



Skript zur Einführung des Arduino im Wahlpflichtfach „Informatik/Physik“ (MINT-Profil) im Schuljahr 2020/21

Sebastian Voß

E-Mail: sebastian@el-voss.de

Gymnasium Marianum

Herzog Arenberg Str. 65

49716 Meppen

Alle Bilder wurden vom Autor selbst erstellt, wenn kein entsprechender Hinweis gegeben wird. Zeichnungen wurden mit *TikZ* und *InkScape* angefertigt. Schaltpläne wurden mit *QElectroTech* erstellt. Dieses Skript wurde mit *LAT_EX* geschrieben. Screenshots wurden mit *Shutter* (unter Linux) bzw. mit dem *Snipping Tool* (Windows) erstellt.

Alle Programme und Schaltungen in diesem Skript wurden sorgfältig erstellt und geprüft. Der Autor kann nicht für Schäden verantwortlich gemacht werden, die bei der Verwendung dieses Skripts entstehen.

Internet-Links waren zum Zeitpunkt der Erstellung dieses Skripts funktionsfähig und führten zu sinnvollen Inhalten. Es kann jedoch nicht ausgeschlossen werden, dass sie inzwischen zu ganz anderen Inhalten führen.

Didaktisch-methodische Bemerkungen zu diesem Skript befinden sich im Anhang. Fortbildungsaufgaben können Sie an die oben angegebene E-Mail-Adresse richten.

Dieses Skript ist unter den Creative-Commons-Bedingungen BY-NC-SA 4.0 veröffentlicht.

<https://creativecommons.org/licenses/by-nc-sa/4.0/deed.de>

Unter den dort angegebenen Bedingungen darf das Skript frei genutzt, verändert und vervielfältigt werden. Wenn es für Sie und Ihren Unterricht hilfreich ist, würde sich der Autor über eine freiwillige Spende freuen: <https://www.paypal.me/arduinorskript>.

Zuletzt bearbeitet am 20. August 2020.

Inhaltsverzeichnis

1. Einleitung	1
1.1. Weitere Informationen: Bücher und Internet	2
2. Eine kurze Einführung zum Arduino und zu Nepo	3
2.1. Der Aufbau des Arduino UNO	4
2.2. Das Open Roberta Lab und Nepo4Arduino	5
2.3. Digitale Ausgänge steuern	6
2.4. Aufbau von Schaltungen auf der Steckplatine	7
2.5. Widerstandsringe ablesen	8
2.6. Vermischte Übungen	10
3. Bausteine von Algorithmen	11
3.1. Entscheidungen programmieren	12
3.2. Kommunikation mit dem Arduino: Der serielle Monitor	14
3.2.1. Reflexion: Datentypen	15
3.3. Entscheidungen mit mehreren Kriterien treffen	16
3.4. Programme mit Variablen und Schleifen effizient steuern	19
3.4.1. Zufällige Ereignisse und Wiederholungen programmieren	20
3.4.2. Wiederholungen mit Bedingungen steuern	21
3.4.3. Zählschleifen programmieren	22
3.5. Programme mit Struktogrammen dokumentieren	25
3.6. Eigene Funktionen definieren	27
3.7. Debugging: Fehler im Programm finden	31
3.8. Das EVA-Prinzip	32
3.9. Vermischte Übungen	35
3.10. Ausblick	36
4. Elektrische Grundlagen zu digitalen und analogen Pins	37
4.1. Spannung, Stromstärke und Widerstand berechnen	38
4.2. Das elektrische Potential	41
4.3. Pulsweitenmodulation	44
4.4. Spannung messen	46
4.5. Drehregler verwenden	48
4.5.1. Die Verwendung eines Potentiometers ohne Mikrocontroller	49
4.6. Helligkeit messen	51



4.7. Temperatur messen	56
4.8. Schaltungen mit Transistoren steuern	58
4.9. Elektromotor und Diode	61
4.9.1. Elektromotor mit Transistor steuern	63
4.9.2. Elektromotor mit Relais steuern	64
4.9.3. Elektromotoren mit dem Motortreiber-IC L293D steuern (inkl. Drehrichtung)	68
4.10. Vermischte Übungen	71
4.11. Ausblick	72
5. Erweiterung des Werkzeugkastens: Bauteilkunde	73
5.1. Servo	74
5.2. Schrittmotor	75
5.3. Liquid Crystal Display (LCD)	77
5.4. Neigungsschalter	80
5.5. Bewegungsmelder	82
5.6. Joystick	84
5.7. Infrarot-Sensor mit Fernbedienung	86
5.8. Temperatur- und Luftfeuchtigkeitssensor DHT-11	88
5.9. Temperatursensor TMP36	89
5.10. Tropfensensor und Feuchtigkeitssensor	90
5.11. Pulssensor	92
5.12. Ultraschallsensor	94
5.13. RFID	95
5.14. Projektarbeit	98
6. Steuerung komplexer Informatik-Systeme	99
6.1. Automaten	100
A. Anhang	105
A.1. Ein kurzer Überblick - die Funktionseinheiten des Arduino Uno	105
A.2. Installation der integrierten Entwicklungsumgebung (IDE) für den Arduino	107
A.3. Mit der Arduino IDE ein Programm auf den Arduino Uno übertragen	107
B. Didaktisch-Methodische Bemerkungen	110
B.1. Zielgruppe	111
B.2. Organisatorisches: Arduino Starter Kits, Raum, Zusammenarbeit	111
B.3. Zum Aufbau dieses Skripts	112
B.3.1. Aufbau der Kapitel	112
B.3.2. Erklärung zu Links, Symbolen und eingebetteten Arbeitsblättern	113
B.4. Erfahrungsbericht	114
B.5. Ausblick	115
B.6. Anpassung / Weiterentwicklung	115

1. Einleitung

Anfang der 2000er Jahre sollte der Professor Massimo Banzi seinen Studenten beibringen, wie man interaktive elektrische Schaltungen für künstlerische Projekte erstellt. Leider erforderten die damals vorhandenen Mikrocontroller einiges an Hintergrundwissen, bevor man irgend etwas mit ihnen anfangen konnte. Professor Banzi hatte dieses Wissen - er mochte seinen Studenten, die ein künstlerisches Designstudium gewählt hatten, jedoch kein Studium zum Elektroingenieur zumuten, ehe sie fähig wären, künstlerische elektronische Projekte umzusetzen.

So kam es, dass Massimo Banzi und David Cuartielles im Jahr 2005 den ersten Arduino entwickelten. Dieser sollte einfach zu handhaben, günstig anzuschaffen und - eine ziemlich neue Idee für Hardware - sein Aufbau sollte frei zugänglich sein, sodass er auch von anderen nachgebaut werden konnte. Natürlich sollte auch die Entwicklungsumgebung, mit der sich der Arduino programmieren lässt, frei verfügbar sein. Diese drei Eigenschaften des Arduino führten innerhalb weniger Jahre zu einer unglaublichen Verbreitung des Mikrocontrollers, die heute nicht nur Studenten und Universitäten betrifft, sondern auch Bastler, Künstler und Schulen weltweit. Elektronik und Programmierung wurde von einer kleinen Nische für Nerds zu einem allgemein verfügbaren Werkzeug, mit dem jeder, der sich ein wenig mit dem Thema beschäftigt, seine Kreativität auf eine neue Weise ausleben kann.

Diese Sichtweise auf den Arduino ist ein wichtiges Ziel dieses Kurses. Es geht nicht nur um den Ausbau eines theoretischen Weltverständnisses, das sonst häufig im Zentrum steht. Es geht um die Erweiterung der praxisbezogenen Fähigkeiten und Fertigkeiten, mit denen wir unsere Umwelt selbst gestalten, erweitern und reparieren können. Jeder neue Lerninhalt soll im Kontext der Fragestellung „Wozu ist das nützlich?“ oder „Was kann man damit anfangen?“ präsentiert werden. Diese Grundhaltung knüpft an die sogenannte *Maker*-Bewegung an - eine ständig größer werdende Gemeinschaft von Menschen, die neue Werkzeuge wie den Arduino, 3D-Drucker und Laser-Cutter nutzen, um selbst Dinge zu erschaffen, Dinge mit selbst ausgedachten Features zu erweitern (zu hacken) oder Dinge zu reparieren, die ganz im Sinne des Herstellers viel zu früh kaputt gegangen sind. Dabei gab es „Maker“, also Bastler, schon immer, allerdings öffnen die neuen Werkzeuge und insbesondere der Arduino Möglichkeiten, die es früher nicht gab. Dabei gehört es zum Konzept des Bastelns, dass man nicht immer alle Dinge bis ins Detail versteht, sondern sich traut, Dinge auszuprobieren und nicht zu enttäuscht zu sein, wenn diese Versuche daneben gehen. Innerhalb eines schulischen Kurses, in dem wir nicht mit selbst bezahlten Materialien experimentieren, sind diesem Ansatz Grenzen gesetzt, allerdings werden wir so weit wie möglich auch frei experimentieren. Natürlich soll das Verständnis dessen, was man da gebaut hat, nicht auf der Strecke bleiben, denn erst dieses Verständnis ermöglicht es systematisch, Fehler zu beseitigen oder neue Funktionen in vorhandene Schaltungen zu integrieren.

1.1. Weitere Informationen: Bücher und Internet

Wenn man sein Interesse und seine Freude am Basteln entdeckt hat, will man häufig weitere Informationen haben, weitere Projekte begutachten oder weitere Ideen finden, die die eigene Kreativität wiederum beflügeln. Die große Arduino-Gemeinschaft ist im Internet sehr aktiv und stellt zahlreiche Informationen bereit. Die im folgenden genannten Quellen boten mir selbst einen guten Einstieg und ich ziehe sie bei Fragen immer noch gerne heran.

Die wichtigste Anlaufstelle ist die Projekthomepage zum Arduino (www.arduino.cc). Dort finden sich nicht nur viele Informationen rund um den Arduino, sondern auch kreative Projektvorstellungen. Weitere Projekte mit fertigen Anleitungen zum Nachbauen finden sich auf Instructables (<http://www.instructables.com/>). Auch Youtube stellt zahlreiche Videos von stolzen Makers bereit, die ihr Projekt vorstellen. Ein Beispiel für einen professionell betriebenen Kanal zu Elektronik ist *GreatScott!* (<https://www.youtube.com/user/greatscottlab>). Ein weiterer professionell betriebener Kanal mit vielen guten Bastelanleitungen ist *bitluni's lab* (<https://www.youtube.com/user/bitlunislab>). Viele gute deutschsprachige Anleitungen zum Arduino findet man bei *Makerblog.at - Arduino & Co* (<https://www.youtube.com/user/makerblogAT>). Nicht auf Youtube, sondern auf ihrer eigenen Homepage bieten die Leute von *Funduino* (<https://funduino.de/>) ebenfalls gute Anleitungen für Einsteiger an.

Wer ein Buch bevorzugt, findet in dem Buch *Arduino Workshops* von John Boxall einen sehr guten Ratgeber, der ganz ähnlich zu diesem Skript schrittweise und anhand von konkreten Projekten in die Welt des Arduino einführt.

Ein umfangreiches und ebenfalls praxisbezogenes Nachschlagewerk ist das Buch *Arduino Praxiseinstieg* von Thomas Brühlmann. Es ist gut sortiert und bietet eine schnelle Hilfe für die meisten Fragen rund um den Arduino.

Alle genannten Quellen haben allerdings gemeinsam, dass sie mit der textbasierten Arduino - Programmierumgebung arbeiten und daher einen etwas komplexeren Einstieg bieten als dieses Skript.

2. Eine kurze Einführung zum Arduino und zu Nepo

Der Arduino ist ein Mikrocontroller, der vom italienischen Professor Massimo Banzi entwickelt wurde, damit seine Studenten im Bereich *Design* eine einfach zugängliche Möglichkeit fanden, Elektronik für künstlerische Projekte zu nutzen. Die vorgenommenen Vereinfachungen in der Handhabung von Mikrocontrollern gefielen jedoch nicht nur den Studenten von Massimo Banzi, sondern auch zahlreichen Studenten anderer Fachrichtungen, Schülern, Hobby-Elektronikern und sogar Fachleuten in der Industrie, sodass der Arduino rasch eine weltweite Verbreitung fand.

In diesem Kapitel lernst du . . .

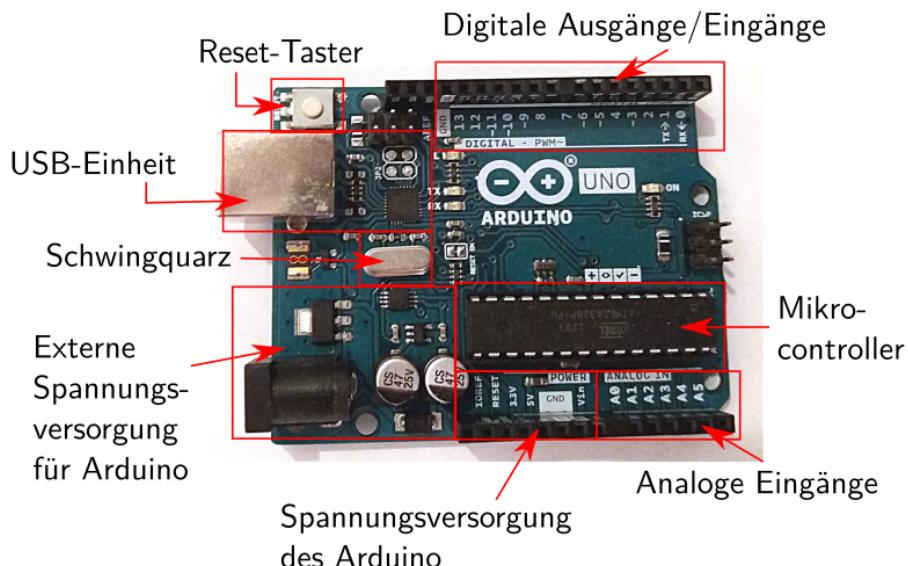
- . . . wie der Arduino aufgebaut ist,
- . . . wie man den Arduino mit dem PC verbindet und mit Nepo im Open Roberta Lab programmiert,
- . . . die digitalen Pins als Ausgänge zu benutzen, um eine LED zu steuern,
- . . . Schaltungen auf dem Steckbrett aufzubauen.

Projekte in diesem Kapitel:

> Ampel	7	> Augentestgerät	8
---------------	---	------------------------	---

2.1. Der Aufbau des Arduino UNO

Mit der Zeit entwickelten sich zahlreiche andere Modelle des Arduino, die kleiner oder größer waren, über mehr oder weniger Anschlüsse verfügten, schneller oder langsamer waren usw. Das Standardmodell ist heute der Arduino Uno, den auch wir verwenden.



B 2.1. Die wichtigsten Komponenten eines Arduino Uno.

Abbildung 2.1 zeigt die wichtigsten Komponenten des Arduino Uno. Eine genauere Beschreibung dieser Funktionseinheiten und ihrer spezifischen Eigenschaften findet sich in Anhang A.1. Wichtig sind an dieser Stelle vor allem folgende Punkte:

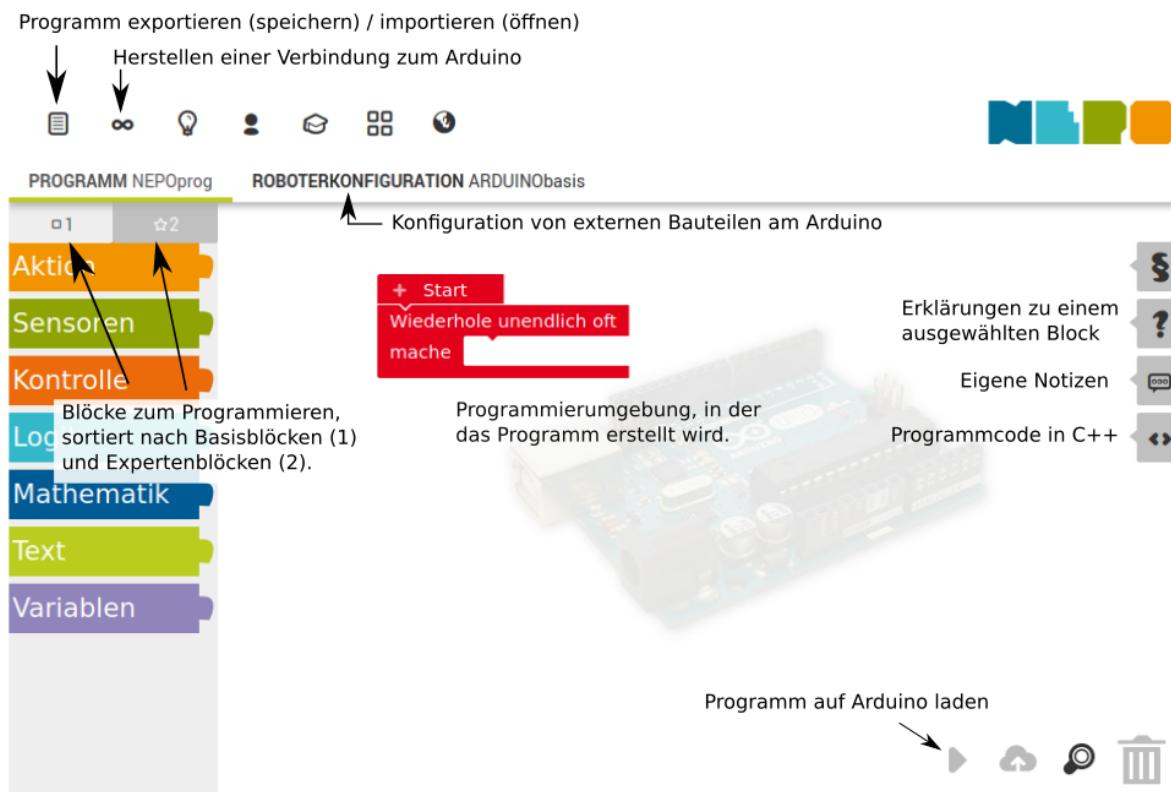
- Über den USB-Anschluss und das mitgelieferte Kabel lässt sich der Arduino mit dem PC verbinden und programmieren.
- Das Programm läuft nach dem Übertragen auf dem eigentlichen Mikrocontroller, dem langen schwarzen Ding in der Mitte. Der ganze Rest auf dieser kleinen Platine dient der einfacheren Handhabung des Mikrocontrollers.
- An den Seiten befinden sich die Pin-Leisten, an die sich zum Beispiel LEDs anschließen lassen. Die Pins sind durchnummiert, sodass sie im Programm angesprochen werden können. *GND* steht für „Ground“ oder den Minus-Kontakt. *5V* steht für den Plus-Kontakt und gibt an, dass dort stets eine Spannung von 5V anliegt, wenn der Arduino über USB oder Batterie mit Strom versorgt wird. Die durchnummerierten Digitalpins können durch das Programm ebenfalls auf 5V gesetzt werden (*HIGH*), aber auch auf 0V, sodass kein Strom fließt (*LOW*).

