Computergestützte Experimente im Physikunterricht unter Verwendung der Arduino Entwicklungsumgebung

Diplomarbeit

zur Erlangung des akademischen Grades
eines Magisters der Naturwissenschaften

an der Karl-Franzens-Universität Graz

vorgelegt von

Florian PFANNHOFER

Begutachter:

Ao. Univ.-Prof. Dipl.-Ing. Dr.techn. Gernot Pottlacher

Technische Universität Graz

Institut für Experimentalphysik

Graz, September 2016

Abstract

Computer aided experiments in physics lessons, applying the Arduino development environment

"When will I need this?", is a common question and sometimes feared by teachers. The connection between educational content and daily life is a justified request, but too often this connection is not made. In this thesis we focus on physical concepts in the context of modern electronic devices and examine deeper the principles of sensors, actuators, displays and other units, which we encounter in everyday life. Furthermore we will demonstrate how to use them practically - what we strongly suggest in addition to reading. For this reason we work with the microcontroller Arduino UNO, which is programmable and takes over the communication with the devices. In several projects the usage of the different units to measure physical quantities, such as voltage, capacities, the speed of sound and molar mass of air will be shown. The building of weather stations which warn of coming storms and verifying whether a car is following the speed limit will also be considered. All these projects shall motivate students to realize their own ideas, using physical principles successfully. As a consequence, physics becomes connected to the reality of life of students, and at the same time its meaning for our modern society is made accessible.

Kurzfassung

„Wofür brauche ich das?“, ist eine wohl bekannte, manchmal vielleicht gefürchtete Frage in der Schule. Die Verbindung von Unterrichtsinhalten zum Alltag wird zurecht gefordert, jedoch leider oft nicht umgesetzt. Noch wünschenswerter wäre die konkrete Anwendung im persönlichen Leben. In dieser Arbeit werden physikalische Effekte im Kontext der modernen elektronischen Geräte betrachtet. Es werden die physikalischen Prinzipien von Sensoren, Aktoren, Displays und weiteren Elementen, die wir tagtäglich verwenden, erörtert und gezeigt, wie man sie ansteuern und in Betrieb nehmen kann - was parallel zur Lektüre auch gemacht werden sollte. Dies geschieht auf Basis des Mikrocontrollers Arduino UNO, den man verhältnismäßig einfach programmieren kann und der die Kommunikation mit den elektronischen Bauteilen übernimmt. In weiterführenden Projekten wird gezeigt, wie man die einzelnen Komponenten einsetzt, um Spannungen oder Kapazitäten zu messen, die Schallgeschwindigkeit und molare Masse der Luft zu bestimmen, Wetterstationen, die vor Unwetter warnen zu bauen oder zu überprüfen, ob sich ein Auto an die vorgeschriebene Höchstgeschwindigkeit hält. Dies soll dazu anregen, eigene Ideen zu verwirklichen, bei denen man ganz nebenbei physikalische Grundlagen anwenden muss, um zu seinem Ziel zu gelangen. Auf diese Weise wird die Physik mit der Lebensrealität der Schüler verknüpft und zugleich erfahrbar, welche Bedeutung sie für unsere moderne Gesellschaft hat.

Inhalt

Eidesstattliche Erklärung

Ich erkläre ehrenwörtlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benutzt und die den Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen inländischen oder ausländischen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht. Die vorliegende Fassung entspricht der eingereichten elektronischen Version.

Graz, am 15. September 2016

(Florian Pfannhofer)

1 Einleitung

In der Physik beschäftigen wir uns damit, wie die Welt funktioniert - nach welchen Regeln Vorgänge der Natur ablaufen. Wir versuchen, das, was um uns ist, zu begreifen, immer mehr zu verstehen und letztlich nach unseren Vorstellungen und für unsere Zwecke zu verwenden. Als Werkzeug hat sich dabei - neben der anschaulichen Argumentation - besonders die Mathematik bewährt. Blickt man in die Physikklassen der Schulen, so erweckt Einiges den Eindruck, dass von dem Urwesen der Physik relativ wenig bei den Schülern ankommt. Ganz im Gegenteil: Oft scheint es für sie um abstrakte, weltfremde Dinge zu gehen, die vielleicht eine Anwendung in dunklen Laboratorien finden, sich jedoch vor dem Alltag möglichst fernhalten.

Um diesem Widerspruch - zumindest teilweise - entgegenzutreten, soll in der folgenden Arbeit den Schülern der Zugang zur elektronischen Welt geöffnet werden, die ein fester Bestandteil unseres Alltags ist. Wir werden dazu Sensoren und Aktoren betrachten, die in jedem Haushalt zu finden sind und die physikalischen Prinzipien untersuchen, welche sie zu ihrer jeweiligen Aufgabe befähigt. Um den Einstieg möglichst einfach zu gestalten, verwenden wir eine kleine Platine, die den Namen Arduino UNO trägt. Diese können wir mit Hilfe eines Computers nach unseren Vorstellungen programmieren und zur Kommunikation mit den Sensoren und Aktoren verwenden.

Nach einem kurzen allgemeinen Teil über die Eigenschaften des Arduinos lernen wir viele Sensoren kennen, besprechen ihre physikalischen Hintergründe und wie wir sie mit dem Arduino ansteuern können (Kapitel „Grundlegende Experimente und Funktionsweisen von Sensoren“). Dabei werden uns schon erste Messungen und einfache Experimente begegnen. Im darauf folgenden Kapitel „Erweiterte Versuche und Projekte“ setzen wir unsere gewonnen Erkenntnisse ein, um komplexere Projekte zu erarbeiten, wie z.B. eine Wetterstation, die uns vor möglichen Unwettern warnen kann.

Das Ziel dieser Arbeit ist es, eine Möglichkeit zu bieten, den Schülern wieder das Wesen der Physik zugänglich zu machen; über Sensoren und Arduino zu lernen und für eigene - auch verspielte - Projekte zu verwenden und dabei ganz natürlich und

ungezwungen auf die physikalischen Grundlagen und Prinzipien zurückzugreifen. Dadurch bekommt die Physik zwar kein neues Gesicht, aber das alte und eigentliche wird wieder ein Stück weit sichtbar.

Die vorgestellten Versuche und Projekte wurden mit den in Tab. 1.1 aufgelisteten Komponenten durchgeführt, wobei versucht wurde die Kosten möglichst gering zu halten, um den Schuleinsatz zu erleichtern.

Tab. 1.1: Benötigtes Material

Bezeichnung	Anzahl	Händler	Preis pro Einheit
Simpleduino UNO (Arduinonachbau)	1	Amazon	13,90 €
Gyro- und Beschleunigssensor MPU-6050	1	Amazon	7,69 €
3 Achsen Magnetometer HMC5883L	1	Amazon	8,99 €
4 Kanal Relais-Modul	1	Amazon	3,34 €
Ultraschall Messmodul HC-SR04	2	Neuhold	3,95 €
Feuchtigkeitssensor AM2301	1	Amazon	4,77 €
Luftdrucksensor BMP180	1	Amazon	7,59 €
LCD Display 16x2 mit I²C Modul	1	Amazon	4,59 €
Fotowiderstand 500 kΩ	2	Neuhold	0,20 €
Schrittmotor DAYPOWER S-SPSM-5V	2	Neuhold	4,95 €
Potentiometer 10 kΩ	2	Neuhold	1,84 €
Taster	4	Neuhold	0,15 €
Temperatursensor KTY 81/222	1	Conrad	1,29 €
Drahtbrückenset mit Box	1	Neuhold	5,95 €
Widerstands Sortiment	1	Conrad	2,49 €
Elektrolytkondensatoren Sortiment	1	Conrad	2,79 €
Breadboard	1	Amazon	2,64 €
Jumper Wires male	1	Amazon	2,65 €
Jumper Wires female	1	Amazon	2,68 €
Sortimentkasten	1	Neuhold	4,40 €
		Gesamtsumme	98,24 €

2 Über Arduino

2.0.1 Allgemeines

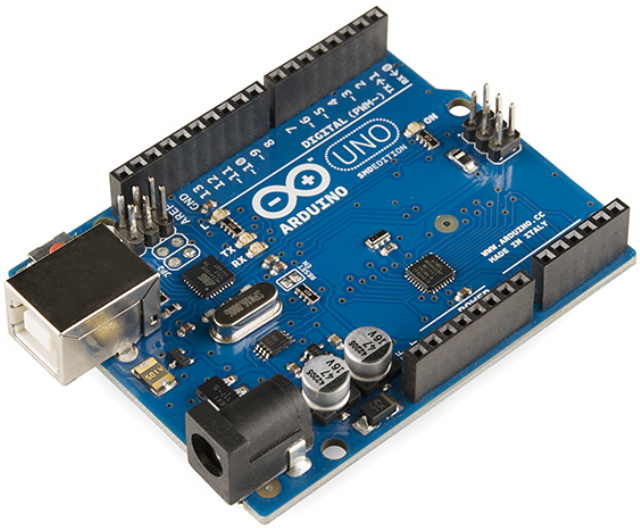

Abb. 2.1: Arduino UNO [1]

Arduino, das ist nicht nur die etwa 5 cm breite und 7 cm lange und einen Mikrocontroller beherbergende Platine aus Abb. 2.1, sondern auch eine IDE (**I**ntegrated **D**evelopment **E**nviroment) Entwicklungsumgebung zum Erstellen von ausführbaren Programmen für den Mikrocontroller. Es ist sozusagen ein Gesamtpaket mit dem man sowohl einfache, als auch komplexe Elektronikaufgaben verwirklichen kann, mit dem Vorteil, viele Aufgaben die eigentlich hardwarebasiert sind, auf der Softwareebene zu lösen. Alle Pläne sind offen gelegt und können nach eigenen Wünschen modifiziert, oder einfach nachgebaut werden. Es gibt mittlerweile eine anschauliche Flotte an verschiedenen Arduino Boards, wobei das oben abgebildete Arduino-UNO Board das weitverbreitetste ist. Seine Eigenschaften sind in Tab. 2.1 angeführt.

Tab. 2.1: Technische Spezifikationen des Arduino UNO [1]

Mikrocontroller	ATmega328P
Betriebsspannung	5 V
empfohlene Versorgungsspannung	7-12 V Gleichspannung
Digitale Ein- Ausgänge	14 (davon unterstützen 6 PWM)
Analoge Eingänge	6
Strom pro digitalem Eingang	20 mA
Flash Memory	32 kB (0,5 kB für den Bootloader)
SRAM	2 kB (ATmega328P)
EEPROM	1 kB (ATmega328P)
Taktfrequenz	16 MHz
Länge	68,6 mm
Breite	53,4 mm
Gewicht	25 g

2.0.2 Aufbau eines Arduino Programms

Ein Programm für den Arduino - auch Sketch genannt - besitzt eine fixe Struktur bestehend aus drei Blöcken (siehe Abb. 2.2). Im ersten Block weist man Variablen einen Datentyp zu, die dann im gesamten restlichen Programm verfügbar sind. Hier werden auch eventuell benötigte externe Bibliotheken mittels `#include` eingebunden. In Block Zwei, der `setup`-Funktion, definiert man grundsätzliche Einstellungen, wie etwa welche Anschlüsse des Arduino als Eingang, bzw. Ausgang verwendet werden sollen. Sie wird einmal ausgeführt und im restlichen Programm nicht mehr durchlaufen. Die `loop`-Funktion bildet den dritten Block der den eigentlichen Code enthält. Solange der Arduino mit Spannung versorgt wird, wird die `loop`-Funktion in einer Endlosschleife wiederholt.[1]

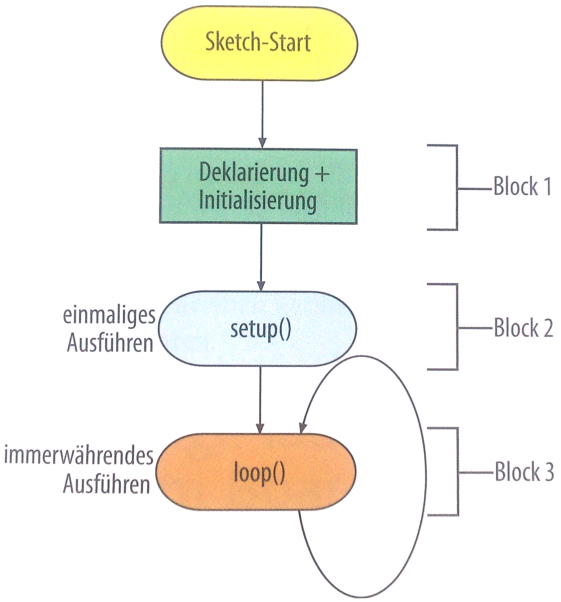

Abb. 2.2: Sketch-Struktur [2]

1) Vgl. [2]

3 Grundlegende Experimente und Funktionsweisen von Sensoren

3.1 Fotowiderstand

Benötigte Materialien

- Arduino UNO
- Breadboard
- Jumper Wires
- Fotowiderstand
- 12,4 kΩ Widerstand

3.1.1 Phyikalisches Prinzip

Fotowiderstände (**L**ight **D**ependent **R**estistor, **LDR**) sind Bauteile, deren Widerstandswert sich mit äußerem Lichteinfall ändert. Sie bestehen aus einer fotoleitenden Halbleiterschicht die auf einem Keramikträger aufgebracht ist und sich zwischen zwei Elektroden befindet. Fallen Photonen auf die Schicht, so können Elektronen aus ihrer Bindung gelöst werden. Dadurch entstehen Ladungsträgerpaare - bestehend aus Elektron und zurückbleibendem Loch - die kurzzeitig zum Stromtransport zur Verfügung stehen und anschließend wieder rekombinieren. Insgesamt verringert sich durch diesen Prozess der elektrische Widerstand bei Beleuchtung mit Licht. Als Halbleiter werden meist Cadmiumsulfid (CdS) oder Cadmiumselenid (CdSe) verwendet, da sich diese besonders für den Einsatz im sichtbaren Spektralbereich eignen, wie in Abb. 3.1 sichtbar ist.[1].

1) Vgl. [3], [4] und [5]

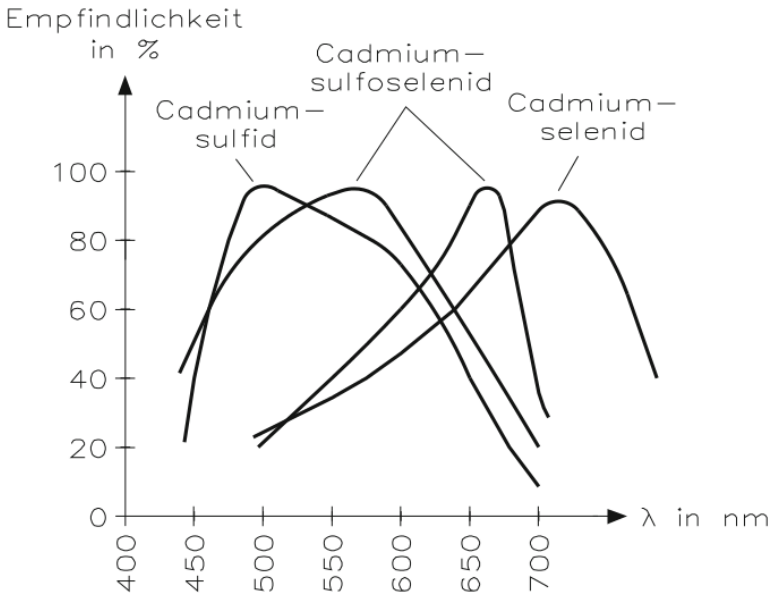

Abb. 3.1: Spektrale Empfindlichkeit von Fotowiderständen aus unterschiedlichen Halbleitern [3]

3.1.2 Aufbau

Die einfachste Art um mit dem Arduino den Widerstand des Fotowiderstandes zu messen, ist mittels Spannungsteiler. Um gute Ergebnisse zu erhalten misst man zunächst mit einem Multimeter den Widerstandswert des LDR bei Dunkelheit, als auch bei starkem Lichteinfall und wählt einen zusätzlichen Widerstand, der in der Mitte der beiden Werte liegt. Dieser wird in Serie mit dem LDR geschlossen und das Potential zwischen den beiden Bauteilen auf den analogen Eingang A0 gelegt (siehe Abb. 3.2). Die hier abgegriffene Spannung entspricht dem Spannungsabfall am LDR. Da über A0 nur ein vernachlässigbarer Strom abfließt, werden beide Widerstände von dem selben Strom I durchflossen. Weiters entspricht die Summe beider Spannungsabfälle der 5 V Arbeitsspannung des Arduinos. Über eine kurze Rechnung kann man aus

$$I = \frac{U_R}{R} = \frac{U_{LDR}}{R_{LDR}} \qquad \text{und} \qquad U_{LDR} + U_R = 5\,\text{V}$$

auf den Widerstandswert des LDR schließen:

$$R_{LDR} = R \cdot \frac{U_{LDR}}{U - U_{LDR}}. \tag{3.1}$$

R	Hilfswiderstand
R_{LDR}	Widerstandswert des LDR
I	Strom der durch die beiden Widerstände fließt
U_R	Spannungsabfall am Hifswiderstand
U_{LDR}	Spannungsabfall am LDR

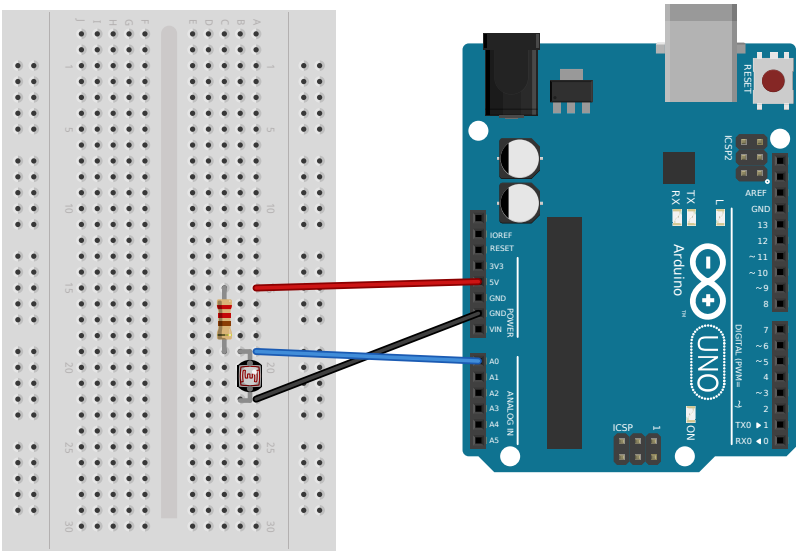

Abb. 3.2: Schaltplan zur Ausmessung eines Fotowiderstandes

Mit der eben beschriebenen Methode, lässt sich jeder passive Widerstands-Sensor auslesen.

3.1.3 Programmcode

```
int X_LDR;              // Definiere X_LDR als ganze Zahl
float U_LDR, R_LDR;     // Definiere U_LDR und R_LDR als float
const int R=12400;      // Wert des verwendeten Hilfswiderstandes
void setup() {
  // Starte serielle Verbindung mit 9600 baud (=Symbole pro Sekunde)
  Serial.begin(9600);
}

void loop() {
  //Schreibe den an Pin0 anliegenden Wert in X_LDR
  X_LDR = analogRead(A0);
  //Berechne die Spannung U_LDR
  U_LDR = (float(X_LDR) * 5/1023);
  //Berechne daraus den Wiederstandswert R_LDR
  R_LDR = float(R) * U_LDR / (5 - U_LDR);
  //Sende den Wert von R_LDR an den seriellen Monitor
```

```
17   Serial.print(R_LDR);
18   Serial.println(" Ohm");
19   delay(500);      //Warte 500 ms
20 }
```

Code 3.1: Auslesen eines Fotowiderstands

Das Arduino Board besitzt einen 10 bit Analog-Digital Wandler. Der kontinuierliche Wertebereich von 0 V bis 5 V des analogen Eingangs, kann daher in $2^{10} = 1024$ Teile diskretisiert werden, wobei 5 V dem Wert 1023 entspricht.
Der diskrete Wert von A0 wird in der als ganzen Zahl definierten Variable X_LDR gespeichert. Um Konflikte mit verschiedenen Datentypen zu vermeiden, wird er in Zeile 13 mit dem Befehl `float(`X_LDR`)` in eine Fließkommazahl umgewandelt. In der selben Zeile wird noch der Spannungswert in Volt berechnet. Alternativ könnte man die Umrechnung auch mit der Funktion `map` durchführen, die wir später noch verwenden werden. Schlussendlich wird der mit Gleichung (3.1) berechnete Widerstandswert R_{LDR} an die serielle Schnittstelle geschickt und kann im seriellen Monitor der IDE-Programmierumgebung abgelesen werden.

3.2 Halbleiter Widerstandsthermometer

Benötigte Materialien

- Arduino UNO
- Breadboard
- Jumper Wires
- Silizium Widerstandsthermometer KTY81-22
- 4,65 kΩ Widerstand

Die Temperatur ist die mit Abstand am häufigsten gemessene Größe bei Anwendungen. Sie ist deshalb so wichtig, weil die meisten Stoffeigenschaften von ihr abhängen. Längenausdehnung, elektrischer Widerstand, Brechzahl und Druck von Gasen, chemische Reaktionsgeschwindigkeit und noch vieles mehr - alles hängt von der Temperatur ab, und all diese Effekte können auch genützt werden, um auf die Temperatur zu schließen. In Abb. 3.3 und Abb. 3.4 sind die wichtigsten Messverfahren angeführt, wobei noch die Fülle an Thermoelementen zu erwähnen ist, die den gesamten dargestellten Temperaturbereich abdeckt.[2]

2) Vgl. [6]

Wir werden uns im Folgenden auf die Beschreibung von Halbleiter Widerstandsthermometer beschränken. Für weitere Ausführungen sei auf [6] verwiesen.

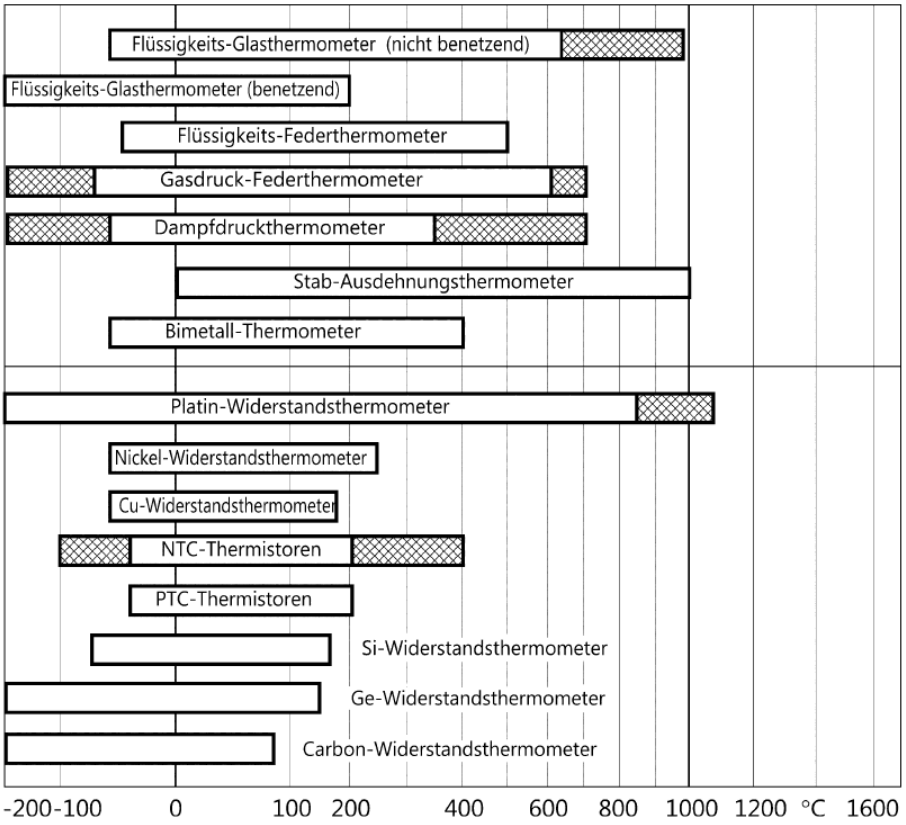

Abb. 3.3: Technische Temperaturmessverfahren und ihre Messbereiche 1 [6]

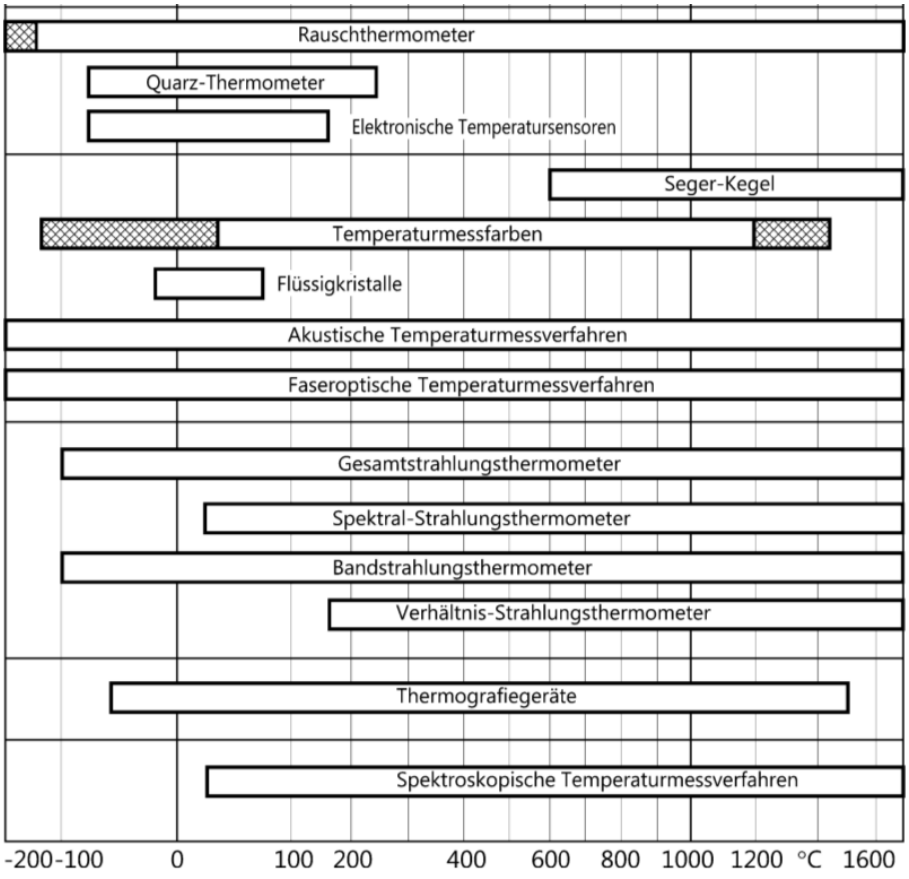

Abb. 3.4: Technische Temperaturmessverfahren und ihre Messbereiche 2 [6]

3.2.1 Physikalisches Prinzip

Um die elektrische Leitfähigkeit einzelner Stoffe nachvollziehen zu können, ist es hilfreich das Energiebändermodell zu verwenden. Dabei betrachtet man alle möglichen Energiezustände die von Elektronen eines Stoffes besetzt werden können in einem Termschema (siehe Abb. 3.5). Für ein einzelnes Atom sind die Energieniveaus eindeutig vergeben und jedes Niveau kann von zwei Elektronen mit entgegengesetzten Spin besetzt werden. Für ein Molekül fächern die Zustände, durch Wechselwirkung der Elektronen mit den positiv geladenen Atomkernen, bereits auf. In einem Kristall, der aus einer Unmenge von Atomen besteht, gibt es so viele Wechselwirkungen zwischen ihnen, dass die einzelnen Energieniveaus derart stark und dicht auffächern, dass man sie zu Bändern zusammenfasst. Dabei können nicht alle Energieniveaus eingenommen werden, wodurch zwischen den Bändern Bereiche unbesetzter Zustände entstehen - die verbotenen Zonen. Das letzte Energieband, das bei einer Temperatur von 0 K noch besetzt ist, bezeichnet man als Valenzband V, das nächsthöhere Band als Leitungsband L. Für die elektrische Leitfähigkeit ist nun die Anzahl der unbesetzten Zustände im Valenzband und dessen Abstand zum Leitungsband ent-

scheidend (siehe Abb. 3.5). Sind im Valenzband alle Zustände besetzt, können sich die Elektronen nicht frei bewegen und tragen daher auch nichts zur Leitfähigkeit bei. Dieser Fall liegt bei Halbleitern vor. Die Besonderheit bei ihnen ist, dass der Abstand zwischen Valenz- und Leitungsband so gering ist, dass moderate thermische Energien reichen, um einzelne Elektronen vom Valenzband in das Leitungsband zu heben, wo sie sich frei bewegen können. Zusätzlich werden dadurch auch Zustände im Valenzband frei. Insgesamt steigt durch diese beiden Effekte die Leitfähigkeit. Da mit zunehmender Temperatur mehr Elektronen vom Valenz- in das Leitungsband gehoben werden können und sich damit der spezifische Widerstand verringert, bezeichnet man Halbleiter auch als Heißleiter oder NTC-Leiter (von **N**egative **T**emperature **C**oefficient).

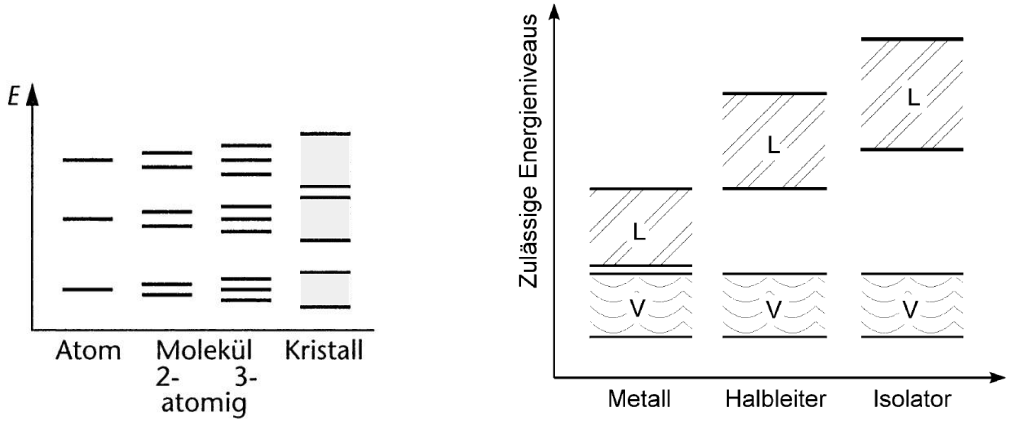

Abb. 3.5: links: Energieniveaus im Festkörper [7] rechts: Energiebändermodell [6]

Durch Dotieren kann die Leitfähigkeit von Halbleitern stark beeinflusst werden. Wird ein vierwertiger Si-Kristall mit dreiwertigen Elementen wie B, Al oder Ga versetzt, so fehlen Elektronen im Gefüge. Diese Fehlstellen bezeichnet man als Defektelektronen oder Löcher, die sich wie positive Ladungsträger verhalten (p-Leitung). Verwendet man stattdessen fünfwertige Elemente wie P, As oder Sb, bleibt ein freies Elektron an den Störstellen über (n-Leitung). In beiden Fällen steigt die Leitfähigkeit an, da dadurch mehr Ladungsträger für den Stromtransport zur Verfügung stehen. Silizium-Temperaturwiderstände - wie etwa der KTY81 - verhalten sich bei steigenden Temperaturen ähnlich wie Metalle, da durch die zunehmende Gitterschwingung die Bewegung der Ladungsträger behindert wird. Ist die Temperatur hoch genug um auch Elektronen aus dem Valenz- ins Leitungsband zu heben, werden sehr viele Ladungsträger für den Stromtransport freigesetzt. Abb. 3.6 zeigt das Zusammenspiel dieser beiden Effekte für verschiedene Dotierungskonzentrationen.[3]

3) Vgl. [6] und [7]

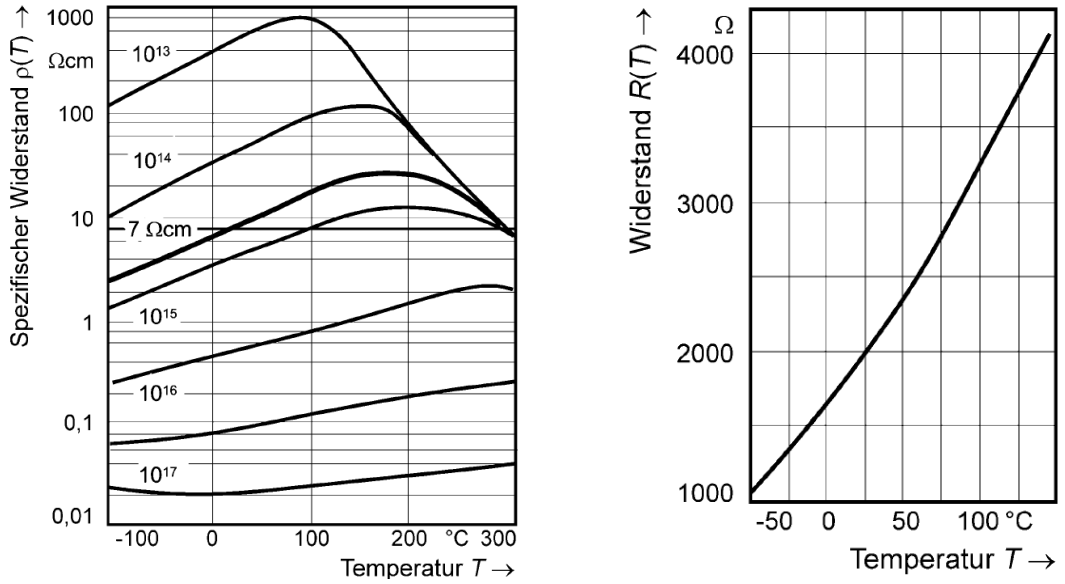

Abb. 3.6: links: Spezifischer Widerstand n-dotierten Siliziums, Parameter: Dotierungskonzentration [6] rechts: Kennlinie eines KTY Sensors [6]

3.2.2 Aufbau

Für unsere Messung bauen wir die Schaltung aus 3.1 auf und ersetzen dabei den Fotowiderstand durch den Silizium Halbleiter-Widerstand KTY81-222. Zu hoher Stromfluss erwärmt den Sensor und verfälscht das Messergebnis - wir wählen daher einen Hilfswiderstand von 4,65 kΩ, der den Strom auf unter 1 mA begrenzt.

