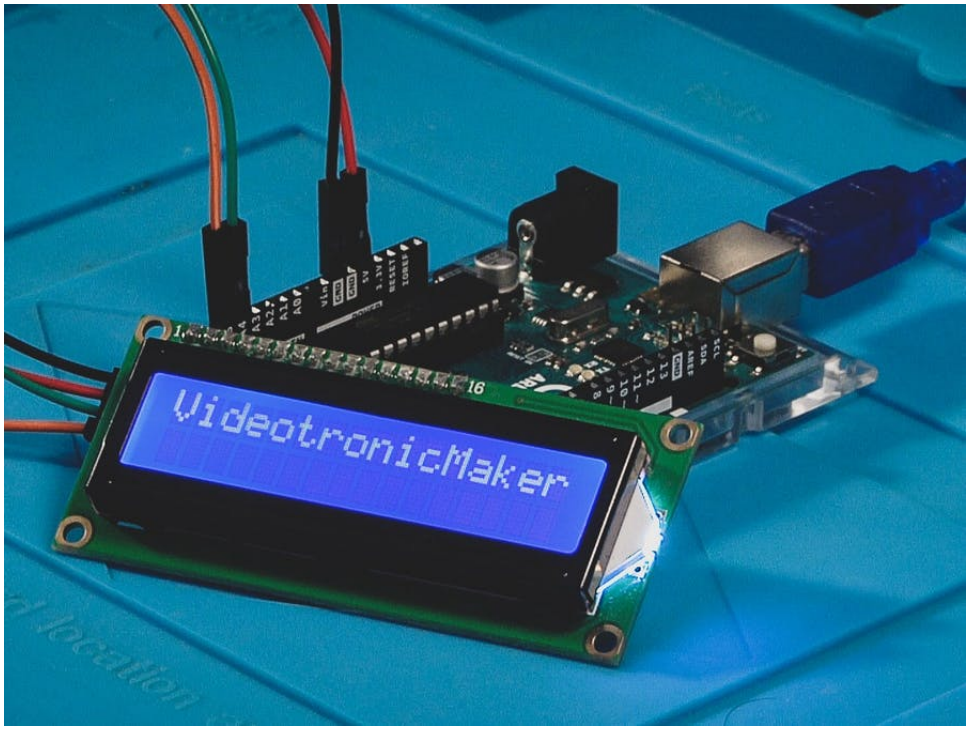
Instituto Superior de Engenharia de Lisboa

Licenciatura em Engenharia Informática e Multimédia

Computação Física - CF - 2122SV

Trabalho Prático 3



Docente Jorge Pais
Realizado por (Grupo 10):
Roman Ishchuk 43498
Pedro Silva 48965
Cláudia Sequeira 49247

28 de junho de 2022

Conteúdo

Lista de Figuras

1 Introdução

Neste trabalho vamos abordar o tema da comunicação série entre o Arduino e vários sensores inteligentes. No nosso caso, estabelecemos a comunicação entre o Arduino, um *display* 16x2 com protocolo I2C, e uma placa de sensores BMP180 e GY-89.

2 Objetivos

Um dos grandes objetivos deste trabalho é estabelecer a comunicação contínua entre o sensor, Arduino, *display*, e finalmente, para uma interface gráfica *Python*. As leituras efetuadas são a temperatura, a pressão, e posteriormente o cálculo da altitude. A cada 10 segundos, a informação irá atualizar-se no *display* e na interface gráfica.

3 Desenvolvimento - Comunicação entre sensor, *display* e Arduino

3.1 Esquema de ligações

Para desenvolver a nossa estação meteorológica todas as ligações utilizam a comunicação I2C para comunicar com o Arduino (ver figura 2 para o diagrama de ligações).

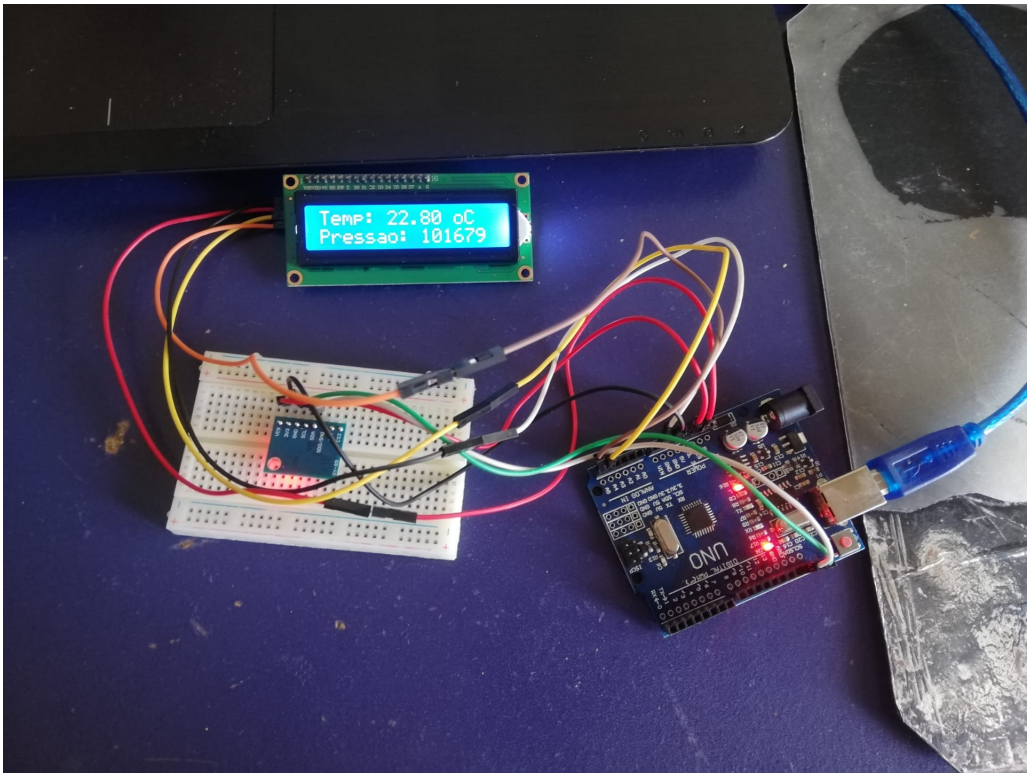


Figura 1: Diagrama de Ligações - Montagem

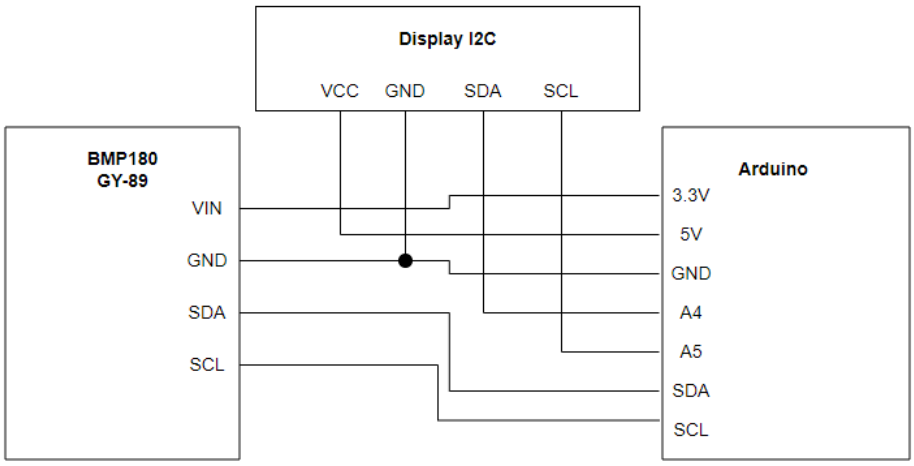


Figura 2: Diagrama de Ligações

3.2 Autómatos

A comunicação contínua com a atualização a cada 10 segundos é possível através de máquinas de estado. No nosso projeto implementamos 3 maquinas de estados: app(), sensorTemperatura() e sensorPressao().

A máquina de estado principal [app()] (ver figura 3) vai chamar todos os métodos auxiliares necessários, e muda os estados das outras máquina de estados

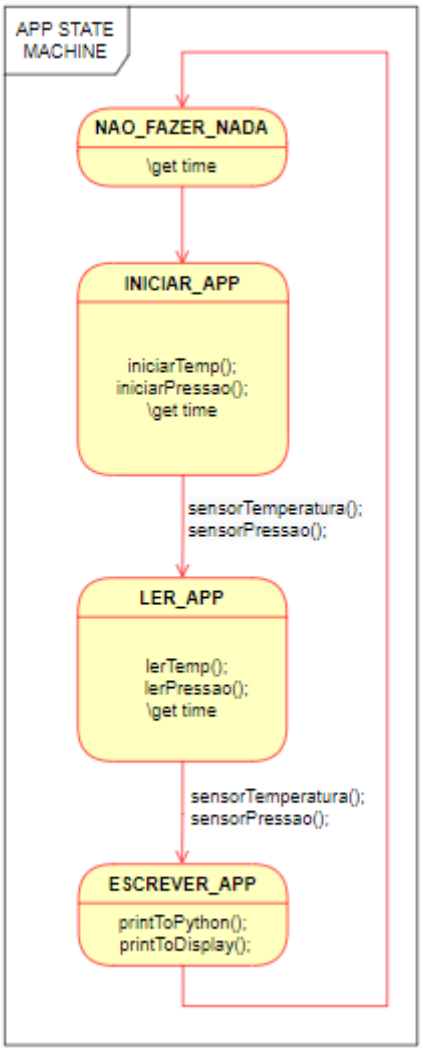


Figura 3: Máquina de estados app

O estado NAO_FAZER_NADA simplesmente vai esperar o tempo necessário antes de passar para o estado seguinte. O estado INICIAR_APP inicia os estados INICIAR_ST e INICIAR_SP, e espera o tempo necessário. O estado LER_APP inicia os estados LER_ST (ver figura 4) e LER_SP (ver figura 5), e espera o tempo necessário. O ultimo estado do ciclo é o estado ESCREVER_APP faz print para o *display* e para o *Python*.

No estado INICIAR_ST, é efetuado a leitura dos registos. No estado LER_ST, é efetuado os cálculos da UT e os cálculos da temperatura real.

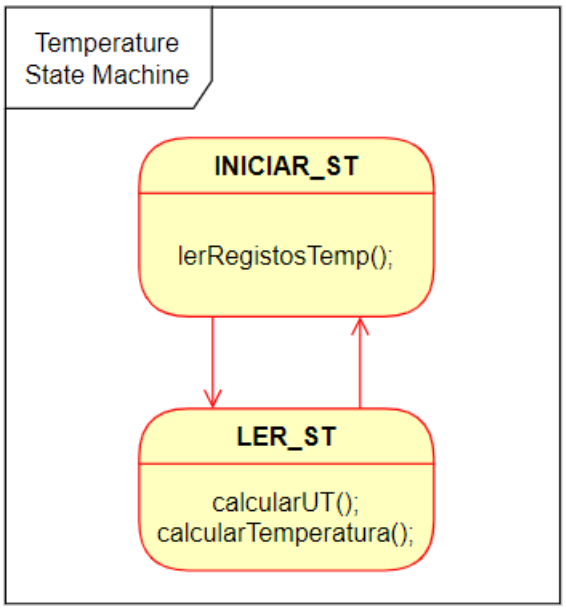


Figura 4: Máquina de estados da temperatura

No estado INICIAR_SP, é efetuado a leitura dos registos. No estado LER_SP, é efetuado os cálculos da UP e os cálculos da pressão real. Finalmente, são executados os cálculos da Altitude

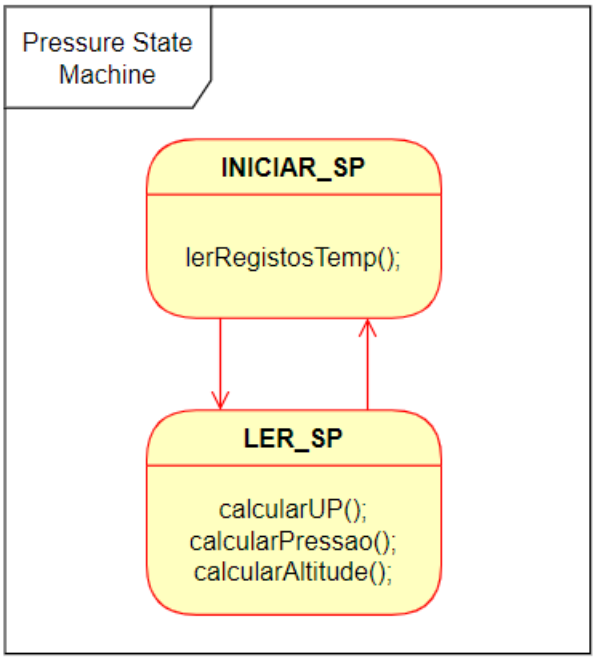


Figura 5: Máquina de estados da pressão

4 Interface Gráfica em *Python*

Nesta parte do projeto temos como objetivo desenhar um painel do tipo *outdoor* em *Python* que permita apresentar a informação obtida através do microprocessador Arduino. O conteúdo deve ser atualizado de 10 em 10 segundos.

4.1 Planeamento e aparência da interface gráfica

Antes de passarmos à implementação da interface gráfica começamos por realizar um rascunho do aspeto do nosso *outdoor* meteorológico. Queríamos incluir elementos visuais que complementassem a informação apresentada (ver figura 6).

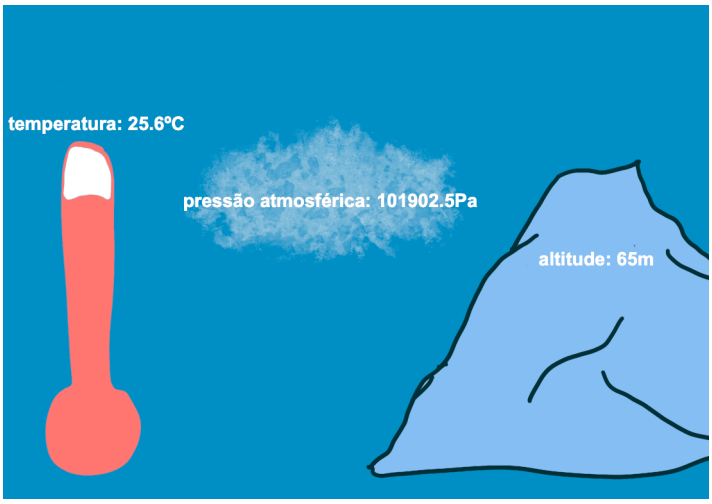


Figura 6: Rascunho desenvolvido no planeamento da Interface Gráfica

Quando decidimos o aspeto pretendido passamos à pesquisa de imagens que nos auxiliassem a realizar a nossa interface gráfica. Na figura 7 que se segue está a primeira versão que realizámos. Para fazer uma interface gráfica utilizámos o módulo *pygame*.

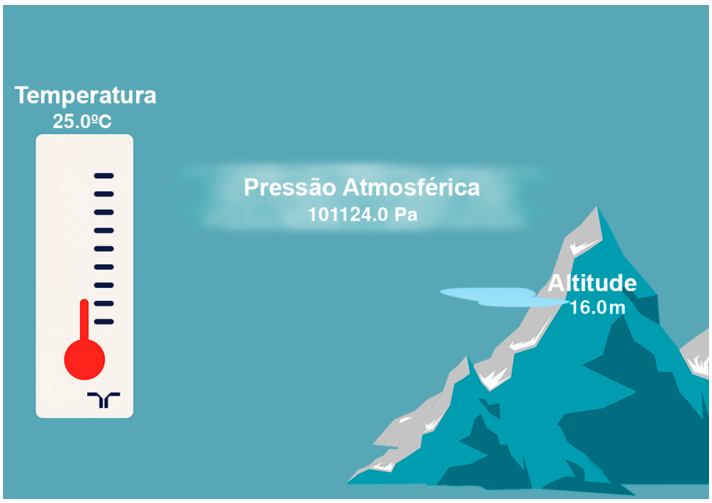


Figura 7: Primeira interface desenvolvida em *Python*

Ao apresentar esta versão ao docente foi nos aconselhado a repensar a utilização das cores escolhidas pois, como o *placard* desenvolvido é um *outdoor*, a utilização de cores análogas à cor do céu faria com que a nossa interface pudesse passar despercebida. Concordámos que isso era um problema da interface.

Para resolver o nosso problema realizámos uma pesquisa sobre a teoria de cor, de modo a que o nosso *outdoor* sobressaísse na paisagem pesquisamos sobre cores complementares. Optámos por utilizar cor-de-laranja como cor de fundo pois é a cor que melhor contrasta com o tom de azul da montanha/céu e funciona bem com o vermelho do termómetro.

Alterámos também a escala da temperatura e incluímos também informação sobre a data e hora atual (ver figura 8).

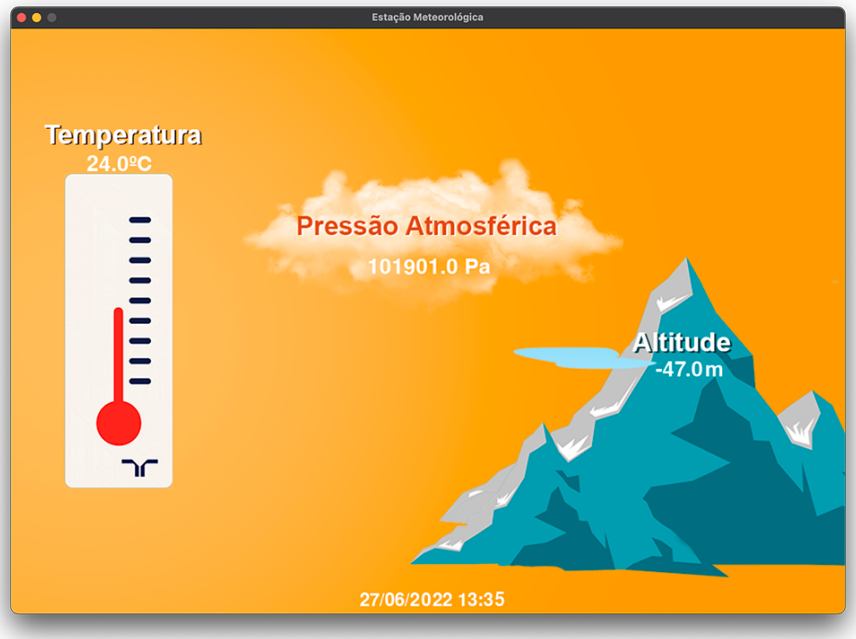


Figura 8: Interface Gráfica da Estação Meteorológica

A imagem que representa o termómetro é atualizada consoante a temperatura lida. Para fazer isso utilizámos 26 imagens, todas ligeiramente diferentes e quando a temperatura é recebida no nosso script de *Python* fazemos uma regra de três simples de modo a que fosse associada a um dos 26 valores. Com esse valor criamos a string que utilizamos para fazer *load* da imagem (ver figura 9).

```
# colocamos o background nas coordenadas 0 0
janela.blit(background, (0, 0))

# transforma a temperatura (0° a 40) num numero inteiro (0 a 15)
# para depois abrir a imagem correspondente desse valor
tempImagem = int((temperatura * 25) / 40)

# se a temperatura for menor que 0 ou maior que 40 utilizamos as imagens mais próximas
if tempImagem < 0:
    tempImagem = 0
elif tempImagem > 25:
    tempImagem = 25

# criamos a string que contém o nome do ficheiro a abrir, consoante a temperatura atual
string = "images/Layer " + str(tempImagem) + ".png"

# carregamos a imagem
imagem = pygame.image.load(string)
```

Figura 9: Código que exemplifica o carregamento da imagem do termómetro

4.2 Apresentação e descrição da comunicação série entre o Arduino e o computador

Para fazer a comunicação entre o Arduino e o computador (*Python*) foi utilizado o módulo *pySerial*. Permite que, quando é feito um *Serial.print* no Arduino o *Python* tenha acesso a esses dados. Como queremos receber três valores diferentes tivemos de arranjar uma maneira de distinguir a informação. Foi preciso programar o Arduino de modo a fazer os *prints* da informação codificada.

