RELATÓRIO 2: Arduino UNO R3 - Sensores

Física Aplicada à Computação – Licenciatura em Engenharia Informática

Ano Lectivo 2016-17

Docente: Nuno Pereira

IPBeja – Departamento de Matemática e Ciências Físicas

Autor(es) João Faria, n.º 16145
Luís Prata, n.º 15970
Ricardo Madeira, n.º 16147
Data 04 de Fevereiro de 2017

CONTEÚDO

RESUMO

Contexto: Utilizar o Arduino e sensor de ultra-sons para cálculo de distância e velocidade. Utilizar o Arduino e sensor de humidade e temperatura para obtenção de valores de humidade e temperatura

Objectivos: Desenvolver competências de programação no Arduino.

Métodos: Utilização dos sensores HC-SR04 e DHT21 e criação de código para que os mesmo consigam adquirir dados.

Resultados: Através dos códigos escritos foram medidos valores de distância, velocidade média, humidade e temperatura.

Conclusões: Os valores obtidos foram satisfatórios.

1. INTRODUÇÃO

Para a realização deste trabalho práctico utilizamos o Arduino, ao qual adicionamos dois sensores. O objectivo do primeiro sensor, o HC-SR04, foi o de medir uma distância através da emissão de um pulso ultra-sónico e posterior recepção do mesmo, após este ser reflectido por uma superfície.

O sensor foi utilizado com meio de obtenção de valores temporais, que por aplicação de alguns conhecimentos básicos de física podem ser transformados em valores de distância.

O segundo sensor utilizado foi o DHT-21, teve como objectivo recolher dados de humidade e temperatura, demonstrando assim a facilidade com que é possível fazer a aquisição de dados com um sistema como o Arduino.

2. APARATO EXPERIMENTAL

Para efectuar o trabalho prático foi necessário efectuar duas montagens diferentes no Arduino.

Na primeira foi montado o sensor HC-SR04 (sensor de ultra-sons) com as respectivas ligações GND e VCC, aos pinos GND e 5V do Arduino. O pino TRIGGER foi ligado ao pino DIGITAL 10 do Arduino e o pino ECHO ao pino DIGITAL 11 do Arduino.

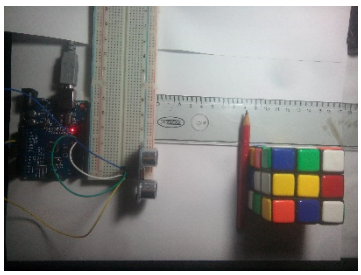

Figura 1 - Montagem do sensor HC-SR04

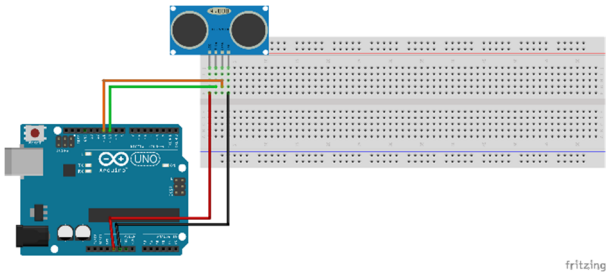

Figura 2- Esquema da montagem do sensor HC-SR04

Na segunda foi montado o sensor DHT-21 (sensor de temperatura e humidade) sendo o fio preto ligado ao GND do Arduino, o fio vermelho ligado aos 5V do Arduino e finalmente o fio amarelo (serial data) ligado ao pino DIGITAL 2 do Arduino.

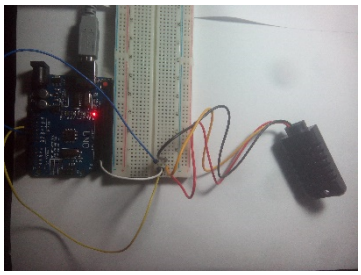

Figura 3- Montagem do sensor DHT21

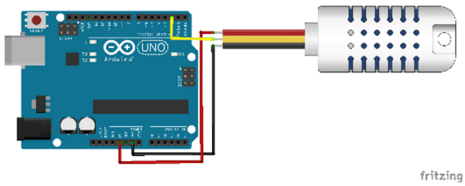

Figura 4- esquema da montagem do sensor DHT21

Foi ainda feita uma terceira montagem, por o nosso colega João Faria, que decidiu aprofundar os seus conhecimentos sobre esta versátil plataforma.

