

UNIVERSIDADE FEDERAL DE PERNAMBUCO

Grupo de Pesquisa em Engenharia Biomédica

# Computação Visual

Tela de Aplicação para Raspberry

Jairo Magno

Carlos Gabriel

Recife, 2023

# Sumário

<b>1</b>	<b>Objetivos</b>	<b>1</b>
<b>2</b>	<b>Descrição</b>	<b>2</b>
2.1	Tela de Aplicação Raspberry . . . . .	2
2.1.1	Importação Bibliotecas . . . . .	2
2.1.2	Classe Window . . . . .	3
2.2	Método update_circle_states() . . . . .	5
2.3	Método draw_circles . . . . .	6
2.3.1	Execução da Aplicação . . . . .	7
2.4	Código rasp_update . . . . .	7
<b>3</b>	<b>Referências Bibliográficas</b>	<b>9</b>
<b>A</b>	<b>Apêndice</b>	<b>10</b>

# 1 Objetivos

Este relatório é referente ao trabalho de computação visual da Sprint 04 : 29/07/2023 - 11/08/2023 do grupo de IC's do GPEB. O principal objetivo desse trabalho foi de transformar a tela responsável por verificar a aquisição dos módulos na raspberry. No estado atual a forma de verificar a aquisição se dá pelo terminal. O intuito é criar uma interface mais '*friendly user*' onde ao invés de se verificar pelo terminal, uma nova janela irá aparecer com um desenho de um corpo humano que mostra onde cada módulo se encontra. Além disso através de cores e de um texto indicativo, será mostrado qual o estado de aquisição atual do módulo.

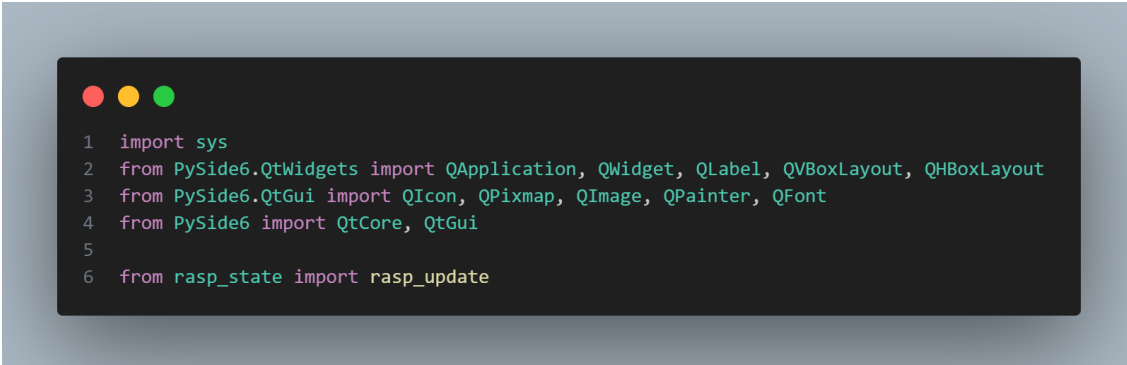
## 2 Descrição

### 2.1 Tela de Aplicação Raspberry

Esta seção é responsável por explicar todo código criado para desenvolver uma tela de aplicação na raspberry. Será feito a explicação dos códigos em trechos. Para a visualização dele de forma completa, pode ser verificado no apêndice nas Figuras A.1 e A.2.

#### 2.1.1 Importação Bibliotecas

Antes de começar a o código em si, será primeiro feito a importação das bibliotecas e módulos utilizados para a criação da aplicação. Elas podem ser vistas nas Figura 2.1.

A screenshot of a code editor with a dark background and light-colored text. The editor has three colored window control buttons (red, yellow, green) in the top-left corner. The code is as follows:

```
1 import sys
2 from PySide6.QtWidgets import QApplication, QWidget, QLabel, QVBoxLayout, QHBoxLayout
3 from PySide6.QtGui import QIcon, QPixmap, QImage, QPainter, QFont
4 from PySide6 import QtCore, QtGui
5
6 from rasp_state import rasp_update
```

Figura 2.1: Importação das Bibliotecas

Dentre elas, a mais importante para essa aplicação foi a biblioteca PySide6. O PySide é uma biblioteca de código aberto que permite criar interfaces gráficas de usuário (GUIs) para aplicativos usando a linguagem de programação Python. Ele fornece um conjunto de módulos que permitem a criação de aplicativos de desktop interativos e visualmente atraentes.

