Diseño de una prensa de mazapanes

Martinez Pérez Cristian Uriel[1]

***Resumen*—** El diseño de esta interfaz se compone de varios elementos clave para facilitar la interacción del usuario con un sistema específico:

Componentes físicos:

Botón físico 1: Este botón se encuentra físicamente en el dispositivo y se puede utilizar para activar una función determinada. Por ejemplo, si el sistema está diseñado para controlar un dispositivo electrónico, este botón podría encenderlo o apagarlo.

Botón físico 2: Similar al primero, este botón físico tiene otra función específica asignada. Dependiendo del contexto del sistema, podría activar una acción diferente. Por ejemplo, si estamos considerando una aplicación de control remoto, este botón podría cambiar entre diferentes modos de funcionamiento.

Componente virtual:
3. Botón virtual: Este elemento de la interfaz es generado digitalmente y se muestra en la pantalla del dispositivo. A diferencia de los botones físicos, este botón puede tener una funcionalidad más dinámica y adaptable. Por ejemplo, podría desencadenar una serie de acciones programadas o permitir al usuario personalizar su función según sus preferencias.

Salidas:

Salida 1: Una de las salidas está destinada a proporcionar retroalimentación al usuario sobre la acción realizada a través del botón físico 1. Esto podría ser en forma de mensajes de confirmación, cambios visuales en la pantalla o cualquier otro tipo de programación relevante.

Salida 2: Similar a la primera salida, esta segunda salida tiene la función de comunicar información sobre la acción realizada a través del botón físico 2. Esto podría incluir datos adicionales, confirmaciones de estado o cualquier otra información relevante para el usuario.

Introducción

En el diseño de interfaces de usuario, la combinación efectiva de elementos físicos y virtuales desempeña un papel crucial para mejorar la experiencia del usuario y la usabilidad de un sistema. En este contexto, se ha propuesto un diseño de interfaz que integra dos botones físicos, un botón virtual y dos salidas. Este diseño busca optimizar la interacción entre el usuario y el sistema, aprovechando las ventajas de la manipulación física y la flexibilidad digital para ofrecer una experiencia intuitiva y satisfactoria.

Objetivo

El objetivo principal de este diseño de interfaz es proporcionar al usuario un medio eficiente y cómodo para interactuar con el sistema, facilitando la realización de acciones específicas y proporcionando una programación clara y relevante. A través de la combinación de dos botones físicos y uno virtual, permitiendo una interacción fluida y efectiva con el sistema. Además, las salidas integradas tienen como objetivo proporcionar información inmediata y comprensible sobre las acciones realizadas.

Metodología

```
import sys
from PyQt5.QtWidgets import QApplication, QWidget, QGridLayout, QLabel
from PyQt5.QtCore import Qt, QTimer
from gpiozero import Button, LED

class CircleLabel(QLabel):
    def __init__(self, text, parent=None):
        super().__init__(text, parent)
        self.setFixedSize(50, 50)
        self.setStyleSheet('background-color: red; color: white;')
        self.setAlignment(Qt.AlignCenter)

    def update_color(self, active):
        if active:
            self.setStyleSheet('background-color: green; color: white;')
        else:
            self.setStyleSheet('background-color: red; color: white;')

class VentanaPrincipal(QWidget):
    def __init__(self):
        super().__init__()
        self.init_ui()
        self.init_gpio()

    def init_ui(self):
        self.layout = QGridLayout(self)

        # Entradas
        self.hs01_button = CircleLabel('HS-01')
        self.hs02_button = CircleLabel('HS-02')
        self.hs03_button = CircleLabel('HS-03')

        # Salidas
        self.pl01_led = CircleLabel('PL-01')
        self.pl02_led = CircleLabel('PL-02')

        # Setup grid
        self.layout.addWidget(QLabel('Entradas'), 0, 0, Qt.AlignCenter)
        self.layout.addWidget(QLabel('Salidas'), 0, 1, Qt.AlignCenter)
        self.layout.addWidget(self.hs01_button, 1, 0, Qt.AlignCenter)
        self.layout.addWidget(self.hs02_button, 2, 0, Qt.AlignCenter)
        self.layout.addWidget(self.hs03_button, 3, 0, Qt.AlignCenter)
        self.layout.addWidget(self.pl01_led, 1, 1, Qt.AlignCenter)
```

```
        self.layout.addWidget(self.pl02_led, 2, 1, Qt.AlignCenter)

        self.setWindowTitle('Interfaz de Control')
        self.setGeometry(100, 100, 300, 200)

    def init_gpio(self):
        self.hs01_button.mousePressEvent = self.update_hs01
        self.hs01_button.mouseReleaseEvent = lambda event:
self.hs01_button.update_color(False)
        self.hs02_button.mousePressEvent = self.toggle_hs02
        self.hs02_button.mouseReleaseEvent = lambda event:
self.hs02_button.update_color(False)
        self.hs03_button.mousePressEvent = self.toggle_hs03

        self.hs01 = Button(17)  # Asumiendo GPIO 17 para hs-01
        self.hs02 = Button(27)  # Asumiendo GPIO 27 para hs-02
        self.pl01 = LED(5)  # Asumiendo GPIO 5 para pl-01
        self.pl02 = LED(6)  # Asumiendo GPIO 6 para pl-02

        self.timer = QTimer()
        self.timer.timeout.connect(self.update_outputs)
        self.timer.start(100)  # Actualizar cada 100 ms

        self.pl02_timer = QTimer()
        self.pl02_timer.timeout.connect(self.toggle_pl02)
        self.pl02_state = False  # Estado inicial apagado
        self.pl02_active = False  # Para controlar el estado de la secuencia

    def update_hs01(self, event):
        self.hs01_button.update_color(True)
        if self.hs03_button.styleSheet().find('background-color: green') != -1:
            self.pl01.on()
            QTimer.singleShot(5000, lambda: self.pl01.off())
        else:
            self.pl01.on()
            QTimer.singleShot(10000, lambda: self.pl01.off())

    def toggle_hs02(self, event):
        self.hs02_button.update_color(True)
        if self.pl02_active:
            self.stop_pl02_sequence()
        else:
            self.start_pl02_sequence()

    def start_pl02_sequence(self):
```

```
        self.pl02_active = True
        self.pl02_state = True
        self.pl02.on()
        self.pl02_led.update_color(True)
        self.pl02_timer.start(3000)  # Comienza la secuencia de 3 segundos
encendido

    def stop_pl02_sequence(self):
        self.pl02_active = False
        self.pl02_timer.stop()
        self.pl02.off()
        self.pl02_led.update_color(False)
        self.pl02_state = False

    def toggle_pl02(self):
        if not self.pl02_active:
            return
        if self.pl02_state:
            self.pl02.off()
            self.pl02_led.update_color(False)
            self.pl02_state = False
            self.pl02_timer.start(1000)  # 1 segundo apagado
        else:
            self.pl02.on()
            self.pl02_led.update_color(True)
            self.pl02_state = True
            self.pl02_timer.start(3000)  # 3 segundos encendido

    def toggle_hs03(self, event):
        current_color = self.hs03_button.styleSheet().find('background-color:
green')
        self.hs03_button.update_color(False if current_color != -1 else True)

    def update_outputs(self):
        self.pl01_led.update_color(self.pl01.is_lit)
        self.pl02_led.update_color(self.pl02.is_lit)

if __name__ == '__main__':
    app = QApplication(sys.argv)
    window = VentanaPrincipal()
    window.show()
    sys.exit(app.exec_())
```

Fig.1 Interfaz 1

Interfaz 1 aquí solo se hizo el diseño de la interfaz que contiene dos botones físicos y uno virtual y dos salidas esta interfaz va a servir para decirnos cual de los botones se activó y cuales están desactivados, lo mismo aran las salidas, y cada una de ellas cambiaran de color verde. (Fig.1)

Resultados

Interfaz de dos leds

```
import sys
from PyQt5.QtWidgets import QApplication, QWidget, QGridLayout, QLabel,
QPushButton
from PyQt5.QtCore import Qt, QTimer
from gpiozero import Button, LED
import threading
import time

class CircleLabel(QLabel):
    def __init__(self, text, parent=None):
        super().__init__(text, parent)
        self.setFixedSize(50, 50)
        self.update_color(False)  # Establecer el color inicial
        self.setAlignment(Qt.AlignCenter)

    def update_color(self, active):
        if active:
            self.setStyleSheet('background-color: green; color: white;')
        else:
            self.setStyleSheet('background-color: red; color: white;')

class VentanaPrincipal(QWidget):
    def __init__(self):
        super().__init__()
```

```
        self.init_ui()
        self.init_gpio()

    def init_ui(self):
        self.layout = QGridLayout(self)

        # Entradas
        self.hs01_button = CircleLabel('HS-01')
        self.hs02_button = CircleLabel('HS-02')
        self.hs03_button = QPushButton('HS-03')  # Botón virtual para HS-03

        # Salidas
        self.pl01_led = CircleLabel('PL-01')
        self.pl02_led = CircleLabel('PL-02')

        # Setup grid
        self.layout.addWidget(QLabel('Entradas'), 0, 0, Qt.AlignCenter)
        self.layout.addWidget(QLabel('Salidas'), 0, 1, Qt.AlignCenter)
        self.layout.addWidget(self.hs01_button, 1, 0, Qt.AlignCenter)
        self.layout.addWidget(self.hs02_button, 2, 0, Qt.AlignCenter)
        self.layout.addWidget(self.hs03_button, 3, 0, Qt.AlignCenter)
        self.layout.addWidget(self.pl01_led, 1, 1, Qt.AlignCenter)
        self.layout.addWidget(self.pl02_led, 2, 1, Qt.AlignCenter)

        self.setWindowTitle('Interfaz de Control')
        self.setGeometry(100, 100, 300, 200)

    def init_gpio(self):
        # Configurar botones físicos
        self.hs01_button_gpio = Button(17)  # GPIO 17 para hs-01
        self.hs02_button_gpio = Button(27)  # GPIO 27 para hs-02

        # Configurar LEDs
        self.pl01 = LED(5)  # GPIO 5 para pl-01
        self.pl02 = LED(6)  # GPIO 6 para pl-02

        # Conectar eventos de botones físicos a métodos
        self.hs01_button_gpio.when_pressed = self.hs01_pressed
        self.hs01_button_gpio.when_released = self.hs_released
        self.hs02_button_gpio.when_pressed = self.hs02_pressed
        self.hs02_button_gpio.when_released = self.hs_released

        # Conectar evento de botón virtual a método
        self.hs03_button.clicked.connect(self.toggle_hs03)
```

```
        # Variables para controlar el parpadeo de pl-02
        self.pl02_blinking = False
        self.pl02_thread = threading.Thread(target=self.blink_pl02)

    def hs01_pressed(self):
        self.hs01_button.update_color(True)
        if self.hs03_button.text() == 'HS-03 Activo':  # Solo activa la salida si
hs-03 está activo
            self.pl01.on()
            self.pl01_led.update_color(True)
            threading.Timer(5, self.deactivate_pl01).start()
        else:
            self.pl01.on()
            self.pl01_led.update_color(True)
            threading.Timer(10, self.deactivate_pl01).start()

    def deactivate_pl01(self):
        self.pl01.off()
        self.pl01_led.update_color(False)

    def hs02_pressed(self):
        self.hs02_button.update_color(True)
        if not self.pl02_blinking:
            self.pl02_blinking = True
            if not self.pl02_thread.is_alive():
                self.pl02_thread = threading.Thread(target=self.blink_pl02)
                self.pl02_thread.start()
        else:
            self.pl02_blinking = False
            self.pl02.off()
            self.pl02_led.update_color(False)

    def hs_released(self):
        self.hs01_button.update_color(False)
        self.hs02_button.update_color(False)

    def toggle_hs03(self):
        if self.hs03_button.text() == 'HS-03':  # Comprueba el texto del botón
para determinar su estado
            self.hs03_button.setText('HS-03 Activo')
            self.hs03_button.setStyleSheet('background-color: green; color:
white;')
        else:
            self.hs03_button.setText('HS-03')
```

```
            self.hs03_button.setStyleSheet('background-color: red; color:
white;')

    def blink_pl02(self):
        while self.pl02_blinking:
            self.pl02.on()
            self.pl02_led.update_color(True)  # Actualizar color de pl02_led
            time.sleep(3)  # Encender pl-02 durante 3 segundos
            self.pl02.off()
            self.pl02_led.update_color(False)  # Actualizar color de pl02_led
            time.sleep(1)  # Apagar pl-02 durante 1 segundo

if __name__ == '__main__':
    app = QApplication(sys.argv)
    window = VentanaPrincipal()
    window.show()
    sys.exit(app.exec_())
```

