FHV
Vorarlberg University
of Applied Sciences

SCHLAUBEET - ein Flying-Classroom-Projekt

Anleitung zum Workshop

Version 1.0.2

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

Versionsgeschichte

Version	Datum	Änderung	Autor
1.0.2	2024-05-15	Dokumentation des Wertebereiches unterschiedlicher Varianten des Feuchtesensors.	Franz Geiger
1.0.1	2024-05-15	Aktualisierung der Teileliste.	Franz Geiger
1.0.0	2024-05-10	Erstversion.	Franz Geiger

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1 Einleitung

Ein dicker Hund, dieses Booklet – fast 70 Seiten. Aber keine Angst, du brauchst nur die Seiten 7 - 21, um erfolgreich zu sein! Der ganze Rest ist für all jene, die ein etwas tieferes Verständnis über das *Embedded System* „Schlaubeet“ gewinnen wollen.

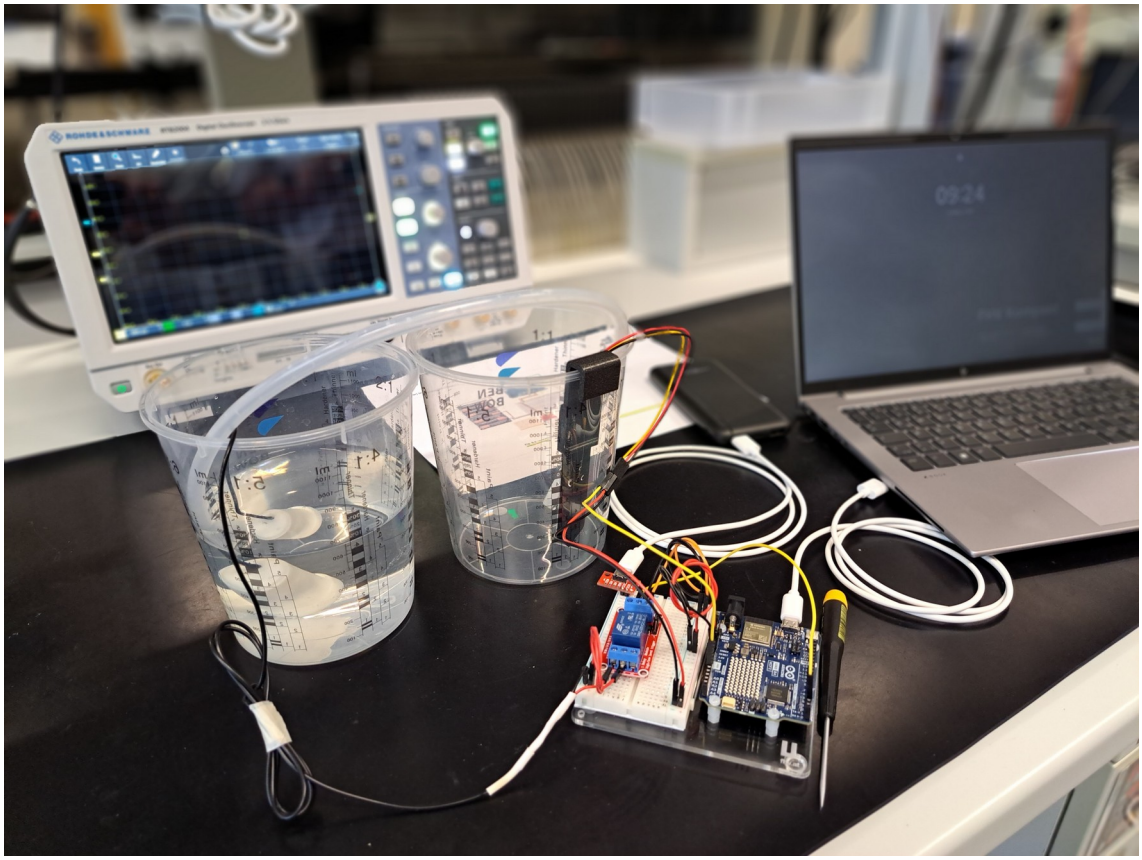

Fig. 1.1: Schlaubeet aufgebaut und betriebsbereit

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 Glossar

Embedded System

Ein System, das in eine Anlage eingebettet ist und dort nur *einen* Zweck erfüllt. Das ABS bei einem Auto ist ein Beispiel hierfür.

GND

Kurz für *Ground*, das man als Minuspol einer Batterie auffassen kann.

Schlaubeet

Name des Workshops, abgeleitet von *smart* und *Gartenbeet*.

3 Die Programmiersprache *C*

Die Arduino-Familie wird in *C/C++* programmiert. Programmiersprachen folgen gewissen Regeln, sog. Syntaxregeln. Gute Einführungen findest du z.B hier:

- https://www.freecodecamp.org/news/the-c-beginners-handbook/: The C Beginner's Handbook: Learn C Programming Language basics in just a few hours.
- https://github.com/Embed-Threads/Learn-C/blob/main/books/c-programming-absolute-beginner.pdf: C Programming – Absolute Beginner’s Guide.

Diese Titel sind in Englisch verfasst.

Einen deutschen Titel findest du z.B. hier:

- https://www.c-howto.de/tutorial/

4 Aufgabenstellung

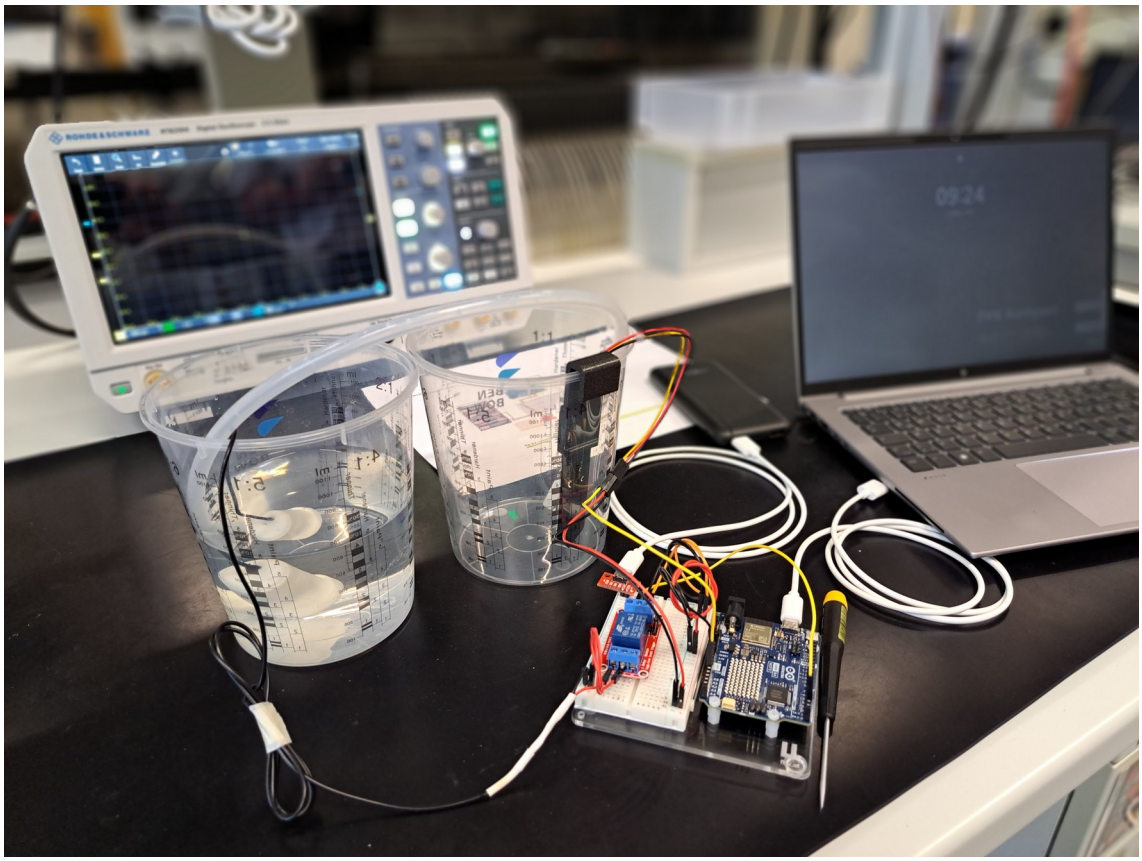

Fig. 4.1: Schlaubeet aufgebaut und betriebsbereit

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. 4.1: Teileliste SCHLAUBEET

Wie du am besten vorgehst, findest du in folgendem Kap. 5, Anleitung zum Aufbau der Hardware.

5 Anleitung zum Aufbau der Hardware

Die Liste der Teile, die du für den Aufbau benötigst, findest du in Kap. 4, 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!

5.1 Mechanisch-elektrischer Aufbau

- Verbinde den **Feuchtigkeitssensor** elektrisch über das Breadboard mit dem Arduino Uno. Folge dabei der Darstellung in Fig. 5.1 auf S. 11. 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. 5.1 auf S. 11. Hier gilt wieder, dass die Farben der Jumper Wires nicht so wichtig sind.
- Stelle die **Wasserpumpe in einen der beiden Eimer**.

5.2 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. 5.1 auf S. 11.
- 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?

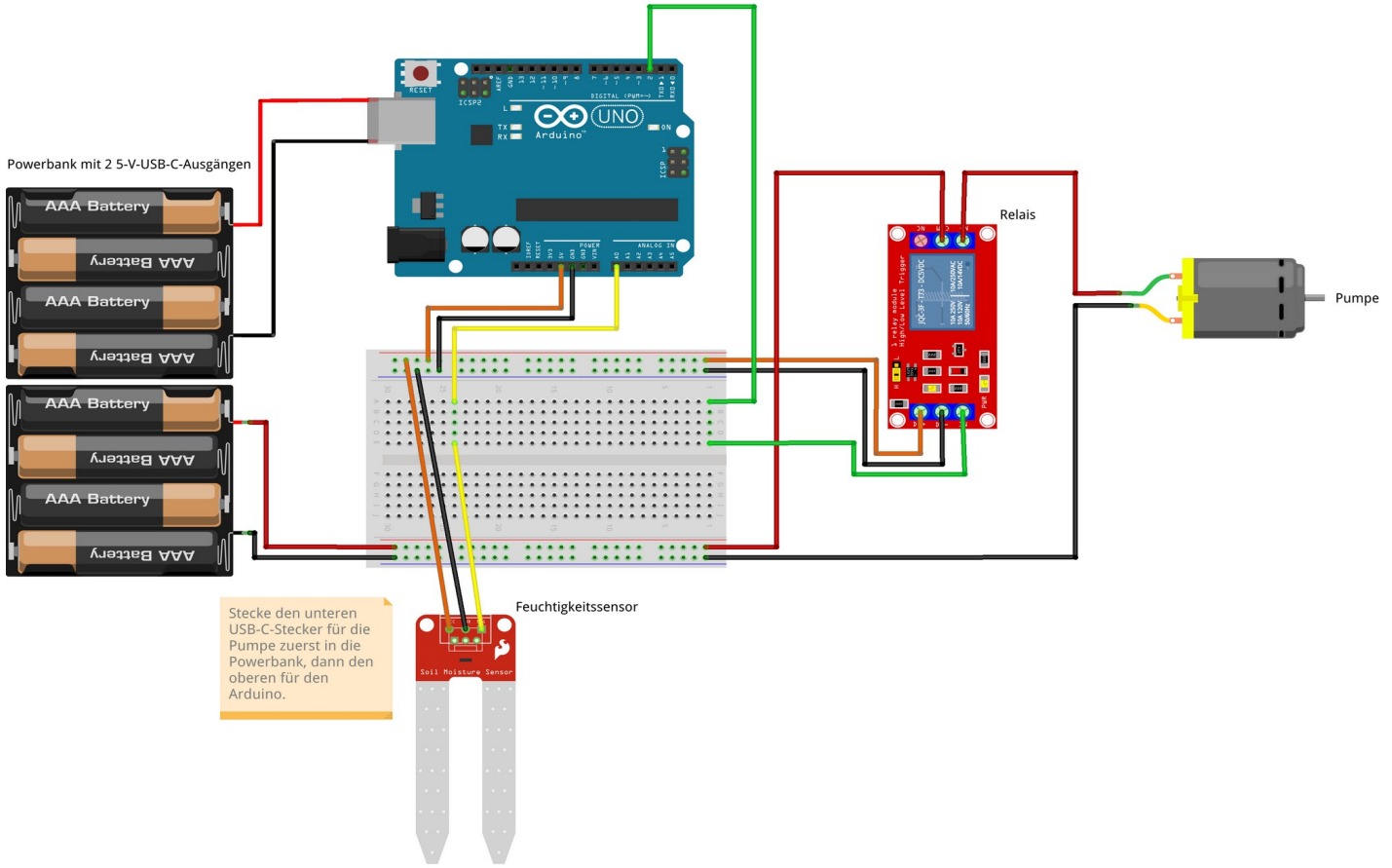

Fig. 5.1: Verdrahtungsplan für SCHLAUBEET – bitte die Orientierung des Arduino und des Breadboards beachten!