Além disso, na linha 6 está sendo feito a importação de uma função chamada `rasp_update`. Ela será responsável por simular algum eventual script que irá trabalhar em conjunto com o código

desse relatório.

### 2.1.2 Classe Window

A classe Window é responsável por criar a janela principal da aplicação, exibindo uma representação visual dos estados de conexão dos módulos Raspberry Pi. Nesta classe, são definidas configurações iniciais da janela, como título e ícone, e carregada uma imagem do corpo humano. A imagem é exibida no centro da janela usando um layout horizontal (QHBoxLayout). Esse trecho do código pode ser visto na Figura 2.2



```
1 class Window(QWidget):
2     def __init__(self):
3         super().__init__()
4         self.setWindowTitle('Teste Módulos Raspberry')
5         self.setWindowIcon(QIcon('images/Logo_GPEB_Menor.png'))
6
7         #label do Corpo
8         img_body = QImage('images/body.png')
9         pixmap = QPixmap(img_body.scaledToWidth(250))
10        self.body_label = QLabel(self)
11        self.body_label.setPixmap(QPixmap(pixmap))
12        self.body_label.setAlignment(QtCore.Qt.AlignCenter)
13
14        layout = QHBoxLayout(self)
15        layout.addWidget(self.body_label)
```

Figura 2.2: Classe Window

Após isso será feito a criação das frases e círculos informativos. A parte da criação desse código pode ser visto na Figura 2.3. Foi criado uma lista `phrases_and_circles` que armazena quais as frases e cores a serem usadas para indicar o estado em que módulo se encontra. Em seguida um loop percorre a lista `phrases_and_circles`, que contém as frases e cores correspondentes aos estados de conexão. Para cada par de frase e cor, um círculo e uma frase são criados usando os widgets `QLabel`. O layout horizontal (`QHBoxLayout`) chamado `phrase_layout` é usado para agrupar cada círculo e frase, e em seguida, adicionado ao `circle_info_layout`.

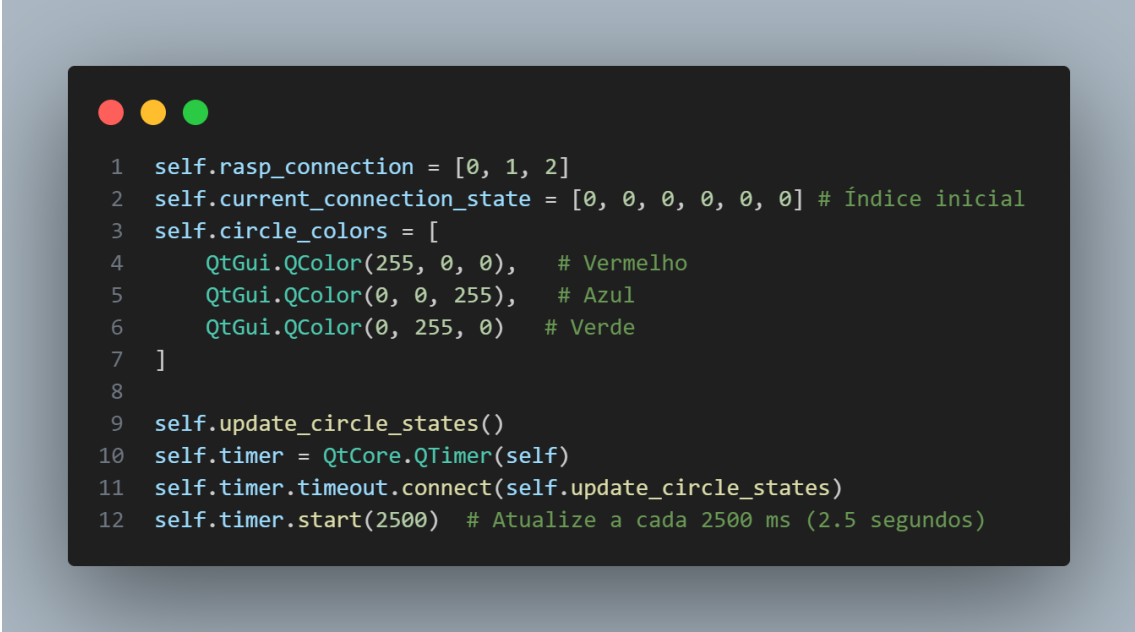


Figura 2.3: Layout para Frases e Círculos

A partir disso será feito a atualização dos estados dos círculos e do desenho do corpo humano (Figura 2.4). São definidos os estados iniciais dos círculos e suas cores, juntamente com as configurações para atualizar periodicamente os estados e cores dos círculos usando um timer. A variável `self.rasp_connection` possui três estados:

- 0 : Não conectado
- 1 : Conectado e não enviando dados
- 2 : Conectado e enviando dados

Já a variável `self.current_connection_state` é uma lista de tamanho seis que guarda o estados dos 6 módulos que serão utilizados. Após isso será chamado o método `update_circle_states()`.

