

Instituto Superior Técnico

Licenciatura em Engenharia Eletrotécnica e de Computadores

## Sistemas Eletrónicos

2023/2024 - Segundo Semestre (3º Período)

### LAB 3 - $\mu$ Oscilloscope

Osciloscópio através de um sistema embebido

Grupo 50

António Vasco Morais de Carvalho (102643)  
Catarina Andreia Duarte Caramalho (102644)  
João Pedro Mendes Pinheiro (102709)

[antonio.v.morais.c@tecnico.ulisboa.pt](mailto:antonio.v.morais.c@tecnico.ulisboa.pt)  
[catarina.caramalho@tecnico.ulisboa.pt](mailto:catarina.caramalho@tecnico.ulisboa.pt)  
[joaoppinheiro@tecnico.ulisboa.pt](mailto:joaoppinheiro@tecnico.ulisboa.pt)

Auto-avaliação: 18/20

Dia da semana: Terça-feira  
Turno: 10:00 - 11:30

Docente: Diogo Miguel Caetano

# **Conteúdo**

<b>1</b>	<b>Introdução</b>	<b>1</b>
<b>2</b>	<b>Programa desenvolvido</b>	<b>1</b>
<b>3</b>	<b>Testes no simulador</b>	<b>3</b>
<b>4</b>	<b>Testes no laboratório</b>	<b>5</b>
4.1	Não calibrado . . . . .	5
4.2	Ajuste experimental . . . . .	7
4.3	Calibrado . . . . .	8
<b>5</b>	<b>Conclusões</b>	<b>10</b>
	<b>Apêndices</b>	<b>12</b>
<b>A</b>	<b>Ficheiro de código desenvolvido completo</b>	<b>12</b>
<b>B</b>	<b>Funcionalidade do envio do email</b>	<b>15</b>

## 1 Introdução

No presente trabalho de laboratório pretende estudar-se sistemas embebidos, ainda que superficialmente, os quais correspondem a uma combinação de um *hardware* computacional e um *software* projetado para uma função específica. Para tal utilizar-se-á um módulo IoT composto por uma placa micro-controladora com diversas funcionalidades e interfaces e capaz de funcionar como um osciloscópio com um *software* desenvolvido para tal. Com isto, definem-se os objetivos do trabalho: Aprendizagem do circuito de *hardware* base para a realização do trabalho; Desenvolvimento do *software* para implementação do  $\mu$ Oscilloscope; Teste e ajuste do *software*; Calibração do  $\mu$ Oscilloscope.

Primeiramente, explica-se o [Programa desenvolvido](#) pelo grupo (`main.py`) com o qual se fizeram [Testes no simulador](#) previamente às aulas de laboratório. De seguida apresentam-se os [Testes no laboratório](#) onde se esperam ver incongruências na representação dos sinais com a ausência de calibração, mas que após um ajuste experimental, isto é, uma calibração experimental, se espera que essas incongruências sejam erradicadas quase totalmente (sempre sujeito a imperfeições experimentais).

## 2 Programa desenvolvido

De modo a cumprir as funções especificadas no enunciado, tomaram-se como base os *scripts* fornecidos pelos docentes (`main_exemplo_1.py` e `main_exemplo_2.py`) para o desenvolvimento do programa `main.py`.

Nesta secção serão apresentadas porções do código consideradas mais importantes para a sua percepção do ponto de vista do leitor. Para erradicar quaisquer dúvidas, o código completo encontra-se no [Apêndice A](#) e devidamente comentado.

Primeiramente reconheçam-se as variáveis globais utilizadas, com especial atenção para as variáveis das linhas 15, 16, 21 e 22 que correspondem à informação de um sinal em pixéis dos dois eixos (para a representação no *display*) e aos índices dos vetores das escalas para efetuar a mudança de escala consoante o botão premido (no ciclo principal do programa incrementam-se estes índices ao receber *input* dos botões respetivos). Notemos ainda a variável da linha 31 que, quando a `True`, possibilita a mudança de escalas e permanecendo na representação do espetro, mudando o seu valor para `False` aquando da representação no domínio do tempo.

```

8 max_width = 240    # Largura do display
9 max_height = 135   # Altura do display
10 grid_height = max_height - 16  # Altura para desenhar a grelha
11 x_top_right_wifi = max_width - 16  # Coordenada x do icon Wifi
12 y_top_right_wifi = max_height - 16  # Coordenada y do icon Wifi
13 x_divisions = 10  # Número de divisões horizontais da grelha
14 y_divisions = 6   # Número de divisões verticais da grelha
15 x = []  # Vetor da escala horizontal
16 y = []  # Vetor da escala vertical
17 scales_time = [5, 10, 20, 50]  # Escalas horizontais no tempo
18 scales_volt = [1, 2, 5, 10]  # Escalas verticais de tensão
19 scales_spectrum_volt = [0.5, 1, 2.5, 5]  # Escalas verticais de tensão do espetro
20 scales_spectrum_freq = [240, 120, 60, 24]  # Escalas horizontais na frequência
21 index_scales_vertical = 3  # Índice do vetor de escalas horizontais., 20ms/div e 60Hz/div default
22 index_scales_horizontal = 2  # Índice do vetor de escalas verticais., 10V/div default
23 voltages = []  # Lista de tensões
24 factor = 1/29.3  # Fator do divisor resistivo
25 Vmax = 0  # Valor máximo de tensão
26 Vmin = 0  # Valor mínimo de tensão
27 Vrms = 0  # Valor eficaz de tensão
28 Vavg = 0  # Valor médio de tensão
29 self = 0  # Argumento da função .send_email
30 body = ""  # Corpo do email a enviar
31 spectrum = False  # Indica se estamos na representação do espetro ou na normal
32 tft = T_Display.TFT()  # Instancia um objeto da classe TFT

```

O programa principal começa por executar `reset_display()` e `read_and_calculate()`. Esta última apresentada abaixo, calcula as tensões a partir dos pontos recolhidos do ADC, determina parte do conjunto de medidas a enviar por email (valor eficaz é calculado à parte com `calculate_rms_value()`) e converte as tensões em pixéis para representar o sinal no *display* através de `draw_normal_plot()`.

```

59 def read_and_calculate():
60     x_aux = []  # Coordenadas x
61     y_aux = []  # Coordenadas y
62     adc_points = []  # Lista de pontos de ADC
63     voltages_aux = []  # Lista de tensões

```

```

64     Vmax_aux = 0    # Valor maximo de tensao
65     Vmin_aux = 0    # Valor minimo de tensao
66     Vavg_aux = 0    # Valor medio de tensao
67     V = 0           # Conversao do valor do ADC para Volt
68
69     adc_points = tft.read_adc(max_width, scales_time[index_scales_horizontal] * x_divisions)  # Le 240 pontos do ADC em [50, 100,
70         200, 500]ms
70     voltages_aux = adc_points
71
72     for n in range(max_width):
73         # V = 0.00044028 * adc_points[n] + 0.091455  # Converte valor do ADC em Volt - Ajuste do Professor
74         # V = V - 1   # Tensao entrada de referencia de 1V
75         # V = V / factor  # Entra com o efeito do div. resistivo
76         V = 0.012020968916394 * adc_points[n] - 24.238021692121710  # Converte valor do ADC em Volt - Ajuste experimental
77         voltages_aux[n] = V
78
79         pixel = grid_height/2 + (grid_height / (scales_volt[index_scales_vertical] * y_divisions))*V  # pix + [pix / (volt/div * div
80     )]*volt = pix
80         if pixel > grid_height: # Quando o valor excede o espaco vertical superior simplesmente fica no limite superior do display
81             pixel = grid_height
82         x_aux.append(n)
83         y_aux.append(round(pixel))
84
85         if n == 0: # Caso seja o primeiro ponto
86             Vmax_aux = Vmin_aux = Vavg_aux = V
87         else:
88             Vavg_aux += V  # Ir somando para depois fazer o valor medio
89             if V > Vmax_aux: # Atualizar valor maximo de tensao
90                 Vmax_aux = V
91             if V < Vmin_aux: # Atualizar valor minimo de tensao
92                 Vmin_aux = V
93
94     Vavg_aux /= max_width  # Calcular o valor medio dividindo pelo numero de amostras
95     del adc_points, V, pixel  # Apagar variaveis ja nao necessarias
96
97     return voltages_aux, Vmax_aux, Vmin_aux, Vavg_aux, x_aux, y_aux

