Aplicação do sensor de cor TCS3200 para auxílio de deficientes visuais

Felipe Coutinho & Rita Duarte
Projeto EDM 2021-22

30 de junho de 2022

Conteúdo

1 Introdução

1.1 Proposta

O tema do trabalho abordado é a leitura de um sensor de cor TCS3200. Assim, enquadrou-se o projeto como uma ferramenta de auxílio para deficientes visuais. A proposta consiste num protótipo de aplicação web que a partir da leitura do sensor, nomeia a cor correspondente por via de áudio - funcionalidade esta obtida a partir da API VoiceRSS para conversões *text-to-speech*. Desse modo, pessoas com daltonismo ou cegueira poderão identificar a cor de produtos que as interessem em tempo real.

1.2 Sensor TCS3200

O princípio de funcionamento do sensor utilizado é a conversão de um sinal luminoso em elétrico a partir de matrizes de fotodíodos.

O sensor dispõe de 4 modos de operação: sem filtro, filtro vermelho, filtro verde e filtro azul. Logo, através de 4 medições assíncronas, acoplando os diferentes filtros, pode-se quantificar cada componente de cor RGB a partir das respectivas intensidades luminosas - reconstruindo a cor do objeto medido. Notar-se que a escolha do filtro é feita através de 2 pinos de seleção, ver 1

Entretanto, a saída do sensor é digital: consiste num sinal PWM, cuja frequência é proporcional a intensidade. Logo, esta deve ser medida com recurso ao microcontrolador. Por exemplo, como discutido nas aulas teóricas, pode-se contar o número de subidas do sinal num certo intervalo de aquisição.

O sensor dispõe ainda uma configuração adicional, que permite re-escalar a frequência do sinal de saída. Esta opção é útil, uma vez que a frequência máxima mensurável com o microcontrolador é limitada pelo tempo de execução das rotinas utilizadas. Sendo assim, escolheu-se trabalhar re-escalando as frequências para 20 % do valor original, de modo a adequar-se à taxa de aquisição mantendo a maior precisão possível.

1.3 LED RGB

Utilizou-se ainda, para efeito de verificação e visualização das leituras, um LED RGB com ânodo comum. Este consiste essencialmente em 3 LEDs, um de cada cor, agrupados num único encapsulamento. Assim, através de 3 pinos diferentes, com sinais PWM adequados pode-se modular qualquer cor desejada. É necessário limitar a corrente nos LEDs, portanto colocaram-se resistências de 220Ω em cada um deles.

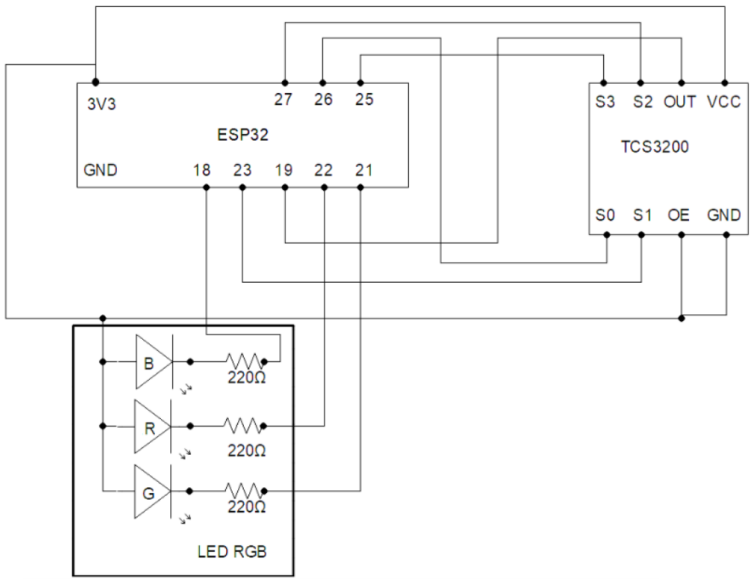

Figura 1: Esquema do circuito utilizado. S0 e S1 permite escolher a escala da frequência de output (neste caso 20%), enquanto S2 e S3 selecionam o fotodíodo utilizado para se medir a intensidade de cada uma das cores (vermelha, verde, azul ou a intensidade total) e assim caracterizar a luz recebida.

2 Implementação das rotinas de leitura

Vai-se descrever as subrotinas envolvidas no processo de medição e determinação da cor. O programa foi estruturado de modo a ler continuamente o sensor enquanto aguarda *inputs* por parte do usuário: solicitar calibração, atualizar LED e página web, etc.

