

Licenciatura Engenharia Informática e Multimédia

Computação Física – CF

Semestre de Verão 2021 / 2022

Relatório Trabalho Prático 3

Docente Carlos Carvalho

 Trabalho realizado por:

 Fábio Dias, nº 42921

Índice

[1. Objetivo 5](#_Toc107013672)

[2. Desenvolvimento 6](#_Toc107013673)

[3. Protocolo I2C 8](#_Toc107013674)

[4. Comunicação Série 9](#_Toc107013675)

[5. Montagem 10](#_Toc107013676)

[6. Resultados 14](#_Toc107013677)

[7. Implementação 16](#_Toc107013678)

[7.1. Arduino 16](#_Toc107013679)

[7.2. Python 26](#_Toc107013680)

[8. Código 28](#_Toc107013681)

[8.1. Arduino 28](#_Toc107013682)

[8.2. Python 35](#_Toc107013683)

[9. Observações 36](#_Toc107013684)

[10. Conclusões 38](#_Toc107013685)

[11. Bibliografia 39](#_Toc107013686)

Índice de Figuras

[Figura 1 - Diagrama de Ligações 6](#_Toc107013640)

[Figura 2 - Diagrama de Actividades 7](#_Toc107013641)

[Figura 3 - Macro Diagrama de Actividades de Leitura e Apresentação de Valores 7](#_Toc107013642)

[Figura 4 - Montagem Geral 10](#_Toc107013643)

[Figura 5 - Montagem Arduino 11](#_Toc107013644)

[Figura 6 - Montagem Display 12](#_Toc107013645)

[Figura 7 - Montagem Placa IMU 10 DOF 13](#_Toc107013646)

[Figura 8 - Resultado Display 14](#_Toc107013647)

[Figura 9 - Resultado Consola em Python 15](#_Toc107013648)

[Figura 10 - Biblioteca Wire.h 16](#_Toc107013649)

[Figura 11 - Definição de Endereços 16](#_Toc107013650)

[Figura 12 - Variáveis Globais 16](#_Toc107013651)

[Figura 13 - Métodos do Display 17](#_Toc107013652)

[Figura 14 - Comandos do Display 18](#_Toc107013653)

[Figura 15 - Dados do Display 19](#_Toc107013654)

[Figura 16 - Métodos do Sensor BMP180 20](#_Toc107013655)

[Figura 17 - Read Calibration Data 21](#_Toc107013656)

[Figura 18 - Read Uncompensated Temperature Value 21](#_Toc107013657)

[Figura 19 - Read Uncompensated Pressure Value 21](#_Toc107013658)

[Figura 20 - Calculate True Temperature And Pressure 22](#_Toc107013659)

[Figura 21 - Calculate Altitude 22](#_Toc107013660)

[Figura 22 - Display Communication 23](#_Toc107013661)

[Figura 23 - Serial Communication 23](#_Toc107013662)

[Figura 24 - Setup 24](#_Toc107013663)

[Figura 25 - Loop 25](#_Toc107013664)

[Figura 26 - Importar Biblioteca Serial 26](#_Toc107013665)

[Figura 27 – Variáveis 26](#_Toc107013666)

[Figura 28 - Método de Inicialização da Comunicação Série 26](#_Toc107013667)

[Figura 29 - Método de Recepção de Informação 27](#_Toc107013668)

[Figura 30 - Loop do Programa Python 27](#_Toc107013669)

[Figura 31 - Alteração no método BMP180\_calculateTrueTemperatureAndPressure 36](#_Toc107013670)

[Figura 32 - Display Valores Corretos 37](#_Toc107013671)

# 1. Objetivo

Para este trabalho prático foi-nos pedido a simulação de uma estação meteorológica com informação da temperatura, pressão atmosférica e altitude. Para obter estas informações, recorremos ao sensor BMP180. Para as apresentar, é usado um *display* de duas linhas de dezasseis caracteres.

É nos também pedido para criarmos um *outdoor* que apresente a mesma informação.

Ambos estas exibições de informação devem ser atualizadas após 10 segundos.

# 2. Desenvolvimento

Dado que usaremos a placa IMU 10 DOF e o display LCD, que ambos usam um protocolo de comunicação I2C, podemos esquematizar o diagrama de ligações[1][2].