Nesta montagem foi incluído um ecrã LCD onde são apresentados os dados recolhidos por os dois sensores.

Figura 5 - Montagem adicional com ecrã LCD

3. MEDIÇÃO DE DISTÂNCIA COM O SENSOR HC-SR04

Para implementar a medição de distâncias com o sensor HC-SR04 foi necessário escrever o código seguinte.

```
1  const int PinTrigger=10;
2  const int PinEcho=11;
3  const int Led=13;
4
5  void setup()
6  {
7    pinMode(PinTrigger,OUTPUT); // HC-SR04 Trigger
8    pinMode(PinEcho,INPUT); // HC-SR04 Echo
9    pinMode(Led,OUTPUT); // On-Board LED
10
11   Serial.begin(9600); // open the serial port at 9600 bps
12   Serial.println("");
13   Serial.println("POWER ON!"); // Hello World!!!
14   Serial.println("");
15
16   delay(250); // take a break, enjoy the view....
17 }
18
19 void loop()
20 {
21   Serial.print("Distancia - ");
22   Serial.print(Distance());
23   Serial.println(" cm");
24   Serial.println("------------------");
25
26   delay(1000);
27 }
28
29 long Distance() //HC_SR04 Distance
30 {
31   long duration, distance;
32
33   digitalWrite(PinTrigger, LOW);
34   delayMicroseconds(2);
35   digitalWrite(PinTrigger, HIGH);
36   delayMicroseconds(10);
37   digitalWrite(PinTrigger, LOW);
38
39   duration = pulseIn(PinEcho,HIGH); // mesuares the time the PinEcho is
   recieving
40   distance = duration /58; // Distance in cm.
41                            // The value 1/58 is given in the data sheet has
   the value to obtain mesurements in cm.
42
43   return distance;
44 }
```

Código 1 - Cálculo de distância com o sensor HC-SR04

Nas primeiras três linhas foram definidos os pinos utilizados, tendo sido o pino DIGITAL 10 atribuído ao pino TRIGGER do sensor e o pino DIGITAL 11 ao pino ECHO do sensor.

Foi também definido o pino 13 com o LED do Arduino.
Nas linhas 7 a 9 foram definidos os modos de cada pino sendo os pinos TRIGGER e LED definidos com output de sinal e o pino ECHO como input de sinal.
Na linha 11 foi pedido ao Arduino para comunicar nos 9600 bps e de seguida nas linhas 12 a 14 imprime-se uma pequena mensagem que adverte para o inicio do código.
Pede-se então ao Arduino para fazer um pequeno intervalo de 250ms.
Na linha 19 começa a função **`loop()`**, que vai imprimir as informações no ecrã e pedir à função **`Distance()`** que calcule a distância. Entre cada *loop* há um intervalo de 1 segundo, para que o sensor possa efectuar o seu trabalho.
Na linha 29 é definida a função **`Distance()`** que irá retornar uma variável *long* chamada **`distance`** que nada mais é que a distância calculada.
Na linha 35 o TRIGGER é colocado no estado LOW, durante 2 milissegundos, para que possamos garantir que o estado inicial do sensor é mesmo LOW.
De seguida o comando da linha 35 coloca o TRIGGER em estado HIGH, durante 10 milissegundos. Este estado HIGH inicializa o sensor, que emite uma série de 8 pulsos ultra-sónicos (frequência 40 kHz).
No final dos 10 milissegundos, o TRIGGER é retornado ao estado LOW, para garantir que mais nenhum pulso é emitido e o ECHO é colocado em estado HIGH, ficando em estado HIGH até receber o ultimo dos 8 pulsos enviado. O período que o ECHO permaneceu em HIGH informa-nos sobre a quantidade de tempo que os pulso ultra-sónicos estiveram em movimento desde serem emitidos até serem recebidos no mesmo ponto.
Neste momento passamos a ter o nosso tempo de "viagem" sob a forma de uma variável long chamada **`duration`**, na linha 39, em milissegundos.
Para calcular a distância temos que ter em atenção que o tempo de "viagem" refere-se à ida e volta dos pulsos ultra-sónicos, pelo que este tempo representa duas vezes a distância entre o sensor e o objecto que reflectiu os pulsos.
Aplicando alguns conhecimentos básicos de física podemos escrever que:

$$\Delta x = \frac{V_{som} \times \Delta t}{2}$$

Fazendo alguns cálculos descobrimos que o factor de proporcionalidade, para que a distância seja apresentada em cm com o tempo em milissegundos, é de $\frac{17}{1000}$, que nos é apresentado no manual do sensor na forma aproximada, mas mais simpática, de $\frac{1}{58}$.
Assim na linha uma nova variável chamada **`distance`** é criada e o seu valor é igual a **`duration`** dividida por 58.
Na linha 43 o valor de **`distance`** é devolvido para a função **`loop()`** e o valor é impresso no ecrã.

4. MEDIÇÃO DE VELOCIDADE MÉDIA COM O SENSOR HC-SR04

Para implementar a medição de velocidades médias com o sensor HC-SR04 foi necessário escrever o código seguinte.

```
1 const byte PinTrigger = 10; // HC-SR04 Trigger Pin
```

```
const byte PinEcho = 11; // HC-SR04 Echo Pin
const unsigned int PAUSE = 500;
const boolean plotter = false; // 'true' for serial plotter, 'false' for serial monitor
const float fVelocity = 0.8;//smoothening factor for velocity

long t2;
float x2;

void setup()
{
  pinMode(PinTrigger, OUTPUT);
  pinMode(PinEcho, INPUT);
  Serial.begin(9600);
}

void loop()
{
  int d = constrain(ComputeDistance(), 0, 400);
  float v = ComputeSpeed(d);
  if(v > 400 || v < -400){v =0;}//filters initial peaks of velocity when the sensor starts
  float vSmooth = SmoothVelocity(v);

  if (plotter)
  {
    Serial.print(d); Serial.print("\t");
    Serial.print(v); Serial.print("\t");
    Serial.print(vSmooth); Serial.println("");
  }
  else
  {
    Serial.print("Distancia: ");
    Serial.print(d);
    Serial.println(" cm\t");
    Serial.print("Velocidade: ");
    Serial.print(v);
    Serial.println(" cm/s\t");
    Serial.print("Velocidade suavizada: ");
    Serial.print(vSmooth);
    Serial.println(" cm/s");
    Serial.println("---------------------------------");
  }
  delay(PAUSE*4);
}

float ComputeDistance()
{
  digitalWrite(PinTrigger, LOW);
```

```
  delayMicroseconds(2);
  digitalWrite(PinTrigger, HIGH);
  delayMicroseconds(10);
  digitalWrite(PinTrigger, LOW);

  unsigned int duration = pulseIn(PinEcho, HIGH); // mesaures the time
  unsigned int distance = duration / 58; // Distance in cm.
  return distance;
}

float ComputeSpeed(int distance)
{
  unsigned int t1 = t2;
  t2 = millis();

  unsigned int x1 = x2;
  x2 = distance;

  float velocity = (x2 - x1) / ((t2 - t1) / 1000.0); // calculates speed
  return velocity;
}

float SmoothVelocity(float v)
{
  float vValue = vValue + (v - vValue) * fVelocity;
  return vValue;
}
```

Código 2 - Código utilizado para cálculo de velocidade média com o sensor HC-SR04

Para calcular a velocidade média alteramos o código anterior, especialmente nos tipos de variáveis, pois sentimos que estas alterações optimiza o processamento dos dados.

A função que calcula a distância passou a chamar-se **`ComputeDistance()`**, mas é muito semelhante à descrita no ponto anterior, pelo que não iremos repetir o seu funcionamento. Foram adicionadas duas novas funções, a função **`computeSpeed()`** e a função **`smoothVelocity()`**.

