Automated Temperature Profiling Using Microcontroller

Applied Measurement and Control 2025

Prof. Dr. Rolf Becker

Hannah Pearly Shelly David, 33189; Kirandeep Kaur, 34284; Amir-Hamza Akash, 32425

Introduction

(by Hannah Pearly Shelly David)

The European Union's Water Framework Directive (2024) (European Union, 2024) regularly monitors water bodies, and one among the many water profiling methods include detailed vertical profiles of lakes. Organizations like LINEG typically conduct this by manually lowering a multisensor into the water in 0.5-meter increments. At each depth, the operator has to wait for the sensor to stabilize before logging the readings using a handheld. This repetitive process takes 20 to 30 minutes and ties up personnel who could otherwise taking other necessary samples parallelly. The current manual system, although effective, lacks efficiency and consistency. This project is about automating the descent of a temperature sensor (demo purpose), controlling the timing of data collection with the help of a microcontroller to manage all system components. The intended result is of a more autonomous process that replicates the same profiling steps with a lot more consistency.

Materials and Methods

(by Amir-Hamza Akash)

The prototype (Figure 1.) was constructed with an Arduino Uno microcontroller, chosen for its ease of programming and compatibility with a commonly available motor shield and sensor. A DS18B20 digital temperature sensor was used to measure water temperature at each depth. This sensor communicates using the OneWire protocol and requires a 4.7kΩ pull-up resistor on the data line for stable operation (Figure 2.). For vertical motion, a 12V worm-geared DC motor was used (Model no. JGY370) to simulate the lowering and retrieval of the temperature sensor via a string (Figure 3.). The motor was controlled using an Adafruit Motor Shield v2, which can be mounted onto the Arduino Uno microcontroller and provides high-current motor control.

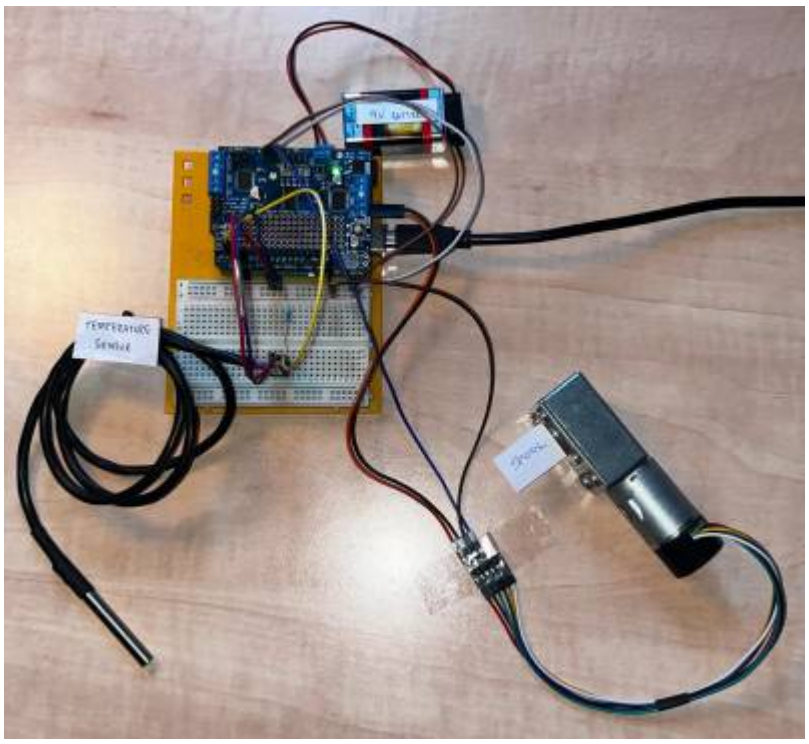

Figure 1. View of the electronics setup showing the Arduino Uno, motor shield, 9V battery and DS18B20 sensor connections

