UNIVERSIDADE ESTADUAL DE CAMPINAS
Faculdade de Engenharia Elétrica e de Computação

IE323A - Tópicos em Eletrônica

Prof.Dr. Fabiano Fruett

Experimento Efeito Estufa e Aquecimento Global

Nomes: Victor Anthony Teixeira dos Santos - 206467

Campinas-SP , 2024

Sumário:

PARTE 1 - DOCUMENTAÇÃO TÉCNICA

- **1 - Resumo descritivo do projeto**
- **2 - Objetivo**
- **3 - Itens da BNCC contemplados (de 2 a 5)**
- **4 - Recursos de hardware (on e off-board) a serem utilizados**
  - **4.1 - BME680**
  - **4.2 - HC05**
- **5 - Descrição simplificada do software**
- **6 - Maior desafio do projeto**
- **7 - Código Completo Comentado**
  - **7.1 - Código Completo**
  - **7.2 - Explicação do Código**
  - **7.3 - Biblioteca BME680**
- **8 - Descrição do hardware (PCB Adaptada para Sensor Bluetooth)**
- **9 - Desenvolvimento Dashboard**
  - **9.1 - Desenvolvimento Interface**
  - **9.2 - Desenvolvimento Diagrama de Blocos**

PARTE 2 - MANUAL PARA PROFESSORES

- **1 - Tutorial BitDogLab**
  - **1.1 - Tutorial de uso inicial para BitDog**
  - **1.2 - Montagem Inicial Experimento**
- **2 - Montagem e realização do experimento**

Link Docs:
https://docs.google.com/document/d/13DsHIVBsn8pt8z_ySMauVp-LgChUZzobolKhmw013Gs/edit?usp=sharing

Link Apresentação:
https://docs.google.com/presentation/d/1zn_d1CRPbyjYiAEmj1wNgHPjXIvcOCrdGSiIc2Ldvhc/edit?usp=sharing

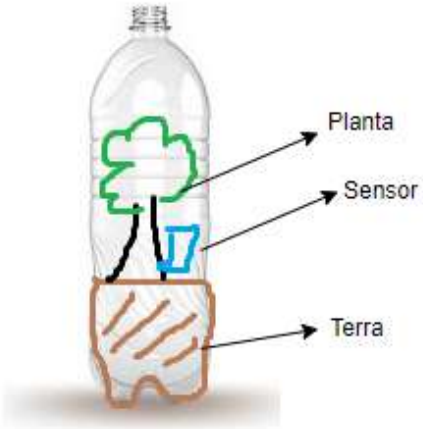
PARTE 1 - DOCUMENTAÇÃO TÉCNICA

1 - Resumo descritivo do projeto

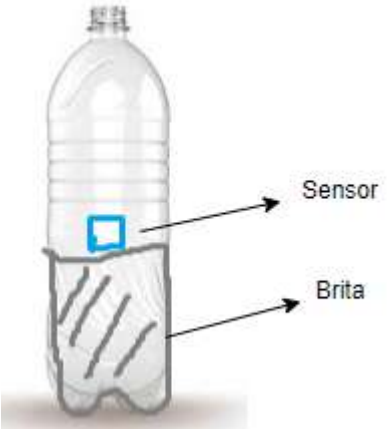
O objetivo do projeto é fazer um experimento em que se possa demonstrar o efeito estufa e o aquecimento global, de forma que seja possível elucidar esse problema de uma maneira clara para os alunos.

O experimento foi feito utilizando dois módulos de sensores de umidade e temperatura BME680, onde cada um deles foi colocado em uma situação, para demonstrarmos as consequências do efeito estufa e do aquecimento global. Além disso, nesse projeto foi adicionado um sensor HC05 que será utilizado em conjunto com uma dashboard, o objetivo é que os alunos possam ver a temperatura, umidade, pressão e gás pelo celular.

Na situação 1, representaremos um ambiente arborizado, com uma área verde. Na situação 2, temos um ambiente sem área verde, só com brita.



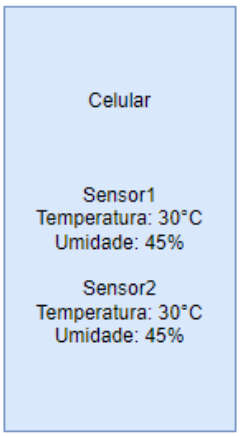
Situação 1



Situação 2

Os dois sensores estão conectados na BitDogLab, onde será mostrado os valores da temperatura, umidade, gás e pressão no display oled, o objetivo é que possamos comparar essas variáveis em dois ambientes diferentes,e mostrar a importância da preservação da natureza. E como dito acima, será possível ver esses parâmetros pelo celular também.

Abaixo, temos uma imagem da ideia inicial da dashboard do projeto.



2 - Objetivo

Seguindo a mesma linha do projeto 2, a ideia é que os professores possam usar a BitDogLab para complementar o que foi passado em aula, de forma que possa facilitar a melhor visualização do aluno sobre o conteúdo a ser estudado, possibilitando um ensino mais interativo e o aumento do engajamento por parte do aluno. Além disso, esse projeto traz um experimento sobre efeito estufa e aquecimento global, que é um tópico muito importante nos dias atuais.

3 - Itens da BNCC contemplados (de 2 a 5)

Ciências – 7º ANO
Unidades Temáticas : Terra e Universo
Objetos de Conhecimento:

- **Composição do ar**
- **Efeito estufa**
- Camada de ozônio
- Fenômenos naturais (vulcões, terremotos e tsunamis)
- Placas tectônicas e deriva continental.

Habilidades:

(EF07CI12) Demonstrar que o ar é uma mistura de gases, identificando sua composição, e discutir fenômenos naturais ou antrópicos que podem alterar essa composição.

(EF07CI13) Descrever o mecanismo natural do efeito estufa, seu papel fundamental para o desenvolvimento da vida na Terra, discutir as ações humanas responsáveis pelo seu aumento artificial (queima dos combustíveis fósseis, desmatamento,

queimadas etc.) e selecionar e implementar propostas para a reversão ou controle desse quadro.

4 - Recursos de hardware (on e off-board) a serem utilizados

Recursos on-board:

- Display Oled
- Botão
- Matriz RGB
- Buzzer

Recursos off-board:

- 2 x Sensores de Temperatura e Umidade BME680
- 1 x Sensor Bluetooth HC05
- 16 x Cabos groove 20 cm
- 2 x Conectores para cabo groove
- 1 x PCB Adaptada para o sensor HC05

4.1 - BME680

O sensor BME680 é um dispositivo compacto que combina sensores de temperatura, umidade relativa, pressão atmosférica, e um sensor de gás (para medir compostos orgânicos voláteis, ou VOCs).

4.2 - HC05

O HC-05 é um módulo Bluetooth clássico que permite comunicação sem fio via tecnologia Bluetooth.

Pontos de atenção: A alimentação desse sensor é feita com 5V. Ficar atento para fazer a ligação correta RX-TX e TX-RX.

5 - Descrição simplificada do software

O software foi projetado para ter uma interação entre os sensores BME680 e o HC05, junto com o display OLED, a matriz RGB e o buzzer.

O display OLED mostrará a temperatura, umidade, gás e pressão em cada ambiente. Nessa parte tem uma interação com o botão, que ao pressionar o botão você alterna entre os sensores para ver as respectivas informações. Outra interação que vai ter é com a matriz RGB, de acordo com a temperatura a matriz RGB muda de cor e quando atingir uma temperatura acima de 30°C o buzzer aciona. Um adendo sobre a matriz RGB é que as duas primeiras colunas serão destinadas para o primeiro sensor e as duas últimas serão destinadas ao segundo sensor.

Essas informações são enviadas para o sensor bluetooth HC05, e posteriormente serão lidas por uma dashboard. Dessa forma, será possível ver as informações no celular também.

6 - Maior desafio do projeto

Na minha visão o maior desafio do projeto foi a adaptação de como usar o sensor HC05 (Bluetooth) e preparar uma dashboard interessante para visualização da temperatura e umidade. Como eu não tinha experiência anterior com isso, acabei tendo dificuldade nessa parte final do projeto.

7 - Código Completo Comentado

7.1 - Código Completo

```
from machine import Pin, SoftI2C  # Configura GPIOs e interface I2C
from ssd1306 import SSD1306_I2C  # Controla displays OLED
from bme680 import *  # Permite interagir com sensores BME680
import machine, neopixel  # Funções de hardware e controle de LEDs RGB
```