2.2. Das Open Roberta Lab und Nepo4Arduino

Nepo4Arduino ist eine graphische Programmiersprache, die vom Fraunhofer Institut für Intelligente Analyse- und Informationssysteme im Rahmen des Open Roberta Labs für verschiedene Robotersysteme entwickelt wird. Durch die bereit gestellten Blöcke wird der Einstieg in die Programmierung des Arduino sehr einfach gemacht. Gleichzeitig lassen sich aber auch sehr komplexe Programme entwickeln und man kann jederzeit das textbasierte Programm in der Sprache C++ einsehen.

Um den Arduino zu programmieren, ruft man die Adresse <https://lab.open-roberta.org/> auf und wählt dann ∞ Nepo4Arduino und im nächsten Schritt den Arduino Uno aus. Damit das dort erstellte Programm aus dem Browser auf den Arduino übertragen werden kann, muss außerdem der Open Roberta Connector installiert werden. Auf der [Wiki-Seite](#) gibt es eine [Installations-Anleitung](#) und eine [Anleitung zum Verbinden des Arduino mit dem Open Roberta Lab](#).

Abbildung 2.2 zeigt eine Übersicht über die Oberfläche des Open Roberta Lab, jedoch lernt man sie am besten kennen, indem man einfach drauf loslegt und ausprobiert, was sich damit anstellen lässt.



B 2.2. Übersicht über die Funktionen von Open Roberta Lab.

2.3. Digitale Ausgänge steuern

Ziel: Es soll das erste Testprogramm auf den Arduino übertragen werden, mit dem man üblicherweise überprüft, ob der Arduino (oder ein anderer Mikrocontroller) richtig funktioniert.

Aufgabe 1: Test der Funktionsweise

Zum Testen der Funktionsweise soll die bordeigene LED zum Blinken gebracht werden:

- Stelle LED an.
 - Warte eine Sekunde.
 - Stelle LED aus.
 - Warte eine Sekunde.
- a) Verbinde den Arduino mit dem Open Roberta Lab nach der [Anleitung zum Verbinden des Arduino](#).
 - b) Überprüfe zunächst in der Roboterkonfiguration, ob der Block für die Board-LED bereits eingestellt ist. Benenne die LED als „BoardLED“.
 - c) Erstelle das oben formulierte Programm mit Nepo4Arduino und übertrage es auf den Arduino.
 - d) Zum Speichern des Programms wähle im  Menü die Funktion  exportiere Programm. Finde das Programm im Download-Ordner deines Computers und benenne es als Blink.xml. Speichere es dann an einem sinnvollen Ort ab.
- Hinweis:* Du kannst das Programm im gleichen Menü wieder importieren, um es zu einem späteren Zeitpunkt wieder aufzurufen.

Algorithmus, Anweisung und Argument

Ein Programm besteht aus einer Folge von Anweisungen. Man spricht auch von Algorithmen: Ein Algorithmus ist eine eindeutige Handlungsvorschrift zur Lösung eines Problems, die aus endlich vielen Anweisungen besteht (s. [Wikipedia](#)).

Eine Anweisung *kann* ein oder mehrere *Argumente* haben, die zum Beispiel festlegen, welche LED geschaltet oder wie lange gewartet werden soll.

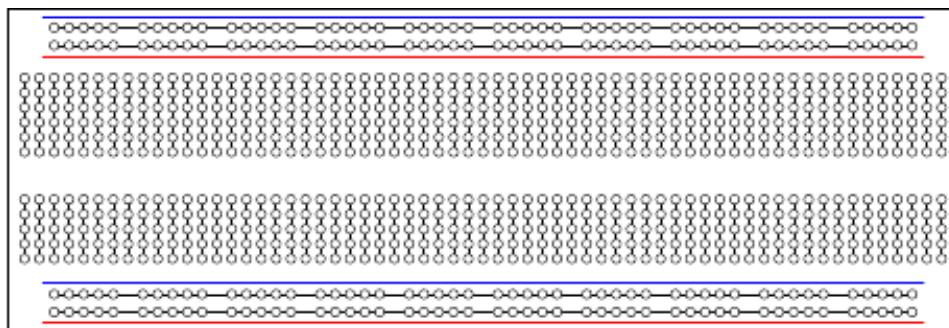


B 2.3. Anweisungen und Argumente in einem Algorithmus.

Aufgabe 2: Wir nutzen in den ersten Kapiteln sehr häufig LEDs, weil sich die Grundlagen mit ihnen einfach erarbeiten lassen, aber auch weil sie eine enorme Bedeutung in der heutigen Welt haben. Notiere dir zur Verdeutlichung eine Woche lang alle Geräte, die dir begegnen oder die dir einfallen, in denen LEDs verbaut sind.

2.4. Aufbau von Schaltungen auf der Steckplatine

In der Regel braucht man für interessante Geräte zusätzliche *Hardware* (Sensoren, Motoren, . . .), die am Arduino angeschlossen wird. Bevor diese fest verlötet werden, nutzt man normalerweise Steckverbindungen bzw. baut die Schaltung auf einem kleinen Steckbrett auf, auf dem man die Verbindungen schnell wieder lösen kann, falls nötig. Steckbretter sind aus dem Physikunterricht bekannt. Abbildung 2.4 zeigt, welche Kontakte auf dem Steckbrett miteinander verbunden sind.

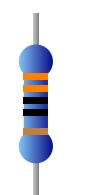


B 2.4. Die Steckverbindungen sind außen in Längsrichtung und innen in Querrichtung miteinander verbunden.

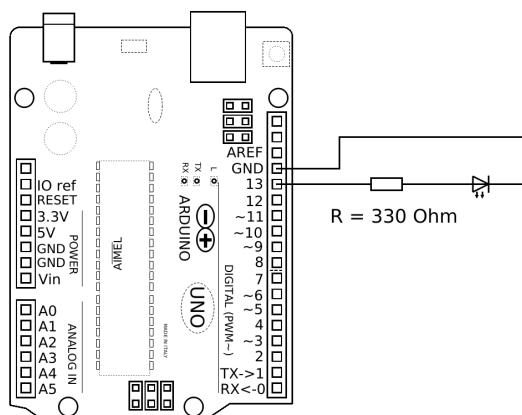
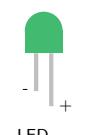
Ziel: Eine externe LED an Pin 13 soll zum Leuchten gebracht werden. Diese kann genutzt werden, um das Blinken einer Alarmanlagen-LED zu simulieren.

Hinweise:

Es kann das Programm aus dem vorherigen Abschnitt wieder verwendet werden, da die Board-LED ebenfalls mit Pin 13 verbunden ist. Allerdings gibt es noch etwas zu beachten: Wenn die LED an Pin 13 angeschaltet wird, bedeutet das, dass der Arduino an Pin 13 eine Spannung von 5V (Pluspol) gegenüber GND (Minuspol) ausgibt. Die LED verträgt jedoch nur (je nach Farbe) gut 2V. Daher ist die LED mit einem Vorwiderstand von $330\ \Omega$ verbunden.



Die LED ist ein sogenanntes gepoltes Bauteil. Das heißt, dass der lange Kontaktstift der LED an den Pluspol angeschlossen werden *muss* und der kurze an den Minuspol (GND) angeschlossen werden *muss*.



B 2.5. Schaltplan zum Anschließen einer LED mit Vorwiderstand an Pin 13.

Projekt 1: Ampel

Baue und programmiere eine Ampelschaltung! Verwende für die LEDs aussagekräftige Namen.
Denke daran, dein Programm abzuspeichern!

Für Schnelle: Erweitere die Ampel um einen Nachtmodus.

Projekt 2: Augentestgerät

Moderne Fernseher nutzen meist eine Bildwiederholungsrate von 144 Hertz; das bedeutet, es werden 144 Bilder pro Sekunde eingespielt. Finde mithilfe des Blink-Programms heraus, ab welchem Blinkintervall du kein Flackern mehr wahrnimmst und berechne, wie viele „Bilder“ pro Sekunde sich daraus ergeben. Vergleiche diesen Wert mit der Bildwiederholungsrate im Fernsehen.

Idee: Frick, Fritsch und Trick (2015): *Einführung in Mikrocontroller - Der Arduino als Steuerzentrale*, Bad Saulgau

LED

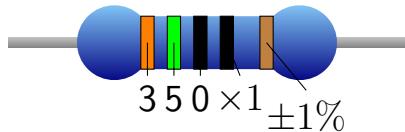
Die Bezeichnung „LED“ steht für *Licht emittierende Diode* (kurz: Leuchtdiode). Sie ist ein Halbleiterbauelement, das elektrische Energie sehr effizient in Lichtenergie umwandelt und daher heute weite Verbreitung in allen Bereichen gefunden hat, in denen Licht benötigt wird. Sie ist ein gepoltes Bauteil. Das heißt, dass der lange Kontaktstift der LED mit dem Pluspol und der kurze Kontaktstift mit dem Minuspol verbunden werden muss, damit sie leuchten kann. Damit die LED nicht durchbrennt, muss in der Regel ein Vorwiderstand angebracht werden, um sie zu betreiben.



2.5. Widerstandsringe ablesen

Leider sind die Widerstände zu klein, um ihren Wert darauf gut lesbar zu drucken. Daher werden die Widerstände mit Ringen versehen, aus deren Farbe sich die Größe des Widerstandswertes ablese lässt. Um im Folgenden jeweils den passenden Widerstand aus dem Bausatz auswählen zu können, müssen wir diesen Farbc ode lesen können.

Bei den blauen Kohleschichtwiderständen, die wir verwenden, gibt es fünf Ringe und jede Ringfarbe steht für eine Zahl. Die ersten drei Ringe bilden die ersten drei Ziffern des Widerstandswertes ab. Die darauf folgende Ringfarbe steht für die Zehnerpotenz, die mit den drei Ziffern multipliziert werden muss. Dies dient dazu, auch größere Widerstandswerte codieren zu können. Der letzte Ring wiederum soll einen etwas größeren Abstand haben und steht für die Fehlertoleranz des Widerstandswertes. In der Praxis lässt sich allerdings nicht immer gut erkennen, welcher Ring der letzte und welcher der erste ist...



Ein Beispiel: Die Ringfarben lauten orange - grün - schwarz - schwarz - braun. Anhand der folgenden Tabelle lässt sich daraus der Wert konstruieren: $3 - 5 - 0 - \cdot 1 (= 10^0) - \pm 1\%$, kurz: $350 \Omega \pm 3,5 \Omega$.

Ringfarbe	1. Ring	2. Ring	3. Ring	4. Ring (Multiplikator)	5. Ring (Toleranz)
Schwarz	0	0	0	$\times 1 / \times 10^0$	-
Braun	1	1	1	$\times 10 / \times 10^1$	1%
Rot	2	2	2	$\times 100 / \times 10^2$	2%
Orange	3	3	3	$\times 1000 / \times 10^3$	-
Gelb	4	4	4	$\times 10.000 / \times 10^4$	-
Grün	5	5	5	$\times 100.000 / \times 10^5$	-
Blau	6	6	6	$\times 1.000.000 / \times 10^6$	-
Lila	7	7	7	-	-
Grau	8	8	8	-	-
Weiß	9	9	9	-	-
Gold	-	-	-	$\times 0.1 / \times 10^{-1}$	5%
Silber	-	-	-	$\times 0.01 / \times 10^{-2}$	10%

Tabelle 2.1. Tabelle zur Codierung der Widerstandswerte durch Farbringe.

Aufgabe 3: Erste Übung

Gib die Farbcodierung für einen Widerstand mit den folgenden Werten an:

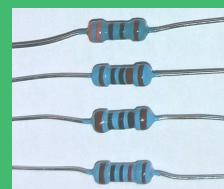
- a) $435 \Omega (\pm 2\%)$, b) $570 \text{ k}\Omega (\pm 5\%)$.

Aufgabe 4: Praxistest

- a) Die Abbildung im Infokasten unten zeigt die vier wichtigsten Widerstände, mit denen wir zu tun haben werden. Bestimme die jeweilige Größe der Widerstände.
 b) Thorsten hat die Ringe eines Widerstands abgelesen: Silber - rot - lila - lila - grün. Bestimme die Größe des Widerstands.

Zur Kontrolle: <https://www.elektronik-kompendium.de/sites/bau/1109051.htm>**Ohmsche Widerstände**

Ohmsche Widerstände dienen dazu, die Stromstärke zu begrenzen. Je größer ihr Widerstand ist, desto geringer wird die Stromstärke, wenn die Spannung gleich bleibt. Die Größe eines Widerstands lässt sich an seinen Ringen ablesen. Sie bleibt immer gleich groß.



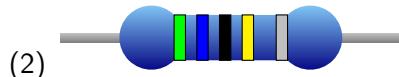
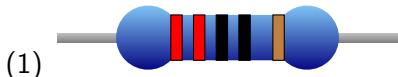
2.6. Vermischte Übungen

Aufgabe 5: Farbcodierung von Widerständen

a) Gib die Farbcodierung der folgenden Widerstandsgrößen an:

- (1) $330\Omega \pm 1\%$, (2) $10\text{k}\Omega \pm 2\%$, (3) $4,7\text{k}\Omega \pm 10\%$.

b) Gib die Größe der folgenden Widerstände an:



Hinweis: Als Hilfsmittel ist die Widerstandskarte aus den Boxen zugelassen.

Motivationsquellen

> [FUTUREMAG](#)

Kurze Dokumentation des FUTUREMAG zur Arduino-Welt

> [Arduino-Wecker](#)

Mit einem selbst gebauten Wecker, der Uhrzeit und Alarmzeit getrennt voneinander anzeigt, erfüllte ein Bastler seinen Wunsch nach mehr Komfort.

> [Aquarium-Licht](#)

Der Bastler hinter diesem Projekt wollte seinen Fischen im Aquarium ein natürliches Licht einschließlich Sonnenaufgang, Sonnenuntergang und Nacht gönnen. Auf die gleiche Art und Weise kann man natürlich auch sein Terrarium beleuchten.

> [VR-Brille](#)

Drei Schüler aus Frankreich hatten kein Geld für eine Virtual Reality Brille – aber dafür das Know How, um sich mit einem Arduino und einem Gehäuse aus dem 3D-Drucker selbst eine VR-Brille zu basteln.

3. Bausteine von Algorithmen

Mit dem Arduino und einigen einfachen Bauteilen lassen sich bereits zahlreiche Projekte umsetzen. Allerdings wird die Programmierung schnell unübersichtlich oder unnötig aufwendig, wenn man sich nicht mit algorithmischen Strukturen auskennt. Daher geht es im folgenden Kapitel um die Einführung von grundlegenden algorithmischen Bausteinen.

In diesem Kapitel lernst du...

- ... Entscheidungen zu programmieren,
- ... den Arduino mit dem Computer kommunizieren zu lassen,
- ... zwischen unterschiedlichen Datentypen zu unterscheiden,
- ... Entscheidungen anhand mehrerer Kriterien zu treffen,
- ... wie man Programme zur Planung oder zum Vergleich auf Papier einfach darstellen kann
- ... Variablen zu benutzen,
- ... zufällige Ereignisse zu programmieren,
- ... mit Schleifen effizient zu programmieren,
- ... systematisch nach Fehlern im Programm zu suchen.

Projekte in diesem Kapitel:

> Fußgängerampel	12	> Bombe bauen	20
> Juke-Box	13	> Reaktionszeitmesser	20
> Straßenlampe	15	> Reaktionsspiel	21
> Carport-Lampe	16	> Alarmanlage mit Lichtschranke ..	34
> Auto-Blinker	22		

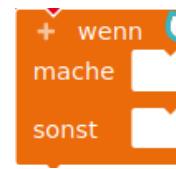
3.1. Entscheidungen programmieren

Schaltungen und Programme werden erst dann richtig interessant, wenn sie auf ihre Umwelt reagieren. Im einfachsten Fall lässt sich dazu ein Taster einbauen, mit dem von außen sich entscheiden lässt, wie das Programm weiterlaufen soll. Dementsprechend müssen im Programm *Fallunterscheidungen* eingebaut werden.

Ziel: Der Arduino soll auf Eingaben aus seiner Umwelt reagieren.

Projekt 1: Fußgängerampel

Baue und programmiere eine Fußgängerampel! Nutze dazu die Informationen aus dem Info-Kasten zum Taster.



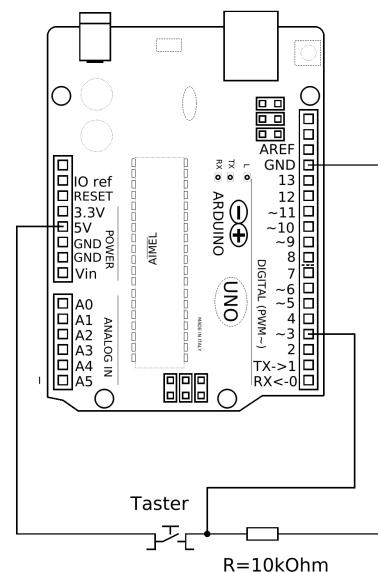
Taster

Ein Taster ist wie ein Schalter, kann also geschlossen sein (Strom fließt) oder offen sein (Strom fließt nicht). Im Gegensatz zum Schalter springt ein Taster aber automatisch wieder in den offenen Zustand zurück, wenn er losgelassen wird.



In dem rechts abgebildeten Schaltplan ist dargestellt, wie man einen Taster am Arduino so anschließt, dass man seinen Zustand im digitalen Pin 3 auslesen kann. Unten ist die zugehörige Taster-Konfiguration abgebildet.

Taste **T**
pin **3**
VCC **5V**

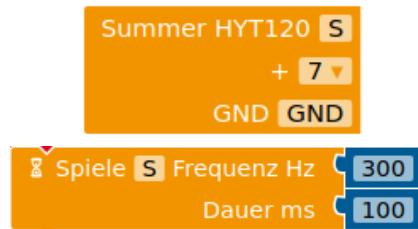


Projekt 2: Juke-Box

Baue und programmiere eine Juke-Box!

Die Juke-Box soll zwei verschiedene, kurze Melodien *anspielen* können. Dazu werden zwei Taster auf die beschriebene Art an zwei digitale Pins des Arduino angeschlossen. Schließe zudem an einen Digitalpin einen Piezo-Summer an (siehe unten).

Idee: Frick, Fritsch und Trick (2015): *Einführung in Mikrocontroller - Der Arduino als Steuerzentrale*, Schülerforschungszentrum Bad Saulgau



Hinweise:

Zwei mögliche Beispiele von Melodien mit Link zu den Noten:

- „Alle meine Entchen“:

https://www.lieder-archiv.de/alle_meine_entchen-notenblatt_100055.html,

- „Oh Tannenbaum“:

https://www.lieder-archiv.de/o_tannenbaum-notenblatt_200078.html.