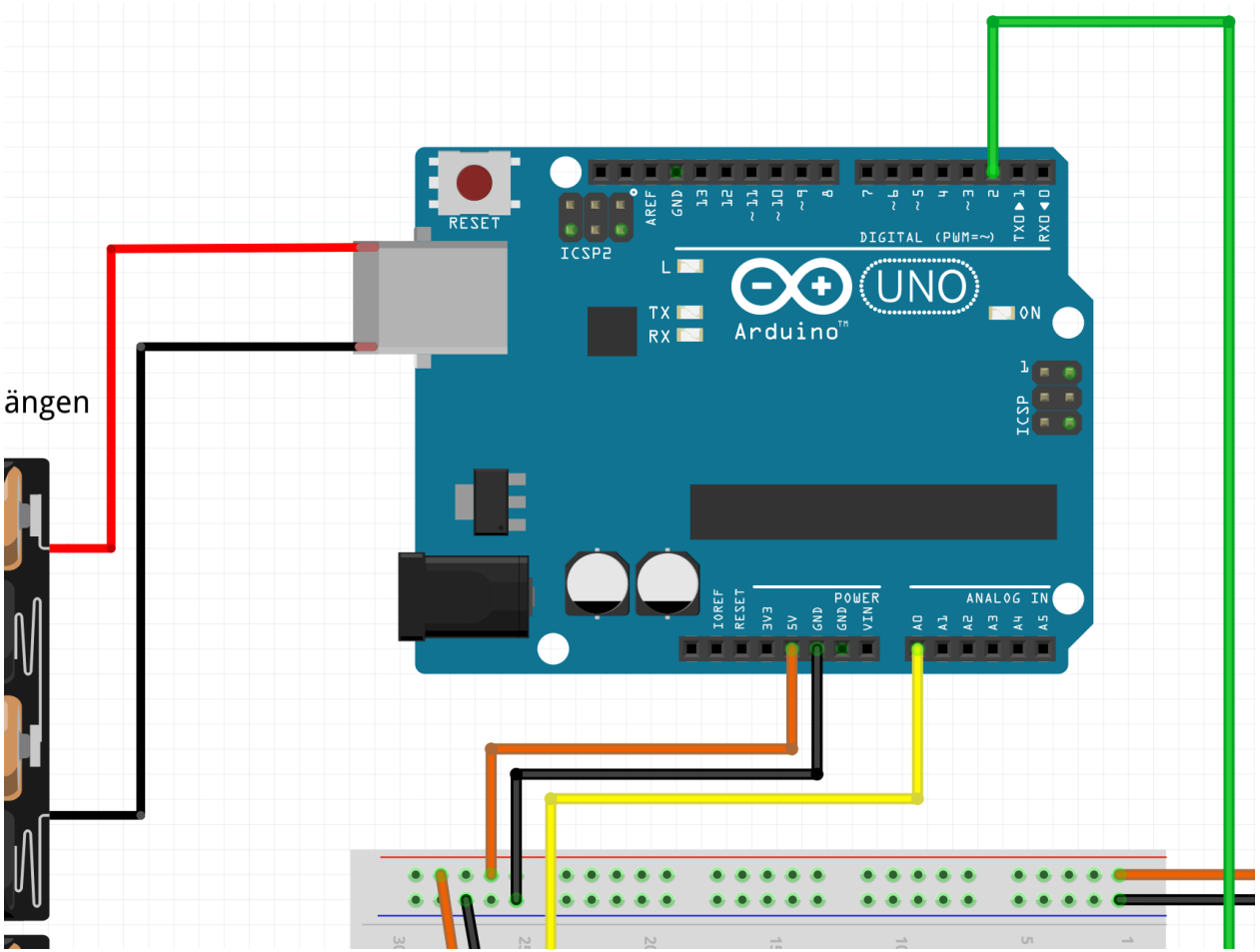

Fig. 5.2: Verdrahtungsplan für SCHLAUBEET – Arduino Uno herausgezoomt

6 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.

6.1 Applikation

6.1.1 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!

6.1.2 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. 4.1 auf S. 8.

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. 6.1: 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):

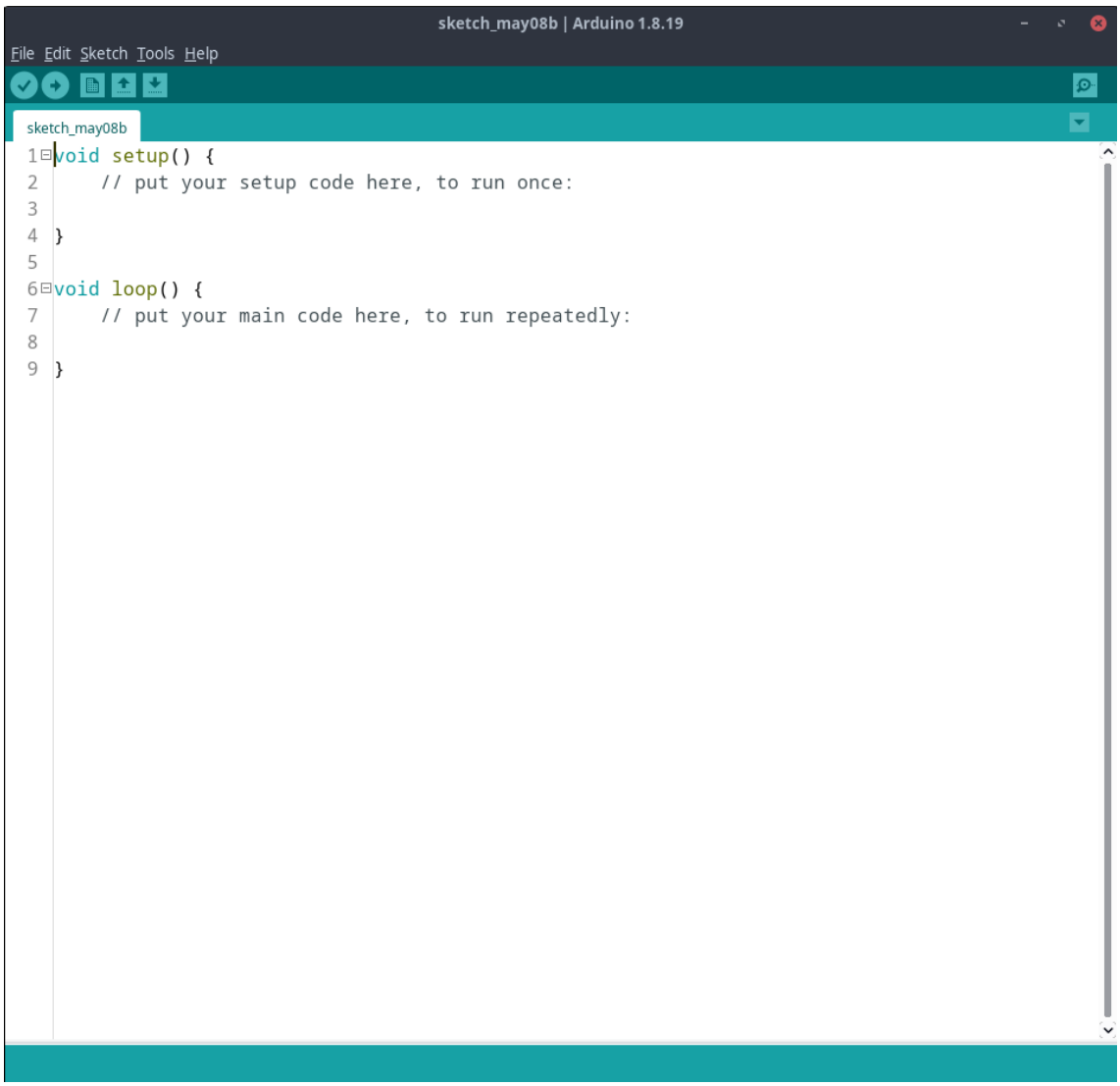

Fig. 6.1: Neues File wie von der Arduino IDE defaultmäßig angelegt.

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.
                                   */
}
```

7 Anhang: Schlaubeet-Versionen

Gratulation, wenn das Programm, das in den letzten Kapiteln dokumentiert worden ist, läuft: Der erste Schritt in die faszinierende Welt der Embedded Systems ist gemacht.

Im Software-Engineering kommt man gut cvoran, wenn man nach dem Motto *Make ir work – make it right – make it fast* vorgeht. In diesem Anhang finden sich nun weitere Schlaubeet-Versionen. Die erste im folgenden Kapitel ist eine, die ohne Sensor und Pumpe auskommt. Die daraiffolgenden Versionen gehen dann immer einen kleinen Schitt weiter.

Taster	Sensor	Pumpe, Schläuche, Eimer	LED-Matrix	Library TRIDENT notwendig	Arduino-Sketch	Kapitel in diesem Booklet	Anmerkung
x					schlaubeetWs0.ino	7.1 auf S.	Minimal-Version • ohne Sensor und Pumpe und • einfach programmiert. Als Anfänger startest man mit dieser Version.
	x	x			schlaubeetWs1.ino	6 auf S. 13	**Workshop-Version** • mit Sensor und Pumpe und • einfach programmiert. Das ist die Standard-Version.
	x	x	x	x	schlaubeetWs2.ino	7.2 auf S. 25	Version mit • fortgeschrittenem Handling möglicher Zustände. Wer tiefer in die technische Programmierung einsteigen möchte, macht mit dieser Version weiter.
	x	x	x	x	schlaubeetWs3.ino	7.3 auf S. 31	Version mit • fortgeschrittenem Handling möglicher Zustände • Handling weiterer Zustände • Anzeige des Bertriebszustandes und von Messwerten auf der LED-Matrix des Arduino (erfordert die Labrary TRIDENT wie beschrieben in Kap. 8 auf S. 37), • Sensoren und Pumpen aus der Library TRIDENT (Kap. 8 auf S. 37). Diese Version realisiert die Annehmlichkeiten der LED-Matrix-Anzeige und bedient sich vorprogrammierter Sensoren, die in der Library TRIDENT (Kap. 8 auf S. 37) verfügbar sind.

Tab. 7.1: Matrix der Schlaubeet-Versionen

7.1 *Schlaubeet* ohne Feuchtesensor und ohne Pumpe (schlaubeetWs0.ino)

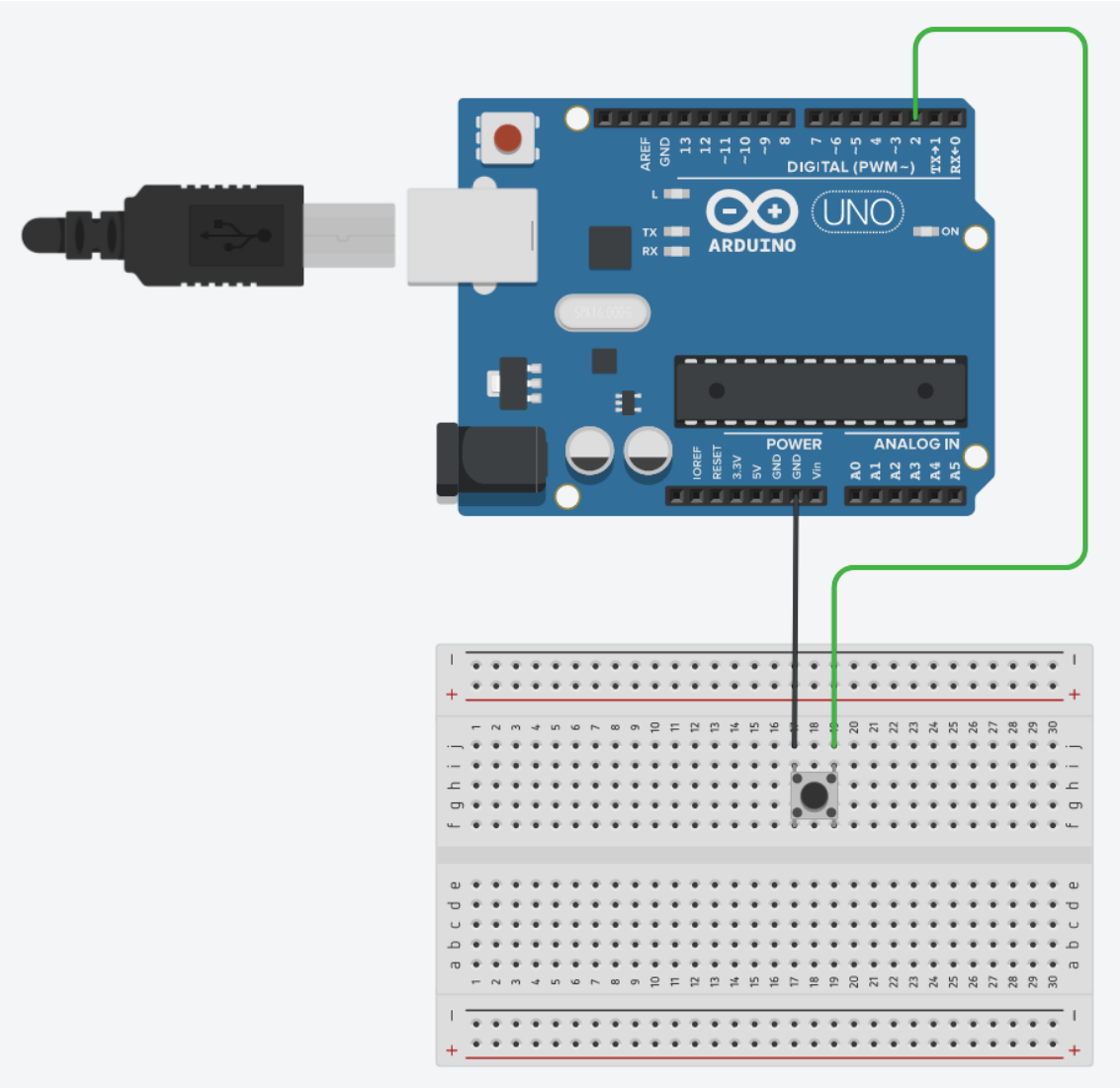

Fig. 7.1: Verdrahtungsplan für die Minimal-Version

Es gibt nur ganz wenig zu ändern, wenn kein Feuchtesensor und keine Pumpe zur Verfügung stehen: Ein Taster simuliert den Feuchtigkeitssensor, die Wasserpumpe wird durch die interne LED des *Arduino Uno* simuliert.

Auch hier wollen wir wieder mit einer Version ohne Kommentare beginnen, die den Blick auf das Wesentliche verstellen könnten.