3.2.3 Programmcode

Es gibt mehrere Möglichkeiten die Temperatur mit einem Silizium Halbleiterwiderstand zu messen. Über die Linearisierung des Spannungsabfalls eines Spannungsteilers, eine Näherungsformel, oder über Interpolation der gegebenen Werte des Datenblattes. Wir realisieren die dritte Variante und deklarieren zu Beginn des Programms die beiden Arrays R und T, welche die benötigten Werte aus dem Datenblatt enthalten. In der `for`-Schleife vergleichen wir die Werte des Arrays R, mit dem gemessenen Widerstandswert R_T. Der erste Wert aus R, der größer als R_T ist, liefert uns den Index i und wir wissen nun, dass R_T zwischen $R\,[i-1]$ und $R\,[i]$ liegt. Für die lineare Interpolation folgt aus dem Strahlensatz

$$\frac{R\_T - R\,[i-1]}{R\,[i] - R\,[i-1]} = \frac{T\_R - T\,[i-1]}{T\,[i] - T\,[i-1]}$$

die Formel zur Berechnung der Temperatur

$$T\_R = \frac{(R\_T - R[i-1])\,(T[i] - T[i-1])}{R[i] - R[i-1]} + T[i-1]\,. \tag{3.2}$$

```
int X_T;              // Definiere X_T als ganze Zahl
float U_T, R_T, T_R; // Definiere U_T, R_T und T als float
const int R_H=4650;   // Wert des verwendeten Hilfswiderstandes
//Widerstands- und Temperaturwerte aus dem Datenblatt des KTY81-22
int R[]={990, 1040, 1146, 1260, 1381, 1510, 1646, 1790, 1941, 2020,
        2100, 2267, 2441, 2623, 2812, 3009, 3214, 3426, 3643, 3855,
        3955, 4048, 4208, 4323};
int T[]={-55, -50, -40, -30, -20, -10, 0, 10, 20, 25, 30, 40, 50,
          60, 70, 80, 90, 100, 110, 120, 125, 130, 140, 150};
void setup() {
  // Starte serielle Verbindung mit 9600 baud (=Symbole pro Sekunde)
  Serial.begin(9600);
}

void loop() {
  //Schreibe den an Pin0 anliegenden Wert in X_T
  X_T = analogRead(A0);
  //Berechne die Spannung U_T
  U_T = (float(X_T) * 5/1023);
  //Berechne daraus den Wiederstandswert R_T
  R_T = float(R_H) * U_T / (5 - U_T);
  //Suche die beiden Widerstandswerte der Liste zwischen denen R_T
  //liegt und interpoliere die zugehörige Temperatur
  for(int i = 0; i <= sizeof(R) - 1; i = i + 1){
  if(R_T < R[i]){
  T_R = (R_T - R[i-1]) * (T[i] - T[i-1]) / (R[i] - R[i-1]) + T[i-1];
  break; //Austieg aus der for-Schleife
  }
  }
  //Sende Spannung, Widerst., Temp. an den seriellen Monitor
  Serial.print(U_T);
  Serial.print(" V, ");
  Serial.print(R_T);
  Serial.print(" Ohm, ");
  Serial.print(T_R);
  Serial.println(" C");
  delay(500);     //Warte 500 ms
}
```

Code 3.2: Temperaturmessung mit einem Silizium Widerstandsthermometer

3.3 Pulsweitenmodulation einer LED

Benötigte Materialien

- Arduino UNO
- Breadboard
- Jumper Wires
- rote LED
- 160 Ω Widerstand

Das Arduino-Board besitzt zwar 6 analoge Eingänge, allerdings nur digitale Ausgänge mit zwei möglichen Zuständen: 0 V und 5 V. Möchte man die Helligkeit einer LED (**L**ight **E**mitting **D**iode) stufenlos regeln, so ist das nicht ohne weiters möglich. Mit Hilfe der Pulsweitenmodulation und der Trägheit des Auges, kann man dieses Problem umgehen und dabei das gewünschte Ergebnis erzielen.

3.3.1 Physikalisches Prinzip

Die Pulsweitenmodulation kurz PWM ist eine Möglichkeit über die Eigenschaften eines Rechtecksignals ein Signal mit kontinuierlichen Amplitudenwerten auf digitalem Weg zu übertragen. Maßgeblich ist dabei das Verhältnis aus Perioden- und Impulsdauer (Zeit des `HIGH`-Zustandes). Je höher die Amplitudenwerte werden, desto länger werden die Impulswerte und vice versa. In den Kapiteln 3.14 und 3.15 werden wir noch genauer auf die Kommunikation über PWM eingehen. Um mit dem Arduino die Helligkeit einer LED zu steuern, kann man nun, da der Spannungspegel nicht variiert werden kann, mit der Funktion `analogWrite` arbeiten. Sie erzeugt ein pulsweitenmoduliertes Signal an einem Digitalausgang und emuliert dadurch ein analoges Signal. Der Befehl benötigt zwei Argumente - als erstes die Pin-Nummer des gewünschten Digitalausganges und als zweites eine ganze Zahl aus dem Intervall $[0, 255]$ welche ein Parameter für die Impulsdauer ist. Der Einfluss dieser Zahl auf das Ausgangssignal ist in Abb. 3.7 dargestellt. Eine angeschlossene LED leuchtet dabei nur zu den Zeitpunkten an denen das Signal auf `HIGH` ist - genaugenommen blinkt sie etwa 500 mal pro Sekunde und je nach gewähltem Parameter verschieden lange. Da unser Auge jedoch viel zu träge für so hohe Frequenzen ist, scheint die LED mit der zeitlich gemittelten Leistung zu leuchten. Auf dem Arduino-UNO unterstützen die Digitalausgänge 3, 5, 6, 10 und 11 die Pulsweitenmodulation und

sind auf der Platine mit dem Symbol ∼ gekennzeichnet.[4]

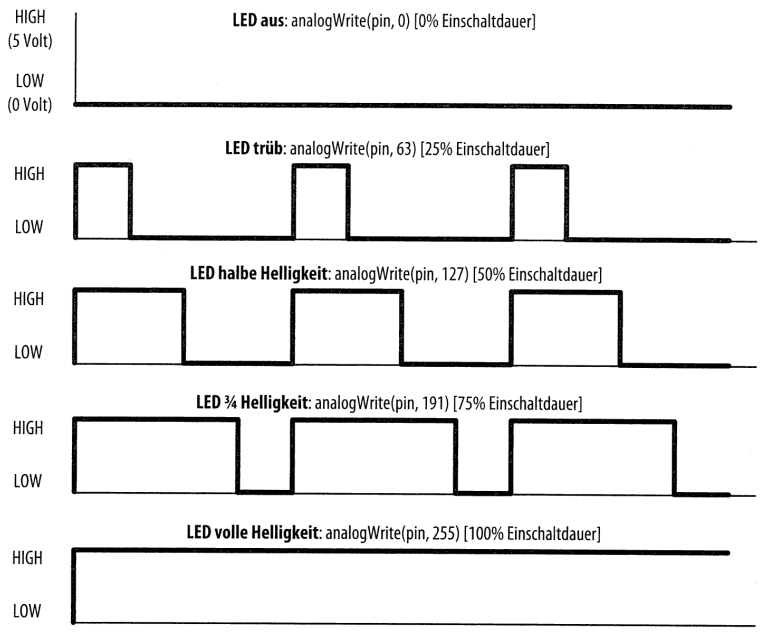

Abb. 3.7: PWM-Ausgabe für verschiedene `analogWrite`-Werte [8]

3.3.2 Aufbau

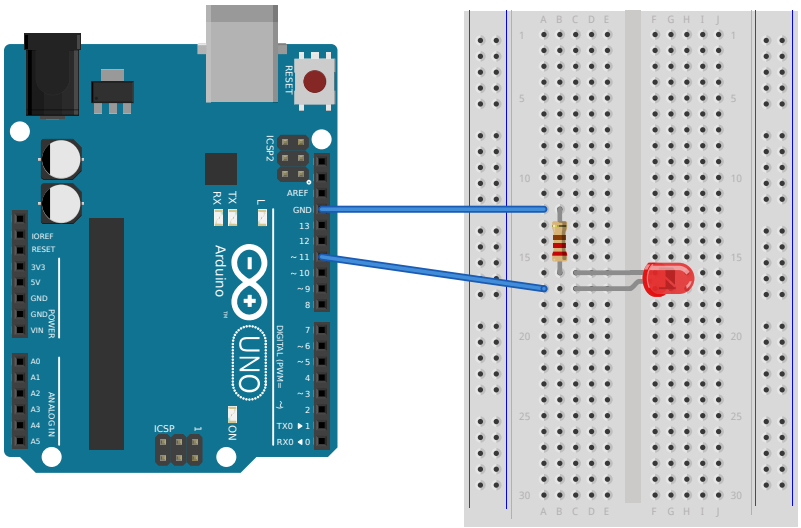

Abb. 3.8: Anschluss einer LED mit Vorwiderstand

Da LEDs empfindlich auf zu hohe Ströme reagieren, muss dieser mit einem geeigneten Vorwiderstand begrenzt werden. Für unseren Aufbau verwenden wir eine rote Standard-LED, die eine Flussspannung[5] von 1,8 V besitzt und mit einem

4) Vgl. [4], [8] und [2]
5) Spannungsabfall bei Betrieb in Durchlassrichtung

Maximalstrom von 20 mA betrieben werden darf. Am Vorwiderstand fallen also $5\,\text{V} - 1{,}8\,\text{V} = 3{,}2\,\text{V}$ ab, womit sich ein benötigter Widerstandswert von $\frac{3{,}2\,\text{V}}{0{,}02\,\text{A}} = 160\,\Omega$ ergibt. Die allgemeine Formel zur Berechnung des Vorwiderstandes lautet:

$$R = \frac{U - U_F}{I}\,. \tag{3.3}$$

R	Vorwiderstand
U	5 V Versorgungsspannung des Arduino
U_F	Flussspannung der LED
I	Maximalstrom der LED

3.3.3 Programmcode

```
1  int led=11;    //Pin-Nummer der LED
2  int z=0;       //Zählervariable
3  int t=10;      //Pause in ms
4  void setup() {
5   //Setze Pin der LED als OUTPUT
6   pinMode(led, OUTPUT);
7  }
8
9  void loop(){
10   //Erhöhe schrittweise den Parameter z
11   for(z=0;z<=255;z=z+1){
12   analogWrite(led, z);
13   delay(t);
14   }
15 }
```

Code 3.3: Pulsweitenmodulation einer LED

Das Programm zählt mit der Variable z, die als zweiter Parameter von `analogWrite` dient, von 0 bis 255 und pausiert nach jeder Erhöhung für die Zeitspanne t. Die Impulsdauer des PWM-Signals wird dadurch schrittweise erhöht, wodurch die LED immer heller leuchtet (siehe Abb. 3.7). Für das träge menschliche Auge gehen diese Vorgänge viel zu schnell vor sich, und die LED scheint ihre Helligkeit kontinuierlich zu erhöhen.

3.4 Taster, Schalter und eindeutige Zustände

Benötigte Materialien

- Arduino UNO
- Breadboard
- Jumper Wires
- rote LED
- Jeweils einen 160 Ω und 10 kΩ Widerstand
- Taster

3.4.1 Prinzip

Um mit einem laufenden Sketch zu interagieren, kann man Taster oder Schalter verwenden. Beide haben gemein, dass man mit ihnen elektrische Schaltkreise schließen, oder öffnen kann - Taster nur für kurze Zeit, Schalter bis zur nächsten Betätigung. Was benötigt wird, hängt vom individuellen Fall und manchmal einfach vom persönlichen Geschmack ab. In der Elektronik ist es wichtig, dass mit eindeutigen Zuständen gearbeitet wird. Für geschlossene Schalter und Taster gibt es hier kein Problem - wohl aber bei offenen! Möchte man eine Eingabe mit Hilfe eines Schalters machen, könnte man einfach die Versorgungsspannung über den Schalter mit einem digitalem Eingang verbinden und dessen aktuellen Zustand abfragen. Die Praxis zeigt, dass man an Eingängen deren Potential nicht definiert ist ziemlich viel Unsinn messen kann. Um dies zu verhindern, zwingt man den Eingangspegel bei offenem Schalter, mit Hilfe eines hohen Widerstandes, der den Digitaleingang mit Masse verbindet auf `LOW` (siehe Abb. 3.9). Da der Widerstand die Spannung am offenen Eingang herunter zieht, wird er Pulldown-Widerstand genannt. Schließt man den Schalter, so ist der Digitaleingang direkt mit der Versorgungsspannung verbunden und besitzt den eindeutigen Zustand `HIGH`. Neben dem Pulldown-Widerstand gibt es noch den Pullup-Widerstand, der mit invertierter Logik arbeitet. Die Schaltung ist analog zu oben (siehe Abb. 3.10), nur dass die Anschlüsse für Masse und Versorgungsspannung vertauscht werden. Bei offenem Schalter zieht der Pullup-Widerstand den Digitaleingang auf `HIGH`, bei geschlossenem Schalter sind Eingang und Masse direkt verbunden.

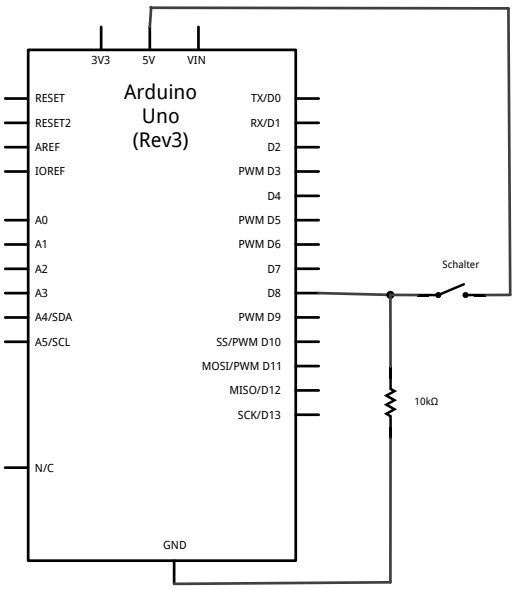

Abb. 3.9: Schalter mit Pulldown-Widerstand

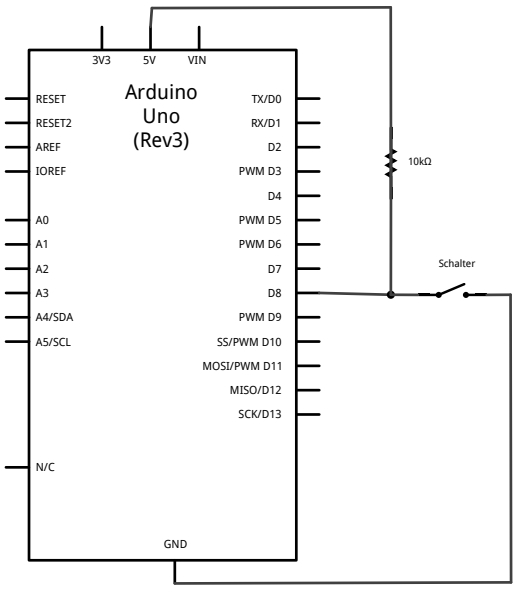

Abb. 3.10: Schalter mit Pullup-Widerstand

3.4.2 Aufbau

Wir wollen mit Hilfe eines Pulldown-Widerstandes, die Schaltung aus 3.3 so erweitern, dass die LED mittels Taster Aus und Ein geschaltet werden kann.

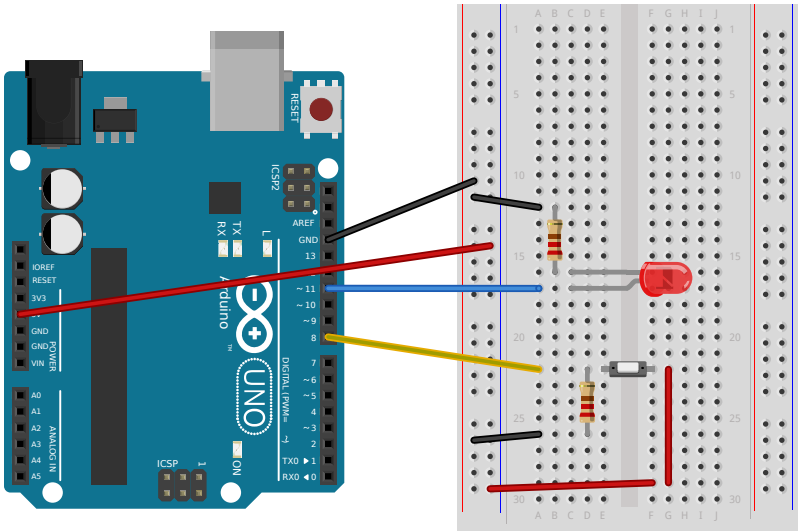

Abb. 3.11: Steuerung einer LED durch Taster mit Pulldown-Widerstand

3.4.3 Programmcode

```
1 int led=11;       //LED auf Pin 11
2 int taster=8;     //Taster auf Pin 8
```

```
3  int zust;          //Varible für Zustand auf Pin 8
4
5  void setup(){
6
7    pinMode(led, OUTPUT);     //Setze Pin der LED als OUTPUT
8    pinMode(taster, INPUT);   //Setzt Pin des Tasters als INPUT
9  }
10 void loop(){
11   //Lese den Zustand des Tasters
12   zust = digitalRead(taster);
13
14   //Taster offen-->LED aus, Taster geschlossen-->LED ein
15   if(zust == LOW){
16   digitalWrite(led, LOW);
17   }
18   else
19   {
20   digitalWrite(led, HIGH);
21   }
22 }
```

Code 3.4: Steuerung einer LED durch Taster mit Pulldown-Widerstand

Wie die meisten Mikrocontroller besitzt auch der Arduino bereits eingebaute Pullup-Widerstände, die softwaremäßig eingeschaltet werden können. Sie besitzen einen Wert von ca. 20 kΩ und werden aktiviert, indem man den Tasterpin nach seiner Deklaration als `INPUT`, mittels `digitalWrite` auf `HIGH` setzt. Zu beachten ist, dass damit der externe Pulldown-Widerstand entfällt, der Taster nun mit Masse verbunden werden muss (vgl. Abb. 3.10) und sich die Logik umkehrt. In Code 24 ist der modifizierte Sketch angeführt.[6]

```
1  int led=11;        //LED auf Pin 11
2  int taster=8;      //Taster auf Pin 8
3  int zust;          //Varible für Zustand auf Pin 8
4
5  void setup(){
6
7    pinMode(led, OUTPUT);     //Setze Pin der LED als OUTPUT
8    pinMode(taster, INPUT);   //Setzt Pin des Tasters als INPUT
9    digitalWrite(taster, HIGH); //Aktiviere internen Pull-Up
10 }
11 void loop(){
12   //Lese den Zustand des Tasters
13   zust = digitalRead(taster);
```

6) Vgl. [2]

```
14
15    //Taster offen -->LED aus, Taster geschlossen -->LED ein
16    if(zust == LOW){
17    digitalWrite(led, HIGH);
18    }
19    else
20    {
21    digitalWrite(led, LOW);
22    }
23  }
```

Code 3.5: Steuerung einer LED durch Taster und internem Pullup-Widerstand

3.5 Drehpotentiometer

Benötigte Materialien

- Arduino UNO
- Breadboard
- Jumper Wires
- rote LED
- 160 Ω Widerstand
- 10 kΩ Drehpotentiometer

3.5.1 Physikalisches Prinzip

Potentiometer oder kurz Potis, sind Widerstände deren Wert sich variabel einstellen lässt. Auf einem isolierendem Trägermaterial ist meist eine Widerstandsbahn aus Graphit, oder eine Drahtwicklung aufgebracht, die von einem Schleifkontakt berührt wird. Dieser Kontakt unterteilt die Widerstandsbahn in zwei Teile. Als Anschlüsse werden jeweils die Enden der Widerstandsbahn und der Schleifer nach außen geführt. Liegt eine Spannung an, so arbeitet das Potentiometer als Spannungsteiler wie Abb. 3.12 zeigt. In der dargestellten Schaltung, lässt sich durch variieren der Schleiferposition eine Ausgangsspannung U_a abgreifen, die zwischen der Versorgungsspannung U und 0 V liegt. Der Gesamtwiderstand R ist charakteristisch für Potentiometer - im Zusammenhang mit Arduino, bieten sich Potis mit 10 kΩ an. Der Schleifer teilt R in die Teilwiderstände R_1 und R_2, wobei $R = R_1 + R_2$ gilt.

Insgesamt gilt mit den Teilspannungen U_1 und U_2 folgender Zusammenhang:[7]

$$\frac{U_1}{U_2} = \frac{R_1}{R_2} .$$

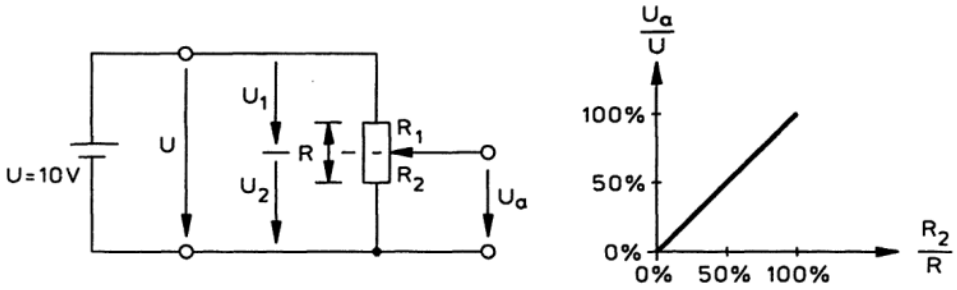

Abb. 3.12: Schaltung und Kennlinie für ein Potentiometer als Spannungsteiler [9]

3.5.2 Aufbau

Mit Hilfe des Drehpotentiometers ist es möglich die Helligkeit einer LED stufenlos einstellen zu können. Wir verwenden dazu die Spannungsteiler Eigenschaft aus Abb. 3.12 in der in Abb. 3.13 dargestellten Schaltung.

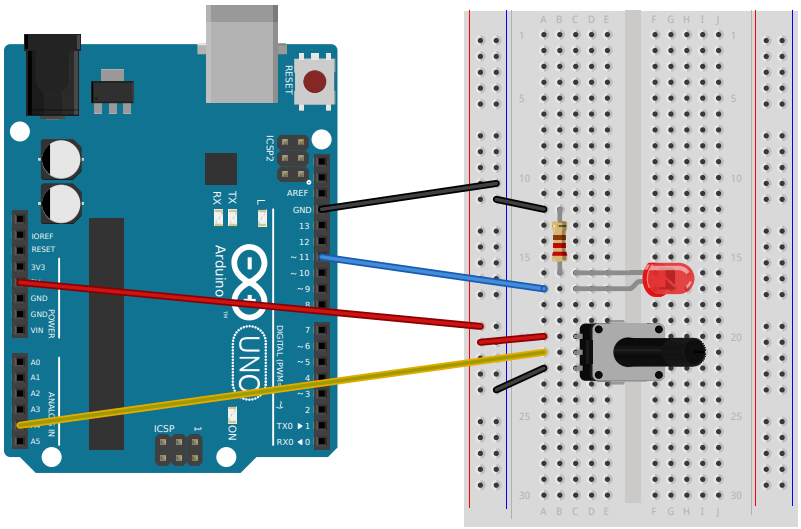

Abb. 3.13: Trimmung einer LED mit Hilfe eines Drehtpotentiometers

7) Vgl. [9]

3.5.3 Programmcode

```
const int led=11;      //Pin-Nummer der LED
const int pot=4;       //Poti auf Pin 4
int val;               //Wert am analogEingang 4
int val_t;             //Transformierter Wert
void setup() {

 pinMode(led, OUTPUT); //Setzt Pin der LED als OUTPUT
 pinMode(pot, INPUT);  //Setzt Pin des Potis als INPUT
 Serial.begin(9600);
}

void loop(){
  val = analogRead(pot);
  //Rechne die gemessene Spannung in Werte zwischen [0,255] um
  val_t=map(val,0,1023,0,255);
  analogWrite(led, val_t);
  //Zusätzliche Ausgabe der Werte über Serial Monitor
  Serial.print(val);
  Serial.print(", ");
  Serial.println(val_t);
}
```

Code 3.6: Trimmung einer LED mit Hilfe eines Drehtpotentiometers

Wie in 3.1 schon angekündigt, kann man die Funktion `map` dazu verwenden um auf einfache Art eine Transformation von einem Intervall $[a, b]$ auf ein Intervall $[c, d]$ durchzuführen. Sie verlangt fünf Parameter: Wert innerhalb des ersten Intervalls, a, b, c und d. Im Programm nutzen wir diese Funktion, um die diskreten Spannungswerte, die am analogen Eingang gemessen werden und innerhalb $[0, 1023]$ liegen, an den Bereich $[0, 255]$ von `analogWrite` anzupassen. Dadurch steuern wir mit dem Potentiometer direkt den zweiten Parameter von `analogWrite`, der für die Impulslänge und damit für die Helligkeit der LED verantwortlich ist.

3.6 Relais

Benötigte Materialien

- Arduino UNO
- Breadboard
- Jumper Wires
- rote LED
- 160 Ω Widerstand
- 4 Relay Module

Bis jetzt haben wir gesehen, dass wir kleinere Lasten, wie zum Beispiel LEDs, mit dem Arduino betreiben können. Durch die Betriebsspannung von 5 V und einem maximalen Strom von 20 mA, ist jedoch der Betrieb von Geräten mit höherem Leistungsbedarf unmöglich. Abhilfe schafft hier das Relais. Obwohl es schon seit dem 19. Jahrhundert im Einsatz ist, spielt es noch heute eine bedeutende Rolle, was nach wie vor steigende Produktionszahlen belegen.[8]

3.6.1 Physikalisches Prinzip

Das Relais ist ein elektromagnetischer Schalter, der es ermöglicht mit einem kleinen Steuerstrom, einen hohen Laststrom zu steuern. Es besteht aus einer Spule die auf einem ferromagnetischen Kern sitzt, einem ferromagnetischen Anker, einer Feder und zwei Kontakten (siehe Abb. 3.14). Der Steuerstrom (bzw. Erregerstrom I_E) läuft durch die Spule und erzeugt ein Magnetfeld. Dieses wiederum sorgt dafür, dass der Anker in Richtung Kern gezogen und damit gleichzeitig ein elektrischer Kontakt auf der Lastseite geschlossen wird. Sobald der Steuerstrom abgeschaltet wird, bringt eine Feder den Anker - und damit auch die Kontakte - in die ursprüngliche Position zurück. Ein großer Vorteil dieses Prinzips ist es, dass der Steuerstromkreis und der Laststromkreis galvanisch voneinander getrennt sind. Somit ist es beispielsweise kein Problem, das Relais mit einer Gleichspannung (z.B. 5 V-Arduino) zu betreiben und damit eine Wechselspannung (z.B. Geräte am 230 V-Stromnetz) zu schalten. Zu beachten ist dabei, dass das Relais auch tatsächlich für die verwendeten Spannungen ausgelegt sein muss. Da es beim Abschaltvorgang zu einer Selbstinduktion der Spule kommt, ist die Spannung auf der Steuerseite unter Umständen deutlich höher als

8) Vgl. [10]

die 5 V des Arduinos. Um mögliche Schäden zu verhindern, wird eine Diode parallel zur Spule geschaltet. Im Regelbetrieb wird sie in Sperrrichtung betrieben, kommt es beim Abschalten zum negativ gerichteten Spannungsstoß durch die Selbstinduktion, so wird die Diode leitend und sorgt für einen Kurzschluss in der Spule. Dadurch wird die angeschlossene Steuerelektronik geschützt.[9]

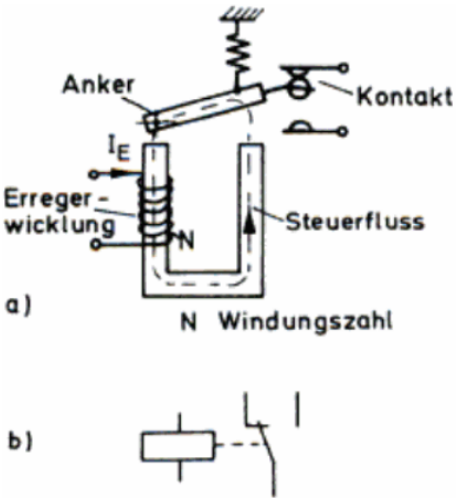

Abb. 3.14: a) Prinzipbild und b) Schaltzeichen eines Relais [11]

3.6.2 Aufbau

Wir verwenden ein Relaismodul mit vier getrennt voneinander schaltbaren Relais. Es besitzt die Anschlüsse GND und Vcc für die Spannungsversorgung des Steuerstromkreises und für jedes Relais noch einen digitalen Kanal, der das jeweilige Relais aktiviert, sobald er auf `LOW` gesetzt wird. In diesem kurzen Projekt möchten wir das Relais periodisch ein- und ausschalten. Als Steuerkanal wählen wir Pin 8, dessen Zustand bestimmt, ob das Relais offen oder geschlossen ist. Auf der Lastseite könnten wir Spannungen bis 230 V betreiben, wir begnügen uns hier mit den 5 V des Arduinos und schalten damit eine LED, um uns von der Funktion des Relais zu überzeugen. Mit den notwendigen Sicherheitsvorkehrungen könnten wir jedoch praktisch jedes Haushaltsgerät Steuern. Der Aufbau ist in Abb. 3.15 dargestellt.

3.6.3 Programmcode

Das folgende Programm, übernimmt die zuvor beschriebene Funktion. Alle 2 s wird der Zustand des Steuerkanals geändert und damit auch der des Relais.