Fig.2

Con el código que se realizo se logró hacer esta interfaz que nos dice cuál de los botones y salidas están activadas, este código lo que hace también es que al presionar el botón virtual hs-03 y luego presionas el botón físico hs-01 la salida pl-01 deberá de estar activadas durante 5 segundos y cuando se desactive el hs-03 y se vuelva a presionar hs-01 la salida deberá estar activada durante 10 segundos.
El botón físico hs-02 al momento de presionarlo hará que se prenda por tres segundos la salida pl-02 y se apagar un segundo y repetirá el mismo ciclo hasta que se vuelva a presionar el botón es cuando se va a pagar definitivamente. (Fig.2)

Interfaz con led y servomotor

```
import sys
from PyQt5.QtWidgets import QApplication, QWidget, QGridLayout, QLabel,
QPushButton
from PyQt5.QtCore import Qt, QTimer
from gpiozero import Button, LED, Servo
import threading
import time

class CircleLabel(QLabel):
    def __init__(self, text, parent=None):
        super().__init__(text, parent)
        self.setFixedSize(50, 50)
        self.update_color(False)  # Establecer el color inicial
        self.setAlignment(Qt.AlignCenter)

    def update_color(self, active):
        if active:
            self.setStyleSheet('background-color: green; color: white;')
        else:
            self.setStyleSheet('background-color: red; color: white;')

class VentanaPrincipal(QWidget):
    def __init__(self):
        super().__init__()
        self.init_ui()
        self.init_gpio()

    def init_ui(self):
        self.layout = QGridLayout(self)

        # Entradas
        self.hs01_button = CircleLabel('HS-01')
        self.hs02_button = CircleLabel('HS-02')
        self.hs03_button = QPushButton('HS-03')  # Botón virtual para HS-03

        # Salidas
        self.pl01_led = CircleLabel('PL-01')
        self.pl02_led = CircleLabel('PL-02')

        # Setup grid
        self.layout.addWidget(QLabel('Entradas'), 0, 0, Qt.AlignCenter)
        self.layout.addWidget(QLabel('Salidas'), 0, 1, Qt.AlignCenter)
        self.layout.addWidget(self.hs01_button, 1, 0, Qt.AlignCenter)
        self.layout.addWidget(self.hs02_button, 2, 0, Qt.AlignCenter)
```

```
        self.layout.addWidget(self.hs03_button, 3, 0, Qt.AlignCenter)
        self.layout.addWidget(self.pl01_led, 1, 1, Qt.AlignCenter)
        self.layout.addWidget(self.pl02_led, 2, 1, Qt.AlignCenter)

        self.setWindowTitle('Interfaz de Control')
        self.setGeometry(100, 100, 300, 200)

    def init_gpio(self):
        # Configurar botones físicos
        self.hs01_button_gpio = Button(17)  # GPIO 17 para hs-01
        self.hs02_button_gpio = Button(27)  # GPIO 27 para hs-02

        # Configurar Servo en lugar de LED
        self.pl01 = Servo(5)  # GPIO 5 para pl-01
        self.pl02 = LED(6)    # GPIO 6 para pl-02

        # Conectar eventos de botones físicos a métodos
        self.hs01_button_gpio.when_pressed = self.hs01_pressed
        self.hs01_button_gpio.when_released = self.hs_released
        self.hs02_button_gpio.when_pressed = self.hs02_pressed
        self.hs02_button_gpio.when_released = self.hs_released

        # Conectar evento de botón virtual a método
        self.hs03_button.clicked.connect(self.toggle_hs03)

        # Variables para controlar el parpadeo de pl-02
        self.pl02_blinking = False
        self.pl02_thread = threading.Thread(target=self.blink_pl02)

    def hs01_pressed(self):
        self.hs01_button.update_color(True)
        if self.hs03_button.text() == 'HS-03 Activo':  # Solo activa la salida si
hs-03 está activo
            self.move_servo(self.pl01, 1)  # Mover servo a posición 1 (máximo)
            self.pl01_led.update_color(True)
            threading.Timer(5, self.deactivate_pl01).start()
        else:
            self.move_servo(self.pl01, 0.5)  # Mover servo a posición intermedia
(0.5)
            self.pl01_led.update_color(True)
            threading.Timer(10, self.deactivate_pl01).start()

    def deactivate_pl01(self):
        self.move_servo(self.pl01, -1)  # Mover servo a posición -1 (mínimo)
        self.pl01_led.update_color(False)
```

```
    def hs02_pressed(self):
        self.hs02_button.update_color(True)
        if not self.pl02_blinking:
            self.pl02_blinking = True
            if not self.pl02_thread.is_alive():
                self.pl02_thread = threading.Thread(target=self.blink_pl02)
                self.pl02_thread.start()
        else:
            self.pl02_blinking = False
            self.pl02.off()
            self.pl02_led.update_color(False)

    def hs_released(self):
        self.hs01_button.update_color(False)
        self.hs02_button.update_color(False)

    def toggle_hs03(self):
        if self.hs03_button.text() == 'HS-03':  # Comprueba el texto del botón
para determinar su estado
            self.hs03_button.setText('HS-03 Activo')
            self.hs03_button.setStyleSheet('background-color: green; color:
white;')
        else:
            self.hs03_button.setText('HS-03')
            self.hs03_button.setStyleSheet('background-color: red; color:
white;')

    def blink_pl02(self):
        while self.pl02_blinking:
            self.pl02.on()
            self.pl02_led.update_color(True)  # Actualizar color de pl02_led
            time.sleep(3)  # Encender pl-02 durante 3 segundos
            self.pl02.off()
            self.pl02_led.update_color(False)  # Actualizar color de pl02_led
            time.sleep(1)  # Apagar pl-02 durante 1 segundo

    def move_servo(self, servo, position):
        servo.value = position

if __name__ == '__main__':
    app = QApplication(sys.argv)
    window = VentanaPrincipal()
    window.show()
    sys.exit(app.exec_())
```