Para os pinos de Serial Data, SDA, estes vão ser ligados ao pino A4 do arduino[1][2]. Por sua vez, os pinos de Serial Clock, SCL, são ligados ao pino A5[1][2]. Ambos os Ground’s, GND, são ligados a um dos pinos GND do Arduino. A única diferença entre estas ligações é que a placa IMU 10 DOF é alimentada a 3.3 Volts enquanto o display LCD é alimentado a 5 Volts. (Ver Figura 1 – Diagrama de Ligações).

Embora usemos a placa IMU 10 DOF, vamos apenas usar o sensor integrado BMP180[1][2].

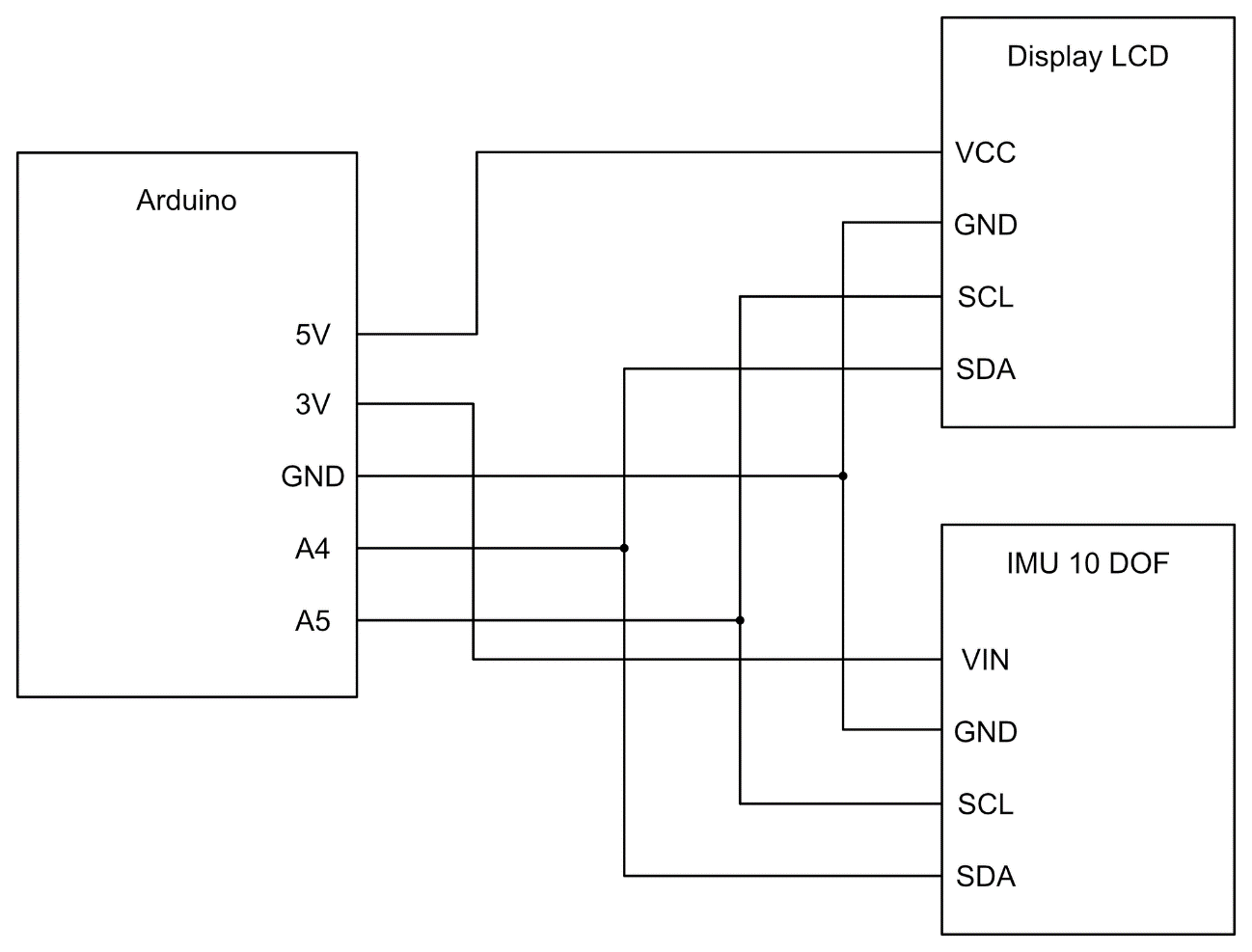


Figura 1 - Diagrama de Ligações

Como já referido, o objetivo é obter as medições da temperatura, pressão atmosférica e altitude, a cada dez segundos, e apresentar essa mesma informação já atualizada. Podemos recorrer assim a um diagrama de atividades onde teremos duas atividades: Leitura e Apresentação de Valores, que será uma só atividade, e a Esperar 10 Segundos.