9) Vgl. [10] und [11]

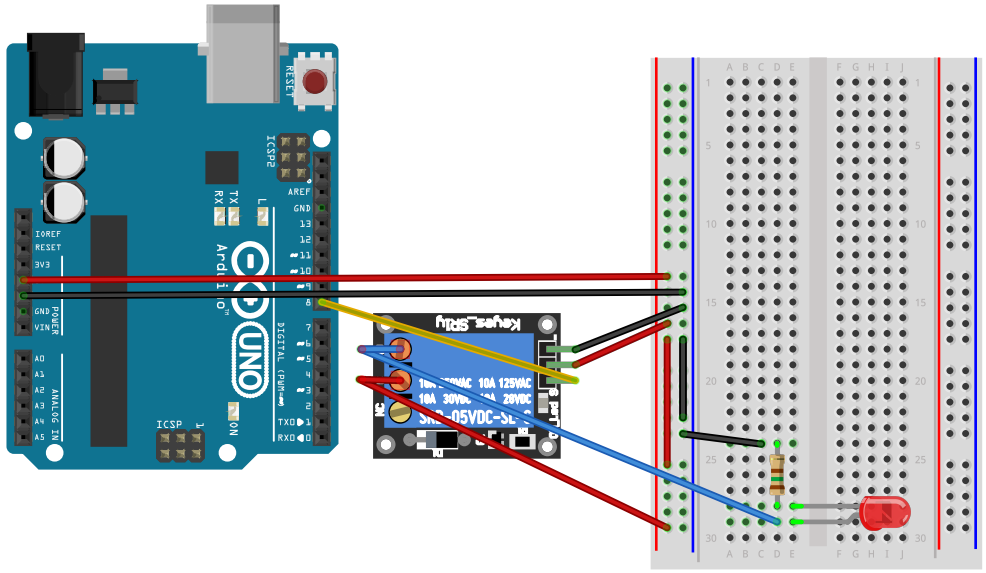

Abb. 3.15: Steuerung einer LED durch ein Relais

```
int st=8;         //Steuerkanal auf Pin 8
int zust=1;       //Varible für Zustand des Steuerkanals

void setup(){
  pinMode(st,OUTPUT); //Setze Steuerkanal auf OUTPUT
}
void loop(){
  digitalWrite(st,zust);  //Setze Steuerkanal auf den Wert von zust
  delay(2000);  //warte 2s
  zust=!zust;   //ändere den logischen Wert von zust
}
```

Code 3.7: Steuerung einer LED durch ein Relais

3.7 Protokolle für digitale Datenübertragung

Es gibt eine Vielzahl von Protokollen zur digitalen Datenübertragung. Sie alle sollen eine möglichst ressourcenschonende Kommunikation ermöglichen und dabei zuverlässig arbeiten. Ein Vorteil den alle Verfahren in sich vereinigen, ist die weitgehende Unabhängigkeit von der Länge der Leitungen, da prinzipiell nur mit binären Zuständen gearbeitet wird. Eine Verfälschung von Messwerten durch lange Übertragungsleitungen zwischen Sensoren und Mikrocontroller ist damit ausgeschlossen, solange

die Zustände `HIGH` und `LOW` eindeutig übertragen werden. Weiters kann die Kommunikation vieler Geräte über wenige gemeinsam genutzte Leitungen erfolgen. Im Folgenden seinen zwei sehr verbreitete digitale Kommunikationsstandards beschrieben.

3.7.1 I²C

I²C steht für **I**nter **I**ntegrated **C**iruit (auch mit IIC abgekürzt). Es ist ein von der Firma Philips entwickeltes Busprotokoll, das die Kommunikation zwischen Chips regelt und mittlerweile zu einem weit verbreitetem Standard geworden ist. Für den gesamten Datenaustausch werden nur zwei Leitungen, *Serial Clock Line* (SCL) und *Serial Data Line* (SDA) benötigt, die mittels Pullup-Widerstand mit der Versorgungsspannung verbunden sind und im Ruhezustand auf `HIGH` liegen. Die SCL-Leitung überträgt das Taktsignal - also gewissermaßen das gemeinsame „Sprechtempo“ - und bestimmt damit die Datenrate, die von 100 kBit/s im Standard Mode, bis zu 5 MBit/s im Ultra Fast Mode reicht. Auf die SDA-Leitung werden die zu übertragenden Daten gelegt. Ein Chip (meist ein Mikrocontroller) fungiert dabei als Master und koordiniert den Austausch, der in beide Richtungen erfolgen kann, mit weiteren Chips die als Slave bezeichnet werden. Der genaue Kommunikationsablauf ist in den Abbildungen 3.16-3.18 dargestellt.

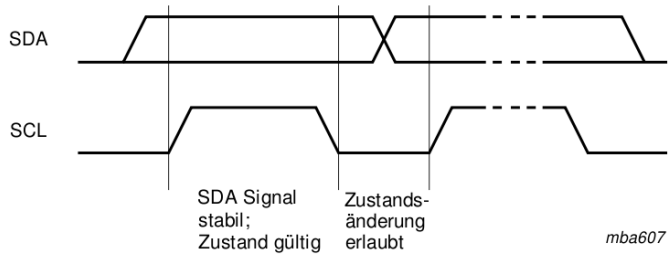

Abb. 3.16: Bit-Transfer mittels I²C. Übersetzt übernommen aus [12]

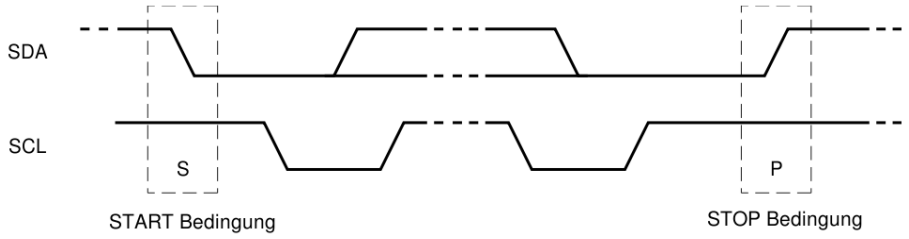

Abb. 3.17: Start und Stop Signale der I²C Datenübertragung. Übersetzt übernommen aus [12]

Daten werden in 8-Bit-Paketen übertragen, beginnend mit dem höchstwertigen Bit. Das sendende Gerät darf den Zustand der SDA-Leitung nur zu jenen Zeitpunkten ändern, zu denen das Taktsignal auf `LOW` ist. Wechselt die SCL-Leitung auf `HIGH`, so liegt dann das zu übermittelnde Bit bereits stabil auf der SDA-Leitung und kann vom empfangenden Gerät als Informationseinheit ausgelesen werden (siehe Abb. 3.16). Wie bereits erwähnt startet der Master den Transportvorgang und beendet ihn auch. Dies geschieht über einen Wechsel des SDA-Signals während das Taktsignal auf `HIGH` ist - mit sogenannten Steuersignalen. Bei fallender Flanke des SDA-Signals beginnt die Übertragung und bei steigender Flanke wird sie beendet (siehe Abb. 3.17). Dies ist der Grund, weshalb die Datensignale stabil anliegen müssen. Würden sie während SCL auf `HIGH` ist wechseln, so würde ein Datenbit als Steuerbit missinterpretiert und die gesamte Kommunikation wäre gestört. Am Ende jedes 8-Bit-Paktes muss der Empfänger noch mit einem `LOW` Acknowledge-Bit (ACK) antworten um zu signalisieren, dass die Daten erfolgreich empfangen wurden - andernfalls mit `HIGH`. Nun muss noch berücksichtigt werden, dass mehrere Slaves existieren können. Dies erreicht man, indem vor der eigentlichen Datenübertragung ein 8-Bit-Adressblock vom Master gesendet wird, der die Adresse des Slave enthält. Danach folgt ein weiteres Bit mit der Anweisung an den Slave zu senden (`HIGH`), oder zu empfangen (`LOW`). Der gesamte Kommunikationsprozess besteht also, wie in Abb. 3.18 dargestellt, aus einem Startsignal gefolgt von der 7-Bit langen Slave-Adresse und einem weiteren Bit mit der Lese- bzw Schreibanweisung. Erst danach erfolgt die Datenübertragung in 8-Bit-Paketen, die jeweils mit einem ACK bestätigt werden müssen. Abgeschlossen wird der Datenaustausch mit dem Stopsignal.[10]

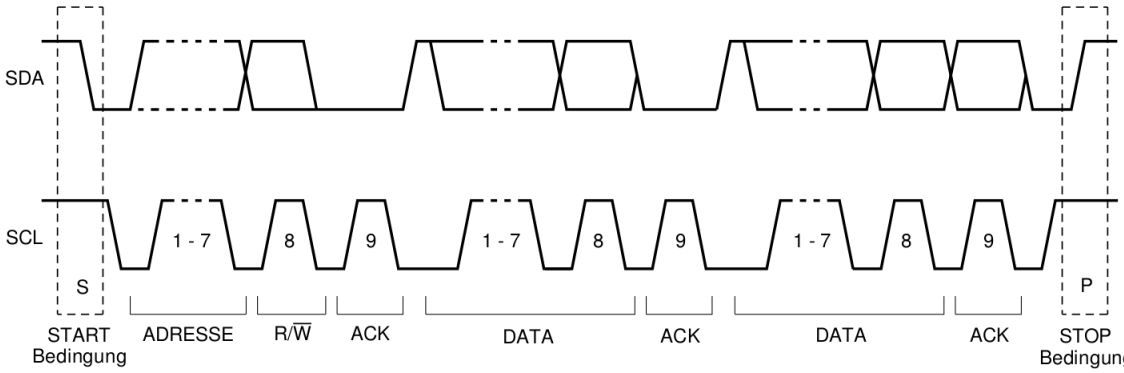

Abb. 3.18: Eine komplette Datenübertragung mit I^2C. Übersetzt übernommen aus [12]

10) Vgl. [12], [13], [8] und [2]

3.7.2 SPI

SPI steht für **S**erial **P**eripheral **I**nterface. Es wurde von Motorola entwickelt und wird in den selben Bereichen wie I^2C eingesetzt. Auch hier gibt es einen Master, der mit mehreren Slaves kommuniziert. Dies geschieht über zwei Datenleitungen *Master Out Slave In* (MOSI), *Master In Slave Out* (MISO) und zwei Steuerleitungen *Serial Clock* (SLK) und *Slave Select* (SS). Die ersten drei Leitungen werden von allen Geräten gemeinsam verwendet. Mit Hilfe der SS-Leitungen, die jeden Slave einzeln mit dem Master verbinden, kann der Master die einzelnen Slaves ansprechen, wenn er eine Kommunikation mit ihnen wünscht. Im Vergleich zu I^2C sind deutlich mehr Leitungen notwendig, dafür entfällt im Protokoll die Adressierung der Slaves und es kann gleichzeitig gesendet und empfangen werden, was die Kommunikation beschleunigt. Es gibt vier Betriebsmodi, die in Tab. 3.1 angeführt sind und die die Zuordnung zwischen Takt- und Datensignal festlegen. Die Clock Polarity regelt die

Tab. 3.1: SPI-Betriebsmodi

Modus	Clock Plarity (CPOL)	Clock Phase (CPHA)
0	0	0
1	0	1
2	1	0
3	1	1

Logik des Taktsignals. Für CPOL gleich null, ist die CLK-Leitung im Ruhezustand auf `LOW` - für CPOL gleich eins, ist sie auf `HIGH`. Mittels des Clock Phase Parameters kann eingestellt werden, mit welcher Taktflanke die Daten übernommen werden sollen. Mit der ersten steigenden Flanke für CPHA gleich null - mit der ersten fallenden Flanke für CPHA gleich eins. Wie bei I^2C werden die Daten auch bei SPI Byte für Byte übertragen. In Abb. 3.19 sind die Signalverläufe für den Modus 0 dargestellt. Zunächst wird dabei der vom Master gewünschte Slave ausgewählt, indem seine SS-Leitung auf `LOW` gesetzt wird. Danach werden die höchstwertigsten Bits (MSB für **M**ost **S**ignifant **B**it), der zu übertragenden Bytes, bereits auf die MOSI- bzw. MISO-Leitungen gelegt und mit der ersten ansteigenden Taktflanke vom Kommunikationspartner übernommen. Danach folgen die weiteren Bits. Setzt der Master die SS-Leitung wieder auf `HIGH`, so beendet er damit die Übertragung.[11]

11) Vgl. [13], [14] und [8]

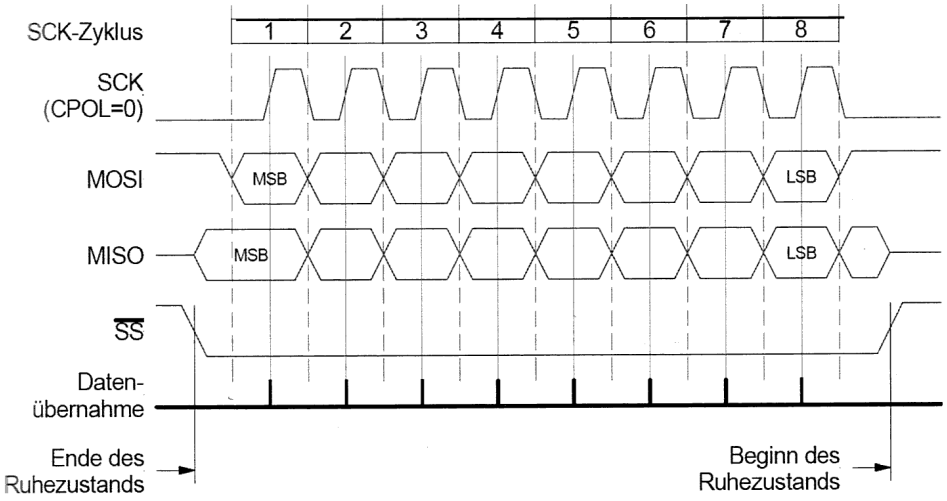

Abb. 3.19: SPI-Übertragung im Modus 0 [13]

3.8 Arbeiten mit externen Bibliotheken

Wie im vorigen Kapitel gezeigt, hat man mit digitalen Übertragungsprotokollen wie I^2C und SPI die Möglichkeit, eine Kommunikation zwischen vielen Geräten zu ermöglichen - bei geringem Hardwareaufwand. Der Preis dafür ist eine relativ unübersichtliche Übertragungsprozedur. Bibliotheken wie Wire- oder SPI-Library helfen hier weiter, sie regeln die `LOW-HIGH`-Ebene über einfache Befehle, die im Programmcode in der IDE-Umgebung implementiert werden. Doch was ist eine Bibliothek? Im Wesentlichen ein ausgelagerter Programmcode, ein Unterprogramm das gewisse Aufgaben erfüllt die immer wieder benötigt werden. Möchte man beispielsweise über I^2C eine Kommunikation mit einem Slave beginnen, so ist jedes Mal die selbe Prozedur notwendig. Dadurch wird der Code sehr unübersichtlich und es ist viel Aufwand, immer wieder die gleichen Zeilen zu schreiben. Aus diesem Grund verwendet man Bibliotheken wo immer es möglich ist und es Sinn macht. Die oben erwähnten Bibliotheken sind bereits in der IDE-Umgebung enthalten. Es gibt jedoch auch Bibliotheken, die Privatpersonen oder Hersteller von Sensoren entwickelt haben und zur Verfügung stellen - sogenannte externe Bibliotheken. Meistens kann man sie als ZIP-Ordner downloaden und anschließend über *Sketch* → *Include Library* → *Add .ZIP Library* in das System integrieren und verwenden (siehe Abb. 3.20). Sie enthalten häufig auch Beispiel-Sketches, sodass man relativ schnell den richtigen Umgang mit ihnen erlernt. Wichtig ist, dass man am Beginn des Sketch die Bibliotheken die man verwenden möchte mittels `#include` in den Programmcode einbindet. Eine praktische Umsetzung wird in 3.9 und den folgenden Kapiteln gezeigt.

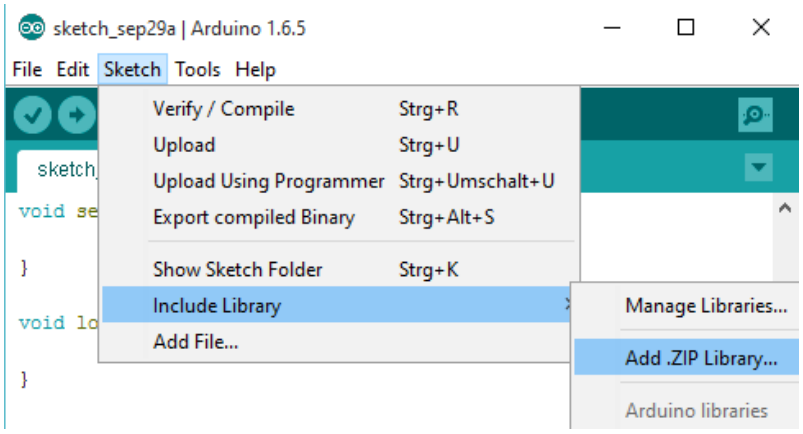

Abb. 3.20: Einbindung externer Bibliotheken als ZIP-Datei

3.9 Luftdrucksensor

Benötigte Materialien

- Arduino UNO
- Breadboard
- Jumper Wires
- Luftdrucksensor BMP180

3.9.1 Physikalisches Prinzip

Der verwendete Sensor BMP180 arbeitet wie viele andere Sensoren, auf Basis des piezoresistiven Effektes. Dabei nutzt man die Widerstandsänderung eines dotierten Halbleiters bei Belastung mit einer mechanischen Spannung. Natürlich ändert sich der elektrische Widerstand durch die bloße geometrische Verformung - dies ist jedoch nur ein Teil. Den weit größeren Einfluss besitzt der piezoresitive Effekt, der nun für den bekannten Halbleiter Silizium kurz beschrieben werden soll.
Reines Silizium besitzt einen spezifischen Widerstand der unabhängig von der Richtung des angelegten elektrischen Feldes ist. Durch Dotierung schleust man zusätzliche Ladungsträger in das Material, welche in ihrer Beweglichkeit jedoch abhängig von der Symmetrie der Kristallstruktur sind. Legt man nun eine mechanische Spannung an, so verzerrt sich das Kristallgitter, was eine Änderung der Ladungsträgerbeweglichkeit nach sich zieht. Dies wiederum hat einen direkten Einfluss auf die Leitfähigkeit des Materials und damit auf den elektrischen Widerstand. In der Praxis bringt man meist vier solcher piezoresistiven Widerstände in einer Brücken-

schaltung auf einer Membran an. Diese Membran trennt eine Referenzkammer mit definiertem Druck von der Atmosphäre und verformt sich je nach vorherrschendem Außendruck. Mit der Membran verformen sich auch die auf ihr befindlichen Widerstände. Die sich daraus ergebenden veränderlichen Widerstandswerte werden von einer integrierten Messschaltung ausgewertet und zur Berechnung des vorherrschenden Druckes verwendet.[12]

3.9.2 Atmosphärischer Druck

In der Physik definiert man den Druck als Quotient von Kraft und Fläche. Wenn man vom Luftdruck spricht, so ist in der Meteorologie der hydrostatische Druck gemeint. Er entsteht durch die ungeordnete Bewegung der Gasmoleküle in der Luft und entspricht im hydrostatischen Gleichgewicht dem Gewicht der Luftsäule über dem betrachteten Ort, welche bis zur Grenze der Atmosphäre reicht. Mit zunehmender Höhe nimmt daher der hydrostatische Luftdruck ab, was durch folgende Differenzialgleichung - der hydrostatischen Grundgleichung - beschrieben wird:

$$\frac{\partial p}{\partial z} = -\rho g\,. \tag{3.4}$$

Für trockene Luft, welche man näherungsweise als ideales Gas betrachtet, kann man die obige Gleichung mit Hilfe der allgemeinen Gasgleichung für ideale Gase $pV = nRT = \frac{m}{M}RT$, der Definition $R_L = \frac{R}{M_L}$ und einer kurzen Umformung auf $\rho = \frac{p}{R_L T}$ auch als

$$\frac{\partial p}{\partial z} = -\frac{p}{R_L T} g \tag{3.5}$$

schreiben.[13] Nimmt man die Temperatur konstant über die Höhe an, so erhält man als Lösung die barometrische Höhenformel

$$p\,(z) = p_0 \exp\left(-\frac{g\,(z - z_0)}{R_L T}\right), \tag{3.6}$$

welche in Abb. 3.21 grafisch dargestellt ist. Der gemessene Druck ist abhängig von der Höhe des Standortes und den meteorologischen Bedingungen. Um die Messwerte verschiedener Orte miteinander vergleichen zu können, müssen die absoluten Werte mit der eben gewonnenen Gleichung 3.6 auf Meeresniveau reduziert werden. Es geht

12) Vgl. [15] und [16]

13) Druck: p, Höhe: z, Dichte: ρ, Molenzahl: n, Erdbeschleunigung: g, allgemeine Gaskonstante: R, Masse: m, Molmasse: M, Molmasse der trockenen Luft: M_L, spezifische Gaskonstante für trockene Luft: R_L

also darum, welcher Luftdruck vorhanden wäre, wenn der jeweilige Messort auf Meereshöhe abgesenkt werden würde. Man spricht dann vom reduzierten Luftdruck.[14]

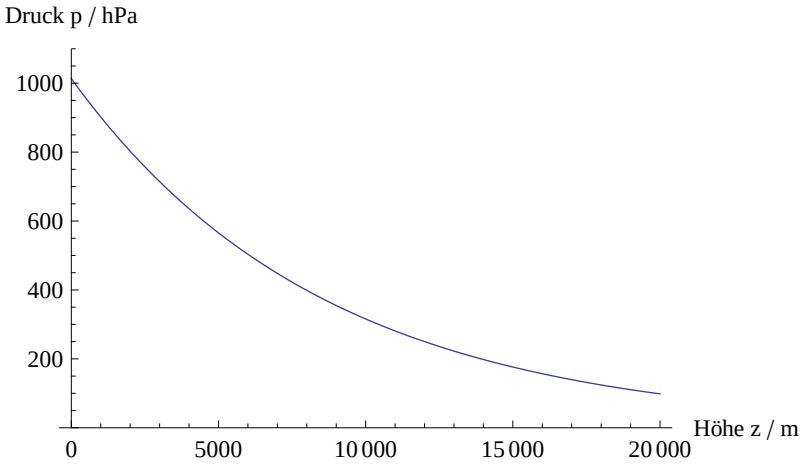

Abb. 3.21: Höhenabhängiger Druckverlauf für eine isotherme Atmosphäre. Berechnet mit Gleichung 3.6, $p_0 = 1013{,}25\,\mathrm{hPa}$, $z_0 = 0\,\mathrm{m}$, $R_L = 287\,\mathrm{J\,kg^{-1}\,K^{-1}}$ und $T = 297{,}15\,\mathrm{K}$

3.9.3 Aufbau

Wie vorhin bereits erwähnt, verwenden wir den Bosch BMP180 Luftdrucksensor. Das Datenblatt [17] kann problemlos von der Herstellerseite heruntergeladen werden und gibt Aufschluss über Funktionsweise und Pinbelegung. Der Sensor kommuniziert über I^2C und besitzt vier Anschlüsse: VIN, GND, SCL und SDA. Wichtig ist, dass an VIN die 3,3 V anliegen, die der Arduino zur Verfügung stellt - die sonst üblichen 5 V würden den Sensor beschädigen. In Abb. 3.22 ist der sehr einfache Aufbau dargestellt, wobei die SDA-Leitung braun und die SCL-Leitung weiß ausgeführt sind.

3.9.4 Programmcode

Für unseren Sensor gibt es eine fertige Bibliothek, die unter `https://github.com/sparkfun/BMP180_Breakout_Arduino_Library/archive/master.zip` heruntergeladen und wie in 3.8 beschrieben, in die Entwicklungsumgebung integriert werden kann. Mit den beiden mitgelieferten Beispielsketches kann man schnell den Sensor in Betrieb nehmen und auf seine Funktion hin überprüfen. Im folgenden Programmcode

14) Dieser Abschnitt basiert auf der von mir verfassten Bachelorarbeit: „Methoden der Wettervorhersage“.

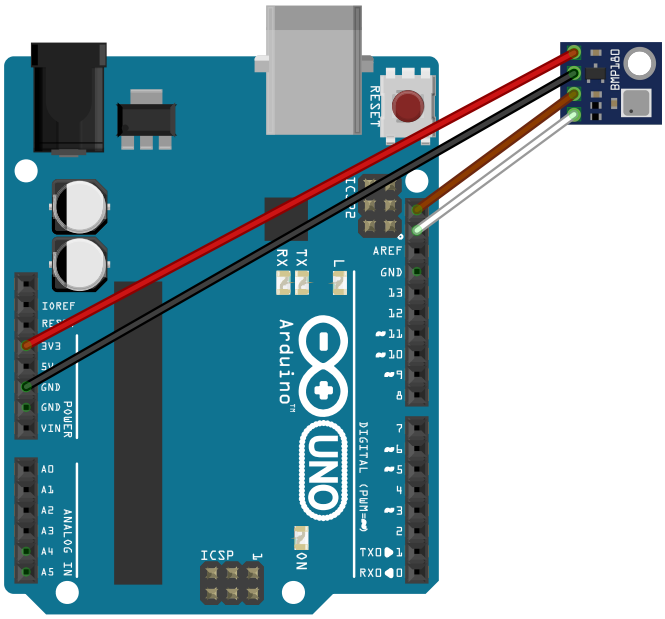

Abb. 3.22: Anschluss des BMP180 an den Arduino

sind die grundlegenden Befehle zum Ansprechen des Sensors - unter Verwendung der externen Bibliothek - dargestellt.

```
#include <SFE_BMP180.h>  //Einbinden der externen Bibliothek
#include <Wire.h>        //Bibliothek für I2C und SPI Kommunikation

SFE_BMP180 pressure;     //definiere pressure als SFE_BMP180 Objekt
const int H=365;         //Seehöhe des Standortes in Meter
char status;
double T,P,p0;

void setup()
{
  Serial.begin(9600);    //Starte serielle Verbindung
  pressure.begin();      //Initialisiere den Sensor
}
void loop()
{
  status = pressure.startTemperature();//Starte Temperaturmessung
  delay(status);
  status = pressure.getTemperature(T); //Speichere Temperatur in T
  delay(status);
  status = pressure.startPressure(3);  //Starte Druckmessung
```

```
21   delay(status);
22   status = pressure.getPressure(P,T);  //Berechnung des Druckes aus
23   //der Messung und aus T
24
25     p0 = pressure.sealevel(P,H);       //Berechnung des reduzierten
26   //Druckes
27   Serial.println(T,2);
28   Serial.println(P,2);
29   Serial.println(p0,2);
30   delay(5000);
31 }
```

Code 3.8: Auslesen eines BMP180 Luftdrucksensors

In Zeile 4 wird ein `SFE_BMP180` Objekt mit dem Namen `pressure` erstellt für das die externe Bibliothek einige Funktionen bereitstellt, die ein einfaches Auslesen des Sensors ermöglichen. Nach dem Start der seriellen Verbindung muss der Sensor mit `pressure.begin()` initialisiert werden. Es werden dabei unter anderem für die Berechnung wichtige Werte geladen, die für eine fehlerfreie Ausgabe des BMP180 notwendig sind. Bei Bedarf liefert diese Funktion `1` oder `0` zurück - je nachdem ob die Initialisierung erfolgreich war oder nicht. Zunächst muss die Temperatur gemessen werden, da sie notwendig ist, um aus den Druckmessungen den tatsächlichen Luftdruck zu berechnen. In der `loop`-Routine wird daher dem Sensor mit `pressure.startTemperature()` mitgeteilt, dass eine Temperaturmessung erfolgen soll. Der Rückgabewert dieser Funktion wird in der Variable `status` gespeichert und ist die Dauer in Millisekunden, welche die Messung benötigen wird. Nach Abwarten dieser Zeit wird die Temperatur mit der Funktion `pressure.getTemperature(T)` in der Variablen `T` gespeichert. Bei einer fehlerfreien Messung ist der Rückgabewert der Funktion `1`, andernfalls `0`. Somit ist die Temperaturmessung abgeschlossen und wir teilen dem Sensor mittels `pressure.startPressure(3)` mit, dass er eine Druckmessung durchführen soll. Als Parameter für die Auflösung der Messung, können die Zahlen von 0 bis 3 übergeben werden. 3 entspricht dabei der höchsten Auflösung bei längerer Messzeit. Die Messdauer wird wieder in `status` gespeichert und anschließend abgewartet. Nun kann mit `pressure.getPressure(P,T)` der absolute Druck aus der Messung und der Temperatur `T` berechnet und in der Variablen `P` gespeichert werden. Auch die Berechnung des reduzierten Luftdruckes wird von der Bibliothek mit der Funktion `pressure.sealevel(P,H)` übernommen und in `p0` gespeichert. Nun erfolgt die Ausgabe an die serielle Schnittstelle mit jeweils zwei Nachkommastellen und im Anschluss darauf eine Pause von 5 Sekunden.

3.10 Luftfeuchtigkeitssensor

Benötigte Materialien

- Arduino UNO
- Breadboard
- Jumper Wires
- Luftfeuchtigkeitssensor AM2301

3.10.1 Physikalisches Prinzip

Mit dem in diesem Projekt verwendeten Sensor AM2301 der Firma AOSONG, kann die relative Luftfeuchtigkeit und gleichzeitig - mit einem integrierten NTC-Sensor - die Temperatur gemessen werden. Er funktioniert nach dem Prinzip der kapazitiven Feuchtigkeitsmessung, bei der die Messeinheit aus einem Kondensator besteht, der je nach Feuchtigkeitsgehalt der Luft seine Kapazität ändert. Aufgebaut ist er - wie für Kondensatoren üblich - aus zwei Elektroden und einem dazwischen befindlichen Dielektrikum, das z.B. aus einer hygroskopischen Polymerschicht besteht (siehe Abb. 3.23). Der gasförmige Wasserdampf der Luft wird vom Dielektrikum

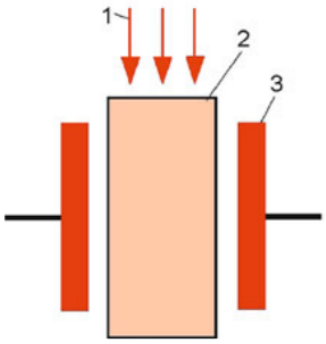

Abb. 3.23: Kapazitver Feuchtigkeitssensor. 1 gasförmiges Wasser, 2 Dielektrikum, 3 Elektroden[18]

absorbiert und verändert so die relative Permittivitätszahl ϵ_r. Da sich für einen Plattenkondensator die Kapazität nach[15]

$$C = \epsilon_0 \epsilon_r \cdot \frac{A}{d} \tag{3.7}$$

15) Kapazität: C, absolute Dielektrizitätskonstante des Vakuums: $\epsilon_0 = 8{,}854 \cdot 10^{-12}\,\mathrm{A\,V^{-1}\,m^{-1}}$, relative Permittivitätszahl des Dielektrikums: ϵ_r, Plattenfläche: A, Abstand der Platten: d.

berechnet, ändert sich mit ϵ_r auch C. In einer Messschaltung wird die Kapazität erfasst und ein dazu proportionales Spannungssignal ausgegeben. Dafür verwendet man oft eine Brückenschaltung, die mit einer hochfrequenten Wechselspannung gespeist wird. Sie besteht aus vier Kondensatoren: Dem Sensor mit einer Kapazität $C_S = C_0$ im Arbeitspunkt, zwei identen Kondensatoren und einem weiteren regelbaren Kondensator zum Abgleich. In Abb. 3.24 ist der schematische Aufbau dargestellt. Der Abgleichkondensator muss nun so eingestellt werden, dass man im Arbeitspunkt (d.h. für $C_S = C_0$) eine Brückenspannung von 0 V misst. Ändert sich durch die Luftfeuchtigkeit der Kapazitätswert des Sensors um ΔC, so ändert sich die Brückenspannung um ΔU. Diese Spannung wird abgegriffen, in Gleichspannung umgewandelt und verstärkt und liegt anschließend als Ausgangsspannung U_a vor. Leider nimmt die Kapazitätsänderung mit steigender relativer Luftfeuchtigkeit zu, sodass die Ausgangsspannung noch linearisiert werden muss, oder man sich gespeicherter Wertetabellen bedient, um den richtigen Feuchtigkeitswert zu erhalten. Bei dem AM2301 erledigt dies eine integrierte Schaltung.[16]

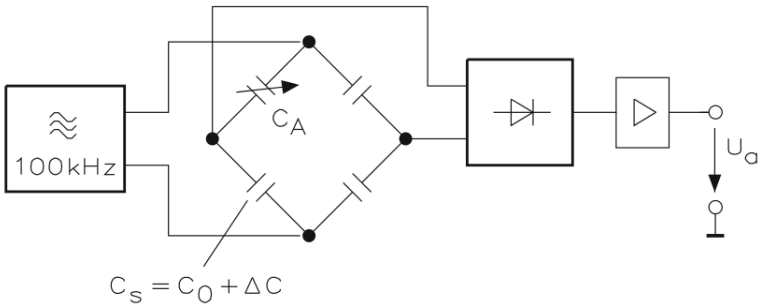

Abb. 3.24: Auswerteschaltung zur Messung der Kapazitätsänderung [3]

3.10.2 Luftfeuchtigkeit

Die trockene atmosphärische Luft ist ein Gemisch aus verschiedenen Gasen und besteht zu 78 % aus Stickstoff, zu 21 % aus Sauerstoff, zu 0,9 % aus Edelgasen und zu 0,4 % aus Kohlendioxid. Aus unserer täglichen Erfahrung mit dem Wetter wissen wir, dass, in sich änderndem Anteil, noch Wasserdampf enthalten ist. Es gibt eine Vielzahl von Möglichkeiten, wie die in der Luft enthaltene Wassermenge angegeben werden kann. Da unser Sensor die relative Luftfeuchtigkeit ausgibt, wollen wir uns auf diese beschränken.