Frequenzen in Hertz zu den Noten:

c^1	cis^1/des^1	d^1	dis^1/es^1	e^1	f^1	fis^1/ges^1	g^1	gis^1/as^1	a^1	ais^1/b^1	h^1
262	277	294	311	330	349	370	392	415	440	466	494

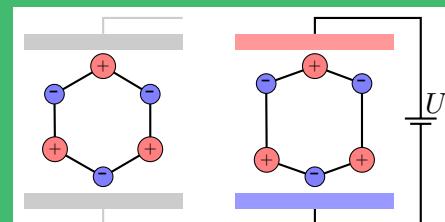
Piezo-Summer

Mit einem Piezo-Summer lassen sich Töne erzeugen, wenn man eine Spannung anschließt. Das lange Bein muss dabei an den Pluspol (Pin) angeschlossen werden; das kurze an den Minuspol bzw. GND. Ein Vorwiderstand ist dabei nicht notwendig, hilft aber die Lautstärke zu reduzieren.



Funktionsweise:

In einem Piezo-Summer befindet sich ein Kristall mit unterschiedlichen Ladungsschwerpunkten, der von einem Kondensator umgeben ist. Wenn von außen an den Kristall eine Spannung angelegt wird, dann verformt sich die Kristallstruktur durch die Anziehung zwischen den Ladungsschwerpunkten und den Kondensatorplatten (*inverser piezo-elektrischer Effekt*). Wenn keine Spannung anliegt, verformt sich der Kristall zurück. Durch diese Verformungen entstehen Druckwellen in der Luft, die wir als Ton wahrnehmen können.



3.2. Kommunikation mit dem Arduino: Der serielle Monitor

Bisher hatte die Kommunikation mit dem Arduino stets nur eine Richtung: Vom Computer zum Arduino. Das reicht nicht mehr, wenn eine Messung vorgenommen und deren Ergebnis zurück gemeldet werden soll. Die einfachste Möglichkeit, um dies zu realisieren, ist der serielle Monitor. Dieser soll im Folgenden genutzt werden, um eine Straßenlampe zu konfigurieren, die leuchtet, wenn es dunkel wird.

Frage: Wie kann der Arduino mit dem Computer kommunizieren?

Aufgabe 1: Test des seriellen Monitors

- Implementiere ein Programm, das in jeder Sekunde „Moin!“ an den seriellen Monitor sendet und übertrage es auf den Arduino.

Zeige auf Serial Monitor  “ Hallo ”

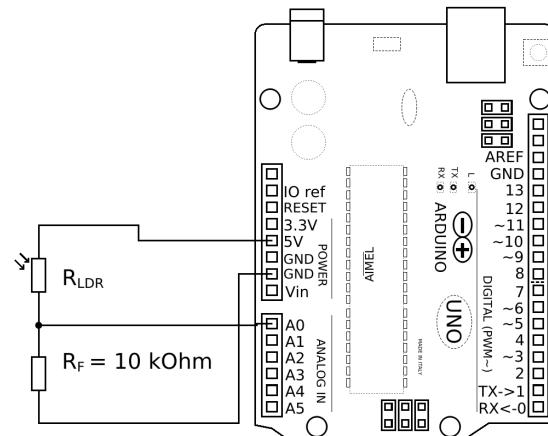
- Öffne den seriellen Monitor im Open Roberta Connector mit einer Baudrate von 9600 und kontrolliere dein Programm.

Ein LDR ist ein Widerstand, dessen Größe von der Lichtstärke abhängt, die auf ihn trifft (siehe unten). Um ihn auslesen zu können, muss er in einem sogenannten Spannungsteiler mit einem Festwiderstand von $R_1 = 10 \text{ k}\Omega$ an den Arduino angeschlossen werden (siehe rechts). Der zugehörige Konfigurationsblock ist unten abgebildet.



Lichtsensor LDR
output A0
VCC 5V

B 3.1. Ein LDR.



B 3.2. Konfiguration des LDR.

Aufgabe 2: Erste Experimente mit dem LDR

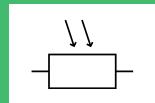
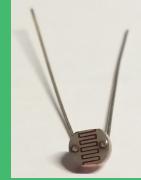
- Baue die oben abgebildete Schaltung zum Auslesen eines LDR am Arduino auf und lasse dir die Lichtstärke in % auf dem seriellen Monitor anzeigen.
- Die Veränderung der Lichtstärke in % verläuft genau umgekehrt zur Veränderung des Widerstands des LDR. Beschreibe, wie sich der Widerstand des LDR verhält, wenn es dunkel bzw. wenn es hell wird.

Projekt 3: Straßenlampe

Baue eine Straßenlampe, deren Licht (Vorwiderstand!) angeht, wenn es dunkel wird, und ausgeht, wenn es hell wird.

Fotowiderstand

Ein **Fotowiderstand**, kurz: **LDR** (*engl. light dependent resistor*), ist ein lichtabhängiger Widerstand. Wenn es dunkel wird, wird der elektrische Widerstand des LDR größer; wenn es hell wird, wird der elektrische Widerstand des LDR kleiner.



3.2.1. Reflexion: Datentypen

Folie

öffnen

Aufgabe 3:

- a) In den bisherigen Anweisungen tauchten verschiedene Typen von Argumenten auf. Die Argumente sind unten noch einmal abgebildet. Gruppieren und charakterisiere sie.

500

Taste T gedrückt?

gib Licht % Lichtsensor LDR

wahr ▾

“ Hallo ”

- b) Begründe: Das Argument des `wenn`-Blocks muss hellblau sein.

Datentypen

Alle Programmiersprachen unterscheiden verschiedene Datentypen, die unterschiedlich codiert sind. Die wichtigsten Datentypen in Nepo sind...

- wahr logische Werte / Wahrheitswerte:
Nur zwei Werte möglich, nämlich wahr oder falsch,
- 500 Zahlenwerte:
Sowohl ganze Zahlen als auch Dezimalzahlen (mit Punkt als Komma),
- “ Hallo ” Zeichenketten:
Aneinanderreihung von Zeichen.

3.3. Entscheidungen mit mehreren Kriterien treffen

Bisher waren die zu treffenden Entscheidungen immer nur von einem Kriterium abhängig. Das sind jedoch Ausnahmen. Nun geht es darum, wie man mehrere Kriterien miteinander kombinieren kann.

Frage: Wie lassen sich Entscheidungen mit mehreren Kriterien programmieren?

Projekt 4: Carport-Lampe

Baue und programmiere eine Carport-Lampe, die für einige Zeit leuchtet, wenn sie eine Bewegung registriert **und** es draußen gerade dunkel ist. In allen anderen Fällen bleibt die Lampe dunkel. Experimentiere mit den Drehreglern, um die Empfindlichkeit und Dauer des Signals richtig einzustellen.

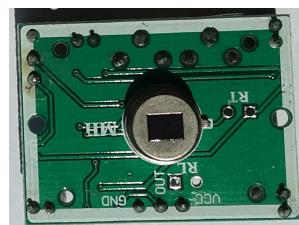
Information zu Bewegungsmeldern: Bewegungsmelder verfügen über drei Pins, deren Beschriftung man lesen kann, wenn man die Kunststofflinse vorsichtig abzieht (*Vorsicht: Nach Abziehen der Linse nicht den Sensor berühren!*). Vcc und GND dienen der Stromversorgung der elektronischen Komponenten und müssen mit 5 V und GND am Arduino verbunden werden.

Der mittlere OUT-Pin ist der Signal-Pin: Wenn eine Bewegung registriert wurde, wird der Wert wahr zurückgegeben, ansonsten falsch. Zum Einlesen des Signals wird dieser Pin mit einem Digitalpin des Arduino verbunden.

Hinter befinden sich zwei Drehregler („Potentiometer“), mit denen sich die Dauer des Bewegungssignals (links) und die Empfindlichkeit (rechts) einstellen lassen. Zusätzlich befindet sich auf der rechten Seite ein sogenannter Jumper, mit dem auf einfache Weise eine Steckverbindung zwischen benachbarten Pins hergestellt werden kann. Wenn sich der Jumper ganz außen befindet, dann bleibt das Bewegungssignal nach dem Erkennen einer Bewegung eine Weile aktiv und wird dann auf jeden Fall deaktiviert. Eine neue Bewegung kann erst nach einer gewissen Zeit wieder registriert werden. Wenn der Jumper hingegen leicht nach innen versetzt ist, bleibt das Bewegungssignal so lange erhalten, wie eine Bewegung erkannt wird (siehe Funduino).



B 3.3. Bewegungsmelder mit Linse.



B 3.4. Pinbelegung.



B 3.5. Drehregler für Signaldauer (links) und Empfindlichkeit (rechts).

Aufgabe 4: Wahrheitswerttabellen

- a) Experimentiere mit der Carport-Lampe, um die folgende Wahrheitswert-Tabelle für die Operation UND auszufüllen.

Arbeitsblatt öffnen

A: Bewegung registriert.

B: Es ist dunkel.

A	B	A UND B
wahr	wahr	

- b) Ändere das UND zu einem ODER und experimentiere wieder mit der Carport-Lampe, um die folgende Wahrheitswert-Tabelle für die Operation ODER auszufüllen.

A: Bewegung registriert.

B: Es ist dunkel.

A	B	A ODER B
wahr	wahr	

- c) Ergänze die Wahrheitswerttabelle für die logische Operation NICHT.

A	NICHT A
wahr	
falsch	

Logische Operationen und Wahrheitswerttabellen

Logische Operationen dienen zum Verknüpfen von Wahrheitswerten - ganz so wie Rechenoperationen zum Verknüpfen von Zahlen dienen. Wir betrachten die logischen Operationen UND (AND), ODER (OR) sowie NICHT (NOT). Das Ergebnis dieser Operationen lässt sich anhand von Wahrheitswerttabellen übersichtlich festhalten. Darin wird festgehalten, ob zwei Aussagen bzw. Bedingungen A und B wahr (w) oder falsch (f) sind. In der rechten Spalte steht dann, ob die logische Operation wahr (w) oder falsch (f) ergibt.

A	B	A UND B
w	w	w
w	f	f
f	w	f
f	f	f

A	B	A ODER B
w	w	w
w	f	w
f	w	w
f	f	f

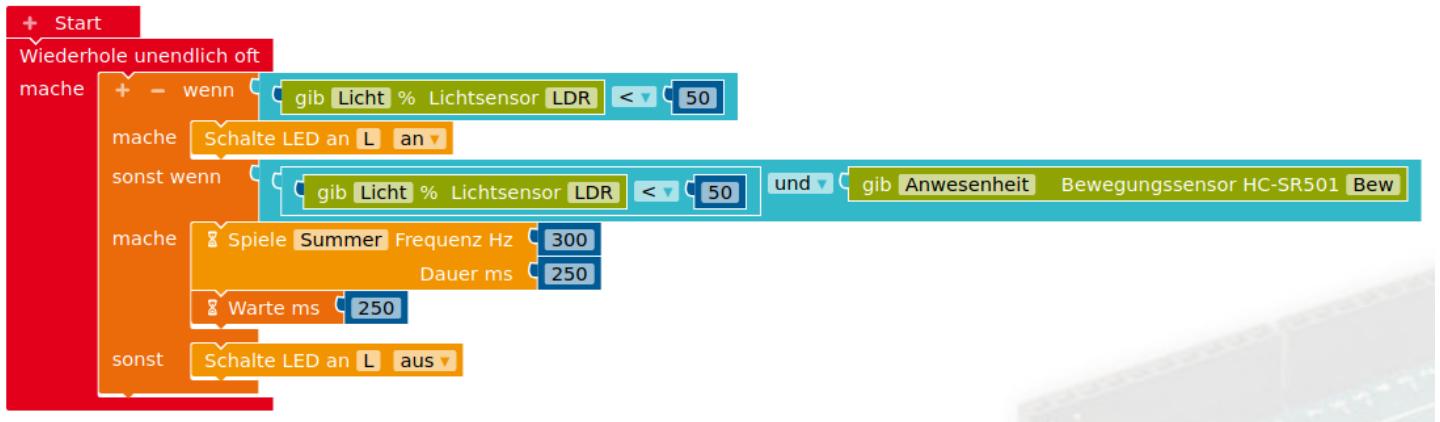
A	NICHT A
w	f
f	w

Achtung: Die ODER-Operation ergibt auch dann „wahr“, wenn beide Aussagen wahr sind. Das aus dem Alltag bekannte „ENTWEDER-ODER“ (XOR) ist eine weitere logische Operation, die „falsch“ ergibt, wenn beide Aussagen wahr sind. Diese Operation ist aber nicht in Nepo enthalten.

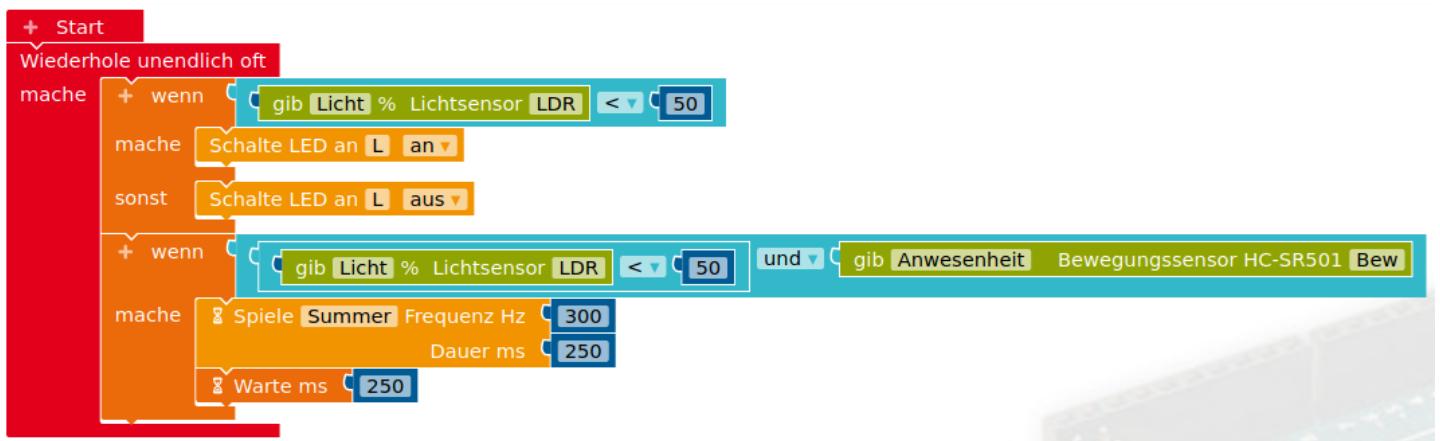
Aufgabe 5: Verschachtelte Entscheidungen?!

Leo und Lara möchten ihre Carport-Lampe so umprogrammieren, dass sie nachts immer leuchtet und tagsüber nicht. Zusätzlich soll ein Alarm ertönen, wenn es Nacht ist und eine Bewegung registriert wird.

Unten sind ihre Programme abgebildet. Entscheide jeweils (begründet!), ob das Programm für das geforderte Verhalten geeignet ist. Mache gegebenenfalls Verbesserungsvorschläge.



B 3.6. Leos Programm.



B 3.7. Laras Programm.

3.4. Programme mit Variablen und Schleifen effizient steuern

Aufgabe 6: Jana und Jonas haben jeweils LEDs an den Arduino angeschlossen und steuern diese mit den unten abgebildeten Programmen. Vergleiche die beiden Programme im Hinblick auf ihre Wirkung und die Art der Programmierung. Welches gefällt dir besser?

Zusatzüberlegung: Wie viel muss man ändern, wenn man die Leuchtdauer ändern will?

Folie
öffnen



B 3.8. Janas Programm zum Steuern der LEDs.

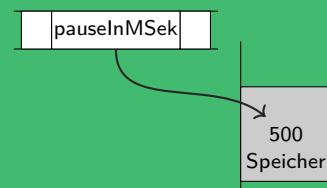


B 3.9. Jonas Programm zum Steuern der LEDs.

Variablen

Eine Variable kann man sich als Koffer vorstellen, der einen Namen bekommt und in dem man einen festgelegten Datentyp speichert. Jedes Mal, wenn der Name des Koffers aufgerufen wird, wird der abgespeicherte Wert hervorgeholt und an die Stelle des Namens gesetzt. Intern wird der Variablenname als Verweis auf einen bestimmten Speicherplatz genutzt, in dem der Wert der Variable abgelegt ist.

Für den Namen hat sich der `lowerCamelCase` etabliert: Der erste Buchstabe ist klein; wenn weitere Wörter folgen, fangen diese mit einem großen Buchstaben an. Leerzeichen sind nicht erlaubt.



3.4.1. Zufällige Ereignisse und Wiederholungen programmieren

Viele Dinge werden interessanter, wenn sie sich nicht immer auf die genau gleiche Art wiederholen. Für diese Fälle kann man im Programm den blauen Block für Zufallszahlen verwenden (Expertenblöcke aktivieren!), der jedes Mal eine neue Zufallszahl erzeugt, wenn er aufgerufen wird.



ganzzahliger Zufallswert zwischen bis

Ein einfaches Beispiel ist die „Bombe“, die man bei dem Spiel „Tick Tack Bumm“ startet und die man so lange herum geben muss, bis sie explodiert. Dabei ist die Dauer des Tickens ein zufälliger Wert zwischen ca. 5 s und 20 s.



Bomben-
Beispielvideo

Projekt 5: Bombe bauen

Baue und programmiere eine „Bombe“, die für eine zufällige Dauer zwischen 5 s und 20 s tickt und dann explodiert. Die Bombe wird über einen Taster aktiviert.

Wiederhole mal
mache

Zusatz: Welchen Unterschied macht es, wenn man die ausgewürfelte Zufallszahl in einer Variable speichert?

Aufgabe 7: Exkurs: Zufallszahlen von Mikrocontrollern/Mikroprozessoren

Übertrage das rechts abgebildete Programm auf den Arduino und betrachte die so erzeugten Zufallszahlen.

+ Start
Wiederhole unendlich oft
mache Zeige auf Serial Monitor ganzzahliger Zufallswert zwischen bis
Warte ms

Drücke dann auf den Reset-

Taster am Arduino und betrachte die nun erzeugten Zufallszahlen. Wiederhole den Vorgang einige Male und beschreibe Auffälligkeiten.

Projekt 6: Reaktionszeitmesser

Baue und programmiere einen Reaktionszeitmesser.

Der Reaktionszeitmesser soll zunächst warten, bis ein Taster gedrückt wurde, der besagt, dass es losgehen kann. Dann wird eine LED angeschaltet (Vorwiderstand!) und nach einer zufälligen Zeit wieder ausgeschaltet. Nun beginnt die Zeitmessung. Die Stoppuhr läuft solange, bis der Taster gedrückt wurde. Die gemessene Zeit wird dann über den seriellen Monitor ausgegeben und es wird erneut gewartet, bis der Anwender bestätigt, dass es losgehen kann.

