# 

# 

# P\_IE323A\_2024S2 - Tópicos em Eletrônica(Semipresencial)

Projeto 2

# **Tecnologia de Camaleão: Imitando Cores da Natureza**

Grupo 4

Gelson Ferreira e Wallif Campos

### 

### 

## Resumo:

Este projeto visa desenvolver um sistema que capta e reproduz as cores primárias RGB, inspirado na capacidade de camuflagem dos camaleões, utilizando o sensor de cor RGB TCS34725. O objetivo é permitir que o módulo sensor identifique a cor apresentada e a reproduza em uma matriz de LEDs RGB, exibindo sua composição exata em valores no display OLED. Essa abordagem promove uma experiência interativa no aprendizado de combinação de cores.

## Itens da BNCC contemplados:

* **(EF15CO01)** Identificar formas de organizar e representar a informação de maneira estruturada (como em matrizes, registros) ou não estruturada (como em valores numéricos e cores).
* **(EF15CO06)** Compreender o funcionamento básico de dispositivos computacionais, analisando a interação entre sensores e atuadores.
* **(EF09CI04)** Planejar e executar experimentos que evidenciam que todas as cores de luz podem ser formadas pela composição das três cores primárias da luz e que a cor de um objeto está relacionada também à cor da luz que o ilumina.

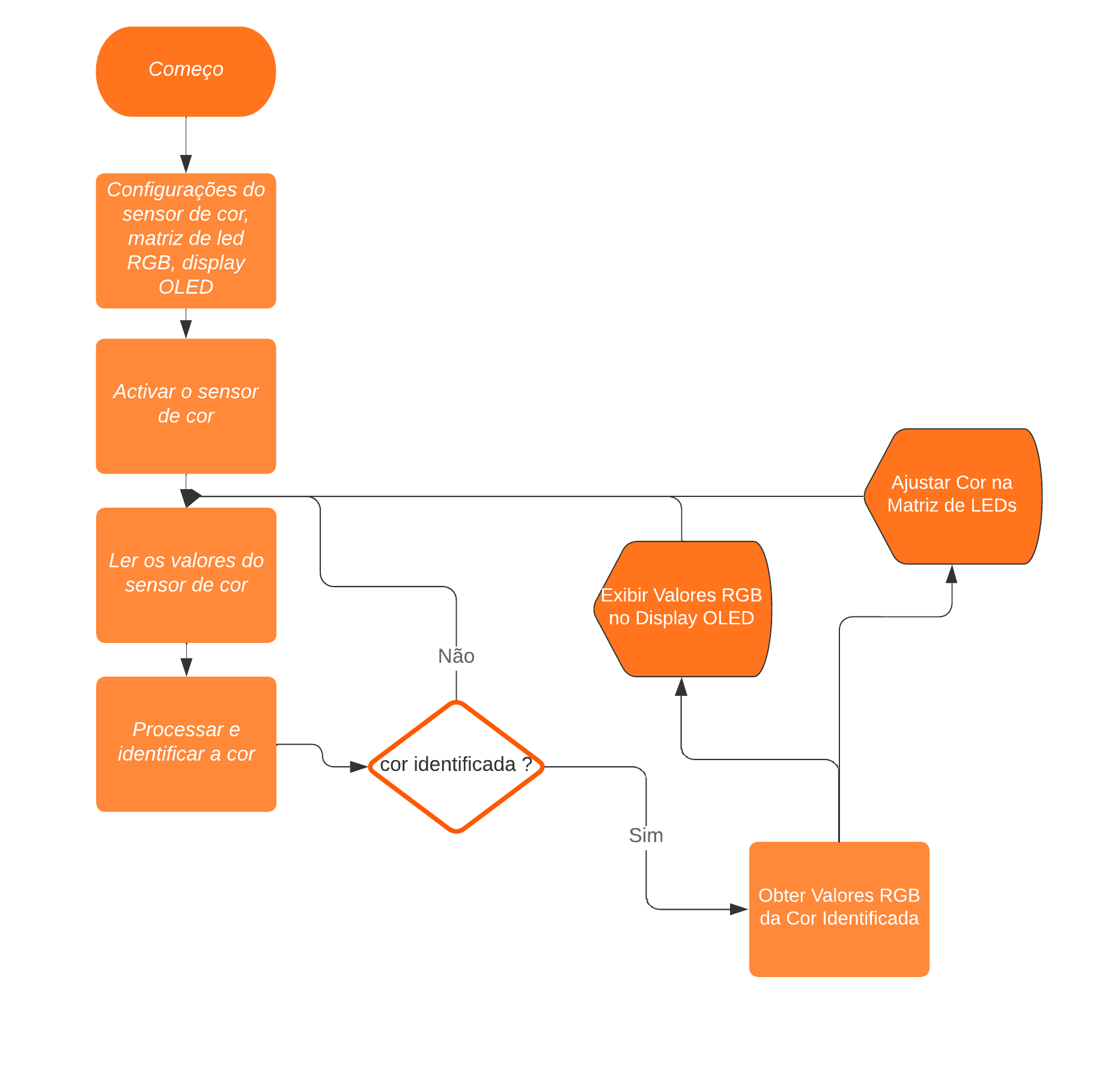
## Recursos de hardware (on e off-board) a serem utilizados:

* **Microcontrolador**: Raspberry Pi Pico
* **Display OLED** (para visualização das informações)
* **Matriz de LEDs RGB WS2812B** (para representar cores ajustadas pelo sistema)
* **Sensor de cor RGB TCS34725** (para detectar a cor de superfícies)

## Descrição simplificada do software:

O software será responsável por controlar o sensor de cor RGB, permitindo a leitura das cores primárias. O processo envolve a captura de quatro valores correspondentes à saturação de vermelho, verde e azul, além da claridade. Com os dados obtidos, o programa identifica a cor detectada pelo sensor e a reproduz na matriz de LEDs RGB. Além disso, os valores RGB exatos serão exibidos no display OLED, proporcionando uma interface clara e informativa. O software foi elaborado em MicroPython na IDE Thonny.

## Fluxograma do código

****

1. **Começo**
   * Inicia a execução do programa.
2. **Configurações Iniciais**
   * Configuração do display OLED.
   * Configuração da matriz de LEDs WS2812B.
   * Configuração do sensor de cor TCS34725.
3. **Ativação do Sensor**
   * Ativação do sensor TCS34725.
   * Definição do tempo de integração e ganho.
4. **Loop Principal**
   * O programa entra em um loop contínuo.
5. **Leitura do Sensor de Cor**
   * Captura os valores de cor do sensor TCS34725 (vermelho, verde, azul e claridade).
6. **Processamento e identificação da cor**
   * Dicionário das cores predefinidas RGB em relação à clareza.
   * Comparar os valores do sensor de cor com as referências de cores predefinidas.
7. **Identificação da Cor**
   * Verifique a cor identificada com base nos cores predefinidas.
   * Se a cor for válida, passe para o próximo bloco.
8. **Ajuste da Matriz de LEDs**
   * Ajuste os LEDs com a cor correspondente à cor identificada (se houver).
9. **Exibição dos Valores no Display OLED**
   * Exibe os valores RGB da cor identificada no display OLED.
10. **Atraso e Repetição**
    * Atraso alguns milissegundos antes da próxima leitura.
    * O loop volta para a leitura do sensor de cor.
11. **Fim (Opcional)**
    * O programa é projetado para funcionar indefinidamente, a menos que seja interrompido.

## Maior desafio do projeto (no seu ponto de vista):

O principal desafio deste projeto foi integrar o sensor de cor ao sistema de controle do Raspberry Pi Pico, assegurando uma leitura precisa das cores e sua reprodução fiel na matriz de LEDs RGB. Também foi necessário enfrentar os desafios de calibração tanto dos LEDs quanto do sensor.

## Apresentação em PowerPoint

<https://docs.google.com/presentation/d/1gn_34yrRUZp3n5dDyKus4PDLKkLMBtjm/edit#slide=id.p6>