Das Gesetz von Dalton besagt, dass sich der Gesamtdruck p eines Gasgemisches, aus

16) Vgl. [3], [18] und [4]

den Partialdrücken p_i der einzelnen Gase zusammensetzt - mathematisch formuliert:

$$p = \sum p_i \,.$$

Man kann daher den Luftdruck als Summe der Partialdrücke aus der trockenen Luft und des Wasserdampfpartialdrucks e anschreiben. Abhängig von der Temperatur, gibt es einen Maximalwert des Dampfdrucks ab dem die Luft gesättigt ist. Man bezeichnet diesen Wert als Sättigungsdampfdruck e_S. Ausgehend von der Clausis-Clapeyron-Gleichung, kann man folgende Formel für den Zusammenhang zwischen Temperatur und Sättingsdampfdruck herleiten:[17]

$$e_s\,(T) = e_{s,0}\, exp\left[\frac{l}{R_v}\left(\frac{1}{T_0} - \frac{1}{T}\right)\right] \,. \tag{3.8}$$

Als Vereinfachung nimmt man dabei an, dass die spezifische Verdampfungsenthalpie von Wasser konstant ist und vernachlässigt die geringe Temperaturabhängigkeit. In Abb. 3.25 ist die oben beschriebene Formel 3.8 auch grafisch dargestellt, wobei für die Verdampfungsenergie mit einem Wert von $l = 2500\,\mathrm{kJ\,kg^{-1}}$ gerechnet wurde. Ein anschauliches Beispiel für den Sättigungsdampfdruck ist der dampfende Atem:

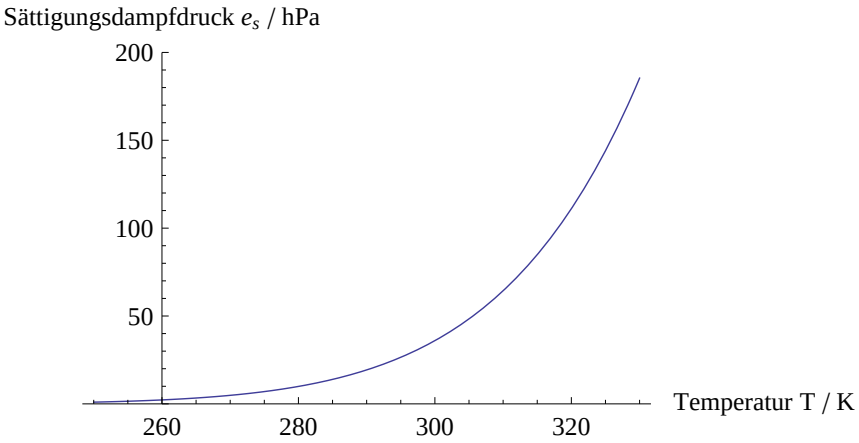

Abb. 3.25: Analytische Lösung der Clausius-Clapeyron-Gleichung

Wird kühle ungesättigte Luft eingeatmet, so erwärmt sie sich in der Lunge. Als Folge der Erwärmung steigt der Sättigungsdampfruck der Luft und sie kann mehr Wasserdampf aufnehmen - was in der feuchten Lunge auch tatsächlich passiert. Insgesamt steigt damit der Wasserdampfpartialdruck. Liegt nun beim Ausatmen der Dampfdruck der Atemluft über dem Sättigungsdampfdruck der Umgebungsluft, so kommt es zur Nebelbildung.

17) Sättigungsdampfdruck bei einer Temperatur $T_0 = 273,15°\mathrm{K}$: $e_{s,0} = 6.1078$ hPa, Spezifische Gaskonstante für Wasserdampf: $R_v = 462$ J/kg K, Lufttemperatur in Kelvin: T, Spezifische Verdampfungsenthalpie von Wasser: l

Um ein Maß für die Luftfeuchtigkeit zu erhalten, bildet man den Quotienten aus aktuellem Dampfdruck e und Sättigungsdampfdruck e_S (bei aktueller Temperatur) und bezeichnet das in Prozent angegebene Ergebnis

$$r_F = \frac{e}{e_S} \cdot 100 \tag{3.9}$$

als relative Luftfeuchtigkeit r_F.[18]

3.10.3 Aufbau

Der Luftfeuchtigkeitssensor besitzt 3 Anschlüsse, wovon zwei für die Spannungsversorgung (VDD und GND) und einer als Datenleitung (SDA) Verwendung finden. Abb. 3.26 zeigt die Verkabelung mit dem Arduino, wobei wir Pin 2 für die bidirektionale Datenleitung verwenden. Das Datenblatt des Sensors [19] gibt Auskunft über wesentliche Eigenschaften. So erstreckt sich der Messbereich für die relative Feuchtigkeit von 0,0 % bis 99,9 %, für die Temperatur von −40,0 °C bis 80,0 °C, bei einer Ungenauigkeit von ±3 % bzw. ±0,5 °C.

Zu erwähnen ist noch, dass der Sensor nur alle 2 s neue Messdaten ausgibt und somit relativ langsam ist - für die meisten Projekte sollte dies jedoch ausreichend sein.

3.10.4 Programmcode

Um die Arbeit mit dem Sensor zu vereinfachen, kann man eine fertige Bibliothek unter `https://github.com/adafruit/DHT-sensor-library/archive/master.zip` herunterladen und in die Programmierumgebung integrieren. Ihr Funktionsumfang ist relativ überschaubar, jedoch lässt sie sich schnell und einfach einsetzen. Unser Sensor ist Teil einer ganzen Reihe anderer Sensoren, die sich nur geringfügig voneinander unterscheiden. Leider sind die Bezeichnungen der einzelnen Typen nicht eindeutig und der AM2301 besitzt auch den Namen DHT22 - welcher in der Bibliothek verwendet wird.

Zunächst wird für den Sensor ein Objekt vom Typ `DHT` erstellt (Zeile 6), das den Namen `dht` trägt. Übergabeargumente sind der Anschlusspin der Datenleitung, sowie das Sensormodell. Das Auslesen der Werte für die relative Feuchtigkeit und der Temperatur, erfolgt in den Zeilen 17 und 18. Im Anschluss daran werden die Werte an die Serielle Schnittstelle gesendet.

18) Dieser Abschnitt basiert auf der von mir verfassten Bachelorarbeit: „Methoden der Wettervorhersage“

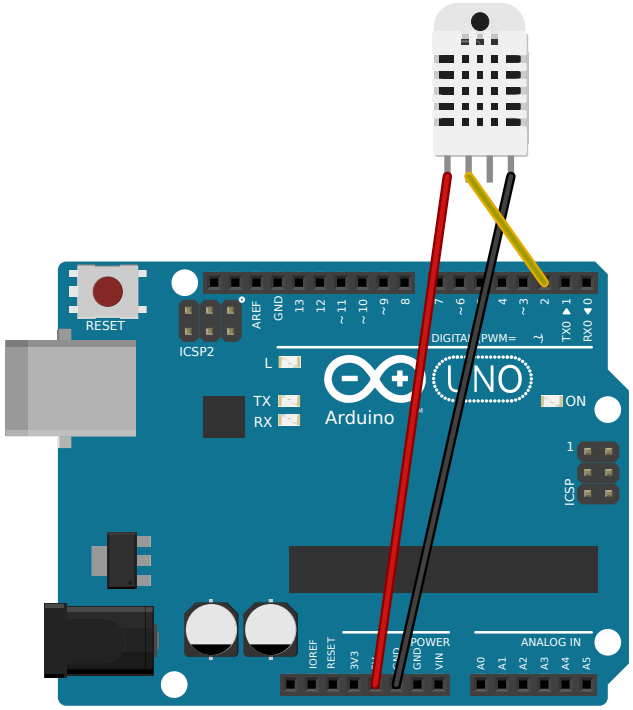

Abb. 3.26: Anschluss des Luftfeuchtigkeitssensors AM2301 an den Arduino

```
#include <DHT.h>

int AM_Pin=2;                  //Pin der Datenleitung
char AM_Typ=DHT22;             //Sensor-Modell ist DHT22=AM2301
float r,T;                     //Variablen für rel. Feuchte und Temp.
DHT dht(AM_Pin, AM_Typ);       //Erstelle Objekt vom Typ DHT mit Namen dht

void setup() {
  Serial.begin(9600);
  Serial.println("Starte Messungen  :");

  dht.begin();                 //Starte Kommunikation mit Sensor
}

void loop() {

  r = dht.readHumidity();      //Auslesen der rel. Feuchte
  T = dht.readTemperature();   //Auslesen Temp.

  Serial.print("relative Feuchte: ");
  Serial.print(r);
  Serial.print("%\t");
  Serial.print("Temperatur: ");
  Serial.print(T);
  Serial.println(" *C");
```

```
26   delay(1000);
27 }
```

Code 3.9: Auslesen des AM2301 Luftfeuchtigkeitssensors

3.11 Gyrosensor

Benötigte Materialien

- Arduino UNO
- Breadboard
- Jumper Wires
- GY-521 Breakout Board mit MPU6050

Gyrosensoren, oder auch Drehratensensoren genannt, messen bei einer Drehung um eine feste Sensorachse, die Winkeländerung pro Zeiteinheit. Mit ihrer Hilfe kann man also Winkelgeschwindigkeiten bestimmen. In diesem Experiment werden wir die Winkelgeschwindigkeiten, die vom Sensor gemessen werden, auslesen, daraus die Winkel berechnen um die der Sensor gedreht wird und die Ergebnisse am seriellen Monitor ausgeben.

3.11.1 Physikalisches Prinzip

Es gibt viele physikalische Effekte, die zur Herstellung von Drehratensensoren verwendet werden können und auch verwendet werden. Von dem historischen Kreisel, bis zur Ausnützung der Relativitätstheorie für beschleunigte Systeme, reicht die Palette. In sehr vielen mikromechanischen Sensoren, die in Smartphones und Alltagsanwendungen eingesetzt werden, nutzt man die in rotierenden Systemen auftretende Coriolisbeschleunigung zur Erfassung der Winkelgeschwindigkeit - so auch in dem in diesem Abschnitt verwendeten MPU6050. Im Prinzip (siehe Abb. 3.27) werden dabei zwei Massen in gegensinnige Schwingung gebracht. Sobald das System mit der Winkelgeschwindigkeit $\vec{\Omega}$ gedreht wird, erfahren beide Massen eine zusätzliche Beschleunigung - die Coriolisbeschleunigung $\vec{a_c}$, die normal auf die ursprüngliche Schwinggeschwindigkeit $\vec{v}$ und $\vec{\Omega}$ steht:

$$\vec{a_c} = 2\vec{v} \times \vec{\Omega}\,. \tag{3.10}$$

Misst man diese auftretende Beschleunigung (mehr dazu in 3.12), so kann man aus dem obigen Zusammenhang die Winkelgeschwindigkeit bestimmen.[19]

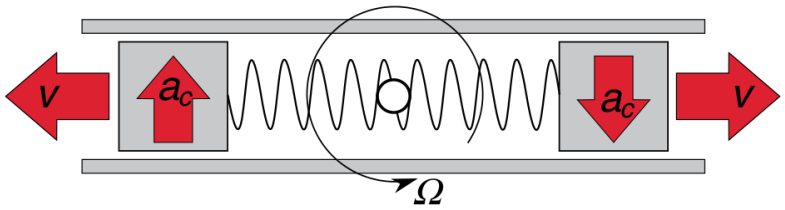

Abb. 3.27: Schematische Darstellung eines Gyroskops [20]

3.11.2 Aufbau

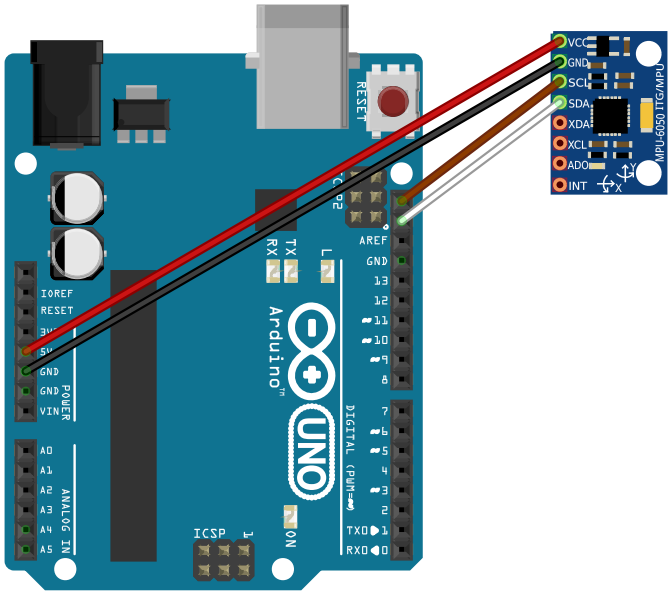

Abb. 3.28: Anschluss des GY-521 Breakout Board mit eingebautem MPU6050 an den Arduino

Alle wichtigen Kondensatoren und Widerstände, die notwendig sind um den MPU6050 mit Hilfe des Arduio auslesen zu können, sind bereits im Breakout Board GY-521 untergebracht. Da der Sensor I^2C unterstützt, ist der Anschluss denkbar einfach und in Abb. 3.28 dargestellt.

19) Vgl. [20]

3.11.3 Programmcode

Der Gyrosensor liefert keine direkten Winkel die man auslesen und verarbeiten könnte. Stattdessen werden die Winkelgeschwindigkeiten um die drei Sensor-Raumachsen gemessen. Aus diesen müssen durch numerische Integration, die tatsächlichen Drehwinkel berechnet werden. Im Folgenden ist ein Sketch angeführt, der diese Aufgabe erledigt. Das Datenblatt [21] liefert dazu die wichtigsten Informationen. So liegen die gemessenen Werte in einem 16 Bit Format vor, wobei ein Bit für das Vorzeichen reserviert ist. Daraus ergibt sich, dass im empfindlichsten Modus der Messbereich $[-250\,°/s, 250\,°/s]$ durch den ganzzahligen Wertebereich $[-2^{15}, 2^{15}-1]$ dargestellt wird. Um die tatsächlichen Werte zu erhalten, muss man noch durch die Empfindlichkeit von 131 dividieren, da im gewählten Modus der Wert 131 einer Geschwindigkeit von $1\,°/s$ entspricht. Auch für diesen Sensor gibt es eine externe Bibliothek, die uns viel Arbeit abnimmt. Sie wurde von Jeff Rowberg entwickelt und kann unter [22] als ZIP-Archiv heruntergeladen werden. Zur Einbindung in die Entwicklungsumgebung, müssen die im Ordner *Arduino* enthaltenen Unterordner *I2Cdev* und *MPU6050*, in den Ordner *sketchbook/libraries* der Entwicklungsumgebung kopiert werden.

Um brauchbare Daten zu erhalten, müssen noch zwei Probleme überwunden werden. Zum einen beinhalten die Messwerte ein Offset, zum anderen streuen die Werte zufällig um diesen Offsetwert. Die Elimination des Offsets erfolgt in der Funktion `offset()` in zwei Schritten: Zunächst werden die arithmetischen Mittel - für die drei Achsen - aus einigen Messwerten berechnet. Diese Werte bilden die Startwerte für eine exponentielle Glättung, welche als Tiefpass fungiert, um die berechneten Offsetwerte weiter zu präzisieren. Das zweite Problem - das Grundrauschen - behandelt die Funktion `minmax()`. Sie analysiert für einige Sekunden die Messwerte und speichert für jede Achse den maximalen und minimalen Wert. Es ergibt sich daraus ein Bereich für das Grundrauschen. Diese beiden Funktionen werden nach den Deklarationen und Initialisationen im Programm ausgeführt. In der `loop`-Schleife erfolgt die Berechnung der Drehwinkel. Hierfür wird die Zeit gemessen, die seit der letzten Messung vergangen ist und in der Variable `dt` gespeichert. Unmittelbar danach erfolgt eine neue Messung, welche nur berücksichtigt wird, wenn sie sich außerhalb des Rauschbereiches befindet. Ist dies der Fall, so berechnet sich die Winkeländerung `dphi` innerhalb des Zeitschritts `dt` wie in Zeile 50 dargestellt. Das Modell das dahinter steht, ist eine numerische Integration durch eine einfache Rechteckapproximation. Die neuen Winkel ergeben sich aus der Summe der alten Winkeln und `dphi`.

Zur Steigerung der Effizienz, wird nur jeder zwanzigste berechnete Winkel, an der

seriellen Schnittstelle ausgegeben.

```
#include <I2Cdev.h>
#include <MPU6050.h>
#include <Wire.h>
MPU6050 mpu;     //Erzeuge ein Objekt vom Typ MPU6050
int16_t w[3];
int8_t mode=0;  //Tiefpass Arbeitsmodus des MPU6050 (von 0=aus bis 6)
    siehe MPU6050.cpp
int t,dt;
int MIN[3],MAX[3];
double dphi[3],phi[]={0,0,0};
double g_off[]={0,0,0},g[3];
int z=0;

void setup() {
  Wire.begin();
  Serial.begin(115200);
  Serial.println("Starte I2C Verbindung...");
  mpu.initialize();
  // Teste die Geräteverbindung
  Serial.println("Teste Geraeteverbindung...");
  if(mpu.testConnection()){
    Serial.println("MPU6050 erfolgreich verbunden!");
  }
  else{
    Serial.println("MPU6050 Verbindung gescheitert!");
  }
  mpu.setDLPFMode(mode);  //Konfiguration für integrierten Tiefpass
  offset();               //Bestimme Offsetwerte des Sensors
  minmax();               //Bestimme Bereich der Rauschunterdrückung

  Serial.println("Winkelmessung aktiv...");
  for(int i=0;i<3;i++){
    Serial.print(0);
    Serial.print("\t");
  }
  Serial.println("\t");
  t=micros();
}

void loop() {
  dt=micros()-t;          //Diskretisierung der Zeit
  t=micros();
  //Auslesen der Winkelgeschwindigkeiten
  mpu.getRotation(&w[0], &w[1], &w[2]);
  //Numerische Integration zur Bestimmung der Winkel
```

```
45   //aus den Winkelgeschwindigkeiten
46   for(int j=0;j<3;j++){
47   //Nur wenn sich ein Messwert außerhalb des Rauschbereiches
48   //befindet, wird er für die Berechnung berücksichtigt.
49     if((w[j]-g_off[j])>MAX[j] || (w[j]-g_off[j])<MIN[j]){
50       dphi[j]=(w[j]-g_off[j])*dt*0.000001/131;
51       phi[j]=phi[j]+dphi[j];
52       z++;  //Erhöhe z wenn ein Messwert die Kriterien erfüllt
53     }
54   }
55   //Ausgabe nach 20 angenommenen Messwerten
56   if(z>=20){
57     for(int i=0;i<3;i++){
58       Serial.print(phi[i]);
59       Serial.print("\t");
60     }
61     Serial.println("\t");
62     z=0;
63   }
64 }
65 /******************Berechnung der Offsetwerte****************/
66 void offset() {
67   mpu.setXGyroOffset(0);
68   mpu.setYGyroOffset(0);
69   mpu.setZGyroOffset(0);
70   int g[3];
71   double sum[]={
72     0,0,0          };
73   int n=100;
74   //Auslesen einiger Werte - zur Stabilisierung des Sensors
75   for(int i=0;i<1000;i++){
76     mpu.getRotation(&w[0], &w[1], &w[2]);
77   }
78   delay(100);
79   //Ermittlung der Startwerte für das exponentiell geglättete Mittel
80   for(int i=0;i<n;i++){
81     mpu.getRotation(&w[0], &w[1], &w[2]);
82     for(int j=0;j<3;j++) sum[j]=sum[j]+w[j];
83   }
84   for(int i=0;i<3;i++) g_off[i]=sum[i]/n;
85   //Berechnung des exponentiell geglätteten Mittels
86   for(int i=0;i<50*n;i++){
87     mpu.getRotation(&w[0], &w[1], &w[2]);
88     for(int j=0;j<3;j++) g_off[j]=0.999*g_off[j]+0.001*float(w[j]);
89     delay(1);
90   }
```

```
91 }
92 /******Berechnung der Werte für die Rauschunterdrückung********/
93 void minmax(){
94   int tt=millis();
95   mpu.getRotation(&w[0], &w[1], &w[2]);
96   for(int i=0;i<3;i++){
97     MIN[i]=w[i]-g_off[i];
98     MAX[i]=w[i]-g_off[i];
99   }
100  //Messe für 5s den Rauschbereich --> MAX, MIN
101  while((millis()-tt)<5000){
102    mpu.getRotation(&w[0], &w[1], &w[2]);
103    for(int i=0;i<3;i++){
104      if(MIN[i]>(w[i]-g_off[i])) MIN[i]=w[i]-g_off[i];
105      if(MAX[i]<(w[i]-g_off[i])) MAX[i]=w[i]-g_off[i];
106    }
107  }
108 }
```

Code 3.10: Berechnung der Lagewinkel aus den Winkelgeschwindigkeitswerten des MPU6050

3.12 Beschleunigungssensor

Benötigte Materialien

- Arduino UNO
- Breadboard
- Jumper Wires
- GY-521 Breakout Board mit MPU6050

3.12.1 Physikalisches Prinzip

Die meisten Beschleunigungssensoren bestehen aus einer Seismischen Masse, die über ein Federelement mit dem Gehäuse verbunden ist. Bewegt man das Gehäuse, so wird durch die Trägheit der Masse, das Federelement gedehnt, bzw. gestaucht. Für kleine Auslenkungen gilt dabei das Hook'sche Gesetz. Es besagt, dass die Kraft F, die eine Dehnung bzw. Stauchung verursacht, über eine Proportionalitätskonstante k

mit der Längenänderung Δx verknüpft ist:

$$F = k \cdot \Delta x \,. \tag{3.11}$$

Da k ein bekannter Materialwert des Federelementes ist, kann man durch das Messen der Längenänderung Δx die Kraft F bestimmen, welche die Seismische Masse beschleunigt. Aus dem Newton'schen Bewegungsgesetz $F = m{\cdot}a$, erhält man schließlich mit dem bekannten Massenwert m der Seismischen Masse, die Beschleunigung a. Insgesamt schließt man also von einer Längenänderung auf die beschleunigende Kraft, und von dieser wiederum auf die wirkende Beschleunigung.

Es gibt nun eine Vielzahl an Möglichkeiten, um die Längenänderung zu messen. In kapazitiven Beschleunigungssensoren, ist die Masse als mittlere Elektrode zweier Plattenkondensatoren C_1, C_2 ausgeführt (siehe Abb. 3.29). Befindet sich das System in Ruhe, so sind beide Plattenabstände identisch und können mit d_0 bezeichnet werden. Lenkt sich die Seismische Masse in Folge einer Beschleunigung aus, so verändern sich die Plattenabstände um den selben Betrag d, sodass gilt[20]

$$C_1 = \frac{\epsilon_0 \epsilon_r A}{d_0 + d}, \quad C_2 = \frac{\epsilon_0 \epsilon_r A}{d_0 - d} \,. \tag{3.12}$$

Durch einfaches Umformen und Gleichsetzen der beiden Gleichungen erhält man $C_1\,(d_0 + d) = C_2\,(d_0 - d)$ und kann damit die Auslenkung d mit

$$d = d_0 \frac{C_2 - C_1}{C_1 + C_2} \tag{3.13}$$

bestimmen.[21]

Der von uns verwendete Beschleunigungssensor MPU6050 der Firma InvenSense hat drei solcher Messeinheiten eingebaut, welche die auftretenden Beschleunigungen entlang der kartesischen Koordinatenachsen des Sensors messen.

3.12.2 Aufbau

Der Aufbau ist ident zu dem in 3.11.

20) Elektrische Feldkonstante: $\epsilon_0 = 8{,}854 \cdot 10^{-12}\,\mathrm{A\,V^{-1}\,m^{-1}}$, Permittivitätszahl: ϵ_r
21) Vgl. [4] und [20]

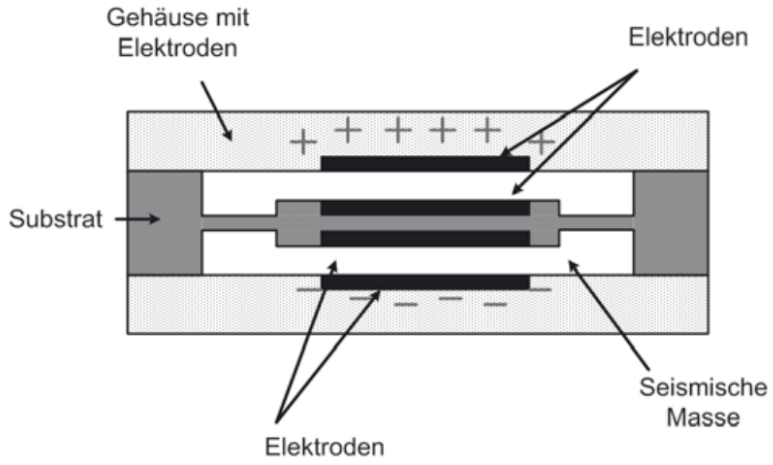

Abb. 3.29: Aufbau eines kapazitiven Beschleunigungssensors [4]

3.12.3 Programmcode

In diesem Abschnitt wollen wir einen Sketch entwerfen, der mit Hilfe der Beschleunigungsmessungen des Sensors, die lokale Schwerebeschleunigung berechnet. Analog zum Gyroskop liefert der Sensor ganzzahlige Werte aus dem Intervall $[-2^{15}, 2^{15}-1]$. Im empfindlichsten Modus liegt der Messbereich bei $[-2\,\mathrm{g}, 2\,\mathrm{g}]$ wobei $g = 9{,}81\,\mathrm{ms}^{-2}$, das bedeutet, dass die gelieferten Sensordaten, nach einer Multiplikation mit dem Faktor $\frac{9{,}81}{16384}$, in SI-Einheiten vorliegen. Da auf den Sensor zu jeder Zeit die Schwerebeschleunigung einwirkt, bestimmt man die Beschleunigungswerte für die drei Sensor-Achsen und erhält daraus einen Beschleunigungsvektor, dessen Betrag - sofern sich der Sensor in Ruhe befindet - dem Betrag der lokalen Schwerebeschleunigung entspricht. Leider streuen die Messwerte jeder Achse, selbst wenn der Sensor keiner Bewegung ausgesetzt ist, um einen Mittelwert. Wir verwenden daher wieder die Vorgangsweise der Funktion `offset()`, die in 3.11.3 eingeführt wurde, um mit Hilfe einer exponentiellen Glättung die gesuchten Mittelwerte zu berechnen. Der Wert der lokalen Schwerebeschleunigung wird anschließend an die serielle Schnittstelle gesendet und danach alle 200 ms eine Messung ausgegeben.

```
1 #include <I2Cdev.h>
2 #include <MPU6050.h>
3 #include <Wire.h>
4 MPU6050 mpu;      //Erzeuge ein Objekt vom Typ MPU6050
5 int16_t a[3];
6 int8_t mode=6;   //Tiefpass Arbeitsmodus des MPU6050 (von 0=aus bis 6)
7                  //siehe MPU6050.cpp
8 double g_off[]={0,0,0};
9 double g;
10
```

```
11 void setup() {
12   Wire.begin();
13   Serial.begin(115200);
14   Serial.println("Starte I2C Verbindung...");
15   mpu.initialize();
16   // Teste die Geräteverbindung
17   Serial.println("Teste Geraeteverbindung...");
18   if(mpu.testConnection()){
19     Serial.println("MPU6050 erfolgreich verbunden!");
20   }
21   else{
22     Serial.println("MPU6050 Verbindung gescheitert!");
23   }
24   mpu.setDLPFMode(mode);  //Konfiguration für integrierten Tiefpass
25
26   Serial.println("bestimme die lokale Schwerebeschleunigung...");
27   //Bestimme Beschleungigungswerte jeder Achse
28   beschl_mittelwerte();
29   //Bilde den Betrag des Beschleunigungsvektors
30   g=sqrt(g_off[0]*g_off[0]+g_off[1]*g_off[1]+g_off[2]*g_off[2]);
31   Serial.print("g= ");
32   Serial.print(g,4);
33   Serial.println(" m/s2");
34 }
35
36 void loop() {
37   mpu.getAcceleration(&a[0], &a[1], &a[2]);
38   for(int i=0;i<3;i++){
39     Serial.print(float(a[i])*9.81/16384,4);
40     Serial.print("\t");
41   }
42   Serial.println("\t");
43   delay(200);
44 }
45 /**********Bestimme Beschleungigungswerte jeder Achse************/
46 void beschl_mittelwerte() {
47   long sum[]={0,0,0};
48   int n=50;
49   //Auslesen einiger Werte - zur Stabilisierung des Sensors
50   for(int i=0;i<100;i++){
51     mpu.getAcceleration(&a[0], &a[1], &a[2]);
52     delay(2);
53   }
54   delay(100);
55   //Ermittlung der Startwerte für das exponentiell geglättete Mittel
56   for(int i=0;i<n;i++){
```

```
57      mpu.getAcceleration(&a[0], &a[1], &a[2]);
58      for(int j=0;j<3;j++) sum[j]=sum[j]+a[j];
59      delay(2);
60    }
61    for(int i=0;i<3;i++) g_off[i]=sum[i]/n;
62    //Berechnung des exponentiell geglätteten Mittels
63    for(int i=0;i<100*n;i++){
64      mpu.getAcceleration(&a[0], &a[1], &a[2]);
65      for(int j=0;j<3;j++) g_off[j]=0.999*g_off[j]+0.001*float(a[j]);
66      delay(2);
67    }
68    //Umrechnung in SI-Einheiten
69    for(int i=0;i<3;i++) g_off[i]=g_off[i]*9.81/16384;
70  }
```

Code 3.11: Bestimmung der lokalen Schwerebeschleunigung

3.13 Magnetometer

Benötigte Materialien

- Arduino UNO
- Breadboard
- Jumper Wires
- GY-273 Breakout Board mit HMC5883L

3.13.1 Physikalisches Prinzip

Auf alle elektrisch leitfähigen Materialien, übt ein vorhandenes Magnetfeld einen geringen Einfluss auf die elektrische Leitfähigkeit aus. Da die Ladungsträger sich durch das Material bewegen, werden sie auf Grund der Lorentzkraft[22]

$$F_L = q\left(\vec{v} \times \vec{B}\right) \tag{3.14}$$

seitlich abgelenkt. Dies führt zu einer Verlängerung des Weges und damit zu einer Erhöhung des Widerstands. Nun gibt es noch weitere magnetfeldabhängige Effekte, die sich auf die Leitfähigkeit von Materialien auswirken. In den mit großer Stückzahl

22) Ladung des Teilchens: q, Geschwindigkeit des Teilchens: v, magnetische Flussdichte: B

produzierten Magnetfeldsensoren, wird meistens der seit 1857 bekannte anisotrope magnetoresistive Effekt (AMR) genützt.
Um den Effekt erklären zu können, muss man wissen, dass Magnetismus eine Eigenschaft ist, die aus dem magnetischen Moment der Elektronen resultiert. Sind die magnetischen Momente statistisch verteilt - das heißt ungeordnet - so kompensieren sie sich gegenseitig. Von außen betrachtet ist ein Stoff dann unmagnetisiert. Man kann den magnetischen Momenten der einzelnen Elektronen jedoch eine Vorzugsrichtung vorgeben, indem man ein äußeres Magnetfeld anlegt. Die einzelnen Momente zeigen dann in die selbe Richtung wie das äußere Magnetfeld. Diesen Vorgang bezeichnet man als Magnetisieren.
Vereinfacht gesprochen, ist der wesentliche Faktor für den anisotropen magnetoresistiven Effekt, dass die Ladungsverteilung der Leitungselektronen im Atomgitter von der Ausrichtung der magnetischen Momente der Elektronen abhängt. Das führt dazu, dass für magnetisierte Stoffe die elektrische Leitfähigkeit nicht in alle Richtungen gleich gut ist, sondern von der relativen Orientierung von Stromfluss und Magnetisierungsrichung abhängt. Sind diese parallel, so ist der elektrische Widerstand maximal, stehen sie normal aufeinander, so ist der Wiederstand minimal. Die Sensoren sind aus einer dünnen, magnetisierten Schicht ferritschen Materials (z.B. Permalloy) gefertigt, für welche die oben besprochenen Zusammenhänge gelten. Wirkt auf diese nun ein äußeres Magnetfeld ein, beispielsweise das Erdmagnetfeld, so überlagert sich dieses mit der internen Magnetisierung der Schicht und führt zu einer neuen Magnetisierungsrichtung, die dem äußeren Feld folgt.

