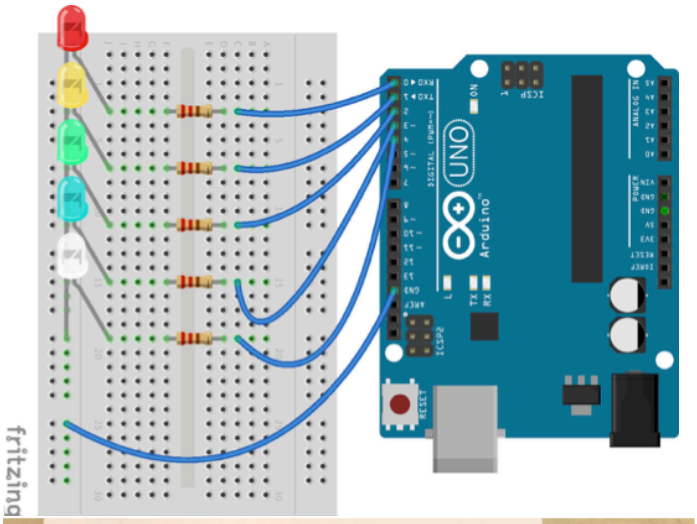
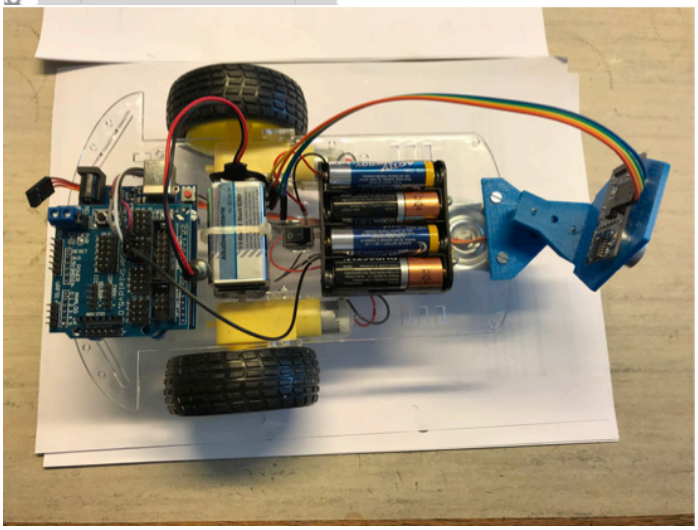




ARDUINO

Von der blinkenden LED bis zum autonomen Roboter

Unterrichtsmaterialien für Lehrerinnen und Lehrer
der Grundschulen (Klasse 4) und weiterführenden Schulen

Das aktuelle Skript und weitere Materialien können unter folgendem Link heruntergeladen werden:
https://bscw.sfz-bw.de:444/pub/bscw.cgi/325122

EXPOSEE

Wir tauchen Schritt für Schritt in die faszinierende Welt des Arduinos ein. Zahlreiche Experimente, Arbeitsblätter und Lösungsvorschläge unterstützen die Einführung.

Rudolf und Marita Lehn, Margret Tomczyk
Sandra Schiele – Korrektur und Gestaltung
Dr. Michael Lehn – fachliche Beratung

Lehrerfortbildung im SFZ Bad Saulgau

Editorial

Im Schülerforschungszentrum Südwürttemberg (SFZ) hat die Entwicklung von Materialien für den Unterrichtsalltag und für außerunterrichtliche Projekte eine lange Tradition. Besonders die Grundschulabteilung im SFZ sah es als eine wichtige Aufgabe an, Handreichungen für Lehrerinnen und Lehrer herzustellen nach dem Motto „von Praktikern für Praktiker". In den letzten Jahren entstand ein fulminanter Hype um Arduino, einem kleinen informationstechnischen Allrounder. Auch technisch weniger versierte Anwender erlangen mit Arduino den Zugang zur Programmierung und zu elektronischen Experimenten. Im SFZ ist für weiterführende Schulen ein Arduino-Skript[1] entwickelt worden, das sich großer Beliebtheit erfreut. Mit dem hier vorliegenden Skript soll der Arduino Eingang in die Grundschule und die Unterstufe weiterführender Schulen finden. Der Titel des Skripts „Von der blinkenden LED bis zum autonomen Roboter" macht aber deutlich, dass mit diesem Skript durchaus auch anspruchsvolle Programmierung und Experimente durchgeführt werden können.

Das vorliegende Skript ist aus mehreren Workshops mit Kindern aus der 4. Klasse in Grundschulen sowie Kindern aus 5. und 6. Klassen weiterführender Schulen entstanden. Zu Beginn war es für uns völlig ungewiss, ob Kinder diesen Alters bereits über die notwendige Abstraktionsfähigkeit sowie praktische Fertigkeit verfügen, um die grundlegende Arduino-Welt zu verstehen. Die Grundschulkinder wurden zu Beginn des Workshops mit dem elektrischen Stromkreis vertraut gemacht. Das Glühlämpchen wurde durch eine LED ersetzt. Damit konnte die Batterie durch den Arduino ersetzt werden und das Abenteuer „Von der blinkenden LED zum autonomen Roboter" nahm seinen Lauf. Die Begeisterung der Kinder war außergewöhnlich. Aus den vorgesehenen drei Workshop-Terminen wurden 10 Termine und jedes Kind wollte letztlich auch einen autonomen Roboter selber zusammenbauen und programmieren. Auch wir wurden von diesem Workshop mitgerissen und kamen zu der Überzeugung, dass der Arduino auch einen Platz in der Grundschule und in der Unterstufe weiterführender Schulen haben sollte.

Wir haben uns bei dem Skript bewusst an den Erfahrungen unserer Grundschulabteilung orientiert. Jedes Thema erfährt eine fachlich korrekte Einführung, die auch für interessierte Laien verständlich sein soll. Es schließen sich Arbeitsblätter an, welche die wesentlichen Inhalte der Themen enthalten. Außerdem sind ausführliche Lösungsvorschläge hinzugefügt. Wir sind davon überzeugt, dass mit diesem Skript für verschiedene Unterrichtseinheiten und vor allem für Projekte außerhalb des Regelunterrichts Themen ausgewählt und zuverlässig zusammen mit den Schülern bearbeitet werden können. Die Arbeitsblätter müssen nicht notwendigerweise an Schüler verteilt werden. Sie können auch nur Teil der Unterrichts- bzw. Projektvorbereitung sein.

Nobody is perfect. Wir haben das Skript mit Sorgfalt erstellt und alle Experimente und Programme zusammen mit Kindern getestet. Dennoch wird sich der eine oder andere Fehler eingeschlichen haben. Für Hinweise darauf sind wir sehr dankbar: rudolf.lehn@sfz-bw.de

Rudolf und Marita Lehn
Margret Tomczyk
Dr. Michael Lehn
Sandra Schiele
(v. l. n. r)

[1] https://sfz-bw.de/arduino-skript/

Nutzungshinweis

Dieses Skript kann frei heruntergeladen werden über die SFZ-Webseite (SFZ-Standort Bad Saulgau).
Sie können das Skript für Ihre persönlichen Zwecke unter Beachtung der gesetzlichen Bestimmungen des Urheberrechts vervielfältigen und weitergeben (CC BY-NC-SA 4.0)[2]. Bei der Entnahme von einzelnen Teilen, bitten wir Sie, auf dieses Skript als Quelle zu verweisen.

Bitte beachten Sie:
Das Skript enthält teilweise urheberrechtlich geschützte Abbildungen und Grafiken, deren Rechteinhaber nicht das SFZ ist. Diese Inhalte sind als solche, unter Angabe der Quelle und der entsprechenden Lizenz, gekennzeichnet. Bei deren Verwendung sind die jeweiligen Nutzungsrechte zu berücksichtigen.

Viele Abbildungen zu Arduino-Experimenten wurden in diesem Skript mit der Fritzing App (Version 0.3.9b vom 2. Juni 2016) erstellt. Dies ist eine freie Software, mit der elektronische Schaltungen am Computer erstellt werden können. Sie kann kostenfrei auf fritzing.org heruntergeladen werden.

[2] Weitere Hinweise zum Copyright https://creativecommons.org/licenses/by-nc-sa/4.0/

Inhaltsverzeichnis

1 Vorkenntnisse aus der Elektrik 7
1.1 Elektrische Grundlagen 8
1.2 AB1: LED-Experimente 10
1.3 ABL1: LED-Experimente 11

2 Arduino 13

3 Blinkende LED 14
3.1 AB 2: Blinkende LED 16
3.2 ABL 2: Blinkende LED - Lösung 17

4 Lauflicht mit LEDs 18
4.1 AB 3a: Lauflicht 19
4.2 ABL 3a: Lauflicht - Lösung 20
4.3 AB 3b: Lauflicht – Programm mit „for-Schleife“ 21
4.4 AB 4: Ampelsteuerung 22
4.5 ABL 4: Ampelsteuerung – Lösung 23

5 Messen mit dem Multimeter 24
5.1 AB 5: Spannungen in der Reihenschaltung 25
5.2 ABL 5: Spannungen in der Reihenschaltung - Lösung 27

6 Analog-Digital-Wandler im Arduino 29
6.1 AB 6: Ausgabe der AD-Werte 30
6.2 ABL 6: Ausgabe der AD-Werte - Lösung 32
6.3 AB 7: LED bei Dunkelheit einschalten 33
6.4 ABL 7: LED bei Dunkelheit einschalten - Lösung 35

7 Bewegungsmelder 37
7.1 AB 8: Alarmanlage 38
7.2 ABL 8: Alarmanlage - Lösung 40

8 Entfernungsmessung mit Ultraschall 42
8.1 AB 9: Entfernungsmesser mit Ultraschall 44
8.2 ABL 9: Entfernungsmesser mit Ultraschall - Lösung 46

9 Steuerung eines Gleichstrommotors 48
9.1 AB 10: Richtungswechsel und Geschwindigkeitsregelung 51
9.2 ABL 10: Richtungswechsel und Geschwindigkeitsregelung - Lösung 53

10 Steuerung eines Servomotors 55
10.1 AB 11: Servo-Steuerung 57
10.2 ABL 11: Servo-Steuerung - Lösung 59

11 Steuerung eines autonomen Roboters 65

12 Test der einzelnen Komponenten des autonomen Roboters 71
12.1 Roboter vorwärts fahren 71
12.2 Roboter rechts drehen 73
12.3 Roboter links drehen 74
12.4 Roboter macht eine Kehrtwende 75
12.5 Programmbeispiel für den Einsatz des autonomen Roboters 77

13 Arduino-Glossar 82
13.1 IT Begriffe 82
13.2 Allgemeines zur Programmierung 82
13.3 C++-Programmierung 83
13.3.1 Variablen 83
13.3.2 Arithmetische Ausdrücke und Zuweisungen 86
13.3.3 Kontrollstrukturen und logische Operatoren 87
13.3.4 Funktionen 90
13.4 Arduino-Programmierung 90
13.5 Fehlermeldungen 91
13.5.1 Syntaxfehler 91
13.5.2 Logikfehler 93
13.5.3 Hardwarefehler 94

14 Anhang 94
14.1 Physik der Diode und der LED 94
14.2 Aufbau und Kennzeichen der LED 98

1 Vorkenntnisse aus der Elektrik

Im Arduino-Workshop wird außer den 5 V-Spannungen des Arduino für verschiedene Projekte noch eine 9 V-Batterie eingesetzt. Mit einem DC-DC-Schaltregler können passende Spannungen, z. B. 5 V oder 6 V für Motoren eingestellt werden. Zur Wiederholung der Elektrik-Grundlagen kann auch auf die im Schulbereich verbreitete 4,5 V-Flachbatterie zurückgegriffen werden.

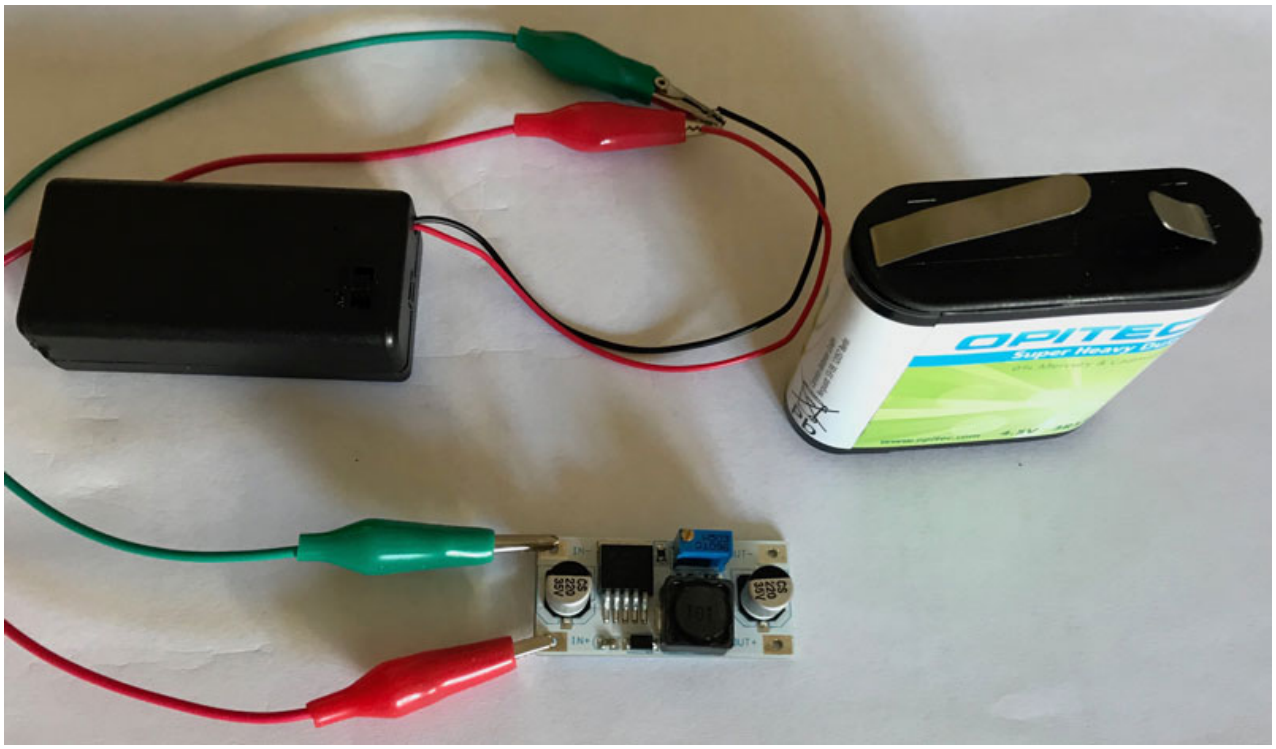


Abbildung 1: Spannungsquellen

Um übersichtliche und einfache Schaltungen aufbauen zu können, verwenden wir so oft wie möglich ein Breadboard (Steckplatine). Die farbig dargestellten Steckplätze sind leitend verbunden.

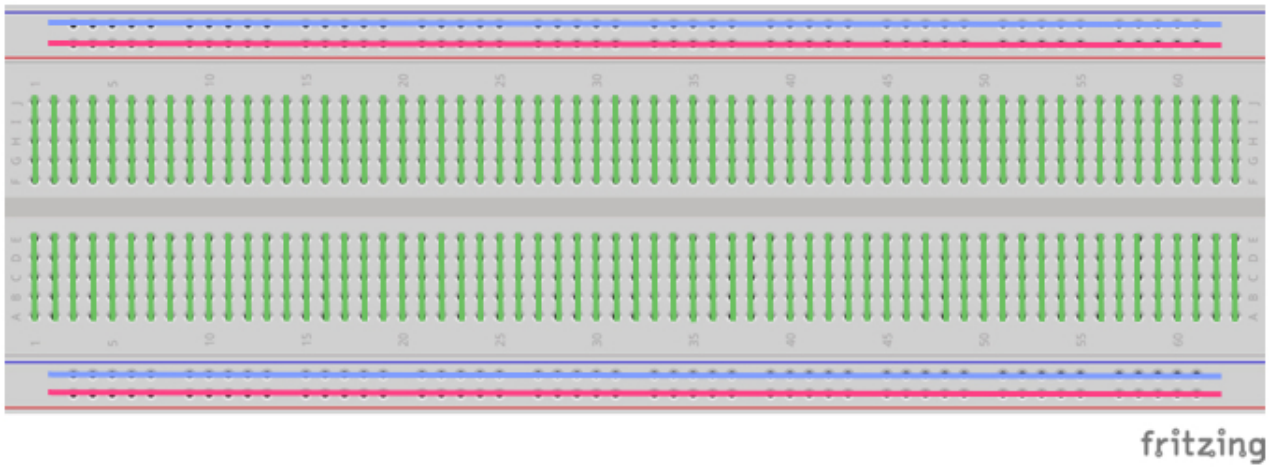


Abbildung 2: Breadboardverbindungen[3]

[3] Diese Abbildung wurde erstellt mit Fritzing (Lizenz: CC BY-SA 3.0)

1.1 Elektrische Grundlagen

Bei einer Leuchtdiode (engl. Light Emitting Diode, LED) ist ein Beinchen länger als das andere. Blickt man auf die Kunststofflinse, erkennt man die größere Trägerplatte des LED-Chips. An dieser Seite ist die Kathode der LED. Die kleinere gegenüberliegende Elektrode heißt Anode. Anders als bei der Glühbirne muss man darauf achten, dass der Strom die LED nur dann durchfließen kann, wenn der **Pluspol an die Anode** (A) und der **Minuspol an die Kathode** (K) angeschlossen werden.

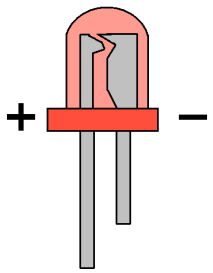


Abbildung 3: Aufbau einer LED[4]

Spannung: Auf Batterien wird die Spannung angegeben, z. B. 4,5 V, 9 V, 12 V, ... Damit wird die Energie pro Ladung ausgedrückt. Die Einheit ist 1 Volt (1 V).

Stromstärke: Je größer in einem Stromkreis die angelegte Spannung ist, desto größer ist die Stromstärke im Stromkreis. Die Einheit ist 1 Ampere (1 A).

Widerstand: Je größer der Widerstand in einem Stromkreis ist, desto kleiner wird bei gleichbleibender Spannung die Stromstärke. Bei kleinem Widerstand wird die Stromstärke bei gleichbleibender Spannung größer. Die Einheit des Widerstandes ist 1 Ohm (1 Ω).
Diesen Zusammenhang kann man durch das Ohmsche Gesetz ausdrücken: $I = {}^{U}/_{R}$

Die Standard-LEDs haben einen Durchmesser von 5 mm. Bei einer Stromstärke von etwa 10 mA leuchten sie in der Regel normal hell. Bei einer Stromstärke von 10 mA beträgt der Spannungsabfall an der LED je nach Farbe etwa 1,5 V bis 3 V. **Da die LEDs als Halbleiter im Betrieb ihren Zustand stark verändern können, sollten sie nur mit einem Vorwiderstand betrieben werden.** Der Vorwiderstand kann sowohl vor als auch hinter der LED eingebaut werden.

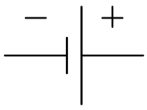
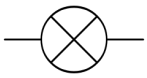
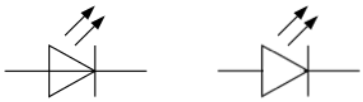
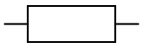
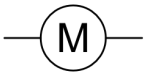
Typische Werte für LED-Spannungen:

LED-Farbe	LED-Spannung
rot	1,2 V ... 1,8 V
gelb, grün	1,9 V ... 2,5 V
blau, weiß	3 V ... 4 V

Setzen wir eine Spannungsquelle von 4,5 V oder 5 V ein und rechnen mit einer Stromstärke von 10 mA, so kann mit einem Vorwiderstand R = 200 Ω ein Spannungsabfall von 2 V am Vorwiderstand erreicht werden. Damit leuchten die LEDs aller Farben mehr oder weniger zufriedenstellend. Zur Physik der LED wird im Anhang (14.1) ein Modell erläutert.

[4] „Anode and cathode of LEDs. LEDの極性" von Adam850 via commons.wikimedia.org, Lizenz: Public domain, https://commons.wikimedia.org/wiki/File:%2B-_of_Led.png (14.08.2018)

Schaltsymbole im Überblick:

Bauteil	Schaltsymbol
Kabel[5] 	
Spannungs-quelle (Batterie[6]) 	 Das Symbol gilt für alle Gleichstromquellen. Der lange Strich ist der Pluspol.
Glühlampe[7] 	
Schalter[8] 	 Das Symbol gilt für alle Vorrichtungen, mit denen ein Stromkreis unterbrochen werden kann.
Leuchtdiode (LED[9]) 	
Widerstand[10]	
Motor[11] 	

[5] „cables-1080555_1280“ von jarmoluk via pixabay.com, Lizenz: CC0
https://cdn.pixabay.com/photo/2015/12/07/10/49/cables-1080555_1280.jpg (15.08.2018)
[6] „Bateria3R12“ von Julo via commons.wikimedia.org, Lizenz: Gemeinfrei
https://upload.wikimedia.org/wikipedia/commons/thumb/2/24/Bateria3R12.jpg/193px-Bateria3R12.jpg (15.08.2018)
[7] „light-bulb-2542155_1280“ von Janson_G via pixabay.com, Lizenz: CC0
https://cdn.pixabay.com/photo/2017/07/26/16/10/light-bulb-2542155_1280.png (15.08.2018)
[8] „switch-97637_640“ von OpenIcons via pixabay.com, Lizenz: CC0
https://cdn.pixabay.com/photo/2013/03/29/13/40/switch-97637_640.png (15.08.2018)
[9] „led-26354_640“ von Clker-Free-Vector-Images via pixabay.com, Lizenz: CC0
https://cdn.pixabay.com/photo/2012/04/10/16/50/led-26354_640.png (15.08.2018)
[10] „resistor-32290_640“ von Clker-Free-Vector-Images via pixabay.com, Lizenz: CC0
https://cdn.pixabay.com/photo/2012/04/13/12/55/resistor-32290_640.png (15.08.2018)
[11] „DC_Motor“ von Dcaldero8983 via comons.wikimedia.org, Lizenz: CC BY-SA 3.0
https://upload.wikimedia.org/wikipedia/commons/f/f4/DC_Motor.jpg (15.08.2018)

1.2 AB1: LED-Experimente

Materialien:

- 1 Spannungsquelle 4,5 V[12]
- Breadboard
- 1 blaue LED
- 2 rote LED
- 2 grüne LED
- 1 gelbe LED
- 2 Kroko-Kabel
- kurze Breadboardkabel
- 6 Widerstände (≈200 Ω)

Aufgabe 1:

a) Fixiere eine blaue LED im Breadboard, so dass sie nach Verbindung mit der Spannungsquelle leuchtet. Skizziere die Schaltung.

b) Ergänze die Schaltung mit der blauen LED durch einen Vorwiderstand und schließe den Stromkreis. Skizziere die Schaltung.

Schaltpläne für Aufgabe 1:

a)	b)

Aufgabe 2:

Auf dem Breadboard sind 6 verschiedene Schaltungen einer LED mit einem Vorwiderstand dargestellt.

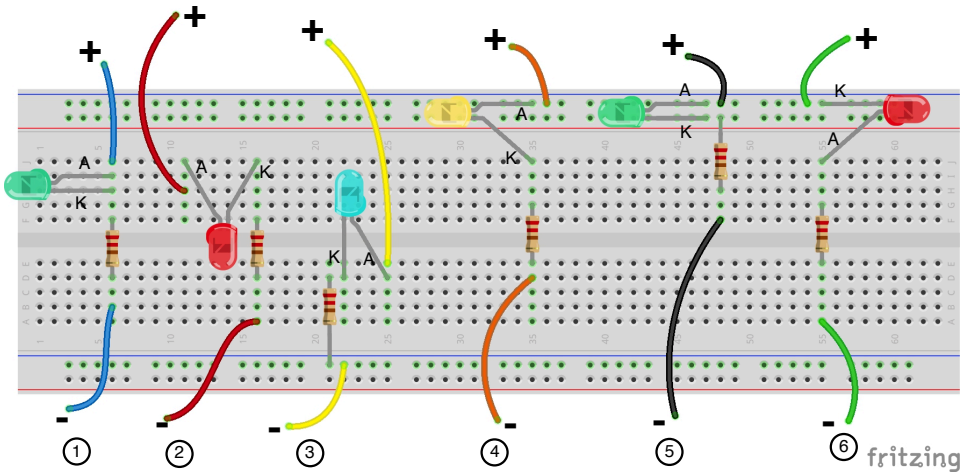


Abbildung 4: Verschiedene Schaltungen einer LED mit einem Vorwiderstand.[13]

a) Welche LED leuchtet? Kreuze diese Nummern an und überprüfe deine Vermutung.

b) Begründe, warum einige der LEDs nicht leuchten.

c) Entferne alle LEDs mit Vorwiderstand, welche nicht leuchten. Schalte danach die restlichen LEDs mit Vorwiderstand parallel. Erstelle dazu auch einen Schaltplan.

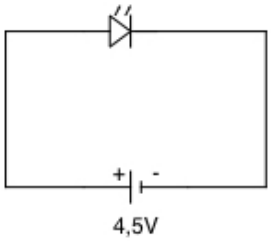
Schaltplan für Aufgabe 2:

[12] Flachbatterie oder 9V-Batterie in einer Batteriebox und einem DC-DC-Adapter
[13] Diese Abbildung wurde erstellt mit Fritzing (Lizenz: CC BY-SA 3.0)

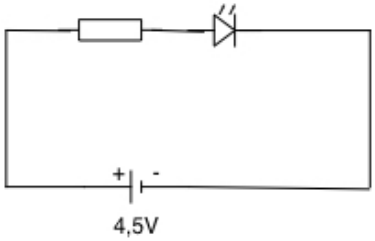
1.3 ABL1: LED-Experimente

Aufgabe 1:

a) Möglicher Schaltplan (LED ohne Vorwiderstand):



b) Möglicher Schaltplan (LED mit Vorwiderstand)



Möglicher Aufbau der Schaltungen auf dem Breadboard:

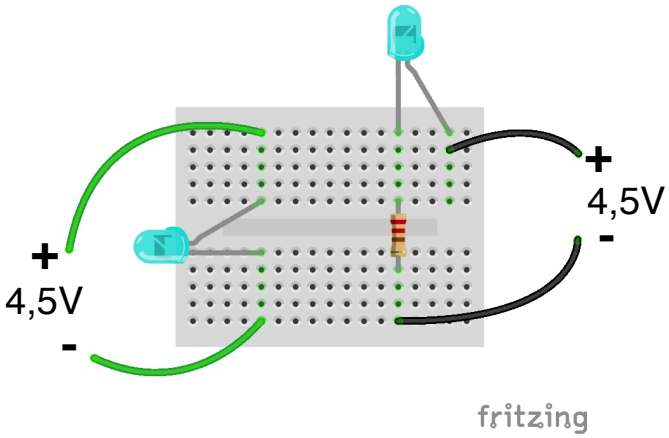


Abbildung 5: Blaue LED ohne und mit Vorwiderstand[14]

[14] Diese Abbildung wurde erstellt mit Fritzing (Lizenz: CC BY-SA 3.0)

Aufgabe 2:

a) Es leuchten die LEDs mit den Nummern 2, 4 und 5.

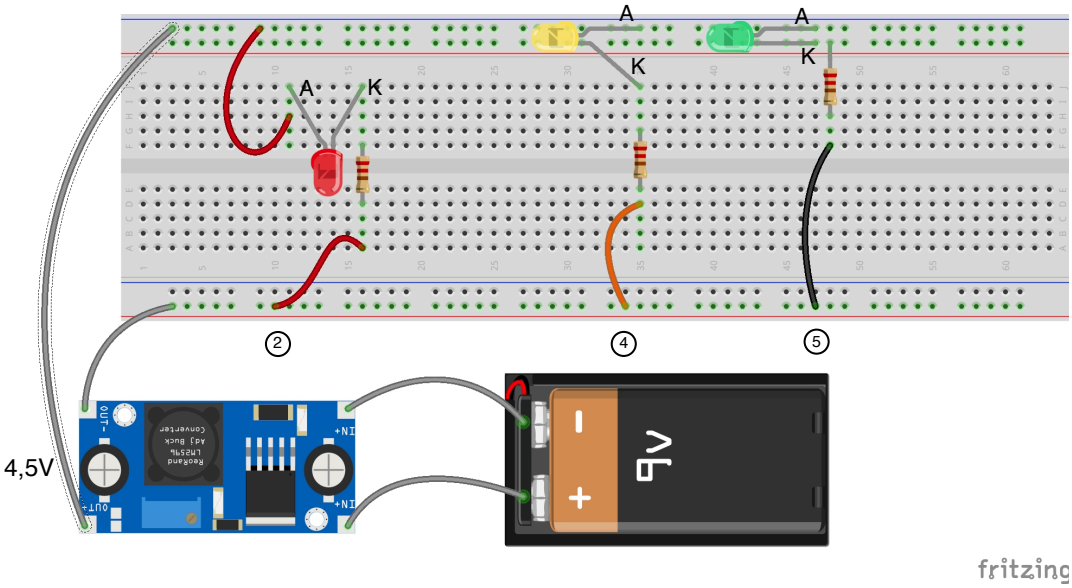


Abbildung 6: Leuchtende LEDs[15]

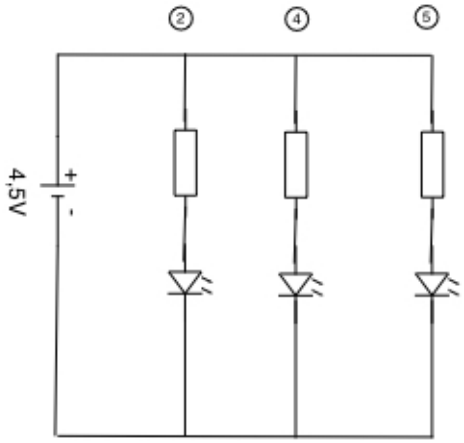
b) <u>Fehlerbegründungen:</u>

① Kurzschluss: Die beiden Beinchen der grünen LED sind verbunden.
Somit fließt der Strom nicht durch die LED.

③ Zwischen der Kathode der grünen LED und dem Widerstand besteht keine Verbindung.
Der Stromkreis ist unterbrochen.

⑥ Stromrichtung: Der Pluspol ist mit der Kathode (kurzes Beinchen) verbunden.

c) <u>Möglicher Schaltplan für die LEDs mit Vorwiderstand, in Parallelschaltung:</u>



[15] Diese Abbildung wurde erstellt mit Fritzing (Lizenz: CC BY-SA 3.0)

2 Arduino

Der Arduino ist eine Platine mit einem Mikrocontroller (Abbildung 7). Der Mikrocontroller kann sogenannte Ausgabepins (Pinzustand: OUTPUT) ansteuern, indem er entweder eine Spannung von 5 V anlegt (HIGH) oder keine Spannung d. h. 0 V anlegt (LOW). Dadurch können an die Pins angeschlossene Geräte, wie bspw. eine LED, ein- oder ausgeschalten werden. An Eingabepins kann eine Spannung gemessen werden, damit können z. B. Sensordaten eines Entfernungsmessers ausgewertet werden.

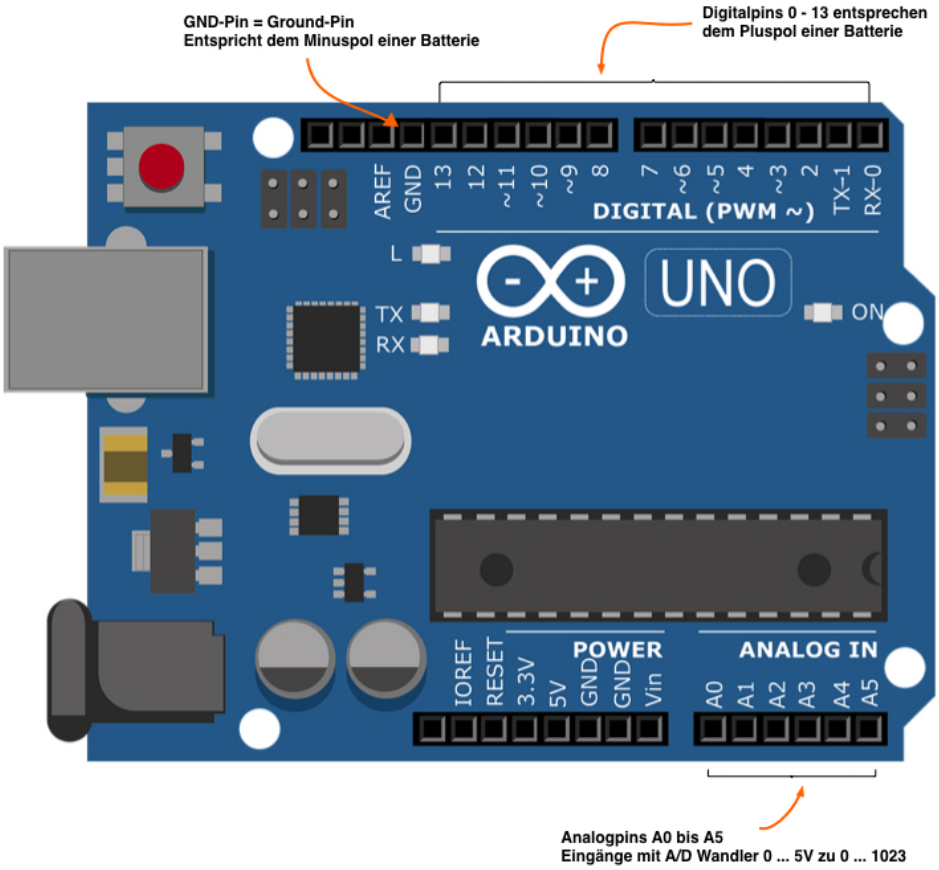


Abbildung 7: Arduino UNO - Board[16]

Wird der Arduino mit dem USB-Port des Computers verbunden, kann er mit der Arduino Software (IDE) programmiert werden (kostenfreier Download unter https://www.arduino.cc). In dieser Entwicklungsumgebung für die Arduino-Programmierung wird der Programmcode geschrieben. Nach dem Start der Arduino IDE erscheint in der Entwicklungsumgebung standardmäßig ein Code mit den beiden Funktionen `setup` und `loop`. Die Programmierung auf der Arduino IDE basiert auf der Programmiersprache C/C++.

```
void setup() {
  // put your setup code here, to run once:
}

void loop() {
  // put your main code here, to run repeatedly:
}
```

Mit `//` werden Kommentar-Zeilen eingefügt.

