Project Overview - Raspberry Pi & Arduino Serial Communication

This document provides a high-level overview of the educational serial communication projects in this repository.

Table of Contents

- arduino_serial/
- Servo_Control/
- Quick Comparison
- Getting Started

---

arduino_serial/ - Basic Serial Communication

Purpose: Introductory project demonstrating fundamental one-way serial communication between Raspberry Pi and Arduino.

What to Expect

This is a **beginner-friendly** starting point for learning serial communication basics. The project sends a simple text message from the Raspberry Pi to the Arduino and receives a confirmation response.

Key Features

- **Simple one-way communication**: Raspberry Pi sends "Hello from Raspberry Pi!" to Arduino
- **Two Arduino sketch variants**:
  - `arduino_receiver/`: Basic receiver that displays messages in Serial Monitor
  - `arduino_receiver_with_led/`: Receiver with visual feedback (LED flashes 10 times on message receipt)
- **Automatic error handling**: Python script includes built-in troubleshooting messages
- **Minimal hardware**: Only requires Raspberry Pi, Arduino Uno, and USB cable

Technical Details

- **Python Script**: `test_serial.py` - Single message sender with response reading
- **Protocol**: Simple string transmission with newline delimiter (`\n`)

- **Baud Rate**: 9600
- **Port**: `/dev/ttyUSB0` (configurable)
- **Communication Flow**: Raspberry Pi → Arduino (one message, one response)

What You'll Learn

- Opening and configuring serial connections in Python
- Encoding/decoding strings to bytes for serial transmission
- Critical timing considerations (2-second delay for Arduino reset)
- Reading responses from Arduino
- Basic error handling for serial communication

Expected Output

```
Connecting to Arduino...
Sending: Hello from Raspberry Pi!

Waiting for Arduino response...
Arduino says: Received: Hello from Raspberry Pi!

Communication complete!
```

Visual Feedback (LED version): Arduino's built-in LED flashes 10 times when message is received.

Ideal For

- First-time serial communication users
- Understanding the basics of PySerial library
- Learning Arduino serial input/output
- Troubleshooting serial port connections

Servo_Control/ - Servo Motor Control

Purpose: Advanced project demonstrating command-based serial communication for hardware control using an SG90 servo motor.

What to Expect

This project introduces **structured command protocols** and **interactive control** of physical hardware. You'll control a servo motor's position (0-180 degrees) from the Raspberry Pi through an intuitive menu system.

Key Features

- **Interactive menu-driven interface** with 6 control modes:
  1. **Manual Control**: Enter any angle (0-180°)
  2. **Preset Positions**: Quick access to 0°, 45°, 90°, 135°, 180°
  3. **Sweep Demo**: Automatic sweep from 0° to 180° and back
  4. **Smooth Movement**: Gradual transition demonstrations
  5. **Quick Center**: Instantly center servo at 90°
  6. **Safe Exit**: Centers servo before closing connection
- **Command-based protocol**: Structured `COMMAND:parameter` format
- **Real-time feedback**: Arduino confirms each movement
- **Smooth servo motion**: Incremental movement to prevent mechanical stress
- **LED status indicator**: Built-in LED shows when servo is moving

Technical Details

- **Python Script**: `servo_control.py` - Full-featured interactive menu system
- **Arduino Sketch**: `arduino_servo.ino` - Servo controller with command parsing
- **Protocol Format**: `SERVO:angle\n` (e.g., `SERVO:90\n`)
- **Hardware Required**: SG90 servo motor connected to Arduino Pin 9
- **Baud Rate**: 9600
- **Servo Range**: 0-180 degrees
- **Power Requirements**:
  - Light testing: Arduino 5V pin (no load)
  - Under load: External 5V supply (100-250mA draw)

What You'll Learn

- Designing structured command protocols for serial communication
- Parsing commands on Arduino using `indexOf()` and `substring()`
- Controlling PWM-based hardware (servo motors)
- Creating interactive menu systems in Python
- Managing servo positioning and smooth motion control
- Power considerations for motor control