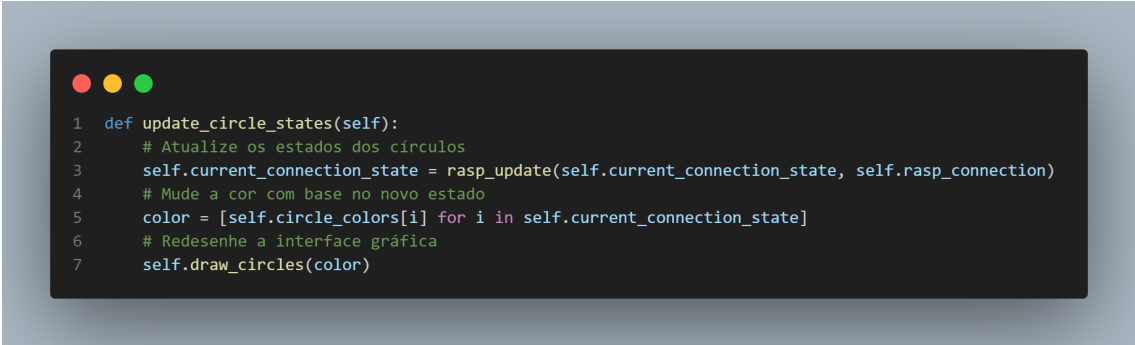


```
1 self.rasp_connection = [0, 1, 2]
2 self.current_connection_state = [0, 0, 0, 0, 0, 0] # Índice inicial
3 self.circle_colors = [
4     QtGui.QColor(255, 0, 0),    # Vermelho
5     QtGui.QColor(0, 0, 255),    # Azul
6     QtGui.QColor(0, 255, 0)     # Verde
7 ]
8
9 self.update_circle_states()
10 self.timer = QtCore.QTimer(self)
11 self.timer.timeout.connect(self.update_circle_states)
12 self.timer.start(2500) # Atualize a cada 2500 ms (2.5 segundos)
```

Figura 2.4: Estados dos Módulos

## 2.2 Método `update_circle_states()`

O método `update_circle_states()` atualiza os estados dos círculos com base em uma função externo chamado `rasp_update` e, em seguida, chama a função `draw_circles` para desenhar os círculos na imagem. A variável `color` é uma list comprehension responsável por pegar estado atual dos módulos e armazenar seu valor para atualizar a cor na imagem do corpo humano. Isso é possível por que ele faz uso da variável `self.circle_color` que também é uma lista com tamanho 3 que já está organizado para casar as cores com estado da `rasp_connection`. O código desse método pode ser visto na Figura 2.5




```
1 def update_circle_states(self):
2     # Atualize os estados dos círculos
3     self.current_connection_state = rasp_update(self.current_connection_state, self.rasp_connection)
4     # Mude a cor com base no novo estado
5     color = [self.circle_colors[i] for i in self.current_connection_state]
6     # Redesenhe a interface gráfica
7     self.draw_circles(color)
```

Figura 2.5: Método `update_circle_states()`

## 2.3 Método draw\_circles

A função `draw_circles` desenha os círculos coloridos na imagem do corpo humano. Os círculos são posicionados usando coordenadas predefinidas e suas cores são determinadas pelos estados de conexão atualizados. O trecho do seu código pode ser visto na Figura 2.6.



```
1 def draw_circles(self, color):
2     pixmap = QPixmap(self.body_label.pixmap())
3     painter = QPainter(pixmap)
4
5     circle_coordinates = [
6         (30, 340), # Mão esquerda
7         (225, 340), # Mão direita
8         (90, 600), # Perna esquerda
9         (160, 600), # Perna direita
10        (128, 280), # Barriga ECG
11        (128, 180) # Peito Temperatura
12    ]
13    circle_radius = 15
14
15    for (x, y), circle_color in zip(circle_coordinates, color):
16        painter.setBrush(circle_color)
17        painter.drawEllipse(x - circle_radius, y - circle_radius, 2 * circle_radius, 2 * circle_radius)
18    painter.end()
19
20    self.body_label.setPixmap(pixmap)
```