```
 1.  #define PID_BUTTON              2
 2.  #define PID_PUMP                LED_BUILTIN
 3.
 4.  void setup()
 5.  {
 6.      pinMode( PID_BUTTON, INPUT_PULLUP);
 7.      pinMode( PID_PUMP, OUTPUT);
 8.  }
 9.
10.  void loop()
11.  {
```

```
12.       if (digitalRead( PID_BUTTON) == LOW) // Schalter ist 0-aktiv.
13.       {
14.           digitalWrite( PID_PUMP, HIGH);
15.       }
16.       else
17.       {
18.           digitalWrite( PID_PUMP, LOW);
19.       }
20.
21.       delay( 100);
22.   }
```

Zeilen 1 – 2…

definiert wieder die Pin-Ids der Pins, an denen der Taster un ddie Pumpe angeschlossen sind. Da die Werte im weiteren Verlauf mehr als einmal benötigt werden, sind sie per Macros definiert. Die Macro-Namen werden dann im Zuge des Compilierens (die IDE macht aus dem Programmtext ein ausführbares Programm) durch die Werte im Macro ersetzt.

Zeilen 4 – 8…

konfiguerien das System. Die Funktion `void setup()` wird nur einmal, beim Systemstart ausgeführt, wenn als der Arduino an eine Energiequelle angeschlossen wird und startet.

Zeile6…

definiert einen Eingang, der – wenn kein Taster gedrückt ist – als *high*, also logisch 1 (= 5 V) erkannt wird. EIn Druck auf den Taster „zieht“ den Eingang auf GND. Das ist der Grund, warum in Zeile 12 auf LOW (= logisch 0) abgefragt wird anstatt auf HIHJ (= logisch 1).

Den Verdrahtungsplan zeigt Fig. 7.1.

Kommen wir nun zur vollständig kommentierten Version:

```
/*****************************************************************************
 *  PROJECT:
 *      Workshop SCHLAUBEET
 *
 *  FILE:
 *      /arduino/sketchbooks/schlaubeetWs0/schlaubeetWs0.ino
 *
 *  SYNOPSIS:
 *      Workshop SCHLAUBEET, Version 0, d.i. die Minimalversion:
 *      Ein-/Ausschalten einer LED in Abhängigkeit eines Tasters.
 *
 *  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_BUTTON              2
                                /* Pin-Id (Nummer) des Pins, an dem der Taster
                                 * angeschlossen ist.
                                 */
#define PID_PUMP                LED_BUILTIN
                                /*
                                 * 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.
                                 */
#define CYCLETIME_LOOP_MS       100
                                /*
                                 * Alle wie viele Millisekunden soll das Programm
                                 * ausgeführt werden?
                                 */

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

/*****************************************************************************
*/
bool    is_moisture_low()
/*
 * Wir retournieren `true`, wenn der Taster gedrückt ist, false sonst.
 *
 * Ein nicht gedrückter Taster bedeutet also, es ist feucht genug.
 * Ein gedrückter Taster hingegen bedeutet, es ist zu trocken.
 *
 *****************************************************************************/
{
    return digitalRead( PID_BUTTON) == 0;
                                /* Der Schalter ist 0-aktiv, schaltet also
                                 * gegen Ground, deshalb lesen wir eine Null,
                                 * wenn er gedrückt ist.
                                 */
}

/*****************************************************************************
 */
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_BUTTON, INPUT_PULLUP);
    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.
 *
 ******************************************************************************/
{
    if (is_moisture_low())
    {
                                /* Feuchtesensor meldet Trockenheit.
                                 * Wir schalten die Pumpe ein und
                                 * wechseln nach S_WATERING.
                                 *
                                 * In der Version 0 von SCHLAUBEET liefert
                                 * die Funktion `is_moisture_low()` den
                                 * Zustand des Tasters. Ist er gedrückt,
                                 * bedeutet das, dass es zu trocken ist.
                                 */
            pump_on();
    }
    else
    {
            pump_off();
    }

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

7.2 *Schlaubeet* mit Feuchtesensor und Pumpe und Berücksichtigung von Zuständen (schlaubeetWs2.ino)

Diese Version realisiert Zustände. Wenn eine Bedigung wie ein bestimmter Sensorwert erfüllt ist, wird von einem Zustand in einen anderen gewechselt:

```
1.  switch( Statevalue)
2.  {
3.  case S_MONITORING:
4.      if (is_moisture_low( humidity_100))
5.      {
6.          pump_on();
7.          Statevalue = S_WATERING;
```

```
 8.      }
 9.      break;
10.
11.  case S_WATERING:
12.      if (is_moisture_ok( humidity_100))
13.      {
14.          pump_off();
15.          Statevalue = S_MONITORING;
16.      }
17.      break;
18.
19.  default:
20.      break;
21.  }
```

`Statevalue` ist am Beginn des Programms als `int` (Ganzzahl) definiert und Mit dem Wert `S_MONITORING` initialisiert.

Wird `Statevalue` nicht verändert, dann wird immer wieder ausschließlich der Code in diesem `case` ausgeführt. Unterschreitet hier der Wert `humidity_100` (der weiter oben im Programm vom Sensor gelesen worden ist – s. die vollständige Programmversion weiter unten), dann wird die Pumpe eingeschaltet und der Zustand wechselt nach `S_WATERING`.

In diesem Zustand bleibt das Programm so lange, bis die Feuchte einen gewissen Wert erreicht (die Pumpe läuft ja). Dann wird die Pumpe ausgeschaltet und nach `S_MONITORING` gewechselt.

Nun zur vollständigen Version des Programms:

```
/*****************************************************************************
 *  PROJECT:
 *      Experiment SCHLAUBEET
 *
 *  FILE:
 *      arduino/sketchbooks/schlaubeetE2/schlaubeetWs2.ino
 *
 *  SYNOPSIS:
 *      Workshop SCHLAUBEET:
 *      -   Ein-/Ausschalten einer Pumpe in Abhängigkeit des Messwertes eines
 *          Feuchtesensors.
 *
 *      -   Realisierung von Zuständen in der Funktion `loop()`.
 *
 *  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
 *
```

```
 ***************************************************************************/

#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 S_MONITORING            1
                                /*
                                 * State, in dem nur die Sensoren gelesen werden.
                                 * In diesem State wird nicht bewässert, läuft die
                                 * Pumpe also nicht.
                                 *
                                 * Wir verwenden diesen Wert in der Funktion Loop() in
                                 * einem switch-Statement, um den entsprechenden
                                 * case-Block zu kennzeichnen, der beim Monitoring
                                 * auszuführen ist.
                                 */
#define S_WATERING              2
                                /*
                                 * State, in dem bewässert wird.
                                 *
                                 * Wir verwenden diesen Wert in der Funktion Loop() in
                                 * einem switch-Statement, um den entsprechenden
                                 * case-Block zu kennzeichnen, der beim Bewässern
                                 * auszuführen ist.
                                 */

#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

/*****************************************************************************
 *  D a t a
 *
 *****************************************************************************/

unsigned char   Statevalue = S_MONITORING;
                                /*
                                 * State-Variable. Sie kann die Werte annehmen, die
                                 * wir weiter oben definiert haben: S_MONITORING,
                                 * S_WATERING, ...
                                 *
                                 * Wir staten also mit dem Monitoring der Eingänge
                                 * des Arduino.
                                 */
```

```
/*****************************************************************************
 *  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));
    return humidity_100;
}


/*****************************************************************************
*/
bool    is_moisture_low( int humidity_100)
/*
 * 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)
/*
 * 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 Arduno 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);

    /*
     * State Machine, die in ihren case-Blöcken in ABhängigkeit der Beurteilung
     * der Feuchte das jeweils notwendige tut.
     */
    switch( Statevalue)
    {
    case S_MONITORING:
                                    /*
                                     * Lesen des Feuchtesensor Wechsel nach
                                     * S_WATERING, wenn er anzeigt, dass es
                                     * zu trocken ist.
                                     */
        if (is_moisture_low( humidity_100))
        {
                                        /* Feuchtesensor meldet Trockenheit.
                                         * Wir schalten die Pumpe ein und
                                         * wechseln nach S_WATERING.
                                         */
            pump_on();
            Statevalue = S_WATERING;
        }
        break;

    case S_WATERING:
        if (is_moisture_ok( humidity_100))
        {
                                        /* Feuchtesensor meldet Feuchte.
                                         * Wir schalten die Pumpe aus und
                                         * wechseln nach S_MONITORING.
                                         */
            pump_off();
            Statevalue = S_MONITORING;
        }
        break;

    default:
        break;
    }

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

7.3 *Schlaubeet* für Fortgeschrittene (schlaubeetWs3.ino)

Wenn Sie tiefer in die Programmierung einsteigen möchten, dann sind Sie in diesem Kapitel genau richtig. Aber was heißt *tiefer einsteigen*? Nun, vielleicht möchten Sie ein richtiges Gartenbeet, vielleicht ein Hochbeet realisieren, mit richtiger Erde und richtigen Pflanzen darin. Dann möchten Sie, damit's keine Überschwemmung gibt, weil die Feuchtigkeit eine gewisse Zeit benötigt, um zum Sensor vorzudringen, eine Durchfeuchtungsphase realisieren. Neben dem Monitoring und dem Watering also ein weiterer Zustand, das Moisturing.

So etwas lässt sich sehr gut über eine sogenannte State Machine realisieren, ein Art zu programmieren, die sich besonders gut eignet, mit Zuständen umzugehen

Der *Arduino Uno R4 Wifi* besitzt eine LED-Matrix, die sehr praktisch ist für die Anzeige des Betriebszustandes und verschiedener Messwerte. Dafür steht die Library[2] TIDENT zur Verfügung, die das und noch vieles mehr bietet.

7.3.1 Applikation

```
/*****************************************************************************
 *  PROJECT:
 *      Workshop SCHLAUBEET
 *
 *  FILE:
 *      arduino/sketchbooks/schlaubeetWs3/schlaubeetWs3.ino
 *
 *  SYNOPSIS:
 *      Workshop SCHLAUBEET, Version 3:
 *      Ein-/Ausschalten einer Pumpe in Abhängigkeit eines Feuchtesensors.
 *      -   Fortgeschrittenem Handling möglicher Zustände
 *      -   Handling weiterer Zustände wie S_WATER_LOW
 *      -   Anzeige des Bertriebszustandes und von Messwerten auf der LED-Matrix
 *          des Arduino (erfordert die Labrary TRIDENT
 *      -   Sensoren und Pumpen aus der Library TRIDENT
 *
 *  Diese Version realisiert die Annehmlichkeiten der LED-Matrix-Anzeige und
 *  bedient sich vorprogrammierter Sensoren, die in der Library TRIDENT
 *  verfügbar sind.
 *
 *  PRELIMINARIES:
 *      -   Dieses Programm setzt einen Arduino Uno (oder Nano) voraus.
 *
 *      -   Dieser Sketch benötigt einen 2. Feuchtesensor, der zur Überwachung des
 *          Wasservorrats im Wassertank platziert werden muss.
 *
 *  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.
 *
 *      Kommenare, 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
 *      um 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.
```

2 Eine Library ist eine Sammlung von Funktionen, das Ihnen als Programmierer das eine oder andere an Arbeit abnehmen kann. Eine Library besteht dabei aus mindestens einem File und ist einfach zu installieren.

```
 */
void print_banner()
{
    const char *strings[] =
    {
        "*****************************************************************************",
        "*  REVISION HISTORY:",
        "*      Ver. 2.0.0 | 2024-05-07/FGE:",
        "*          -   Umfangreiche Überarbeitung.",
        "*",
        "*      Ver. 1.0.2 | 2024-03-05/FGE:",
        "*          -   MOISTURETHRESHOLD -> MOISTURETHRESHOLD_100",
        "*          -   MOISTURETHRESHOLD_HYSTERESIS_100 eingeführt",
        "*          -   Anzeige auf LED-Matrix per SETUP_LEDMATRIX_IS_AVAILABLE konf'bar.",
        "*",
        "*      Ver. 1.0.1 | 2024-03-04/FGE:",
        "*          -   MOISTURETRESHOLD eingeführt.",
        "*          -   Parametertypen von `is_moisture_low()` und `is_moisture_ok()` von ",
        "*              `unsigned` auf `signed` geändert.",
        "*          -   Anzeige der Feuchte auf der LED-Matrix.",
        "*",
        "*      Ver. 1.0.0 | 2024-02-28/FGE:",
        "*          -   Detailliertere Kommentare.",
        "*",
        "*      2024-02-25/FGE:",
        "*          -   Feuchtesensor 'Soil Moisture Sensor Hygrometer Module V1.2 Capacitive'",
        "*              eingebunden.",
        "*          -   Ob die Pumpe noch mittels interner LED simuliert wird oder eine",
        "*              reale Pumpe angesteuert wird, kann/muss per PID_PUMP konfiguriert",
        "*              werden.",
        "*",
        "*      2024-02-19/FGE:",
        "*          -   Bug Fixes:",
        "*              -   LED_BUILTIN statt einfach 13, weil auf dem Arduino Uno Wifi Rev2",
        "*                  die interne LED auf D25 liegt.",
        "*",
        "*      2024-02-08/FGE:",
        "*          Erstversion.",
        "*****************************************************************************",
        NULL
    };
    for (int i = 0; strings[ i]; i++)
        Serial.println( strings[ i]);
}
/*
 ****************************************************************************/

/*****************************************************************************
 *  M a c r o s
 ****************************************************************************/

#define __DEBUG__               1
                                /* Wir definieren hier, ob im Lauf des Programms
                                 * besondere Diagnosen aktiviert werden sollen, wenn
                                 * es solche gibt. Ein Beispiel hierfür sind Daten,
                                 * die wir am Serial Monitor der Arduino IDE anzeigen
                                 * möchten.
                                 *
                                 * Wenn dieses Macro also 1 ist, dann werden Daten über
                                 * die serielle Schnittstelle an die Arduino IDE
                                 * auf dem Entwicklungs-PC gesendet, die mittels
                                 * Tools -> Serial Monitor eingesehen werden können.
                                 * Nicht vergessen: Dort die Baudrate 115200
                                 * einstellen!
                                 */
#define SETUP_LEDMATRIX_IS_AVAILABLE    1
                                /*
                                 * 0...Keine Anzeige auf LED-Matrix
                                 * 1...Auf LED-Matrix wird Heartbeat und Feuchtigkeitslevel
```

