MECHATRONICS SYSTEM INTEGRATION MCTA3203

Bluetooth Data Interfacing:

Remote Temperature Monitoring and Control via Bluetooth

EXPERIMENT 8

DATE: 1ST December 2025
SECTION: 2
GROUP: 18
SEMESTER 1, 2025/2026

NO	NAME	MATRIC NO
1.	SHAMSUL HAKIMEE BIN SHAMSUL HAIREE	2315027
2.	QASHRINA AISYA BINTI MOHD NAZARUDIN	2315184
3.	NUREL HUMAIRAH BINTI MUHAMMAD FIRDAUS	2315680

DATE OF SUBMISSION :
Monday, 8TH December 2025

Abstract

This lab report details the process of creating a wireless temperature monitoring and control system using Bluetooth communication between an ESP32 microcontroller and a smartphone/computer. The experiment utilized the ESP32's built-in Bluetooth capabilities to transmit environmental data collected by a DHT11 sensor and to receive control commands from a mobile terminal application. Additionally, a Python script was developed to interface with the ESP32 via USB Serial to log and monitor real-time system feedback. The system successfully demonstrated the ability to toggle a servo motor (simulating a fan) using "ON/OFF" commands sent wirelessly, while simultaneously monitoring temperature and humidity data.

Table of Contents

1.0 Introduction

Wireless communication plays a pivotal role in modern mechatronics and Industrial Internet of Things (IIoT) applications. Bluetooth technology provides a reliable, low-cost method for short-range communication between embedded systems and user interfaces such as smartphones or computers.

The objective of this laboratory session is to create a wireless temperature monitoring system using Bluetooth communication. This involves interfacing a microcontroller (ESP32) with a temperature sensor (DHT11) to read environmental data and transmitting it wirelessly. Furthermore, the system is designed to receive simple control commands from a paired device to actuate a physical output, in this case, a servo motor simulating a cooling fan.

By executing this experiment, students gain practical experience in serial communication protocols, sensor integration, and the implementation of non-blocking control logic using `millis()` to manage data acquisition and actuator movement simultaneously.

2.0 Materials and Equipment

Software**:**

- Arduino IDE: For writing, compiling, and uploading the firmware to the ESP32.
- Python (VS Code/IDLE): For running the serial monitoring script.
- Serial Bluetooth Terminal (Android/Smartphone): For sending wireless commands to the ESP32.

Hardware**:**

- ESP32 Development Board: The core microcontroller with integrated Bluetooth and WiFi.
- DHT11 Sensor: Used for measuring Temperature and Humidity.
- Servo Motor: Used to simulate a Fan/Actuator.
- Breadboard: For circuit prototyping.
- Jumper Wires: For interconnections.
- USB Cable: For programming and power.