A atividade Leitura e Apresentação de Valores é uma macro-atividade, ou seja, pode ser descrita por um diagrama de atividades mais elementar[1][2]. Este também possui duas atividades, Leitura de Valores e Escrita de Valores.

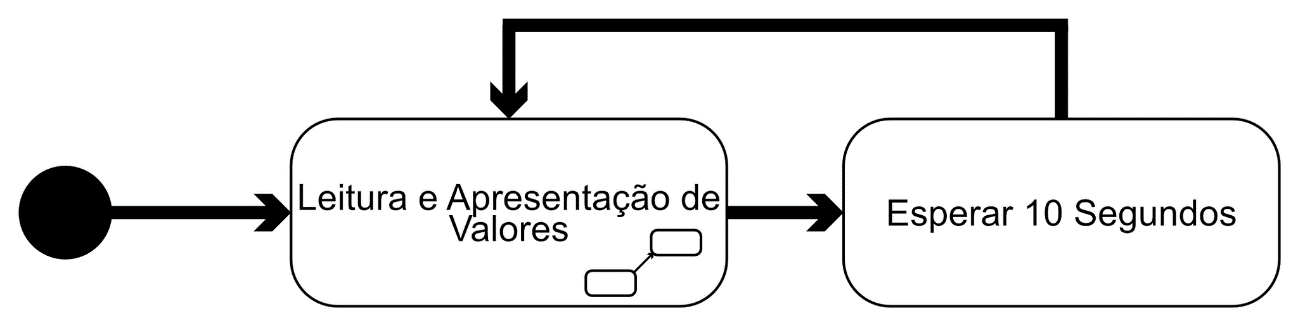


Figura 2 - Diagrama de Actividades

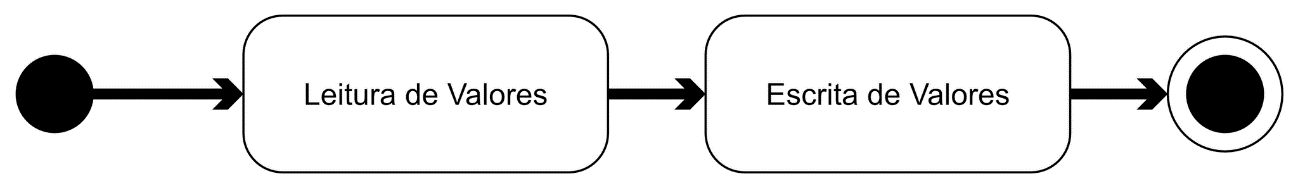


Figura 3 - Macro Diagrama de Actividades de Leitura e Apresentação de Valores

# 3. Protocolo I2C

O protocolo I2C é partilhado pela placa e pelo display. Este protocolo o envio de comandos e dados, assim como a leitura de dados[1][2].

No caso do display, enviamos comandos e dados enquanto a placa também devolve dados.

Para estas operações serem efetuadas, começamos por comunicar com um determinado endereço. Este é o endereço do componente em questão. De seguida enviamos ou um comando ou dados, que eventualmente serão processados como desejados. Para receber dados da placa, o mesmo processo é realizado, seguindo da leitura do que é recebido, caso realmente haja essa informação.

Usaremos a biblioteca Wire.h para facilitar a manipulação dos sinais SDA e SCL[1][2].

# 4. Comunicação Série

De forma a termos a simulação do outdoor, é necessária a comunicação série entre o Arduino e uma aplicação implementada em Python, com recurso ao uso da biblioteca Serial[1][2].

Desta forma, quando os valores da temperatura, pressão atmosférica e altitude são actualizados, é também enviado por comunicação série os mesmos valores. Estes serão lidos pela aplicação Python e depois apresentados na consola.