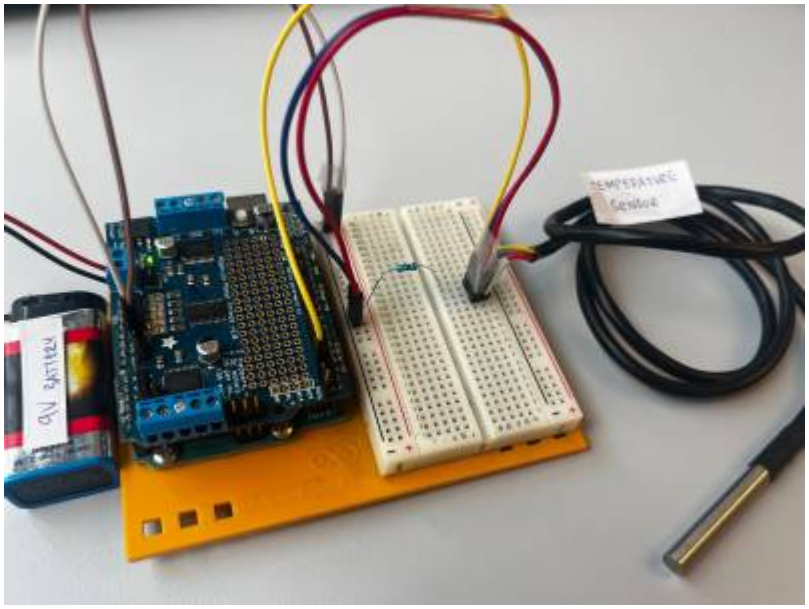

Figure 2. DS18B20 digital temperature sensor wired to digital pin D2, with a 4.7kΩ pull-up resistor between data and VCC

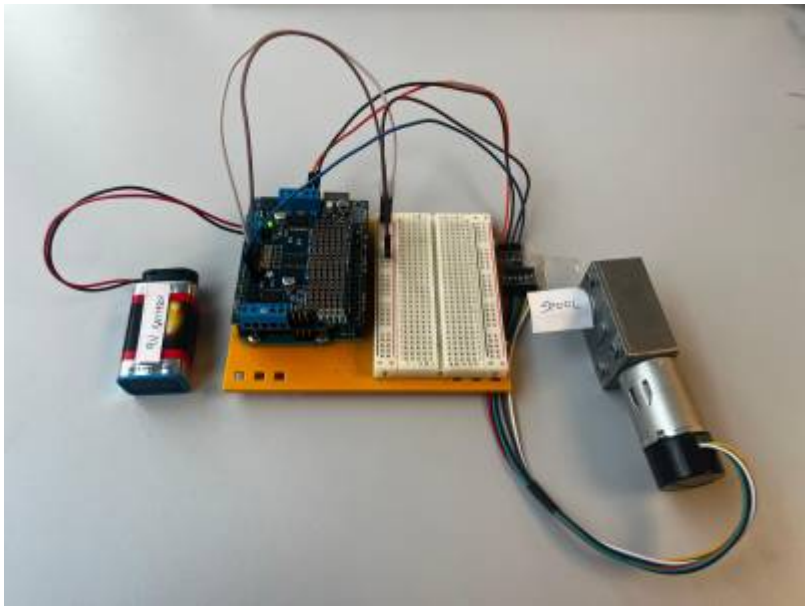

Figure 3. 12V DC worm gear motor for lowering and raising the sensor.

Power was supplied via a 9V battery instead of a 12V battery due to availability constraints, though the system is ideally meant to be powered with a regulated 12V source to achieve full motor torque. The DS18B20 sensor was connected to digital pin D2 on the Arduino. The motor was connected to M1 on the motor shield. A built-in Hall effect encoder in the motor was initially intended to measure rotation and thereby estimate depth, but it was later excluded due to lack of encoder signal detection followed by non-functional outputs, possibly caused by hardware faults or wiring inconsistencies. An alternate method was adopted where depth estimation was done using a time-based method.

