Guía implementación un sistema de monitorización de un huerto utilizando arduino y Raspberry Pi

IES Arroyo de la Miel

Juan Antonio Manceras
José Antonio Caballero Tejero
José Ruiz Castillo
María Inmaculada Gormaz García
Alejandra Baez Durán
Diego Cándido Rosado Fuentes

Introducción

Con el desarrollo de un huerto en el centro se ha planteado el proyecto de crear un sistema de monitorización de las condiciones ambientales para el mejor mantenimiento del huerto del centro.
Los alumnos colaborarán en el desarrollo de las herramientas de monitorización, integración del hardware a utilizar (arduino y raspberry pi) y desarrollo de software para el control y publicación en una web de las condiciones ambientales del huerto.
A su vez habrá una colaboración muy estrecha entre este grupo de trabajo y el proyecto de creación del centro para que las condiciones ambientales publicadas en la web puedan ser utilizadas por aquellas personas que trabajan en el huerto.

Miembros del grupo

El curso lectivo 2019-2020 empezamos con esta iniciativa. Fue una iniciativa muy interesante y un proyecto ambicioso, y como es normal, el primer año cometimos muchos errores y nos faltó mucho por hacer.
Este nuevo curso lectivo, 2020-2021, volvemos con muchas ganas, algunas lecciones aprendidas pero sabiendo que seremos más ambiciosos y volveremos a cometer errores. Lo sabemos y sabemos también que es la manera de seguir aprendiendo.

Pues este grupo de intrépidos educadores está formado por:

- Juan Antonio Manceras
- José Antonio Caballero Tejero
- José Ruiz Castillo
- María Inmaculada Gormaz García
- Alejandra Baez Duran
- Diego Cándido Rosado Fuentes
- J. Javier Bueno Moreno

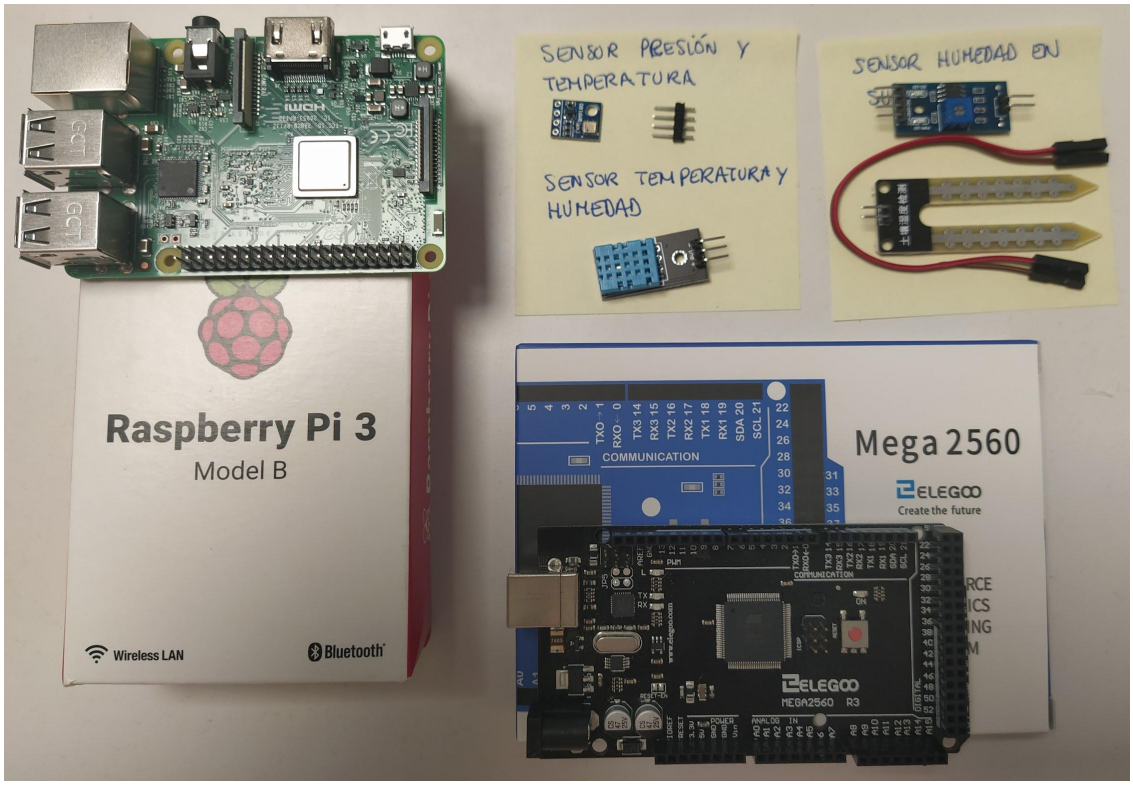
Componentes

En este apartado vamos a listar los componentes que vamos a utilizar en el proyecto. La finalidad de este listado es tener todo lo necesario para poder replicar la experiencia del proyecto.

Listado de componentes a utilizar en el proyecto:

- **Raspberry Pi**: cualquier versión puede utilizarse. Para este proyecto vamos a usar la versión de Raspberry Pi 3.
- Una placa **arduino**: nosotros vamos a utilizar la placa arduino Mega 25600.
- Sensor de presión y temperatura **GY-68 BMP180**.
- Sensor de temperatura y humedad **DHT11**.
- Sensor de humedad en terreno **DollaTek DZ0325**.

Aquí os mostramos una foto con los componentes principales del proyecto.



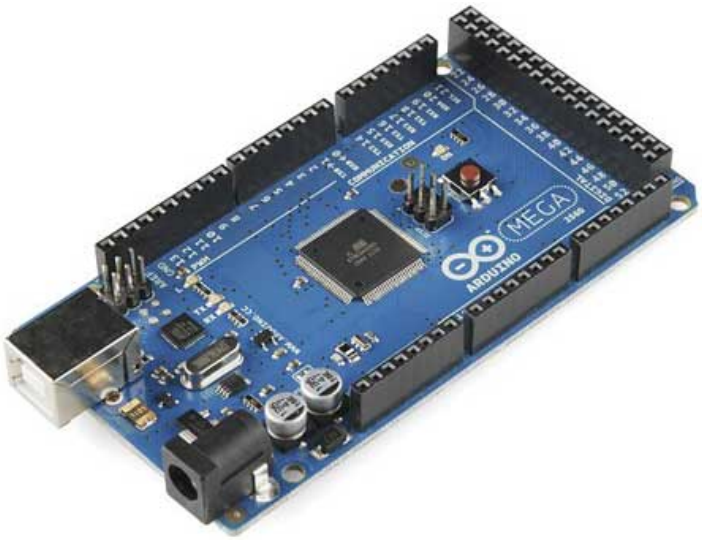
Estos son los sensores que hemos usado nosotros pero podéis usar los sensores que queráis para hacer este proyecto.

Arduino Mega 2560

Arduino MEGA 2560 es una placa de desarrollo basada en el microcontrolador **ATmega2560** (de aquí su nombre). Esta placa pertenece a la extensa familia de placas Arduino, siendo junto al Arduino UNO de las más representativas.

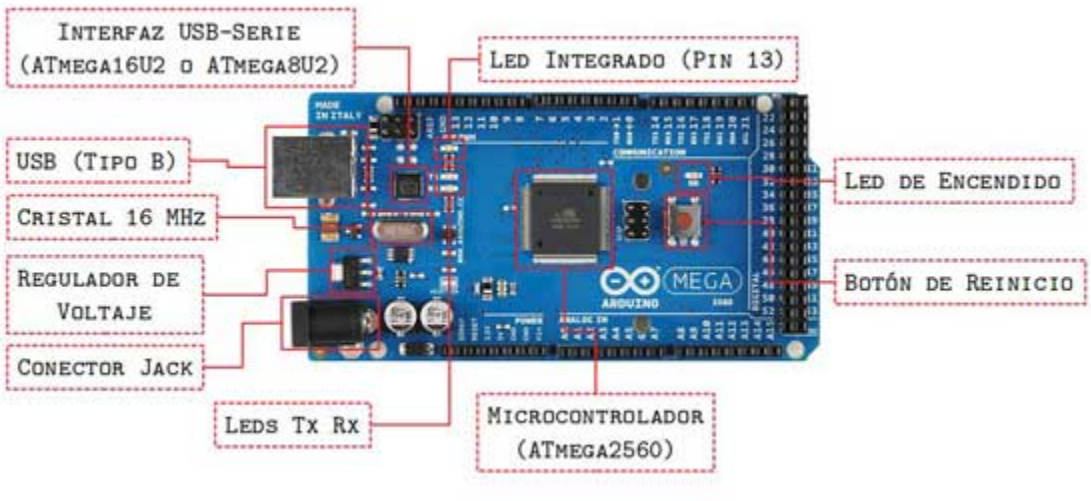
Como es costumbre en esta familia de placas, el Arduino MEGA 2560 está compuesto, básicamente, por:

- **Un microcontrolador (ATmega2560)** con la configuración de "sistema mínimo" (El término "sistema mínimo" se refiere a que solo se utilizan los componentes indispensables para el microcontrolador).
- **Una interfaz USB-Serie** que permite re-programar dicho microcontrolador utilizando simplemente un ordenador, un cable USB y el software Arduino IDE.
- Y un **conjunto de cabezales** que permiten conectar los pines de entrada/salida, ya sea con los conocidos shields o con cualquier otro sistema externo.

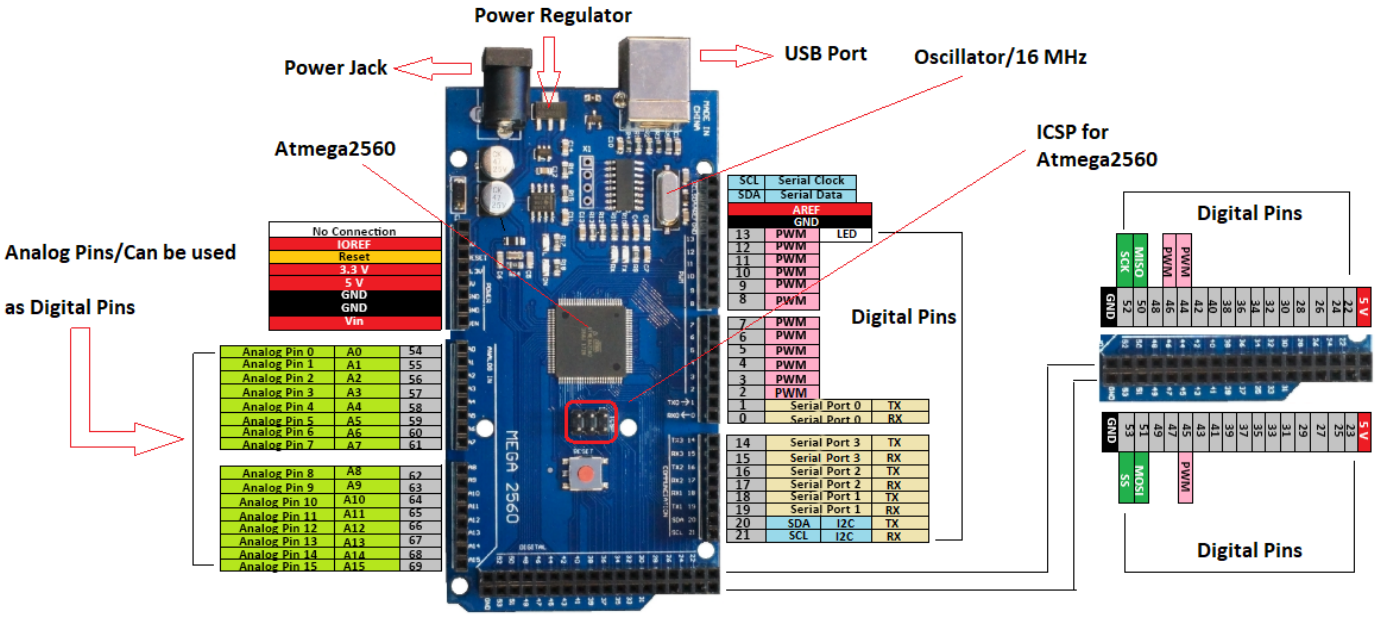


Componentes principales

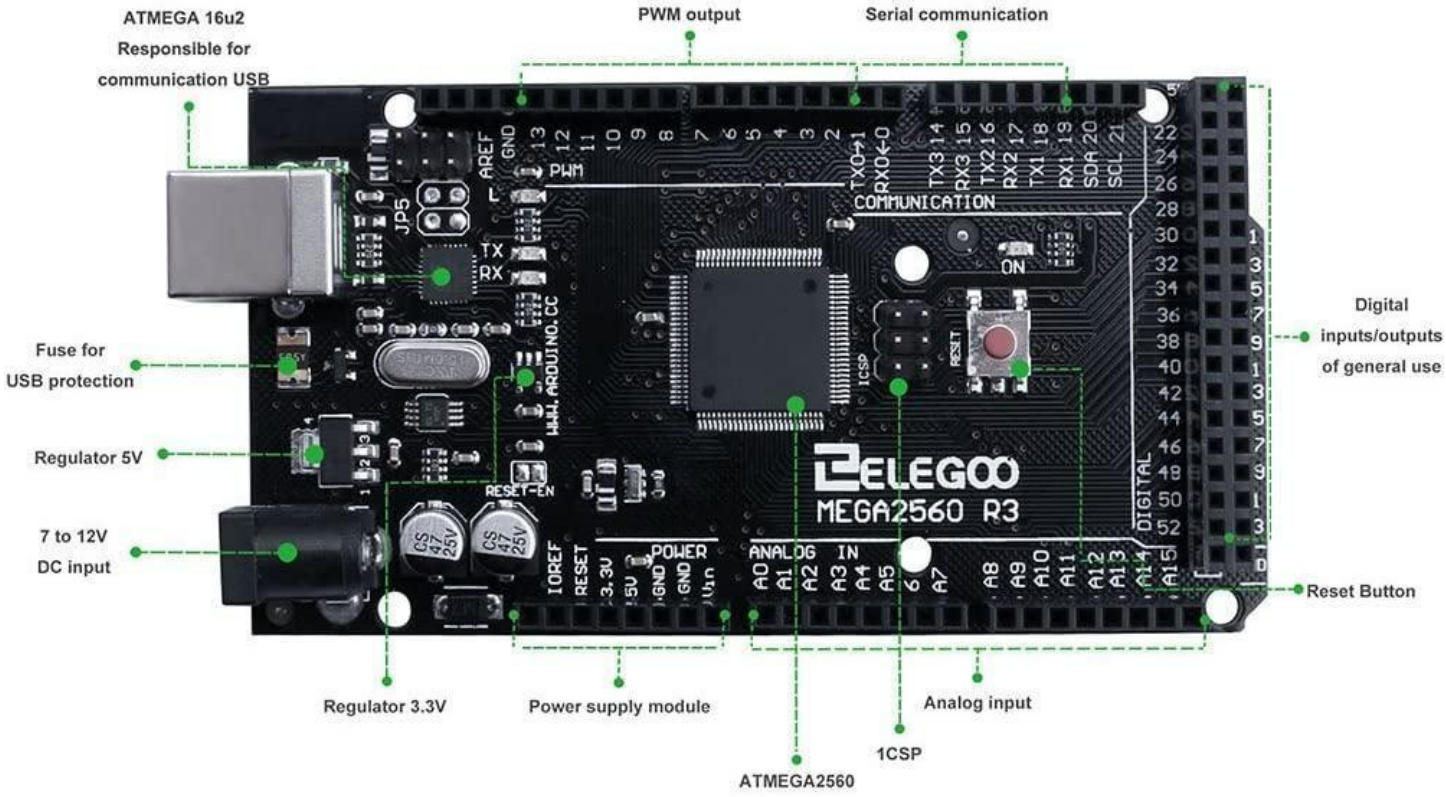
En la imagen se muestra un Arduino MEGA 2560 con sus principales componentes señalados. Durante todo el documento intentaremos explicaros cada uno de los componentes que vayamos utilizando para que comprendas cómo funcionan en conjunto.