No topo declaramos algumas variáveis, entre as quais os pinos utilizados pelo sensor, um valor chamado **`PAUSA`** que irá ser o nosso valor de intervalo entre as iterações da função **`loop()`**, um *boolean* que permite trocar entre um bloco de código preparado para fazer *output* para o *serial monitor* e outro preparado para fazer *output* para o *serial plotter* e finalmente o factor de *smoothening*. De seguida são inicializadas duas variáveis, **`t2`** e **`x2`**. Estas variáveis irão guardar os valores de calculados nas funções **`ComputeDistance()`** e **`ComputeSpeed()`**.

No **`setup()`**, apenas são definidos os modos dos pinos e o canal no qual o arduino deve comunicar.

Passamos então ao **`loop()`**, aqui fazemos uma chamado da função **`ComputeDistance()`** e guardamos o *return*, com uma limitação entre 0 e 400, pois estes são os valores para os quais o manual do sensor garante precisão de leitura, numa variável **`d`**. De seguida é chamada a função

computeSpeed() e o *return* é guardado numa variável **v**. Foi colocada uma condicional que atribuí o valor 0 à variável **v**, caso o resultado da chamada da função **ComputeSpeed()** seja superior a 400 ou inferior a -400. Esta característica foi adicionada pois ao iniciar o código verificamos que a primeira leitura de velocidade era muito grande e não tinha significado, mas no caso do *serial plotter* transformava a escala de um modo que era difícil ler a informação do gráfico.

Por fim é chamada a função **vSmooth()** que faz o *smoothening* do valor da velocidade e devolve-o para uma variável **vSmooth**. Uma condicional determina se desejamos mostrar os resultados em formato próprio para *serial monitor* ou para *serial plotter* e o *output* é então feito. Faz-se um *delay* no valor de **PAUSE**, e corremos novamente o **loop()**.

A função **ComputeSpeed(int distance)** inicialmente atribui o valor de tempo da iteração anterior à variável **t1** (para que esta fique guardada) e por meio do comando **milis()** determina o número de milissegundos que passaram desde que o código iniciou. A cada iteração do código **milis()** dá-nos um valor maior, pelo que se fizermos entre as leituras de duas iteração seguidas obtemos a diferença temporal entre uma leitura de distância e a anterior. De seguida utiliza-se uma estratégia igual para o calculo da diferença de posições, mas em vez de utilizar um comando, utilizamos a nossa variável **distance** (que foi recolhida na função **ComputeDistance()**). Finalmente calculamos o quociente entre a diferença das distâncias e a diferença dos tempos, tendo o cuidado de dividir estes por 1000, pois tratam-se de valores em milissegundos que necessitam de ser convertidos para segundos para garantir as unidades da velocidade. O valor é devolvido ao **loop()** e guardado na variável **v**.

A ultima função que escrevemos foi a **SmoothVelocity(float v)** que calcula o *smoothening* da velocidade. O *smoothening* consiste em somar ao valor da iteração anterior(**vValue**) uma fracção da diferença entre a diferença da leitura actual e anterior (**v-vValue**), sendo essa fracção definida pela constante **fVelocity**. O resultado é então devolvido ao **loop()**, onde é guardado na variável **vSmooth**.

5. MEDIÇÃO DE TEMPERATURA E HUMIDADE COM O SENSOR DHT-21

Para implementar a medição de humidade e temperatura utilizamos o sensor DHT-21, também conhecido como AM2301.

As principais características deste sensor são o seu baixo consumo energético (300 µm em média), a distância de transmissão (chegando aos 20 metros), calibração automática e a sua precisão na medição de valores de humidade e temperatura. O sensor faz a comunicação através de uma ligação digital single-bus, o que facilita a sua instalação e utilização. Este sensor trabalha com uma voltagem entre os 3,3V e os 5,2V (sendo a voltagem recomendada 5,0V) e com um intervalo de amostragem mínimo de 2 segundos.

Em relação às leituras de humidade o sensor possui uma resolução de 0,1% e a 25 ˚C apresenta um erro de ±3%. Quanto às leituras de temperatura o sensor possui uma resolução de 0,1 ˚C, um erro de ±0,3 ˚C e um intervalo de leitura entre os -40 ˚C e os 80 ˚C.