Setze Zeitgeber zurück
gib Wert ms Zeitgeber

Miss mindestens zehn Mal deine Reaktionszeit und bestimme den Mittelwert. Bist du besser als dein Partner?

3.4.2. Wiederholungen mit Bedingungen steuern

In vielen Fällen geht es bei Schleifen nicht um eine genau oder zufällig bestimmte Anzahl von Wiederholungen, sondern darum, einen Vorgang zu wiederholen, bis eine Bedingung wahr ergibt, bzw. solange, wie eine Bedingung wahr ergibt. Die Bedingung, die wahr oder falsch ergibt, kann auch Sensorwerte beinhalten. Dies wird auch im folgenden Spiel genutzt.

Projekt 7: Konfigurierbares Reaktionsspiel

Baue und programmiere ein konfigurierbares Reaktionsspiel!

Dazu werden drei Taster (mit Widerstand!) am Arduino angeschlossen. Nach einer zufälligen Zeit wird auf dem seriellen Monitor angezeigt, welcher (zufällig ausgewürfelte) Taster gedrückt werden soll. Geschieht dies innerhalb einer vorgegebenen maximalen Reaktionszeit, hat man gewonnen, andernfalls verloren.

Am Anfang des Spiels soll diese maximale Reaktionszeit konfiguriert werden können. Das heißt, man kann die max. Reaktionszeit mit dem linken Taster verringern und mit dem rechten Taster vergrößern. Erst wenn der mittlere Taster gedrückt wird, startet das Spiel.

Für einen besseren Zugang zu diesem komplexen Spiel kannst du folgende Vorlage öffnen, mittels „Speichern unter“ als Reaktionsspiel.xml auf dem Computer speichern und die Datei im Open Roberta Lab importieren: [reaktionsspiel-vorlage.xml](#).

Serial Monitor
Willkommen beim Reaktionsspiel!
Stelle zuerst die max. Reaktionszeit ein.
Max. Reaktionszeit in Millisekunden:
500
Max. Reaktionszeit in Millisekunden:
490
Spiel startet...
Drücke Taster:
2
Taster 2 gedrückt
Sehr gut!

Mögliche Erweiterungen:

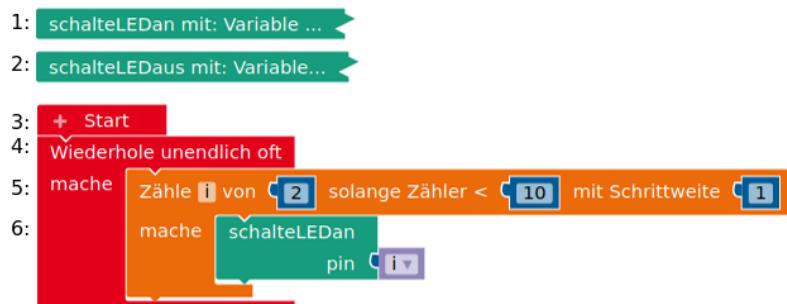
- zusätzliche Ausgabe der Reaktionszeit,
- Ober- und Untergrenze für die einstellbare maximale Reaktionszeit, damit das Spiel nicht unmöglich, aber auch nicht zu langweilig wird,
- nach Spielstart folgen mehrere Spiele hintereinander und es wird mitgezählt, wie oft gewonnen wird,
- je nach Reaktionszeit bekommt man mehr oder weniger Punkte,
- ...

3.4.3. Zählschleifen programmieren

Lauflichter findet man inzwischen überall in unserer Welt: An den Rändern von Landebahnen an Flughäfen, an Spieleanutomaten, aufdringlichen Werbeschildern, als Blinker von modernen Autos und vieles mehr. Wenn man diese programmieren will, eignen sich dazu am besten Zählschleifen.

Aufgabe 8: Zählschleifen verstehen

Das rechts abgebildete Programm enthält zwei selbst definierte Blöcke, mit denen sich eine LED an einem beliebigen Pin zwischen 2 und 13 anstellen bzw. ausschalten lässt. In der Endlosschleife wird dann eine Zählschleife genutzt.



Stelle eine Vermutung an, was die Zählschleife bewirkt.

Überprüfe deine Vermutung mit Hilfe einer *Trace-Tabelle* (siehe unten).

Trace-Tabellen

Trace-Tabellen stellen den Wert von Variablen beim Durchlaufen des Programms dar. Auf diese Art und Weise kann man sich zum Beispiel genau veranschaulichen, wann Schleifen abgebrochen werden.

Zeile	i
...	...
5	2
6	2
5	3
6	3



Blinker-
Beispielvideo

Projekt 8: Auto-Blinker

Programmiere ein Lauflicht so, wie es auch als Blinker in modernen Autos genutzt wird. Nutze zunächst nur 5 LEDs (mit Vorwiderstand!).

Hinweis: Du kannst das folgende Programm als Vorlage nutzen, damit du auch über die selbst definierten Blöcke zum Anstellen bzw. Ausstellen einer LED an einem beliebigen Pin zwischen 2 und 13 verfügst. Öffne das Programm, speichere es als `blinker.xml` und importiere es in Open Roberta Lab: [blinker-vorlage.xml](#).

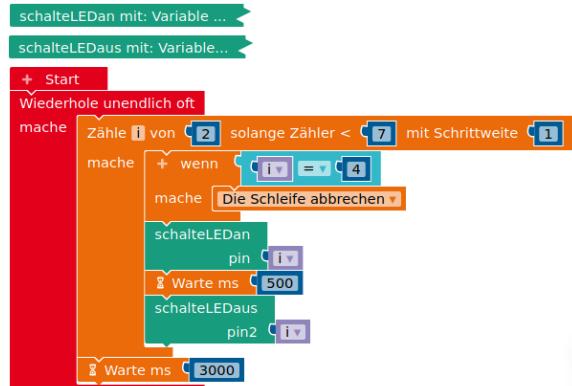
Aufgabe 9: La - o - La

Programmiere ein Lauflicht, das hin- und zurückläuft.

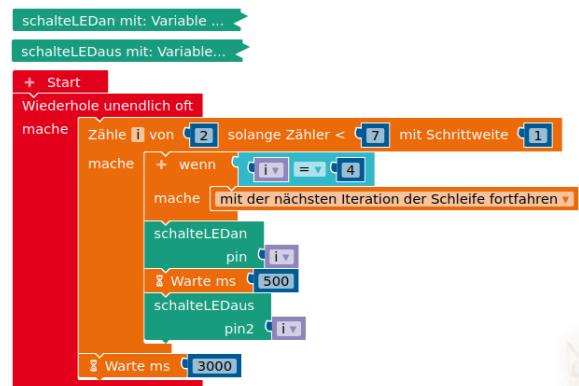
Aufgabe 10: Steuerung von Schleifen

Unten siehst du zwei Programme, für die jeweils fünf LEDs mit Vorwiderstand an Pin 2 bis 6 angeschlossen wurden. Baue das nach, importiere die Programme in Open Roberta Lab und übertrage sie auf den Arduino.

Beschreibe die Wirkung von **Die Schleife abbrechen** und mit der nächsten Iteration der Schleife fortfahren.



schleife-abbrechen.xml



schleife-fortfahren.xml

In Nepo wie in anderen Programmiersprachen gibt es verschiedene Arten von Schleifen. Bisher wurde in Nepo die einfache Zählschleife `wiederhole x mal`, die bedingungsgesteuerte Wiederholschleife `wiederhole bis / solange` und die Zählschleife mit Zählervariable genutzt. Tatsächlich lässt sich das gleiche Verhalten aber mit allen drei Schleifenvarianten erreichen.

Aufgabe 11: Vergleich von Schleifen

Betrachte noch einmal das Programm `schleife-abbrechen.xml` (siehe oben). Implementiere das gleiche Verhalten mit ...

- ... einer `wiederhole x mal` Schleife,
- ... einer `wiederhole bis` Schleife,
- ... einer `wiederhole solange` Schleife.

Erkläre, welche Schleifenvariante sich als „Grundschleife“ eignet, die die anderen Varianten *immer* ersetzen kann.

Sichtbarkeit: Lokale und globale Variablen

Ein Unterschied zwischen den Schleifenimplementierungen bleibt bestehen: Die Zählvariable `i` wird bei einer Zählschleife als *lokale Variable* angelegt, das heißt, man kann die Zählvariable nur *innerhalb der Schleife* nutzen. Dafür benötigt sie auch nur innerhalb der Schleife Speicherplatz.

Im Gegensatz dazu sind die unter `Start` angelegten Variablen überall im Programm bzw. global verfügbar und heißen deshalb *globale Variablen*. Für diese Variablen muss während der ganzen Zeit Speicherplatz bereitgehalten werden, auch wenn sie vielleicht nur an einer Stelle wirklich benötigt werden.

Schleifen

Bei der Programmierung werden häufig Schleifen genutzt, die die Anweisungen in ihrem Rumpf (oder Körper) solange wiederholen, bis eine gewisse Abbruchbedingung eintritt.

- `wiederhole x mal:` Einfache *Zählschleife*, die die Anweisungen im Rumpf für eine festgelegte Anzahl wiederholt.
- `wiederhole, bis:` Bedingungsgesteuerte Schleife, die die Anweisungen im Rumpf wiederholt, *bis* die Bedingung wahr ergibt.
- `wiederhole, solange:` Bedingungsgesteuerte Schleife, die die Anweisungen im Rumpf wiederholt, *solange* die Bedingung wahr ergibt.
- `zähle i von ... solange Zähler ... mit Schrittweite ...:` Zählergesteuerte Schleife, die die Anweisungen im Rumpf wiederholt, solange der Zähler kleiner als eine vorgegebene Zahl ist und die Zählervariable nach jedem Durchlauf um eine angegebene Zahl erhöht.

Die Überprüfung, ob die Bedingung wahr ist, erfolgt hier *vor* der Ausführung der Anweisungen im Rumpf. Daher nennt man die Schleifen auch *kopfgesteuert*.

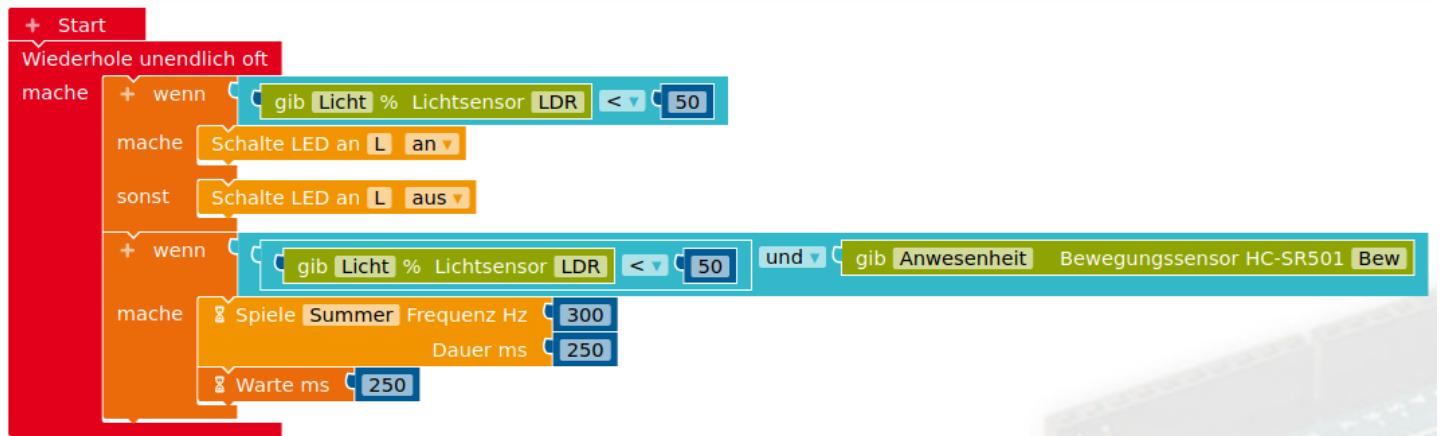
3.5. Programme mit Struktogrammen dokumentieren

Wenn wir uns über Programme austauschen, dann haben wir nicht immer den Computer zur Hand. In solchen Momenten wäre es viel zu aufwendig, die bunten Blöcke von Nepo zu malen. Außerdem könnte es sein, dass jemand anderes das Programm nicht mit Blöcken, sondern mit Text in der Programmiersprache C++ aufschreiben will, also so wie der Quellcode aussieht.

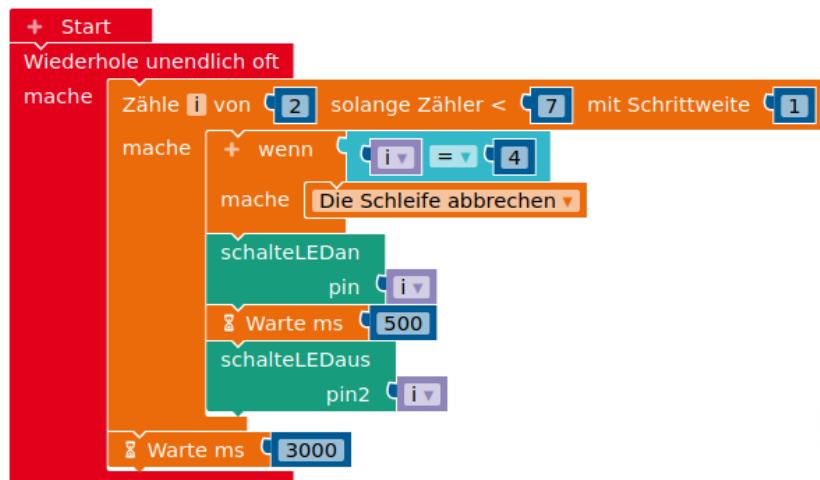
Frage: Wie kann man Programme übersichtlich zu Papier bringen?

Man nutzt zur Darstellung des Ablaufs eines Computerprogramms sogenannte **Struktogramme** (vgl. Tab. 3.1), die in den 1970er Jahren von Isaac Nassi und Ben Shneidermann entwickelt wurden. Struktogramme sollen ein Computerprogramm möglichst einfach und ohne programmiersprachen-spezifische Befehlssyntax abbilden. Auf diese Art und Weise lassen sich Programme auch einfach planen, bevor man sich damit beschäftigt, wie die Schritte im Einzelnen zu codieren sind.

Aufgabe 12: Stelle die unten abgebildeten Programme jeweils mithilfe eines Struktogramms dar.



B 3.10. Programm A.



B 3.11. Programm B.

Darstellung von Programmen in Struktogrammen

Lineare Struktur

Jede Anweisung wird in einen rechteckigen Block geschrieben.

Anweisung 1

Anweisung 2

Wiederholung / Schleifen

Zählergesteuerte Schleife

Die Anzahl der Schleifendurchläufe wird durch eine Zählvariable festgelegt. Im Schleifenkopf werden der Startwert der Zählvariablen, der Endwert der Zählvariablen und die Veränderung der Zählvariablen, z. B. Schrittweite 1, angegeben.

zähle Variable von Startwert bis Endwert, Schrittweite

Anweisung 1

Kopfgesteuerte Schleife

Wiederholungsschleife mit vorausgehender Prüfung der Bedingung. Der Schleifenkörper wird so lange wiederholt, *wie* oder *bis* die Bedingung wahr ist (bei uns nur der letzte Fall verfügbar).

solange, wie/bis Bedingung wahr

Anweisung 1

Fußgesteuerte Schleife

Wiederholungsschleife mit nachfolgender Prüfung der Bedingung. Der Schleifenkörper wird so lange wiederholt, *wie* oder *bis* die Bedingung wahr ist (in mBlock nicht verfügbar).

Anweisung 1

solange, wie/bis Bedingung wahr

Verzweigung

Einfache Verzweigung

Die Anweisung 1 (und ggf. weitere) wird nur ausgeführt, falls die Bedingung wahr ist. Andernfalls wird nichts gemacht.



Alternative Verzweigung

Falls die Bedingung wahr ist, wird Anweisung 1 (und ggf. weitere) ausgeführt, sonst wird Anweisung 2 (und ggf. weitere) ausgeführt.



Verschachtelte Verzweigung

Falls Bedingung 1 wahr ist, folgt eine weitere Bedingung 2.

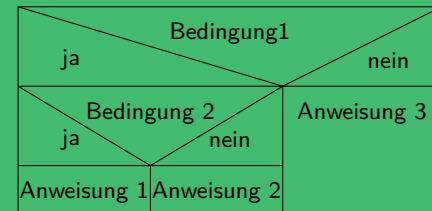


Tabelle 3.1. Tabelle zur Darstellung eines Programms als Struktogramm nach Nassi und Schneidermann.

3.6. Eigene Funktionen definieren

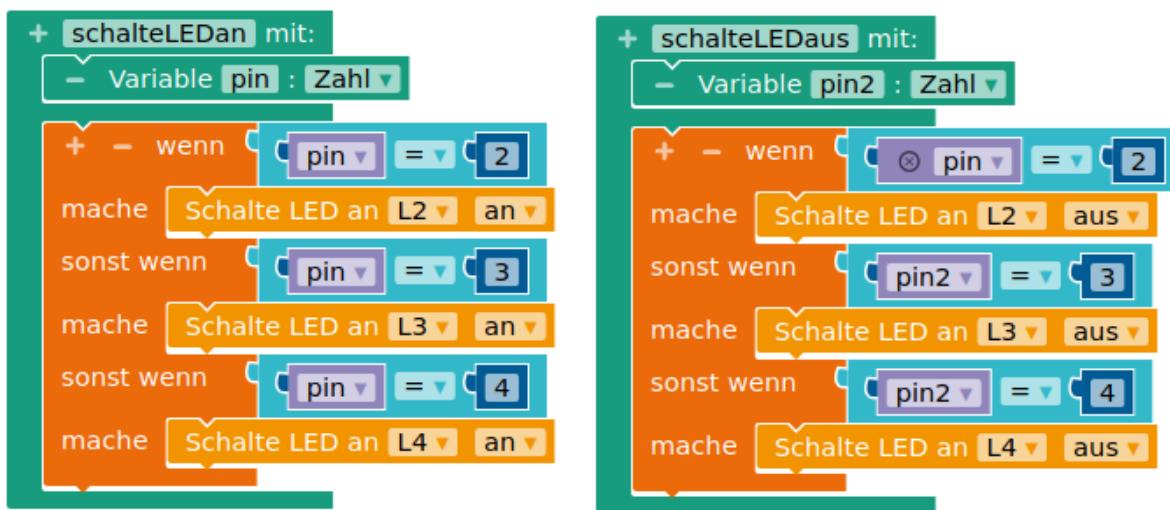
In den letzten Abschnitten ist bereits deutlich geworden, dass es manchmal praktisch sein kann, sich eigene Blöcke zu definieren. In der Programmiersprache C++, in der der Quellcode für den Arduino generiert wird, werden diese „Funktion“ genannt.

Frage: Wie kann man in Nepo Funktionen implementieren?