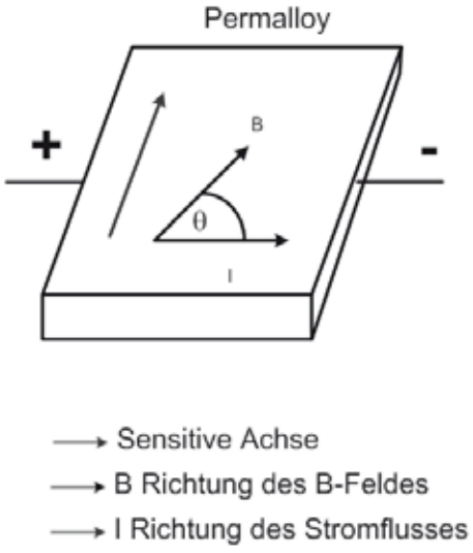

Abb. 3.30: Darstellung des magnetoresistiven Messprinzips [4]

Legt man nun an das Sensorblättchen eine konstante elektrische Spannung an (sie-

he Abb. 3.30), so ändert sich der durch das Blättchen fließende Strom, mit dem vom äußeren Magnetfeld abhängigen Widerstand. Näherungsweise kann man diesen Zusammenhang auch quantitativ mit der Formel

$$R = R_0 + \delta R \cos(2\Theta) \tag{3.15}$$

beschreiben.[23]

Der beschriebene Effekt ist relativ gering und die maximale Widerstandsänderung liegt für AMR-Sensoren bei 3-4 %.[24]

3.13.2 Erdmagnetfeld

Um einen AMR-Sensor als Kompass verwenden zu können, muss man zuerst ein paar Dinge über die Beschaffenheit des Erdmagnetfeldes wissen. Es entsteht durch einen äußerst komplexen Vorgang im äußeren Erdkern. Dieser besteht zum Großteil aus flüssigem Eisen. Durch Temperatur- und Konzentrationsunterschiede entstehen Konvektionsströmungen, die sehr starke elektrische Ströme hervorrufen, welche wiederum Magnetfelder erzeugen. So kompliziert diese Vorgänge im Erdinneren im Detail auch sein mögen, von außen betrachtet führen sie zu einem Erdmagnetfeld, das weitgehend dem eines starken Stabmagneten entspricht, der sich im Erdmittelpunkt befindet. Jede Stelle an der Erdoberfläche wird vom Erdmagnetfeld durchdrungen, wobei im Allgemeinen die Richtung des Magnetfeldes von der lokalen Horizontalen abweicht und auch nicht exakt zum Nordpol zeigt. Mit Deklination bezeichnet man die Winkelabweichung einer frei beweglich gelagerten Kompassnadel von der geografischen Nord-Süd-Richtung, mit Inklination den Winkel der Nadel zur Horizontalen (siehe Abb. 3.31). Diese beiden Größen sind zeitlich nicht konstant und werden zusätzlich von lokalen Umweltbedingungen wie Gesteinsdichte und -art, in der Nähe befindlichen metallischen Gegenständen oder sogar von Sonnenstürmen beeinflusst. Laut [25] liegt für Graz die Deklination bei D = 3,8° und die Inklination bei I = 63,5°.

3.13.3 Aufbau

Für dieses Projekte verwenden wir das GY-273 Breakout Board, das einen HMC5883L Magnetfeldsensor eingebaut hat. Dieser wiederum besitzt drei Messeinheiten, mit wel-

23) Widerstandswert des Sensors: R, Materialabhängige Parameter: R_0 und δR, Winkel zwischen Stromrichtung und Magnetfeld: Θ

24) Vgl. [23], [4], [24] und [20]

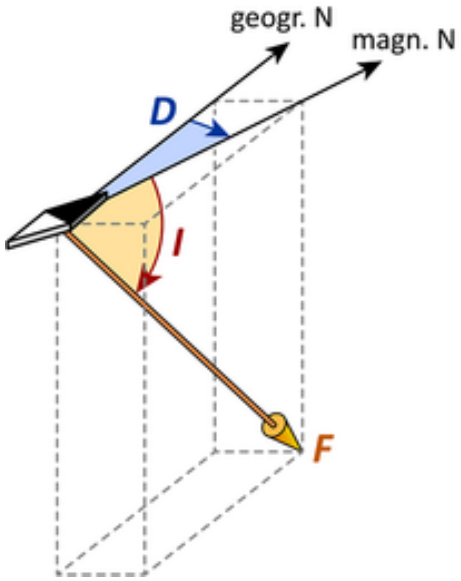

Abb. 3.31: Deklination D und Inklination I zur lokalen Richtung des Erdmagnetfelds F [25]

chen sowohl die Stärke, als auch die räumliche Richtung des Magnetfeldes gemessen werden kann. Aus dem Datenblatt [26] erfährt man die wichtigsten Eigenschaften und die Arbeitsweise des Sensors. So besitzt er einen Messbereich von [−800 µT, 800 µT], bei einer Auflösung von 0,2 µT. Die Pins werden wie in Abb. 3.32 dargestellt mit dem Arduino verbunden. Betrieben wird er mittels I^2C und der 3 V Spannungsversorgung des Arduinos.

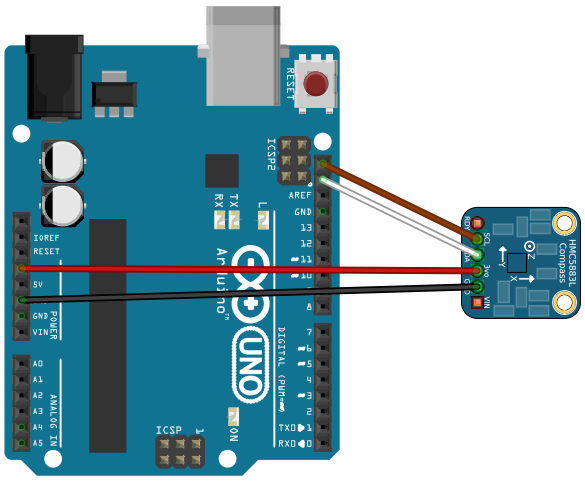

Abb. 3.32: Anschluss des HMC5883L Magnetfeldsensors an den Arduino

3.13.4 Programmcode

Für den Sensor gibt es eine Bibliothek, die man unter https://github.com/adafruit/Adafruit_HMC5883_Unified/archive/master.zip herunterladen und nach 3.8 in das System integrieren kann. Der folgende Sketch zeigt, wie man den Magnetfeldsensor mit Hilfe der externen Bibliothek ausliest und damit den Vektor der magnetischen Flussdichte erhält. Weiters kann mit dem lokalen Wert der Deklination, die Nord-Süd-Richtung berechnet und damit geografisch Nord gefunden werden. Wir haben also mit dem Sensor einen elektronischen Kompass gebaut, der uns die Richtung zum geografischen Nordpol angibt und zusätzlich auch den Betrag des Magnetfeldes.

```
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_HMC5883_U.h>

//Erstelle ein Adafruit_HMC5883_Unified Objekt mit dem Namen Sensor für
    den Sensor mit eindeutiger Identifikationsnummer
Adafruit_HMC5883_Unified Sensor = Adafruit_HMC5883_Unified(1);
//Erstelle ein sensors_event_t Objekt
sensors_event_t ereignis;

float nord;
float D=6.3; //Deklination in Grad fuer Graz
void setup(){
  Serial.begin(9600);
  Serial.println("Starte die Verbindung zum Sensor...");
}

void loop(){

  Sensor.getEvent(&ereignis); //Lese Sensor aus
  ausgabe_nord();
  Serial.print("\t");
  ausgabe_flussdichte();
  Serial.print("\t");
  ausgabe_feldvektor();
  Serial.println("");
  delay(300);
}

//****Auslesen der Feldkomponenten und Ausgabe im seriellen Monitor
    ****//
void ausgabe_feldvektor(){
  Serial.print("Feldvektor: ");
  Serial.print("(X=");
```

```
33   Serial.print(ereignis.magnetic.x);
34   Serial.print("uT, Y=");
35   Serial.print(ereignis.magnetic.y);
36   Serial.print("uT, Z=");
37   Serial.print(ereignis.magnetic.z);
38   Serial.print("uT )");
39 }
40
41 //***Berechnung und Ausgabe des Winkels zur Nord-Richtung***//
42 void ausgabe_nord(){
43   nord=atan2(ereignis.magnetic.y, ereignis.magnetic.x);
44   nord=nord-(PI*D/180); //Beruecksichtigung der Deklination
45   nord=180*nord/PI; //Umrechnen von Rad auf Grad
46   //Korrekturen, damit der Winkel zwischen 0 und 360 Grad bleibt
47   if(nord<0) nord=nord+360;
48   if(nord>=360) nord=nord-360;
49   nord=int(nord+0.5);  //Winkel auf ganze Zahl runden
50   Serial.print("Grad: ");
51   Serial.print(nord,0);
52 }
53 //***Berechnung und Ausgabe der Flussdichte B***//
54 void ausgabe_flussdichte(){
55   Serial.print("Flussdichte=");
56   Serial.print(sqrt(ereignis.magnetic.x*ereignis.magnetic.x+ereignis.
       magnetic.y*ereignis.magnetic.y+ereignis.magnetic.z*ereignis.
       magnetic.z));
57   Serial.print("uT");
58 }
```

Code 3.12: Magnetometer

Nach dem Einbinden der benötigten Bibliotheken, wird ein `Adafruit_HMC5883_Unified`-Objekt erstellt das den Namen `Sensor` trägt und die eindeutige Identifikationsnummer 1 erhält (zur Unterscheidbarkeit falls mehrere Sensoren angeschlossen werden). Danach folgt die Erstellung eines `sensors_event_t`-Objektes mit dem Namen `ereignis`. Damit ist es möglich, mit dem Befehl aus Zeile 19, sämtliche verfügbare Messgrößen des `Sensor`s im Objekt `ereignis` zu speichern. In der `loop`-Funktion werden jeweils am Beginn die Sensorwerte ausgelesen. Die Berechnung und Ausgabe der Nordrichtung, der Flussdichte, sowie des Feldvektors erfolgt in drei separaten Funktionen:

- `ausgabe_feldvektor()` Den Wert der Flussdichte in Richtung der x-Achse des Sensors, der in µT ausgegeben wird, erhält man mit dem Befehl `ereignis.magnetic.x` - für die beiden anderen Achsen lautet der Befehl ana-

log. Damit können wir sehr einfach den Vektor der Flussdichte an die serielle Schnittstelle schicken.

- `ausgabe_nord()` Für die Bestimmung der Nordrichtung sind nur die Komponenten der x- und y-Richtung von Relevanz (vorausgesetzt der Sensor wird waagrecht gehalten). Mit Hilfe der `atan2`-Funktion kann man leicht den Winkel berechnen, den der in die x-y-Ebene projizierte Flussdichtevektor mit der x-Achse einschließt (siehe Zeile 43). Nach der Korrektur der Deklination und der Einführung von periodischen Randbedingungen - um den Winkel innerhalb von [0°,360°) zu halten - wird der Winkel noch auf ganze Gradzahlen gerundet und ausgegeben. Der so berechnete Winkel gibt an, um wie viel Grad die x-Achse des Sensors von der geografischen Nordrichtung abweicht.
- `ausgabe_flussdichte()` In der letzten Funktion wird schließlich der Betrag der Flussdichte berechnet und in µT ausgegeben.

3.14 Schrittmotor

Benötigte Materialien

- Arduino UNO
- Breadboard
- Jumper Wires
- Schrittmotor 28BYJ-48 mit Treiberplatine

Schrittmotoren sind Motoren, die auf Grund ihrer Bauweise, Drehungen um exakt definierte Winkelwerte erlauben. Das macht sie für eine Vielzahl von Anwendungen interessant, wo Positionierungsaufgaben gefragt sind. Roboter- und Automatisierungstechnik, und Alltagsprodukte wie Drucker oder DVD-Laufwerke sind nur einige Einsatzgebiete die hier genannt werden sollen.

3.14.1 Physikalisches Prinzip

Schrittmotoren bestehen aus einem starren, außen liegendem Stator, der mit mehreren Spulen bestückt ist und einem darin befindlichen, drehbaren Rotor. Je nach Bauart kann man mehrere Typen unterscheiden. Wir verwenden einen permanentmagneterregten, unipolaren, vierphasigen Schrittmotor. Dabei ist der Rotor als mehr-

poliger Permanentmagnet ausgeführt, und der Stator besitzt zwei Spulen, die jeweils in der Mitte einen nach außen geführten Anschluss besitzen. Fließt nun durch eine Spule Strom, so entsteht ein Magnetfeld und der Rotor richtet sich nach diesem aus. Durch geschicktes Schalten der Spulen, kann man auf diese Weise eine quasi-kontinuierliche Drehung erreichen. In Abb. 3.33 und Tab. 3.2 ist das Funktionsprinzip dargestellt. Es werden dabei immer zwei Spulen gleichzeitig bestromt. Dies führt dazu, dass sich an den Polschuhpaaren zwei magnetische Dipole bilden. Der Rotor stellt sich so ein, dass sein Nordpol zwischen den Südpolen des Stators und sein Südpol zwischen den Nordpolen des Stators liegt. Im ersten Schritt sind die Spulen A1 und B1 aktiv. Im Schritt zwei wird die Spule A2 aktiviert, sodass sich die Polarität in einem Polschuh ändert und sich der Rotor um 90° gegen den Uhrzeigersinn dreht. In den Schritten drei und vier werden die Spulen nach Tab. 3.2 geschaltet, was die vollständige Drehung des Rotors vollendet. In diesem einfachen Beispiel kann also eine definierte Drehung um jeweils 90° - beliebig oft - erfolgen. Um auch andere Winkelauflösungen zu erzielen, kann man die Anzahl der Phasen m und der Polpaare p im Rotor ändern. Allgemein ist der Schrittwinkel α durch

$$\alpha = \frac{360}{2 \cdot p \cdot m} \tag{3.16}$$

gegeben.[25]

Tab. 3.2: Schrittfolge beim Schrittmotor

Schritt	**Stromdurchflossene Spulen**	
1	A1	B1
2	A2	B1
3	A2	B2
4	A1	B2

Für den von uns verwendeten Schrittmotor ist $\alpha = 11{,}25°$. Zusätzlich besitzt er noch ein Getriebe mit einem Übersetzungsverhältnis von 1/64. Insgesamt benötigt unser Schrittmotor daher 2048 Schritte, für eine vollständige Umdrehung der nach außen führenden Welle.

3.14.2 Aufbau

Der Aufbau ist in Abb. 3.33 dargestellt. Der Motor wird über die Steckverbindung mit dem Treiberboard verbunden und dessen Kanäle `IN1-IN4` mit den digitalen

25) Vgl. [27] und [28]

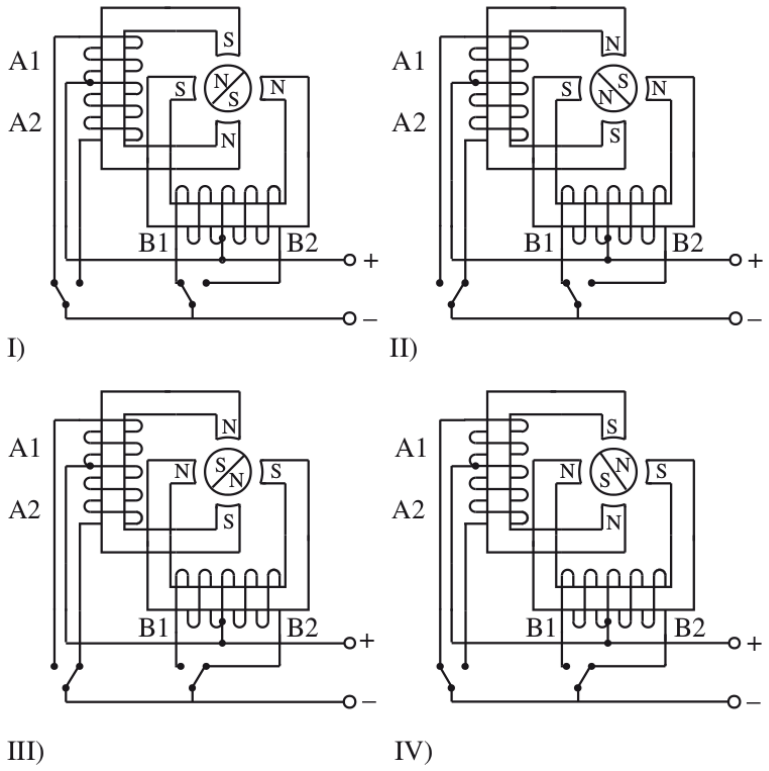

Abb. 3.33: Schrittfolge beim Schrittmotor [27]

Ausgängen `8-11` des Arduinos. Der Motor ist für eine Spannung von 5 V ausgelegt, welche direkt vom Arduino bereitgestellt werden kann.

3.14.3 Programmcode

Um die Funktion des Schrittmotors spielerisch zu überprüfen, wollen wir ein Programm schreiben, das die Winkeleinstellung der Drehachse über Tastatureingaben ermöglicht. Im seriellen Monitor gibt man dabei den gewünschten Drehwinkel ein und bestätigt mit Enter. Daraufhin soll vom Schrittmotor exakt diese Drehung ausgeführt werden.

In der Programmierumgebung existiert bereits die Bibliothek `Stepper.h`, welche die Steuerung für die einzelnen Schritte aus Tab. 3.2 übernimmt und somit die Bedienung sehr erleichtert. Nach der Einbindung der `Stepper`-Bibliothek erfolgt die Zuweisung der digitalen Ausgänge zu den vier Eingängen am Treiberboard (Zeile 2). In Zeile 6 wird ein Objekt für unseren Schrittmotor erzeugt, das den Namen `SM` trägt. Die `setup`-Schleife enthält die Deklaration der Drehgeschwindigkeit `SM.setSpeed(5)` mit 5 Umdrehungen pro Minute. Dies ist bereits der Höchstwert - größere Werte führen zu Fehlschritten.

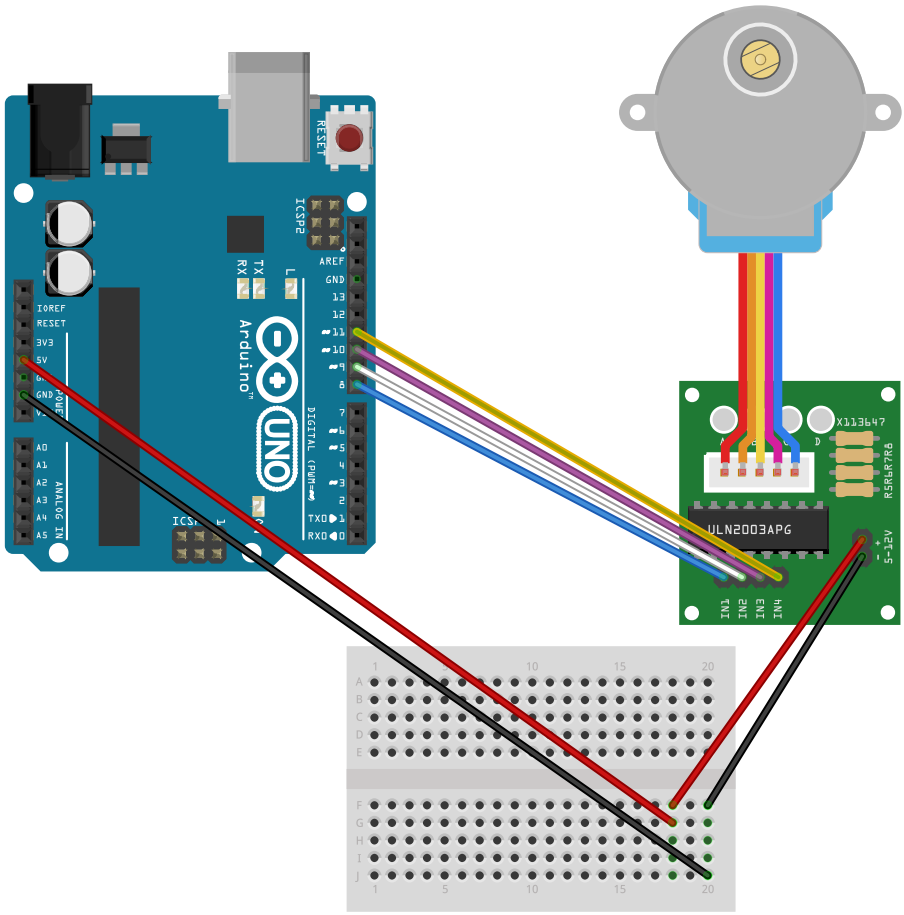

Abb. 3.34: Anschluss des 28BYJ-48 Schrittmotors an den Arduino

In der `loop`-Schleife wird in Zeile 14 abgefragt ob Daten über die serielle Schnittstelle gesendet wurden. Ist dies der Fall, so ist der Rückgabewert für `Serial.available()` glößer als Null und die `while`-Schleife wird betreten. Hier muss zunächst die Zeichenkette vom seriellen Port in eine Zahl umgewandelt werden, um weitere Berechnungen mit ihr durchführen zu können. Die Funktion `parseInt()` ermöglicht genau das. Sie berücksichtigt ausschließlich die Zahlenwerte (und das Vorzeichen) des Datenstroms, welche in der Variablen `alpha` gespeichert werden. In Zeile 18 wird danach die Anzahl der Motorschritte berechnet, die notwendig sind um die Achse um `alpha` zu drehen. Mit `SM.step` wird danach die Drehung tatsächlich ausgeführt.

Als Tastatureingabe sind alle ganzen Zahlen erlaubt, das heißt, dass auch negative Werte zulässig sind. Diese bewirken eine Drehung gegen den Uhrzeigersinn.

```
#include <Stepper.h>
int IN1=8, IN2=10, IN3=9, IN4=11;
int spu=2048;       //Schritte für eine volle Umdrehung
int Schritte;
int alpha;
//Erstelle ein Objekt vom Typ Stepper mit Namen SM
Stepper SM(spu,IN1,IN2,IN3,IN4);

void setup(){
  SM.setSpeed(5);  //Geschwindigkeit; maximal 5 Runden pro Minute
   Serial.begin(9600);
}

void loop(){
  while (Serial.available() > 0){   //liegen Daten an der seriellen
                                    //Schnittstelle?
    alpha=Serial.parseInt();        //lese und speichere die Daten
                                    //als integer
    Serial.print(alpha);
    Serial.print(", ");
    Schritte=int(float(alpha)*2048/360); //Schritte die für Drehung um
                                         //alpha benötigt werden
    SM.step(Schritte);                   //Führe die Drehung aus
    Serial.println(Schritte);
  }
}
```

Code 3.13: Programm zum Betrieb des Schrittmotors

3.15 Ultraschallsensor

Benötigte Materialien

- Arduino UNO
- Breadboard
- Jumper Wires
- Ultraschallsensor HC-SR04

Ultraschallsensoren finden in der Prozessautomatisation, der Fahrzeugtechnik und vielen anderen Bereichen eine breite Anwendung. Meist werden sie dazu eingesetzt um Füllstände, Abstände, Schichtdicken oder Strömungsgeschwindigkeiten zu messen. Sie arbeiten dabei berührungslos und sind sowohl für feste Materialien, als auch flüssige oder pulverfömige Stoffe geeignet.

3.15.1 Physikalisches Prinzip

Als Ultraschall bezeichnet man Schallwellen, die in einem Frequenzbereich von 20 kHz bis zirka 10^7 kHz liegen. Für den Menschen sind diese Frequenzen nicht wahrnehmbar - für einige Tiere allerdings schon. Delfine und Fledermäuse verwenden beispielsweise Ultraschall zur Orientierung im Raum oder zur Jagd. Auch wir wollen dieses Prinzip verwenden und mit Hilfe eines Ultraschallsensors zunächst Abstände zu Objekten zu messen. Der verwendete Ultraschallsensor HC-SR04 besteht aus einem Sendemodul, einem Empfangsmodul und einer integrierten Steuer- und Auswerteelektronik. Aus dem Datenblatt [29] erhält man die wichtigsten Informationen für den Betrieb.

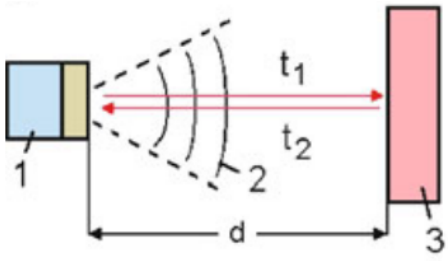

Abb. 3.35: Abstandsmessung mit einem Ultraschallsensor. 1 Sensor, 2 Sendeimpuls, 3 Objekt, d Distanz Sensor-Objekt, t_1 und t_2 Laufzeiten [18]

Das Messprinzip ist relativ simpel (siehe Abb. 3.35): Am Beginn jeder Messung werden einige Ultraschallimpulse mit einer Frequenz von 40 kHz vom Sendemodul abgestrahlt. Gleichzeitig wird ein Timer aktiviert, der die verstrichene Zeit misst. Die

Schallwellen breiten sich nun innerhalb des Abstrahlkegels, der einen Öffnungswinkel von 15° besitzt, mit der Schallgeschwindigkeit aus. Treffen sie auf einen Gegenstand, so werden sie reflektiert und gelangen zurück zum Sensor, wo sie vom Empfangsmodul detektiert werden. Dies ist der Moment in dem der Timer gestoppt wird. Die Zeit, die zwischen dem Senden und Empfangen liegt, entspricht der Schalllaufzeit zum Objekt und wieder zurück. Um die Entfernung zum Objekt zu bestimmen muss man nun die Schalllaufzeit halbieren und mit der Schallgeschwindigkeit multiplizieren. Beachten sollte man hierbei, dass die Schallgeschwindigkeit von einigen Einflussfaktoren, wie Temperatur, Luftdruck, Zusammensetzung der Luft oder Luftfeuchtigkeit abhängt. Wir wollen uns an dieser Stelle nur um die Lufttemperatur kümmern, deren Einfluss auf die Schallgeschwindigkeit c, mit folgender Formel[26] angegeben werden kann:[27]

$$c\,(T) = c_0 \cdot \sqrt{1 + \frac{T}{273}}\,. \tag{3.17}$$

Für eine Umgebungstemperatur von 20 °C beträgt $c \approx 343{,}5\,\mathrm{m/s}$.

3.15.2 Aufbau

Der Sensor besitzt vier Pins: Vcc, GND, trig und echo. Zwei davon für die Spannungsversorgung (Vcc und GND). Möchte man eine Messung durchführen, so muss man den trig-Pin für mindestens 10 µs auf `HIGH` legen. Sobald man das Signal auf `LOW` setzt, wird ein Ultraschallimpuls gesendet und die Zeit t gestoppt, bis die erste Reflexion wieder auf den Sensor trifft. Diese Zeit wird am echo-Pin als PWM-Signal (siehe 3.3) ausgegeben, dessen `HIGH`-Dauer genau der gemessenen Zeitdauer t entspricht. Da der Sensor für einen Messbereich von $[2\,\mathrm{cm}, 400\,\mathrm{cm}]$ - bei einer Auflösung von 3 mm - konzipiert ist, sind die Werte für t entsprechend klein und liegen im Bereich von einigen µs bis hin zu ms. Um ein Gefühl für den Zusammenhang zwischen Entfernung und Zeitdauer zu bekommen, soll nur kurz angemerkt werden, dass eine Änderung der Objektentfernung um 1 cm, eine Laufzeitänderung von zirka 58 µs bewirkt. Die Verkabelung erfolgt wie in Abb. 3.36 dargestellt: Vcc und GND werden mit den entsprechenden Steckplätzen des Arduino verbunden und echo sowie trig auf die digitalen Kanäle 8 bzw. 9 gelegt.

26) Lufttemperatur in Celsius: T, Schallgeschwindigkeit bei 0 °C: $c_0 = 331{,}6\,\mathrm{m/s}$
27) Vgl. [18] und [7]

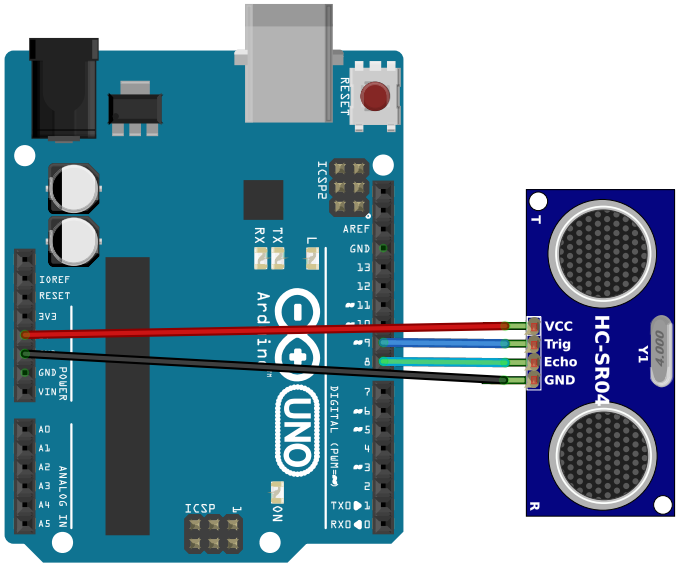

Abb. 3.36: Anschluss des HC-SR04 Ultraschallsensors an den Arduino

3.15.3 Programmcode

Wie bereits besprochen, wird am Echo-Pin ein Rechtecksignal ausgegeben. Die Zeitspanne zwischen dem Senden und Empfangen des Ultraschallsignals, definiert dabei die Dauer, für die der Echo-Pin des Sensors auf `HIGH` gesetzt wird. Es gibt in der Arduino-IDE die Funktion `pulseIn`, die genau diese Aufgabe erfüllt. Sie misst die Zeitdauer in Mikrosekunden, für die ein ausgewählter Digitaleingang den Zustand `HIGH` besitzt. Um die sehr kurzen Schall-Laufzeiten exakt bestimmen zu können, ist ihre Genauigkeit leider nicht zufriedenstellend, sodass wir auf eine neuere Version (`NewPing`) zurückgreifen müssen. Diese arbeitet viel genauer, besitzt weitere Verarbeitungsmöglichkeiten und kann als externe Bibliothek unter https://bitbucket.org/teckel12/arduino-new-ping/downloads heruntergeladen und in die Programmierumgebung integriert werden. Der Sketch zum Auslesen der Zeitdauer ist relativ kurz und wird im Folgenden dargestellt.

```
#include <NewPing.h>
const int echo=8;       //Pin-Nummer des Echo-Signals
const int trig=9;       //Pin-Nummer des Trigger-Signals
int t;                  //Variable der Laufzeit
float s, c=343.6;       //Variablen der Entfernung und der
    Schallgeschwindigkeit
NewPing HC_SR(trig, echo, 500);  //Erstelle ein NewPing Objekt mit
    Namen HC_SR
```

```
7
8  void setup() {
9    Serial.begin(9600);
10 }
11
12 void loop(){
13   t=HC_SR.ping_median(5);   //Median aus 5 Messungen
14   s=c*t*0.0001/2;           //Berechnung der Entfernung in cm aus der
       Laufzeit
15   Serial.print("t=");
16   Serial.print(t);
17   Serial.print("Mirkosekunden,  Entfernung=");
18   Serial.print(s);
19   Serial.println("cm");
20   delay(500);
21 }
```

Code 3.14: Ultraschallsensor

Den Kern des Programms bilden die Zeilen 6 und 13. Zuerst wird mit dem Befehl `NewPing HC_SR(trig, echo, 500)` ein Objekt erstellt, das den Namen `HC_SR` trägt. Mit den ersten beiden Argumenten wird die Trigger- und Echo-Pinbelegung übermittelt. Als drittes Argument übergibt man die Entfernung in Zentimeter, bis zu der gemessen werden soll - dies ist wichtig um die Wartezeit zu begrenzen, falls das Signal nicht reflektiert wird. In Zeile 13 der `loop`-Funktion erfolgt die Messung der Laufzeit mit dem Befehl `HC_SR.ping_median(5)`. Das Triggersignal wird automatisch an den trig-Pin geschickt und daraufhin gemessen wie lange der echo-Pin auf `HIGH` liegt. Dieser Vorgang wird so oft wiederholt, wie man es mit dem Funktionsargument vorgibt und anschließend der Median der Messungen zurückgegeben. Dadurch können Fehlmessungen und Ausreißer minimiert werden. In der nächsten Zeile erfolgt die Berechnung der Entfernung in Zentimeter, aus Laufzeit und Schallgeschwindigkeit.