This method is grounded in the geometry of the spool and the known motor characteristics. A spool diameter of approximately 4 cm was assumed. From this, the circumference of the spool was calculated using the formula for the circumference of a circle:

Circumference = π × Diameter = π × 4cm ≈ 12.57cm

This implies that one complete rotation of the spool releases approximately 12.57 cm of wire.

The DC worm gear motor in the system has a rated speed of 90 RPM at 12V. However, due to the use of a 9V battery instead of the rated 12V power supply, a proportional reduction in speed was assumed. A reduction of approximately 25% was estimated, resulting in an adjusted motor speed of around 67.5 RPM, or:

67.5 RPM ÷ 60 ≈ 1.125 rotations per second

To descend 0.5 meters (50 cm) of wire, the number of required spool rotations was calculated as:

50cm ÷ 12.57 cm per rotation ≈ 3.98 rotations

The time required to complete this number of rotations at the given speed is:

3.98 rotations ÷ 1.125 rotations/sec ≈ 3.54 seconds

To ensure reliable descent, the descent duration was rounded up to 4 seconds per 0.5 meters. This time-based approach was used to approximate sensor depth at each step, aiding with the automated collection of temperature data at predefined intervals. By assuming a spool diameter of

approximately 4 cm and a reduced RPM (75% of original speed at 12V) due to under-voltage conditions, each 0.5-meter descent was timed for 4 seconds due to t.

The code was written in Arduino IDE and configured to move the motor forward in short bursts, pause for 30 seconds at each stop to simulate thermal stabilization, and read the temperature using the DS18B20. After the final depth was reached, the motor reversed direction and reeled the wire back to the starting point.

Results

(by Kirandeep Kaur)

The Arduino code successfully controlled the descent and temperature reading collection. For the sole purpose of a demo, the code was written in such a way it samples a 3 meter deep lake or water body. This would mean a total of 5 depth levels were configured, corresponding to a total descent of 2.5 meters. Readings are not taken at the 3 meter mark as by then the temperature readings would be affected by the temperature of the waterbed. Therefore, at each step, the system printed the estimated depth and corresponding temperature reading onto the serial monitor. The readings were separated by 30-second delays to mimic the real sampling procedure. Once all data points were collected, the motor reversed and reeled in the sensor based on the total descent time. The final message confirmed the completion of the measurement cycle (Figure 5.).

```
#include <Wire.h>
#include <Adafruit_MotorShield.h>
#include <OneWire.h>
#include <DallasTemperature.h>

// Motor Shield Setup
Adafruit_MotorShield AFMS = Adafruit_MotorShield();
Adafruit_DCMotor *motor = AFMS.getMotor(1); // Connected to M1

// DS18B20 Setup
#define ONE_WIRE_BUS 2
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);

// Parameters
int totalSteps = 5;                 // 5 steps = 2.5 meters
int delayPerStep_ms = 4000;         // 4 seconds per 0.5m descent (estimated
at 9V)
int waitAfterStep_ms = 30000;       // 30 seconds wait per depth before
reading

void setup() {
  Serial.begin(9600);
  AFMS.begin();
  sensors.begin();
```

```
  Serial.println("=== Automated Temperature Logging System ===");
  Serial.println("Starting descent...");

  // Descent loop
  for (int i = 1; i <= totalSteps; i++) {
    Serial.print("\nDescending to estimated depth: ");
    Serial.print(i * 0.5);
    Serial.println(" meters");

    // Lower motor
    motor->setSpeed(150);       // Adjust speed as needed
    motor->run(FORWARD);
    delay(delayPerStep_ms);
    motor->run(RELEASE);

    // Wait before reading
    Serial.println("Waiting 30 seconds for water to stabilize...");
    delay(waitAfterStep_ms);

    // Read temperature
    sensors.requestTemperatures();
    float tempC = sensors.getTempCByIndex(0);

    Serial.print("Temperature at ");
    Serial.print(i * 0.5);
    Serial.print(" meters: ");
    Serial.print(tempC);
    Serial.println(" °C");
  }

  // Rewind sensor to surface
  Serial.println("\nRewinding sensor to surface...");

  motor->setSpeed(150);
  motor->run(BACKWARD);
  delay(totalSteps * delayPerStep_ms);  // Run long enough to rewind all
steps
  motor->run(RELEASE);                  // Stop motor

  Serial.println("Measurement cycle complete.");
}

void loop() {
}
```

Figure 5. Real-time output from Arduino serial monitor showing temperature readings at every 0.5m depth.