# 5. Montagem

Uma imagem com texto, eletrónica

Descrição gerada automaticamente

Figura 4 - Montagem Geral

Uma imagem com eletrónica, circuito

Descrição gerada automaticamente

Figura 5 - Montagem Arduino

Uma imagem com texto, interior

Descrição gerada automaticamente

Figura 6 - Montagem Display

Uma imagem com eletrónica

Descrição gerada automaticamente

Figura 7 - Montagem Placa IMU 10 DOF

# 6. Resultados

Uma imagem com texto, monitor, eletrónica

Descrição gerada automaticamente

Figura 8 - Resultado Display

Uma imagem com texto

Descrição gerada automaticamente

Figura 9 - Resultado Consola em Python

# 7. Implementação

## 7.1. Arduino

Como foi mencionado, é usado a biblioteca Wire.h.



Figura 10 - Biblioteca Wire.h

De seguida, definimos os endereços, tanto do display como do sensor BMP180, e ainda os endereços para o envio de comandos e de dados do display.

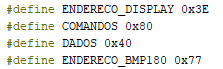


Figura 11 - Definição de Endereços

Também temos de definir as variáveis globais, tanto para calcular os valores de temperatura, pressão atmosférica e altitude, assim como o estado do autómato. Temos ainda as variáveis para guardar os tempos de espera e atualizar os valores acima referidos.

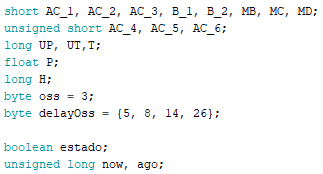


Figura 12 - Variáveis Globais

Começamos a definir os métodos que serviram para enviar comandos e dados para o display. Como referido, definimos a comunicação com o endereço do display, enviamos o endereço para comandos ou para dados, e de seguida a operação que queremos executar, ou dados a apresentar.

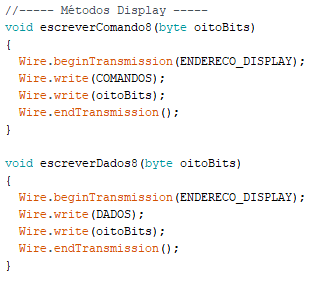


Figura 13 - Métodos do Display

Temos de definir cada comando que o display executa. Neste caso, temos o *displayInit* que inicializa o display; o *displayClear* que apaga tudo o que o display está a exibir; *displaySetCursor* que define onde queremos colocar o cursor; *displayCursorOffBlinkOff* para esconder o cursor e para este não piscar; *displayCursorOnBlinkOn* para mostrar o cursor e fazê-lo piscar; *displayCursorHome* que insere o cursor na primeira coluna da primeira linha.

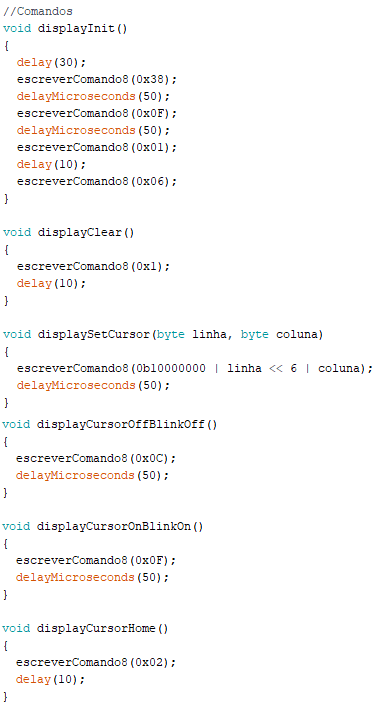


Figura 14 - Comandos do Display

Tal como para os comandos, também é necessário criar métodos que enviem dados para o display. Neste caso, o *displayPrintChar* que exibe um caracter no display; o *displayPrintString* que apresenta uma *string* no display.

