



UNIVERSIDAD DEL BÍO-BÍO

23 Tutorial control plc

Estudiantes Practicantes:

Luis Pereira

Profesor:

Luis Vera

Laboratorio CIMUBB

2023-2

Introducción:

A continuación se explicara el programa con implementación en Python utilizando la biblioteca Tkinter para crear una interfaz gráfica de usuario (GUI) que interactúa con un robot a través de una conexión serial y nos permitirá que el programa nos responda comunicándonos en que estación se encuentra un palette. Aquí hay un tutorial paso a paso para entender y modificar el código:

1. Importar bibliotecas:

```
import tkinter as tk
from tkinter import ttk, messagebox
import serial
import threading
import time
```

A diferencia de los tutoriales anteriores, utilizaremos ahora la biblioteca threading para estar leyendo continuamente datos que se recibirán del robot, en otras palabras nos permite manejar la lectura continua de datos en un hilo separado.

2. Clase ControlPLC:

Utilizaremos clases en esta ocasión porque el uso de clases en este caso mejora la claridad, la organización y la mantenibilidad del código, al tiempo que facilita la reutilización y la expansión de la funcionalidad en el futuro.

La clase **ControlPLC** encapsula toda la lógica relacionada con la interacción del robot. Esto significa que las variables y métodos están encapsulados en un solo lugar, lo que facilita su mantenimiento y modificación.

La modularidad se mejora al dividir la funcionalidad en métodos específicos de la clase, como **conectar_robot**, **desconectar_robot**, **enviar_instruccion**, y **leer_datos_continuamente**. Esto hace que cada parte del código sea más fácil de entender y modificar sin afectar otras partes.

Funciones en la clase:

1) conectar_robot(self,port):

```
def conectar_robot(self, port):
    try:
        self.serial_port = serial.Serial(port, baudrate=9600, timeout=1, bytesize=7, parity="E", stopbits=2)
        self.thread_leer_datos = threading.Thread(target=self.leer_datos_continuamente, daemon=True)
        self.thread_leer_datos.start()
        messagebox.showinfo(message="Conexión exitosa con el robot.")
    except Exception as e:
        messagebox.showerror(message=f"Error al conectar con el robot: {e}")
```

1) **self.serial_port = serial.Serial(port, baudrate=9600, timeout=1)**

- Crea una instancia de la clase serial.Serial del módulo serial, que se utiliza para la comunicación serial.
- **port**: Es el puerto serial al que se intentará conectar el robot.
- **baudrate=9600**: Establece la velocidad de transmisión de datos en 9600 baudios
- **bytesize = 7** Establece una longitud de bits de 7 para la transmisión de datos.
- **stopbits = 2** Se utilizan 2 bits de parada al final de cada byte de datos.
- **parity="E"** Establece la paridad en even con la letra "E" mayúscula.
- **timeout=1**: Configura un tiempo de espera de 1 segundo para las operaciones de lectura.

2) **self.thread_leer_datos= threading.Thread(target=self.leer_datos_continuamente, daemon=True)**

- Crea un hilo (**threading.Thread**) que ejecutará la función **self.leer_datos_continuamente**.
- **target**: Especifica la función que será ejecutada en el hilo.
- **daemon=True**: Configura el hilo como un hilo demonio, lo que significa que se cerrará cuando el programa principal termine.

3) **self.thread_leer_datos.start()**

- Inicia el hilo creado en el paso anterior, lo que permite que la función **leer_datos_continuamente** se ejecute en paralelo con el resto del programa.

4) **messagebox.showinfo(message="Conexión exitosa con el robot .")**

- Si la conexión serial y el hilo se establecen con éxito, se muestra un mensaje informativo de éxito utilizando la función showinfo del módulo messagebox de Tkinter.

except Exception as e:

- Si algo sale mal durante el intento de conexión (por ejemplo, si el puerto no está disponible o hay algún otro error), se captura la excepción y se

muestra un mensaje de error utilizando **showerror** del módulo **messagebox**.

En resumen, esta función intenta establecer una conexión serial con el robot a través del puerto especificado. Si la conexión es exitosa, inicia un hilo para leer datos continuamente y muestra un mensaje de éxito. Si hay algún error durante el proceso, muestra un mensaje de error.

Observación: el baud rate en 9600 bits por segundo. Esto significa que cada segundo, el robot y el dispositivo conectado intercambiarán información a una velocidad de 9600 bits por segundo. Es crucial que el robot esté configurado con la misma velocidad de baudios para garantizar una comunicación efectiva.

2) desconectar_robot(self):

```
def desconectar_robot(self):
    try:
        if self.thread_leer_datos and self.thread_leer_datos.is_alive():
            self.thread_leer_datos.join()
        if self.serial_port and self.serial_port.isOpen():
            self.serial_port.close()
            messagebox.showinfo(message="Desconexión exitosa del robot Scrobot.")
    except Exception as e:
        messagebox.showerror(message=f"Error al desconectar el robot Scrobot: {e}")
```

La función **desconectar_robot** se encarga de cerrar la conexión con el robot. En primer lugar, verifica si el hilo de lectura de datos está en ejecución y, en caso afirmativo, espera a que termine antes de continuar. Luego, comprueba si la conexión serial está abierta y la cierra. Después de realizar estas acciones, muestra un mensaje informativo indicando que la desconexión fue exitosa. En caso de que ocurra algún error durante el proceso, captura la excepción y muestra un mensaje de error detallado. En resumen, la función asegura una desconexión controlada y proporciona retroalimentación al usuario.

3) enviar instruccion(self,instruccion):

```
def enviar_instruccion(self, instruccion):
    try:
        if self.serial_port and self.serial_port.isOpen():
            # Limpiar el búfer antes de enviar una nueva instrucción
            self.response_buffer = ""

            # Agregar un carácter de retorno de carro al final de la instrucción
            instruccion = f"{instruccion}\r"
            self.serial_port.write(instruccion.encode())
            messagebox.showinfo(message="Instrucción enviada correctamente.")

            # Agregar una pausa antes de leer los datos
            time.sleep(0.23)

            # Leer datos después de la pausa
            datos = self.serial_port.read_all().decode()
            time.sleep(0.15)
            if datos:
                TextRecibidos.insert(tk.END, datos)
                if "Done" in datos or "OK" in datos:
                    TextInterpretacion.insert(tk.END, "Se recibio un Done o un OK en los datos recibidos" + "\n")
    except Exception as e:
        messagebox.showerror(message=f"Error al enviar la instrucción al robot Scorbot: {e}")
```

1) Verificar la conexión serial:

```
if self.serial_port and self.serial_port.isOpen():
```

Verifica que la conexión serial (self.serial_port) esté establecida y abierta (isOpen()). Si la conexión no está abierta, la función no continuará y mostrará un mensaje de error.

2) Limpiar el búfer y enviar la instrucción:

```
self.response_buffer = ""

# Agregar un carácter de retorno de carro al final de la instrucción
instruccion = f"{instruccion}\r"
self.serial_port.write(instruccion.encode())
```

- Limpia el búfer (**self.response_buffer**) antes de enviar una nueva instrucción.
- Agrega un carácter de retorno de carro (**\r**) al final de la instrucción.
- Utiliza write para enviar la instrucción codificada al robot a través de la conexión serial.

3) Agregar pausas antes y después de leer datos:

```

# Agregar una pausa antes de leer los datos
time.sleep(0.23)

# Leer datos después de la pausa
datos = self.serial_port.read_all().decode()
time.sleep(0.15)

```

En resumen, la función **enviar_instruccion()** se encarga de enviar una instrucción al robot a través de la conexión serial. Comienza verificando que la conexión esté establecida y abierta. Luego, limpia el búfer, agrega un retorno de carro a la instrucción y la envía al robot. Después de enviar la instrucción, muestra un mensaje de éxito. Añade pausas antes y después de leer datos y muestra los datos recibidos en la interfaz gráfica. Además, verifica si la respuesta contiene "Done" o "OK" y muestra un mensaje correspondiente. En caso de cualquier error durante el proceso, captura la excepción y muestra un mensaje de error detallado. En resumen, la función controla el envío de instrucciones, gestiona la recepción de datos y proporciona retroalimentación al usuario en la interfaz gráfica.

4) leer_datos_continuamente(self):

```

def leer_datos_continuamente(self):
    global TextInterpretacion

    while True:
        try:
            if self.serial_port and self.serial_port.isOpen():
                datos = self.serial_port.read_all().decode()

                if datos:
                    # Agregar datos al búfer de respuesta
                    self.response_buffer += datos

                    # Buscar "Done." y "OK" en el búfer de respuesta
                    if "Done." in self.response_buffer:
                        TextInterpretacion.insert(tk.END, "Se recibió un Done." + "\n")
                        TextRecibidos.insert(tk.END, "> " + self.response_buffer + "\n")
                        self.response_buffer = "" # Limpiar el búfer
                    elif "ok" in self.response_buffer:
                        TextInterpretacion.insert(tk.END, "Se recibió un ok" + "\n")
                        TextRecibidos.insert(tk.END, self.response_buffer + "\n")
                        self.response_buffer = "" # Limpiar el búfer
                    elif "OK" in self.response_buffer:
                        TextInterpretacion.insert(tk.END, "Se recibió un OK" + "\n")
                        TextRecibidos.insert(tk.END, self.response_buffer + "\n")
                        self.response_buffer = "" # Limpiar el búfer
                    else:
                        # Usar after para actualizar la interfaz gráfica en el hilo principal
                        ventana.after(0, self.actualizar_interfaz)
            except Exception as e:
                print(f"Error al leer datos desde el puerto serie: {e}")
                break

```

La función **leer_datos_continuamente** es un bucle infinito que intenta leer continuamente datos desde el puerto serie. Aquí hay un resumen de lo que hace:

Verifica si el puerto serie (**self.serial_port**) está abierto.

Lee los datos desde el puerto serie y los decodifica.

Si hay datos leídos:

- Agrega los datos al búfer de respuesta (**self.response_buffer**).
- Busca las cadenas "**Done.**", "**ok**" y "**OK**" en el búfer de respuesta.

Si encuentra una de las cadenas mencionadas:

- Inserta un mensaje correspondiente en la caja de texto **TextInterpretacion**.
- Si es "**Done.**", también agrega el búfer al final de la caja de texto **TextRecibidos** con un prefijo ">".
- Limpia el búfer de respuesta.

Si no se encuentra ninguna cadena mencionada:

- Utiliza el método **after** de Tkinter para programar la ejecución de la función **actualizar_interfaz** en el hilo principal, evitando problemas de concurrencia.

Captura cualquier excepción que pueda ocurrir al leer datos desde el puerto serie y muestra un mensaje de error en la consola.

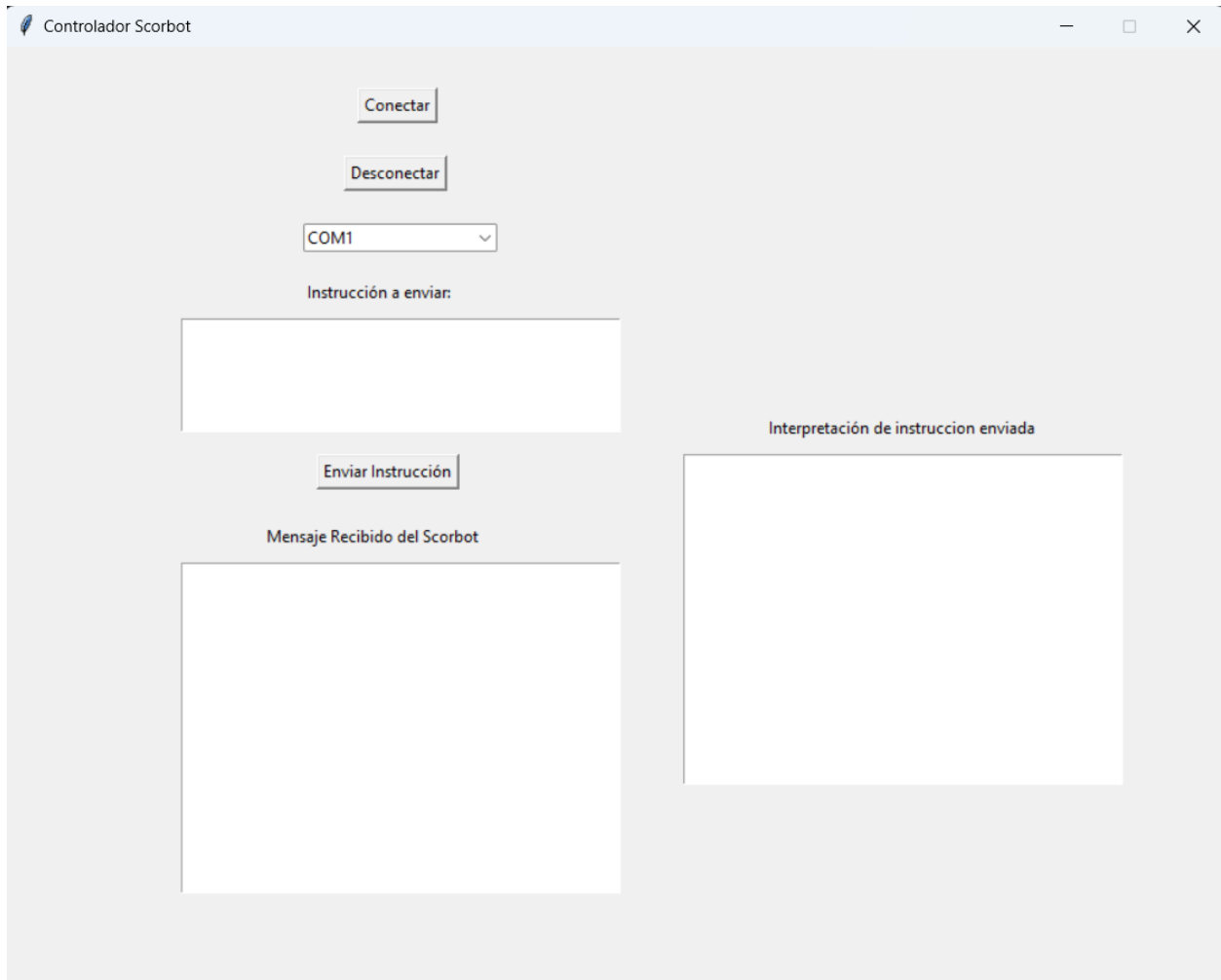
5) actualizar_interfaz(self):

```
def actualizar_interfaz(self):
    global TextInterpretacion
    # Obtener el palette y la estación del buffer
    print(self.response_buffer)
    estacion = self.response_buffer[7:9]
    palette = self.response_buffer[11:13]
    TextInterpretacion.insert(tk.END, f"El palette {palette} se encuentra en la estación {estacion}" + "\n")
    self.response_buffer = ""
```

La función **actualizar_interfaz** fue creada para abordar un problema de concurrencia al actualizar la interfaz gráfica (GUI) de Tkinter desde un hilo secundario. En el código proporcionado, el método **leer_datos_continuamente** se ejecuta en un hilo secundario para leer continuamente los datos desde un puerto serie. Dentro de este método, cuando se recibe un conjunto de datos, se intenta actualizar la interfaz gráfica mediante la inserción de un nuevo texto en la caja de texto **TextInterpretacion**.

Sin embargo, las operaciones de interfaz gráfica deben realizarse en el hilo principal de Tkinter, ya que Tkinter no es thread-safe. Para resolver este problema, se utiliza el método **after** de Tkinter, que programa la ejecución de una función en el hilo principal después de un cierto tiempo. La función **actualizar_interfaz** se llama desde **leer_datos_continuamente** mediante el método **after**, lo que garantiza que las operaciones de interfaz gráfica se realicen en el hilo principal.

Resultado final



Controlador Scrobot

Conectar

Desconectar

COM1

Instrucción a enviar:

Enviar Instrucción

Mensaje Recibido del Scrobot

Interpretación de instruccion enviada

Tenemos 3 cajas de texto donde podemos enviar instrucciones, ver el mensaje recibido por el robot que se parecerá a esto:

```
@00EX0002000100025C*  
@00EX0004000100025A*  
@00EX0005000100025B*  
@00EX0003000200025E*  
@00EX0006000200025B*  
@00EX0001000200025C*  
@00EX0002000200025F*
```

Recibimos estas señales del robot cada vez que un palette pasa por alguna estación, para poder entender este código que nos envía el robot, tenemos que notar que cambia estos valores:


```
@00EX000200000025C*
@00EX000400000025A*
@00EX000500000025B*
@00EX000300000025E*
@00EX000600000025B*
@00EX000100000025C*
@00EX000700000025F*
```

Estos valores indican la estación por la cual se encuentra pasando el palette, y los siguientes valores indican el palette que está pasando:

```
@00EX0002000100005C*
@00EX0004000100005A*
@00EX0005000100005B*
@00EX0003000200005E*
@00EX0006000200005B*
@00EX0001000200005C*
@00EX0002000200005F*
```

Por ejemplo si el código nos enviara esta señal de respuesta:

```
@00EX0002000100005C*
```

Lo interpretaremos como:

```
El palette {palette} se encuentra en la estación {estacion}" + "\n"
```

El palette 2 se encuentra en la estación 1

Para lograr que el programa interpreta el código recibido en una caja de texto que llamamos TextInterpretacion lo que se hizo fue recortar la parte del string que nos interesa:

```
print(self.response_buffer)
estacion = self.response_buffer[7:9]
palette = self.response_buffer[11:13]
TextInterpretacion.insert(tk.END, f"El palette {palette} se encuentra en la estación {estacion}" + "\n")
```