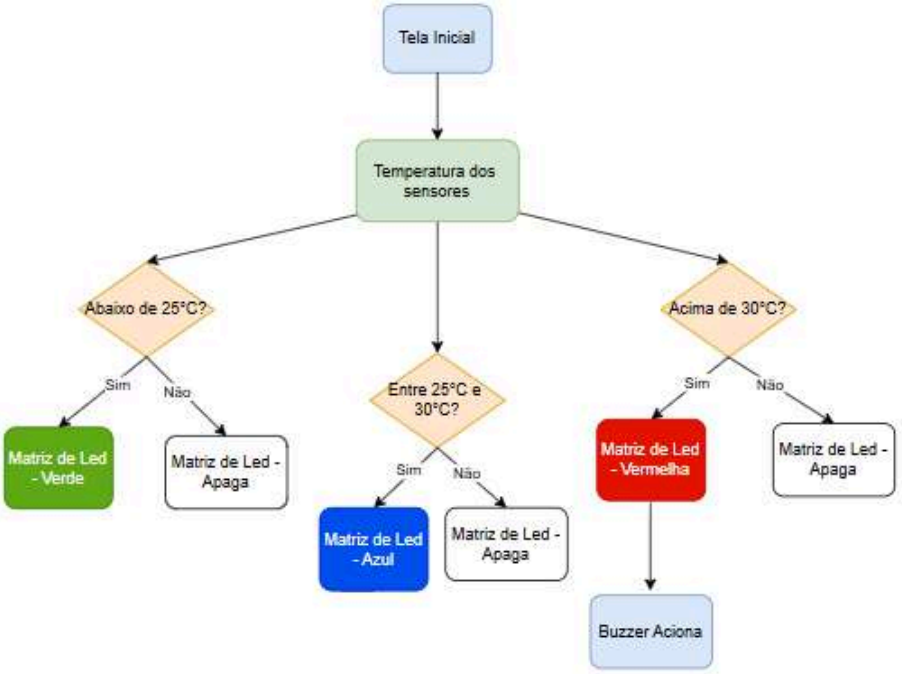
```
from time import sleep
from led import *  # Funções para gerenciar LEDs individuais em uma matriz
from machine import PWM, UART  # PWM para buzzer e UART para #comunicação serial

# Inicialização do Buzzer
buzzer_a = PWM(Pin(21))
buzzer_a.freq(1000)  # Ajustar a frequência para o som desejado

# Configuração UART para o HC-05
uart = UART(0, baudrate=9600, tx=Pin(16), rx=Pin(17))

# Inicialização do I2C para os sensores BME680 e para o display OLED
i2c_bme1 = SoftI2C(scl=Pin(3), sda=Pin(2))
i2c_bme2 = SoftI2C(scl=Pin(1), sda=Pin(0))
i2c_oled = SoftI2C(scl=Pin(15), sda=Pin(14))

# Inicialização do display OLED
oled = SSD1306_I2C(128, 64, i2c_oled)

# Inicialização dos sensores BME680
bme1 = BME680_I2C(i2c=i2c_bme1)
bme2 = BME680_I2C(i2c=i2c_bme2)

# Configuração da matriz de LED
np = neopixel.NeoPixel(Pin(7), NUM_LEDS)


intensity = 0.1  # Ajuste do brilho da Matriz RGB

# Definição das cores
GREEN = (int(0 * intensity), int(255 * intensity), int(0 * intensity))
BLUE = (int(0 * intensity), int(0 * intensity), int(255 * intensity))
RED = (int(255 * intensity), int(0 * intensity), int(0 * intensity))

# Configuração do botão
button = Pin(5, Pin.IN, Pin.PULL_UP)

beeped = False  # Variável para rastrear se o beep já foi emitido
current_sensor = 1  # Variável para controlar qual sensor está sendo exibido

# Função que controla o Buzzer
def beep(buzzer, duration=500):
    buzzer.duty_u16(30000)  # Liga o buzzer
```

```
    sleep(duration / 1000)
    buzzer.duty_u16(0)  # Desliga o buzzer

# Função para mostrar os parâmetros dos sensores no display OLED
def display_sensor_data(sensor):
    oled.fill(0)

    if sensor == 1:
        temp = str(round(bme2.temperature, 2)) + ' C'
        hum = str(round(bme2.humidity, 2)) + ' %'
        pres = str(round(bme2.pressure, 2)) + ' hPa'
        gas = str(round(bme2.gas / 1000, 2)) + ' KOhms'
        oled.text('Sensor 1', 0, 0)
    else:
        temp = str(round(bme1.temperature, 2)) + ' C'
        hum = str(round(bme1.humidity, 2)) + ' %'
        pres = str(round(bme1.pressure, 2)) + ' hPa'
        gas = str(round(bme1.gas / 1000, 2)) + ' KOhms'
        oled.text('Sensor 2', 0, 0)

    oled.text('Temp: {}'.format(temp), 0, 10)
    oled.text('Umidade: {}'.format(hum), 0, 20)
    oled.text('Pressao: {}'.format(pres), 0, 30)
    oled.text('Gas: {}'.format(gas), 0, 40)
    oled.show()

# Essa função serve para alternar entre os sensores quando o botão é
#pressionado
def read_button():
    global current_sensor
    if button.value() == 0:
        sleep(0.2)
        current_sensor = 2 if current_sensor == 1 else 1

# Essa função serve para configurar as cores para cada coluna de LEDs
def set_led_colors(sensor1_color, sensor2_color):
    for row in range(ROW_SIZE):
        ligar_led(3, row, sensor1_color)
        ligar_led(4, row, sensor1_color)
        ligar_led(0, row, sensor2_color)
        ligar_led(1, row, sensor2_color)
    np.write()
```

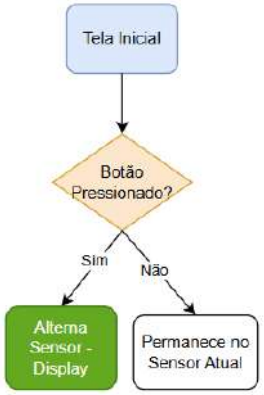
```
# Essa função envia os dados do sensor com marcadores claros para o App
#Inventor
def enviar_dados_bluetooth(sensor, temp, hum, pres, gas):
    # Formatar os dados com marcadores e separador '|'
    if sensor == 1:
        uart.write(f"S1|{temp:.2f}|{hum:.2f}|{pres:.2f}|{gas:.2f}|\n")
    elif sensor == 2:
        uart.write(f"|S2|{temp:.2f}|{hum:.2f}|{pres:.2f}|{gas:.2f}\n")

while True:
    read_button()
    display_sensor_data(current_sensor)

    temp1 = bme2.temperature
    temp2 = bme1.temperature

# Cores da matriz de led se alteram de acordo com a temperatura
    color1 = GREEN if temp1 < 25 else BLUE if temp1 <= 30 else RED
    color2 = GREEN if temp2 < 25 else BLUE if temp2 <= 30 else RED

    set_led_colors(color1, color2)

# Buzzer aciona quando a temperatura é maior que 30°C
    if (temp1 >= 30 or temp2 >= 30) and not beeped:
        beep(buzzer_a, duration=500)
        beeped = True
    elif temp1 < 30 and temp2 < 30:
        beeped = False

# Dados são enviados para o bluetooth
        enviar_dados_bluetooth(1, temp1, bme2.humidity, bme2.pressure,
bme2.gas / 1000)
        enviar_dados_bluetooth(2, temp2, bme1.humidity, bme1.pressure,
bme1.gas / 1000)
    sleep(1)
```

7.2 - Explicação do Código

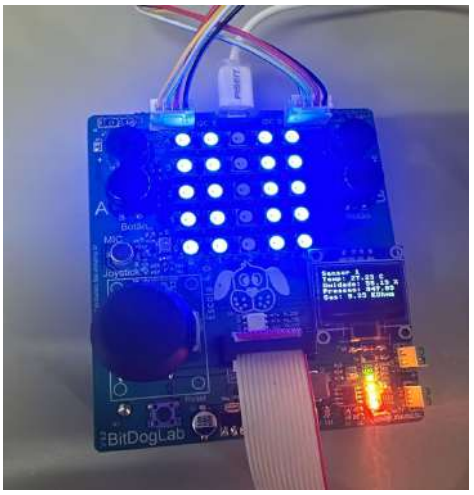
Fluxograma 1



Fluxograma 2



No desenvolvimento desse código, utilizei as duas comunicações do I2C para trabalhar com os sensores. No display OLED, vai aparecer os valores da temperatura, umidade, gás e pressão que os sensores estão enviando de cada ambiente, caso a temperatura seja inferior a 25°C, a matriz de led ficará verde, se a temperatura estiver entre 25°C e 30°C, a matriz de led ficará azul, por fim, se a temperatura for superior a 30°C, a matriz de led ficará vermelha e um buzzer irá acionar. Em relação ao projeto passado, eu fiz algumas modificações, a matriz de LED foi melhor dividida, eu separei duas colunas para cada sensor, assim é possível visualizar melhor o que está acontecendo em cada ambiente. Outro ponto importante, foi a adição do sensor HC05, esse sensor está sendo utilizado para mandar os dados do sensor para a dashboard, dessa forma, será possível ver os dados pelo celular.



Observações importantes para quem for utilizar esse código:

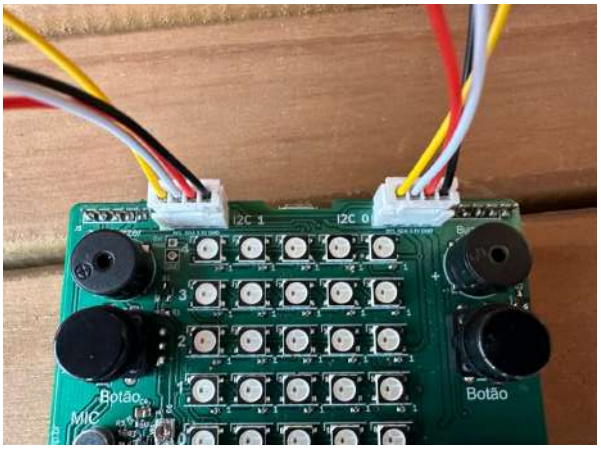
- Não esquecer de carregar a biblioteca do BME680 na Raspberry, a biblioteca será disponibilizada junto com o código no github.
- Ficar atento com os pinos da comunicação I2C.

Pinos que são utilizados na comunicação I2C

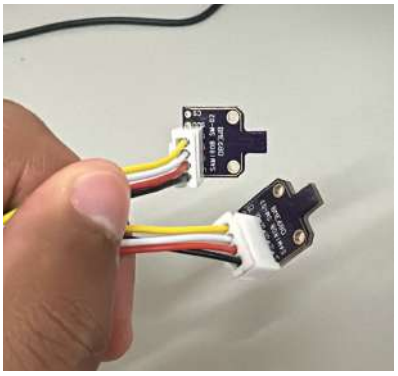
I2C1 - GPIOS
GPIO3 -SCL
GPIO2- SDA
VCC- 3.3V
GND - GND

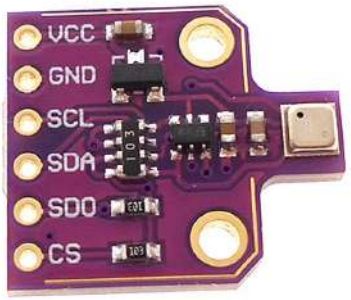
I2C0 - GPIOS
GPIO0 - SDA
GPIO1- SCL
VCC- 3.3V
GND - GND

- Ficar atento para conectar corretamente os sensores, lembrando que o I2C1 fica na esquerda da BitDogLab e o I20 fica na direita.

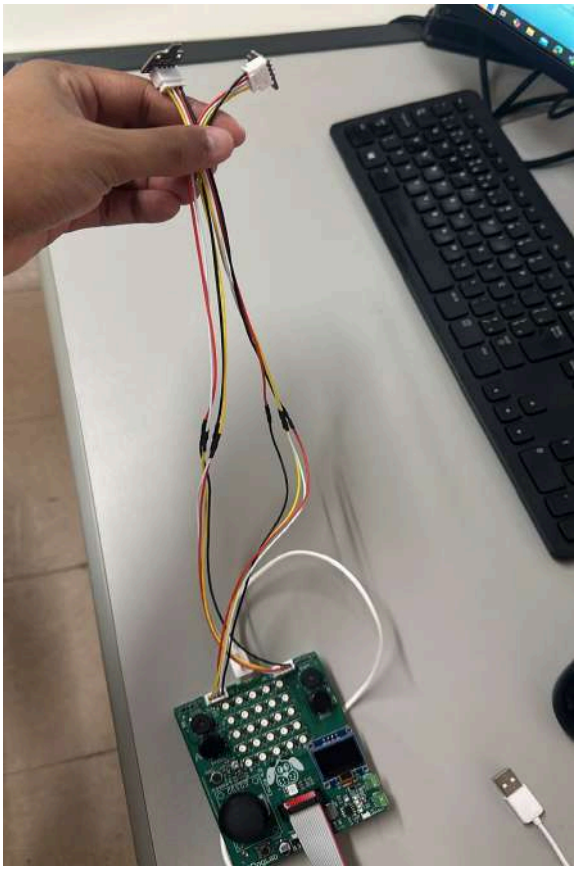
-


- Ficar atento ao ligar os pinos do I2C com os pinos do sensor BME680

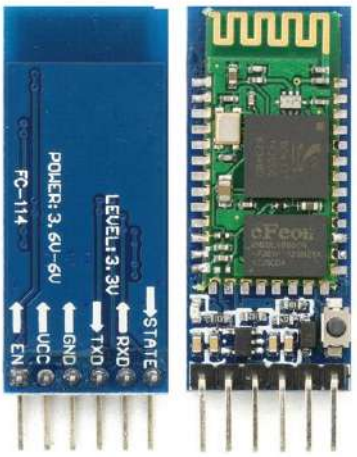




- Caso seja necessário aumentar fio, ficar atento para ligar Vcc → Vcc , GND → GND, SLC → SLC e SDA →SDA



- Outra ponto muito importante é sobre o sensor HC05



* Não esquecer que ele é alimentado com 5V e a ligação dele é TX - RX e RX- TX

7.3 - Biblioteca BME680

Spaces, comments and some functions have been removed from the original file to save memory

Original source: https://github.com/adafruit/Adafruit_CircuitPython_BME680/blob/master/adafruit_bme680.py

```
import time
import math
from micropython import const
from ubinascii import hexlify as hex
try:
  import struct
except ImportError:
  import ustruct as struct
_BME680_CHIPID = const(0x61)
_BME680_REG_CHIPID = const(0xD0)
_BME680_BME680_COEFF_ADDR1 = const(0x89)
_BME680_BME680_COEFF_ADDR2 = const(0xE1)
_BME680_BME680_RES_HEAT_0 = const(0x5A)
_BME680_BME680_GAS_WAIT_0 = const(0x64)
_BME680_REG_SOFTRESET = const(0xE0)
_BME680_REG_CTRL_GAS = const(0x71)
_BME680_REG_CTRL_HUM = const(0x72)
_BME280_REG_STATUS = const(0xF3)
_BME680_REG_CTRL_MEAS = const(0x74)
_BME680_REG_CONFIG = const(0x75)
_BME680_REG_PAGE_SELECT = const(0x73)
```

```
        _BME680_REG_MEAS_STATUS = const(0x1D)
        _BME680_REG_PDATA = const(0x1F)
        _BME680_REG_TDATA = const(0x22)
        _BME680_REG_HDATA = const(0x25)
        _BME680_SAMPLERATES = (0, 1, 2, 4, 8, 16)
        _BME680_FILTERSIZES = (0, 1, 3, 7, 15, 31, 63, 127)
        _BME680_RUNGAS = const(0x10)
        _LOOKUP_TABLE_1 = (2147483647.0, 2147483647.0, 2147483647.0,
2147483647.0, 2147483647.0,
                2126008810.0, 2147483647.0, 2130303777.0, 2147483647.0,
2147483647.0,
                2143188679.0, 2136746228.0, 2147483647.0, 2126008810.0,
2147483647.0,
         2147483647.0)
        _LOOKUP_TABLE_2 = (4096000000.0, 2048000000.0, 1024000000.0,
512000000.0, 255744255.0, 127110228.0,
          64000000.0, 32258064.0, 16016016.0, 8000000.0, 4000000.0, 2000000.0,
1000000.0,
         500000.0, 250000.0, 125000.0)
        def _read24(arr):
         ret = 0.0
         for b in arr:
           ret *= 256.0
           ret += float(b & 0xFF)
         return ret
        class Adafruit_BME680:
         def __init__(self, *, refresh_rate=10):
           self._write(_BME680_REG_SOFTRESET, [0xB6])
           time.sleep(0.005)
           chip_id = self._read_byte(_BME680_REG_CHIPID)
           if chip_id != _BME680_CHIPID:
             raise RuntimeError('Failed 0x%x' % chip_id)
           self._read_calibration()
           self._write(_BME680_BME680_RES_HEAT_0, [0x73])
           self._write(_BME680_BME680_GAS_WAIT_0, [0x65])
           self.sea_level_pressure = 1013.25
           self._pressure_oversample = 0b011
           self._temp_oversample = 0b100
           self._humidity_oversample = 0b010
           self._filter = 0b010
           self._adc_pres = None
           self._adc_temp = None
           self._adc_hum = None
           self._adc_gas = None
```

```
        self._gas_range = None
        self._t_fine = None
        self._last_reading = 0
        self._min_refresh_time = 1000 / refresh_rate
    @property
    def pressure_oversample(self):
        return _BME680_SAMPLERATES[self._pressure_oversample]
    @pressure_oversample.setter
    def pressure_oversample(self, sample_rate):
        if sample_rate in _BME680_SAMPLERATES:
                                                    self._pressure_oversample          =
_BME680_SAMPLERATES.index(sample_rate)
        else:
            raise RuntimeError("Invalid")
    @property
    def humidity_oversample(self):
        return _BME680_SAMPLERATES[self._humidity_oversample]
    @humidity_oversample.setter
    def humidity_oversample(self, sample_rate):
        if sample_rate in _BME680_SAMPLERATES:
                                                    self._humidity_oversample          =
_BME680_SAMPLERATES.index(sample_rate)
        else:
            raise RuntimeError("Invalid")
    @property
    def temperature_oversample(self):
            return _BME680_SAMPLERATES[self._temp_oversample]
    @temperature_oversample.setter
    def temperature_oversample(self, sample_rate):
        if sample_rate in _BME680_SAMPLERATES:
            self._temp_oversample = _BME680_SAMPLERATES.index(sample_rate)
        else:
            raise RuntimeError("Invalid")
    @property
    def filter_size(self):
        return _BME680_FILTERSIZES[self._filter]
    @filter_size.setter
    def filter_size(self, size):
        if size in _BME680_FILTERSIZES:
            self._filter = _BME680_FILTERSIZES[size]
        else:
            raise RuntimeError("Invalid")
    @property
    def temperature(self):
```

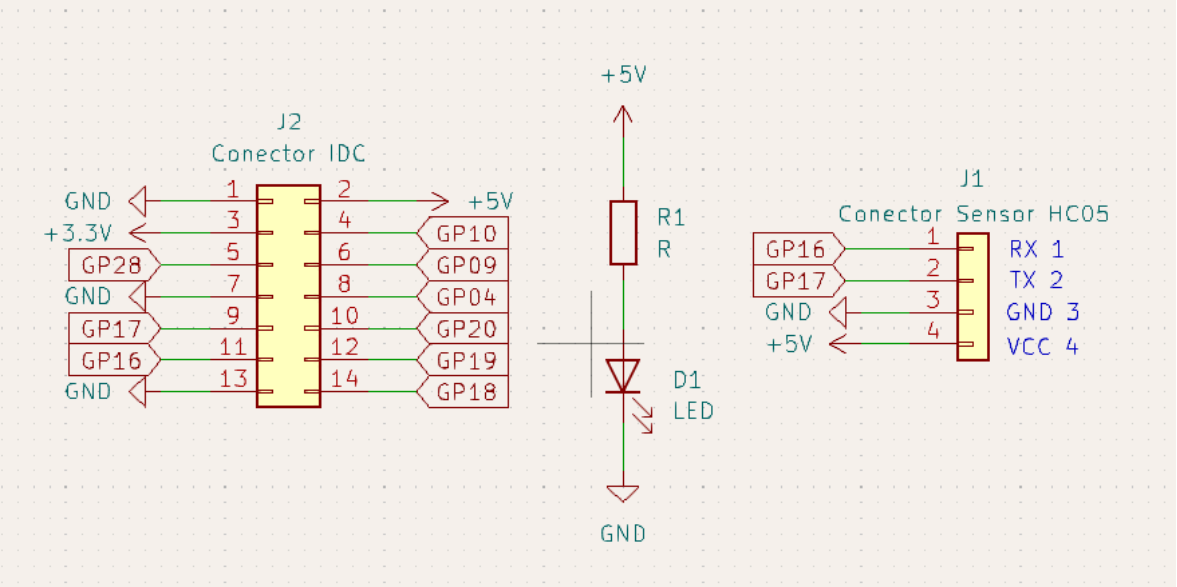
```
        self._perform_reading()
        calc_temp = (((self._t_fine * 5) + 128) / 256)
        return calc_temp / 100
      @property
      def pressure(self):
        self._perform_reading()
        var1 = (self._t_fine / 2) - 64000
        var2 = ((var1 / 4) * (var1 / 4)) / 2048
        var2 = (var2 * self._pressure_calibration[5]) / 4
        var2 = var2 + (var1 * self._pressure_calibration[4] * 2)
        var2 = (var2 / 4) + (self._pressure_calibration[3] * 65536)
        var1 = (((((var1 / 4) * (var1 / 4)) / 8192) *
          (self._pressure_calibration[2] * 32) / 8) +
          ((self._pressure_calibration[1] * var1) / 2))
        var1 = var1 / 262144
        var1 = ((32768 + var1) * self._pressure_calibration[0]) / 32768
        calc_pres = 1048576 - self._adc_pres
        calc_pres = (calc_pres - (var2 / 4096)) * 3125
        calc_pres = (calc_pres / var1) * 2
          var1 = (self._pressure_calibration[8] * (((calc_pres / 8) * (calc_pres / 8)) /
8192)) / 4096
        var2 = ((calc_pres / 4) * self._pressure_calibration[7]) / 8192
        var3 = (((calc_pres / 256) ** 3) * self._pressure_calibration[9]) / 131072
         calc_pres += ((var1 + var2 + var3 + (self._pressure_calibration[6] * 128)) /
16)
        return calc_pres/100
      @property
      def humidity(self):
        self._perform_reading()
        temp_scaled = ((self._t_fine * 5) + 128) / 256
        var1 = ((self._adc_hum - (self._humidity_calibration[0] * 16)) -
          ((temp_scaled * self._humidity_calibration[2]) / 200))
        var2 = (self._humidity_calibration[1] *
          (((temp_scaled * self._humidity_calibration[3]) / 100) +
           (((temp_scaled * ((temp_scaled * self._humidity_calibration[4]) / 100)) /
             64) / 100) + 16384)) / 1024
        var3 = var1 * var2
        var4 = self._humidity_calibration[5] * 128
        var4 = (var4 + ((temp_scaled * self._humidity_calibration[6]) / 100)) / 16
        var5 = ((var3 / 16384) * (var3 / 16384)) / 1024
        var6 = (var4 * var5) / 2
        calc_hum = (((var3 + var6) / 1024) * 1000) / 4096
        calc_hum /= 1000
        if calc_hum > 100:
```

```
            calc_hum = 100
          if calc_hum < 0:
            calc_hum = 0
          return calc_hum
        @property
        def altitude(self):
          pressure = self.pressure
          return 44330 * (1.0 - math.pow(pressure / self.sea_level_pressure, 0.1903))
        @property
        def gas(self):
          self._perform_reading()
                        var1 = ((1340 + (5 * self._sw_err)) *
(_LOOKUP_TABLE_1[self._gas_range])) / 65536
          var2 = ((self._adc_gas * 32768) - 16777216) + var1
          var3 = (_LOOKUP_TABLE_2[self._gas_range] * var1) / 512
          calc_gas_res = (var3 + (var2 / 2)) / var2
          return int(calc_gas_res)
        def _perform_reading(self):
          if (time.ticks_diff(self._last_reading, time.ticks_ms()) * time.ticks_diff(0, 1)
              < self._min_refresh_time):
            return
          self._write(_BME680_REG_CONFIG, [self._filter << 2])
          self._write(_BME680_REG_CTRL_MEAS,
            [(self._temp_oversample << 5)|(self._pressure_oversample << 2)])
          self._write(_BME680_REG_CTRL_HUM, [self._humidity_oversample])
          self._write(_BME680_REG_CTRL_GAS, [_BME680_RUNGAS])
          ctrl = self._read_byte(_BME680_REG_CTRL_MEAS)
          ctrl = (ctrl & 0xFC) | 0x01
          self._write(_BME680_REG_CTRL_MEAS, [ctrl])
          new_data = False
          while not new_data:
            data = self._read(_BME680_REG_MEAS_STATUS, 15)
            new_data = data[0] & 0x80 != 0
            time.sleep(0.005)
          self._last_reading = time.ticks_ms()
          self._adc_pres = _read24(data[2:5]) / 16
          self._adc_temp = _read24(data[5:8]) / 16
          self._adc_hum = struct.unpack('>H', bytes(data[8:10]))[0]
          self._adc_gas = int(struct.unpack('>H', bytes(data[13:15]))[0] / 64)
          self._gas_range = data[14] & 0x0F
          var1 = (self._adc_temp / 8) - (self._temp_calibration[0] * 2)
          var2 = (var1 * self._temp_calibration[1]) / 2048
          var3 = ((var1 / 2) * (var1 / 2)) / 4096
          var3 = (var3 * self._temp_calibration[2] * 16) / 16384
```

```
        self._t_fine = int(var2 + var3)
      def _read_calibration(self):
        coeff = self._read(_BME680_BME680_COEFF_ADDR1, 25)
        coeff += self._read(_BME680_BME680_COEFF_ADDR2, 16)
                coeff = list(struct.unpack('<hbBHhbBhhbbHhhBBBHbbbBbHhbb',
bytes(coeff[1:39])))
        coeff = [float(i) for i in coeff]
        self._temp_calibration = [coeff[x] for x in [23, 0, 1]]
        self._pressure_calibration = [coeff[x] for x in [3, 4, 5, 7, 8, 10, 9, 12, 13, 14]]
        self._humidity_calibration = [coeff[x] for x in [17, 16, 18, 19, 20, 21, 22]]
        self._gas_calibration = [coeff[x] for x in [25, 24, 26]]
        self._humidity_calibration[1] *= 16
        self._humidity_calibration[1] += self._humidity_calibration[0] % 16
        self._humidity_calibration[0] /= 16
        self._heat_range = (self._read_byte(0x02) & 0x30) / 16
        self._heat_val = self._read_byte(0x00)
        self._sw_err = (self._read_byte(0x04) & 0xF0) / 16
      def _read_byte(self, register):
        return self._read(register, 1)[0]
      def _read(self, register, length):
        raise NotImplementedError()
      def _write(self, register, values):
        raise NotImplementedError()
    class BME680_I2C(Adafruit_BME680):
      def __init__(self, i2c, address=0x77, debug=False, *, refresh_rate=10):
        self._i2c = i2c
        self._address = address
        self._debug = debug
        super().__init__(refresh_rate=refresh_rate)
      def _read(self, register, length):
        result = bytearray(length)
        self._i2c.readfrom_mem_into(self._address, register & 0xff, result)
        if self._debug:
            print("\t${:x} read ".format(register), " ".join(["{:02x}".format(i) for i in
result]))
        return result
      def _write(self, register, values):
        if self._debug:
            print("\t${:x} write".format(register), " ".join(["{:02x}".format(i) for i in
values]))
        for value in values:
          self._i2c.writeto_mem(self._address, register, bytearray([value & 0xFF]))
          register += 1
```

8 - Descrição do hardware (PCB Adaptada para Sensor Bluetooth)

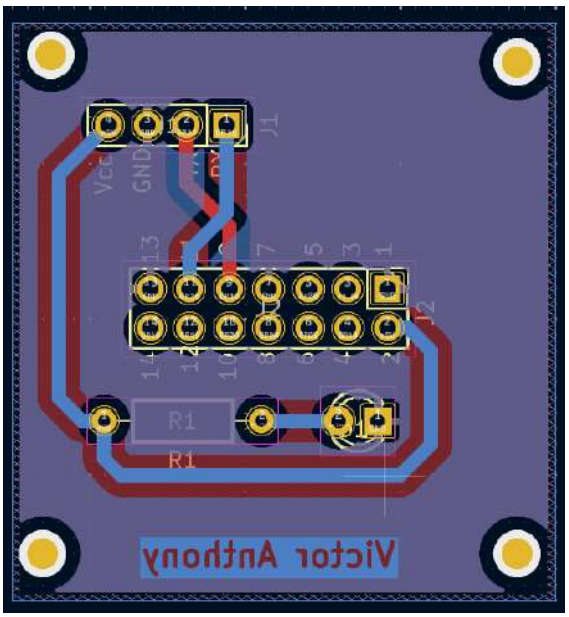
Para esse projeto foi desenvolvida uma PCB, essa PCB foi utilizada para utilizar o sensor HC05, com isso, não foi preciso utilizar uma protoboard, nem deixar fio solto.



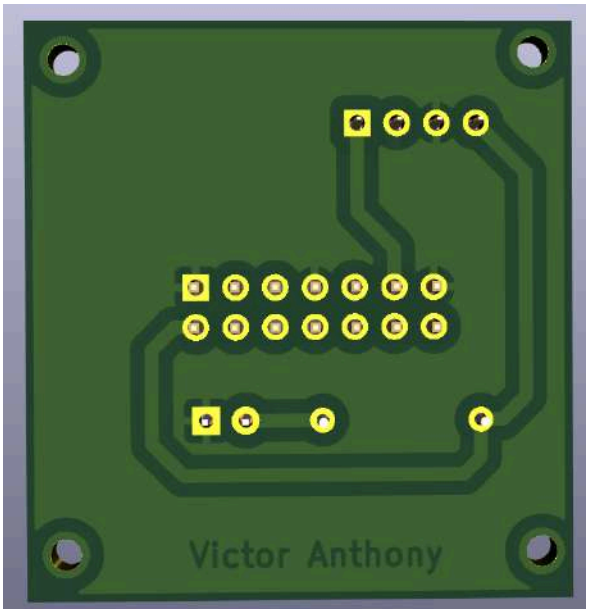
O esquemático é bem simples, basicamente, utilizamos os pinos do IDC para fazer a conexão com o sensor HC05, coloquei um LED só para verificar se está chegando a tensão correta no sensor.

Pontos importantes no desenvolvimento dessa PCB, ficar atento com a pinagem do IDC, tem que estar na mesma ordem da BitDog, outro ponto importante, como estamos utilizando o sensor bluetooth não podemos esquecer de fazer a ligação correta RX-TX e TX-RX.

Depois do esquemático, a próxima etapa é fazer o layout.



Para conseguir adequar e deixar uma PCB funcional, sem precisar de jumper, não consegui utilizar uma camada, foi preciso utilizar duas camadas. Além disso, afastei um pouco os conectores, na primeira versão eles estavam muito próximos atrapalhando para colocar o cabo flat.



Listas de Componentes:

1 x Conector IDC Macho 2x7 → https://www.acheicomponentes.com.br/conector/conector-idc-macho-14-vias-2x7-180-preto-s-ejetor?srsltid=AfmBOoq7RUI6C4pHshAyXPG0kX9xBSxy4kApf6YOrd_SrG47PjZLHOzg

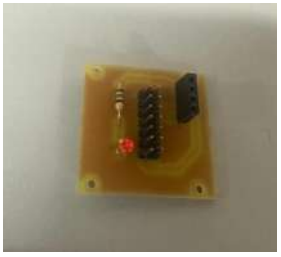
1 x Conector Soquete Fêmea 1x4 → https://www.acheicomponentes.com.br/barra-de-pinos/femea/4-vias-pinos/barra-de-pinos-femea-mci-1x4-180o-passo-2-54mm-ds1023-1-4s21

1 x Led 3mm (Verde ou Vermelho) → https://www.eletrogate.com/led-difuso-3mm-verde?utm_source=Site&utm_medium=GoogleMerchant&utm_campaign=GoogleMerchant&utm_source=google&utm_medium=cpc&utm_campaign=[MC4]_[G]_[PMax]_Categorias&utm_content=&utm_term=&gad_source=1&gclid=Cj0KCQiAgJa6BhCOARIsAMiL7V8dAxi9SX3avvpwHgGnTyeNB0uWCCNmNvv-Ugl1cjIGfX4SFhwAV44aAulzEALw_wcB

1 x Resistor 150Ω → https://www.eletrogate.com/resistor-150r-1-4w-10-unidades?utm_source=Site&utm_medium=GoogleMerchant&utm_campaign=GoogleMerchant&utm_source=google&utm_medium=cpc&utm_campaign=[MC4]_[G]_[PMax]_Categorias&utm_content=&utm_term=&gad_source=1&gclid=Cj0KCQiAgJa6BhCOARIsAMiL7V_dyR9h3dS-_T8-vt364jDpHKyTtuIpGfXzoCEQxcbAgQpwdTvlnSAaAtPnEALw_wcB

1x Cabo Flat 2x7 → https://www.elecbee.com/pt-3326-254mm-passo-2x7-pin-14-pin-14-fio-idc-flat-ribbon-comprimento-do-cabo-20cm

PCB Montada



9 - Desenvolvimento Dashboard

A dashboard foi desenvolvida utilizando o AppInventor. Foram duas etapas de desenvolvimento, sendo a primeira a montagem da interface e a segunda foi o diagrama de blocos.

9.1 - Desenvolvimento Interface

Para desenvolver a interface, foi utilizado os elementos que o AppInventor dispõe. A dashboard é feita a partir de um layout e das labels do User Interface.

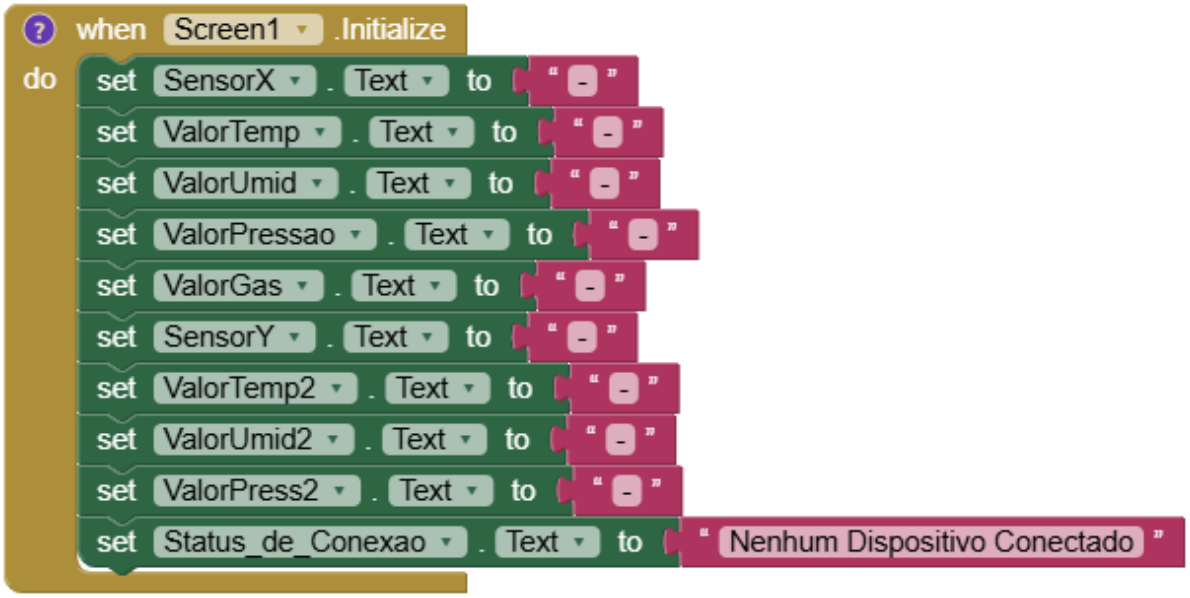
Todos os layouts e labels que foram utilizados, assim como o app desenvolvido no AppInventor, ficaram disponíveis no github.

Resultado Final da Interface

9.2 - Desenvolvimento Diagrama de Blocos

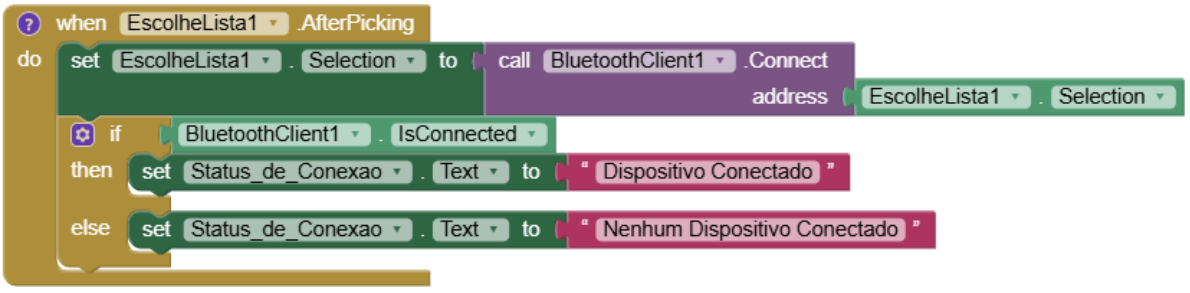
Primeiro bloco → guarda as variáveis que vão chegar da raspberry.

Segundo bloco → Inicializando a screen 1 - quando começa o app o que tem que acontecer, nesse caso, estamos mostrando quais variáveis que estarão sendo mostradas assim que inicializarmos o app.

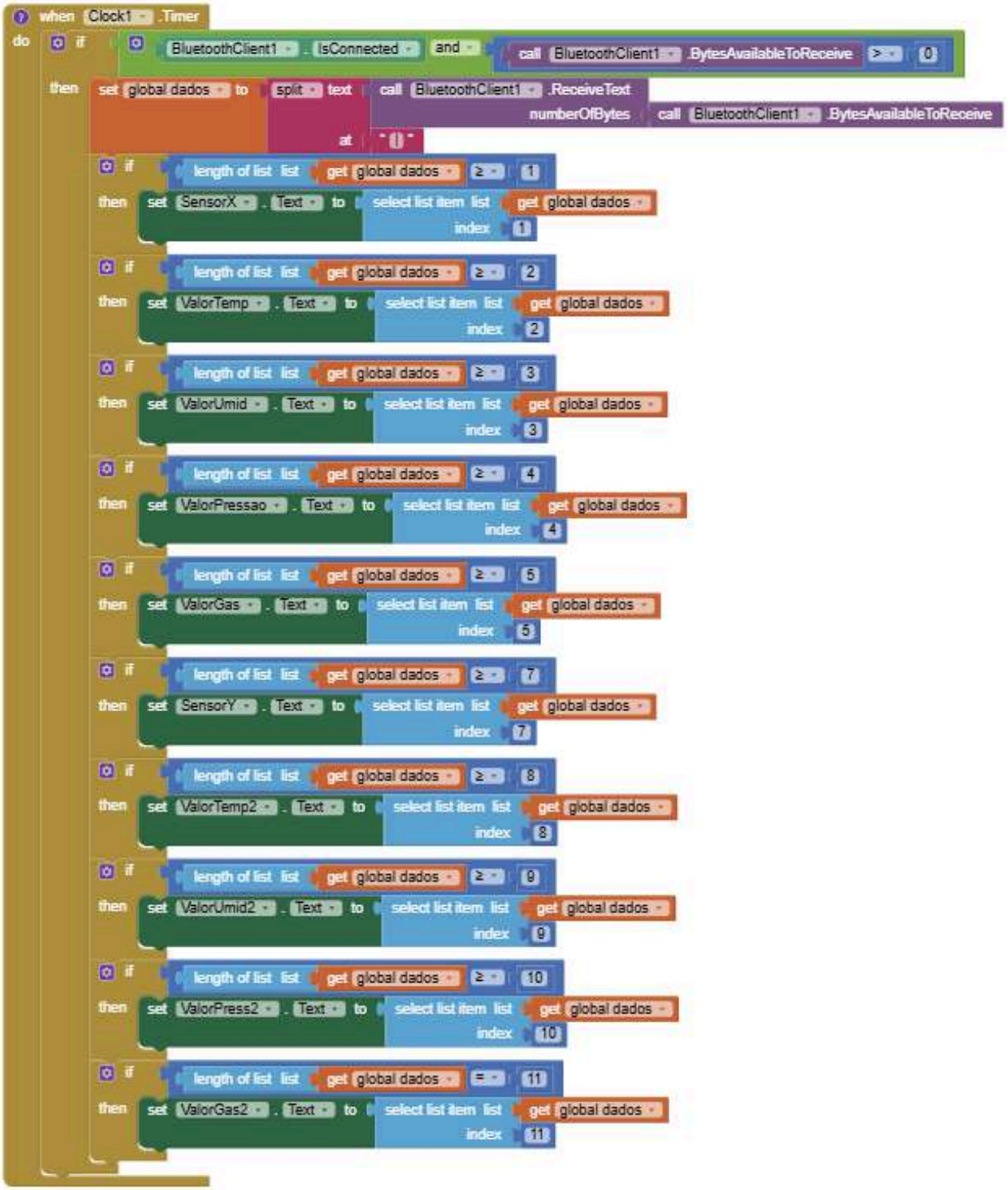


Terceiro bloco → Antes da pessoa clicar no botão, antes da conexão com o bluetooth.

Quarto bloco → Depois de escolher, depois da pessoa conectar com o bluetooth.



Quinto bloco → Esse diagrama serve para o ler as informações que estão chegando do sensor, é importante separar essas informações com um marcador para os dados chegarem corretamente na dashboard, outro ponto importante é que os dados estão chegando em forma de lista, isso é importante para serem lidos de forma ordenada.


when Clock1 .Timer
do if BluetoothClient1 . IsConnected and call BluetoothClient1 .BytesAvailableToReceive > 0
then set global dados to split text call BluetoothClient1 .ReceiveText numberOfBytes call BluetoothClient1 .BytesAvailableToReceive at "()"
if length of list list get global dados ≥ 1
then set SensorX . Text to select list item list get global dados index 1
if length of list list get global dados ≥ 2
then set ValorTemp . Text to select list item list get global dados index 2
if length of list list get global dados ≥ 3
then set ValorUmid . Text to select list item list get global dados index 3
if length of list list get global dados ≥ 4
then set ValorPressao . Text to select list item list get global dados index 4
if length of list list get global dados ≥ 5
then set ValorGas . Text to select list item list get global dados index 5
if length of list list get global dados ≥ 7
then set SensorY . Text to select list item list get global dados index 7
if length of list list get global dados ≥ 8
then set ValorTemp2 . Text to select list item list get global dados index 8
if length of list list get global dados ≥ 9
then set ValorUmid2 . Text to select list item list get global dados index 9
if length of list list get global dados ≥ 10
then set ValorPress2 . Text to select list item list get global dados index 10
if length of list list get global dados = 11
then set ValorGas2 . Text to select list item list get global dados index 11

Resultado Final Interface

Tela Inicial - Antes de Conectar o Bluetooth

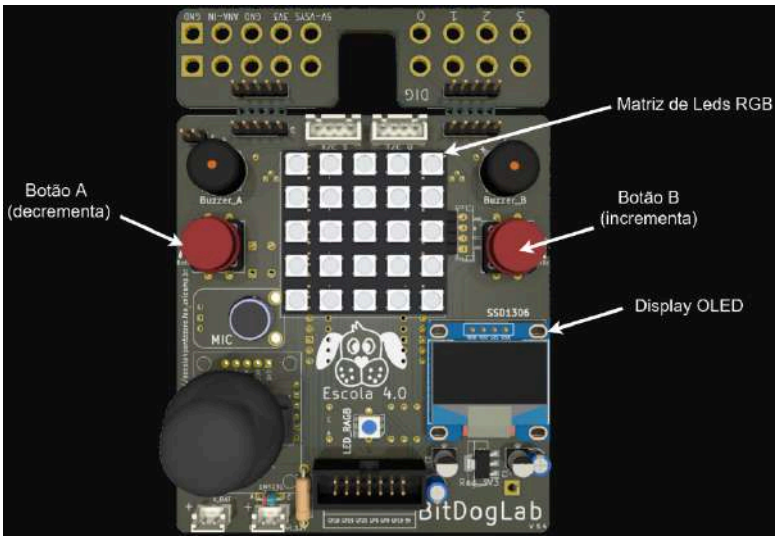
Tela com dados - Após Conexão com Bluetooth

PARTE 2 - MANUAL PARA PROFESSORES

1 - Tutorial BitDogLab

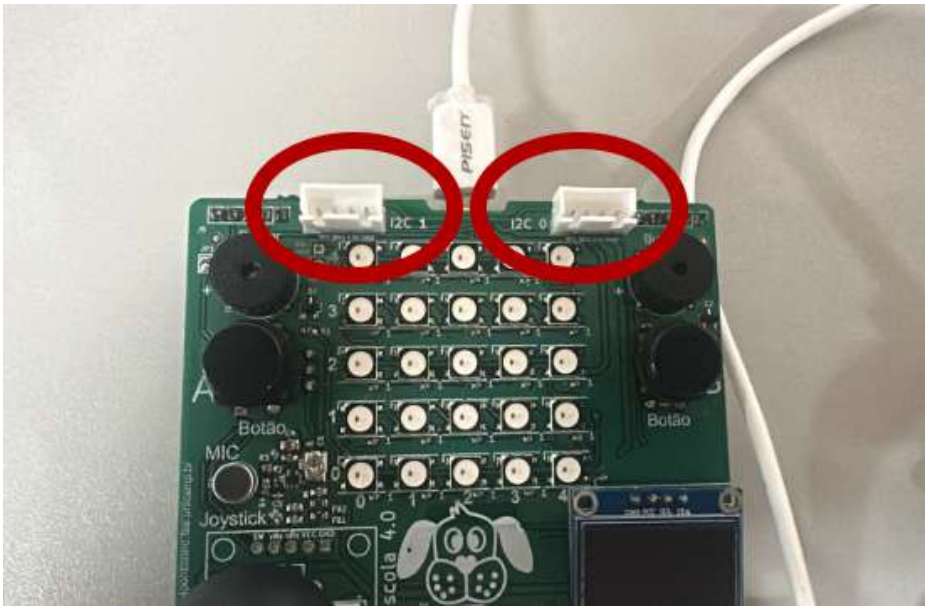
1.1 - Tutorial de uso inicial para BitDog

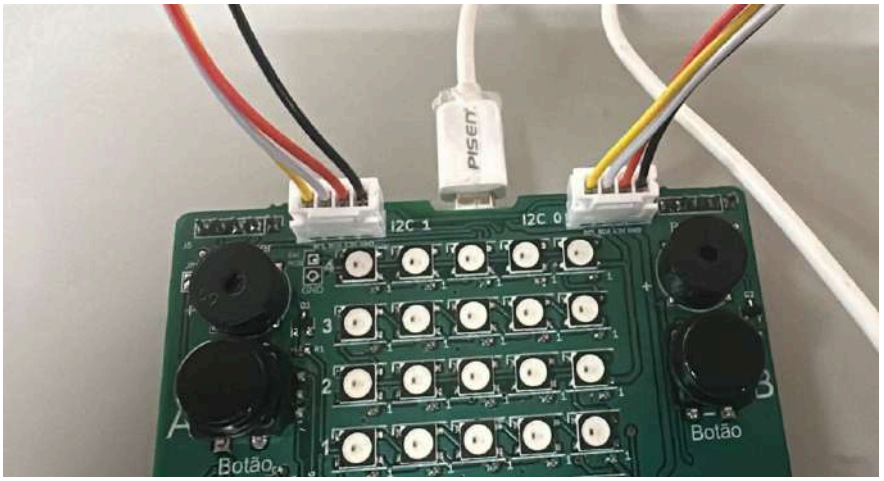
1. Energizar a placa da BitDog - Plugue o cabo USB no computador ou utilize uma bateria adequada.
2. Aguarde enquanto a BitDog realiza o seu processo de inicialização automático.
3. Siga as instruções apresentadas no display OLED.
4. Use o botão "B" para avançar nas etapas do menu do Display OLED e use o Botão "A" para voltar à etapa anterior do menu do display OLED.



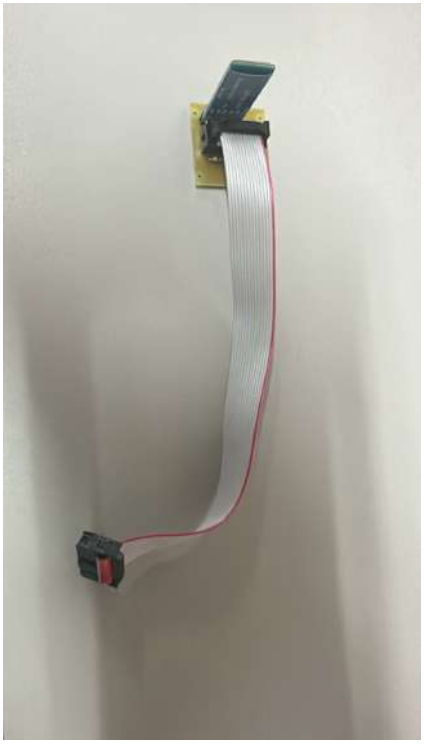
1.2 - Montagem Inicial Experimento

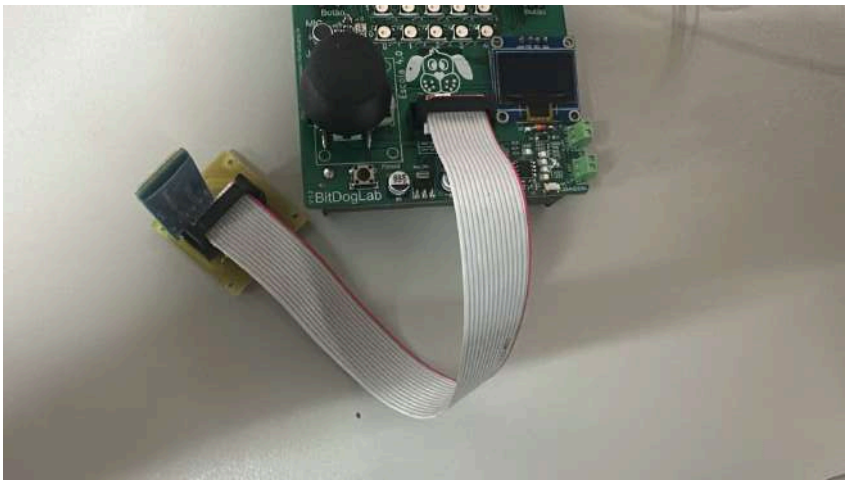
1 - Ligar os sensores BME 680 no I2C da BitDog



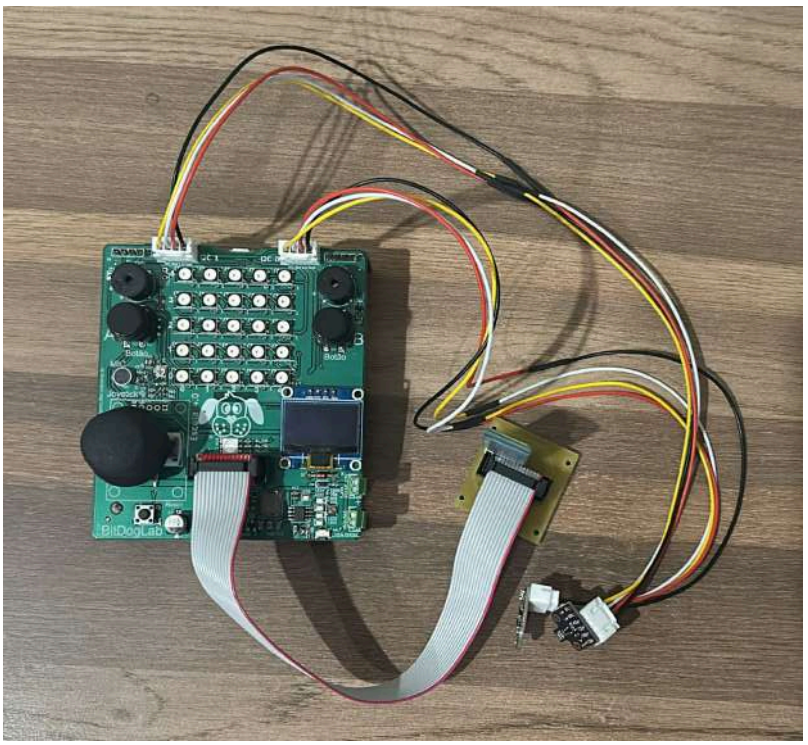


2 - Conectar PCB no conector IDC da BitDogLab





Com esses passos concluídos, partir para a montagem do experimento.



2 - Montagem e realização do experimento

O experimento vai precisar de 4 garrafas pet 's, terra, planta e brita.
Basta cortar as garrafas no meio e utilizar somente a parte de baixo delas, a parte da tampa pode descartar.
Uma sugestão, é que o professor incentive o aluno a fazer essa parte, dessa forma ficará melhor para ter um engajamento do aluno.

Um exemplo de montagem é a seguinte:

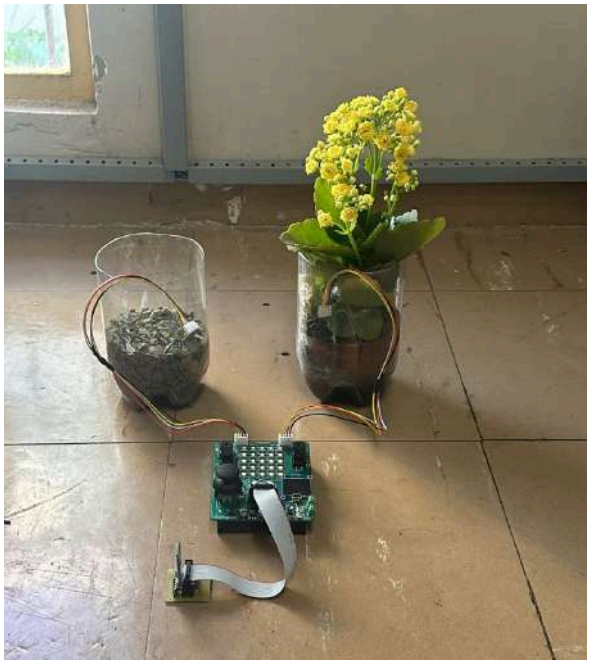
1) **Simulação do ambiente arborizado**
   1.1) Materiais utilizados:
   - **Garrafa Pet**
   - **Terra**
   - **Plantas**

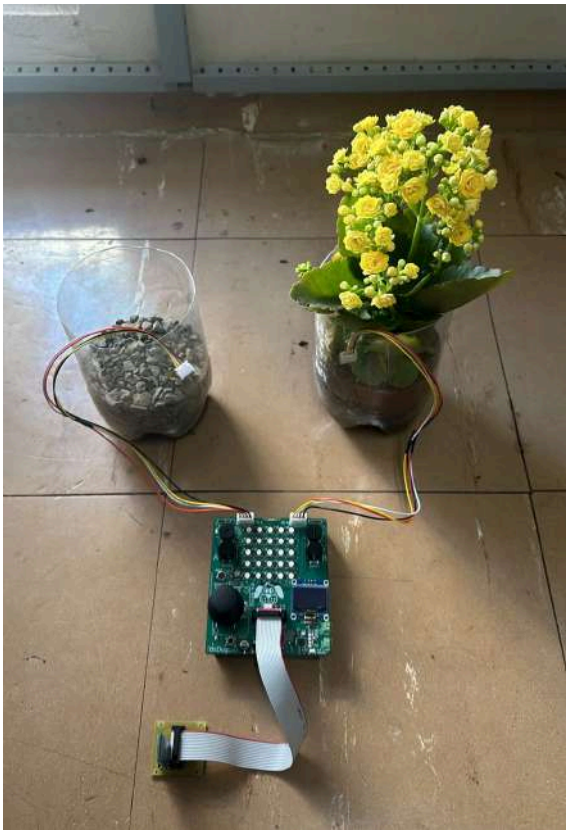
2) **Simulação do ambiente sem árvores**

2.1) Materiais utilizados:

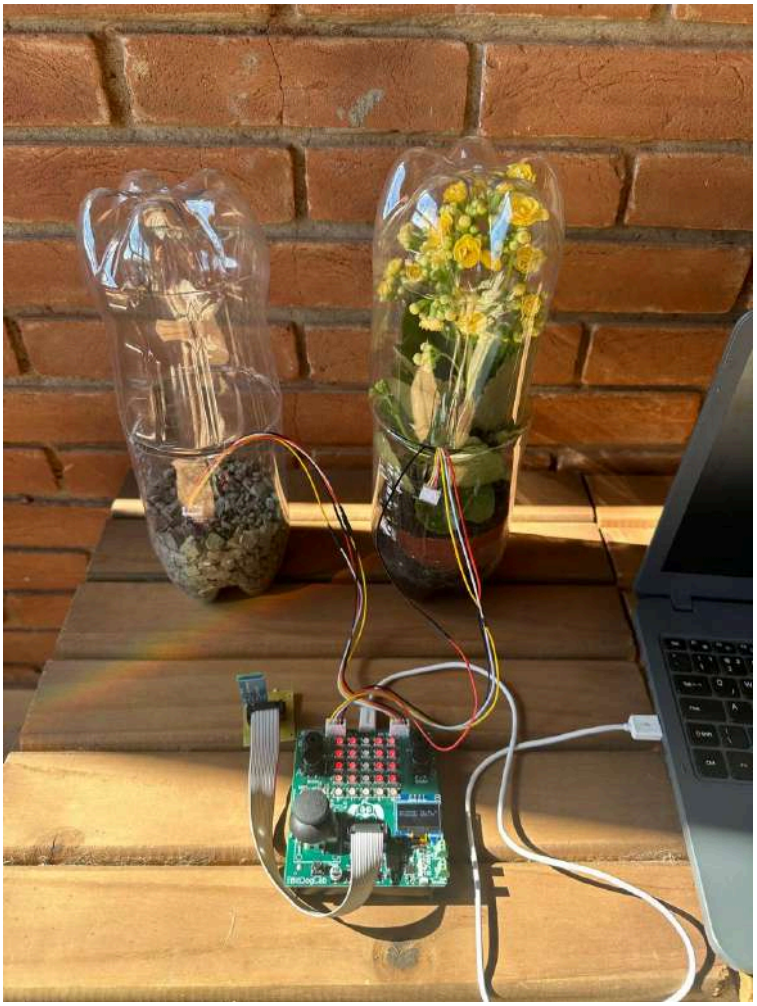
- **Garrafa Pet**
- **Brita**

Os sensores ficarão da seguinte forma nas garrafas:





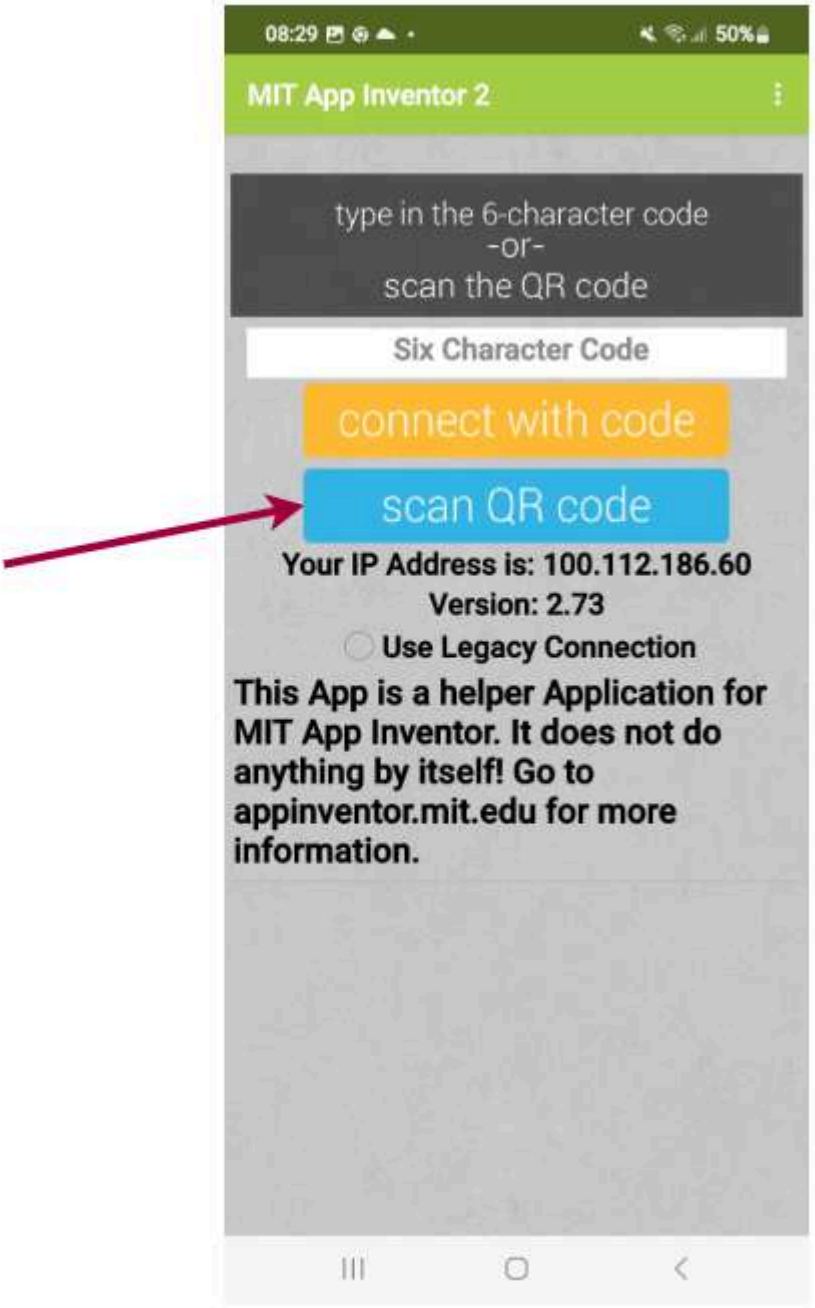
Atenção: A medida será feita tampando cobrindo as garrafas, para conseguir uma medida mais precisa.



Feita toda a montagem inicial, agora é hora de partir para o experimento em si, com os sensores dentro das garrafas, basta ligar a BitDogLab e o código já estará carregado nela, assim que BitDog for ligada já começara a mensurar a temperatura, umidade, gás e pressão do ambiente.

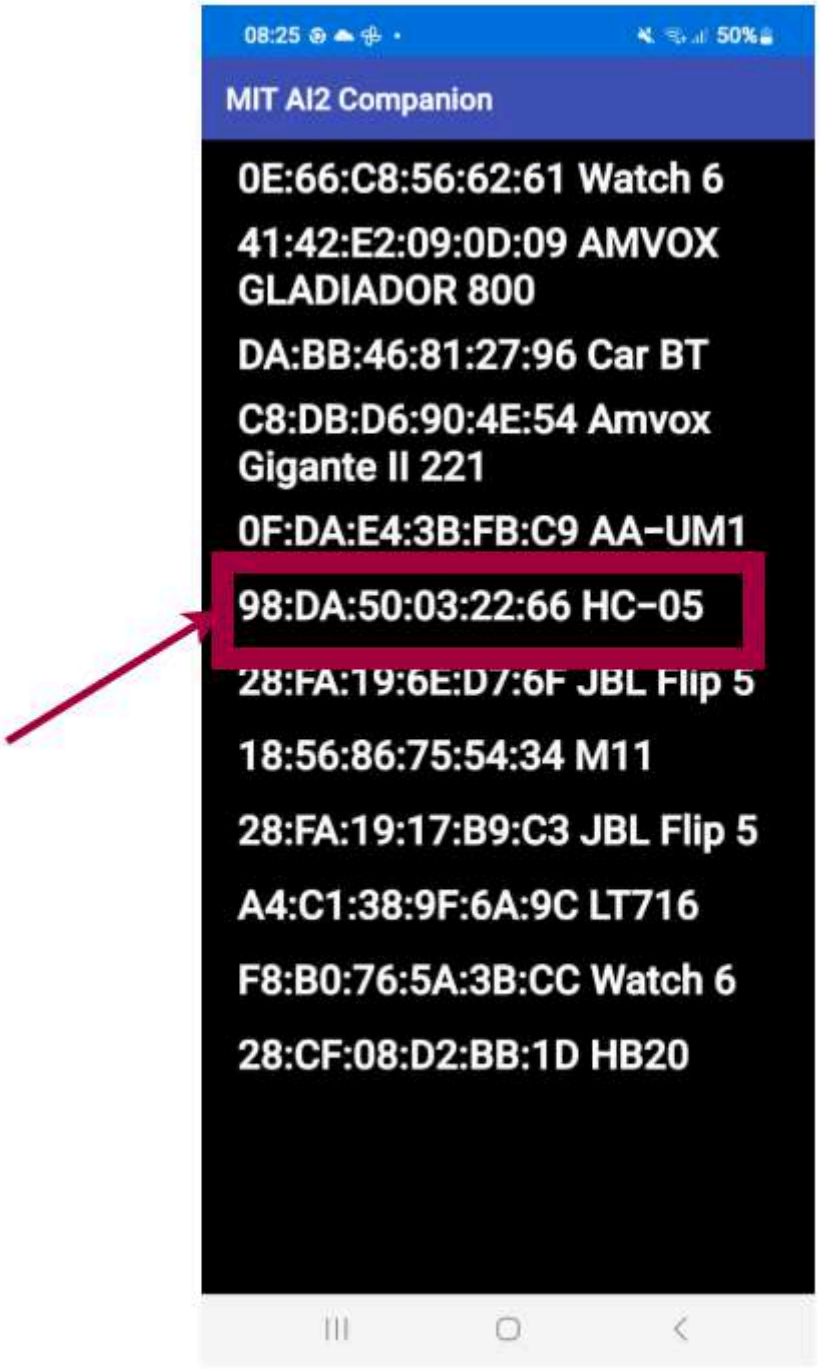
Após ligar a BitDog, vamos conectar no bluetooth:

Abrir o aplicativo do AppInventor, clicar no QRCode que será disponiblizado.

