paso es definir las variables que participan en el código

```
c:\Users\ander\Google Drive\GoogleDrive\Documents\Arduino\pruebasExamen2-tomademedidas\pruebasExamen2-tomademedidas.ino

14
15 String serverIP = "54.209.141.166";
16 bool control = false;
17 char* ssid = "MARTINEZ_M";
18 char* password = "8M5^*t/a%1";
19 uint32_t previousMillis = 0;
20 const uint8_t numSamples=20, temperature=0, humidity=1,
21             light=2, positionGPS=3;
22 uint8_t sensor;
23 float samples[numSamples], measures[4];
24 uint16_t id;
25 char date[32];
26 Long latitude, longitude;
27 enum class machineState {INIT, READ_TEMPERATURE, READ_HUMIDITY, READ_LIGHT,
28                          READ_GPS,CHECK_CONNECTION, PRINT_PACKET};
29 machineState state = machineState::INIT;
```

- **serverIP:** es la dirección de nuestro servidor remoto usada para el envío de los datos
- **ssid:** nombre de nuestra red. Es importante tenerla en cuenta a la hora de usar una red distinta para conectar nuestro dispositivo.
- **Password:** la clave de la red WIFI-
- **previousMillis:** esta variables es usada para llevar una cuenta del tiempo sin tener que recurrir al delay para no tener que bloquear el funcionamiento de la máquina. Posteriormente se explicará mejor su uso.
- **numSamples:** indica el número de muestras tomadas por los sensores asíncronos que luego son usadas en el proceso de pruning
- **temperature, humidity, light, positionGPS:** son índices para acceder a las medidas de cada sensor dentro del arreglo **measures** que contiene los datos de cada uno luego del proceso de pruning
- **sensor:** almacena alguno de los valores descritos anteriormente para acceder a ellos.
- **Id:** en esta variable se guarda el id del paquete.
- **date:** acá se almacena la fecha extraída desde el GPS.
- **Latitude, longitude:** almacena las coordenadas de longitud y latitud extraídas del GPS.
- **machineState:** guarda los estados de máquina. Más adelante serán detallados cada uno.

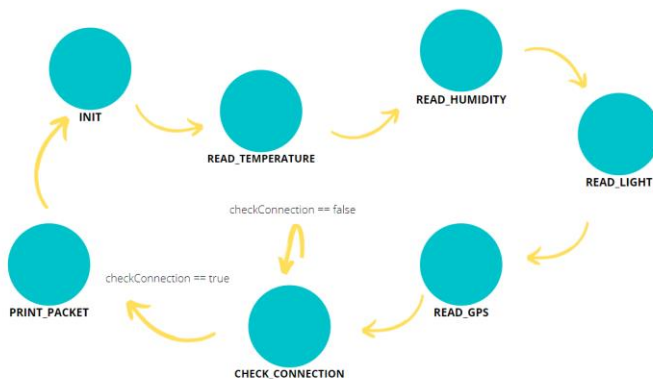
Luego procedemos a realizar la configuración inicial de nuestro sistema, empezamos por iniciar las conexiones Serial, I2C, y la conexión síncrona del GPS, además de conectarnos a la red WIFI (Esta función será descrita más adelante)

Luego configuramos los parámetros del sensor de luz y de temperatura.

```
c:\Users\ander\Google Drive\GoogleDrive\Documents\Arduino\pruebasExamen2-tomademedidas\pruebasExamen2-tomademedidas.ino

31 void setup() {
32   Serial.begin(115200);
33   wifiConnect();
34   Wire.begin();
35   ss.begin(9600);
36   lightSensor.init();
37   lightSensor.setMode(AP3216_ALS_PS);
38   lightSensor.setLuxRange(RANGE_20661);
39   lightSensor.setPSGain(2);
40   lightSensor.setNumberOfLEDPulses(1);
41   lightSensor.setPSMeanTime(PS_MEAN_TIME_50);
42   lightSensor.setPSThresholds(0, 100);
43   lightSensor.setPSIntegrationTime(1);
44   lightSensor.setPSInterruptMode(INT_MODE_HYSTERSIS);
45
46   if(!tempHumSensor.begin()) {
47     Serial.println("No es posible establecer la comunicación con el sensor de temperatura.");
48   }
49   else {
50     Serial.println("Conectado al sensor de temperatura.");
51   }
52   delay(1000);
53 }
54
```

Ahora es momento de pasar al Loop donde se encuentran los estados de máquina definidos



```

c:\Users\ander\Google Drive\GoogleDrive\Documents\Arduino\pruebasExamen2-tomademedidas\pruebasExamen2-tomademedidas.ino

55
56 void loop() {
57     switch(state)
58     {
59         case machineState::INIT:
60             state = machineState::READ_TEMPERATURE;
61             smartdelay(1000);
62             break;
63
64         case machineState::READ_TEMPERATURE:
65             getMeasures(temperature);
66             state = machineState::READ_HUMIDITY;
67             break;
68
69         case machineState::READ_HUMIDITY:
70             getMeasures(humidity);
71             state = machineState::READ_LIGHT;
72             break;
73
74         case machineState::READ_LIGHT:
75             getMeasures(light);
76             state = machineState::READ_GPS;
77             break;
78
79         case machineState::READ_GPS:
80             smartdelay(1000);
81             readDate(gps);
82             gps.get_position(&longitude, &latitude);
83             state = machineState::CHECK_CONNECTION;
84             break;
85
86         case machineState::CHECK_CONNECTION:
87             checkConnection();
88             break;
89
90         case machineState::PRINT_PACKET:
91             if (waitTime(13000)){
92                 packet();
93                 messageToServer();
94             }
95             state = machineState::INIT;
96             break;
97     }
98 }
99
100

```

- **INIT:** estado inicial donde comienza el ciclo de máquina. Se revisa que el GPS tenga datos y se cambia al siguiente estado
- **READ\_TEMPERATURE, READ\_HUMIDITY, READ\_LIGHT:** En estos estados se recoge las medidas de cada uno de los sensores luego de realizar el proceso de pruning.
- **READ\_GPS:** Volvemos a revisar si el GPS y pasamos a leer la fecha y obtenemos las coordenadas de latitud y longitud

- **CHECK\_CONNECTION:** En este estado de máquina, se verifican tanto la conexión a la red WIFI como la conexión al servidor. Si la conexión es fallida, intenta conectarse de nuevo para proseguir al siguiente estado.
- **PRINT\_PACKET:** imprime los datos leídos desde los sensores tanto al puerto serial como al servidor

## **FUNCIONES:**

### Función Chirp.

```
c:\Users\ander\Google Drive\GoogleDrive\Documents\Arduino\pruebasExamen2-tomademedidas\pruebasExamen2-tomademedidas.ino

101 void chirp (uint8_t sensor) {
102     for (uint8_t sample=0; sample < numSamples; sample++) {
103         switch(sensor) {
104             case temperature:
105                 samples[sample] = tempHumSensor.readTemperature();
106                 delay(15);
107                 break;
108
109             case humidity:
110                 samples[sample] = tempHumSensor.readHumidity();
111                 delay(15);
112                 break;
113
114             case light:
115                 samples[sample] = lightSensor.getAmbientLight();
116                 delay(15);
117                 break;
118         }
119     }
120 }
```

En la función chirp nos encargamos de tomar cierta cantidad de muestras dada por la variable numSamples con el fin de eliminar los errores producidos por fallas en la medida de los sensores. Luego, dependiendo el tipo de sensor que hallamos enviado como parámetro, tomará su respectiva medida y lo almacena en un arreglo llamado samples para, posteriormente, realizar el proceso de pruning.

### Función prunning:

```
c:\Users\ander\Google Drive\GoogleDrive\Documents\Arduino\pruebasExamen2-tomademedidas\pruebasExamen2-tomademedidas.ino

121
122 float prunning() {
123     float sum;
124     for (uint8_t sample = 0; sample < numSamples; sample++) {
125         sum += samples[sample];
126     }
127     return sum/numSamples;
128 }
```

En la Función de prunning nos encargamos de obtener el promedio de las muestras que fueron proporcionadas por los sensores y esta será la que se enviará para realizar el proceso de bunding.

### getMeasures:

```
c:\Users\ander\Google Drive\GoogleDrive\Documents\Arduino\pruebasExamen2-tomademedidas\pruebasExamen2-tomademedidas.ino

129
130 void getMeasures(uint8_t sensor) {
131     chirp(sensor);
132     measures[sensor] = prunning();
133 }
```

Esta función se encarga de llamar a las dos anteriores y luego el resultado de cada uno de los sensores los guarda en un arreglo llamado measures que le asigna el lugar dependiendo de la variable sensor.

### Packet:

```

c:\Users\ander\Google Drive\GoogleDrive\Documents\Arduino\pruebasExamen2-tomademedidas\pruebasExamen2-tomademedidas.ino

134
135 void packet() {
136     Serial.print("packet ID:\t");Serial.println(id);
137     Serial.print("Date: \t\t");Serial.println(date);
138     Serial.print("Temperature:\t");Serial.print(measures[temperature]);Serial.println(" °C");
139     Serial.print("Humidity: \t");Serial.print(measures[humidity]);Serial.println(" %");
140     Serial.print("Light:\t\t");Serial.println(measures[light]);
141     Serial.print("Position: \t\n");
142     Serial.print("    Latitude:\t");Serial.println(latitude);
143     Serial.print("    Longitude:\t");Serial.println(longitude);
144
145     id++;
146 }

```

La función packet se encarga de imprimir los valores obtenidos por los diferentes sensores y el GPS en el puerto serial de manera ordenada de tal manera que podamos hacer un rastreo sobre el resultado esperado antes de enviarlo a la nube.

ReadDate:

```

c:\Users\ander\Google Drive\GoogleDrive\Documents\Arduino\pruebasExamen2-tomademedidas\pruebasExamen2-tomademedidas.ino

147 void readDate(TinyGPS &gps) {
148     int year;
149     byte month, day, hour, minute, second, hundredths;
150     unsigned long age;
151     gps.crack_datetime(&year, &month, &day, &hour, &minute, &second, &hundredths, &age);
152     if (age == TinyGPS::GPS_INVALID_AGE) {
153     }
154     else {
155         sprintf(date, "%02d/%02d/%02d %02d:%02d:%02d ",
156             month, day, year, hour, minute, second);
157     }
158     smartdelay(0);
159 }
160 }

```

Esta función nos permite obtener la hora y fecha desde nuestro sensor GPS y almacenarla en una variable para posteriormente trabajarla.

### SmartDelay:

La función smartDelay se encargar de leer periódicamente los datos suministrador por el sensor GPS, ya que al ser una comunicación síncrona, requiere estar constantemente recibiendo datos.

```
c:\Users\ander\Google Drive\GoogleDrive\Documents\Arduino\pruebasExamen2-tomademedidas\pruebasExamen2-tomademedidas.ino

160 void smartdelay(unsigned Long ms)
161 {
162     unsigned Long start = millis();
163     do
164     {
165         while (ss.available())
166             gps.encode(ss.read());
167     } while (millis() - start < ms);
168 }
169
```

### parseToDegrees (función de prueba, no terminada aún)

```
c:\Users\ander\Google Drive\GoogleDrive\Documents\Arduino\pruebasExamen2-tomademedidas\pruebasExamen2-tomademedidas.ino

170
171 void parseToDegrees(Long decimal) {
172     bool isNeg = decimal < 0 ? true : false;
173     int hours, minutes;
174     float sec, resHours, resMinutes;
175     if (isNeg) decimal * -1;
176     //decimal/=100000;
177     resHours = decimal - (int)(decimal)<0 ? 1+(decimal - (int)decimal) : decimal - (int)decimal;
178     hours = (int)(decimal - resHours);
179     resMinutes = (resHours * 60)<1 ? 0 : (resHours * 60);
180     resMinutes = minutes - (int)(minutes)<0 ? 1+(minutes - (int)minutes) : minutes - (int)minutes;
181     minutes = (int)(minutes - resMinutes);
182     sec = resMinutes*60;
183     Serial.print(hours);Serial.print(minutes);Serial.println(sec);
184 }
185
```



Esta función lo que pretende es poder convertir los valores de longitud y latitud dados en grados centesimales a grados cegesimales.

### WifiConnect

```
c:\Users\ander\Google Drive\GoogleDrive\Documents\Arduino\pruebasExamen2-tomademedidas\pruebasExamen2-tomademedidas.ino

186 void wifiConnect()
187 {
188     IPAddress staticIP(192,168,1,50);
189     IPAddress gateway(192,168,1,254);
190     IPAddress subnet(255,255,255,0);
191     Serial.println();
192     Serial.printf("Connecting to %s\n", ssid);
193     WiFi.config(staticIP, gateway, subnet);
194     WiFi.begin(ssid, password);
195     while (WiFi.status() != WL_CONNECTED)
196     {
197         delay(500);
198         Serial.print(WiFi.status());
199     }
200     Serial.println();
201     Serial.print("Connected, IP address: ");
202     Serial.println(WiFi.localIP());
203 }
```

En esta parte del código nos encargamos de conectar nuestro ESP a la red de internet. Es importante recalcar que en nuestro caso hubieron ciertos inconvenientes a la hora de configurar de manera dinámica la conexión, por eso se recurrió a una dirección fija.

### messageToServer:

```
c:\Users\ander\Google Drive\GoogleDrive\Documents\Arduino\pruebasExamen2-tomademedidas\pruebasExamen2-tomademedidas.ino

204 void messageToServer() {
205     String postData = String("packetID: "+String(id)+";Temperature: "+String(measures[0])+";Humidity: "+String(measures[1])+
206                        ";Light: "+String(measures[2])+";Longitude: "+String(latitude)+
207                        ";Latitude: "+String(longitude)) ;
208     client.connect(serverIP,80);
209     client.println("POST /data HTTP/1.1");
210     // poner la direccion IP del servidor
211     client.print("Host: "+ serverIP +"\n");
212     client.println("User-Agent: Arduino/1.0");
213     client.println("Connection: close");
214     client.println("Content-Type: text/plain");
215     client.print("Content-Length: ");
216     client.println(postData.length());
217     client.println();
218     client.println(postData);
219 }
```

En esta sección, nos encargamos de escribir el mensaje que será enviado al servidor, los datos serán enviados a través de el cuerpo del mensaje

### waitTime

```
c:\Users\ander\Google Drive\GoogleDrive\Documents\Arduino\pruebasExamen2-tomademedidas\pruebasExamen2-tomademedidas.ino

220 bool waitTime (uint32_t timeToAwait) {
221     uint32_t currentMillis = millis();
222     if (currentMillis - previousMillis > timeToAwait) {
223         previousMillis = currentMillis;
224         control = true;
225     }
226     else {
227         control = false;
228     }
229     return control;
230 }
```

Con la función waitTime, se crea un registro que almacena el tiempo actual. Esta función es usada con el fin de que los mensajes solo sean transmitidos cada 15 segundos, así que solo devuelve como verdadero cuando ha transcurrido el tiempo esperado.

### checkConnection

```
c:\Users\ander\Google Drive\GoogleDrive\Documents\Arduino\pruebasExamen2-tomademedidas\pruebasExamen2-tomademedidas.ino

231 void checkConnection() {
232     if(WiFi.status() == WL_CONNECTED) {
233         if(!client.connected()) {
234             while(!client.connect(serverIP,80)) {
235                 Serial.print(".");
236                 delay(500);
237             }
238         }
239         state = machineState::PRINT_PACKET;
240     }
241     else {
242         wifiConnect();
243     }
244 }
```

por último tenemos la función `checkConnection`, que nos permite monitorear el estado de la conexión tanto con el servidor como con la red WIFI que estemos usando