```
                                 *     angezeigt. Erfordert neben den entsprechenden
                                 *     Arduino-Libraries auch die FHV-Library `trident`.
                                 */
#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 das Relais für 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: Wird 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_MOISTURESENSOR      A0
                                /*
                                 * Pin-Id des analogen Eingangs, an den der analoge
                                 * Feuchtesensor angeschlossen ist.
                                 */
#define PID_WATERLEVELSENSOR    A1

#define MOISTURETRESHOLD_100                50
#define MOISTURETHRESHOLD_HYSTERESIS_100    10

/*****************************************************************************
 *  I n c l u d e s
 *
 *****************************************************************************/
#include <trident.h>


/*****************************************************************************
 *  D a t a
 *
 *****************************************************************************/

unsigned long Loopcounter = -1;
                               /*
                                * Durch diese Initialisierung ist der Loopcounter,
                                * der am Beginn von`loop()` inkrementiert wird,
                                * 0-based.
                                */
ArduinoUnoR4Matrix  LedMatrix( true);

D3StateId       Stateid( D3StateValues::SV__MONITORING);
                               /*
                                * State-Variable. Sie kann die Werte annehmen, die
                                * wir weiter oben definiert haben: S_MONITORING,
                                * S_WATERING, ...
                                *
                                * Wir starten also mit dem Monitoring der Eingänge
                                * des Arduino.
                                */

D3SensorSoil_Analog \
                Moisturesensor( "D3SensorSoil_Analog", PID_MOISTURESENSOR);
```

```
D3SensorWaterlevelAnalog \
                Waterlevelsensor( "D3SensorWaterlevel",  PID_WATERLEVELSENSOR);
D3Pump          Pump( "D3Pump", PID_PUMP);

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

/*****************************************************************************
 */
void setup()
/*
 *****************************************************************************/
{
    LedMatrix.set_up();

#if __DEBUG__ == 1
    Serial.begin( 115200);
                                    /*
                                     * Initialisierung der seriellen Schnittstelle
                                     * für die Ausgabe von zusätzlichen Informationen,
                                     * die bei der Fehlersuche hilfreich sein könnten.
                                     */
    print_banner();
    delay( 3000);

    Serial.println( "setup(): Exit now.\n");
#endif
}

/*****************************************************************************
*/
void loop()
/*
 * Funktion, die der Arduno 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.
 *
 *****************************************************************************/
{
    Loopcounter++;

    long    runtime_ms = millis();
#if __DEBUG__ == 1
    Serial.print( "State = ");
    Serial.print( Stateid.name());
    Serial.println( ".");
#endif

    /*
     * Feuchtesensoren lesen
     */
     Moisturesensor.write_to_hardware();
     Moisturesensor.read_from_hardware();

     Waterlevelsensor.write_to_hardware();
     Waterlevelsensor.read_from_hardware();

    /*
     * LED-Matrix löschen
     */
     LedMatrix.clear();

    /*
     * Heartbeat
     */
```

```
 LedMatrix.led( 0, Loopcounter % 12, 1);

/*
 * Grad der Feuchtigkeit auf LED-Matrix anzeigen
 */
 LedMatrix.percents_to_leds_in_a_row( 1, Moisturesensor.humidity_100());

 /*
  * LED-Matrix ausgeben
  */
 LedMatrix.write_to_hardware();

/*
 * State Machine, die in ihren case-Blöcken das jeweils notwendige tut.
 */
switch( Stateid.value())
{
default:
case D3StateValues::SV__MONITORING:
                                /*
                                 * Lesen des Feuchtesensor Wechsel nach
                                 * S_WATERING, wenn er anzeigt, dass es
                                 * zu trocken ist.
                                 */
    if (Moisturesensor.is_moisture_low())
    {
                                    /* Feuchtesensor meldet Trockenheit.
                                     * Wir schalten die Pumpe ein und
                                     * wechseln nach S_WATERING.
                                     */
        Pump.on();
        Stateid.value( D3StateValues::SV__WATERING);
    }
    break;

case D3StateValues::SV__WATERING:
    if (Waterlevelsensor.is_waterlevel_low())
    {
                                    /* Kein Wasser im Tank.
                                     */
        Pump.off();
        Stateid.value( D3StateValues::SV__ERROR_LOW_WATER);
    }
    else
    {
        if (Moisturesensor.is_moisture_ok())
        {
                                        /* Feuchtesensor meldet Feuchte.
                                         * Wir schalten die Pumpe aus und
                                         * wechseln nach S_MONITORING.
                                         */
            Pump.off();
            Stateid.value( D3StateValues::SV__MONITORING);
        }
    }
    break;

case D3StateValues::SV__ERROR_LOW_WATER:
    if (Waterlevelsensor.is_waterlevel_ok())
    {
                                    /* Wieder Wasser im Tank.
                                     */
        Stateid.value( D3StateValues::SV__MONITORING);
    }
    break;
}

/*
 * Zykluszeit sicherstellen
```

```
     */
    runtime_ms = millis() - runtime_ms;
    long sleeptime_ms = CYCLETIME_LOOP_MS - runtime_ms;
    delay( max( 0, sleeptime_ms));
}
```

7.3.2 Library

Wir verwenden die Library TRIDENT wie beschrieben in Kap. 8 auf S. 37.

8 Anhang: Die Library TRIDENT

Diese Library bietet hauptsächlich komfortablen Zugriff auf Peripherie wie (momentan hauptsächlich) Sensoren.

8.1 File *trident.h*

```
/************************************************************
 *  FILE:
 *      /atlantis/s/swdevmnt/tau/tebe/arduino/sketchbooks/libraries/trident/trident.h
 *
 *  SYNOPSIS:
 *      Haupt-File der trident-Library.
 *
 *  PRELIMINARIES:
 *      Folgende Libs sind zu installieren, um trident nützen zu können:
 *      -   Adafruit LPS2X
 *      -   Adafruit SGP30 Sensor
 *      -   Adafruit BME680 Library
 *      -   Adafruit seesaw Library
 *
 *      Diese Librarys installieren ihrerseits Abhängigkeiten.
 *
 *  REVISION HISTORY:
 *      2024-02-01/FGE: Erstversion.
 *
 ************************************************************/

#ifndef _TRIDENT_H
#define _TRIDENT_H

#include <Arduino.h>

#include <d3actuators.h>
#include <d3logging.h>
#include <d3sensors.h>
#include <d3signalling.h>
#include <d3statedefs.h>
#include <d3time.h>

#include <d3ee.h>

#endif
```

8.2 File *d3actuators.h*

```
/*****************************************************************************
 *  FILE:
 *      /atlantis/s/swdevmnt/tau/tebe/arduino/sketchbooks/libraries/trident/d3actuators.h
 *
 *  SYNOPSIS:
 *      Sensoren:
 *          -   Base Class
 *          -   LPS22
 *          -   BME688
 *          -   SoilSensor
 *
 *  REVISION HISTORY:
 *      2024-02-01/FGE: Erstversion.
 *
 *****************************************************************************/

#ifndef _D3ACTUATORS_H
#define _D3ACTUATORS_H

#include <Wire.h>
#include "SparkFun_Qwiic_Relay.h"

#include <d3time.h>

class D3Actuator
{
public:
    D3Actuator( const char *p_id)
    : _p_id( p_id)
    , _is_inited( false)
                                    {}

    void            is_set_up( bool arg)
                                    {     _is_inited = arg;
                                    }
                                    /*
                                     * Ist der Aktuator initialisiert worden?
                                     *
                                     * Hintergrund: Nicht alle Aktuatoren dürfen
                                     * bei der Instanzierung bereits initialisiert
                                     * werden, die Arduino-Lib mag das nicht so
                                     * gerne. Die Initialisierung darf meist erst in
                                     * der Funktion `setup` erfolgen. Und das
                                     * muss halt auch überprüft werden können.
                                     */

    bool            is_set_up() const
                                    {    return _is_inited;
                                    }

    const char      *p_id() const
                                    {    return _p_id;
                                    }

    virtual void    write_to_hardware() = 0;
                                    /*
                                     * Hardware triggern.
                                     */
    virtual void    read_from_hardware() = 0;
                                    /*
                                     * Hardware lesen.
                                     */
    virtual ~D3Actuator()
                                    {}

private:
```

```
    const char      *_p_id;
    bool            _is_inited;
};


/*****************************************************************************
 *
 * BAse CLass für Pumpen.
 */
class D3Pump : public D3Actuator
{
public:
    D3Pump( const char *p_id)
    : D3Actuator( p_id)
    , _time_last_watering_started( 0)
    , _time_last_watering_stopped( 0)
    , _is_on( false)
    {}

    /*****************************************************************************
     */
    bool    is_on() const
    {   return _is_on;
    }

    /*****************************************************************************
     */
    void off( bool dont_track_time=false)
    /*
     * Merkt die Pumpe fürs Ausschalten vor bei Ausführung von write_to_hardware().
     *
     * Parameters:
     *      dont_track_time (bool) = false:
     *          Zeiten nicht protokollieren. Zeiten dürfen bspw
     *          beim manuellen Bewässern nicht protokolliert
     *          werden, weil das sonst das Tracking der Zeiten
     *          im Automatikbetrieb durcheinanderbringt.
     *****************************************************************************/
    {
        _is_on = false;
        if (!dont_track_time)
            _time_last_watering_stopped = millis() / 1000;
    }

    /*****************************************************************************
     */
    void on( bool dont_track_time=false)
    /*
     * Merkt die Pumpe fürs Einschalten vor bei Ausführung von write_to_hardware().
     *
     * Parameters:
     *      dont_track_time (bool) = false:
     *          Zeiten nicht protokollieren. Zeiten dürfen bspw
     *          beim manuellen Bewässern nicht protokolliert
     *          werden, weil das sonst das Tracking der Zeiten
     *          im Automatikbetrieb durcheinanderbringt.
     *****************************************************************************/
    {
        _is_on = true;
        if (!dont_track_time)
            _time_last_watering_started = millis() / 1000;
    }

    virtual ~D3Pump()
    {}

private:
    unsigned long   _time_last_watering_started;
    unsigned long   _time_last_watering_stopped;
```

```
    bool            _is_on;
};


/*****************************************************************************
 *
 * Pumpe, die (ziemlich sicher über ein Relais) an einem DOut hängt.
 */
class D3PumpAtDOut : public D3Pump
{
public:
    D3PumpAtDOut( const char *p_id, byte pid_trigger)
    : D3Pump( p_id)
    , _pid_trigger( pid_trigger)
    {
        set_up();
    }

    /*****************************************************************************
     */
    void                set_up()
    {
        pinMode( _pid_trigger, OUTPUT);
        is_set_up( true);
    }

    /*****************************************************************************
     */
    void                write_to_hardware() override
    {
        digitalWrite( _pid_trigger, is_on() ? HIGH : LOW);
    }

    /*****************************************************************************
     */
    void                read_from_hardware() override
    {   return;
    }

    virtual ~D3PumpAtDOut()
    {}

private:
    byte                _pid_trigger;
};


/*****************************************************************************
 *
 * Pumpe, die über das Quad-Realy von Sparkfun geschaltet wird.
 */
class D3PumpAtQwiicQuadRelayFromSparkfun : public D3Pump
/*
 * Important:
 *      Der Qrwiic-Bus ist auf dem Arduino Uno R4 Wifi  n u r   ü b e r   W i r e 1
 *      und nicht über Wire erreichbar. Man fällt leicht auf dieses "Falle" herein,
 *      wenn man sich an Examples und Totorials von Sparkun orientiert!
 *
 *      Die Initialisierung von  W i r e 1  muss bei Ausführung von  i n i t ( )
 *      bereits erfolgt sein!
 *
 *****************************************************************************/
{
public:
    D3PumpAtQwiicQuadRelayFromSparkfun( const char *p_id, int relaynumber, Qwiic_Relay& quadrelaisu-
nit, bool dont_set_up_quadrelayunit_in_ctor=false)
    : D3Pump( p_id)
    , _relaynumber( relaynumber)
    , _is_hardware_on( false)
```

```
    , _quadrelayunit( quadrelaisunit)
    {
        if (!dont_set_up_quadrelayunit_in_ctor)
            set_up();
    }

    /**************************************************************************
     */
    void            set_up()
    /*
     * Important:
     *      Die Initialisierung von  W i r e 1  muss bei Ausführung dieser Methode
     *      bereits erfolgt sein!
     *
     **************************************************************************/
    {
        is_set_up( true);
        return;
    }

    /**************************************************************************
     */
    bool    is_hardware_on() const
    {
        return _is_hardware_on;
    }

    /**************************************************************************
     */
    void    write_to_hardware()
    {
        if (is_on())
        {
            //~ Serial.print( "Switch relay "); Serial.print( _relaynumber); Serial.println( "
on.");
            _quadrelayunit.turnRelayOn( _relaynumber);
        }
        else
        {
            //~ Serial.print( "Switch relay "); Serial.print( _relaynumber); Serial.println( "
off.");
            _quadrelayunit.turnRelayOff( _relaynumber);
        }

        return;
    }

    /**************************************************************************
     */
    void    read_from_hardware()
    {
        _is_hardware_on = _quadrelayunit.getState( _relaynumber);
        return;
    }

    virtual ~D3PumpAtQwiicQuadRelayFromSparkfun()
    {}

private:
    const char      *_p_id;
    int             _relaynumber;
    int             _i2caddr_of_quadrelayunit;
    unsigned long   _time_last_watering_started;
    unsigned long   _time_last_watering_stopped;
    bool            _is_on;
    bool            _is_hardware_on;
    Qwiic_Relay&    _quadrelayunit;
};
#endif
```

8.3 File *d3ee.h*

```
/*****************************************************************
 *  FILE:
 *      /atlantis/s/swdevmnt/tau/tebe/arduino/sketchbooks/libraries/trident/d3ee.h
 *
 *  SYNOPSIS:
 *      Everything Else :-)
 *
 *  REVISION HISTORY:
 *      2024-02-01/FGE: Erstversion.
 *
 *****************************************************************************/