Foi assim necessário escrever o código seguinte para obter os dados através deste sensor.

```
#include "DHT.h"
const byte DHTPIN = 2;
const byte DHTTYPE = DHT21;
DHT dht(DHTPIN, DHTTYPE);

const boolean plotter = true; // 'true' for serial plotter, 'false' for serial monitor
const float fHumidity = 0.7; //smoothening factor for humidity
const float fTemperature = 0.9; //smoothening factor for temperature
const byte ref_H = 60; //reference value for the humidity
const byte ref_T = 25; //reference value for the temperature
float hValue, tValue;

void setup()
{
  dht.begin();
  Serial.begin(9600); // open the serial port at 9600 bps

  while ( isnan(dht.readHumidity()) || isnan(dht.readTemperature()) )
  {
    Serial.print("\t");
    Serial.print("Erro ao comunicar com o DHT, verifique se o DHT esta ligado ao Arduino no pino ");
    Serial.println(DHTPIN);
    delay(2500);
  }
  hValue = dht.readHumidity();
  tValue =  dht.readTemperature();
}

void loop()
{
  float h = dht.readHumidity();
  float t = dht.readTemperature();

  if ( isnan(h) || isnan(t) ) // If there is an error, it will keep displaying a message until there's a valid reading
  {
    Serial.print("\tErro ao comunicar com o DHT, verifique se o DHT esta ligado ao Arduino no pino");
    Serial.println(DHTPIN);
  }
  else
  {
    if (plotter)
    {
      Serial.print(ref_H); Serial.print("\t");
      Serial.print(h); Serial.print("\t");
      Serial.print(H_Smooth(h)); Serial.print("\t");
```

```
46
47        Serial.print(ref_T); Serial.print("\t");
48        Serial.print(t); Serial.print("\t");
49        Serial.print(T_Smooth(t)); Serial.println("\t");
50      }
51      else
52      {
53    Serial.print("Humidade: "); Serial.print(h);
54    Serial.print("\t\tHumidade suavizada: "); Serial.print(H_Smooth(h));
55    Serial.print("\tTemperatura: "); Serial.print(t);
56    Serial.print("\tTemperatura suavizada: "); Serial.println(T_Smooth(t));
57      }
58    }
59    delay(2000);
60  }
61
62  float H_Smooth(float h)
63  {
64    hValue = hValue + ((h - hValue) * fHumidity);
65    return hValue;
66  }
67
68  float T_Smooth(float t)
69  {
70    tValue = tValue + (t - tValue) * fTemperature;
71    return tValue;
72  }
```

Código 3 - Código utilizado para medição de valores de humidade e temperatura através do sensor DHT-21

Nas primeiras linhas do código incluímos a biblioteca **`DHT.h`**, que possui funcionalidades do sensor DHT21. De seguida o Arduino é informado que o sensor DHT está ligado ao pino DIGITAL 2 e que o tipo de sensor é o DHT21. Na linha 4 o sensor é inicializado.

São definidas várias constantes. A primeira é um *boolean* que nos vai permitir mudar facilmente entre um bloco de código que está preparado para apresentar os dados no *serial plotter* (caso o *boolean* seja *True*) ou apresentar os dados no *serial monitor* (caso o *boolean* seja *False*). De seguida são definidos os factores que vão ser usados no *smoothening* dos dados obtidos. Por tentativa e erros chegamos à conclusão que os melhores valores seriam 0,7 no caso da humidade e 0,9 no caso da temperatura. Definimos então os valores de referência, que nos irão permitir ter uma linha de comparação no *serial plotter*. Os valores escolhidos forma 60% de humidade (pois no ambiente onde testamos o sensor a humidade estava próxima dos 57%) e 25 ˚C no caso da temperatura (pois no ambiente onde testamos o sensor temperatura estava próxima dos 20 ˚C). São então declaradas as variáveis **`hValue`** e **`tValues`**, variáveis que guardam os valores do *signal smoothening*.

