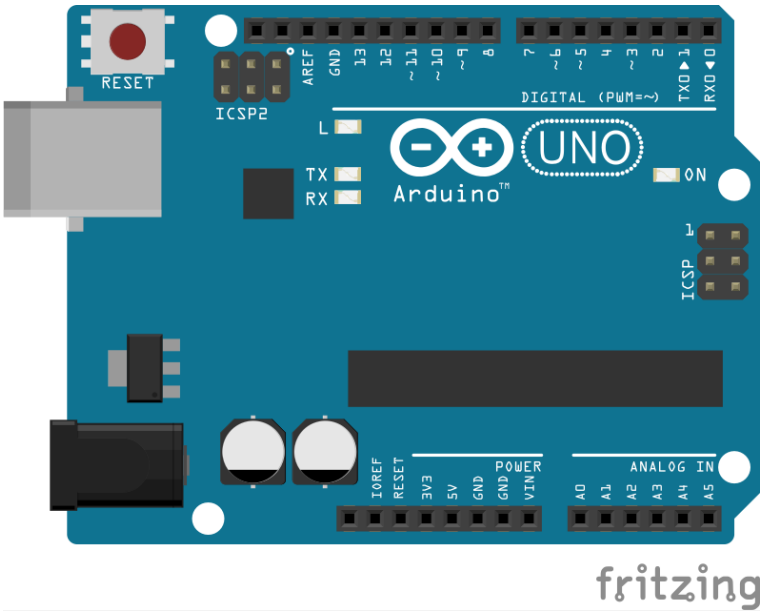
Anleitung zum NWT-Projekt: Arduino

A. Kimmig, Stand: 21.11.2017

Droste-
Hülshoff-
Gymnasium
Rottweil



Inhaltsverzeichnis

Kapitel I: Arduino: Hardware und Software

I.1. Mikrocontroller

„Als Mikrocontroller (auch μController, μC, MCU) werden Halbleiterchips bezeichnet, die einen Prozessor und zugleich auch Peripheriefunktionen enthalten.“[1]

Mikrocontroller können als kleine Computer angesehen werden, die für verschiedene, einfache Zwecke – wie z. B. Messungen und Steuerungen – eingesetzt werden können.

Der populärste Vertreter ist die *ATmega*-Baureihe des Herstellers *Atmel*. Zum Betrieb eines solchen Mikrocontroller werden noch einige andere Bauelemente benötigt (Widerstände, Kondensatoren, Quartz,...), welche früher selbst zusammengebaut werden mussten.

Um den Einstieg in die Mikrocontroller-Technik zu erleichtern gibt es inzwischen jedoch viele verschiedene fertige Aufbauten – sogenannte *Boards* – die auch ohne (oder zumindest deutlich weniger) Elektronikkenntnisse benutzt werden können.

I.2. Arduino

Das meistverbreiteste Board ist der Arduino:[2]

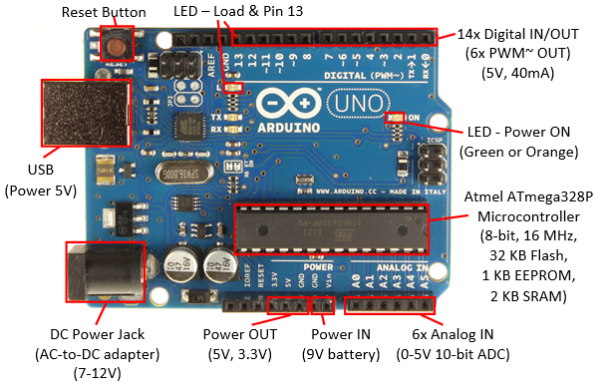


Abbildung 1: Aufbau eines **Arduino Unos**

Dieses Board bringt alles mit, was man für einen erfolgreichen Einstieg in die Mikrocontroller-Programmierung benötigt. Auch die Software um Programme für den Arduino zu erstellen ist für alle Betriebssysteme kostenlos erhältlich[3].

Es gibt inzwischen viele unterschiedlichen Arduino-Boards, die sich hauptsächlich in der Anzahl der verfügbaren Pins unterscheiden. In der Schule werden wir mit dem Arduino Uno *arbeiten, weshalb alle folgenden Beschreibungen auf dieses Modell ausgelegt sind.*

[1] Zitat: Wikipedia „Mikrocontroller“
[2] Quelle: `https://www3.ntu.edu.sg/home/ehchua/programming/arduino/images/ArduinoUno.png`
[3] `https://www.arduino.cc`

I.3. Ein- und Ausgänge

Ein weiterer Vorteil des Arduinos ist, dass dieser viele steckbare *Pins* bereitstellt, welche durch die Programmierung angesteuert oder ausgelesen werden können:

- 14 digitale Pins, die sowohl als Steuerung (Ausgang/*Output*) als auch zur Messung (Eingang/*Input*) verwendet werden können. Durch die Programmierung können wir für jeden Pin selbst frei entscheiden, ob dieser als Ausgang oder als Eingang funktionieren soll.
- 6 dieser digitalen Pins sind mit einem ~ gekennzeichnet. Diese können auch als sogenannter PWM-Output verwendet werden.[4]
- 6 analoge Eingänge

I.3.1. digital ↔ analog

Wenn wir von digitalen Signalen sprechen, so meinen wir zumeist die *binäre Digitaltechnik* die legiglich mit den Zuständen 1 und 0, bzw. „an“ und „aus“ auskommt.

Genauso können wir an den digitalen Pins die Spannung[5] auch nur ein- bzw. ausschalten. Umgekehrt können wir an den digitalen Eingängen auch nur erkennen, ob eine Spannung anliegt oder nicht.[6]

An den analogen Eingängen können wir mit dem Arduino nicht nur die Zustände „1“ und „0“ unterscheiden, sondern auch 1024 Abstufungen davon. Liegt eine Spannung von 0 V an, so liefert der Arduino als Ergebnis weiterhin eine „0“, bei einer Spannung von 5 V ist das Ergebnis jedoch „1023“.[7]

Bei einer dazwischenliegenden Spannung erhalten wir also auch einen Wert, der dazwischen liegt, bei einer Spannung von 2.5 V (also der Hälfte) erhalten wir einen Wert von „511“.

Aufgabe I.1:

Welche Werte liefern also folgende Spannungen:

a) 1 V

b) 2 V

c) 4.5 V

Aufgabe I.2:

Welche Spannung liegt an, wenn folgende Ergebnisse herauskommen:

1. „1000“
2. „1“
3. „280“

[4] s. Kapitel IV.5
[5] Die Spannung am Arduino Uno beträgt 5 V
[6] Der Arduino registriert bei einer Spannung > 3.0 V den Eingang als „1“, ansonsten als “0“.
[7] Achtung: es gibt insgesamt 1024 verschiedene Zustände, die jedoch von 0-1023 gehen!

I.4. Spannungsversorgung

Die Spannungsversorgung kann beim Arduino über die USB-Verbindung erfolgen. Ein herkömmlicher USB-Anschluss stellt für die meisten Versuche eine ausreichende Stromstärke bereit, so dass kein zusätzliches Netzteil angeschlossen werden muss.

Sollen spezielle Sensoren und Aktoren benutzt werden, die eine andere Versorgungsspannung als die vom Arduino bereitgestellten 5 V, oder eine höhere Stromstärke benötigen, so muss man zusätzlich eine externe Stromquelle verwenden. (Näheres s. Kapitel VI)

Weiter ist zu beachten, dass beim Arduino Uno jeder Pin nur bis zu einer maximalen Stromstärke von 40 mA belastet werden darf! Wird versucht, dem Pin eine höhere Stromstärke zu entziehen, so besteht die Gefahr, dass der Arduino beschädigt und unbrauchbar wird!

Die Gesamtstromstärke für alle Pins beträgt jedoch nur 200 mA!

I.4.1. Pin-Bezeichnungen

Die Spannungs-Pins elektrischer Bauteile haben immer eine besondere Bezeichnung, unter anderem:[8]

V_{CC} : Versorgungspotenzial, im Falle des Arduinos +5 V

GND : Nullpotenzial, 0 V

I.5. Software

Mithilfe der kostenlos erhältlichen Arduino-Software lassen sich Arduino Programme – sogenannte *Sketche* – erstellen und auf den Arduino übertragen.