```

De seguida inicia-se o ciclo principal do programa onde se espera pelo acionamento de alguma das funções pedidas para desenvolver, através da pressão de um botão por parte do utilizador. Por motivos de extensividade, esse ciclo não é aqui apresentado, mas pode ser consultado em detalhe no [Apêndice A](#).

Ao pressionar o **Botão 1**: Clique rápido - estamos no domínio do tempo e as escalas voltam ao padrão (modificando as variáveis `spectrum` e `index` para tal), de seguida executa-se `read_and_calulate()` e `draw_normal_plot()`, obtendo a representação do sinal no *display*; Clique lento - aqui calcula-se a medida em falta `Vrms` e envia-se um email com esta e as calculadas na medição anterior; Duplo clique - calcula-se o conjunto de medidas e são apresentadas no *display* através de `reset_button_13_display()`.

Por outro lado, pressionando o **Botão 2**: Clique rápido e clique lento - mudança das escalas vertical e horizontal (respetivamente) através das variáveis `index` que são incrementadas de modo às escalas dos vetores `scale` aumentarem sequencialmente e, caso seja a última, voltar à primeira. A variável `spectrum` é aqui usada para mudar a escala e continuar no domínio da frequência (se for o caso), depois são feitas novas medições e representa-se o sinal no domínio respetivo; Duplo clique - aqui é calculada a *Discrete Fourier Transform* (DFT) e o espetro do sinal, com `dft_and_spectrum()` (esta apresentada abaixo), e representado o espetro do sinal com `draw_spectrum_plot()`. No cálculo da DFT começa-se por usar a fórmula apresentada no enunciado, mas uma vez que metade dos valores são redundantes devido a  $X_k = X_{N-k}^*$  (pois o espetro obtém-se a partir dos módulos da transformada de Fourier), apenas se calcula `X_k` num ciclo de  $N//2-1$  iterações (ignorando o último ponto  $k = N/2$  como pedido no enunciado). Este cálculo é ainda feito dois a dois pontos devido à recomendação do enunciado. De seguida, aplica-se a fórmula de  $X_{SS_k}$ , que consta no enunciado, para os valores de `k` respetivos e estes valores são convertidos para pixels que irão ser os elementos do vetor das ordenadas `y_aux` a devolver (juntamente com `x_aux` que consiste no vetor das abcissas, ou seja, frequências).

```

142 def dft_and_spectrum(x_n):
143     N = len(x_n)  # Numero de amostras
144     x_aux = [n for n in range(0,max_width,1)]  # Vetor de frequencias
145     y_aux = []  # Espectro de frequencia a devolver
146     X_k = [0.0]*N  # DFT
147     X_ss_k = [0.0]*N  # Espectro de frequencia
148
149     for k in range(N//2):  # Ignorar o ponto N/2
150         real = 0  # Parte real de X_k
151         imag = 0  # Parte imaginaria de X_k
152         for n in range(N - 1):  # Somatorio da equacao
153             real += x_n[n] * math.cos(2 * math.pi * k * n / N)
154             imag += x_n[n] * -math.sin(2 * math.pi * k * n / N)
155         X_k[2*k] = math.sqrt(real**2 + imag**2)  # Modulo de X_k dois a dois pontos, porque queremos X_ss_k assim
156         X_k[2*k + 1] = X_k[2*k]
157

```

```

158     if k == 0: # Expressao de X_ss_k
159         X_ss_k[2*k] = X_k[2*k] / N
160         X_ss_k[2*k + 1] = X_ss_k[2*k]
161         pixel = X_ss_k[2*k] * (grid_height / (scales_spectrum_volt[index_scales_vertical] * y_divisions)) # pix + [pix / (volt/
162         div * div)]*volt = pix
163         if pixel > grid_height: # Quando o valor excede o espaco vertical superior simplesmente fica no limite superior do
164             display
165                 pixel = grid_height
166                 y_aux.append(round(pixel))
167         pixel = X_ss_k[2*k + 1] * (grid_height / (scales_spectrum_volt[index_scales_vertical] * y_divisions)) # pix + [pix / (
168         volt/div * div)]*volt = pix
169         if pixel > grid_height: # Quando o valor excede o espaco vertical superior simplesmente fica no limite superior do
170             display
171                 pixel = grid_height
172                 y_aux.append(round(pixel))
173         else: # Expressao de X_ss_k
174             X_ss_k[2*k] = 2*X_k[2*k] / N
175             X_ss_k[2*k + 1] = X_ss_k[2*k]
176             pixel = X_ss_k[2*k] * (grid_height / (scales_spectrum_volt[index_scales_vertical] * y_divisions)) # pix + [pix / (volt/
177             div * div)]*volt = pix
178             if pixel > grid_height: # Quando o valor excede o espaco vertical superior simplesmente fica no limite superior do
179             display
180                 pixel = grid_height
181                 y_aux.append(round(pixel))
182
183     del N, X_k, X_ss_k, real, imag, pixel # Apagar variaveis ja nao necessarias
184
185     return x_aux, y_aux

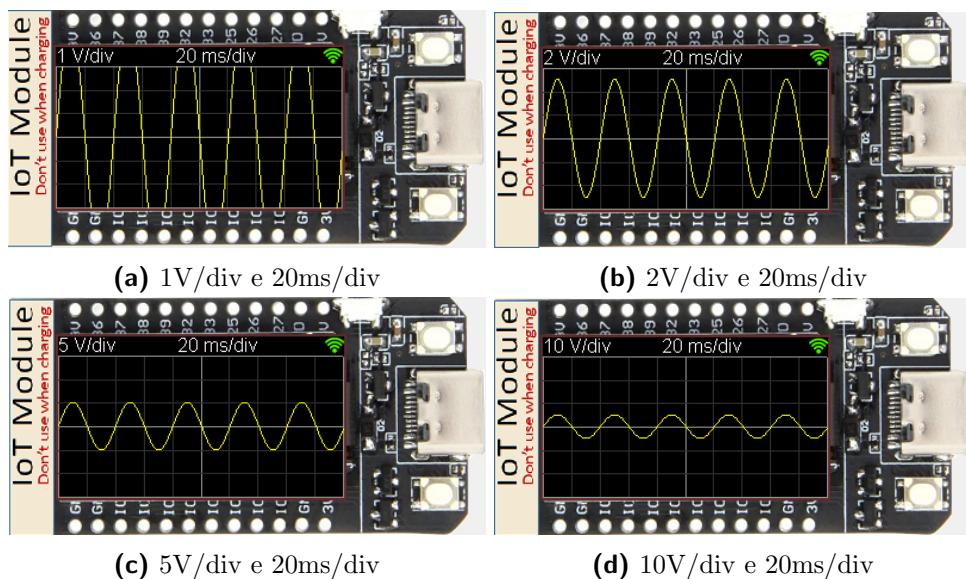
```

Outras funções ainda não mencionadas, mas claramente utilizadas ao longo da implementação do ciclo principal do programa, são `reset_display()` e `reset_spectrum_display()` que simplesmente efetuam o `reset` do `display` para o domínio do tempo e frequência, respetivamente, no que toca à grelha e ícone de Wi-fi.