Na função **`setup()`** é pedido ao Arduino para comunicar nos 9600 bps e é definido um ciclo while, onde é verificado se existem leituras de humidade ou de temperatura. Caso estas não seja números, será apresentada uma mensagem que informa o utilizador que existe um erro com as

ligações do sensor. Caso as leituras sejam números, os valores são guardados nas variáveis **`hValue`** e **`tValue`** de modo a serem utilizadas na primeira iteração das funções **`H_Smooth()`** e **`T_Smooth()`**. Este pequeno truque, permite que ao correr qualquer uma das funções de *smoothening* já exista um valor antigo, que vai permitir um cálculo correcto.

Iniciamos então a função **`loop()`**, que realiza uma leitura de humidade, usando o comando **`dht.readHumidity`** e uma leitura de temperatura , usando o comando **`dht.readTemperature`** (comandos estes incluído na biblioteca que foi importada na primeira linha). Os dois valores são guardados respectivamente nas variáveis **`h`** e **`t`**, que são inicializadas com o tipo *float*.

Fazemos nova verificação dos valores lidos, caso os mesmos não sejam números, é apresentada uma mensagem ao utilizador. Caso os valores lidos sejam números passamos a uma condição. No caso do *bollean* (definido no topo do código) ser *True* é executado um pequeno bloco de código que está preparado para fazer output para o *serial plotter*. No caso do *bollean* ser *False* é executado um pequeno bloco de código que está preparado para fazer output para o serial monitor. Fazemos então um *delay* de 2 segundos, que garante o período de amostragem do sensor.

Dentro dos blocos de texto mencionados, para além de se fazer output dos valores **`h`** e **`t`** são chamadas as funções **`H_Smooth()`** e **`T_Smoth()`**, que efectuam os cálculos de *smoothening* que já foram descritos anteriormente. No caso do output para o *serial plotter* é também mostrado os valores de **`ref_H`** e **`ref_T`**.

6. TRATAMENTO DE DADOS

Sendo este um trabalho práctico que incidia especialmente sobre o desenvolvimento de competências de programação de sistemas de aquisição de dados, não houve necessidade de tratamento dos dados recolhidos. Sendo a nossa única preocupação fazer algumas simples comparações que nos demonstravam se os sensores estavam a obter valores semelhantes aos dos restantes colegas e se esses mesmos valores seriam minimamente credíveis, tendo em conta as condições que estavam a ser medidas.

Apresentamos, no entanto, capturas de ecrã relativas ao *serial monitor* e aos valores que obtivemos com cada um dos códigos utilizados.

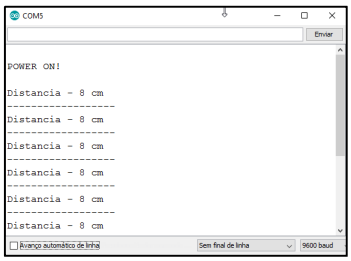

Figura 6 – Serial monitor com dados de distância

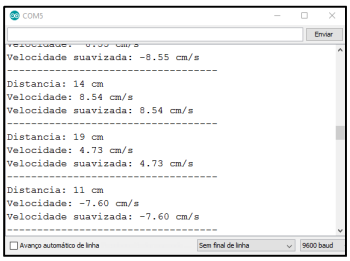

Figura 7 - Serial monitor com dados de distância e velocidade média

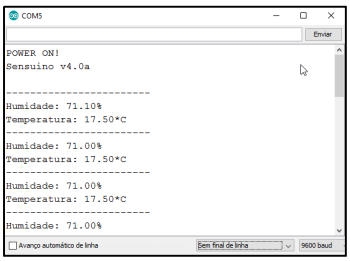

Figura 8 - Serial monitor com dados de humidade e temperatura

Apresentamos também capturas de ecrã relativas à representação de dados no *serial plotter*.