2.1 Medição de frequência

A medição da frequência foi implementada nos moldes apresentados nas aulas teóricas. Ou seja, fez-se recurso a um *Timer* e um *interrupt* do tipo *irq*. Sempre que o sinal de saída do sensor sobe, incrementa-se um contador. Ao fim de Δt segundos, a frequência é estimada como $f = \frac{N}{\Delta t}$, onde N é o número total de contagens. Assim, tem-se controle sobre a taxa de amostragem e evitam-se erros associados a contagens "perdidas"enquanto o microcontrolador ocupava-se com outras tarefas.

```
(...)
sensor_signal = Pin(19, Pin.IN) # Sensor output
(...)
def increment(tag):
```

```
    global current_count
    current_count += 1
(...)
sensor_signal.irq(trigger=Pin.IRQ_RISING, handler=increment)
(...)
```

Assim, fez-se um ciclo que permuta entre os filtros possíveis, mede a frequência associada e ao fim de aproximadamente $4\Delta t$ produz um conjunto de valores (C, R, G, B) dos quais extrair-se-á informação sobre a cor.

```
def select_filter(mode):
    (...)
    if mode == "red":
        s2.value(0)
        s3.value(0)
    elif mode == "green":
        s2.value(1)
        s3.value(1)
    elif mode == "blue":
        s2.value(0)
        s3.value(1)
    elif mode == "clear":
        s2.value(1)
        s3.value(0)
```

```
def get_measurement(tag):
    (...)
    # Get measurement
    measurement = current_count/measure_interval # kHz
    if tag == measure_timer:
        (...)
    elif tag == calib_timer:
        (...)
    # Cycle through filters
    filter_index = (filter_index + 1)%4
    select_filter(filters[filter_index])
    # Reset counter
    current_count = 0
```

2.2 Calibração do sensor

Para converter as frequências em valores RGB, é preciso calibrar o sensor estabelecendo referências para o negro (0, 0, 0) e o branco (255, 255, 255). Para isso, fez-se uma rotina que realiza o processo de calibração sob demanda, devendo o usuário posicionar objetos com as cores indicadas para que se estabeleça as referências. Uma vez obtido os valores de referência, tem-se para o valor RGB x_i:

$$x_i = \text{int}\left(255 \times \frac{f_i^0 - f_i^{dark}}{f_i^{white} - f_i^{dark}}\right) \tag{1}$$

...onde f_i^0 é a frequência medida com o filtro i e f_i^{dark} e f_i^{white} são as frequências obtidas pela calibração. [1]

Outro ponto sobre a implementação é que alterna-se entre dois *Timers*, um para as medições associadas a calibração e outro para as medições comuns. Isso foi feito pois a lógica a ser aplicada aos resultados das medições em cada um dos casos é diferente.

```
(...)
# Timers for data acquisition
```

```
measure_timer = Timer(1) # Measurement cycle
calib_timer = Timer(2) # Calibration cycle
(...)
async def loop():
    measure_timer.init(period=measure_interval,
                       mode=Timer.PERIODIC,
                       callback=get_measurement) # Begin measurement
    while True:
        command = (...)
        # Calibration
        if command == "c":
            measure_timer.deinit() # Stop normal measurement
            calibration() # Proceed with calibration
            measure_timer.init(period=measure_interval,
                               mode=Timer.PERIODIC,
                               callback=get_measurement) # Resume measurement
        (...)
    (...)
(...)
```

2.3 Ativação do LED

Também pode-se solicitar que a cor seja reproduzida no LED através do comando "a". Notar que este comando também envia as informações para a parte web, pelo que tem dupla funcionalidade.

Focando-se na primeira delas, o que é feito consiste no mapeamento dos valores RGB x para valores de duty-cycle d a serem usados nos sinais PWM enviados para o LED. A expressão utilizada para este efeito é dada em (2)

$$d_i = \text{int}\left(\frac{1023}{255}x_i\right) \tag{2}$$

Como a intensidade luminosa de cada LED era diferente das restantes, em algumas cores, como o laranja ou o púrpura, o resultado da mistura dos 3 LEDS deixou um bocado a desejar.

Para se conseguirem resultados melhores, obtiveram-se as intensidades luminosas pela *datasheet* [5], (2800, 6500, 1200) mcd para (R,G,B) respetivamente. Tendo isso, reduziu-se essa proporção para $\frac{28}{65} : 1 : \frac{12}{65}$ e multiplicou-se cada duty-cycle pela sua fração. O duty-cycle final terá que ser a diferença de 1023 pelo valor inteiro do resultado anterior por se tratarem de LEDs ativos ao nível baixo.

A utilização do LED pode ser desativada através do comando "d". É importante notar que seu uso pode atrapalhar futuras medições, pois a luz emitida será detectada em parte pelo sensor.