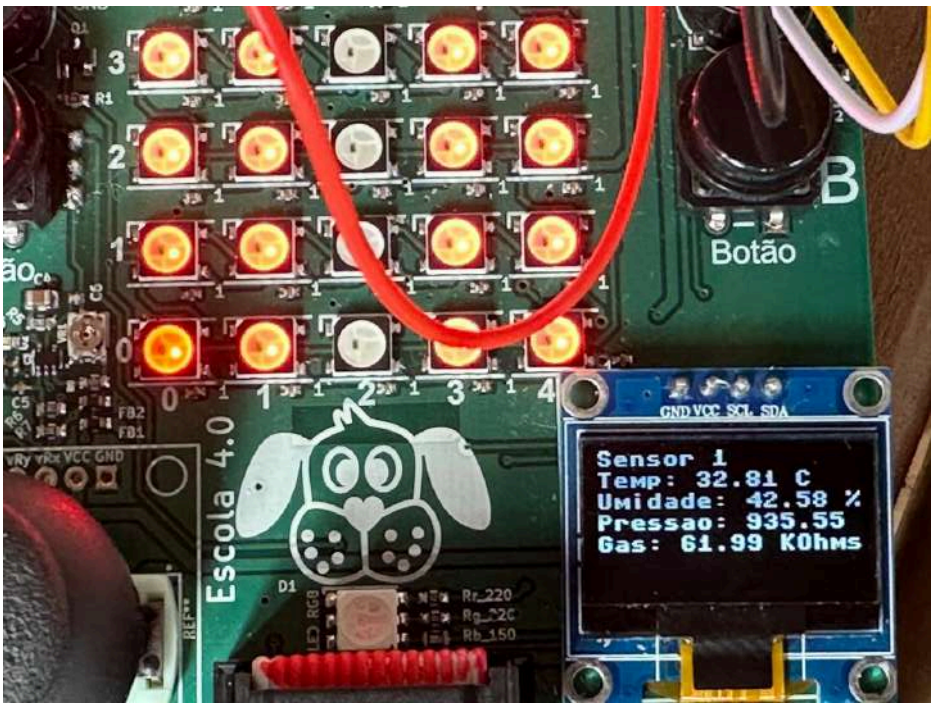


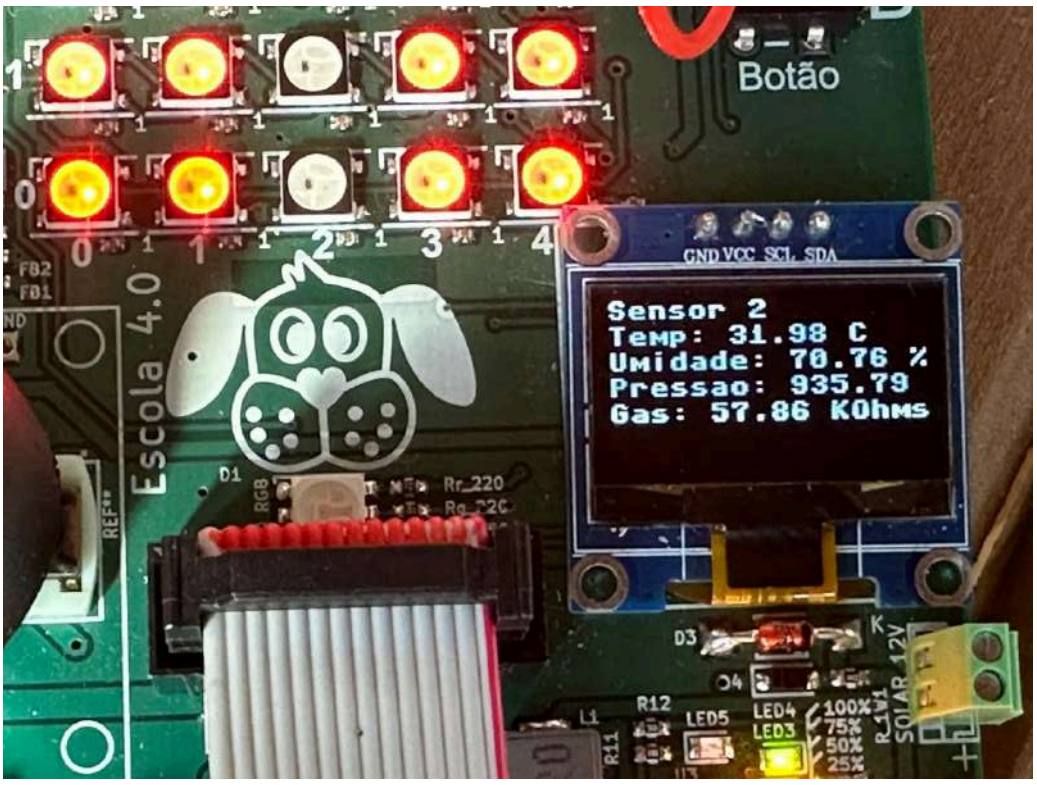
Após clicar no QRCode, esperar alguns segundos para a leitura do aplicativo. Assim que o aplicativo abrir, vai aparecer a tela abaixo.

Clicar em conectar, após isso, vai aparecer uma tela preta com diversos dispositivos, clicar no HC05, feito isso, o app começará a ler os dados que estão vindo dos sensores.


08:25
50%
MIT AI2 Companion
0E:66:C8:56:62:61 Watch 6
41:42:E2:09:0D:09 AMVOX GLADIADOR 800
DA:BB:46:81:27:96 Car BT
C8:DB:D6:90:4E:54 Amvox Gigante II 221
0F:DA:E4:3B:FB:C9 AA-UM1
98:DA:50:03:22:66 HC-05
28:FA:19:6E:D7:6F JBL Flip 5
18:56:86:75:54:34 M11
28:FA:19:17:B9:C3 JBL Flip 5
A4:C1:38:9F:6A:9C LT716
F8:B0:76:5A:3B:CC Watch 6
28:CF:08:D2:BB:1D HB20

Parte 1 - Medir a temperatura antes de levar ao sol (A ideia de medir a temperatura antes de levar no sol é para conseguirmos posteriormente uma boa comparação).


Botão
GND VCC SCL SDA
Sensor 1
Temp: 32.81 C
Umidade: 42.58 %
Pressao: 935.55
Gas: 61.99 KOhms
Escola 4.0


Botão
GND VCC SCL SDA
Sensor 2
Temp: 31.98 C
Umidade: 70.76 %
Pressao: 935.79
Gas: 57.86 KOhms
Escola 4.0
SOLAR 12V

Valores vistos pela Dashboard:

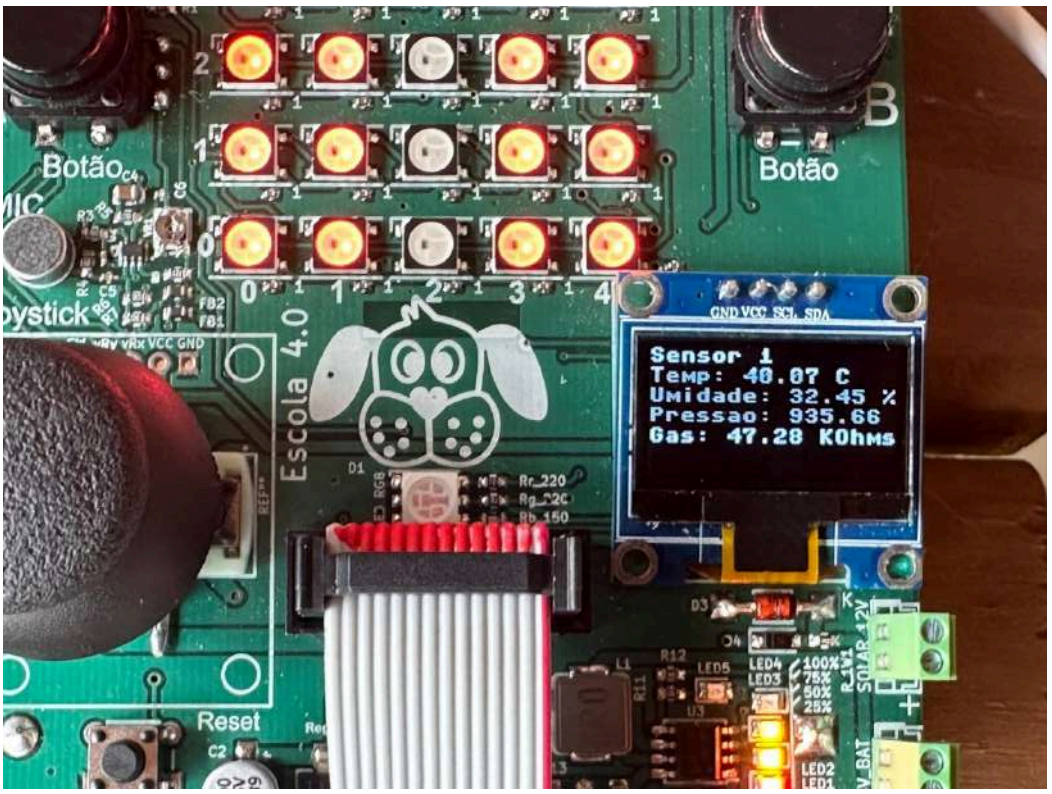
O sensor 1 está no ambiente com brita e o sensor 2 está no ambiente arborizado, podemos perceber que os parâmetros estão bem parecidos o que muda é a umidade, onde no ambiente com brita ela é bem menor.

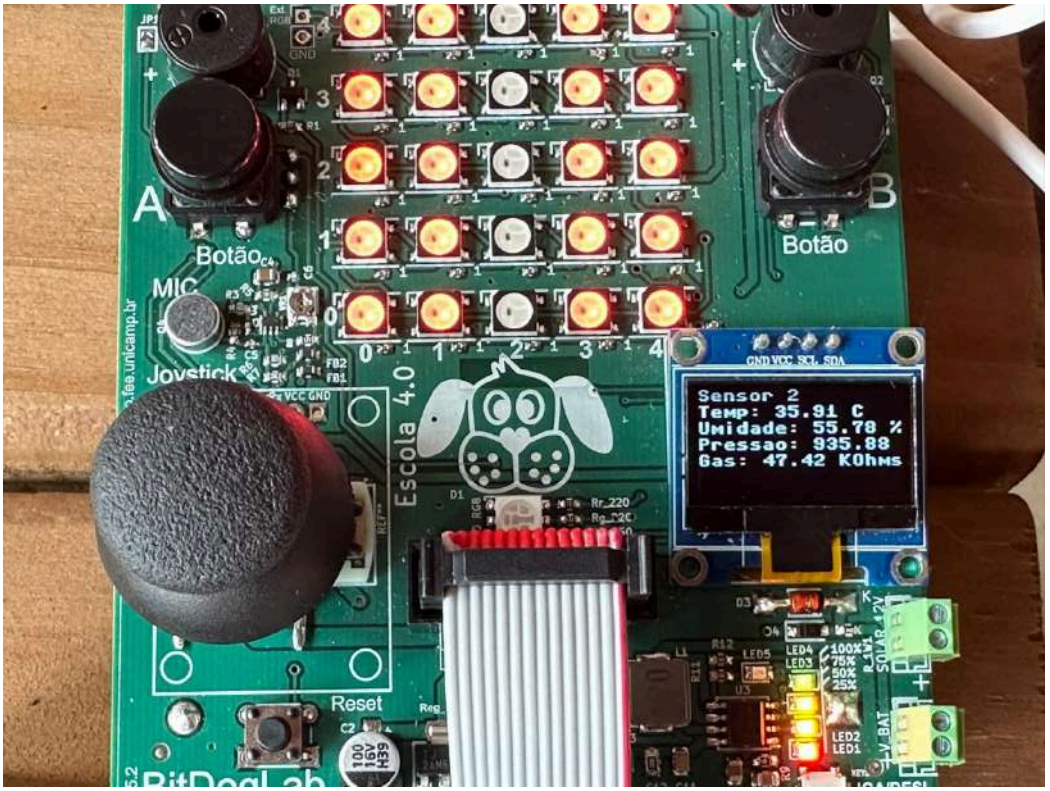
Após a medida inicial, levar as garrafas sem a parte de cima para o sol.

Parte 2 - Levar as garrafas para o sol (A ideia é deixar os dois ambientes ficarem um tempo no sol, para mostrar a importância da preservação da natureza e o problema que é o aquecimento global.)

Minha sugestão é que os ambientes fiquem pelo menos 30 minutos no sol, com isso já é possível fazer uma boa comparação.
Por fim, depois de pelo menos 30 minutos retirar os ambientes do sol, e fazer novamente uma medição.

Parte 3 - Medir as temperaturas novamente, depois de ficar algum tempo no sol.


Botão
Botão
Joystick
Escola 4.0
Reset
GND VCC SCL SDA
Sensor 1
Temp: 40.07 C
Umidade: 32.45 %
Pressao: 935.66
Gas: 47.28 KOhms


A
B
Botão
Botão
MIC
Joystick
Escola 4.0
Reset
GND VCC SCL SDA
Sensor 2
Temp: 35.91 C
Umidade: 55.78 %
Pressao: 935.88
Gas: 47.42 KOhms

Valores vistos pela Dashboard:

Após isso, a temperatura e umidade dos ambientes ficará diferente, onde tem brita ficará com uma temperatura maior, e no ambiente arborizado, ficará com uma temperatura menor, a partir disso, sugiro que nessa parte o professor faça um paralelo com o aquecimento global,e explique a importância da preservação da natureza e de ambientes arborizados.