/*****************************************************************************
 */
class D3Heartbeat
/*
 * Synopsis:
 *      Realisierung eines Heartbeats an einem beliebigen Pin.
 *
 * Parameters:
 *      pid (int):
 *          Pin-Id.
 *
 *      frequency (byte):
 *          Frequenz. Die Frequenz des Heartbeats ist also uinabhängig von der
 *          Zykluszeit, solange die kurz genug ist.
 *
 *          Default-Value = 2.
 *
 *****************************************************************************/
{
public:
    D3Heartbeat( int pid, unsigned char frequency=2)
    : _pid( pid)
    , _frequency( frequency)
    , _togglevalue( 0)
    {
        pinMode( _pid, OUTPUT);
    }

    virtual ~D3Heartbeat() {}

    unsigned char   frequency() const { return _frequency; }
    void            frequency( unsigned char frequency) { _frequency = frequency; }
    void            trigger()
    {
        unsigned long now_ms = millis();
        if (now_ms - _time_of_last_beat_ms >= 1000.0/_frequency)
        {
            digitalWrite( _pid, _togglevalue = !_togglevalue);
            _time_of_last_beat_ms = now_ms;
        }
    }

private:
    int             _pid;
    unsigned char   _frequency;

    unsigned long   _time_of_last_beat_ms;
    unsigned char   _togglevalue;
};
```

8.4 File *d3sensors.h*

```
/*****************************************************************************
 *  FILE:
 *      /atlantis/s/swdevmnt/tau/tebe/arduino/sketchbooks/libraries/trident/d3sensors.h
 *
 *  SYNOPSIS:
 *      Sensoren:
 *          -   Base Class
 *          -   LPS22
 *          -   BME688
 *          -   SoilSensor
 *
 *  REVISION HISTORY:
 *      2024-02-01/FGE: Erstversion.
 *
 *****************************************************************************/

#ifndef _D3SENSORS_H
#define _D3SENSORS_H

#include <Arduino.h>
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_LPS2X.h>
#include <Adafruit_BME680.h>
#include <Adafruit_seesaw.h>

#include <d3logging.h>
#include <d3time.h>

#include <vector>

/*****************************************************************************
 */
#define SEALEVELPRESSURE_HPA (1013.25)

/*****************************************************************************
 */
class D3Sensor
/*
 * SYNOPSIS:
 *      Basis-Klasse aller Sensoren.
 *
 * Parameters:
 *      p_id (const char *)::
 *          Identifikation des Sensors. Kann beim Debugging hilfreich sein.
 *
 *      versionstring (String):
 *          Falls ein Sensor keinen Versions-String oder Versions-Nummer liefern
 *          kann, kann er hier angegebeen werden. Vllt ist das mal nützlich.
 *
 * Protokoll des Sensor-Handlings:
 * -    Schreiben auf die Hardware, wie das z.B. bei Ultraschallsensoren notwendig
 *      ist.
 *
 * -    Lesen der Messwerte von der Hardware, wandeln falls notwendig und speichern
 *      in der Instanz.
 *
 *****************************************************************************/
{
public:
    class SData
    {
    };

public:
    D3Sensor( const char *p_id, String versionstring = "0.0")
```

```
    : _p_id( p_id)
    , _value( -1.)
    , _is_inited( false)
    , _is_working_properly( true)
    , _is_bypassed( false)
    , _versionstring( versionstring)
    {}

    virtual         ~D3Sensor() {}

/*
* I n s p e c t o r s
*/
    const char      *p_id() const
                                    {    return _p_id;
                                    }
                                    /*
                                     * Id des Sensors.
                                     */
    virtual String  dumps() const { return String( "No dump available!"); }
                                    /*
                                     * Alles Wichtige des Sensors in einen String
                                     * packen und retournieren.
                                     */
    virtual bool    is_analog() const
                                    {    return false;
                                    }

    bool            is_bypassed() const
                                    {   return _is_bypassed;
                                    }

    bool            is_inited() const // DEPRECATED
                                    {    return _is_inited;
                                    };

    bool            is_set_up() const
                                    {    return _is_inited;
                                    };

    bool            is_working_properly() const
                                    {    return is_bypassed() ? true : _is_working_properly;
                                    };

    double          value() const
                                    {    return _value;
                                    }
                                    /*
                                     * Lesen des Messwertes durch dei App.
                                     * Kann nicht für alle Sensorarten ausreichen.
                                     * Solche Sensoren müssen dann das API selber
                                     * erweitern.
                                     */

/*
* M u t a t o r s
*/
public:
    void            is_bypassed( bool arg)
                                    {   _is_bypassed = arg;
                                    }

    virtual void    write_to_hardware() = 0;
                                    /*
                                     * Hardware triggern.
                                     */
    virtual void    read_from_hardware() = 0;
                                    /*
                                     * Hardware lesen.
```

```
                                    */
    void            value( double value)
                                    {   _value = value;
                                    }
                                    /*
                                     * Schreiben des Messwertes in die Instanz
                                     * durch die Basis-Klassen.
                                     */
protected:
    void            is_inited( bool arg) // DEPRECATED
                                    {   _is_inited = arg;
                                    }
                                    /*
                                     * Ist der Sensor initialisiert worden?
                                     *
                                     * Hintergrund: Nicht alle Sensoren dürfen
                                     * bei der Instanzierung bereits initialisiert
                                     * werden, die Arduino-Lib mag das nicht so
                                     * gerne. Die Initialisierung darf erst in
                                     * der Funktion `setup` erfolgen. Und das
                                     * muss halt auch überprüft werden können.
                                     */
    void            is_set_up( bool arg)
                                    {   _is_inited = arg;
                                    }
                                    /*
                                     * Ist der Sensor initialisiert worden?
                                     *
                                     * Hintergrund: Nicht alle Sensoren dürfen
                                     * bei der Instanzierung bereits initialisiert
                                     * werden, die Arduino-Lib mag das nicht so
                                     * gerne. Die Initialisierung darf erst in
                                     * der Funktion `setup` erfolgen. Und das
                                     * muss halt auch überprüft werden können.
                                     */
    void            is_working_properly( bool arg)
                                    {   _is_working_properly = arg;
                                    }


private:
    const char      *_p_id;
    double          _value;

    bool            _is_inited;
    bool            _is_working_properly;
    bool            _is_bypassed;

    String         _versionstring;
};


/************************************************************
 */
class D3SensorButton : public D3Sensor
/*
 * Synopsis:
 *      Facade (Design Pattern) um einen low-aktiven Schalter.
 *
 * Parameters:
 *      p_id (const char *):
 *          Id des Sensors.
 *
 *      pinid (int):
 *          Id des Pins (wie zu finden auf dem Arduino), an dem der
 *          Taster angeschlossen ist.
 *
 *      dont_set_up (bool) = false:
 *          Nicht schon bei der Instanzieren initialisieren, der User
```

```
 *          wird die Intialisierung in der FUnktion `setup` vornehmen.
 *
 *          Der Default-Wert ist `false`, weil dieses verschobene
 *          Initialisieren für einfache DInps nicht nötig ist.
 *
 * Usage:
 *      D3SensorSwitch switch_manual_watering( "Switch.ManualWatering");
 *
 *      void loop()
 *      {
 *          switch_manual_watering.write_to_hardware(); // Bei diesem Sensor optional
 *          switch_manual_watering.read_from_hardware();
 *          ... = switch_manual_watering.is_on();
 *      }
 *
 * Revision History:
 *      2024-02-07/FGE: Erstversion.
 *
 ************************************************************/
{
public:
    D3SensorButton( const char *p_id, int pid, bool dont_set_up=false)
    : D3Sensor( p_id)
    , _pid( pid)
    {
        if (!dont_set_up)
            set_up();
    }

    void set_up()
    {
        pinMode( _pid, INPUT_PULLUP);
        value( 1);
        is_set_up( true);
    }

    void pid( int pid)
                                    {   _pid = pid;
                                        pinMode( _pid, INPUT_PULLUP);
                                    }

    void write_to_hardware()        {}

    void read_from_hardware()
                                    {   value( digitalRead( _pid));
                                    }

    bool is_on() const              {   return (int)value() == 0;
                                    }

    virtual ~D3SensorButton()       {}

private:
    int     _pid;
};

/************************************************************
 */
class D3SensorLPS22 : public D3Sensor
/*
 *  Synopsis:
 *      Facade (Design Pattern) um die LPS22-Lib von Adafruit.
 *
 *  Der LPS22 ist ein Luftdrucksensor.
 *
 *  Usage:
 *      D3SensorLPS22 sensor( "LPS22.1", true);
 *
 *      void setup()
```

```
 *      {
 *          sensor.set_up()
 *      }
 *
 *      void loop()
 *      {
 *          sensor.write_to_hardware(); // Bei diesem Sensor optional
 *          sensor.read_from_hardware();
 *          ... = sensor.tempr();
 *          ... = sensor.pressure_hPa();
 *      }
 *
 *  Revision History:
 *      2024-02-01/FGE: Erstversion.
 *
 ***************************************************************/
{
public:
    D3SensorLPS22( const char *p_id, bool dont_set_up=false)
    : D3Sensor( p_id)
    , _tempr( -273.15)
    , _pressure_hPa( 0)
    , _is_inited( false)
    {
        if (!dont_set_up)
            set_up();
    }

    void set_up()
    {
        if (_lps22.begin_I2C( LPS2X_I2CADDR_DEFAULT, &Wire1))
        {
            _lps22.setDataRate( LPS22_RATE_10_HZ);

            if (Serial)
            {   Serial.print( "The data rate of "); Serial.print( this->p_id()); Serial.print( " is
set to: ");
            }

            switch( _lps22.getDataRate())
            {
            case LPS22_RATE_ONE_SHOT:   Serial.println( "One Shot / Power Down"); break;
            case LPS22_RATE_1_HZ:       Serial.println( "1 Hz"); break;
            case LPS22_RATE_10_HZ:      Serial.println( "10 Hz"); break;
            case LPS22_RATE_25_HZ:      Serial.println( "25 Hz"); break;
            case LPS22_RATE_50_HZ:      Serial.println( "50 Hz"); break;
            case LPS22_RATE_75_HZ:      Serial.println( "75 Hz"); break;
            }

            _is_inited = true;
        }
        else
            if (Serial)
                Serial.println( "Failed to init LPS22!");
    }

    bool    is_inited() const { return _is_inited; }


    void write_to_hardware() {}

    void read_from_hardware()
    {
        _lps22.getEvent( &_pressureevent, &_temprevent);

        _tempr = _temprevent.temperature;
        _pressure_hPa = _pressureevent.pressure;
    }
```

```
    double tempr() const { return _tempr; }
    double pressure_hPa() const { return _pressure_hPa; }

    virtual ~D3SensorLPS22() {}

private:
    Adafruit_LPS22  _lps22;
    sensors_event_t _temprevent;
    sensors_event_t _pressureevent;
    double          _tempr;
    double          _pressure_hPa;
    bool            _is_inited;
};


/*************************************************************
 */
class D3SensorBME688 : public D3Sensor
/*
 *  Synopsis:
 *      Facade (Design Pattern) um die BME680-Lib von Adafruit.
 *
 *  Der BME688 ist ein Kombi-Sensor für Temperatur, Luftfeuchtigkeit,
 *  Luftdruck und Luftqualität.
 *
 *  Usage:
 *      D3SensorBME688 sensor( "BME688.1", true);
 *
 *      void setup()
 *      {
 *          sensor.set_up()
 *      }
 *
 *      void loop()
 *      {
 *          sensor.write_to_hardware(); // Bei diesem Sensor optional
 *          sensor.read_from_hardware();
 *          ... = sensor.tempr();
 *          ... = sensor.pressure_hPa();
 *      }
 *
 *  Revision History:
 *      2024-02-01/FGE: Erstversion.
 *
 *************************************************************/
{
public:
    D3SensorBME688( const char *p_id, int address_i2c=0x76, bool dont_set_up=false)
    : D3Sensor( p_id)
    , _address_i2c( address_i2c)
    , _bme( &Wire1) // I2C
    , _tempr( -273.15)
    , _humidity_100( -1)
    , _pressure_hPa( -1)
    , _gas_kOhm( -1)
    , _altitude( -1)
    , _is_inited( false)
    , _clock( 1000)
    , _runtime_getting_data_ms( 0)
    {
        if (!dont_set_up)
            set_up();
    }

    void set_up()
    {
        if (!(_is_inited = _bme.begin( _address_i2c)))
        {
            if (Serial)
```

```
            {
                Serial.println( "Failed to init BME688!");
                return;
            }
        }
        _bme.setTemperatureOversampling(    BME680_OS_8X);
        _bme.setHumidityOversampling(       BME680_OS_2X);
        _bme.setPressureOversampling(       BME680_OS_4X);
        _bme.setIIRFilterSize(              BME680_FILTER_SIZE_3);
        _bme.setGasHeater(                  320, 150);
                                        /* 320 °C für 150 ms
                                         */
        _bme.beginReading();

        is_set_up( true);
    }

    bool    is_inited() const { return _is_inited; }

    void write_to_hardware() {}

    void read_from_hardware()
    {
        _clock.update();
        if (_clock.is_run_down_again())
        {
#if __DEBUG__ == 1
            Serial.print( "Clock is run down @ "); Serial.print( millis()); Serial.println( " ms,
read BME688.");
#endif
            _runtime_getting_data_ms = millis();
            if (!_bme.endReading())
            {
                Serial.println( "Failed to read BME688!");
                _runtime_getting_data_ms = 0;
            }
            else
            {
                _tempr = _bme.temperature;
                _humidity_100 = _bme.humidity;
                _pressure_hPa = _bme.pressure / 100.0;
                _gas_kOhm = _bme.gas_resistance / 1000.0;
                _altitude = _bme.readAltitude( SEALEVELPRESSURE_HPA);