Figura 2.6: Método `draw_circles`

O trecho da linha 15 até 18 é um loop para percorrer as coordenadas dos círculos e as cores correspondentes que foram calculadas com base nos estados de conexão. A função `zip` é usada para combinar as listas `circle_coordinates` (coordenadas dos círculos) e `color` (cores dos círculos). Dentro do `For`, está sendo definido a cor do pincel (brush) que será usado para desenhar o círculo. A cor é definida com base na lista de cores '`color`' correspondente ao estado atual. A linha 17 é responsável por desenhar o círculo na posição especificada pelas coordenadas (`x`, `y`). O método `drawEllipse` desenha uma elipse (que é um círculo quando os raios horizontal e vertical são iguais) nas coordenadas especificadas. Os parâmetros passados para `drawEllipse` são: a posição `x` e `y` do canto superior esquerdo do retângulo circunscrito ao círculo (calculado subtraindo o raio do círculo das coordenadas), a largura e altura do retângulo circunscrito (calculado multiplicando o raio por 2).



### 2.3.1 Execução da Aplicação

Nesta última parte do código, a aplicação é iniciada. Um objeto da classe `QApplication` é criado, e a janela principal (`Window`) é instanciada e exibida com as dimensões de 1000x600 pixels. O programa é finalizado quando a janela é fechada. Seu trecho no código pode ser visto na Figura 2.7.



Figura 2.7: Execução da Aplicação

## 2.4 Código rasp\_update

Como dito no início do relatório, para verificar se a atualização dos estados realmente está ocorrendo, foi criado um script externo com a ideia de simular o eventual programa que venha a realmente ter a comunicação com os estados dos módulos. Dessa forma, foi apenas criado um programa que mudasse a lista `self.current_connection_state` de forma pseudo-aleatória. O código

criado pode ser visto na Figura 2.8.

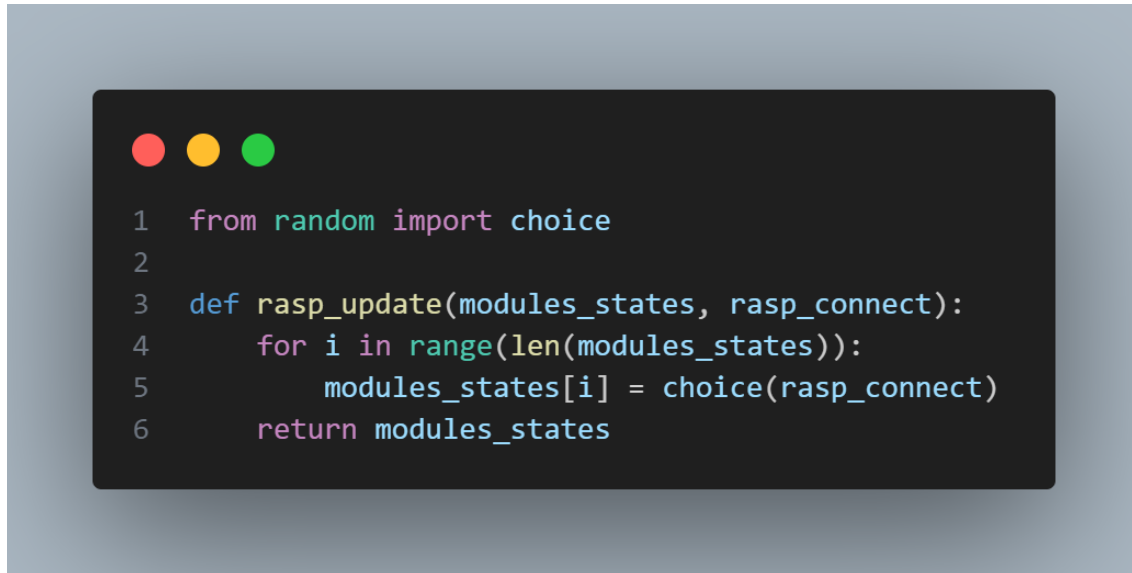


Figura 2.8: Simulação do Script Externo

### 3 Referências Bibliográficas

[1] Documentário PySide6. Disponível em:  
<<https://doc.qt.io/qtforpython-6/quickstart.html>>.