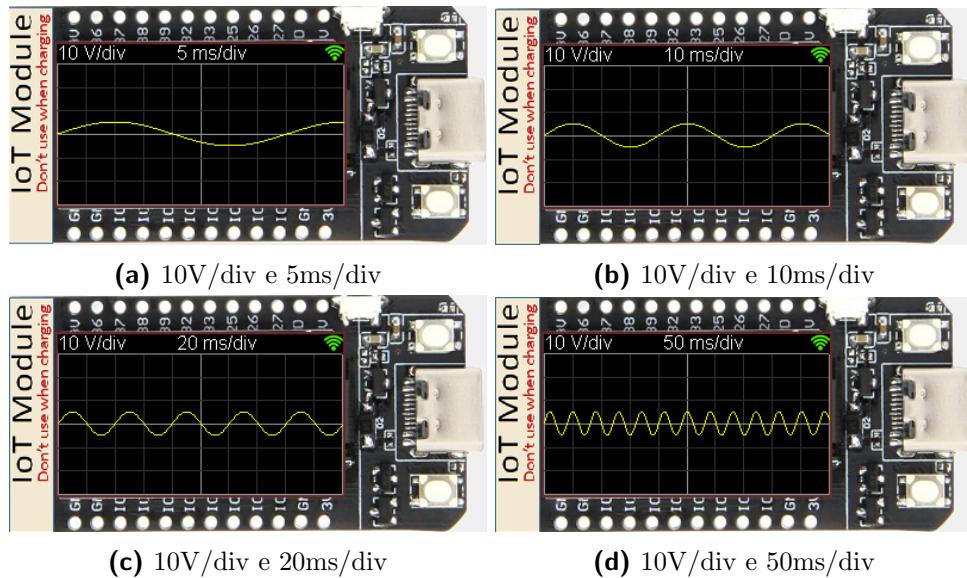
Uma outra funcionalidade que podia ter sido desenvolvida seria o clássico *Autoscale*, que haveria de selecionar as melhores escalas vertical e horizontal para proceder à representação de um sinal da melhor maneira possível, assim como também seria possível definir um número de períodos mínimo a representar.

### 3 Testes no simulador

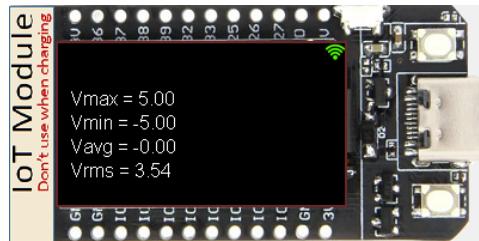
De maneira a verificar o funcionamento do programa, primeiramente testaram-se as funcionalidades deste com o simulador. Para tal apresenta-se de seguida os resultados obtidos para um sinal sinusoidal de 25Hz, 5V de amplitude e 0V de offset, e ainda para a variação de uma fonte DC (escala de 5V/div como pedido).



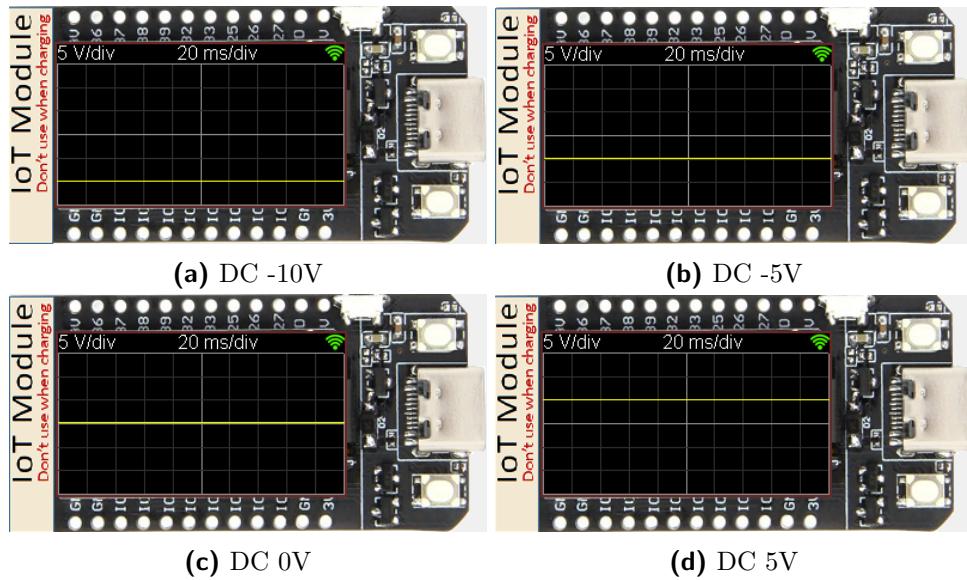
**Figura 1:** Variação da escala vertical (simulador)

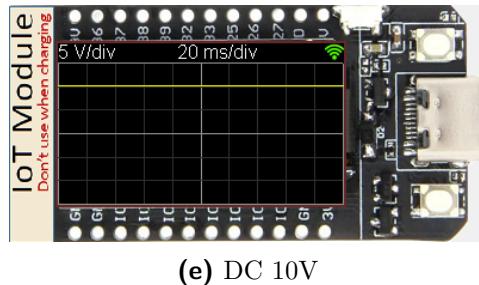


**Figura 2:** Variação da escala horizontal (simulador)



**Figura 3:** Funcionamento do duplo clique do botão 1 (simulador)





**Figura 4:** Variação de uma tensão DC (simulador)

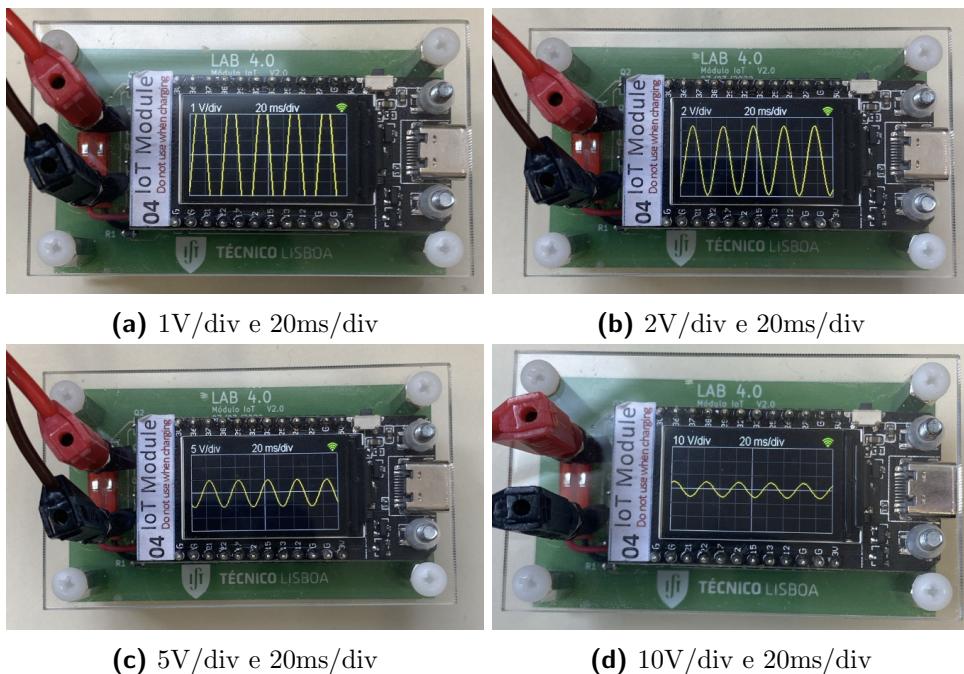
Aqui não se apresenta a onda quadrada com que o programa tem de ser testado uma vez que o obtido é exatamente igual ao que está no enunciado. A funcionalidade da DFT será demonstrada nos [Testes no laboratório](#).