                _runtime_getting_data_ms = millis() - _runtime_getting_data_ms;
            }

            _bme.beginReading();
        }

        return;
    }

    double tempr() const { return _tempr; }
    double humidity_100() const { return _humidity_100; }
    double pressure_hPa() const { return _pressure_hPa; }
    double gas_kOhm() const { return _gas_kOhm; }
    double altitude() const { return _altitude; }

    unsigned long   runtime_getting_data_ms() const { return _runtime_getting_data_ms; }

    virtual ~D3SensorBME688() {}

private:
    int             _address_i2c;
    Adafruit_BME680 _bme;

    double          _tempr;
    double          _humidity_100;
```

```
    double          _pressure_hPa;
    double          _gas_kOhm;
    double          _altitude;

    bool            _is_inited;

    D3Clock         _clock;
    unsigned long   _runtime_getting_data_ms;
};


/*****************************************************************************
 */
class D3SensorSoil : public D3Sensor
/*
 * Synopsis:
 *      Facade (Design Pattern) um die Seesaw-Lib von Adafruit für den I2C-Soil-Sensor.
 *
 * Parameters:
 *      Siehe Basis-Klasse!
 *
 * Revision History:
 *      2024-02-05/FGE:
 *          Erstversion.
 *
 *****************************************************************************/
{
public:
    D3SensorSoil( const char *p_id, String versionstring="?.?")
    : D3Sensor( p_id, versionstring)
    {}

    virtual int     humidity_100() const = 0;

    virtual bool    is_moisture_low() = 0;

    virtual bool    is_moisture_ok() = 0;

    virtual ~D3SensorSoil() {}

private:
};

/*****************************************************************************
 */
class D3SensorSoil_I2C : public D3SensorSoil
/*
 *  Synopsis:
 *      Facade (Design Pattern) um die Seesaw-Lib von Adafruit für den I2C-Soil-Sensor.
 *
 *  Revision History:
 *      2024-02-05/FGE:
 *          Erstversion.
 *
 *****************************************************************************/
{
public:
    D3SensorSoil_I2C( const char *p_id, int address_i2c=0x36, bool dont_set_up=false)
    : D3SensorSoil( p_id)
    , _address_i2c( address_i2c)
    , _seesaw( &Wire1) // I2C
    , _tempr( -273.15)
    , _capacity( -1)
    {
        if (!dont_set_up)
            set_up();
    }

    void set_up()
```

```
    {
        Serial.println( "D3SensorSoil_I2C(): Init soil sensor.");
        is_inited( _seesaw.begin( _address_i2c));
        if (!is_inited())
        {
            is_working_properly( false);
            if (Serial)
            {
                Serial.println( "D3SensorSoil_I2C(): Failed to init soil sensor! :-(");
                return;
            }
        }

        is_working_properly( true);

        int _versionnumber = _seesaw.getVersion();
        _versionnumber = _versionnumber;

        is_set_up( true);

        Serial.println( "D3SensorSoil_I2C(): Inited. :-)");
    }

    void    write_to_hardware()     {}

    void    read_from_hardware()    { _tempr = _seesaw.getTemp(); _capacity = _seesaw.touchRead( 0);
return; }

    double  tempr() const           { return _tempr; }

    double  capacity() const        { return _capacity; }

    bool    is_moisture_low()       { return false; }
                                    /*
                                    * 2DOs:
                                    *       -   Implementieren!
                                    */

    bool   is_moisture_ok()         { return true; }
                                    /*
                                    * 2DOs:
                                    *       -   Implementieren!
                                    */

    virtual ~D3SensorSoil_I2C() {}

private:
    int             _address_i2c;
    Adafruit_seesaw _seesaw;

    double          _tempr;
    double          _capacity;
};

/*****************************************************************************
 */
class D3SensorSoil_Analog : public D3SensorSoil
/*
 * Synopsis:
 *      Facade (Design Pattern) um die einen analogen Eingang, an dem ein analoger
 *      Feuchtesensor hängt.
 *
 * 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.
 *
 * Es wird default der Bereich 20 %...50 % als ordnungsgemäß feucht angenommen.
 * Mit `moisture_min_100()` und `moisture_max_100()` kann dieser Bereich verändert
 * werden und wird es im Ctor von Greenhouse auch.
```

```
 *
 * Der Sensor ist hier gut dokumentiert:
 * https://www.az-delivery.de/en/products/bodenfeuchte-sensor-modul-v1-2
 *
 * Parameters:
 *      p_id (const char *):
 *          Id des Sensors. Wvtl nürzlich fürs Debugging. Wird sonst nicht
 *          gebraucht.
 *
 *      pid:
 *          Pin-Id wie angegeben direkt auf dem Arduino Uno R4.
 *          Die Macros A0, ... A5 sind erlaubte Werte.
 *
 * Revision History:
 *      2024-02-23/FGE:
 *          Erstversion.
 *
 ******************************************************************************/
{
public:
    /**************************************************************************
     */
    D3SensorSoil_Analog( const char *p_id, int pid, bool dont_set_up=false)
    /*
     * Note:
     *      Eine Initialisierung des Pins, an dem der Sensor hängt, ist
     *      bei analogen Eingängen nicht nötig.
     *
     * Important:
     *      Welche Eingänge Analogwerte verarbeiten können, ist von Modell zu
     *      Modell verschieden und muss der Dokumentation des jeweiligen Arduinos
     *      entnommen werden.
     *
     **************************************************************************/
    : D3SensorSoil( p_id, "1.2")
    , _pid( pid)
    , _adcvalue( 0)
    , _moisture_min_100( 20)
    , _moisture_max_100( 50)
    {
        if (!dont_set_up)
            set_up();
    }

    void    set_up()
    {   is_set_up( true);
    }


    String  dumps() const
    {
        String s( "D3SensorSoil_Analog():");
        s += "\n\tPin-id = ";
        s += _pid;
        s += "\n\tAnalog value = ";
        s += _adcvalue;
        s += "\n\tHumidity = ";
        s += humidity_100();
        s += " %";
        return s;
    }

    void  dump_to_serial() const
    {
        String s = dumps();
        Serial.println( s);
    }

    virtual bool    is_analog() const   { return true; }
```

```
    void            write_to_hardware() {}

    /*************************************************************************
     */
    void            read_from_hardware()
    /*
     * Ein völlig TROCKENER Sensor liefert Werte zwischen 790 und 810.
     * Will man keine negativen, ungeclippten Feuchtewerte haben, die dann
     * natürlich geclippt werden, muss der Wert eher größer gewählt werden,
     * also eher 810 statt 790. Man verschenkt sonst einen Teil des Messbereichs.
     *
     * Die Verhältnisse ändern sich nicht, wenn der Sensor in TROCKENER ERDE
     * steckt.
     *
     * Ein NASSER, in Wasser getauchter Sensor liefert Werte zwischen 390 und 400.
     * Will man keine ungeclippten Feuchtewerte größer als 100 % haben, die dann
     * natürlich geclippt werden, muss der Wert eher kleiner
     * gewählt werden, also eher 395 statt 400.
     *
     * Völlig DURCHNÄSSTE, jedoch ausgepresste ERDE liegt zwischen 400 und 500,
     * typisch bei 430.
     *
     * Revision History:
     *      -   2024-05-06/FGE:
     *          Wertebereich geändert von [411, 790] auf [400, 800].
     *
     ************************************************************************/
    {
        _adcvalue = analogRead( _pid);
        const int       value_defining_wetness = 400;
        const int       value_defining_dryness = 800;
        int             humidity_100 = map( \
            _adcvalue, 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.
    #if __DEBUG__ == 1
        d3debug( String( "Humidity = ") + String( humidity_100) + " % <- " + String( _adcvalue));
        if (humidity_100 < 0)
            d3warning( String( "You need to recalibrate your moisture sensor: In `humiditysensorva-
lue_100()` increase the value currently being ") + String( value_defining_dryness) + "!");

        if (humidity_100 > 100)
            d3warning( String( "You need to recalibrate your moisture sensor: In `humiditysensorva-
lue_100` decrease the value currently being ") + String( value_defining_wetness) + "!");
    #endif

        humidity_100 = max( 0, min( 100, humidity_100));
        this->value( humidity_100);
        return;
    }

    int             humidity_100() const
                                {   return int( value());
                                }

    /*************************************************************************
     *
     * Wir nehmen trockene Erde an, wenn wir unter 30 % sind.
     * Wir nehmen nasse Erde an, wenn wir über 50 % sind.
     */
    bool   is_moisture_low()
                                {   return humidity_100() < _moisture_min_100;
                                }
```

```
    /*************************************************************************
     * Wir nehmen trockene Erde an, wenn wir unter 30 % sind.
     * Wir nehmen nasse Erde an, wenn wir über 50 % sind.
     *
     * Note:
     *      Hier ist eine HJysteres von 10% mit drinnen.
     */
    bool   is_moisture_ok()
                                    {   return humidity_100() >= min( 100, (_moisture_max_100 +
10));
                                    }

    void moisture_min_100( int arg) { _moisture_min_100 = arg; }

    void moisture_max_100( int arg) { _moisture_max_100 = arg; }

    virtual ~D3SensorSoil_Analog()
    {}

private:
    int             _pid;
    unsigned int    _adcvalue;
    int             _moisture_min_100;
    int             _moisture_max_100;
};


/*************************************************************************
 */
class D3SensorarraySoil_Analog
/*
 * Synopsis:
 *      Handling einer gewissen Anzahl von Feuchtigkeitssensoren.
 *
 * Fasst eine gewisse Anzahl an Sensoren zu einer Einheit zusammen und richtet
 * sich nach dem "trockensten" Sensor.
 *
 * Revision History:
 *      2024-04-04/FGE:
 *          Erstversion.
 *
 *************************************************************************/
{
public:
    D3SensorarraySoil_Analog()
    {}

    virtual bool    is_analog() const
                                    {   return true;
                                    }

    void            write_to_hardware()
                                    {   for (unsigned int i = 0; i < _ap_sensors.size(); i++)
                                            _ap_sensors[ i]->write_to_hardware();
                                    }

    void            read_from_hardware()
                                    {   d3info( String( "Read ") + String( _ap_sensors.size()) +
String( " sensors."));
                                        for (unsigned int i = 0; i < _ap_sensors.size(); i++)
                                            _ap_sensors[ i]->read_from_hardware();
                                    }

    bool sensor_add( const char *p_id, int pid)
                                    {
                                        _ap_sensors.push_back( new D3SensorSoil_Analog( p_id, pid));
                                        return true;
                                    }
```

```
    String  dumps()
                                    {   String s;
                                        for (unsigned int i = 0; i < _ap_sensors.size(); i++)
                                        {   s += _ap_sensors[ i]->dumps();
                                            s += "\n";
                                        }

                                        return s;
                                    }

    int humidity_100() const
                                    {   int humidity_100 = 100;
                                        int aux;
                                        for (unsigned int i = 0; i < _ap_sensors.size(); i++)
                                            if ((aux = _ap_sensors[ i]->humidity_100()) <
humidity_100)
                                                humidity_100 = aux;

                                        d3info( String( "humidity_100 = ") + String( humidity_100));
                                        return humidity_100;
                                    }

    bool   is_moisture_low()
                                    {   for (unsigned int i = 0; i < _ap_sensors.size(); i++)
                                            if (_ap_sensors[ i]->is_moisture_low())
                                                return true;

                                        return false;
                                    }

    bool   is_moisture_ok()
                                    {   for (unsigned int i = 0; i < _ap_sensors.size(); i++)
                                            if (!_ap_sensors[ i]->is_moisture_ok())
                                                return false;

                                        return true;
                                    }

    virtual ~D3SensorarraySoil_Analog()
                                    {   for (unsigned int i = 0; i < _ap_sensors.size(); i++)
                                            delete _ap_sensors[ i];
                                    }

private:
    std::vector<D3SensorSoil_Analog *>  _ap_sensors;
};


/*************************************************************************
 */
class D3SensorUssPlain : public D3Sensor
/*
 * Synopsis:
 *      Ultraschallsensor HC-SR04.
 *
 * Im Projekt SCHLAUBEET wird ein USS eingesetzt für die Messung der Entfernung
 * der Wasseroberfläche vom wasserdurchlässigen(!) Deckel, in den der USS
 * eingelassen ist.
 *
 * Usage:
 *      D3SensorUssPlain( PID_USS_TRIGGER, PID_USS_ECHO);
 *
 *      void loop()
 *      {
 *          sensor.write_to_hardware();
 *          sensor.read_from_hardware();
 *          ... = sensor.distance_cm();
 *      }
```

```
 *
 * Revision History:
 *      2024-02-14/FGE:
 *          Erstversion.
 *
 ******************************************************************************/
{
public:
    D3SensorUssPlain( const char *p_id, int pid_trigger, int pid_echo, bool dont_set_up=false, int
max_distance_cm=300);

    void            set_up();

    void            write_to_hardware() override;

    void            read_from_hardware() override;

    int             distance_cm() const
                                {   return (int)value();
                                }

    void            distance_max_cm( int arg)
                                {   _max_distance_cm = arg;
                                }
    int             distance_max_cm() const
                                {   return _max_distance_cm;
                                }
    int             distance_complementary_cm() const
                                {   return max( 0, distance_max_cm() - distance_cm());
                                }
    virtual ~D3SensorUssPlain()
    {}

private:
    int     _pid_trigger;
    int     _pid_echo;
    int     _max_distance_cm;
};


/*****************************************************************************
 */
class D3SensorUss : public D3Sensor
/*
 * Synopsis:
 *      Ultraschallsensor HC-SR04.
 *
 * Im Projekt SCHLAUBEET wird ein USS eingesetzt für die Messung der Entfernung
 * der Wasseroberfläche vom wasserdurchlässigen(!) Deckel, in den der USS
 * eingelassen ist.
 *
 * Usage:
 *      D3SensorUss( PID_USS_TRIGGER, PID_USS_ECHO, CYCLETIME_USS_MS);
 *
 *      void loop()
 *      {
 *          sensor.write_to_hardware();
 *          sensor.read_from_hardware();
 *          ... = sensor.distance_cm();
 *      }
 *
 * 2DOs:
 *      Ableiten von D3SensorUssPlain.
 *
 * Revision History:
 *      2024-02-14/FGE:
 *          Erstversion.
 *
 ******************************************************************************/
```

