Dokuz Eylül University

Faculty Of Engineering

Electrical and Electronics Engineering Department

ETE3007

Fundamentals of Robotics

Smart Servo Motor Control by Python

Final Report

Student Name / ID:

GÜRKAN BIYIK (CENG) : 2020510019

GÜZİDE TOĞA (ELK) : 2018502074

HASAN FURKAN ÇOLAK (ELK) : 2018502090

Course Instructor:

ASSOCIATE PROFESSOR AHMET ÖZKURT

Contents

Abstract

The project "Smart Servo Motor Control Software Using Python" aims to develop an intelligent control system for servo motors, leveraging the versatility and simplicity of Python programming. This software is designed to provide precise and efficient control over servo motor movements, suitable for applications in robotics, automation, and mechatronics. Key features include real-time monitoring, adaptive control algorithms, and an intuitive user interface.

The system integrates various sensor inputs to dynamically adjust motor parameters, ensuring optimal performance and responsiveness. Through the implementation of advanced control strategies such as PID (Proportional-Integral-Derivative) control, the software achieves high precision in positioning and speed regulation. Additionally, it offers a flexible platform for customizations, allowing users to tailor the control logic to specific applications.

The project also emphasizes ease of use, with Python's extensive libraries and support for hardware interfacing facilitating rapid development and deployment. Comprehensive testing and validation are conducted to ensure reliability and robustness in diverse operating conditions. Ultimately, this project aims to provide a powerful tool for engineers and hobbyists alike, enhancing the capability and efficiency of servo motor-driven systems.

1.Introduction

This project focuses on creating a user-friendly and adaptable control platform that can be used in a wide range of applications. By incorporating real-time monitoring and adaptive control strategies, the software ensures that the servo motors operate at peak efficiency, maintaining high precision and reliability. The inclusion of features such as PID (Proportional-Integral-Derivative) control provides a robust foundation for achieving fine-tuned motor control, while the modular design of the software allows users to customize and extend its capabilities according to their specific needs.

Moreover, the project aims to make sophisticated servo motor control accessible to a broader audience, including engineers, researchers, and hobbyists. By utilizing Python, the software benefits from a large and active community, extensive documentation, and compatibility with a wide array of sensors and actuators. This facilitates the development process and enhances the software's potential for widespread adoption and continuous improvement.

In summary, the "Smart Servo Motor Control Software Using Python" project aspires to deliver a powerful and flexible tool for servo motor control, combining advanced control techniques with the ease of use and adaptability of Python. This introduction sets the stage for a detailed exploration of the system's design, features, and potential applications, highlighting its significance in the field of motor control technology.

2.User Manual

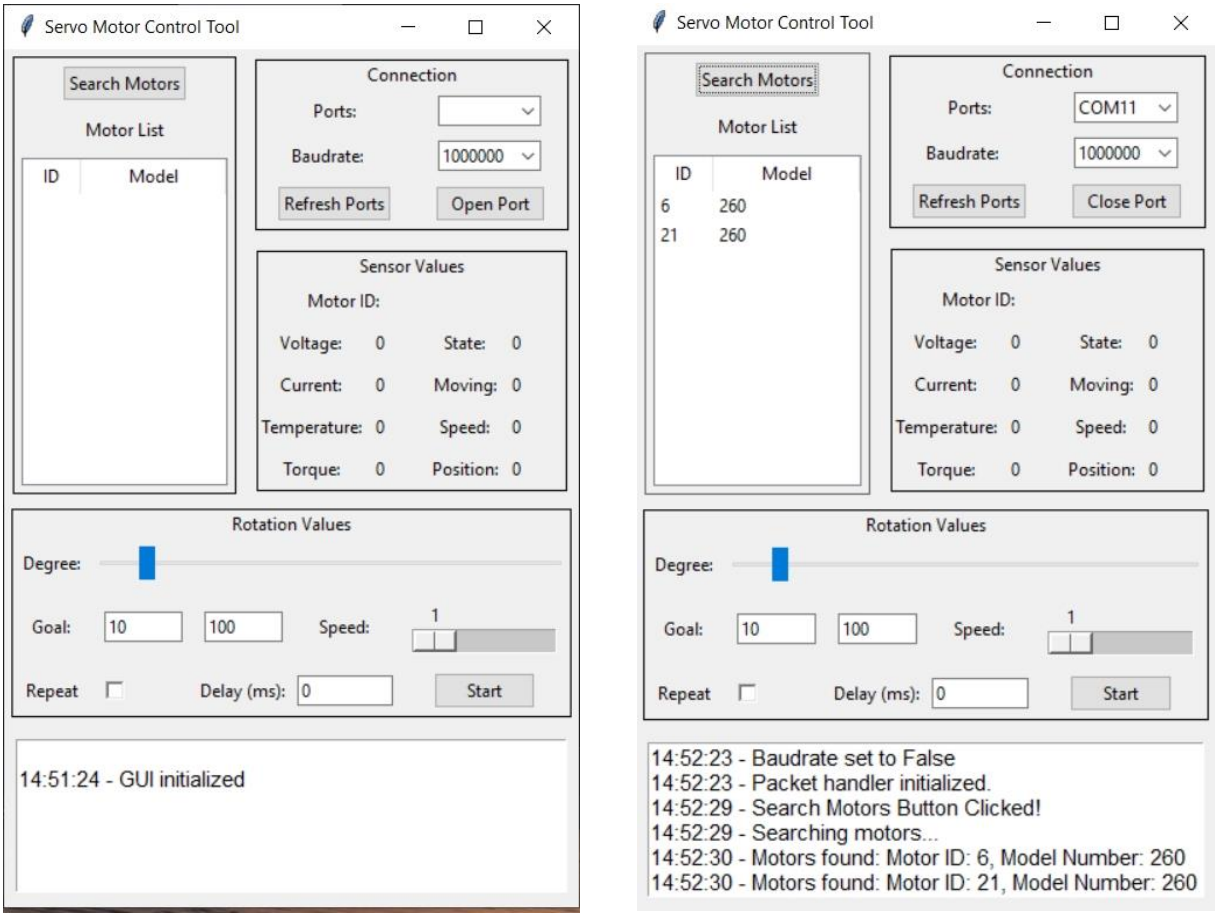


Figure 1

This manual provides step-by-step instructions for using the software to control and monitor motors via a serial port connection.

