FIB UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

Facultat Informàtica de Barcelona

Internet of Things

Informe

Conexionado de anemómetro y veleta

25 de mayo de 2017

Índice

Índice de figuras

Índice de tablas

1. Introducción

Actualmente, la contaminación del aire es controlada por redes de estaciones equipadas con instrumentos de referencia de alta calidad. como consecuencia del elevado coste, las estaciones cubren una densidad espacial relativamente baja, que se transforma en una reproducción poco precisa de la variabilidad de las concentraciones. Debido al avance en sensores móviles y aplicaciones de software, ha surgido un creciente interés en redes de sensores basados en los ciudadanos.

El **Proyecto Captor**[1] es un proyecto desarrollado por la `Universitat Politècnica de Catalunya` juntamente con la `Université Blaise Pascal Clermont-Ferrand` y otras organizaciones. El objetivo es establecer redes de sensores de bajo coste mantenidos por los ciudadanos para medir la contaminación de ozono en áreas dispersas.

Los datos del proyecto se recogerán entre el día 01/01/2015 y 31/12/2017 en 3 zonas europeas muy afectadas por la contaminación de ozono:

- Barcelonès - Vallès Occidental - Osona (Cataluña, España)
- Pianura Padana (Calle del Po, Itàlia)
- Burgenland, Steiermark and Niederösterreich (Austria)

El dispositivo encargado de capturar los datos se muestra en la figura 1. La finalidad de este trabajo es ampliar los captor para que puedan capturar información sobre el viento.

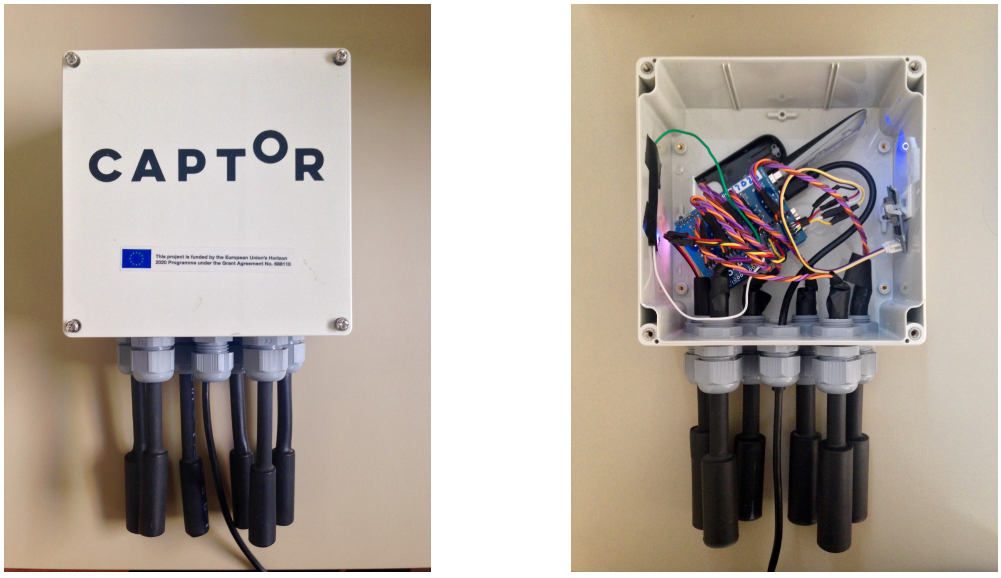

Figura 1: Dispositivo captor

[1] `https://www.captor-project.eu/es/`

2. El viento

Porqué añadimos sensores para monitorizar el viento? Que datos extraemos? Que relación tienen los datos del viento con el ozono?

3. El dispositivo

Para obtener datos sobre la velocidad y dirección del viento se usa anemómetro y veleta de la marca Davis[2] (figura 2). El sensor de velocidad del viento usa un interruptor magnético que se activa por cada revolución de las aspas. La dirección del viento es mesurada a través de un potenciómetro.

Figura 2: Dispositivo

Algunas de las especificaciones más relevantes que presenta el dispositivos se describen a la tabla 1.

	Velocidad	**Dirección**
Rango	1 - 322 km/h	0 - 360º
Precisión	±3 km/h	±7º
Resolución	1km/h	1º

Tabla 1: Especificaciones del dispositivo

Ambos sensores se conectan a través de un cable con conexión final RJ11 (figura 3) con la que vamos a sacar los valores de nuestro interés a través de un breakout.

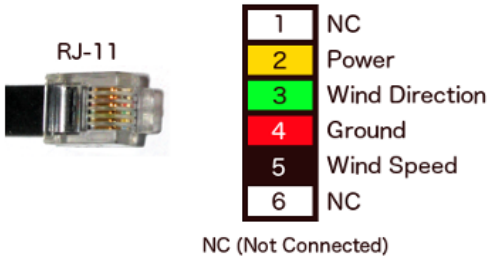

Figura 3: Esquema conexionado RJ11

[2] http://www.davisnet.com/product/anemometer-for-weather-monitor-or-wizard/

3.1. Dirección del viento

Para obtener la dirección del viento es necesario conectar la salida del conector RJ11 con uno de los pines analógicos del Arduino. El sensor incorpora un potenciómetro de 20kΩ (figura 4) que conectaremos al conversor A/D del Arduino. El conversion tiene una resolución de 10bits, lo que nos da un rango de valores de 0 a 1023, que corresponde a su vez a un voltage de 0 a 5V. Mediante código (script A) debemos convertir el rango de valores del A/D al rango de 0 a 360º que ofrece el sensor para obtener la dirección del viento.

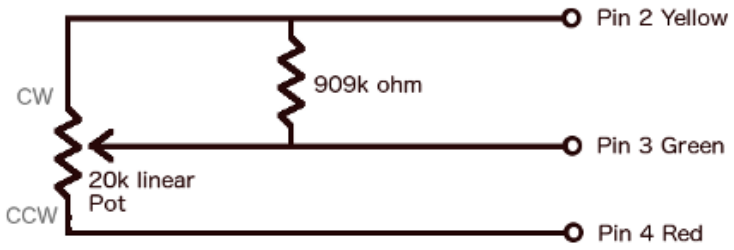

Figura 4: Esquema elétrico de la veleta

El potenciómetro tiene una "banda muerta"que dará como resultado 0º en el rango 1020 - 4 como se puede apreciar en la figura 5. El sensor está calibrado de fábrica para entregar 0 cuando la dirección está alineada con el mástil que lo sostiene.

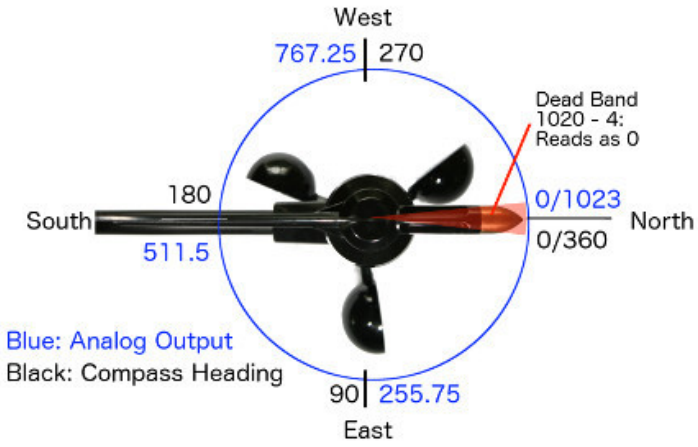

Figura 5: Esquema de la dirección

3.1.1. Resultados

Para observar los resultados se ha optado en primera instancia por enviar los valores crudos obtenidos del sensor en una primera columna y los valores ya procesados por el conversor en una segunda columna a través del puerto analógico. Un script de Python (D) captura los valores del puerto serie y los almacena en un archivo de texto; posteriormente, un segundo script de Pyhton (E) los representa gráficamente. Los valores mostrados en la figura 6 se han obtenido girando la veleta de forma manual.

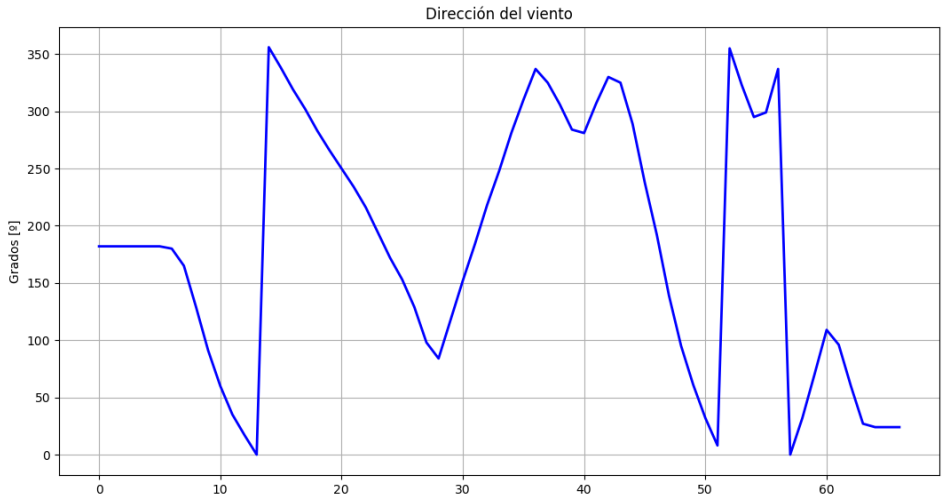

Figura 6: Resultado de la primera prueba en interior

3.1.2. Calibración del sensor

La forma ideal de montage y funcionamiento del sensor es alinear la punta de la veleta con el mástil de sujeción. Aun así, si por condiciones físicas no es posible encarar el sensor con el norte magnético, debemos aplicar un *offset* a nuestros cálculos para corregir la dirección del viento.

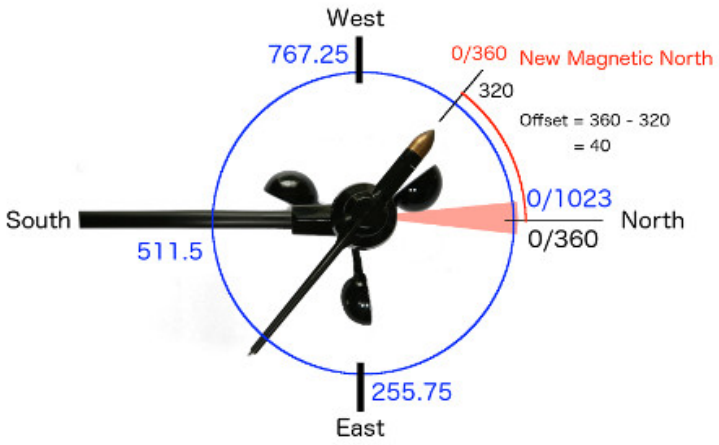

Figura 7: *Offset* en la calibración de la dirección del viento

Para determinar el valor de *offset* que debemos aplicar, necesitamos apuntar la veleta hacía el norte magnético. Con la ayuda de una brújula vamos a determinar los grados de desviación, para luego restarlo al valor calculado.

3.2. Velocidad del viento

Para obtener la dirección del viento es necesario conectar la salida negra del conector RJ11 con uno de los pines digitales del Arduino a través de un pull-up de 4k7$Omega$ que pondrá a 5V el pin del Arduino.

Las copas para medir la velocidad del viento tienen un pequeño interruptor de lámina junto al eje de rotación, que se activa una vez cada rotación (figura 8). Para calcular la velocidad del viento aplicaremos una fórmula que convierte el número de veces que se activa el interruptor por unidad de tiempo a km/h.

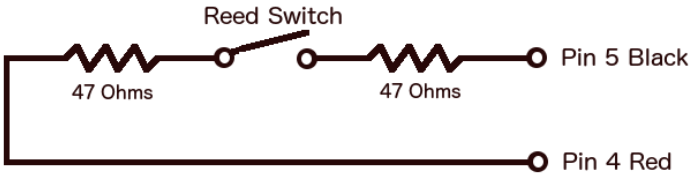

Figura 8: Esquema elétrico anemómetro

De acuerdo con la documentación oficial de Davis, 1 milla por hora equivale a 1600 revoluciones por hora. Como vamos a trabajar a nivel europeo, la conversión se hará a quilómetros por hora, según la relación: 1 mph = 1,609344 km/h.

Usando la fórmula 1 calcularemos la velocidad en km/h. Los parámetros que la definen son:

- **V** → Velocidad en km/h.
- **P** → Número de pulsos por período.
- **T** → Período en segundos.

$$V = P \cdot \left(\frac{3{,}621024}{T} \right) \tag{1}$$

A causa de activaciones naturales incontroladas del interruptor, se ha decidido trabajar con interrupciones en el pin digital del Arduino que generan una interrupción en el flanco descendente del pulso que incrementa un contador.

Se usa un *delay* de 3 segundos para obtener una media de la dirección del tiempo; este valor es T en la fórmula de cálculo.

3.2.1. Resultados

A diferencia de la primera parte, ahora no obtenemos unos datos en crudo, sino que se calcula según el número de rotaciones por período. Para observar los resultados se ha optado por guardar en una primera columna el número de rotaciones por período y en

una segunda columna el valor de la velocidad. La figura 9 representa la primera prueba, en la que se ha soplado el anemómetro en un espacio cerrado.

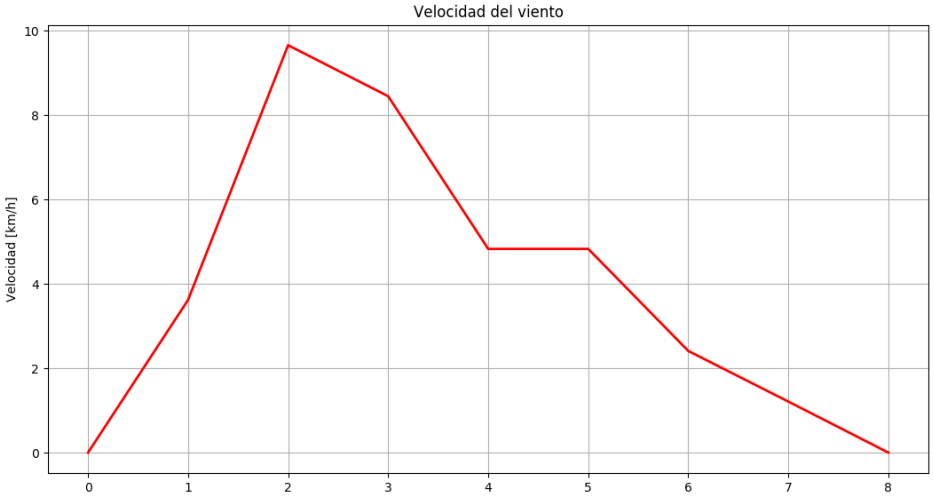

Figura 9: Resultado de la primera prueba en interior

4. Resultados finales

Una vez comprobado el buen funcionamiento de los sensores de forma separada, es vital crear un script que capture los datos de los dos sensores para obtener los resultados a la vez (script C).

El resultado se ha representado de forma gráfica (figura 10) mediante el script de Python previamente citado con la peculiaridad que se representa cada sensor en un eje Y distinto, pero compartiendo el mismo eje X; de esta forma es más visual la evolución de los datos.

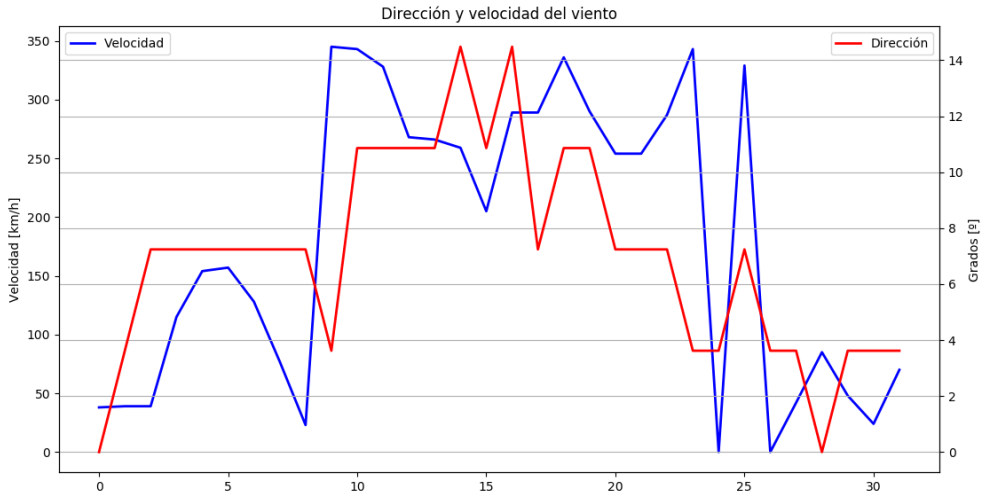

Figura 10: Resultado conjunto en interior

Referencias

[1] Arduino: Python interfacing,
https://playground.arduino.cc/Interfacing/Python

[2] Cactus io,
http://cactus.io/hookups/weather/anemometer/davis/hookup-arduino-to-davis-anemometer

[3] Davis Instruments,
http://www.davisnet.com

[4] Python: Matplotlib,
https://matplotlib.org

Anexos

A. Script windDirection.ino

```
int VaneValue;
int Direction;
int CalDirection;

#define Offset 0;

void setup() {
  Serial.begin(9600);
  Serial.println("Wind Direction");
  Serial.println("Vane Value \t Direction");
}

void loop() {
  VaneValue = analogRead(A4);
  Direction = map(VaneValue, 0, 1023, 0, 360);
  CalDirection = Direction + Offset;

  if (CalDirection > 360)
    CalDirection = CalDirection - 360;

  if (CalDirection < 0)
    CalDirection = CalDirection + 360;

  Serial.print(VaneValue);
  Serial.print("\t");
  Serial.println(CalDirection);

  delay(500);
}
```

B. Script windVelocity.ino

```
#include <math.h>

volatile unsigned long Rotations;
volatile unsigned long ContactBounceTime;

float WindSpeed;

void setup() {
  Serial.begin(9600);
  pinMode(2, INPUT);
  attachInterrupt(digitalPinToInterrupt(2), isr_rotation, FALLING);

  Serial.println("Wind Velocity");
  Serial.println("Rotacions \t km/h");
}

void loop() {
  Rotations = 0;

  sei();
  delay(1000);
  cli();

  WindSpeed = (3.621024/1) * Rotations;
  Serial.print(Rotations);
  Serial.print("\t");
  Serial.println(WindSpeed);
}

void isr_rotation() {
  if ((millis() - ContactBounceTime) > 15) {
    Rotations++;
    ContactBounceTime = millis();
  }
}
```

C. Script windFull.ino

```
#include <math.h>

volatile unsigned long Rotations;
volatile unsigned long ContactBounceTime;

float WindSpeed;
int VaneValue;
int Direction;
int CalDirection;

#define Offset 0;

void setup() {
  Serial.begin(9600);
  pinMode(2, INPUT);
  attachInterrupt(digitalPinToInterrupt(2), isr_rotation, FALLING);

  Serial.flush();
}

void loop() {
  //Direction
  VaneValue = analogRead(A4);
  Direction = map(VaneValue, 0, 1023, 0, 360);
  CalDirection = Direction + Offset;

  if (CalDirection > 360)
    CalDirection = CalDirection - 360;

  if (CalDirection < 0)
    CalDirection = CalDirection + 360;

  //Speed
  Rotations = 0;

  sei();
  delay(1000);
  cli();

  WindSpeed = (3.621024/1) * Rotations;

  Serial.print(CalDirection);
  Serial.print("\t");
  Serial.println(WindSpeed);
}

void isr_rotation() {
  if ((millis() - ContactBounceTime) > 15) {
    Rotations++;
    ContactBounceTime = millis();
  }
}
```

D. Script readSerial.py

```
#-*- encoding:utf-8 -*-
import sys
import serial

try:
    ser = serial.Serial('/dev/cu.usbmodem1411', 9600)
except:
    print 'Error de conexion'
    sys.exit()

data_file = sys.argv[1]
f = open(data_file, "w")

while True:
    try:
        f.write(ser.readline())
    except (KeyboardInterrupt):
        f.close()
        sys.exit()
```

E. Script plotData.py

```
#-*- encoding:utf-8 -*-
import sys
import matplotlib.pyplot as plt

def read_data(filename, t):
    f = open(filename, "r")
    data = []
    dataFull = [[], []]

    for line in f.readlines():
        if t == 'D' or t == 'V':
            data.append(line.split("\t")[1])
        else:
            dataFull[0].append(line.split("\t")[0])
            dataFull[1].append(line.split("\t")[1])

    if t == 'D' or t == 'V':
        return data
    else:
        return dataFull

def plot(data, t):
    if t == 'D':
        plt.plot(data, 'b', linewidth = 2)
        plt.title(u'Direccion del viento')
        plt.ylabel(u'Grados')
    elif t == 'V':
        plt.plot(data, 'r', linewidth = 2)
        plt.title(u'Velocidad del viento')
        plt.ylabel(u'Velocidad [km/h]')
    else:
        fig, ax1 = plt.subplots()
        ax1.plot(data[0], 'b', linewidth = 2, label = u'Velocidad')
        ax1.set_ylabel(u'Velocidad [km/h]')
        ax1.legend(loc = 2)

        ax2 = ax1.twinx()
        ax2.plot(data[1], 'r', linewidth = 2, label = u'Direccion')
        ax2.set_ylabel(u'Velocidad [km/h]')
        ax2.set_ylabel(u'Grados')
        ax2.legend(loc = 0)

        plt.title(u'Direccion y velocidad del viento')

    plt.grid(True)
    plt.show()

if __name__ == '__main__':
    filename = sys.argv[1]
    t = filename[4]
    data = read_data(filename, t)
    plot(data, t)
```