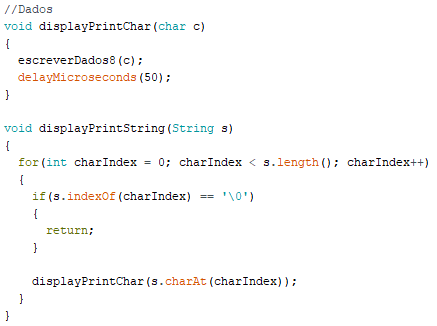


Figura 15 - Dados do Display

Tal como fizemos para o display, agora temos de repetir o processo para o sensor BMP180. Começamos por definir os métodos para a leitura e envio de dados.

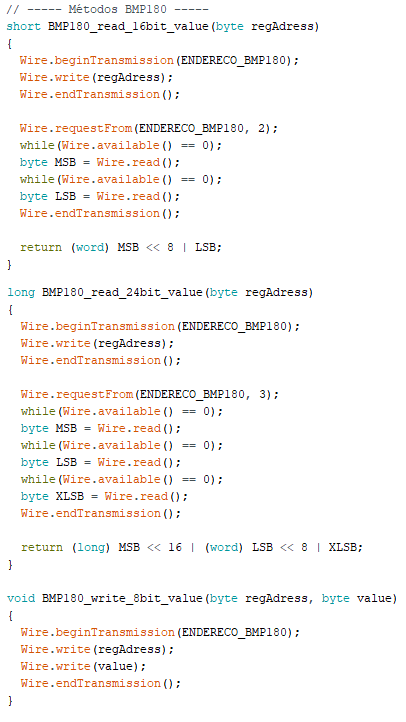


Figura 16 - Métodos do Sensor BMP180

Passamos para a implementação da leitura dos valores do sensor. O primeiro método é a leitura dos valores de calibração do sensor.

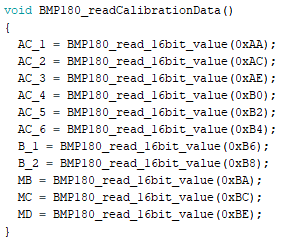


Figura 17 - Read Calibration Data

De seguida, obtemos o valor de temperatura antes de ser compensada.

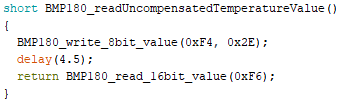


Figura 18 - Read Uncompensated Temperature Value

É feito o mesmo para o valor da pressão atmosférica.

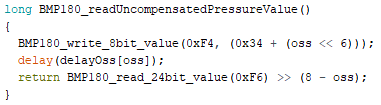


Figura 19 - Read Uncompensated Pressure Value

Após obtermos estes valores, vamos fazer as operações necessárias para obtermos os valores corretos da temperatura e pressão atmosférica.

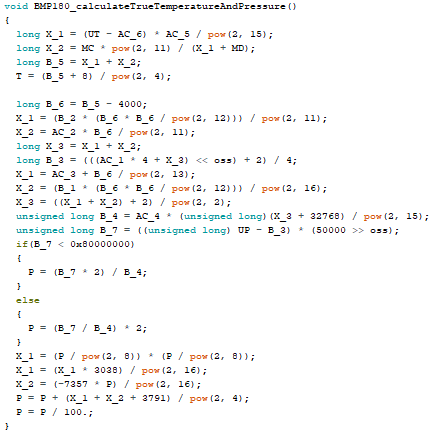


Figura 20 - Calculate True Temperature And Pressure

Por fim, obtemos o valor da altitude, a partir do valor da pressão atmosférica.

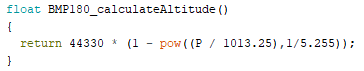


Figura 21 - Calculate Altitude

Como foi apresentado previamente, o objectivo é implementar um autómato onde obtém os valores actuais da temperatura, pressão atmosférica e altitude, apresenta estes valores e depois entra num estado suspenso durante dez segundos, repetindo depois o processo.

Para facilitar este objectivo, foram criados dois métodos, um que lê e exibe os valores, e outro que envia, via comunicação série, os mesmos valores.