Initial Setup

1.Refresh the Ports:

- •Start by refreshing the available ports to ensure the software can detect your connected devices.
- •Click the "Refresh Ports" button.

2.Select Port and Baudrate:

- •From the dropdown menu, select the port you will use for communication.
- •Choose the appropriate baudrate from the available options.

3.Open the Port:

- •Click the "Open Port" button.
- •Check the logbox for the following messages:

•"Port opened successfully."

•"Packet handler initialized."

Searching for Motors

1.Search Motors:

•Click the "Search Motors" button.

•The logbox will display "Searching Motors...".

•If motors are found, you will see messages like:

•"Motor found: Motor ID: <x>, Motor Model: <y>"

•The discovered motors will be added to the Motor List widget.

•Wait for the logbox to display "Motor List Updated".

Controlling Motors

1.Select a Motor:

•Choose any motor from the Motor List widget.

2.View Sensor Values:

•Sensor values for the selected motor will be displayed in the interface.

3.Control Motor Rotation:

•Use the degree slider to rotate the motor to the desired angle.

4.Repeat Rotation:

•Adjust the start and end degrees using the provided fields following goal.

•Check the "Repeat" checkbox.

•Click the "Start" button to initiate continuous rotation between the start and end degrees.

•The motor will rotate from start to end and then back from end to start.

•To stop the rotation, click the "Start" button again.

5.Adjust Speed:

•Use the speed slider to set the desired rotation speed.

6.Set Initial Delay:

•Specify an initial delay in milliseconds (ms) if needed.

Log Messages

The logbox provides real-time feedback and updates on the software's actions. Regularly check the logbox to ensure the software is functioning as expected and to troubleshoot any issues that may arise.

Summary of Actions

- Refresh Ports: Updates the list of available ports.
- Select Port/Baudrate: Configures the communication settings.
- Open Port: Establishes a connection and initializes handlers.
- Search Motors: Detects and lists connected motors.
- Control Motor: Adjusts motor position, speed, and rotation parameters.
- Monitor Sensor Values: Displays real-time sensor data for the selected motor.

3.Smart Servo Motors

A smart servo motor, also known as an intelligent servo or integrated servo driver, is a type of servo motor that incorporates a built-in motor driver and control unit. This integration simplifies control and offers several advantages over traditional servo motors.

Here's a breakdown of the key features of a smart servo motor:

•Integrated Driver: Unlike a standard servo motor, which requires a separate motor driver for control, a smart servo motor has the driver built directly into the unit. This eliminates the need for additional external components, reducing complexity and simplifying wiring.

•Onboard Control: Smart servo motors have a microcontroller on board that allows for more advanced control capabilities. This can include features like:

-Precise position control

-Speed control

-Torque control

-Feedback mechanisms that allow the motor to monitor its performance and adjust accordingly

•Daisy-Chaining: Some smart servo motors support daisy-chaining, which enables you to connect multiple servos together using a single communication line. This can significantly reduce wiring complexity, especially in applications with many servos.

•Communication Protocols: Smart servo motors can communicate with control systems using various communication protocols, such as RS-485 or CAN bus. This allows for more sophisticated control and easier integration into complex projects.

In essence, smart servo motors offer a more integrated and user-friendly solution for applications requiring precise control of motion. They are ideal for robotics, automation, animatronics, and other projects that demand high-performance motion control.

4.Hardware

4.1. SCS 15 Smart Servo Motor

The SCS15 is a smart servo motor made by Feetech. It is a high-torque servo that provides 15kg of torque at 6 volts. It is also serially controlled, which means that it can receive commands and send feedback data over a serial communication line. This allows for more precise control of the servo motor than is possible with traditional pulse width modulation (PWM) control.

Here are some of the key features of the SCS15 smart servo motor:

- High torque: 15kg at 6 volts
- Serial control: For more precise control
- Onboard control unit: Allows for more advanced control capabilities
- Daisy-chaining: Allows you to connect multiple servos together using a single communication line
- Communication protocols: RS-485 or CAN bus

Some things to keep in mind when using the SCS15 include:

- It requires a separate programmer to configure it for your specific application.
- It communicates using a special half-duplex UART bus, so you will need a compatible controller to communicate with it.
- It is more expensive than a traditional servo motor.

Overall, the SCS15 is a powerful and versatile smart servo motor that a valuable asset fort his Project.

Figure 2

4.2. The SparkFun FTDI Basic

The SparkFun FTDI Basic Breakout - 3.3V is a handy tool for tinkerers and developers working with electronics, particularly those that operate at 3.3 volts (3.3V). It acts as an adapter, allowing your computer to talk to these electronic devices using a serial connection.

Here's a closer look at what the SparkFun FTDI Basic Breakout - 3.3V offers:

- FTDI Chip: The heart of this breakout board is the FTDI FT232RL chip. This tiny chip acts as an interpreter, translating signals between the Universal Serial Bus (USB) port on your computer and the serial protocol that many electronic devices use.

- 3.3V Power Supply: This version of the breakout board is specifically designed for 3.3V logic level devices. It provides a stable 3.3V power output, making it safe to connect and communicate with these devices without the risk of damage from higher voltage levels.

- FTDI Cable Compatibility: The pin layout (the arrangement of connectors) on this board matches that of a standard FTDI cable. This compatibility allows you to use it with software designed for FTDI cables, saving you time and effort setting things up. Additionally, the pinout matches many 3.3V Arduino boards, making it perfect for programming and debugging them.