Optámos por envolver os valores com um carácter que os diferencia e que cada *print* seja feito numa linha separada. O método *printToPython* (ver figura 10) é chamado no estado ESCREVER_APP.

```
void printToPython(){
  Serial.print("T");Serial.print(t*0.1);Serial.println("T");
  Serial.print("P");Serial.print(p);Serial.println("P");
  Serial.print("A");Serial.print(a);Serial.println("A");
}
```

Figura 10: Código *print* do método *printToPython*

Quando recebemos os dados no *Python* utilizámos o método *split()* que nos permite dividir cada linha lida com um separador. Para fazer o *split()* utilizámos os separadores 'P' para pressão, o 'T' para temperatura e o 'A' para a altitude. É retornado um *array* e a informação que nós queremos está no índice 1.

A condição que separa os três tipos de informação encontra-se dentro de um *try/except* de modo a que, caso haja um problema na distinção de dados, o programa não encerre. Caso isso aconteça ele vai manter o valor anterior e faz um *print* na consola a informar que houve um problema a obter os valores. A leitura da linha está incluída no ciclo do *pygame*, a informação é atualizada a cada 10 segundos no Arduino.

```
while True:
    # .decode() converte byte string para uma string
    linha = lineReceive(Serie).decode()

    try:
        if 'T' in linha:
            leitura = linha.split('T', 2)
            temperatura = float(leitura[1])

        elif 'P' in linha:
            leitura = linha.split('P', 2)
            pressure = float(leitura[1])

        elif 'A' in linha:
            leitura = linha.split('A', 2)
            altitude = float(leitura[1])
    except:
        # caso haja um problema a obter os valores
        # é feito um print na consola e são mantidos
        # os valores anterioes
        print('Erro a obter os valores')
```

Figura 11: Código de leitura dos dados recebidos no *Python*

5 Conclusões

A realização deste trabalho permitiu-nos aprofundar o nosso conhecimento sobre a utilização do Arduino com sensores. Lidámos pela primeira vez com sensores de interface digital, designados também por sensores "inteligentes", e aprendemos como inicializá-los. Foi necessário conceber métodos de simulação de leitura dos registos para testar o *software* desenvolvido, de modo a garantir o correto funcionamento do dispositivo.

Adquirimos conhecimento sobre o Protocolo I2C utilizado para fazer a comunicação em série através de uma trama. A comunicação é feita utilizando o *SDA (Serial Data)* e o *SCLK (Serial Clock)*, sendo possível através dessas duas ligações comunicar com vários sensores ao mesmo tempo.

Aprendemos sobre a comunicação do Arduino com o computador, no nosso caso com o *Python*. Para implementar a nossa interface gráfica criámos o nosso próprio protocolo de comunicação de modo a diferenciar entre os vários tipos de informação recebidos no *Python* e podemos colocar a informação no sitio certo.

Aprofundámos também o conhecimento adquirido ma UC de SA (Sensores e Atuadores), que são as máquinas de estados. Foi utilizado este tipo de programação de modo a manter uma comunicação contínua entre os vários componentes, com atualização a cada 10 segundos.