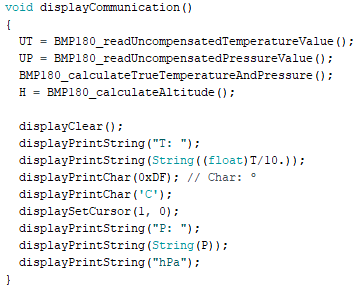


Figura 22 - Display Communication

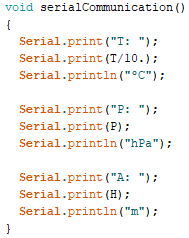


Figura 23 - Serial Communication

No nosso *setup*, iniciamos a comunicação série. O *baud rate* tem de ser o mesmo que na aplicação feita em Python. Inicializamos o arduino como o mestre no protocolo I2C. Inicializamos também o display e escondemos o cursor. Lemos os valores de calibração do sensor BMP180 e definimos o estado do autómato para verdadeiro, pois só vamos ter dois estados que podem ser diferenciados pelo valor booleano.

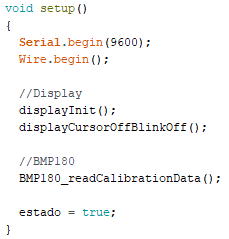


Figura 24 - Setup

No *loop* ocorre o autómato. Este verifica o estado actual, se for o correspondente à atividade Esperar, calculamos o tempo que passou desde a última atualização. Se esse intervalo de tempo for igual ou superior a dez segundos, o autómato evolui no sentido da atividade Leitura e Apresentação de Valores. Estes valores são atualizados, exibidos e enviados via comunicação série e atualizado o momento em que saímos desse estado. No final de cada uma destas atividades, o estado é atualizado para o complementado.

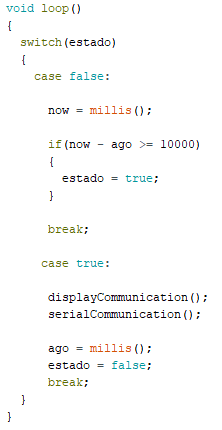


Figura 25 - Loop

## 7.2. Python

Como já referido anteriormente, para usarmos a comunicação série, temos de importar a biblioteca *serial* do Python.



Figura 26 - Importar Biblioteca Serial

Criamos duas variáveis que igualam a porta onde o Arduino está ligado e, como já mencionado, o *baud rate*.



Figura 27 – Variáveis

De seguida, implementamos o método que inicializa a comunicação série com o Arduino.

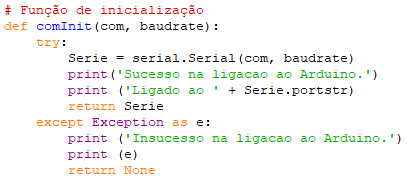


Figura 28 - Método de Inicialização da Comunicação Série

Definimos também o método que recebe informação pela comunicação Série.

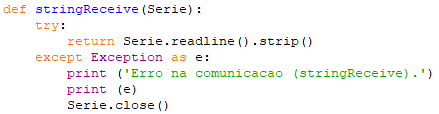


Figura 29 - Método de Recepção de Informação

Finalmente, temos o *loop* do programa que verifica se existe alguma informação a receber e, caso exista, exibe-a na consola da aplicação.

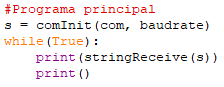


Figura 30 - Loop do Programa Python

# 8. Código

## 8.1. Arduino

#include <Wire.h>

#define ENDERECO\_DISPLAY 0x3E

#define COMANDOS 0x80

#define DADOS 0x40

#define ENDERECO\_BMP180 0x77

short AC\_1, AC\_2, AC\_3, B\_1, B\_2, MB, MC, MD;

unsigned short AC\_4, AC\_5, AC\_6;

long UP, UT,T;

float P;

long H;

byte oss = 3;

byte delayOss = {5, 8, 14, 26};

boolean estado;

unsigned long now, ago;

*//----- Métodos Display -----*

void escreverComando8(byte oitoBits)

{

Wire.beginTransmission(ENDERECO\_DISPLAY);

Wire.write(COMANDOS);

Wire.write(oitoBits);

Wire.endTransmission();

}

void escreverDados8(byte oitoBits)

{

Wire.beginTransmission(ENDERECO\_DISPLAY);

Wire.write(DADOS);

Wire.write(oitoBits);

Wire.endTransmission();

}

*//Comandos*

void displayInit()

{

delay(30);

escreverComando8(0x38);

delayMicroseconds(50);

escreverComando8(0x0F);

delayMicroseconds(50);

escreverComando8(0x01);

delay(10);

escreverComando8(0x06);

}

void displayClear()