## 

## Programa principal comentado

| **import** time *# Importa a biblioteca time para funções de temporização* **from** machine **import** Pin, I2C *# Importa as funções para controle de pinos e comunicação I2C* **import** neopixel *# Importa a biblioteca para controlar a matriz de LEDs WS2812B* **from** tcs34725 **import** TCS34725 *# Importa a classe para controlar o sensor de cor TCS34725* **import** ssd1306 *# Importa a biblioteca para controlar o display OLED* **from** utime **import** sleep\_ms **as** delay *# Renomeia a função de atraso para simplificação*  *# Configuração do display OLED via I2C* i2c\_oled = I2C(1, scl=Pin(15), sda=Pin(14), freq=400000) *# Configura a comunicação I2C para o display* oled = ssd1306.SSD1306\_I2C(128, 64, i2c\_oled) *# Inicializa o display OLED com tamanho 128x64*  *# Configuração da matriz de LEDs WS2812B (Neopixel)* NUM\_LEDS = 25 *# Número de LEDs na matriz (5x5)* LED\_PIN = 7 *# Pino GPIO onde a matriz de LEDs está conectada* leds = neopixel.NeoPixel(Pin(LED\_PIN), NUM\_LEDS) *# Inicializa a matriz de LEDs*  *# Configuração do sensor de cor TCS34725* i2c = I2C(0, scl=Pin(1), sda=Pin(0), freq=400000) *# Configura a comunicação I2C para o sensor de cor* sensor = TCS34725(i2c) *# Inicializa o sensor de cor TCS34725*  *# Valores calibrados para identificação da cor BRANCO* calibR = 0.36 *# Valor de referência para a cor vermelha no branco* calibG = 0.37 *# Valor de referência para a cor verde no branco* calibB = 0.26 *# Valor de referência para a cor azul no branco* margem = 0.02 *# Margem de erro para identificação de cores*  *# Ativando o sensor de cor* sensor.active(**True**) *# Ativa o sensor de cor TCS34725* sensor.integration\_time(50) *# Define o tempo de integração (quanto tempo o sensor capta luz)* sensor.gain(4) *# Define o ganho do sensor (aumenta a sensibilidade para luz fraca)*  delay(500) *# Pequeno atraso para garantir que o sensor esteja pronto*  *# Dicionário para as cores predefinidas* cores\_predefinidas = {  'Branco': (25, 25, 25), *# Branco total*  'Vermelho': (25, 0, 0), *# Vermelho*  'Verde': (0, 25, 0), *# Verde*  'Azul': (0, 0, 25), *# Azul*  'Ciano': (0, 25, 25), *# Ciano (mistura de azul e verde)*  'Magenta': (25, 0, 25), *# Magenta (mistura de vermelho e azul)*  'Amarelo': (25, 25, 0), *# Amarelo (mistura de vermelho e verde)*  'Preto': (0, 0, 0) *# Preto (ausência de luz)* }  **def** **ler\_sensor\_cor**():  """Função para capturar os valores de cor do sensor de cor TCS34725"""  leitura = sensor.read(**True**) *# Faz a leitura do sensor*  r = leitura[0] *# Valor da cor vermelha*  g = leitura[1] *# Valor da cor verde*  b = leitura[2] *# Valor da cor azul*  c = leitura[3] *# Valor da claridade (intensidade total da luz)*  **return** r, g, b, c *# Retorna os valores lidos*  **def** **captura\_cor**(r, g, b, c):  """Processa os valores RGB e retorna a cor aproximada"""  **if** c: *# Verifica se há luz suficiente