De especial relevancia es conocer la distribución de los pines de comunicación y alimentación.



En concreto nosotros utilizaremos una versión de esta placa de desarrollo, la [Elegoo Mega 2560 Rev.3](), que podéis ver a continuación.



Empezamos

Pues nada, vamos a empezar ya con nuestro proyecto.

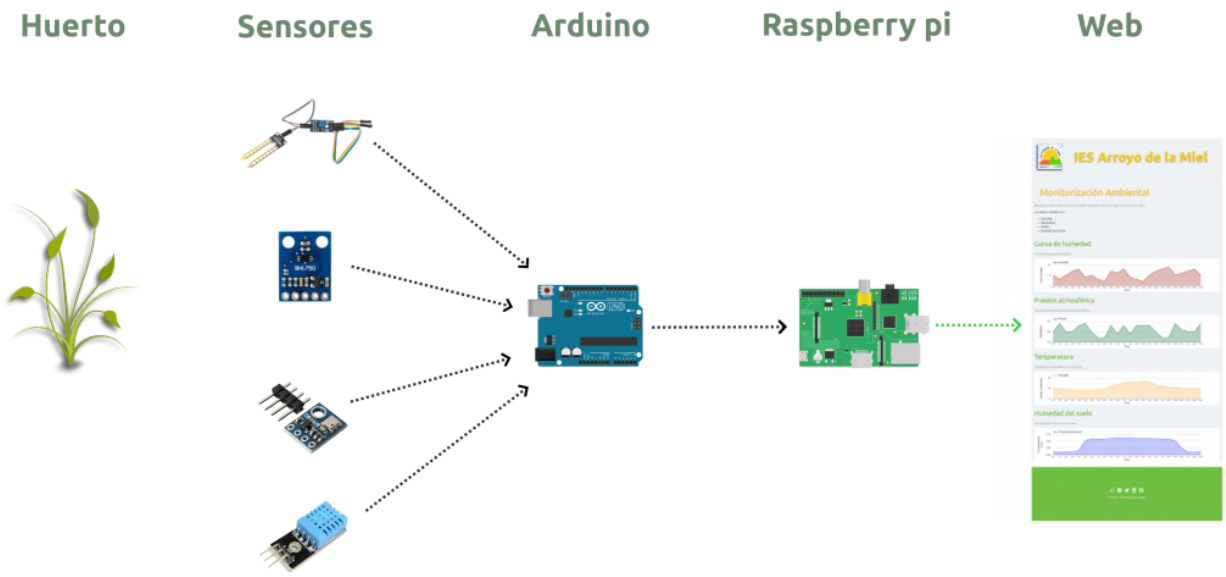
Suponemos que ya tenemos ...

Damos por hecho que en la Raspberry Pi tenemos instalado un sistema linux. En nuestro caso utilizamos el Sistema Operativo Raspberry Pi OS.
Respecto al arduino damos por hecho que tenemos el entorno de desarrollo para arduino, Arduino IDE, instalado en el ordenador que vayamos a utilizar para el desarrollo software.

Si queréis encontrar como instalar Raspberry Pi OS o el entorno de desarrollo Arduino IDE, internet está lleno de tutoriales, videos, guías, ...

Diagrama de despliegue

Para tener una primera idea de lo que queremos hacer vamos a mostrar este diagrama de despliegue de la aplicación. No se trata de un diagrama de despliegue en el sentido tradicional de la palabra pero nos va a dar una buena visión de lo que queremos conseguir.



Lo primero que aparece en el diagrama es el **huerto** que queremos monitorizar. En el huerto tendremos que poner los **sensores** que queramos para monitorizar el huerto. Es decir, si queremos un sensor de temperatura y varios de humedad de la tierra, eso será lo que tendremos que poner.

Tras poner los sensores tendremos que conectarlos a un **arduino** para que controle la lectura de datos. Ese arduino estará conectado a una **raspberry pi** que irá recolectando todos esos datos y guardandolos en una base de datos.

Por último tendremos una **web** que recogerá los datos almacenados en la base de datos y los mostrará por pantalla.

Esperamos que esta primera introducción al proyecto os aclare lo que esperamos conseguir.

Leyendo de los sensores

Vamos a empezar leyendo la información que nos dan los sensores. Para ello conectaremos los sensores al arduino e iremos leyendo los valores que nos dan esos sensores.

El sensor GY-68 BMP180

En esta sección veremos cómo conectar el sensor GY-68 BMP180 y leer datos desde el arduino.

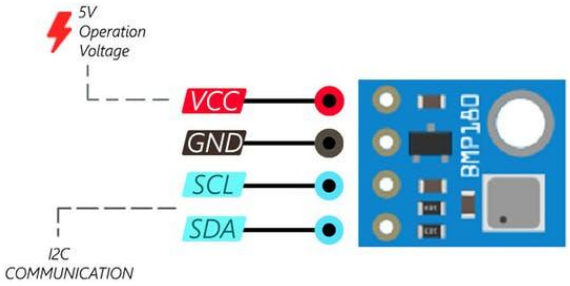
¿Para qué sirve el sensor GY-68 BMP180?

Con el sensor GY-68 BMP180 podemos leer la presión atmosférica y la temperatura.
Este sensor es un barómetro y un termómetro ambiente. Gracias a la relación que existe entre la presión atmosférica y la altura este sensor se puede utilizar también para conocer la altitud a la que nos encontramos.
(Nota: imagen atribuida a www.electropeak.com. Licencia CC - By - SA)

¿Cómo funciona?

El sensor GY-68 BMP180 tiene un barómetro integrado en la circuitería tal y como se aprecia en la imagen. Por medio de ese barómetro se conoce la presión atmosférica que después será traducida a impulsos que se enviarán por el puerto I2C. Con el sensor de temperatura ocurre exactamente lo mismo.
(Nota: imagen atribuida a www.electropeak.com. Licencia CC - By - SA)

En la siguiente imagen se muestra como debería ser el conexionado del sensor con el puerto I2C de arduino.

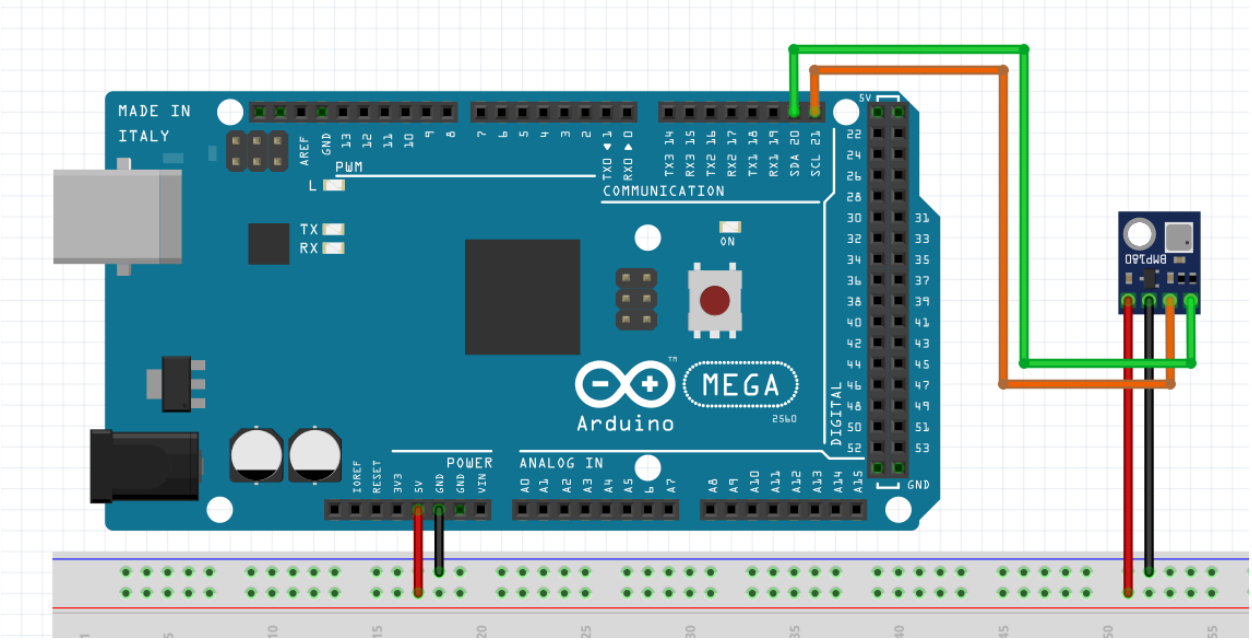


(Nota: imagen atribuida a www.electropeak.com. Licencia CC - By - SA)

Esquema de conexionado

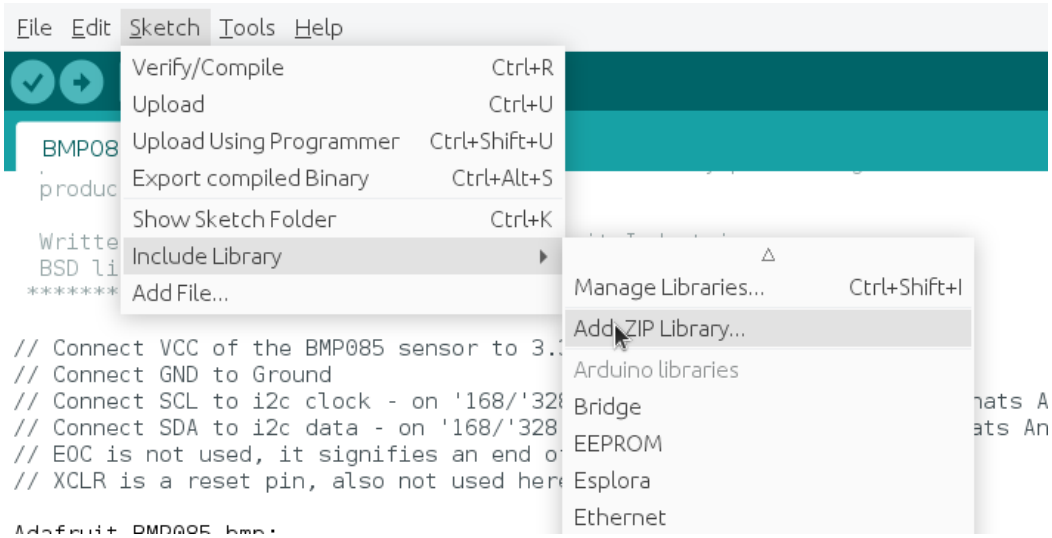
Tenemos que conectar los pines VCC y GND del sensor a los pines VCC (5V) y GND de la placa arduino, y los pines SCL y SDA del sensor a los pines SCL (21) y SDA (20) de la placa arduino.

En la siguiente imagen podemos ver el esquema de conexionado seguido para este ejercicio.

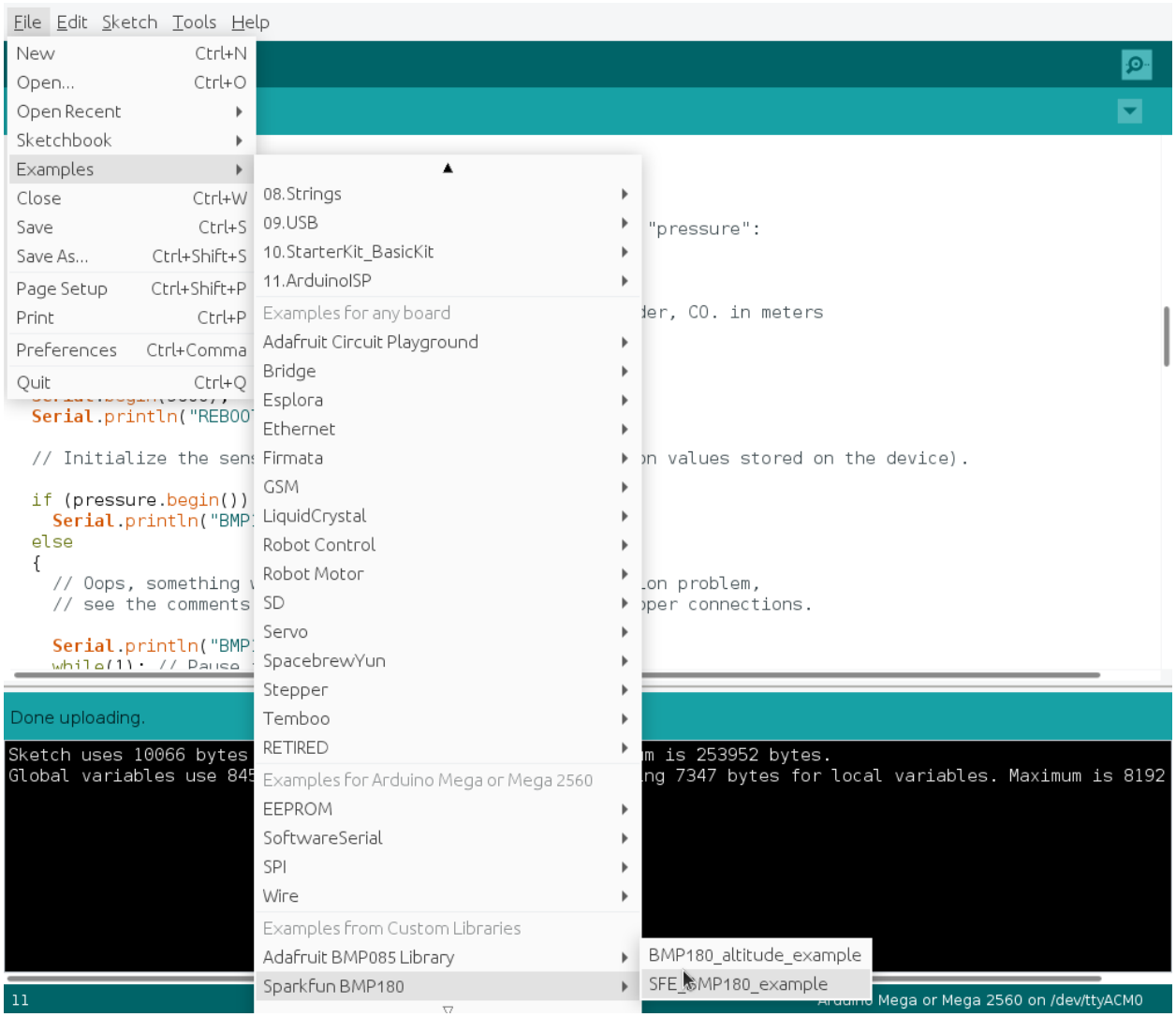


Instalando librerías

Llegados a este punto vamos a intentar leer los datos del sensor de datos y nuestro arduino.
Lo primero que tenemos que hacer es descargarnos la librería de arduino para el sensor BMP180 que podéis encontrar en este enlace de github.
Tenemos que instalar la librería en nuestro Arduino `IDE, Sketch -> Include Library -> Add .ZIP Library .`



Tras añadir la librería podemos cargar el programa de ejemplo del sensor que podemos encontrar en `File -> Examples -> Sparkfun BMP180 -> SPF_BMP180_Example .`



Leyendo del sensor GY-68 BMP180

A continuación vamos a analizar cómo queda el código fuente y la funcionalidad de cada una de las partes.