# A Apêndice

```
1  """
2  Autor: Jairo Magno Caracciolo Marques
3  Data de criação: 08/08/2023
4  """
5  import sys
6  from PySide6.QtWidgets import QApplication, QWidget, QLabel, QVBoxLayout, QHBoxLayout
7  from PySide6.QtGui import QIcon, QPixmap, QImage, QPainter, QFont
8  from PySide6 import QtCore, QtGui
9
10 from rasp_state import rasp_update
11
12 class Window(QWidget):
13     def __init__(self):
14         super().__init__()
15         self.setWindowTitle('Teste Módulos Raspberry')
16         self.setWindowIcon(QIcon('images/Logo_GPEB_Menor.png'))
17
18         #label do Corpo
19         img_body = QImage('images/body.png')
20         pixmap = QPixmap(img_body.scaledToWidth(250))
21         self.body_label = QLabel(self)
22         self.body_label.setPixmap(QPixmap(pixmap))
23         self.body_label.setAlignment(QtCore.Qt.AlignCenter)
24
25         layout = QHBoxLayout(self)
26         layout.addWidget(self.body_label)
27
28         # Layout para os círculos informativos e as frases
29         circle_info_layout = QVBoxLayout()
30         serisFont_1 = QFont('Times', 15)
31         serisFont_2 = QFont('Times', 40)
32
33         # Criação das frases e círculos informativos
34         phrases_and_circles = [
35             ('Não Conectado', QtGui.QColor(255, 0, 0)), # Vermelho
36             ('Conectado e Não Enviando Dados', QtGui.QColor(0, 0, 255)), # Azul
37             ('Conectado e Enviando Dados', QtGui.QColor(0, 255, 0)) # Verde
38         ]
39
40         for phrase, circle_color in phrases_and_circles:
41             circle_label = QLabel('●')
42             circle_label.setFont(serisFont_2)
43             circle_label.setStyleSheet(f'color: {circle_color.name()}')
44
45             phrase_label = QLabel(phrase)
46             phrase_label.setFont(serisFont_1)
47
48             phrase_layout = QHBoxLayout()
49             phrase_layout.addWidget(circle_label)
50             phrase_layout.addSpacing(-400)
51             phrase_layout.addWidget(phrase_label)
52
53             circle_info_layout.addLayout(phrase_layout)
54
55         layout.addLayout(circle_info_layout)
56         self.setLayout(layout)
```

Figura A.1: Código da Classe Window Pt.1

```

1 self.rasp_connection = [0, 1, 2]
2     self.current_connection_state = [0, 0, 0, 0, 0, 0] # Índice inicial
3     self.circle_colors = [
4         QtGui.QColor(255, 0, 0),    # Vermelho
5         QtGui.QColor(0, 0, 255),    # Azul
6         QtGui.QColor(0, 255, 0)    # Verde
7     ]
8
9     self.update_circle_states()
10    self.timer = QtCore.QTimer(self)
11    self.timer.timeout.connect(self.update_circle_states)
12    self.timer.start(2500) # Atualize a cada 2500 ms (2.5 segundos)
13
14    def update_circle_states(self):
15        # Atualize os estados dos círculos
16        self.current_connection_state = rasp_update(self.current_connection_state, self.rasp_connection)
17        # Mude a cor com base no novo estado
18        color = [self.circle_colors[i] for i in self.current_connection_state]
19        # Redesenhe a interface gráfica
20        self.draw_circles(color)
21
22    def draw_circles(self, color):
23        pixmap = QPixmap(self.body_label.pixmap())
24        painter = QPainter(pixmap)
25
26        circle_coordinates = [
27            (30, 340),    # Mão esquerda
28            (225, 340),   # Mão direita
29            (90, 600),    # Perna esquerda
30            (160, 600),   # Perna direita
31            (128, 280),   # Barriga ECG
32            (128, 180)    # Peito Temperatura
33        ]
34        circle_radius = 15
35
36        for (x, y), circle_color in zip(circle_coordinates, color):
37            painter.setBrush(circle_color)
38            painter.drawEllipse(x - circle_radius, y - circle_radius, 2 * circle_radius, 2 * circle_radius)
39        painter.end()
40
41        self.body_label.setPixmap(pixmap)
42
43    if __name__ == '__main__':
44        app = QApplication([])
45
46        window = Window()
47        window.resize(1000, 600)
48        window.show()
49
50    sys.exit(app.exec())
51

```

Figura A.2: Código da Classe Window Pt.2