3.0 Experimental Setup

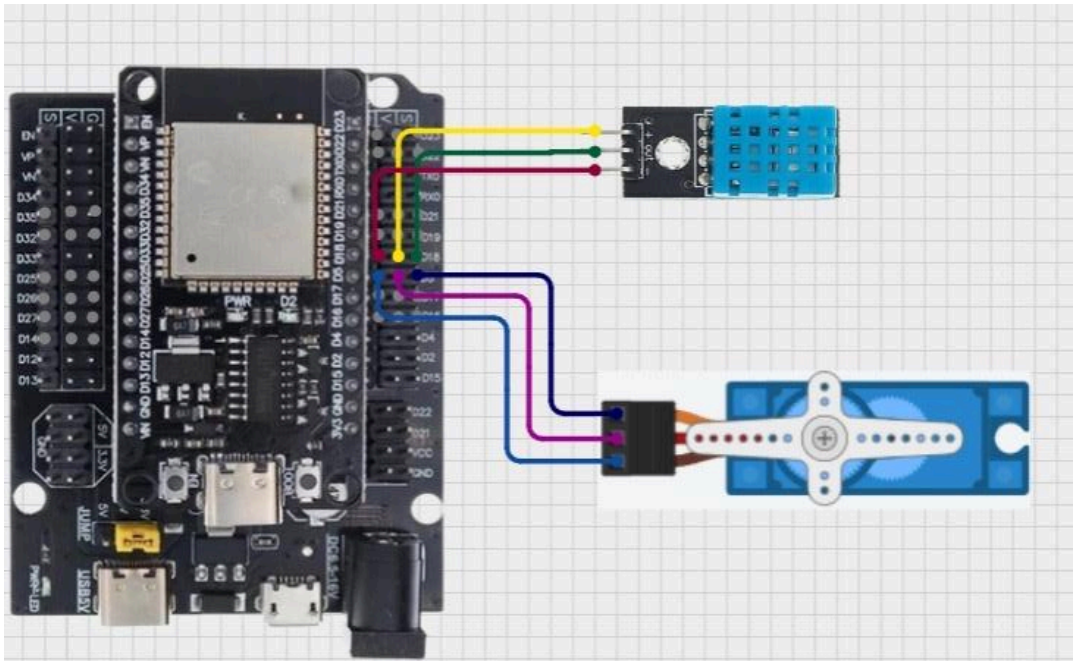


Figure 3.0 Circuit Diagram

3.1 Circuit Setup

1. The DHT11 sensor was placed on the breadboard, with its data pin connected to GPIO 5.
2. The Servo motor was connected to GPIO 18 to act as the controllable output.
3. The ESP32 was connected to the computer via USB for power and serial monitoring.
4. The system was powered up, and the Bluetooth radio was initialized with the name "ESP32_FAN_MONITOR".

4.0 Methodology

4.1 Arduino/ESP32 Programming The firmware was developed using the Arduino IDE with the following logic:

1. Initialization: The `BluetoothSerial`, `DHT`, and `ESP32Servo` libraries were included. The Bluetooth device was named "ESP32_FAN_MONITOR".
2. Sensor Logic: A non-blocking timer reads the DHT11 sensor every 2 seconds to avoid pausing the servo movement.
3. Command Parsing: The loop checks `SerialBT.available()`. If the command 'O' is received, the boolean `isFanOn` is set to TRUE. If 'X' is received, it is set to FALSE.
4. Actuation: If `isFanOn` is TRUE, the servo performs a sweep motion (0° to 180°) updating every 10ms.

4.2 Python Monitoring Script A Python script was written to interface with the ESP32's USB Serial port (COM6).

1. Connection: The script uses the `serial` library to connect to the ESP32 at a baud rate of 115200.
2. Data Logging: It continuously reads the output printed by the ESP32 (which includes sensor readings and command feedback) and displays it on the PC console.
3. Error Handling: The script includes `try-except` blocks to handle connection errors or interruptions.

4.3 Code Used

Python Code:

```
import serial
import time
import sys

# --- Configuration ---
# IMPORTANT: Change this to the COM port your ESP32 is using when plugged in via USB.
# (e.g., 'COM3' on Windows, '/dev/ttyUSB0' on Linux, '/dev/tty.SLAB_USBtoUART' on
macOS)
SERIAL_PORT = 'COM6'
BAUD_RATE = 115200  # MUST MATCH the Serial.begin(115200) in the Arduino code


def main():
```

```
    ser = None

    try:
        # Establish connection to the USB Serial Port
        ser = serial.Serial(SERIAL_PORT, BAUD_RATE, timeout=1)

        # Wait a moment for the connection to fully initialize and for ESP32 to restart
        time.sleep(2)

        print(f"================================================")
        print(f"Connected to ESP32 USB Serial on {SERIAL_PORT} at {BAUD_RATE} baud.")
        print(f"Data from ESP32 will be printed below:")
        print(f"================================================")

        while True:
            # Check for incoming data
            if ser.in_waiting > 0:
                # Read a line of data, decode it, and print it to the console
                line = ser.readline().decode('utf-8', errors='ignore').strip()
                if line:
                    print(line)

            # Use a small sleep to prevent the loop from hogging CPU resources
            time.sleep(0.01)

    except serial.SerialException as e:
        print(f"Error connecting to the serial port: {e}", file=sys.stderr)
        print("Ensure the ESP32 is plugged in via USB and the SERIAL_PORT is correct.",
file=sys.stderr)
    except KeyboardInterrupt:
        print("\nProgram interrupted by user.")
    finally:
        if ser is not None and ser.is_open:
            ser.close()
            print("Serial connection closed.")


if __name__ == "__main__":
    main()
```