Lo primero que tenemos que hacer es incluir un par de librerías. La primera librería es SFE_BMP180.h que es la librería del sensor. La segunda librería es Wire.h , que es la librería que nos ayudará a leer utilizando el puerto I2C .

```
#include <SFE_BMP180.h>
#include <Wire.h>
```

A continuación definimos un objeto de tipo `SFE_BMP180` al que llamamos `pressure`.

```
// You will need to create an SFE_BMP180 object, here called "pressure":
SFE_BMP180 pressure;
```

También definimos una constante con la directiva `#DEFINE` a la que llamamos `ALTITUDE` y donde pondremos la altitud a la que estamos realizando la lectura. Por defecto vamos a poner 1665 pero tendremos que poner la altitud a la que los sensores van a estar conectados.

```
#define ALTITUDE 1655.0 // Altitude of SparkFun's HQ in Boulder, CO. in
meters
```

En el método de setup() empezamos inicializando la comunicación por el puerto serie y mostrando un mensaje por el puerto serie.

```
void setup()
{
  Serial.begin(9600);
  Serial.println("REBOOT");
```

A continuación inicializamos el sensor de presión atmosférica. Si se produce un error en la inicialización se muestra un mensaje por el puerto serie y se hace un bucle infinito sin hacer nada. A esta técnica se le conoce con el nombre de "espera activa".

```
  // Initialize the sensor (it is important to get calibration values
stored on the device).
  if (pressure.begin())
    Serial.println("BMP180 init success");
  else
  {
    // Oops, something went wrong, this is usually a connection problem,
    // see the comments at the top of this sketch for the proper
connections.

    Serial.println("BMP180 init fail\n\n");
    while(1); // Pause forever.
  }
```

```
}
```

En el bucle principal empezamos definiendo una serie de variables que utilizaremos posteriormente.

```
void loop()
{
  char status;
  double T,P,p0,a;
```

Mostramos por el puerto serie la altura que hemos definido tanto en metros como en pies.

```
  // If you want sea-level-compensated pressure, as used in weather
reports,
  // you will need to know the altitude at which your measurements are
taken.
  // We're using a constant called ALTITUDE in this sketch:
  Serial.println();
  Serial.print("provided altitude: ");
  Serial.print(ALTITUDE,0);
  Serial.print(" meters, ");
  Serial.print(ALTITUDE*3.28084,0);
  Serial.println(" feet");
```

Empezamos a intentar leer la presión atmosférica. Si el valor de retorno de `startTemperature()` no es `0` quiere decir que todavía no está listo el sensor y nos devuelve un valor numérico que será el tiempo que tenemos que esperar hasta que el sensor esté listo para leer.

```
  // Start a temperature measurement:
  // If request is successful, the number of ms to wait is returned.
  // If request is unsuccessful, 0 is returned.

  status = pressure.startTemperature();
  if (status != 0)
  {
    // Wait for the measurement to complete:
    delay(status);
```

Después de esperar ese tiempo el sensor estará listo para poder realizar la lectura, por eso llamamos a la función `getTemperature(T)`, donde la temperatura se almacenará en el variable `T`, mientras que la función devolverá un valor de `status`.
Si el valor de `status` es `1` es que la operación de lectura se ha realizado con éxito y mostraremos la temperatura por el puerto serie.

```
    // Retrieve the completed temperature measurement:
    // Note that the measurement is stored in the variable T.
    // Function returns 1 if successful, 0 if failure.
    status = pressure.getTemperature(T);
    if (status != 0)
    {
      // Print out the measurement:
      Serial.print("temperature: ");
      Serial.print(T,2);
      Serial.print(" deg C, ");
      Serial.print((9.0/5.0)*T+32.0,2);
      Serial.println(" deg F");
```

Tras esto hacemos el mismo proceso parecido para la lectura de la presión atmosférica.

```
      // Start a pressure measurement:
      // The parameter is the oversampling setting, from 0 to 3 (highest
res, longest wait).
      // If request is successful, the number of ms to wait is returned.
      // If request is unsuccessful, 0 is returned.
      status = pressure.startPressure(3);
      if (status != 0)
      {
        // Wait for the measurement to complete:
        delay(status);

        // Retrieve the completed pressure measurement:
        // Note that the measurement is stored in the variable P.
        // Note also that the function requires the previous temperature
measurement (T).
        // (If temperature is stable, you can do one temperature
measurement for a number of pressure measurements.)
```

```
        // Function returns 1 if successful, 0 if failure.
        status = pressure.getPressure(P,T);
        if (status != 0)
        {
          // Print out the measurement:
          Serial.print("absolute pressure: ");
          Serial.print(P,2);
          Serial.print(" mb, ");
          Serial.print(P*0.0295333727,2);
          Serial.println(" inHg");

          // The pressure sensor returns abolute pressure, which varies
with altitude.
          // To remove the effects of altitude, use the sealevel function
and your current altitude.
          // This number is commonly used in weather reports.
          // Parameters: P = absolute pressure in mb, ALTITUDE = current
altitude in m.
          // Result: p0 = sea-level compensated pressure in mb

          p0 = pressure.sealevel(P,ALTITUDE); // we're at 1655 meters
(Boulder, CO)
          Serial.print("relative (sea-level) pressure: ");
          Serial.print(p0,2);
          Serial.print(" mb, ");
          Serial.print(p0*0.0295333727,2);
          Serial.println(" inHg");

          // On the other hand, if you want to determine your altitude from
the pressure reading,
          // use the altitude function along with a baseline pressure
(sea-level or other).
          // Parameters: P = absolute pressure in mb, p0 = baseline
pressure in mb.
          // Result: a = altitude in m.

          a = pressure.altitude(P,p0);
          Serial.print("computed altitude: ");
          Serial.print(a,0);
          Serial.print(" meters, ");
          Serial.print(a*3.28084,0);
```

```
          Serial.println(" feet");
        }
        else Serial.println("error retrieving pressure measurement\n");
      }
      else Serial.println("error starting pressure measurement\n");
    }
    else Serial.println("error retrieving temperature measurement\n");
  }
  else Serial.println("error starting temperature measurement\n");
```

Por último hacemos una espera de 5 segundos para intentar volver a leer del sensor.

```
  delay(5000);  // Pause for 5 seconds.
}
```

Código fuente completo

Tras haber analizado el código fuente por partes presentamos ahora el código fuente completo en un único bloque para que se pueda analizar mejor.

```
#include <SFE_BMP180.h>
#include <Wire.h>

// You will need to create an SFE_BMP180 object, here called "pressure":

SFE_BMP180 pressure;

#define ALTITUDE 1655.0 // Altitude of SparkFun's HQ in Boulder, CO. in
meters

void setup()
{
  Serial.begin(9600);
  Serial.println("REBOOT");

  // Initialize the sensor (it is important to get calibration values
stored on the device).
```

```
  if (pressure.begin())
    Serial.println("BMP180 init success");
  else
  {
    // Oops, something went wrong, this is usually a connection problem,
    // see the comments at the top of this sketch for the proper
connections.

    Serial.println("BMP180 init fail\n\n");
    while(1); // Pause forever.
  }
}

void loop()
{
  char status;
  double T,P,p0,a;

  // Loop here getting pressure readings every 10 seconds.

  // If you want sea-level-compensated pressure, as used in weather
reports,
  // you will need to know the altitude at which your measurements are
taken.
  // We're using a constant called ALTITUDE in this sketch:

  Serial.println();
  Serial.print("provided altitude: ");
  Serial.print(ALTITUDE,0);
  Serial.print(" meters, ");
  Serial.print(ALTITUDE*3.28084,0);
  Serial.println(" feet");

  // If you want to measure altitude, and not pressure, you will instead
need
  // to provide a known baseline pressure. This is shown at the end of the
sketch.

  // You must first get a temperature measurement to perform a pressure
reading.
```

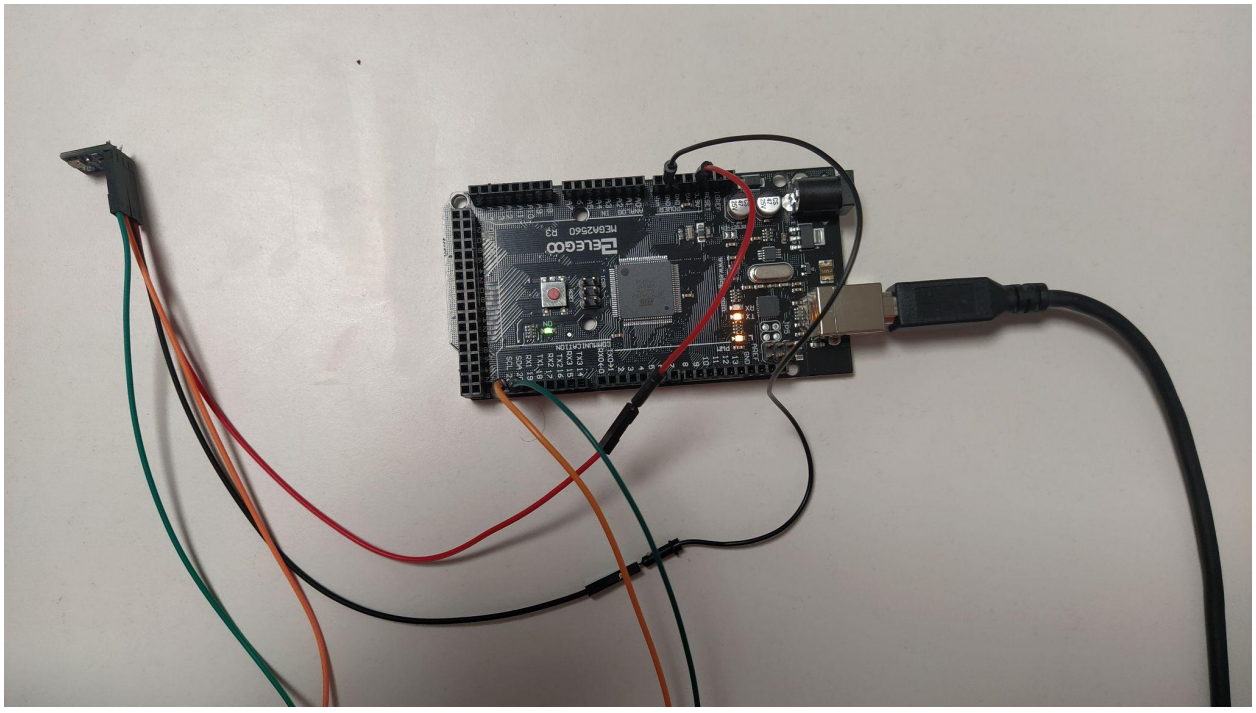
```
  // Start a temperature measurement:
  // If request is successful, the number of ms to wait is returned.
  // If request is unsuccessful, 0 is returned.

  status = pressure.startTemperature();
  if (status != 0)
  {
    // Wait for the measurement to complete:
    delay(status);

    // Retrieve the completed temperature measurement:
    // Note that the measurement is stored in the variable T.
    // Function returns 1 if successful, 0 if failure.

    status = pressure.getTemperature(T);
    if (status != 0)
    {
      // Print out the measurement:
      Serial.print("temperature: ");
      Serial.print(T,2);
      Serial.print(" deg C, ");
      Serial.print((9.0/5.0)*T+32.0,2);
      Serial.println(" deg F");

      // Start a pressure measurement:
      // The parameter is the oversampling setting, from 0 to 3 (highest
res, longest wait).
      // If request is successful, the number of ms to wait is returned.
      // If request is unsuccessful, 0 is returned.

      status = pressure.startPressure(3);
      if (status != 0)
      {
        // Wait for the measurement to complete:
        delay(status);

        // Retrieve the completed pressure measurement:
        // Note that the measurement is stored in the variable P.
        // Note also that the function requires the previous temperature
measurement (T).
        // (If temperature is stable, you can do one temperature
```

```
measurement for a number of pressure measurements.)
        // Function returns 1 if successful, 0 if failure.

        status = pressure.getPressure(P,T);
        if (status != 0)
        {
          // Print out the measurement:
          Serial.print("absolute pressure: ");
          Serial.print(P,2);
          Serial.print(" mb, ");
          Serial.print(P*0.0295333727,2);
          Serial.println(" inHg");

          // The pressure sensor returns abolute pressure, which varies
with altitude.
          // To remove the effects of altitude, use the sealevel function
and your current altitude.
          // This number is commonly used in weather reports.
          // Parameters: P = absolute pressure in mb, ALTITUDE = current
altitude in m.
          // Result: p0 = sea-level compensated pressure in mb

          p0 = pressure.sealevel(P,ALTITUDE); // we're at 1655 meters
(Boulder, CO)
          Serial.print("relative (sea-level) pressure: ");
          Serial.print(p0,2);
          Serial.print(" mb, ");
          Serial.print(p0*0.0295333727,2);
          Serial.println(" inHg");

          // On the other hand, if you want to determine your altitude from
the pressure reading,
          // use the altitude function along with a baseline pressure
(sea-level or other).
          // Parameters: P = absolute pressure in mb, p0 = baseline
pressure in mb.
          // Result: a = altitude in m.

          a = pressure.altitude(P,p0);
          Serial.print("computed altitude: ");
          Serial.print(a,0);
```

```
          Serial.print(" meters, ");
          Serial.print(a*3.28084,0);
          Serial.println(" feet");
        }
        else Serial.println("error retrieving pressure measurement\n");
      }
      else Serial.println("error starting pressure measurement\n");
    }
    else Serial.println("error retrieving temperature measurement\n");
  }
  else Serial.println("error starting temperature measurement\n");

  delay(5000);  // Pause for 5 seconds.
}
```

También podéis encontrar el código en la carpeta del repo de github, aquí.

Resultado final



Como resultado de ejecutar este programa podemos ver la salida que se genera en el monitor serial.

```
/dev/ttyACM0

REBOOT
BMP180 init success

provided altitude: 1655 meters, 5430 feet
temperature: 28.97 deg C, 84.14 deg F
absolute pressure: 1008.76 mb, 29.79 inHg
relative (sea-level) pressure: 1232.04 mb, 36.39 inHg
computed altitude: 1655 meters, 5430 feet

provided altitude: 1655 meters, 5430 feet
temperature: 29.35 deg C, 84.84 deg F
absolute pressure: 1008.67 mb, 29.79 inHg
relative (sea-level) pressure: 1231.93 mb, 36.38 inHg
computed altitude: 1655 meters, 5430 feet

provided altitude: 1655 meters, 5430 feet
temperature: 29.74 deg C, 85.53 deg F
absolute pressure: 1008.67 mb, 29.79 inHg
relative (sea-level) pressure: 1231.92 mb, 36.38 inHg
computed altitude: 1655 meters, 5430 feet
```

Como podéis apreciar este sensor nos da la temperatura y la presión atmosférica. Basándose en que la presión disminuye con la altura en este caso nos dice que estamos a 1655 metros de altura.
En realidad estamos a unos 500 metros de altura pero al ser un día de baja presión el programa concluye que estamos a más altura.

Más información

Podéis encontrar más información sobre este sensor y como utilizarlo en los siguientes enlaces:

- GY-68 BMP180 Barometric Pressure Breakout Board Tutorial With Arduino Uno
- Guide for BMP180 Barometric Sensor with Arduino

El sensor DHT11

En esta sección veremos cómo conectar el sensor DHT11 y leer datos desde el arduino.

¿Para qué sirve?

El sensor DHT11 nos permite medir la temperatura y humedad con Arduino.
Una de las ventajas que nos ofrece el DHT11, además de medir la temperatura y la humedad, es que es digital.
A diferencia de otros sensores como el LM35, este sensor utiliza un pin digital para enviarnos la información y por lo tanto, estaremos más protegidos frente al ruido.

¿Cómo funciona?

El DHT11 es un sensor de temperatura y humedad digital de bajo costo. Utiliza un sensor capacitivo de humedad y un termistor para medir el aire circundante, y muestra los datos mediante una señal digital en el pin de datos.