Como seria de esperar, no simulador com a calibração de origem do Professor verifica-se a correta representação de todos os sinais e funcionalidade do programa desenvolvido. Nos testes experimentais é inevitável que haja erros a nível de ruído e no conjunto de medidas determinado, o que será discutido de seguida.

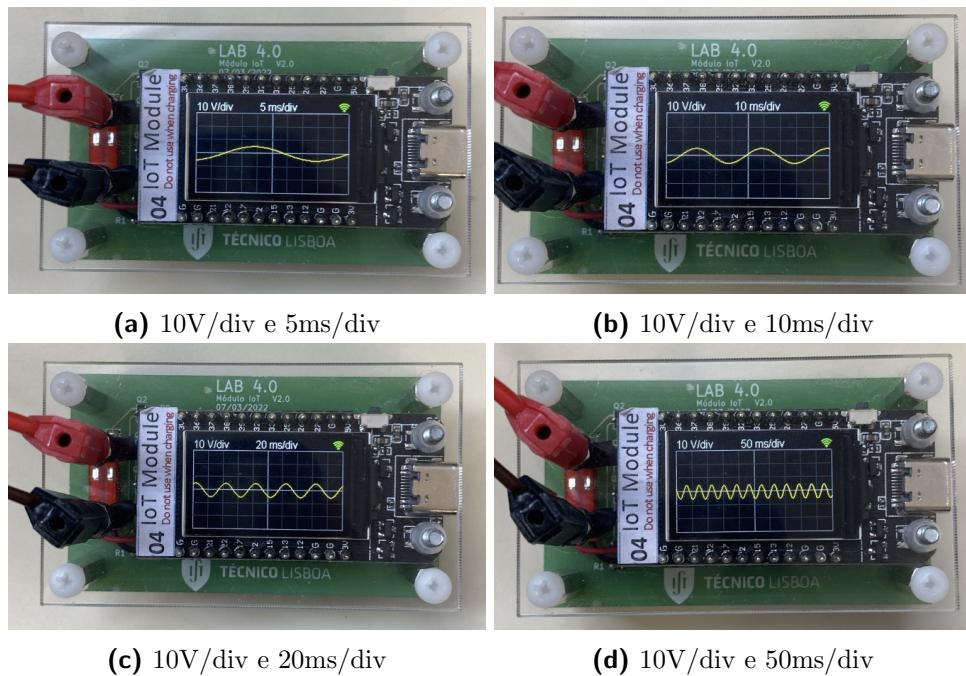
## 4 Testes no laboratório

### 4.1 Não calibrado

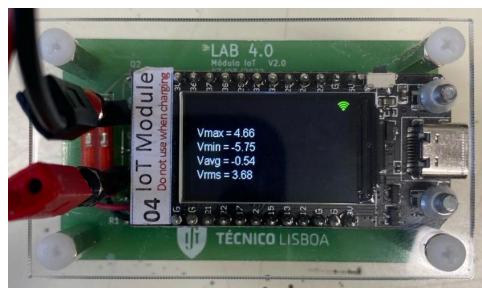
Novamente, mas agora experimentalmente, apresenta-se de seguida os resultados obtidos para um sinal sinusoidal de 25Hz, 5V de amplitude e 0V de *offset*, para a variação de uma fonte DC (escala de 5V/div como pedido) e ainda uma onda quadrada de 60Hz, 4V de amplitude e -1V de *offset*. Para obter estes dados sem calibração comenta-se a linha 76 e descomentam-se as linhas 73, 74 e 75 da função `read_and_calculate()`.



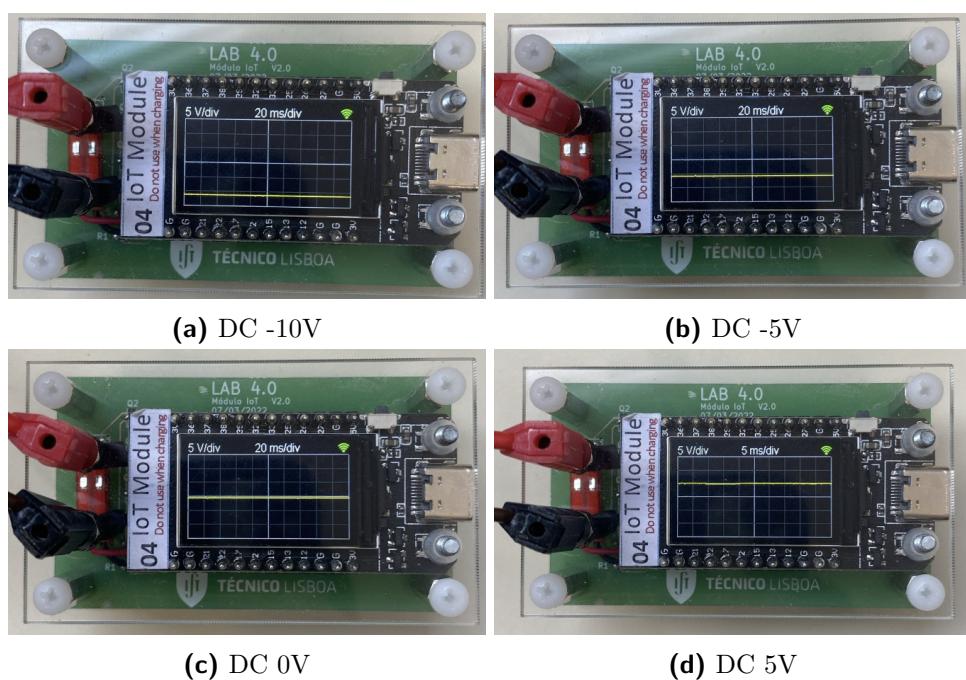
**Figura 5:** Variação da escala vertical (não calibrado)