Arduino Code:

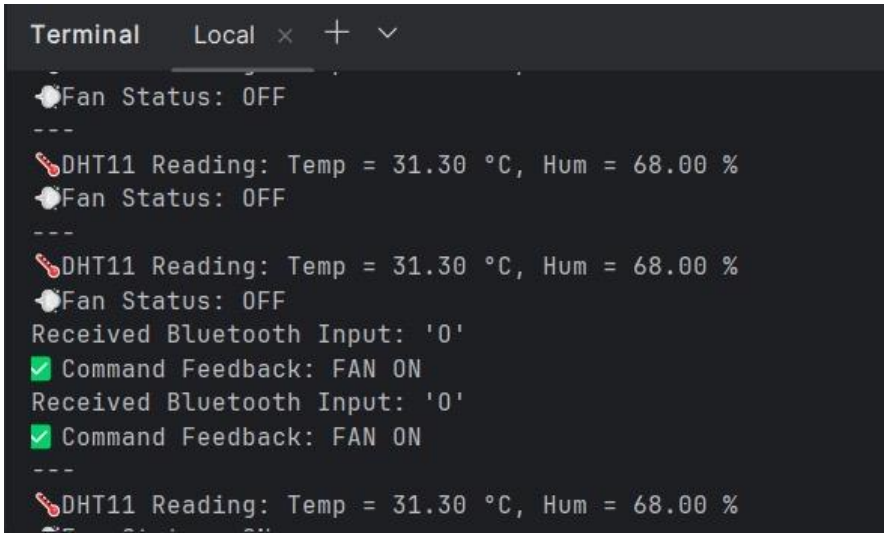
```
#include <BluetoothSerial.h>
#include <DHT.h>
#include <ESP32Servo.h>

// --- Configuration ---
#define DHTPIN 5        // DHT11 data pin connected to ESP32 D5
#define DHTTYPE DHT11   // DHT 11 sensor
#define SERVO_PIN 18    // Servo connected to ESP32 D18
#define BLUETOOTH_NAME "ESP32_FAN_MONITOR"

// --- Object Initialization ---
BluetoothSerial SerialBT;
DHT dht(DHTPIN, DHTTYPE);
Servo fanServo;

// --- State Variables ---
bool isFanOn = false; // Tracks the state of the fan
unsigned long lastMoveTime = 0; // Time of the last servo movement
const int moveInterval = 10;    // Delay in milliseconds between servo steps
int currentAngle = 0;           // Current servo angle (0 to 180)
bool increasing = true;         // Direction of servo movement

void setup() {
  // Use a fast baud rate for the USB Serial Monitor
  Serial.begin(115200);
  SerialBT.begin(BLUETOOTH_NAME);
  dht.begin();

  // Attach the servo object to the defined pin
  fanServo.setPeriodHertz(50);
  fanServo.attach(SERVO_PIN, 500, 2400);
  fanServo.write(0); // Initialize servo to 0 degrees

  Serial.println("========================================");
  Serial.println("DHT11, Servo, and Bluetooth Monitoring Ready");
  Serial.println("Connect via Bluetooth and send 'O' (ON) or 'X' (OFF).");
  Serial.println("========================================");
}
```