Características técnicas

Respecto a las especificaciones técnicas tenemos las siguientes:

- Voltaje de Operación: 3V - 5V DC
- Rango de medición de temperatura: 0 a 50 °C
- Precisión de medición de temperatura: ±2.0 °C
- Resolución Temperatura: 0.1°C
- Rango de medición de humedad: 20% a 90% RH.
- Precisión de medición de humedad: 5% RH.
- Resolución Humedad: 1% RH
- Tiempo de sensado: 1 seg.
- Interface digital: Single-bus (bidireccional)
- Modelo: DHT11
- Dimensiones: 16*12*5 mm
- Peso: 1 gr.
- Carcasa de plástico celeste

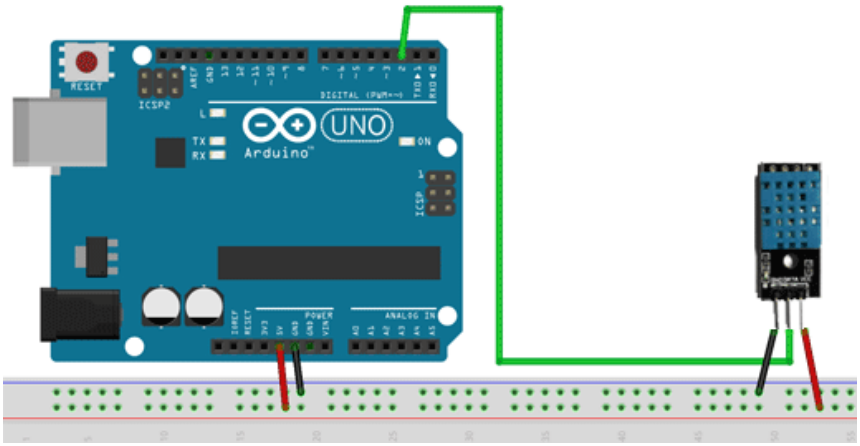
Respecto a los pines de conexión:

- VCC: alimentación
- I/O: transmisión de datos
- NC: no conecta, pin al aire
- GND: conexión a tierra

Esquema de conexionado

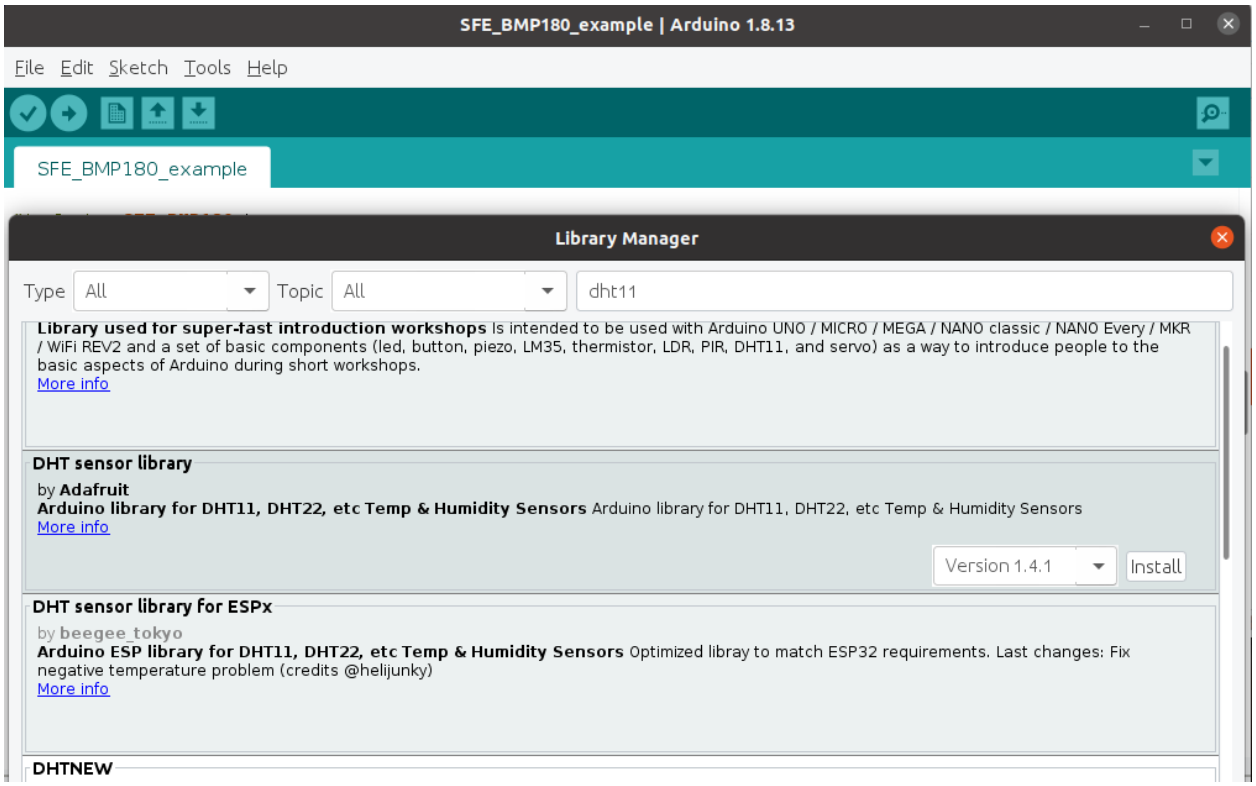
Al contrario que el otro modelo, el DHT11 integrado dentro de un PCB ya viene con la **resistencia pull-up integrada**. Puede resultar muy útil en ocasiones, pero si añadimos un cable de más de 20 metros, deberemos tener en cuenta este factor.

Este modelo de DHT11 dispone de 3 pines, la toma de tierra GND, para los datos DATA y para la alimentación VCC (de 3,5V a 5V). En la siguiente imagen puedes ver el esquema de conexión con Arduino.



Instalando librerías

Ahora tenemos que instalar la librería para el sensor de DHT11 de Adafruit. Vamos al gestor de librerías, `Sketch -> Include Library -> Manage Libraries` , y ahí buscamos por **DHT11**. Nos saldrán varias librerías pero nosotros instalaremos la de Adafruit.
A continuación tenéis una captura de como hacerlo.



Lo primero que hacemos es utilizar un delay para esperar los 5 segundos recomendados. La librería de Adafruit para el DHT11 nos proporciona datos en grados centígrados y grados Fahrenheit. Para obtener los dos datos utilizamos la misma función, `readTemperature()`.

Si no pasamos ningún parámetro nos devuelve la temperatura en grados centígrados. Si pasamos el valor true nos devuelve la temperatura en grados Fahrenheit. La humedad se obtiene llamando a la función `readHumidity()`.

Es conveniente comprobar que la información no está corrupta y que realmente nos está devolviendo un número. Eso lo hacemos con la sentencia isnan(...). Esto nos dará verdadero si no es un número (isnan, Is Not a Number) y falso en caso contrario.

Por último obtenemos el índice de calor con la función `computeHeatIndex(...)`. Nos puede devolver grados centígrados o grados Fahrenheit. Al final del todo mostramos la información en el monitor serie.

Leyendo del sensor DHT11

A continuación vamos a analizar cómo queda el código fuente y la funcionalidad de cada una de las partes.

Lo primero que hacemos es incluir la librería para trabajar con el sensor DHT11. Esta librería nos permitirá utilizar el tipo DHTTYPE que "sabe" cómo trabajar con el sensor DHT11.

```
// Incluimos librería
#include <DHT.h>
```

A continuación definimos el pin 2 como el pin con al que conectaremos el sensor DHT11. Los otros dos cables que llegan al sensor son de los alimentación y tierra.

```
// Definimos el pin digital donde se conecta el sensor
#define DHTPIN 2
```

Creamos un tipo DHTTYPE para usar en nuestro código. El tipo DHT11, importado de la librería DHT.h, es el tipo de datos que puede interactuar con el sensor.

```
// Dependiendo del tipo de sensor
#define DHTTYPE DHT11
```

Definimos una variable de tipo DHT.

```
// Inicializamos el sensor DHT11
DHT dht(DHTPIN, DHTTYPE);
```

El el método de `setup()` , que se ejecuta una sola vez al iniciar el arduino, inicializamos tanto el puerto serie como el sensor DHT11.

```
void setup() {
  // Inicializamos comunicación serie
  Serial.begin(9600);

  // Comenzamos el sensor DHT
  dht.begin();
}
```

Ahora llegamos al bucle principal. Lo primero que hacemos es esperar 5 segundos entre lectura y lectura.

```
void loop() {
  // Esperamos 5 segundos entre medidas
  delay(5000);
```

Una vez transcurrido ese tiempo leemos la humedad relativa, la temperatura en grados centígrados y la temperatura en grados Fahrenheit.

```
  // Leemos la humedad relativa
  float h = dht.readHumidity();
  // Leemos la temperatura en grados centígrados (por defecto)
  float t = dht.readTemperature();
  // Leemos la temperatura en grados Fahrenheit
  float f = dht.readTemperature(true);
```

Tras esto comprobamos que no haya habido ningún tipo de error en la lectura del sensor. El tipo de datos de la lectura es un `float` y si ha habido algún error devuelve un `NaN` (Not a Number).

```
  // Comprobamos si ha habido algún error en la lectura
  if (isnan(h) || isnan(t) || isnan(f)) {
    Serial.println("Error obteniendo los datos del sensor DHT11");
    return;
  }
```

Si no ha habido error calculamos los índices de calor en Fahrenheit y en grados centígrados.

```
  // Calcular el índice de calor en Fahrenheit
  float hif = dht.computeHeatIndex(f, h);
  // Calcular el índice de calor en grados centígrados
  float hic = dht.computeHeatIndex(t, h, false);
```

Por último mostraremos toda esa información por el puerto serie.

```
  Serial.print("Humedad: ");
  Serial.print(h);
```

```
  Serial.print(" %\t");
  Serial.print("Temperatura: ");
  Serial.print(t);
  Serial.print(" *C ");
  Serial.print(f);
  Serial.print(" *F\t");
  Serial.print("Índice de calor: ");
  Serial.print(hic);
  Serial.print(" *C ");
  Serial.print(hif);
  Serial.println(" *F");

}
```

Código fuente completo

Tras haber analizado el código fuente por partes presentamos ahora el código fuente completo en un único bloque para que se pueda analizar mejor.

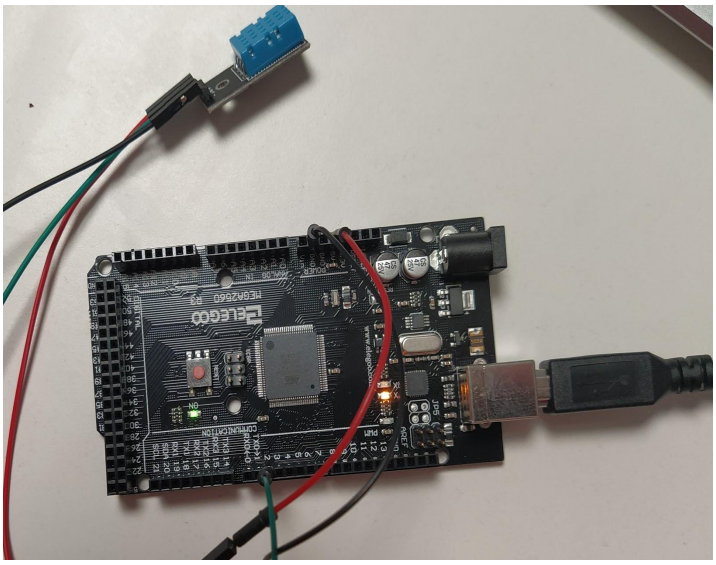
```
// Incluimos librería
#include <DHT.h>

// Definimos el pin digital donde se conecta el sensor
#define DHTPIN 2
// Dependiendo del tipo de sensor
#define DHTTYPE DHT11

// Inicializamos el sensor DHT11
DHT dht(DHTPIN, DHTTYPE);

void setup() {
  // Inicializamos comunicación serie
  Serial.begin(9600);

  // Comenzamos el sensor DHT
  dht.begin();
}

void loop() {
  // Esperamos 5 segundos entre medidas
```

```
  delay(5000);

  // Leemos la humedad relativa
  float h = dht.readHumidity();
  // Leemos la temperatura en grados centígrados (por defecto)
  float t = dht.readTemperature();
  // Leemos la temperatura en grados Fahrenheit
  float f = dht.readTemperature(true);

  // Comprobamos si ha habido algún error en la lectura
  if (isnan(h) || isnan(t) || isnan(f)) {
    Serial.println("Error obteniendo los datos del sensor DHT11");
    return;
  }

  // Calcular el índice de calor en Fahrenheit
  float hif = dht.computeHeatIndex(f, h);
  // Calcular el índice de calor en grados centígrados
  float hic = dht.computeHeatIndex(t, h, false);

  Serial.print("Humedad: ");
  Serial.print(h);
  Serial.print(" %\t");
  Serial.print("Temperatura: ");
  Serial.print(t);
  Serial.print(" *C ");
  Serial.print(f);
  Serial.print(" *F\t");
  Serial.print("Índice de calor: ");
  Serial.print(hic);
  Serial.print(" *C ");
  Serial.print(hif);
  Serial.println(" *F");

}
```

Podéis encontrar el código fuente para este ejemplo [en el repo de github](#) .

Resultado final

La conexión con el arduino Mega AT2560 y el sensor quedaría tal y como se puede apreciar en la siguiente foto.



El resultado que se puede ver en el serial monitor después de ejecutar el código es el siguiente.

```
Humedad: 59.00 %        Temperatura: 24.90 *C 76.82 *F  Índice de calor: 24.99 *C 76.98 *F
Humedad: 61.00 %        Temperatura: 24.20 *C 75.56 *F  Índice de calor: 24.27 *C 75.68 *F
Humedad: 61.00 %        Temperatura: 24.20 *C 75.56 *F  Índice de calor: 24.27 *C 75.68 *F
Humedad: 63.00 %        Temperatura: 24.20 *C 75.56 *F  Índice de calor: 24.32 *C 75.78 *F
Humedad: 65.00 %        Temperatura: 24.20 *C 75.56 *F  Índice de calor: 24.37 *C 75.87 *F
Humedad: 64.00 %        Temperatura: 24.20 *C 75.56 *F  Índice de calor: 24.35 *C 75.82 *F
Humedad: 64.00 %        Temperatura: 24.10 *C 75.38 *F  Índice de calor: 24.24 *C 75.63 *F
Humedad: 64.00 %        Temperatura: 24.10 *C 75.38 *F  Índice de calor: 24.24 *C 75.63 *F
Humedad: 63.00 %        Temperatura: 24.00 *C 75.20 *F  Índice de calor: 24.10 *C 75.38 *F
Humedad: 63.00 %        Temperatura: 24.00 *C 75.20 *F  Índice de calor: 24.10 *C 75.38 *F
```

Más información

Podéis encontrar más información sobre este sensor y como utilizarlo en los siguientes enlaces:

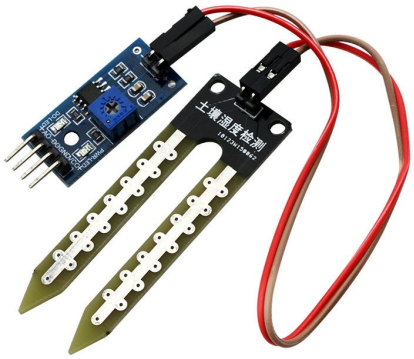
- Cómo utilizar el sensor DHT11 para medir la temperatura y humedad con Arduino
- DHT11: todo sobre el sensor para medir temperatura y humedad

El sensor DollaTek DZ0325

En esta sección veremos cómo conectar el sensor DollaTek DZ0325 y leer datos desde el arduino.

¿Para qué sirve?

El sensor DollaTek DZ0325 es un sensor que detecta la humedad en la tierra.



Este sensor puede darnos una señal digital cuando la tierra necesita agua. La salida puede ajustarse con un potenciómetro para ajustarlo a las necesidades de la planta.

También puede darnos una señal analógica de la humedad actual de la tierra.

¿Cómo funciona?

No hemos encontrado mucha información sobre cómo funciona este sensor pero por lo que nos hemos podido documentar pensamos que se puede tratar de un sensor capacitivo de humedad relativa.

Se puede hacer un simple sensor capacitivo de humedad relativa a partir de un condensador lleno de aire a medida que la humedad en la atmósfera cambia su permitividad. Pero para aplicaciones prácticas, el aire como dieléctrico no es factible.

Por lo tanto, el espacio entre las placas del condensador generalmente se llena con un material dieléctrico apropiado (aislador), cuya constante dieléctrica varía cuando está sujeto a cambios de humedad.

El método común para construir un sensor de HR capacitivo es usar una película de polímero higroscópico como dieléctrico y depositar dos capas de electrodos a cada lado.