6 Bibliografia

1. Folhas de Computação Física - Jorge Pais, 2021/22

7 Código Arduino

```
#include <Wire.h>
# define ENDERECO 0x77
#define RS 0x01
#define RW 0x02
#define EN 0x04
#define LUZ 0x08
#define ENDERECO_DISPLAY 0x27
#define INICIAR_APP 1
#define LER_APP 2
#define INICIAR_ST 3
#define LER_ST 4
#define INICIAR_SP 5
#define LER_SP 6
#define NAO_FAZER_NADA 0
#define ESCREVER_APP 7

short s_ac1, s_ac2, s_ac3, s_b1, s_b2, s_mb, s_mc, s_md;
long ac1, ac2, ac3, b1, b2, mb, mc, md;
short oss = 0;
unsigned short s_ac4, s_ac5, s_ac6;
unsigned long ac4, ac5, ac6;
unsigned long b4, b7;
long ut, up, x1, x2, b5, t, b6, x3, b3, p, a;
static const int ESPERAR = 0, LER = 1;
static int estado_st, estado_app, estado_sp;


void setup(){
  Serial.begin(9600);
  Wire.begin();
  iniciaDisplay();
  estado_st = NAO_FAZER_NADA;
  estado_sp = NAO_FAZER_NADA;
  estado_app = INICIAR_APP;
}

void loop(){
  app();
}
/**
 * Máquina de estados principal.
 * Esta máquina de estados vai mudar os estados das duas
 * máquinas de estados ( sensorTemperatura() e
 * sensorPressao()).
 */
void app(){
  static unsigned long t,t0=millis();
  switch(estado_app){
```

```
    case NAO_FAZER_NADA:
    t=millis();
      if((t-t0)>(3000)){
      estado_app = INICIAR_APP;
      t0=millis();
      }
    break;
    case INICIAR_APP:
     t=millis();
      if((t-t0)>(3000)){
      iniciarTemp();
      iniciarPressao();
      estado_app = LER_APP;
      sensorTemperatura();
      sensorPressao();
      t0=millis();
      }
    break;
    case LER_APP:
    t=millis();
      if((t-t0)>(4000)){
      lerTemp();
      lerPressao();
      estado_app = ESCREVER_APP;
      sensorTemperatura();
      sensorPressao();
      t0=millis();
      }
    break;
    case ESCREVER_APP:
      printToPython();
      printToDisplay();
      estado_app = NAO_FAZER_NADA;
    break;
}
}

/**
 * Métodos auxiliares, que mudam os estados.
 */
void iniciarTemp(){
  estado_st = INICIAR_ST;
}

void lerTemp(){
  estado_st = LER_ST;
}

void iniciarPressao(){
  estado_sp = INICIAR_SP;
```

```
}

void lerPressao(){
  estado_sp = LER_SP;
}


/**
 * máquina de estados da temperatura
 */
void sensorTemperatura(){
  switch(estado_st){
      case INICIAR_ST:
      lerRegistosTemp();
      break;
      case LER_ST:
        calcularUT();
        calcularTemperatura();
      break;
}
}

/**
 * máquina de estados da pressao e altitude
 */
void sensorPressao(){
  switch(estado_sp){
      case INICIAR_SP:
      lerRegistosTemp();
      break;
      case LER_SP:
        calcularUP();
        calcularPressao();
        calcularAltitude();
      break;
    }
  }

/**
 * Leitura dos registos de temp e pressão.
 */
void lerRegistosTemp(){

  s_ac1= readRegisto16bits(ENDERECO, 0xAA);
  ac1 = (long) s_ac1;
  s_ac2 = readRegisto16bits(ENDERECO, 0xAC);
  ac2 = (long) s_ac2;
  s_ac3 = readRegisto16bits(ENDERECO, 0xAE);
  ac3  = (long) s_ac3;
  s_ac4 = readRegisto16bits(ENDERECO, 0xB0);
```

```
  ac4 = (long) s_ac4;
  s_ac5 = readRegisto16bits(ENDERECO, 0xB2);
  ac5 = (long) s_ac5;
  s_ac6 = readRegisto16bits(ENDERECO, 0xB4);
  ac6 = (long) s_ac6;
  s_b1  = readRegisto16bits(ENDERECO, 0xB6);
  b1 = (long) s_b1;
  s_b2  = readRegisto16bits(ENDERECO, 0xB8);
  b2 = (long) s_b2;
  s_mb  = readRegisto16bits(ENDERECO, 0xBA);
  mb = (long) s_mb;
  s_mc  = readRegisto16bits(ENDERECO, 0xBC);
  mc = (long) s_mc;
  s_md  = readRegisto16bits(ENDERECO, 0xBE);
  md = (long) s_md;
  }

/**
 * Escrever registo 8 bit
 */
void writeReg8bits(byte endr, byte reg){
  Wire.beginTransmission(ENDERECO);
  Wire.write(reg);
  Wire.write(endr);
  Wire.endTransmission();

  }

/**
 * cálculo de UT
 */
void calcularUT(){
  writeReg8bits(0x2E, 0xF4);
  delay(4.5);

  byte MSB = readRegisto8bits(ENDERECO, 0xF6);
  byte LSB = readRegisto8bits(ENDERECO, 0xF7);

  ut = (MSB << 8) | LSB;
  }
/**
* cálculo de UP
*/
void calcularUP(){
  writeReg8bits(0x34|(oss<<6), 0xF4);
  delay(4.5);

  byte MSB = readRegisto8bits(ENDERECO, 0xF6);
  byte LSB = readRegisto8bits(ENDERECO, 0xF7);
  byte XLSB = readRegisto8bits(ENDERECO, 0xF8);
```

```
  up = (((long)MSB << 16) | (long)LSB << 8| (long)XLSB) >> (8-oss);

  }

/**
 * Cálculo da temperatura real
 */
void calcularTemperatura(){
  x1 = (ut - ac6) * ac5 >> 15;
  x2 = (mc << 11) / (x1 + md);
  b5 = x1 + x2;
  t = (b5 + 8) >> 4;
}
/**
 * Cálculo da pressão real
 */
void calcularPressao(){
  b6 = b5-4000;
  x1 = (b2 * (b6 * b6 >> 12))>>11;
  x2 = ac2 * b6 >>11;
  x3 = x2 + x1;
  b3 = (((ac1 * 4 + x3) << oss ) + 2) /4;
  x1 = ac3 * b6>>13;
  x2 = (b1 * (b6 * b6 >> 12))>>16;
  x3 = ((x1 + x2) + 2) >> 2;
  b4 =  ac4 * (x3 + 32768) >> 15;
  b7 = (up - b3) * (50000 >> oss);
  if (b7 < 0x80000000) {
    p = (b7*2)/b4;
    }else {
      p=(b7/b4)*2;
      }
   x1 = (p>>8) * (p>>8);
   x1 = (x1*3038) >> 16;
   x2 = (-7357 * p) >> 16;
   p += (x1+x2+3791) >> 4;
  }


/**
 * cálculo da amplitude
 */
