Physik: Seminar

Arduino – Wetterstation

Vincent Piegsa
Flemming Esser
Finn Haack

Hamburg, den 19. September 2018

Inhaltsverzeichnis

1 Einleitung

Open-Source Projekte wie der Arduino sind in den letzten Jahren immer mehr an Popularität und Anwendungsmöglichkeiten gewachsen – Vor allem aber wurden Sensoren für alle möglichen Zwecke entwickelt. So ist es möglich, mit einfachen Programmen und Bauteilen, die für jeden zugänglich sind, eine Wetterstation zu kreieren, die präzise Messwerte ausgibt.

2 Materialien

Bauteil	Funktion
Arduino Mega 2560	Rechen- und Datenverarbeitungszentrum
DHT11	Erfassung der Temperatur und Luftfeuchtigkeit
DS3231	Zeitmessung
LCD	Grafische Darstellung der Daten
SD-Karten Modul	Speicherung der gesammelten Daten
SD-Karte	Speichermedium für die gesammelten Daten
Jumper-Kabel	Elektrische Verbindung zwischen den Bauteilen
Potentiometer	Einstellen des Kontrastes auf dem LCD
Widerstände	220 Ω für das LCD, 10 kΩ für den DHT11

Tab. 1: Materialien

3 Aufbau

3.1 Elektrische Schaltung

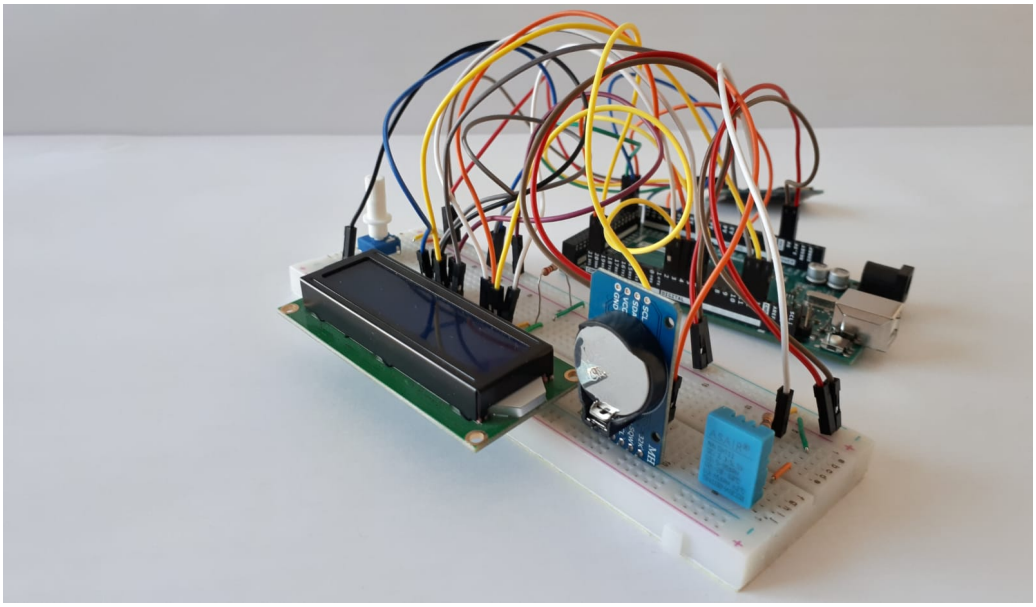


Abb. 1: Wetterstation

Bauteil	Bauteil-Pin	Arduino-Pin
DHT11	1: VCC	5 V
	2: DATA	10 kΩ zu 5 V; Digitalpin 7
	3: NC	nicht verbunden
	4: GND	GND
DS3231	1: 32k	nicht verbunden
	2: SQW	nicht verbunden
	3: SCL	SCL
	4: SDA	SDA
	5: VCC	5 V
	6: GND	GND
SD-Karten Modul	1: GND	GND
	2: VCC	5 V
	3: MISO	Digitalpin 50
	4: MOSI	Digitalpin 51
	5: SCK	Digitalpin 52
	6: CS	Digitalpin 6
LCD	1: LED–	GND
	2: LED+	5 V über 220 Ω
	3: DATABUS7	Digitalpin 2
	4: DATABUS6	Digitalpin 3
	5: DATABUS5	Digitalpin 4
	6: DATABUS4	Digitalpin 5
	7: DATABUS3	nicht verbunden
	8: DATABUS2	nicht verbunden
	9: DATABUS1	nicht verbunden
	10: DATABUS0	nicht verbunden
	11: E	Digitalpin 11
	12: R/W	GND
	13: RS	Digitalpin 12
	14: V0	Potentiometer
	15: VCC	5 V
	16: GND	GND

Tab. 2: Verknüpfung der Bauteile im Detail

Hinweis: Einen detaillierten Schaltplan sowie weitere Bilder von der Wetterstation und den einzelnen Sensoren befinden sich im Anhang unter Kapitel 7.4.

3.2 Programmierung

3.2.1 Grundaufbau

Arduino-Programme, so genannte Sketche, werden in einer vereinfachten Version von C++ geschrieben, die via USB auf das Board übertragen werden und dort vom Mikroprozessor aus-

geführt werden. Zur einfacheren Verwendung der Sensoren haben wir sogenannte Libraries (Bibliotheken) in den Code eingebunden, die bereits vorgefertigt von Plattformen wie z.B. GitHub oder direkt beim Hersteller heruntergeladen und in der Entwicklungsumgebung installiert werden können.
Bis auf einige Klassenbezeichnungen gleicht die Arduino-Syntax der von C++. Die Hauptbestandteile eines jeden Programms sind die setup- und loop-Schleife, die hier beschrieben werden:

```
void setup() {
  /* Code, der einmal am Start durchlaufen wird */
}
```

Abb. 2: Der Code innerhalb der setup-Schleife wird nur einmal am Start ausgeführt

```
void loop() {
  /* Code, der immer durchlaufen wird */
}
```

Abb. 3: Der Code innerhalb der loop-Schleife wird, sich ständig wiederholend, während der Laufzeit ausgeführt

3.2.2 Liquid-Crystal-Display (LCD)

```
// Einbinden der LCD-Bibliothek
#include <LiquidCrystal.h>

// Instanziieren der LCD-Klasse und Festlegung der Pins
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
```

Abb. 4: Initialisierung des LCDs

```
void setup() {
  // Pixelanzahl des LCDs festlegen
  lcd.begin(16, 2);

  // Koordinaten (x, y) auf dem Display setzen
  lcd.setCursor(0, 0);

  // Text bei den vorher festgelegten Koordinaten schreiben
  lcd.print("Hello World!");
}
```

Abb. 5: Auf dem LCD einen Beispieltext anzeigen lassen

```
// Funktion "displayTemperature", die ein Array (Liste) als Parameter nimmt
void displayTemperature(String* data) {

  // Beseitigen aller Inhalte auf dem Display
  lcd.clear();

  // Festlegen der Koordinaten, an denen der Text erscheinen soll
  lcd.setCursor(0, 0);
  // Text auf das Display schreiben
  lcd.print("Temperatur:");

  // Zweite Zeile auf dem Display
  lcd.setCursor(0, 1);
  // Anzeigen des dritten Elements des Arrays mit der Einheit Grad Celsius
  lcd.print(String(data[2]) + " 'C");
}
```

Abb. 6: Temperatur auf dem LCD anzeigen lassen

3.2.3 SD-Karten Modul

```
// Einbinden der erforderlichen Bibliotheken
#include <SD.h>
#include <SPI.h>

// CS-Pin am Arduino festlegen
const int chipSelect = 6;

void setup() {
  // Wenn keine Verbindung zur SD-Karte hergestellt werden kann
  if (!SD.begin(chipSelect)) {

    // Led anschalten
    digitalWrite(LED_BUILTIN, HIGH);

    // Auf dem Seriellen Monitor eine Fehlermeldung ausgeben
    Serial.println("Card initialization failed");
  }
}

```

Abb. 7: Initialisierung des SD-Karten Moduls

```
// Funktion "writeSD", die einen String als Parameter nimmt
void writeSD(String dataString) {

  // Initialisieren der SD-Klasse, Erstellen einer Datei und Festlegung des
    Modus, mit dem die Datei bearbeitet werden soll
  File dataFile = SD.open("datalog.txt", FILE_WRITE);

  // Wenn die Datei erfolgreich geoeffnet wurde
  if (dataFile) {

    // String "dataString" in die Datei schreiben
    dataFile.println(dataString);

    // Datei schliessen, um Fehler zu vermeiden
    dataFile.close();
  }

  // Wenn die datei nicht geoeffnet werden kann
  else {

    // Fehlermeldung auf dem Seriellen Monitor ausgeben
    Serial.println("Error opening datalog.txt");
  }
}
```

Abb. 8: Daten auf die SD-Karte schreiben

3.2.4 DHT11

```
// Einbinden der Bibliothek
#include <DHT.h>

// Definierung des Pins am Arduino
#define DHTPIN 13

// Definierung des DHT-Modells
#define DHTTYPE DHT11

// Instanziierung der DHT-Klasse mit der Modellnummer und dem Anschlusspin
DHT dht(DHTPIN, DHTTYPE);

```

Abb. 9: Initialisierung des DHT11 (Temperatur- & Feuchtigkeitssensors)

Die bisher programmierten Bauteile benötigten nur einige Bibliotheken, die nicht zusätzlich installiert werden müssen. Zur Nutzung des DHT11 und des DS3231 (siehe Kapitel 3.2.5) werden Bibliotheken verwendet, die nicht standardmäßig in der Arduino-IDE enthalten sind. Die Links zu den Bibliotheken befinden sich in Kapitel 3.2.6.

```
void loop() {
  // Ausgeben der Temperatur
  Serial.println(dht.readTemperature());

  // Ausgeben der Luftfeuchtigkeit
  Serial.println(dht.readHumidity());

  // 5 Sekunden warten
  delay(5000);
}
```

Abb. 10: Messen von Temperatur und Luftfeuchtigkeit

3.2.5 DS3231

```
// Einbinden der Bibliotheken
#include <Wire.h>
#include "RTClib.h"

// Instanziieren der RTC-Klasse
RTC_DS3231 rtc;

// Variable, die nach jedem Neustart auf "false" ist
bool syncOnFirstStart = false;

void setup() {

  // Wenn der Start des DS3231 nicht funktioniert
  if (!rtc.begin()) {

    // Fehlermeldung ausgeben
    Serial.println("Initializing RTC failed.");
  }

  // Wenn der DS3231 inaktiv (nicht an den Strom angeschlossen) war
  if (rtc.lostPower() || syncOnFirstStart) {

    // Meldung ausgeben, dass der DS3231 synchronisiert wird
    Serial.println("Due to a powerloss, the RTC is being synchronized.");

    // Einstellen des neuen Datums und der neuen Uhrzeit
    rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));
  }
}
```

Abb. 11: Initialisierung des DS3231 und automatische Synchronisation

```
// Funktion, die Zeit und Datum mithilfe des DS3231 ausgibt
void getDateTime() {

  // Instanziieren der now-Klasse, die das Datum und die Uhrzeit des DS3231
     beinhaltet
  DateTime now = rtc.now();

  // Datum, bestehend aus Tag, Monat und Jahr
  String date_ = String(now.day(), DEC) + "." + String(now.month(), DEC) +
    "." + String(now.year(), DEC);

  // Uhrzeit, bestehend aus Stunden, Minuten und Sekunden
  String time_ = String(now.hour(), DEC) + ":" + String(now.minute(), DEC)
    + ":" + String(now.second(), DEC);

  // Ausgeben von Datum und Uhrzeit
  Serial.println(date_ + " - " + time_);
}

```

Abb. 12: Ausgeben von Datum und Uhrzeit

3.2.6 Anmerkungen

In den vorherigen Unterkapiteln wurde gezeigt, wie die einzelne Programmierung der Module und Sensoren zusammengesetzt ist. Wie bereits in Kapitel 3.2.1 und 3.2.4 angesprochen, wurden zur Vereinfachung der Programmierung einiger Module öffentliche Bibliotheken verwendet, deren Quellen hier zusammengestellt sind:

Bibliothek	Quelle
DHT.h	`https://github.com/adafruit/DHT-sensor-library`
RTClib.h	`https://github.com/adafruit/RTClib`

Tab. 3: Quellen der genutzten Bibliotheken

Alle anderen Bibliotheken sind standardmäßig Teil der Arduino IDE und bereits vorinstalliert. Der gesamte Quellcode inklusive Kommentaren befindet im Anhang (Kapitel 7.1.1).

4 Durchführung

Nachdem die Wetterstation getreu dem Schaltplan in Kapitel 3.1 aufgebaut und das Programm über den USB-Anschluss auf den Arduino geladen wurde, entschlossen wir uns, eine Langzeitmessung durchzuführen, um die volle Funktionalität auszutesten. Als Messungsort diente der Physikraum 2, der während der Woche ausgewogen belegt war. Damit die Station nicht von äußerlichen Einwirkungen beeinflusst wird, haben wir sie auf einen Schrank gestellt, der in der Nähe einer Steckdose war. Da wir kein passendes USB-Netzteil hatten, wurde ein Netzteil mit 6 V und einer 2 1 mm-Buchse zur Verfügung gestellt, das mit dem dazugehörigen Anschluss am Arduino verbunden wurde. Die auf der SD-Karte gespeicherten Daten haben wir mit einem ebenfalls selbstgeschriebenen Python-Script in eine Datenbank geschrieben und visualisiert. Genaueres zum Programm befindet sich im Anhang unter Kapitel 7.2.

5 Beobachtung

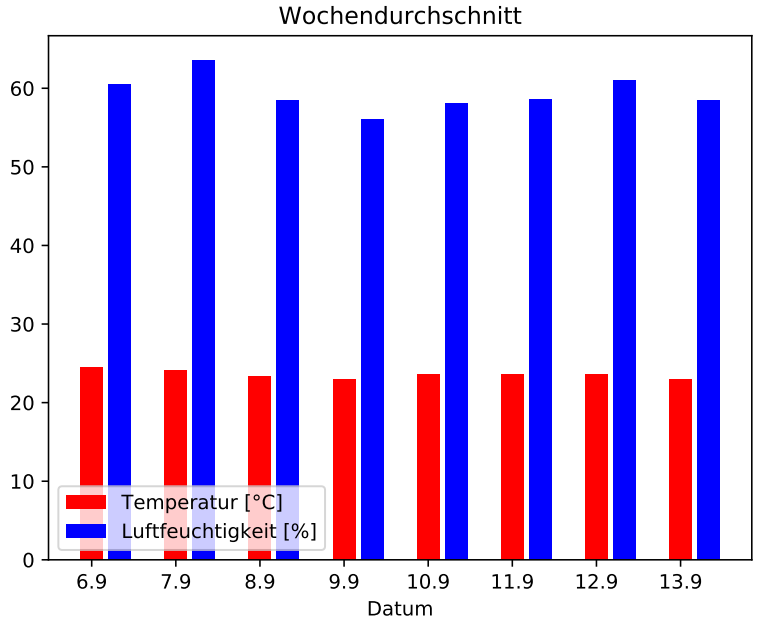


Abb. 13: Temperatur- und Luftfeuchtigkeitsdurchschnitt – Wochenverlauf [1]

Wie man dem Wochenverlauf entnehmen kann, fallen Temperatur und Luftfeuchtigkeit am Wochenende und steigen im Laufe der Woche wieder.

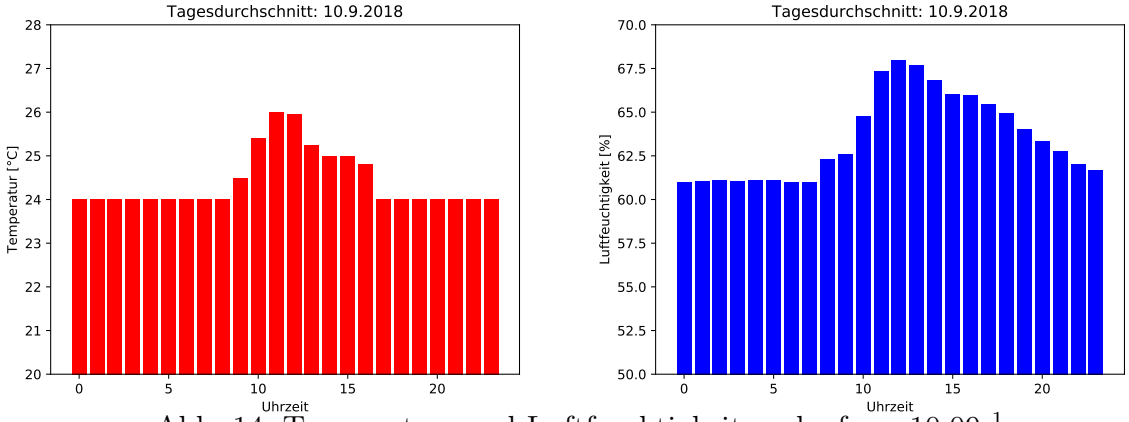


Abb. 14: Temperatur- und Luftfeuchtigkeitsverlauf am 10.09 [1]

Als Beispiel haben wir Montag, den 10.09 ausgewählt, da der Raum an diesem Tag fast vollständig belegt war (siehe Tabelle 5). Temperatur und Luftfeuchtigkeit beginnen zwischen 8 und 9 Uhr zu steigen, bevor sie um 12 Uhr ihren Höhepunkt erreichen und unterschiedlich schnell wieder sinken. Die Temperatur hat um ca. 17 Uhr wieder ihren Ausgangswert erreicht und hat sich nicht mehr verändert, wohingegen die Luftfeuchtigkeit noch gegen 23 Uhr etwas über dem Durchschnitt liegt. Während der Nacht bzw. am frühen Morgen verlaufen beide Werte nahezu konstant. Dieses Verhalten lässt sich auch bei anderen Werktagen feststellen – Temperatur und Feuchtigkeit beginnen je nach Wochentag ab 8 Uhr oder 9 Uhr zu steigen. Am Wochenende steigen sie ab 10 Uhr.

[1] Weitere Graphen befinden sich im Anhang unter Kapitel 7.3

Operator	Messung	Datierung
Durchschnitt (Temperatur)	23,5 °C	06.09 – 13.09
Max (Temperatur)	27 °C	07.09 – 11:13
Min (Temperatur)	23 °C	08.09 – 03:18
Durchschnitt (Luftfeuchtigkeit)	59,3 %	06.09 – 13.09
Max (Luftfeuchtigkeit)	68 %	06.09 – 13:12
Min (Luftfeuchtigkeit)	51 %	10.09 – 10:35

Tab. 4: Allgemeine Werte

Die hier in der Tabelle aufgelisteten Daten lassen auf erhöhte Luftfeuchtigkeit schließen, die sich im oberen Grenzbereich und darüber aufhält. Als Normwert gilt eine Feuchtigkeit zwischen 40 % und 60 %. Das kann unter anderem zu Schimmel führen, wenn nicht ausreichend gelüftet wird. Auch die Temperatur ist mit Spitzenwerten von 27 °C recht hoch, was sich aber nur negativ auf die Schüler auswirken könnte.

6 Auswertung

6.1 Temperaturverlauf

Man kann eine Verbindung zwischen Temperaturerhöhung und Benutzung des Raums erkennen: Da der Unterricht am Montag, Mittwoch und Donnerstag bereits um 7:50 Uhr beginnt, fängt auch die Temperatur an zu steigen – Demnach steigt sie am Dienstag und Freitag erst ab ca. 9 Uhr.

Uhrzeit	Montag	Dienstag	Mittwoch	Donnerstag	Freitag
7:50 – 8:35	8a		7d	S1	
8:35 – 9:20	8a		7d	S1	
9:40 – 10:25	8c	7e			7a
10:25 – 11:10		7e			7a
11:30 – 12:15	7b	7c	S1	8b	
12:15 – 13:00	7b	7c	S1	8b	
13:40 – 14:25	10fn	8c		NT8	
14:25 – 15:10	10fn			NT8	

Tab. 5: Raumbelegung Physik 2

Die Temperatur sinkt nach Unterrichtsschluss wieder, vermutlich gibt es einen Temperaturaustausch zwischen den Entropiepotentialen. Das heißt, dass aus dem Physikraum mit höherem Entropiegehalt die Entropie nach draußen fließt, bis die beiden Potentiale angeglichen sind – wie es der 2. Hauptsatz der Thermodynamik besagt. Dieser Austausch verläuft wahrscheinlich durch die Fenster, deren Isolierung nicht perfekt ist und Wärme entweichen lässt, oder auch durch die Tür.
Allerdings steigen Temperatur und Luftfeuchtigkeit auch am Wochenende, obwohl kein Unterricht stattfindet. Dies liegt vermutlich an der Sonneneinstrahlung, die erst zwischen 10 und 15 Uhr den Raum trifft.

6.2 Luftfeuchtigkeitsverlauf

Das scheinbar langsamere Abnehmen der Luftfeuchtigkeit lässt sich dadurch erklären, dass die gemessene, so genannte relative Luftfeuchtigkeit φ abhängig von der Temperatur ϑ ist: Je höher die Temperatur, desto höher ist auch die absolute Luftfeuchte f_{max}, also die maximale Menge an Wasserdampf, die in der Luft enthalten sein kann. Die Formel für die relative Luftfeuchtigkeit, die der Anteil der Luftfeuchte f von der absoluten Luftfeuchtigkeit ist, lautet:

$$\varphi = \frac{f}{f_{\max}} \cdot 100$$

Wenn f (die Menge des Wasserdampfes in der Luft) gleich f_{max} ist, beträgt die relative Luftfeuchtigkeit 100 %. Das ist die Höchstmenge an Wasserdampf, die die Luft mit einer bestimmten Temperatur aufnehmen kann. f_{max} berechnet sich mit:

$$f_{\max} = \frac{f}{\varphi \div 100}$$

Wenn man nun diese Formel für Temperaturen von 0 °C bis 100 °C anwendet und die relative Luftfeuchtigkeit misst, erhält man ungefähr diese Daten für f_{max} :

ϑ [°C]	f_{max} $[\frac{\mathrm{g}}{\mathrm{m}^3}]$
0	5
10	9
20	17
30	30
40	50
50	80
60	120
80	300
100	580

Die absolute Luftfeuchtigkeit nimmt also mit sinkender Temperatur ab, was bedeutet, dass die gleiche Menge an Wasserdampf bei unterschiedlichen Temperaturen eine unterschiedliche relative Luftfeuchtigkeit ergibt. Also kann es sein, dass ein Teil der Feuchtigkeit von den Wänden aufgenommen wurde, die Temperatur allerdings viel schneller gesunken war, sodass die relative Luftfeuchtigkeit nur geringfügig gesunken oder gleich geblieben ist.
Während der Benutzung des Raums produzieren die Schüler/Lehrer die Feuchtigkeit durch das Atmen oder Schwitzen. Für die Veränderung am Wochenende gibt es aber eine andere Eklärung: Im Laufe des Tages erhitzt sich die Luft innerhalb des Raums. Die Wände erwärmen sich langsamer als die Luft, und somit bildet sich eine Temperaturdifferenz. Mit zunehmender Differenz geht die in den Wänden gespeicherte Feuchtigkeit in die Luft über. Es ist derselbe Effekt wie beim Abkühlen: Das Steigen der Luftfeuchtigkeit wird durch das parallele Wachsen von f_{max} kompensiert. Dieser Prozess läuft solange, bis entweder die Wände trocken sind oder die Luft keine Feuchtigkeit mehr aufnehmen kann.

6.3 Verbesserungsmöglichkeiten

Um unsere Theorien zu bestätigen, müssten wir mehrere Sensoren zu unserer Messung hinzufügen. Einerseits bräuchten wir ein Gerät, das die Feuchtigkeit in den Wänden erfasst. Außerdem

wäre es nützlich, die Außentemperatur zu messen oder die Daten einer bereits bestehenden Wetterstation als Referenzwert zu nehmen. Somit könnten wir den Übergang von Luftfeuchtigkeit und Temperatur zwischen Raumluft und Wand bzw. Raumluft und Außenwelt nachvollziehen.

Der Temperatur- und Luftfeuchtigkeitsanstieg während der Woche im Vergleich zum Wochenende wäre vermutlich noch viel größer als von der Wetterstation gemessen, da der Unterricht höchstwahrscheinlich bei offenem Fenster stattgefunden hat oder zwischendurch gelüftet wurde. Somit ist der direkte Vergleich nur bedingt möglich, da diese Option bei der Messung leider nicht berücksichtigt wurde.

Die gesammelten Daten weisen vermutlich einen Fehler auf: Die Temperatur ist nie kleiner als 23 °C und verändert sich kaum. Nach weiterem Testen zeigte sich aber, dass der Sensor einwandfrei funktioniert und auch niedrigere Temperaturen erfassen kann, wie vom Hersteller angegeben. Die Ursache ist daher weiter unbekannt.
Einige Verbesserungen an unserer Wetterstation wären z.B ein zweiter Temperatur- bzw. Feuchtigkeitssensor, wenn möglich von einem anderen Bautyp, um typbedingte Messfehler oder den Ausfall eines Sensors zu kompensieren.
Außerdem wäre eine Datenübertragung über WLAN effizienter als die Speicherung auf eine SD-Karte, da man damit in Echtzeit die Daten in einer Webanwendung visualisieren und auswerten kann und nicht die Gefahr besteht, dass der Speicherplatz ausgeht. Das wäre z.B mit einem zusätzlichen ESP8266-01 oder dem Ersetzen des Arduino Mega mit dem ESP32 möglich, der WLAN, Bluetooth und eine Uhr integriert hat.
Zudem würden wir ein OLED-Display einem LCD vorziehen, da es weniger Strom verbraucht und auch Nicht-ASCII-Zeichen darstellen kann.

7 Anhang

7.1 Programme

7.1.1 Wetterstation.ino

```
/*
 * Titel: Wetterstation Rev 1.1
 * Autor: Vincent Piegsa
 * Datum: 28.08.2018
 * Beschreibung:
 *  - Wetterstation, basierend auf dem Arduino Mega 2560
 *  - Temperatur- und Feuchtigkeitsmessung mit dem DHT11
 *  - Speicherung der Daten auf einer SD-Karte
 *  - Genaue Datierung mit dem DS3231
 *  - Echtzeit-Visualisierung mit einem LCD
 */

// Einbinden aller benoetigten Bibliotheken
#include <Wire.h>
#include "RTClib.h"
#include <DHT.h>
#include <LiquidCrystal.h>
#include <SD.h>
#include <SPI.h>

// Festlegen von DHT-Anschlusspin und DHT-Modell
#define DHTPIN 13
#define DHTTYPE DHT11

// Instanziierung der DHT-Klasse
DHT dht(DHTPIN, DHTTYPE);

// Instanziierung des DS3231
RTC_DS3231 rtc;

// Instanziierung des LCDs und Festlegung der Daten-Pins
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

//CS-Pin des SD-Karten Moduls
const int chipSelect = 6;

// Variablen, die fuer das Timing des LCDs gebraucht werden
int lcd_timer = 0;
int sd_timer = 0;
int lcd_page = 0;

// automatische Synchronisation des DS3231
// nach jedem Neustart auf false
bool syncOnFirstStart = false;

// Funktion, die Daten in ein Array einsetzt
void formatArray(String* arr) {
  // Instanziieren der Uhrzeit und des Datums
  DateTime now = rtc.now();
```

Wetterstation.ino

```
  // Erfassen von Zeit und Datum
  String time_ = String(now.hour(), DEC) + ":" + String(now.minute(), DEC)
   + ":" + String(now.second(), DEC);
  String date_ = String(now.day(), DEC) + "." + String(now.month(), DEC) +
   "." + String(now.year(), DEC);

  // Messen der Temperatur
  String tmp = String(dht.readTemperature());

  // Messen der Luftfeuchtigkeit
  String hdt = String(dht.readHumidity());

  // Einsetzen der Werte in das Array
  arr[0] = date_;
  arr[1] = time_;
  arr[2] = tmp;
  arr[3] = hdt;
}

// Funktion, die Uhrzeit und Datum auf dem LCD anzeigt
void displayDatetime(String* data) {
  // Bereinigen aller Daten auf dem LCD
  lcd.clear();

  // Koordinaten (x, y) auf dem LCD festlegen (erste Zeile)
  lcd.setCursor(0, 0);
  // Element 1 aus dem Array (Datum) auf dem LCD anzeigen lassen
  lcd.print(data[0]);

  // Zweite Zeile auf dem LCD
  lcd.setCursor(0, 1);
  // Element 2 aus dem Array (Zeit) auf dem LCD anzeigen lassen
  lcd.print(data[1]);;
}

//Funktion, die die Temperatur auf dem LCD anzeigt
void displayTemperature(String* data) {
  // Bereinigen aller Daten auf dem LCD
  lcd.clear();

  // Koordinaten (x, y) auf dem LCD festlegen (erste Zeile)
  lcd.setCursor(0, 0);
  // Text auf dem LCD (erste Zeile) anzeigen
  lcd.print("Temperatur: ");

  // Zweite Zeile auf dem LCD
  lcd.setCursor(0, 1);
  // Element 3 aus dem Array (Temperatur) und Einheit [Grad Celsius] auf
   dem LCD anzeigen lassen
  lcd.print(String(data[2]) + "'C");
}

// Funktion, die die Feuchtigkeit auf dem LCD anzeigt
void displayHumidity(String* data) {
  // Bereinigen aller Daten auf dem LCD
```

Wetterstation.ino

```
  lcd.clear();

  // Koordinaten (x, y) auf dem LCD festlegen (erste Zeile)
  lcd.setCursor(0, 0);
  // Text auf dem LCD (erste Zeile) anzeigen
  lcd.print("Feuchtigkeit: ");

  // Zweite Zeile auf dem LCD
  lcd.setCursor(0, 1);
  // Element 4 aus dem Array (Luftfeuchtigkeit) in [%] auf dem LCD anzeigen
    lassen
  lcd.print(String(data[3]) + "%");
}

// Funktion, die die Daten als [csv] Format und ohne Einheit abspeichert
String formatSD(String* data) {
  // Variable, die spaeter die formatierten Daten enthaelt
  String dataString = "";

  // Durchlaufen aller vier Elemente im Array
  for (int i = 0; i < 4; i++) {
    switch (i) {
      // 1. Element
      case 0:
        // Datum formatieren
        dataString += "\n" + data[i] + ";";
        break;
      // 2. Element
      case 1:
        // Uhrzeit formatiern
        dataString += data[i] + ";";
        break;
      // 3. Element
      case 2:
        // Temperatur formatieren
        dataString += data[i] + ";";
        break;
      // 4. Element
      case 3:
        // Luftfeuchtigkeit formatieren
        dataString += data[i] + ";";
        break;
    }
  }

  // Zurueckgeben der formatierten Daten
  return dataString;
}

// Funktion, die die Daten auf die SD-Karte schreibt
void writeSD(String dataString) {
  // Oeffnen der Datei "datalog.txt" im "WRITE" (schreiben) Modus
  File dataFile = SD.open("datalog.txt", FILE_WRITE);

  // Wenn die Datei geoeffnet wurde
```

Wetterstation.ino

```
  if (dataFile) {
    // Daten in die Datei schreiben
    dataFile.println(dataString);

    // Datei schliessen
    dataFile.close();
  }

  // Wenn die Datei nicht geoeffnet wurde
  else {
    // Fehlermeldung ausgeben
    Serial.println("Error opening datalog.txt");
  }
}

// Schriftug, der beim Starten des Arduinos angezeigt wird
void lcd_begin() {
  // Koordinaten auf dem LCD setzen
  lcd.setCursor(0, 0);
  // Text anzeigen lassen
  lcd.print("Wetterstation");

  // Zweite Zeile auf dem LCD
  lcd.setCursor(0, 1);
  // Text anzeigen lassen
  lcd.print("Vincent Piegsa");
}

void setup() {
  // Initialisieren der Seriellen Verbindung
  Serial.begin(9600);

  // Initialisieren des DS3231
  // Wenn die Initialisierung nicht funktioniert hat
  if (! rtc.begin()) {
    // Fehlermeldung ausgeben
    Serial.println("initializing RTC failed");
  }

  // Automatische Synchronisierung des DS3231
  if (rtc.lostPower() || syncOnFirstStart) {
    // Meldung ausgeben, dass der DS3231 synchronisiert wird
    Serial.println("Due to a powerloss, the RTC is being synchronized.");
    // Datum und Uhrzeit einstellen
    rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));
  }

  // Initialisieren des SD-Karten Moduls
  // Wenn die Initialisierung nicht funktioniert hat
  if (!SD.begin(chipSelect)) {
    // Led anschalten
    digitalWrite(LED_BUILTIN, HIGH);
    // Fehlermeldung ausgeben
    Serial.println("Card initialization failed.");
```

Wetterstation.ino

```
  }

  // Initialisierung des DHT11
  dht.begin();

  // Initialisierung des LCDs und Festlegung der Dimensionen (16 Spalten, 2
     Zeilen)
  lcd.begin(16, 2);

  // Schrifzug beim "Hochfahren"
  lcd_begin();
}

void loop() {
  // Automatische Erhoehung der Timer-Variablen
  lcd_timer += 1;
  sd_timer += 1;

  // Erstellung eines Arrays mit 4 Elementen
  String data[4] = {};

  // Einsetzen der Daten in das Array
  formatArray(data);

  // Wenn die Timer-Variable gleich 20000 ist (~1 Minute)
  if (sd_timer == 20000) {
    // Daten formatieren
    String dataString = formatSD(data);

    // Daten auf dem Seriellen Monitor ausgeben
    Serial.println("Writing to SD-Card...\n\t" + dataString);

    // Daten auf die SD-Karte schreiben
    writeSD(dataString);

    // Zuruecksetzen des SD-Karten-Timers
    sd_timer = 0;
  }

  // Wenn die Timer-Variable gleich 500 ist (~ 1 Sekunde)
  if (lcd_timer == 500) {

    // Mehrere Seiten auf dem LCD, die unterschiedlichen Inhalt haben
    switch(lcd_page) {
      case 0:
        // Datum und Zeit anzeigen
        displayDatetime(data);

        // naechste Seite
        lcd_page += 1;
        break;
      case 1:
        // Temperatur anzeigen
        displayTemperature(data);
```

Wetterstation.ino

```
        // naechste Seite
        lcd_page += 1;
        break;
      case 2:
        // Luftfeuchtigkeit anzeigen
        displayHumidity(data);

        // Anfangsseite
        lcd_page = 0;
        break;
    }

    // LCD-Timer zuruecksetzen
    lcd_timer = 0;
  }

}
```

Wetterstation.ino

7.2 Daten

Da die Menge der Daten viel zu groß für dieses Dokument ist (über 11500 Zeilen), habe ich die Datei auf GitHub unter meinem Projekt hochgeladen. Dort finden Sie außerdem das selbstgeschriebene Python-Script, mit dem wir die Daten ausgewertet und visualisiert haben, sowie die Graphen und die Datenbank. Der Link zur Webseite:

`https://github.com/VincentPiegsa/weatherstation/tree/master/Messung`

7.3 Visualisierte Daten

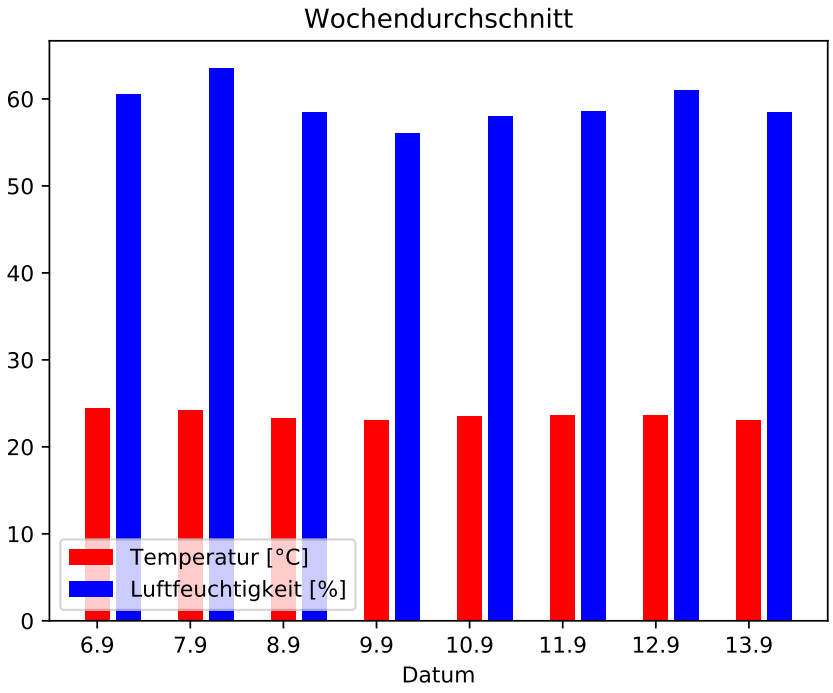


Abb. 15: Temperatur und Luftfeuchtigkeit – Wochenverlauf

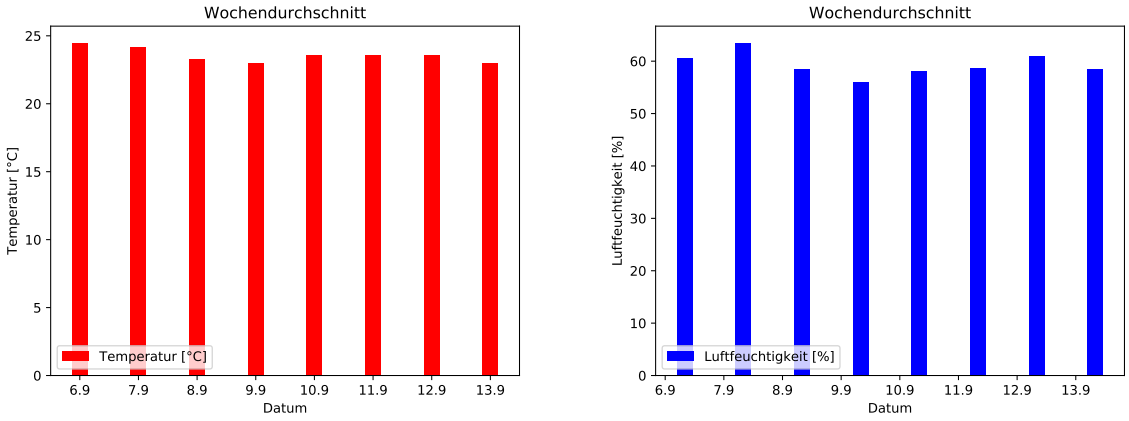


Abb. 16: Temperatur und Luftfeuchtigkeit – Wochenverlauf

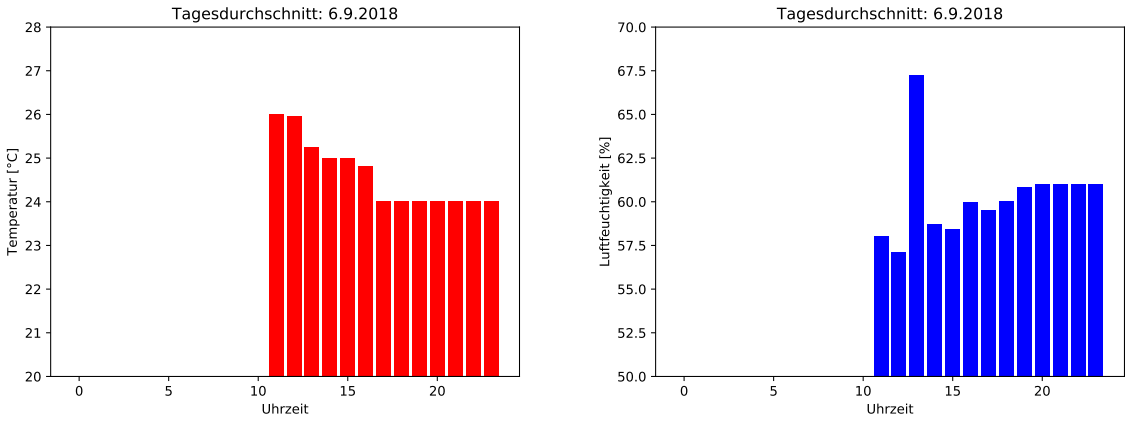


Abb. 17: Temperatur und Luftfeuchtigkeit – 06.09.2018

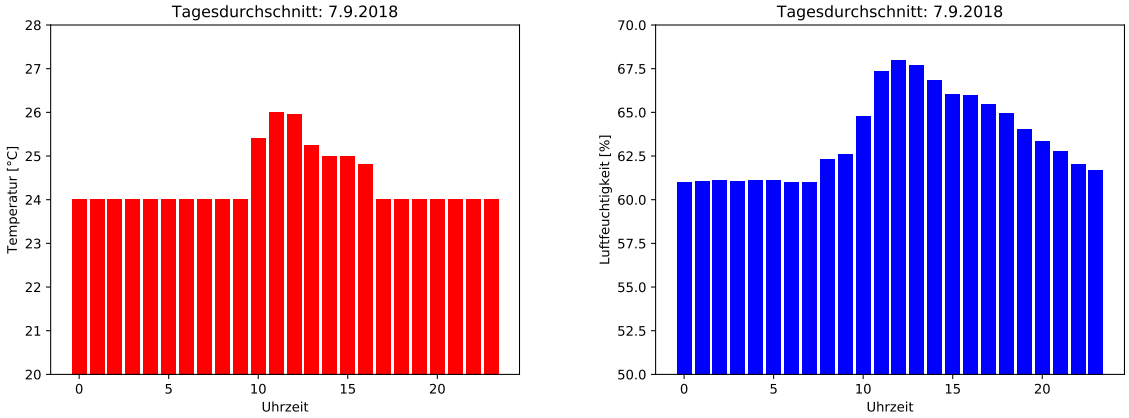


Abb. 18: Temperatur und Luftfeuchtigkeit – 07.09.2018

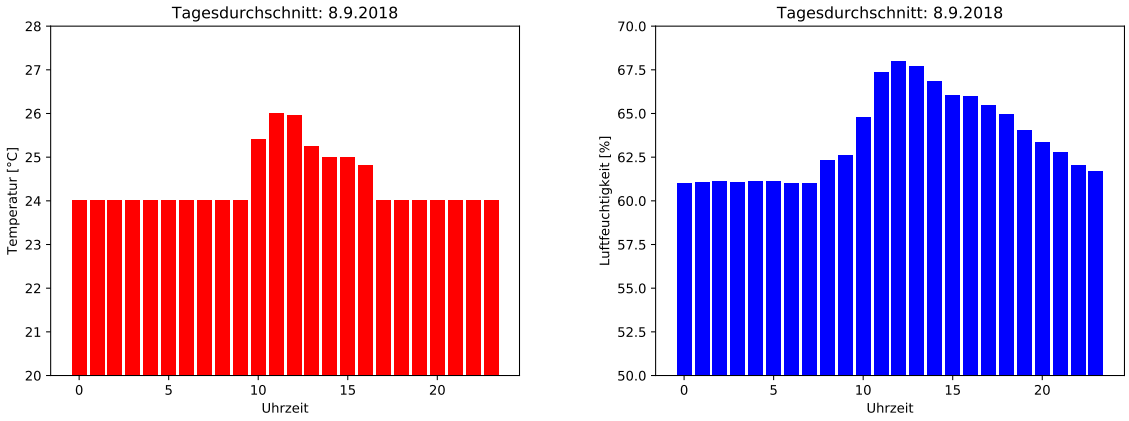


Abb. 19: Temperatur und Luftfeuchtigkeit – 08.09.2018

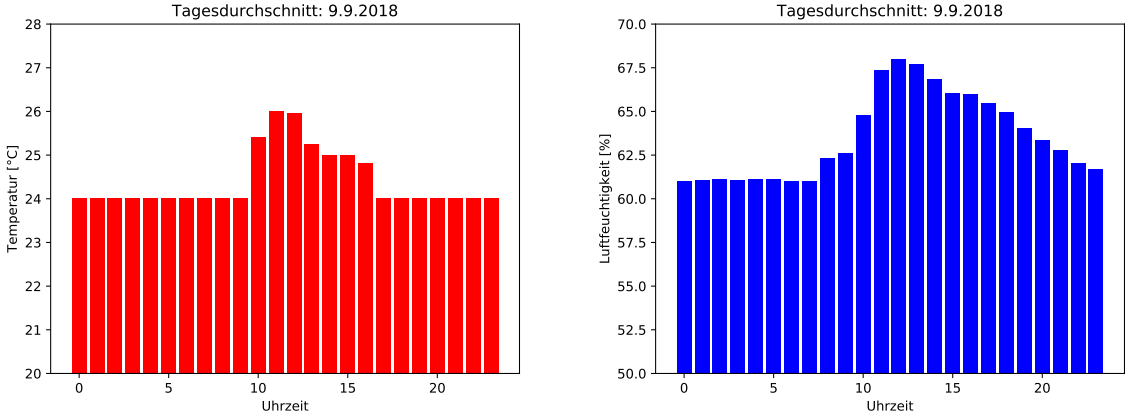


Abb. 20: Temperatur und Luftfeuchtigkeit – 09.09.2018

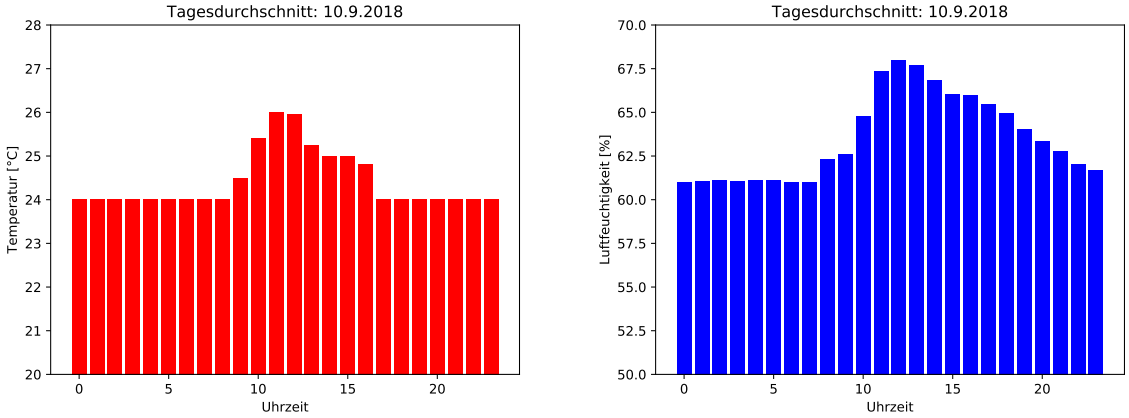


Abb. 21: Temperatur und Luftfeuchtigkeit – 10.09.2018

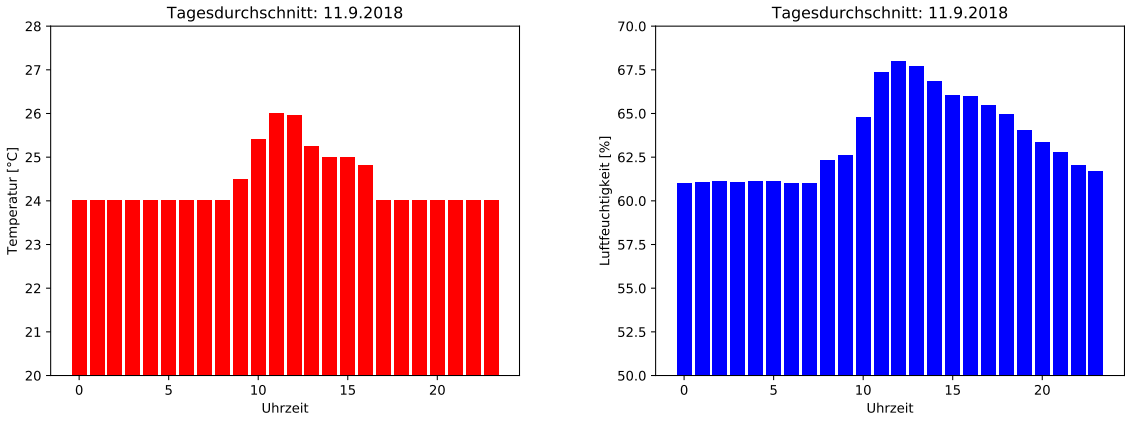


Abb. 22: Temperatur und Luftfeuchtigkeit – 11.09.2018

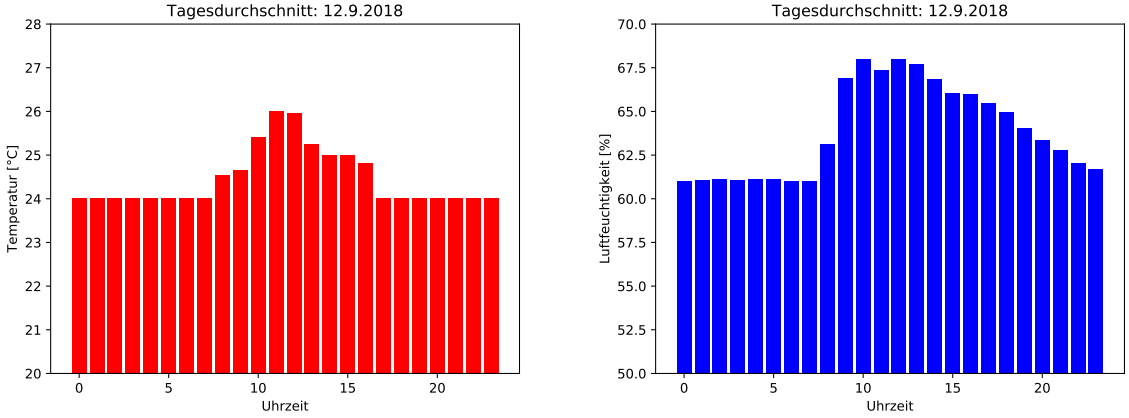


Abb. 23: Temperatur und Luftfeuchtigkeit – 12.09.2018

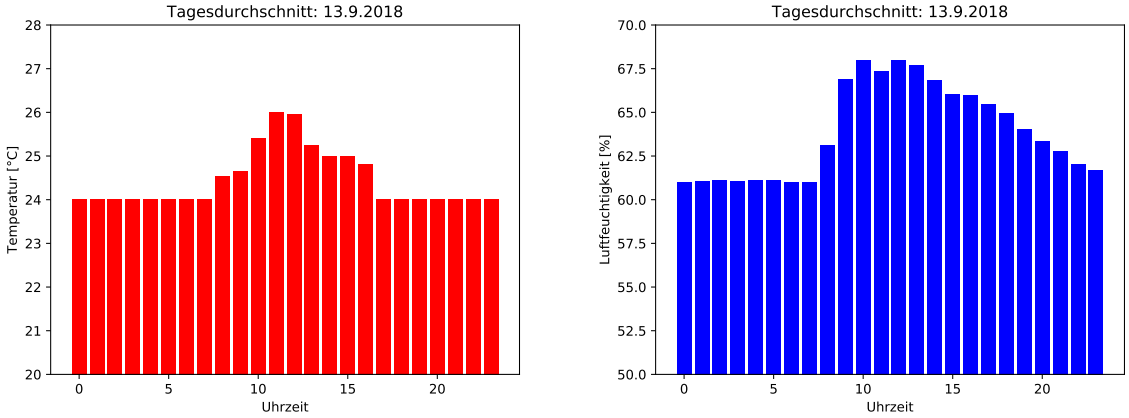


Abb. 24: Temperatur und Luftfeuchtigkeit – 13.09.2018

7.4 Bilder

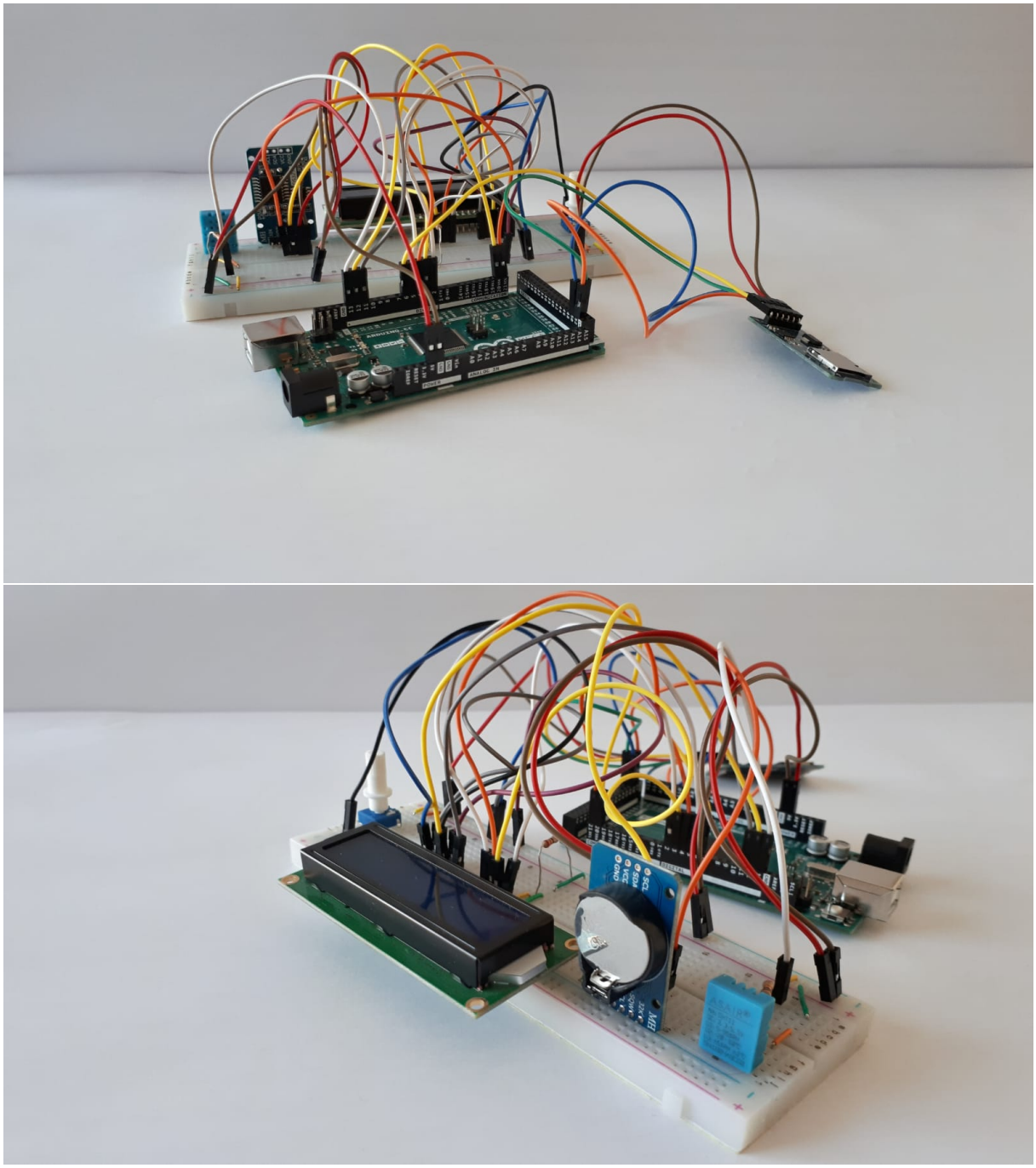


Abb. 25: Wetterstation

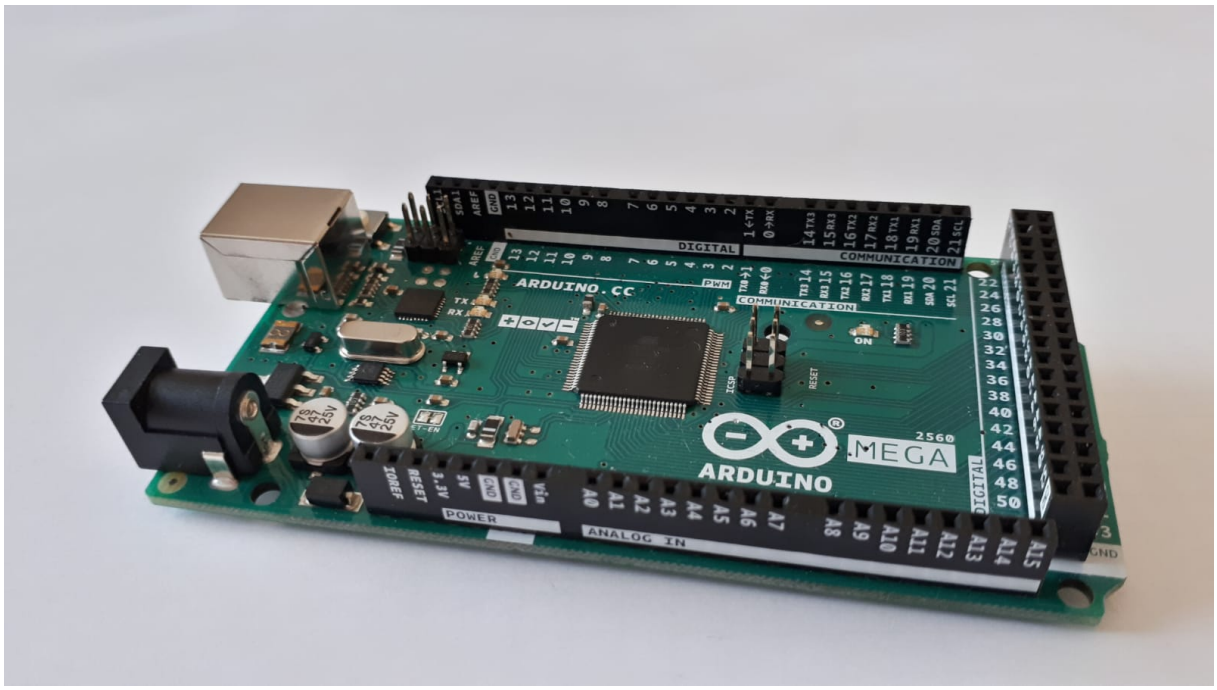
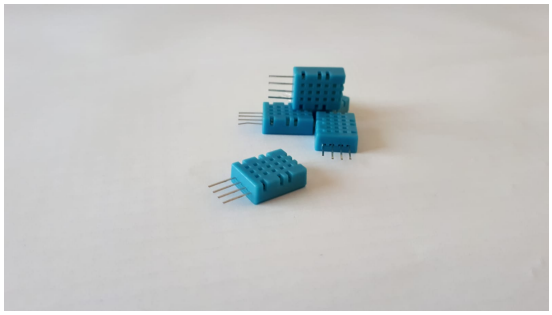


Abb. 26: Arduino Mega 2560



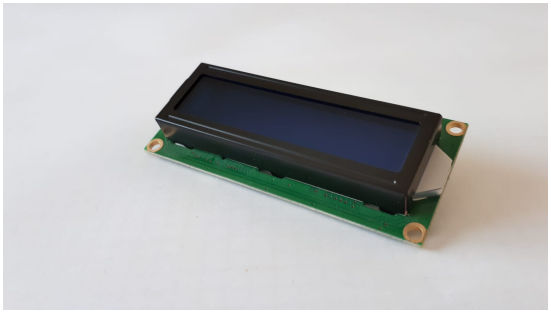
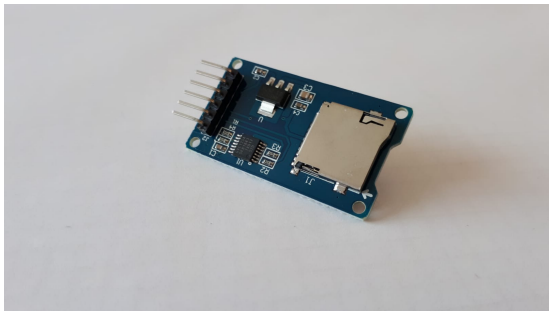


Abb. 27: DHT11 (Temperatur- & Luftfeuchtigkeitssensor) und LCD



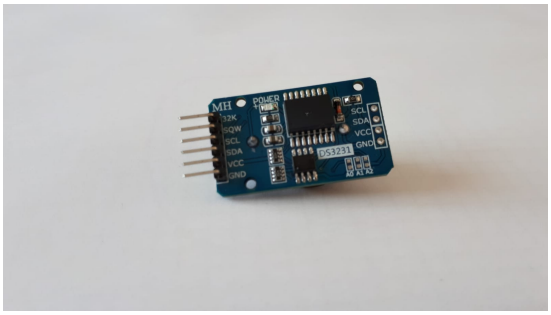


Abb. 28: SD-Karten Modul und DS3231 (elektronische Uhr)

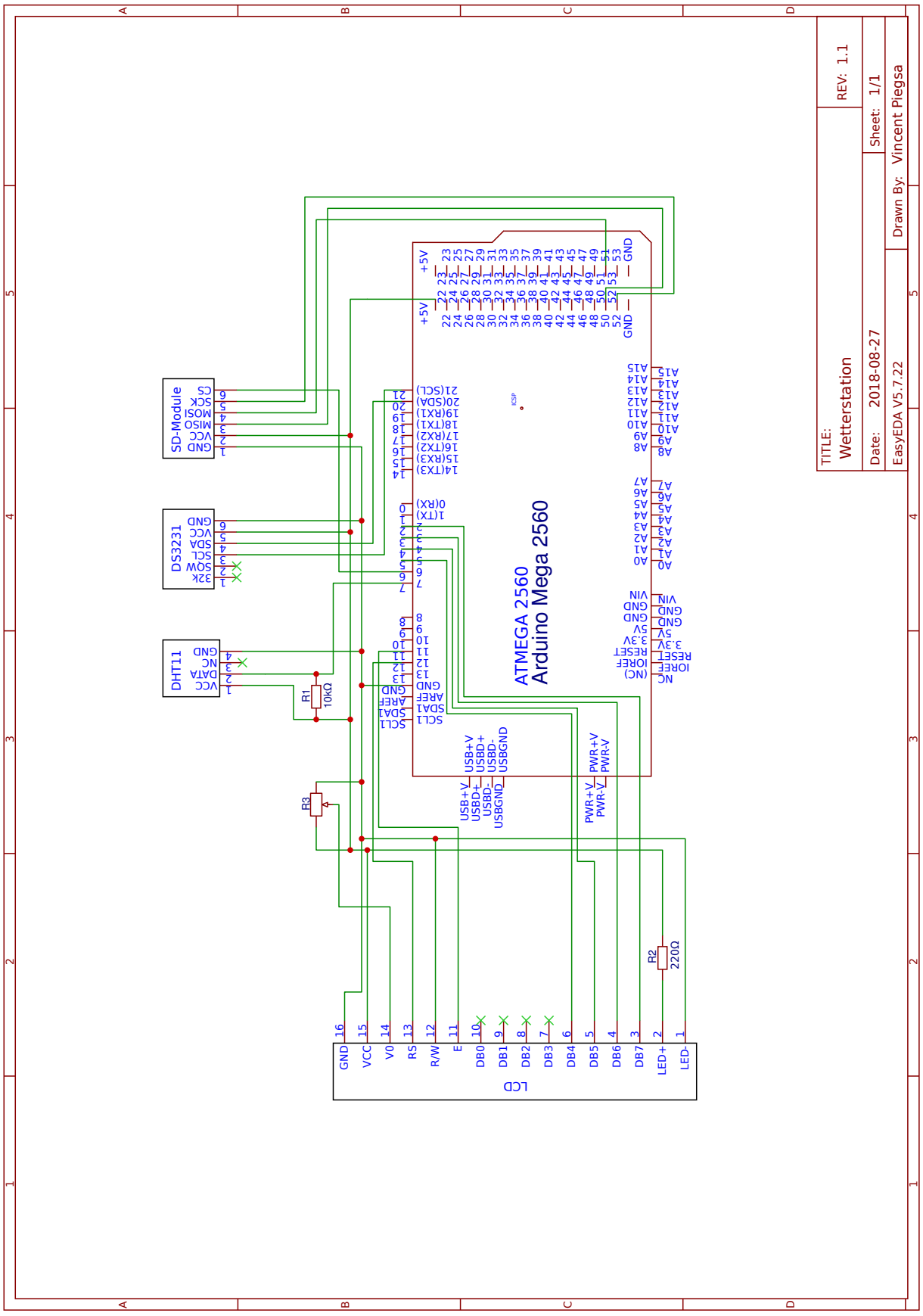


Abb. 29: Schaltplan