Carrero-Carralero et al., 2027. High-frequency, spatially distributed monitoring of water temperature: the MultiTEMP system. Limnetica 46(2), 2027

Supplementary material 1

In the main text, the possibility of both types of system control is mentioned. The automatic code operates in an automated and fixed manner, continuously recording temperatures in a single location, allowing for a multitemporal analysis of the data. In contrast, the manual code requires operator interaction, enabling the system to be moved and data to be recorded in different locations, offering a multispatial analysis. Additionally, the second code includes optimizations in error handling and storage efficiency, making it more suitable for measurements on the move.

Below are the corresponding codes for the two types of system programming, along with a brief introduction to their individual characteristics:

Automatic mode Arduino code

The automatic code allows temperature readings from multiple DS18B20 sensors using the 1-Wire protocol. It records data on an SD card automatically and continuously, ensuring that each new record is stored without overwriting the previous ones. It implements an RTCZero clock to track time and update temperatures at defined intervals. Additionally, it uses a 7-segment display to show the current record number and an LED indicator that lights up when temperatures remain stable. This system is designed to operate in a fixed location, enabling multitemporal monitoring of environmental conditions.

```
#include <RTCZero.h>
#include <OneWire.h>
#include <DallasTemperature.h>
#include <SD.h>

// **Display Variables**
int pinA = 2, pinB = A5, pinC = 1, pinD = A2;
int pinE = A6, pinF = A1, pinG = A3;
int D1 = 0, D2 = A4;

// **Temperature Variables**
float deltaT = 0.1;
int nreg, lastreg, reg = 1;
float t0, t00, t1, t10, t2, t20, t3, t30, t4, t40, t5, t50;
float t6, t60, t7, t70, t8, t80, t9, t90, t11, t110;

// **RTC Variables**
RTCZero myRTC;
int deltaRTC = 60;
// **Update Temp every x seconds**
long int rtc0 = 0, rtc = 0, t = 0;

// **Button Variables**
int bt1, bt0;
int pinBUT = 14;
```

```
// **Temperature Sensor (1-Wire)**
const int tempPin = 6;
OneWire oneWireObjeto(tempPin);
DallasTemperature sensorDS18B20(&oneWireObjeto);

// **LED Indicator**
const int pinLED = 7;

// **SD Card Module**
const int chipSelect = 4;
String msgSD;

void setup() {
   Serial.begin(9600);
   int t = 2;
   while (!Serial) {
      delay(1000);
      if ((t--) == 0) break;
   }

   sensorDS18B20.begin();
   sensorDS18B20.setResolution(12);

   pinMode(pinLED, OUTPUT);
```

```
pinMode(pinBUT, INPUT_PULLUP);

Serial.print("Initializing SD card...");
if (!SD.begin(chipSelect)) {
   Serial.println("Card failed, or not present");
} else {
   Serial.println("Card initialized.");
}

myRTC.begin();
rtc0 = myRTC.getEpoch();
myRTC.setTime(0, 0, 0);
rtc0 = myRTC.getEpoch();
Serial.print("Inicio RTC: ");
Serial.println(rtc0);

pinMode(pinA, OUTPUT);
pinMode(pinB, OUTPUT);
pinMode(pinC, OUTPUT);
pinMode(pinD, OUTPUT);
pinMode(pinE, OUTPUT);
pinMode(pinF, OUTPUT);
pinMode(pinG, OUTPUT);
pinMode(D1, OUTPUT);
pinMode(D2, OUTPUT);
```

```
    delay(1000);
}

void loop() {
    rtc = myRTC.getEpoch();
    t = rtc - rtc0;

    if (t % deltaRTC == 0) {
        getTemp();
        msgSD = String(t) + ',' + String(t0) + ',' + String(t1) + ',' + String(t2) + ',' +
                String(t3) + ',' + String(t4) + ',' + String(t5) + ',' + String(t6) + ',' +
                String(t7) + ',' + String(t8) + ',' + String(t9) + ',' + String(t11);

        File dataFile = SD.open("DATALOG.csv", FILE_WRITE);
        if (dataFile) {
            dataFile.println(msgSD);
            dataFile.close();
            nreg += 1;
            Serial.print("Registro SD: ");
            Serial.println(msgSD);
            digitalWrite(pinLED, HIGH);
        } else {
            Serial.println("Error opening DATALOG.csv");
        }
```

```
    }

    delay(1000);
    digitalWrite(pinLED, LOW);
}

void getTemp() {
    print_off();
    t00 = t0;
    t10 = t1;
    t20 = t2;
    t30 = t3;
    t40 = t4;
    t50 = t5;
    t60 = t6;
    t70 = t7;
    t80 = t8;
    t90 = t9;
    t110 = t11;

    sensorDS18B20.requestTemperatures();

    t0 = sensorDS18B20.getTempCByIndex(10);
    t1 = sensorDS18B20.getTempCByIndex(5);
    t2 = sensorDS18B20.getTempCByIndex(7);
```

```
  t3 = sensorDS18B20.getTempCByIndex(2);
  t4 = sensorDS18B20.getTempCByIndex(6);
  t5 = sensorDS18B20.getTempCByIndex(1);
  t6 = sensorDS18B20.getTempCByIndex(3);
  t7 = sensorDS18B20.getTempCByIndex(4);
  t8 = sensorDS18B20.getTempCByIndex(9);
  t9 = sensorDS18B20.getTempCByIndex(0);
  t11 = sensorDS18B20.getTempCByIndex(8);
}

void print_off() {
  digitalWrite(pinA, LOW);
  digitalWrite(pinB, LOW);
  digitalWrite(pinC, LOW);
  digitalWrite(pinD, LOW);
  digitalWrite(pinE, LOW);
  digitalWrite(pinF, LOW);
  digitalWrite(pinG, LOW);
}
```