[16] „arduino-2168193_1280“ von Seven_au via pixabay.com, Lizenz: CC0 Creative Commons
https://cdn.pixabay.com/photo/2017/03/23/12/32/arduino-2168193_960_720.png (31.08.2018)

3 Blinkende LED

Wir wollen eine LED ein- und ausschalten. Dazu schließen wir die LED an Digitalpin 12 des Arduino an und teilen dem Arduino in der `setup`-Funktion durch den Befehl `pinMode(12,OUTPUT)` mit, dass er auf diesem Pin die Steuerung für die LED ausgeben soll. In der `loop`-Funktion soll der Arduino dann die LED immer wieder einschalten und ausschalten. Eingeschaltet wird die LED mit `digitalWrite(12,HIGH)` und ausgeschaltet mit `digitalWrite(12,LOW)`. Mit `HIGH` wird am Pin 12 die Spannung von 5V bereitgestellt und mit `LOW` wird die Spannung am Pin 12 wieder auf 0V zurückgesetzt. Mit dem Befehl `delay(1000)` kann eine Sekunde lang gewartet werden, bis die LED umgeschaltet wird.

So sieht dann das ganze Arduino-Programm aus:

```
void setup() {
  // put your setup code here, to run once:
  pinMode(12, OUTPUT); // Pin 12 mit OUTPUT initialisieren
}

void loop() {
  // put your main code here, to run repeatedly:
  digitalWrite(12, HIGH); // Pin 12 auf HIGH setzen
  delay(1000);            // Ablauf 1000 ms verzögern
  digitalWrite(12, LOW);  // Pin 12 auf LOW setzen
  delay(1000);            // Ablauf 1000 ms verzögern
}
```

Es ist guter Programmierstil, die Anweisungen zu kommentieren.
Nach der Erstellung des Programms muss es übersetzt und auf den Mikrocontroller des Arduino geladen werden. In der Toolbar (Werkzeugleiste) der IDE (Entwicklungsumgebung) können diese Optionen mit einem Mausklick gewählt werden (Abbildung 8). Außerdem werden wir aufgefordert, das Programm auf dem Rechner zu speichern.

Abbildung 8: Toolbarleiste der IDE[17]

Zusammen mit der Entwicklungsumgebung wird auch ein umfangreiches Pull-Down-Menü von Arduino bereitgestellt (Abbildung 9). Dieses Hauptmenü von Arduino enthält die Rubriken: Datei, Bearbeiten, Sketch, Werkzeuge, Hilfe. Beim Hochladen eines Programms auf den Mikrocontroller muss der Arduino den passenden Port der USB-Verbindung erhalten. Bei einer Fehlermeldung muss dieser Port über die Hauptmenü-Punkte „Werkzeuge“ – „Port“ ausgewählt werden.

[17] Screenshot aus der IDE (Arduino Version 1.8.3)

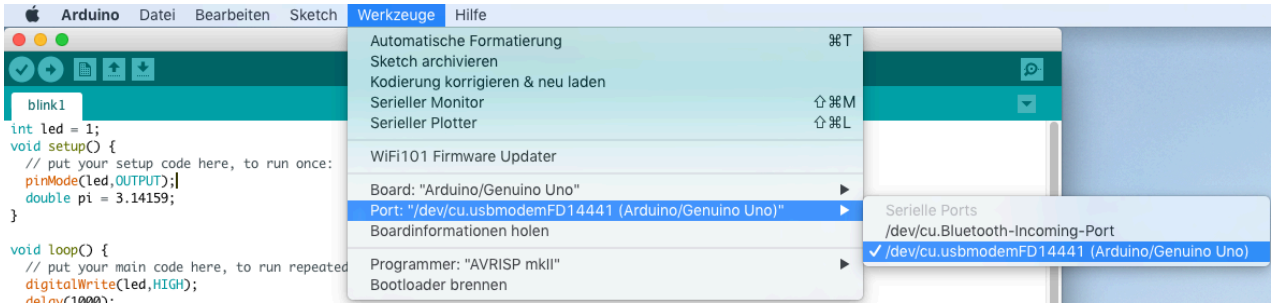


Abbildung 9: Auswahl des passenden Ports im Hauptmenü[18]

Weitere Optionen stehen zur Verfügung (Abbildung 10):

- „Neu“ führt zur Erstellung eines neuen Programms.
- „Öffnen“ kann zum Laden eines Programms vom Rechner verwendet werden.
- Mit „Speichern unter...“ kann das Programm unter einem neuen Namen auf dem Rechner abgespeichert werden.

Abbildung 10: Weitere Optionen aus der Toolbarleiste der IDE[19]

Mit dem folgenden Schaltplan kann auf dem Breadboard der Stromkreis aufgebaut werden (Abbildung 11). Am Pin 12 sind 5 V, wenn die LED leuchten soll, und 0 V, wenn die LED nicht leuchten soll.

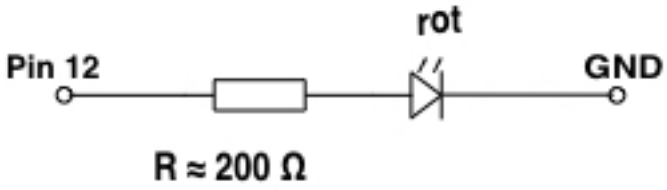


Abbildung 11: Schaltplan zur LED mit Vorwiderstand

[18] Screenshot aus dem Hauptmenü (Arduino Version 1.8.3)
[19] Screenshot aus der IDE (Arduino Version 1.8.3)

3.1 AB 2: Blinkende LED

Materialien:

- 1 Laptop mit Arduino-Software
- 1 Arduino
- 1 Breadboard
- 1 rote LED
- 1 Widerstand (≈ 200 Ω)
- verschiedene Breadboard Kabel für den Arduino

Aufgabe 1:

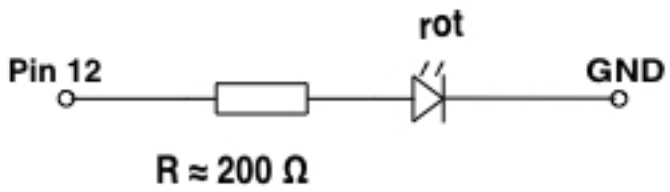
Schreibe auf dem Laptop das Arduino-Programm mit dem die LED blinkt
(1 Sekunde leuchten und 1 Sekunde ausgeschaltet).

Aufgabe 2:

a) Baue die Schaltung nach dem Schaltplan auf dem Breadboard.

b) Starte das Arduino-Programm.

c) Wenn das Arduino-Programm läuft, kannst du es so abändern, dass die LED verschieden schnell blinkt.



3.2 ABL 2: Blinkende LED - Lösung

Aufgabe 1:

Schreibe auf dem Laptop das Arduino-Programm mit dem die LED blinkt (1 Sekunde leuchten und 1 Sekunde ausgeschaltet).

▸ siehe Programmcode auf Seite 14

Aufgabe 2:

a) Mögliche Schaltung für die LED auf dem Breadboard:

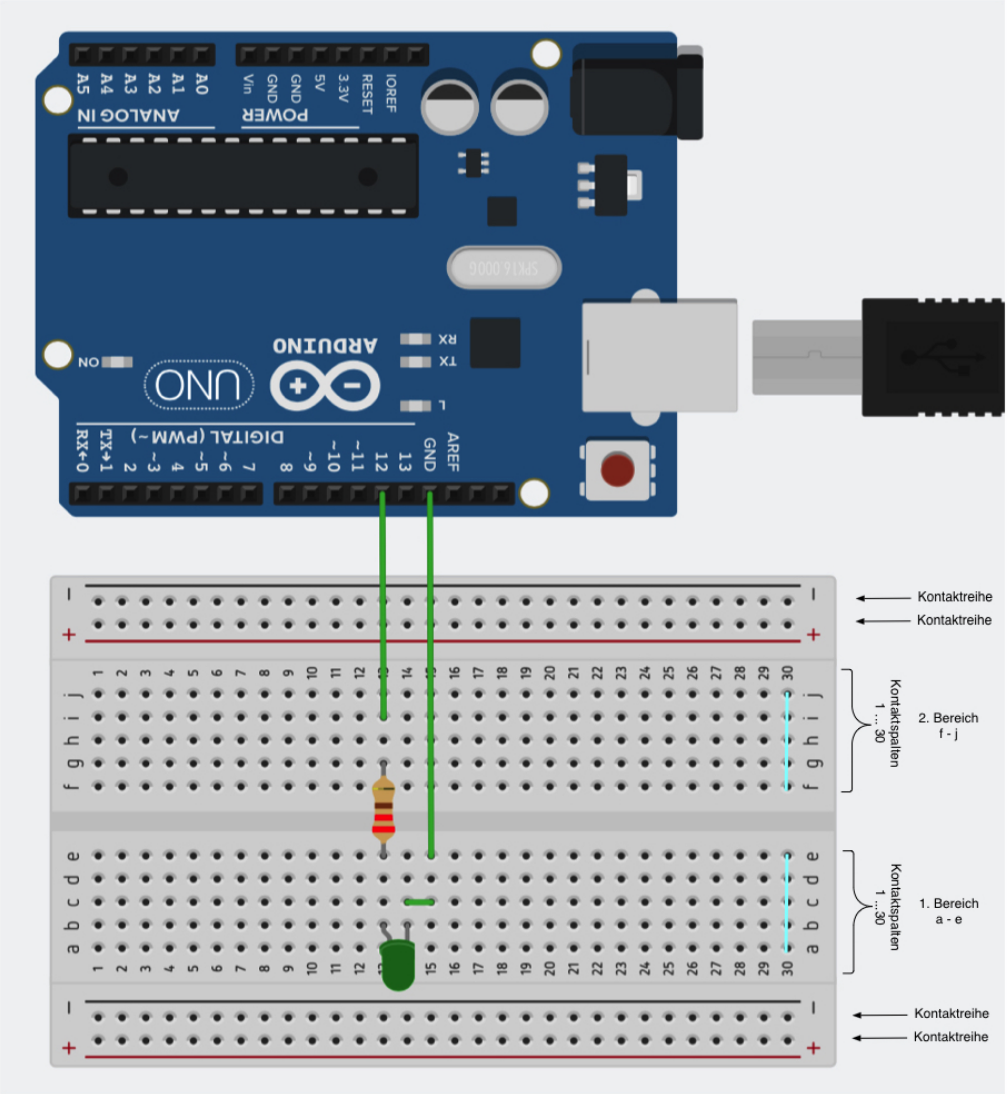


Abbildung 12: Mögliche Schaltung für eine blinkende LED[20]

Die horizontalen Reihen oben und unten sind durchgängig leitend verbunden. Sie eignen sich für Verbindungen zu GND (Minuspol) oder zum Pluspol. Die vertikalen Kontaktspalten sind in zwei Bereiche aufgeteilt und jeweils mit fünf Kontaktlöchern leitend verbunden.

[20] Diese Abbildung wurde erstellt mit Fritzing (Lizenz: CC BY-SA 3.0)

4 Lauflicht mit LEDs

Verschiedenfarbige LEDs sollen der Reihe nach eine vorgegebene Zeit leuchten. Wir verwenden eine rote, eine gelbe, eine grüne, eine blaue und eine weiße LED. Die rote LED wird an den Pin 0, die gelbe LED wird an den Pin 1, die grüne LED wird an den Pin 2, die blaue LED an den Pin 3 und die weiße LED an den Pin 4 angeschlossen. Jeder dieser Pins von 0 bis 4 muss in der `setup`-Funktion auf OUTPUT eingestellt werden.

In der `loop`-Funktion kannst du dann z. B. für die grüne LED an Pin 2 die folgenden drei Befehle eingeben:

```
digitalWrite(2, HIGH);
delay(500);
digitalWrite(2, LOW);
```

4.1 AB 3a: Lauflicht

Materialien:

- 1 Laptop mit Arduino-Software
- 1 Arduino
- 1 Breadboard
- 1 rote LED
- 1 gelbe LED
- 1 grüne LED
- 1 blaue LED
- 1 weiße LED
- 5 Widerstände (≈ 200 Ω)
- verschiedene Breadboard Kabel für den Arduino

Aufgaben:

a) Schreibe das Arduino-Programm, mit dem die LEDs der Reihe nach eingeschaltet und ausgeschaltet werden.

b) Baue auf dem Breadboard die Schaltung auf. Jede LED braucht einen Vorwiderstand von etwa 200 Ω. Überprüfe mit dem Multimeter die Widerstände.

c) Lade das Arduino-Programm auf den Mikrocontroller. Starte das Experiment.

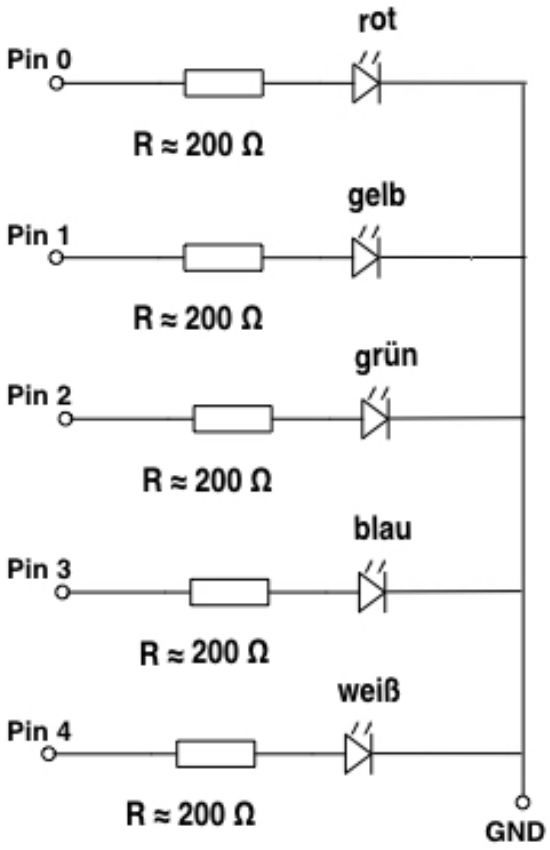


Abbildung 13: Schaltplan Lauflicht

4.2 ABL 3a: Lauflicht - Lösung

Aufgaben:

a) Mögliches Arduino-Programm für ein LED-Lauflicht:
In diesem Arduino-Programm leuchtet jede LED 300 ms lang.

```
void setup() {
  // put your setup code here, to run once:
  pinMode(0, OUTPUT); // rote LED
  pinMode(1, OUTPUT); // gelbe LED
  pinMode(2, OUTPUT); // grüne LED
  pinMode(3, OUTPUT); // blaue LED
  pinMode(4, OUTPUT); // weiße LED
}

void loop() {
  // put your main code here, to run repeatedly:
  digitalWrite(0, HIGH); delay(500); digitalWrite(0, LOW);
  digitalWrite(1, HIGH); delay(500); digitalWrite(1, LOW);
  digitalWrite(2, HIGH); delay(500); digitalWrite(2, LOW);
  digitalWrite(3, HIGH); delay(500); digitalWrite(3, LOW);
  digitalWrite(4, HIGH); delay(500); digitalWrite(4, LOW);
}
```

b) Möglicher experimenteller Aufbau des LED-Lauflichts**:**

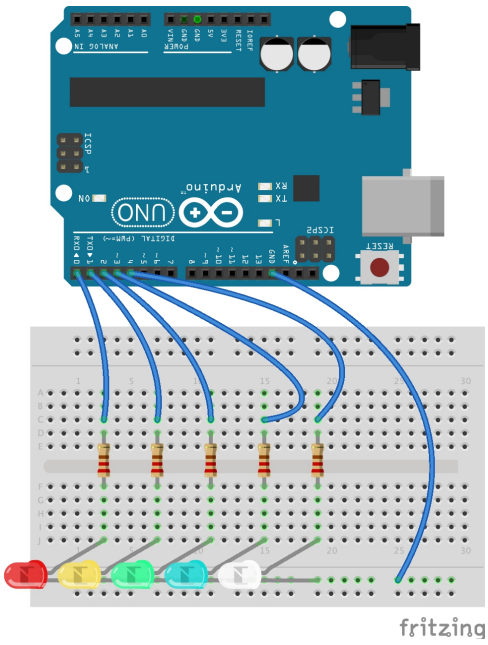


Abbildung 14: Möglicher Aufbau des LED-Lauflichts[21]

Wichtiger Hinweis:
Da Pin 0 (RX) und Pin 1 (TX) auch vom USB-Transfer abhängt, muss beim Hochladen der Stromkreis der Schaltung unterbrochen werden. Dazu kann z.B. die GND-Verbindung getrennt werden.

[21] Diese Abbildung wurde erstellt mit Fritzing (Lizenz: CC BY-SA 3.0)

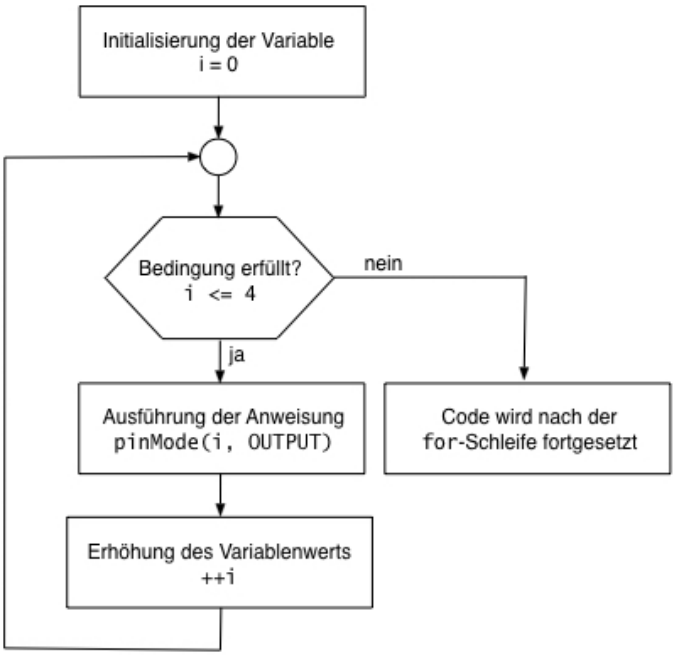
4.3 AB 3b: Lauflicht – Programm mit „for-Schleife“

Anstatt für jeden einzelnen Pin die Anweisungen zum Ein- und Ausschalten zu schreiben, kann man auch eine `for`-Schleife verwenden. Mit einer `for`-Schleife werden Anweisungen automatisch solange wiederholt ausgeführt, wie sie eine bestimmte Bedingung erfüllen. Die Bedingung steht in einer runden Klammer, die Anweisung in einer geschweiften Klammer.

Arudino-Programm mit `for`-Schleife:

```
int i;

void setup() {
  // put your setup code here, to run once:
  for (i = 0; i <= 4; ++i) {
    pinMode(i, OUTPUT);
  }
}

void loop() {
  // put your main code here, to run repeatedly:
  for (i = 0; i <= 4; ++i) {
    digitalWrite(i, HIGH); delay(500); digitalWrite(i, LOW);
  }
}
```

Erklärung der oberen `for`-Schleife:
Ziel dieser `for`-Schleife ist, für die Pins 0 bis 4 den `pinMode` auf OUTPUT zu stellen. Hierzu wird in der `for`-Schleife zunächst die Variable i für die Pin-Nummer festgelegt (sog. Zähler). Die Variable i wird zu Beginn mit dem Startwert i = 0 initialisiert. In der `for`-Schleife wird geprüft, ob die Variable die Bedingung `i <= 4` erfüllt, d. h. ob die Pin-Nummer kleiner gleich 4 ist. Falls diese Bedingung erfüllt ist, wird die Anweisung `pinMode(i, OUTPUT)` für diesen Pin ausgeführt und anschließend mit `++i` der Variablenwert um 1 erhöht. Die Schritte Prüfung der Bedingung, Ausführung der Anweisung und Erhöhung des Variablenwerts werden solange durchlaufen, bis die Bedingung `i <= 4` nicht mehr erfüllt ist, d. h. nachdem Pin 4 auf OUTPUT gesetzt wurde. Sobald die `for`-Schleife beendet ist, wird das Programm danach fortgesetzt.



Aufgabe:
Teste das Programm.

4.4 AB 4: Ampelsteuerung

Materialien:

- 1 Laptop mit Arduino-Software
- 1 Arduino
- 1 Breadboard
- 1 rote LED
- 1 gelbe LED
- 1 grüne LED
- 3 Widerstände (≈ 200 Ω)
- verschiedene Breadboard Kabel für den Arduino

Mit der folgenden Schaltung kann eine Ampelsteuerung gebaut werden.

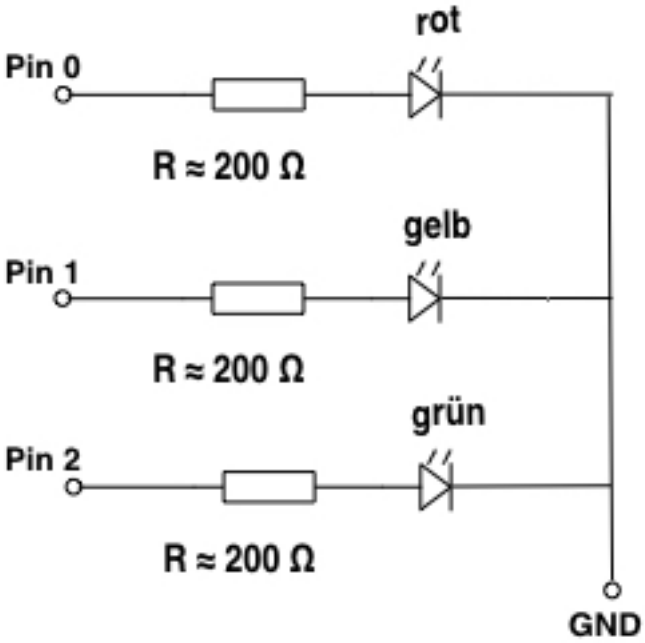


Abbildung 15: Mögliche Schaltung für eine Ampelsteuerung

Aufgaben:

a) Erstelle ein Arduino-Programm, bei dem das rote Licht der Ampel 10 s, das gelbe Licht der Ampel 1 s und das grüne Licht 5 s leuchtet. Die Ampel soll folgenden Ablauf haben:
   a. Die Ampel startet mit „rot“
   b. danach leuchten „rot“ und „gelb“
   c. danach leuchtet nur „grün“
   d. danach leuchtet nur „gelb“
   e. danach schaltet die Ampel wieder auf „rot“

b) Mache einen experimentellen Aufbau und teste die Ampel.

4.5 ABL 4: Ampelsteuerung – Lösung

Aufgaben:

a) Mögliches Arduino-Programm für eine Ampelsteuerung:

```
void setup() {
  // put your setup code here, to run once:
  pinMode(0, OUTPUT); //rot
  pinMode(1, OUTPUT); //gelb
  pinMode(2, OUTPUT); //grün
}

void loop() {
  // put your main code here, to run repeatedly:
  digitalWrite(0, HIGH);  delay(10000); //Ablauf 10 s verzögern
  digitalWrite(1, HIGH);  delay(1000);
  digitalWrite(0, LOW);   digitalWrite(1, LOW);
  digitalWrite(2, HIGH);  delay(5000);  //Ablauf 5 s verzögern
  digitalWrite(2, LOW);
  digitalWrite(1, HIGH);  delay(1000);
  digitalWrite(1, LOW);
}
```

Das Programm wird klarer, wenn für die Pins 0, 1, 2 Namen für die Ampelfarben eingesetzt werden. Da 0, 1, 2 ganze Zahlen sind, definieren wir die Platzhalter `rot, gelb, gruen` am Programmbeginn. Statt Platzhalter wird i.d.R. der Fachbegriff **Variable** verwendet. Beachte, dass im Arduino-Programm, das die C++ Programmiersprache verwendet, keine Umlaute in den Namen der Veriablen verwendet werden dürfen. In der setup-Funktion setzen wir noch `rot, gelb` und `gruen` jeweils auf LOW, d. h. die Ampel ist ausgeschaltet.

```
int rot = 0;
int gelb = 1;
int gruen = 2;

void setup() {
  pinMode(rot, OUTPUT);
  pinMode(gelb, OUTPUT);
  pinMode(gruen, OUTPUT);
  digitalWrite(rot, LOW);   // Ampel ausgeschaltet
  digitalWrite(gelb, LOW);
  digitalWrite(gruen, LOW);
}

void loop() {
  digitalWrite(rot, HIGH);   delay(10000); //Ablauf 10 s verzögern
  digitalWrite(gelb, HIGH);  delay(1000);
  digitalWrite(rot, LOW);    digitalWrite(gelb, LOW);
  digitalWrite(gruen, HIGH); delay(5000); //Ablauf 5 s verzögern
  digitalWrite(gruen, LOW);
  digitalWrite(gelb, HIGH);  delay(1000);
  digitalWrite(gelb, LOW);
}
```

5 Messen mit dem Multimeter

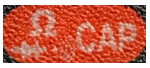
Abbildung 16: **Digitalmultimeter**

Ein Multimeter braucht man um die Größe von Spannungen oder von Widerständen in unseren Experimenten bestimmen zu können.

Um Spannungen zu messen, muss man den Drehschalter auf

einstellen.

Um Widerstände zu messen muss man den Drehschalter auf



einstellen.

Das Multimeter gibt die Spannungen und Widerstandswerte als Dezimalzahlen an.

5.0 V oder 4.5 V oder 2.45 V ...

200 Ω oder 9.5 kΩ oder 1.1 kΩ ...

Rechnen mit Dezimalzahlen

4.5 V bedeutet 4 V + 5/10 V.
An der ersten Stelle hinter dem Punkt sind die 1/10 V Werte.
4.49 V bedeutet 4 V + 4/10 V + 9/100 V.
An der zweiten Stelle hinter dem Punkt sind die 1/100 V Werte.

Bei den Widerstandswerten ist es ganz ähnlich.
9.5 kΩ bedeutet 9 kΩ + 5/10 kΩ.
9.19 kΩ bedeutet 9 kΩ + 1/10 kΩ + 9/100 kΩ.

Damit wir einen besseren Überblick haben, runden wir meistens auf eine Stelle hinter dem Punkt. 4.49 V ≈ 4.5 V oder 9.19 kΩ ≈ 9.2 kΩ

5.1 AB 5: Spannungen in der Reihenschaltung

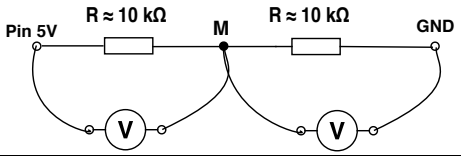
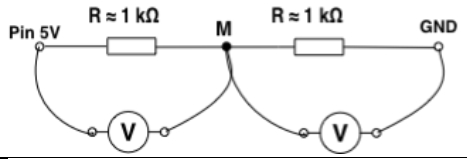
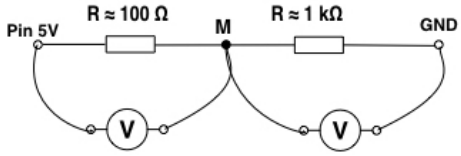
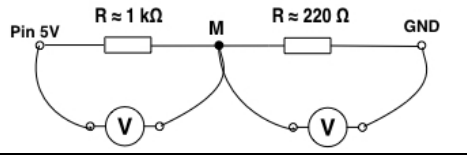
Materialien:

- 2 Widerstände 1 kΩ
- 2 Widerstände 10 kΩ
- 1 Widerstand 100 Ω
- 1 Widerstand 220 Ω
- 1 Fotowiderstand (LDR)
- 1 Arduino
- 1 Breadboard
- verschiedene Breadboard Kabel für den Arduino
- 1 Multimeter

Aufgabe 1:

Baue nach den Schaltbildern in der linken Spalte die zugehörigen Schaltungen auf dem Breadboard auf. Miss dann jeweils folgende Spannungen und trage die Werte auf der rechten Seite der Tabelle ein:

- zwischen dem Pin 5 V[22] und dem Pin GND
- zwischen dem Pin 5 V und der Stelle M
- zwischen der Stelle M und dem Pin GND

	$U_{5V\text{-}GND}$	$U_{5V\text{-}M}$	$U_{M\text{-}GND}$
	$U_{5V\text{-}GND}$	$U_{5V\text{-}M}$	$U_{M\text{-}GND}$
	$U_{5V\text{-}GND}$	$U_{5V\text{-}M}$	$U_{M\text{-}GND}$
	$U_{5V\text{-}GND}$	$U_{5V\text{-}M}$	$U_{M\text{-}GND}$

- Wenn man zwischen Pin 5 V und GND zwei gleichgroße Widerstände schaltet, wird die Spannung von 5 V in zwei _____ Teile aufgeteilt.
- Wenn der Widerstand zwischen dem Pin 5 V und der Mitte M kleiner ist als der Widerstand zwischen M und dem Pin GND, dann ist die Spannung zwischen M und GND _____ als zwischen dem Pin 5 V und M.
- Wenn der Widerstand zwischen dem Pin 5 V und der Mitte M größer ist als der Widerstand zwischen M und dem Pin GND, dann ist die Spannung zwischen M und GND _____ als zwischen dem Pin 5 V und M.

[22] Dieser Pin 5 V wird verwendet, wenn konstant die Spannung 5 V angelegt wird.

Aufgabe 2:

Wir verwenden jetzt einen LDR-Widerstand, der bei Dunkelheit seinen größten und bei Helligkeit seinen kleinsten Widerstand hat. LDR ist die Abkürzung für Light Dependant Resistor (lichtabhängiger Widerstand).

Baue nach dem folgenden Schaltbild die zugehörige Schaltung auf dem Breadboard auf und miss mit dem Multimeter bei Dunkelheit und bei Helligkeit jeweils die Spannung

- zwischen dem Pin 5 V und dem Pin GND
- zwischen dem Pin 5 V und der Stelle M
- zwischen der Stelle M und dem Pin GND

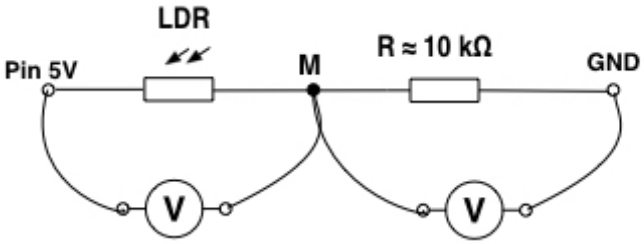
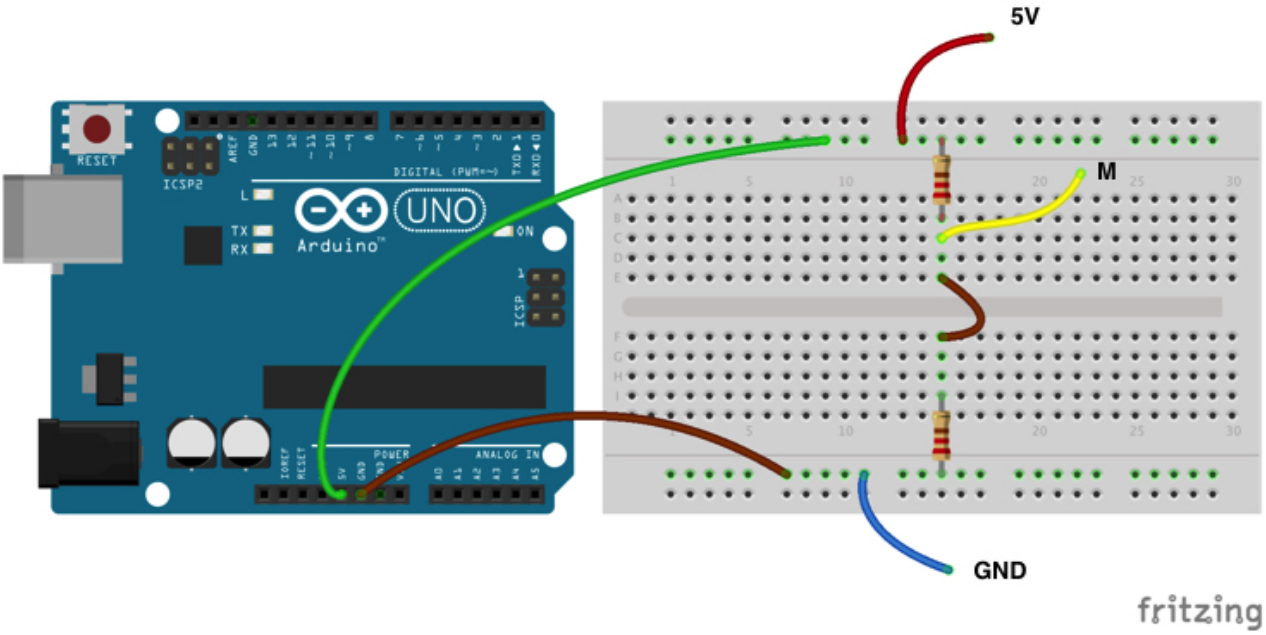


Abbildung 17: Schaltplan zu LDR mit Vorwiderstand

Bei Dunkelheit			Bei Helligkeit		
$U_{5V\text{-}GND}$	$U_{5V\text{-}M}$	$U_{M\text{-}GND}$	$U_{5V\text{-}GND}$	$U_{5V\text{-}M}$	$U_{M\text{-}GND}$

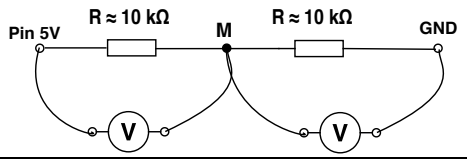
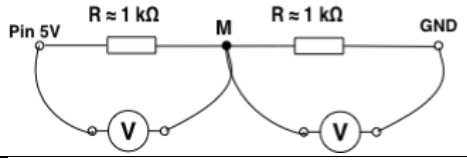
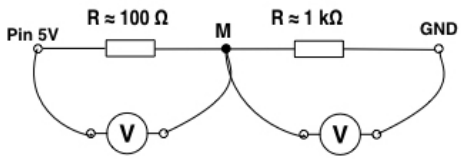
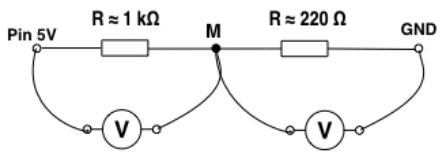
Hinweis:
Zur Bestimmung der Spannungswerte mit dem Multimeter ist es sinnvoll, kurze Breadboardkabel einzufügen wie in der folgenden Skizze dargestellt.



5.2 ABL 5: Spannungen in der Reihenschaltung - Lösung

Aufgabe 1:

Mögliche Messwerte:

	U_{5V-GND}	U_{5V-M}	U_{M-GND}
	5 V	2.5 V	2.5 V
	5 V	2.5 V	2.5 V
	5 V	0.5 V	4.5 V
	5 V	4.1 V	0.9 V

Die Messwerte für U_{5V-M} und U_{M-GND} können leicht variieren.

- Wenn man zwischen Pin 5 V und GND zwei gleich große Widerstände schaltet, wird die Spannung von 5 V in zwei gleich große Teile aufgeteilt.
- Wenn der Widerstand zwischen dem Pin 5 V und der Mitte M kleiner ist als der Widerstand zwischen M und dem Pin GND, dann ist die Spannung zwischen M und GND größer als zwischen dem Pin 5 V und M.
- Wenn der Widerstand zwischen dem Pin 5 V und der Mitte M größer ist als der Widerstand zwischen M und dem Pin GND, dann ist die Spannung zwischen M und GND kleiner als zwischen dem Pin 5 V und M.

Aufgabe 2:

Mögliche Messwerte:

Bei Dunkelheit			Bei Helligkeit		
$U_{5V\text{-}GND}$	$U_{5V\text{-}M}$	$U_{M\text{-}GND}$	$U_{5V\text{-}GND}$	$U_{5V\text{-}M}$	$U_{M\text{-}GND}$
5V	4.7V	0.3V	5V	1.2V	3.8V

Die Messwerte für $U_{5V\text{-}M}$ und $U_{M\text{-}GND}$ variieren je nach Grad der Dunkelheit und Helligkeit.

Möglicher experimenteller Aufbau für eine Reihenschaltung:

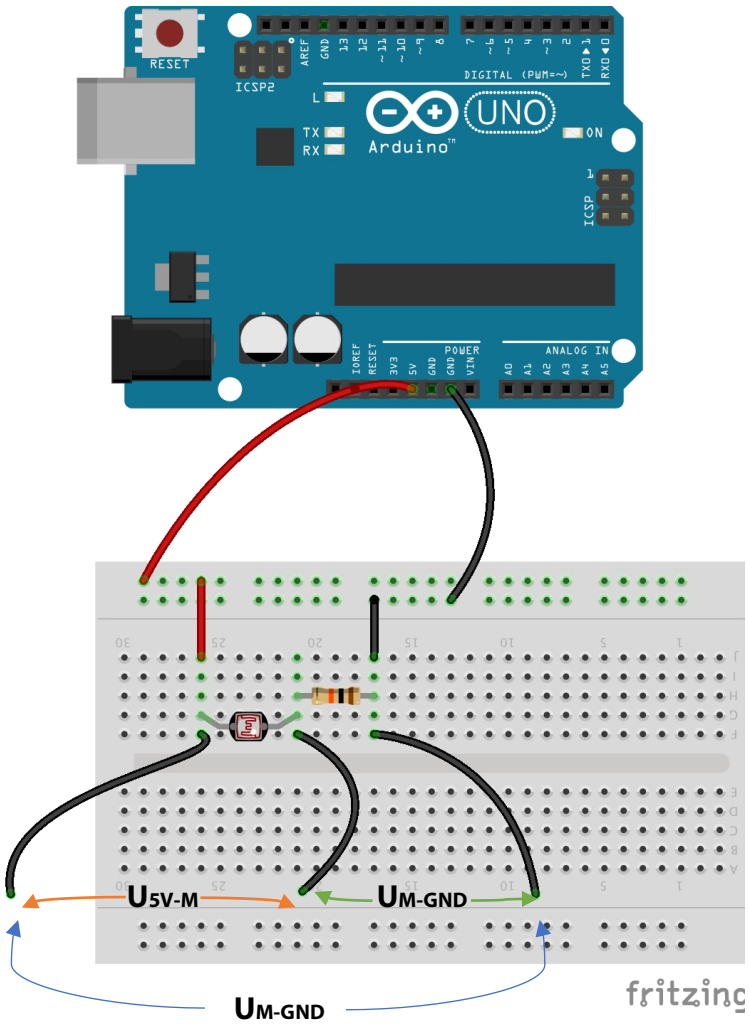


Abbildung 18: Schaltung eines LDR mit Vorwiderstand[23]

[23] Diese Abbildung wurde erstellt mit Fritzing (Lizenz: CC BY-SA 3.0)

6 Analog-Digital-Wandler im Arduino

An den Pins A0 bis A5 ist im Arduino jeweils ein 10-bit-Analog-Digital-Wandler angeschlossen. Das bedeutet, dass der Arduino die Spannungen zwischen 0.0 V und 5.0 V automatisch in $2^{10} = 1024$ Integerwerte von 0 bis 1023 umwandelt. Wenn die Spannung 2.5 V beträgt, dann bildet der Arduino die Zahl 512. Da jedes Gerät kleine Toleranzen hat, kann der Arduino auch 511 oder 513 anzeigen. Die 1024 Integerwerte sind in $\frac{5}{1024}\,\text{V} \approx 0.0049\,\text{V} = 4.9\,\text{mV}$ Portionen aufgeteilt.

Jetzt bestimmen wir die Spannungswerte zwischen M und GND mit dem Arduino, indem wir den Punkt M mit dem Analogeingang A0 des Arduino verbinden.

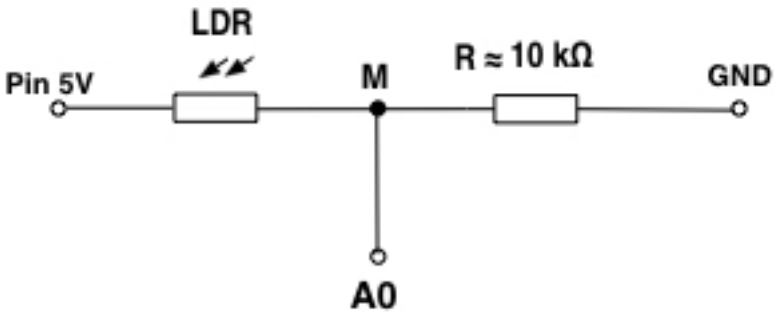


Abbildung 19: Schaltplan zu LDR mit Vorwiderstand und Verbindung zu A0

Wenn wir M mit dem Analogeingang A0 verbinden, wird mit Hilfe des Analog-Digital-Wandlers (AD-Wandler) automatisch für die Spannung zwischen M und GND eine Integerzahl zwischen 0 und 1023 vom Arduino bestimmt. Diese Zahl kann man auf dem Bildschirm des Laptop ausgeben. Dazu benutzt man die serielle USB-Verbindung (vgl. AB 6: Ausgabe der AD-Werte) zwischen Arduino und Laptop.

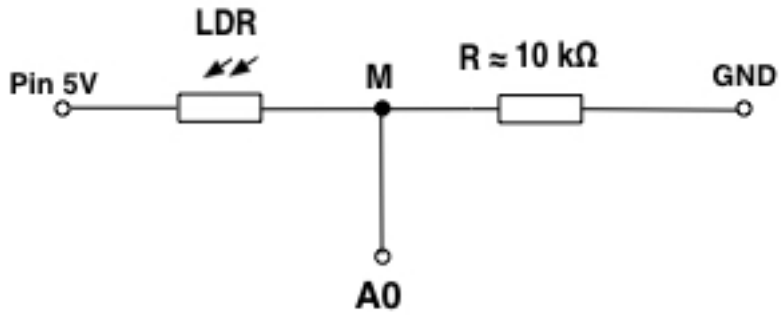
Hinweis:
In der Abbildung 19 ist zum LDR ein 10 kΩ Widerstand in Reihe geschaltet. Die Größenordnung des Widerstandes richtet sich nach dem Widerstand des LDR bei normaler Umgebungshelligkeit. Dieser LDR-Widerstand ist meistens im Bereich von 10 kΩ bis 50 kΩ. Wenn der in Reihe geschaltete Widerstand in etwa mit dem Widerstand des LDR übereinstimmt, liegt bei normaler Umgebungshelligkeit der Messwert bei M im mittleren Spannungsbereich zwischen 5 V und 0 V. Die Messwerte bei Dunkelheit oder Helligkeit sind dann nicht im Grenzbereich von 5 V und 0 V.

6.1 AB 6: Ausgabe der AD-Werte

Materialien:

- 1 Laptop mit Arduino-Software
- 1 Arduino
- 1 Breadboard
- 1 Fotowiderstand (LDR)
- 1 Widerstand 10 kΩ
- verschiedene Breadboard Kabel für den Arduino

Baue diese Schaltung auf:



Mit dem folgenden Programm kannst du die Integer-Werte des AD-Wandlers für die Spannung zwischen M und GND auf dem Laptop ausgeben.

```
1)  int sensorWert;        // Variable für den Sensorwert
2)  int analogPin = A0;    // Variable für den Analogpin A0
3)  void setup() {
4)    Serial.begin(9600); // Initialisierung der seriellen Verbindung
5)    pinMode(analogPin, INPUT); // Analogpin A0 auf INPUT schalten
6)  }
7)
8)  void loop() {
9)    sensorWert = analogRead(A0);
10)     // Spannung mit AD-Wandler am Pin A0 messen
11)   Serial.print("Sensorwert = ");
12)     // den Text zwischen den Anführungszeichen ausgeben
13)   Serial.println(sensorWert);
14)     // Spannung als Integerzahl des AD-Wandlers ausgeben und
        // an den Anfang der nächsten Zeile gehen
15)   delay(500); //Ablauf 500 ms verzögern
16) }
```

Erläuterungen zu den Programmzeilen

Zeile 1)
Die vom AD-Wandler bestimmte Zahl zwischen 0 und 1023 soll in einem ganzzahligen Speicherplatz mit dem Namen „`sensorWert`“ abgelegt werden. Deshalb schreiben wir am Programmanfang die Anweisung: `int sensorWert;`

Zeile 2)
Auch die Analogpins A0 bis A5 sind durch Integerzahlen festgelegt. A0 = 14, A1 = 15, ... A5 = 19. Deshalb können auch die Analogpins als Integerzahlen definiert werden, z.B `int analogPin = A0`.

Zeile 3)
In der setup-Funktion initialisieren wir die serielle Verbindung mit `Serial.begin(9600)`, d. h. die Daten werden mit einer Geschwindigkeit von 9600 Bit/Sekunde übertragen. Da für ein Zeichen (z. B. A) 8 Bit erforderlich sind, werden 9600/8 = 1200 Zeichen/Sekunde übertragen.

Zeile 4
Beim Aufruf von `analogRead(...);` schaltet der Arduino automatisch auf einen analogen INPUT. Deshalb ist `pinMode(A0, INPUT);` eigentlich nicht erforderlich. Aber zu besseren Lesbarkeit des Programms ist es sinnvoll, auch bei Analogpins die `pinMode(...)` Anweisung zu verwenden.

Zeile 6) bis 13)
In der `loop`-Funktion können wir dann mit `Serial.print` oder mit `Serial.println` die Zahlenwerte ausgeben. Bei `Serial.println` geht der Cursor nach der Ausgabe des Zahlenwertes an den Anfang der nächsten Zeile.

Aufgabe 1:

Schreibe das Arduino-Programm und teste es mit deiner Schaltung.

Aufgabe 2:

Erweitere das Arduino-Programm so, dass neben den ganzzahligen Sensorwerten zwischen 0 und 1023, die im Speicherplatz `sensorWert` abgelegt werden, auch die zugehörigen Spannungswerte berechnet und ausgegeben werden. Die Spannungswerte sollen in einer Variablen `spannungWert` abgelegt werden. Da die Spannungswerte normalerweise keine ganzen Zahlen sind, müssen wir für sie einen Speicherplatz festlegen, in dem eine Dezimalzahl stehen kann. An den Programmanfang setzen wir dazu eine weitere Anweisung mit `double spannungWert;`

Im Arduino-Programm wird der Spannungswert berechnet mit der Anweisung:

```
spannungWert = sensorWert * 5/1024;
```

Wichtiger Hinweis:
Da die Variable `sensorWert` und die anderen Zahlen in diesem Ausdruck alle vom Typ Integer sind, rechnet C++ nur mit ganzen Zahlen. Bei einem `sensorWert = 300` würde C++ den Zahlenwert 1 liefern. Wir müssen C++ darauf hinweisen, dass hier mit Dezimalzahlen zu rechnen ist.

Dazu gibt man folgenden Befehl ein:

```
spannungWert = static_cast<double>(sensorWert) * 5/1024;
```

Mit `static_cast<double>` drückt man in C++ aus, dass `sensorWert` als Dezimalzahl behandelt werden soll. Danach weiß C++, dass der gesamte Term mit Dezimalzahlen gerechnet werden muss.

Einfacher:
Wenn ein Teil des Terms als double definiert ist, wird der Term insgesamt als double von C++ bearbeitet.
`spannungWert = sensorWert*5.0/1024;`

6.2 ABL 6: Ausgabe der AD-Werte - Lösung

Aufgabe 1:

- siehe Programmcode auf Seite 30

Aufgabe 2:

```
int sensorWert;        // Variable für den Sensorwert
double spannungWert;   // Spannung zwischen M und GND
int analogPin = A0;    // Analogpin A0 für Aufnahme der Spannungen

void setup() {                 // Hier beginnt das Setup
  Serial.begin(9600);          // Serielle Ausgabe initialisieren
  pinMode(analogPin, INPUT); // Analogpin A0 auf INPUT schalten
}

void loop() {
  sensorWert = analogRead(analogPin);
    // Die Spannung an dem Fotowiderstand auslesen
  Serial.print("Sensorwert = ");
    // Ausgabe am Serial-Monitor: Das Wort Sensorwert:
  Serial.print(sensorWert);
    // Sensorwert des Fotowiderstandes als integer Zahl
  spannungWert = static_cast<double>(sensorWert)*5/1024;
  Serial.print(" ");
  Serial.print(spannungWert);
  Serial.println(" V");
  delay(500); //Ablauf 500 ms verzögern
}
```

Möglicher experimenteller Aufbau**:**

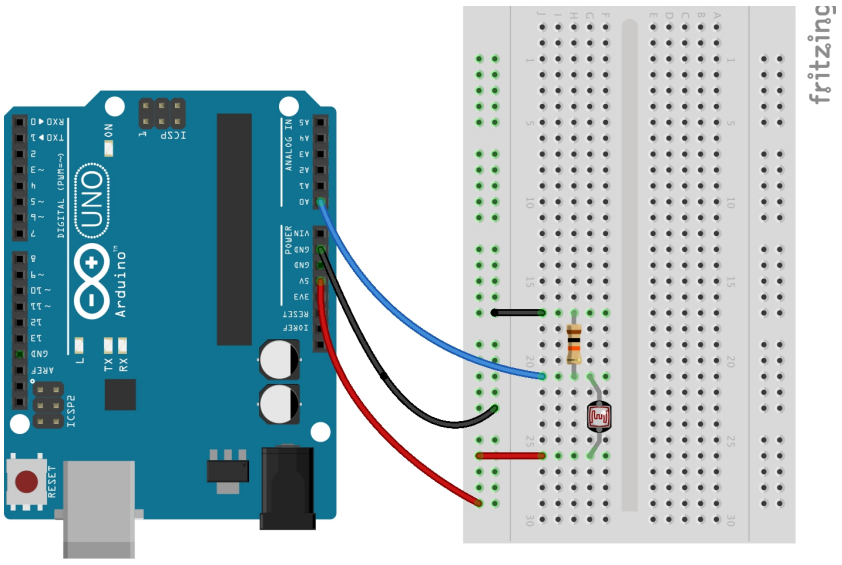


Abbildung 20: Schaltung LDR mit Vorwiderstand und Verbindung zu A0[24]

[24] Diese Abbildung wurde erstellt mit Fritzing (Lizenz: CC BY-SA 3.0)

6.3 AB 7: LED bei Dunkelheit einschalten

Materialien:

- 1 Laptop mit Arduino-Software
- 1 Arduino
- 1 Breadboard
- 1 rote LED
- 1 aktiver Piezo-Lautsprecher
- 1 Fotowiderstand (LDR)
- 1 Widerstand 10 kΩ
- 1 Widerstand ≈ 200 Ω
- verschiedene Breadboard Kabel für den Arduino

Um bei einem bestimmten Sensorwert eine LED einzuschalten, schließen wir eine LED an den Digitalpin 10.

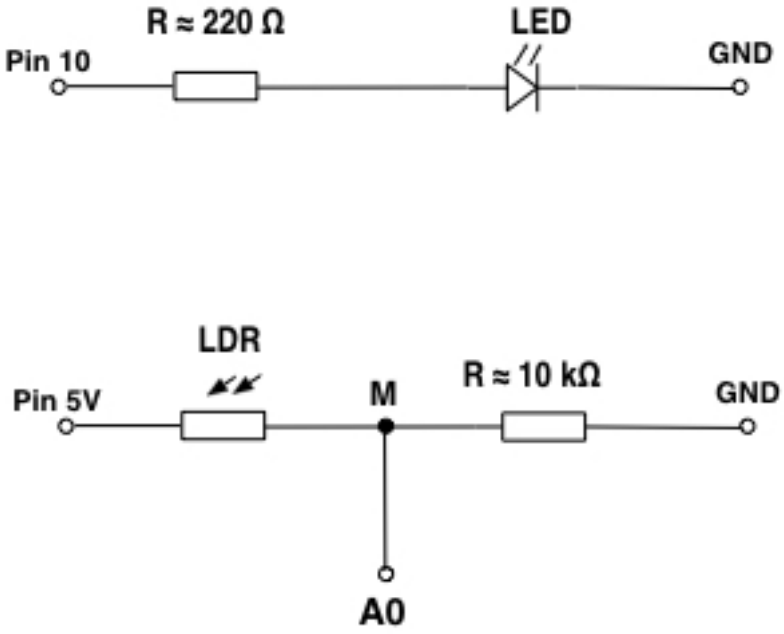


Abbildung 21: Schaltplan zu LDR und LED-Signal

Wenn der LDR nur noch schwach beleuchtet wird, ist es dunkel und sein Widerstand ist groß. Wie wir wissen, ist dann bei M die Spannung klein, d. h. der digitale Sensorwert ist z. B. kleiner als 300. Im folgenden Programm wird für einem Sensorwert von weniger als 300 die LED eingeschaltet.

Die zugehörige Anweisung im Programm ist:

```
if (sensorWert < 300)
{
  digitalWrite(10,HIGH); // LED einschalten
}
```

Die Bedingung `(sensorWert < 300)` muss in runden Klammern hinter `if` stehen!
Alle Anweisungen, die zu `if` gehören, setzt man in geschweifte Klammern `{ ... }`.
Falls die Bedingung nicht erfüllt ist, kann mit `else` die Anweisung zum Ausschalten der LED aufgerufen werden. Auch nach `else` setzt man die Anweisungen in geschweifte Klammern `{ ... }`.

```
int sensorWert;     // Variable für den Sensorwert
int analogPin = A0; // Analogpin A0 für Aufnahme der Spannungen
int ledPin = 10;    // Digitalpin 10 für die LED

void setup() {
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
  pinMode(analogPin, INPUT);
}

void loop() {
  sensorWert = analogRead(analogPin);   // Spannung messen
  Serial.print("Sensorwert = ");
  Serial.println(sensorWert);           // Spannung ausgeben
  if (sensorWert < 300)
  {
     digitalWrite(ledPin, HIGH);        // LED einschalten
  }
  else
  {
     digitalWrite(ledPin, LOW);         // LED ausschalten
  }
  delay(500); //Ablauf 500 ms verzögern
}
```

Aufgabe:

Baue nach der Schaltskizze die Schaltung auf dem Breadboard auf.
Teste das Programm mit deiner Schaltung. Möglicherweise musst du den Sensorwert noch anpassen, damit die LED bei Dunkelheit aufleuchtet und bei Helligkeit ausgeht.

Zusatzaufgabe:

Verwende außer der LED noch einen aktiven Piezo-Lautsprecher, der eingeschaltet wird, wenn es dunkel wird.

Hinweis:

- Ein aktiver Piezo-Lautsprecher hat einen integrierten Oszillator mit einer festen Frequenz (z. B. 2300 Hz). Er tönt bei HIGH (5V) und wird mit LOW (0V) ausgeschaltet.

Der Piezo-Lautsprecher wird wie im folgenden Schaltbild zwischen den Pin 8 und GND geschaltet:

Abbildung 22: Schaltplan für Signal mit Piezo

Schreibe das Programm für den Piezo-Lautsprecher entsprechend um.

6.4 ABL 7: LED bei Dunkelheit einschalten - Lösung

Aufgabe

Möglicher experimenteller Aufbau:

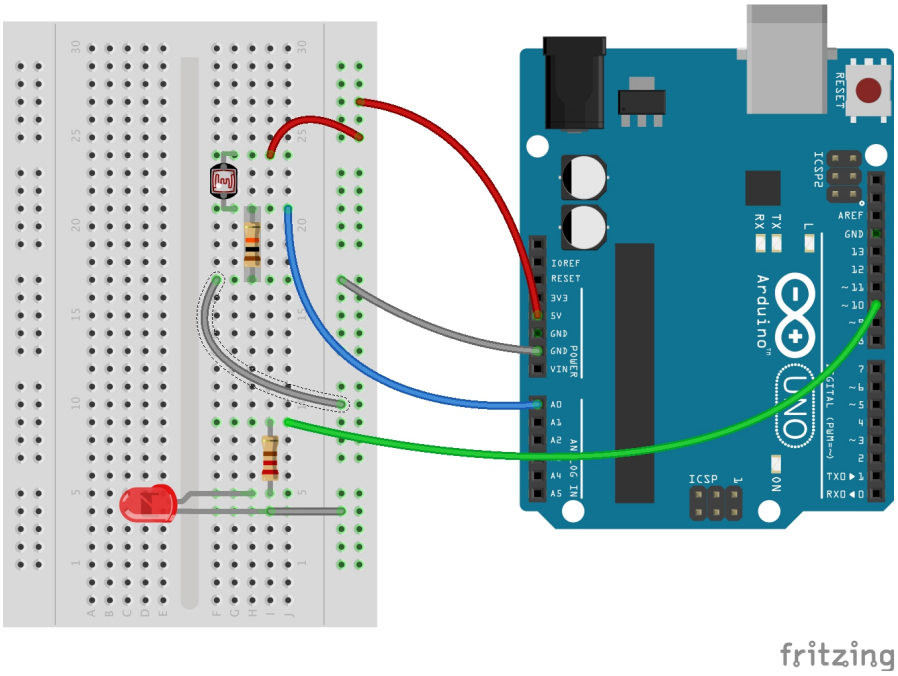


Abbildung 23: Schaltung zu LDR und LED-Signal[25]

Zusatzaufgabe

Möglicher experimenteller Aufbau für eine Reihenschaltung:

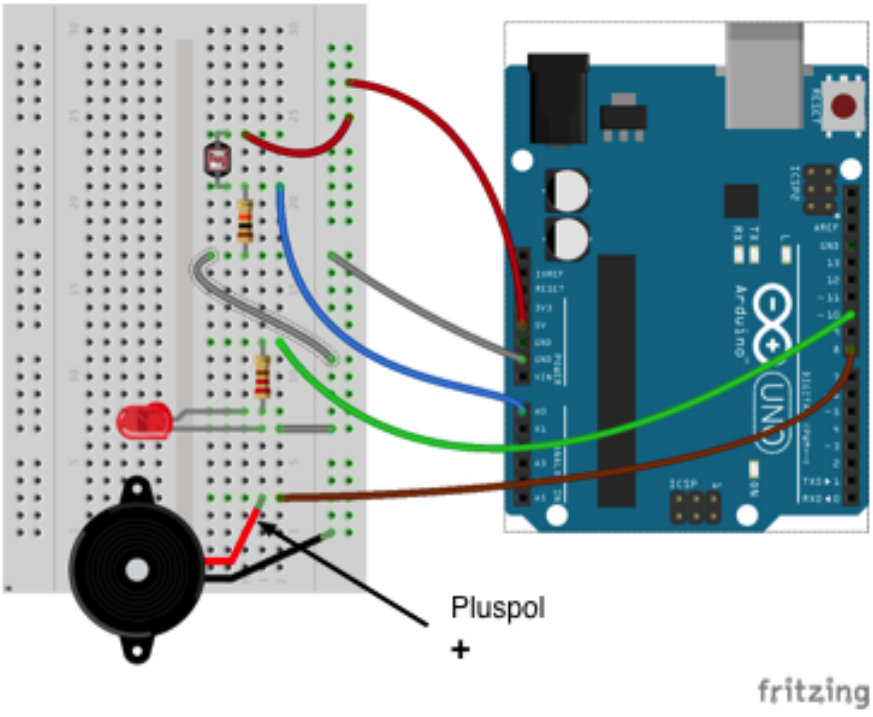


Abbildung 24: Schaltung zu LDR und LED+Piezo-Signal[26]

[25] Diese Abbildung wurde erstellt mit Fritzing (Lizenz: CC BY-SA 3.0)
[26] Diese Abbildung wurde erstellt mit Fritzing (Lizenz: CC BY-SA 3.0)

Mögliches Arduino-Programm:

```
int sensorWert;      // Variable für den Sensorwert
int analogPin = A0;  // Analogpin 0 für Aufnahme der Spannungen
int ledPin = 10;     // Digitalpin 10 für die LED
int piezoPin = 8;    // Digitalpin 10 für den Piezo

void setup() {
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
  pinMode(piezoPin, OUTPUT);
  pinMode(analogPin, INPUT);
}

void loop() {
  sensorWert = analogRead(analogPin);   // Spannung messen
  Serial.print("Sensorwert = ");
  Serial.println(sensorWert);           // Spannung ausgeben
  if (sensorWert < 300)
  {
     digitalWrite(ledPin, HIGH);        // LED einschalten
     digitalWrite(piezoPin, HIGH);      // Piezo einschalten
     delay(2000);                       // 2 Sekunden zuwarten
     digitalWrite(piezoPin, LOW);       // Piezo ausschalten
  }
  else
  {
     digitalWrite(ledPin, LOW);        // LED ausschalten
  }
  delay(500);                          //Ablauf 500 ms verzögern
}
```

7 Bewegungsmelder

Beim Arduino kann man einen Bewegungsmelder einfach mit einem Passive InfraRed-Sensor (PIR) steuern. Sobald der PIR eine Bewegung eines Lebewesens erkennt, gibt er ein Signal von 5 V aus, sonst gibt er 0 V aus. Dieses Signal liest man an einem Digitalpin ein (`digitalRead`) und kann dann eine LED einschalten (`digitalWrite`) oder einen Piezo-Lautsprecher ertönen lassen.

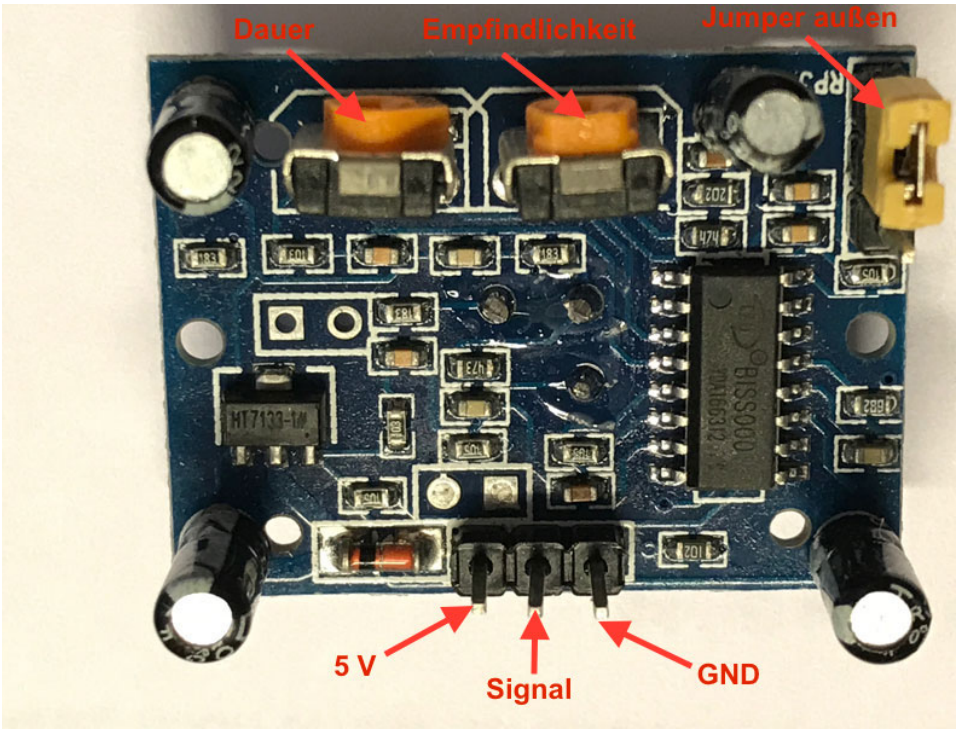


Abbildung 25: Bewegungsmelder Platine mit Anschlusspins und Einstellreglern

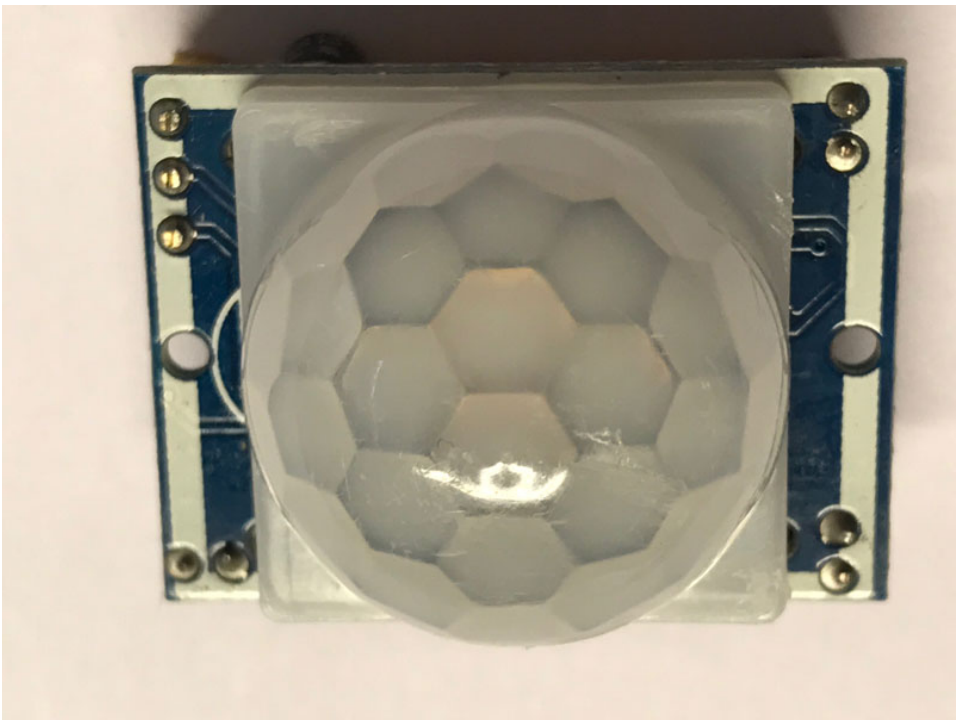


Abbildung 26: Kunststofflinse über dem PIR

7.1 AB 8: Alarmanlage

Materialien:

- 1 Laptop mit Arduino-Software
- 1 Arduino
- 1 Breadboard
- 1 rote LED
- 1 aktiver Piezo-Lautsprecher
- 1 Passive InfraRed-Sensor (PIR)
- 1 Widerstand 10 kΩ
- 1 Widerstand ≈ 200 Ω
- verschiedene Breadboard Kabel für den Arduino
- 3 female-male Breadboard Kabel

Aufgabe 1:

Baue nach der Schaltskizze die Schaltung auf dem Breadboard auf.

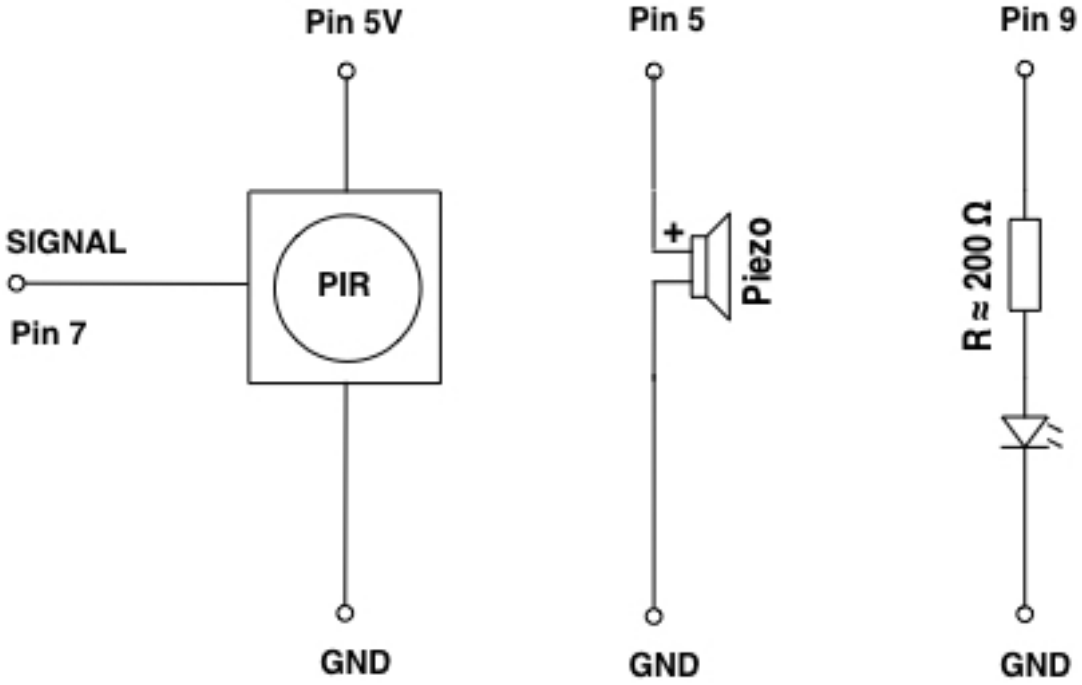


Abbildung 27: Schaltplan zu PIR und Piezo+LED-Signal

Das Signal des Bewegungssensors kommt am Digitalpin 7 an. Da der Arduino dabei ein digitales Signal von 5 V einliest, müssen wir in der setup-Funktion eine Anweisung `pinMode(7, INPUT);` eintragen.
Der **aktive** Piezo-Lautsprecher wird am Pin 5 angeschlossen.
Die LED wird am Pin 9 angeschlossen.

Aufgabe 2:

a) Schreibe die `pinMode`-Anweisungen in die `setup`-Funktion für das Signal des Bewegungssensors, den aktiven Piezo-Lautsprecher und die LED auf.

Mit der Anweisung `status = digitalRead(7);` wird der Signalwert auf Pin 7 eingelesen und in einem Speicherplatz mit dem Namen `status` abgespeichert. Für 0 V wird die Zahl 0 abgespeichert. Für 5 V wird die Zahl 1 abgespeichert. `status` hat also nur die zwei ganzen Zahlenwerte 0 oder 1. Deshalb muss `status` ganz am Anfang des Programms als Integerzahl festgelegt werden mit der Definition: `int status;`

Für den Zahlenwert 1 kann HIGH und für den Zahlenwert 0 kann LOW im Programm geschrieben werden.

- Wenn `status` auf HIGH ist, soll die LED für 5 s aufleuchten und der aktive Piezo-Lautsprecher für 5 s tönen.
- Wenn `status` auf LOW ist, sollen LED und aktiver Piezo-Lautsprecher ausgeschaltet werden.

In C++ lauten die Anweisungen dazu:

```
if (status == HIGH)
{
  digitalWrite(5, HIGH);      // aktiven Piezo einschalten
  digitalWrite(9, HIGH);      // LED einschalten
  delay(5000);                // 5 Sekunden Dauer
  digitalWrite(5, LOW);       // aktiven Piezo ausschalten
  digitalWrite(9, LOW);       // LED ausschalten
}
else
{
  digitalWrite(5, LOW);
  digitalWrite(9, LOW);
}
```

b) Versuche das Programm zu erstellen, indem du deine Anweisungen in die `setup`-Funktion und in die `loop`-Funktion einträgst. Teste es mit deinem experimentellen Aufbau.

Beachte:

- Die Definition `int status;` muss am Anfang des Programms stehen.
- Die Bedingung `status == HIGH` muss in runden Klammern hinter `if` stehen!
  Beim Vergleich muss ein doppeltes Gleichheitszeichen verwendet werden! Wie im Glossar (vgl. S. 83 ff) dargestellt, wird das einfache Gleichheitszeichen in C++ nicht dazu verwendet, um eine Gleichung darzustellen, sondern um eine Zuweisung zu machen.
- Die Anweisungen hinter der `if-else`-Anweisung sind zwischen geschweifte Klammern zu setzen.

7.2 ABL 8: Alarmanlage - Lösung

Aufgabe 1:

Möglicher experimenteller Aufbau**:**

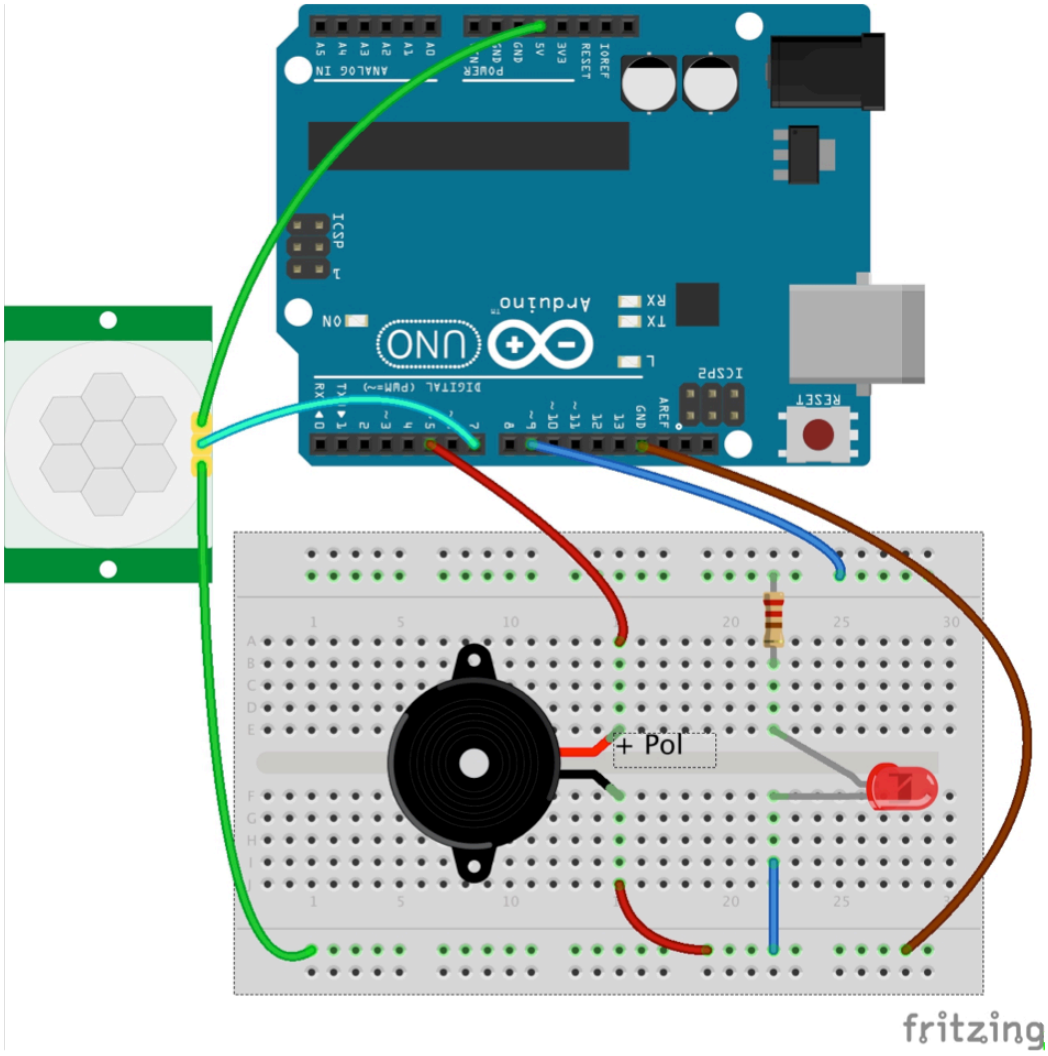


Abbildung 28: Schaltung zu PIR und Piezo+LED-Signal[27]

[27] Diese Abbildung wurde erstellt mit Fritzing (Lizenz: CC BY-SA 3.0)

Aufgabe 2:

Mögliches Arduino-Programm für eine Alarmanlage:

```
int status;                    // Signalstatus
int pirPin = 7;                // PIR an Pin 7
int piezoPin = 5;              // PIEZO an Pin5
int ledPin = 9;                // LED an Pin 9

void setup() {
  pinMode(pirPin, INPUT);      // Signal des Bewegungssensors
  pinMode(piezoPin, OUTPUT);   // aktiver Piezo
  pinMode(ledPin, OUTPUT);     // LED
}

void loop() {
  status = digitalRead(pirPin);
  if (status == HIGH)
  {
    digitalWrite(piezoPin, HIGH);    // aktiven Piezo einschalten
    digitalWrite(ledPin, HIGH);      // LED einschalten
    delay(5000);                     // 5 Sekunden Dauer
    digitalWrite(piezoPin, LOW);     // aktiven Piezo ausschalten
    digitalWrite(ledPin, LOW);       // LED ausschalten
  }
  else
  {
     digitalWrite(piezoPin, LOW);  // aktiven Piezo ausschalten
     digitalWrite(ledPin, LOW); // LED ausschalten
  }
}
```

8 Entfernungsmessung mit Ultraschall

Der Ultraschallsensor HC-SR04 besitzt einen Sender (Trigger) und einen Empfänger (Echo), die ähnlich wie bei einer Fledermaus funktionieren (Mund = Trigger; Ohren = Echo). Der Ultraschall ist ein Ton mit ca. 40 kHz.

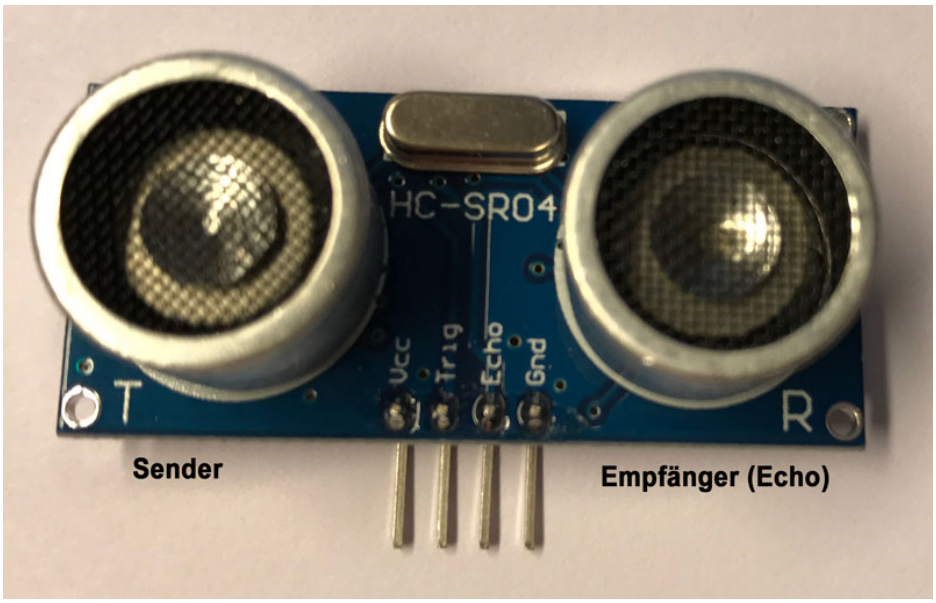


Abbildung 29: Ultraschallentfernungsmesser HC-SR04

Durch den Trigger wird ein kurzer Impuls von mindestens 10 µs ausgegeben. Kurze Zeit nach dem Abfall der Trigger-Flanke (250 µs) sendet der Ultraschallsensor 200 µs lang ein 40 kHz Burst-Signal (= 8 Perioden). Sofort danach geht der Echo-Eingang auf einen HIGH-Pegel und wartet auf den Empfang des akustischen Echos. Sobald das Echo des Burst-Signals vollständig registriert ist, fällt der Ausgang am Echo auf den LOW-Pegel. Die Distanz d berechnet sich dann aus der Zeit t vom Aussenden des Signals bis zu seiner Rückkehr mit $d = \frac{c \cdot t}{2}$. Die Distanz entspricht nämlich der Hälfte des vom Burst-Signal zurückgelegten Weges. Die Schallgeschwindigkeit c beträgt bei 20 °C etwa 344 $^{m}/_{s}$. Diese Ultraschall-Entfernungsmessung kann zwischen 2 cm und 3 m mit einer Genauigkeit von etwa ±2 mm eingesetzt werden.

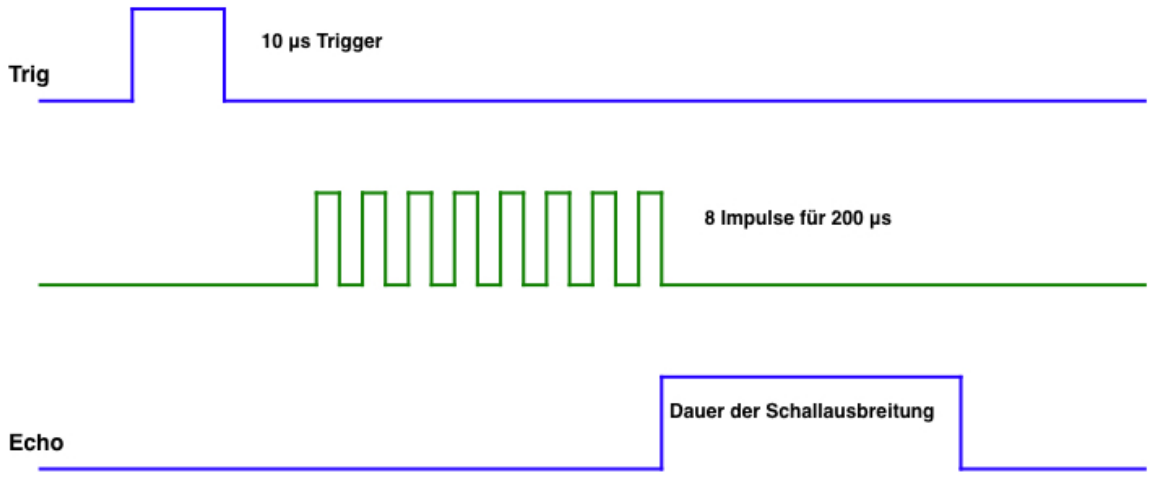


Abbildung 30: Signalverlauf beim HC-SR04

Um die Pulslänge am Echo-Eingang zu messen, wird die Funktion `pulseIn()` verwendet. Bei `pulseIn(6, HIGH)` wartet die Funktion, bis das Signal am Pin 6 auf HIGH geht und startet dann das Timing. Das Timing wird gestoppt, sobald das gesamte Burst-Signal zurück ist. Damit geht das Signal am Pin 6 auf LOW. Das Ergebnis wird von `pulseIn()` in Mikrosekunden geliefert bzw. 0, wenn kein vollständiges Echo des Burst-Signals angekommen ist.

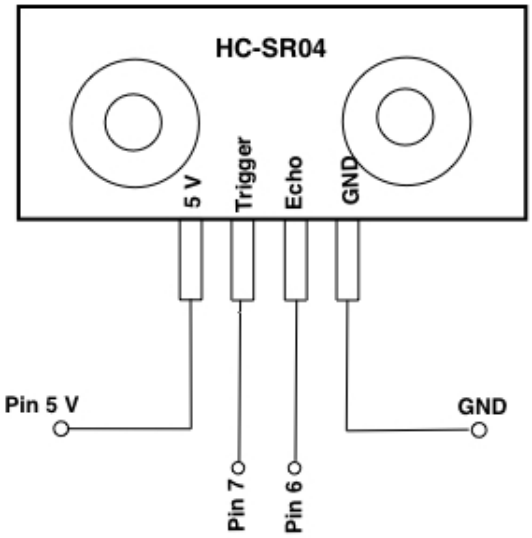


Abbildung 31: Pinblegung beim HC-SR04

Einige Hintergrundinformationen zum Programm

1) Auf dem Arduino Uno werden ganze Zahlen (Integer-Zahlen) als 16 Bit-Wert abgespeichert. Daraus folgt, dass ein Zahlenbereich von -2^{15} bis $+(2^{15}-1)$ dargestellt werden kann. $2^{15} = 32768$

2) Mit der Definition `unsigned int echo;` kann die Variable `echo` Zahlenwerte von 0 bis $+(2^{16}-1) = 65535$ darstellen.

3) Abstandsmessung mit der Schallgeschwindigkeit:
   a) Laufzeit des Schalls hin und zurück in Mikrosekunden: `dauer`
   b) Schallgeschwindigkeit bei 20°C: $344\ ^{m}/_{s}$.
      In 1 s legt der Schall 344 m zurück.
      In einer Mikrosekunde legt er nur 0,0344 cm zurück.
   c) Für die Entfernung in cm ergibt sich damit:
      `entfernung = 0.0344*(dauer/2);`
      i. Hier rechnet der C++ Compiler automatisch alles mit `double`, weil 0.0344 eine Dezimalzahl ist.
      ii. Wenn das Echo nicht vollständig zurückkommt (z.B. Reflexion an einem Schwamm), liefert `pulseIn` als Ergebnis die Integerzahl 0.
      iii. HC-SR04 kann Entfernungen von 2 cm bis 300 cm durch Reflexion an glatten, ebenen Flächen zuverlässig messen. Bei Reflexionen an weichen oder porösen Oberflächen (z. B. Schwamm) kommt es leicht zu Messfehlern. Nur wenn `dauer != 0` (ungleich 0) ist, können sinnvolle Entfernungen gemessen werden.
   d) Temperaturabhängigkeit der Schallgeschwindigkeit:
      $c_{Luft} \approx (331{,}5 + 0{,}6\vartheta)\ ^{m}/_{s}$ mit ϑ in °C

8.1 AB 9: Entfernungsmesser mit Ultraschall

Materialien:

- 1 Laptop mit Arduino-Software
- 1 Arduino
- 1 Breadboard
- 1 Entfernungsmesser HC-SR04
- verschiedene Breadboard Kabel für den Arduino
- 4 female-male Breadboardkabel
- Halterung für den Entfernungsmesser
- Doppelklebebandstreifen
- Sockel

Aufgabe 1:

Baue die Schaltung des HC-SR04 auf dem Breadboard auf und verbinde mit dem Arduino. Der Entfernungsmesser ist in ein Gesicht als Halterung eingebaut.

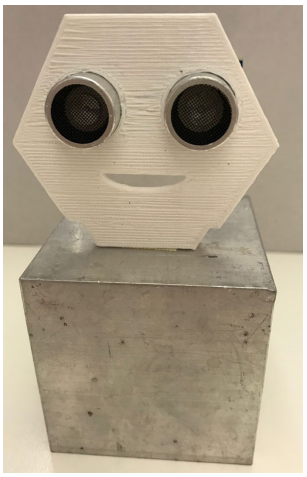


Abbildung 32: Entfernungsmesser fixiert

Aufgabe 2:

Schreibe ein Programm zur Entfernungsmessung. Folgende Hinweise können dir helfen:

a) Festlegung von Variablen
Weise mit folgenden Anweisungen für den Pin 7 der Variablen `trigger` die Zahl 7 und für den Pin 6 der Variablen `echo` die Zahl 6 zu. Die Laufzeit des Schalls wird als ganze Zahl in Mikrosekunden gemessen. Wir nennen die zugehörige Variable `dauer`. Die Entfernung ist eine Dezimalzahl und wird deshalb mit einer Variablen vom Typ `double` bearbeitet. Diese Variablen müssen ganz am Anfang des Programms, also vor der setup-Funktion, stehen.

```
int trigger = 7;       // Trigger-Pin des Ultraschallsensors
int echo = 6;          // Echo-Pin des Ultraschallsensors
int dauer;             // Laufzeit des Schallsignals
double entfernung;     // Berechnete Entfernung
```

b) Initialisierungen
`trigger` muss auf OUTPUT und `echo` auf INPUT initialisiert werden. Damit wir die Messwerte auf dem Bildschirm des Computers lesen können, verwenden wir die Ausgabe über die USB-Verbindung. Diese muss initialisiert werden mit:

```
Serial.begin(9600);
```

c) Schreibe die Anweisungen zur Entfernungsmessung in die loop-Funktion
Gehe dazu in folgenden Schritten vor:

1. Setze den Trigger auf LOW. Warte zwei Millisekunden und setze dann den Trigger auf HIGH. Halte mit `delayMicroseconds(10)` den Trigger für 10 µs auf HIGH.
   Setze danach den `trigger` wieder auf LOW.
2. Miss mit folgender Anweisung die Laufzeit des Schalls in Mikrosekunden:

   ```
   dauer = pulseIn(echo, HIGH);
   ```

3. Berechne mit folgender Anweisung die Entfernung:

   ```
   entfernung = 0.0344*(dauer/2);
   ```

4. Lege mit Hilfe von `if-else`-Anweisungen folgendes fest:
   Wenn kein Messfehler festgestellt wird (`dauer != 0`) und wenn die Entfernung zwischen 2 cm und 300 cm liegt, gibst du die Entfernung mit folgenden Anweisungen aus:

   ```
   Serial.print("Entfernung = ");
   Serial.print(entfernung);
   Serial.println(" cm");
   ```

5. Sonst soll auf dem Bildschirm mit `Serial.println("kein Messwert!")` eine Fehlermeldung erscheinen.
6. Nach einer Entfernungsmessung lässt du das Programm 3 s pausieren, bis du wieder eine neue Messung machst.

Hinweis zur Funktion pulseIn():

pulseIn(echo, HIGH) wartet am Pin echo bis das Signal dort auf HIGH geht. Dann wird der Timer gestartet bis das Signal am Pin echo auf LOW wechselt. Danach wird der Timer gestoppt.

8.2 ABL 9: Entfernungsmesser mit Ultraschall - Lösung

Aufgabe 1:

Möglicher experimenteller Aufbau:

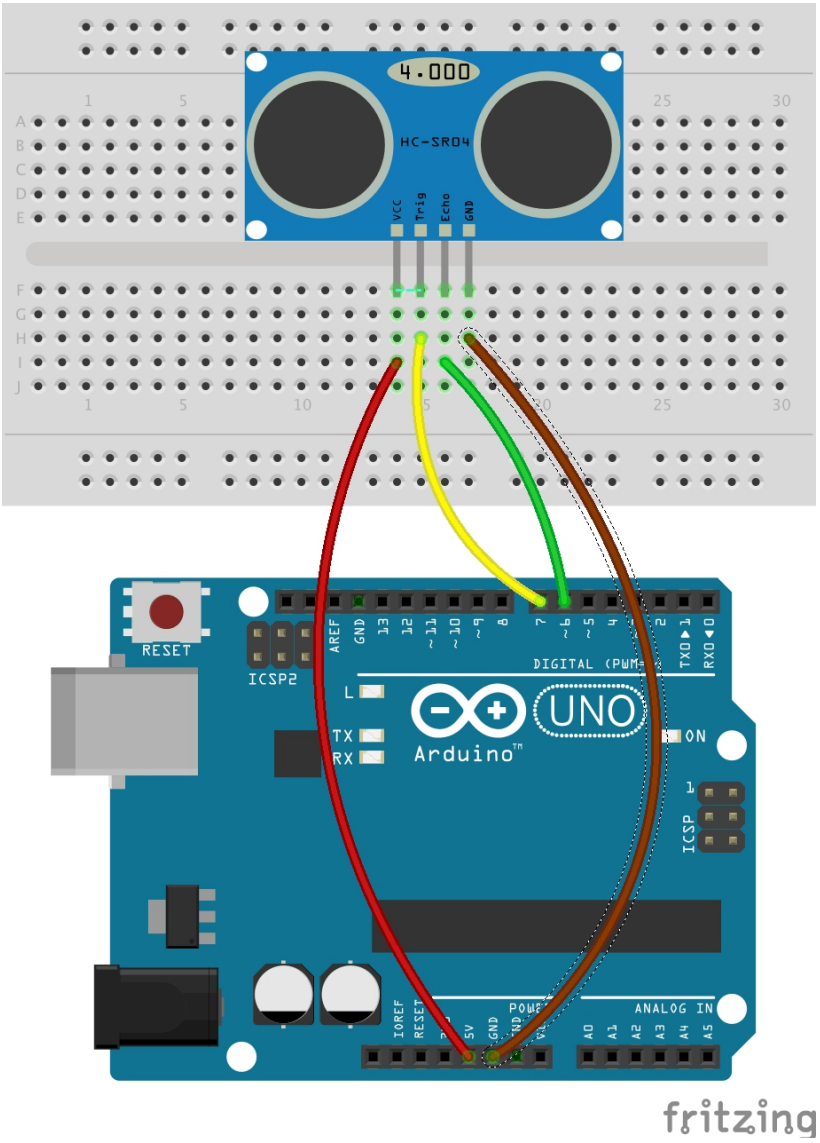


Abbildung 33: Schaltung für den HC-SR04[28]

[28] Diese Abbildung wurde erstellt mit Fritzing (Lizenz: CC BY-SA 3.0)

Aufgabe 2:

Mögliches Arduino-Programm für eine Entfernungsmessung mit Ultraschall:

```
int trigger = 7;     // Trigger-Pin des Ultraschallsensors
int echo = 6;        // Echo-Pin des Ultraschallsensors
int dauer;           // Laufzeit des Schallsignals in Mikrosek.
double entfernung;   // Berechnete Entfernung

void setup() {
  pinMode(trigger, OUTPUT); // trigger Pin mit OUTPUT initialisieren
  pinMode(echo, INPUT);    // echo Pin mit INPUT initialisieren
  Serial.begin(9600);      // Serielle Verbingung initialisieren
}

void loop() {
  digitalWrite(trigger, LOW);
  delay(2);                      //Ablauf 2 ms verzögern
  digitalWrite(trigger, HIGH);
  delayMicroseconds(10);         //Ablauf 10 µs verzögern
  digitalWrite(trigger, LOW);
    // Triggersignal auf LOW.
    // Nach 250 Mikrosek. wird das Burstsignal ausgesandt
  dauer = pulseIn(echo, HIGH); // Laufzeit des Schalls in µs
    // Der Echo-Eingang wird sofort danach auf HIGH geschaltet
    // Sobald der gesamte Burst zurück ist, auf LOW
  entfernung = 0.0344*(dauer/2); // Entfernung in cm
  if ((dauer != 0) && (entfernung < 300) && (entfernung > 2))
  {
    Serial.print("Entfernung = ");
    Serial.print(entfernung);
    Serial.println(" cm");
  }
  else
  {
    Serial.println("Kein Messwert!");
  }
  delay(3000); //Ablauf 3000 ms verzögern
}
```

Wichtiger Hinweis:
Dezimalzahlen müssen im C++ Programm immer mit einem Punkt dargestellt werden.

9 Steuerung eines Gleichstrommotors

Die meisten Elektromotoren können mit den kleinen Stromstärken des Arduino (maximal 40 mA) nicht betrieben werden. Wir müssen deshalb sogenannte Motortreiber einsetzen. Sehr häufig werden die Motortreiber L293D oder L298N verwendet. Im Folgenden wird der Motortreiber L298N erläutert. Mit dem Motortreiber L298N lassen sich zwei Gleichstrommotoren mit Spannungen bis zu 35 V und bis zu einer Stromstärke von 2 A verwenden. Elektronisch kommen hierfür zwei H-Brücken zum Einsatz, mit denen die Drehrichtungen der Motoren gesteuert werden.

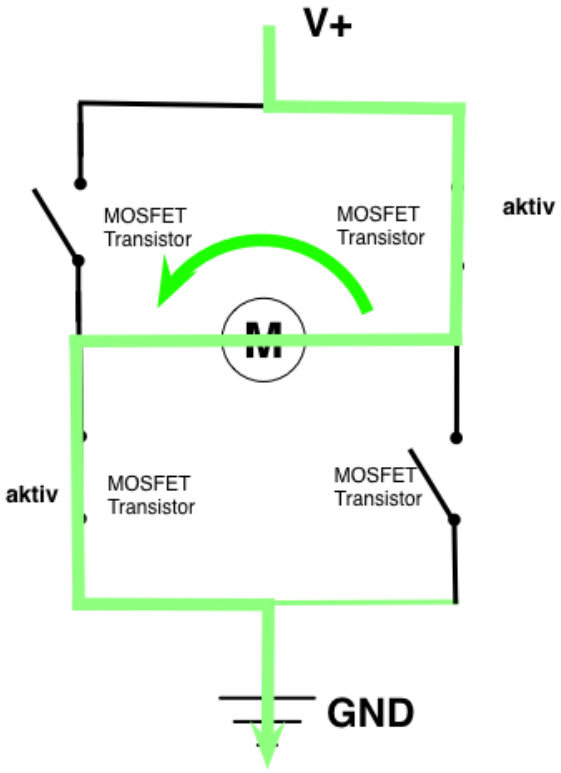


Abbildung 34: H-Brücke im Motortreiber L298N

Die Drehzahl des Gleichstrommotors kann elegant geregelt werden, indem der Enable-Pin des L298N an einen der PWM-Pins des Arduino angeschlossen wird. Mit der Pulsweitenmodulation wird der Motor in regelmäßigen Zeitspannen ein- und ausgeschaltet. Die Pulsweiten können von 0 bis 255 eingeteilt werden. Bei einer Pulsweite 255 handelt es sich um ein konstantes HIGH-Signal und bei einer Pulsweite 0 um ein konstantes LOW-Signal. Die Zahlenwerte für die Pulsweiten zwischen 0 und 255 müssen mit `analogWrite(...)` ausgegeben werden, weil es nicht wie bei `digitalWrite(...)` nur die zwei Werte `LOW` und `HIGH` bzw. 0 V und 5 V sind. Der Analogwert wird dann an ENA des L298N weitergegeben.
Die PWM-Pins für analogWrite() sind in Reihe der Digitalpins!

Hintergrundinformation zur Pulsweitenmodulation (PWM):

Bei der Pulsweitenmodulation (PWM) des Arduino wird ein Rechtecksignal, das eine feste Frequenz und eine Ausgangsspannung von 0 V bzw. 5 V hat, in der Breite der Pulse variiert. Beim Arduino Uno arbeitet diese PWM an den Pins 3, 5, 6, 9, 10 und 11[29]. Die PWM Frequenz beträgt ungefähr 500 Hz, d. h. die Periodendauer ungefähr 2 ms.

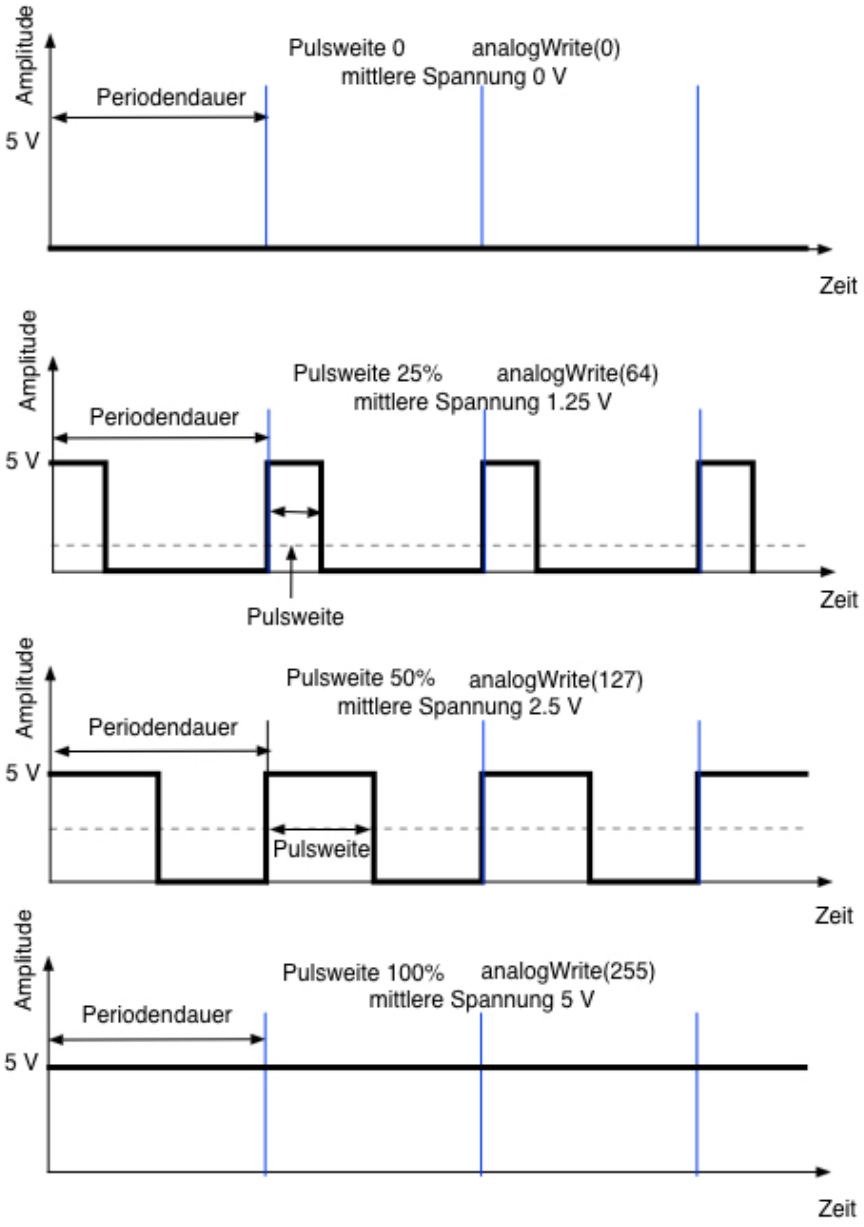


Abbildung 35: Signale bei verschiedenen Pulsweiten

[29] Diese Pins sind mit einer Tilde (~) gekennzeichnet.

Der L298N wird häufig in einem Modul eingesetzt, auf dem die notwendigen Anschlüsse herausgeführt sind.

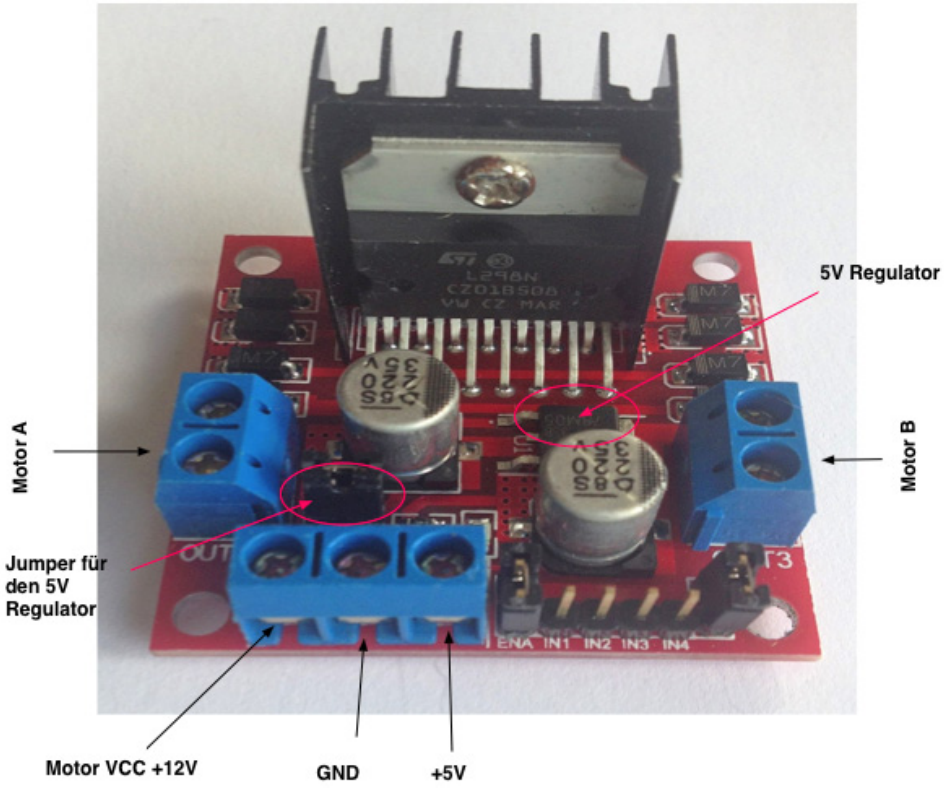


Abbildung 36: **Motortreibermodul mit L298N**

Motor A / Motor B
Anschluss jeweils eines Gleichstrommotors

Motor VCC +12V
Spannung maximal 12 V für die Motoren A und B

Jumper für den 5V Regulator
Mit diesem Jumper wird der 5V Regulator versorgt und eine 5V Spannung zur Verfügung gestellt.

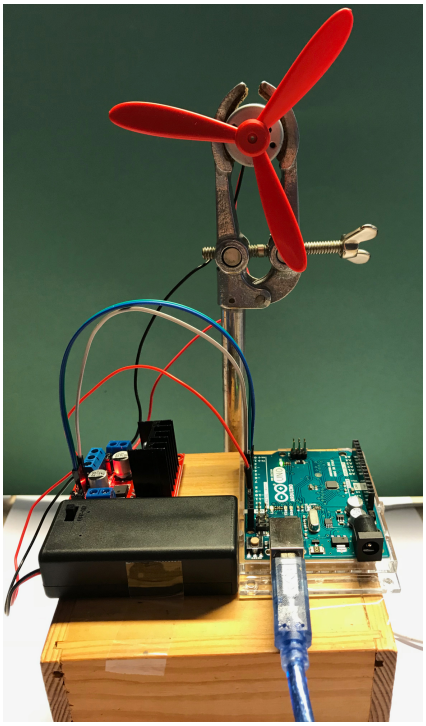


Abbildung 37: **Versuchsaufbau für einen Gleichstrommotor**

9.1 AB 10: Richtungswechsel und Geschwindigkeitsregelung

Materialien:

- 1 Laptop mit Arduino-Software
- 1 Arduino
- 1 Breadboard
- 9V Batterie mit Anschlussadapter
- Box für die 9V Batterie mit Ein- /Aus-Schalter
- optional: DC-DC Adapter
- Gleichstrommotor
- Motortreibermodul L296N
- Propeller
- Klemmhalterung für den Elektromotor
- verschiedene Breadboard Kabel für den Arduino
- Schraubendreher
- 5 female-male Breadboard Kabel

Versuchsaufbau:
Wir setzen einen Gleichstrommotor (RK-370CA-14420 TD098812) über OUT1/OUT2 (Motor A) in Gang. Auf die Motorachse stecken wir einen Propeller.
Stelle die Ausgangspannung mit der Potentiometerschraube am DC-DC-Konverter auf 6 V ein.

Aufgabe 1:

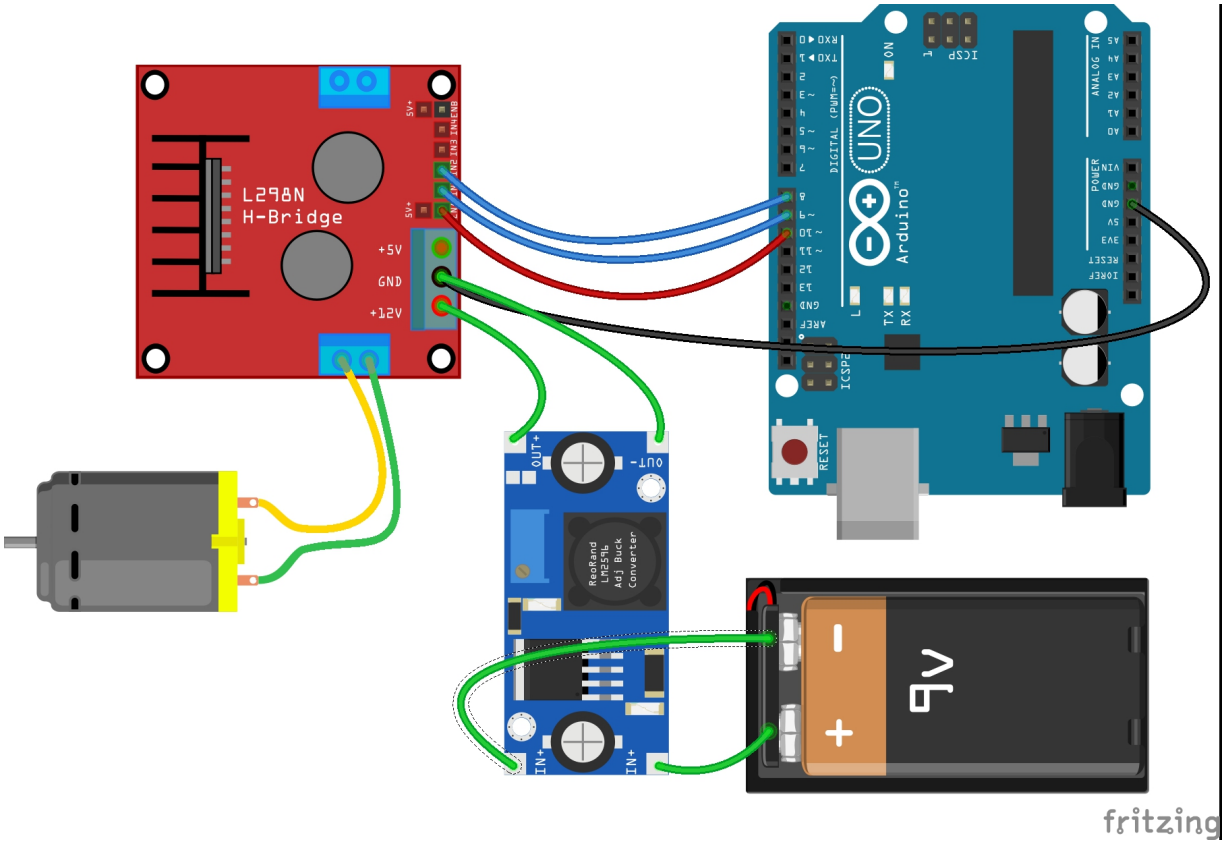


Abbildung 38: Schaltung (2) für einen Gleichstrommotor[30]

[30] Diese Abbildung wurde erstellt mit Fritzing (Lizenz: CC BY-SA 3.0)

Schaltung
Verbinde den Plus- und Minuspol der 9V-Batterie mit den Eingängen IN+ und IN- des LM2596 DC-DC-Moduls. Stelle am Potentiometer auf dem LM2596-Modul mit einem Schraubendreher vor in Betriebnahme die Spannung an den Ausgängen OUT+ und OUT- auf die gewünschte Maximalspannung (z.B. 6V) für den Betrieb des Motors ein. Verbinde den Ausgang OUT+ des LM2596 DC-DC-Moduls mit dem +12-V-Schraubpin und den Ausgang OUT- mit dem GND des L298N-Moduls **und** des Arduino.

In der dargestellten Schaltung bestehen folgende Verbindungen zwischen dem L298N Modul und dem Arduino.

L298N-Modul	Arduino
ENA	Pin 10
IN1	Pin 9
IN2	Pin 8

a) Durch die Verbindung von ENA mit Pin 10 kann der L298N durch ein HIGH-Signal der Anschluss des Motors M1 startklar gemacht und auf eine passende Geschwindigkeit eingestellt werden. Es ist darauf zu achten, dass die Höchstgeschwindigkeit mit der maximalen Versorgungsspannung des Motors angepasst wird. Deshalb haben wir für den eingesetzten Motor am DC-DC-Konverter 6V eingestellt. Die Anweisung zum Erreichen der Maximalgeschwindigkeit ist: `analogWrite(10,255);`

b) Wenn man am L298N den Anschluss IN1 auf LOW und IN2 auf HIGH setzt, dreht sich der Motor links oder rechts herum. Wenn man am L298N den Anschluss IN1 auf HIGH und IN2 auf LOW setzt, dann dreht sich der Motor _____ .

c) Verbinde die Anschlüsse am L298N Modul mit den Pins am Arduino wie in der Schaltung dargestellt.

Aufgabe 2:

Erstelle ein Programm, mit dem der Motor bei maximaler Geschwindigkeit folgendermaßen gesteuert wird:

a. Motor 5 s mit maximaler Geschwindigkeit rechts oder links herum drehen lassen
b. Motor stoppt 2 s lang
c. Motor 5 s auf geringere Geschwindigkeit setzen ohne Richtungsänderung
d. Motor stoppt 2 s lang
e. Motor 5 s auf maximale Geschwindigkeit setzen in umgekehrter Richtung
f. Motor stoppt 2 s lang
g. Motor 5 s auf geringere Geschwindigkeit setzen in umgekehrter Richtung
h. Motor stoppt 10 s lang

Teste das Programm!

9.2 ABL 10: Richtungswechsel und Geschwindigkeitsregelung - Lösung

Aufgabe 1:

- Schaltung siehe Seite 51
- Wenn man am L298N den Anschluss IN1 auf HIGH und IN2 auf LOW setzt, dann dreht sich der Motor in die entgegengesetzte Richtung

Aufgabe 2:

```
int pinEn = 10; // zu EnableA beim L298N Modul
int pinIN1 = 9; // zu Pin IN1 beim L298N Modul
int pinIN2 = 8; // zu Pin IN2 beim L298N Modul

int vmax = 255; // vmax an externe Spannung für Motor anpassen
void setup() {
  pinMode(pinEn,OUTPUT);    //nicht erforderlich, aber hilfreich
  pinMode(pinIN1,OUTPUT);  // pinIN1 mit OUTPUT initialisieren
  pinMode(pinIN2,OUTPUT); // pinIN1 mit OUTPUT initialisieren
}

void loop() {
    analogWrite(pinEn, vmax);     // maximale Geschwindigkeit
    digitalWrite(pinIN1, HIGH);   // Rechts- oder Linksdrehung
    digitalWrite(pinIN2, LOW);
    delay(5000);                  // Ablauf 5000 ms verzögern
    digitalWrite(pinIN1, LOW);
    digitalWrite(pinIN2, LOW);    // Motor stoppen
    delay(2000);                  // Ablauf 2000 ms verzögern
    analogWrite(pinEn, vmax/2);   // halbe Geschwindigkeit
    digitalWrite(pinIN1, HIGH);   // ohne Richtungsänderung
    digitalWrite(pinIN2, LOW);
    delay(5000);                  // Ablauf 5000 ms verzögern
    digitalWrite(pinIN1, LOW);
    digitalWrite(pinIN2, LOW);    // Motor stoppen
    delay(2000);                  // Ablauf 2000 ms verzögern
    analogWrite(pinEn, vmax);     // maximale Geschwindigkeit
    digitalWrite(pinIN1, LOW);    // Drehrichtung entgegengesetzt
    digitalWrite(pinIN2, HIGH);
    delay(5000);                  // Ablauf 5000 ms verzögern
    digitalWrite(pinIN1, LOW);
    digitalWrite(pinIN2, LOW);    // Motor stoppen
    delay(2000);                  // Ablauf 2000 ms verzögern
    analogWrite(pinEn, vmax/2);   // halbe Geschwindigkeit
    digitalWrite(pinIN1, LOW);    // ohne Richtungsänderung
    digitalWrite(pinIN2, HIGH);
    delay(5000);                  // Ablauf 5000 ms verzögern
    digitalWrite(pinIN1, LOW);
    digitalWrite(pinIN2, LOW);    // Motor stoppen
    delay(10000);                 // Ablauf 10000 ms verzögern
}
```

Hinweis:
In einem Roboter kommen meistens zwei Gleichstrommotoren zum Einsatz. Das Modul mit dem L298N hat je zwei Schraubanschlüsse für den Motor A (OUT1, OUT2) und den Motor B (OUT3, OUT4). Da wir für unseren Roboter (vgl. Abschnitt 11) nur mit 4 x 1.5 V = 6 V betreiben, können wir auch auf eine Geschwindigkeitsregelung verzichten. Wir setzen die Jumper bei ENA und ENB. Damit sind ENA und ENB auf 5 V eingestellt und die Motoren laufen mit maximaler Geschwindigkeit.

Möglicher experimenteller Aufbau:

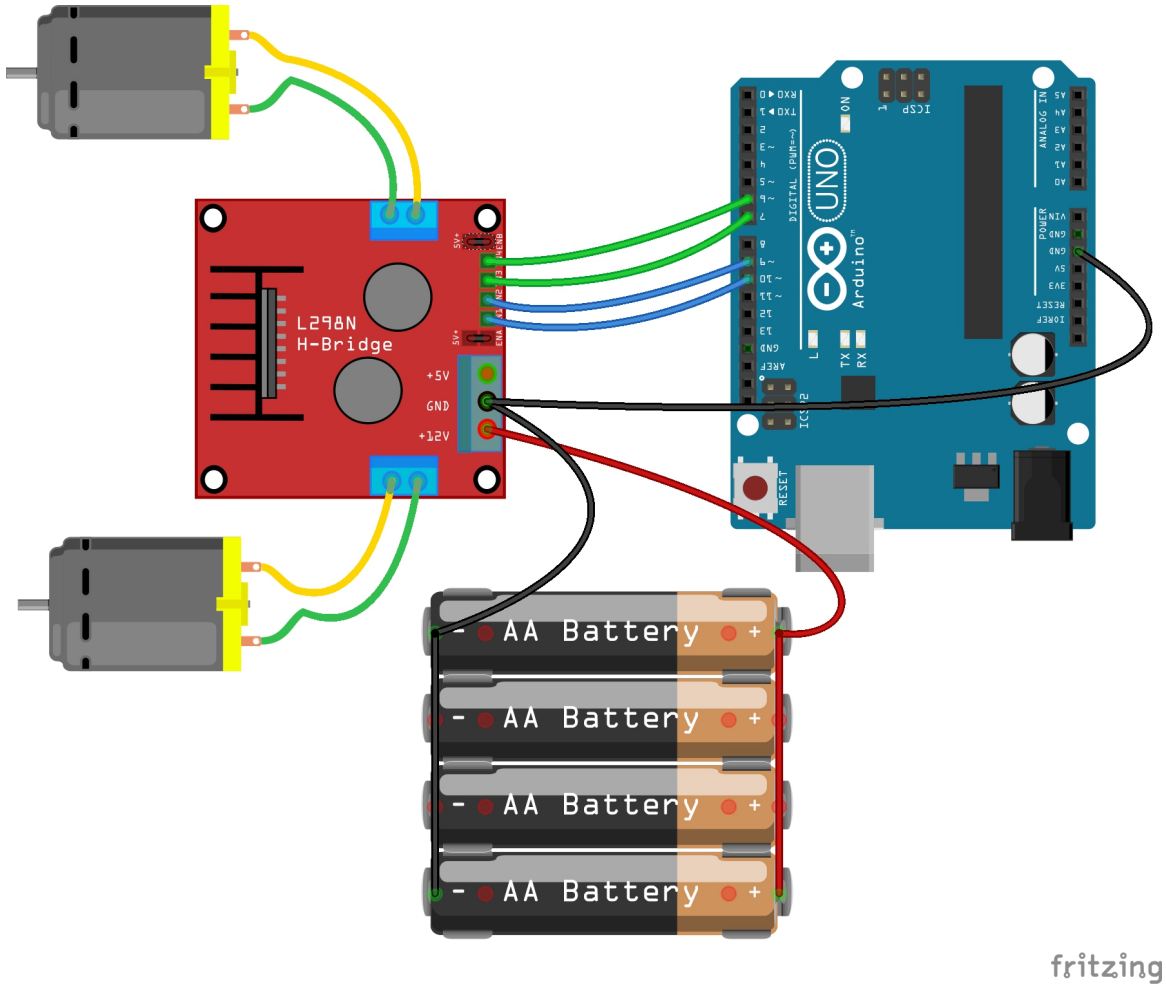


Abbildung 39: Schaltung für zwei Gleichstrommotoren[31]

In der dargestellten Schaltung bestehen folgende Verbindungen zwischen dem L298N Modul und dem Arduino.

L298N-Modul	Arduino
IN1	Pin 10
IN2	Pin 9
IN3	Pin 7
IN4	Pin 6

[31] Diese Abbildung wurde erstellt mit Fritzing (Lizenz: CC BY-SA 3.0)

10 Steuerung eines Servomotors

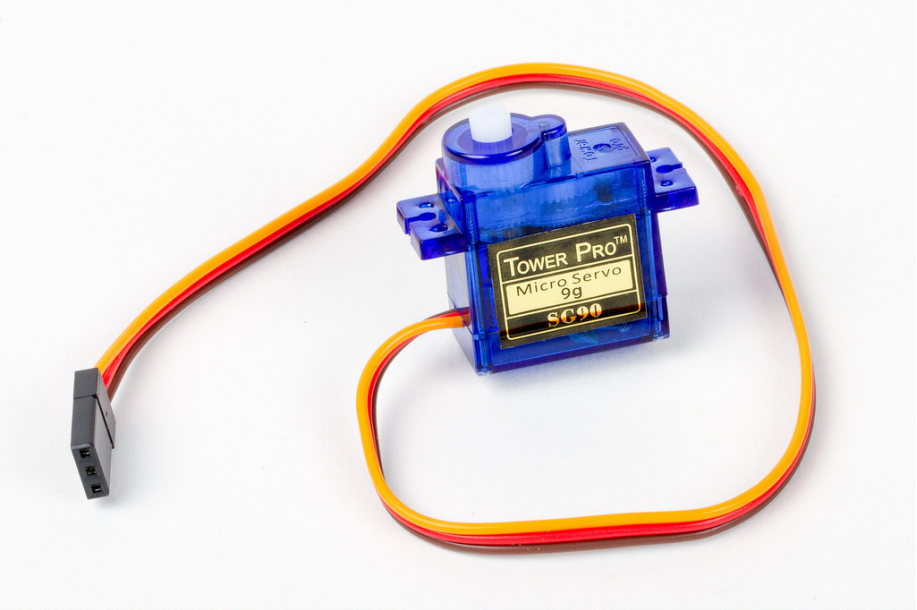


Abbildung 40: Micro Servo[32]

Ein Servomotor, kurz Servo, besteht aus einem Gleichstrommotor und einem Getriebe sowie aus einer Steuerelektronik, welche die Drehung des Motors bestimmt. Die Ansteuerung erfolgt über ein Spannungssignal, das aus Rechteckimpulsen mit Pulsweiten von etwa 1 ms bis zu 2 ms besteht.

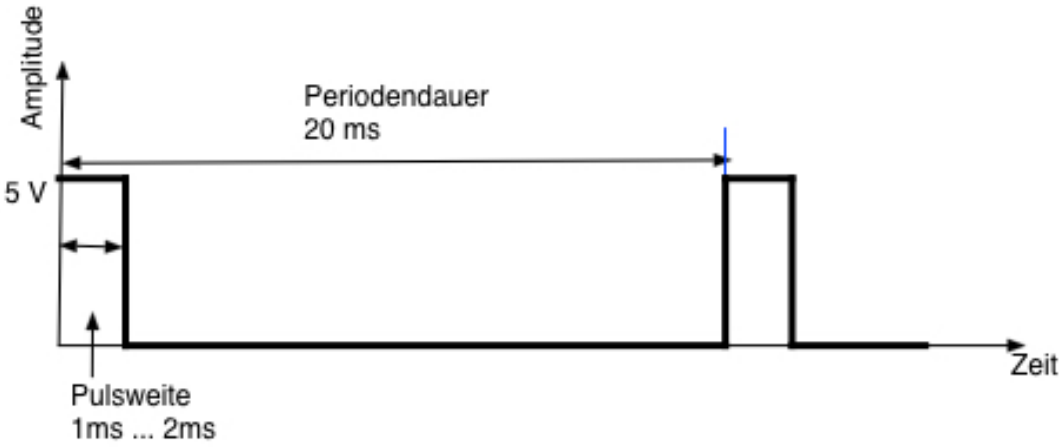


Abbildung 41: Gradeinstellung beim Servo

Wird die Pulsweite von 1 ms auf 2 ms erhöht, dreht sich der Servo von 0° auf 180°. Um die Pulsweiten mit den Drehwinkeln des Servo zu verbinden, gibt es bei Arduino eine Bibliothek `Servo.h` mit den notwendigen Funktionen und der Bereitstellung des Spannungssignals: Periodendauer 20 ms, Pulsweiten zwischen etwa 1 ms und 2 ms. Am Beginn des Programms wird der Inhalt der Bibliothek bekanntgemacht durch

```
#include <Servo.h>
```

[32] „servo-2“ von Windell Oskay via flickr. com, Lizenz: CC BY 2.0
https://www.flickr.com/photos/oskay/22731304279 (17.08.2018)

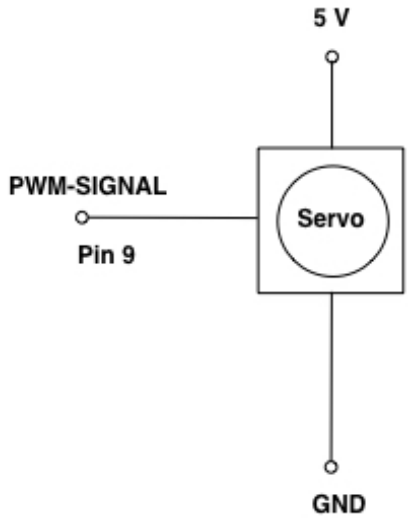
Mit dieser Bibliothek können mehrere Servos gesteuert werden. Wir steuern nur einen Servo. Den nennen wir z.B. `servoSFZ`. Am Programmanfang legen wir dies mit der Vereinbarung

```
Servo servoSFZ;
```

fest. Die Ansteuersignale des Servo müssen von einem PWM-Pin des Arduino ausgehen, z. B. Pin 9. Der Übersichtlichkeit halber geben wir dieser Steuerleitung mit der Vereinbarung

```
int servoSteuer = 9;
```

den Namen `servoSteuer` und weisen dabei den Wert 9 zu. Die Ansteuerung des Servo ist aus der folgenden Schaltskizze zu entnehmen:



Der Servo wird mit 3 Leitungen versorgt.
Der Micro Servo 9g SG90 hat folgende Anschlussbelegungen:

- orange (PWM Signal)
- rot (VCC 5V)
- braun (GND)

Abbildung 42: Anschlussbelegung beim Servo

Im Zusammenhang mit dem Namen `servoSFZ` können verschiedene Funktionen des Servo aufgerufen werden:

1. In der setup-Funktion wird mit der Anweisung

   ```
   servoSFZ.attach(servoSteuer);
   ```

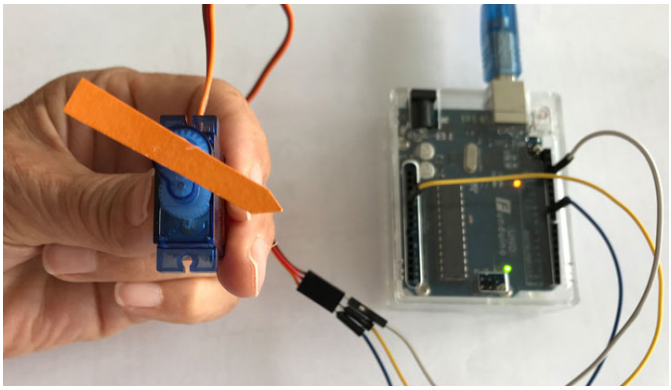
   der Servo mit `servoSteuer` verbunden.

2. In der loop-Funktion kann z. B. durch eine Anweisung

   ```
   servoSFZ.write(90);
   ```

   der Servo auf den Winkel von 90° eingestellt werden.

10.1 AB 11: Servo-Steuerung



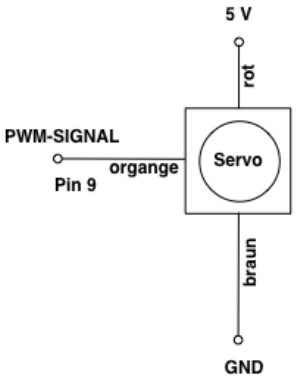


Abbildung 43: Richtungsanzeige beim Servo

Materialien:

- 1 Laptop mit Arduino-Software
- 1 Arduino
- 1 Breadboard
- 4 female/male Kabel
- verschiedene Breadboard Kabel für den Arduino
- Entfernungsmesser mit Halterung (z.B. Gesicht)
- Servo mit aufgeschraubtem Adapter
- Doppelklebebandstreifen

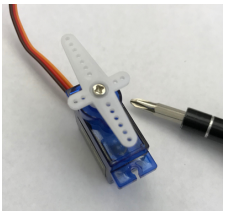
Aufgabe 1:

a) Verbinde den Servo direkt mit dem Arduino.

b) Überprüfe mit dem folgenden Programm die Steuerung deines Servo:

```
// Servo-Ansteuerung mit Servo-Bibliothek
#include <Servo.h>       // Servo-Bibliothek wird bekannt gemacht
Servo servoSFZ;          // Initialisierung des Objekts servoSFZ
int servoSteuer = 9;     // Servo-PWM-Steuerleitung orange

void setup() {
  servoSFZ.attach(servoSteuer);   // servoSteuer mit Servo verbinden
}

void loop() {
  servoSFZ.write(0);              // Servo auf 0° einstellen
  delay(1000);                    // Ablauf 1000 ms verzögern
  servoSFZ.write(90);             // Servo auf 90° einstellen
  delay(1000);                    // Ablauf 1000 ms verzögern
  servoSFZ.write(180);            // Servo auf 180° einstellen
  delay(1000);                    // Ablauf 1000 ms verzögern
}
```

c) Verändere zum Justieren des aufgeschraubten Adapters das Programm so, dass der Servo sich immer auf 90° einstellt. Möglicherweise muss der Adapter abgeschraubt und in der 90° Richtung erneut festgeschraubt werden. Diese Justierungsarbeit muss sorgfältig durchgeführt werden.



Aufgabe 2:

Erweitere das Programm so, dass der Servo von 0° bis 180° und anschließend von 180° wieder nach 0° dreht. Nach jedem Grad-Schritt soll der Servo 10 ms warten. Um diese Schritte automatisch ablaufen zu lassen, setze die `for`-Schleife ein, wie du sie schon beim Lauflicht kennengelernt hast.

Linksdrehung von 0° bis 180°

```
for(int i = 0; i <= 180; ++i) {
  servoSFZ.write(i);
  delay(10);
}
```

Rechtsdrehung von 180° bis 0°

```
for(int i = 180; i >= 0; --i)
{
  servoSFZ.write(i);
  delay(10);
}
```

Aufgabe 3:

Verwende nun einen Servo mit aufgesetztem Gesicht, in das ein Entfernungsmesser eingebaut ist. Das Gesicht in Abbildung 44 wurde mit einem 3D Drucker erstellt. Schreibe ein Programm, das die Entfernung in den Richtungen 0°, 90° und 180° zu dort aufgestellten Gegenständen, z. B. Bücher, misst. Ausgegeben werden soll mit `Serial.print` und `Serial.println` die Richtung und der Abstand in Zentimeter.

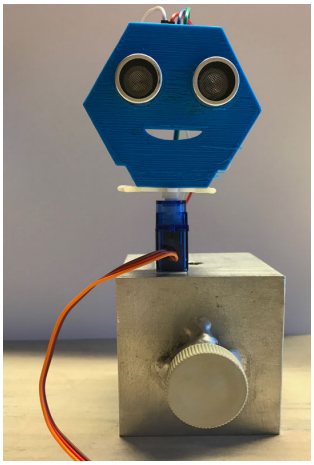


Abbildung 44: Entfernungsmesser auf dem Servo

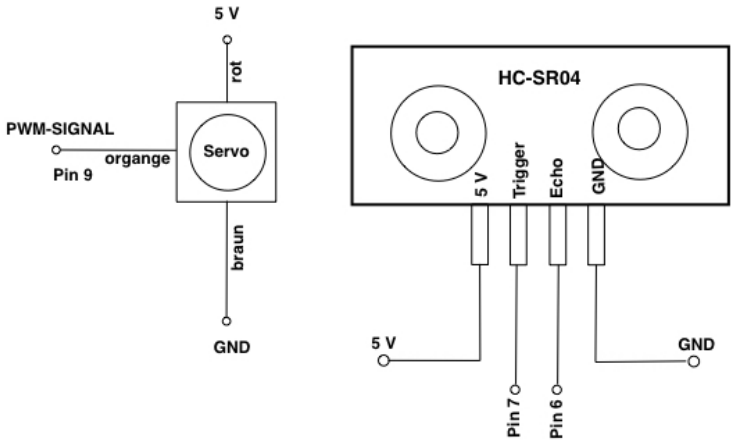


Abbildung 45: Schaltplan für Servo und Entfernungsmesser

10.2 ABL 11: Servo-Steuerung - Lösung

Aufgabe 2:

Mögliches Arduino-Programm

```
#include <Servo.h>

int servoSteuer = 9;      // Servo-PWM-Steuerleitung orange
Servo servoSFZ;           // Servo-Objekt initialisieren

void setup() {
  servoSFZ.attach(servoSteuer); // servoSteuer mit Servo verbinden
}

void loop() {
  for (int i = 0; i <= 180; ++i) {
    servoSFZ.write(i);
    delay(10);
  }
  for (int i = 180; i >= 0; --i) {
    servoSFZ.write(i);
    delay(10);
  }

}
```

Aufgabe 3:

Mögliches Arduino-Programm in Langversion:

```
#include <Servo.h>

int servoSteuer = 9;     // Servo-PWM-Steuerleitung orange
Servo servoSFZ;          // Servo-Objekt initialisieren

int trigger = 7;         // Trigger-Pin des Ultraschallsensors
int echo = 6;            // Echo-Pin des Ultraschallsensors
int dauer;               // Laufzeit des Schalls in Mikrosekunden
double entfernung;       // Entfernung in cm zum Objekt

void setup() {
  pinMode(trigger, OUTPUT); // trigger Pin mit OUTPUT initialisieren
  pinMode(echo, INPUT);
  Serial.begin(9600);

  servoSFZ.attach(servoSteuer); // servoSteuer mit Servo verbinden
}

void loop() {
  servoSFZ.write(0);           // Servo auf 0° einstellen
```

```
  digitalWrite(trigger, LOW);   // Burst vorbereiten
  delay(2);                     // Ablauf 2 ms verzögern
  digitalWrite(trigger, HIGH);
  delayMicroseconds(10);        // Ablauf 10 µs verzögern
  digitalWrite(trigger, LOW);  // Burst startet in 250 Mikrosek.
  dauer= pulseIn(echo, HIGH);  // Laufzeit in Mikrosekunden
  entfernung = dauer*0.0344/2; // Rückgabe der Entfernung in cm

  if (entfernung > 300 || entfernung < 2)
  {
    Serial.println("Kein Messwert!");
  }
  else
  {
    Serial.print("Entfernung = ");
    Serial.print(entfernung);
    Serial.println(" cm");
  }
  delay(3000);                  // Ablauf 3000s verzögern

  servoSFZ.write(90);           // Servo auf 90° einstellen
```

```
  digitalWrite(trigger, LOW);  // Burst vorbereiten
  delay(2);                    // Ablauf 2 ms verzögern
  digitalWrite(trigger, HIGH);
  delayMicroseconds(10);       // Ablauf 10 µs verzögern
  digitalWrite(trigger, LOW);  // Burst startet in 250 Mikrosek.
  dauer= pulseIn(echo, HIGH);  // Laufzeit in Mikrosekunden
  entfernung = dauer*0.0344/2; // Rückgabe der Entfernung in cm

  if (entfernung > 300 || entfernung < 2)
  {
    Serial.println("Kein Messwert!");
  }
  else
  {
    Serial.print("Entfernung = ");
    Serial.print(entfernung);
    Serial.println(" cm");
  }
  delay(3000);                 // Ablauf 3000 s verzögern

  servoSFZ.write(180);         // Servo auf 180° einstellen

  digitalWrite(trigger, LOW);  // Burst vorbereiten
  delay(2);                    // Ablauf 2 ms verzögern
  digitalWrite(trigger, HIGH);
  delayMicroseconds(10);       // Ablauf 10 µs verzögern
  digitalWrite(trigger, LOW);  // Burst startet in 250 Mikrosek.
```

```
  dauer= pulseIn(echo, HIGH);  // Laufzeit in Mikrosekunden
  entfernung = dauer*0.0344/2; // Rückgabe der Entfernung in cm
  if (entfernung > 300 || entfernung < 2)
  {
    Serial.println("Kein Messwert!");
  }
  else
  {
    Serial.print("Entfernung = ");
    Serial.print(entfernung);
    Serial.println(" cm");
  }
  delay(3000);              // Ablauf 3000 s verzögern
}
```

Hinweis zu Funktionen:
Bei unserem Programm wird für jeden eingestellten Winkel eine Entfernungsmessung und eine Ausgabe der Entfernung durchgeführt. Wir können dies vereinfachen, indem wir für die Entfernungsmessung eine Funktion und für die Ausgabe der Entfernung ebenfalls eine Funktion festlegen. Diese beiden Funktionen können wir dann für jeden eingestellten Winkel aufrufen.