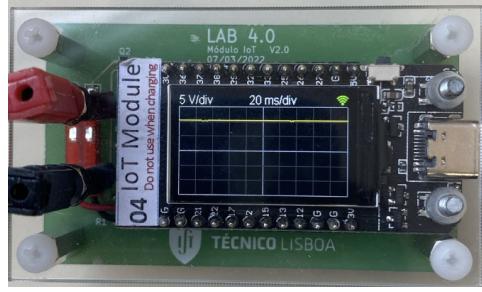


**Figura 6:** Variação da escala horizontal (não calibrado)



**Figura 7:** Funcionamento do duplo clique do botão 1 (não calibrado)





(e) DC 10V

**Figura 8:** Variação de uma tensão DC (não calibrado)



(a) Onda quadrada

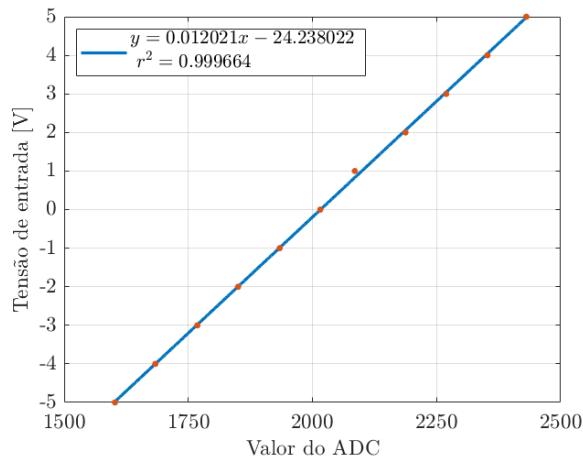
(b) Espetro da onda quadrada

**Figura 9:** Onda quadrada do enunciado e o seu espetro (não calibrado)

Como era de esperar, verifica-se a necessidade de calibrar o sistema por mera observação das imagens acima no que toca à incorreta representação dos sinais e, nomeadamente, o erro associado ao conjunto de medidas evidenciado na [Figura 7](#) (facilmente observável pelas tensões máxima e mínima apresentadas).

## 4.2 Ajuste experimental

Verificada a necessidade de efetuar a calibração, ou seja, fazer corresponder o valor digital obtido através do conversor analógico-digital ao valor de entrada do mesmo, utilizou-se o gerador de funções disponível no laboratório. Com este simulou-se uma fonte DC ao gerar um sinal com uma frequência e amplitude pico a pico praticamente nulas ( $f \approx 0$  e  $V_{pp} \approx 0$ ). Simulada esta fonte, alterou-se o valor do *offset* na gama de valores  $\{-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5\}$  como é visível pelos pontos e escala das ordenadas da [Figura 10](#).

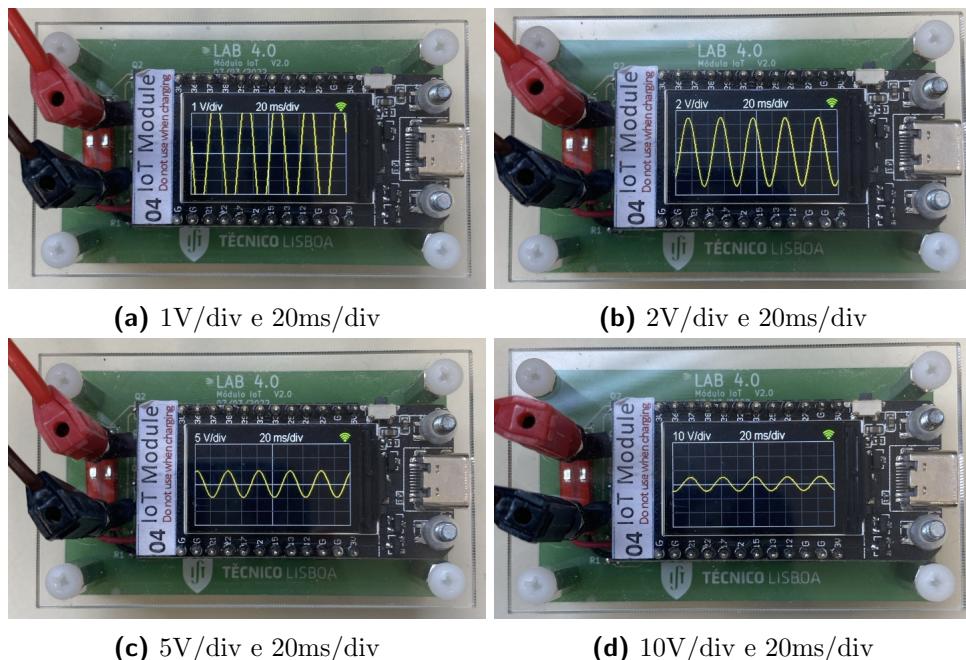


**Figura 10:** Reta de ajuste experimental

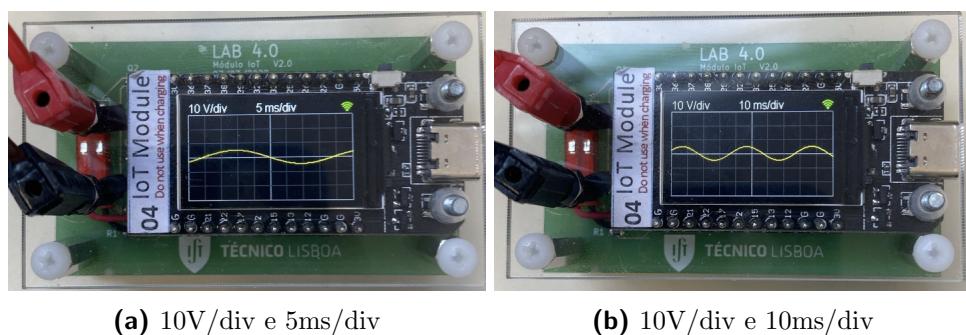
À medida que se alteravam os valores usufruiu-se do script `main_exemplo_2.py` para calcular a média de 100 amostras cujos valores representam o eixo das abcissas da Figura 10. Note-se que o gráfico acima é o inverso do apresentado no enunciado, pelo que no código da função `read_and_calculate()`, e já tendo a equação de ajuste, aparecem as linhas 73, 74 e 75 comentadas e a 76 descomentada. Para confirmação do bom ajuste usou-se o script `main_exemplo_1.py` com o qual se verificou que para uma tensão DC de 5V o valor médio era 5.06V (Erro = 1.2% < 5% - indicação de bom ajuste).

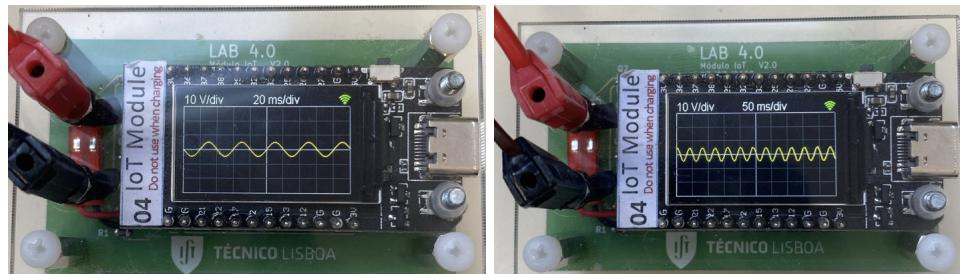
### 4.3 Calibrado

Como na secção anterior, apresenta-se de seguida os resultados obtidos para um sinal sinusoidal de 25Hz, 5V de amplitude e 0V de offset, para a variação de uma fonte DC (escala de 5V/div como pedido) e ainda uma onda quadrada de 60Hz, 4V de amplitude e -1V de offset. Agora com o ajuste experimental, comentam-se as linhas 73, 74 e 75 e descommenta-se a linha 76 da função `read_and_calculate()`.