void calcularAltitude(){
  float p0 = 101325;
  float base = p/p0;
  float expoente = pow(5.255,-1);
  a = 44330 * (1 -pow(base,expoente));
 }
```

```
/**
 * leitura do registo 8 bits
 */
byte readRegisto8bits(byte addressI2C, byte addressReg){
  Wire.beginTransmission(addressI2C);
  Wire.write(addressReg);
  Wire.endTransmission();
  Wire.requestFrom(addressI2C, (byte) 1);
  byte valor= Wire.read();
  Wire.endTransmission();
  return valor;
}
/**
 * leitura do registo 16 bits
 */
short readRegisto16bits(byte addressI2C,  byte addressReg){

  short high = readRegisto8bits(addressI2C, addressReg);
  short low  = readRegisto8bits(addressI2C, addressReg + 1);


  return (high << 8) | low;
}
/**
 * Método que faz print das saídas
 */
void printToPython(){

  Serial.print("T");Serial.print(t*0.1);Serial.print("T");
  Serial.print("P");Serial.print(p);Serial.print("P");
  Serial.print("A");Serial.print(a);Serial.print("A");
}

/**
 * Escrever Dados 4 bits no display
 */
void escreverDados4(byte quatroBits) {
  Wire.beginTransmission(ENDERECO_DISPLAY);
  Wire.write((quatroBits << 4) | LUZ| RS);
  Wire.endTransmission();
  Wire.beginTransmission(ENDERECO_DISPLAY);
  Wire.write((quatroBits << 4) | LUZ | RS | EN );
  Wire.endTransmission();
  delayMicroseconds(1); // enable ativo >450ns
  Wire.beginTransmission(ENDERECO_DISPLAY);
  Wire.write((quatroBits << 4) | LUZ | RS);
  Wire.endTransmission();
  delayMicroseconds(40); // tempo > 37us para comando fazer efeito}
}
```

```
/**
 * Escrever dados 8 bits no display
 */
void escreverDados8(byte oitoBits){
    escreverDados4((oitoBits >> 4) & 0xF);
    escreverDados4(oitoBits & 0xF);

  }
/**
 * Encontra todos os sensores com o protocolo I2C
 */
  void findI2CDevices() {
  const int MAX= 128; byte endereco[MAX]; byte identificador[MAX];
  int nDevices= 0;
  for (int i= 0 ; i<MAX ; ++i) {
    identificador[nDevices]= readRegisto8bits((byte)i, 0x0F); // 0x0F { registo que
    if (identificador[nDevices]!= 255)
      endereco[nDevices++]= (byte) i;
  }
  Serial.print("Detetados "); Serial.print(nDevices); Serial.println(" dispositivos
  Serial.println("Endereço - Identificador");
  for (int i= 0 ; i <nDevices ; ++i) {
    Serial.print(endereco[i], 16);
    Serial.print("h - ");
    Serial.print(identificador[i], 16);
    Serial.println("h");
  }
 }

/**
 * Escrever comandos 4 bits no display
 */
void escreverComandos4(byte quatroBits) {
  Wire.beginTransmission(ENDERECO_DISPLAY);
  Wire.write((quatroBits << 4) | LUZ);
  Wire.endTransmission();
  Wire.beginTransmission(ENDERECO_DISPLAY);
  Wire.write((quatroBits << 4) | LUZ | EN);
  Wire.endTransmission();
  delayMicroseconds(10); // enable ativo >450ns
  Wire.beginTransmission(ENDERECO_DISPLAY);
  Wire.write((quatroBits << 4) | LUZ);
  Wire.endTransmission();
  delayMicroseconds(400); // tempo > 37us para comando fazer efeito}
}
```

```
/**
 * Escrever comandos 8 bits no display
 */
void escreverComandos8(byte oitoBits){
    escreverComandos4((oitoBits >> 4) & 0xF);
    escreverComandos4(oitoBits & 0xF);
  }

/**
 * Mete cursor numa determinada posição no display
 */
void setCursorAdress(int linha, int coluna){
    escreverComandos8(0x80 | linha << 6 | coluna);
    delayMicroseconds(150);
  }
/**
 * imprime um caracter no display
 */
void printChar(char c){
  escreverDados8(c);
 }

/**
 * Imprime uma string no display
 */
void printString(String s){
  for(int i=0; s[i] != '\0';i++){
    printChar(s[i]);

    }
}
/**
 * Limpa o display
 */
void clearDisplay(){
  escreverComandos4(0x0);
  escreverComandos4(0x1);
  delay(5);
  }

/**
 * Inicia display
 */
void iniciaDisplay(){
  display4bits();
  display2linhas();
  dispCursorBlink();
  inicializaCursor();
}
```

```
/**
 * Inicia o display a 4 bits
 */
void display4bits(){
  delay(60);
  escreverComandos4(0x3);
  delay(7);
  escreverComandos4(0x3);
  delayMicroseconds(250);
  escreverComandos4(0x3);
  escreverComandos4(0x2);
 }
/**
 * inicia o display a 2 linhas
 */
 void display2linhas(){
  escreverComandos4(0x2);
  escreverComandos4(0x8);
  delayMicroseconds(150);
  }

/**
 * Display on, cursor on, blink = true
 */
void dispCursorBlink(){
  escreverComandos4(0x0);
  escreverComandos4(0xF);
  delayMicroseconds(150);
  }

/**
 * inicializa o cursor no display
 */
void inicializaCursor(){
  escreverComandos4(0x8);
  escreverComandos4(0x0);
  delayMicroseconds(150);
  }
/**
 * Shift left no display
 */
void shiftLeft(){
  escreverComandos4(0x1);
  escreverComandos4(0x8);
  delayMicroseconds(150);
}
```

```
/**
 * Imprime as saídas do programa no display
 */
void printToDisplay(){
  setCursorAdress(0,0);
  float aux = (float)t;
  printString("Temp: "+ (String)(aux/10) + " C" );
  setCursorAdress(1,0);
  String aux2 = "Pressao: " + (String) p + " Pa; Altitude: "+ (String)a + " m";
  printString(aux2);
  int i;
  for(i = 0; i < 20; i++){
  shiftLeft();
  delay(300);
  }
}
```