3 Implementação web

3.1 Conexão a rede Wi-Fi

Na inicialização do programa, faz-se a conexão a rede local utilizando-se a rotina demonstrada nas aulas.

```
def do_connect():
    essid = (...)
    password = (...)
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)
    print(wlan.scan())
    if not wlan.isconnected():
        print('connecting to network...')
        wlan.connect(essid, password)
        while not wlan.isconnected():
            pass
    print('network config:', wlan.ifconfig())
```

3.2 Comunicação com a API VoiceRSS

A API [4] utilizada permite a conversão de *strings* em fala artificial. Para tal, faz-se um pedido *GET* através do protocolo *http*, sendo retornado um ficheiro de áudio no formato escolhido.

Em termos de implementação, escreveram-se funções auxiliares para obter o nome da cor medida e aplicar a *querystring* adequada no pedido *GET*. O ficheiro retornado pela API é por sua vez guardado internamente no microcontrolador no formato *mp3*. Claro que isto exige que não se trabalhe com ficheiros de áudio demasiado grandes; para os fins deste trabalho, não há qualquer problema em termos de memória pois tratam-se de áudios muito curtos.

```
def parseQuerystring(url, qs):
    query = f"?key={qs['key']}&src={qs['src']}"
    query = query + f"&hl={qs['hl']}&r={qs['r']}&c={qs['c']}&f={qs['f']}"
    return url + query

def convertTTS(name):
    (...)
    url = "http://api.voicerss.org/"
    querystring = {
                    "key":(...),
                    "src":name, # String to be converted
                    "hl":"en-us", # Language to be used
                    "r":"0",
                    "c":"mp3", # Audio format
                    "f":"8khz_8bit_mono" # Audio quality
```

```
                }
    parsed = parseQuerystring(url, querystring)
    api_response = urequests.request("GET", parsed)
    (...)
    with open("tts.mp3", mode="wb") as file:
        (...)
        file.write(api_response.content)
    (...)

def updatePage(client, rgb_values):
    (...)
    name = getColorName(rgb_values)
    # Generate TTS audio
    convertTTS(name)
    # Update HTML with new color
    index = html.find("rgb")
    new_rgb = f"rgb({rgb_values[0]:03d}, {rgb_values[1]:03d}, {rgb_values[2]:03d})"
    html = html[:index] + new_rgb + html[index+18:]
    # Send page HTML
    (...)
    client.send('HTTP/1.0 200 OK content-type: text/html\r\n\r\n')
    client.send(html)
    (...)
    # Send audio file
    (...)
    client.send('HTTP/1.0 200 OK content-type: audio/mpeg')
    with open("tts.mp3", "rb") as file:
        audio = file.read()
        client.send(audio)
    (...)
    client.close()
```

3.3 Comunicação via *socket*

Para tornar acessível os resultados, faz-se uma conexão via *socket* com um cliente também conectado na rede local. Por via desta, envia-se um ficheiro HTML, contendo a informação sobre a página web e de seguida, o ficheiro *mp3*; os dois elementos principais da página são um quadrado preenchido com uma cor sólida (a ser atualizado com as cores medidas) e um *player* (para o ficheiro de áudio obtido com a API).

Esta estrutura não é ideal, mas funciona com alguns pormenores. Em particular, o elemento de HTML utilizado para inclusão do *player* exige referência ao ficheiro de áudio. Contudo, este não se encontra disponível para o cliente a princípio, sendo recebido a posteriori.

```
<!DOCTYPE html>
<html>
```

```
  <head> <title>Color sensor</title> </head>
  <body>
    (...)
    <style>
    square {
      height: 50px;
      width: 50px;
      background-color: rgb(255, 099, 071);
    }
    </style>
  <body>
    <h2>Color</h2>
    <div class="square"></div>
  </body>
  <body>
    (...)
    <audio autoplay controls>
      <source src="tts.mp3" type="audio/mpeg">
      (...)
    </audio>
  </body>
  (...)
</html>
```

O pedido para a atualização da página foi incorporado no comando "a", discutido na secção 2.3.

Os testes do código foram feitos utilizando o navegador Opera, e validaram o funcionamento básico do sistema. Contudo, por vezes é necessário repetir o comando entre 2-3 vezes para que o áudio torne-se disponível na página. Para gerir a conexão via *socket*, incluíram-se funcionalidades do módulo *uasyncio*. Isso porque esta estrutura exige constantemente escutar por novas conexões por parte do cliente, sem que o restante do programa fique em suspensão.

Assim, sempre que não existir um cliente já conectado, o microcontrolador tentará aceitar novas conexões. Sempre que existir um cliente conectado, o programa deixa de escutar novas conexões. Após utilizar um *socket*, o programa o encerra.

Esta implementação acabou por implicar, por vezes, na necessidade de recarregamentos da página no navegador após usos do comando "a".

Estes elementos não são ideais, porém funcionam suficientemente bem para que o protótipo seja apresentado. Tendo em conta a falta de conhecimentos para implementar um servidor web mais robusto, optou-se por manter a estrutura acima.

Figura 2: Esquema de uma roda de 24 cores.

3.4 Cálculo do valor *hue* e nomeação das cores

Para obter o nome das cores associado a cada triplo (R, G, B), calculou-se a coordenada *hue* da cor medida no espaço de cores HSV. Como os valores de *hue* tomam valores entre $[0, 360]$ graus, mapeiam continuamente as cores numa circunferência. Assim, a discretizando pode-se criar intervalos correspondendo a cada cor, como representado na figura 2

Não se utilizou a expressão encontrada no artigo [2], hue $= \frac{\sqrt{3}(G-B)}{2R-G-B}$, pois seria necessário utilizar funções não lineares, o que torna o código menos eficiente. Uma alternativa mais otimizada do que o método anterior foi obtida em [3].

Antes de se realizar a conversão, verifica-se que não se trata nem de branco (todos os valores acima de 230) nem de preto (todos os valores abaixo de 20). Se sim devolve-se de imediato a cor correspondente, se não prossegue-se com o que foi planeado.

Estando os valores RGB entre 0 e 1, obtém-se a componente máxima (mx) e mínima (mn) da cor e calcula-se a diferença (cr) para se definir o intervalo dos valores.

Redefiniram-se os valores RGB para facilitar os cálculos (agora R1,G1,B1) e, sabendo qual das 3 é a componente máxima, define-se *hue* pela sua posição no círculo de cores numa escala de 0 a 1. Na identificação do máximo foi necessário realizar a comparação na forma $|mx - X| <$ (, sendo X um dos termos RGB,) para não acontecer nenhum erro relacionado com a comparação de números decimais.

```
R1 = (( (mx - R) / 6 ) + (cr / 2)) / cr
G1 = (( (mx - G) / 6 ) + (cr / 2)) / cr
B1 = (( (mx - B) / 6 ) + (cr / 2)) / cr
if  abs(R - mx) < eps:
```

```
    hue = B1-G1
elif abs(G - mx) < eps:
    hue = 1/3 + R1 - B1
elif abs(B - mx) < eps:
    hue = 2/3 + G1 - R1
```

Tendo obtido *hue*, rescala-se o seu valor para um número inteiro do intervalo de [0,360]. Para facilitar a discretização das cores, ou seja, para que o intervalo da cor vermelha começasse em 0°, deslocámos os valores em 15°.

Para se aliar a especificidade à simplicidade, foram definidos 12 cores diferentes de Hue com o mesmo intervalo (30°):

```
colornames = {0 : "red",
              1 : "orange",
              2 : "yellow",
              3 : "chartreuse",
              4 : "green",
              5 : "turquoise",
              6 : "cyan",
              7 : "azure",
              8 : "blue",
              9 : "purple",
              10 : "magenta",
              11 : "crimson"}
```

```
def getColorName(RGB_values):
    (...)
    hue = getHue(RGB_values)
    if type(hue) is not int:
        # Either black or white was detected
        name = hue
        return name
    else:
        # Shift for fitting defined color intervals
        hue = (hue + 15)%360
        # Select which interval hue falls into
        index = int(hue*12/360)
        # Pick appropriate color
        name = colornames[index]
        return name
```

4 Conclusão

Cumpriu-se o objetivo principal deste trabalho. Através de um sensor de cor TCS3200 e de uma API com a funcionalidade *text-to-speech* (API VoiceRSS) consegue-se identificar a cor do objeto à frente do sensor e reproduzir pela página web o nome da cor. Ainda foi possível controlar a cor de um LED RGB e a de um quadrado na mesma página web através dos valores lidos por esse sensor.

Provavelmente seria necessário escolher resistências mais adequadas para o LED RGB de forma a que a cor transmitida se aproxime mais da do objeto. Também seria adequado alterar as categorias das cores escolhidas para evitar ambiguidades nos valores próximos de 2 grupos diferentes. Outro ponto crucial que seria necessário modificar é o layout da página web, assim como outros aspetos da mesma.

Referências

[1] TCS230/TCS3200 Sensor Calibration

[2] Frank Preucil, "Color Hue and Ink Transfer ... Their Relation to Perfect Reproduction, TAGA Proceedings, p 102-110 (1953).

[3] InAffinity for Affinity Photo. "How is Hue Calculated" YouTube, 2 Mar. 2021, https://youtu.be/BxEsEXsOJyA.

[4] VoiceRSS API

[5] LED 5mm RGB Diffused Common Anode