{

escreverComando8(0x1);

delay(10);

}

void displaySetCursor(byte linha, byte coluna)

{

escreverComando8(0b10000000 | linha << 6 | coluna);

delayMicroseconds(50);

}

void displayCursorOffBlinkOff()

{

escreverComando8(0x0C);

delayMicroseconds(50);

}

void displayCursorOnBlinkOn()

{

escreverComando8(0x0F);

delayMicroseconds(50);

}

void displayCursorHome()

{

escreverComando8(0x02);

delay(10);

}

*//Dados*

void displayPrintChar(char c)

{

escreverDados8(c);

delayMicroseconds(50);

}

void displayPrintString(String s)

{

for(int charIndex = 0; charIndex < s.length(); charIndex++)

{

if(s.indexOf(charIndex) == '\0')

{

return;

}

displayPrintChar(s.charAt(charIndex));

}

}

*// ----- Métodos Display -----*

*// ----- Métodos BMP180 -----*

short BMP180\_read\_16bit\_value(byte regAdress)

{

Wire.beginTransmission(ENDERECO\_BMP180);

Wire.write(regAdress);

Wire.endTransmission();

Wire.requestFrom(ENDERECO\_BMP180, 2);

while(Wire.available() == 0);

byte MSB = Wire.read();

while(Wire.available() == 0);

byte LSB = Wire.read();

Wire.endTransmission();

return (word) MSB << 8 | LSB;

}

long BMP180\_read\_24bit\_value(byte regAdress)

{

Wire.beginTransmission(ENDERECO\_BMP180);

Wire.write(regAdress);

Wire.endTransmission();

Wire.requestFrom(ENDERECO\_BMP180, 3);

while(Wire.available() == 0);

byte MSB = Wire.read();

while(Wire.available() == 0);

byte LSB = Wire.read();

while(Wire.available() == 0);

byte XLSB = Wire.read();

Wire.endTransmission();

return (long) MSB << 16 | (word) LSB << 8 | XLSB;

}

void BMP180\_write\_8bit\_value(byte regAdress, byte value)

{

Wire.beginTransmission(ENDERECO\_BMP180);

Wire.write(regAdress);

Wire.write(value);

Wire.endTransmission();

}

void BMP180\_readCalibrationData()

{

AC\_1 = BMP180\_read\_16bit\_value(0xAA);

AC\_2 = BMP180\_read\_16bit\_value(0xAC);

AC\_3 = BMP180\_read\_16bit\_value(0xAE);

AC\_4 = BMP180\_read\_16bit\_value(0xB0);

AC\_5 = BMP180\_read\_16bit\_value(0xB2);

AC\_6 = BMP180\_read\_16bit\_value(0xB4);

B\_1 = BMP180\_read\_16bit\_value(0xB6);

B\_2 = BMP180\_read\_16bit\_value(0xB8);

MB = BMP180\_read\_16bit\_value(0xBA);

MC = BMP180\_read\_16bit\_value(0xBC);

MD = BMP180\_read\_16bit\_value(0xBE);

}

short BMP180\_readUncompensatedTemperatureValue()

{

BMP180\_write\_8bit\_value(0xF4, 0x2E);

delay(5);

return BMP180\_read\_16bit\_value(0xF6);

}

long BMP180\_readUncompensatedPressureValue()

{

BMP180\_write\_8bit\_value(0xF4, (0x34 + (oss << 6)));

delay(delayOss[oss]);

return BMP180\_read\_24bit\_value(0xF6) >> (8 - oss);

}

void BMP180\_calculateTrueTemperatureAndPressure()

{

long X\_1 = (UT - AC\_6) \* AC\_5 / pow(2, 15);

long X\_2 = MC \* pow(2, 11) / (X\_1 + MD);

long B\_5 = X\_1 + X\_2;

T = (B\_5 + 8) / pow(2, 4);

long B\_6 = B\_5 - 4000;

X\_1 = (B\_2 \* (B\_6 \* B\_6 / pow(2, 12))) / pow(2, 11);

X\_2 = AC\_2 \* B\_6 / pow(2, 11);

long X\_3 = X\_1 + X\_2;

long B\_3 = (((AC\_1 \* 4 + X\_3) << oss) + 2) / 4;