- Beyond Arduino: While commonly used with Arduino boards, the FTDI Basic Breakout - 3.3V is a versatile tool. Its ability to handle serial communication makes it suitable for various applications. You can use it with any device that uses serial communication for programming, transferring data, or controlling it.

- Compact Size: This breakout board is small and lightweight, making it ideal for use on breadboards where space might be limited. It also makes it easy to carry around for working on projects on the go.

Overall, the SparkFun FTDI Basic Breakout - 3.3V is a valuable tool for this project, especially those that operate at 3.3V logic levels. It simplifies communication and programming tasks.

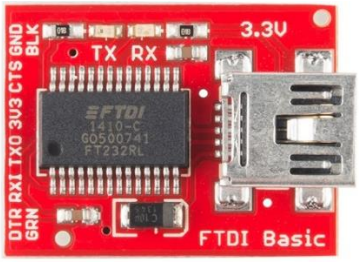


Figure 3

4.3. TTLinker Mini

The TTLinker Mini, made by Feetech, is a small circuit board that acts as a bridge between two electronic components:

- Arduino microcontrollers: These are popular single-board computers used in electronics projects for various purposes.
- Feetech SCS series servos: These are special types of servos with integrated control circuits that offer more advanced features than traditional servos.

Here's how the TTLinker Mini helps them work together:

- Signal Conversion: Standard Arduinos communicate using UART signals, while SCS servos use half-duplex UART signals. The TTLinker Mini translates between these two signal formats, allowing them to understand each other.

- Simplified Connection: Without the TTLinker Mini, you'd need to manage the signal conversion yourself, which can be complex. The TTLinker Mini takes care of this, making the connection between your Arduino and SCS servos much simpler. It essentially reduces the number of wires you need to manage.
- Optional Sensor Support: Some versions of the TTLinker Mini may have additional pins for connecting sensors to your project. This can be useful if your project requires the servo to react to sensor data, such as a robotic arm using a sensor to detect objects.

Figure 4

In essence, the TTLinker Mini acts as an adapter that enables smooth communication between Arduino and SCS servos.

Here are some additional points to consider:

- Compatibility: The TTLinker Mini is specifically designed for Feetech SCS series servos and might not work with other servo brands.

- Software Support: You might need an Arduino library to control the servos using the TTLinker Mini. This library provides functions and code to simplify communication between the Arduino and the servos.
- Power: The TTLinker Mini itself typically draws power from the Arduino it's connected to.

Overall, the TTLinker Mini offers a convenient way to connect and control Feetech SCS servos from Arduino. While it adds an extra component, it simplifies wiring and potentially unlocks more advanced control features for servos.

4.4 Samsung SB-LSM80

The Samsung SB-LSM80 is a **lithium-ion rechargeable battery** specifically designed for use with Samsung digital cameras. It provides power to the camera, allowing it to operate its various functions. The battery is suitable fort his project.

Here are some of the key features of the Samsung SB-LSM80 battery:

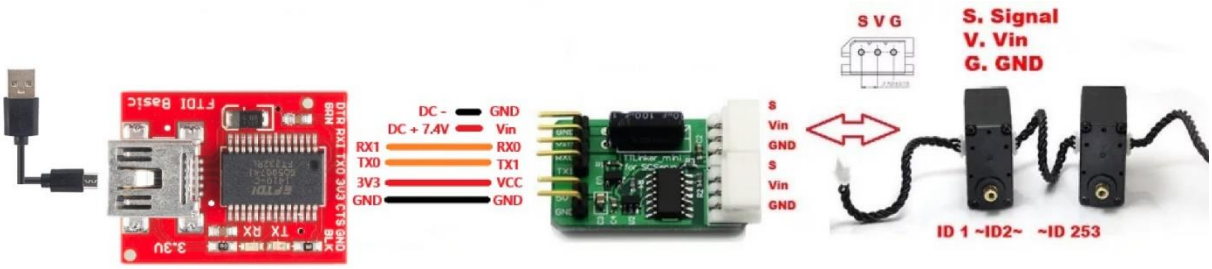
- **Battery Type:** Lithium-ion (Li-ion)
- **Capacity:** 800mAh
- **Voltage:** 7.4V
- **Chemistry:** Lithium Cobalt Oxide (LiCoO2)
- **Rechargeable:** Yes

Here are some additional things to keep in mind about the Samsung SB-LSM80 battery:

- **Safety:** Always handle lithium-ion batteries with care. Avoid puncturing, crushing, or exposing them to extreme temperatures.
- **Charging:** Use the appropriate charger for lithium-ion batteries to avoid overcharging or damage to the battery.
- **Disposal:** Dispose of used lithium-ion batteries responsibly according to local regulations.

Figure 5

4.5.Schema



5.Software

5.1.Installation Instructions

- Check Python Version:
  - Ensure you have Python 3.9.13 installed. Open a command prompt and type:
  - python --version
  - It should return Python 3.9.13.

- Install Python:
  - If you don't have Python installed:
    - You can install the Python Windows app from the Microsoft Store. Or, download Python 3.9.13 from the official Python website.
    - During installation, make sure to check the box that says "Add Python to PATH".

- Download the Source:
  - Download the source code from the provided link. Unzip the contents to a folder of your choice.

- Open Command Prompt in the Folder:
  - Navigate to the folder where you unzipped the contents. Press Shift and right-click in the folder, then select "Open command window here" or "Open PowerShell window here".

- Create a Virtual Environment:
  - In the command prompt, create a virtual environment with Python 3.9.13:
  - python -m venv venv

- Activate the Virtual Environment:
  - On Windows Command Prompt:
    - venv\Scripts\activate
  - On Windows PowerShell:

- .\venv\Scripts\Activate.ps1

❖ Install Required Packages:
  - ➢ Install the required packages using pip:
    - pip install -r requirements.txt

❖ Move scservo_sdk to the Site-Packages Folder:
  - ➢ Move the scservo_sdk folder to the venv/Lib/site-packages directory.
  - ➢ You can do this manually or using the following command in the command prompt:
    - move scservo_sdk venv\Lib\site-packages

❖ Run the Program:
  - ➢ You can start the program directly from the run.bat file by double-clicking it.
  - ➢ Alternatively, you can open your preferred IDE, load the project, and execute the run script from there. Make sure you run the code with virtual environment

❖ Summary of Commands
  - ➢ Here is a summary of the commands you will use:
    - python --version
    - python -m venv venv
    - venv\Scripts\activate # or .\venv\Scripts\Activate.ps1 for PowerShell
    - pip install -r requirements.txt
    - move scservo_sdk venv\Lib\site-packages
  - ➢ After following these steps, your environment should be set up correctly and you can run the program as needed.

5.2.Code Inspection

a) Imports and Dependencies
   a. The script imports several standard libraries such as tkinter, threading, time, and tkinter.ttk.
   b. External dependencies include:
      i. scservo_sdk for servo motor communication.

ii. serial.tools.list_ports for listing available serial ports.

```
def refresh_ports(self):
    ports = [port.device for port in list_ports.comports()]
    self.portList["values"] = sorted(ports)
    self.write_log("Ports refreshed.")
    self.portList.current(0)
    self.MotorList.delete(*self.MotorList.get_children())
```

b) Class Structure

a. The ServoControl class inherits from tk.Tk, indicating it is a main application window.
b. The constructor (__init__) initializes the GUI and class variables.

```
class ServoControl(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("Servo Motor Control Tool")
        self.MINIMUM_POSITION_VALUE  = 10
        self.MAXIMUM_POSITION_VALUE  = 1000
        self.MOVING_PART             = 2400 // 5
        self.geometry("+400+100")  # set the position of the window
        self.logBox: tk.Text = None
        self.portList: ttk.Combobox = None
        self.baudrateList: ttk.Combobox = None
        self.portHandler: PortHandler = None
        self.packet_handler: scscl = None
        self.sensor_values_values = {}
        self.MotorList: ttk.Treeview = ttk.Treeview()
        self.DegreeSlider = None
        self.DelayEntry = None
        self.degree_var = tk.IntVar()
        self.speed_var = tk.IntVar()
        self.minsize(300, 300)
        self.initialize_gui()  # initialize the GUI
```

c) GUI Initalization

a. The method initialize_gui sets up the various frames, labels, buttons, sliders, and other widgets in the main window.
b. The layout uses the grid system to position elements.

d) Key Widgets and Controls

a. Port List and Baudrate List: Dropdown menus for selecting the port and baudrate.

```
portsLabel = ttk.Label(frame1, text="Ports:")
portsLabel.grid(row=1, column=0, padx=5, pady=5)

self.portList = ttk.Combobox(frame1, state="readonly", width=8)
self.portList.grid(row=1, column=1, padx=5, pady=5)

refresh_button = ttk.Button(
    frame1,
    text="Refresh Ports",
    command=lambda: (
        self.write_log("Refresh Button Clicked!"),
        self.refresh_ports(),
    ),
)
refresh_button.grid(row=3, column=0, padx=5, pady=5)
```

5.2.Code Inspection1

a) **Imports and Dependencies**

a. The script imports several standard libraries such as **tkinter**, **threading**, **time**, and **tkinter.ttk**.

b. External dependencies include:

i. **scservo_sdk** for servo motor communication.

ii. **serial.tools.list_ports** for listing available serial ports.

```
def refresh_ports(self):
    ports = [port.device for port in list_ports.comports()]
    self.portList["values"] = sorted(ports)
    self.write_log("Ports refreshed.")
    self.portList.current(0)
    self.MotorList.delete(*self.MotorList.get_children())
```

b) **Class Structure**

a. The **ServoControl** class inherits from **tk.Tk**, indicating it is a main application window.

b. The constructor (**__init__**) initializes the GUI and class variables.

```python
class ServoControl(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("Servo Motor Control Tool")
        self.MINIMUM_POSITION_VALUE  = 10
        self.MAXIMUM_POSITION_VALUE  = 1000
        self.MOVING_PART             = 2400 // 5
        self.geometry("+400+100")  # set the position of the window
        self.logBox: tk.Text = None
        self.portList: ttk.Combobox = None
        self.baudrateList: ttk.Combobox = None
        self.portHandler: PortHandler = None
        self.packet_handler: scscl = None
        self.sensor_values_values = {}
        self.MotorList: ttk.Treeview = ttk.Treeview()
        self.DegreeSlider = None
        self.DelayEntry = None
        self.degree_var = tk.IntVar()
        self.speed_var = tk.IntVar()
        self.minsize(300, 300)
        self.initialize_gui()  # initialize the GUI
```

c) **GUI Initialization**

a. The method **initialize_gui** sets up the various frames, labels, buttons, sliders, and other widgets in the main window.

b. The layout uses the grid system to position elements.