**Figura 11:** Variação da escala vertical (calibrado)

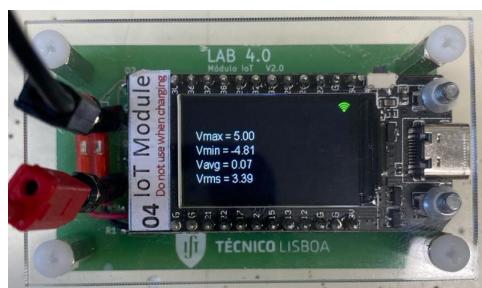




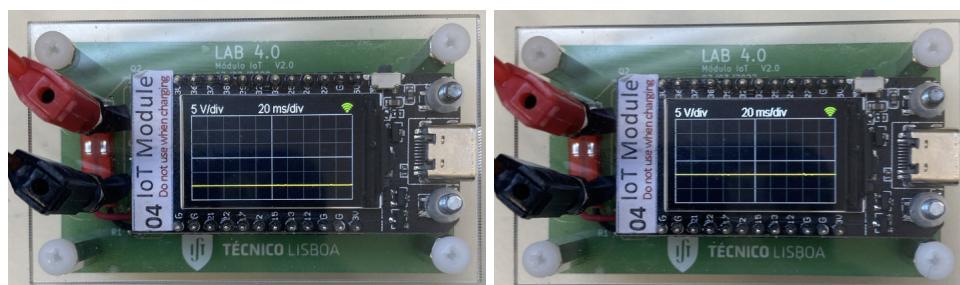
(c) 10V/div e 20ms/div

(d) 10V/div e 50ms/div

**Figura 12:** Variação da escala horizontal (calibrado)

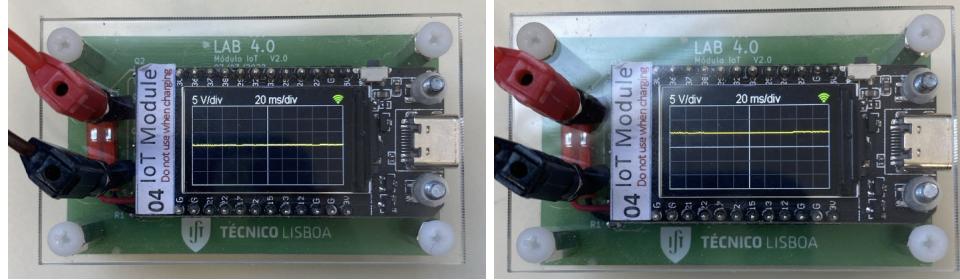


**Figura 13:** Funcionamento do duplo clique do botão 1 (calibrado)



(a) DC -10V

(b) DC -5V



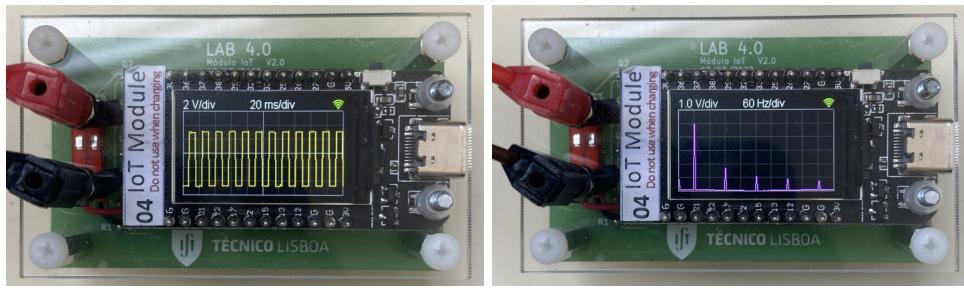
(c) DC 0V

(d) DC 5V



(e) DC 10V

**Figura 14:** Variação de uma tensão DC (calibrado)



(a) Onda quadrada

(b) Espetro da onda quadrada

**Figura 15:** Onda quadrada do enunciado e o seu espetro (calibrado)

Como previsto, a calibração permitiu obter representações dos sinais realmente corretas como se pode ver nas ilustrações acima apresentadas. No entanto, como dito na [Introdução](#), as imperfeições não são completamente aniquiladas devido às não idealidades do sistema que causam ruído (particularmente visível na [Figura 14](#)), nomeadamente a impedância associada ao gerador de funções usado para gerar o sinal de entrada e fazer curto-círcuito no módulo IoT. Pela [Figura 13](#) vemos ainda o efeito das não idealidades no cálculo de  $V_{min}$ , uma vez que este não é o simétrico de  $V_{max}$ . Pela [Figura 15](#) vemos também o efeito do ruído na representação no domínio do tempo e, por sua vez, um ligeiro decréscimo do pico principal do espetro.

Desta maneira, após testes em simulação e no laboratório verifica-se o correto funcionamento do ficheiro `main.py` desenvolvido, assim como um bom ajuste experimental para o módulo IoT 04 utilizado.

## 5 Conclusões

Primeiramente, há que referir a fluída execução do presente trabalho de laboratório. Em comparação com trabalhos passados, este tem menos variáveis que fogem do controlo do grupo (como é o caso dos inúmeros componentes de circuito utilizados em outros trabalhos), uma vez que o essencial deste trabalho é o desenvolvimento de *software*.

Para cumprir os objetivos estabelecidos foi desenvolvido o programa `main.py` cujas funcionalidades pedidas foram completa e corretamente implementadas. Estas contemplam a leitura de dados do ADC e conversão para tensões, envio de um email com um conjunto de medidas e apresentação deste no *display*, alteração das escalas vertical e horizontal e ainda a representação do espetro de um sinal. Estas complementando-se de forma a implementar o  $\mu$ Oscilloscope com recurso a um módulo IoT.

Conforme já mencionado, uma funcionalidade que poderia ter sido implementada seria o *Autoscale*, devido à sua enorme utilidade. Uma outra seria ainda o canal *Source*, que possibilita a visualização simultânea de duas ondas. Para implementar esta última funcionalidade, seria necessário realizar duas medições consecutivas, sendo que a segunda medição seria armazenada em variáveis distintas (não efetuando o *reset* do *display* no momento da representação do segundo sinal).

Desta vez, a maior parte do trabalho foi realizado em casa dos alunos testando o programa por eles desenvolvido a partir do simulador. Este verificando-se como uma ótima ferramenta de teste para o âmbito laboratorial, uma vez que, devido à sua utilidade, não houve percalços na recolha de dados experimentais.

Em ambiente laboratorial verificou-se a necessidade de calibrar os parâmetros de ajuste, confirmando as expectativas estabelecidas na [Introdução](#). Para tal efetuou-se um [Ajuste experimental](#) com o qual se obtiveram dados congruentes, mas com ligeiros desvios e presença de ruído devido a não idealidades já referidas ao longo do relatório.

Com isto, através deste estudo superficial de sistemas embebidos é possível afirmar que se concluíram os objetivos inicialmente estabelecidos, nomeadamente o desenvolvimento de *software* para a implementação

do  $\mu$ Oscilloscope, o teste em simulação e posterior verificação do funcionamento do programa em ambiente experimental e, intermediariamente, a calibração necessária do  $\mu$ Oscilloscope.