Struktur einer Funktion:

```
Rückgabetyp Funktionsname (Parameterliste) {
   .... Anweisungen ....
   return Ergebnis;
}
```

Beispiel:

```
int laufzeitMessung(int echo, int trigger) {
   // ... Anweisungen ...
   return dauer;
}
```

Wenn eine Funktion kein Ergebnis zurückgibt, steht für den Rückgabetyp `void` und die letzte Anweisung lautet `return` oder kann entfallen. Wenn keine Parameter der Funktion übergeben werden, steht zwischen den runden Klammern hinter dem Funktionsnamen `void` oder kann leer gelassen werden.

Beispiel:

```
void setup(void) {
   // ... Anweisungen ...
   return;
}
```

oder

```
void setup() {
   // ... Anweisungen ...
}
```

Hinweis zur Arduino IDE:
Die Funktionen können beim Arduino IDE in einem eigenen TAB gespeichert werden. Der C++ Compiler sucht dort automatisch nach den notwendigen Funktionen.

Durch Anklicken des kleinen Dreiecks oben rechts lässt sich ein neuer Tab erstellen.

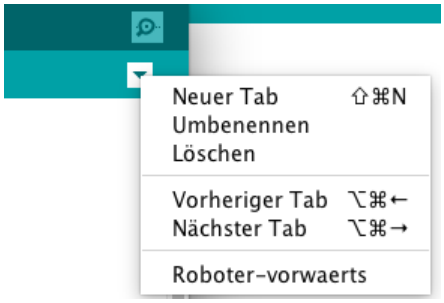


Abbildung 46: Neuen Tab erzeugen[33]

Im ersten Tab steht dann das bekannte Programm mit der `setup` und `loop` Funktion. Im zweiten Tab können die von dir erstellten Funktionen abgelegt werden.

Abbildung 47: Funktionen in neuem Tab[34]

Für die Funktion zur Entfernungsmessung können wir eingeben:

```
double entfernungMessung(int trigger, int echo) {
  int dauer;             // Laufzeit des Schalls in Mikrosek.

  digitalWrite(trigger, LOW);   // Burst vorbereiten
  delay(2);                     // Ablauf 2 ms verzögern
  digitalWrite(trigger, HIGH);
  delayMicroseconds(10);        // Ablauf 10 µs verzögern
  digitalWrite(trigger, LOW);   // Burststart
  dauer= pulseIn(echo, HIGH);   // Laufzeit in Mikrosek.
  return dauer*0.0344/2;        // Rückgabe der Entfernung in cm
}
```

Vor dem Funktionsnamen steht der Datentyp `double` des Ergebnisses. In der Klammer nach dem Funktionsnamen stehen die Variablen für den Echo-Pin und für den Trigger-Pin. In der letzten Zeile der Funktion wird mit

```
return (dauer/2)*0.0344;
```

die Entfernung in cm zurückgegeben. Für die Funktion zur Ausgabe einer Entfernung können wir eingeben:

```
void ausgabeEntfernung(double entfernung){
```

[33] Screenshot aus der IDE (Arduino Version 1.8.3)
[34] Screenshot aus der IDE (Arduino Version 1.8.3)

```
  if (entfernung > 300 || entfernung < 2)
  {
    Serial.println("Kein Messwert!");
  }
  else
  {
    Serial.print("Entfernung = ");
    Serial.print(entfernung);
    Serial.println(" cm");
  }
}
```

Diese Funktion hat keinen Ergebniswert, deshalb ist der Typ `void`. Damit die Funktion weiß, was sie ausgeben soll, wird ihr die Variable `entfernung` übergeben, wie es in der Klammer nach dem Funktionsnamen steht.

Aufgabe 3:

Mögliches Arduino-Programm in Kurzversion:

Hauptteil – Tab1

```
// Servo-Ansteuerung mit Servo-Bibliothek
#include <Servo.h>

int servoSteuer = 9;       // Servo-PWM-Steuerleitung orange
Servo servoSFZ;            // Servo-Objekt initialisieren

int trigger = 7;           // Trigger-Pin des Ultraschallsensors
int echo = 6;              // Echo-Pin des Ultraschallsensors
int dt = 200; // Zeitpuffer zwischen Drehung u. Entfernungsmessung

void setup() {
  servoSFZ.attach(servoSteuer); // servoSteuer mit Servo verbinden
  pinMode(trigger, OUTPUT); // trigger Pin mit OUTPUT initialisieren
  pinMode(echo, INPUT);     // echo Pin mit INPUT initialisieren
  Serial.begin(9600);
}

void loop() {
  double entfernung;            // Entfernung in cm zum Objekt

  servoSFZ.write(0);
  delay(dt);                    // Ablauf dt s verzögern
  entfernung = entfernungMessung(trigger, echo);
  ausgabeEntfernung(entfernung);
  delay(3000);                  // Ablauf 3000s verzögern

  servoSFZ.write(90);
  delay(dt);                    // Ablauf dt s verzögern
  entfernung = entfernungMessung(trigger, echo);
  ausgabeEntfernung(entfernung);
  delay(3000);                 // Ablauf 3000 s verzögern

  servoSFZ.write(180);
```

```
  delay(dt);                    // Ablauf dt s verzögern
  entfernung = entfernungMessung(trigger, echo);
  ausgabeEntfernung(entfernung);
  delay(3000);                  // Ablauf 3000 s verzögern
}
```

Tab2 - Funktionen

```
double entfernungMessung(int trigger, int echo) {
  int dauer;                // Laufzeit des Schalls in Mikrosek.
  digitalWrite(trigger, LOW);   // Burst vorbereiten
  delay(2);                     // Ablauf 2 ms verzögern
  digitalWrite(trigger, HIGH);
  delayMicroseconds(10);        // Ablauf 10 µs verzögern
  digitalWrite(trigger, LOW);   // Burststart
  dauer = pulseIn(echo, HIGH);  // Laufzeit in Mikrosek.
  return dauer*0.0344/2;        // Rückgabe der Entfernung in cm
}

void ausgabeEntfernung(double entfernung){
  if (entfernung > 300 || entfernung < 2)
  {
    Serial.println("Kein Messwert!");
  }
  else
  {
    Serial.print("Entfernung = ");
    Serial.print(entfernung);
    Serial.println(" cm");
  }
}
```

Die Parameter `trigger` und `echo` in der Funktion entfernungMessung(trigger, echo) können entfallen, wenn die Variablen trigger und echo vorab festgelegt worden sind.

11 Steuerung eines autonomen Roboters

Im Folgenden werden verschiedene Steuerungen und Sensormessungen in einem autonomen Roboter eingesetzt. Der Roboter besteht aus zwei Antriebsrädern und einem Stützrad. Im Internet findet man komplette Bausätze. Die Autoren des Skripts können entsprechende Links geben. Diese Bausätze enthalten alle notwendigen Bauteile. Die Kosten für den Bausatz betragen etwa 20 Euro.
Achtung: Arduino Billigprodukte können bei der Programmübertragung Probleme bereiten!

Wir können mit den Erfahrungen aus den Abschnitten 8, 9 und 10 bereits einen Roboter konfigurieren. Hierbei macht es Sinn, eine Roboter Plattform mit 2 Gleichstrommotoren, Reifen und Batteriefach von einschlägigen Lieferanten einzusetzen. Mit Doppelklebeband und passenden Adapterstücken aus dem 3D-Drucker können Arduino, 298N-Modul oder Entfernungsmesser befestigt werden.

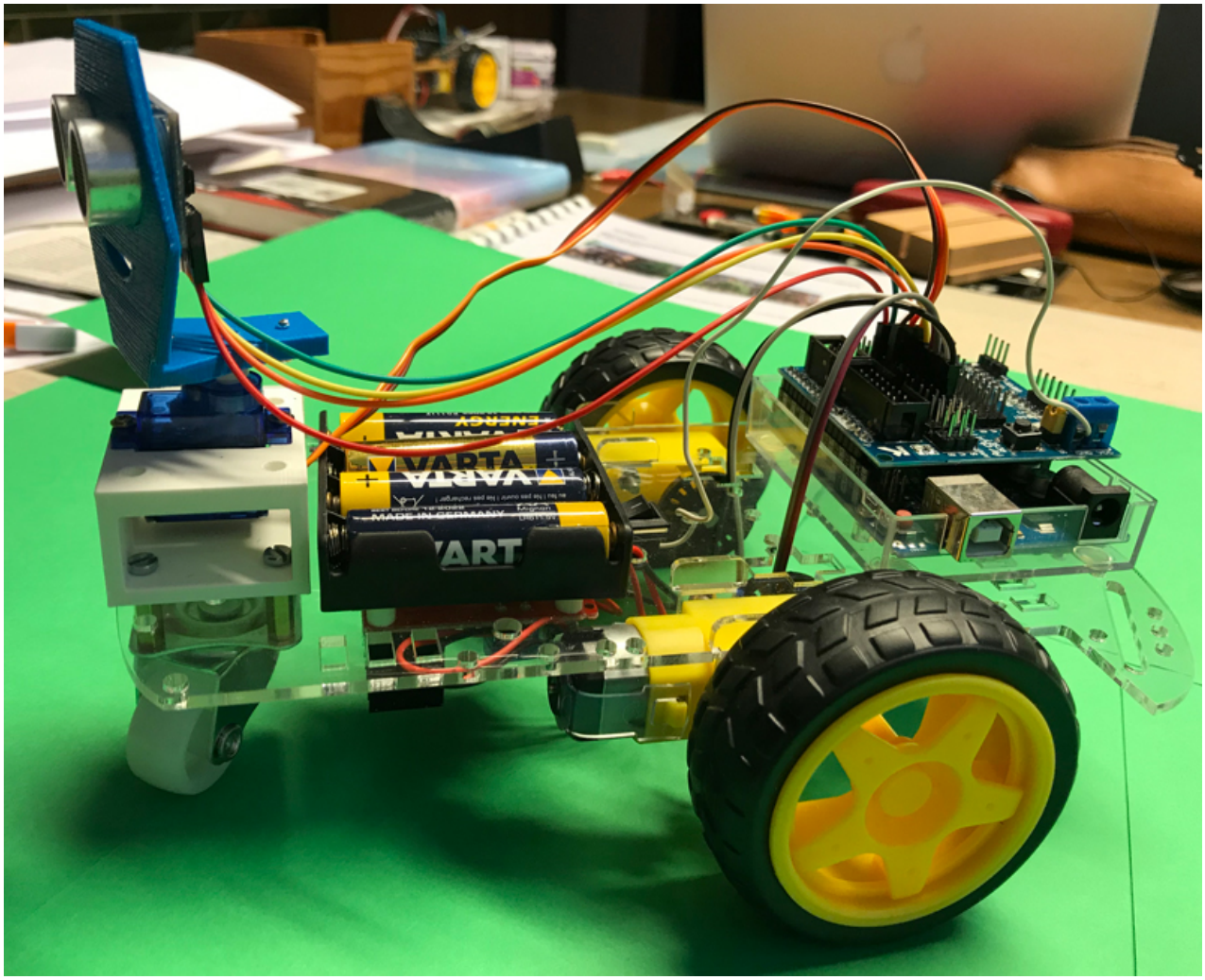
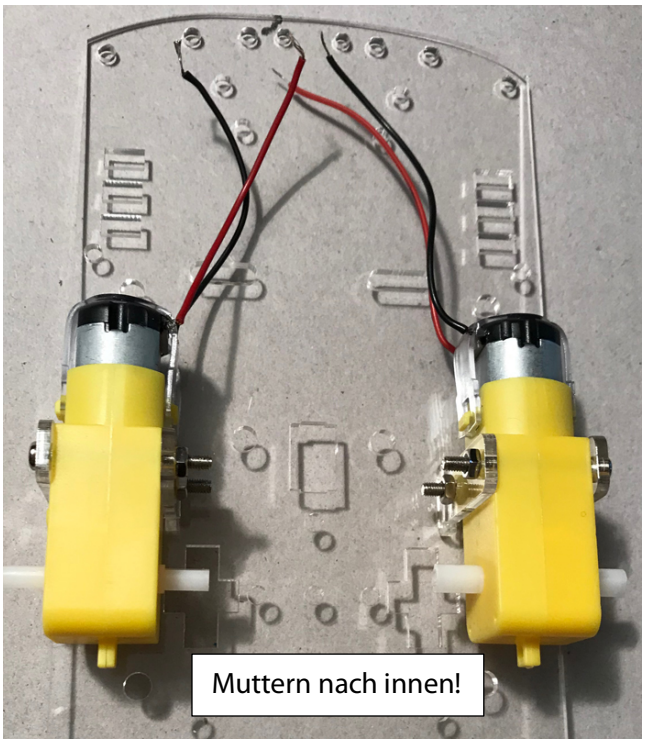


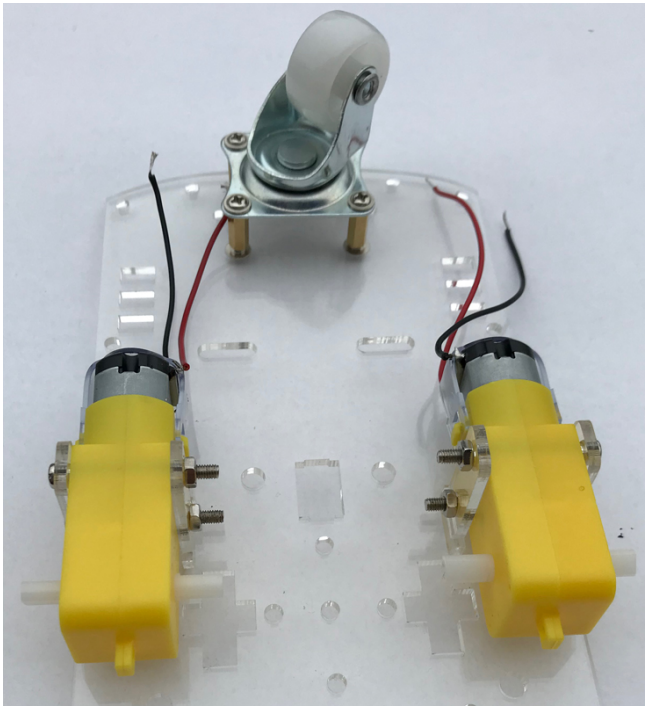
Abbildung 48: Unser autonomer Roboter

Mögliche Schritte beim Zusammenbau des Roboters

1. Anschlussdrähte für die Elektromotoren passend kürzen und anlöten.
2. Einbau der beiden Elektromotoren auf der Trägerplatte mithilfe der vier Kunststofflaschen.

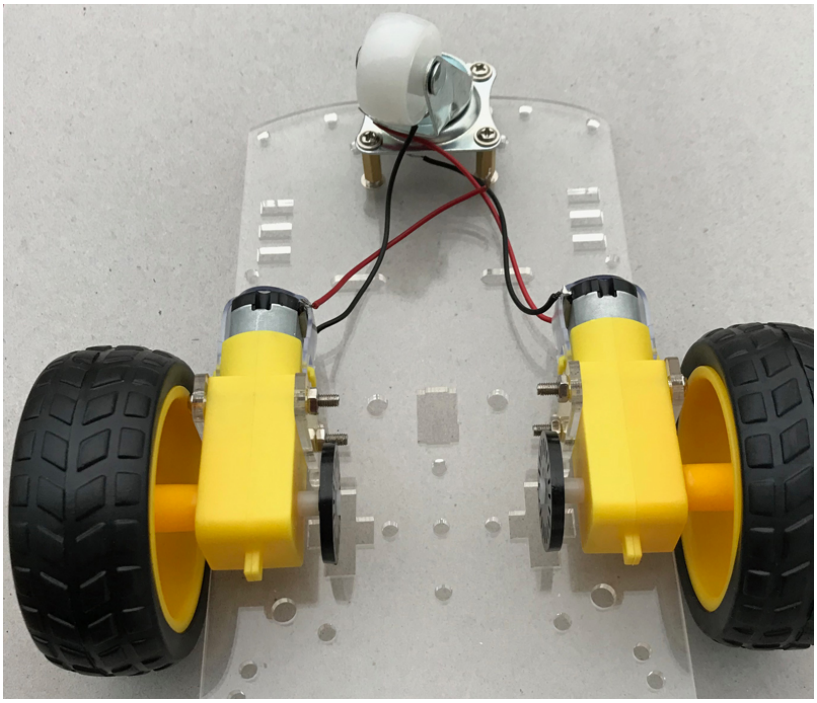


3. Befestigung des Stützrades mit Abstandshaltern auf der Unterseite und der Servo-Halterung auf der Oberseite



Servo-Halterung jetzt auf der Oberseite mit 4 Schrauben (3 x 20 mm) aus dem Baumarkt befestigen.

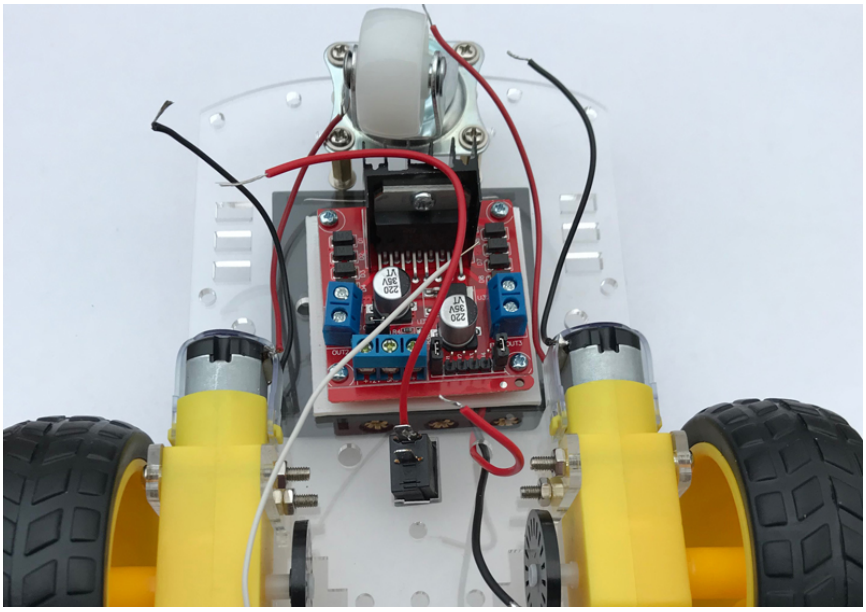
4. Montage der Antriebsräder und der inneren Konterscheiben.



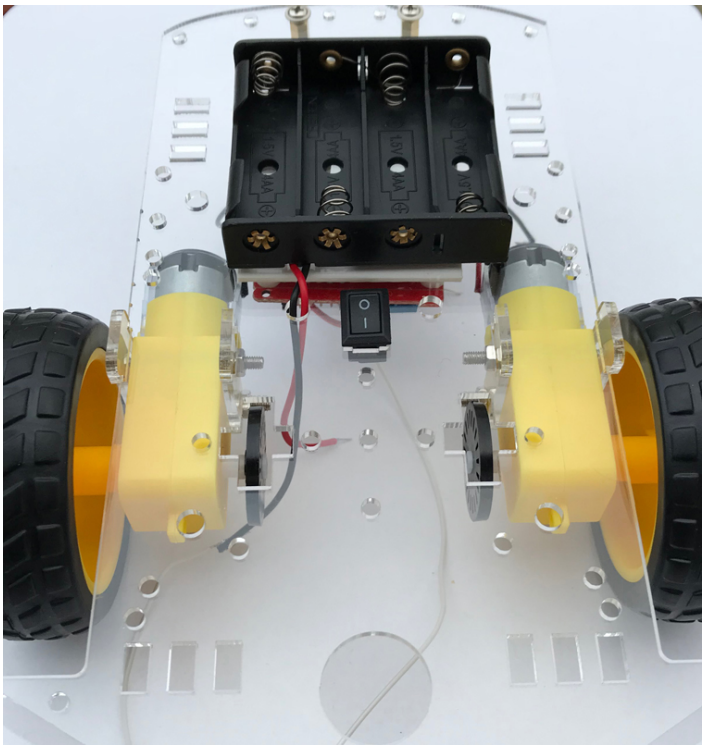
Hinweis:

- Öffnung der Radnabe und Form der Motorachse anpassen.
- Beim Andrücken der Räder auf entsprechenden Gegendruck von der Innenseite her achten, um die Verankerung der Elektromotoren nicht zu beschädigen.
- Räder nicht mit der Motorachse drehen!

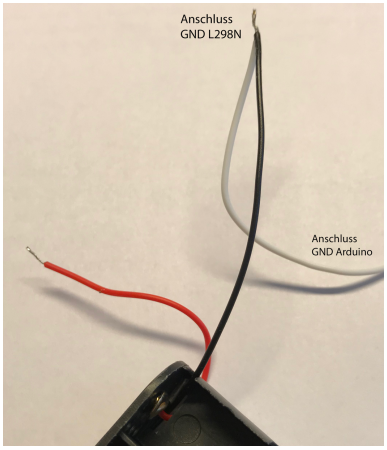
5. Montage des L298N-Treibermoduls auf der Unterseite der Trägerplatte mit 10-mm-Abstandshaltern, die mit einem 3D-Drucker hergestellt werden können. Evtl. müssen auf der Trägerplatte vier Löcher (Durchmesser 3 mm) gebohrt werden. Befestigung mit 4 Schrauben (3 x 20 mm) aus dem Baumarkt. Man kann auch eine für den L298N passende Trägerplatte mit dem 3D-Drucker erstellen und mit Doppelklebeband befestigen. Das Stützrad muss sich frei drehen können!
6. Den Ein-Aus-Schalter in die vorgesehene Öffnung einklinken. Der Draht vom Pluspol des AA-Batteriefachs wird über den Schalter mit dem Schraubanschluss +12V verbunden.



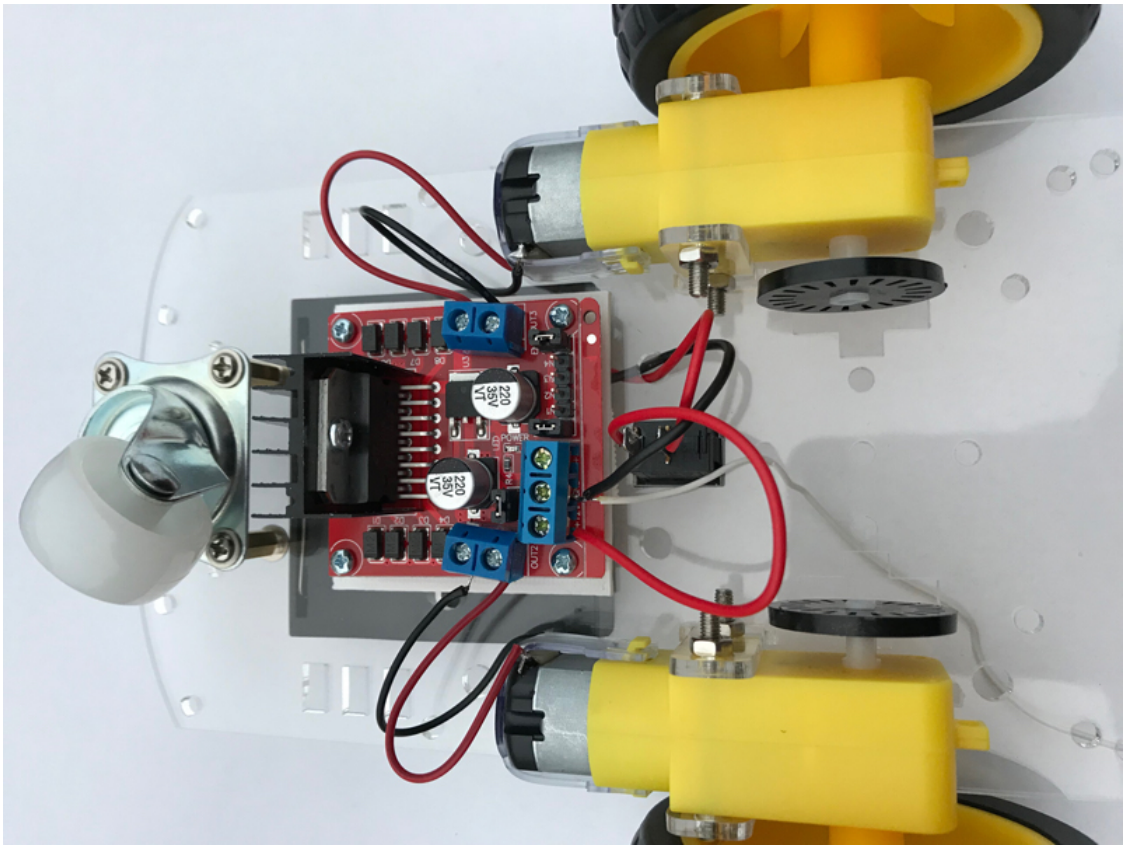
7. Befestigung des AA-Batteriefachs auf der Oberseite



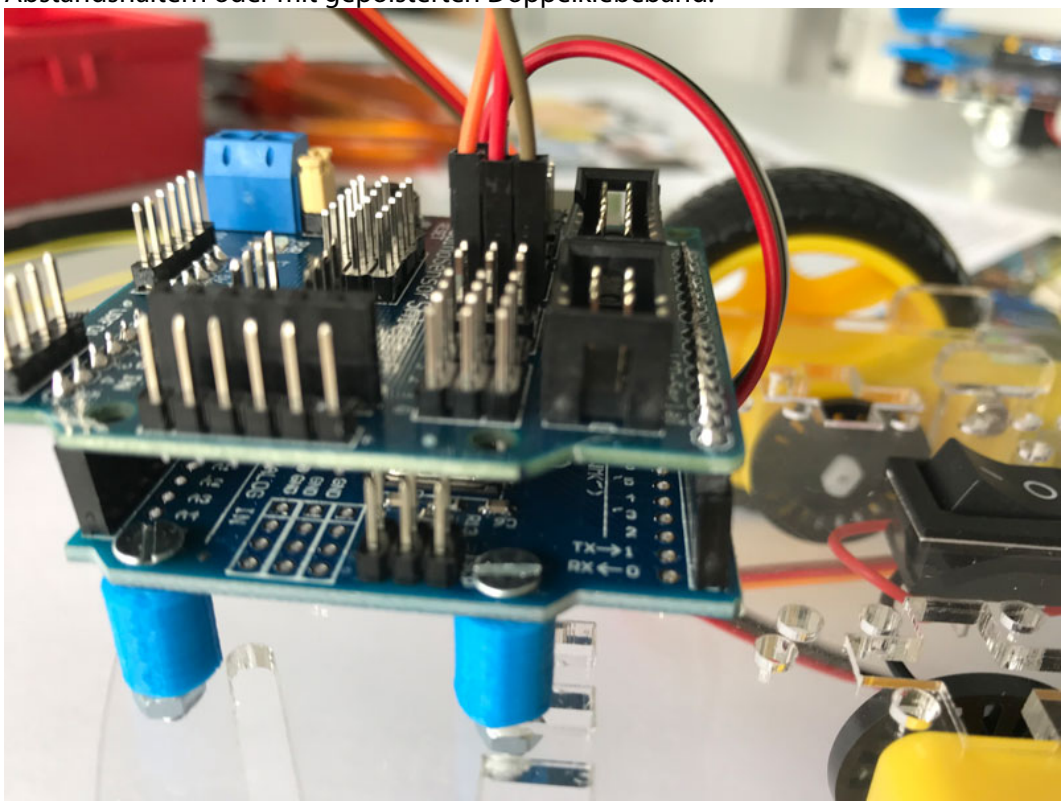
Hinweis:
Den Draht vom Minuspol der Batterien mit einem Verbindungsdraht zum GND des Aduino verknüpfen. Die Verbindungsstelle wird mit dem GND des L298N Moduls verbunden.



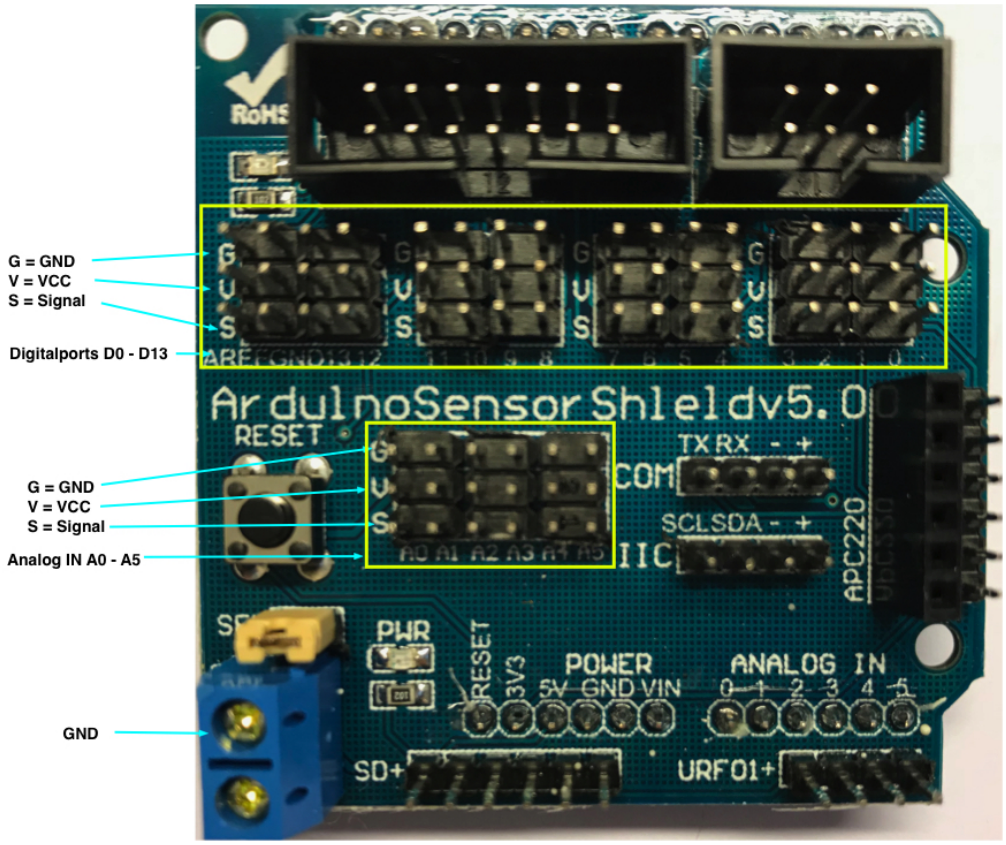
8. Anschluss der Elektromotoren am Treibermodul.



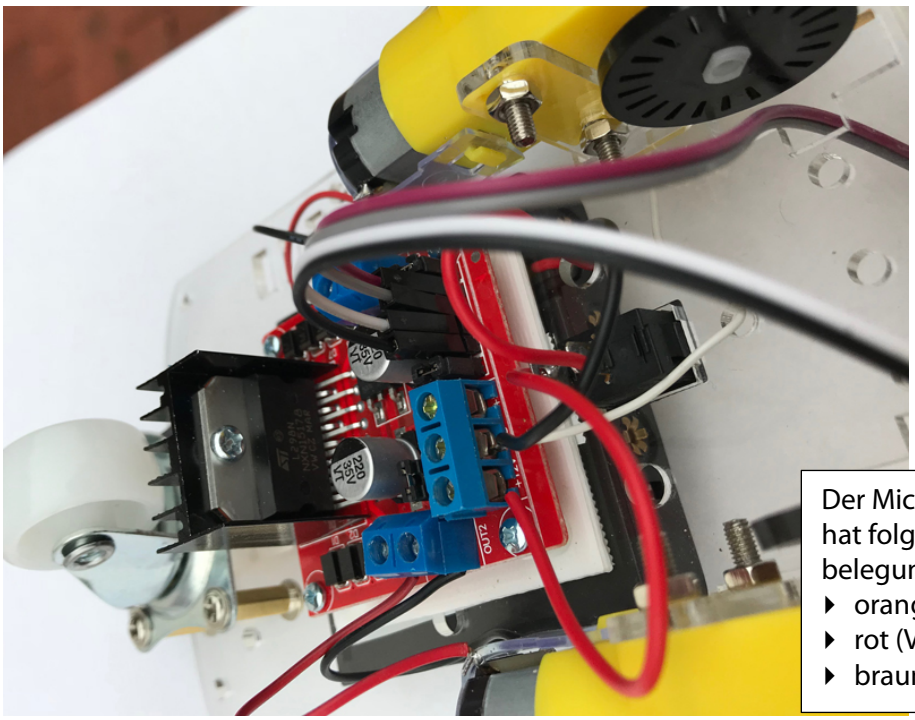
9. Befestigung des Arduino mit aufgesetztem Sensor Shield V 5.0 auf der Trägerplatte mit 10-mm-Abstandshaltern oder mit gepolsterten Doppelklebeband.