X\_1 = AC\_3 + B\_6 / pow(2, 13);

X\_2 = (B\_1 \* (B\_6 \* B\_6 / pow(2, 12))) / pow(2, 16);

X\_3 = ((X\_1 + X\_2) + 2) / pow(2, 2);

unsigned long B\_4 = AC\_4 \* (unsigned long)(X\_3 + 32768) / pow(2, 15);

unsigned long B\_7 = ((unsigned long) UP - B\_3) \* (50000 >> oss);

if(B\_7 < 0x80000000)

{

P = (B\_7 \* 2) / B\_4;

}

else

{

P = (B\_7 / B\_4) \* 2;

}

X\_1 = (P / pow(2, 8)) \* (P / pow(2, 8));

X\_1 = (X\_1 \* 3038) / pow(2, 16);

X\_2 = (-7357 \* P) / pow(2, 16);

P = P + (X\_1 + X\_2 + 3791) / pow(2, 4);

P = P / 100.;

}

float BMP180\_calculateAltitude()

{

return 44330 \* (1 - pow((P / 1013.25),1/5.255));

}

*// ----- Métodos BMP180 -----*

void setup()

{

Serial.begin(9600);

Wire.begin();

*//Display*

displayInit();

displayCursorOffBlinkOff();

*//BMP180*

BMP180\_readCalibrationData();

estado = true;

}

void loop()

{

switch(estado)

{

case false:

now = millis();

if(now - ago >= 10000)

{

estado = true;

}

break;

case true:

displayCommunication();

serialCommunication();

ago = millis();

estado = false;

break;

}

}

void serialCommunication()

{

Serial.print("T: ");

Serial.print(T/10.);

Serial.println("ºC");

Serial.print("P: ");

Serial.print(P);

Serial.println("hPa");

Serial.print("A: ");

Serial.print(H);

Serial.println("m");

}

void displayCommunication()

{

UT = BMP180\_readUncompensatedTemperatureValue();

UP = BMP180\_readUncompensatedPressureValue();

BMP180\_calculateTrueTemperatureAndPressure();

H = BMP180\_calculateAltitude();

displayClear();

displayPrintString("T:");

displayPrintString(String((float)T/10.));

displayPrintChar(0xDF); *// Char: º*

displayPrintChar('C');

displayPrintString(" A:");

displayPrintString(String(H));

displayPrintChar('m');

displaySetCursor(1, 0);

displayPrintString("P:");

displayPrintString(String(P));

displayPrintString("hPa");

}

## 8.2. Python

*# -\*- coding: cp1252 -\*-*

import serial

com = 'COM11' *# Tem de coincidir com o do Arduino*

baudrate = 9600 *# Tem de coincidir com o do Arduino*

*# Função de inicialização*

def comInit(com, baudrate):

try:

Serie = serial.Serial(com, baudrate)

print('Sucesso na ligacao ao Arduino.')

print ('Ligado ao ' + Serie.portstr)

return Serie

except Exception as e:

print ('Insucesso na ligacao ao Arduino.')

print (e)

return None

def stringReceive(Serie):

try:

return Serie.readline().strip()

except Exception as e:

print ('Erro na comunicacao (stringReceive).')

print (e)

Serie.close()

*#Programa principal*

s = comInit(com, baudrate)

while(True):

print(stringReceive(s))

print()

# 9. Observações

Após algumas observações, foi possível concluir que a placa IMU 10 DOF apresenta uma avaria pois os valores devolvidos da pressão atmosférica são extremamente altos, o que implicaria que estaríamos vários metros debaixo do nível do mar.

Dado isto, na aula foi combinada uma pequena solução que exibia os resultados corretos. Esta foi feita por uma pequena modificação no código, o que serviu para os valores ficarem corretos.



Figura 31 - Alteração no método BMP180\_calculateTrueTemperatureAndPressure



Figura 32 - Display Valores Corretos

# 10. Conclusões

Com este trabalho finalizado, foi possível aprofundar os conhecimentos relacionados com o protocolo I2C, assim como a comunicação série. Foi também possível experienciar como os display’s funcionam no nosso quotidiano assim como os *outdoors*.

# 11. Bibliografia

[1] Carlos Carvalho (2022). *Computação Física*.

[2] Jorge Pais (2022). *Computação Física*.