Características técnicas

Respecto a las especificaciones técnicas tenemos las siguientes:

- Sensibilidad ajustable mediante el potenciómetro digital (en azul en la imagen)
- Rango de voltaje de operación: 3.3V-5V
- Consumo: 1W-6W
- Rango de temperatura de funcionamiento: -40°C ~ 125°C
- Salida dual: Salida digital y salida analógica (más exacta).
- Indicador led (rojo) y indicador de salida digital (verde)

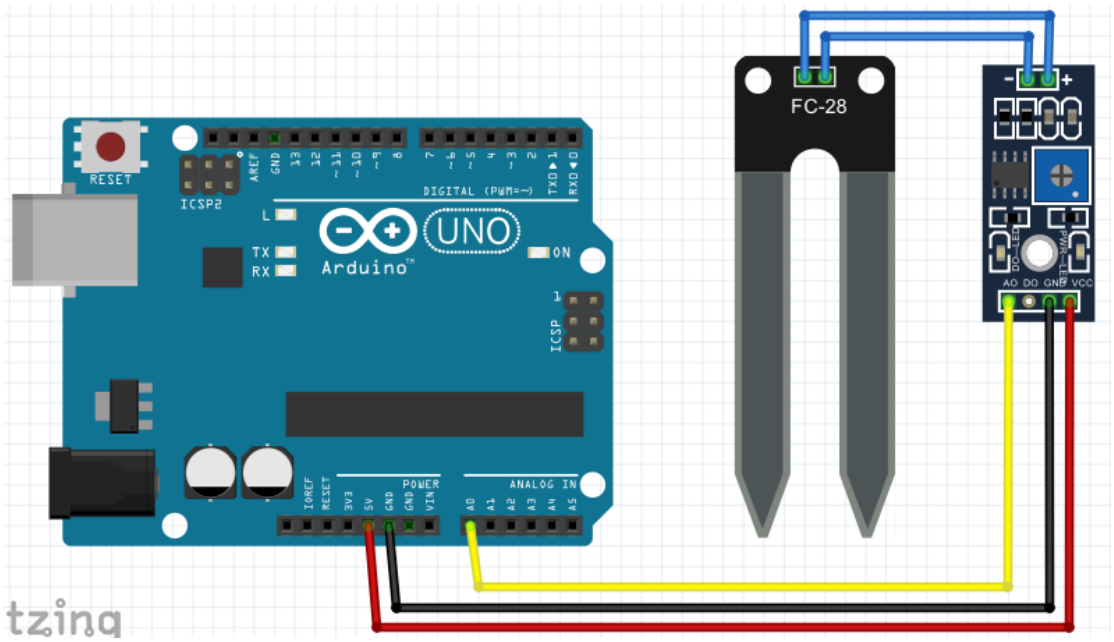
Respecto a los pines de conexión:

- VCC: alimentación externa de 3.3V-5V
- GND: toma de tierra externa.
- DO: Salida digital (0 and 1)
- AO: Salida analógica

Esquema de conexionado

Este sensor nos va a permitir conocer la humedad de la tierra. Tiene una salida analógica que nos indica el grado de humedad y una digital que, parametrizada con el potenciómetro, nos indicará 0 o 1, es decir, si tenemos que regar o no.

En la siguiente imagen podéis ver el diagrama de conexionado con el arduino UNO y el sensor DZ0325



Instalando librerías

Para este sensor no hace falta instalar ningún tipo de librería. Simplemente tenemos que leer del pin analógico e interpretar los datos leídos.

Leyendo del sensor DollaTek DZ0325

A continuación vamos a analizar cómo queda el código fuente y la funcionalidad de cada una de las partes.

Empezamos definiendo una constante para el pin analógico de salida del sensor Higrometer. En el ejemplo anterior hemos usado la directiva `#define` para este propósito. Ambas opciones son válidas. Si utilizamos la directiva el preprocesador realizar un intercambio y si utilizamos la constante se definirá en tiempo de compilación.

También definimos una variable global donde almacenaremos la lectura del sensor.

```
//Constants
const int hygrometer = A0;  //Hygrometer sensor analog pin output at pin A0
of Arduino
//Variables
int value;
```

En el método de setup solo se inicializa la comunicación por el puerto serie.

```
void setup(){
  Serial.begin(9600);
}
```

En el bucle principal leemos del sensor con la función analogRead(...), forzamos que la lectura esté en el rango entre 400 - 1023 (`contrain(value,400,2013);`) y hacemos un mapping entre el valor obtenido y los valores comprendidos entre 100 y 0, una forma de poner en porcentaje el valor que nos da el sensor.

Después mostramos los resultados por el puerto serie y esperamos 2 segundos entre lectura y lectura.

```
void loop(){
  // When the plant is watered well the sensor will read a value 380~400, I
```

```
will keep the 400
  // value but if you want you can change it below.

  value = analogRead(hygrometer);   //Read analog value
  value = constrain(value,400,1023);  //Keep the ranges!
  value = map(value,400,1023,100,0);  //Map value : 400 will be 100 and
1023 will be 0
  Serial.print("Soil humidity: ");
  Serial.print(value);
  Serial.println("%");
  delay(2000); //Read every 2 sec.
}
```

Código fuente completo

Aunque en este caso no hay tantos bloques pequeñitos de código, con el fin de la claridad, mostramos ahora el código completo que lee de este sensor.

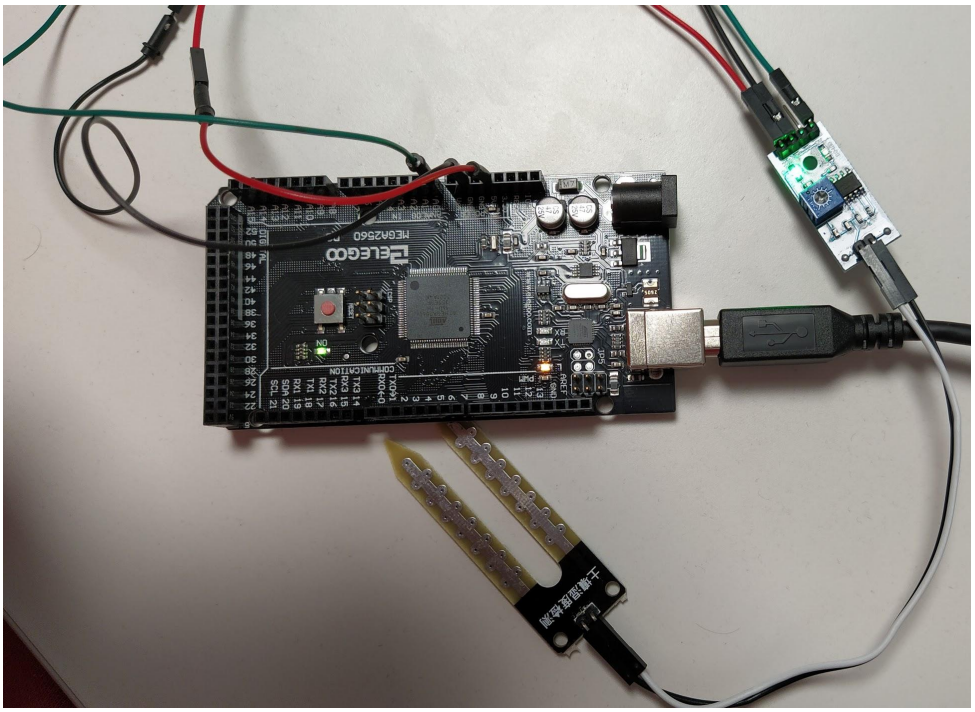
```
//Constants
const int hygrometer = A0;  //Hygrometer sensor analog pin output at pin A0
of Arduino
//Variables
int value;

void setup(){

  Serial.begin(9600);
}

void loop(){

  // When the plant is watered well the sensor will read a value 380~400, I
will keep the 400
  // value but if you want you can change it below.

  value = analogRead(hygrometer);   //Read analog value
  value = constrain(value,400,1023);  //Keep the ranges!
  value = map(value,400,1023,100,0);  //Map value : 400 will be 100 and
1023 will be 0
  Serial.print("Soil humidity: ");
  Serial.print(value);
```

```
  Serial.println("%");
  delay(2000); //Read every 2 sec.
}
```

Podéis encontrar el código fuente para leer de este sensor en [esta carpeta de github](#).

Resultado final

El conexionado real con el arduino Mega AT2560 quedaría tal y como podemos ver en la siguiente imagen.



El resultado que se puede ver en el serial monitor después de ejecutar el código es el siguiente.

```
Soil humidity: 1%
Soil humidity: 1%
Soil humidity: 2%
Soil humidity: 100%
Soil humidity: 2%
Soil humidity: 2%
Soil humidity: 2%
Soil humidity: 10%
Soil humidity: 7%
Soil humidity: 7%
Soil humidity: 9%
Soil humidity: 10%
Soil humidity: 3%
Soil humidity: 2%
```

Como no tenemos suelo sobre el que probar vamos a probar dejando los extremos de los sensores al aire, conectándolos entre sí por un dedo y conectándolos entre sí con un cable.

- Los extremos al aire: los valores en los que aparece 1% son cuando no los dos extremos del sensor están al aire.
- Los extremos unidos con un dedo: cuando unimos los extremos con uno o varios dedos la humedad pasa a un 7% a 10%.
- Los extremos unidos con un cable: si unimos los extremos con un cable la humedad pasa un 100%.

Más información

Podéis encontrar más información sobre este sensor y como utilizarlo en los siguientes enlaces:

- How to use the soil hygrometer module - Arduino tutorial
- Arduino automatic watering system

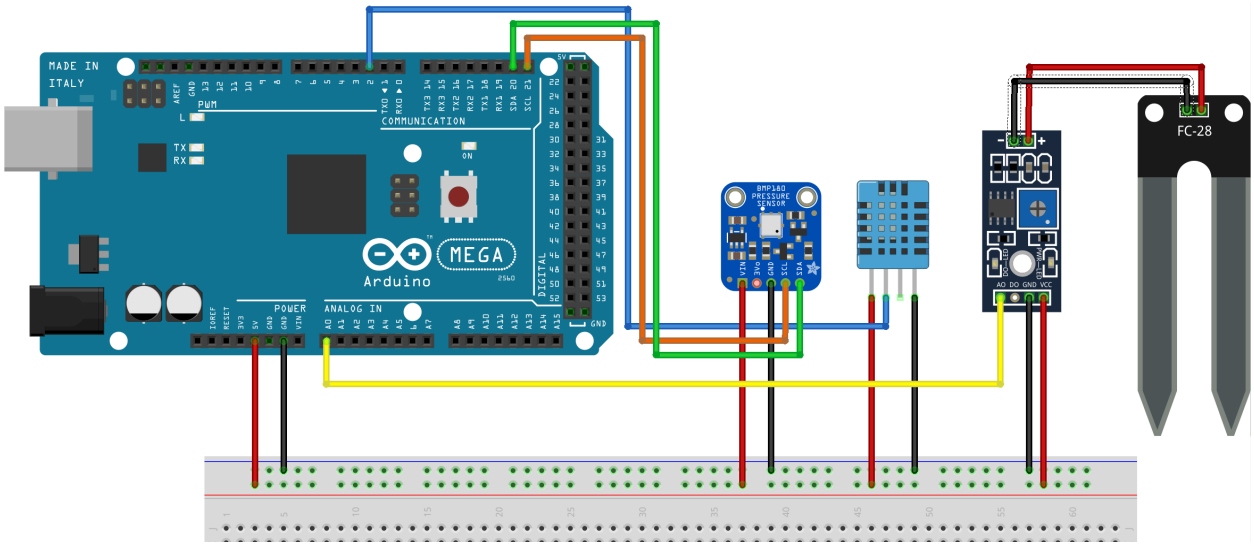
Leyendo de todos los sensores

Ya hemos conseguido leer de cada uno de los sensores. Ahora tenemos que conseguir leer de todos los sensores a la vez.

Para ello lo primero que tenemos que hacer es definir el diagrama de conexionado y después adaptar el código para que lea de todos los sensores.

Diagrama de conexionado

El diagrama de conexionado para poder leer de todos los sensores a la vez queda de la siguiente manera.

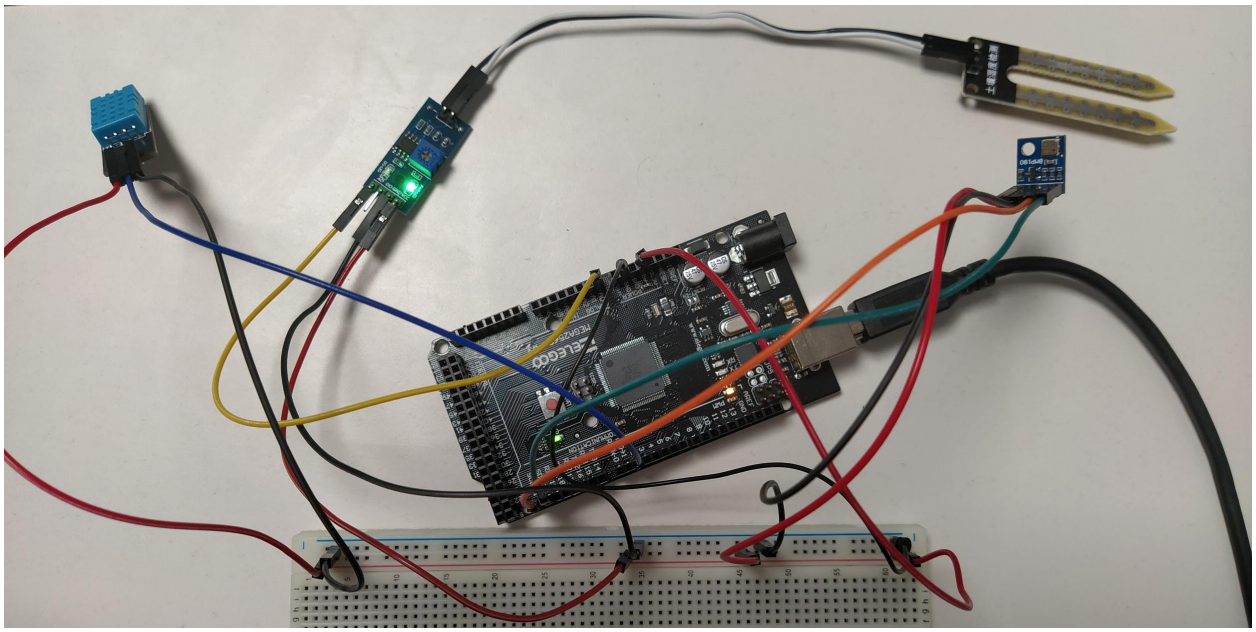


Tal y como se puede apreciar el conexionado es básicamente el mismo que teníamos cuando leíamos de cada uno de los sensores.

Hemos intentando mantener el código de colores de los cables a los ejemplos anteriores. De igual manera intentaremos mantener el mismo código de colores para cuando hagamos el conexionado físico. De esta manera creemos que será más sencillo seguir las indicaciones de esta guía.

Montaje en el mundo real

Vale, ya tenemos nuestro diagrama teórico montado. Ahora vamos a montarlo en el mundo real con el arduino mega, sensores, cableado y el breadboard. El resultado es la imagen que podéis ver a continuación.



En este montaje para la vida real hemos seguido el mismo código de colores de los cables que el que aparece en el diagrama, para que sea más fácil el seguimiento del proyecto.

Código fuente

Ya tenemos el conexionado real hecho. El siguiente paso es poner el código fuente que lee de todos los sensores.