10. Anschlüsse auf dem Arduino Sensor Shield V5.0



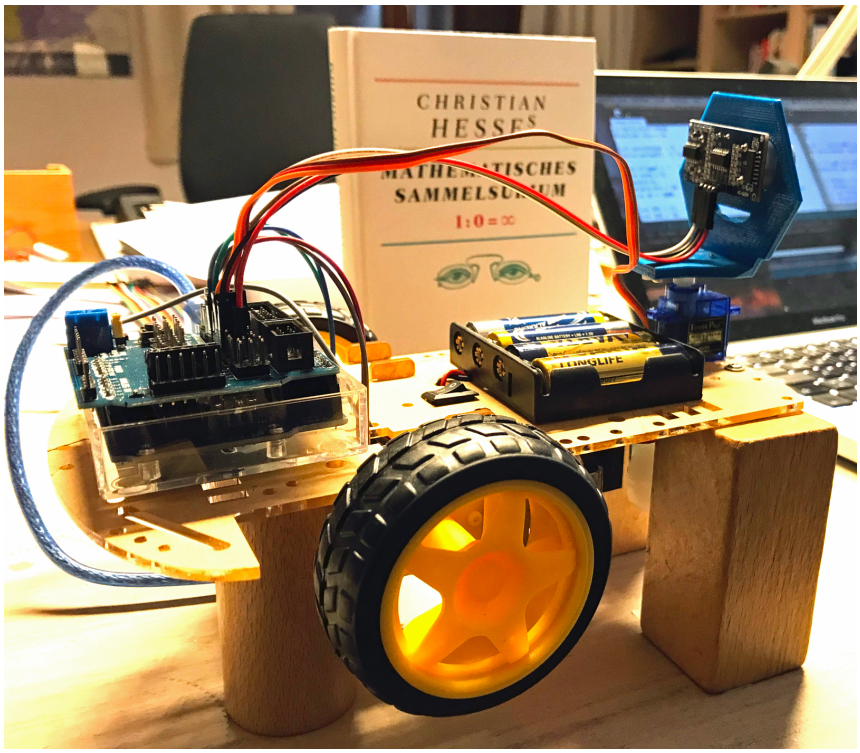
11. IN1, IN2, IN3, IN4 Anschlüsse des L298N mit dem Arduino verbinden wie im Programm vorgesehen.



Der Micro Servo 9g SG90 hat folgende Anschluss-belegungen:

- orange (PWM Signal)
- rot (VCC 5V)
- braun (GND)

12. Den Servo auf der Trägerplatte befestigen. Die Pinbelegung am Arduino für den Servo findet man im Programm.
13. Den Entfernungsmesser zuerst mit den im Programm festgelegten Pins am Arduino verbinden und dann in einer Halterung befestigen.
14. Den Roboter zum Labortest auf die Hebebühne setzen.



12 Test der einzelnen Komponenten des autonomen Roboters

Die Funktionen des Servos und des Entfernungsmessers sind bereits überprüft worden.

12.1 Roboter vorwärts fahren

Beide Räder müssen sich vorwärts drehen. Gegebenenfalls müssen in der Funktion `void vorwaertsRoboter (in ms)` LOW und HIGH angepasst werden.
Um einen realen Test durchzuführen, müssen wir auch die Programmanweisungen für die Servo-Einstellung (Servo auf geradeaus einstellen) und die Funktion `double entfernungMessung ()` einfügen.
Damit die Einstellungen beim Test auch im autonomen Roboter richtig arbeiten, sollte die Funktion `void vorwaertsRoboter (in ms)im` Testprogramm sofort mit der entsprechenden Funktion im Programm Roboter-autonom abgeglichen werden.

Hauptteil – Tab1

```
#include <Servo.h> // Servo-Bibliothek

Servo servoSFZ; // Servo-Objekt festlegen
int IN1 = 7;
int IN2 = 8;
int IN3 = 9;
int IN4 = 10;
int w_v = 90;  // Winkelstellung des Servo in Vorwärtsrichtung
int trigger = 4;  // Trigger-Pin des Ultraschallsensors
int echo = 5;     // Echo-Pin des Ultraschallsensors
int servoSteuer = 6; // Servo-PWM-Steuerleitung orange

double e;         // Entfernung in cm zum Objekt
double min = 30;  // minimaler Abstand in cm zum Hindernis
int dt = 100;    // Zeitpuffer zw. Drehen und Entfernungmessen

void setup() {
   pinMode(IN1, OUTPUT);
   pinMode(IN2, OUTPUT);
   pinMode(IN3, OUTPUT);
   pinMode(IN4, OUTPUT);
   servoSFZ.attach(servoSteuer); // servoSteuer mit Servo verbinden
   pinMode(trigger, OUTPUT); // trigger Pin mit OUTPUT initialisieren
   pinMode(echo, INPUT);     // echo Pin mit INPUT initialisieren

}

void loop() {
    servoSFZ.write(w_v); // Servo auf geradeaus einstellen
    delay(dt);               // Ablauf dt ms verzögern
    e = entfernungMessung();
    delay(dt);               // Ablauf dt ms verzögern
    if (e > 300 || e < 2)
    {
      stoppMotoren();
      delay(dt);
```

```
    }
    else
    {
      if (e > min)
      {
        vorwaertsRoboter(10); // 10 ms vorwaerts fahren
      }
      else
      {
        stoppMotoren();
      }
    }
}
```

Tab2 – Funktionen

```
void stoppMotoren(void) {
  // beide Motoren stoppen
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, LOW);
}

void vorwaertsRoboter(int ms) {
  // Roboter vorwaerts laufen lassen
  digitalWrite(IN1, LOW); // rechtes Rad vorwärts
  digitalWrite(IN2, HIGH);
  digitalWrite(IN3, LOW); // linkes Rad vorwärts
  digitalWrite(IN4, HIGH);
  delay(ms);
  stoppMotoren();
}

double entfernungMessung() {
  int dauer; // Laufzeit des Schalls in Mikrosekungen
  digitalWrite(trigger,LOW); // Burst vorbereiten
  delay(2);                  // Ablauf 2 ms verzögern
  digitalWrite(trigger,HIGH);
  delayMicroseconds(10);     // Ablauf 10 µs verzögern
  digitalWrite(trigger, LOW); // Burststart
  dauer= pulseIn(echo,HIGH);
  return dauer*0.0344/2; // Rueckgabe der Entfernung in cm
}
```

12.2 Roboter rechts drehen

Das linke Rad dreht sich vorwärts und das rechte Rad rückwärts. Damit die Einstellungen beim Test auch im autonomen Roboter richtig arbeiten, sollten das Testprogramm Roboter-rechts-drehen sofort mit der entsprechenden Funktion im Programm Roboter-autonom abgeglichen werden.

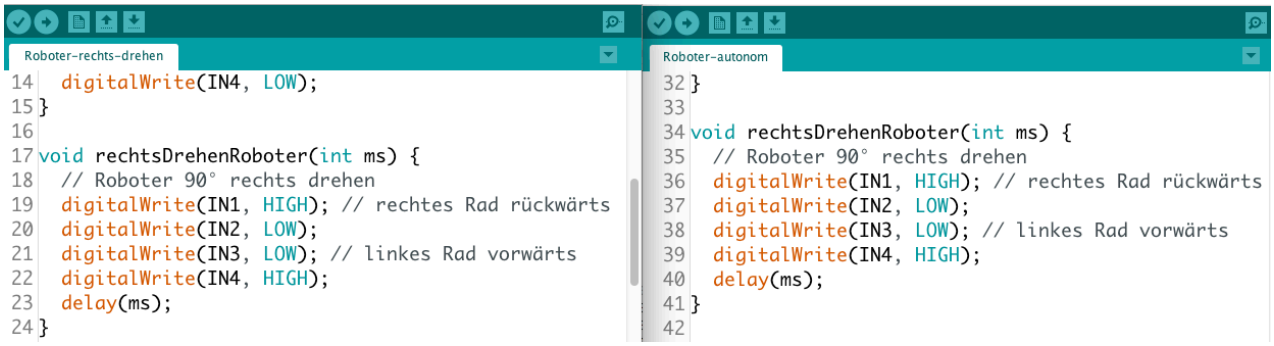


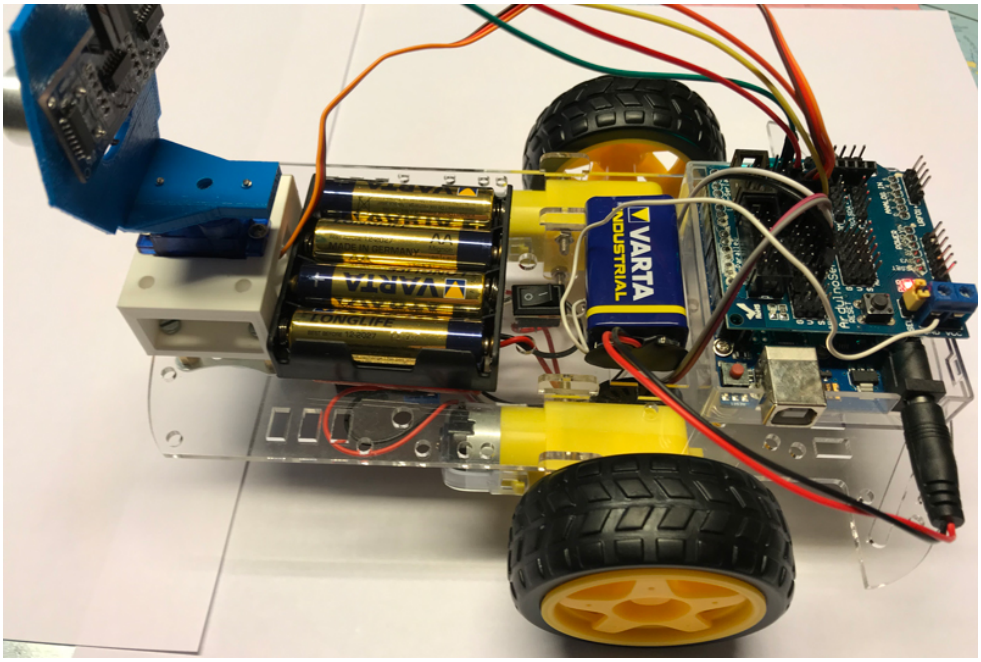
Abbildung 49: Abgleich Roboter-rechts-drehen mit Roboter-autonom[35]

```
int IN1 = 11;
int IN2 = 10;
int IN3 = 9;
int IN4 = 8;
int dt_r = 330; // Zeit für gegenläufiges Drehen der Räder

void stoppMotoren() {
  // beide Motoren stoppen
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, LOW);
}

void rechtsDrehenRoboter(int ms) {
  // Roboter 90° rechts drehen
  digitalWrite(IN1, LOW); // rechtes Rad rückwärts
  digitalWrite(IN2, HIGH);
  digitalWrite(IN3, HIGH); // linkes Rad vorwärts
  digitalWrite(IN4, LOW);
  delay(ms);                // Ablauf ms ms verzögern
  stoppMotoren();
}

void setup() {
   pinMode(IN1, OUTPUT);
   pinMode(IN2, OUTPUT);
   pinMode(IN3, OUTPUT);
   pinMode(IN4, OUTPUT);
}

void loop() {
    stoppMotoren();
```

[35] Screenshot aus der IDE (Arduino Version 1.8.3)
23.04.19

```
    delay(1000);          // Ablauf 1000 ms verzögern
    rechtsDrehenRoboter(dt_r);
}
```

Hinweis:
Die Rechtsdrehung des Roboters sollte bereits jetzt auf dem Boden getestet werden. Der Roboter wird hierbei mit der Spannung einer 9V Batterie versorgt.



12.3 Roboter links drehen

Das linke Rad dreht sich rückwärts und das rechte Rad vorwärts. Damit die Einstellungen beim Test auch im autonomen Roboter richtig arbeiten, sollten das Testprogramm Roboter-links-drehen wie in Abschnitt 12.2 sofort mit der entsprechenden Funktion im Programm Roboter-autonom abgeglichen werden.

```
int IN1 = 11;
int IN2 = 10;
int IN3 = 9;
int IN4 = 8;
int dt_l = 330; // Zeit für gegenläufiges Drehen der Räder


void stoppMotoren() {
  // beide Motoren stoppen
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, LOW);
}

void linksDrehenRoboter(int ms) {
```

```
  // Roboter 90° rechts drehen
  digitalWrite(IN1, HIGH); // rechtes Rad vorwärts
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, LOW); // linkes Rad rückwärts
  digitalWrite(IN4, HIGH);
  delay(ms);               // Ablauf ms ms verzögern
  stoppMotoren();
}

void setup() {
   pinMode(IN1, OUTPUT);
   pinMode(IN2, OUTPUT);
   pinMode(IN3, OUTPUT);
   pinMode(IN4, OUTPUT);
}

void loop() {
    stoppMotoren();
    delay(1000);
    linksDrehenRoboter(dt_l);
}
```

Hinweis:
Die Linksdrehung des Roboters sollte bereits jetzt auf dem Boden getestet werden. Der Roboter wird hierbei mit der Spannung einer 9V Batterie versorgt.

12.4 Roboter macht eine Kehrtwende

Der Roboter macht dabei eine doppelte Rechtsdrehung oder eine doppelte Linksdrehung. Damit die Einstellungen beim Test auch im autonomen Roboter richtig arbeiten, sollten das Testprogramm Roboter-umdrehen wie in Abschnitt 12.2 sofort mit der entsprechenden Funktion im Programm Roboter-autonom abgeglichen werden.

```
int IN1 = 11;
int IN2 = 10;
int IN3 = 9;
int IN4 = 8;
int dt_ret = 660; // Zeit für gegenläufiges Drehen der Räder

void stoppMotoren() {
  // beide Motoren stoppen
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, LOW);
}

void umkehrenRoboter(int ms) {
  // Roboter umdrehen
  digitalWrite(IN1, LOW); // rechtes Rad rückwärts
  digitalWrite(IN2, HIGH);
  digitalWrite(IN3, HIGH); // linkes Rad vorwärts
  digitalWrite(IN4, LOW);
```

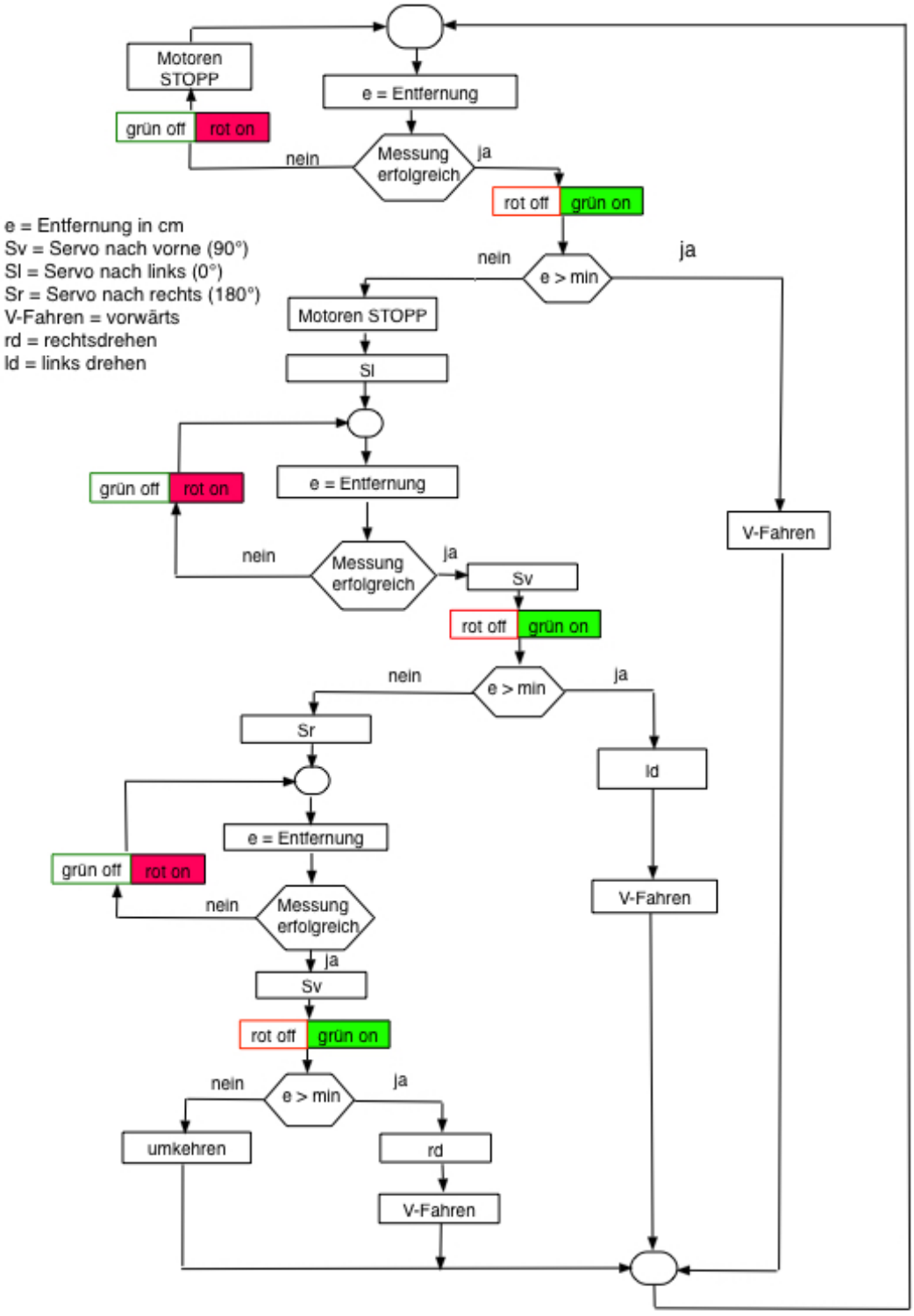
```
  delay(ms);                // Ablauf ms ms verzögern
  stoppMotoren();
}

void setup() {
   pinMode(IN1, OUTPUT);
   pinMode(IN2, OUTPUT);
   pinMode(IN3, OUTPUT);
   pinMode(IN4, OUTPUT);
}

void loop() {
    stoppMotoren();
    delay(1000);              // Ablauf 1000 ms verzögern
    umkehrenRoboter(dt_ret);
}
```

Hinweis:
Die Kehrtwende des Roboters sollte bereits jetzt auf dem Boden getestet werden. Der Roboter wird hierbei mit der Spannung einer 9V Batterie versorgt.

12.5 Programmbeispiel für den Einsatz des autonomen Roboters

Wir können für die Inbetriebnahme des autonomen Roboters viele Teile bereits erstellter Programme verwenden.

Struktur des Programms:



Hinweise:

- Wenn der autonome Roboter in einer ausweglosen Situation ist, sind möglicherweise seine Entfernungsmessungen ständig unbrauchbar. Es ist deshalb sinnvoll, bei einer erfolgreichen Entfernungsmessung eine grüne LED einzuschalten und bei einer fehlerhaften Entfernungsmessung eine rote LED einzuschalten. Zur Abfrage kann die `do-while`-Schleife verwendet werden.
- Der Roboter fährt mit maximaler Geschwindigkeit. Wenn mit geringerer Geschwindigkeit gefahren werden soll, müssen die ENA und ENB Pins aktiviert werden. Ein behelfsmäßige Methode ist im folgenden Programm vorgenommen worden, um die Geschwindigkeiten zu drosseln, ohne auf ENA und ENB zuzugreifen. Die Motoren werden nach einem Vortrieb kurz gestoppt:
  ```
  vorwaertsRoboter(10); // 10 ms vorwaerts fahren
  stoppMotoren();
  delay(1);
  ```
- Die minimale Entfernung e sollte nicht zu knapp eingestellt werden, damit der Roboter ausreichend Zeit hat zu reagieren. e = 20 cm hat sich bewährt.
- Der vorgestellte mögliche Programmablaufplan kann noch verbessert werden.

Lösungsvorschlag

```
// Servo-Bibliothek
#include <Servo.h>

int servoSteuer = 6; // Servo-PWM-Steuerleitung orange
Servo servoSFZ; // Servo-Objekt festlegen

int trigger = 4;  // Trigger-Pin des Ultraschallsensors
int echo = 5;     // Echo-Pin des Ultraschallsensors
double e;         // Entfernung in cm zum Objekt
double min = 20;  // minimaler Abstand in cm zum Hindernis
int dt = 1000;    // Zeitpuffer zw. Drehen und Entfernungmessen

int IN1 = 7;
int IN2 = 8;
int IN3 = 9;
int IN4 = 10;

int LEDgruen = 12;
int LEDrot = 2;

int w_v = 90;  // Winkelstellung des Servo in Vorwärtsrichtung
int w_r = 0;   // Winkelstellung des Servo nach rechts
               // aus Sicht des Roboterfahrers
int w_l =180;  // Winkelstellung des Servo nach links
int dt_r = 300;   // rechts: Zeit für gegenläufiges Drehen
int dt_l =320;    // links: Zeit für gegenläufiges Drehen
int dt_ret = 550; // retour: Zeit für gegenläufiges Drehen

int messung_fehler(int e){
  return (e > 300 || e < 2);
}
void LEDon (int LEDpin){  // LED einschalten
```

```
  digitalWrite(LEDpin, HIGH);
}

void LEDoff (int LEDpin){    // LED ausschalten
  digitalWrite(LEDpin, LOW);
}

void stoppMotoren() {
  // beide Motoren stoppen
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, LOW);
}

void rechtsDrehenRoboter(int ms) {
  // Roboter 90° rechts drehen
  digitalWrite(IN1, LOW); // rechtes Rad rückwärts
  digitalWrite(IN2, HIGH);
  digitalWrite(IN3, LOW); // linkes Rad vorwärts
  digitalWrite(IN4, HIGH);
  delay(ms);
  stoppMotoren();
}

void linksDrehenRoboter(int ms) {
  // Roboter 90° rechts drehen
  digitalWrite(IN1, HIGH); // rechtes Rad vorwärts
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, HIGH); // linkes Rad rückwärts
  digitalWrite(IN4, LOW);
  delay(ms);
  stoppMotoren();
}

void umkehrenRoboter(int ms) {
  // Roboter umdrehen
  digitalWrite(IN1, LOW); // rechtes Rad rückwärts
  digitalWrite(IN2, HIGH);
  digitalWrite(IN3, LOW); // linkes Rad vorwärts
  digitalWrite(IN4, HIGH);
  delay(ms);
  stoppMotoren();
}

double entfernungMessung() {
  int dauer; // Laufzeit des Schalls in Mikrosekungen
  digitalWrite(trigger,LOW); // Burst vorbereiten
  delay(2);
  digitalWrite(trigger,HIGH);
  delayMicroseconds(10);
  digitalWrite(trigger, LOW); // Burststart
  dauer= pulseIn(echo,HIGH);
  return dauer*0.0344/2; // Rueckgabe der Entfernung in cm
```

```
}

void vorwaertsRoboter(int ms) {
  // Roboter vorwaerts laufen lassen
  digitalWrite(IN1, HIGH); // rechtes Rad vorwärts
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, LOW); // linkes Rad vorwärts
  digitalWrite(IN4, HIGH);
  delay(ms);
  stoppMotoren();
}

void setup() {
   pinMode(IN1, OUTPUT);
   pinMode(IN2, OUTPUT);
   pinMode(IN3, OUTPUT);
   pinMode(IN4, OUTPUT);
   servoSFZ.attach(servoSteuer); // Servo initialisieren
   servoSFZ.write(w_v); // servo auf gerade aus
   pinMode(trigger, OUTPUT);
   pinMode(echo, INPUT);
   pinMode(LEDgruen, OUTPUT);
   pinMode(LEDrot, OUTPUT);
}

void loop() {

  do {
    e = entfernungMessung();
    if (messung_fehler(e)){
      LEDoff(LEDgruen);
      LEDon(LEDrot);
      stoppMotoren();
      delay(dt);
    }
  } while (messung_fehler(e)); // Messung nicht erfolgreich
  LEDoff(LEDrot);
  LEDon(LEDgruen);
  if (e > min)
  {
    vorwaertsRoboter(10); // 10 ms vorwaerts fahren
  }
  else
  {
    stoppMotoren();
    servoSFZ.write(w_l); // Servo links drehen
    delay(dt);
    do {
      e = entfernungMessung();
      if (messung_fehler(e)){
        LEDoff(LEDgruen);
        LEDon(LEDrot);
      }
```

```
    } while (messung_fehler(e)); // Messung nicht erfolgreich
    servoSFZ.write(w_v); // Servo auf gerade aus
    LEDoff(LEDrot);
    LEDon(LEDgruen);
    if (e > min)
    {
      linksDrehenRoboter(dt_l);
      vorwaertsRoboter(10); // 10 ms vorwaerts fahren
    }
    else
    {
      servoSFZ.write(w_r); // Servo rechts drehen
      do {
        e = entfernungMessung();
        if (messung_fehler(e)){
          LEDoff(LEDgruen);
          LEDon(LEDrot);
        }
      } while (messung_fehler(e)); // Messung nicht erfolgreich
      servoSFZ.write(w_v); // servo auf gerade aus
      LEDoff(LEDrot);
      LEDon(LEDgruen);
      if (e > min) {
        rechtsDrehenRoboter(dt_r);
        vorwaertsRoboter(10); // 10 ms vorwaerts fahren
      }
      else {
        umkehrenRoboter(dt_ret);
      }
    }
  }
}
```

13 Arduino-Glossar

13.1 IT Begriffe

Platine	Platte zur mechanischen Befestigung und elektrischer Verbindung von elektronischen Teilen
Mikrocontroller	kleines Computersystem, das fest vorgegebene Befehle ausführen kann
Arduino-Programmierung	Nach dem Download der Entwicklungsumgebung des Arduino können mit der Programmiersprache C++ Programme erstellt und dann auf den Mikrocontroller hochgeladen werden.
Digitalpins	entsprechen dem Pluspol einer Batterie. Sie können nur auf LOW (= 0V) oder HIGH (= 5V) sein.
Analogpins	entsprechen dem Pluspol einer Batterie. Sie können nur auf LOW (= 0V) oder HIGH (= 5V) sein.
-GND (Ground)	entspricht dem Minuspol einer Batterie
Breadboard	Steckplatine, die der mechanischen Befestigung und der elektrischen Verbindung von elektronischen Bauteilen dient.
Vorwiderstand	reduziert die Spannung am eingesetzten Bauteil (z.B. LED)
Fotowiderstand (LDR)	Light dependant Resistor (lichtabhängiger Widerstand)
PWM-Pin	Pulsweitenmodulation, die auf dem Arduino mit dem Zeichen ~ markiert ist. Die Pulsweiten können mit analogWrite(…) mit Werten von 0 bis 255 eingeteilt werden.
Bit	binary digit – bezeichnet eine Binärziffer 0 und 1
Byte	1 Byte (B) = 8 Bit; 1 kB = 2^{10} B = 1024 B; 1 MB (Megabyte) = 2^{10}·2^{10} B ≈ 10^6 B; 1 GB (Gigabyte) = 2^{10}·2^{10}·2^{10} B ≈ 10^9 B; 1 TB (Terabyte) = 2^{10}·2^{10}·2^{10}·2^{10} B ≈ 10^{12} B

13.2 Allgemeines zur Programmierung

Ein Computer wird durch Befehle gesteuert, die der Reihe nach abgearbeitet werden. Bei uns soll zum Beispiel der Arduino-Computer so gesteuert werden, dass eine LED (Leuchtdiode) blinkt:

1. Befehl: LED einschalten
2. Befehl: Warte eine Sekunde
3. Befehl: LED ausschalten
4. Befehl: Warte eine Sekunde

Ein Computer versteht nur sogenannte Maschinenbefehle. Ein Maschinenbefehl ist im Prinzip einfach eine Zahl, und ein Programm ist somit eigentlich nur eine Liste von verschiedenen Zahlen.

Offensichtlich wäre es für Menschen sehr umständlich, wenn größere Programme direkt mit Maschinenbefehlen geschrieben werden müssten. Stattdessen kann eine sogenannte höhere Programmiersprache verwendet werden. Ein Programm ist dabei ein Textdokument, das aus Wörtern und Sätzen besteht. Von einem Compiler (Übersetzer) wird dies dann in Maschinenbefehle übersetzt werden.

Für die Arduino-Programmierung wird die Programmiersprache C++ (sprich: "C Plus Plus") verwendet. Das Aufblinken einer LED könnte damit im Wesentlichen mit den C++-Anweisungen

```
turnOn_LED();
wait(1);
turnOff_LED();
wait(1);
```

programmiert werden.

13.3 C++-Programmierung

Wie oben erwähnt wurde, ist ein C++-Programm zunächst nur ein Text-Dokument, das von einem C++-Compiler in ein Maschinenprogramm übersetzt werden kann. Wie bei Sprachen, die wir als Menschen benutzen, besteht das Programm aus Wörtern und Satzzeichen und muss einer speziellen Grammatik genügen. Im Vergleich zu menschlichen Sprachen kann diese Grammatik sehr präzise durch Regeln beschrieben werden und es gibt keinerlei Ausnahmen. Allerdings ist die formale Beschreibung solch einer Grammatik für den Einsteiger in C++ für gewöhnlich zu komplex. Wir begnügen uns deshalb darauf, die Grammatik an Hand von Beispielen und anschließenden Erläuterungen einzuführen.

13.3.1 Variablen

- Variablen sind ein wichtiges technisches Hilfsmittel für die Programmierung. Eine Variable kann benutzt werden, um Zahlenwerte zu speichern und mit diesen zu rechnen. Es müssen aber folgende Besonderheiten beachtet werden:

- Auf einem Computer wird das Rechnen mit ganzen Zahlen und nicht-ganzen Zahlen völlig unterschiedlich realisiert. Für jede Variable, die man verwenden möchte, muss man deshalb festlegen, ob diese ganzzahlige oder nicht-ganzzahlige Werte speichern kann. Diese Festlegung kann nicht geändert werden.

- Der Wertebereich einer Variablen ist nicht beliebig groß. Das bedeutet zum Beispiel, dass es für eine ganzzahlige Variable eine kleinste und größte darstellbare Zahl gibt. Dasselbe gilt für nicht-ganzzahlige Variablen. Hier ist zusätzlich zu beachten, dass nicht beliebig viele Nachkommastellen gespeichert werden können.

Ob eine Variable ganzzahlige oder nicht-ganzzahlige Werte speichern kann und ebenso der darstellbare Wertebereich wird durch einen **Datentyp** definiert:

- Für ganzzahlige Variablen verwenden wir meist den Datentyp `int` (integer). Eine Variable von diesem Typ kann auf einem Arduino UNO ganze Zahlen von -32768 bis 32767 speichern.

- Für nicht-ganzzahlige Variablen verwenden wir den Datentyp `double`. Ohne auf die technischen Hintergründe einzugehen ist es hierbei deutlich schwieriger den Wertebereich genau und gleichzeitig anschaulich zu beschreiben. Die darstellbaren Zahlen liegen beim Arduino UNO zwischen -3.4· 10^{38} und 3,4 · 10^{38} . Die Anzahl der darstellbaren Ziffern beträgt nur 6 oder 7. Bei der Ausgabe mit
  `Serial.print` werden standardmäßig nur zwei Nachkommastellen ausgegeben. Mit
  `Serial.print(1.234567, 3)` wird 1.235 ausgegeben.

Für unsere Zwecke gehen wir davon aus, dass der Datentyp `int` für ganze Zahlen und der Datentyp `double` für nicht-ganze Zahlen stets ausreichend ist. Insbesondere gehen wir davon aus, dass die Wertebereiche bei Rechnungen nicht verlassen werden und bei nicht-ganzen Zahlen die Rundungsfehler vernachlässigbar sind.

Definition einer Variablen
Eine Definition (oder Vereinbarung) für eine Variable benötigt mindestens die Angabe eines Datentyps (bei uns `int` oder `double`) und die eines Bezeichners (Name der Variable). In C++ wird bei einem Bezeichner zwischen Groß- und Kleinbuchstaben unterschieden. Der Bezeichner muss mit einem Buchstaben ('a' bis 'z' oder 'A' bis 'Z') oder einem Unterstrich ('_') beginnen. Danach dürfen weitere Buchstaben, Ziffern oder Unterstriche folgen. Umlaute oder ß können nicht verwendet werden. Die Definition wird mit einem Strichpunkt abgeschlossen. Beispielsweise werden mit

```
int     a;
double  x;
```

die Variablen "a" und "x" definiert. Dabei wird festgelegt, dass "a" ganzzahlige und "x" nicht-ganzzahlige Werte enthalten kann. Eine nicht zulässige Definition wäre beispielsweise:

```
int Meßwert;
// Fehler: Buchstabe 'ß' ist ein nicht zulässiger Bezeichner
```

Definition einer Liste von Variablen
Es ist möglich, mit einer Vereinbarung gleich mehrere Variablen mit gleichem Datentyp festzulegen. Dazu gibt man eine mit Kommata getrennte Liste von Variablen an, wie beispielsweise in

```
int     a,b,c;
double  x,y;
```

womit dann die Variablen "a", "b" und "c" als ganzzahlig und die Variablen "x" und "y" als nicht-ganzzahlig definiert werden.