3.16 LC-Display

Benötigte Materialien

- Arduino UNO
- Jumper Wires
- 16×2 LC-Display mit I^2C Modul

3.16.1 Physikalisches Prinzip

Die uns bekannte Materie besitzt drei Zustände in denen sie vorliegen kann: Fest, flüssig oder gasförmig. Die Ursache, dass ein und der selbe Stoff, in verschiedenen Formen existieren kann, liegt in seiner mikroskopischen Struktur. Während Gase völlig ungeordnet sind, das heißt, dass man die einzelnen Atome bzw. Moleküle als unabhängig voneinander betrachten kann, gibt es bei Flüssigkeiten eine Wechselwirkung mit den benachbarten Teilchen, sodass sich eine Nahordnung ausbildet. Für kristalline Festkörper gibt es zusätzlich noch eine Fernordnung, bei der sich die Bauteile des Kristalls in einem Gitter anordnen. Für jedes Teilchen „sieht“ seine Umgebung exakt gleich aus - egal an welchem Gitterpunkt es sich befindet. Es gibt nun bestimmte Stoffe, die selbst in der flüssigen Phase noch eine räumliche Ordnung ausbilden. Auf Grund dieser Eigenschaft nennt man sie Flüssigkristalle. Man unterscheidet dabei fadenförmige (nematische Flüssigkristalle), schraubenlinienförmige (cholesterinische Flüssigkristalle) und schichtweise (smektische Flüssigkristalle) Anordnungen (siehe Abb. 3.37). Allen gemein ist, dass sich ihre Ordnung sehr stark

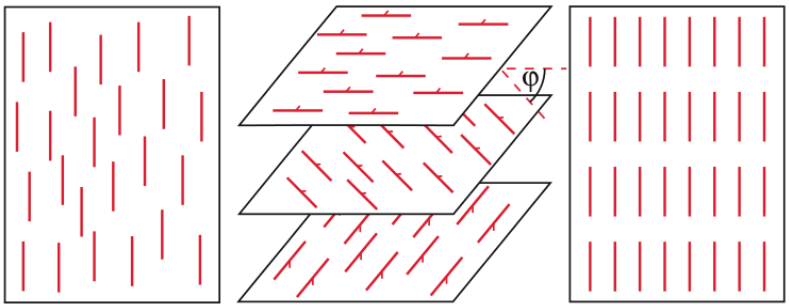

Abb. 3.37: Einteilung der Flüssigkristalle. Von links nach rechts: nematisch, cholesterinisch, smektisch [30]

durch Temperatur, Druck, oder elektrische Felder verändern lässt. Diese Eigenschaft macht man sich für die Herstellung von LC-Displays (kurz für **L**iquid **C**ristal Display) zunutze, in denen hauptsächlich nematischen Flüssigkristalle verwendet wer-

den. Sie bestehen aus länglichen Molekülen mit einem schwachen Dipolmoment und richten sich daher im zeitlichen Mittel parallel zueinander aus.

Ein LC-Display besteht aus kleinen Drehzellen, die sich zwischen zwei Glasplatten befinden, auf deren Innenseite wiederum transparente Elektroden angebracht sind. Auf den Elektroden sind zwei extrem dünne (einige Nanometer) Polyimidfolien, in die feine, parallele Linien eingeritzt sind. Die beiden Folien werden so auf den Elektroden platziert, dass deren Linien zueinander um 90° gedreht sind. Der übrige Innenraum, wird von dem Flüssigkristall eingenommen. Auf jede Außenseite der Glasplatten werden noch lineare Polarisationsfilter angebracht, deren Polarisationsrichtung mit der Ritzrichtung der jeweiligen Polyimidfolie übereinstimmt - das heißt ebenfalls um 90° zueinander gedreht ist (siehe Abb. 3.38). Liegt keine Spannung an den Elektroden an, so übernehmen die Flüssigkristallmoleküle die vorgegebene Ritzrichtung und ordnen sich schraubenförmig im Raum an. Das linear polarisierte Licht, das durch den ersten Polarisator tritt, dreht dabei durch den Flüssigkristall seine Polarisationsebene um 90° und kann den zweiten Filter passieren: Die Zelle erscheint transparent. Bei Anlegen einer Spannung, richten sich die Moleküle - da sie ein Dipolmoment besitzen - im elektrischen Feld aus. Da die Polarisationsebene des Lichtes jetzt nicht mehr vom Flüssigkristall gedreht wird, kann es nicht mehr durch den zweiten Polarisationsfilter gelangen: Die Zelle erscheint schwarz.[28]

Um ein LC-Display zu erhalten, werden nun viele solcher Zellen aneinander gefügt und deren Transparenz, je nach darzustellender Bildinformation, eingestellt. Für farbige Bilder unterteilt man jeden Bildpunkt in drei getrennte Subpixel, bestehend aus einzelnen Drehzellen mit Flüssigkristallen in den Farben Rot, Grün und Blau. Durch additive Farbmischung, entsteht für unser Auge, je nach Zusammensetzung der RGB-Werte der Subpixel, ein kontinuierlicher Farbeindruck.

3.16.2 Aufbau

Da das LC-Display über I^2C kommuniziert, benötigen wir für die Verkabelung lediglich die beiden Leitungen für die Spannungsversorgung und weitere Leitungen für `SDA` und `SCL`, die in Abb. 3.39 weiß bzw. braun gefärbt sind.

28) Vgl. [31] und [30]

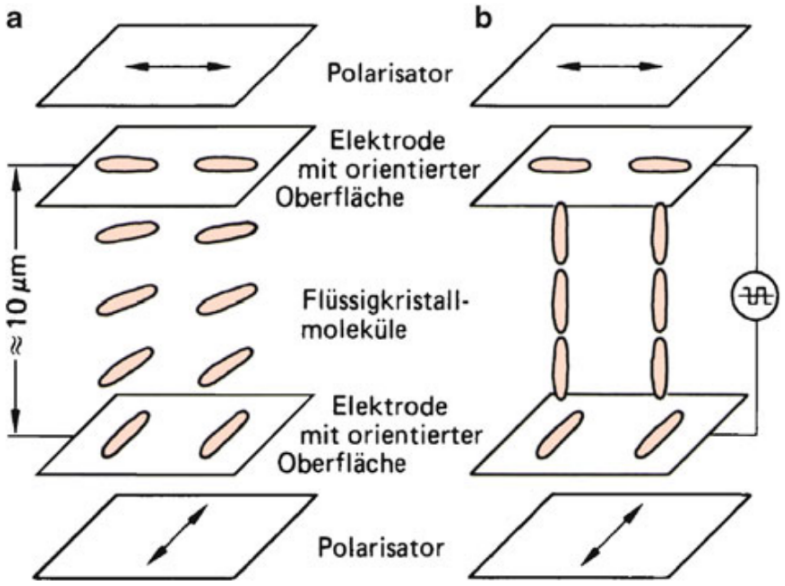

Abb. 3.38: Aufbau einer LC-Drehzelle. a) ohne Spannung, b) mit Spannung [31]

3.16.3 Programmcode

Wir verwenden ein zweizeiliges LC-Display mit 16 Zeichen pro Zeile und einem integriertem I²C Modul, das die Kommunikation wesentlich vereinfacht. Die dafür notwendige Bibliothek kann unter `https://bitbucket.org/fmalpartida/new-liquidcrystal/downloads/NewliquidCrystal_1.3.4.zip` heruntergeladen und ins System integriert werden.

Wir wollen nun ein kurzes Programm schreiben, das einen Text, welcher über die serielle Schnittstelle eingegebenen wird, auf dem LC-Display anzeigt. Da unser LC-Display über I²C kommuniziert müssen wir die `wire.h` Bibliothek einbinden, sowie die zuvor integriete Bibliothek `LiquidCrystal_I2C.h`. In Zeile 6 erfolgt die Initialisierung eines LiquidCrystal_I2C Objektes, das treffender Weise den Namen `LCD` bekommt und im weiteren Code unser Display repräsentiert. Als Übergabeparameter müssen die I²C-Adresse sowie die Nummer der benötigten Pins auf der Platine angegeben werden und zum Schluss die Polarität des backlight-Pins, der für die Hintergrundbeleuchtung zuständig ist:
LiquidCrystal_I2C LCD(Adresse, Enable-Pin, Read/Write-Pin, RegisterSelect-Pin, Datenpin d4, Datenpin d5, Datenpin d6, Datenpin d7, backlight-Pin, Polarität des backlight-Pins).

In der `setup`-Funktion wird zunächst die serielle Verbindung, und danach die Verbindung zum LC-Display gestartet (Zeilen 9-10). Dabei wird gleich das Format des Displays mit 16×2 angeben, was bedeutet, dass die Displaygröße 16 Zeichen pro Zeile und 2 Zeilen beträgt. Die darauf folgende Funktion `setCursor` (Zeile 11) setzt

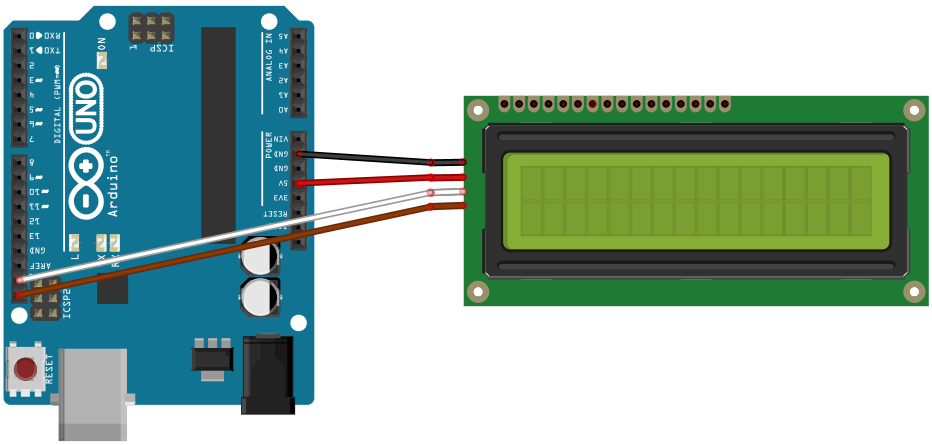

Abb. 3.39: Anschluss des I²C fähigen LC-Displays an den Arduino

den Cursor an die angegebene Position - als Parameter dienen Zeichen- und Zeilennummer.

Am Beginn der `loop`-Funktion wird geprüft - wie schon im Programm von 3.14 - ob Daten an der seriellen Schnittstelle anliegen. Ist dies der Fall, so wird die aktuelle Anzeige des LC-Displays gelöscht (Zeile 20) und mit den neu empfangenen Zeichen überschrieben (Zeile 24). Dafür werden die Funktionen `clear` und `write` der `LiquidCrystal_I2C`-Bibliothek verwendet.

```
1  #include <Wire.h>
2  #include <LiquidCrystal_I2C.h>
3
4  //Initialisiere ein LC-Display mit Namen LCD
5  //Übergabeparameter sind die I2C-Adresse sowie Nummer des enable-PIN,
       des read/write-PIN, des RegisterSelct-PIN, der vier Daten-PINs d4-
       d7, des backlight-Pin und die Polarität von backlight.
6  LiquidCrystal_I2C LCD(0x27, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);
7
8  void setup(){
9    Serial.begin(115200);
10   LCD.begin(16,2);                //Starte Kommunikation 16x2 LCD
11   LCD.setCursor(0,0);             //Setze Cursor auf 1. Zeile
12   LCD.write("Text eingeben:");
13   LCD.setCursor(0,1);             //Setze Cursor auf 2. Zeile
14   LCD.write("max. 16 Zeichen!");
15 }
16
17 void loop(){
18   //Wenn Zeichen an die serielle Schnittstelle gesendet werden, lösche
```

```
    das LC-Display
19  if (Serial.available() > 0){
20    LCD.clear();
21  }
22  //Schreibe die neuen Zeichen auf das LC-Display
23  while (Serial.available() > 0){
24    LCD.write(Serial.read());
25  }
26 }
```

Code 3.15: Programm um Text über die serielle Schnittstelle an das LC-Display zu senden

4 Erweiterte Versuche und Projekte

Im ersten Teil dieser Arbeit haben wir verschiedene Sensoren und Aktoren kennengelernt, gesehen welche physikalischen Prinzipien ihre Funktion ermöglichen und wie man sie mit dem Arduino einsetzen kann. In diesem Kapitel möchten wir nun weiter gehen und die gewonnenen Erkenntnisse verbinden und für umfangreichere Projekte einsetzen.

4.1 Spannungsmessgerät

Benötigte Materialien

- Arduino UNO
- LC-Display mit I^2C Modul
- Relaismodul
- Widerstände: 4,64 kΩ, 4,8 kΩ, 12,4 kΩ, 37,4 kΩ (oder ähnliche)
- Breadboard
- Jumper Wires

Im folgenden Projekt werden wir ein Spannungsmessgerät entwickeln, dessen Kernstück ein Spannungsteiler ist. Wie schon in 3.1 verwenden wir dabei einen Messwiderstand R_m, dessen Spannungsabfall U_m wir am analogen Eingang `A0` messen und wählen je nach gewünschtem Messbereich einen geeigneten Vorwiderstand R_V. Dadurch ist es möglich den Wertebereich von 0 V - 5 V, die der Arduino an den analogen Eingängen bewältigen kann, zu erweitern. Ein großer Messbereich ist zwar wünschenswert für ein möglichst universelles Spannungsmessgerät, wirkt sich jedoch negativ auf die Messauflösung aus, da diese durch den Analog-Digitalwandler des Arduinos mit 10-Bit festgelegt ist.

Um nun einen großen Spannungsbereich bei vernünftiger Auflösung messen zu können, verwenden wir mehrere Vorwiderstände R_i mit verschiedenen Werten. Über ein Relaismodul werden diese vom Arduino in die Schaltung integriert, je nachdem welcher Messbereich gerade optimal - also möglichst klein - ist. Der Schaltplan dazu ist mit den folgenden Bezeichnungen in Abb. 4.1 dargestellt.

R_m	Messwiderstand
U_e	Eingangsspannung
$IN_1 - IN_4$	Steuerkanäle für die 4 Relais des Relaismoduls
$R_1 - R_3$	Vorwiderstände

Aus der Maschenregel erhalten wir den Spannungsabfall am Vorwiderstand

$$U_{R_i} = U_e - U_m \tag{4.1}$$

und für den Strom gilt

$$I = \frac{U_{R_i}}{R_i} = \frac{U_m}{R_m} = \frac{U_e}{R_m + R_i}\,. \tag{4.2}$$

Insgesamt folgt daraus die zu bestimmende Eingangsspannung

$$U_e = U_m \cdot \left(\frac{R_i}{R_m} + 1\right). \tag{4.3}$$

Die Spannung am analogen Eingang des Arduinos darf maximal 5 V betragen um den Mikrocontroller nicht zu beschädigen. Daraus ergeben sich zu den Vorwiderständen gehörende maximale Eingangsspannungen U_{max}, die wir für $U_m = 5\,\mathrm{V}$ aus 4.3 erhalten (siehe Tab. 4.1).

Tab. 4.1: Maximale Eingangsspannung U_{max} der einzelnen Vorwiderstände R_i

i	R_i / kΩ	U_{max} / V
1	37,4	45,3
2	12,4	18,4
3	4,8	10,2
4	0	5

In der konkreten Umsetzung sollen über ein LC-Display die gemessenen Spannungswerte ausgegeben und der optimale Messbereich automatisch eingestellt werden. Im Programm werden dafür nach der Deklaration, zunächst in der `setup`-Schleife die maximalen Spannungen zu den Vorwiderständen berechnet und in einem Array gespeichert. Danach werden die Pins, die zur Steuerung des Relaismoduls verwendet werden, als `OUTPUT` definiert (Zeile 21) und sogleich auf `HIGH` gesetzt, was eine Öffnung aller Relaiskontakte bewirkt. In der nächsten Zeile wird der Ausgangszustand hergestellt: In ihm ist - zum Schutz des Arduinos - nur der Kontakt des höchsten

Widerstands geschlossen (Zeile 27).

Die `loop`-Schleife beginnt mit einem Aufruf der `Messung`-Funktion, in der U_m gemessen und daraus U_e berechnet wird. Danach erfolgt sofort die Kontrolle mit dem maximal zulässigen Spannungswert. Falls $U_e > U_{max}$ ist, wird der Relaiskontakt geöffnet und der Arduino in eine Endlosschleife geschickt (Zeile 34). Falls dies nicht der Fall ist, so schreibt die Funktion `Ausgabe`, die Werte auf das LC-Display.

Wie schon oben erwähnt, ist es von Vorteil mit einem möglichst kleinen Messbereich zu arbeiten. Aus diesem Grund wird in Zeile 40 überprüft, ob der nächstkleinere Vorwiderstand verwendet werden kann. Gibt es dafür grünes Licht, werden die notwendigen Schaltvorgänge des Relais veranlasst, sofern U_e nicht sehr nahe bei Null liegt.

Am Ende der `loop`-Schleife wird in einer letzten Abfrage (Zeile 47) noch kontrolliert, ob überhaupt gemessen wird. Sollte dies nicht der Fall sein, erfolgt die Herstellung des Ausgangszustands.

Einen wichtigen Zusammenhang muss man sich noch bewusst machen: Auf Grund der Belastung durch Relais und LC-Display, nimmt die Spannung des Arduinos leicht ab und sinkt unter 5 V. Der AD-Wandler diskretisiert jedoch den Bereich von 0 V bis zu der tatsächlich vorliegenden Arbeitsspannung in 1024 Teile. Das bedeutet, dass auch die mit `analogRead` ausgegebenen, diskreten Werte auf diesen Bereich bezogen sind. In Zeile 56 wird daher nicht mit 5 V, sondern mit 4,88 V gerechnet, die für die Entwurfschaltung mit dem Multimeter gemessen wurden. Dieser Wert wird im Allgemeinen von der Versorgungsspannung mit welcher der Arduino betrieben wird und der Belastung durch die angeschlossenen Geräte abhängen und sollte für jeden Aufbau neu gemessen werden.

```
1  #include <Wire.h>
2  #include <LiquidCrystal_I2C.h>
3
4  LiquidCrystal_I2C LCD(0x27, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);
5
6  int IN[]={8,9,10,11};  //Verwendete Pins für die Relais-Steuerung
7  int MessPin=A0;
8  float Ue, Um;  //Eingangsspannung, Spannung an Rm
9  float Umax[4];
10 const float Rm=4640, R1=37400, R2=12400, R3=4800; //Widerstandswerte
11 const float R[]={R1,R2,R3,0};
12 int n=0;
13
14 void setup() {
15
```

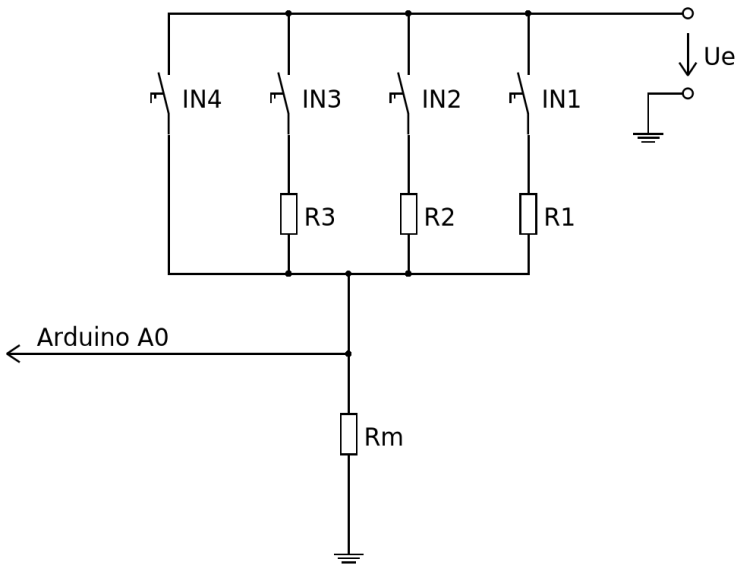

Abb. 4.1: Schaltplan des Spannungsmessgeräts

```
16   LCD.begin(16,2);
17   //Berechne die maximalen Spannungsbereiche für die Widerstände
18   for(int i=0;i<4;i++) Umax[i]=5*(R[i]/Rm + 1);
19
20   //Definiere Pins für Relais als OUTPUT
21   for(int i=0;i<4;i++) pinMode(IN[i], OUTPUT);
22
23   //Alle Relais-Kontakte öffnen
24   for(int i=0;i<4;i++) digitalWrite(IN[i], HIGH);
25
26   //Schließe das erste Relais
27   digitalWrite(IN[0], LOW);
28
29 }
30
31 void loop() {
32   Messung();
33
34   if(Ue>Umax[n]){
35     digitalWrite(IN[n], HIGH); //Sicherung: falls Ue>Umax
36     while(1);                  //wiederhole die Schleife für immer
37   }
38   Ausgabe();
39   //Falls genauerer Messbereich möglich, wechsle den Vorwiderstand
40   if(Ue<0.9*Umax[n+1] && Ue>0.00001 && n<3){
41     n++;
42     digitalWrite(IN[n-1], HIGH);
```

```
43      digitalWrite(IN[n], LOW);
44      delay(100);
45    }
46    //Falls nichts gemessen wird --> höchster Vorwiderstand
47    if(Ue<0.00001){
48      digitalWrite(IN[n], HIGH);
49      digitalWrite(IN[0], LOW);
50      n=0;
51      delay(100);
52    }
53
54 }
55 void Messung(){
56   Um=float(analogRead(MessPin))*4.88/1023; //Im Betrieb sinkt die
       Spannung des Arduinos auf 4.88V --> für genaue Ergebinsse muss
       dieser Wert mit einem Multimeter überprüft werden.
57   Ue=Um*(R[n]/Rm + 1);
58 }
59 void Ausgabe(){
60
61   LCD.clear();
62   LCD.setCursor(0,0);
63   LCD.write("0V bis ");
64   LCD.print(Umax[n],0);
65   LCD.write("V");
66   LCD.setCursor(0,1);
67   LCD.write("Ue=");
68   LCD.print(Ue,3);
69   LCD.write(" V");
70
71   delay(100);
72 }
```

Code 4.1: Programm für das Spannungsmessgerät

4.2 Kapazitätsmessung von Kondensatoren

Benötigte Materialien

- Arduino UNO
- LC-Display mit I^2C Modul
- Relaismodul
- Widerstände: 12,4 kΩ, 200 kΩ (oder ähnliche)
- Breadboard
- Jumper Wires

In diesem Projekt wollen wir eine Schaltung und einen zugehörigen Programmcode entwerfen, sodass wir die Kapazität von Kondensatoren messen und anschließend die Werte auf einem LC-Display anzeigen können. Zunächst untersuchen wir, nach

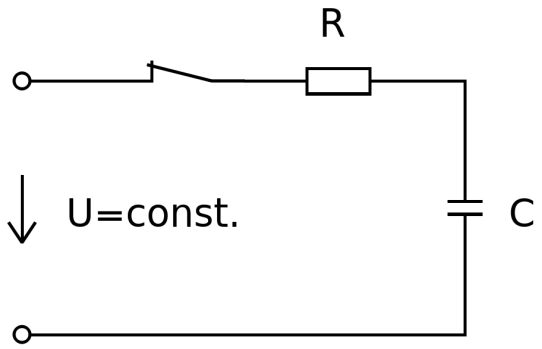

Abb. 4.2: RC-Schaltung für Lade- und Entladevorgänge

welchen Gesetzmäßigkeiten die Lade- bzw Entladevorgänge eines Kondensators ablaufen. Wir betrachten dazu die Schaltung, wie sie in Abb. 4.2 vorliegt. Hier sind eine konstante Spannung U, ein Widerstand R und ein Kondensator C in Serie geschaltet, sodass nach der Maschenregel die Summe der Spannungen Null ergibt[1]

$$U + u_R + u_c = 0\,. \tag{4.4}$$

1) Spannungsabfall am Widerstand: u_R, Spannungsabfall am Kondensator: u_C

Für den Strom i, der in der Schaltung fließt, gilt mit $dQ = C \cdot du_c$:[2]

$$i = i_C = i_R = \frac{dQ}{dt} = C \cdot \frac{du_C}{dt}\,. \tag{4.5}$$

Schreibt man Gleichung 4.4 als

$$U + i \cdot R + u_C = 0 \tag{4.6}$$

und ergänzt mit Gleichung 4.5, so erhalten wir die Differenzialgleichung

$$U + RC\frac{du_C}{dt} + u_C = 0\,, \tag{4.7}$$

welche die Lade- und Entladevorgänge beschreibt. Wir führen die Abkürzungen $\tau := RC$ und $\tilde{U} := \frac{U}{RC}$ und schreiben Gleichung 4.7 als

$$\dot{u_C} + \frac{1}{\tau}u_C + \tilde{U} = 0\,. \tag{4.8}$$

Um diese Differenzialgleichung zu lösen, betrachten wir zuerst die homogene Differenzialgleichung $\dot{u_C} = -\frac{1}{\tau}u_C$ und finden

$$u_C(t) = De^{-\frac{t}{\tau}} \tag{4.9}$$

als Lösung mit der Konstanten D. Nun lösen wir die inhomogene Differenzialgleichung über die Methode der Variation der Konstanten. Dazu leiten wir Gleichung 4.9 nach der Zeit ab

$$\dot{u_C} = \dot{D}e^{-\frac{t}{\tau}} - D\frac{1}{\tau}e^{-\frac{t}{\tau}} \tag{4.10}$$

und setzen das Ergebnis in Gleichung 4.8 ein:

$$\dot{D}e^{-\frac{t}{\tau}} - D\frac{1}{\tau}e^{-\frac{t}{\tau}} + \frac{1}{\tau}De^{-\frac{t}{\tau}} + \tilde{U} = 0\,. \tag{4.11}$$

Der zweite und der dritte Term heben sich weg und wir erhalten nach der Integration von

$$\dot{D} = \tilde{U}e^{\frac{t}{\tau}} \tag{4.12}$$

zu

$$D = \tilde{U}\frac{1}{\tau}e^{\frac{t}{\tau}} + E \tag{4.13}$$

2) Ladungsmenge: Q, Kapazität: C

und Einsetzen in Gleichung 4.9, die Lösung von Gleichung 4.8:

$$u_C(t) = \frac{1}{\tau}\tilde{U} + Ee^{-\frac{t}{\tau}} . \tag{4.14}$$

E ist dabei eine Integrationskonstante.

Wenn wir davon ausgehen, dass der Kondensator zum Zeitpunkt $t = 0$ eine Spannung $u_C(0) = U_0$ besitzt, so können wir damit E bestimmen:

$$\begin{aligned} u_C(0) = U_0 &= \frac{1}{\tau}\tilde{U} + E \\ &= U + E \\ \Longrightarrow E &= U_0 - U . \end{aligned}$$

Aus Gleichung 4.14 wird dadurch

$$u_C(t) = U\left(1 - e^{-\frac{t}{RC}}\right) + U_0 e^{\frac{t}{RC}} . \tag{4.15}$$

Diese Gleichung enthält den Lade- und Entladevorgang als Spezialfälle, sodass wir damit die gesuchten Gesetzmäßigkeiten gefunden haben:

- Laden: $U_0 = 0$

$$u_C(t) = U\left(1 - e^{-\frac{t}{RC}}\right) \tag{4.16}$$

- Entladen: $U = 0$

$$u_C(t) = U_0 e^{-\frac{t}{RC}} \tag{4.17}$$

Die Messung erfolgt nach folgendem Prinzip: Zuerst wird der Kondensator auf eine Spannung U_0 nahe 5 V geladen und dieser Wert gespeichert. Über einen Widerstand entladet man nun den Kondensator, bis dessen Spannung u_C auf etwa die Hälfte gesunken ist und stoppt die Zeit, die dafür benötigt wird. Durch Umformung von Gleichung 4.17 auf

$$C = -\frac{t}{R \ln \frac{u_C}{U_0}} \tag{4.18}$$

können wir daraus den Kapazitätswert C bestimmen.

Für die praktische Umsetzung bauen wir die Schaltung aus Abb. 4.3 auf und schreiben ein Programm. Zunächst soll der Kondensator geladen werden. Dazu wird der als digitaler Ausgang definierte Pin `12` auf `HIGH` gesetzt und das Relais über den Steuerkanal `IN1` geschlossen. Der Kondensator liegt nun zwischen der Versorgungsspannung des Arduinos und Masse und wird über einen Vorwiderstand $R_1 = 200\,\Omega$

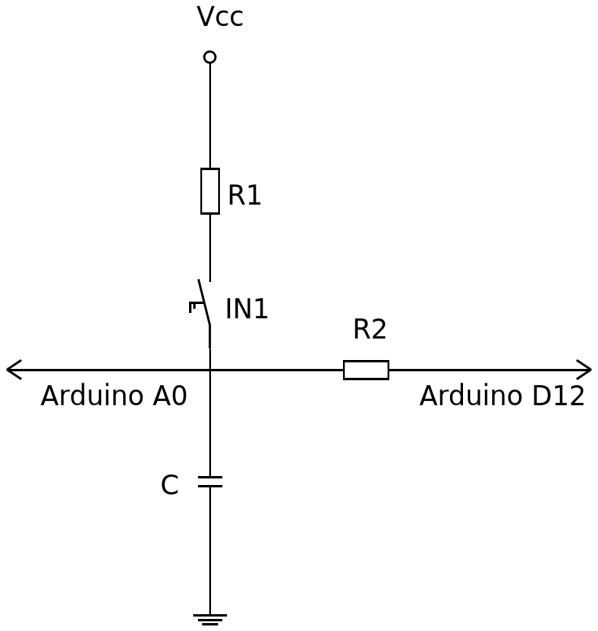

Abb. 4.3: Schematischer Aufbau zur Kapazitätsmessung

geladen, welcher der Strombegrenzung dient. Sobald der Spannungswert `uc` hoch genug ist, oder die Ladezeit 5 s überschreitet, wird das Relais geöffnet und damit der Vorgang beendet (siehe Funktion-`Laden` ab Zeile 32).
Nun beginnt der Entladevorgang (siehe Funktion-`Entladen` ab Zeile 46). Die Kondensatorspannung wird gemessen und in der Variable `U0` gespeichert. Danach wird der Timer gestartet und der Ausgang `12` auf `LOW` gesetzt, wodurch sich der Kondensator über den Widerstand $R_2 = 12{,}4\,\text{k}\Omega$ entlädt. Im Programm wird parallel dazu die Kondensatorspannung `uc` gemessen und der Timer gestoppt, sobald diese kleiner ist als die halbe Ausgangsspannung `U0`, oder eine Zeit von 10 s überschritten wird. Gleichung 4.18 dient danach zur Berechnung des Kapazitätswertes.
Die grafische Anzeige erfolgt auf einem LC-Display, das von der Funktion-`Ausgabe` (ab Zeile 62) beschrieben wird.

```
1 #include <math.h>
2 #include <Wire.h>
3 #include <LiquidCrystal_I2C.h>
4
5 LiquidCrystal_I2C LCD(0x27, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);
6
7 int messPin=A0;
8 int entladePin=12;
9 int IN1=9;                  //Pin für Relais-Steuerung
10 double C, uc, U0;          //Kapazitaet, Spannungsabfall an C,
    Anfangsspannung
```

```
11 const float R=12400;     //Verwendeter Entlade-Widerstand
12 long t=0;
13 float Vcc=4.95;
14
15 void setup() {
16   LCD.begin(16,2);
17   //Definiere Pin für Relais und Entladung als OUTPUT
18   pinMode(IN1, OUTPUT);
19   pinMode(entladePin, OUTPUT);
20   //Relais-Kontakt öffnen
21   digitalWrite(IN1, HIGH);
22   //EntladePin auf HIGH;
23   digitalWrite(entladePin, HIGH);
24 }
25
26 void loop() {
27   Laden();
28   Entladen();
29   Ausgabe();
30 }
31
32 void Laden(){
33   t=millis();
34   digitalWrite(entladePin, HIGH);
35   digitalWrite(IN1, LOW);
36   //warten bis der Kondensator über 4,6 V geladen ist, oder
37   //das Laden länger als 5 s dauert
38   while((millis()-t)<5000 && analogRead(messPin)<950){
39   }
40   //Relaiskontakt öffnen - Ladevorgang beenden
41   digitalWrite(IN1, HIGH);
42   //warten bis der Relaiszustand stabil ist
43   delay(80);
44 }
45
46 void Entladen(){
47   //Messung der Aussgangsspannung
48   U0=float(analogRead(messPin))*Vcc/1023;
49   uc=U0;
50   t=micros();       //Timer Start
51   //Entladevorgang starten
52   digitalWrite(entladePin, LOW);
53   //warte bis uc auf U0/2 gefallen ist,
54   //oder die Messung über 10s dauert
55   while(uc>U0/2 && (micros()-t)<10000000){
56     uc=float(analogRead(messPin))*Vcc/1023;
```

```
57    }
58    t=(micros()-t);   //Timer Stop
59    //Berechnung der Kapazitaet in Mikro-Farad
60    C=-t/(R* log(uc/U0));
61  }
62  void Ausgabe(){
63    LCD.clear();
64    LCD.setCursor(0,0);
65    LCD.write("Kapazitaet: ");
66    LCD.setCursor(0,1);
67    LCD.print(C,2);
68    LCD.write(" uF");
69  }
```

Code 4.2: Programm zur Messung von Kapazitäten

4.3 Messung der Schallgeschwindigkeit

Benötigte Materialien

- Arduino UNO
- Ultraschallsensor HC-SR04
- LC-Display mit I^2C Modul
- Luftdrucksensor BMP180
- Luftfeuchtigkeits- und Temperatursenor AM2301
- Breadboard
- Jumper Wires
- Maßband, Klebeband, Messobjekt z.B. Schachtel

Wie bereits in 3.15 besprochen wurde, hängt die Schallgeschwindigkeit c von vielen Einflussgrößen ab, die bei einer Messung unbedingt festgehalten werden müssen, um ein aussagekräftiges Ergebnis zu erhalten. Neben der eigentlichen Messung bestimmen wir daher noch die Temperatur T, die relative Luftfeuchtigkeit r und den Luftdruck p. Der Aufbau folgt Abb. 3.35, wobei wir den Ultraschallsensor auf das Breadboard stecken und dieses mit Klebeband auf einem Tisch fixieren. Als Reflexionsobjekt kann jeder Gegenstand verwendet werden der eine glatte ebene Fläche aufweist und einen stabilen Stand besitzt - z.B. eine solide Schachtel. Für die Aufnahme der Daten variieren wir nun den Abstand des Gegenstands zum Sensor, messen mit dem Maßband den Abstand d und mit dem unten angeführten Programm die dazugehörige Schalllaufzeit t. Ziel ist es - für möglichst konstante Umgebungsbedingungen - eine Messreihe mit Wertepaaren (d, t) zu erhalten.