I.5.1. Oberfläche

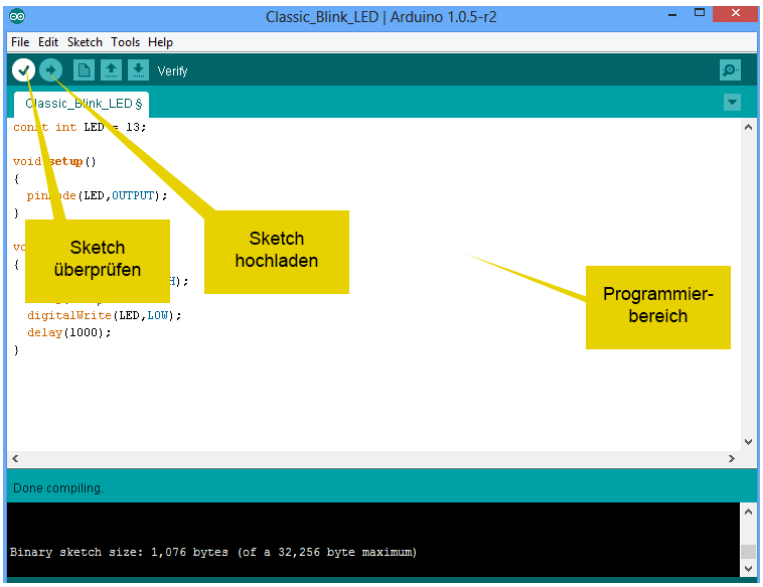


Abbildung 2: Die Oberfläche der Arduino-Software und die wichtigsten Befehle/Bereiche

[8] Eine Übersicht der spezifischen Bezeichnungen findet man auf `https://de.wikipedia.org/wiki/Spannungsbezeichnung`

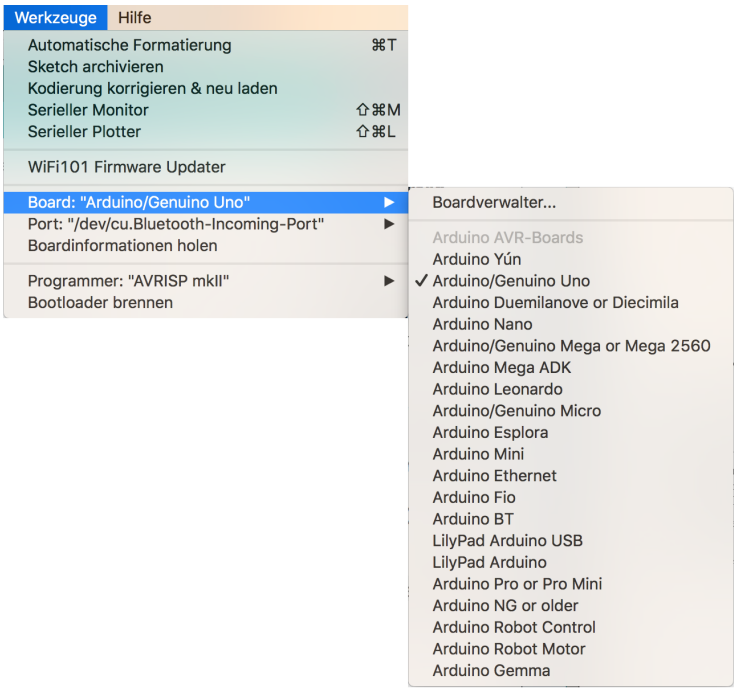
I.5.2. Erste Einstellungen

Nachdem die Arduino-Software gestartet und das Board per USB-Kabel an den PC angeschlossen wurde muss zunächst überprüft werden, ob die Software den Arduino automatisch erkannt hat.

Im Menü „Werkzeuge“ (bzw. in der englischen Version „Tools“) gibt es einen Unterpunkt „Port“. Für gewöhnlich wird hier nur ein einziger Port angezeigt. Sollten mehrere zur Verfügung stehen, so müssen wir diese durchprobieren.

Ebenso im „Werkzeuge“-Menü müssen wir das korrekte Board (in unserem Fall „Arduino/Genuino Uno“ auswählen.

Ist alles korrekt eingestellt, so können wir mit dem Programmieren beginnen.



I.5.3. Programmierbereich

Im Programmierbereich wird später der Sketch programmiert. Erstellen wir einen neuen Sketch, so werden automatisch bereits zwei Bereiche erstellt:

```
1 void setup() {
2   // put your setup code here, to run once:
3 }
4
5 void loop() {
6   // put your main code here, to run repeatedly:
7 }
```

Listing 1: Aufbau eines neuen Sketchs

Der Programmablauf im Arduino wird zunächst einmalig die Befehle im ersten `setup`-Block ausgeführt. Anschließend wird der zweite `loop`-Block immer und immer wieder durchgeführt. Man nennt den `loop`-Block auch *Hauptschleife*.

Merke:

Wie wir eben schon gesehen haben, wird die Programmierung immer in Blöcke unterteilt. Jeder Block wird dabei in geschweiften Klammern eingeschlossen.

Im `setup`-Bereich werden dabei einmalige Initialisierungen durchgeführt. Solche können beispielweise Einstellungen der Pins sein, ob diese als Ausgang oder Eingang funktionieren sollen.

Im `loop`-Bereich können wir ständige Messungen und Steuerungen durchführen.

Da die beiden Blöcke bisher noch leer sind wird der Arduino nichts tun. Wir können jedoch mit einem Klick auf „Sketch hochladen“ testen, ob die Verbindung zum Arduino funktioniert.

Kapitel II: Programmiergrundlagen

II.1. LED-Blinklicht

Im ersten Programm wollen wir eine LED (*engl.* ***L****ight-****E****mitting* ***D****iode* blinken lassen.

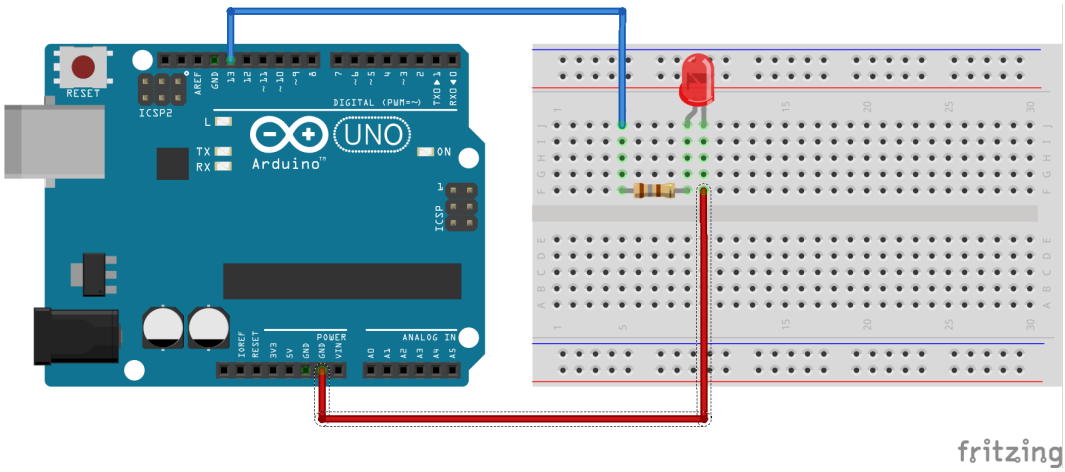


Abbildung 3: Erster Aufbau: LED-Blinklicht

```
1  void setup() {
2    pinMode(13, OUTPUT);
3  }
4
5  void loop() {
6    digitalWrite(13, HIGH);
7    delay(1000);
8    digitalWrite(13, LOW);
9    delay(1000);
10 }
```

Listing 2: LED-Blinklicht

Aufgabe II.1:

Baue zunächst die angegebene Schaltung nach. Achte auf den richtigen Vorwiderstand um die LED zu betreiben.[9]Dieser muss bei einer roten oder gelben LED 180 Ω, bei einer weißen, grünen oder blauen LED 100 Ω betragen!
Tippe anschließend den angegebenen Sketch ab, speichere ihn als `02_01_LED` und übertrage ihn auf den Arduino. Das Programm wird automatisch gestartet und die LED sollte anfangen zu blinken.

Hinweis: sollte die LED nicht blinken, so tausche die Anschlüsse. LEDs funktionieren nur in eine Stromrichtung!

Merke:

Achte bei der Programmierung auf Groß- und Kleinschreibung. Jeder Befehl muss desweiteren auch mit einem Semikolon abgeschlossen werden.

[9]Näheres zum Thema Vorwiderstand im Kapitel IV.3.5

II.1.1. Programmanalyse

Der `setup`-Block wird ganz zu Beginn ausgeführt. Anschließend werden die Befehle im `loop`-Block abgearbeitet. Ist das Programm am Ende angekommen, so wird der `loop`-Block von vorne gestartet.

Aufgabe II.2:

Beschreibe, wofür die Befehle dienen:

a) `pinMode(13, OUTPUT);`

b) `digitalWrite(13, HIGH);` bzw. `digitalWrite(13, LOW);`

c) `delay(1000);`

Ändere dazu einzelne Befehle ab und beobachte, was passiert. Z. B. kannst du die Zahlen ändern, die Befehle löschen oder auch eine andere Reihenfolge ausprobieren.

Aufgabe II.3:

Ändere den Sketch so ab, dass die LED 3 Sekunden lang leuchtet und anschließend 1 Sekunde lang dunkel bleibt.
Speichere den neuen Sketch unter `02_03_LED` ab.

Größere Projekte können sehr schnell unübersichtlich werden. Um später noch die Befehle zu verstehen, können *Kommentare* benutzt werden. Diese dienen lediglich dem Programmierer dazu, den Code zu verstehen, werden aber vom Arduino komplett ignoriert!

Merke:

Um den Code für den Programmierer verständlicher zu machen, kannst du die Befehle mithilfe von *Kommentaren* beschreiben. Es gibt zwei Arten von Kommentaren:

- Kommentare, die mit `//` beginnen gehen bis zum Zeilenende.
- Kommentare, die mit `/*` beginnen müssen mit `*/` abgeschlossen werden und können auch über mehrere Zeilen gehen.

Beispiel für Kommentare:

```
1  void setup() { // Der Setup-Bereich wird einmal zum Programmstart aufgerufen
2    pinMode(13, OUTPUT);
3  }
4
5  /* Nach dem Setup-Bereich wird der loop-Bereich ausgefuehrt. Sind alle
       Befehle abgearbeitet, so beginnt dieser Bereich wieder von vorne */
6  void loop() {
7    digitalWrite(13, HIGH);
8    delay(1000);
9    digitalWrite(13, LOW);
10   delay(1000);
11 }
```

Listing 3: Kommentare beim LED-Blinklicht

Füge in Zukunft in jeden deiner Sketche Kommentare ein um die Befehle zu beschreiben.