Aufgabe 13: Bekannte Funktionen aufgeschlüsselt

In der Abbildung unten ist zu sehen, wie man Funktionen implementiert, mit denen sich LEDs an Pin 2 bis 4 über ihre Pin-Nummer anschalten und ausschalten lassen. Beschreibe, wie die Funktion zum Anschalten aufgebaut ist und genutzt wird.

In der Funktion zum Ausschalten kann die Variable pin nicht genutzt werden (sichtbar durch \otimes). Was bedeutet dies für die Variablen, die in Funktionen angelegt werden?



Aufgabe 14: *Block zum Blinken*

Implementiere einen Block `blinke` mit den Argumenten `anzahl` und `pauseInMSek`, die die Board-LED für die angegebene Anzahl und mit der angegebenen Pause in Millisekunden zum Blinken bringt. Überprüfe deinen Block.

Was passiert, wenn eine Kommazahl übergeben wird?

Aufgabe 15: *Lesbarkeit und Rückgabewerte*

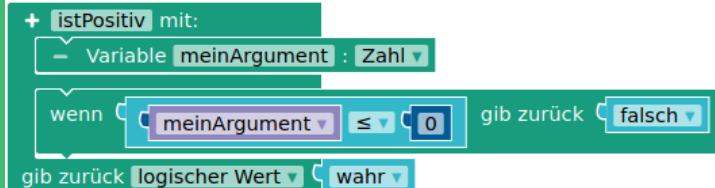
Die Logik für die Straßenlaterne (vgl. S. 15) lautete: Wenn es dunkel ist, schalte die Lampe an, sonst schalte die Lampe aus.

Mit einem eigenen Block lässt sich diese Logik direkt im Programm umsetzen, sodass es noch besser lesbar wird. Implementiere einen Block `istDunkel`, der basierend auf den Werten eines angeschlossenen LDR an A0 einen Wahrheitswert zurückgibt.

Tipp: Nutze ggf. die  Hilfefunktion auf der rechten Seite, um dich mit den abgebildeten Blöcken vertraut zu machen.

**Funktionen**

Funktionen fassen mehrere Anweisungen zusammen und können als eigene Anweisung im Algorithmus genutzt werden, um ihn lesbarer und modularer zu machen, wenn an einigen Stellen die gleichen Anweisungen immer wieder benötigt werden. Für den Namen der Funktion gilt wiederum die `lowerCamelCase`-Konvention.

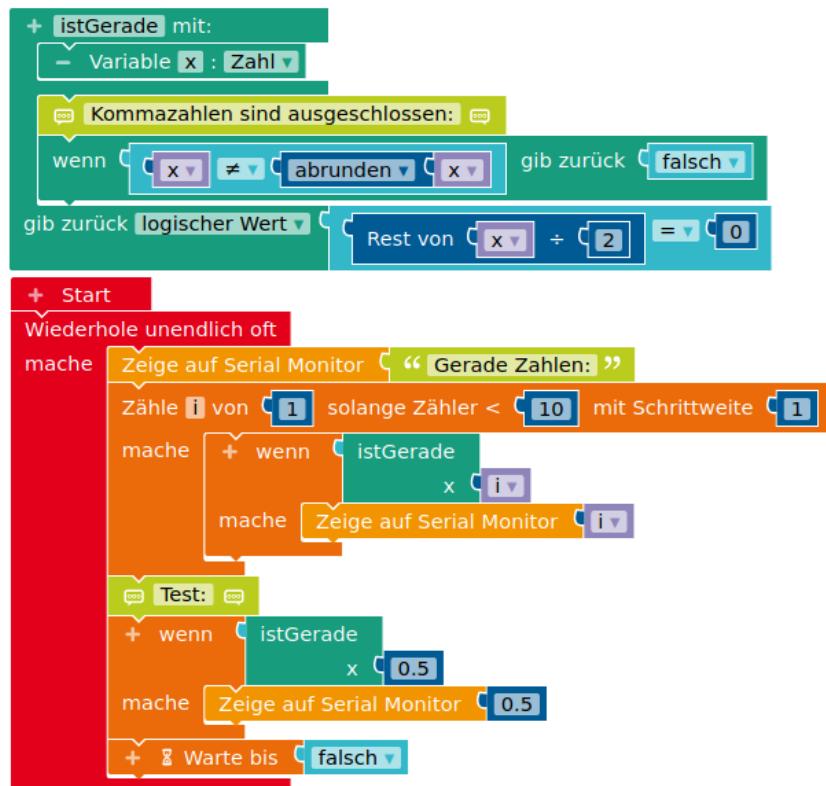


Funktionen können mehrere Argumente von unterschiedlichem Typ haben, die die Art der Ausführung variieren können. Die Variablen, in denen diese Argumente gespeichert werden, sind lokale Variablen und daher nur innerhalb der Funktion verfügbar. Außerdem können Funktionen einen Wert zurückgeben, der für den „Hauptalgorithmus“ genutzt werden kann. Die Rückgabe eines Wertes muss nicht am Ende der Funktion erfolgen. Wenn bereits vor dem Ende ein Wert zurückgegeben wird, wird der Rest der Funktion nicht mehr ausgeführt.

Die Bezeichnung „Computer“, zu deutsch: „Rechner“, besagt schon, dass man die Entwicklung von Mikrocontrollern und Mikroprozessoren immer auch dazu diente, Rechnungen zu automatisieren, die ein Rechner wesentlich schneller, präziser und fehlerfreier vornehmen kann als ein Mensch. Die Grundrechenarten sind schon als Blöcke in Nepo implementiert. Damit lassen sich auch auf schnelle Art weitere Berechnungen anstellen, für die ein Mensch mehrere Minuten oder sogar Stunden bräuchte.

Aufgabe 16: Teilbarkeit durch 2

Unten ist ein Programm abgebildet, mit dem die Teilbarkeit durch 2 überprüft wird. Erläutere seinen Ablauf.

**Aufgabe 17:** Primzahlen

Primzahlen sind natürliche Zahlen, die nur durch 1 und sich selbst teilbar sind. Die kleinste mögliche Primzahl ist 2.

Primzahlen haben die Menschen seit jeher fasziniert, weil man bis heute keine Formel gefunden hat, um Primzahlen zu berechnen. Im Wesentlichen ist man darauf angewiesen, alle möglichen Teiler auszuprobieren und auf diese Art herauszufinden, ob eine Zahl eine Primzahl ist oder nicht. Unter anderem aufgrund dieser „Sperrigkeit“ eignen sich Primzahlen gut zur Verschlüsselung.

Implementiere einen Block `istPrimzahl` mit einer Zahl x als Argument und einem Wahrheitswert als Rückgabewert. Sorge dafür, dass Kommazahlen sofort als falsch erkannt werden. Für den Primzahltest kannst du zunächst für alle Zahlen von 2 bis $x - 1$ überprüfen, ob sie Teiler von x sind. Der Mathe-Block `...ist Primzahl` darf nicht verwendet werden - es geht hier darum, ihn selbst zu implementieren.

Implementiere dann ein Programm, das alle Primzahlen zwischen 1 und 1000 ausgibt. Füge einen Primzahlzähler ein, der am Ende ausgibt, wie viele Primzahlen gefunden wurden. Recherchiere, ob dein Programm das korrekte Ergebnis liefert.

Zusatz: Bei größeren Zahlen braucht der Arduino schon relativ lange, um alle Rechenschritte durch-

zuführen. Dies führt zu einem typischen Problem der Informatik:

Wie kann man ein Verfahren so optimieren, dass es auch bei begrenzter Rechenkapazität annehmbar schnell abgearbeitet wird?

Überlege dir Antworten zu folgenden Fragen und optimiere dein Programm entsprechend:

- Wie groß kann ein Teiler von x maximal sein?
- Zu einem Teiler t_1 gehört immer ein zweiter Teiler t_2 mit $t_1 \cdot t_2 = x$. Zum Beispiel ist 9 ein Teiler von 18 und ein zweiter zugehöriger Teiler ist 2, denn $2 \cdot 9 = 18$. Die Überprüfung der Teiler von 18 muss aber nicht bis 9 gehen, weil die Überprüfung der 2 in diesem Fall schon ausreicht. Wie groß ist der größte Teiler, der nicht schon durch einen zugehörigen kleineren Teiler gefunden werden kann? Denke zum Beispiel an die Teiler von 36.

Aufgabe 18: Berechnungen zur Fibonacci-Folge

Die **Fibonacci-Folge** beginnt mit den Zahlen $f_1 = 1$ und $f_2 = 1$. Die darauf folgende Zahl ist die Summe der beiden vorhergehenden Zahlen:

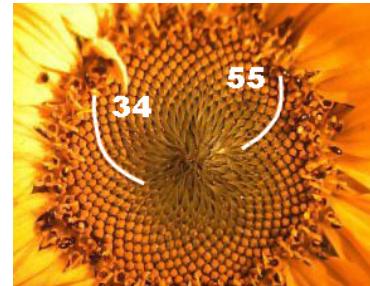
$$f_3 = f_1 + f_2 = 2$$

$$f_4 = f_2 + f_3 = 3$$

...

$$f_n = f_{n-1} + f_{n-2}$$

- Berechne schriftlich die ersten 10 Glieder der Fibonacci-Folge.
- Implementiere einen Block `gibFibonaccizahl` mit einem Argument `n`, das angibt, die wie viele Fibonacci-Zahl berechnet werden soll, und einem Argument `mitAusgabe`, das angibt, ob eine Ausgabe der vorhergehenden Folgenglieder bei der Berechnung erfolgen soll oder nicht. Die n -te Fibonacci-Zahl wird zurückgegeben.



B 3.12. Fibonacci-Zahlen finden sich in den Blütenständen von vielen Blumen. (Bild: CC-BY-SA, Urheber Dr. Helmut Haß)

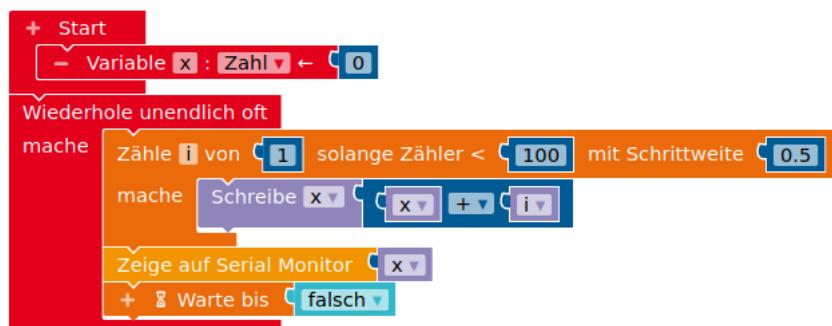
3.7. Debugging: Fehler im Programm finden

Fehler werden in der Informatik auch als „Bugs“ bezeichnet. Fehler treten beim Programmieren ständig auf und sind völlig normal. Erst nach einem Testen läuft ein Programm völlig stabil und fehlerfrei. Das Entfernen von Fehlern wird dann auch als „Debugging“ bezeichnet.

Informatiker unterscheiden zwischen zwei Fehlerarten: Syntaxfehler und logische Fehler. Logische Fehler sind dir vielleicht schon passiert und manche wurden auch schon in diesem Skript behandelt, zum Beispiel zur sonst wenn-Bedingung auf S. 18. Syntaxfehler entstehen, wenn man die Grammatik (Syntax) einer Programmiersprache nicht beachtet. In Neko werden die meisten Syntaxfehler automatisch vermieden, weil die meisten Blöcke nur ineinander greifen, wenn sie syntaktisch zueinander passen.

Unten ist ein Programm mit zwei Fehlern abgebildet. Es soll die folgende Summe berechnen:

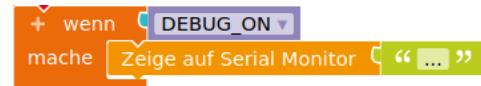
$$1 + 1,5 + 2 + 2,5 + 3 + \dots + 98,5 + 99 + 99,5 + 100.$$



B 3.13. Beispielprogramm zum Debuggen: Debug-Bsp.xml

Aufgabe 19:

Um Fehler zu finden, kann man sich die Werte von Variablen auf dem seriellen Monitor ausgeben lassen. Programmierer bauen dann häufig eine Variable DEBUG_ON ein und nutzen eine Konstruktion wie rechts abgebildet.



Welchen Vorteil könnte das haben?

Öffne das oben abgebildete Programm in Neko und nutze die Debug-Konstruktion, um die Fehler zu finden oder nachzuweisen. Korrigiere sie.

Frage deinen Lehrer, wenn du den Fehler gefunden hast, aber nicht nachvollziehen kannst, wieso sich das Programm so verhält.

3.8. Das EVA-Prinzip

In diesem Kapitel hast du bereits einige Bauteile kennengelernt, aber es gibt noch viele mehr. Um dabei nicht den Überblick zu verlieren, wären Kategorien praktisch, mit denen man Bauteile und informationsverarbeitende Systeme im Allgemeinen einordnen kann.

Frage: Wie lassen sich elektrische Bauteile und informationsverarbeitende Systeme kategorisieren?

 Folie
öffnen

Aufgabe 20: *Informationsverarbeitung*

Lies die beiden Beschreibungen zur Informationsverarbeitung bei der Straßenlampe und beim Menschen. Beschreibe Gemeinsamkeiten.

Beispiel Dämmerungsschaltung:

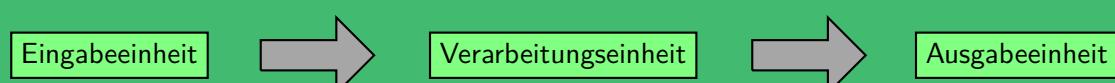
Der Aufbau von Festwiderstand und LDR ermöglicht die Eingabe von Daten zur Helligkeit. Auf dem Arduino werden die elektrischen Signale entsprechend des laufenden Programms verarbeitet. Letztlich erfolgt die Ausgabe durch das Leuchten einer LED, wenn es dunkel ist, bzw. durch das Nicht-Leuchten der LED.

Beispiel Mensch:

Unsere Sinne (Augen zum Sehen, Ohren zum Hören, ...) ermöglichen die Eingabe von Informationen in das System Mensch. Im Gehirn und den weiteren Nervenbahnen im Körper werden die Signale verarbeitet. Schließlich kommt es zu einer Ausgabe - zum Beispiel zu einer Bewegung (Musik leiser drehen, Augen zukneifen, sprechen mit dem Mund ...).

Das EVA-Prinzip

Informationsverarbeitende Systeme lassen sich nach ihrer Funktion in drei Einheiten zerlegen: Eingabeeinheit, Verarbeitungseinheit, Ausgabeeinheit.



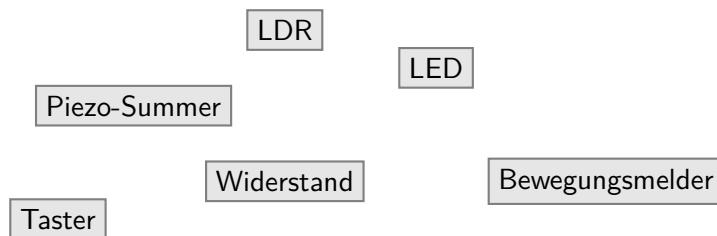
Mit dem EVA-Prinzip wird die grundlegende Reihenfolge der Verarbeitung von Daten charakterisiert. Die Einheiten bestehen dabei nicht nur aus den Bauteilen, sondern beinhalten auch die Art der Verarbeitung, also zum Beispiel das Programm auf dem Arduino.

Aufgabe 21: *Kleines Begriffstraining*

- Kategorisiere die *Juke-Box* (s. S. 13) nach dem EVA-Prinzip.
- Kategorisiere den *Reaktionszeitmesser* (s. S. 20) nach dem EVA-Prinzip.

Aufgabe 22: Du hast bisher (mindestens) folgende Bauteile verwendet:

Folie
öffnen



Benenne Gemeinsamkeiten und Unterschiede. Welche Bauteile lassen sich zusammenfassen?

Sensoren und Aktoren mit digitalen und analogen Signalen

Für die Eingabe von Daten werden *Sensoren* benötigt; für die Ausgabe hingegen *Aktoren*:

- **Sensoren** (auch Fühler genannt) sind elektrische Bauteile, die eine physikalische Größe aus der Umwelt (Temperatur, Helligkeit, Luftdruck oder auch ein mechanischer Druck mit dem Finger) in eine elektrische Größe (Widerstand, Spannung, elektrisches Potential, Stromstärke) umwandeln. Dadurch werden die physikalischen Größen aus der Umwelt einer elektronischen Verarbeitung zugänglich.
- **Aktoren** (auch Aktuatoren genannt) sind elektrische Bauteile, die eine elektrische Größe in eine mechanische (Bewegung, Schallwellen) oder andere Größe (Temperatur, Licht, ...) umwandeln. Sie ermöglichen, dass die elektronische Verarbeitung zu Handlungen bzw. Konsequenzen führen kann.

Die Signale von Sensoren und Aktoren können digital oder analog sein:

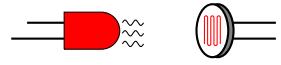
- **Digitale Signale** haben nur zwei mögliche Zustände - z. B. an / aus, gedrückt / nicht gedrückt oder 1 / 0.
- **Analoge Signale** haben unendlich viele mögliche Werte, weil sie beliebig fein eingeteilt werden können. Digitale Geräte wie der Arduino können nur *quasi* analoge Signale einlesen. Bei den „analogen“ Eingängen A0, A1, ... A5 des Arduino sind 1024 verschiedene Stufen möglich; bei „analogen“ Ausgängen (Pins mit einer Tilde: ~) sind 256 verschiedene Stufen möglich. Diese Einteilung ist für die meisten Aufgaben fein genug.



B 3.14. Digitale und analoge Aktoren und Sensoren in Neko.

Projekt 9: Alarmanlage mit Lichtschranke

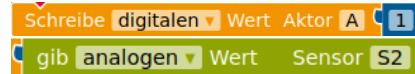
Baue eine Alarmanlage, indem du mit einer LED (Vorwiderstand!) und einem LDR eine Lichtschranke baust. Wird diese unterbrochen, soll ein akustischer Alarm ertönen. Über einen zusätzlichen Taster (mit Widerstand!) soll die Alarmanlage „scharf“ gestellt bzw. wieder ausgestellt werden können.



Hinweise:

Konfiguriere alle benötigten Bauteile als Aktoren und Sensoren.

Links:  Taster anschließen,  LDR anschließen



Aufgabe 23: Analoge Aktoren

Analoge Aktoren kamen bisher nicht vor. Schließe eine LED mit Vorwiderstand an Pin 5 an und konfiguriere einen entsprechenden analogen Aktor.

- An Pin 5 können Werte von 0 bis 255 ausgegeben werden. Implementiere eine Zählschleife, die diese Werte systematisch durchläuft und beschreibe die Auswirkung auf die LED.
- Probiere Werte größer als 255 aus und beschreibe, welche Auswirkung diese haben.