The motor behaved as expected in both directions when supplied with 9V, although noticeably under-powered compared to its rated 12V specification. The DS18B20 sensor responded with temperature values at each level. The encoder component of the motor did not produce usable signals during testing and was omitted from the final implementation. An SD card module was not connected during this prototype phase due to time constraints, code and planning are however in place for its integration.

```
#include <SD.h>

// SD Card Setup
const int chipSelect = 10; // CS pin for SD card

void setup() {
  // Initialize SD Card
  Serial.print("Initializing SD card...");
  if (!SD.begin(chipSelect)) {
    Serial.println(" SD card failed or not present.");
    while (1); // Halt program
  }
  Serial.println(" SD card initialized.");

  // Open file and write header
  dataFile = SD.open("datalog.txt", FILE_WRITE);
  if (dataFile) {
    dataFile.println("Depth (m), Time (ms), Temperature (C)");
    dataFile.close();
  } else {
    Serial.println("Error opening datalog.txt");
  }

  // Log to SD
    dataFile = SD.open("datalog.txt", FILE_WRITE);
    if (dataFile) {
      dataFile.print(depth, 2);
      dataFile.print(", ");
      dataFile.print(timestamp);
      dataFile.print(", ");
      dataFile.println(tempC, 2);
```

```
      dataFile.close();
    } else {
      Serial.println("Failed to write to SD card.");
    }
}
```

Results

(by Kirandeep Kaur)

The Arduino code successfully controlled the descent and temperature reading collection. For the sole purpose of a demo, the code was written in such a way it samples a 3 meter deep lake or water body. This would mean a total of 5 depth levels were configured, corresponding to a total descent of 2.5 meters. Readings are not taken at the 3 meter mark as by then the temperature readings would be affected by the temperature of the waterbed. Therefore, at each step, the system printed the estimated depth and corresponding temperature reading onto the serial monitor. The readings were separated by 30-second delays to mimic the real sampling procedure. Once all data points were collected, the motor reversed and reeled in the sensor based on the total descent time. The final message confirmed the completion of the measurement cycle (Figure 5.).

```
#include <Wire.h>
#include <Adafruit_MotorShield.h>
#include <OneWire.h>
#include <DallasTemperature.h>

// Motor Shield Setup
Adafruit_MotorShield AFMS = Adafruit_MotorShield();
Adafruit_DCMotor *motor = AFMS.getMotor(1); // Connected to M1

// DS18B20 Setup
#define ONE_WIRE_BUS 2
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);

// Parameters
int totalSteps = 5;                 // 5 steps = 2.5 meters
int delayPerStep_ms = 4000;         // 4 seconds per 0.5m descent (estimated
at 9V)
int waitAfterStep_ms = 30000;       // 30 seconds wait per depth before
reading

void setup() {
  Serial.begin(9600);
  AFMS.begin();
  sensors.begin();

  Serial.println("=== Automated Temperature Logging System ===");
  Serial.println("Starting descent...");
```

```
  // Descent loop
  for (int i = 1; i <= totalSteps; i++) {
    Serial.print("\nDescending to estimated depth: ");
    Serial.print(i * 0.5);
    Serial.println(" meters");

    // Lower motor
    motor->setSpeed(150);       // Adjust speed as needed
    motor->run(FORWARD);
    delay(delayPerStep_ms);
    motor->run(RELEASE);

    // Wait before reading
    Serial.println("Waiting 30 seconds for water to stabilize...");
    delay(waitAfterStep_ms);

    // Read temperature
    sensors.requestTemperatures();
    float tempC = sensors.getTempCByIndex(0);

    Serial.print("Temperature at ");
    Serial.print(i * 0.5);
    Serial.print(" meters: ");
    Serial.print(tempC);
    Serial.println(" °C");
  }

  // Rewind sensor to surface
  Serial.println("\nRewinding sensor to surface...");

  motor->setSpeed(150);
  motor->run(BACKWARD);
  delay(totalSteps * delayPerStep_ms);  // Run long enough to rewind all
steps
  motor->run(RELEASE);                  // Stop motor

  Serial.println("Measurement cycle complete.");
}

void loop() {
}
```