II.2. mehrere LEDs ansteuern

Jetzt sollen mehrere LEDs betrieben und abwechselnd ein- bzw. ausgeschaltet werden.

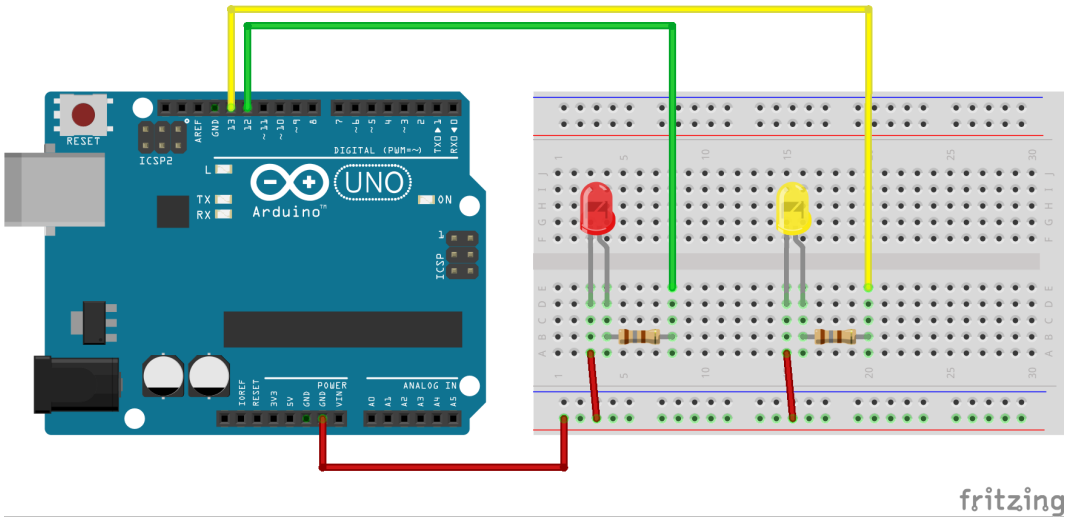


Abbildung 4: Zweiter Aufbau: mehrere LEDs

Aufgabe II.4:

Baue den angegebenen Aufbau nach.

a) Ergänze dein Programm so, dass zunächst beide LEDs gleichzeitig blinken und speichere den Sketch als `02_04_2LED` ab.

b) Ändere es anschließend ab, dass beide LEDs abwechselnd blinken.

Aufgabe II.5:

Erstelle ein Lauflicht aus einer roten, einer gelben und einer grünen LED. (Achtung, die grüne LED benötigt einen anderen Vorwiderstand von 100 Ω!)
Speichere den Sketch als `02_05_Lauflicht`.

a) Alle LEDs sollen nacheinander eine halbe Sekunde aufleuchten.

b) Skizziere den Schaltplan

Hinweis: eine LED wird im Schaltplan mit dem folgenden Symbol gezeichnet: 

Aufgabe II.6:

Erstelle mit den 3 farbigen LEDs von voriger Aufgabe eine Ampelschaltung entsprechend einer herkömmlichen Verkehrsampel.

II.3. Variablen

Wollen wir beim bisherigen Aufbau eine LED an einem anderen Pin anschließen, so müssen wir den Sketch an mehreren Stellen abändern um die entsprechende Pinnummer anpassen.

Besser wäre es wenn wir die Pinnummer an einer Stelle hinterlegen und an den anderen Stellen einfach wiederverwenden könnten.

Genau das kann man mit einer *Variablen* erledigen. Variablen kennst du schon aus der Mathematik, wo z. B. x für eine bestimmte Zahl steht, dieses x aber mehrfach in einer Formel vorkommen kann.

Variablen können wir frei benennen (Umlaute und Sonderzeichen dürfen dabei jedoch nicht vorkommen) und können diese mit einem Wert belegen. Anschließend können wir mit diesem Variablennamen wieder auf den Wert zugreifen.

```
1  int meinPin = 13; // hier wird die Variable angelegt und der Wert festgelegt
2
3  void setup() {
4    pinMode(meinPin, OUTPUT); // hier wird der Wert abgerufen und verwendet
5  }
6
7  void loop() {
8    digitalWrite(meinPin, HIGH);
9    delay(1000);
10   digitalWrite(meinPin, LOW);
11   delay(1000);
12 }
```

Listing 4: Variablen

In Zeile 1 wird dabei die Variable `meinPin` angelegt und ihr der Wert 13 zugewiesen.

Merke:

Mit einem einfachen Gleichzeichen „=“ erfolgt eine Wertzuweisung des rechten Wertes in die links stehende Variable.

Anschließend kann in den Zeilen 4, 8 und 10 der in der Variablen gespeicherte Wert verwendet werden. Soll nachträglich die LED an einem anderen Pin angeschlossen werden, so muss der Sketch also nur noch an einer einzigen Stelle angepasst werden. Dies reduziert auch die möglichen Fehlerquellen!

Merke:

Wird eine Variable innerhalb eines mit geschweiften Klammern umschlossenen Blockes (z. B. in `setup`) definiert, so ist diese auch nur innerhalb dieses Blockes gültig und kann nicht in einem anderen Block (wie z. B. in `loop`) verwendet werden!

II.3.1. Datentypen

Anders als in der Mathematik müssen wir bei den Variablen angeben, welche Art von Wert diese speichern. Diese Angabe erfolgt **einmalig** beim Erzeugen der Variablen indem man den Datentyp davor schreibt. (s. oben: `int meinPin = 13;`)

Es stehen uns u. a. folgende Datentypen zur Verfügung:

int ganze Zahlen zwischen $-32\,768$ und $+32\,767$
z. B. `int meinPin = 13;`

long ganze Zahlen zwischen $-2\,147\,483\,648$ und $+2\,147\,483\,647$
z. B. `long meinPin = 13;`

float Fließkommazahlen von $-3.4028235 \cdot 10^{38}$ bis $+3.4028235 \cdot 10^{38}$
z. B. `float meineZahl = 13.311;`[10]

char Einzelne Buchstaben, die in einfache oder doppelte Anführungszeichen eingeschlossen werden
z. B. `char meinBuchstabe = 'a';`

String [11] Zeichenkette, die in doppelte Anführungszeichen eingeschlossen ist
z. B. `String meinText = "Hallo Welt!";`

bool Wahrheitswerte wahr/`true` bzw. falsch/`false`
z. B. `bool fertig = false;`

Aufgabe II.7:

Ändere deinen Ampelschaltungs-Sketch, führe die drei `int`-Variablen `pinRot`, `pinGelb` und `pinGruen` ein und speichere ihn als `02_07_Variablen` ab.

[10] Achtung: eingegeben müssen die Zahlen in englischer Schreibweise werden, d. h. mit einem Punkt als Dezimaltrennzeichen!
[11] Achtung: Schreibweise mit großem „S“!

II.4. Taster und Bedingungen

Ein Arduino eignet sich nicht nur dazu, Geräte zu steuern, sondern auch um Sensoren auszulesen und entsprechend zu reagieren. Als einfachste externe Steuerung verwenden wir einen Taster.

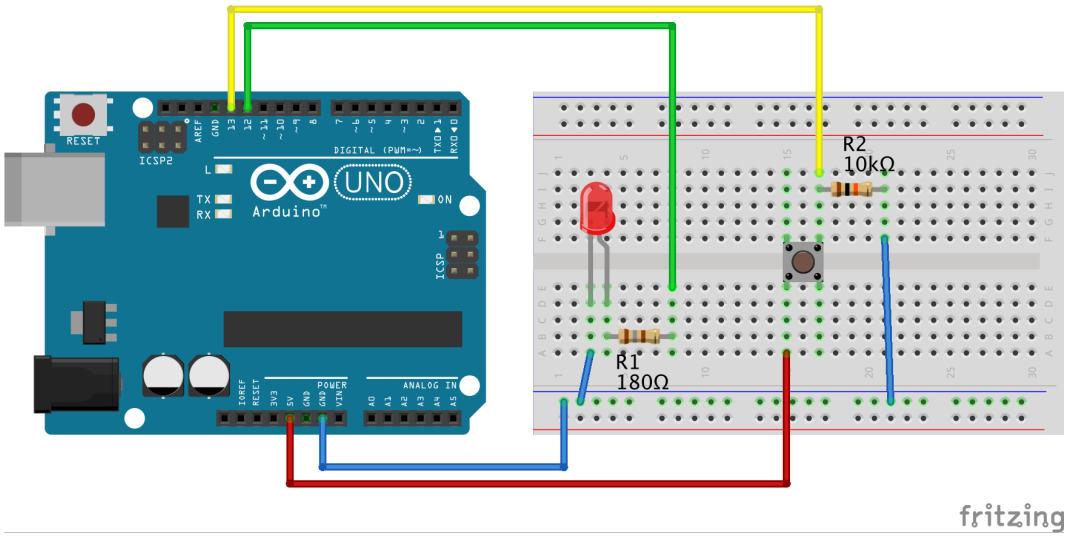


Abbildung 5: Einfache Steuerung des Programms durch einen Taster.

```
int ledPin = 12;
int tasterPin = 13;

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(tasterPin, INPUT);
}

void loop() {
  int schalter = digitalRead(tasterPin);

  if(schalter == 1) {
    digitalWrite(ledPin, HIGH);
  }
  else {
    digitalWrite(ledPin, LOW);
  }
}
```

Listing 5: Lesen eines Tasters

Aufgabe II.8:

Baue die angegebene Schaltung nach, verwende dabei als zweiten Widerstand am Taster einen mit 10 kΩ[12]und programmiere den Sketch. Speichere diesen als `02_08_Taster` ab und übertrage ihn auf den Arduino. Kommentiere auch die Befehle im Sketch!

Drückst du nun den Taster, so geht die LED an. Das an sich ist noch nicht spektakulär, denn diese Funktion könnten wir natürlich auch mit einem normalen Stromkreis ohne Arduino aufbauen. In Zukunft werden wir aber die Funktion noch erweitern.

[12]Warum der Widerstand nötig ist steht im Kapitel IV.2

II.4.1. Programmanalyse

In diesem Programm wird zum ersten Mal die Spannung an einem Pin „gemessen“. Dazu müssen wir diesen zunächst als Eingang einstellen, das passiert in der 6. Zeile mit dem Befehl `pinMode(tasterPin, INPUT);`

Merke:

Bevor wir einen Pin benutzen können müssen wir diesen zuerst als Ausgang/`OUTPUT` oder Eingang/`INPUT` einstellen!

In der Hauptschleife wird zunächst mit dem Befehl `digitalRead(tasterPin);` der Status des des Pins 13 (`tasterPin`) abgefragt und in der Variablen `schalter` gespeichert.

Liegt eine Spannung von 5 V an diesem Pin an – also ist der Taster gedrückt – so ist der Wert der Variablen anschließend „1“, ansonsten „0“.

Mit der `if`-Anweisung in Zeile 12 führt der Arduino den direkt darauffolgenden Block nur dann aus, wenn die Bedingung wahr ist. Unsere Bedinung `(schalter == 1)` ist ein direkter Vergleich, ob der Wert der Variablen `schalter` genau „1“ entspricht.

Merke:

Da ein einfaches Gleichzeichen eine Wertzuweisung ist müssen wir für einen Vergleich zwei Gleichziechen schreiben!

Weitere Vergleichsmöglichkeiten sind wie wir aus der Mathematik kennen:

- < kleiner
- > größer
- == genau gleich
- <= kleiner oder gleich
- <= größer oder gleich
- ! = ungleich

Tritt die Bedingung **nicht** ein, so wird der `if`-Block (Zeilen 12-14) übersprungen. Die Befehle im anschließenden `else`-Block werden nur ausgeführt, wenn die Bedingung nicht zutrifft! der `else`-Block kann auch weggelassen werden.

Aufgabe II.9:

Ändere deinen Sketch so ab, dass du den `else`-Block (Zeilen 15-17) entfernst und beobachte was passiert.

Aufgabe II.10:

Benutze zwei Taster und programmiere den Sketch `02_10_OnOff` so, dass mit einem Taster die LED ein- mit dem anderen ausgeschaltet werden kann.

Zusatz-Aufgabe II.11:

Programmiere den Sketch so, dass die LED erst mit einer Verzögerung von 1 Sekunde nach dem Drücken des Tasters ein- bzw. ausgeht.

II.5. Lautsprecher

Hier wollen wir einen Lautsprecher betreiben und eine kurze Melodie spielen.

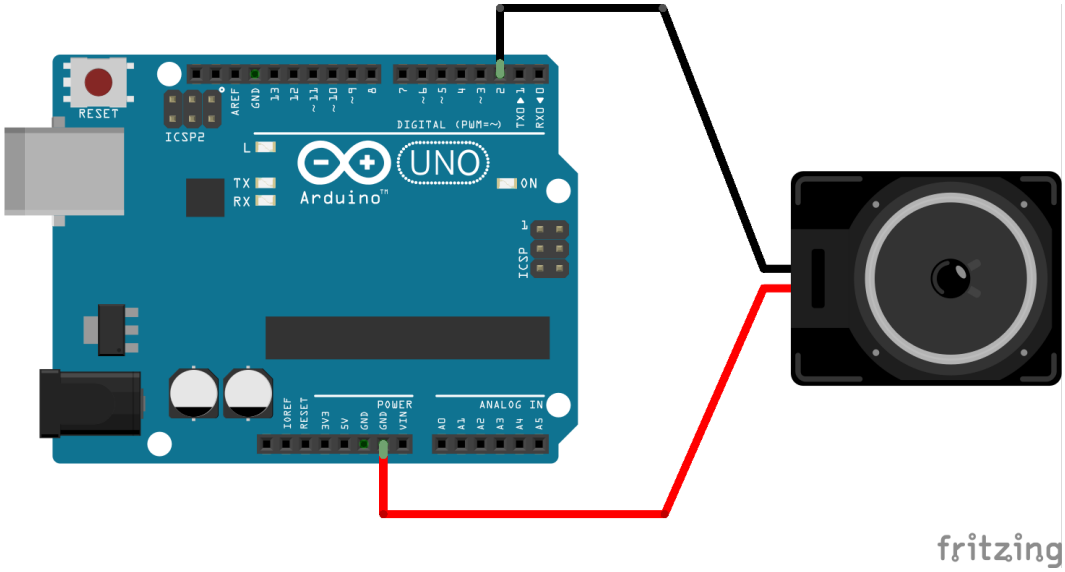


Abbildung 6: Tonerzeugung mit dem Arduino.

```arduino
int speaker = 2;

void setup() {
}

void loop() {
  tone(speaker, 100);      // 100 Hz einschalten
  delay(2000);             // 2 Sekunden warten
  noTone(speaker);         // Ton ausschalten
  delay(1000);             // 1 Sekunde warten
}
```

Listing 6: Ansteuerung eines Lautsprechers

Aufgabe II.12:

Schließe den Lautsprecher auf Pin 2 und GND an und programmiere den Sketch.

II.5.1. Programmanalyse

Mit dem Befehl `tone(speaker, 100);` wird der Lautsprecher der an Pin `speaker` eingeschaltet mit einer Frequenz von 100 Hz. Der Befehl `noTone(speaker);` deaktiviert diesen anschließend wieder.

II.5.2. Frequenzen der Töne

In der folgenden Tabelle sind die Frequenzen der unterschiedlichen Töne aufgelistet

	C	Cis	D	Dis	E	F	Fis	G	Gis	A	Ais	H
3.	130,8	138,6	146,8	155,6	164,8	174,6	185,0	196,0	207,7	220,0	233,1	246,9
4.	261,6	277,2	293,7	311,1	329,6	349,2	370,0	392,0	415,3	440,0	466,2	493,9
5.	523,3	554,4	587,3	622,3	659,3	698,5	740,0	784,0	830,6	880,0	932,3	987,8
6.	1046,5	1108,7	1174,7	1244,5	1318,5	1396,9	1480,0	1568,0	1661,2	1760,0	1864,7	1975,5

Aufgabe II.13:

Programmiere eine eigene Melodie und speichere sie als `02_13_Melodie` ab.

II.6. Schleifen

Als nächstes wollen wir eine Sirene programmieren, d. h. die Tonhöhe (also die Frequenz) soll langsam erhöht und anschließend wieder langsam erniedrigt werden. Mit den bisherigen Mitteln könnten wir das folgendermaßen programmieren:

```
[...]
void loop() {
  tone(speaker, 100);     // 100 Hz einschalten
  delay(10);              // 10 Millisekunden warten
  tone(speaker, 105);     // 105 Hz einschalten
  delay(10);              // 10 Millisekunden warten
  tone(speaker, 110);     // 110 Hz einschalten
  delay(10);              // 10 Millisekunden warten
  tone(speaker, 115);     // 100 Hz einschalten
  delay(10);              // 10 Millisekunden warten
  [...]
}
```

Listing 7: Sirene Teil 1

Dass diese Methode sehr aufwändig und auch fehleranfällig ist, ist offensichtlich. Wann immer in einem Programm ein Ablauf mehrfach hintereinander wiederholt werden soll (Frequenz erhöhen, warten) können wir eine sogenannte *Schleife* verwenden:

```
[...]
void loop() {
  for( int frequenz = 100 ; frequenz <= 1000 ; frequenz = frequenz + 10 ) {
    tone(speaker, frequenz);
    delay(10);
  }
}
```

Listing 8: Sirene Teil 2: `for`-Schleife

Dieser Befehl für eine `for`-Schleife besteht aus 3 Teilen, welche durch Semikolons getrennt sind:

- Der erste Teil (`int frequenz = 100`) ist die *Initialisierung*. Dieser wird einmalig vor dem ersten Durchlauf ausgeführt. In diesem Fall wird der Wert der ganzzahligen Variable `frequenz` auf „100“ festgelegt.
- Der zweite Teil (`frequenz <= 1000`) ist die *Bedingung*. Wie bei der `if`-Anweisung wird der nachfolgende Block so lange ausgeführt, wie die Bedingung zutrifft.

- Der dritte Teil (`frequenz = frequenz + 10`) wird immer **nach** jedem Durchlauf ausgeführt. In unserem Fall wird die Frequenz zunächst um den Wert „10“ erhöht und das Ergebnis anschließend wieder in der Variablen `frequenz` gespeichert.

Der mit geschweifen Klammern umschlossene, nachfolgende Block wird also so lange ausgeführt, bis die Frequenz 1000 Hz erreicht hat.

Aufgabe II.14:

Programmiere zunächst den Sketch `02_14_Schleifen`.

Aufgabe II.15:

Ändere den Sketch so ab, dass die Tonhöhe nicht ansteigt sondern abfällt.

Aufgabe II.16:

Ergänze den Sketch zu einer kompletten Sirene, bei der die Tonhöhe zunächst ansteigt und anschließend wieder abfällt.

Merke:

Anstatt der Schreibweise `frequenz = frequenz + 10` kann auch die Kurzschreibweise `frequenz += 10` benutzt werden. `+= 10` bedeutet dabei, dass der Wert der Variablen um den Wert „10“ erhöht wird.

Merke:

Anstatt der Rechnung `frequenz = frequenz +1` bzw. `frequenz += 1` kann auch die Kurzschreibweise `frequenz++` geschrieben werden. `++` stellt dabei das sogenannte *Inkrement* dar, wodurch die Variable um den Wert „1“ erhöht wird.

Aufgabe II.17:

Ändere deinen Sketch so ab, dass kleinere Frequenzsprünge verwendet werden und sich die Sirene noch „glatter“ anhört.

II.7. Serielle Konsole

Oft sollen z. B. Messwerte nicht nur verarbeitet sondern auch ausgegeben werden, damit wir diese direkt ablesen können. Im Kapitel VI.3 werden wir ein eigenes Display an den Arduino anschließen um Werte direkt darauf ausgeben zu können. Der Arduino bietet daneben aber auch noch die *serielle Kommunikation* an, womit wir die Werte ganz leicht auf den Computer übertragen und dort dann ablesen können.

Hierfür müssen wir in der Arduino-Software am PC den *seriellen Monitor* im Menü „Werkzeuge“ öffnen.

```
1 void setup() {
2   Serial.begin(9600);
3 }
4
5 void loop() {
6   Serial.println("Hello World!");
7   delay(1000);
8 }
```

Listing 9: Erste Ausgabe auf der seriellen Konsole

Aufgabe II.18:

Programmiere den Sketch und speichere ihn als `02_18_seriell1`.

II.7.1. Programmanalyse

Um die serielle Kommunikation nutzen zu können, muss diese zuerst gestartet werden. Dies passiert einmalig mit dem Befehl `Serial.begin(9600);` im `setup`-Block. „9600“ gibt dabei die sogenannte *Baudrate* an, das bedeutet, dass 9600 Zeichen pro Sekunde übertragen werden können.

Nachdem die serielle Schnittstelle initialisiert wurde kann man mit dem Befehl `Serial.println();` Daten übertragen. In der Klammer steht dabei der zu übertragende Inhalt.

Soll ein Text ausgegeben werden, so muss dieser in doppelten Anführungszeichen geschrieben werden![13] Soll hingegen der Wert einer Variablen ausgegeben werden so muss diese ohne Anführungszeichen geschrieben werden.

Aufgabe II.19:

Neben dem Befehl `Serial.println();` gibt es auch den Befehl `Serial.print();`
Teste beide Befehle und erkläre den Unterschied.

[13] vgl. hierzu auch Kapitel II.3.1

Als nächstes soll der Wert einer Variablen ausgegeben werden.

```
int x=0;

void setup() {
  Serial.begin(9600);
}

void loop() {
  Serial.print("Wert von x: ");
  Serial.println(x);
  delay(1000);
}
```

Listing 10: Ausgabe eines Variablenwertes auf der Konsole

Aufgabe II.20:

Programmiere den Sketch und speichere ihn als `02_20_seriell2`. Beobachte die Ausgabe.

Aufgabe II.21:

Ändere den Sketch so ab, dass dieser einen Timer mit Minuten:Sekunden auf der seriellen Konsole ausgibt.
Beispielausgabe: `verstrichen: 9min 41s`
Schaffst du es auch, einen noch genaueren Timer auszugeben, der auch Zehntelsekunden mit anzeigt?
Beispielausgabe: `verstrichen: 9min 41.2s`
Speichere den Sketch als `02_21_timer`.

Aufgabe II.22:

Kopiere den Sketch aus Aufgabe II.16 und ändere diesen so ab, dass die aktuelle Frequenz auf der Konsole ausgegeben wird.

Zusatz-Aufgabe II.23:

Ändere den Sketch so ab, dass die Frequenz nur bei Vielfachen von 100 auf der Konsole ausgegeben wird.

II.8. eigene Methoden

Sketches lassen sich einfacher und übersichtlicher gestalten, wenn immer wieder auftretende Programmteile in Unterprogrammen, sogenannten *Methoden* „ausgelagert“ werden. Diese können dann mit nur einem Befehl an einer beliebigen Stelle aufgerufen werden.

Wir haben auch bisher bereits zwei Methoden programmiert, auch wenn wir diese bisher noch nicht so genannt haben: `void setup() { [...] }` und `void loop() { [...] }`.

Methoden bestehen dabei zum ersten aus einem Rückgabetyp. Bei `setup` und `loop` ist dies jeweils der „Datentyp“ `void`, das bedeutet, dass die Methode kein Ergebnis zurückgibt. Wie eine Methode Ergebnisse zurückgeben kann werden wir später sehen.

Anschließend können wir der Methode – wie bereits bei den Variablen – einen Namen geben über den die Methode später aufgerufen werden kann. Abgeschlossen wird die Definition mit Klammern.

Danach folgt noch ein in geschweiften Klammern eingeschlossener Block mit Befehlen, die mit dieser Methode ausgeführt werden sollen.

```
1  void ledAn() {
2    digitalWrite(13, HIGH);
3  }
4
5  void setup() {
6    pinMode(13, OUTPUT);
7  }
8
9  void loop() {
10   ledAn();
11 }
```

Listing 11: eine eigene Methode

In den Zeilen 1-3 legen wir uns also eine eigene Methode an. Anschließend rufen wir diese Methode in Zeile 10 auf und es werden dann die Befehle in der Methode ausgeführt.

Merke:

Unsere eigenen Methoden unterscheiden sich nur unwesentlich von den bisherigen Methoden `setup` und `loop`. Der (vorerst) einzige Unterschied ist, dass die `setup`-Methode *automatisch* beim Start des Arduinos ausgeführt wird. Die `loop`-Methode wird danach ebenfalls *automatisch* unendlich oft ausgeführt.
Ebenfalls **alle** Befehle die wir bisher ausgeführt haben sind durch das System vordefinierte Methoden, wie z. B. `digitalWrite`, `pinMode`, `Serial.println`,...
Anmerkung: Methoden werden auch oft Funktionen *genannt.*

II.8.1. Methoden mit Parametern

Oft wollen wir Befehle innerhalb einer Methode mit bestimmten Werten ausführen. Beispielsweise macht es im vorangehenden Beispiel wenig Sinn, eine Methode zu programmieren, nur um einen bestimmten Pin einzuschalten. Benutzen wir mehrere Pins, so müssten wir für jeden Pin eine einzelne Methode schreiben.

Hierfür können wir *Parameter* nutzen. Parameter kennst du schon aus der Mathematik z. B. von $\sin(\alpha)$ und anderen trigonometrischen Funktionen.

Legen wir eine Methode an, so können wir in den runden Klammern noch „Variablen“ angeben. Wollen wir anschließend die Methode aufrufen, so müssen wir einen Wert für diesen Parameter angeben:

```
1  void ledBlink(int pin, int dauer) {
2    digitalWrite(pin, HIGH);
3    delay(dauer);
4    digitalWrite(pin, LOW);
5  }
6
7  void setup() {
8    pinMode(13, OUTPUT);
9  }
10
11 void loop() {
12   ledBlink(13, 500);
13   ledBlink(13, 1000);
14 }
```

Listing 12: Methode mit Parametern

In den ersten 5 Zeilen wird die Methode `ledBlink` programmiert. Will man diese aufrufen, so muss man zwei Werte angeben: der erste gibt die Pinnummer (`int pin`) an, der zweite die Dauer, die die LED eingeschaltet bleiben soll (`int dauer`).

Innerhalb dieses Methodenblocks können wir dann auf die Parameter `pin` und `dauer` wie auf Variablen zugreifen.

Merke:

Die Parameter sind vergleichbar mit Variablen, können aber nur **innerhalb** des Methodenblocks verwendet werden!

Rufen wir jetzt die Methode auf, so müssen wir eben diese beiden Parameter angeben (s. Zeilen 12 und 13).

Merke:

Wichtig dabei ist: die Befehle werden wie bisher auch weiterhin alle **nacheinander** abgearbeitet! Das bedeutet, in Zeile 12 „springt“ der Programmablauf nach Zeile 2 und erst nachdem Zeile 4 abgearbeitet ist springt das Programm wieder zurück zu Zeile 13 (und führt anschließend die Methode erneut aus).

Merke:

Auch die Methoden, die wir bisher wie selbstverständlich verwendet haben, benutzen Parameter. Z. B. verlangt die Methode `digitalWrite` zwei Parameter: der erste gibt die Pinnummer an, der zweite Parameter bestimmt den Status des Pins (also ob die Spannung ein- oder ausgeschaltet sein soll).

Kapitel III: Erstes Projekt

Das bis hierher Gelernte soll nun zusammengefasst und damit ein Projekt erstellt werden.

Aufgabe III.1:

Erstelle eine Ampelschaltung mit folgenden Bedingungen:

- Die Ampel soll in eine Richtung für Autos gehen. (rot/gelb/grün)
- Diese Auto-Ampel soll zunächst dauerhaft grün leuchten.
- Zusätzlich soll es eine Fußgängerampel geben. (rot/grün)
- Auf Knopfdruck soll die Auto-Ampel zunächst nach rot wechseln.
- Sobald die Ampel rot ist, soll die Fußgängerampel auf grün wechseln.
- Nach einer kurzen Zeitspanne soll dann die Fußgängerampel wieder auf rot und anschließend die Auto-Ampel wieder auf grün wechseln.
- Solange die Fußgängerampel auf grün steht soll außerdem ein Signalton erklingen.

Fasse dabei Befehle zu eigenen Methoden zusammen – sofern sinnvoll!

Aufgabe III.2:

Die obige Schaltung hat (ggf.) ein Problem: durch ständiges Drücken des Tasters können die Autos quasi dauerhaft blockiert werden.
Programmiere dein Programm so um, dass nach einer rot-Phase die Autos mindestens eine festgelegte Zeitspanne lang grün haben.

Kapitel IV: Physikalische Grundlagen

IV.1. Wiederholung: elektrische Grundlagen

IV.1.1. Ladung

Es gibt positive und negative Ladung. In einem Atom sind dies Protonen und Elektronen. Daneben gibt es noch die neutral bzw. ungeladenen Neutronen. Jeder Körper besteht zunächst aus gleich vielen Elektronen wie Protonen, dadurch wirkt der Körper ungeladen. Sor man für einen Überschuss einer Ladungsart, so ist der Körper geladen.

Die Einheit der Ladung ist $[q] = 1\,\text{Coulomb} = 1\,\text{C}$

IV.1.2. Potenzial

Gleich geladene Körper stoßen sich ab, unterschiedlich geladene Körper ziehen sich an. Auch innerhalb eines geladenen Körpers stoßen sich die Elektronen gegenseitig ab. Je stärker der Körper geladen ist, desto stärker ist auch die Abstoßung, bzw. desto höher ist auch der *Ladungsdruck*. Diesen Ladungsdruck bezeichnet man als *Potenzial*.

Die Einheit des Potenzials ist $[\varphi] = 1\,\text{Volt} = 1\,\text{V}$

IV.1.3. Stromstärke

Verbindet man zwei Stellen mit unterschiedlichem Potenzial, so fließen Ladungen um diesen Unterschied auszugleichen. Fließende Ladung bezichnen wir als *elektrischen Strom*.

Unter der *Stromstärke* I versteht man, welche Ladungsmenge pro Zeit durch eine bestimmte Stelle fließt, es gilt $I = \frac{q}{t}$

Die Einheit der Stromstärke ist $[I] = 1\,\text{Ampère} = 1\,\text{A}$

IV.1.4. Spannung

Je größer der Potenzialunterschied, desto größer ist auch die Stromstärke. Den Potenzialunterschied zwischen zwei Stellen bezeichnet man als *Spannung*.

Die Einheit der Spannung ist $[U] = 1\,\text{Volt} = 1\,\text{V}$

IV.1.5. Widerstand

Die Spannung treibt den Strom an, wohingegen der *Widerstand* eine Bremse für diesen darstellt. Je größer der Widerstand, desto kleiner die Stromstärke.

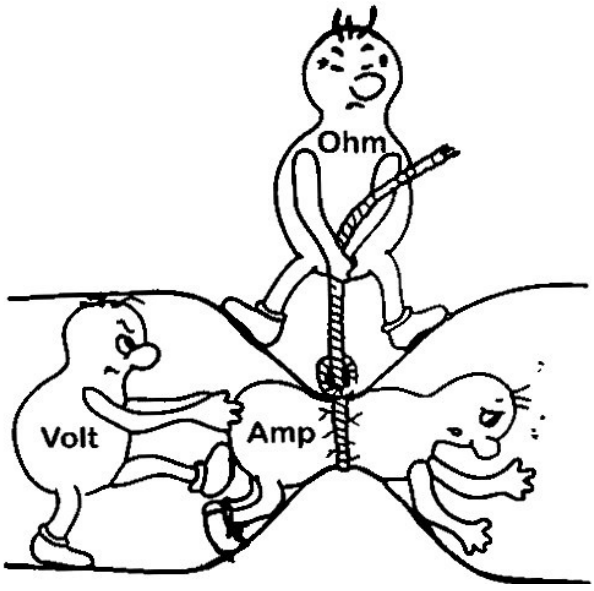


Abbildung 7: Bildlicher Zusammenhang zwischen Spannung, Stromstärke und Widerstand[14]

[14] Quelle: `http://www.sengpielaudio.com/ohms-law-illustrated.gif`

Bei einem *ohm'schen Widerstand* ist das Verhältnis zwischen Spannung und Stromstärke immer konstant, es gilt: $R = \frac{U}{I}$

Bei *nicht-ohm'schen Widerständen* ist dieser Zusammenhang nicht proportional!

Die Einheit des Widerstandes ist $[R] = 1\,\text{Ohm} = 1\,\Omega$

Aufgabe IV.1:

Welchen Widerstand muss man einsetzen, damit bei einer Spannung von 5 V eine Stromstärke von 20 mA fließt?

IV.1.6. Farbcodierung von Widerständen

Um Widerstände als Bauteile auch optisch unterscheiden zu können, werden diese mit farbigen Ringen markiert. Es gibt zwei unterschiedliche Widerstandstypen, die hauptsächlich verwendet werden: Kohleschichtwiderstände haben ein beigefarbenes Gehäuse und für gewöhnlich 4 farbige Ringe, Metallschichtwiderstände hingegen haben ein blaues Gehäuse und normalerweise 5 Farbringe.

Der jeweils letzte Ring hat meist einen etwas größeren Abstand und gibt die Toleranz an, d. h. wie genau der Widerstand ist.

Ringfarbe	1.-2. bzw. 1.-3. Ring (Widerstandszähler)	vorletzter Ring Multiplikator	letzter Ring Toleranz
schwarz	0	-	-
braun	1	x 10	1 %
rot	2	x 100	2 %
orange	3	x 1000	-
gelb	4	x 10000	-
grün	5	x 100000	0.5 %
blau	6	x 1000000	0.25 %
violett	7	x 10000000	0.1 %
grau	8	-	-
weiß	9	-	-
gold	-	x 0.1	5 %
silber	-	x 0.01	10 %

Tabelle 1: Widerstandscodierung

Beispiel IV.1:

Ein Widerstand mit den Ringen gelb/violett/braun/gold hat einen Wert von 4/7/*10/±5%, also 470 Ω mit einer Toleranz von ±23.5 Ω.
Ein Widerstand mit den Ringen braun/grün/schwarz/rot/braun hat einen Wert von 1/5/0/*100/±1%, also 15 kΩ mit einer Toleranz von ±150 Ω.
Ein Widerstandswert von 180 Ω hat die Ringe 1/8/*10, also braun/grau/braun (plus den Toleranzring, den wir hier allerdings mal vernachlässigen)

Aufgabe IV.2:

Bestimme den Wert der Widerstände mit folgenden Ringen:

a) braun/schwarz/braun/gold

b) rot/rot/gelb/braun

c) grün/blau/schwarz/gelb/grün

d) weiß/braun/schwarz/silber/violett

Aufgabe IV.3:

Bestimme die Codierung für folgende Widerstandswerte. Die Toleranz kann vernachlässigt werden. Gib die Codierung jeweils mit 3 und mit 4 Ringen an.

a) 47 kΩ

b) 91 Ω

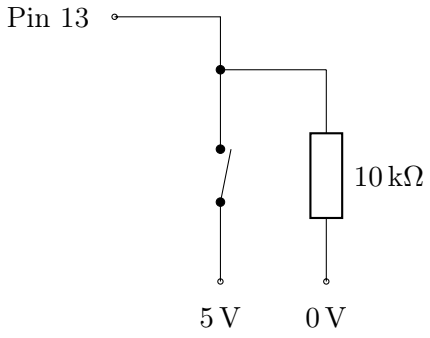
c) 10 kΩ

d) 2.2 MΩ

IV.2. Pullup- und Pulldown-Widerstände

Z. B. im Kapitel II.4 (Taster) haben wir einen sogenannten *Pulldown-Widerstand* verwendet.

Wir haben dabei nebenstehende Schaltung gebaut. Man könnte meinen, dass der rechte Teil davon überflüssig ist, denn sobald wir den Taster drücken liegt das Potenzial[15] von 5 V am Pin 13 an. Dies ist soweit auch korrekt.



Doch was passiert, wenn der Taster nicht gedrückt ist? Dann liegt am Pin 13 nicht (wie man annehmen könnte) ein Potenzial von 0 V an, sondern es herrscht ein undefinierter Zustand! In diesem Fall *kann* das Programm funktionieren, es könnte aber ebensogut auch sein, dass der Pin 13 einen falschen Zustand erkennt.

Um dem Pin einen definierten Zustand von 0 V zu geben wenn der Taster nicht gedrückt ist, müssen wir diesen an diesem Nullpotenzial anschließen. Um jedoch einen Kurzschluss zu verhindern sobald der Taster gedrückt wird, verwenden wir hier einen relativ großen Widerstand.

Da mithilfe von diesem Widerstand das Potenzial am Pin auf 0 V hinuntergezogen wird spricht man von einem *Pulldown-Widerstand*.

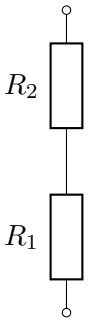
Vertauscht man den 0 V und den 5 V Anschluss, so spricht man von einem *Pullup-Widerstand*, da das Potenzial wenn der Taster nicht gedrückt ist auf 5 V hochgezogen wird. Sobald der Taster gedrückt wird liegt dann ein Potenzial von 0 V am Pin an.

Anmerkung: der Befehl `digitalRead(pin);` *ergibt dann natürlich „0“ wenn der Taster gedrückt ist!*

[15] Umgangssprachlich spricht man nicht vom Potenzial sondern überall von der Spannung.

IV.3. Spannungsteiler, Potentiometer

IV.3.1. Reihenschaltung



Werden zwei (oder mehrere) Widerstände hintereinander geschaltet, so spricht man von einer *Reihenschaltung*. Die Ladung muss dabei durch beide Widerstände durchfließen und wird somit auch durch beide Widerstände „gebremst“.

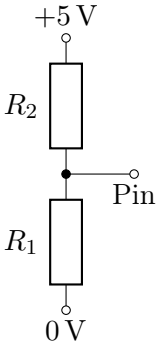
Der Gesamtwiderstand ist deshalb

$$R_{\text{ges}} = R_1 + R_2$$

Für die Gesamtstromstärke gilt dann nach dem *ohm'schen Gesetz*

$$I_{\text{ges}} = \frac{U_{\text{ges}}}{R_{\text{ges}}}$$

IV.3.2. Spannungsteiler



Die Stromstärke, die durch die Widerstände R_1 bzw. R_2 fließt entspricht genau der Gesamtstromstärke I_{ges}. Deshalb gilt für die Spannungen an den einzelnen Widerständen

$$U_1 = R_1 \cdot I_{\text{ges}} = \frac{R_1}{R_{\text{ges}}} \cdot U_{\text{ges}} \qquad \text{und} \qquad U_2 = R_2 \cdot I_{\text{ges}} = \frac{R_2}{R_{\text{ges}}} \cdot U_{\text{ges}}$$

Greift man nun das Potenzial in der Mitte der beiden Widerstände ab, so bekommt man hier genau U_1.[16]

Da diese Schaltung die Gesamtspannung auf die Widerstände aufteilt nennt man diese auch *Spannungsteiler*.

Beispiel IV.2:

Wählt man als Widerstände $R_1 = 200\,\Omega$ und $R_2 = 50\,\Omega$, so erhält man am Pin ein Potenzial von

$$U_1 = \frac{200\,\Omega}{250\,\Omega} \cdot 5\,\text{V} = 4\,\text{V}$$

[16] Physikalisch korrekt wäre für das Potenzial die Bezeichnung φ, denn U gibt eine *Spannung* an. Da jedoch am Widerstand R_1 genau die Spannung U_1 anliegt und am unteren Ende das Potenzial $\varphi = 0\,\text{V}$ anliegt, entspricht das Potenzial, welches am Pin anliegt gerade dem Wert dieser Spannung U_1.

Aufgabe IV.4:

Welches Potenzial liegt zwischen den beiden Widerständen an, wenn

a) $R_1 = 100\,\Omega$, $R_2 = 100\,\Omega$

b) $R_1 = 2\,\text{k}\Omega$, $R_2 = 6\,\text{k}\Omega$

c) $R_1 = 1\,\Omega$, $R_2 = 1\,\text{k}\Omega$

Was würde passieren, wenn R_1 durch ein Kabel mit vernachlässigbarem Widerstand ersetzt wird, d. h. $R_1 = 0\,\Omega$?
Wenn die Leitung unterbrochen ist, kann man das auch als *„unendlich großen“ Widerstand* ansehen. Welches Potenzial liegt am Pin an, wenn R_1 unendlich groß wird?

Aufgabe IV.5:

Welche Widerstände könnte man verwenden, damit am Pin ein Potenzial von

a) 4.5 V

b) 2 V

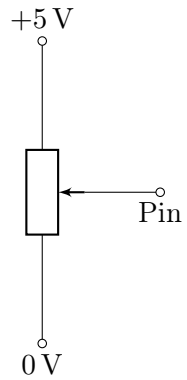
c) 0.1 V

anliegt?

IV.3.3. Potentiometer

Ein *Potentiometer* (oder kurz „Poti“) bildet so einen variablen Spannungsteiler als Bauteil ab. Es besitzt 3 Anschlüsse: zwischen den äußeren beiden Anschlüssen wirkt der gesamte Widerstand, d. h. hier wird wie bei den einzelnen Widerständen die Gesamtspannung angeschlossen.

Der dritte Anschluss am Potentiometer greift das Potenzial dazwischen ab. Diese Stelle kann nun durch den Schiebe- oder Drehregler eingestellt werden, d. h. man kann das Verhältnis von R_1 und R_2 einfach durch Schieben bzw. Drehen einstellen.



IV.3.4. Messung des Potenzials

Das Potenzial kann mit dem Arduino an den *analogen Eingängen* gemessen werden.[17] Dieses wird wie in Kapitel I.3.1 bereits besprochen als Wert zwischen 0 ($\hat{=}$ 0 V) und 1023 ($\hat{=}$ 5 V) angegeben.

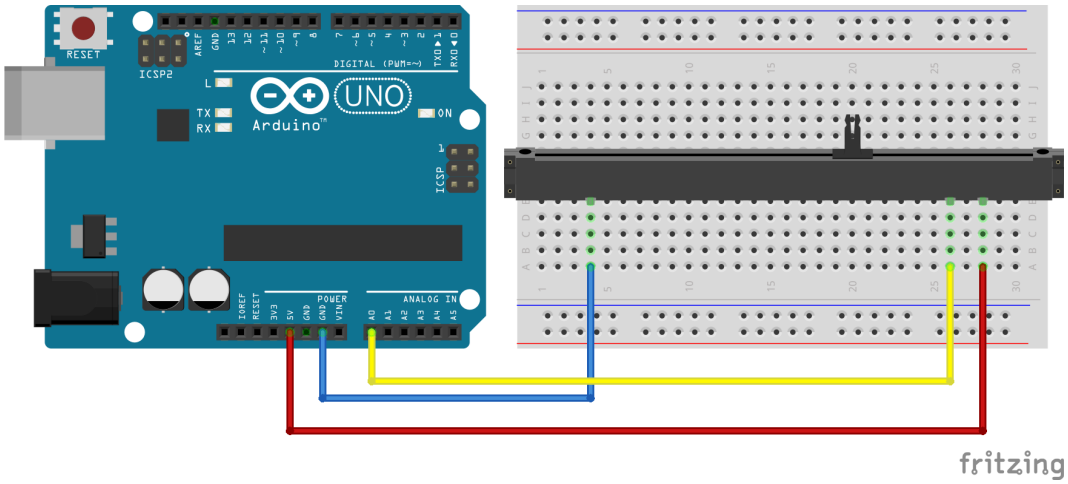


Abbildung 8: Messung analoger Werte.

```arduino
int analogPin = 0;

void setup() {
  Serial.begin(9600);
}

void loop() {
  int wert = analogRead(analogPin);

  Serial.println(wert);

  delay(1000);
}
```

Listing 13: Messen von analogen Werten

Aufgabe IV.6:

Baue die Schaltung nach und verbinde dabei +5 V und GND mit den beiden äußeren Anschlüssen. Den mittleren Anschluss verbindest du mit dem Pin *A0* auf dem Arduino.
Hinweis: Du kannst alternativ auch einen Dreh-Potentiometer nutzen.
Übertrage anschließend den Sketch `04_06_analog` auf den Arduino und beobachte die serielle Konsole während du das Potentiometer veränderst.

Merke:

Anders als bei den digitalen Pins muss man die analogen Pins nicht als `INPUT` einstellen, denn die analogen Pins funktionieren immer nur als Eingang!

[17] Technisch gesehen wird hier die Spannung gegenüber GND = 0 V gemessen. Dies entspricht aber genau dem oben angegebenen Potenzial.

Aufgabe IV.7:

Der Gesamtwiderstand der Potentiometer ist im Normalfall darauf aufgedruckt. Falls nichts aufgedruckt ist nimm $R_{\text{ges}} = 10\,\text{k}\Omega$ an.
Rechne einen Wert, den der Arduino auf der Konsole angibt, zunächst per Hand in das mittlere Potenzial am Pin $A0$ um und bestimme daraus U_1 und U_2. Anschließend bestimme die Widerstände R_1 und R_2.

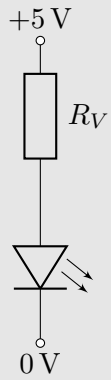
Diese Rechnung soll später in Kapitel V automatisch mit dem Arduino durchgeführt werden.

IV.3.5. Vorwiderstand

Oftmals darf ein Gerät nicht mit 5 V sondern nur mit einer deutlich niedrigeren Spannung betrieben werden. LEDs vertragen (je nach Farbe und Bauart) nur etwa zwischen 1.5 V und 2.5 V. Würde man diese direkt an die 5 V-Spannungsversorgung des Arduino anschließen, so würde die LED sehr schnell „durchbrennen“.

Um diese Geräte betreiben zu können verwendet man einen *Vorwiderstand* R_V. Wie beim Spannungsteiler soll an diesem R_V diezu hohe Spannung abfallen, so dass am eigentlichen Bauteil, welches in Reihe geschaltet ist, nur noch die richtige Betriebsspannung übrig bleibt.

Beispiel IV.3:



Eine rote LED verträgt nur eine Spannung von 1.5 V bei einer Stromstärke von 20 mA. Um diese trotzdem betreiben zu können muss also am Vorwiderstand R_V eine Spannung von 3.5 V anliegen.
Da eine Stromstärke von 0.02 A fließen soll, muss nach dem *ohm'schen Gesetz* der Widerstand also einen Wert von

$$R_V = \frac{U_V}{I} = \frac{3.5\,\text{V}}{0.02\,\text{A}} = 175\,\Omega$$

haben. Deshalb haben wir bisher bei einer roten und gelben LED einen 180 Ω-Widerstand benutzt.
Hinweis: LEDs sind wie auch die meisten anderen Geräte nicht ganz empfindlich gegen höhere Spannungen, deshalb können auch geringfügig niedrigere Widerstände benutzt werden.

Aufgabe IV.8:

Eine weiße LED benötigt eine Spannung von 2.8 V bei einer Stromstärke von 40 mA. Berechne, welcher Vorwiderstand benutzt werden sollte.

IV.4. Besonderheiten einer LED

Abbildung 9: Nahaufnahme einer LED[18]

Eine herkömmliche Glühlampe leuchtet, weil durch den fließenden elektrischen Strom ein Draht zum Glühen gebracht wird. Der Nachteil hiervon ist, dass neben dem gewünschten Licht auch noch sehr viel Energie in Wärme umgewandelt wird. Das Licht macht dabei nur etwa 5% der Energiemenge aus.

LEDs sind hier deutlich effizienter und wandeln fast die komplette Energie in Licht um. Sie basieren auf *Halbleitertechnologie* und nutzen verschiedene *Energieniveaus* aus um Licht mit einer bestimmten Frequenz zu erzeugen. Das Licht von Glühbirnen hingegen besteht aus vielen unterschiedlichen Frequenzen.

Da die LEDs deshalb weniger Energie zum Betrieb benötigen als Glühbirnen können sie auch am Arduino betrieben werden. Hierbei ist aber zu beachten, dass LED (wie auch „normale“ *Dioden*) den Strom nur in eine Richtung durchlassen und deshalb richtig herum angeschlossen werden müssen!

Um zu bestimmen, wie die LED angeschlossen werden muss gibt es 3 Merkmale:

- Bei neuen LED ist ein Beinchen kürzer. Dieses ist die sogenannte *Kathode* und muss am Minuspol angeschlossen werden. Merken kann man es sich, wenn man das Schaltsymbol einer Batterie kennt: auch hier ist der kürzere Strich der Minuspol.
- An der Unterseite des Gehäuses ist ein Ring angebracht. Dieser ist auf der Kathodenseite abgeflacht.
- Bei durchsichtigen LEDs lässt sich im Inneren der LED erkennen, dass diese aus einem kleinen und einem großen Teil besteht. Der größere Teil wird aufgrund seiner Form auch oft „Kelch“ genannt. Dieser ist ebenfalls auf der Seite der Kathode.

Merke:

Die **K**athode ist die Seite mit dem **k**urzen Bein und dem **K**elch innen drin. Diese **K**athode gehört an den Minuspol der Batterie.

IV.5. Pulsweitenmodulation PWM

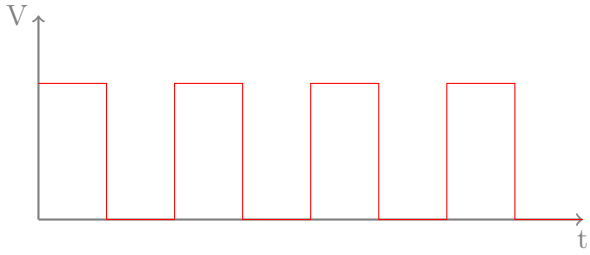
Herkömmliche Glühbirnen lassen sich *dimmen*, indem man die Spannung verändert. Daraus resultiert eine geringere Stromstärke und eine geringere Helligkeit.

Beim Arduino mit LED haben wir hierbei zwei Probleme: zum einen können wir die Ausgangsspannung nicht variieren, zum anderen lässt sichdie Helligkeit von LEDs auch gar nicht über die Spannung regeln!

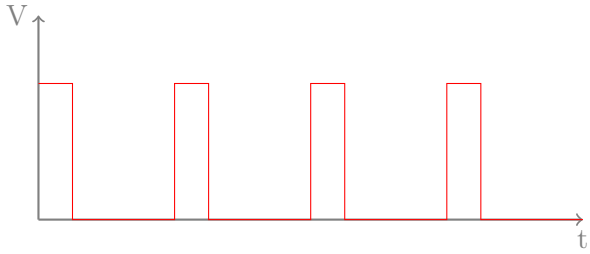
Wollen wir trotzdem die Helligkeit von LEDs mit dem Arduino regeln nutzen wir die sogenante *Pulsweitenmodulation*. Der Trick dabei ist, dass die Spannung an der LED sehr schnell ein- und ausgeschaltet wird, so dass mit bloßem Auge kein Flackern erkennbar ist, insgesant die LED aber dunkler wird.

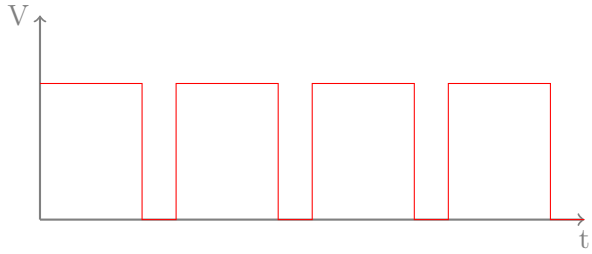
[18] Quelle: `https://de.wikipedia.org/wiki/Leuchtdiode`

Der Spannungsverlauf sieht dabei so ähnlich aus:



Hierbei ist die Spannung genau die Hälfte der gesamten Zeit eingeschaltet, die LED wirkt dann halb so hell. Alternativ kann auch das die Spannung z. B. nur ein Viertel oder drei Viertel der Zeit eingeschaltet sein:





Merke:

Die Funktion der Pulsweitenmodulation ist im Arduino bereits eingebaut und wird auf allen digitalen Pins unterstützt, die mit einem „∼“-Symbol gekennzeichnet sind.

```
1  int ledPin = 3;
2
3  void setup() {
4    pinMode(ledPin, OUTPUT);
5  }
6
7  void loop() {
8    analogWrite(ledPin, 128);
9    delay(1000);
10 }
```

Listing 14: Dimmen einer LED durch Pulsweitenmodulation

Aufgabe IV.9:

Baue wieder die erste Schaltung mit einer LED nach und schließe die LED aber am Pin 3 an. Übertrage anschließend den Sketch `04_09_LED_dimmen` und teste verschiedene Werte für den zweiten Parameter der `analogWrite`-Methode.

Der zweite Parameter der `analogWrite`-Methode gibt dabei die Helligkeit an. Du kannst hier Werte von 0 (≙ LED ist aus) und 255 (≙ volle Helligkeit) eingeben.

Aufgabe IV.10:

Programmiere einen Sketch so, dass die LED langsam heller und anschließend wieder dunkler wird. Benutze hier – ähnlich der Lautsprecher-Aufgabe II.16 – zwei Schleifen um die Helligkeit zu regeln.

Kapitel V: Rechnen mit dem Arduino

Kapitel VI: Weitere Sensoren, Aktoren und Module

VI.1. Helligkeitssensor (Widerstand)

VI.2. Temperatursensor (Widerstand)

VI.3. LC-Display

VI.4. Motoren

VI.4.1. Gleichstrommotor

VI.4.2. Schrittmotor

VI.4.3. Servomotor

VI.5. Lichtschranke

VI.6. Ultraschall-Entfernungssensor

Kapitel VII: Liste der Sensoren, Aktoren, Geräten und Modulen

Es folgt eine Liste mit Geräten und Beispiele für Einsatzzwecke

- LED: Signalanzeige
- Lautsprecher: Tonausgabe, Warntöne
- Taster: Steuerung
- Potentiometer: Schieberegler, Steuerung
- Helligkeitssensor: Messung der Helligkeit, automatisches Licht
- Temperatursensor: Messung der Wärme, automatische Fensteröffnung
- Luftfeuchtigkeitssensor:
- LCD: Anzeige von Messwerten
- Lichtschranke: Messung von Geschwindigkeit/Bewegung
- Reflexlichtschranke: Erkennung von Bewegungen, Bewegungsmelder
- Ultraschallsensor: Abstandsmessung
- Infrarotsensor: Abstandsmessung
- Motor: Bewegung, Fahren
- Stromstärkesensor: Messung der Stromstärke
- Kameramodul: Bilder aufnehmen
- Fingerabdrucksensor: Sicherheit z. B. Türöffner
- RFID-Sensor: Zeiterfassung
- Tastenfeld: Zugangscode z. B. für Türöffner
- Netzwerkmodul: Anbindung und Datenübertragung ins Internet
- WLAN-Modul: drahtlose Anbindung und Datenübertragung ins Internet/an mobile Geräte
- Bluetooth-Modul: Datenübertragung ans Handy
- Alkohol-Gas-Sensor: Messung Atemalkoholgehalt
- CO-Sensor: Messung der Konzentraton von Kohlenmonooxid und brennbaren Gasen
- GPS: Navigation
- LCD-Touchscreen: Anzeige von Messwerten und Steuerung
- Joystick: Steuerung von Spielen
- Relais: Schaltverstärkung
- Mikrofon: Lautstärkemessung
- Flame-Sensor: optische Erkennung von offenem Feuer
- Hall-Sensor: Messung eines Magnetfeldes
- Reed-Sensor: Erkennung eines Magneten in der Nähe
- Puls-Sensor: Messung der Herzfrequenz
- Lagesensor: Messung der Ausrichtung (nicht Winkel!)
- Infrarot-Sensor: Benutzung von Fernbedienungen
- Erschütterungssensor: Erkennung von Vibrationen
- Laser-Modul: Erzeugung eines Laserstrahls
- Regen-Sensor