3.9. Vermischte Übungen

3.10. Ausblick

Offene Fragen:

- Was passiert, wenn ein Digitalpin angestellt oder ausgestellt wird?
→ Kap.: Elektrische Grundlagen
- Was passiert bei der Messung der Lichtstärke?
→ Kap.: Elektrische Grundlagen
- Wie werden weitere Bauteile angeschlossen und im Programm angesprochen?
→ Kap.: Bauteilkunde
- Was sind Listen und wie verwendet man sie?
- Wie werden die Daten codiert, um sie auf dem Arduino abzuspeichern oder zwischen Arduino und Computer auszutauschen?

Motivationsquellen

> [Staubsaugerroboter](#)

Mithilfe eines Arduino haben zwei Schüler aus den Niederlanden ihren eigenen Staubsaugerroboter gebaut. Das Teile für das Gehäuse haben sie mit einem 3D-Drucker gedruckt.

> [Levitation](#)

Durch Levitation lassen sich Gegenstände scheinbar von Geisterhand in der Luft schweben. Die nötige Elektronik dafür lässt sich mit einem Arduino realisieren.

> [Arduino-kontrollierter LED-Streifen zur Visualisierung von Musik](#)

Der Arduino lässt sich zudem über ein Smartphone ansteuern.

> [Spielekonsole von Makerbuino](#)

Mithilfe eines Bausatzes lässt sich eine kleine, Arduino-basierte Spieldeskonsole bauen.

4. Elektrische Grundlagen zu digitalen und analogen Pins

In der Arduino-Welt kommt man bereits sehr weit, wenn man die vorkonfigurierten Bauteile nach Anleitung anschließt und dann mit dem Programmieren loslegt. Allerdings kann man manche Schaltungen auch effizienter aufbauen oder anders herum anschließen oder aus den Sensorwerten mehr als nur qualitative Prozentwerte ermitteln, deren genaue Bedeutung völlig unklar ist. Um dies zu erreichen, muss man die physikalischen Grundlagen der Arduino-Pins und einiger grundlegender Bausteine von elektrischen Schaltungen verstehen, die im Folgenden thematisiert werden.

Hinweis: Falls dieses Skript für einen reinen Informatikunterricht genutzt werden soll, kann dieses Kapitel übersprungen werden.

In diesem Kapitel lernst du...

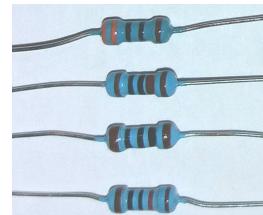
- ... Widerstand, Spannung und Stromstärke im Stromkreis zu berechnen,
- ... das elektrische Potential an einem digitalen Pin einzulesen,
- ... Pulsweitenmodulation zu benutzen,
- ... Spannungen zu messen und Spannungen an einem Spannungsteiler zu berechnen,
- ... ein Potentiometer, einen LDR sowie einen NTC auszulesen, um eine Drehung, die Umgebungshelligkeit bzw. die Temperatur zu messen,
- ... wie Transistoren verwendet werden,
- ... wie man einen Elektromotor steuert,

Projekte in diesem Kapitel:

> Raketencountdown	41	> Dimmbare Lampe ohne μC	49
> Fußgängerampel reloaded	43	> Digitales Thermometer	56
> Kerzenfunkeln	44	> Straßenlampe ohne μC	59
> Batterietester ($U < 5V$)	46	> Automatischer Lüfter	63
> Batterietester ($U > 5V$)	46	> Waschmaschinensteuerung	66
> Dimmbare Lampe	49		

4.1. Spannung, Stromstärke und Widerstand berechnen

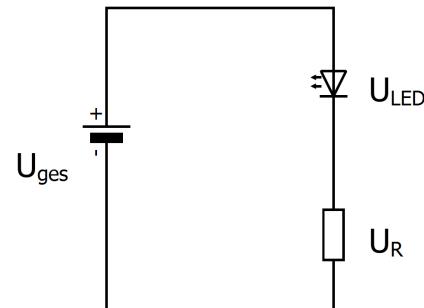
Bisher war die Größe des Vorwiderstands von LEDs mit $330\ \Omega$ vorgegeben. In unserem Bausatz finden sich jedoch viele weitere Widerstände, die teilweise größer und teilweise kleiner sind.



Frage: Welche Auswirkung haben Widerstände auf den Stromkreis?
Wie kann man dies berechnen?

Aufgabe 1: Zusammenhang von Widerstand, Stromstärke und Spannung

Rechts ist eine einfache Reihenschaltung mit einer Spannungsquelle, einer LED und einem Vorwiderstand abgebildet. Stelle eine Vermutung an, ob die LED heller oder dunkler leuchten wird, wenn man den Vorwiderstand verkleinert. Begründe deine Vermutung.



B 4.1. Reihenschaltung von LED und Vorwiderstand an einer Spannungsquelle.

Aufgabe 2: Mindestgröße des Vorwiderstands

Berechne, wie groß der Vorwiderstand einer LED mindestens sein muss, damit sie nicht durchbrennt.

Hinweise:

- Wenn ein Digitalpin angeschaltet wird (zum Beispiel durch die Anweisung `schalte LED an` oder schreibe digitalen Wert ... 1), dann gibt er eine Spannung von 5V gegenüber GND aus.
- Durch eine LED darf höchstens ein Strom von 20 mA fließen.
- Je nach Farbe halten LEDs eine andere maximale Spannung aus:

Farbe	rot	gelb/grün	blau/weiß
U_{LED}	1,6 V - 2,2 V	1,9 V - 2,5 V	2,7 V - 3,5 V

Widerstand, Spannung und Stromstärke

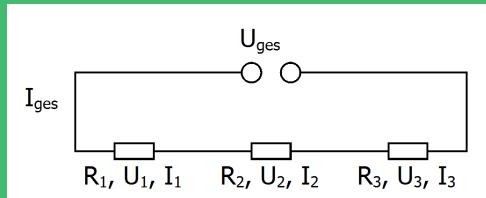
Der Widerstand R ist definiert als das Verhältnis von Spannung U zu Stromstärke I :

$$R = \frac{U}{I}.$$



Ein Widerstand heißt *ohmscher Widerstand*, wenn das Verhältnis $\frac{U}{I}$ stets gleich groß ist (also wenn R unabhängig von Stromstärke und Spannung konstant ist).

Elektrische Stromstärke und Spannung in der Reihenschaltung



- In einer Reihenschaltung ist die Stromstärke an jeder Stelle gleich groß:

$$I_{ges} = I_1 = I_2 = I_3 = \dots$$

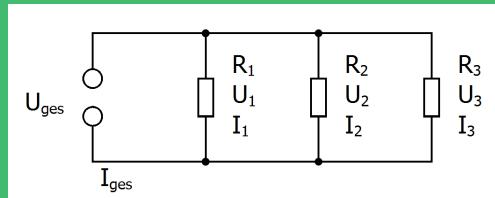
- In einer Reihenschaltung teilt sich die Gesamtspannung auf die einzelnen Bauteile auf:

$$U_{ges} = U_1 + U_2 + U_3 + \dots$$

- In einer Reihenschaltung addieren sich die Einzelwiderstände zum Gesamtwiderstand:

$$R_{ges} = R_1 + R_2 + R_3 + \dots$$

Elektrische Stromstärke und Spannung in der Parallelschaltung



- In einer Parallelschaltung teilt sich die Gesamtstromstärke auf die einzelnen Zweige auf:

$$I_{ges} = I_1 + I_2 + I_3 + \dots$$

- In einer Parallelschaltung ist die Spannung in jedem Zweig gleich groß:

$$U_{ges} = U_1 = U_2 = U_3 = \dots$$

- In einer Parallelschaltung ist der Kehrwert des Gesamtwiderstands gleich der Summe der Kehrwerte der einzelnen Widerstände:

$$\frac{1}{R_{ges}} = \frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3} + \dots$$

Aufgabe 3:

Unter den Bauteilen im Arduino-Kasten befindet sich auch ein 9V Akku. Berechne den mindestens notwendigen Vorwiderstand, wenn eine rote LED an den 9V Block angeschlossen wird.

Aufgabe 4:

- An einem 9V Block sollen drei rote LEDs in Reihe geschaltet betrieben werden. Zeichne einen Schaltplan und berechne den passenden Vorwiderstand.
- An einem 9V Block sollen drei rote LEDs parallel geschaltet betrieben werden. Zeichne einen Schaltplan und berechne den passenden Vorwiderstand.

Aufgabe 5: Ampelschaltung

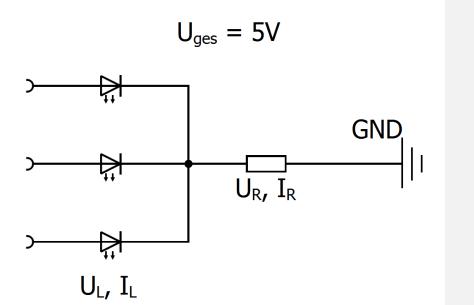
Max überlegt sich, dass er für eine Ampelschaltung am Arduino denselben Vorwiderstand für drei parallel geschaltete LEDs verwenden kann, sodass er nur einen Widerstand heraussuchen muss. Die Berechnung der Mindestgröße nimmt er folgendermaßen vor.

$$I_L = 20 \text{ mA} = 0,02 \text{ A} \implies I_R = 0,06 \text{ mA}$$

$$U_L = 2,2 \text{ V} \quad (\text{max. Spannung, die rote LEDs aus-halten})$$

$$U_R = 5 \text{ V} - 2,2 \text{ V} = 2,8 \text{ V}$$

$$R = \frac{U_R}{I_R} = \frac{2,8 \text{ V}}{0,06 \text{ A}} \approx 46,67 \Omega$$



Der Vorwiderstand sollte eine Größe von mindestens 50Ω haben.

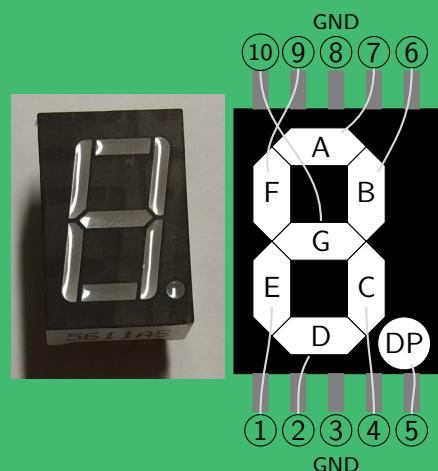
Begründe, warum der oben berechnete Vorwiderstand zu niedrig ist. Erkläre, wie man stattdessen vorgehen müsste und gib den korrekten Wert für einen möglichen gemeinsamen Vorwiderstand an.

Aufgabe 6: Vorbereitungen zur 7-Segment-Anzeige

- a) Zeichne einen vereinfachten Schaltplan einer 7-Segment-Anzeige, in dem die LEDs einzeln eingezeichnet sind (siehe Infokasten unten).
- b) Es wäre sehr umständlich, für jede LED einen eigenen Vorwiderstand anzuschließen; praktischer ist es, einen einzigen Vorwiderstand zwischen GND-Anschluss der 7-Segment-Anzeige und GND-Anschluss des Arduino anzubringen. Die Größe des gemeinsamen Vorwiderstands der acht LEDs (Anzeige & Punkt) soll 330Ω betragen. Berechne die Gesamtstromstärke und die Stromstärke in jeder LED bei Darstellung einer 1 und einer 8 (jeweils ohne Punkt).

7-Segment-Anzeige

Eine 7-Segment-Anzeige besteht aus sieben roten LEDs, die so angeordnet sind, dass sich mit ihnen eine Zahl darstellen lässt. Zusätzlich gibt es zur leichteren Unterscheidung von 6 und 9 eine LED für den Punkt. Jede LED lässt sich einzeln über einen der Pins ansteuern, wobei sich alle LEDs einen gemeinsamen GND-Anschluss teilen. Der zweite GND-Anschluss soll hier nicht genutzt werden, um die Schaltung so einfach wie möglich zu halten.



Projekt 1: Raketencountdown

Suche dir nun den passenden Widerstand für die 7-Segment-Anzeige heraus und verbinde beide mit dem Arduino. Programmiere dann einen Raketencountdown, der von 9 rückwärts bis 0 zählt.

Tipp: Erstelle dir zuerst eine Tabelle, in der du übersichtlich festhältst, welche LEDs für welche Zahl an sein müssen und mit welchen Pins am Arduino diese verbunden sind.

Für Schnelle: Man kann mit einer 7-Segment-Anzeige auch Buchstaben darstellen und nacheinander durchlaufen lassen!

Idee: Frick, Fritsch und Trick (2015): *Einführung in Mikrocontroller - Der Arduino als Steuerzentrale*, Bad Saulgau

Widerstandsringe
ablesen

4.2. Das elektrische Potential

Die Ausgabe von 5V gegenüber GND an einem digitalen Ausgang des Arduino ist vergleichbar mit einer Batterie oder einem Spannungsgerät. Um zu verstehen, wie der Arduino digitale Signale einliest und dadurch auf seine Umwelt reagieren kann, muss jedoch zuerst geklärt werden, was sich hinter dem *elektrischen Potential* verbirgt.

Frage: Wie werden digitale Signale am Arduino eingelesen?

Folie
öffnen

Aufgabe 7: Eine Analogie für das elektrische Potential

- Anna und Bert schauen auf dasselbe Fenster. Anna meint, das Fenster befindet sich in 1 Meter Höhe. Bert hingegen meint, das Fenster befindet sich in 4 Meter Höhe. Beide haben jeweils für sich betrachtet Recht. Wie kann das sein?
- Die Vase fällt einen Meter tief. Gib an, ...
 - ... mit welcher Höhe Anna die Höhenenergie nach dem Fallen berechnet und mit welcher Höhendifferenz sie die Höhenenergie berechnet, die in Bewegungsenergie umgewandelt wurde.
 - ... mit welcher Höhe Bert die Höhenenergie nach dem Fallen berechnet und mit welcher Höhendifferenz er die Höhenenergie berechnet, die in Bewegungsenergie umgewandelt wurde.

Hinweis: $E_H = m \cdot g \cdot h$

Aufgabe 8: Übersicht

- Übertrage und vervollständige die folgende Tabelle von Analogien.
- Erkläre, welche der genannten Größen in der Mechanik bzw. der Elektrik abhängig von der Festlegung eines Nullniveaus sind.

Mechanik	Elektrik
Höhenenergie	
	Elektrisches Potential
Höhendifferenz	
Grundhöhe	

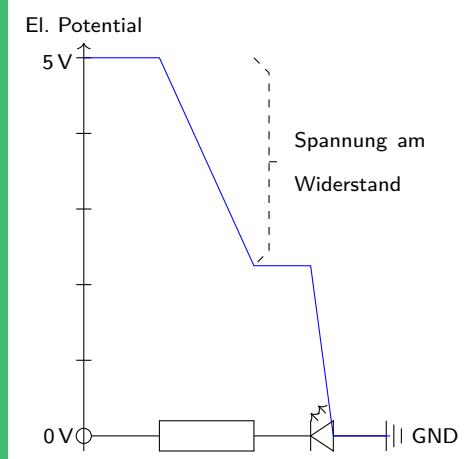
Elektrisches Potential

So wie die Höhendifferenz ein Maß für die Höhenenergie ist, die umgewandelt wird (z. B. in Bewegungsenergie), ist die Spannung ein Maß für die elektrische Energie, die an einer LED, einem Widerstand etc. umgewandelt wird.

Das elektrische Potential hingegen ist wie die Höhe ein Maß für die elektrische Energie der Elektronen im Stromkreislauf. Es kann nur in Bezug auf ein Nullniveau („Ground“/GND) angegeben werden. Die Einheit des elektrischen Potentials ist Volt.

Elektrisches Potential am GND-Pin: 0V

Elektrisches Potential am 5V-Pin: 5V

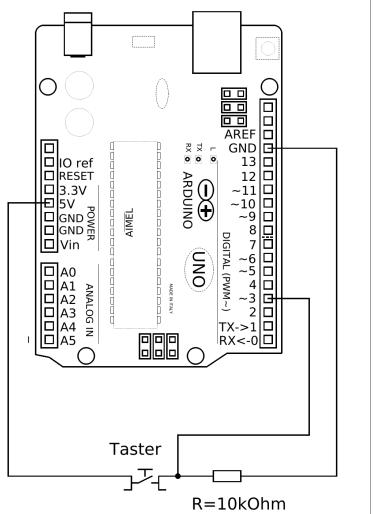


Aufgabe 9: Pulldown-Widerstand

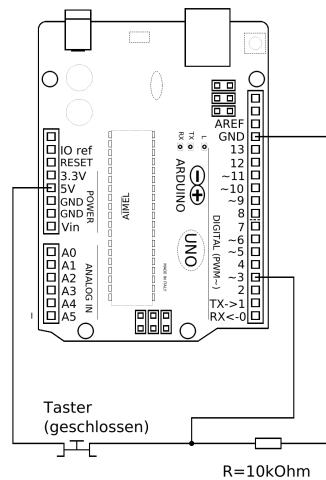
In dem unten abgebildeten Schaltplan ist dargestellt, wie man einen Taster am Arduino so anschließt, dass man seinen Zustand im digitalen Pin 3 auslesen kann. Der Widerstand wird auch als *Pulldown-Widerstand* bezeichnet und sollte relativ groß sein. 10 k Ω sind üblich.

Markiere die Kabel farbig, sodass die Kabel, die auf dem gleichen elektrischen Potential liegen, die gleiche Farbe haben. Notiere zudem den Wert des elektrischen Potentials.

 Vorlage
öffnen



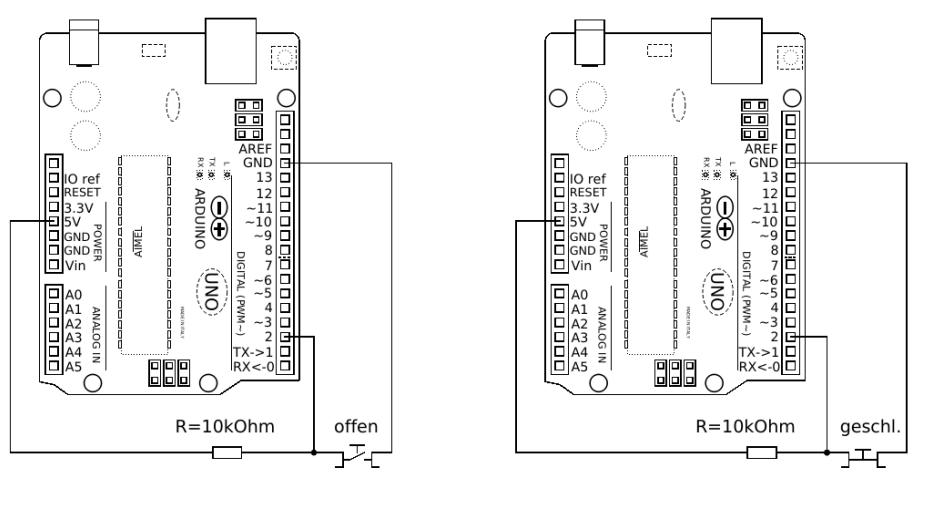
B 4.2. Taster offen (kein Stromfluss).