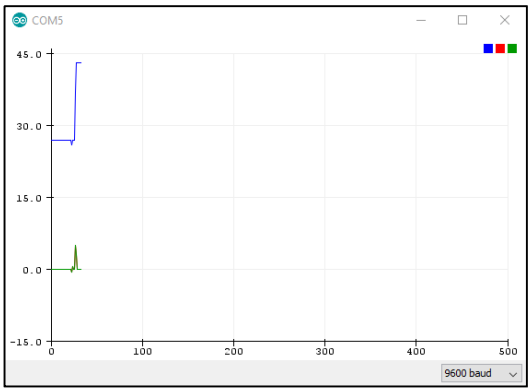

Figura 9 - Serial plotter com dados de distância e velocidade média

A linha azul representa o gráfico da distância, a vermelho a velocidade (é coincidente com a linha verde) e a verde a velocidade suavizada.

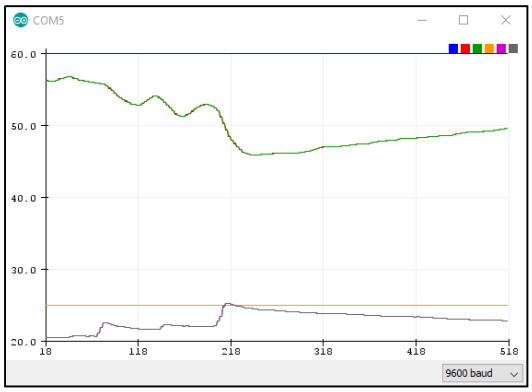

Figura 10 - Serial plotter com dados de humidade e temperatura.

A linha a azul representa o gráfico da referência para a humidade (60%), a vermelho a humidade, a verde a humidade suavizada. A laranja está representado o gráfico da referência da temperatura, a violeta a temperatura e a cinzento a temperatura suavizada.

7. DISCUSSÃO

No geral não encontramos dificuldades de maior neste trabalho. A utilização de sensores no Arduino é bastante simples e os nossos códigos, apesar de não serem muito pequenos são bastante eficientes e permitiram-nos fazer algumas experiências e até promover alguma interdisciplinaridade, através do uso de métodos e ferramentas que estamos no momento a aprender na disciplina de introdução à programação.

Em relação aos nossos códigos, não encontramos qualquer tipo de desvio com o código de cálculo das distâncias (desde que o sensor de ultra-sons fosse usado de modo adequado). Sendo os valores obtidos muito satisfatórios.

No código que calcula a velocidade média notava-se alguns resultados estranhos nas leituras iniciais, mas tentamos que estes dados fossem ignorados no nosso código. Apesar da sugestão do protocolo, para a utilização da técnica de *signal smoothening* no cálculo da velocidade, consideramos que esta técnica não trouxe qualquer vantagem. Na nossa opinião deve-se à natureza discreta dos dados de distância e velocidade, enquanto que a técnica do *signal smoothening* depende da natureza continua da grandeza medida. Assim consideramos que o melhor factor de *smoothening* para a velocidade seria 1, que é basicamente o mesmo que ignorar o valor antigo.

Em relação à utilização do sensor DHT o maior desafio foi a utilização da biblioteca sugerida no protocolo, no entanto após alguma pesquisa foi encontrada outra biblioteca que, contendo as mesmas funcionalidades, funcionava de modo eficiente. Por o que optamos por esta nova biblioteca. A utilização do sensor DHT é bastante simples e os valores parecem ser suficientemente exactos para uma utilização simples. Novamente a utilização da técnica de *signal smoothening* não apresentou melhorias às leituras realizadas. Temos que ponderar as hipóteses

de que devido ao *delay* entre leituras, estas fossem realizadas com uma maior exactidão (uma vez que era respeitado o período de amostragem de 2 segundos especificado no manual do sensor) ou que a técnica de *signal smoothening* não estivesse a ser bem implementada no código. No geral consideramos que estes pequenos sensores são bastante fáceis de utilizar e precisos o quanto baste para uma utilização normal.

Gostávamos ainda de mostrar que graças a este trabalho foi possível aprofundar os nossos conhecimentos acerca do Arduino e ganhar ainda mais gosto por esta simples e versátil plataforma. Deste modo gostávamos de dar a conhecer um trabalho desenvolvido singularmente pelo nosso colega João Faria que decidiu dar uma utilização práctica ao material que adquiriu, através da construção de uma pequena estação de monitorização meteorológica a qual ligou ao serviço thingspeak.com .