# Apêndices

## A Ficheiro de código desenvolvido completo

```
1 import T_Display
2 import math
3 import gc
4
5 ##### VARIAVEIS #####
6 # max_width = 240 # Largura do display
7 # max_height = 135 # Altura do display
8 max_width = 240 # Largura do display para desenhar a grelha
9 max_height = 135 # Altura do display
10 grid_height = max_height - 16 # Altura para desenhar a grelha
11 x_top_right_wifi = max_width - 16 # Coordenada x do icon Wifi
12 y_top_right_wifi = max_height - 16 # Coordenada y do icon Wifi
13 x_divisions = 10 # Numero de divisões horizontais da grelha
14 y_divisions = 6 # Numero de divisões verticais da grelha
15 x = [] # Vetor da escala horizontal
16 y = [] # Vetor da escala vertical
17 scales_time = [5, 10, 20, 50] # Escalas horizontais no tempo
18 scales_volt = [1, 2, 5, 10] # Escalas verticais de tensão
19 scales_spectrum_volt = [0.5, 1, 2.5, 5] # Escalas verticais de tensão do espetro
20 scales_spectrum_freq = [240, 120, 60, 24] # Escalas horizontais na frequência
21 index_scales_vertical = 3 # Índice do vetor de escalas horizontais,, 20ms/div e 60Hz/div default
22 index_scales_horizontal = 2 # Índice do vetor de escalas verticais,, 10V/div default
23 voltages = [] # Lista de tensões
24 factor = 1/29.3 # Fator do divisor resistivo
25 Vmax = 0 # Valor máximo de tensão
26 Vmin = 0 # Valor mínimo de tensão
27 Vrms = 0 # Valor eficaz de tensão
28 Vavg = 0 # Valor médio de tensão
29 self = 0 # Argumento da função .send_email
30 body = "" # Corpo do email a enviar
31 spectrum = False # Indica se estamos na representação do espetro ou na normal
32 tft = T_Display.TFT() # Instancia um objeto da classe TFT
33
34 #####
35 # FUNCÕES #
36 #####
37 ''
38 # Reset do display normal #
39 ''
40 def reset_display():
41     tft.display_set(tft.BLACK, 0, 0, max_width, max_height) # Apaga display
42     tft.display_write_grid(0, 0, max_width, grid_height, x_divisions, y_divisions, True, tft.GREY1, tft.GREY2) # Desenha grelha (com linhas centrais)
43     tft.set_wifi_icon(x_top_right_wifi, y_top_right_wifi) # Desenhar icon wifi
44
45 ''
46 # Desenhar o gráfico normal #
47 ''
48 def draw_normal_plot():
49     tft.display_oline(tft.YELLOW, x, y) # Desenhar o sinal
50     for i in x: # Arranjar o gráfico (apagar os pixels que estão para lá das margens do gráfico)
51         if y[i] >= grid_height:
52             tft.display_pixel(tft.BLACK,x[i],y[i])
53     tft.display_write_str(tft.Arial16, "%d V/div" % scales_volt[index_scales_vertical], 0, grid_height) # Apresentar a escala vertical
54     tft.display_write_str(tft.Arial16, "%d ms/div" % scales_time[index_scales_horizontal], 100, grid_height) # Apresentar a escala horizontal
55
56 ''
57 # Ler valores ADC e calcular valores necessários #
58 ''
59 def read_and_calculate():
60     x_aux = [] # Coordenadas x
61     y_aux = [] # Coordenadas y
62     adc_points = [] # Lista de pontos de ADC
63     voltages_aux = [] # Lista de tensões
64     Vmax_aux = 0 # Valor máximo de tensão
65     Vmin_aux = 0 # Valor mínimo de tensão
66     Vavg_aux = 0 # Valor médio de tensão
67     V = 0 # Conversão do valor do ADC para Volt
68
69     adc_points = tft.read_adc(max_width, scales_time[index_scales_horizontal] * x_divisions) # Le 240 pontos do ADC em [50, 100, 200, 500]ms
70     voltages_aux = adc_points
71
72     for n in range(max_width):
73         # V = 0.00044028 * adc_points[n] + 0.091455 # Converte valor do ADC em Volt - Ajuste do Professor
74         # V = V - 1 # Tensão entrada de referência de 1V
75         # V = V / factor # Entra com o efeito do divisor resistivo
76         V = 0.012020968916394 * adc_points[n] - 24.238021692121710 # Converte valor do ADC em Volt - Ajuste experimental
77         voltages_aux[n] = V
78
79         pixel = grid_height/2 + (grid_height / (scales_volt[index_scales_vertical] * y_divisions))*V # pix + [pix / (volt/div * div)]*volt = pix
80         if pixel > grid_height: # Quando o valor excede o espaço vertical superior simplesmente fica no limite superior do display
81             pixel = grid_height
82             x_aux.append(n)
83             y_aux.append(round(pixel))
84
```

```

85     if n == 0: # Caso seja o primeiro ponto
86         Vmax_aux = Vmin_aux = Vavg_aux = V
87     else:
88         Vavg_aux += V # Ir somando para depois fazer o valor medio
89         if V > Vmax_aux: # Atualizar valor maximo de tensao
90             Vmax_aux = V
91         if V < Vmin_aux: # Atualizar valor minimo de tensao
92             Vmin_aux = V
93
94     Vavg_aux /= max_width # Calcular o valor medio dividinho pelo numero de amostras
95     del adc_points, V, pixel # Apagar variaveis ja nao necessarias
96
97     return voltages_aux, Vmax_aux, Vmin_aux, Vavg_aux, x_aux, y_aux
98
99 """
100 # Reset do display para o botao 13 #
101 """
102 def reset_button_13_display():
103     tft.display_set(tft.BLACK, 0, 0, max_width, max_height) # Apaga display
104     tft.display_write_str(tft.Arial16, "Vmax = %.2f" % Vmax, 10, 60+20) # Apresentar o valor Vmax
105     tft.display_write_str(tft.Arial16, "Vmin = %.2f" % Vmin, 10, 40+20) # Apresentar o valor Vmin
106     tft.display_write_str(tft.Arial16, "Vavg = %.2f" % Vavg, 10, 20+20) # Apresentar o valor Vavg
107     tft.display_write_str(tft.Arial16, "Vrms = %.2f" % Vrms, 10, 20) # Apresentar o valor Vrms
108     tft.set_wifi.icon(x_top_right_wifi, y_top_right_wifi) # Desenhar icon wifi
109
110 """
111 # Calcular o valor eficaz #
112 """
113 def calculate_rms_value():
114     sum_squares = sum(value ** 2 for value in voltages) # Soma dos quadrados
115     Vrms_aux = (sum_squares / len(voltages)) ** 0.5 # Valor eficaz
116     del sum_squares # Apagar variaveis ja nao necessarias
117
118     return Vrms_aux
119
120 """
121 # Reset do display para o espetro #
122 """
123 def reset_spectrum_display():
124     tft.display_set(tft.BLACK, 0, 0, max_width, max_height) # Apaga display
125     tft.display_write_grid(0, 0, max_width, grid_height, x_divisions, y_divisions, False, tft.GREY1, tft.GREY2) # Desenha grelha (s
     / linhas centrais)
126     tft.set_wifi.icon(x_top_right_wifi, y_top_right_wifi) # Desenhar icon wifi
127
128 """
129 # Desenhar o grafico do espetro #
130 """
131 def draw_spectrum_plot():
132     tft.display_line(tft.MAGENTA, x, y) # Desenhar o sinal
133     for i in x: # Aranjar o grafico (apagar os pixels que estao para la das margens do grafico)
134         if y[i] >= grid_height:
135             tft.display_pixel(tft.BLACK,x[i],y[i])
136     tft.display_write_str(tft.Arial16, "%i V/div" % scales_spectrum_volt[index_scales_vertical], 0, grid_height) # Apresentar a
     escala vertical
137     tft.display_write_str(tft.Arial16, "%d Hz/div" % scales_spectrum_freq[index_scales_horizontal], 100, grid_height) # Apresentar a
     escala horizontal
138
139 """
140 # Calcular a dft e espetro #
141 """
142 def dft_and_spectrum(x_n):
143     N = len(x_n) # Numero de amostras
144     x_aux = [n for n in range(0,max_width,1)] # Vetor de frequencias
145     y_aux = [] # Espectro de frequencia a devolver
146     X_k = [0.0]*N # DFT
147     X_ss_k = [0.0]*N # Espectro de frequencia
148
149     for k in range(N//2): # Ignorar o ponto N/2
150         real = 0 # Parte real de X_k
151         imag = 0 # Parte imaginaria de X_k
152         for n in range(N - 1): # Somatorio da equacao
153             real += x_n[n] * math.cos(2 * math.pi * k * n / N)
154             imag += x_n[n] * -math.sin(2 * math.pi * k * n / N)
155         X_k[2*k] = math.sqrt(real**2 + imag**2) # Modulo de X_k dois a dois pontos, porque queremos X_ss_k assim
156         X_k[2*k + 1] = X_k[2*k]
157
158         if k == 0: # Expressao de X_ss_k
159             X_ss_k[2*k] = X_k[2*k] / N
160             X_ss_k[2*k + 1] = X_ss_k[2*k]
161             pixel = X_ss_k[2*k] * (grid_height / (scales_spectrum_volt[index_scales_vertical] * y_divisions)) # pix + [pix / (volt/
     div * div)]*volt = pix
162             if pixel > grid_height: # Quando o valor excede o espaco vertical superior simplesmente fica no limite superior do
     display
163                 pixel = grid_height
164                 y_aux.append(round(pixel))
165                 pixel = X_ss_k[2*k + 1] * (grid_height / (scales_spectrum_volt[index_scales_vertical] * y_divisions)) # pix + [pix / (volt/
     div * div)]*volt = pix
166                 if pixel > grid_height: # Quando o valor excede o espaco vertical superior simplesmente fica no limite superior do
     display
167                     pixel = grid_height
168                     y_aux.append(round(pixel))
169             else: # Expressao de X_ss_k
170                 X_ss_k[2*k] = 2*X_k[2*k] / N
171                 X_ss_k[2*k + 1] = X_ss_k[2*k]
172                 pixel = X_ss_k[2*k] * (grid_height / (scales_spectrum_volt[index_scales_vertical] * y_divisions)) # pix + [pix / (volt/
     div * div)]*volt = pix
173                 if pixel > grid_height: # Quando o valor excede o espaco vertical superior simplesmente fica no limite superior do
     display