Definition von initialisierten Variablen
Bei der Definition einer Variablen kann zusätzlich angegeben werden, mit welchem Wert diese initialisiert wird. Beispielsweise wird mit

```
int a = 4;
```

die ganzzahlige Variable mit "a" definiert und mit dem Wert 4 initialisiert. Für die Initialisierung können auch arithmetische Ausdrücke verwendet werden, die wiederum zuvor definierte Variablen enthalten können:

```
int b = a + 1;
```

Dies kann auch bei der Definition einer Liste von Variablen verwendet werden. Beispielsweise können die letzten beiden Definition in einer Zeile mit

```
int a = 4, b = a + 1;
```

geschrieben werden. Zu beachten ist, dass diese Zeile von links nach rechts ausgewertet wird. Nicht zulässig wäre deshalb

```
int b = a + 1, a = 4;      // Fehler
```

denn die Variable "a" wäre noch nicht bekannt, wenn sie für die Initialisierung von "b" verwendet wird.

Besonderheiten bei nicht-ganzzahligen Variablen
Ganzzahlige und nicht-ganzzahlige Variablen werden nach dem gleichen Schema definiert. Anfänger stolpern aber gelegentlich über folgende Probleme:

Für nicht-ganzzahlige Werte (Dezimalzahlen, "Kommazahlen") wird kein Komma sondern ein Dezimalpunkt verwendet:

```
double x = 1,25;   // Fehler
double x = 1.25;   // Ok: x wird mit "Eins-Komma-zwei-fünf"
                   // initialisiert
```

Wie oben erwähnt und im Abschnitt "Arithmetische Ausdrücke" weiter beschreiben, wird zwischen ganzzahliger und nicht-ganzzahliger Rechnung unterschieden. Insbesondere bei der Division wird der Unterschied deutlich. Die Division "5 durch 3" ergibt bei der ganzzahligen Rechnung das Ergebnis 1 (der Rest von 2 wird verworfen). Bei nicht-ganzzahliger Rechnung ist das Ergebnis der auf eine gewissen Stellenzahl gerundete Wert 1.66..67. Beim Arduino UNO werden bei double-Zahlen standardmäßig zwei Ziffern nach dem Dezimalpunkt ausgegeben. Das hat folgende Auswirkungen:

```
double x = 5/3;     // "x" wird mit 1 initialisiert
double x = 5.0/3.0; // "y" wird mit 1.67 beim UNO initialisiert
```

13.3.2 Arithmetische Ausdrücke und Zuweisungen

Bei arithmetischen Ausdrücken können die Operatoren "+", "-", "*", "/" jeweils für die Grundrechenarten Addition, Subtraktion, Multiplikation und Division verwendet werden. In komplexeren Ausdrücken können Klammerungen verwendet werden. Ansonsten wird der Ausdruck nach der bekannten Punkt-vor-Strich-Regel ausgewertet.

Unter einer Zuweisung versteht man, dass eine Variable mit einer Zahl oder der Auswertung eines arithmetischen Ausdrucks überschrieben wird. Wie beispielsweise in

```
b = 4.5; c = 3; d = 11;
a = b*2 + c*d;
```

Im Unterschied zur initialisierten Definition einer Variablen müssen hier alle Variablen bereits zuvor schon definiert worden sein. Das Gleichheitszeichen "=" wird hier *nicht* verwendet, um eine Gleichung darzustellen. Es sollte vielmehr als "wird überschrieben mit" interpretiert werden. Das wird besonders deutlich bei einer Zuweisung wie

```
a = a + 1;
```

Hier wird der Inhalt von "a" um Eins erhöht (genauer: "a" wird überschrieben mit dem um Eins erhöhten Wert von "a"). Das Erhöhen einer Variable um Eins wird übrigens so häufig benötigt, dass es in C++ dafür einen eigenen Operator gibt:

```
++a;        // äquivalent zur Zuweisung: a = a+1
```

Implizite und explizite Datentyp-Konvertierung
Sind in einem Ausdruck wie

```
a / b
```

die Operanden "a" und "b" beide ganzzahlig, dann wird eine Operation ganzzahlig durchgeführt. Sind beide Operanden nicht-ganzzahlig, dann wird die Operation nicht-ganzzahlig durchgeführt. Dies wurde bereits oben erwähnt. Interessant ist die Regelung, wenn ein Operand ganzzahlig und der andere nicht-ganzzahlig ist. In diesem Fall wird die Rechnung nicht-ganzzahlig durchgeführt. Beispielsweise in

```
double x = 5.0/3;  // 5.0 ist nicht ganzzahlig, 3 ist ganzzahlig.
                   // Die Division wird nicht-ganzzahlig durchgeführt
                   // und x mit 1.67  initialisiert.
```

Man sagt, dass in diesem Fall der "kleinere" Datentyp 'int' implizit in den "größeren" Datentyp 'double' konvertiert wird. Mit "kleiner" bzw. "größer" ist hiermit gemeint, dass der Wertebereich von 'int' eine Teilmenge des Wertebereiches von 'double' ist. Mit "implizit" ist gemeint, dass diese Konvertierung automatisch, d. h. ohne Zutun des Programmierers geschieht.

In manchen Fällen muss man beim Programmieren eine Konvertierung "explizit" durchführen, also mit Zutun des Programmierers. Die Notwendigkeit soll an folgendem Beispiel motiviert werden.

```
int a = 5, b = 3;
double x = a/b;     // a und b sind beide ganzzahlig, also wird die
                    // Division ganzzahlig durchgeführt und
                    // x mit 1 initialisiert.
```

Damit "x" mit einer nicht-ganzzahligen Division initialisiert wird, müsste "a" oder "b" in einen nicht-ganzzahligen Wert konvertiert werden. Da aber die Datentypen beider Variable fest als ganzzahlig definiert sind, kann dies nicht automatisch erfolgen. Dies kann mit dem Operator `static_cast` aber explizit durchgeführt werden:

```
int a = 5, b = 3;
double x = static_cast<double>(a)/b;
// Der Wert von a wird explizit in double konvertiert
```

Den Ausdruck `static_cast<double>(a)` könnte man auffassen als "konvertiere den Wert von a in einen Wert vom Typ double".

13.3.3 Kontrollstrukturen und logische Operatoren

Anweisungen eines Programms werden in derselben Reihenfolge ausgeführt, wie sie aufgeschrieben worden sind. Manchmal ist es notwendig, dass die Reihenfolge unterbrochen oder ein Teil der Anweisungen wiederholt werden soll. Mit Kontrollstrukturen kann dies gesteuert werden. Wir erklären diese beispielhaft:

if-Anweisung
Mit der if-Anweisung der Form

```
if (Bedingung) {
    Anweisungen (falls Bedingung erfüllt ist)
} else {
    Anweisungen (falls Bedingung nicht erfüllt ist)
}
```

werden abhängig von einer Bedingungen die Anweisungen aus dem if-Zweig oder dem else-Zweig ausgeführt. Wird kein else-Zweig benötigt (d.h. würde in diesem keine Anweisung stehen), dann kann man diesen auch weglassen:

```
if (Bedingung) {
    Anweisungen (falls Bedingung erfüllt ist)
}
```

Die Bedingung kann zum Beispiel sein, dass ein Variable i gewisse Werte hat (oder nicht):

1) Bedingung "i hat den Wert 42"

```
if (i==42) {
    ...
}
```

Hier ist wichtig, dass zum Vergleich der Form "Wert ist gleich" der Vergleich-Operator "==" und *nicht* der Zuweisung-Operator "=" verwendet werden muss.

2) Bedingung "i ist ungleich 42"

```
if (i!=42) {
    ...
}
```

3) Bedingung "i ist kleiner als 42"

2.

```
if (i<42) {
    ...
}
```

Weitere, ähnliche Vergleiche können mit den Operator "<=" (kleiner gleich), ">" (größer als) und ">=" (größer gleich) erstellt werden.

Komplexere Bedingungen kann man durch logische Verknüpfungen formulieren:

1) Bedingung "i ist kleiner 42 und i ist größer als 13"

```
if ((i<42) && (i>13)) {
    ...
}
```

2) Bedingung "i ist kleiner 13 oder i ist größer als 42"

3.

```
if ((i<13) || (i>42)) {
    ...
}
```

Beispiele:
AB 7: LED bei Dunkelheit einschalten
ABL 7: LED bei Dunkelheit einschalten - Lösung
AB 8: Alarmanlage
ABL 8: Alarmanlage - Lösung
ABL 9: Entfernungsmesser mit Ultraschall - Lösung

for-Schleife
Manchmal soll eine Teilaufgabe mehrfach wiederholt werden. Wenn die Anzahl der Wiederholungen bekannt ist, kann dies mit einer for-Schleife durchgeführt werden. Die Bauweise einer for-Schleife ist wie folgt:

```
for (Initialisierung; Bedingung; Änderung) {
    Anweisungen (falls Bedingung erfüllt ist)
}
```

Ausgeführt wird die Schleife nach folgenden Regeln:

- Schritt 1: Die Initialisierung wird durchgeführt.
- Schritt 2: Falls die Bedingung erfüllt ist führe die Anweisungen und dann die Änderung aus. Wiederhole Schritt 2.

Konkreter kann dies wie folgt aussehen:

```
for (i=1; i<=4; ++i) {
    summe = summe + i;
}
```

Hier sind die Bausteine:

- Initialisierung: „i=0"
- Bedingung: „i<=4"
- Änderung: „++i"
- Anweisungen: „summe = summe + i"

Die Variable summe wird dann nacheinander mit i=1, i=2, ..., i=4 erhöht.
Beispiele:
AB 3b: Lauflicht – Programm mit „for-Schleife“
AB 11: Servo-Steuerung
ABL 11: Servo-Steuerung - Lösung

while-Schleife
Die Anweisung für eine while-Schleife lautet:

```
while (Bedingung) {
    Anweisungen (falls Bedingung erfüllt ist)
}
```

Solange die Bedingung wahr ist, wird die Anweisung ausgeführt.

do-while Schleife

Die Anweisung für eine d while-Schleife lautet:

```
do {
    Anweisungen
} while (Bedingung);
```

Wichtige Operatoren in C++:

=	Der Speicherplatz (links vom Gleichzeichen) wird mit dem Wert von rechts belegt
==	ist gleich
!=	ungleich
&&	und
||	oder

13.3.4 Funktionen

Eine Funktion bearbeitet eine Teilaufgabe im Programm. Die notwendigen Daten werden der Funktion als Parameter mitgegeben und die Funktion gibt dann das Ergebnis der Teilaufgabe zurück. Aus dem Mathematikunterricht kennen wir Terme wie $f(x) = 3x^2 + 4x$. Der Funktion f wird x als Parameter übergeben und daraus das Ergebnis $y = f(x)$ bestimmt. In C++ wird eine Funktion nur einmal definiert und kann dann beliebig oft mit ihrem Namen aufgerufen werden.

Zur Syntax und Struktur einer Funktion siehe:

ABL 11: Servo-Steuerung - Lösung

13.4 Arduino-Programmierung

Nach dem Download der Entwicklungsumgebung des Arduino können mit der Programmiersprache C++-Programme erstellt und dann auf den Mikrocontroller hochgeladen werden.

void setup() { **...** **}**	In der setup-Funktion wird die Initialisierung (Ausgangssituation) festgelegt (z.B. welche Pins auf HIGH (5V) oder LOW (0V) gesetzt oder ob die Pins für Eingangssignale (INPUT) oder Ausgangssignale (OUTPUT) benutzt werden).
void loop() { **...** **}**	In der loop-Funktion werden die Befehle eingegeben, die immer wieder ausgeführt werden sollen.
pinMode	legt einen bestimmten Pin als OUTPUT oder INPUT Pin fest. Bsp.: `pinMode(3, OUTPUT);` Auf Pin 3 werden HIGH- oder LOW-Signale mit digitalWrite nach außen geschickt.
digitalWrite	Schreibt einen HIGH oder LOW Wert auf den angegebenen Pin. Bsp: `digitalWrite(10, HIGH);` Auf Pin 10 ist das HIGH-Signal, 5V.
digitalRead	Liest einen HIGH oder LOW Wert vom angegebenen Pin. Bsp: `digitalRead(7);` Das Ergebnis ist HIGH oder LOW. Der Pin 7 ist mit `pinMode(7, INPUT);` auf INPUT festgelegt.
delay(...)	unterbricht den Programmablauf um so viele Millisekunden, wie in Klammer angegeben
analogWrite	Setzt auf den digitalen PWM-Pins die mit einer Zahl zwischen 0 und 255 die Pulsweiten (vgl. im Abschnitt 9). analogWrite hat nichts mit den Analogpins zu tun! Bsp: `analogWrite(10, 120);` Setzt die Pulsweite auf 120/255, d.h. etwa 50%.
analogRead	Liest einen ganzzahligen Wert zwischen 0 und 1023 vom angegebenen Analog-Pin. Das Arduinoboard hat einen 6 Kanal (A0 … A5) 10-Bit Analog-Digital-Wandler (vgl. Abschnitt 6), welcher Eingangsspannungen

	zwischen 0 und 5 V in ganze Zajhlen zwischen 0 und 1023 umwandelt. Bsp: `sensorwert = analogRead(A0);` Der Analogpin braucht nicht mit pinMode auf INPUT gesetzt werden.
Serial.begin(9600)	Datenübertragung zum Computer: 9600 Bits pro Sekunde (9600 Baud), 8 Bit pro Zeichen= 1 Byte. Das Bit ist die kleinste Einheit zur Datenspeicherung mit zwei Zuständen (0 oder 1)
Serial.print(...)	Ausgabe der Messwerte auf dem Computer ohne Zeilenvorschub.
Serial.println(...)	Ausgabe der Messwerte auf dem Computer mit Zeilenvorschub

static_cast<double> (sensorWert)	Die Variable „sensorWert“ soll vom C++ in eine Dezimalzahl umgewandelt werden.
tone, noTone	Bei einem **passiven** Piezolautsprecher kann mit tone die Tonhöhe eingestellt werden. Wenn der Piezo am Pin 8 angeschlossen ist, kann z.B. mit `tone(8, 440);` eine Tonhöe von 440Hz eingestellt werden. `noTone(8)` schaltet den Piezo am Pin 8 wieder aus.
pulseIn	Liest einen HIGH oder LOW-Puls am angegebenen Pin. Wenn z.B. die Anweisung `pulseIn(6,HIGH);` eingegeben wird, wartet die pulseIn-Funktion, bis am Pin 6 das Signal von LOW auf HIGH geht. Dann startet das Timing und wird gestoppt, sobald das Signal am Pin 6 wieder auf LOW geht. pulseIn liefert dann eine Zeit in Mikrosekunden. Bei einem Messfehler liefert pulseIn den Wert 0.

13.5 Fehlermeldungen

Wir unterscheiden zwischen Syntaxfehlern, Logikfehlern und Hardwarefehlern. Syntaxfehler entstehen beim Übersetzen des Programms in den Maschinencode. Logikfehler entstehen, wenn ein korrekt übersetztes Programm unvorhergesehene Dinge ausführt. Hardwarefehler betreffen den experimentellen Aufbau der Schaltung.

13.5.1 Syntaxfehler

Bei einem Syntaxfehler erscheint im unteren Bereich des IDE-Bereichs ein roter Balken mit einer Fehlermeldung. Zusätzlich wird der Fehler beschrieben. Die Fehlermeldungen sind nicht immer aussagekräftig.

Jede Anweisung wird mit einem Semikolon (;) beendet. In der Zeile oberhalb der markierten Zeile fehlt der Strichpunkt.

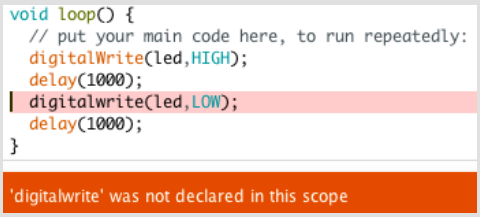
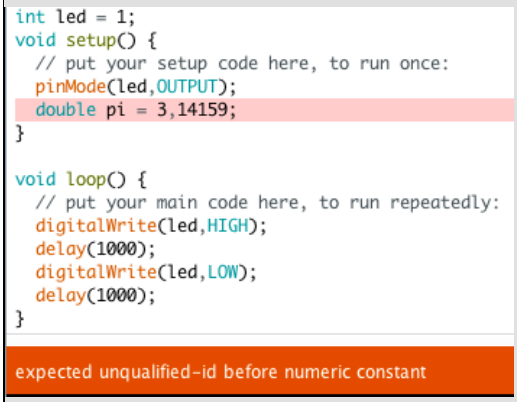
	Nichtssagende Fehlermeldung. Wir müssen die markierte Zeile selber auf den Fehler untersuchen. Die Parameter einer Funktion werden immer zwischen runden Klammern und nicht zwischen geschweiften Klammern eingeschlossen. richtig: `delay(1000);`
	Die Fehlerangabe macht deutlich, dass in diesem Bereich (scope) `digitalwrite` nicht bekannt ist. Wir müssen jetzt erkennen, dass `digitalWrite` eingesetzt werden muss.
	Fehlerhafte Verwendung des Strichpunkts (;). `void loop()` ist noch keine vollständige Anweisung.
	In der markierten Zeile fehlt der Funktionstyp.
	Ein unangenehmer Fehler: Dezimalzahlen müssen mit einem Punkt anstelle eines Kommas geschrieben werden.

Abbildung 50: Syntax-Fehlermeldungen mit Hinweisen[36]

[36] Screenshot aus der IDE (Arduino Version 1.8.3)

13.5.2 Logikfehler

Auch wenn das Programm fehlerfrei vom Compiler übersetzt worden ist, kann es Fehler beim Ablauf des Programms nach dem Hochladen auf den Mikrocontroller geben. Wir sprechen hier von Logikfehlern. Manchmal ist gezieltes Debugging (Auffinden von Fehlern) notwendig. Dazu setzen wir an verschiedenen Stellen im Programm mit `Serial.println(...)` Kontrollausgaben.

13.5.3 Hardwarefehler

Beim Experimentieren mit dem Arduino kann es neben Syntax- und Logikfehlern auch zu sogenannten Hardwarefehlern kommen. Vielleicht ist auf dem Breadboard ein Kabel nicht korrekt verbunden oder eine Lötstelle hat sich gelöst. Der Einsatz eines Digitalmultimeters lohnt sich manchmal, um Ärger zu vermeiden. Einer der häufigsten Hardwarefehler ergibt sich beim Hochladen des Programms auf den Mikrocontroller.

Hier ist der USB-Port nicht korrekt gewählt. Die korrekte Vorgehensweise ist beschrieben in „Blinkende LED“ S. 14

Abbildung 51: Hardware-Fehlermeldungen mit Hinweisen[37]

14 Anhang

14.1 Physik der Diode und der LED

Diode, LED oder Transistor erhalten ihre Funktion vor allem dadurch, dass das eingesetzte Halbleitermaterial, z.B. Silizium (Si), aus der 4. Hauptgruppe des Periodensystems mit Elementen aus der 3. Hauptgruppe oder mit Elementen aus der 5. Hauptgruppe „verunreinigt“ (dotiert) werden. Dadurch erhöht sich ihre Leitfähigkeit ganz wesentlich.

Der Si-Kristall besteht aus Si-Atomen, welche jeweils vier Nachbarn haben, mit denen sie jeweils eine Elektronen-Paar-Bindung eingehen. Man sagt, Si ist vierwertig. Jedes Si-Atom und jeder seiner Nachbarn stellt ein Elektron für diese Paarbindung zur Verfügung. In einer einfachen zweidimensionalen Darstellung können wir folgendes Modell wählen.

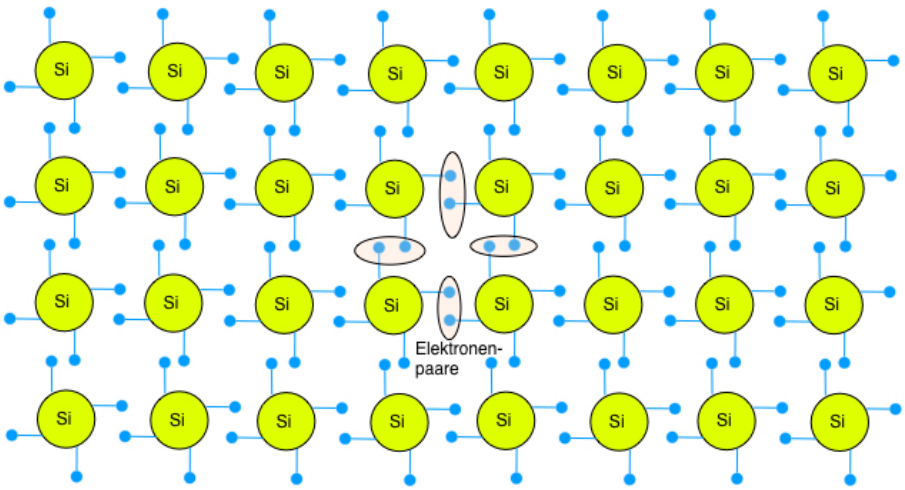


Abbildung 52: Si-Halbleiter

[37] Screenshot aus der IDE (Arduino Version 1.8.3)

Bei sehr tiefen Temperaturen ist der Si-Kristall ein Isolator, da alle Elektronen fixiert sind. Bei normalen Temperaturen, z.B. Zimmertemperatur, können sich einige Paarbindungen lösen. Es entstehen Leitungselektronen, welche sich frei durch die Atomrümpfe bewegen können. Wenn ein Elektron sich verselbständigt, dann hinterlässt es eine positive Ladung, man sagt ein Loch. Wenn in der Umgebung dieses Lochs zufällig ein Elektron frei wird, füllt es wahrscheinlich das Loch auf und es entsteht ein Loch in der Nähe. Wir können somit auch diese Löcher als wandernde positive Ladungen behandeln. Diese sogenannte Eigenleitung aus Elektronen und Löchern nimmt mit steigender Temperatur zu. Die Zahl der Löcher ist gleich der Anzahl der freien Elektronen. Der Si-Kristall ist also immer neutral.

Wenn ein Si-Kristall mit einem Atom aus der 5. Hauptgruppe, z.B. mit Phosphor, dotiert wird, entsteht ein n-Halbleiter, in dem Elektronen sich frei bewegen können. Wenn ein Si-Kristall mit einem Atom aus der 3. Hauptgruppe, z.B. mit Bor, dotiert wird, entsteht ein p-Halbleiter, in dem sich positive Ladungen (Löcher) frei bewegen können.

Wie bildet sich der n-Halbleiter?
Wenn ein Atom aus der 5. Hauptgruppe in den Si-Kristall eingebaut wird, bleibt eines seiner fünf Valenzelektronen, d.h. der Elektronen auf der äußeren Schale, nur noch schwach gebunden, da es nicht für die Bindung im Si-Kristall gebraucht wird. Bei kleiner angelegter Spannung wird das nicht benötigte Elektron bereits frei durch den Si-Kristall wandern. Zurück bleibt auch hier eine positive Ladung, weil ein Elektron fehlt. Da aber für die Bindung des Fremdatoms im Si-Kristall eines der Valenzelektronen nicht gebraucht wird, zeigt der Atomrumpf keine große Attraktion für Elektronen in der Nachbarschaft. Insgesamt sind in diesem Si-Kristall also vor allem Elektronen als bewegliche Ladungsträger vorhanden. Der Si-Kristall ist n-dotiert. n steht für negativ. Wir können vereinfacht den n-dotierten Si-Kristall als n-Leiter darstellen.

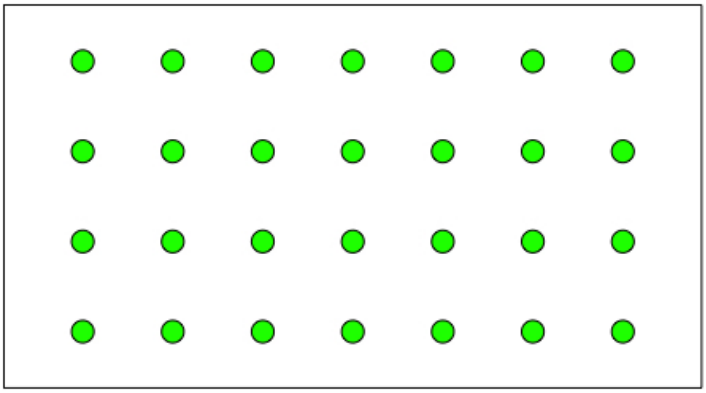


Abbildung 53: n-dotierter Halbleiter

Wie bildet sich der p-Halbleiter?
Wenn ein Atom aus der 3. Hauptgruppe in den Si-Kristall eingebaut wird, fehlt ein Elektron zur Elektronen-Paarbindung. Bei Zimmertemperaturen ist die Wahrscheinlichkeit hoch, dass sich das eingefügte Atom zur Ausbildung der Elektronenpaarbindung mit einem Elektron aus der Umgebung bedient. Dadurch entsteht an der Stelle des fremden dreiwertigen Atoms eine negative Ladung, die ortsgebunden ist. In der Umgebung ist dann ein positives Loch bei einem Si-Atom. Dieses holt sich wieder ein Elektron aus seiner Umgebung. Damit wandert das positive Loch quasi als beweglicher positiver Ladungsträger durch den Kristall.

Wir können vereinfacht den p-dotierten Si-Kristall als p-Leiter darstellen.

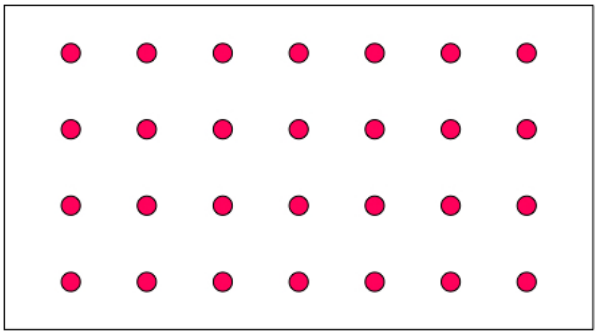


Abbildung 54:p-dotierter Halbleiter

Bringen wir einen p-Leiter und einen n-Leiter in Kontakt, dann diffundieren freie Elektronen vom n-Halbleiter in den p-Halbleiter. An der Grenzschicht kommt es damit zu einer sogenannten Rekombination: der p-Halbleiter wird an der Grenzschicht negativ und der n-Halbleiter durch die zurückgelassenen positiven Ladungen positiv. Dadurch kommt die Diffusion schnell zum Stillstand, denn die beweglichen Elektronen können das elektrische Feld nicht mehr überwinden.

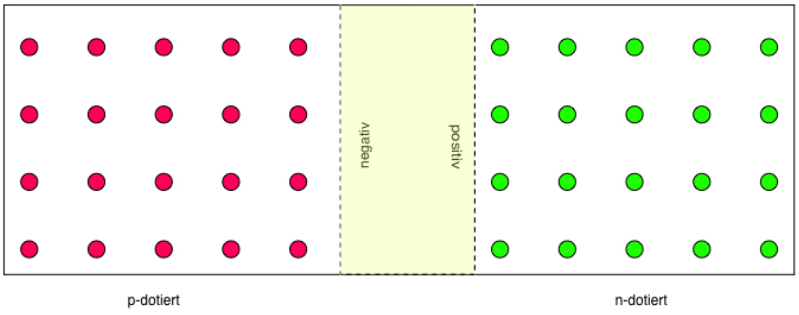


Abbildung 55: pn-Schicht

Es entsteht eine elektrische Spannung an der Grenzschicht: Der Pluspol ist im n-Halbleiter und der Minuspol im p-Halbleiter. Wenn eine äußere Spannung an den pn-Übergang - mit dem Pluspol an den p-Halbleiter und dem Minuspol an den n-Halbleiter - gelegt wird, fließt erst dann ein Strom, wenn die innere Spannung der pn-Grenzschicht überwunden ist. Bei Silizium beträgt diese innere Spannung etwa 0,7V. Wenn die äußere Spannung mit dem Pluspol an den n-Halbleiter und dem Minuspol an den p-Halbleiter gelegt wird, verbreitert sich die landungsfreie pn-Grenzschicht und es findet kein Stromfluss statt (abgesehen von der vernachlässigbaren Eigenleitung).
Fließt durch eine Diode elektrischer Strom, so wird durch die Rekombination der Elektronen und Löcher die Diode erwärmt.

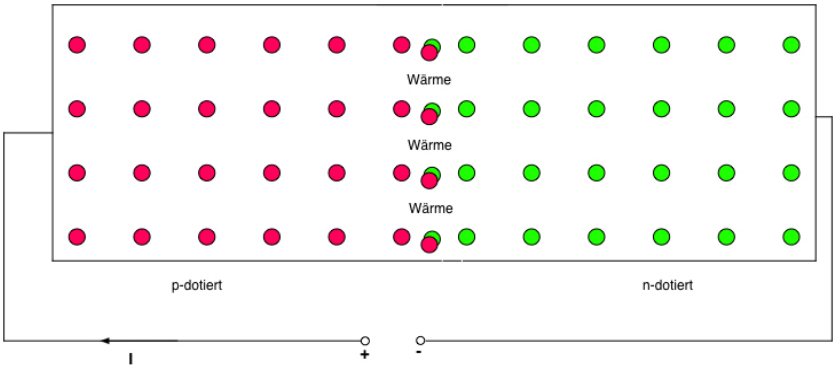


Abbildung 56: Diode in Durchlassrichtung beschaltet

Die elektrischen Eigenschaften einer LED sind ähnlich wie bei einer Diode. Allerdings ist die innere Spannung an der pn-Schicht so groß, dass bei der Rekombination der Elektronen und Löcher die frei werdende Energie als Licht ausgesandt wird. Um die notwendigen Spannungen zu erreichen, werden Stoffe wie Galliumphosphid (grüne LED) oder Aluminiumgalliumarsenid (rote LED) aus den 3. und 5. Hauptgruppen verwendet.

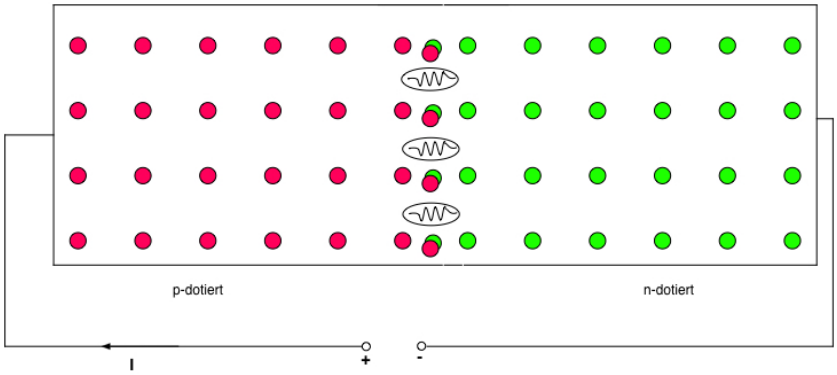


Abbildung 57: LED in Durchlassrichtung beschaltet

Die Wellenlängen des abgestrahlten Lichts errechnen sich nach Einstein und Planck aus
$hf = \Delta E = e \cdot U$
Mit $c = \lambda \cdot f$ folgt $\lambda = {}^{hc}/_{eU}$, wobei $h = 4{,}14 \cdot 10^{-15} eVs$ und $c = 3 \cdot 10^{8} m/s$ ist.

Bei einer roten LED ist U im Bereich 1,2 V .... 1,8 V. Damit errechnen sich Wellenlängen von 690 nm bis 1035 nm.
Bei einer gelb/grünen LED ist U im Bereich 1,9 V ... 2,5 V. Damit errechnen sich Wellenlängen von 654 nm bis 497 nm.
Bei einer blau/weiß LED ist U im Bereich 3 V ... 4 V. Damit errechnen sich Wellenlängen von 414 nm bis 311 nm.

14.2 Aufbau und Kennzeichen der LED

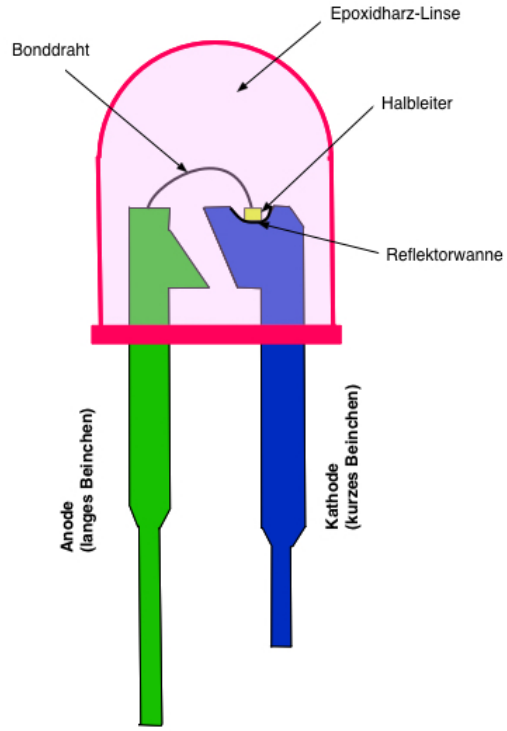


Abbildung 58: Aufbau einer LED

Bei einer Leuchtdiode (engl. Light Emitting Diode, LED) ist das Beinchen der Anode länger als das Beinchen der Kathode. Blickt man auf die Kunststofflinse, erkennt man die größere Trägerplatte des LED-Halbleiters. An dieser Seite ist die Kathode der LED. Die kleinere gegenüberliegende Elektrode heißt Anode. Der Strom kann die LED nur dann durchfließen, wenn der **Pluspol an die Anode** (A) und der **Minuspol an die Kathode** (K) angeschlossen werden.

Die LEDs sind in verschiedenen Farben erhältlich: rot, grün, orange, gelb, blau und weiß. Das Epoxidharz-Gehäuse ist durchsichtig, damit das Licht austreten kann. Es gibt zwei verschiedene Typen. Der gebräuchlichste Typ ist das Epoxidharz-Gehäuse, welches das LED-Licht diffuse abstrahlt. Es hat die gleiche Farbe wie das im Halbleiterkristall erzeugte Licht. Damit strahlt die LED in alle Richtungen. Daneben gibt es LEDs mit einem klaren Epoxidharz-Gehäuse. Diese LEDs strahlen zwar ebenfalls das entsprechende farbige Licht ab, aber nur in einem eingeschränkten Winkelbereich nach oben.