8 Código Python

```
import sys
import pygame
from pygame.locals import *
import serial
import time
from datetime import datetime

com = '/dev/cu.usbmodem1101'
baudrate = 9600
pressure=0
temperatura=0
altitude=0

def comInit(com, baudrate):
    try:
        Serie = serial.Serial(com, baudrate, timeout=0)
        time.sleep(2)  # wait for arduino reset
        print('Sucesso na ligação ao arduino.')
        return Serie
    except:
        print('Insucesso na ligação ao arduino.')
        return None

def lineReceive(Serie):
    try:
        linha = Serie.readline()
        return linha
    except:
        print('Erro na comunicação.')
        Serie.close()

comInit(com, baudrate)
Serie = serial.Serial(com, baudrate, timeout=0)
time.sleep(2) # esperar pelo arduino

# inicialização do módulo pygame
pygame.init()


# criação de uma janela com o tamanho da imagem
largura = 1100
altura = 770
tamanho = (largura, altura)
janela = pygame.display.set_mode(tamanho)
pygame.display.set_caption('Estação Meteorológica')
background = pygame.image.load("images/background2.png")

# formatar os diferentes tipos de texto
```

```
def texto(pos, txt, color, f_size, bg_color):
    font = pygame.font.Font(None, f_size)
    text = font.render(txt, True, color, bg_color)
    janela.blit(text, pos)  # blit inserir imagens sobre outras

# número de imagens por segundo
frame_rate = 30

# relógio para controlo do frame rate
clock = pygame.time.Clock()

# Definimos a fonte que vamos ter
font_size = 25
font = pygame.font.Font(None, font_size)
antialias = True
WHITE = (255, 255, 255)


while True:
    # .decode() converte byte string para uma string
    linha = lineReceive(Serie).decode()

    try:
        if 'T' in linha:
            leitura = linha.split('T', 2)
            temperatura = float(leitura[1])

        elif 'P' in linha:
            leitura = linha.split('P', 2)
            pressure = float(leitura[1])

        elif 'A' in linha:
            leitura = linha.split('A', 2)
            altitude = float(leitura[1])
    except:
        # caso haja um problema a obter os valores
        # é feito um print na consola e são mantidos
        # os valores anterioes
        print('Erro a obter os valores')


    time.sleep(1)  # esperar pelo arduino


    # colocamos o background nas coordenadas 0 0
    janela.blit(background, (0, 0))

    # regra de 3 simples que transforma a temperatura (0º a 40) num
    # numero inteiro, para depois abrir a imagem correspondente desse valor
    tempImagem = int((temperatura * 25) / 40)
```

```
# se a temperatura for menor que 0 ou maior que 40
# utilizamos as imagens mais próximas
if tempImagem < 0:
    tempImagem = 0
elif tempImagem > 40:
    tempImagem = 40

# criamos a string que contém o nome do ficheiro a abrir,
# consoante a temperatura atual
string = "images/Layer " + str(tempImagem) + ".png"

# carregamos a imagem
imagem = pygame.image.load(string)


# TEMPERTATURA - Imagem
janela.blit(imagem, (70,190))

# TEMPERTATURA - Texto
texto((100,165), str(temperatura) + 'ºC', WHITE, 42, None)

# PRESSÃO - Texto
texto((largura/2 - 80, 300), str(pressure) + ' Pa', WHITE, 42, None)

# ALTITUDE - Texto
texto((largura/2 + 300, altura/2 + 50), str(altitude) + 'm', WHITE, 42, None)

# DATA E HORA ATUAIS - Texto
presente = datetime.now()
data_hora = presente.strftime("%d/%m/%Y %H:%M")
texto((largura/2 - 90 ,740), str(data_hora), WHITE, 36, None)

#update do display
pygame.display.update()

for event in pygame.event.get():
    if event.type == QUIT:
        pygame.quit()
        sys.exit()
```