para identificar a cor*  rp = r / c *# Normaliza o valor de vermelho pela claridade*  gp = g / c *# Normaliza o valor de verde pela claridade*  bp = b / c *# Normaliza o valor de azul pela claridade*    cor = "NA" *# Inicializa a variável da cor com "não identificada"*    *# Define a cor com base nos valores normalizados e na calibração*  **if** c < 400:   cor = "Preto"  **elif** rp <= (calibR + margem) **and** gp <= (calibG + margem) **and** bp <= (calibB + margem):  cor = "Branco"  **elif** rp > (calibR + margem) **and** gp <= (calibG + margem) **and** (bp + 0.06) > (calibB + margem):  cor = "Magenta"  **elif** rp > (calibR + margem) **and** gp <= (calibG + margem) **and** bp <= (calibB + margem):  cor = "Vermelho"  **elif** rp <= (calibR + margem) **and** gp > (calibG + margem) **and** bp <= (calibB + margem):  cor = "Verde"  **elif** rp <= (calibR + margem) **and** gp <= (calibG + margem) **and** bp > (calibB + margem):  cor = "Azul"  **elif** rp <= (calibR + margem) **and** gp > (calibG + margem) **and** bp > (calibB + margem):  cor = "Ciano"  **elif** rp > (calibR + margem) **and** gp > (calibG + margem) **and** bp <= (calibB + margem):  cor = "Amarelo"    delay(250) *# Pequeno atraso para estabilidade*  **else**:  delay(2000) *# Se a claridade for insuficiente, espera mais tempo*  **return** cor *# Retorna a cor identificada*  **def** **ajustar\_matriz\_leds**(r, g, b):  """Ajusta a cor da matriz de LEDs WS2812B com os valores RGB fornecidos"""  **for** i **in** range(NUM\_LEDS):  leds[i] = (r, g, b) *# Define a cor de cada LED da matriz*  leds.write() *# Atualiza a matriz de LEDs com as novas cores*  **def** **exibir\_cor\_oled**(r, g, b):  """Exibe os valores RGB no display OLED"""  oled.fill(0) *# Limpa o display*  oled.text('R: {}'.format(r), 0, 0) *# Mostra o valor de vermelho*  oled.text('G: {}'.format(g), 0, 10) *# Mostra o valor de verde*  oled.text('B: {}'.format(b), 0, 20) *# Mostra o valor de azul*  oled.show() *# Atualiza o display com as informações*  *# Loop principal do programa* **try**:  **while** **True**:  r, g, b, c = ler\_sensor\_cor() *# Lê os valores do sensor de cor*  cor = captura\_cor(r, g, b, c) *# Processa e identifica a cor*    **if** cor **in** cores\_predefinidas:  r\_esperado, g\_esperado, b\_esperado = cores\_predefinidas[cor] *# Obtém os valores RGB da cor identificada*  ajustar\_matriz\_leds(r\_esperado, g\_esperado, b\_esperado) *# Ajusta a cor na matriz de LEDs*  exibir\_cor\_oled(r\_esperado, g\_esperado, b\_esperado) *# Exibe os valores RGB no display OLED*  delay(500) *# Adicionando um pequeno atraso* **except** KeyboardInterrupt:  oled.fill(0) *# Limpa o display*  oled.show() *# Atualiza o display com as informações*  **for** i **in** range(NUM\_LEDS):  leds[i] = (0, 0, 0) *# Define a cor de cada LED da matriz*  leds.write() *# Atualiza a matriz de LEDs com as novas cores*  print("Execução interrompida manualmente.") |
| --- |