Este código hace lo mismo que el de arriba solo con la diferencia que se utiliza un servomotor la activación de las salidas son las mismas al igual que la configuración de los botones

Análisis

El código proporcionado es una aplicación de interfaz gráfica de usuario (GUI) desarrollada con PyQt5, una biblioteca de Python para crear aplicaciones de escritorio. La interfaz consiste en dos botones físicos (representados por etiquetas circulares), un botón virtual y dos salidas (también representadas por etiquetas circulares).

El diseño de la interfaz es claro y organizado, con una disposición de cuadrícula que separa las entradas de las salidas. Los botones físicos están vinculados a eventos de pulsación y liberación, mientras que el botón virtual está conectado a un método de alternancia. Las salidas pueden cambiar de color para indicar su estado.

El código utiliza la biblioteca gpiozero para interactuar con componentes electrónicos externos, como botones y LEDs. Además, se implementan subprocesos (threads) para controlar el parpadeo de una de las salidas.

Conclusiones

En resumen, el código proporciona una base sólida para una interfaz de control que combina elementos físicos y virtuales para interactuar con dispositivos electrónicos. Sin embargo, para mejorar su robustez y confiabilidad, sería necesario agregar manejo de errores y excepciones, así como también realizar pruebas exhaustivas en una variedad de entornos de hardware para garantizar su funcionamiento óptimo.

Limitaciones

Una limitación de este código es la falta de manejo de errores y excepciones. No se realizan comprobaciones para garantizar que los dispositivos GPIO estén disponibles o que las operaciones de E/S se completen correctamente. Esto puede llevar a errores no controlados y a un comportamiento inesperado de la aplicación, especialmente en entornos donde los recursos de hardware pueden ser escasos o compartidos.

Referencias

OSCAR, S. F. (1 de 02 de 2024). *CURSO PYTHON* . Obtenido de CURSO PYTHON : https://github.com/SigfridoO