B 4.3. Taster geschlossen (Stromfluss).

Aufgabe 10: Pullup-Widerstand

Eine Alternative zu der bekannten oberen Schaltung ist die Schaltung mit einem sogenannten Pullup-Widerstand. In der Abbildung ist die Schaltung mit einem Taster und einem Pullup-Widerstand dargestellt.



- Markiere die Kabel jeweils farbig, sodass die Kabel, die auf dem gleichen elektrischen Potential liegen, die gleiche Farbe haben. Notiere zudem den Wert des elektrischen Potentials.
- Erläutere die Bedeutung der beiden Begriffe *Pulldown* und *Pullup*.

Hinweis: to pull - engl. für „ziehen“

Projekt 2: Fußgängerampel reloaded

Baue und programmiere eine Fußgängerampel mit einer Pullup-Schaltung für den Taster!

Digitale Pins des Arduino

Die digitalen Pins des Arduino von 0 bis 13 kennen nur zwei Zustände, für die es unterschiedliche Bezeichnungen gibt (siehe rechts). Sie können als digitaler Ausgang oder als digitaler Eingang konfiguriert werden. Bei einem digitalen Ausgang kann eine Spannung von 5 V oder 0 V gegenüber GND ausgegeben werden. Ein digitaler Eingang kann Spannungen zwischen 0 V und 5 V einlesen; dabei werden Spannungen von 0 V bis 1,4 V als *LOW* oder 0 interpretiert, größere Spannungen als *HIGH* oder 1.

Bezeichnungen für Zustände von digitalen Pins

An	Aus
HIGH (5 V)	LOW (0 V)
1	0

Hinweis: Bei vielen anderen Mikrocontrollern entspricht das HIGH-Potential 3,3 V.

4.3. Pulsweitenmodulation

Ziel: Mithilfe des Arduino soll eine funkelnde LED-Kerze gebaut werden.

Der Arduino verfügt über mehrere sogenannte PWM-Pins, die mit einer Tilde (~) gekennzeichnet sind. Du hast diese Pins schon bei den analogen Aktoren kennen gelernt, weil diese über Pulsweitenmodulation (PWM) angesprochen werden. Die PWM-Werte, die der Anweisung übergeben werden können, variieren von 0 bis 255.

 Schreibe analogen Wert Aktor A 135

Aufgabe 11: Fading

- Schließe eine LED mit Vorwiderstand an einen PWM-Pin an und konfiguriere einen analogen Aktor an diesem Pin. Implementiere dann eine Zählschleife, die systematisch alle PWM-Werte von 0 bis 255 durchläuft. Beschreibe die Auswirkung auf die LED.
- Sorge dafür, dass die PWM-Werte nach Erreichen der 255 genauso rückwärts von 255 bis 0 durchlaufen werden.
- Zum Experimentieren: Was passiert, wenn PWM-Werte größer als 255 übergeben werden?

Aufgabe 12: Pulsweitenmodulation

Erkläre mithilfe der Zusammenfassung zur Pulsweitenmodulation, was bei der Nutzung des Befehls  Schreibe analogen Wert Aktor A 158 passiert. Berechne auch die mittlere Spannung, die am PWM-Pin ausgegeben wird.

Für Physik-Profis: Eine blaue LED hält bis zu 3,5 V aus, ohne durchzubrennen. Trotzdem darf man sie bei Verwendung dieses Befehls nicht ohne Vorwiderstand an den Pin anschließen. Begründe.

Projekt 3: Kerzenfunkeln

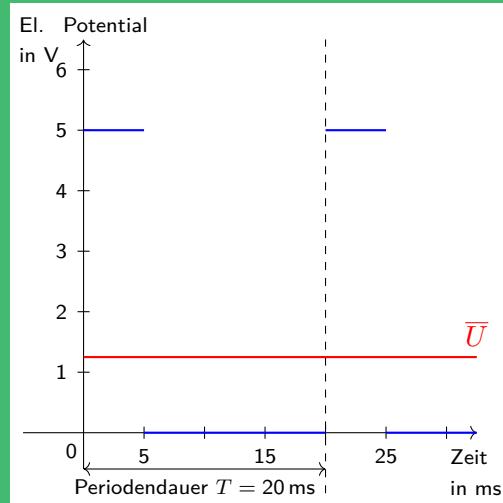
Modelliere mithilfe von drei LEDs das Funkeln von Kerzen.

Tipp: Die Verwendung des Blocks für Zufallszahlen wird dir helfen, das Funkeln natürlicher aussehen zu lassen.

Pulsweitenmodulation (PWM)

Bei der Pulsweitenmodulation wechselt der ausgewählte digitale Pin sehr schnell (mit einer Frequenz von 50 Hz) zwischen den elektrischen Potentialen 5V und 0V hin und her - es ergibt sich also ein gepulstes Signal, dessen Weite (Dauer) moduliert werden kann. Aus dem Verhältnis der Zeit, in der der Pin auf einem 5V-Potential liegt, zu der Zeit, in der der Pin auf einem 0V-Potential liegt, ergibt sich eine mittlere Spannung (gegenüber Ground), die scheinbar am Pin anliegt. Wenn der Pin in der Hälfte der Zeit auf 5V und in der anderen Hälfte auf 0V liegt, dann ergibt sich eine mittlere Spannung von $\bar{U} = 2,5V$. Wenn der Pin nur in einem Viertel der Zeit auf 5V liegt, dann ergibt sich eine mittlere Spannung von $\bar{U} = 1,25V (= 5V \cdot 0,25)$.

Das Verhältnis der Zeit mit 5V zu der Gesamtdauer einer Periode mit 5V und 0V wird als *Tastverhältnis* bezeichnet. Im Programm wird das Tastverhältnis durch einen Wert zwischen 0 und 255 angegeben. Eine 0 bedeutet, dass die Zeit mit 5V 0% ausmacht, also liegt der Pin durchgängig auf einem 0V-Potential. Eine 255 bedeutet, dass die Zeit mit 5V 100% ausmacht, also liegt der Pin durchgängig auf einem 5V-Potential. Diese beiden Werte entsprechen dem, was bei den bekannten Befehlen zur Steuerung von digitalen Pins passiert. Ein Wert von 100 bedeutet einen Anteil von $\frac{100}{255} \approx 0,39$ der Periodendauer. Daraus ergibt sich eine mittlere Spannung von $\bar{U} = 5V \cdot 0,39 \approx 1,96V$.



B 4.4. Darstellung des zeitlichen Verlaufs einer Pulsweitenmodulation mit einem Tastverhältnis von 25%.

4.4. Spannung messen

Wenn Batterien kaum noch Ladung gespeichert haben, lässt die Spannung an ihren Polen nach und sinkt unter den Wert, der auf der Batterie vermerkt ist. Mithilfe der analogen Eingänge A0 bis A5 lässt sich die Spannung messen und so entscheiden, ob die Batterie noch brauchbar ist.

 Vorlage
öffnen

Frage: Wie kann man mit dem Arduino eine Spannung messen?

Projekt 4: Batterietester (Voltmeter für $U < 5 \text{ V}$)

Für eine einfache Messung bei einer 1,5 V-Batterie wird der negative Pol der Batterie mit GND verbunden, sodass ein gemeinsames Nullpotential vorliegt. Der positive Pol der Batterie wird mit einem der analogen Eingänge A0 bis A5 verbunden. Über einen eingebauten Analog-Digital-Wandler (*engl. analog-to-digital converter, ADC*) wird der Spannungswert durch eine Zahl zwischen 0 und 1023 ausgedrückt.

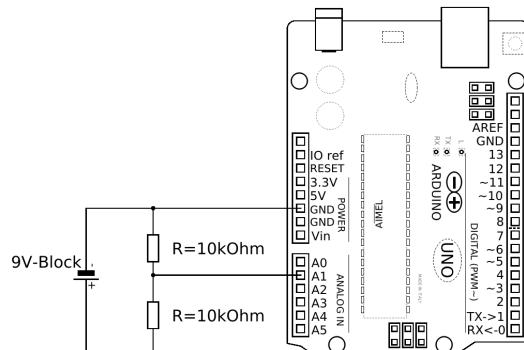
Analogwert	Spannung
0	0 V
1	
100	
1023	5 V

- Schließe eine mit 1,5 V beschriftete Batterie an den Arduino an und miss die Spannung. Berechne aus dem Analogwert die Spannung und lass sie auf dem seriellen Monitor ausgeben.
- Ergänze den Batterietester um eine Ampel, die anzeigt, ob die Batterie voll aufgeladen bzw. noch in Ordnung bzw. leer ist.

Projekt 5: Batterietester (Voltmeter für $U > 5 \text{ V}$)

Da der Arduino beim direkten Anschließen nur maximal 5V „verträgt“, muss man zum Testen von z.B. 9V-Blöcken weitere Bauteile verwenden. Mit zwei $10\text{k}\Omega$ Widerständen kann man einen einfachen *Spannungsteiler* aufbauen, der die Messung ermöglicht.

- Berechne die Stromstärke und die Spannung an den Widerständen. Warum sind große Widerstände hier sinnvoll?
- Markiere die Kabel in der Abbildung farbig, sodass die Kabel, die auf dem gleichen elektrischen Potential liegen, die gleiche Farbe haben. Notiere zudem den Wert des elektrischen Potentials.
- Gib an, wie man mit dem Arduino die Spannung am 9V-Block berechnet. Baue den Batterietester dann auf und probiere ihn mit dem 9V-Block aus der Box aus.
- Zum Nachdenken: Wie groß darf die Spannung beim oben verwendeten Spannungsteiler maximal sein, damit am Arduino nicht mehr als 5V anliegen? Wie könnte man den Spannungsteiler bauen, sodass man Spannungen bis zu 15V messen kann?



Hinweis: Ganz ähnlich funktioniert ein Multimeter, bei dem man mit einem Drehregler ein passendes Widerstandsverhältnis für den aufgedruckten Messbereich einstellen kann. Auch im Multimeter werden für die Spannungsmessung möglichst große Widerstände verwendet.



B 4.5. Einfaches Multimeter.

(Quasi) Analoge Pins am Arduino

Die mit einer Tilde versehenen Digitalpins am Arduino verfügen über Pulsweitenmodulation, über die sich eine mittlere Spannung einstellen lässt, die quasi einem analogen Signal entspricht. Genau genommen sind 256 Stufen von 0 (0 V) bis 255 (5 V) möglich, woraus sich ergibt, dass die Stufen sich um 0,02 V unterscheiden.

Die Pins mit der Beschriftung A0 bis A5 werden als analoge Eingänge bezeichnet, weil sich mit ihnen Spannungen zwischen 0 V und 5 V messen lassen. Auch hier handelt es sich nur um eine quasi analoge Messung, denn der Messbereich ist in 1024 Stufen von 0 (0 V) bis 1023 (5 V) unterteilt, woraus sich ergibt, dass die Stufen sich um 0,005 V unterscheiden.

4.5. Drehregler verwenden

Die Messung einer variablen, (quasi-)analogen Spannung eröffnet neue Möglichkeiten, da die Eingabewerte nun viel differenzierter sind als bei einem Taster, bei dem die Eingabe nur aus „0“ oder „1“ bestand. Zum Beispiel kann man darüber angeben, wie hell eine Lampe leuchten soll bzw. wie stark sie gedimmt werden soll. Dazu werden Potentiometer verwendet.

Frage: Wie funktioniert ein Potentiometer?

Folie
öffnen

Aufgabe 13: Bleistiftpotentiometer

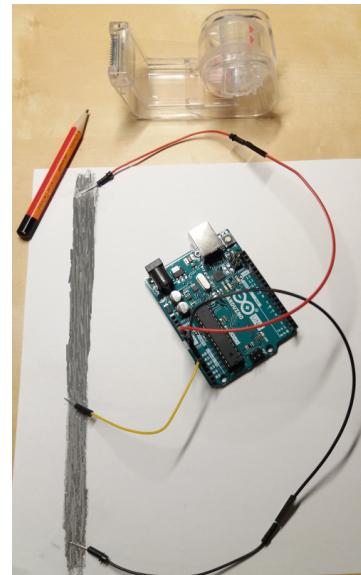
Ein einfaches Potentiometer kannst du selbst bauen.

Basteln: Markiere dafür mit Bleistift einen dicken Strich auf einem Blatt Papier und klebe am einen Ende ein Kabel fest, das mit GND verbunden ist. Klebe ans andere Ende ein Kabel, das mit 5V verbunden ist. Mit einem dritten Kabel („Sensorkabel“), das mit einem analogen Eingang verbunden ist, lässt sich nun messen, welches elektrische Potential an einer beliebigen Stelle des Bleistiftstreifens anliegt.

Experimentieren: Schreibe ein Programm, dass dir fortlaufend auf dem seriellen Monitor die Analogwerte und die umgerechneten Werte für das elektrische Potenzial bzw. die Spannung gegenüber GND anzeigt. Bewege dann das Sensorkabel über den Streifen und beobachte, wie sich die Spannungswerte verändern.

Analysieren: Der Bleistiftstreifen leitet den Strom bei einem bestimmten Gesamtwiderstand R_{ges} . Durch das Sensorkabel wird der Streifen in zwei Teile mit den Teilwiderständen R_1 und R_2 geteilt. Erläutere anhand deiner Beobachtungen, wie die drei Widerstände zusammenhängen.

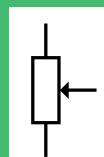
Idee: Frick, Fritsch und Trick (2015): *Einführung in Mikrocontroller - Der Arduino als Steuerzentrale*, Bad Saulgau



Potentiometer

Ein **Potentiometer**, kurz: Poti, ist im Grunde nichts anderes als ein Spannungsteiler mit zwei Widerständen. Jedoch kann die Größe der Widerstände z. B. durch Drehen variiert werden. Der Gesamtwiderstand bleibt dabei immer gleich.

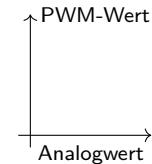
Beim Anschluss an den Arduino wird der mittlere Pin des Potentiometers an einen analogen Eingang angeschlossen. Die anderen beiden Pins werden mit GND und 5V verbunden.



Projekt 6: Dimmbare Lampe

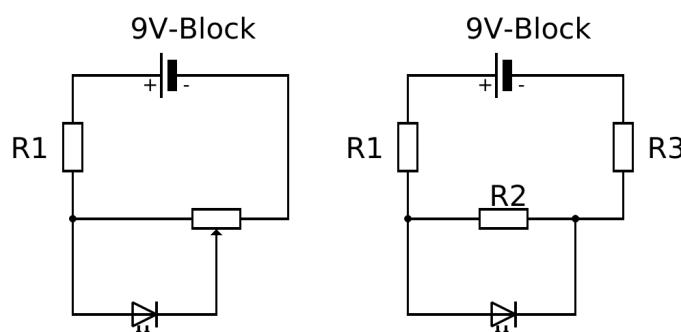
Baue und programmiere eine Lampe, deren Helligkeit sich durch ein Potentiometer einstellen lässt.

Hinweis: Du musst dafür sorgen, dass der eingelesene Analogwert zwischen 0 und 1023 in einen PWM-Wert zwischen 0 und 255 umgerechnet wird. Ermittle dazu eine passende Funktion.



4.5.1. Die Verwendung eines Potentiometers ohne Mikrocontroller

Für einige Projekte, wie das Dimmen einer Lampe, ist ein Mikrocontroller eigentlich überdimensioniert, weil sich die Funktion schon durch eine reine Hardwarelösung erreichen lässt.



B 4.6. Auf der linken Seite ist die Anwendung eines Potentiometers ohne Mikrocontroller dargestellt. Auf der rechten Seite ist der zugehörige Ersatzschaltplan gezeichnet, der zeigt, dass das Potentiometer als Spannungsteiler mit zwei variablen Widerständen R_2 und R_3 aufgefasst werden kann.

Projekt 7: Dimmbare Lampe ohne Mikrocontroller

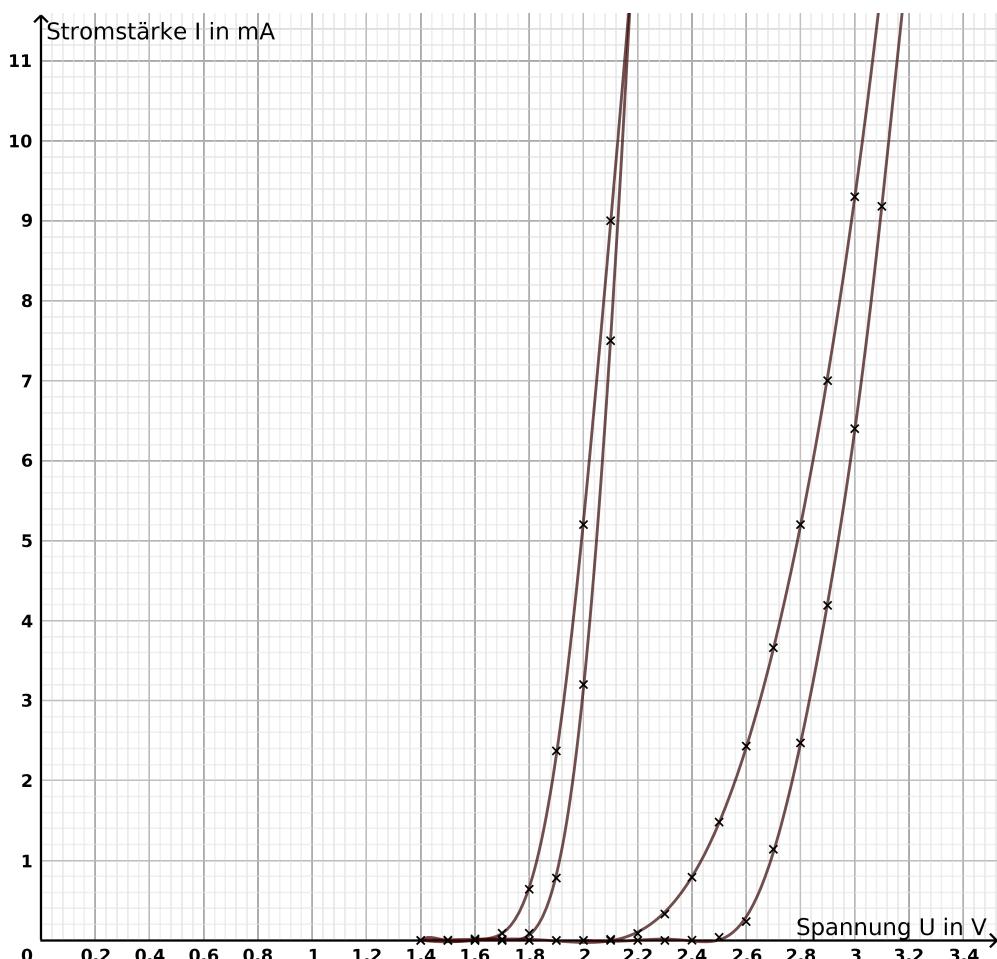
- Erkläre, wie sich die LED verhält, wenn das Potentiometer so gedreht ist, dass gilt:
 - $\dots R_2 = 0 \Omega, R_3 = 10 \text{ k}\Omega,$
 - $\dots R_2 = 10 \text{ k}\Omega, R_3 = 0 \Omega.$
- Bevor die Schaltung aufgebaut werden kann, muss die Größe des Vorwiderstands R_1 berechnet werden. Der Vorwiderstand muss den Strom auch dann noch klein genug halten, wenn das Potentiometer so gedreht ist, dass gilt: $R_3 = 0 \Omega$ (und dementsprechend $R_2 = 10 \text{ k}\Omega$). Berechne R_1 so, dass die Stromstärke durch die LED maximal 20 mA beträgt.
Hinweis: Die Stromstärke durch R_2 kann vernachlässigt werden, sodass $I_{ges} \approx I_{LED} = 20 \text{ mA}$ mit $U_{LED} = 2,3 \text{ V}$ gilt.
Begründung: Wenn $R_1 = 0 \Omega$ wäre, würde die gesamte Spannung an R_2 abfallen. Dann gilt: $I_{R_2} = \frac{9 \text{ V}}{10 \text{ k}\Omega} = 0,9 \text{ mA}$. Die Stromstärke durch R_2 beträgt also nur etwa 1/20 der Stromstärke durch die LED und wenn R_1 größer wird, dann wird die Stromstärke durch R_2 sogar noch kleiner.
- Baue die Schaltung auf und beobachte das Leuchtverhalten der LED beim Drehen am Potentiometer.