```
{
public:
    D3SensorUss( const char *p_id, int pid_trigger, int pid_echo, unsigned long cycletime_ms, bool
dont_set_up=false, int max_distance_cm=300);

    void            set_up();

    void            write_to_hardware() override;

    void            read_from_hardware() override;

    int             distance_cm() const
    {   return (int)value();
    }

    void            distance_max_cm( int arg)
    {   _max_distance_cm = arg;
    }
    int             distance_max_cm() const
    {   return _max_distance_cm;
    }
    int             distance_complementary_cm() const
    {   return max( 0, distance_max_cm() - distance_cm());
    }
    virtual ~D3SensorUss()
    {}

private:
    int     _pid_trigger;
    int     _pid_echo;
    D3Clock _clock;
    int     _max_distance_cm;
};


/*****************************************************************************
 */
class D3SensorWaterlevel : public D3Sensor
/*
 *****************************************************************************/
{
public:
    D3SensorWaterlevel( const char *p_id)
    : D3Sensor( p_id)
                                    {}
    virtual ~D3SensorWaterlevel()
                                    {}
    virtual bool    is_waterlevel_low() = 0;
    virtual bool    is_waterlevel_ok() = 0;

    virtual int     waterlevel_100() = 0;
};


/*****************************************************************************
 */
class D3SensorWaterlevelAnalog : public D3SensorWaterlevel
/*
 * Note:
 *      Der Sensor realisiert eine Hysterese von 10 % Feuchte.
 *
 *****************************************************************************/
{
public:
    /*************************************************************************
     */
    D3SensorWaterlevelAnalog( const char *p_id, int pid)
    : D3SensorWaterlevel( p_id)
    , _moisturesensor( p_id, pid)
```

```
    {}

    /*************************************************************************
     */
    virtual ~D3SensorWaterlevelAnalog()
    {}

    /*************************************************************************
     */
    bool    is_analog() const override
    {   return true;
    }

    /*************************************************************************
     */
    int     waterlevel_100() override;
    /*
     * Wir retournieren einfach den Feuchtewert als Wasserstand. Der Übergang von
     * 100 % auf 0 % und umgekehrt erstreckt sich über die Länge des Sensors.
     *
     *************************************************************************/

    /*************************************************************************
     */
    bool    is_waterlevel_low() override;
    /*
     * Zuwenig Wasser = weniger als 20 % Feuchte?
     *
     * Zusammen mit `is_waterlevel_ok()` realisiert diese Methode eine Hysterese
     * von 10 %.
     *
     *************************************************************************/

    /*************************************************************************
     */
    bool    is_waterlevel_ok() override;
    /*
     * Genug Wasser = mehr als 80 % Feuchte?
     *
     * Zusammen mit `is_waterlevel_low()` realisiert diese Methode eine Hysterese
     * von 10 %.
     *
     *************************************************************************/

    void    write_to_hardware() override;

    void    read_from_hardware();

private:
    D3SensorSoil_Analog \
                    _moisturesensor;
};


/*****************************************************************************
 */
class D3SensorWaterlevelUSS : public D3SensorWaterlevel
/*
 * Note:
 *      Der Sensor realisiert eine Hysterese von 10 cm.
 *
 *****************************************************************************/
{
public:
    /*************************************************************************
     */
    D3SensorWaterlevelUSS( const char *p_id, int pid_trigger, int pid_echo, int height_of_watertank,
unsigned long cycletime_ms, bool dont_set_up=false)
    : D3SensorWaterlevel( p_id)
```

```
    , _height_of_watertank( height_of_watertank)
    , _distancesensor( p_id, pid_trigger, pid_echo, cycletime_ms, dont_set_up)
    {   if (!dont_set_up)
            set_up();
    }

    /*************************************************************************
     */
    virtual ~D3SensorWaterlevelUSS()
    {}

    /*************************************************************************
     */
    void    set_up()
    {   _distancesensor.distance_max_cm( _height_of_watertank);
        is_set_up( true);
    }

    /*************************************************************************
     */
    bool    is_analog() const override
    {   return true;
    }

    /*************************************************************************
     */
    int     waterlevel_100() override
    /*
     * Der Wasserstand in % wird berechnet durch
     * `waterlevel_100 = ((distance_max_cm() - distance_cm())*100)/distance_max_cm();`,
     * wobei die Werte vom enthaltenen USS kommen.
     *
     * Durch diesen USS kann die Rechnung etwas abgekürzt werden:
     * `waterlevel_100 = (_distancesensor.distance_complementary_cm() * 100)/_distancesensor.distan-
ce_max_cm();`
     *
     *************************************************************************/
    {   return (_distancesensor.distance_complementary_cm() * 100)/
_distancesensor.distance_max_cm();
    }

    /*************************************************************************
     */
    bool    is_waterlevel_low() override
    {   return _distancesensor.distance_complementary_cm() < 20;
                                    /* Das ist etwa die Wasserpumpengröße.
                                     */
    }

    /*************************************************************************
     */
    bool    is_waterlevel_ok() override
    {   return _distancesensor.distance_complementary_cm() > 30;
                                    /* Das ist etwa die Wasserpumpengröße.
                                     */
    }

    /*************************************************************************
     */
    void    write_to_hardware() override
    {   _distancesensor.write_to_hardware();
    }

    /*************************************************************************
     */
    void    read_from_hardware()
    {   _distancesensor.read_from_hardware();
    }
```

```
private:
    int             _height_of_watertank;
    D3SensorUss \
                    _distancesensor;
};


#endif
```

8.5 File *d3sensors.cpp*

```
#include <d3logging.h>
#include <d3sensors.h>


/*****************************************************************************
 */
D3SensorUss::D3SensorUss( const char *p_id, int pid_trigger, int pid_echo, unsigned long
cycletime_ms, bool dont_set_up, int max_distance_cm)
: D3Sensor( p_id)
, _pid_trigger( pid_trigger)
, _pid_echo( pid_echo)
, _clock( cycletime_ms)
, _max_distance_cm( max_distance_cm)
{
    if (!dont_set_up)
        set_up();

    value( 0);
}


/*****************************************************************************
 */
void    D3SensorUss::set_up()
/*
 *****************************************************************************/
{
    if (!is_bypassed())
    {
        pinMode( _pid_trigger, OUTPUT);
        pinMode( _pid_echo, INPUT);
    }

    is_inited( true);
}

/*****************************************************************************
 */
void    D3SensorUss::write_to_hardware()
/*
 * Important:
 *      Zwischen `write_to_hardware()` und `read_from_hardware()` darf es
 *      keinerlei Verzögerung geben wie z.B. eine Ausgabe auf die serielle
 *      Schnittstelle!
 *
 *****************************************************************************/
{
    //~ d3debug( "D3SensorUss::write_to_hardware(): E n t e r e d.");
    if (!is_bypassed())
    {
        _clock.update();
        if (_clock.is_run_down_again())
        {
            //~ d3debug( "D3SensorUss::write_to_hardware(): Trigger sensor.");
            digitalWrite( _pid_trigger, LOW);
            delayMicroseconds( 2);
```

```
            digitalWrite( _pid_trigger, HIGH);
            delayMicroseconds( 10);
            digitalWrite( _pid_trigger, LOW);
        }
    }
    else
        //~ d3debug( "D3SensorUss::write_to_hardware(): Sensor is bypassed!");

    //~ d3debug( "D3SensorUss::write_to_hardware(): Exit now.\n");
    return;
}

/*****************************************************************************
 */
void    D3SensorUss::read_from_hardware()
/*
 * Important:
 *      Zwischen `write_to_hardware()` und `read_from_hardware()` darf es
 *      keinerlei Verzögerung geben wie z.B. eine Ausgabe auf die serielle
 *      Schnittstelle!
 *
 *****************************************************************************/
{
    //~ d3debug( "D3SensorUss::read_from_hardware(): E n t e r e d.");
    if (!is_bypassed())
    {
        //_clock.update();
        //////  Darf pro Zyklus nur 1-mal ausgeführt werden und das machen wir bereits in
            //      write_to_hardware().
        if (_clock.is_run_down_again())
        {
            //~ d3debug( "D3SensorUss::read_from_hardware(): Read duration_us.");
            unsigned long  duration_us = pulseIn( _pid_echo, HIGH);
            if (duration_us)
                value( (0.0343 * double( duration_us)) / 2.0);
        }
    }

    //~ d3debug( "D3SensorUss::read_from_hardware(): Exit now.\n");
    return;
}


/*############################################################################
############################################################################
############################################################################*/

D3SensorUssPlain::D3SensorUssPlain( const char *p_id, int pid_trigger, int pid_echo, bool
dont_set_up, int max_distance_cm)
: D3Sensor( p_id)
, _pid_trigger( pid_trigger)
, _pid_echo( pid_echo)
, _max_distance_cm( max_distance_cm)
{
    if (!dont_set_up)
        set_up();

    value( 0);
}


/*****************************************************************************
 */
void    D3SensorUssPlain::set_up()
/*
 *****************************************************************************/
{
    if (!is_bypassed())
    {
```

```
        pinMode( _pid_trigger, OUTPUT);
        pinMode( _pid_echo, INPUT);
    }

    is_inited( true);
}

/*****************************************************************************
 */
void    D3SensorUssPlain::write_to_hardware()
/*
 * Important:
 *      Zwischen `write_to_hardware()` und `read_from_hardware()` darf es
 *      keinerlei Verzögerung geben wie z.B. eine Ausgabe auf die serielle
 *      Schnittstelle!
 *
 *****************************************************************************/
{
    //~ d3debug( "D3SensorUssPlain::write_to_hardware(): E n t e r e d.");
    if (!is_bypassed())
    {
        //~ d3debug( "D3SensorUssPlain::write_to_hardware(): Trigger sensor.");
        digitalWrite( _pid_trigger, LOW);
        delayMicroseconds( 2);
        digitalWrite( _pid_trigger, HIGH);
        delayMicroseconds( 10);
        digitalWrite( _pid_trigger, LOW);
    }
    else
        //~ d3debug( "D3SensorUssPlain::write_to_hardware(): Sensor is bypassed!");

    //~ d3debug( "D3SensorUssPlain::write_to_hardware(): Exit now.\n");
    return;
}

/*****************************************************************************
 */
void    D3SensorUssPlain::read_from_hardware()
/*
 * Important:
 *      Zwischen `write_to_hardware()` und `read_from_hardware()` darf es
 *      keinerlei Verzögerung geben wie z.B. eine Ausgabe auf die serielle
 *      Schnittstelle!
 *
 *****************************************************************************/
{
    //~ d3debug( "D3SensorUssPlain::read_from_hardware(): E n t e r e d.");
    if (!is_bypassed())
    {
        //~ d3debug( "D3SensorUssPlain::read_from_hardware(): Read duration_us.");
        unsigned long  duration_us = pulseIn( _pid_echo, HIGH);
        if (duration_us)
            value( (0.0343 * double( duration_us)) / 2.0);
    }

    //~ d3debug( "D3SensorUssPlain::read_from_hardware(): Exit now.\n");
    return;
}


/*#############################################################################
#
#   D3SensorWaterlevelAnalog
#
#############################################################################*/

/*****************************************************************************
 */
int     D3SensorWaterlevelAnalog::waterlevel_100()
```

```
/*
 ******************************************************************************/
{   return _moisturesensor.humidity_100();
}


/******************************************************************************
 */
bool    D3SensorWaterlevelAnalog::is_waterlevel_low()
{   return _moisturesensor.humidity_100() < 20;
}


/******************************************************************************
 */
bool    D3SensorWaterlevelAnalog::is_waterlevel_ok()
{   return _moisturesensor.humidity_100() > 80;
}


/******************************************************************************
 */
void    D3SensorWaterlevelAnalog::write_to_hardware()
{}


/******************************************************************************
 */
void    D3SensorWaterlevelAnalog::read_from_hardware()
{   _moisturesensor.read_from_hardware();
    value( _moisturesensor.value());
}
```

8.6 File *d3signalling.h*

```
/*****************************************************************************
 *  FILE:
 *      /atlantis/s/swdevmnt/tau/tebe/arduino/sketchbooks/libraries/trident/d3signalling.h
 *
 *  SYNOPSIS:
 *      Funktionalität, die mit Anzeigen wie LEDs und OLEDs zu tun hat.
 *
 *  REVISION HISTORY:
 *      2024-02-08/FGE:
 *          Erstversion.
 *
 *****************************************************************************/

#ifndef _D3SIGNALLING_H
#define _D3SIGNALLING_H

#include <Arduino_LED_Matrix.h>

/*****************************************************************************
 */
class D3LEDStripe
{
public:
    D3LEDStripe( unsigned char num_leds)
    : _num_leds( num_leds)
    {}

    virtual ~D3LEDStripe()
    {}

private:
    unsigned char   _num_leds;
};


/*****************************************************************************
 */
class ArduinoUnoR4Matrix
{
public:
    ArduinoUnoR4Matrix( bool init_this_instance_manually_in_the_setup_function)
    : _leds {
        { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
        { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
        { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
        { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
        { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
        { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
        { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
        { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }
    }
    ,
_init_this_instance_manually_in_the_setup_function( init_this_instance_manually_in_the_setup_functi-
on)
    {
        if (!_init_this_instance_manually_in_the_setup_function)
            _matrix.begin();
    }

    void set_up()
    {
        if (_init_this_instance_manually_in_the_setup_function)
            _matrix.begin();
    }