Manual mode Arduino code

The manual code is a version of the first one that requires the user to press a button to register each measurement on the SD card. This allows the system to be taken to different locations and measurements to be taken at various points, providing flexibility for multispatial data recording. The button control is optimized to prevent duplicate records due to electrical bounces. Additionally, storage management on the SD card is improved, ensuring that errors do not occur

when reading the last record. The update of the 7-segment display is also optimized to avoid unnecessary delays, and error-handling checks are added to prevent incorrect readings in case of sensor failures.

```
#include <OneWire.h>
#include <DallasTemperature.h>
#include <SD.h>
#include <SPI.h>
#include <Time.h>
#include <RTCZero.h>

// **Display Variables**
const int pinA = 2, pinB = A5, pinC = 1, pinD = A2, pinE = A6, pinF = A1, pinG = A3;
const int D1 = 0, D2 = A4;

// **Temperature Variables**
float deltaT = 0.1;
int nreg, lastreg, reg = 1;
float t1, t10, t2, t20, t3, t30, t4, t40, t5, t50;

// **RTC Variables**
RTCZero myRTC;
int deltaRTC = 4;

// **Update temp every x seconds**
long rtc0 = 0, rtc = 0;
```

```
// **Button Variables**
int bt1, bt0;
const int pinBUT = 14;

// **Pin for 1-Wire bus**
const int tempPin = 6;
OneWire oneWireObjeto(tempPin);
DallasTemperature sensorDS18B20(&oneWireObjeto);

// **Pin for LED**
const int pinLED = 7;

// **SD Card Variables**
const int chipSelect = 4;
String msgSD;

void setup() {
  // Start serial communication
  Serial.begin(9600);
  int t = 2;
  while (!Serial) {
    delay(1000);
    if (--t == 0) break;
  }
```

```
//** Initialize temperature sensors**
sensorDS18B20.begin();
sensorDS18B20.setResolution(12);

// **Set up pins**
pinMode(pinLED, OUTPUT);
pinMode(pinBUT, INPUT_PULLUP);

// **Initialize SD card**
Serial.print("Initializing SD card...");
if (!SD.begin(chipSelect)) {
  Serial.println("Card failed, or not present");
} else {
  Serial.println("card initialized.");
}

// **Get the last register number from SD**
File file = SD.open("DATALOG.csv", FILE_READ);
file.seek(file.size() - 34);
lastreg = file.readStringUntil(',').toInt();
file.close();
Serial.print("Last reg. number: ");
Serial.println(lastreg);
nreg = lastreg + 1;
```

```
  //** Initialize RTC**
  myRTC.begin();
  rtc0 = myRTC.getEpoch();

  //** Set up display pins**
  for (int pin : {pinA, pinB, pinC, pinD, pinE, pinF, pinG, D1, D2}) {
    pinMode(pin, OUTPUT);
  }
}

void loop() {
  printDisplay(nreg);
  rtc = myRTC.getEpoch();

  // **Update temperature data every deltaRTC seconds**
  if (rtc != rtc0 && rtc % deltaRTC == 0) {
    getTemp();
  }

  //** Control LED based on temperature difference**
  if (abs(t1 - t10) < deltaT && abs(t2 - t20) < deltaT && abs(t3 - t30) < deltaT && abs(t4 - t40)
< deltaT && abs(t5 - t50) < deltaT) {
    digitalWrite(pinLED, HIGH);
  } else {
```

```
    digitalWrite(pinLED, LOW);
  }

  bt1 = digitalRead(pinBUT);

  // **Log data to SD card if button is pressed and conditions are met**
  if (bt1 != bt0 && bt1 == LOW && reg == 1 && (abs(t1 - t10) < deltaT && abs(t2 - t20) <
deltaT && abs(t3 - t30) < deltaT && abs(t4 - t40) < deltaT && abs(t5 - t50) < deltaT)) {
    msgSD = (nreg < 10 ? '0' : "") + String(nreg) + ',' + String(t1) + ',' + String(t2) + ',' + String(t3)
+ ',' + String(t4) + ',' + String(t5);

    File dataFile = SD.open("DATALOG.csv", FILE_WRITE);
    if (dataFile) {
      dataFile.println(msgSD);
      dataFile.close();
      nreg++;
      Serial.print("Registro SD: ");
      Serial.println(msgSD);
    } else {
      Serial.println("Error opening DATALOG.csv");
    }
    reg = 0;
  }

  bt0 = bt1;
```

```
}

// **Function to extract the sensor temperatures every X seconds**
void getTemp() {
  print_off();
  t10 = t1;
  t20 = t2;
  t30 = t3;
  t40 = t4;
  t50 = t5;