d) **Key Widgets and Controls**

a. **Port List and Baudrate List**: Dropdown menus for selecting the port and baudrate.

```
portsLabel = ttk.Label(frame1, text="Ports:")
portsLabel.grid(row=1, column=0, padx=5, pady=5)

self.portList = ttk.Combobox(frame1, state="readonly", width=8)
self.portList.grid(row=1, column=1, padx=5, pady=5)

refresh_button = ttk.Button(
    frame1,
    text="Refresh Ports",
    command=lambda: (
        self.write_log("Refresh Button Clicked!"),
        self.refresh_ports(),
    ),
)
refresh_button.grid(row=3, column=0, padx=5, pady=5)
```

```
baudrateLabel = ttk.Label(frame1, text="Baudrate:", width=10)
baudrateLabel.grid(row=2, column=0, padx=5, pady=5)

self.baudrateList = ttk.Combobox(frame1, state="readonly", width=8)
self.baudrateList["values"] = (
    1000000,
    500000,
    250000,
    128000,
    115200,
    76800,
    57600,
    38400,
)
self.baudrateList.current(0)
self.baudrateList.grid(row=2, column=1, padx=5, pady=5)

self.baudrateList.bind(
    "<<ComboboxSelected>>",
    lambda e: self.write_log(f"Baudrate selected: {self.baudrateList.get()}"),
)

OpenPortBtn = ttk.Button(
    frame1,
    text="Open Port",
)
OpenPortBtn.grid(row=3, column=1, padx=5, pady=5)
```

Motor List: Treeview widget to display discovered motors.

```
SearchMotorBtn = ttk.Button(
    frame0,
    text="Search Motors",
    command=lambda: (
        self.write_log("Search Motors Button Clicked!"),
        Thread(target=self.search_motors).start(),
    ),
)
SearchMotorBtn.pack(padx=5, pady=5)

MotorListLabel = ttk.Label(frame0, text="Motor List")
MotorListLabel.pack(padx=5, pady=5)

MotorLst = ttk.Treeview(frame0, columns=("ID", "Model"), show="headings", selectmode="browse")
MotorLst.heading("ID", text="ID")
MotorLst.heading("Model", text="Model")
MotorLst.column("ID", width=40)
MotorLst.column("Model", width=100)
MotorLst.pack(padx=5, pady=5)
self.MotorList = MotorLst
```

- **Degree Slider**: Slider to set the target degree for motor rotation.

```
DegreeLabel = ttk.Label(frame3, text="Degree:")
DegreeLabel.grid(row=1, column=0, padx=5, pady=5)

DegreeSlider = ttk.Scale(
    frame3,
    from_= self.MINIMUM_POSITION_VALUE,
    to= self.MAXIMUM_POSITION_VALUE,
    orient=tk.HORIZONTAL,
    length=320,
)
DegreeSlider.set(100)
self.degree_var.set(100)
DegreeSlider.bind(
    "<ButtonRelease-1>",
    lambda e: (    "e" is not accessed
        DegreeSlider.set(int(DegreeSlider.get())),
        self.degree_var.set(int(DegreeSlider.get())),
        self.rotate_degree(DegreeSlider.get()),
        self.write_log(f"Degree set to {DegreeSlider.get():.0f}"),
    ),
)
DegreeSlider.grid(row=1, column=1, padx=5, pady=5, columnspan=4)
self.DegreeSlider = DegreeSlider
```

- **Speed Slider**: Slider to adjust the rotation speed.

```
SpeedLabel = ttk.Label(frame3, text="Speed:")
SpeedLabel.grid(row=3, column=3, pady=3)

SpeedSlider = tk.Scale(
    frame3,
    from_=1,
    to=5,
    orient=tk.HORIZONTAL,
    length=100,
    variable=self.speed_var,
    command=lambda e: self.write_log(f"Speed set to {SpeedSlider.get()}"),
)
SpeedSlider.grid(row=3, column=4, padx=5, pady=5)
```

- **Delay Entry**: Entry box to set an initial delay.

```
DelayLabel = ttk.Label(frame3, text="Delay (ms):")
DelayEntry = ttk.Entry(frame3, width=10)
DelayLabel.grid(row=4, column=2, pady=3)
DelayEntry.grid(row=4, column=3, padx=5, pady=5)
self.DelayEntry = DelayEntry
self.DelayEntry.insert(0, 0)

def delay_changed(event):    "event" is not accessed
    try:
        value = max(0, int(DelayEntry.get()))
        DelayEntry.delete(0, tk.END)
        DelayEntry.insert(0, value)
    except ValueError:
        self.write_log("Invalid value entered for Delay Entry.")
        DelayEntry.delete(0, tk.END)
        return

DelayEntry.bind("<KeyRelease>", delay_changed)
```

- **Repeat Checkbox**: Checkbox to enable continuous rotation between start and end degrees.

- **Log Box**: Text widget to display log messages.

```
RepeatLabel = ttk.Label(frame3, text="Repeat", anchor="e")
CheckBoxState = tk.BooleanVar()
RepeatCheckBox = tk.Checkbutton(
    frame3, anchor="w", variable=CheckBoxState, width=5
)
RepeatCheckBox.bind(
    "<Button-1>",
    lambda e: self.write_log(    "e" is not accessed
        f"Repeat checkbox {'checked' if CheckBoxState.get() else 'unchecked'}"
    ),
)
RepeatLabel.grid(row=4, column=0, pady=3)
RepeatCheckBox.grid(row=4, column=1, padx=5, pady=5)
```

```
self.logBox = tk.Text(        You, 6 days ago • last
    self, width=47, height=6, fg="black", font=("sans-serif", 11)
)
self.logBox.grid(row=3, column=0, columnspan=2, padx=3, pady=10)
```

5. Functions and Methods

- **initialize_gui**: Sets up the GUI elements.
- **update_sensor_values**: Updates the sensor values displayed for the selected motor.

```
def update_sensor_values(self):
        return

    selected_items = self.MotorList.selection()
    current_motor_id = self.MotorList.item(selected_items, "values")[0]

    mtr_id = int(current_motor_id)
    flag = True
    while selected_items:
        scs_present_position, scs_present_speed, scs_comm_result, scs_error = self.packet_handler.ReadPosSpeed(mtr_id)
        moving_value, moving_result, moving_error = self.packet_handler.ReadMoving(mtr_id)     "moving_result" is not a
        voltage_value, temp_result, temp_error = self.packet_handler.read1ByteTxRx(mtr_id, SCSCL_PRESENT_VOLTAGE)
        x1_value, temp_result, temp_error = self.packet_handler.read1ByteTxRx(mtr_id, SCSCL_PRESENT_CURRENT_H)
        x2_value, temp_result, temp_error = self.packet_handler.read1ByteTxRx(mtr_id, SCSCL_PRESENT_CURRENT_L)
        temp_value, temp_result, temp_error = self.packet_handler.read1ByteTxRx(mtr_id, SCSCL_PRESENT_TEMPERATURE)
        torque_value, temp_result, temp_error = self.packet_handler.read1ByteTxRx(mtr_id, SCSCL_TORQUE_ENABLE)
        self.sensor_values_values["Voltage"].config(text=f'{voltage_value//10}.{voltage_value%10} V')
        self.sensor_values_values["Current"].config(text=f"{x1_value}.{x2_value}")
        self.sensor_values_values["Temperature"].config(text=temp_value)
        self.sensor_values_values["Torque"].config(text=torque_value)
        self.sensor_values_values["State"].config(text=scs_comm_result)
        self.sensor_values_values["Moving"].config(text=moving_value)
        self.sensor_values_values["Speed"].config(text=scs_present_speed)
        self.sensor_values_values["Position"].config(text=scs_present_position)
        if flag:
            self.degree_var.set(scs_present_position)
            self.DegreeSlider.set(scs_present_position)
            flag = False
```

- **write_log**: Writes messages to the log box.

```
def write_log(self, message: str):
    msg = f'{strftime("%H:%M:%S")} - {message}'
    self.logBox.insert(tk.END, "\n" + msg)
    self.logBox.see(tk.END)
```

- **refresh_ports**: Refreshes the list of available serial ports.
- **open_port**: Opens the selected serial port and initializes the packet handler.
- **search_motors**: Searches for connected motors and updates the motor list.

```
def search_motors(self):
    if self.portHandler is None:
        self.write_log("Port handler is not initialized.")
        self.write_log("Please open a port first.")
        return

    if self.packet_handler is None:
        self.write_log("Packet handler is not initialized.")
        return

    self.write_log("Searching motors...")
    ids = (motor_id for motor_id in range(MAX_ID + 1))
    is_motor_found = False
    for motor_id in ids:
        scs_model_number, scs_comm_result, scs_error = self.packet_handler.ping(    "scs_error" is not
            motor_id
        )
        if scs_comm_result == COMM_SUCCESS:
            is_motor_found = True
            str_motor_list = f"Motor ID: {motor_id}, Model Number: {scs_model_number}"
            self.write_log("Motors found: " + str_motor_list)
            self.MotorList.insert("", 'end', text=str_motor_list, values=(motor_id, scs_model_number))

    if is_motor_found:
        self.write_log("Motor list updated.")
    else:
        self.write_log("No motors found.")
```

- **rotate_degree**: Rotates the selected motor to the specified degree.

6. Threading

- The use of **threading.Thread** ensures that long-running tasks like searching for motors and updating sensor values do not block the main GUI thread.

```python
def rotate_degree(self, degree: int):
    if not all((self.portHandler, self.packet_handler, self.MotorList.selection())):
        return
    try:
        dly = int(self.DelayEntry.get()) / 1000
    except ValueError:
        self.write_log("Invalid value entered for Delay Entry.")
        return

    try:
        if degree < self.MINIMUM_POSITION_VALUE or degree > self.MAXIMUM_POSITION_VALUE:
            raise ValueError
    except ValueError:
        self.write_log("Invalid value entered for Degree.")
        return


    motor_id = self.MotorList.item(self.MotorList.selection(), "values")[0]
    motor_id = int(motor_id)
    speed = int(self.speed_var.get()) * self.MOVING_PART
    degree = int(degree)
    sleep(dly)
    sleep(0.2)
    self.packet_handler.WritePos(motor_id, degree, 0, speed)
    while self.packet_handler.ReadMoving(motor_id)[0]:
        continue
    print("Rotating motor to degree:", degree)
```

```python
SearchMotorBtn = ttk.Button(
    frame0,
    text="Search Motors",
    command=lambda: (
        self.write_log("Search Motors Button Clicked!"),
        Thread(target=self.search_motors).start(),
    ),
)
SearchMotorBtn.pack(padx=5, pady=5)
```

```python
self.MotorList.bind("<ButtonRelease-1>", lambda e: Thread(target=self.update_sensor_values).start())
```

```python
t = Thread(target=start)

def start_btn_handler():
    text = startBtn.cget("text")
    self.write_log(f"{text} button clicked.")
    if text == "Start":
        t = Thread(target=start)
        t.start()
    if text == "Stop":
        CheckBoxState.set(False)

startBtn.bind("<Button-1>", lambda e: Thread(target=start_btn_handler).start())
```

To reach the Thread

if The Button is Start, start function start with Thread

7. Event Handling

- Several event bindings are used:
    - Buttons for starting actions like opening ports and searching for motors.
    - Combobox for selecting baudrate.
    - Sliders for setting degrees and speed.

```python
try:
    startDegree = int(StartEntry.get())
    endDegree = int(EndEntry.get())
    if startDegree < self.MINIMUM_POSITION_VALUE or startDegree > self.MAXIMUM_POSITION_VALUE:
        raise ValueError
    if endDegree < self.MINIMUM_POSITION_VALUE or endDegree > self.MAXIMUM_POSITION_VALUE:
        raise ValueError
except Exception as e:
    self.write_log("Invalid value entered for Start or End Position.")
    return False
```

- Checkboxes and entries for additional control parameters.

```python
def rotate_degree(self, degree: int):
    if not all((self.portHandler, self.packet_handler, self.MotorList.selection())):
        return
    try:
        dly = int(self.DelayEntry.get()) / 1000
    except ValueError:
        self.write_log("Invalid value entered for Delay Entry.")
        return

    try:
        if degree < self.MINIMUM_POSITION_VALUE or degree > self.MAXIMUM_POSITION_VALUE:
            raise ValueError
    except ValueError:
        self.write_log("Invalid value entered for Degree.")
        return
```

```
def delay_changed(event):    "event" is not accessed      You, 3 weeks
    try:
        value = max(0, int(DelayEntry.get()))
        DelayEntry.delete(0, tk.END)
        DelayEntry.insert(0, value)
    except ValueError:
        self.write_log("Invalid value entered for Delay Entry.")
        DelayEntry.delete(0, tk.END)
        return
```

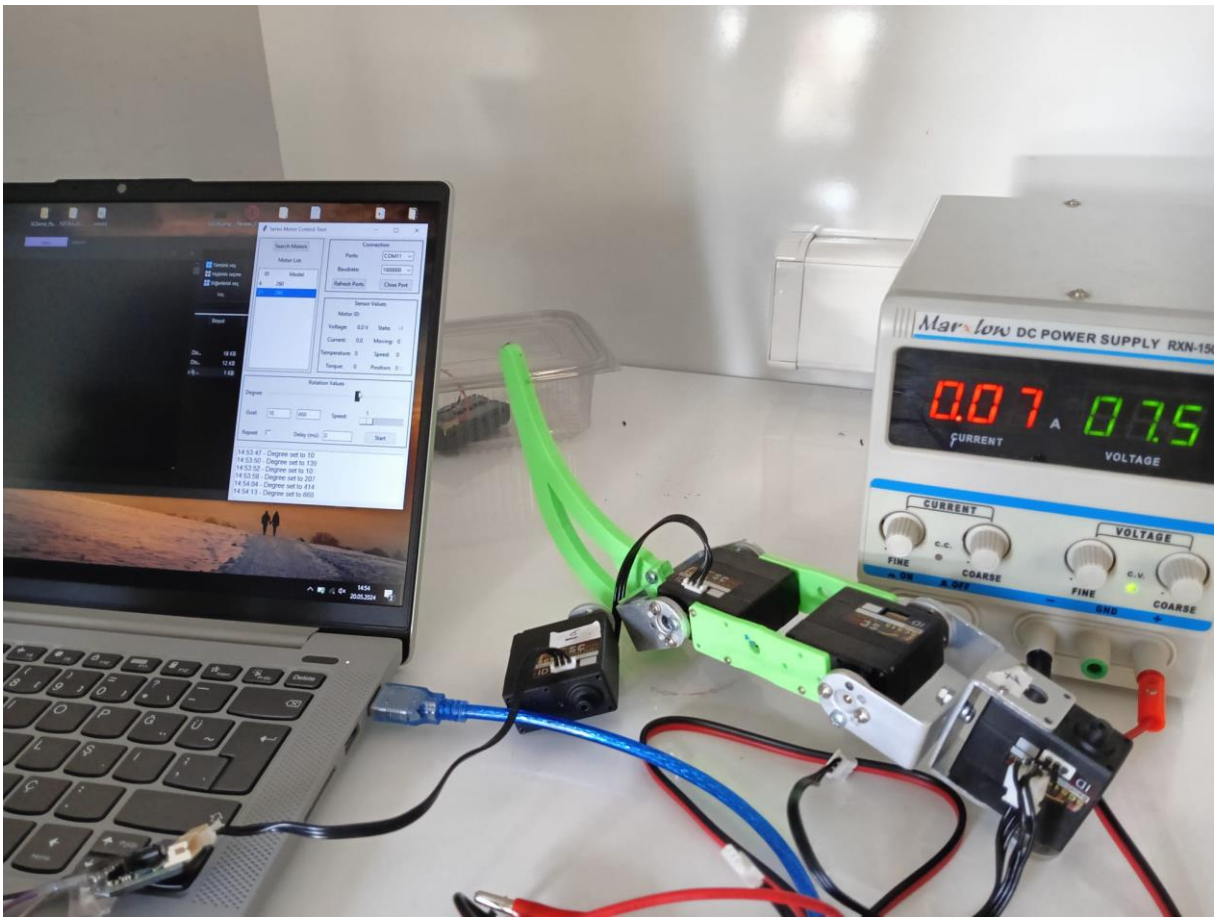
8. Logging and Feedback

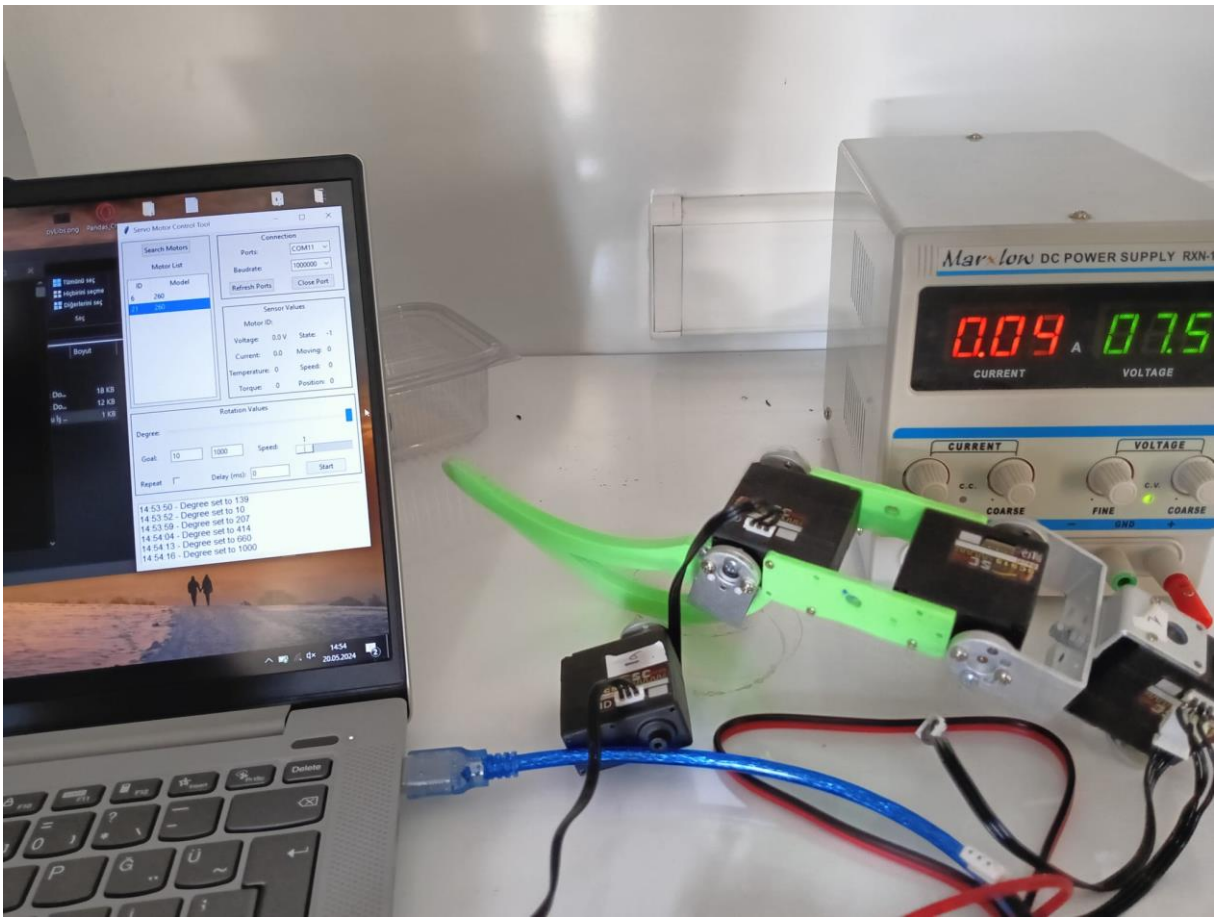
- The **write_log** method provides real-time feedback to the user by logging messages to a text box in the GUI.

6.Project Demo


Servo Motor Control Tool
Search Motors
Motor List
Connection
Ports: COM11
Baudrate: 1000000
Refresh Ports
Close Port
Sensor Values
Motor ID:
Voltage: 0.0 V
State: -1
Current: 0.0
Moving: 0
Temperature: 0
Speed: 0
Torque: 0
Position: 0
Rotation Values
Degree:
Goal: 10
207
Speed: 1
Repeat
Delay (ms): 0
Start
14:52:43 - Motor list updated.
14:53:45 - Degree set to 20
14:53:47 - Degree set to 10
14:53:50 - Degree set to 139
14:53:52 - Degree set to 10
14:53:59 - Degree set to 207
14:53
20.05.2024
DC POWER SUPPLY
CURRENT
VOLTAGE
CURRENT
VOLTAGE
C.C.
C.V.
ON
OFF
GND


Servo Motor Control Tool
Search Motors
Motor List
Connection
Ports:
COM11
Baudrate:
1000000
Refresh Ports
Close Port
Sensor Values
Motor ID:
Voltage: 0.0 V
State: -1
Current: 0.0