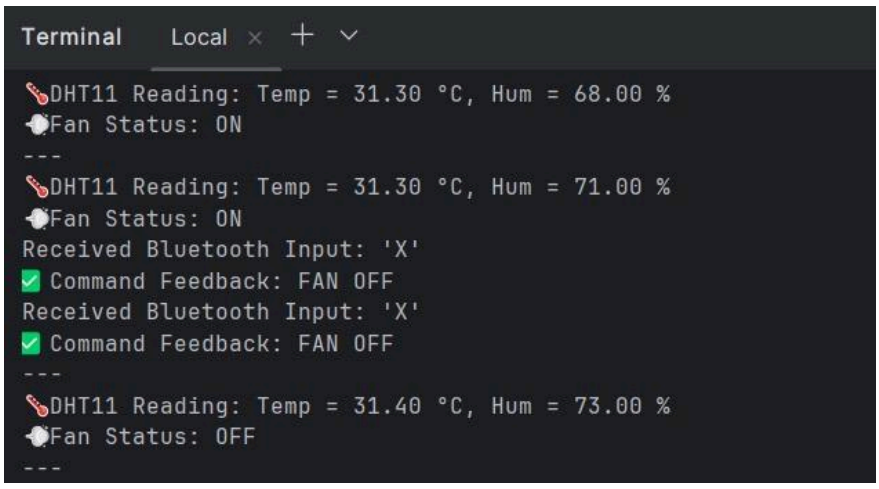
```
void loop() {
  // 1. --- Read and Display Sensor Data (Every 2 seconds) ---
  static unsigned long lastReadTime = 0;
  if (millis() - lastReadTime >= 2000) {
    lastReadTime = millis();

    float hum = dht.readHumidity();
    float temp = dht.readTemperature();

    Serial.println("---");
    if (isnan(hum) || isnan(temp)) {
      Serial.println("🌡️ FAILED to read from DHT sensor!");
    } else {
      // Display the DHT11 sensor data (what you wanted the Python script to show)
      Serial.print("🌡️ DHT11 Reading: Temp = ");
      Serial.print(temp, 2); // Print with 2 decimal places
      Serial.print(" °C, Hum = ");
      Serial.print(hum, 2);
      Serial.println(" %");
      Serial.print("💨 Fan Status: ");
      Serial.println(isFanOn ? "ON" : "OFF");
    }
  }

  // 2. --- Receive and Process Bluetooth Commands ---
  if (SerialBT.available()) {
    // Read the incoming command
    String cmd = SerialBT.readStringUntil('\n');
    cmd.trim(); // Remove any leading/trailing whitespace

    // Display the input received via Bluetooth (what you wanted the Python script to show)
    Serial.print("Received Bluetooth Input: '");
    Serial.print(cmd);
    Serial.println("'");

    // Command Logic
    if (cmd.equalsIgnoreCase("O")) {
      isFanOn = true;
      // Display FAN ON status (what you wanted the Python script to show)
```

```
    Serial.println("✅ Command Feedback: FAN ON");
  } else if (cmd.equalsIgnoreCase("X")) {
    isFanOn = false;
    fanServo.write(0); // Set servo to a fixed position when off
    // Display FAN OFF status (what you wanted the Python script to show)
    Serial.println("✅ Command Feedback: FAN OFF");
  } else {
    Serial.println("⚠️ Unrecognized command. Send 'O' (ON) or 'X' (OFF)");
  }
 }

 // 3. --- Control Servo Sweep (if Fan is ON) ---
 if (isFanOn) {
  if (millis() - lastMoveTime >= moveInterval) {
    lastMoveTime = millis();

    // Sweep logic (0 -> 180 -> 0)
    if (increasing) {
      currentAngle++;
      if (currentAngle >= 180) {
        currentAngle = 180;
        increasing = false;
      }
    } else {
      currentAngle--;
      if (currentAngle <= 0) {
        currentAngle = 0;
        increasing = true;
      }
    }

    fanServo.write(currentAngle);
  }
 }
}
```

5.0 Result

The experiment successfully demonstrated the implementation of a wireless monitoring and control system using the ESP32 microcontroller. The system reliably acquired real-time temperature and humidity data from the DHT11 sensor, which was transmitted via USB serial and accurately logged by the Python monitoring script. Simultaneously, the Bluetooth communication interface functioned as intended; sending the command 'O' from the smartphone terminal successfully triggered the "FAN ON" state, causing the servo motor to initiate a continuous sweep motion, while the 'X' command immediately halted the operation and reset the servo to the 0-degree position. These actions were confirmed by corresponding status messages displayed on both the smartphone terminal and the Python console, validating the successful integration of sensor data acquisition, non-blocking actuator control, and wireless serial communication.





6.0 Discussion

The experiment highlighted the efficiency of the ESP32's dual-core architecture in handling communication and actuation. Unlike the basic delay-based blinking examples, this experiment utilized `millis()` for multitasking. This was crucial because using `delay()` for the servo sweep would have blocked the processor, preventing it from receiving Bluetooth commands or reading the sensor effectively.

The integration of the Python script provided a robust way to debug and log data on the PC side while simultaneously controlling the device wirelessly from a smartphone. This mimics a real-world SCADA or telemetry system where a local controller (ESP32) communicates with a central logger (Python) and a remote operator (Smartphone). One observation was the difference in range; Bluetooth Class 2 radios typically support up to 10 meters, which was sufficient for this lab setup.

7.0 Conclusion

This laboratory session successfully demonstrated the creation of a wireless temperature monitoring system using Bluetooth communication. By integrating the DHT11 sensor and a servo motor with the ESP32, the group was able to read environmental data and control a physical actuator remotely.

The Arduino code successfully implemented the logic to parse ASCII commands ("O" and "X") to toggle the fan status , while the Python script verified the data transmission via the USB serial port. This experiment confirmed the viability of using Bluetooth Classic on the ESP32 for simple, low-latency, and short-range industrial control applications

8.0 Recommendation

To further optimize the wireless monitoring system, several key enhancements are proposed for future development. Firstly, the Python monitoring interface should be upgraded to include real-time data visualization using libraries like Matplotlib, which would enable clearer analysis of temperature trends compared to simple text logging. Secondly, transitioning the communication protocol from Bluetooth Classic to Bluetooth Low Energy (BLE) is highly recommended to significantly reduce power consumption, making the device far more suitable for battery-dependent remote applications. Furthermore, the control logic could be improved by implementing a closed-loop system where the fan automatically activates when specific temperature thresholds are exceeded, rather than relying solely on manual user commands. Finally, integrating a local OLED display to show connection status and sensor readings would provide immediate on-device feedback, reducing the constant dependence on a paired smartphone for basic monitoring.

9.0 References

1. Chaudhary, N. (n.d.). *Arduino Bluetooth basic tutorial*. Arduino Project Hub. https://projecthub.arduino.cc/NeilChaudhary/arduino-bluetooth-basic-tutorial-9cff12
2. Department of Mechatronics Engineering. (2025). *Mechatronics system integration (MCTA3203) lab module: Experiment 8 - Bluetooth data interfacing*. International Islamic University Malaysia.
3. HowToMechatronics. (n.d.). *Arduino and HC-05 Bluetooth module complete tutorial*. https://howtomechatronics.com/tutorials/arduino/arduino-and-hc-05-bluetooth-module-tutorial/
4. pySerial. (n.d.). *pySerial documentation*. Read the Docs. https://pyserial.readthedocs.io/en/latest/

10.0 Acknowledgement

We would like to express my sincere gratitude to Dr. Wahju Sediono, Dr. Zulkifli Bin Zainal Abidin, our teaching assistant, and our peers for their invaluable guidance and support throughout the completion of this report. Their insights, feedback, and expertise greatly contributed to the depth and quality of this work. We truly appreciate their time, patience, and commitment to our academic growth.

11.0 Student's Declaration

Certificate of Originality and Authenticity

This is to certify that we are responsible for the work submitted in this report, that the original work is our own except as specified in the references and acknowledgement, and that the original work contained herein have not been untaken or done by unspecified sources or persons.

We hereby certify that this report has not been done by only one individual and all of us have contributed to the report. The length of contribution to the reports by each individual is noted within this certificate.

We also hereby certify that we have read and understand the content of the total report and no further improvement on the reports is needed from any of the individual's contributors to the report.

We therefore, agreed unanimously that this report shall be submitted for marking and this final printed report has been verified by us.

Name: Shamsul Hakimee Bin Shamsul Hairee	Read	[/]
Matric Number: 2315027	Understand	[/]
Signature: 	Agree	[/]

Name: Qashrina Aisya binti Mohd Nazarudin	Read	[/]
Matric Number: 2315184	Understand	[/]
Signature: 	Agree	[/]

Name: Nurel Humairah Binti Muhammad Firdaus	Read	[/]
Matric Number: 2315680	Understand	[/]
Signature: 	Agree	[/]