Expected Output

```
==================================================
Servo Control Menu:
==================================================
1. Manual Control (enter specific angle)
2. Preset Positions (0°, 45°, 90°, 135°, 180°)
3. Sweep Demo (0° to 180° and back)
4. Smooth Movement Demo
```

```
5. Center Servo (90°)
6. Exit

Enter choice (1-6): 1

=== Manual Servo Control ===
Enter angle (0-180) or 'q' to quit

Enter angle: 45
Sent: SERVO:45
Arduino says: Servo moved to 45 degrees
```

Visual Feedback: - Arduino LED lights up during servo movement - 3 LED flashes on startup to indicate ready state - Servo physically moves to commanded position

Ideal For

- Learning command-based serial protocols
- Understanding hardware control via serial communication
- Building interactive control interfaces
- Servo motor control and positioning
- Intermediate serial communication projects

Quick Comparison

Feature	arduino_serial/	Servo_Control/
Complexity	Beginner	Intermediate
Communication	Simple strings	Structured commands
Interaction	One message	Interactive menu
Hardware	Arduino only	Arduino + SG90 servo
Protocol	Basic string (`"message\n"`)	Command format (`"SERVO:90\n"`)
Purpose	Learn serial basics	Control physical hardware
Runtime	Runs once, exits	Continuous interactive session
Python Lines	~74 lines	~228 lines
Arduino Features	Serial read/write, LED	Serial parsing, servo control, smooth motion

---

Getting Started

Prerequisites (Both Projects)

1. **Hardware**:
   - Raspberry Pi 5 (or similar)
   - Arduino Uno R3
   - USB A to B cable
   - (Servo_Control only) SG90 servo motor + jumper wires
2. **Software Setup**:

```
# Grant serial port access (one-time)
sudo usermod -a -G dialout $USER
# Then logout/login

# Activate virtual environment
cd /home/dom/Serial_Comms
source venv/bin/activate

# Install dependencies
pip install -r requirements.txt
```

3. **Verify Arduino Connection**:

```
ls /dev/tty*     # Look for /dev/ttyUSB0 or /dev/ttyACM0
lsusb            # Should show "Arduino SA Uno R3"
```

Recommended Learning Path

1. **Start with arduino_serial/**
   - Run `python3 arduino_serial/test_serial.py`
   - Understand basic serial communication flow
   - Experiment with the LED indicator version
   - Read the detailed `SETUP_GUIDE.md`
2. **Progress to Servo_Control/**
   - Set up servo hardware (see `set_up_guide_servo.md`)
   - Run `python3 Servo_Control/servo_control.py`
   - Explore different control modes
   - Understand command-based protocols
3. **Explore two_way_comms/** (not covered in this overview)
   - Bidirectional messaging
   - Diagnostic tools for troubleshooting

Documentation Resources

- **Project-wide guide**: `/home/dom/Serial_Comms/CLAUDE.md` - Complete technical reference
- **arduino_serial/ setup**: `arduino_serial/SETUP_GUIDE.md` - Step-by-step beginner guide
- **Servo_Control/ setup**: `Servo_Control/set_up_guide_servo.md` - Comprehensive servo project guide

---

Common Serial Communication Patterns (Both Projects)

Python Pattern

```python
import serial
import time

# Open connection
ser = serial.Serial('/dev/ttyUSB0', 9600, timeout=1)
time.sleep(2)  # CRITICAL: Wait for Arduino reset

# Send command (encode to bytes)
ser.write("message\n".encode())

# Read response (decode from bytes)
if ser.in_waiting > 0:
    response = ser.readline().decode('utf-8').strip()

ser.close()
```

Arduino Pattern

```cpp
void setup() {
  Serial.begin(9600);  // Match Python baud rate
}

void loop() {
  if (Serial.available() > 0) {
    String message = Serial.readStringUntil('\n');
    // Process message
    Serial.println("Response");  // Send back to Python
  }
}
```

Critical Success Factors

Both projects require: - Matching baud rates (9600 on both devices) - 2+ second delay after opening serial connection (Arduino resets) - Only one program accessing serial port at a time (close Arduino Serial Monitor!) - Proper string encoding (`.encode()` in Python, `.decode()` when receiving) - Newline delimiters (`\n`) for message boundaries - Serial port permissions (`dialout` group membership)

Happy Learning! Start with `arduino_serial/` for fundamentals, then advance to `Servo_Control/` for practical hardware control.