Empezamos importando las librerías que nos van a hacer falta para leer de los sensores que tenemos.

```
//Include libraries
// BMP180
#include <SFE_BMP180.h>
#include <Wire.h>
// DHT11
#include <DHT.h>
```

Creamos un objeto SFE_BMP180, definimos la altura (en este caso ya ponemos 300 metros), el pin del sensor DHT11, el objeto DHT11, el pin analógico del Higrometro y por último inicializamos el objeto dht.

```
// You will need to create an SFE_BMP180 object, here called "pressure":
SFE_BMP180 pressure;

// Constatst
#define ALTITUDE 300.0 // Altitude of SparkFun's HQ in Boulder, CO. in
meters
// Definimos el pin digital donde se conecta el sensor
#define DHTPIN 2
#define DHTTYPE DHT11
// Hygrometer
//Constants
const int hygrometer = A0;  //Hygrometer sensor analog pin output at pin A0
of Arduino

// Inicializamos el sensor DHT11
DHT dht(DHTPIN, DHTTYPE);
```

Sacamos a una función la inicialización del sensor BMP180 que es el que es un poco más complejo.

```
void start_BMP180() {
  // Initialize the sensor (it is important to get calibration values
stored on the device).
  if (pressure.begin()) {
    Serial.println("BMP180 init success");
  } else {
    // Oops, something went wrong, this is usually a connection problem,
    // see the comments at the top of this sketch for the proper
connections.
    Serial.println("BMP180 init fail\n\n");
    while(1); // Pause forever.
  }
}
```

Sacamos el método que lee de sensor DHT11. Haremos lo mismo con otros sensores.
A la vez que leemos mostramos por el puerto serie los datos de nuestra lectura.

```
void read_DHT11() {
    // Esperamos 5 segundos entre medidas
```

```
  delay(5000);

  // Leemos la humedad relativa
  float h = dht.readHumidity();
  // Leemos la temperatura en grados centígrados (por defecto)
  float t = dht.readTemperature();

  // Comprobamos si ha habido algún error en la lectura
  if (isnan(h) || isnan(t)) {
    Serial.println("Error obteniendo los datos del sensor DHT11");
    return;
  }

  Serial.print("Humedad: ");
  Serial.print(h);
  Serial.println("%");
  Serial.print("Temperatura: ");
  Serial.print(t);
  Serial.println("ºC");
}
```

Al igual que en el caso anterior, leemos del higrómetro (sensor DollaTek) y mostramos por el puerto serie lo que hemos leído.

```
void read_hygrometer() {
  int value;
  value = analogRead(hygrometer);   //Read analog value
  value = constrain(value,400,1023);  //Keep the ranges!
  value = map(value,400,1023,100,0);  //Map value : 400 will be 100 and
1023 will be 0
  Serial.print("Soil humidity: ");
  Serial.print(value);
  Serial.println("%");
}
```

Por último, leemos del sensor BMP180 y mostramos por el puerto serie lo que hemos leido.

```
void read_BMP180() {
  char status;
```

```
  double T,P,p0,a;

  // Loop here getting pressure readings every 10 seconds.

  // If you want sea-level-compensated pressure, as used in weather
reports,
  // you will need to know the altitude at which your measurements are
taken.
  // We're using a constant called ALTITUDE in this sketch:
  Serial.print("provided altitude: ");
  Serial.print(ALTITUDE,0);
  Serial.println(" meters,");

  // If you want to measure altitude, and not pressure, you will instead
need
  // to provide a known baseline pressure. This is shown at the end of the
sketch.
  // You must first get a temperature measurement to perform a pressure
reading.
  status = pressure.startTemperature();
  if (status != 0) {
    // Wait for the measurement to complete:
    delay(status);

  // Retrieve the completed temperature measurement: Note that the
measurement is stored in the variable T.
    // Function returns 1 if successful, 0 if failure.

    status = pressure.getTemperature(T);
    if (status != 0) {
      // Print out the measurement:
      Serial.print("temperature: ");
      Serial.print(T,2);
      Serial.println(" deg C.");

      // Start a pressure measurement:
      // The parameter is the oversampling setting, from 0 to 3 (highest
res, longest wait).
      // If request is successful, the number of ms to wait is returned.
      // If request is unsuccessful, 0 is returned.
```

```
      status = pressure.startPressure(3);
      if (status != 0)
      {
        // Wait for the measurement to complete:
        delay(status);

        // Retrieve the completed pressure measurement:
        // Note that the measurement is stored in the variable P.
        // Note also that the function requires the previous temperature
measurement (T).
        // (If temperature is stable, you can do one temperature
measurement for a number of pressure measurements.)
        // Function returns 1 if successful, 0 if failure.
        status = pressure.getPressure(P,T);
        if (status != 0)
        {
          // Print out the measurement:
          Serial.print("absolute pressure: ");
          Serial.print(P,2);
          Serial.println(" mb");

          // The pressure sensor returns abolute pressure, which varies
with altitude.
          // To remove the effects of altitude, use the sealevel function
and your current altitude.
          // This number is commonly used in weather reports.
          // Parameters: P = absolute pressure in mb, ALTITUDE = current
altitude in m.
          // Result: p0 = sea-level compensated pressure in mb
          p0 = pressure.sealevel(P,ALTITUDE);
          Serial.print("relative (sea-level) pressure: ");
          Serial.print(p0,2);
          Serial.println(" mb.");

        } else {
          Serial.println("error retrieving pressure measurement\n");
        }
      } else {
        Serial.println("error starting pressure measurement\n");
      }
    } else {
```

```
      Serial.println("error retrieving temperature measurement\n");
    }
  } else {
    Serial.println("error starting temperature measurement\n");
  }
}
```

En el método de setup() inicializamos el puerto serie a 9600 baudios, inicializamos el sensor BMP180 e inicializamos el sensor DHT11.

```
void setup()
{
  Serial.begin(9600);
  Serial.println("Starting Serial...");
  start_BMP180();
  // Comenzamos el sensor DHT
  dht.begin();
}
```

El bucle principal queda de esta manera algo muy sencillo de entender. Leemos de cada uno de los sensores y esperamos 5 segundos entre lectura y lectura.

```
void loop()
{

  Serial.println("\n\nReading from DHT11 ...");
  read_DHT11();

  Serial.println("\nReading from hygrometer ...");
  read_hygrometer();

  Serial.println("\nReading from BMP180 ...");
  read_BMP180();

  delay(5000);  // Pause for 5 seconds.
}
```

A continuación mostramos todo el código fuente de este parte de la guía, el código fuente que leerá de todos los sensores en cada ciclo de lectura.

```
//Include libraries
// BMP180
#include <SFE_BMP180.h>
#include <Wire.h>
// DHT11
#include <DHT.h>

// You will need to create an SFE_BMP180 object, here called "pressure":
SFE_BMP180 pressure;

// Constatst
#define ALTITUDE 300.0 // Altitude of SparkFun's HQ in Boulder, CO. in
meters
// Definimos el pin digital donde se conecta el sensor
#define DHTPIN 2
#define DHTTYPE DHT11
// Hygrometer
//Constants
const int hygrometer = A0;  //Hygrometer sensor analog pin output at pin A0
of Arduino

// Inicializamos el sensor DHT11
DHT dht(DHTPIN, DHTTYPE);

void start_BMP180() {
  // Initialize the sensor (it is important to get calibration values
stored on the device).
  if (pressure.begin()) {
    Serial.println("BMP180 init success");
  } else {
    // Oops, something went wrong, this is usually a connection problem,
    // see the comments at the top of this sketch for the proper
connections.
    Serial.println("BMP180 init fail\n\n");
    while(1); // Pause forever.
}
```

```
void read_DHT11() {
    // Esperamos 5 segundos entre medidas
  delay(5000);

  // Leemos la humedad relativa
  float h = dht.readHumidity();
  // Leemos la temperatura en grados centígrados (por defecto)
  float t = dht.readTemperature();

  // Comprobamos si ha habido algún error en la lectura
  if (isnan(h) || isnan(t)) {
    Serial.println("Error obteniendo los datos del sensor DHT11");
    return;
  }

  Serial.print("Humedad: ");
  Serial.print(h);
  Serial.println("%");
  Serial.print(//Include libraries
}

void read_hygrometer() {
  int value;
  value = analogRead(hygrometer);   //Read analog value
  value = constrain(value,400,1023);  //Keep the ranges!
  value = map(value,400,1023,100,0);  //Map value : 400 will be 100 and
1023 will be 0
  Serial.print("Soil humidity: ");
  Serial.print(value);
  Serial.println("%");
}

void read_BMP180() {
  char status;
  double T,P,p0,a;

  // Loop here getting pressure readings every 10 seconds.

  // If you want sea-level-compensated pressure, as used in weather
reports,
```

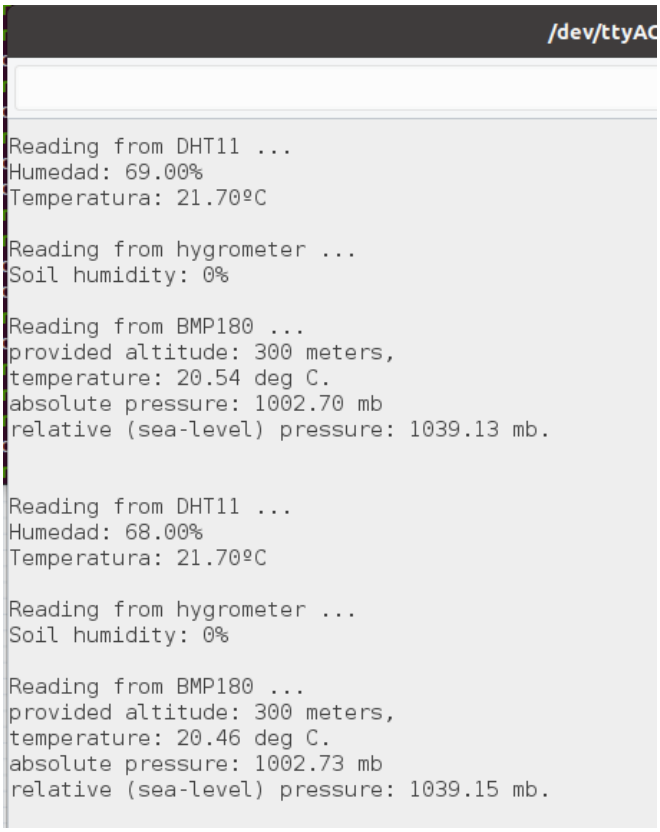
```
  // you will need to know the altitude at which your measurements are
taken.
  // We're using a constant called ALTITUDE in this sketch:
  Serial.print("provided altitude: ");
  Serial.print(ALTITUDE,0);
  Serial.println(" meters,");

  // If you want to measure altitude, and not pressure, you will instead
need
  // to provide a known baseline pressure. This is shown at the end of the
sketch.
  // You must first get a temperature measurement to perform a pressure
reading.
  status = pressure.startTemperature();
  if (status != 0) {
    // Wait for the measurement to complete:
    delay(status);

    // Retrieve the completed temperature measurement: Note that the
measurement is stored in the variable T.
    // Function returns 1 if successful, 0 if failure.
    status = pressure.getTemperature(T);
    if (status != 0) {
      // Print out the measurement:
      Serial.print("temperature: ");
      Serial.print(T,2);
      Serial.println(" deg C.");

      // Start a pressure measurement:
      // The parameter is the oversampling setting, from 0 to 3 (highest
res, longest wait).
      // If request is successful, the number of ms to wait is returned.
      // If request is unsuccessful, 0 is returned.
      status = pressure.startPressure(3);
      if (status != 0)
      {
        // Wait for the measurement to complete:
        delay(status);

        // Retrieve the completed pressure measurement:
        // Note that the measurement is stored in the variable P.
```

```
        // Note also that the function requires the previous temperature
measurement (T).
        // (If temperature is stable, you can do one temperature
measurement for a number of pressure measurements.)
        // Function returns 1 if successful, 0 if failure.
        status = pressure.getPressure(P,T);
        if (status != 0)
        {
          // Print out the measurement:
          Serial.print("absolute pressure: ");
          Serial.print(P,2);
          Serial.println(" mb");

          // The pressure sensor returns abolute pressure, which varies
with altitude.
          // To remove the effects of altitude, use the sealevel function
and your current altitude.
          // This number is commonly used in weather reports.
          // Parameters: P = absolute pressure in mb, ALTITUDE = current
altitude in m.
          // Result: p0 = sea-level compensated pressure in mb
          p0 = pressure.sealevel(P,ALTITUDE);
          Serial.print("relative (sea-level) pressure: ");
          Serial.print(p0,2);
          Serial.println(" mb.");

        } else {
          Serial.println("error retrieving pressure measurement\n");
        }
      } else {
        Serial.println("error starting pressure measurement\n");
      }
    } else {
      Serial.println("error retrieving temperature measurement\n");
    }
  } else {
    Serial.println("error starting temperature measurement\n");
  }
}

void setup()
```

```
{
  Serial.begin(9600);
  Serial.println("Starting Serial...");
  start_BMP180();
  // Comenzamos el sensor DHT
  dht.begin();
}

void loop()
{

  Serial.println("\n\nReading from DHT11 ...");
  read_DHT11();

  Serial.println("\nReading from hygrometer ...");
  read_hygrometer();

  Serial.println("\nReading from BMP180 ...");
  read_BMP180();

  delay(5000);  // Pause for 5 seconds.
}
```

El código fuente que consigue esto lo podéis encontrar aquí. Es muy importante que usemos los mismos pines de conexión para cada sensor si no, evidentemente, el programa no va a funcionar.

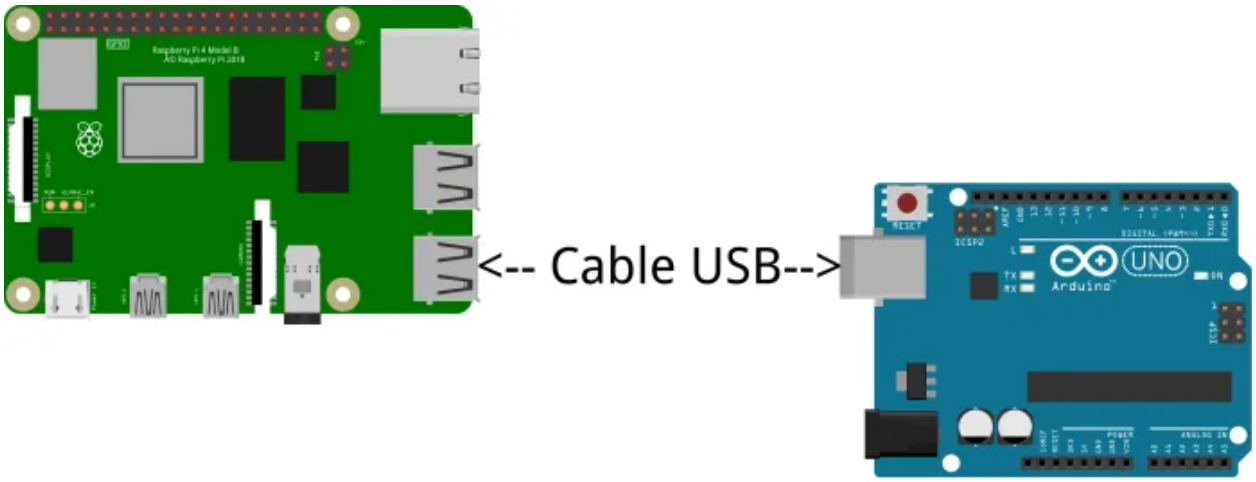
Como sabéis, por el momento estamos mandado los datos que leemos de los sensores al puerto serial para que se vea en el serial monitor. En la siguiente imagen podéis ver una captura de la salida que se genera.



Comunicación entre arduino y Raspberry pi

La comunicación UART o comunicación serial se establece con 2 señales digitales, por cada dispositivo: «TX», que es la que transmite datos y «RX», que es la que recibe datos. Sin embargo es posible emular un puerto serie a través de una clase del protocolo USB, esto lo podemos ver implementado en la plataforma Arduino cuando desplegamos el monitor serie y visualizamos mensajes que llegan desde nuestro Arduino. Lo que sucede es que la computadora recibe la información del puerto serie a través del USB, es decir a través de un puerto serie emulado.

Para este proyecto aprovecharemos que tanto la tarjeta Arduino como las tarjetas Raspberry Pi tienen la capacidad de emular puertos serie a través del USB, de tal manera que el diagrama de conexiones se reducirá a conectar la tarjeta Arduino a la Raspberry Pi con el cable USB.



Serialización de datos

Para comunicar los datos entre el arduino y la raspberry pi tenemos que **serializar** los datos. Según la wikipedia la serialización consiste en un proceso de codificación de un objeto con el fin de transmitirlo a través de una conexión en red.

Nosotros utilizaremos json para serializar los datos, porque:

- Es un formato sencillo y fácil de entender
- ocupa poco espacio tanto en memoria como en disco
- y funciona muy bien de forma nativa con python y con la web.

Para serializar en la parte de arduino no utilizaremos ninguna librería. Lo haremos nosotros mismos ya que no es complicado.
Para leer en la parte de python utilizaremos el soporte nativo para json que ya incluye python.

Leyendo de los sensores y serializando los datos

Llegados a este momento tenemos que leer de todos los sensores (esto ya lo hemos hecho justo en el apartado anterior) y serializar los datos para enviarlos a la raspberry pi por medio de un cable usb.

Para ello tenemos el siguiente código donde la única particularidad con respecto al anterior que leía de todos los sensores, es que ahora en lugar de mostrar los datos por el puerto serie como queremos vamos a convertir esas lecturas en una estructura json como la que vamos a definir a continuación.

La estructura json está formado por un elemento de primer nivel que se denomina "lectura". La lectura es una lista de elementos donde cada elemento representa las lecturas de un sensor particular.

El formato quedaría como el siguiente:

```
{
   "lectura":[
      { <lectura sensor 1> },
      { <lectura sensor 2> },
      ... ,
      { <lectura sensor n> },
   ]
}
```

A tener en cuenta que los elementos que empiezan y terminan por paréntesis angulares (< >), representan elementos que están por definir.

Pasamos ahora a definir cómo se forma cada lectura del sensor. La lectura de un sensor tendrá el siguiente formato json:

```
      {
         "Sensor":"nombre sensor",
         "valores":{
            "magnitud_1":numero,
            "magnitud_2":numero,
            ... ,
            "magnitud_n":numero
         }
      }
```

Vemos que tenemos un "sensor" que tiene como valor su nombre de sensor y una estructura clave valor donde aparecerán la magnitud y el valor de la magnitud, ya que los sensores pueden leer más de una magnitud física.

Como ejemplo real de cómo quedaría una lectura de este tipo tenemos el siguiente bloque json:

```
      {
         "sensor":"BMP180",
```

```
         "valores":{
            "temperatura":24,
            "presion":1000
         }
      }
```

Por último mostramos un ejemplo completo de una lectura de todos los sensores que hemos definido y quedaría como la siguiente:

Json datos

```
{
   "lectura":[
      {
         "sensor":"DHT11",
         "valores":{
            "humedad":61.0,
            "temperatura":26.0
         }
      },
      {
         "sensor":"DZ0325",
         "valores":{
            "humedad_terreno":1
         }
      },
      {
         "sensor":"BMP180",
         "valores":{
            "temperatura":24,
            "presion":1000
         }
      }
   ]
}
```

Aquí podemos ver la lectura de los 3 sensores y de los distintos valores de cada una de las magnitudes leídas.
Destacar que los espacios y los retornos de carro se han puesto así para que sea visualmente más sencillo de identificar. Todo el json puede estar en una sola línea (es lo que haremos cuando escribamos por el puerto serie) y quedaría de la siguiente manera.

```
{"lectura":[{"sensor":"DHT11","valores":{"humedad":61.0,"temperatura":26.0}
},{"sensor":"DZ0325","valores":{"humedad_terreno":1}},{"sensor":"BMP180","v
alores":{"temperatura":24,"presion":1000}}]}
```

Y este bloque json es perfectamente válido. Es más, es equivalente al anterior.

Código arduino para la serialización

Pues bien, nuestro código de arduino tiene que escribir en el puerto serie esta estructura de datos.
Vamos a mostrar como sería el código C de arduino que serializaría nuestros datos en formato json.

En este caso vamos a centrarnos en el código que serializa nuestros datos a formato json y obviaremos el resto del código fuente que ya ha sido analizado en los apartados anteriores.

La serialización es sencilla de entender si tenemos en cuenta que lo hacemos por bloques, es decir, cada método que lee de un sensor se encarga de serializar su parte y el resultado es un bloque json válido.

Para poder escribir unas comillas dobles en una cadena de caracteres, que son los caracteres que indican el inicio y el fin de la cadena de caracteres, utilizaremos la barra invertida o carácter de escape. Por ejemplo `print("Hola \"pepe\"");` mostrará por el puerto serie `Hola "pepe"`.

Serialización de los datos del sensor DHT11

Empezamos analizando la lectura del sensor DHT11.

```
void read_DHT11() {

  // Leemos la humedad relativa
  float h = dht.readHumidity();
  // Leemos la temperatura en grados centígrados (por defecto)
  float t = dht.readTemperature();

  // Comprobamos si ha habido algún error en la lectura
```

```
  if (isnan(h) || isnan(t)) {
    Serial.println("Error obteniendo los datos del sensor DHT11");
    return;
  }
```

A continuación vamos a crear la serialización del bloque json de la lectura del sensor DHT11. Para ello empezamos la cadena con abriendo unas llaves, utilizamos la barra invertida para escapar el carácter comillas dobles y ponemos la palabra sensor.
Pasando un poco por encima de lo que ya sabemos, ahora escribimos el nombre del sensor que es DHT11. Tras esto la coma que separa un elemento del siguiente, y el siguiente elemento que es valores.
Los valores vuelven a ser una estructura clave valor donde la clave es la magnitud (humedad y temperatura) y los valores serán los valores leídos por el sensor DHT11.

El código resultante es el siguiente:

```
  Serial.print("{\"sensor\":\"DHT11\",\"valores\":{");
  Serial.print("\"humedad\":");
  Serial.print(h);
  Serial.print(",\"temperatura\":");
  Serial.print(t);
  Serial.print("}}");
}
```

Este código C generaría el siguiente bloque json:

```
{"sensor":"DHT11","valores":{"humedad":61.0,"temperatura":26.0}}
```

Serialización de los datos del sensor DollaTek DZ0325

A continuación leemos del higrómetro, el sensor DollaTek DZ0325, que solo lee una magnitud por lo que el json es un poco más corto y sencillo.

```
void read_hygrometer() {
  int value;
  value = analogRead(hygrometer);   //Read analog value
  value = constrain(value,400,1023);  //Keep the ranges!
```

```
  value = map(value,400,1023,100,0);  //Map value : 400 will be 100 and
1023 will be 0

  Serial.print("{\"sensor\":\"DZ0325\",\"valores\":{");
  Serial.print("\"humedad_terreno\":");
  Serial.print(value);
  Serial.print("}}");
}
```

Este código C generaría el siguiente bloque json:

```
{"sensor":"DZ0325","valores":{"humedad_terreno":1}}
```

Serialización de los datos del sensor GY-68 BMP180

Por último leemos del sensor BMP180 que tiene mucho código y obviaremos el código que lee para centrarnos solo en el que serializa los datos.

```
void read_BMP180() {
  char status;
  double T,P,p0,a;

  ...

        if (status != 0)
        {
          Serial.print("{\"sensor\":\"BMP180\",\"valores\":{");
          Serial.print("\"temperatura\":");
          Serial.print((int)T);
          Serial.print(",\"presion\":");
          Serial.print((int)P);
          Serial.print("}}");

        }
   ...
}
```

Este código C generaría el siguiente bloque json:

```
{"sensor":"BMP180","valores":{"temperatura":24,"presion":1000}}
```

Para terminar sólo queda empezar el bloque de json con las lecturas y terminarlo cerrando las llaves. En ese caso hemos resaltado las dos lineas de código que hacen lo comentado (abrir y cerrar el json).

```
void loop() {
  Serial.print("{\"lectura\":[");
  read_DHT11();

  Serial.print(",");
  read_hygrometer();

  Serial.print(",");
  read_BMP180();
  Serial.println("]}");

  delay(DELAY_TIME);  // Pause for DELAY_TIME milliseconds.
}
```

El bloque completo resultante es el siguiente, donde hemos resaltado también la parte de apertura del bloque json y el elemento lectura y el cierre. En medio quedarían los bloques de los sensores que ya hemos explicado con anterioridad.

```
{"lectura":[{"sensor":"DHT11","valores":{"humedad":61.0,"temperatura":26.0}
},{"sensor":"DZ0325","valores":{"humedad_terreno":1}},{"sensor":"BMP180","v
alores":{"temperatura":24,"presion":1000}}]}
```

Código completo de la serialización desde arduino

Ahora falta por mostrar el código completo que realizar tanto la lectura de datos como la serialización en json de los datos.

```
//Include libraries
// BMP180
#include <SFE_BMP180.h>
#include <Wire.h>
// DHT11
#include <DHT.h>

// You will need to create an SFE_BMP180 object, here called "pressure":
SFE_BMP180 pressure;
```

```
// Constatst
#define DELAY_TIME 300000   //  5 minutes
#define ALTITUDE 300.0 // Altitude of SparkFun's HQ in Boulder, CO. in
meters
// Definimos el pin digital donde se conecta el sensor
#define DHTPIN 2
#define DHTTYPE DHT11
// Hygrometer
//Constants
const int hygrometer = A0;  //Hygrometer sensor analog pin output at pin A0
of Arduino


// Inicializamos el sensor DHT11
DHT dht(DHTPIN, DHTTYPE);


void start_BMP180() {
  // Initialize the sensor (it is important to get calibration values
stored on the device).
  if (pressure.begin()) {
    //Serial.println("BMP180 init success");
  } else {
    // Oops, something went wrong, this is usually a connection problem,
    // see the comments at the top of this sketch for the proper
connections.
    Serial.println("BMP180 init fail\n\n");
    while(1); // Pause forever.
  }

}

void read_DHT11() {

  // Leemos la humedad relativa
  float h = dht.readHumidity();
  // Leemos la temperatura en grados centígrados (por defecto)
  float t = dht.readTemperature();

  // Comprobamos si ha habido algún error en la lectura
```

```
  if (isnan(h) || isnan(t)) {
    Serial.println("Error obteniendo los datos del sensor DHT11");
    return;
  }

  Serial.print("{\"sensor\":\"DHT11\",\"valores\":{");
  Serial.print("\"humedad\":");
  Serial.print(h);
  Serial.print(",\"temperatura\":");
  Serial.print(t);
  Serial.print("}}");
}

void read_hygrometer() {
  int value;
  value = analogRead(hygrometer);   //Read analog value
  value = constrain(value,400,1023);  //Keep the ranges!
  value = map(value,400,1023,100,0);  //Map value : 400 will be 100 and
1023 will be 0

  Serial.print("{\"sensor\":\"DZ0325\",\"valores\":{");
  Serial.print("\"humedad_terreno\":");
  Serial.print(value);
  Serial.print("}}");
}

void read_BMP180() {
  char status;
  double T,P,p0,a;

  // You must first get a temperature measurement to perform a pressure
reading.
  status = pressure.startTemperature();
  if (status != 0) {
    // Wait for the measurement to complete:
    delay(status);

    // Retrieve the completed temperature measurement: Note that the
measurement is stored in the variable T.
    // Function returns 1 if successful, 0 if failure.
```

```
    status = pressure.getTemperature(T);
    if (status != 0) {
      // Start a pressure measurement:
      // The parameter is the oversampling setting, from 0 to 3 (highest
res, longest wait).

      status = pressure.startPressure(3);
      if (status != 0)
      {
        // Wait for the measurement to complete:
        delay(status);

        // Retrieve the completed pressure measurement:
        // Note that the measurement is stored in the variable P.
        // Note also that the function requires the previous temperature
measurement (T).
        // (If temperature is stable, you can do one temperature
measurement for a number of pressure measurements.)
        // Function returns 1 if successful, 0 if failure.

        status = pressure.getPressure(P,T);
        if (status != 0)
        {
          Serial.print("{\"sensor\":\"BMP180\",\"valores\":{");
          Serial.print("\"temperatura\":");
          Serial.print((int)T);
          Serial.print(",\"presion\":");
          Serial.print((int)P);
          Serial.print("}}");

        } else {
          Serial.println("error retrieving pressure measurement\n");
        }
      } else {
        Serial.println("error starting pressure measurement\n");
      }
    } else {
      Serial.println("error retrieving temperature measurement\n");
    }
  } else {
    Serial.println("error starting temperature measurement\n");
```

```
  }
}

void setup()
{
  Serial.begin(9600);
  start_BMP180();
  // Comenzamos el sensor DHT
  dht.begin();
}

void loop()
{

  Serial.print("{\"lectura\":[");
  read_DHT11();

  Serial.print(",");
  read_hygrometer();

  Serial.print(",");
  read_BMP180();
  Serial.println("]}");

  delay(DELAY_TIME);  // Pause for DELAY_TIME milliseconds.
}
```

El código completo puede encontrarse en el repo de github.

Leyendo los datos desde la raspberry pi

Ya hemos serializado los datos a través del puerto serie. Ahora tenemos que leer los datos en el otro extremo, en el lado de la raspberry pi.

En este otro extremo hemos implementado un programa python que estará leyendo de puerto serie que se genera al conectar el arduino por usb. En nuestro caso el puerto es `/dev/ttyACM0` .

Ya hemos definido el formato json que enviaremos desde la parte de arduino. Desde la parte de python tenemos que leerlo y procesarlo.

Afortunadamente python trae soporte nativo para leer de json y convertirlo a diccionarios con anidamientos si es necesario.

Para leer una cadena de caracteres que contiene un json y convertirlo en una estructura de datos de diccionarios anidados si es necesario solo tenemos que ejecutar `json.loads(cadena_de_caracteres)` . Eso devolverá una estructura de datos con diccionarios, listas, cadena de caracteres y números anidados según el formato json.

El código resultante que lee de json es el siguiente:

```
import time
import serial
import json

ser = serial.Serial("/dev/ttyACM0", baudrate=9600) #Modificar el puerto
serie de ser necesario

try:
    while True:
        read = ser.readline()
        print("Leido: " + str(read))
        data = json.loads(read[0:-2])
        print(data)

except KeyboardInterrupt:
    print("\nInterrupcion por teclado")
except ValueError as ve:
    print(ve)
    print("Otra interrupcion")
finally:
    ser.close()
```

Podéis encontrar el código en el repo de github.

Este código, si se ejecuta mientras está conectado con el arduino y ejecutando el programa de arduino, debe generar una salida como la siguiente (estas son dos lecturas de ejemplo):

```
Leido:
b'{"lectura":[{"sensor":"DHT11","valores":{"humedad":73.00,"temperatura":22
```

```
.40}},{"sensor":"DZ0325","valores":{"humedad_terreno":0}},{"sensor":"BMP180
","valores":{"temperatura":21,"presion":998}}]}\r\n'
{'lectura': [{'sensor': 'DHT11', 'valores': {'humedad': 73.0,
'temperatura': 22.4}}, {'sensor': 'DZ0325', 'valores': {'humedad_terreno':
0}}, {'sensor': 'BMP180', 'valores': {'temperatura': 21, 'presion': 998}}]}
Leido:
b'{"lectura":[{"sensor":"DHT11","valores":{"humedad":73.00,"temperatura":22
.50}},{"sensor":"DZ0325","valores":{"humedad_terreno":0}},{"sensor":"BMP180
","valores":{"temperatura":21,"presion":998}}]}\r\n'
{'lectura': [{'sensor': 'DHT11', 'valores': {'humedad': 73.0,
'temperatura': 22.5}}, {'sensor': 'DZ0325', 'valores': {'humedad_terreno':
0}}, {'sensor': 'BMP180', 'valores': {'temperatura': 21, 'presion': 998}}]}
```

Almacenando en base de datos

Una vez recibimos los datos en la raspberry pi vamos a almacenarlos en base de datos para que pueda ser consumidos por el cliente web o por cualquier otro cliente que queramos desarrollar en el futuro.

De todas las opciones posibles nos decantamos por utilizar sqlite porque:

- Es sencillo de utilizar
- Existen librerías en python y php de fácil uso
- Tienen un impacto en memoria pequeño

Estructura de la base de datos

Respecto a la estructura de datos que queremos utilizar tiene que ser lo suficientemente flexible para que permita añadir o quitar sensores sin que el proyecto deje de funcionar.