  sensorDS18B20.requestTemperatures();

  t1 = sensorDS18B20.getTempCByIndex(4);
  t2 = sensorDS18B20.getTempCByIndex(2);
  t3 = sensorDS18B20.getTempCByIndex(1);
  t4 = sensorDS18B20.getTempCByIndex(3);
  t5 = sensorDS18B20.getTempCByIndex(0);

  Serial.print(t1); Serial.print(";");
  Serial.print(t2); Serial.print(";");
  Serial.print(t3); Serial.print(";");
  Serial.print(t4); Serial.print(";");
  Serial.print(t5); Serial.println(";");
```

```
  reg = 1;
}

// **Function to display the register number on the 7-segment display**
void printDisplay(int nreg) {
  String ntext = (nreg <= 9) ? "0" + String(nreg) : String(nreg);
  for (int i = 0; i < 2; i++) {
    if (i == 0) write_D1();
    else if (i == 1) write_D2();

    int n = ntext[i] - '0'; // Convert Char to Int

    // **Call the correct function to print the digit**
    printDigit(n);
    delay(5);
  }
}

// **Function to write to display digit D1**
void write_D1() {
  digitalWrite(D1, LOW);
  digitalWrite(D2, HIGH);
}

// **Function to write to display digit D2**
```

```
void write_D2() {
  digitalWrite(D1, HIGH);
  digitalWrite(D2, LOW);
}

// **Function to print the correct digit on the display**
void printDigit(int n) {
  switch (n) {
    case 0: print_0(); break;
    case 1: print_1(); break;
    case 2: print_2(); break;
    case 3: print_3(); break;
    case 4: print_4(); break;
    case 5: print_5(); break;
    case 6: print_6(); break;
    case 7: print_7(); break;
    case 8: print_8(); break;
    case 9: print_9(); break;
  }
}
// **Function to display a 0 on the 7-segment display**
void print_0() { setSegments(HIGH, HIGH, HIGH, HIGH, HIGH, HIGH, LOW); }
// **Function to display a 1 on the 7-segment display**
void print_1() { setSegments(LOW, HIGH, HIGH, LOW, LOW, LOW, LOW); }
//** Function to display a 2 on the 7-segment display**
```

```
void print_2() { setSegments(HIGH, HIGH, LOW, HIGH, HIGH, LOW, HIGH); }
//** Function to display a 3 on the 7-segment display**
void print_3() { setSegments(HIGH, HIGH, HIGH, HIGH, LOW, LOW, HIGH); }
// **Function to display a 4 on the 7-segment display**
void print_4() { setSegments(LOW, HIGH, HIGH, LOW, LOW, HIGH, HIGH); }
//** Function to display a 5 on the 7-segment display**
void print_5() { setSegments(HIGH, LOW, HIGH, HIGH, LOW, HIGH, HIGH); }
//** Function to display a 6 on the 7-segment display**
void print_6() { setSegments(HIGH, LOW, HIGH, HIGH, HIGH, HIGH, HIGH); }
// **Function to display a 7 on the 7-segment display**
void print_7() { setSegments(HIGH, HIGH, HIGH, LOW, LOW, LOW, LOW); }
//** Function to display an 8 on the 7-segment display**
void print_8() { setSegments(HIGH, HIGH, HIGH, HIGH, HIGH, HIGH, HIGH); }
// **Function to display a 9 on the 7-segment display**
void print_9() { setSegments(HIGH, HIGH, HIGH, HIGH, LOW, HIGH, HIGH); }

// **Function to turn off all segments of the 7-segment display**
void print_off() {
  setSegments(LOW, LOW, LOW, LOW, LOW, LOW, LOW);
}

// **Helper function to set the segments for each digit**
void setSegments(int a, int b, int c, int d, int e, int f, int g) {
  digitalWrite(pinA, a);
  digitalWrite(pinB, b);
```

```
  digitalWrite(pinC, c);
  digitalWrite(pinD, d);
  digitalWrite(pinE, e);
  digitalWrite(pinF, f);
  digitalWrite(pinG, g);
}
```

Supplementary material 2

In the main text, discharge and flow velocity during the monitored hydropeak are discussed. These data were obtained using an acoustic Doppler current profiler (aDcp, River Surveyor Sontek® M9). The figure shows the Adcp measurement records, illustrating how discharge ranged between 2 and 7 m³/s and flow velocity between 0 and 2 m/s. These field measurements allow for a detailed analysis of the hydraulic behaviour during the hydropeaking event.

Below are the corresponding aDcp data plots, providing a visual representation of the spatial and temporal variability captured during the monitoring campaign before and during the hydropeaking: