FHV
Vorarlberg University
of Applied Sciences

SCHLAUBEET - ein Flying-Classroom-Projekt

Kurzanleitung zum Workshop

Version 1.0.2

File: /atlantis/t/tebe/flying_classroom/fcr.experimente/fcr.schlaubeetE/dox/fcr.schlaubeetWS.kurzanleitung.odt

Versionsgeschichte

Version	Datum	Änderung	Autor
1.0.1	2024-05-15	Aktualisierung der Teileliste.	Franz Geiger
1.0.0	2024-05-10	Erstversion.	Franz Geiger

Inhaltsverzeichnis

1 Einleitung 1
2 Aufgabenstellung 3
3 Anleitung zum Aufbau der Hardware 5
3.1 Mechanisch-elektrischer Aufbau 5
3.2 Programmierung und Inbetriebnahme 5
4 Software 9
4.1 Applikation 9
4.1.1 Voraussetzungen 9
4.1.2 Software (schlaubeetWs1.ino) 9

Tabellenverzeichnis

Tab. 2.1: Teileliste SCHLAUBEET.....4
Tab. 4.1: Wertebereiche für verschiedene Varianten des Feuchtesensors.....11

Abbildungsverzeichnis

Fig. 2.1: Schlaubeet aufgebaut und betriebsbereit.....3
Fig. 3.1: Verdrahtungsplan für SCHLAUBEET – bitte die Orientierung des Arduino und des Breadboards beachten!.....7
Fig. 3.2: Verdrahtungsplan für SCHLAUBEET – Arduino Uno herausgezoomt.....8
Fig. 4.1: Neues File wie von der Arduino IDE defaultmäßig angelegt.....12

1 Einleitung

Dieses Booklet begleitet den Workshop *SCHLAUBEET - ein Flying-Classroom-Projekt* und widmet sich dabei der Software *Schlaubeet*.

SCHLAUBEET - ein Flying-Classroom-Projekt ist ein Workshop zum Thema Elektronik und Informationstechnologie des Programms *Flying Classroom* der FHV.

Wie ist dieses Booklet strukturiert? Es geht gleich los mit Aufgabenstellung, Hardware und Software. Im Anhang finden alle interessierten Maker vertiefendes Material, das für die Durchführung des Workshops erst mal nicht notwendig ist.

Teile dieses Dokuments sind farblich besonders gekennzeichnet:

- Gelb hinterlegter Text kennzeichnet Dinge, auf die du ganz besonders achten musst.
- `Grau hinterlegter Text kennzeichnet Software, d.s. die Programme, die du schreibst.`

2 Aufgabenstellung

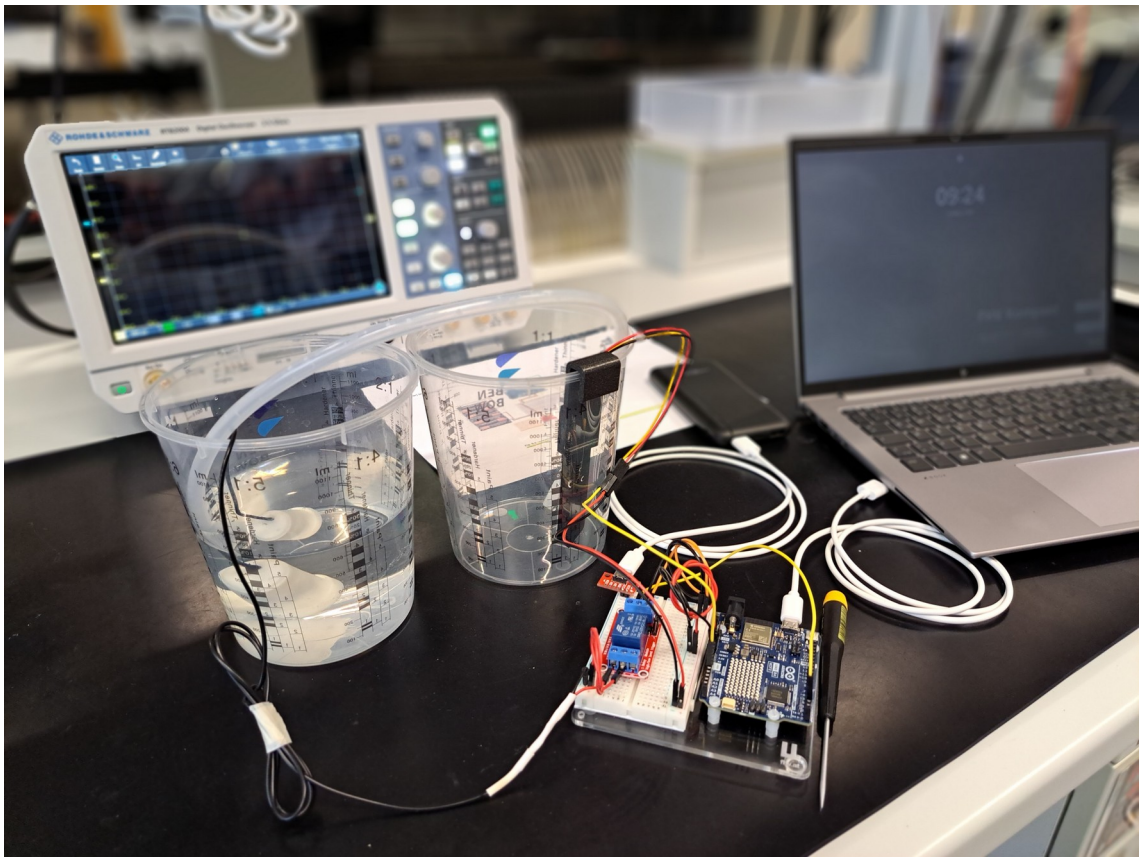

Fig. 2.1: Schlaubeet aufgebaut und betriebsbereit

Deine Aufgabe ist der Bau einer Gartenbeetbewässerung. Das heißt, du baust die Elektronik auf, die dazu notwendig ist programmierst sie so, dass das Ganze dann auch tut, was es soll: Wenn das Gartenbeet zu trocken ist, soll es bewässert werden.

In der Box zum Workshop findest du folgende Teile:

Pos.	Anz.	Verfügbar	Teil
1	1	x	Basisplatte (Kunststoff, transparent, mit FHV-Logo)
2	1	x	Arduino Uno R4 Wifi
3	1	x	Breadboard
4	11	x	Jumper-Wires
5	1	x	Feuchtesensor analog
6	1		Temperatursensor (BME688)
7	1		Qwiic Kabelsatz

Pos.	Anz.	Verfügbar	Teil
8	1	x	Pumpe (5V)
9	1	x	Kabelverbinder für die MAsseleitung der Pumpw
10	1	x	Relais 1-kanalig
11	1	x	Kabelverbinder
12	1	x	Schlauch
13	2	x	Messbecher
14	1	x	Powerbank
15	1	x	USB-C-Kabel für den Arduino
16	1	x	USB-Kabel mit Adapter für die Wasserpumpe
17	1	x	Schraubendreher
Für alle Gruppen steht zur gemeinsamen Nutzung bereit:			
18	1		Messgerät mit Messspitzen
19	1		Rolle Küchenpapier

Tab. 2.1: Teileliste SCHLAUBEET

Wie du am besten vorgehst, findest du in folgendem Kap. 3, Anleitung zum Aufbau der Hardware.

3 Anleitung zum Aufbau der Hardware

Die Liste der Teile, die du für den Aufbau benötigst, findest du in Kap. 2, Aufgabenstellung. Die Teile sind mit einem `x` gekennzeichnet.

Bevor du mit dem Aufbau beginnst, achte darauf, dass die Elektronik stromlos ist. Trenne also den Arduino von deinem PC!

3.1 Mechanisch-elektrischer Aufbau

- Verbinde den **Feuchtigkeitssensor** elektrisch über das Breadboard mit dem Arduino Uno. Folge dabei der Darstellung in Fig. 3.1 auf S. 7. Hier gilt wieder, dass die Farben der Jumper Wires nicht so wichtig sind.
- Verbinde die Wasserpumpe mechanisch mit dem **Schlauch**.
- Verbinde die **Wasserpumpe** elektrisch über das Breadboard mit dem Arduino. Folge dabei der Darstellung in Fig. 3.1 auf S. 7. Hier gilt wieder, dass die Farben der Jumper Wires nicht so wichtig sind.
- Stelle die **Wasserpumpe in einen der beiden Eimer**.

3.2 Programmierung und Inbetriebnahme

- Starte die **Arduino IDE**.
- Die Workshop-Betreuung erklärt dir nun mit Hilfe des Source-Listings in Kap. Error: Reference source not found, wie der Arduino Uno zu **programmieren** ist.

Wenn das Programm läuft:

- Die Wasserpumpe steht in einem der beiden Eimer, platziere den **Feuchtigkeitssensor** mittels Klammer **im anderen Eimer**.
- Fülle in den Eimer mit der Wasserpumpe soviel **Wasser**, dass es das untere Viertel des Sensors bedeckt.
- Stecke den **Schlauch** am Ausgang der Pumpe in den leeren Eimer.
- Verbinde die **Powerbank** mit
  1. dem Breadboard und dann erst mit
  2. dem Arduino Uno.

  Folge dabei der Darstellung in Fig. 3.1 auf S. 7.
- Mach' folgende **Experimente**:
  1. Die Pumpe läuft solange, bis das Wasser den Feuchtesensor im jetzt noch leeren Eimer erreicht. Die Pumpe muss dann abschalten. **Achtung: Die Wasserpumpen dürfen nie leer laufen, sie können sonst Schaden durch Überhitzung nehmen!**
  2. Schau’ mal, was passiert, wenn du den Feuchtigkeitssensor etwas anhebst, sodass er

nicht mehr mit Wasser bedeckt ist. Eigentlich müsste jetzt die Wasserpumpe wieder einschalten – tut sie das?

3. Wenn du zufrieden bist, wie dein Programm funktioniert, dann versuche herauszufinden, wie viele Liter / Stunde die Pumpe fördern kann. Wie könntest du hier vorgehen? Denn dass sich das Experiment über eine ganze Stunde erstreckt, willst du ja nicht, oder?

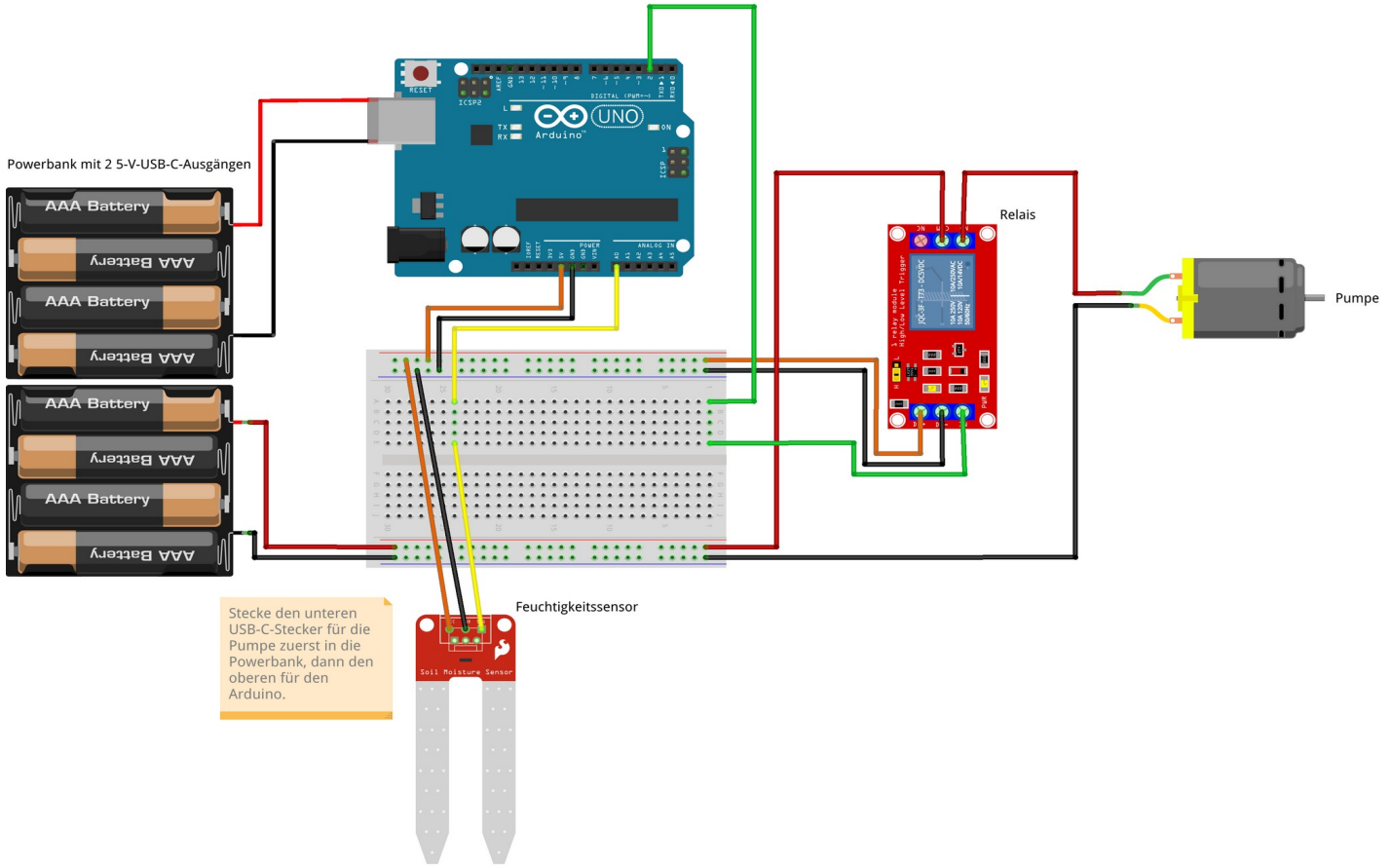

Fig. 3.1: Verdrahtungsplan für SCHLAUBEET – bitte die Orientierung des Arduino und des Breadboards beachten!

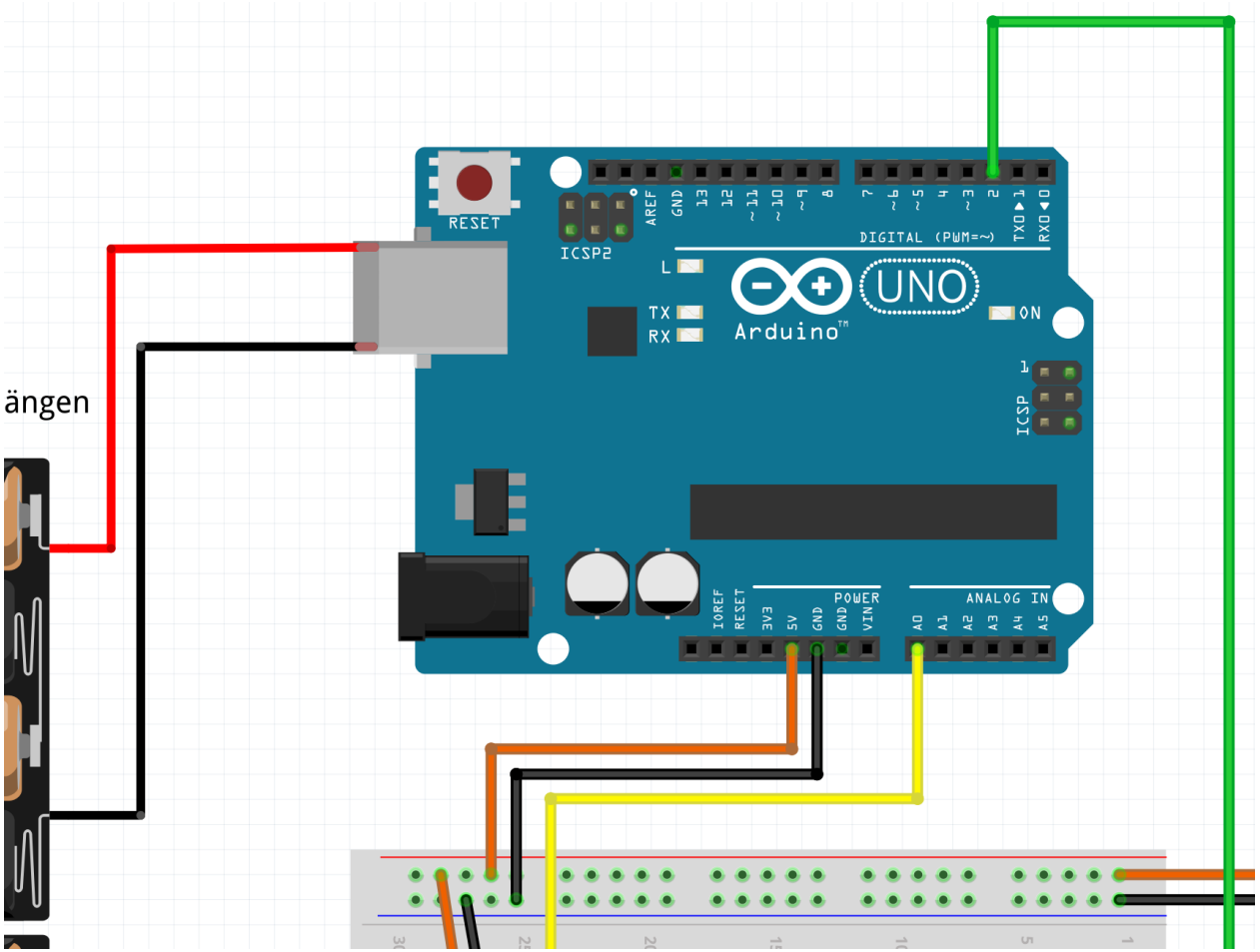

Fig. 3.2: Verdrahtungsplan für SCHLAUBEET – Arduino Uno herausgezoomt

4 Software

Die Software-Listings, die alle grau hinterlegt sind, sehen auf den ersten Blick sehr umfangreich aus. Der Grund dafür sind die vielen Kommentare[1], die erklären warum etwas an dieser Stelle so gemacht worden ist. Viele Kommentare sind anfangs wie die sprichwörtlichen vielen Bäume, die den Blick auf den Wald versperren. Damit die vielen Kommentare also nicht den Blick auf den Code versperren, sind sie 8 Tabulatorschritte eingerückt und befinden sich damit eher in der rechten Hälfte des Programms.

Um es noch einfacher zu machen, sich in die Software einzufinden, beginnen wir mit dem Programm, aus dem alle Kommentare gelöscht worden sind, und erklären es Zeile für Zeile.

4.1 Applikation

4.1.1 Voraussetzungen

Es wird ein Laptop oder PC benötigt, auf dem...

die **Arduino IDE** installiert ist. Wenn auf den Rechnern *Windows* läuft, muss sichergestellt werden, dass auch die nötigen **Software-Treiber** installiert sind, um einen Arduino-Mikrocontroller ansprechen zu können!

4.1.2 Software (schlaubeetWs1.ino)

Die Aufgabe besteht darin, je nach Messwert eines Feuchtesensors eine Pumpe ein oder auszuschalten – s. Error: Reference source not found.

Die Liste der Teile die hierfür notwendig sind, findest du in Tab. 2.1 auf S. 4.

Die Software dazu ist im Folgenden gelistet. Sie läuft prinzipiell auf jedem Arduino und ist gut dokumentiert, d.h. reichlich mit Kommentaren versehen. Aber vielleicht ist genau das das Problem? Möglicherweise sieht man vor lauter Bäumen den sprichwörtlichen Wald nicht mehr? Stellen wir das vollständige Listing also etwas zurück und schauen, wie das Programm aussähe, wenn man keine Kommentare verwendete und auch die Funktionen, die die Pumpen abstrahieren weglässt.

```
 1.  #define PID_PUMP                2
 2.  #define CYCLETIME_LOOP_MS       100
 3.  #define PID_HUMIDITYSENSOR      A0
 4.  #define MOISTURETRESHOLD_100                50
 5.  #define MOISTURETHRESHOLD_HYSTERESIS_100    10
 6.
 7.
 8.  int humiditysensorvalue_100( unsigned char pid)
 9.  {
10.      unsigned int    value = analogRead( pid);
11.      const int       value_defining_wetness = 411;
12.      const int       value_defining_dryness = 790;
13.      int             humidity_100 = map( \
14.          value, value_defining_wetness, value_defining_dryness, 100, 0
15.      );
```

1 Einzeilenkommentare beginnen mit `//`, mehrzeilige Kommentare beginnen mit `/*` und enden mit `*/`.

```
16.      //         ^                       ^
17.      //         |                       |
18.      //         |                       trocken: Will man keine negativen Werte,
19.      //         |                       muss dieser Wert vergrößert werden.
20.      //         nass: Will man keine Werte größer als 100 %, muss
21.      //         dieser Wert verkleinert werden.
22.
23.      humidity_100 = max( 0, min( 100, humidity_100));
24.                                      // Wert aus Bereich 0...100 beschränken
25.      return humidity_100;
26.  }
27.
28.
29.  bool    is_moisture_low( int humidity_100)
30.  {
31.      return humidity_100 < (MOISTURETRESHOLD_100 -
MOISTURETHRESHOLD_HYSTERESIS_100);
32.  }
33.
34.
35.  bool    is_moisture_ok( int humidity_100)
36.  {
37.      return humidity_100 >= (MOISTURETRESHOLD_100 +
MOISTURETHRESHOLD_HYSTERESIS_100);
38.  }
39.
40.
41.  void setup()
42.  {
43.      pinMode( PID_PUMP, OUTPUT);
44.  }
45.
46.
47.  void loop()
48.  {
49.       int humidity_100 = humiditysensorvalue_100( PID_HUMIDITYSENSOR);
50.
51.      if ( is_moisture_low( humidity_100))
52.          digitalWrite( PID_PUMP, HIGH);
53.
54.      if ( is_moisture_ok( humidity_100))
55.          digitalWrite( PID_PUMP, LOW);
56.
57.      delay( CYCLETIME_LOOP_MS);
58.  }
```

Gehen wir’s kurz durch, so lange ist das Programm nun ja nicht mehr.

Zeilen 1 – 5...

definieren Macros, die man sich als einfachen Textersatz vorstellen kann. Nehmen wir zur Erklärung die Zeile 1. Hier wird die Pin-Id jenes Pins definiert, an dem die Pumpe angeschlossen ist – D2 oder kurz 2. Da wir diesen Wert mehrmals brauchen, nämlich in den Zeilen 43, 52 und 55, ist es klug, einen Textersatz dafür zu definieren: `PID_PUMP`. Überall, wo wir `PID_PUMP` schreiben, setzt der Präprozessor den Wert 2 ein. Praktisch, nicht?

Wie ist es aber mit Macros, die nur einmal gebraucht werden? Die wären nicht unbedingt nötig, stimmt. Aber da kommt ein weiterer Vorteil vcon Macros zum Zuge: Sie sind „sprechend“. `CYCLETIME_LOOP_MS` für die Zykluszeit der Funktion `loop()` ist offensichtlicher als ein einfaches 500, nicht?

Zeilen 8 – 25…

definieren die Funktion `humiditysensorvalue_100()`, die den Analogwert des Feuchtesensors einliest und ihn in einen %-Wert umwandelt. Funktionen fassen mehrere zusammengehörige Code-Zeilen innerhalb einer öffnenden `{` und einer schließenden `}` zusammen. Sie haben meistens ein oder mehrere sog. Argumente als Eingangsdaten und liefern ein Ergeb-

nis. Hier ist das Eingangsdatum die Pin-Id des Pins, an dem der Feuchtesensor angeschlossen ist und die Funktion liefert die Feuchte in % (0 %…Sensor an der Luft, abgewischt, 100 %...Sensor im Wasser).

Zeilen 11, 12…

definieren die Analogwerte für trockenen Feuchtesensor und nassen Feuchtesensor.

Hier ist nun darauf zu achten, welcher Sensor verwendet wird - s. folgende Tabelle.

Farbe des Sensorhalters	Analogwert für trockenen Sensor	Analogwert für nassen Sensor
Antrazit	790	411
Orange	493	222

Tab. 4.1: Wertebereiche für verschiedene Varianten des Feuchtesensors

Zeilen 29 – 38…

bewerten einen Feuchtewert in %. Wann ist die Erde zu trocken, wann feucht genug? Der Body, der Inhalt der Funktionen ist jeweils nur eine Zeile. Aber weil gerechnet wird, liegt es doch nahe, diese Zeile in Funktionen mit aussagekräftigen Namen zu verpacken. Zusammen realisiern sie übrigens eine Hysteres der Breite

`2 * MOISTURETHRESHOLD_HYSTERESIS_100`.

So wird verhindert, dass an der Grenze von trocken zu nass die Pumpe laufend ein- und gleich wieder ausgeschaltet wird.

Zeile 41 – 44…

dient der Konfiguration: Welcher Pin ist Eingang, welcher Ausgang? Die Funktion heißt `setup()`, ist fest vorgegeben und wird nur einmal – beim Systemstart - ausgeführt. Um diese Funktion kommt man nicht herum.

Erzeugt man mit der *Arduino IDE* ein INO-File (einen sog. Sketch, wie Arduino das nennt), dann sind `void setup()` und `void loop()` schon enthalten – mit leerem Body natürlich. Beide Funktionen haben keine Argumente und liefern keine Werte (Typ `void`).

Ziele 47 – 58…

ist die zweite, fest vorgegebene Funktion, um die man nicht herumnkommt. Sie wird periodisch ausgeführt. Hier sind also alle Eingänge zu lesen und die Ausgänge entsprechend zu schalten.

Aber wie geht man vor? Schreibt man wirklich alles Zeile für Zeile oben beginnend in ein File? Nein. Spielen wir’s doch einmal durch.

Man startet die *Arduino IDE* und öffnet – sofern nicht sowieso schon ein neues File geöffnet und angezeigt wird – ein neues File (File → New):

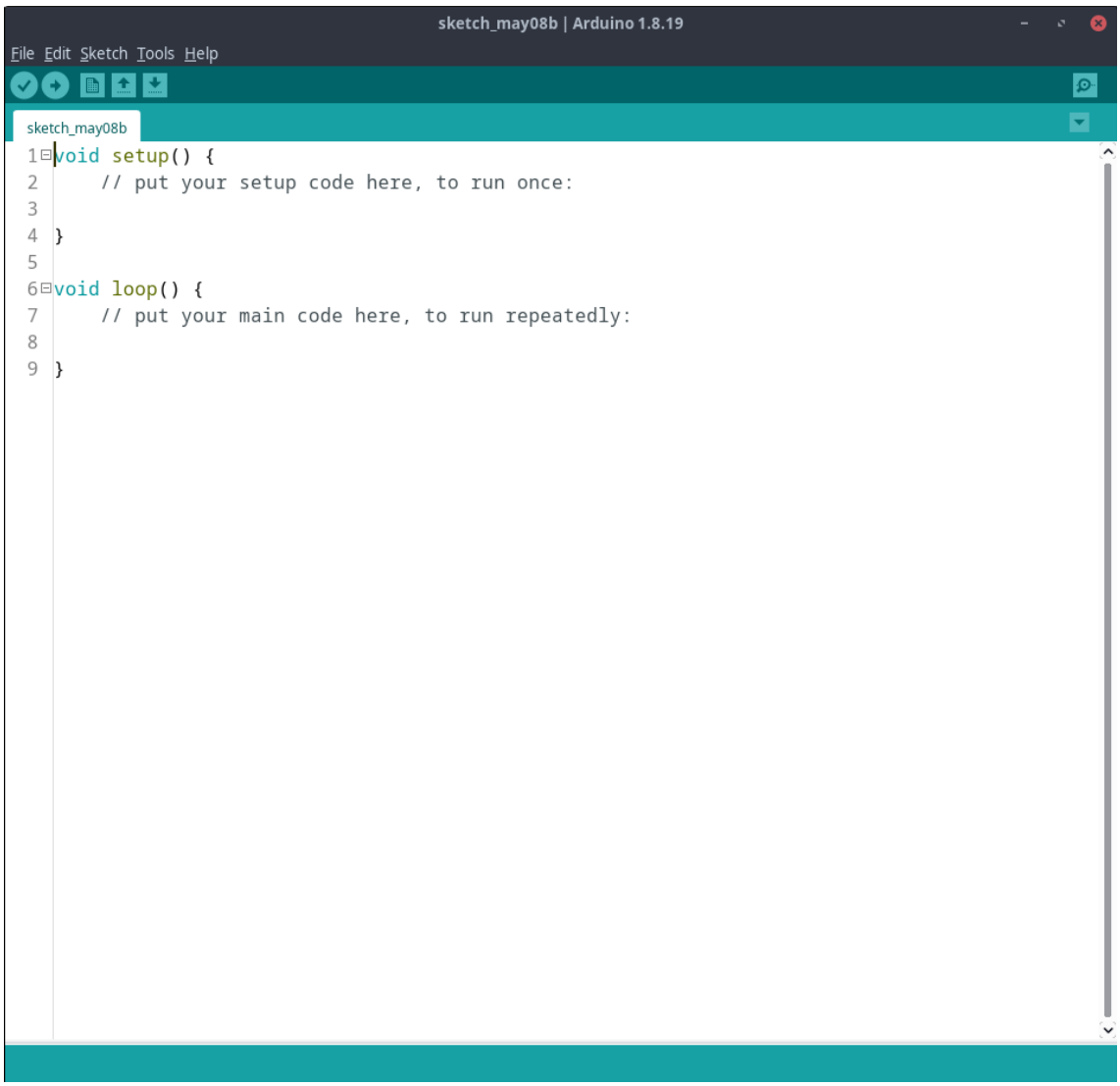

Fig. 4.1: Neues File wie von der Arduino IDE defaultmäßig angelegt.

In der Annahme, dass man bereits weiß, was man will, würde man für

```
1.  Feuchte ermitteln
2.  Wenn diese Feuchte anzeigt, dass es zu trocken ist:
3.      Schalte die Pumpe ein
4.
5.  Sonst:
6.      Schalte die Pumpe aus
```

Oder auch – wenn man stzark unterscheiden will zwischen zu trocken und ausreichend feucht:

```
1.  Feuchte ermitteln
2.  Wenn diese Feuchte anzeigt, dass es zu trocken ist:
3.      Schalte die Pumpe ein
4.
5.  Wenn diese Feuchte anzeigt, dass es feucht genug ist:
6.      Schalte die Pumpe aus
```

Nun überlegt man sich, wie das in der Programmiersprache *C* zu formulieren wäre. Wenn man

noch nicht an Sensoren denken will und Pumpen und Pins, mit denen diese Teile verbunden sind, formuliert man einfach in Funktionen, deren Definition man vorerst hinausschiebt:

```
1.  int humidity_100 = humiditysensorvalue_100( PID_HUMIDITYSENSOR);
2.  if (is_moisture_low( humidity_100))
3.      pump_on();
4.
5.  if (is_moisture_ok( humidity_100))
6.      pump_off();
```

Dann bleibt nur noch, diese Funktionen mit „Leben“ zu füllen:

- `is_moisture_low()` und `is_moisture_ok()` lesen einen analogen Eingang und wandeln den Bit-Wert, der zwischen 0 und 1023 liegen kann, in eien Feuchtewert in % um.
- `pump_on()` und `pump_off()` schreiben auf digitale Ausgänge.

Zahlenwerte, die man dann noch braucht, werden am Beginn des Programms oft als Macros definiert.

Nun ist der Zugang zur vollständig dokumentierten Version vielleicht ein wenig leichter:

```
/*****************************************************************************
 *  PROJECT:
 *      Workshop SCHLAUBEET
 *
 *  FILE:
 *      arduino/sketchbooks/schlaubeetWs1/schlaubeetWs1.ino
 *
 *  SYNOPSIS:
 *      Workshop SCHLAUBEET, Standardversion: Ein-/Ausschalten einer Pumpe in
 *      Abhängigkeit des Messwertes eines Feuchtesensors.
 *
 *  PRELIMINARIES:
 *      Dieses Programm setzt einen Arduino Uno (oder Nano) voraus.
 *
 *  NOTE:
 *      Source-Code beginnt in der äußerst linken Spalte und wird mit jeder
 *      öffnenden geschwungenen Klammer `{` um 4 Zeichen eingerückt. Jede `}` rückt
 *      ihn wieder aus um 4 Zeichen. Für den Compiler (= Programm, das aus Text
 *      ausführbaren Maschinen-Code macht), wäre das nicht nötig, für uns aber
 *      schon: Nur so bewahren wir die Übersicht, sehen, was wohin gehört und
 *      finden uns so bei der Fehlersuche zurecht.
 *
 *      Kommentare, die als eine Art Überschrift fungieren, sind linksbündig mit
 *      dem Source-Code platziert, den sie "ankündigen". Kommentare, die einzelne
 *      Zeilen dokumentieren, beginnen nach der Zeile, die sie dokumentieren und
 *      8 Tabs = 32 Zeichen eingerückt. So ergibt sich ein zweigeteiltes Source-
 *      Listing: Links der Code und rechts die Kommentare, die so nicht den Blick
 *      auf den Source-Code verstellen.
 *
 *****************************************************************************/

/*****************************************************************************
 *  M a c r o s  (Textersatz)
 *
 *****************************************************************************/

#define PID_PUMP                2
                                /*
                                 * Pin-Id (Nummer) des Pins, an dem die Pumpe angeschlossen
                                 * ist.
                                 *
                                 * Will man ohne reale Pumpe arbeiten, kann man hier
                                 * LED_BUILTIN angeben. Andernfalls gibt man den Pin
```

```
                                * an, an dem die reale Pumpe hängt.
                                *
                                * V o r s i c h t: Die Pumpe muss über eine Relais
                                *      mit einer externen Spanunngsquelle versorgt werden.
                                *      Effektiv steuert man also ein Relais an, an das
                                *      die reale Pumpe angeschlossen ist!
                                *
                                *      Software-maäßig macht das keinen Unterschied,
                                *      hardware-mäßig aber schon: Wir die Pumpe vom
                                *      Arduino direkt angesteuert, beschädigt das den
                                *      Arduino!
                                */
#define CYCLETIME_LOOP_MS       100
                                /*
                                 * Alle wie viele Millisekunden soll das Programm ausgeführt werden?
                                 */
#define PID_HUMIDITYSENSOR      A0
                                /*
                                 * Pin-Id des analogen Eingangs, an den der analoge
                                 * Feuchtesensor angeschlossen ist.
                                 */
#define MOISTURETRESHOLD_100                50
#define MOISTURETHRESHOLD_HYSTERESIS_100    10

/*****************************************************************************
 *  F u n c t i o n s
 *
 *****************************************************************************/

/*****************************************************************************
*/
int humiditysensorvalue_100( unsigned char pid)
/*
 * Liest einen Wert von einem der analogen Pins des Arduino Uno R4 ein und
 * interpretiert diesen Wert als von einem 'Soil Moisture Sensor Hygrometer
 * Module V1.2 Capacitive' gemessene Feuchte.
 *
 * Der Sensor ist hier gut dokumentiert:
 * https://www.az-delivery.de/en/products/bodenfeuchte-sensor-modul-v1-2
 *
 * Parameters:
 *      pid:
 *          Pin-Id wie angegeben direkt auf dem Arduino Uno R4.
 *          Die Macros A0, ... A5 sind erlaubte Werte.
 *
 * Usage:
 *      ```
 *      if (is_moisture_low( humiditysensorvalue_100( A0)))
 *          ...
 *      ```
 *
 *****************************************************************************/
{
    unsigned int    value = analogRead( pid);
    const int       value_defining_wetness = 411;
    const int       value_defining_dryness = 790;
    int             humidity_100 = map( \
        value, value_defining_wetness, value_defining_dryness, 100, 0
    );
    //         ^                        ^
    //         |                        |
    //         |                        trocken: Will man keine negativen Werte,
    //         |                        muss dieser Wert vergrößert werden.
    //         nass: Will man keine Werte größer als 100 %, muss
    //         dieser Wert verkleinert werden.

    humidity_100 = max( 0, min( 100, humidity_100));
                                    // Wert auf Bereich 0...100 beschränken
    return humidity_100;
```

```
}


/*****************************************************************************
*/
bool    is_moisture_low( int humidity_100)
/*
 * Diese Funktion beurteilt den übergebenen Feuchtewert.
 *
 * Wir nehmen trockene Erde an, wenn wir unter 30 % sind.
 * Wir nehmen nasse Erde an, wenn wir über 50 % sind.
 *
 * Parameters:
 *      humidity_100 (int):
 *          Feuchtewert in %, den es zu beurteilen gilt. 0 % ist trocken
 *          (Sensor an der Luft, abgewischt), 100 % ist nass (Sensor im Wasser).
 *
 * Usage:
 *      ```
 *      if (is_moisture_low( humiditysensorvalue_100( A0)))
 *          ...
 *      ```
 *
 *****************************************************************************/
{
    return humidity_100 < (MOISTURETRESHOLD_100 - MOISTURETHRESHOLD_HYSTERESIS_100);
}

/*****************************************************************************
*/
bool    is_moisture_ok( int humidity_100)
/*
 * Diese Funktion beurteilt den übergebenen Feuchtewert.
 *
 * Wir nehmen trockene Erde an, wenn wir unter 30 % sind.
 * Wir nehmen nasse Erde an, wenn wir über 50 % sind.
 *
 * Parameters:
 *      humidity_100 (int):
 *          Feuchtewert in %, den es zu beurteilen gilt. 0 % ist trocken
 *          (Sensor an der Luft, abgewischt), 100 % ist nass (Sensor im Wasser).
 *
 * Usage:
 *      ```
 *      if (is_moisture_ok( humiditysensorvalue_100( A0)))
 *          ...
 *      ```
 *
 *****************************************************************************/
{
    return humidity_100 >= (MOISTURETRESHOLD_100 + MOISTURETHRESHOLD_HYSTERESIS_100);
}


/*****************************************************************************
 */
void pump_off()
/*
 * Schaltet die Pumpe aus.
 *
 * Die Pumpe wird über ein Relais angesteuert, was software-mäßig aber unerheblich
 * ist. Der Software ist es egal, ob an enem zu schaktenden Pin ein Relais ist oder
 * nicht. Hardwaremäßig ist es wichtig, denn eine direkt angeschlossene Pumpe würde
 * den Arduino beschädigen.
 *
 *****************************************************************************/
{
    digitalWrite( PID_PUMP, LOW);
}
```

```
/*****************************************************************************
 */
void pump_on()
/*
 * Schaltet die Pumpe ein.
 *
 *
 * Die Pumpe wird über ein Relais angesteuert, was software-mäßig aber unerheblich
 * ist. Der Software ist es egal, ob an enem zu schaktenden Pin ein Relais ist oder
 * nicht. Hardwaremäßig ist es wichtig, denn eine direkt angeschlossene Pumpe würde
 * den Arduino beschädigen.
 *
 *****************************************************************************/
{
    digitalWrite( PID_PUMP, HIGH);
}


/*****************************************************************************
 */
void setup()
/*
 *****************************************************************************/
{
    pinMode( PID_PUMP, OUTPUT);
}

/*****************************************************************************
*/
void loop()
/*
 * Funktion, die der Arduino periodisch ausführt. Das ist also DIE Funktion,
 * in der die gesamte Funktionalität steckt.
 *
 * Diese Funktion stellt zusammen mit der Funktion setup() das Minimum dar, das
 * ein Arduino-programm (auch als Sketch) bezeichnet) enthalten muss.
 *
 *****************************************************************************/
{
    /*
     * Feuchtesensor lesen
     */
     int humidity_100 = humiditysensorvalue_100( PID_HUMIDITYSENSOR);

    /*
     * Feuchte beurteilen und entsprechend reagieren
     */
    if ( is_moisture_low( humidity_100))
        pump_on();

    if ( is_moisture_ok( humidity_100))
        pump_off();

    /*
     * Zykluszeit abwarten
     */
    delay( CYCLETIME_LOOP_MS);
                                    /* Das ist UNGEFÄHR die Zykluszeit. Nur UNGEFÄHR,
                                     * weil die Lauzeit von Loop nicht berücksichtigt
                                     * ist.
                                     */
}
```

Index

A

Anleitung zum Aufbau der Hardware.....5
Applikation.....9
Aufbau.....5
Aufgabenstellung.....3

E

Einleitung.....1

I

Inbetriebnahme.....5

P

Programmierung.....5

S

Software.....9
Software (schlaubeetWs1.ino).....9

T

Teileliste.....4

V

Versionsgeschichte.....2
Voraussetzungen.....9