```

```

174     pixel = grid_height
175     y_aux.append(round(pixel))
176     pixel = X_ss_k[2*k + 1] * (grid_height / (scales_spectrum_volt[index_scales_vertical] * y_divisions))    # pix + [pix / (
177     volt/div * div)]*volt = pix
178     if pixel > grid_height: # Quando o valor excede o espaco vertical superior simplesmente fica no limite superior do
179         display
180         pixel = grid_height
181         y_aux.append(round(pixel))
182
183     del N, X_k, X_ss_k, real, imag, pixel      # Apagar variaveis ja nao necessarias
184
185 #####PROGRAMA PRINCIPAL (main)#####
186 #
187 #####reset do display normal#####
188 reset_display() # Reset do display normal
189 voltages, Vmax, Vmin, Vavg, x, y = read_and_calculate() # Ler valores ADC e calcular valores necessarios
190 draw_normal_plot() # Desenhar o grafico normal
191
192 while tft.working(): # Ciclo principal do programa
193     button = tft.readButton() # Le estado dos botoes
194     if button != tft.NOTHING: # Quando algum botao for premido
195         print("Button pressed:", button)
196         if button == 11: # Botao 1 click rapido (<1segundo)
197             gc.collect()
198             spectrum = False # Ja nao estamos no espetro
199             reset_display() # Reset do display normal
200             index_scales_horizontal = 2 # Indice do vetor de escalas horizontais,, 20ms/div default
201             index_scales_vertical = 3 # Indice do vetor de escalas verticais,, 10V/div default
202             voltages, Vmax, Vmin, Vavg, x, y = read_and_calculate() # Ler valores ADC e calcular valores necessarios
203             draw_normal_plot() # Desenhar o grafico normal
204             gc.collect()
205
206         if button == 12: # Botao 1 click lento (>1segundo)
207             gc.collect()
208             Vrms = calculate_rms_value() # Calcular o valor eficaz
209             self = scales_time[index_scales_horizontal] * x_divisions # Argumento da funcao .send_mail
210             body = f"Tensoa maxima (Vmax): {Vmax:.2f} V\nTensoa minima (Vmin): {Vmin:.2f} V\nValor medio (Vav): {Vavg:.2f} V\nValor
211             eficaz (Vrms): {Vrms:.2f} V" # Corpo do email a enviar
212             tft.send_email(self/max_width, voltages, body, "antonio.v.moraes.c@tecnico.ulisboa.pt") # Enviar email da tabela com
213             tempos e tensoes
214             gc.collect()
215
216         if button == 13: # Botao 1 duplo click
217             gc.collect()
218             voltages, Vmax, Vmin, Vavg, x, y = read_and_calculate() # Ler valores ADC e calcular valores necessarios
219             Vrms = calculate_rms_value() # Calcular o valor eficaz
220             reset_button_13_display() # Reset do display para o botao 1
221             gc.collect()
222
223         if button == 21: # Botao 2 click rapido (<1segundo)
224             gc.collect()
225             index_scales_vertical += 1 # Incrementar o indice do vetor de escalas verticais
226             if index_scales_vertical >= len(scales_volt): # Indice do vetor de escalas verticais, caso seja o ultimo voltar ao
227                 inicio
228                 index_scales_vertical = 0
229                 if spectrum == True: # Se estivermos no espetro
230                     reset_spectrum_display() # Reset do display do espetro
231                     voltages, Vmax, Vmin, Vavg, x, y = read_and_calculate() # Ler valores ADC e calcular valores necessarios
232                     x, y = dft_and_spectrum(voltages) # Calcular o espetro
233                     draw_spectrum_plot() # Desenhar o grafico do espetro
234                     gc.collect()
235
236                 else:
237                     reset_display() # Reset do display normal
238                     voltages, Vmax, Vmin, Vavg, x, y = read_and_calculate() # Ler valores ADC e calcular valores necessarios
239                     draw_normal_plot() # Desenhar o grafico normal
240                     gc.collect()
241
242         if button == 22: # Botao 2 click lento (>1segundo)
243             gc.collect()
244             index_scales_horizontal += 1 # Incrementar o indice do vetor de escalas horizontais
245             if index_scales_horizontal >= len(scales_time): # Indice do vetor de escalas horizontais, caso seja o ultimo voltar ao
246                 inicio
247                 index_scales_horizontal = 0
248                 if spectrum == True: # Se estivermos no espetro
249                     reset_spectrum_display() # Reset do display do espetro
250                     voltages, Vmax, Vmin, Vavg, x, y = read_and_calculate() # Ler valores ADC e calcular valores necessarios
251                     x, y = dft_and_spectrum(voltages) # Calcular o espetro
252                     draw_spectrum_plot() # Desenhar o grafico do espetro
253                     gc.collect()
254
255         if button == 23: # Botao 2 duplo click
256             gc.collect()
257             spectrum = True # Estamos no espetro
258             reset_spectrum_display() # Reset do display do espetro
259             voltages, Vmax, Vmin, Vavg, x, y = read_and_calculate() # Ler valores ADC e calcular valores necessarios
260             x, y = dft_and_spectrum(voltages) # Calcular o espetro
261             draw_spectrum_plot() # Desenhar o grafico do espetro
262             gc.collect()

```

**Código 1:** Ficheiro de código desenvolvido completo - main.py

## B Funcionalidade do envio do email



**Figura 16:** Funcionamento do clique lento do botão 1 (dados experimentais)