Im zugehörigen Programm werden zunächst die Sensoren initiiert und die Variablen deklariert. Danach wird in der `loop`-Schleife die Temperatur gemessen und am LC-Display ausgegeben. So kann man sich während der Messungen versichern, dass die Lufttemperatur konstant bleibt. Hat man d bestimmt, so gibt man dessen Wert im seriellen Monitor ein und bestätigt. Dies bewirkt, dass der Arduino die `loop`-Schleife unterbricht und die Funktion-`serialEvent` aufruft, in der die aktuellen Sensorwerte ausgelesen und anschließend, gemeinsam mit d auf dem Bildschirm angezeigt werden (Zeilen 44-54). Die Sensorabfragen sind dabei in die beiden Funktionen `Messung` und `messeDruck` ausgelagert, um den Sketch übersichtlicher zu gestalten.

```
#include <DHT.h>
#include <SFE_BMP180.h>
#include <NewPing.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

const int AM_Pin=2;    //Pin der AM2301 Datenleitung
char AM_Typ=DHT22;     //Sensor-Modell ist DHT22=AM2301
const int echo=8;      //Pin-Nummer des Echo-Signals
const int trig=9;      //Pin-Nummer des Trigger-Signals
const int H=365;       //Seehöhe des Sensors

NewPing HC_SR(trig, echo, 200);  //Erstelle ein Objekt vom Typ NewPing mit dem Namen HC_SR
DHT dht(AM_Pin, AM_Typ);         //Erstelle Objekt vom Typ DHT mit dem Namen dht
SFE_BMP180 pressure;             //Erstelle Objekt vom Typ SFE_BMP180 mit dem Namen pressure
LiquidCrystal_I2C LCD(0x27, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE); //Erstelle Objekt vom Typ LiquidCrystal_I2C mit dem Namen LCD

int t;
double T,Tp,P,p0,r;
char status;

void setup() {
  Serial.begin(9600);
  LCD.begin(16,2);
  //Starte Kommunikation mit den Sensoren
  dht.begin();
  pressure.begin();

  Serial.println("Eingabe der gemessenen Distanz zum Objekt in mm!");
}

void loop(){
  //Anzeige der Temp. am LCD - zur Kontrolle während der Messung
  T = dht.readTemperature();  //Auslesen Temp.
  LCD.clear();
  LCD.setCursor(0,0);
  LCD.print(T,1);
  LCD.print(" C");

}
void serialEvent(){
  int d;
```

```
43   Messung();
44   d=Serial.parseInt();          //Eingabe von d am seriellen Monitor
45   Serial.print(d);
46   Serial.print(" mm, ");
47   Serial.print(t);
48   Serial.print(" Mikrosekunden, ");
49   Serial.print(T);
50   Serial.print(" C, ");
51   Serial.print(P);
52   Serial.print(" hPa, ");
53   Serial.print(r);
54   Serial.println(" %");
55 }
56 void Messung(){
57   t=HC_SR.ping_median(100);     //Messe Schalllaufzeit
58   T = dht.readTemperature();    //Messe Temp.
59   r = dht.readHumidity();       //Messe rel. Feuchte
60   messeDruck();                 //Messe den Druck
61 }
62 void messeDruck(){
63   status = pressure.startTemperature();  //Starte Temperaturmessung
64   delay(status);
65   status = pressure.getTemperature(Tp);  //Speichere Temperatur in T
66   delay(status);
67   status = pressure.startPressure(3);    //Starte Druckmessung
68   delay(status);
69   status = pressure.getPressure(P,Tp);   //Berechnung des Druckes aus
       der Messung und aus Tp
70   p0 = pressure.sealevel(P,H);           //Berechnung des reduzierten
       Druckes
71 }
```

Code 4.3: Programm zur Bestimmung der Schallgeschwindigkeit

Nach Beendigung der Messungen überträgt man die Daten vom seriellen Monitor in ein wissenschaftliches Analyseprogramm (z.B. Origin, SciDAVis, etc.) und stellt sie in einem Diagramm dar. Für den durchgeführten Versuch ist dies in Abb. 4.4 geschehen. Die Unsicherheit für den Abstand d beträgt $\pm 1\,\text{mm}$. Das Datenblatt des Ultraschallsensors enthält leider keine Angaben zur Unsicherheit. Die angegebene Auflösung von 3 mm sieht zwar gut aus, liegt jedoch mit Sicherheit weit unter der erreichbaren Genauigkeit. Bis auf Weiteres wählen wir einen relativen Fehler von 3 %, der sich im Betrieb als realistisch erwiesen hat.

Da die Laufzeit proportional zur Entfernung ist, führen wir im Analyseprogramm eine lineare Regressionsrechnung zur Funktion $y = Ax + B$ durch. Als Parameter zu

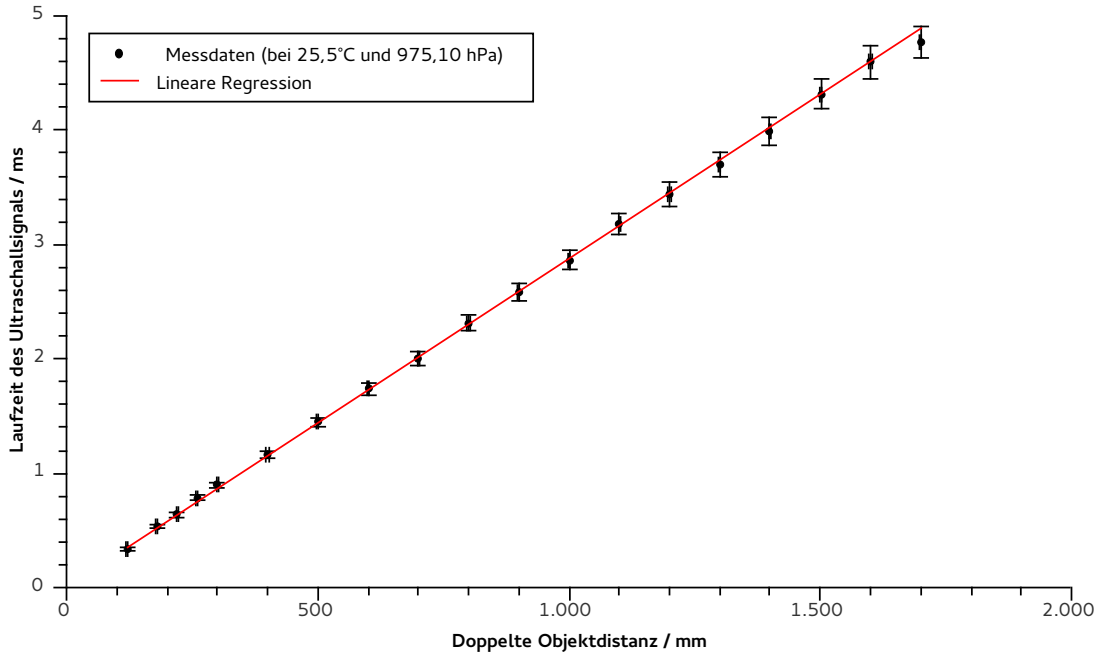

Abb. 4.4: Analyse der Messdaten zur Schallgeschwindigkeit

den konkreten Werten erhalten wir:

```
B = 0,00386362297281062 +/- 0,00967263003034253
A = 0,00287604516641659 +/- 2,89740138696185e-05
R^2 = 0,999310235388522
```

Der Korrelationskoeffizient von $0,999$ bestätigt, dass sich der lineare Zusammenhang auch in den gemessenen Daten widerspiegelt. Für die Schallgeschwindigkeit gilt

$$c = \frac{1}{A}, \tag{4.19}$$

da im Diagramm die Laufzeit über der Distanz aufgetragen wurde und somit die Steigung der Kehrwert der Geschwindigkeit ist. Schlussendlich erhalten wir bei atmosphärischen Bedingungen von
$T = 25{,}5\,°\mathrm{C}$, $p = 975{,}10\,\mathrm{hPa}$ und $r = 60{,}0\,\%$ eine Schallgeschwindigkeit von

$$c = (347,7 \pm 3,5)\ \mathrm{m\,s^{-1}}\,. \tag{4.20}$$

4.4 Bestimmung der Molmasse von Luft

Aus der Messung im vorigen Kapitel wollen wir nun versuchen, die molare Masse der Luft (zum Messzeitraum) zu berechnen. Zunächst führen wir uns vor Augen, dass die Umgebungsluft ein Gasgemisch ist, das aus vielen Komponenten besteht. Tab. 4.2 listet die Zusammensetzung und die daraus resultierende Molmasse für trockene Luft auf. Natürlich ist in der atmosphärischen Luft auch Wasserdampf enthalten, der eine molare Masse von $M_w = 18{,}016\,\mathrm{g\,mol^{-1}}$ besitzt, welche unter der von trockener Luft liegt. Wir erwarten daher, dass sich durch die Beimengung von Wasserdampf die Molmasse der Luft verringert.

Tab. 4.2: Molmasse trockener Luft [32]

Gas	Molmasse / $\mathrm{g\,mol^{-1}}$	Volumsanteil / %
Stickstoff	28,013	78,08
Sauerstoff	31,999	20,95
Argon	39,948	0,93
Kohlendioxid	44,010	0,04
trockene Luft	28,966	

Um nun von der Schallgeschwindigkeit auf die Molmasse zu kommen, gehen wir von der Grundgleichung für flüssige und gasförmige Stoffe aus, die den Kompressionsmodul K und die Dichte ρ enthält

$$c = \sqrt{\frac{K}{\rho}}\,. \tag{4.21}$$

Aus der allgemeinen Gasgleichung für ideale Gase

$$pV = nRT \tag{4.22}$$

gewinnen wir nach kurzer Rechnung[3]

$$p \cdot \frac{m}{\rho} = nRT \tag{4.23}$$

$$p \cdot \frac{M}{\rho} = RT \tag{4.24}$$

$$p = \frac{RT\rho}{M}\,. \tag{4.25}$$

Mit dem Adiabatenexponent κ (für Luft ist $\kappa = 1,4$) und der Beziehung $K = \kappa \cdot p$

3) Luftdruck: p, Volumen: V, Anzahl der Mole: n, universelle Gaskonstante: $R = 8{,}314\,41\,\mathrm{J\,mol^{-1}\,K^{-1}}$, Temperatur: T, Masse: m, Molmasse: M

folgt aus den Gleichungen 4.21 und 4.25

$$c = \sqrt{\frac{\kappa RT}{M}} \tag{4.26}$$

und letztlich die gesuchte Beziehung

$$M = \frac{\kappa RT}{c^2} \tag{4.27}$$

für die molare Masse der Luft.

Die Temperatur $T = 298{,}65\,\mathrm{K}$ und die Schallgeschwindigkeit $c = 347{,}7\,\mathrm{m\,s^{-1}}$ wurden im vorigen Kapitel für den Messzeitraum bestimmt, sodass wir für die molare Masse der Luft

$$M = 28{,}755\,\mathrm{g\,mol^{-1}} \tag{4.28}$$

erhalten. Wie erwartet liegt sie unter der molaren Masse von trockener Luft.[4]

Dieser Wert hat ohne der Angabe eines Fehlerbereiches relativ wenig Aussagekraft. Wir führen daher noch eine Fehlerrechnung nach der Methode des Größtfehlers durch. Allgemein berechnet sich dieser, für eine Funktion $f(x_1, x_2, x_3, ..., x_n)$ die von den Variablen $x_1, x_2, x_3, ..., x_n$ abhängt, zu

$$\Delta f = \sum_{i=1}^{n} \left| \frac{\partial f}{\partial x_i} \right| \cdot \Delta x_i \tag{4.29}$$

Da die Molmasse laut Gleichung 4.27 eine Funktion von T und c ist, führen wir folgende Rechnung durch:

$$\Delta M = \left| \frac{\partial M}{\partial T} \right| \cdot \Delta T + \left| \frac{\partial M}{\partial c} \right| \cdot \Delta c \tag{4.30}$$

$$\Delta M = \left| \frac{\kappa R}{c^2} \right| \cdot \Delta T + \left| \frac{2\kappa RT}{c^3} \right| \Delta c\,. \tag{4.31}$$

Die Unsicherheiten für Temperatur und Schallgeschwindigkeit sind mit $\Delta T = 0{,}5\,\mathrm{K}$ (Datenblatt des Sensors) und $\Delta c = 3{,}5\,\mathrm{m\,s^{-1}}$ bekannt, wodurch wir für den Größtfehler der Molmasse

$$\Delta M = 6{,}27 \cdot 10^{-4}\,\mathrm{kg\,mol^{-1}} \tag{4.32}$$

4) Wir haben für die Rechnung $\kappa = 1,4$ als konstant betrachtet. Dieser Wert gilt streng genommen nur für trockene Luft und verringert sich leicht mit der Aufnahme von Wasserdampf. Da die Änderung im Vergleich zum Einfluss von T und c^2 wirklich gering ist, können wir sie hier vernachlässigen.

und damit als Ergebnis

$$M = (28,8 \pm 0,6)\ \mathrm{g\,mol^{-1}} \tag{4.33}$$

erhalten. Daraus ergibt sich ein relativer Fehler von 2,1 %.

4.5 Wetterstation mit Unwetterwarnung

Benötigte Materialien

- Arduino UNO
- LC-Display mit I²C Modul
- Luftdrucksensor BMP180
- Luftfeuchtigkeits- und Temperatursenor AM2301
- rote LED, orange LED (oder ähnliche)
- 160 Ω Widerstand
- Breadboard
- Jumper Wires

In diesem Projekt wollen wir die bereits vertrauten Sensoren AM2301 und BMP180 zur Messung von Temperatur, Luftfeuchtigkeit und Luftdruck verwenden. Die Werte sollen in regelmäßigen Abständen ausgelesen und auf dem LC-Display angezeigt werden, um jederzeit den aktuellen Zustand der Atmosphäre kontrollieren zu können. Darüber hinaus soll unsere Wetterstation auch über eine Unwetterwarnung verfügen, die den Besitzer über optische Signale (LEDs) vor Gefahren warnt. Der Aufbau ist in Abb. 4.5 abgebildet.

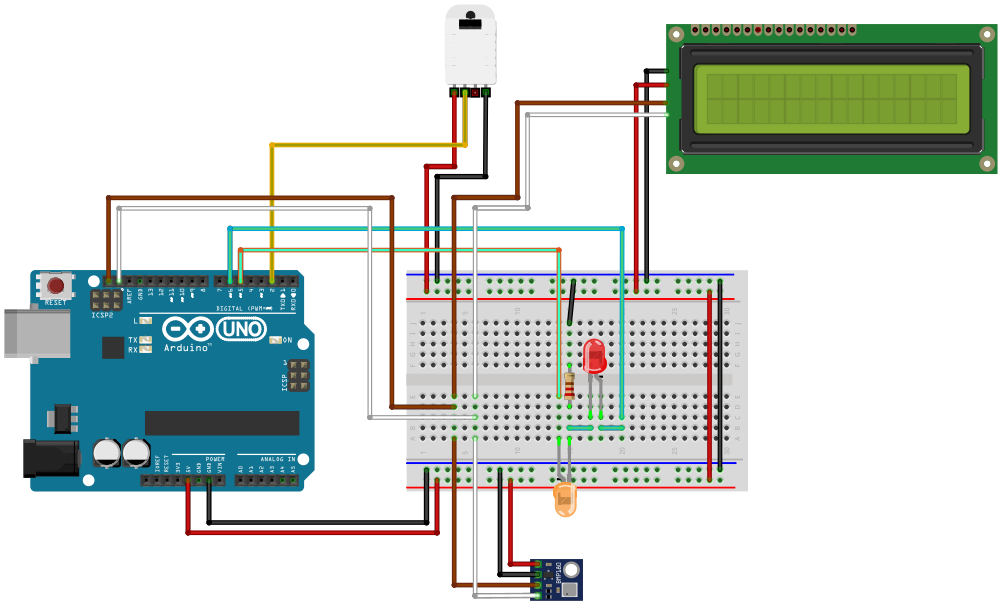

Abb. 4.5: Schaltplan der Wetterstation mit Unwetterwarnung

In der Wettervorhersage versucht man das äußerst komplexe und chaotische System der Atmosphäre zu beschreiben und darüber hinaus noch Aussagen über zukünftige Zustände zu tätigen. Dass dieses Unterfangen nicht ganz so einfach ist, können wir von den nicht eintreffenden Prognosen aus Rundfunk und Internet ableiten. Nichts desto trotz sind die Vorhersagen heute erstaunlich gut und in den meisten Fällen, bereits für einen Zeitraum von drei Tagen zutreffend. Um dies zu erreichen wird der gesamte Globus mit einem Messnetz umzogen. Nicht nur auf der Erdoberfläche, sondern auch in einzelnen Schichten bis zur Stratosphäre. An den Knotenpunkten werden durch Messstationen oder Satelliten die atmosphärischen Zustandsgrößen bestimmt und daraus mit Hilfe von Riesencomputern das wahrscheinlichste Wettergeschehen berechnet.

Unser Arduino ist für so eine Aufgabe natürlich völlig ungeeignet und doch kann er wichtige Dienste in der Unwetterwarnung leisten. Gewisse Wettervorgänge weisen charakteristische Verläufe in Druck, Temperatur, Luftfeuchtigkeit, Windrichtung oder Windgeschwindigkeit auf, die wir mit Hilfe des Arduinos detektieren können. Im Speziellen wollen wir unser Augenmerk auf die Kaltfront richten. Für eine genauere Beschreibung der Fronten-Theorie sei an dieser Stelle auf [32] verwiesen. Lokal betrachtet schiebt sich eine schnelle, kühle Luftschicht unter eine warme, feuchte Luftschicht und zwingt sie so zum Aufsteigen. Durch den niedrigeren Druck in der Höhe expandiert die warme Luft, kühlt dabei ab und beginnt zu kondensieren - als Folge können starke Gewitter, Stürme und lang anhaltende Regenschauer entstehen. Der Durchgang einer Kaltfront ist am Boden mit einem immer schneller werdenden Luftdruckabfall verbunden, der nach dem Passieren wieder ebenso schnell ansteigt. In Abb. 4.6 ist die Messung an der Wetterstation der Universität Graz beim Durchgang einer Kaltfront am 31.7.2016 dargestellt. Man sieht hier sehr deutlich das charakteristische Verhalten, welches wir nun auch für die Unwetterwarnung unserer Wetterstation nützen wollen.

Das Prinzip ist dabei folgendes: Wir messen jede halbe Stunde den Luftdruck und speichern den Wert in einem Array `p` welches für 7 Werte Platz bietet. Somit bleiben die letzten drei Stunden verfügbar für weitere Auswertungen. Die gespeicherten Werte werden nun permanent mit dem aktuellen Luftdruck `pm` verglichen und die Druckdifferenzen bestimmt. Sobald diese einen kritischen Wert übersteigen, werden verschiedene Alarmstufen aktiviert (siehe Tab. 4.3). Je größer die mittlere Druckänderung ist, desto höher ist auch die Alarmstufe.

Zunächst möchten wir uns kurz das Programmgerüst ansehen und erst im Anschluss daran auf die Details eingehen. Das Programm beginnt mit den nötigen Deklarationen und Initialisierungen, die in Zeile 43 mit dem Aufruf der Funktion-`start` ab-

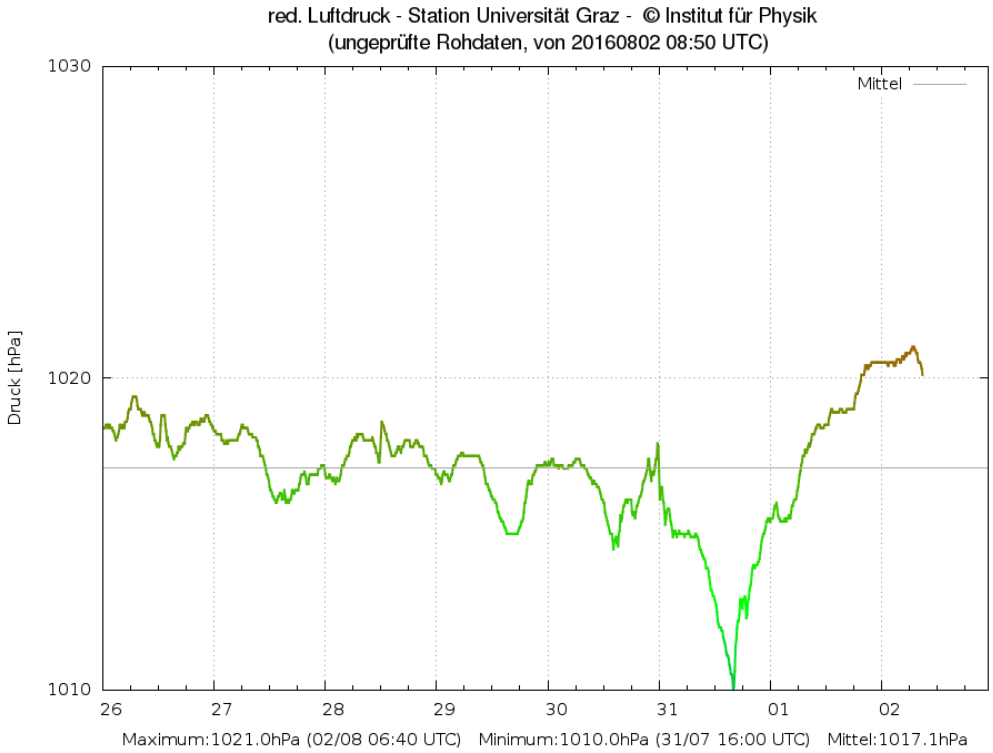

Abb. 4.6: Luftdruckverlauf vom 26.7. bis 2.8.2016 an der Messstation der Universität Graz [33]

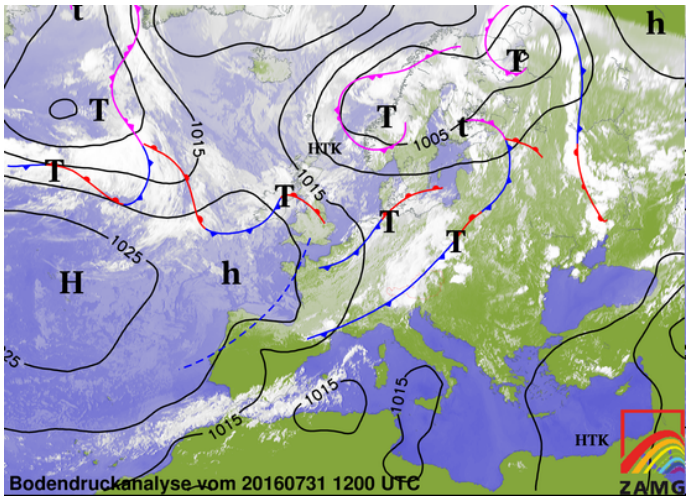

Abb. 4.7: Wetterkarte vom 31.7.2016 um 14:00 Uhr mit Kaltfront über Österreich [34]

geschlossen werden. Danach wird die `loop`-Schleife betreten, welche aus vier Teilen besteht: Drei `if`-Abfragen in denen geregelt wird wann eine Messung durchgeführt wird, wann eine Ausgabe auf das LC-Display und die serielle Schnittstelle erfolgt und wann die gespeicherten Luftdruckwerte aktualisiert werden. Der vierte Punkt ist der Aufruf der Funktion-`Unwettercheck`, in der die Überwachung der Luftdruckänderung und die damit verbundene Steuerung der Warnsignale erledigt werden.

Tab. 4.3: Alarmstufen der Unwetterwarnung

Abfrage	Alarmstufe	Signal
`p[1] - pm > 1 hPa`	4	rote LED blinkt
`p[2] - pm > 1 hPa`	3	rote LED ein
`p[4] - pm > 1 hPa`	2	rote und orange LED ein
`p[6] - pm > 1 hPa`	1	orange LED ein
sonst	0	keines

Abschließend betrachten wir kurz die erstellten Funktionen:

- `messeDruck()` - Zeile 85: Liest den Drucksensor aus und speichert den absoluten Druck in `P`, den reduzierten Druck in `p0`, die Temperatur in `Tp`.
- `Messung()` - Zeile 76: Liest den Feuchtigkeitssensor aus und speichert die Temperatur in `T` und die relative Luftfeuchtigkeit in `r`. Danach wird die Funktion-`messeDruck` aufgerufen und die exponentiell geglätteten Mittel `pm`, `Tm` und `rm` gebildet. Somit sind nach dem Aufruf dieser Funktion alle verfügbaren Atmosphärenwerte aktuell.
- `update(double v, float h[7])` - Zeile 177: Besitzt zwei Übergabeparameter - den aktuellen Luftdruckwert `pm` und das Array `p`. Die bereits gespeicherten Werte in `p` werden um eine Position weiter geschoben. Dadurch fällt der älteste Wert aus der Liste. An die erste Stelle `p[0]` kommt `pm`.
- `start()` - Zeile 67: Wird noch in der Initialisierungsphase aufgerufen und führt einige Male die Funktion-`Messung` aus - um gute Startwerte für `pm`, `Tm` und `rm` zu bekommen. Danach wird `update` aufgerufen und `pm` dem Array `p` hinzugefügt.
- `DataOut()` - Zeile 96: Schickt alle verfügbaren Daten als leerzeichengetrennte Zeile an die serielle Schnittstelle.
- `LCDAusgabe()` - Zeile 127: Gibt die geglätteten Werte `pm`, `Tm`, `rm` und die aktuelle Alarmstufe am LC-Display aus.
- `Unwettercheck()` - Zeile 140: Überprüft die Druckänderungen verschiedener Zeitspannen und ändert bei Bedarf die Alarmstufe und die LED-Signale.

```
#include <DHT.h>
#include <SFE_BMP180.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

const int AM_Pin=2;         //Pin der AM2301 Datenleitung
char AM_Typ=DHT22;          //Sensor-Modell ist DHT22=AM2301
const int H=365;            //Seehöhe des Sensors
const int Led1=5, Led2=6; //Pins der Signal-LEDS

DHT dht(AM_Pin, AM_Typ);   //Erstelle Objekt vom Typ DHT mit dem Namen
    dht
SFE_BMP180 pressure;        //Erstelle Objekt vom Typ SFE_BMP180 mit dem
    Namen pressure
LiquidCrystal_I2C LCD(0x27, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE); //
    Erstelle Objekt vom Typ LiquidCrystal_I2C mit dem Namen LCD

unsigned long ttmess=2000;      //Intervall der Messungen
unsigned long ttausgabe=10000; //Intervall der LCD/serial-Ausgabe
unsigned long ttupdate=1800000;//1000*60*30 entspr. 30min in ms
unsigned long t1,t2,t3;         //Zeitvariablen für die Intervalle
double T,Tp,P,p0,r,pm,Tm,rm;
char status;
int z=0, zustand=0;
//Array für die Luftdruckwerte der letzten 3h
float p[]={
  0,1,2,3,4,5,6};
void setup() {
  Serial.begin(9600);
  LCD.begin(16,2);
  //Starte Kommunikation mit den Sensoren
  dht.begin();
  pressure.begin();

  pinMode(Led1, OUTPUT);
  pinMode(Led2, OUTPUT);

  LCD.print("Starte...");
  Messung();

  pm=p0;    //
  Tm=T;     //initialisiere Variablen für die exponentielle Glättung
  rm=r;     //

  t1=t2=t3=0;
  start();  //kurze Messserie für gute Startwerte
```

```
44 }
45
46 void loop(){
47   //Messe p,T,r alle ttmess ms
48   if((millis()-t1)>ttmess){
49     Messung();
50     t1=millis();
51   }
52   //Ausgabe an LCD und serielle Schnittstelle alle ttausgabe ms
53   if((millis()-t2)>ttausgabe){
54     DataOut();
55     LCDAusgabe();
56     t2=millis();
57   }
58   //Erstelle neue Druckhistorie alle ttupdate ms
59   if((millis()-t3)>ttupdate){
60     update(pm,p);
61     t3=millis();
62   }
63   Unwettercheck(); //Überprüfe die Unwettergefahr
64 }
65
66 /*******************Funktionen*******************/
67
68 void start(){
69   //kurze Messserie für einen guten Startwert in p
70   for(int i=0;i<10;i++){
71     Messung();
72     delay(2000);
73   }
74   update(pm,p);
75 }
76
77 void Messung(){
78   T = dht.readTemperature();  //Messe Temp.
79   r = dht.readHumidity();     //Messe rel. Feuchte
80   messeDruck();               //Messe den Druck
81   pm=0.9*pm+0.1*p0;           //
82   Tm=0.9*Tm+0.1*T;            //geglättetete Werte
83   rm=0.9*rm+0.1*r;            //
84 }
85
86 void messeDruck(){
87   status = pressure.startTemperature();  //Starte Temperaturmessung
88   delay(status);
89   status = pressure.getTemperature(Tp);  //Speichere Temperatur in T
```

```
90    delay(status);
91    status = pressure.startPressure(3);     //Starte Druckmessung
92    delay(status);
93    status = pressure.getPressure(P,Tp);    //Berechnung des Druckes aus
        der Messung und aus Tp
94    p0 = pressure.sealevel(P,H);            //Berechnung des reduzierten
        Druckes
95  }
96
97  void DataOut(){
98    //Ausgabe aller relevanten Daten an die serielle Schnittstelle
99    Serial.print(T);
100   Serial.print(" ");
101   Serial.print(p0);
102   Serial.print(" ");
103   Serial.print(r);
104   Serial.print(" ");
105   Serial.print(Tm);
106   Serial.print(" ");
107   Serial.print(pm);
108   Serial.print(" ");
109   Serial.print(rm);
110   Serial.print(" ");
111   Serial.print(z);
112   Serial.print(" ");
113   Serial.print(p[0]);
114   Serial.print(" ");
115   Serial.print(p[1]);
116   Serial.print(" ");
117   Serial.print(p[2]);
118   Serial.print(" ");
119   Serial.print(p[3]);
120   Serial.print(" ");
121   Serial.print(p[4]);
122   Serial.print(" ");
123   Serial.print(p[5]);
124   Serial.print(" ");
125   Serial.println(p[6]);
126 }
127
128 void LCDAusgabe(){
129   //Ausgabe von pm, Tm, rm und der Alarmstufe an das LC-Display
130   LCD.clear();
131   LCD.setCursor(0,0);
132   LCD.print(Tm,1);
133   LCD.print("C, ");
```

```
134   LCD.print(rm,1);
135   LCD.print("%");
136   LCD.setCursor(0,1);
137   LCD.print(pm);
138   LCD.print("hPa, AS=");
139   LCD.print(z);
140 }
141 void Unwettercheck(){
142   //Abfragen ob innerhalb des betrachteten Zeitintervalls der Luftdruck
143   //um mehr als 1 hPa gesunken ist und aktiviert die entsprechenden
        LEDs
144
145   //Alarmstufe 4 Druckänderung >1 hPa/30 min
146   if((p[1]-pm)>1){
147     z=4;
148     zustand=zustand+1;
149     digitalWrite(Led1,LOW);
150     analogWrite(Led2,zustand);
151     delay(5);
152   }
153   //Alarmstufe 3 Druckänderung >1 hPa/60 min
154   else if((p[2]-pm)>1){
155     z=3;
156     digitalWrite(Led1,LOW);
157     digitalWrite(Led2,HIGH);
158   }
159   //Alarmstufe 2 Druckänderung >1 hPa/120 min
160   else if((p[4]-pm)>1){
161     z=2;
162     digitalWrite(Led1,HIGH);
163     digitalWrite(Led2,HIGH);
164   }
165   //Alarmstufe 4 Druckänderung >1 hPa/180 min
166   else if((p[6]-pm)>1){
167     z=1;
168     digitalWrite(Led1,HIGH);
169     digitalWrite(Led2,LOW);
170   }
171   else{
172     z=0;
173     digitalWrite(Led1,LOW);
174     digitalWrite(Led2,LOW);
175   }
176 }
177
178 void update(double v, float h[7]){
```

```
179   //Schiebt die alten Werte des arrays eine Position weiter und
        schreibt den neuen an die erste Stelle
180   for(int i=6;i>0;i--){
181     h[i]=h[i-1];
182   }
183   h[0]=v;
184 }
```

Code 4.4: Wetterstation mit Unwetterwarnung

4.6 Radar - Geschwindigkeitsmessung über Winkeländerung zum geradlinig bewegten Objekt

Benötigte Materialien

- Arduino UNO
- LC-Display mit I^2C Modul
- GY-521 Breakout Board mit MPU6050
- 3 Taster
- lineares 10 kΩ Potentiometer
- Breadboard
- Jumper Wires

Die Geschwindigkeit ist eine wichtige Größe in unserem Alltag und schon von klein auf mit interessanten Fragen verbunden. Wie schnell kann ich laufen? Haltet sich das Auto an die vorgeschriebene Geschwindigkeitsgrenze? Wie schnell fliegt ein Flugzeug, ein Fußball oder ein Paragleiter? Mit dem nächsten Projekt wollen wir Antworten auf diese Fragen liefern, indem wir mit Hilfe des Gyrometers, ein wenig Mathematik und natürlich dem Wissen über den Umgang mit dem Arduino, ein Geschwindigkeitsmessgerät für eindimensionale Bewegungen bauen.

Fährt ein Motorrad entlang einer geraden Straße, so sind die geometrischen Beziehungen zu einem am Straßenrand stehenden Beobachter wie in Abb. 4.8. Über die trigonometrischen Funktionen gewinnen wir die Beziehung

$$\tan\varphi = \frac{x}{a}\,, \tag{4.34}$$

wobei x und φ Funktionen der Zeit sind. Um die Geschwindigkeit des Motorrads zu bestimmen, formen wir zunächst die obige Gleichung auf

$$x = a\tan\varphi \tag{4.35}$$

um und leiten anschließend nach der Zeit ab

$$v(t) = \frac{dx}{dt} = a\frac{1}{\cos^2\varphi}\frac{d\varphi}{dt}\,. \tag{4.36}$$

Die Geschwindigkeit ist nach Gleichung 4.36 darstellbar als Funktion des Win-

kels $\varphi(t)$ und der Winkelgeschwindigkeit $\omega(t) = \frac{d\varphi(t)}{dt}$ - beides können wir mit dem Gyrometer bestimmen (siehe 3.11), indem wir damit das vorbeifahrende Motorrad anvisieren und die Messdaten auswerten.

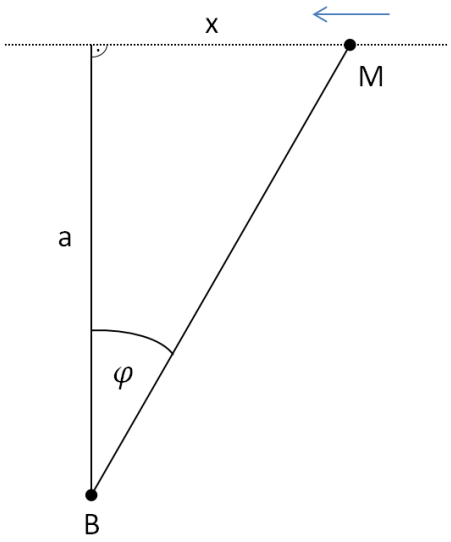

Abb. 4.8: Geometrischer Zusammenhang für die Geschwindigkeitsmessung. Beobachter B, Motorrad M, Normalabstand zur Straße a, Entfernung x zum Nullpunkt, Winkel zwischen Nullpunkt und Motorrad φ

Wir haben nun die theoretischen Zusammenhänge erarbeitet und können uns an die Umsetzung unseres Radargeräts machen. An einem LC-Display sollen der Winkel φ und die Momentangeschwindigkeit ausgegeben werden und zusätzlich auch die mittlere Geschwindigkeit über einen Messzeitraum, der mit einem Taster bestimmt werden kann. Um die Funktionalität zu erhöhen, soll der Abstand a zur Fahrbahn über ein Potentiometer einstellbar sein und der Winkel φ - zur genauen Ausrichtung der Normalrichtung zur Straße - über einen Taster rückgesetzt werden können. Der Aufbau ist in Abb. 4.9 dargestellt.

Im Code verwenden wir erstmals eine Interrupt Routine, die eine praktische Interaktion während des Betriebs ermöglicht (Zeile 40). Liegt der Pegel an Kanal 2 auf `LOW`, wird das laufende Programm sofort unterbrochen und die angegebene Funktion ausgeführt - in unserem Fall `offsetReset`. Sobald der Pegel nicht mehr auf `LOW` liegt, wird das Programm dort fortgesetzt wo es unterbrochen wurde. Der Arduino UNO besitzt zwei Interrupt Routinen: Interrupt 0 auf Kanal 2 und Interrupt 1 auf Kanal 3. Als Triggermodi für die Aktivierung des Interrupts gibt es `LOW`, `CHANGE`, `RISING` und `FALLING`.

Weiters kommt eine Tastenentprellung zum Einsatz. Da Taster schwingfähige mechanische Systeme sind, ändert ein Taster bei Betätigung innerhalb einiger Millisekunden mehrmals seinen Zustand bis er stabil ist. Für einen korrekten Ablauf

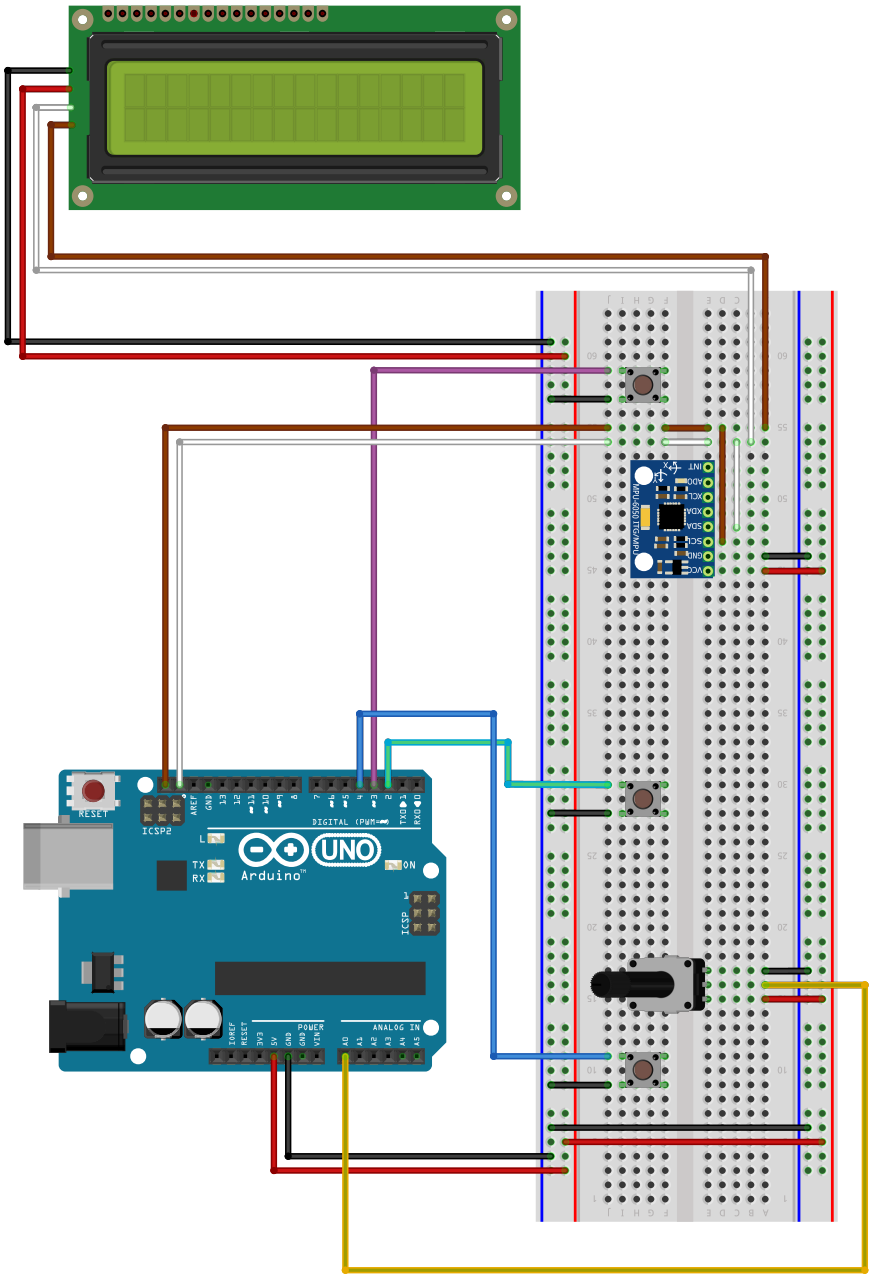

Abb. 4.9: Aufbau des „Radars“

des Programms müssen die Tasterzustände jedoch eindeutig sein. Meist wird dieses Problem auf Softwareebene gelöst, indem der Tasterzustand wiederholt abgefragt und erst geändert wird, wenn er stabil ist. Die von Thomas Fredericks entwickelte Bibliothek `Bounce2` übernimmt diese Überprüfungen und kann unter https://github.com/thomasfredericks/Bounce-Arduino-Wiring/archive/master.zip heruntergeladen werden.

Nun gehen wir genauer auf den Code ein. Zu Beginn definieren wir globale Variablen, auf die wir im weiteren Programm zugreifen möchten. Dabei erstellen wir auch zwei `Bounce`-Objekte entprell_abstand_pin und entprell_vm_pin, mit deren Hilfe wir entprellte, also eindeutige Tasterzustände bekommen. Die `setup`-Funktion enthält zunächst die Deklarationen der verwendeten Eingänge und die Aktivierung der benötigten Pullup-Widerstände. Danach wird die Tastenentprellung konfiguriert (Zeile 44), die I^2C Verbindung zu LC-Display und Sensor gestartet und der Sensor mit den Funktionen `offset` und `minmax` für den Betrieb vorbereitet. Die ersten Zeilen der `loop`-Funktion (77-90) sind aus dem Programm von 3.11 übernommen und enthalten zusätzlich noch die Berechnung der Momentangeschwindigkeit nach Gleichung 4.36. Danach werden in `berechneVm` und `checkAbstand` die mittlere Geschwindigkeit über eine mit dem Taster bestimmbare Dauer berechnet und überprüft ob der Normalabstand zur Straße verändert werden soll. Den Abschluss bildet die Funktion-`lcdAusgabe`, welche die grafische Ausgabe der wichtigsten Werte auf dem LC-Display übernimmt.

Wir betrachten die erstellten Funktionen genauer:

- `entprellSetup()` - Zeile 184: Für die beiden `Bounce`-Objekte entprell_abstand_pin und entprell_vm_pin werden die Kanalnummern vergeben und mit der Funktion-`intervall` die Zeit in Millisekunden übergeben, für die der Eingangszustand mindestens stabil sein muss, um als neuer Zustand akzeptiert zu werden.
- `offset()` - Zeile : Siehe 3.11
- `minmax()` - Zeile : Siehe 3.11
- `offsetReset()` - Zeile 150: Wird vom Interrupt 0 aufgerufen sobald der Zustand am Eingang 2 auf `LOW` liegt. Die Funktion setzt alle drei Raumwinkel auf 0° und aktualisiert damit den Koordinatennullpunkt. Sollte die Normalrichtung zur Straße neu definiert werden müssen, so kann dies mit dem Aufruf dieser Funktion - bzw. Drücken des entsprechenden Tasters - geschehen.
- `berechneVm()` - Zeile 191: Am Beginn wird der Zustand von entprell_vm_pin überprüft. Sollte dieser von `HIGH` auf `LOW` fallen - durch das Drücken des ent-

sprechenden Tasters - wird die aktuelle Zeit und der zugehörige Winkel in den Variablen `t0` und `alpha_0` gespeichert. Sollte der Zustand von `LOW` auf `HIGH` wechseln - also der Taster wieder losgelassen worden sein - werden aktuelle Zeit und Winkel in `t1` und `alpha_1` gespeichert. Mit diesen Werten ist es möglich,

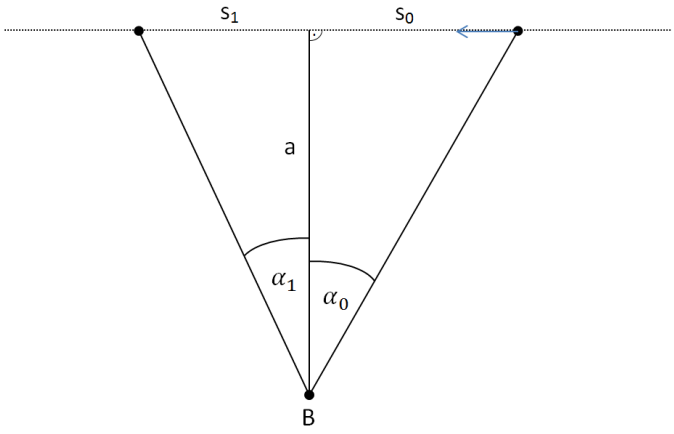

Abb. 4.10: Geometrische Zusammenhänge zur Berechnung der mittleren Geschwindigkeit

die mittlere Geschwindigkeit im Intervall [`t0`, `t1`] zu berechnen. In Abb. 4.10 befindet sich ein sich nach links bewegendes Objekt auf einer geraden Straße. Im Normalabstand a zur Straße steht der Beobachter B mit dem Sensor. Wegen der Antisymmetrie der tan-Funktion, gilt für den im Intervall [`t0`, `t1`] zurückgelegten Weg

$$s = a \cdot (\tan \alpha_0 - \tan \alpha_1) \tag{4.37}$$

und damit für die mittlere Geschwindigkeit

$$v_m = a \cdot \frac{\tan \alpha_0 - \tan \alpha_1}{t_1 - t_0} \,. \tag{4.38}$$

In Zeile 202 wird Gleichung 4.38 ausgewertet und das Ergebnis in der Variable `vm` gespeichert.

- `checkAbstand()` - Zeile 206: Mit Hilfe dieser Funktion kann der Abstand zur Straße - während des Betriebs - eingestellt werden. Am Beginn wird der Zustand von entprell_abstand_pin aktualisiert, ausgelesen und in der Variablen `zustand` gespeichert. Wird der entsprechende Taster betätigt, so ist `zustand` auf `LOW` und die `while`-Schleife wird ausgeführt. Hier wird in Zeile 210 die Spannung am Potentiometer abgefragt, die je nach Position zwischen 0 und 5 V liegt, auf einen Wert zwischen 0 und 25 m transformiert und in `abstand` gespeichert. Zur grafischen Rückmeldung wird dieser Wert am LC-Display dargestellt. Es folgt das Aktualisieren und Auslesen von entprell_abstand_pin,

von dessen Wert der Verbleib in der `while`-Schleife abhängt.

- `lcdAusgabe()` - Zeile 170: Diese Funktion gibt die Momentangeschwindigkeit und die letzte gemessene mittlere Geschwindigkeit in Meter pro Sekunde und den Winkel zwischen Sensorachse und Straßennormale auf dem LC-Display aus.

Mit einem USB-Akku kann damit einfach und unabhängig vom Computer die Geschwindigkeit verschiedenster geradlinig bewegter Objekte gemessen werden.

```
1  #include <I2Cdev.h>
2  #include <MPU6050.h>
3  #include <Wire.h>
4  #include <LiquidCrystal_I2C.h>
5  #include <Bounce2.h>
6
7  LiquidCrystal_I2C LCD(0x27, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);
8  MPU6050 mpu;        //Erzeuge ein Objekt vom Typ MPU6050
9  int16_t w[3];
10 int8_t mode=6;     //Tiefpass Arbeitsmodus des MPU6050 (von 0=aus bis 6)
       siehe MPU6050.cpp
11 int offset_pin=2; //Taster zur Winkelrücksetzung
12 int vm_pin=3;      //Taster zur Messung der mittleren Geschwindigkeit vm
13 int abstand_pin=4;
14 int poti_pin=A0;
15 //Initialisiere Bounce-Objekte für die zu entprellenden Taster
16 Bounce entprell_abstand_pin=Bounce();
17 Bounce entprell_vm_pin=Bounce();
18
19 unsigned long t,t0,t1,dt;
20 int MIN[3],MAX[3];
21 double dphi[3],phi[]={
22   0,0,0};
23 double w_off[]={
24   0,0,0};
25 int z=0;
26 double abstand=3;
27 double vm=0; //mittlere Geschwindigkeit vm, für ein mit einem Taster
       bestimmbares Zeitintervall
28 double alpha_0, alpha_1;
29 double v[]={
30   0,0,0};
31 double w_abs;
32 const float Pi=3.14159265358979323846;
33
34 void setup() {
```

```
35   pinMode(abstand_pin,INPUT);
36   pinMode(poti_pin,INPUT);
37   pinMode(offset_pin, INPUT);
38   digitalWrite(abstand_pin, HIGH);      //interner Pullup aktiviert
39   digitalWrite(offset_pin, HIGH);       //interner Pullup aktiviert
40   attachInterrupt(0, offsetReset, LOW);//Interrupt0 auf Pin2
41   pinMode(vm_pin, INPUT);
42   digitalWrite(vm_pin,HIGH);            //interner Pullup aktiviert
43
44   entprellSetup();
45
46   Wire.begin();
47   LCD.begin(16,2);
48   LCD.clear();
49   LCD.setCursor(0,0);
50   Serial.begin(115200);
51   LCD.print("Starte I2C");
52   mpu.initialize();
53   // Teste die Geräteverbindung
54   LCD.clear();
55   LCD.setCursor(0,0);
56   LCD.print("Geraeteverbindung...");
57   if(mpu.testConnection()){
58     LCD.clear();
59     LCD.setCursor(0,0);
60     LCD.print("MPU verbunden!");
61   }
62   else{
63     LCD.clear();
64     LCD.setCursor(0,0);
65     LCD.print("Fehler!");
66   }
67   mpu.setDLPFMode(mode);   //Konfiguration für integrierten Tiefpass
68   offset();                //Bestimme Offsetwerte des Sensors
69   minmax();                //Bestimme Bereich der Rauschunterdrückung
70
71   LCD.clear();
72   LCD.setCursor(0,0);
73   LCD.print("Messung...");
74   t=micros();
75 }
76
77 void loop() {
78   dt=micros()-t;           //Diskretisierung der Zeit
79   t=micros();
80   //Auslesen der Winkelgeschwindigkeiten
```

```
81    mpu.getRotation(&w[0], &w[1], &w[2]);
82
83    //Numerische Integration zur Bestimmung der Winkel aus den
        Winkelgeschwindigkeiten
84    w_abs=w[0]-w_off[0];
85    if(w_abs>MAX[0] || w_abs<MIN[0]){
86      dphi[0]=w_abs*dt*0.000001/131;
87      phi[0]=phi[0]+dphi[0];
88      v[0]=abstand*(1/sq(cos(Pi*phi[0]/180)))*(Pi/180)*w_abs/131;
89      z=z+1;  //Erhöhe z wenn ein Messwert die Kriterien erfüllt
90    }
91    berechneVm();
92    checkAbstand();
93    //Ausgabe nach 80 angenommenen Messwerten
94    if(z>=80){
95      //serielleAusgabe();
96      lcdAusgabe();
97      z=0;
98    }
99  }
100 /******************Berechnung der Offsetwerte****************/
101 void offset() {
102   LCD.clear();
103   LCD.setCursor(0,0);
104   LCD.print("Berechne Offset.");
105   mpu.setXGyroOffset(0);
106   mpu.setYGyroOffset(0);
107   mpu.setZGyroOffset(0);
108   double sum[]={
109     0,0,0                                  };
110   int n=100;
111   //Auslesen einiger Werte - zur Stabilisierung des Sensors
112   for(int i=0;i<1000;i++){
113     mpu.getRotation(&w[0], &w[1], &w[2]);
114   }
115   delay(100);
116   //Ermittlung der Startwerte für das exponentiell geglättete Mittel
117   for(int i=0;i<n;i++){
118     mpu.getRotation(&w[0], &w[1], &w[2]);
119     for(int j=0;j<3;j++) sum[j]=sum[j]+w[j];
120   }
121   for(int i=0;i<3;i++) w_off[i]=sum[i]/n;
122   //Berechnung des exponentiell geglätteten Mittels
123   for(int i=0;i<50*n;i++){
124     mpu.getRotation(&w[0], &w[1], &w[2]);
125     for(int j=0;j<3;j++) w_off[j]=0.999*w_off[j]+0.001*float(w[j]);
```

```
126     delay(1);
127   }
128 }
129 /******Berechnung der Werte für die Rauschunterdrückung********/
130 void minmax(){
131   LCD.clear();
132   LCD.setCursor(0,0);
133   LCD.print("Kalibrierung...");
134   int tt=millis();
135   mpu.getRotation(&w[0], &w[1], &w[2]);
136   for(int i=0;i<3;i++){
137     MIN[i]=w[i]-w_off[i];
138     MAX[i]=w[i]-w_off[i];
139   }
140   //Messe für 5s den Rauschbereich --> MAX, MIN
141   while((millis()-tt)<5000){
142     mpu.getRotation(&w[0], &w[1], &w[2]);
143     for(int i=0;i<3;i++){
144       if(MIN[i]>(w[i]-w_off[i])) MIN[i]=w[i]-w_off[i];
145       if(MAX[i]<(w[i]-w_off[i])) MAX[i]=w[i]-w_off[i];
146     }
147   }
148 }
149 /******Setze den Winkel auf 0********/
150 void offsetReset(){
151   for(int i=0;i<3;i++) phi[i]=0;
152 }
153 /***Ausgabe der wichtigsten Werte auf der seriellen Schnittstelle***/
154 void serielleAusgabe(){
155   Serial.print("v= ");
156   Serial.print(v[0]*3.6,1);
157   Serial.print(" km/h");
158   Serial.print("\t");
159   Serial.print("vm= ");
160   Serial.print(vm*3.6,1);
161   Serial.print(" km/h");
162   Serial.print("\t");
163   for(int i=0;i<3;i++){
164     Serial.print(phi[i],1);
165     Serial.print("\t");
166   }
167   Serial.println("");
168 }
169 /******Ausgabe der wichtigsten Werte am LC-Display********/
170 void lcdAusgabe(){
171   LCD.clear();
```

```
172   LCD.setCursor(0,0);
173   LCD.print("v:");
174   LCD.print(v[0]*3.6,1);
175   LCD.print(" km/h");
176   LCD.setCursor(0,1);
177   LCD.print("vm:");
178   LCD.print(vm*3.6,1);
179   LCD.print(" ");
180   LCD.print(phi[0],1);
181   LCD.print("grd");
182 }
183 /******Setup für die Tastenentprellung********/
184 void entprellSetup(){
185   entprell_abstand_pin.attach(abstand_pin);
186   entprell_abstand_pin.interval(5);
187   entprell_vm_pin.attach(vm_pin);
188   entprell_vm_pin.interval(5);
189 }
190 /******Berechnung der mittleren Geschwindigkeit********/
191 void berechneVm(){
192   entprell_vm_pin.update();
193   //bei fallender Flanke Winkel und Zeit speichern
194   if(entprell_vm_pin.fell()){
195     t0=t;
196     alpha_0=Pi*phi[0]/180;     //Umrechnung auf rad
197   }
198   //wenn die Flanke wieder steigt Winkel und Zeit merken und vm
      berechen
199   if(entprell_vm_pin.rose()){
200     t1=t;
201     alpha_1=Pi*phi[0]/180;     //Umrechnung auf rad
202     vm=abstand*(tan(alpha_0)-tan(alpha_1))*(1000000/float(t1-t0));
203   }
204 }
205 /******Justierung des Abstands mittels Poti und Taster********/
206 void checkAbstand(){
207   entprell_abstand_pin.update();
208   int zustand=entprell_abstand_pin.read();
209   while(!zustand){
210     abstand=float(map(analogRead(poti_pin),0,1023,0,250))/10; //von
      0-25m
211     LCD.clear();
212     LCD.setCursor(0,0);
213     LCD.print("Abstand: ");
214     LCD.print(abstand,1);
215     LCD.print("m");
```

```
216     delay(50); //Gebe LC-Display Zeit zum schreiben
217     entprell_abstand_pin.update();
218     zustand=entprell_abstand_pin.read();
219   }
220 }
```

Code 4.5: Radar

5 Ausblick

Die Gestaltungsmöglichkeiten, die sich mit dem Arduino ergeben, sind beinahe grenzenlos. Daher ist es nicht Anspruch dieser Arbeit all diese erschöpfend zu behandeln, sondern einen geleiteten Einstieg in die elektronische Welt und deren physikalische Hintergründe zu bieten. Bei der Bearbeitung und Entwicklung der einzelnen Projekte wurde deutlich, dass es für den Schuleinsatz sinnvoll wäre, fächerübergreifend mit Informatik zu arbeiten, da der Programmieraufwand zwar auf ein Minimum reduziert wurde, jedoch noch immer recht beträchtlich ist. Falls Informatikräume der Schule genutzt werden sollen, empfiehlt es sich die Integration der benötigten externen Bibliotheken vom Systemadministrator durchführen zu lassen, um ein fehlerfreies Arbeiten an jedem Arbeitsplatz zu gewährleisten. Des Weiteren wäre es hilfreich, gewisse theoretische Überlegungen in die Mathematikstunde auszulagern. Dadurch wird die Vielschichtigkeit einzelner Probleme sichtbar und die etwas künstliche Trennung der einzelnen Unterrichtsfächer wieder aufgehoben. Der nächste Schritt ist nun der Einsatz im Unterricht: Je nach zeitlichem Rahmen und individueller Unterrichtsvorstellung können Teile der Arbeit im Klassenverband behandelt oder den Schülern als Nachschlagewerk überlassen werden. Das grundlegende Werkzeug wurde erarbeitet - jetzt sind Leser und Schüler gefordert, eigene Ideen umzusetzen.

Literatur

[1] Arduino. *Arduino*. 2015. URL: https://www.arduino.cc/ (abgerufen am 29. Sep. 2015).

[2] Erik Bartmann. *Die elektronische Welt mit Arduino entdecken*. 3. Aufl. Köln: O'Reilly, 2011.

[3] Herbert Bernstein. *Messelektronik und Sensoren*. 1. Aufl. München: Springer, 2014.

[4] Ekbert Hering. *Sensoren in Wissenschaft und Technik*. 1. Aufl. Wiesbaden: Vieweg+Teubner, 2012.

[5] Johannes Niebuhr und Gehrhard Lindner. *Physikalische Messtechnik mit Sensoren*. 3. Aufl. München: Oldenbourg, 1994.

[6] Frank Bernhard, Hrsg. *Handbuch der Technischen Temperaturmessung*. 1. Aufl. Berlin: Springer Vieweg, 2014.

[7] Horst Kuchling. *Taschenbuch der Physik*. 19. Aufl. München: Carl Hanser Verlag, 2007.

[8] Michael Margolis. *Arduino Kochbuch*. 1. Aufl. Köln: O'Reilly, 2015.

[9] Herbert Bernstein. *Elektrotechnik/Elektronik für Maschinenbauer*. 1. Aufl. Wiesbaden: Vieweg, 2004.

[10] Hans-Jürgen Gevatter und Ulrich Grünhaupt, Hrsg. *Handbuch der Mess- und Automatisierungstechnik in der Produktion*. Berlin Heidelberg: Springer, 2006.

[11] Erwin Böhmer, Dietmar Ehrhardt und Wolfgang Oberschelp. *Elemente der angewandten Elektronik*. 16. Aufl. Wiesbaden: Vieweg+Teubner, 2010.

[12] NXP. *I2C-bus specification and user manual*. 2014. URL: http://www.nxp.com/documents/user_manual/UM10204.pdf (abgerufen am 21. Sep. 2015).

[13] Klaus Dembowski. *Mikrocontroller - Der Leitfaden für Maker*. 1. Aufl. Heidelberg: dpunkt.verlag, 2014.

[14] Herbert Bernstein. *Mikrocontroller*. 1. Aufl. Wiesbaden: Springer Vieweg, 2015.

[15] Thomas Elbel. *Mikrosensorik*. 1. Aufl. Wiesbaden: Springer, 1996.

[16] Freidemann Völklein und Thomas Zetterer. *Praxiswissen Mikrosystemtechnik*. 2. Aufl. Wiesbaden: Vieweg, 2000.

[17] Bosch. *BMP180 Digital pressure sensor*. 2015. URL: https://ae-bst.resource.bosch.com/media/products/dokumente/bmp180/BST-BMP180-DS000-12~1.pdf (abgerufen am 5. Okt. 2015).

[18] Stefan Hesse und Gehrhard Schell. *Sensoren für die Prozess- und Fabrikautomation*. 6. Aufl. Wiesbaden: Springer Vieweg, 2014.

[19] AOSONG. *Temperature and humidity module AM2301 Product Manual*. URL: http://akizukidenshi.com/download/ds/aosong/AM2302.pdf (abgerufen am 1. Juni 2016).

[20] Hans-Rolf Tränkler, Hrsg. *Sensortechnik: Handbuch für Praxis und Wissenschaft*. 2. Aufl. Berlin Heidelberg: Springer Vieweg, 2014.

[21] InvenSense. *MPU-6000 and MPU-6050 Product Specification*. 2013. URL: http://43zrtwysvxb2gf29r5o0athu.wpengine.netdna-cdn.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf (abgerufen am 29. Jan. 2016).

[22] Jeff Rowberg. *i2cdevlib*. 2016. URL: https://github.com/jrowberg/i2cdevlib/archive/master.zip (abgerufen am 30. Jan. 2016).

[23] Oskar Höfling. *Physik*. 13. Aufl. Bonn: Dümmler, 1981.

[24] Benjamin Bahr, Jörg Resag und Kristin Riebe. *Faszinierende Physik*. Berlin Heidelberg: Springer Spektrum, 2013.

[25] Zentralanstalt für Meteorologie und Geodynamik. *IGRF Deklinationsrechner*. 2016. URL: http://www.zamg.ac.at/cms/de/geophysik/produkte-und-services-1/online-deklinationsrechner (abgerufen am 30. Apr. 2016).

[26] honeywell. *3-Axis Digital Compass IC HMC5883L*. 2013. URL: https://aerospace.honeywell.com/~/media/aerospace/files/datasheet/3-axisdigitalcompassichmc5883l_ds.pdf.

[27] Wilfried Plaßmann. *Handbuch Elektrotechnik*. Hrsg. von Detlef Schulz. 6. Aufl. Wiesbaden: Springer Vieweg, 2013.

[28] Andreas Binder. *Elektrische Maschinen und Antriebe*. Berlin Heidelberg: Springer, 2012.

[29] MARLIN P. JONES & ASSOC. INC. *HC-SR04 User Guide*. 2016. URL: http://www.mpja.com/download/HC-SR04.pdf (abgerufen am 13. Mai 2016).

[30] Wolfgang Demtröder. *Experimentalphysik 3*. 4. Aufl. Berlin Heidelberg: Springer, 2010.

[31] Ekbert Hering, Klaus Bressler und Jürgen Gutekunst. *Elektronik für Ingenieure und Naturwissenschaftler*. 6. Aufl. Berlin Heidlberg: Springer Vieweg, 2014.

[32] Helmut Kraus. *Die Atmosphäre der Erde: Eine Einführung in die Meteorologie*. 3. Aufl. Berlin Heidelberg: Springer, 2004.

[33] Institut für Physik der Universität Graz. *Meteorologische Station*. 2016. URL: http://physik.uni-graz.at/de/igam/forschen/mess-stationen/meteorologische-station/ (abgerufen am 2. Aug. 2016).

[34] Zentralanstalt für Meteorologie und Geodynamik. *Wetterlage vom 31. Juli 2016, 12 UTC*. 2016. URL: http://www.zamg.ac.at/cms/de/wetter/wetterkarte?tag=31&monat=07&jahr=2016&utc=12 (abgerufen am 2. Aug. 2016).

Abbildungen

Tabellen

Codes