La idea que se nos ha ocurrido es agrupar los datos por los sensores que los leen (hay sensores que leen más de una magnitud física).
Por ejemplo, el sensor DHT11 lee la humedad y la temperatura ambiente por lo que nuestra propuesta sería crear una tabla cuyo nombre fuese el nombre del sensor, DHT11, y que tuviese los campos definidos las magnitudes que lee, es decir, humedad y temperatura.

A todos estas tablas habría que añadirle una fecha y hora de lectura de los datos que sería su clave primaria.

Sabiendo que en sqlite no tiene un tipo explícito para la fecha y/o hora y que las opciones que propone la documentación son:

- Utilizar el tipo TEXT como cadena de caracteres ISO8601 ("YYYY-MM-DD HH:MM:SS.SSS").
- Utilizar el tipo REAL como día del calendario Juliano, el número de días desde el mediodía en Greenwich el 24 de Noviembre de 4714 B.C. de acuerdo con el calendario Gregoriano.
- Utilizar el tipo INTEGER como Unix Time, el número de segundo desde el 1970-01-01 00:00:00 UTC.

Nosotros vamos a optar por la primera opción. Fecha y hora en formato **ISO8601 ("YYYY-MM-DD HH:MM:SS.SSS").**

Creación de tablas en sqlite

Con todo lo comentado lo primero que tenemos que hacer es crear las tablas, ya que el programa da por hecho que existen las tablas en base de datos.

Para entrar en el entorno CLI de sqlite utilizaremos el siguiente comando (en este caso utilizando sqlite v.3 pero también se podría usar la v.2):

```
$ sqlite3 data.db
SQLite version 3.31.1 2020-01-27 19:55:54
Enter ".help" for usage hints.
sqlite>
```

Lo que nos queda, `sqlite>` es el prompt de sqlite.

En ese prompt solo tenemos que pegar el código SQL para crear las 3 tablas que nos hacen falta. En el caso de que uses otros sensores con otras magnitudes pues solo tienes que adaptar este código.

```
CREATE TABLE DHT11 (datetime text, humedad real, temperatura real);
CREATE TABLE DZ0325 (datetime text, humedad_terreno integer);
CREATE TABLE BMP180 (datetime text, temperatura integer, presion integer);
```

Puedes encontrar el script en github.

Para comprobar que se han creado las tablas solo tenemos que ejecutar el comando `.tables`.

```
sqlite> .tables
BMP180  DHT11   DZ0325
sqlite>
```

Y con esto ya podemos salir del entorno de sqlite pulsando Ctrl+D (fin de archivo).

Leyendo los datos y almacenando desde python

Una vez creada la base de datos y las tablas queremos leer los datos que llegan por el puerto usb y escribirlo en base de datos.

Para ello seguimos la siguiente secuencia:

1. Empezamos por crear un objeto base de datos y un objeto puerto serie, `db` y `ser`.
2. Leemos del puerto serie, `json.loads(...)`
3. Cargamos los datos json es una estructura orientadas a objetos. Esto lo hacemos a mano. El objeto resultante es `data`.
4. Y por último almacenamos en base de datos con `db.save(data)`.

```
def main():

    db = DB(db_name)
    ser = serial.Serial("/dev/ttyACM0", baudrate=9600, timeout=5)
#Modificar el puerto serie de ser necesario

    try:
        while True:
            read = ser.readline()
            print("Leido: " + str(read))
            try:
                data_json = json.loads(read[0:-2])
                print(data_json)
                data = load_data(data_json)
                print("Data object")
                print(data)
                print()
                print("...storing in database...")
                db.save(data)
            except Exception as e:
                print("Error leyendo del puerto serie: " + str(e))
```

```
            time.sleep(5*60) # 5 minutos
```

El código completo, a continuación, incluye importación de librerías, gestión de errores y llamada a la función main.

```
import time
import serial
import json
from data import SensorData, Data, load_data
from db_storage import DB
import traceback

db_name = "data.db"

def main():

    db = DB(db_name)
    ser = serial.Serial("/dev/ttyACM0", baudrate=9600, timeout=5)
#Modificar el puerto serie de ser necesario

    try:
        while True:
            read = ser.readline()
            print("Leido: " + str(read))
            try:
                data_json = json.loads(read[0:-2])
                print(data_json)
                data = load_data(data_json)
                print("Data object")
                print(data)
                print()
                print("...storing in database...")
                db.save(data)
            except Exception as e:
                print("Error leyendo del puerto serie: " + str(e))
            time.sleep(5*60) # 5 minutos

    except KeyboardInterrupt as e:
```

```
        traceback.print_exc()
        print("\nInterrupcion por teclado")
        print(e)
    except ValueError as ve:
        print(ve)
        print("Otra interrupcion")
    finally:
        ser.close()

if __name__ == "__main__":
    main()
```

En el archivo python data.py podemos encontrar como pasamos de los datos en cargados de json a los objetos. Para ello tenemos que llamar a `load_data(json)`.

```
# This is an example a json read
"""{
   "lectura":[
      {
         "sensor":"DHT11",
         "valores":{
            "humedad":61.0,
            "temperatura":26.0
         }
      },
      {
         "sensor":"DZ0325",
         "valores":{
            "humedad_terreno":1
         }
      },
      {
         "sensor":"BMP180",
         "valores":{
            "temperatura":24,
            "presion":1000
         }
      }
   ]
}
```

```
"""

# This is an open structure that can deal with adding new sensors and
values
# we just add the timestamp to json data

from datetime import datetime

class SensorData:

    def __init__(self, sensor_name):
        self.sensor = sensor_name
        self.values = {}

    def add_value(self,magnitude, value):
        self.values[magnitude] = value

    def __repr__(self): # be unambiguous
        s = "Sensor:" + self.sensor + ";"
        s = s + "Values:["
        for k,v in self.values.items():
            s = s + str(k) + ":" + str(v) + ","
        if s[-1]==",": # remove last , if exists
            s = s[:-1]
        s = s + "]"
        return s

    def __str__(self): # be readable
        return self.__repr__()


class Data:
    date_time_str_format = "YYYY-MM-DD HH:MM:SS.SSS"

    def __init__(self):
        self.lecturas = []
        self.date_time = datetime.now()

    def add(self, lectura):
        self.lecturas.append(lectura)
```

```
    def __repr__(self):  # be unambiguous
        s = "datetime:" + str(self.date_time) + ";"
        s = s + "lecturas: ["
        for d in self.lecturas:
            s = s + str(d) + ","
        if s[-1]==",": # remove last , if exists
            s = s[:-1]
        s = s + "]"
        return s

    def __str__(self):  # be readable
        return self.__repr__()

def load_data(json):
    """{
       "lectura":[
          {
             "sensor":"DHT11",
             "valores":{
                "humedad":61.0,
                "temperatura":26.0
             }
          },
          {
             "sensor":"DZ0325",
             "valores":{
                "humedad_terreno":1
             }
          },
          {
             "sensor":"BMP180",
             "valores":{
                "temperatura":24,
                "presion":1000
             }
          }
       ]
    }
    """
    data = Data()
    lecturas = json["lectura"]
```

```
    for l in lecturas:
        sensor_name = l["sensor"]
        sensorData = SensorData(sensor_name)
        values = l["valores"]
        for k,v in values.items():
            magnitude = k
            value = v
            sensorData.add_value(magnitude, value)
        data.add(sensorData)
    return data
```

En el archivo `db_storage.py` encontramos todo el código que almacena en base de datos los datos leídos de json y almacenados en los objetos.

En este caso cabría destacar la función `__create_inserts(data)` que genera una lista de cadena de caracteres donde cada una de ellas es una instrucción `SQL INSERT` que añade datos a una tabla.

La tabla viene definida por el nombre del sensor. Las magnitudes serán cada uno de los campos del json (por ejemplo, humedad, temperatura, … o cualquier que queráis definir), y los valores de esas magnitudes serán los valores leídos por los sensores en cada instante.
La clave primaria de todos ellos será la hora a la que se han leído en la parte python.

El código resultante es el siguiente:

```
import sqlite3
import data
import traceback
import os

class DB:
    date_time_str_format = "YYYY-MM-DD HH:MM:SS.SSS"

    def __init__(self):
        self.db_name = "data"

    def __init__(self, db_name):
        self.db_name = db_name
```

```
    def connect(self):
        conn = sqlite3.connect(self.db_name)
        return conn

    def __generate_inserts(self, data: data.Data):
        inserts = []
        date_time = str(data.date_time)
        insert_string = "INSERT INTO <table> (<fields>) VALUES (<values>)"
        insert_string = insert_string.replace("<fields>", "datetime," +
"<fields>")
        insert_string = insert_string.replace("<values>", "\"" + date_time
+ "\"," + "<values>")
        for lectura in data.lecturas:
            fields_string = ""
            values_string = ""
            current_insert = insert_string
            sensor = lectura.sensor
            current_insert = current_insert.replace("<table>", sensor)
            values = lectura.values
            for magnitude,value in values.items():
                fields_string = fields_string + str(magnitude) + ","
                values_string = values_string + str(value) + ","
            fields_string = fields_string.rstrip(",")
            values_string = values_string.rstrip(",")
            current_insert = current_insert.replace("<fields>",
fields_string)
            current_insert = current_insert.replace("<values>",
values_string)
            inserts.append(current_insert)
        return inserts

    def save(self, data: data.Data):
        conn = self.connect()
        cur = conn.cursor()
        working_dir = os.getcwd()

        inserts = self.__generate_inserts(data)

        try:
            for insert in inserts:
                cur.execute(insert)
```

```
                conn.commit()
        except Exception as e:
            traceback.print_exc()

        finally:
            conn.close()
```

En la siguiente carpeta de github puedes encontrar todo el código fuente para leer desde los sensores conectados al arduino, la serialización, la lectura por parte del programa en python y el almacenamiento en la base de datos sqlite.

Mostrando los datos en la web

En esta sección explicaremos lo necesario para poner a funcionar el servidor web con el software apache 2. También utilizaremos el módulo de php y el módulo para el sistema gestor de base de datos SQLite.

Software necesario

Lo primero que tenemos que hacer es instalar el software necesario para tener un servidor web con php y con soporte para SQLite.

Afortunadamente el software que necesitamos está en los repositorios oficiales de la distribución y podemos instalarlos.

Para instalar servidor web apache tenemos que ejecutar.

```
$ sudo apt-get install apache2
```

Tras ello será necesario instalar el módulo de php.

```
$ sudo apt install php libapache2-mod-php
```

Para poder acceder a la base de datos tendremos que instalar el módulo de SQLite 3.

```
$ sudo apt-get install php7.0-sqlite3
```

Por último será necesario reiniciar el servicio de apache para que utilice los nuevos módulos. Para ello ejecutaremos los siguiente.

```
$ sudo systemctl restart apache2
```

Con esto ya tendríamos todo el software necesario para poder ejecutar en nuestra raspberry pi el servidor web que leerá los datos de la base de datos de SQLite donde previamente el programa python habrá almacenado los datos leídos de los sensores.

Google charts

Creemos que es necesario comentar que para mostrar las gráficas estamos haciendo uso de la librería de google charts.

Esta librería no es necesaria instalarla porque la página web hace uso de ella en forma de dependencia con la url donde se descarga la librería y también hace uso de la API de google charts, todo ello de una manera bastante sencilla y transparente para el programador.

La web

El código fuente de la web está subido al repositorio de github.
Este código utiliza html, css (y bootstrap), php, google fonts y varias librerías de javascript como pueden ser Jquery o google charts. La explicación de como funciona todo esto se escapa un poco del objetivo de esta guía, por lo que no se entrará en el detalle de dicha implementación.

Para hacer funcionar la web solo hay que poner la web en alguna ruta accesible por el servidor apache. Por defecto la ruta suele ser `/var/www/html/` aunque puede ser otra.
Otra opción interesante es utilizar enlaces simbólicos para tener la web en un directorio dentro de tu home de usuario y que esté disponible en la carpeta de `html` de apache tal y como comentan en este hilo.

Captura de la web

Tras poner en marcha el servidor web y desplegar los archivos en la carpeta `html` del servidor apache deberíamos ver algo parecido a lo siguiente.

IES Arroyo de la Miel

Monitorización Ambiental

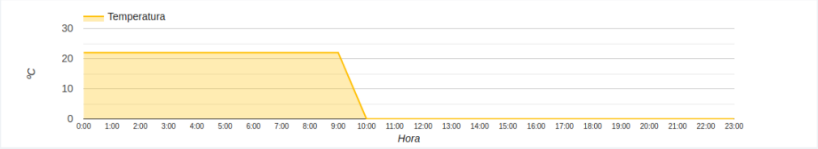
Resultados de los datos de monitorización ambiental del huerto del IES Arroyo de la Miel.

Los datos a recoletar son:

- humedad
- temperatura
- presión
- humedad de la tierra

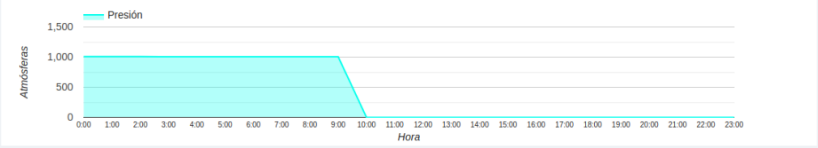
Curva de Temperatura

Curva de temperatura ambiente leída por el sensor BMP180.



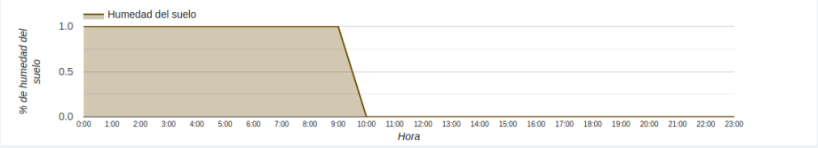
Curva de Presión Atmosférica

Curva de presión atmosférica leída por el sensor BMP180.



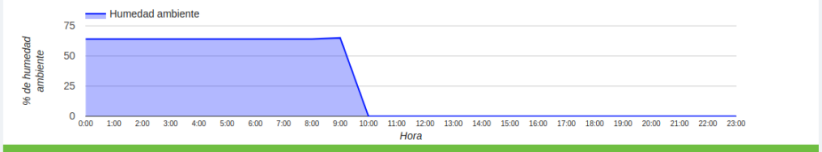
Curva de Humedad del terreno

Curva de humedad del terreno leída por el sensor DZ0325.



Curva de Humedad

Curva de humedad ambiente leída por el sensor DHT11.



Posibles ampliaciones

En esta sección vamos a comentar posibles ampliaciones que podemos hacer a este proyecto.

Broker de mensajes

Podemos instalar un broker de mensajes para enviar la información y con el broker redirigir la información a distintas fuentes (base de datos, logs de datos, web en tiempo real, aplicación móvil en tiempo real, ...)

Instalando mosquitto

Tenemos que tener instalado el broker de mensajería mosquitto. Para ello, en la Raspberry Pi con el sistema operativo Raspberry Pi OS, ejecutaremos el siguiente comando:

```
$ sudo apt update
$ sudo apt install -y mosquitto mosquitto-clients
```

Como queremos que el servicio de mosquito arranque cuando arranca el sistema operativo tenemos que ejecutar el siguiente comando:

```
$ sudo systemctl enable mosquitto.service
```

Por último vamos a comprobar que mosquitto está realmente instalado y funcionando. Lo hacemos con el siguiente comando:

```
$ mosquitto -v
1604314520: mosquitto version 1.5.7 starting
1604314520: Using default config.
1604314520: Opening ipv4 listen socket on port 1883.
1604314520: Error: Address already in use
```

Integración con Home Assistant

Home Assistant es un proyecto de código abierto que tiene como objetivo implementar un sistema de domótica y automatización para el hogar. Es un proyecto desarrollado por una comunidad de entusiastas del mundo DIY y está orientado a que la herramienta funcione principalmente sobre una raspberry pi.

Reconocimientos

Varias de las imágenes y esquemas (con derechos CC BY-SA 3.0) han sido obtenidos de la iniciativa de proyectos open source fritzing .

Los esquemas de arduino y conexionado se han realizado utilizando la herramienta disponible en este mismo proyecto que podéis encontrar en el siguiente enlace.