Lab Report for Practical Measurement Electronics and Interfaces in Ocean Sciences (63-769)

Danai Filippou

March 2023

1. INTRODUCTION

In this report, we develop a data logger suitable to use with a DS18B20 temperature sensor. The DS18B20 is a versatile instrument that can be used in various conditions (e.g. in water). Here, we integrate it in a circuit that includes an Arduino board and a resistor. We calibrate the instrument in the Sea Ice Lab of the University of Hamburg and compare it to other sensors of the same type.

We describe the experiment (hardware and software) in Section 2. Section 3 includes our measurements and their analysis. We conclude with the discussion in Section 4.

2. COMPONENTS AND TECHNICAL DESCRIPTION

2.1. Hardware components

We use the following material:

- Arduino Uno R3
- Datalogger shield
- DS18B20 temperature sensor
- USB cable
- three jump cables
- 8GB SD card
- 4.7 kOhm resistor
- three WAGO 221 connectors

The complete circuit is depicted in Figure 1. As listed above, we use a waterproof DS18B20 temperature sensor that is powered by 5V through the Arduino. The sensor can be used in temperatures that range from -55°C to 125°C with an accuracy of ±0.5°C and its measurements have a resolution of 9-bit to 12-bit, based on the user's choice. It employs a one-wire bus protocol, which means that only one data line is needed to communicate with a microprocessor (in our case, the Arduino's microprocessor). Another significant component of our setup is the Arduino Uno R3, an easy-to-use and open source hardware and software platform that can be powered by a computer using a USB cable. The Arduino is connected to the temperature sensor and provides the microcontroller where the temperature data is read and processed. The data is stored in an 8GB SD card mounted on the Arduino's datalogger shield which also includes a Real-time clock (RTC). In order to ensure that a signal in the circuit remains in a known state, even if there is no input present (for example, when the circuit is not connected to a voltage source), a pull-up resistor is needed. This way, the resulting current is decreased and the Arduino ends up receiving more voltage through the data connection. Here, we use a 4.7 kOhm resistor which is a typical value for applications like ours as it provides a suitable voltage level on the signal line when no input signal is present. Finally, the three WAGO 221 connectors are used instead of a breadboard, to connect the resistor and the cables of the sensor and the Arduino.

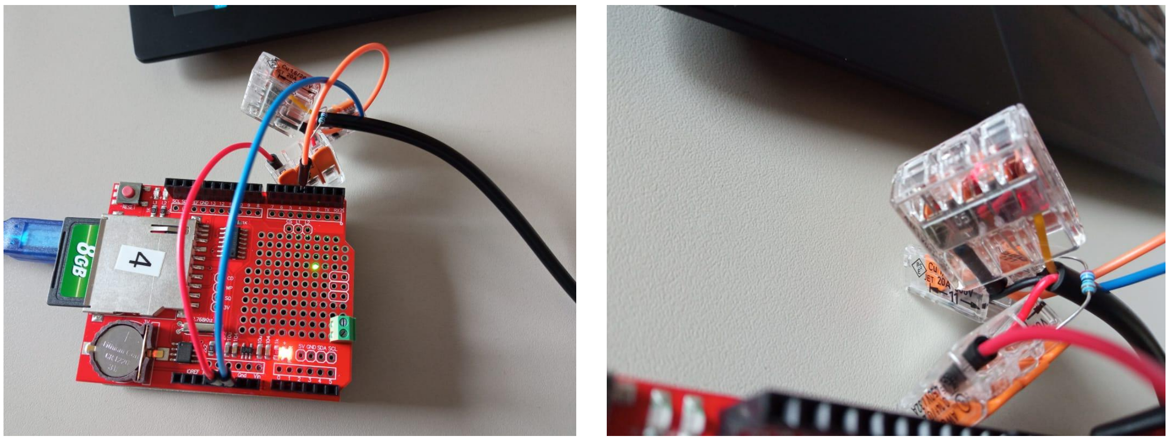

Figure 1: Photograph of the circuit and the wiring.