### Biblioteca do sensor de cor TCS34725

| **import** utime *# Módulo para funções de temporização* **import** ustruct *# Módulo para manipulação de dados em formato binário*  *# Definição de constantes usadas para configurar o sensor TCS34725* \_COMMAND\_BIT = const(0x80) *# Bit de comando, necessário para acessar registros*  *# Registros do sensor* \_REGISTER\_ENABLE = const(0x00) *# Registro de habilitação do sensor* \_REGISTER\_ATIME = const(0x01) *# Tempo de integração de dados* \_REGISTER\_AILT = const(0x04) *# Limite inferior para interrupções de luz ambiente* \_REGISTER\_AIHT = const(0x06) *# Limite superior para interrupções de luz ambiente* \_REGISTER\_ID = const(0x12) *# Registro de identificação do sensor* \_REGISTER\_APERS = const(0x0c) *# Configuração de persistência da interrupção* \_REGISTER\_CONTROL = const(0x0f) *# Controle do ganho do sensor* \_REGISTER\_SENSORID = const(0x12) *# Identificador do sensor* \_REGISTER\_STATUS = const(0x13) *# Registro de status* \_REGISTER\_CDATA = const(0x14) *# Dados do canal claro (Clear)* \_REGISTER\_RDATA = const(0x16) *# Dados do canal vermelho (Red)* \_REGISTER\_GDATA = const(0x18) *# Dados do canal verde (Green)* \_REGISTER\_BDATA = const(0x1a) *# Dados do canal azul (Blue)*  *# Máscaras de bits para habilitar recursos no sensor* \_ENABLE\_AIEN = const(0x10) *# Habilitar interrupção de luz ambiente* \_ENABLE\_WEN = const(0x08) *# Habilitar detecção de luz ambiente* \_ENABLE\_AEN = const(0x02) *# Habilitar o ADC (Conversor Analógico-Digital)* \_ENABLE\_PON = const(0x01) *# Habilitar o sensor (Power ON)*  *# Valores de ganho suportados pelo sensor* \_GAINS = (1, 4, 16, 60) *# Ganhos possíveis para amplificação dos dados de cor*  *# Ciclos de persistência para interrupção, usados para controle da frequência de interrupções* \_CYCLES = (0, 1, 2, 3, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60)  *# Definição da classe TCS34725, que controla o sensor de cor* **class** **TCS34725**:  **def** **\_\_init\_\_**(self, i2c, address=0x29):  self.i2c = i2c *# Objeto de comunicação I2C*  self.address = address *# Endereço padrão do sensor*  self.\_active = **False** *# Estado de ativação do sensor*  self.integration\_time(2.4) *# Define o tempo de integração padrão para 2.4 ms*  sensor\_id = self.sensor\_id() *# Obtém o ID do sensor*  *# Verifica se o sensor conectado é válido*  **if** sensor\_id **not** **in** (0x44, 0x10, 0x4d):  **raise** RuntimeError("wrong sensor id 0x{:x}".format(sensor\_id))   *# Função para ler ou escrever em um registro de 8 bits*  **def** **\_register8**(self, register, value=None):  register |= \_COMMAND\_BIT *# Adiciona o bit de comando ao endereço do registro*  **if** value **is** **None**:  *# Se nenhum valor for passado, lê o valor do registro*  **return** self.i2c.readfrom\_mem(self.address, register, 1)[0]  *# Caso contrário, escreve o valor no registro*  data = ustruct.pack('<B', value) *# Converte o valor em formato de 8 bits*  self.i2c.writeto\_mem(self.address, register, data)   *# Função para ler ou escrever em um registro de 16 bits*  **def** **\_register16**(self, register, value=None):  register |= \_COMMAND\_BIT *# Adiciona o bit de comando ao endereço do registro*  **if** value **is** **None**:  *# Se nenhum valor for passado, lê dois bytes do registro*  data = self.i2c.readfrom\_mem(self.address, register, 2)  **return** ustruct.unpack('<H', data)[0] *# Desempacota os dados em formato de 16 bits*  *# Caso contrário, escreve o valor no registro*  data = ustruct.pack('<H', value) *# Converte o valor em formato de 16 bits*  self.i2c.writeto\_mem(self.address, register, data)   *# Ativa ou desativa o sensor*  **def** **active**(self, value=None):  **if** value **is** **None**:  **return** self.\_active *# Retorna o estado atual do sensor*  value = bool(value) *# Converte o valor para booleano*  **if** self.\_active == value:  **return** *# Se o estado atual já for o desejado, não faz nada*  self.\_active = value *# Atualiza o estado*  enable = self.\_register8(\_REGISTER\_ENABLE) *# Lê o registro de habilitação*  **if** value:  *# Se o sensor deve ser ativado, habilita o sensor e o ADC*  self.\_register8(\_REGISTER\_ENABLE, enable | \_ENABLE\_PON)  utime.sleep\_ms(3) *# Espera 3 ms*  self.\_register8(\_REGISTER\_ENABLE,  enable | \_ENABLE\_PON | \_ENABLE\_AEN)  **else**:  *# Se deve ser desativado, desliga o sensor*  self.\_register8(\_REGISTER\_ENABLE,  enable & ~(\_ENABLE\_PON | \_ENABLE\_AEN))   *# Obtém o ID do sensor*  **def** **sensor\_id**(self):  **return** self.\_register8(\_REGISTER\_SENSORID)   *# Define ou obtém o tempo de integração*  **def** **integration\_time**(self, value=None):  **if** value **is** **None**:  **return** self.\_integration\_time *# Retorna o tempo de integração atual*  *# Limita o valor de tempo de integração entre 2.4 e 614.4 ms*  value = min(614.4, max(2.4, value))  cycles = int(value / 2.4) *# Converte o tempo em ciclos*  self.\_integration\_time = cycles \* 2.4 *# Armazena o tempo de integração calculado*  **return** self.\_register8(\_REGISTER\_ATIME, 256 - cycles) *# Define o valor no registro ATIME*   *# Define ou obtém o ganho do sensor*  **def** **gain**(self, value):  **if** value **is** **None**:  **return** \_GAINS[self.\_register8(\_REGISTER\_CONTROL)] *# Retorna o ganho atual*  **if** value **not** **in** \_GAINS:  **raise** ValueError("gain must be 1, 4, 16 or 60") *# Verifica se o valor é válido*  **return** self.\_register8(\_REGISTER\_CONTROL, \_GAINS.index(value)) *# Define o ganho*   *# Verifica se os dados são válidos*  **def** **\_valid**(self):  **return** bool(self.\_register8(\_REGISTER\_STATUS) & 0x01)   *# Lê os dados de cor do sensor*  **def** **read**(self, raw=False):  was\_active = self.active() *# Armazena o estado atual do sensor*  self.active(**True**) *# Ativa o sensor*  **while** **not** self.\_valid():  *# Aguarda até que os dados estejam disponíveis*  utime.sleep\_ms(int(self.\_integration\_time + 0.9))  *# Lê os dados dos canais de cor*  data = tuple(self.\_register16(register) **for** register **in** (  \_REGISTER\_RDATA,  \_REGISTER\_GDATA,  \_REGISTER\_BDATA,  \_REGISTER\_CDATA,  ))  self.active(was\_active) *# Restaura o estado anterior do sensor*  **if** raw:  **return** data *# Retorna os dados crus*  **return** self.\_temperature\_and\_lux(data) *# Calcula e retorna a temperatura de cor e o lux*   *# Calcula a temperatura de cor e a luminosidade*  **def** **\_temperature\_and\_lux**(self, data):  r, g, b, c = data *# Desempacota os dados de cor*  *# Converte os dados de cor em XYZ (modelo de cor usado para calcular a temperatura de cor)*  x = -0.14282 \* r + 1.54924 \* g + -0.95641 \* b  y = -0.32466 \* r + 1.57837 \* g + -0.73191 \* b  z = -0.68202 \* r + 0.77073 \* g + 0.56332 \* b  d = x + y + z *# Soma dos valores de cor*  *# Calcula o valor do nCIE (temperatura de cor correlata)*  n = (x / d - 0.3320) / (0.1858 - y / d)  *# Calcula a temperatura de cor correlata em Kelvin*  cct = 449.0 \* n\*\*3 + 3525.0 \* n\*\*2 + 6823.3 \* n + 5520.33  **return** cct, y *# Retorna a temperatura de cor e a luminosidade (lux)* |
| --- |

## Manual simplificado para o usuário final

### Tecnologia de Camaleão Imitando Cores da Natureza

#### 1. Introdução

Este projeto simula a capacidade de camuflagem dos camaleões, imitando as cores da natureza mediante um sistema embarcado. Utilizando um sensor de cor TCS34725, o sistema identifica a cor primária RGB de um objeto e reproduz essa cor em uma matriz de LEDs RGB WS2812B (Neopixel). Um display OLED exibe os valores exatos das cores (R, G, B) captada pelo sensor, proporcionando uma interface visual clara e informativa.

O projeto será desenvolvido utilizando a plataforma BitDogLab com o microcontrolador Raspberry Pi Pico, programado em Python no ambiente MicroPython usando o IDE Thonny.

#### 2. Objetivo

O objetivo deste sistema é capturar cores RGB primários usando um sensor de cor e reproduzi-las em uma matriz de LEDs RGB. Ao mesmo tempo, os valores RGB exatos são exibidos em um display OLED, promovendo uma experiência interativa e educacional no aprendizado de combinação de cores.

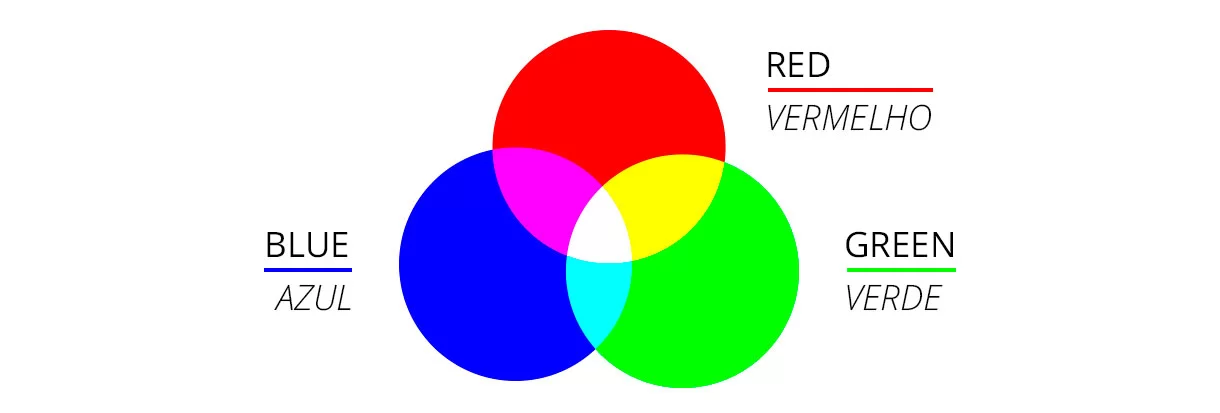


Figura 1: Cores Primárias RGB

#### 3. Conexões e configurações de hardware

* **Matriz de LEDs WS2812B (Neopixel)** :
  + Pino de entrada (IN) conectado ao **GPIO7** do Raspberry Pi Pico.
* **Tela OLED (128x64)** :
  + Pino **SDA** conectado ao **GPIO14** e **SCL** ao **GPIO15** (I2C1).
* **Sensor de cor TCS34725** :
  + Pino **SDA** conectado ao **GPIO0** e **SCL** ao **GPIO1** (I2C0).

#### 4. Componentes Utilizados

* **Raspberry Pi Pico** com MicroPython instalado.
* **Matriz de LEDs WS2812B (5x5)** para exibir as cores da combinação primária RGB.
* **Display OLED 128x64** para exibir os valores RGB capturados.
* **Sensor de cor TCS34725** para capturar cores.

#### 5. Instruções de como instalar o jogo

Passo 1: Instalar a IDE Thonny

Baixe e instale o IDE Thonny a partir do site oficial.

Passo 2: Configurar IDE thonny com o MicroPython ( Raspberry PI Pico)

Abra o IDE Thonny e clique no ícone de onde tem configure interpreter (localizado no lado direito abaixo).

Seleciona MicroPython ( Raspberry PI Pico)

Seleciona a porta do microcontrolador

Copie o código do repositório pretendido e salve na opção que vai aparecer Raspberry com nome main.py e faça o mesmo no código da biblioteca, mas com o nome de TCS34725.py. Caso já tenha um código salvo, apenas substitua.

Passo 3: Enviar o Código para o plataforma Bitdoglab

No IDE Thonny, digitando o F5 para enviar e correr com programa no Bitdoglab ou clica no botão run.

Passo 4: Ver o Código Funcionando

Após enviar o código, o Pico vai rodar o main.py automaticamente. Se tudo estiver certo, o programa vai começar.

#### 6. Explicação do Código

1. **Importação das Bibliotecas** : O código utiliza bibliotecas como machine para controle do hardware, neopixel para controlar a matriz de LEDs, tcs34725 para o sensor de cor e ssd1306 para o display OLED.
2. **Configuração do OLED e Sensor de Cor** : O OLED está configurado no canal I2C1, enquanto o sensor de cor utiliza o canal I2C0. O sensor de cor é inicializado com tempo de integração e ganho definido.
3. **Funções Principais** :
   * ler\_sensor\_cor: Lê os valores RGB e a clareza (intensidade de luz) capturados pelo sensor.
   * captura\_cor(r, g, b, c): Processa e identifica a cor
   * ajustar\_matriz\_leds: Ajusta a matriz de LEDs para reproduzir a cor detectada.
   * exibir\_cor\_oled: Exibe os valores RGB no display OLED.
   * loop\_principal: Loop contínuo que realiza a leitura dos valores, normaliza os dados e os exibe tanto na matriz de LEDs quanto no OLED.
4. **Dicionário de cores predefinidas do RGB primários** : para garantir que a cor exibida seja precisa, os valores RGB são ajustados com base na clareza capturada pelo sensor (analisando os valores de R, G e B pelo valor de C).
5. **Loop Infinito** : O programa funciona em um loop infinito, capturando e exibindo as cores continuamente.

**Link deste documento Google doc:**

<https://docs.google.com/document/d/1lmVPA4vaM4aL9rKiDlJza7ejJ8PD7WprtfyfEPbnlH8/edit?usp=sharing>

## Referência Bibliográfica

https://www.youtube.com/watch?v=GF4DfSiGFNE

https://www.afixgraf.com.br/blog/o-que-significa-rgb/