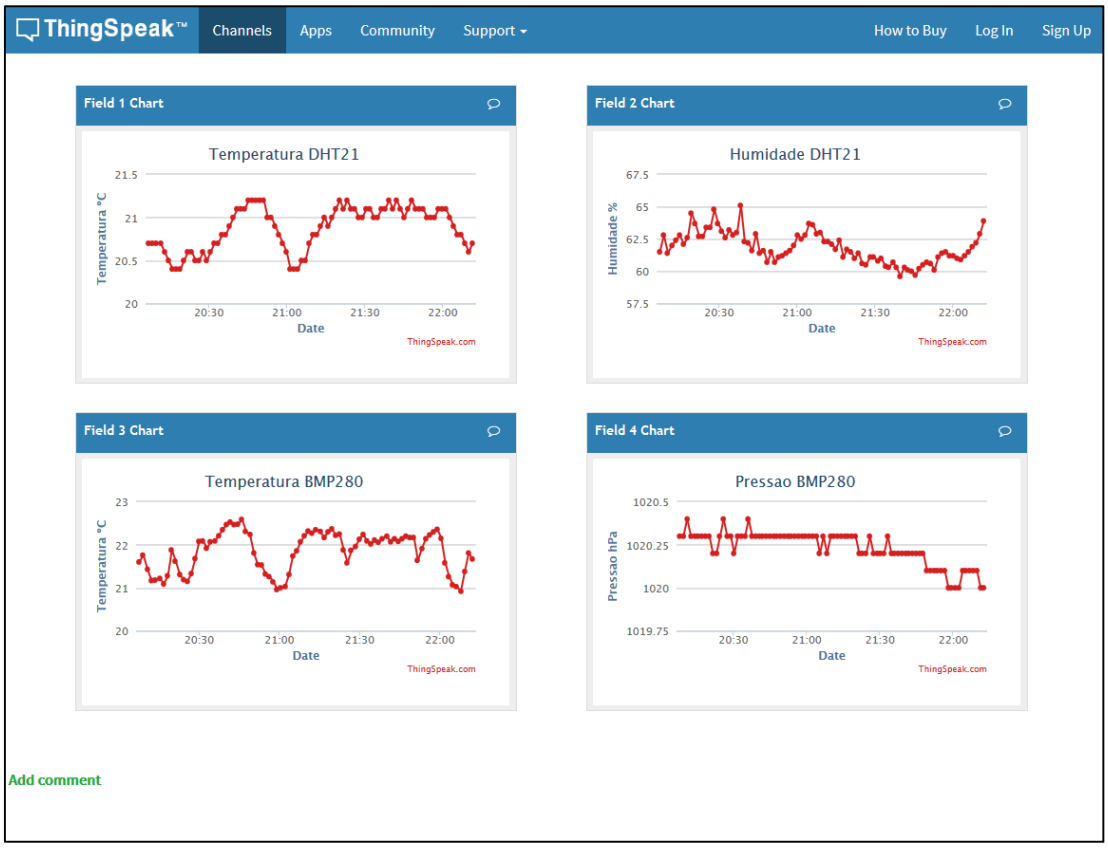

Figura 11 - Screenshot do serviço ThingSpeak criado por João Faria

LISTA DE FIGURAS

LISTA DE BLOCOS DE CÓDIGO

REFERÊNCIAS

DHT21 (AM2301) Product Manual. (24 de 01 de 2017). Obtido de kropochev.com: https://kropochev.com/downloads/humidity/AM2301.pdf

DHT-Sensor-Library. (24 de 01 de 2017). Obtido de Git-Hub.com: https://github.com/adafruit/DHT-sensor-library

Foruns. (24 de 01 de 2017). Obtido de Arduino.cc: https://forum.arduino.cc/

HC-SR04 User's_Manual. (24 de 01 de 2017). Obtido de docs.google.com: https://docs.google.com/document/d/1Y-yZnNhMYy7rwhAgyL_pfa39RsB-x2qR4vP8saG73rE/edit

Tutorials. (24 de 01 de 2017). Obtido de Arduino.cc: https://www.arduino.cc/en/Tutorial/HomePage