Moving: 0
Temperature: 0
Speed: 0
Torque: 0
Position: 0
Rotation Values
Degree:
Goal:
Speed:
Repeat
Delay (ms):
Start
14:53:45 - Degree set to 20
14:53:47 - Degree set to 10
14:53:50 - Degree set to 139
14:53:52 - Degree set to 10
14:53:59 - Degree set to 207
14:54:04 - Degree set to 414
Marlow DC POWER SUPPLY
CURRENT
VOLTAGE
ON
OFF
GND


DC POWER SUPPLY
0.07 A
CURRENT
07.5
VOLTAGE
CURRENT
VOLTAGE
FINE
COARSE
FINE
COARSE


Marlow DC POWER SUPPLY
0.09 A
CURRENT
07.5
VOLTAGE
CURRENT
VOLTAGE
COARSE
FINE
COARSE
Servo Motor Control Tool
Search Motors
Motor List
Connection
Ports:
COM11
Baudrate:
1000000
Refresh Ports
Close Port
Sensor Values
Rotation Values
Degree:
Goal:
Speed:
Repeat
Delay (ms):
Start
14:53:50 - Degree set to 139
14:53:52 - Degree set to 10
14:53:59 - Degree set to 207
14:54:04 - Degree set to 414
14:54:13 - Degree set to 660
14:54:16 - Degree set to 1000

7.Problems and Optimization

•Error Handling: While some error handling is present, it could be enhanced, especially around critical operations like port opening and motor commands.

•Thread Safety: Ensure all UI updates from threads are thread-safe by using mechanisms like tkinter's after method.

•Code Modularity: Consider breaking down large methods like initialize_gui into smaller methods for better readability and maintenance.

8.References

- "SCServo_Python_220415.7Z · FTSERVO/飞特总线舵机SDK_FEETECH Bus Servo SDK," Gitee, https://gitee.com/ftservo/SCServoSDK/blob/master/SCServo_Python_220415.7z (accessed May 23, 2024).
- "SCServo_Python_220415.7Z · FTSERVO/飞特总线舵机SDK_FEETECH Bus Servo SDK," Gitee, https://gitee.com/ftservo/SCServoSDK/blob/master/SCServo_Python_220415.7z (accessed May 23, 2024).
- "FTSERVO/飞特总线舵机SDK_FEETECH Bus Servo SDK," Gitee, https://gitee.com/ftservo/SCServoSDK (accessed May 23, 2024).
- "Graphical user interfaces with TK," Python documentation, https://docs.python.org/3/library/tk.html (accessed May 23, 2024).