2.2 Wiring

Figure 2 shows our circuit's diagram. We connect the 5V power supply and the ground of Arduino to the temperature sensor. The sensor uses a 1-wire interface, which means that only one data pin of the Arduino board is required to establish communication between the two devices. We choose one of the digital pins of the Arduino to connect to our sensor. We place the 4.7 kOhm between the data connection and the 5V power supply.

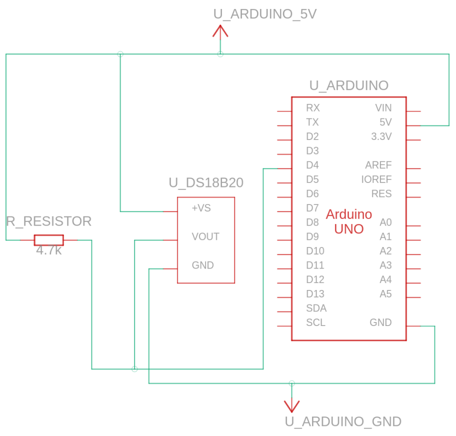

Figure 2: Circuit diagram (*created at https://www.tinkercad.com/*)

2.3 Software and code configuration

We use the Arduino IDE to write our code, compile it and upload it to the Arduino. In this section we provide a brief documentation of the code, that can be found on the corresponding github repository (https://github.com/DFilippou/CourseElectronicsWiSe22_23).

In the following code snippet, we first set the baud rate to 9600, so that the Arduino can establish communication with the computer through the USB cable. Afterwards, the Real Time Clock (RTC) and the SD card are initialized. The code is written in such a way that messages are produced, if these initializations are not completed successfully. If the RTC fails to set up, the current date and time are used instead.

```
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);

  if(!rtc.begin()) {
    Serial.println("RTC is NOT running :( Let's set the time now! :) ");
    rtc.adjust(DateTime(F(__DATE__),F(__TIME__)));
  }
  //rtc.adjust(DateTime(F(__DATE__),F(__TIME__)));
  if (!SD.begin()) {
    Serial.println("SD module initialization failed or Card is not present :( ");
    return;
  }
```

In the next lines, the OneWire library is used to initiate a temperature conversion. First, the code tests whether the connected temperature sensor is a DS18B20 sensor, and an error message is printed if that is not the case. To perform the test, the sensor's ROM code is read. Then, a command is sent to the sensor to initiate the temperature measurements conversion.

```
byte rom_code[8];
byte sp_data[9];
// Start 1st sequence to read out the rom code
ow.reset();
ow.write(READ_ROM);
for (int i=0; i<8; i++) {
  rom_code[i] = ow.read();
}
if(rom_code[0] != IS_DS18B20_SENSOR){
  Serial.println("Sensor is not a DS18B20 sensor! :( ");
}

String registration_number;
for (int i=1; i<7; i++) {
  registration_number += String(rom_code[i],HEX);
}

ow.reset();
ow.write(SKIP_ROM);
ow.write(CONVERT_T);
```

In the final part of the code, first the SKIP_ROM command is used to communicate with the sensor directly, and then the code reads the temperature data from the sensor using READ_SCRATCHPAD. The measurement is stored as a 16-bit integer and then it is converted to Celsius. Finally, the function `printOutput` is used to print the data in the desired format (timestamp,milliseconds,registration number, temperature in °C). In the last line, `delay(1000)` is used so that the code waits for one second before starting the loop again.

```
//Start sequence for converting temperatures
ow.reset();
ow.write(SKIP_ROM);
ow.write(READ_SCRATCHPAD);
for (int i=0; i<9; i++){
  sp_data[i] = ow.read();

}
int16_t tempRead = sp_data[1] << 8| sp_data[0];

float tempCelsius = tempRead / 16.0;

printOutput(getISOtime());
printOutput(", ");
printOutput((String)millis());
printOutput(", ");
printOutput(registration_number);
printOutput(", ");
printOutputln((String)tempCelsius);

delay(1000);
```

3. MEASUREMENTS

3.1 Sensor Calibration

In order to improve the accuracy of our temperature sensor, we need to calibrate it. To do so, we compare the measurements taken by our sensor to a reference, which we consider as the ideal response. From the characteristic curve (Figure 3a) we observe that:

- there is an offset to the entirety of the measured temperature range (3 - 20°C)
- this offset decreases as time goes by (Figure 3b), indicating that the performance of the sensor is slightly improving

As a result, a One Point Calibration is enough to calibrate our sensor. We take the average of the temperature offset and we conclude that we have an offset of -0.658°C. For the rest of the report, we subtract 0.658 °C from our sensor's measurements, in order to apply the calibration.

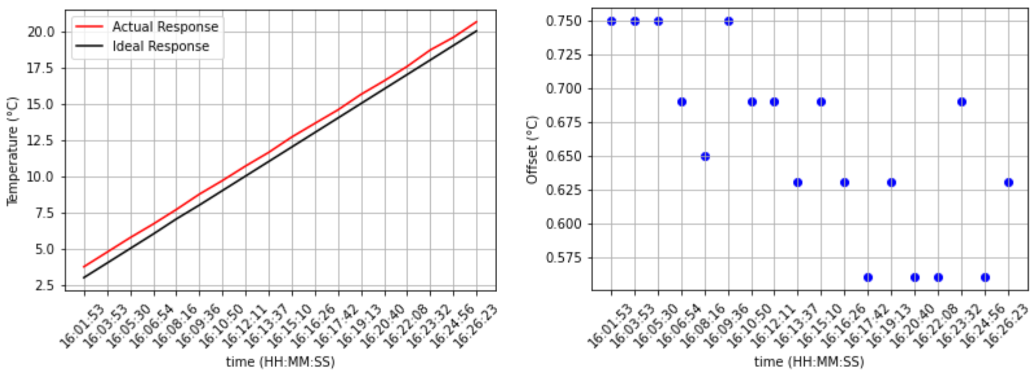

Figure 3: Calibration of our sensor using a reference sensor. a) An offset can be seen for all measurements. b) The offset slightly decreases over time

3.2 Spot and average measurements

Next, we attempt to reduce the noise of our sensor's measurements by taking the averages of the spot measurements of our experiment. Averaging minimizes the effect of random variations in temperature that can be caused by factors like electronic noise (generated by the circuit), aging of the experiment's components, or fluctuations that occur during the conversion from analogue to digital signal. The result can be seen in Figure 4. We notice that in the beginning of the measurements, before temperature falls below 5°C, there is a significant difference between the spot and averaged measurements. However, as temperature steadily starts to rise from 3°C up to 20°C the two measurement types start to converge. This is expected, since the temperature at that point follows an increasing linear trend.

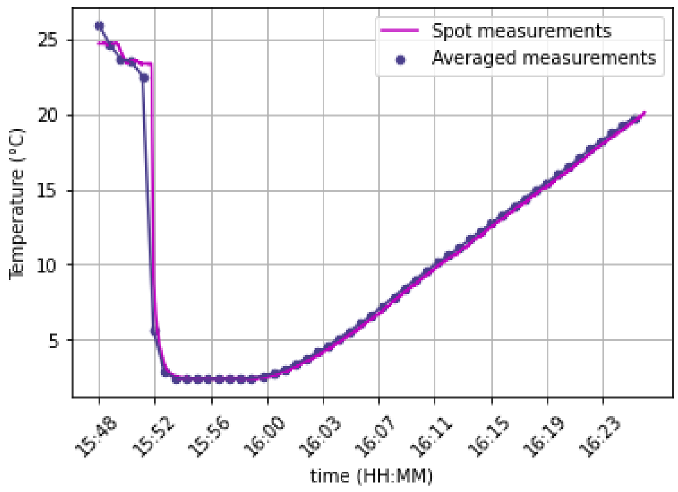

Figure 4: Spot and averaged measurements of our sensor

3.3 Calculation of Time Constant

The time constant is the time required for a sensor to adapt its output to 63.2% (1 - 1/e) of its final value. It is a quantity that is useful for our experiment since it is an indication of how fast or slow our temperature sensor reacts to a sudden change in its environment (for example, when placed inside water of low temperature). In our experiment, we assume that the response of the sensor to this change follows the equation:

$$T = T_m - (T_m - T_0)e^{-(t/\tau)} \qquad (1)$$

where T is the temperature measured by the sensor, T_m is the surrounding environment temperature, T_0 is the initial temperature, t is the time and τ is the time constant. If we rearrange this equation we get:

$$ln(\frac{T-T_m}{T_0-T_m}) = -\frac{t}{\tau} \qquad (2)$$

so we can calculate τ as the negative inverse of the slope in the $ln(\frac{T-T_m}{T_0-T_m})$ vs. t curve. We use T_0 = 24.63°C for our sensor, T_0 = 24.38°C for the reference sensor and T_m = 2.3°C , same for both sensors. Our calculations were done in LibreOffice Calc and resulted in τ_{sensor} = 109 sec and τ_{ref} = 49 sec. These numbers indicate that the reference sensor has a faster response rate than our sensor. Still, these values remain extremely high.

3.4 Comparison to other sensors

More than one temperature sensor was available in the lab, so as further evaluation, we proceed to compare the one we used to the rest. We plot all the temperature measurements from all the sensors in Figure 5. We observe that three sensors (s1,s3 and s8) have a delay in the temperature measurements. Since all of the sensors were placed in the water at the same time, we conclude that this offset stems from differences in the measuring timestamps across all the sensors, and not from differences in the sensors' accuracy.

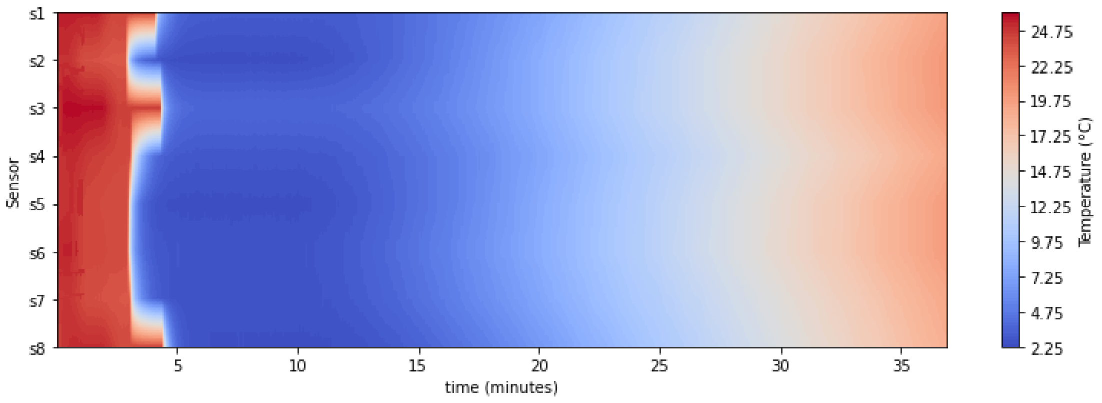

Figure 5: Comparison of all sensors used in the lab. The sensor we used is sensor number 1.

4. CONCLUSION

In this report we designed a data logger for the DS18B20 temperature sensor and we made some tests regarding the specifications of the system. The sensor calibration was straight-forward and easy to implement, since the ideal and the actual response had a similar linear trend. However, it is unclear whether the calibration improved the sensor's accuracy or not. Averaging was used to reduce the noise of the measurements. As indicated by Figure 4, the measurements at the beginning of the experiment (range 25 - 5°C) exhibit some fluctuations that are indeed smoothed out by the averaging process. In the temperature range 5-20°C there is no noise in the measurements, so averaging does not appear to improve the data quality. Finally, during the comparison of the sensors, it was evident that there was a drift in the measurements for some of the sensors. As pointed out in the previous section, this could be due to differences in the measuring timestamps of each instrument. In such an experiment it should be verified that all sensors are coordinated in time. If all sensors are correctly arranged, the measurements across all instruments could be averaged, and that could lead to even more accurate results.