    /*************************************************************************
```

```
     *
     * LED ein- oder ausschalten.
     *
     * Aber Achtung: Die Anzeige erfolgt erst bei Ausführung von write_to_hardware()!
     *
     * Parameters:
     *      i (unsigned char):
     *          Zeile der LED-Matrix.
     *
     *      j (unsigned int):
     *          Spalte der LED-Matrix.
     */
    void led( unsigned char i, unsigned char j, unsigned char value)
    {
        _leds[ min(  7, i)][ min( 11, j)] = value ? 1 : 0;
    }
    /*
     ************************************************************************/

    void percents_to_leds_in_a_row( unsigned char i, int value_100)
    {
        value_100 = max( 0, min( 100, value_100));
        char J = (value_100 * 12) / 100 - 1;
        for (int j = 0; j < 12; j++)
            led( i, j, j >= 0 && j <= J);
    }

    void clear()
    {
        for (int i = 0; i < 8; i++)
            for( int j = 0; j < 12; j++)
                led( i, j, 0);
    }

    void    write_to_hardware()
    {
        // for (int i = 0; i < 8; i++)
        // {
        //     for (int j = 0; j < 12; j++)
        //         Serial.print( _leds[ i][ j]);
        //
        //     Serial.println();
        // }
        _matrix.renderBitmap( _leds, 8, 12);
    }

    virtual ~ArduinoUnoR4Matrix()
    {}

private:
    unsigned char       _leds[ 8][ 12];
    bool                _init_this_instance_manually_in_the_setup_function;
    ArduinoLEDMatrix    _matrix;
};


#endif
```

8.7 File *d3statedefs.h*

```
#ifndef _STATE_H
#define _STATE_H

/*****************************************************************************
 *
 * FILE:
 *      stateid.h
 *
 * SYNOPSIS:
 *      Zustandswert für FSMs als Value und als String, letzteres für DIagnose-/Anzeigezwecke.
 *
 * NOTE:
 *      File und Klasse können nicht `state` bzw `State` heißen, weil das die
 *      einzelnen Zustände sein müssten. States als Klassen zu realisieren, ist
 *      aufwendiger als es sein muss für die App SCHLAUBEET/sp.
 *
 * REVISION HISTORY:
 *      2024-04-30/FGE:
 *          Erstversion
 *
 *****************************************************************************/

#include <Arduino.h>


enum D3StateValues
{
    SV__IDLE,
    SV__INITING,
    SV__MONITORING,
    SV__WATERING,
    SV__MOISTURISING,

    SV__ERROR_LOW_WATER = 99
};

/*****************************************************************************
 */
class D3StateId
/*
 * Usage:
 *      switch( p_gbed->stateid().value())
 *      {
 *          :
 *          :
 *          :
 *      case S_STATE_1:
 *          :
 *          :
 *          :
 *          p_pved->stateid().value( S_STATE_2);
 *      break;
 *          :
 *          :
 *          :
 *      }
 *
 *****************************************************************************/
{
public:
    D3StateId( int statevalue = D3StateValues::SV__INITING)
    : _statevalue( statevalue)
    {
        value2name();
    }
```

```
    /*************************************************************************
     */
    const String&   name() const
    {
        return _statename;
    }

    /*************************************************************************
     */
    const char      *p_name() const
    {
        return _statename.c_str();
    }

    /*************************************************************************
     */
    int             value() const
    {
        return _statevalue;
    }

    /*************************************************************************
     */
    void            value( int arg)
    {
        _statevalue = arg;
        value2name();
        return;
    }

    /*************************************************************************
     */
    virtual ~D3StateId()
    {}

private:
    void    value2name()
    {
        switch( _statevalue)
        {
        case D3StateValues::SV__IDLE:               _statename = "IDLE";
            break;

        case D3StateValues::SV__INITING:            _statename = "INITING";
            break;

        case D3StateValues::SV__MONITORING:         _statename = "MONITORING";
            break;

        case D3StateValues::SV__WATERING:           _statename = "WATERING";
            break;

        case D3StateValues::SV__MOISTURISING:       _statename = "MOISTURISING";
            break;

        case D3StateValues::SV__ERROR_LOW_WATER:    _statename = "LOW WATER";
            break;

        default:                                    _statename = "???";
            break;
        }
    }

private:
    int     _statevalue;
    String  _statename;
};

#endif // _STATE_H
```

8.8 File *d3time.h*

```
#ifndef _D3TIME_H
#define _D3TIME_H

/*****************************************************************************
 *  FILE:
 *      /atlantis/s/swdevmnt/tau/tebe/arduino/sketchbooks/libraries/trident/d3time.h
 *
 *  SYNOPSIS:
 *      Funktionalität, die mit Zeit zu tun hat.
 *
 *  REVISION HISTORY:
 *      2024-02-05/FGE:
 *          D3Clock realisiert.
 *
 *      2024-02-01/FGE:
 *          Erstversion.
 *
 *****************************************************************************/

#include <Arduino.h>


/*****************************************************************************
 */
class D3Clock
/*
 * Usage:
 *      D3Clock clock( 1000);
 *      clock.update();
 *      if (clock.is_run_down_again())
 *          YOUR ACTION GOES HERE
 *
 * 2DOs:
 *      -   2024-05-01/FGE:
 *          Design-Fehler ausbügeln: Es muss eine Methode `reset`
 *          geben. Jetzt übernimmt diese  Aufgabe `update` und das ist die
 *          zweitbeste Idee.
 *
 *****************************************************************************/
{
public:
    D3Clock( unsigned long interval_ms)
    : _interval_ms( interval_ms)
    , _is_run_down_again( false)
    {
        _time_of_running_down_last_ms = millis();
    }

    void    update()
    {
        if (millis() - _time_of_running_down_last_ms > _interval_ms)
        {
            _is_run_down_again = true;
            _time_of_running_down_last_ms += _interval_ms;
        }
        else
            _is_run_down_again = false;

    }

    bool    is_run_down_again() const
    {
        return _is_run_down_again;
```

```
    }

private:
    unsigned long   _interval_ms;
    unsigned long   _time_of_running_down_last_ms;
    bool            _is_run_down_again;
};

/*************************************************************
 */
class D3Looprateguard
{
public:
    D3Looprateguard( int cycletime_to_ensure_ms)
    : _cycletime_to_ensure_ms( cycletime_to_ensure_ms)
    , _time_of_last_sleepend_ms( 0)
    , _sleeptime_ms( 10)
    {}

    void sleep()
    {
        long    now_ms = millis();
        long    looptime_ms = now_ms - _time_of_last_sleepend_ms;
        _sleeptime_ms = _cycletime_to_ensure_ms - looptime_ms;
        //Serial.print( (int)_cycletime_to_ensure_ms); Serial.print( "; "); Serial.println( loopti-
me_ms);
        if (_sleeptime_ms < 0)
        {
            _sleeptime_ms = 0;
            Serial.print( "WARNING: Sleep-time underflow by ");
            Serial.print( abs( _cycletime_to_ensure_ms - looptime_ms));
            Serial.println( " ms!");
        }
        delay( _sleeptime_ms);
        _time_of_last_sleepend_ms = millis();
    }

    unsigned long   sleeptime_ms() const { return _sleeptime_ms; }

private:
    int             _cycletime_to_ensure_ms;
    unsigned long   _time_of_last_sleepend_ms;
    long            _sleeptime_ms;
};


/*************************************************************
 */
class D3Stopwatch
{
public:
    D3Stopwatch( const char *p_id=NULL)
    : _p_id( p_id)
    , _is_running( false)
    , _time_ms( 0.0)
    {}

    String          dumps() const
    {
        String s( "D3Stopwatch(");
        if (_p_id)
            s += _p_id;

        s += "):\n\tRunning: ";
        s += _is_running;
        s += "\n\tTime: ";
        s += _time_ms;
        s += " ms";
        return s;
```

```
    }

    bool            is_running() const
                                    {   return _is_running;
                                    }
    void            reset()
                                    {   _time_ms = _time_started_ms = 0;
                                    }
    void            start()
                                    {   _time_started_ms = millis();
                                        _is_running = true;
                                    }
    /**************************************************************************
     */
    void            stop()
    /*
     * Nach einem Stop kann die Zeit gelesen werden u/o die Stopuhr per `start()`
     * wieder gestartet werden. Jede Start-Stop-Sequenz addiert die Zeitspanne
     * zwischen `start()` und `stop()` zur Zeit, die dann gelesen werden kann.
     *
     *************************************************************************/
    {   _time_ms += millis() - _time_started_ms;
        _is_running = false;
    }

    double          time()
                                    {   return (double)time_ms() / 1000.0;
                                    }
    unsigned long   time_ms()
                                    {
                                        //~ if (_is_running)
                                            //~ _time_ms += millis() - _time_started_ms;

                                        //~ return _time_ms;
                                        if (_is_running)
                                            return _time_ms + millis() - _time_started_ms;

                                        return _time_ms;
                                    }

    virtual ~D3Stopwatch()
                                    {}

private:
    const char      *_p_id;
    bool            _is_running;
    unsigned long   _time_ms;
    unsigned long   _time_started_ms;
};
#endif
```

9 Anhang: Einbindung des Workshops *Schlaubeet* in den Unterricht

9.1 Lernziele

Wozu Lernziele? Nun, man kann diesen Workshop auch so durchführen, einfach als kleines Automatisierungsprojekt. Auch so gibt es in diesem Workshop vieles zu lernen.

Allerdings ist die Wirkung eine viel größere wenn man auch den Gegenstand der Automatisierung mit einbindet – den Garten. Wie das geschehen könnte, ist in den folgenden Kapiteln skizziert.

9.1.1 Biologie

Die Schüler

- wissen, wie Pflanzen funktionieren (Nährstofftransport, Photosynthese usw.)
- wissen, was Pflanzen brauchen, um zu gedeihen
- wissen Bescheid über günstige Bodenbeschaffenheit (pH-Wert?)
- wissen, wie gesund (Wasser?) eine Pflanze sein muss, um gegen Schädlinge resistent zu sein
- wissen, welche biologische Mögl. es zur Schädlingsbekämpfung gibt insbes. bez. Schlaubeet

9.1.2 Physik

Die Schüler

- kennen den Unterschied zw. Batt. u. Akku und wissen, was er bedeutet
- wissen wie eine PV-Zelle funktioniert, wie sie verschaltet werden und wie man das Optimum aus einem Solar-Panel herausholt
- wissen, was Tempr bedeutet, die versch. Systeme und wie man sie misst und wissen in diesem Zshang um die Anomalie des Wassers
- kennen den Zshang zw. Tempr und Feuchtigkeit (Taupunkt usw.)
- können mit Begriffen aus der E-Technik umgehen (kWh vs. kW).
- wissen, was Klima ist, was Wetter ist und wie letzteres entsteht (H-Druck, T-Druck, Winde)
- wissen, wie Töne entstehen, wie Tonhöhen verändert werden können (Thema Schinwingungen)

9.1.3 Elektronik im Zuge des Fachs Digitale Grundbildung (DGB) / Physik

Die Schüler

- wissen,
  - welche Arten von Sensoren es gibt und welche für das Projekt gut geeignet sind
  - wie Elektromotoren funktionieren und wie sie ausgewählt werden
  - wie ein Mikrocontroller aufgebaut ist
- kennen das Ohmsche Gesetz und können damit einfach Berechnungen durchführen (Vorwiderstand für LED)
- können
  - einfache C-Programme für die Arduino-Plattform (Arduino Uno, Nano, …) schreiben
  - auch komplexere Aufgaben unter Zuhilfenahme der Library `schlaubeet.ino` lösen
  - einfache Messungen mit einem Multimeter machen (Batteriespannung, Fehlersuche)

9.2 Lehrinhalte

9.2.1 Bioaspekte

- Bedeutung Wasser für die Pflanzen (Stw. *Wasserhaushalt*)
- Bedeutung künstl. Bewässerung
- Folgen künstl. Bew. im Großen
- Starkregen vs. Dürre
- Bodenversiegelung - was bedeutet das und kann man da was rechnen?

9.2.2 Energieversorgung

9.2.2.1 Batterien

- Typen von Batterien
- Benötigte Energiemengen und Batteriegröße

9.2.2.2 Alternative Energiequellen

Sonne, Wind, Wasser

9.2.2.3 Photovoltaik

- Solar-Paneele, wie sie funktionieren und wie sie optimal genützt werden
- Notwendige Energiespeicherung In Batterien

- Solrladeregler – was leistet diese Elektronik?

9.2.3 Sensorik

Sensorik wird z.V. stehen, soll nicht gebaut werden.

9.2.3.1 Temperatur

- Boden
- Luft

9.2.3.2 Bodenfeuchte

Auswahl eines Feuchte-Sensors nach Vor- und Nachteilen.

- Resistiv (vergiftet den Boden, wenn Kupferelektroden)
- Kapazitiv
- Densitometrisch[3]

9.2.3.3 Bodenqualität

- pH-Wert
- Was noch?

9.2.3.4 Wasserverbrauch

- Durchflusssensor

9.2.4 Aktuatorik

- Ventile (Spulen)
- Pumpen (Elektromotoren)

9.2.5 Mikrocontroller

- Programmierung grafisch
- Programmierung in *C*

9.2.6 Erfassung und Interpretation von Betriebsdaten

- Wassermenge
- Außentempr.
- Bodentempr.

3 Farbdichtemessung – s. z.B. https://de.wikipedia.org/wiki/Densitometrie_(Farbdichtemessung).

- Zshang. Wasserverbrauch (Durchflusssensor!) und Wetter u/o Bodenqualität
- Datenvisualisierung von Betriebsdaten
  - Thingspeak
  - Thingsboard
- Virtuelles Wasser: Gegenüberstellung Ertrag zu Wassereinsatz für versch. Gemüse

9.2.7 Ergänzende Bemerkungen

Wir würden mit diesen Lehrzielen (oder einer zu bestimmenden Auswahl) die Themen

- Klima
- Precision Farming
-

abdecken.

SCHLAUBEET wäre mit diesen Lehrinhalten (oder einer Auswahl davon) präsent in einer Reihe von Fächern:

- Physik mit Sensorik, Aktuatorik, Energieversorgung
- DGB mit Programmieren von Mikrocontrollern
- Naturwissenschaft mit Biologie, Wasser, Bodenqualität[4]
- Musik für akustische Darstellung gewisser Betriebsdaten- s. z.B. http://radio.thingslogic.com/
- Wir würden mit diesen Lehrzielen (oder einer zu bestimmenden Auswahl) die Themen

4 Was vergiftet die Böden? Welche Sensorik kommt nicht in Frage aus genau diesem Grund?

Index