Figure 5. Real-time output from Arduino serial monitor showing temperature readings at every 0.5m depth.

The motor behaved as expected in both directions when supplied with 9V, although noticeably under-powered compared to its rated 12V specification. The DS18B20 sensor responded with temperature values at each level. The encoder component of the motor did not produce usable signals during testing and was omitted from the final implementation. An SD card module was not connected during this prototype phase due to time constraints, code and planning are however in place for its integration.

```
#include <SD.h>

// SD Card Setup
const int chipSelect = 10; // CS pin for SD card

void setup() {
  // Initialize SD Card
  Serial.print("Initializing SD card...");
  if (!SD.begin(chipSelect)) {
    Serial.println(" SD card failed or not present.");
    while (1); // Halt program
  }
  Serial.println(" SD card initialized.");

  // Open file and write header
  dataFile = SD.open("datalog.txt", FILE_WRITE);
  if (dataFile) {
    dataFile.println("Depth (m), Time (ms), Temperature (C)");
    dataFile.close();
  } else {
    Serial.println("Error opening datalog.txt");
  }

  // Log to SD
    dataFile = SD.open("datalog.txt", FILE_WRITE);
    if (dataFile) {
      dataFile.print(depth, 2);
      dataFile.print(", ");
      dataFile.print(timestamp);
      dataFile.print(", ");
      dataFile.println(tempC, 2);
```

```
      dataFile.close();
    } else {
      Serial.println("Failed to write to SD card.");
    }
}
```

Discussion

(by Hannah Pearly Shelly David)

The system presented here is an improvement over the manual sampling method in terms of operator efficiency especially. While the current version operates in a lab setting, the core components mimic the actual field conditions. One of the key limitations was the lack of a working encoder, which would have allowed for real-time depth feedback. The workaround was the use of timed descent and an assumed spool diameter which proved effective for this prototype, but use of the encoder in the future would benefit greatly from direct feedback systems.

Another limitation was power delivery. The 9V battery used was insufficient to fully drive the motor at its rated speed and torque, which could be problematic in outdoor or deeper water environments. Additionally, the absence of an actual winding spool and waterproofing limits the deployment of this system in real water bodies. The system is currently limited to serial output, but future enhancements can include the use of SD card data logging or wireless data transmission via Wi-Fi or LoRa modules.

Despite these limitations, the prototype demonstrated the core functionality: automated descent, timed temperature reading, and motorized rewinding.

Conclusion

(by Kirandeep Kaur)

This project has somewhat successfully demonstrated a microcontroller-based prototype for automated water temperature profiling. Using an Arduino Uno, DS18B20 sensor, and a motorized mechanism, the system automates a previously manual process, reducing sampling time and improving consistency. Although limited by certain hardware constraints such as encoder failure and underpowered supply, the prototype fulfills its educational and functional goals. Future versions will aim to integrate data logging, improved power systems, real-time feedback, and a deployable mechanical structure to enable field use.

References

1. European Union. (2024). Water Framework Directive. https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32000L0060 2. Rufián, R. (n.d.). Guide for DS18B20 Temperature Sensor with Arduino. Random Nerd Tutorials. https://randomnerdtutorials.com/guide-for-ds18b20-temperature-sensor-with-arduino/