Beim Experimentieren mit dem Potentiometer wirst du feststellen, dass man das Potentiometer nicht vollständig zur Seite drehen muss, damit die LED aufhört zu leuchten. Im Experiment zeigt sich, dass eine blaue LED schon bei $R_2 = 2,5 \text{ k}\Omega$ und $R_3 = 7,5 \text{ k}\Omega$ aufhört zu leuchten. Die Spannung an der LED beträgt dann $U_{L,blau} = 2,3 \text{ V}$.

Aufgabe 14: Kennlinien von Leuchtdioden

- Baue eine blaue LED in den oben dargestellten Schaltkreis und drehe das Potentiometer so, dass die blaue LED gerade nicht mehr leuchtet. Die Spannung an der blauen LED beträgt dann $U_{L,blau} = 2,3 \text{ V}$. Ersetze dann die blaue LED durch eine grüne/gelbe/rote LED. Notiere deine Beobachtungen.
- Ordne die Farben der LEDs den rechts abgebildeten Kennlinien von einer blauen, einer grünen, einer gelben und einer roten LED zu. Experimentiere dazu mit dem Potentiometer und den LEDs.

Hinweis: Das menschliche Auge ist in der Lage, bereits bei einer Stromstärke von wenigen Mikroampere ein schwaches Leuchten zu erkennen. Im Diagramm ist so eine geringe Stromstärke kaum von 0 mA zu unterscheiden.



B 4.7. U-I-Kennlinien einer roten, gelben, grünen und blauen Leuchtdiode.

4.6. Helligkeit messen

Die Helligkeit bestimmt unseren Tages- und Jahresrhythmus: Wenn es dunkel wird, schlafen wir (oder gehen feiern) und wenn es hell wird, stehen wir wieder auf und unternehmen etwas. Es ist daher nur logisch, dass es einige Anwendungen für elektrische Schaltungen gibt, die auf die Helligkeit reagieren.

In einfachen Fällen wird dabei auf einen Fotowiderstand, kurz: LDR (*engl. light dependent resistor*), zurückgegriffen. Diesen hast du bereits in Abschnitt 3.2 kennen gelernt: Sein Widerstand wird umso kleiner, je heller es ist.

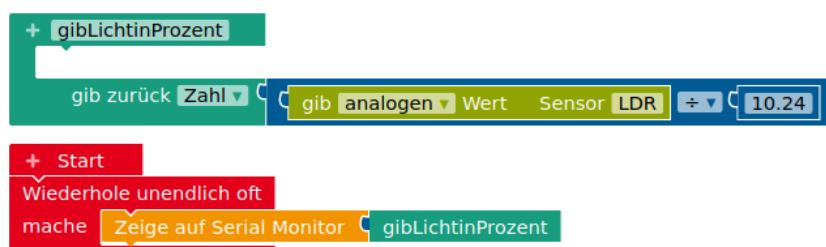
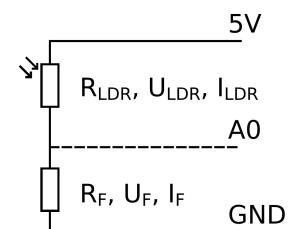


B 4.8. Ein LDR.

Frage: Wie bestimmt man mit Hilfe von Analogwerten die Helligkeit?

Aufgabe 15: LDR revisited

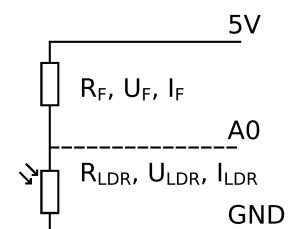
- a) Die Abbildung rechts zeigt noch einmal, wie der LDR in Abschnitt 3.2 am Arduino angeschlossen wurde. Erläutere, was im Spannungsteiler passiert, wenn die Helligkeit erhöht wird. Halte dich dabei an folgende Schritte:
Widerstand des LDR \rightarrow Spannung am LDR \rightarrow Spannung am Festwiderstand \rightarrow Analogwert in A0.
- b) Unten ist abgebildet, wie die Anweisung gib Licht % Lichtsensor LDR intern umgesetzt wird. Begründe, dass diese Angabe als Angabe für die physikalische Größe Helligkeit ungeeignet ist.



B 4.9. Die Anweisung `Gib Licht % Lichtsensor LDR` als selbst definierte Funktion.

Aufgabe 16: Der LDR im Spannungsteiler

- a) Um nicht ständig umdenken zu müssen, soll der Spannungsteiler von LDR und Festwiderstand $R_F = 10\text{ k}\Omega$ nun wie rechts abgebildet aufgebaut werden. Lasse dir die Spannung am LDR auf dem seriellen Monitor ausgeben.
- b) Erläutere wiederum, was im Spannungsteiler passiert, wenn die Helligkeit erhöht wird.



Die Änderung der Spannung resultiert aus der Änderung des Widerstands des LDR. Um einen Eindruck vom Wertebereich des Widerstands eines LDR zu bekommen, soll dieser nun berechnet werden. Anschließend lässt sich aus dem Widerstand des LDR auch die Helligkeit berechnen.

Aufgabe 17: Genaue Analyse des Spannungsteilers

- a) Leite eine Formel für den Spannungsteiler her, mit der du den Widerstand R_{LDR} des LDR mithilfe der Spannung U_{LDR} am LDR, dem Festwiderstand R_F und der Spannung U_F am Festwiderstand berechnen kannst.

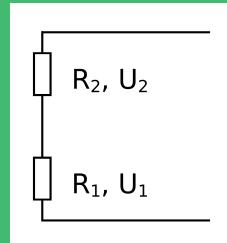
Tipp: Betrachte zuerst die Stromstärken I_F und I_{LDR} durch den Festwiderstand und den LDR. Durch das Sensorkabel fließt (näherungsweise) kein Strom.

- b) Berechne, welchen Widerstand der LDR hat, wenn er komplett abgedunkelt ist und wenn er mit einer Smartphone-Taschenlampe bestrahlt wird.

Der Spannungsteiler

Der Spannungsteiler ist ein häufig verwendeteter Teil einer Schaltung, in dem zwei Widerstände in Reihe geschaltet sind. Dadurch teilt sich die insgesamt anliegende Spannung auf die beiden Widerstände entsprechend ihrer jeweiligen Größe auf. Dabei gilt:

$$\frac{U_1}{R_1} = \frac{U_2}{R_2} = \frac{U_{ges}}{R_{ges}}$$



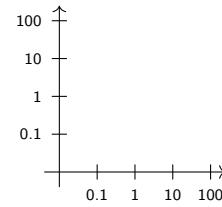
Um den funktionalen Zusammenhang zwischen Widerstand des LDR und Umgebungshelligkeit herauszufinden, gibt es drei Möglichkeiten:

- Theoretische Herleitung: Die Herleitung des Zusammenhangs mit Hilfe von anderen physikalischen Gesetzen und Annahmen führt an dieser Stelle zu weit,
- Experimentelle Ermittlung: Dazu müssten einige Widerstandswerte bei vorgegebener Helligkeit gemessen werden, jedoch ist es schwierig, eine vorgegebene Helligkeit genau herzustellen,
- Blick ins Datenblatt: Zu jedem Bauteil gibt es ein Datenblatt, in dem die zugehörigen Kennzahlen und Zusammenhänge dargestellt sind - hier können die Ergebnisse von Laborexperimenten eingesehen werden.

Aufgabe 18: Datenblatt lesen

Suche im [Datenblatt des LDR](#) den Graphen, der den Zusammenhang von Helligkeit in der Einheit Lux und Widerstand des LDR in der Einheit kΩ abbildet. Entnimm dem Graphen fünf zusammengehörige Werte von Helligkeit und Widerstand und halte diese tabellarisch fest.

Achtung: Die Achsen im Graphen sind logarithmisch skaliert. Das bedeutet, dass die Werte an den Achsen nicht gleichmäßig zunehmen, sondern exponentiell (von 0,1 zu 1 zu 10 zu 100 zu 1000). Diese Skalierung ermöglicht erst das Ablesen der Werte, aber es muss berücksichtigt werden, dass die Werte nur sehr ungenau abzulesen sind.



Das verlinkte Datenblatt ist evtl. nicht das korrekte Datenblatt zu dem LDR. Da die Bauteilnummer bei dem verwendeten Starter Kit nicht angegeben wird, ist eine Zuordnung leider nicht mehr möglich.

Aufgabe 19: Regression durchführen

Unabhängig davon, ob man die Daten aus einem Experiment oder dem Datenblatt gewonnen hat, lässt sich nun eine Regression durchführen, um den allgemeinen Zusammenhang zwischen Widerstand des LDR und Helligkeit herauszufinden.

Führe mit den ermittelten Werten eine Regression durch, indem du die unten abgebildete Anleitung befolgst. Bestimme mit der erhaltenen Funktion die Umgebungshelligkeit im Raum sowie die Helligkeit deiner Smartphone-Taschenlampe auf niedrigster und höchster Stufe.

Eine Regression durchführen

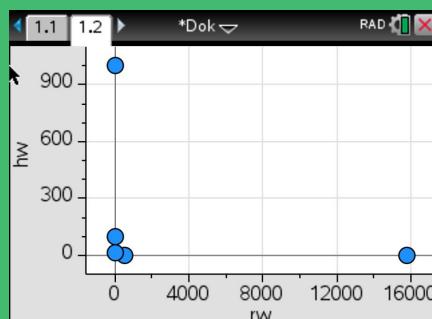
Beim Durchführen einer Regression wird diejenige Funktion(sgleichung) ermittelt, die am besten zu den gegebenen Daten passt. Die Art der Funktion muss jedoch vom Anwender sinnvoll festgelegt werden.

Regression mit TI Nspire

- Erstelle ein neues Dokument mit einer Seite „Lists & Spreadsheet“. Benenne eine Spalte als *rw* (Werte für R, also der Widerstand) und eine Spalte als *hw* (Werte für die Helligkeit). Ergänze die Werte.

A rw	B hw	C	D
15800	0.1		
500	1		
16	10		
0.5	100		
0.02	1000		

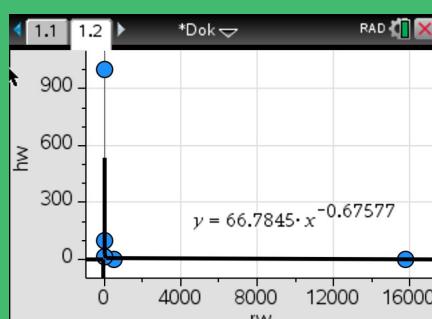
- Füge eine neue Seite „Data & Statistics“ hinzu (mit *ctrl → +page*). Da wir die Helligkeit in Abhängigkeit vom Widerstand berechnen wollen, kommt der Widerstand auf die Rechtsachse und die Helligkeit auf die Hochachse.



- Führe eine Regression durch (menu → 4: Analysieren → 6: Regression). Welche Funktionsklasse könnte zu der Verteilung der Werte passen? Falls die ausgewählte Funktionsklasse nicht zu den Werten passt, mache die Regression rückgängig (*ctrl → esc*) und probiere eine andere Funktionsklasse.

Achtung: Durch die geringe Auflösung des Taschenrechners können auch passende Funktionen ggf. falsch aussehen.

- Übersetze die Funktionsgleichung in den physikalischen Zusammenhang. Für den Widerstand ist das Formelzeichen *R* festgelegt. Für die Helligkeit wählen wir an dieser Stelle *H*.



$$y = 66,78 \cdot x^{-0,66}$$

$$\downarrow \qquad \qquad \downarrow$$

$$H = 66,78 \cdot R^{-0,66}$$

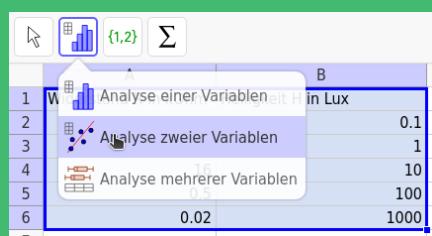
R in kΩ, *H* in Lux

Regression mit Geogebra Classic

1. Starte Geogebra Classic und wähle als Perspektive die Tabellenkalkulation. Übertrage die Daten in die Tabellenkalkulation.

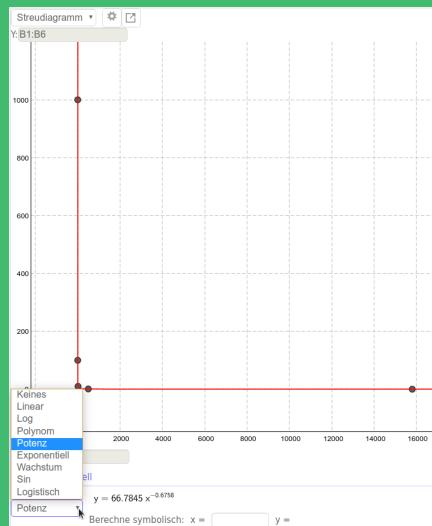
	A	B
1	Widerstand R in kOhm	Helligkeit H in Lux
2	15800	0.1
3	500	1
4	16	10
5	0.5	100
6	0.02	1000
7		

2. Markiere alle Daten und wähle das Werkzeug Analyse zweier Variablen. Die Daten aus Spalte A werden automatisch als x-Koordinate gewählt, die aus Spalte B als y-Koordinate. Bei Bedarf kann dies mit $X \rightleftarrows Y$ vertauscht werden (oben rechts).



3. Führe eine Regression durch, indem du unten links ein passendes Regressionsmodell wählst. Welche Funktionsklasse könnte zu der Verteilung der Werte passen? Falls die ausprobierte Funktionsklasse nicht zu den Werten passt, probiere eine andere Funktionsklasse.

Hinweis: Die Anzahl der Nachkommastellen lässt sich in den Einstellungen unter „Runden“ ändern.



4. Übersetze die Funktionsgleichung in den physikalischen Zusammenhang. Für den Widerstand ist das Formelzeichen R festgelegt. Für die Helligkeit wählen wir an dieser Stelle H .

$$y = 66,78 \cdot x^{-0,66}$$

$$\downarrow \qquad \qquad \downarrow$$

$$H = 66,78 \cdot R^{-0,66}$$

R in $k\Omega$, H in Lux

4.7. Temperatur messen

Nicht nur die Helligkeit beeinflusst unseren Alltag, sondern auch die Temperatur. Ganz allgemein ist die Temperatur eine wichtige Größe, die bei vielen Anwendungen eine Rolle spielt und daher erfasst und automatisiert in die Anwendung einfließen sollte: Thermostate regeln die Temperatur im Raum, Wetterstationen geben die Temperatur an und 3D-Drucker regeln die Temperatur der Düse auf eine festgelegte Temperatur, damit der Kunststoff flüssig wird, aber immer noch zäh genug bleibt, um die Figur zu bilden. Häufig wird dabei ein Heißleiter (kurz: NTC, von engl. *negative temperature coefficient*) verwendet - ein elektrischer Widerstand, der auf die Temperatur reagiert.



B 4.10. Ein NTC.

Frage: Wie verwendet man einen NTC am Arduino?

Aufgabe 20: Erste Experimente mit dem NTC

- Baue mithilfe eines Festwiderstands $R_F = 10\text{ k}\Omega$ und dem NTC einen Spannungsteiler und lies die Spannung am NTC in A0 aus (genau wie beim LDR). Erwärm den NTC, indem du ihn zwischen Daumen und Zeigerfinger hälst. Beschreibe, wie sich die Spannung am NTC ändert, wenn dieser wärmer wird.
- Begründe, dass auch hier gilt:

$$\frac{R_{NTC}}{R_F} = \frac{U_{NTC}}{U_F}.$$

Begründe anhand der Formel, wie sich der Widerstand am NTC ändert, wenn dieser wärmer wird.

Hinweis: Der NTC wird wieder als analoger Sensor konfiguriert. Es handelt sich *nicht* um den Temperatursensor TMP36, der nach einem anderen Prinzip arbeitet.

Projekt 8: Digitales Thermometer

Baue ein digitales Thermometer, das die Lufttemperatur im Raum auf dem seriellen Monitor anzeigt!

Führe dazu mithilfe des rechts abgebildeten Ausschnitts aus einem Datenblatt eine Regression durch.

Das verlinkte Datenblatt ist evtl. nicht das korrekte Datenblatt zu dem NTC. Da die Bauteilnummer bei dem verwendeten Starter Kit nicht angegeben wird, ist eine Zuordnung leider nicht mehr möglich.

R/T No. 7003

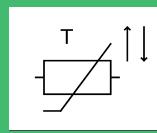
Widerstand bei 25°:

$$R_{25} = 10\text{ k}\Omega.$$

T (C)	R_T/R_{25}	(%/K)
5.0	2.3311	4.5
10.0	1.8684	4.4
15.0	1.5075	4.2
20.0	1.224	4.1
25.0	1.0000	4.0
30.0	0.82176	3.9

Heißleiter

Ein **Heißleiter**, kurz: **NTC** (*engl. negative temperature coefficient*), ist ein temperaturabhängiger Widerstand. Wenn es wärmer wird, wird der elektrische Widerstand des NTC kleiner; wenn es kälter wird, wird der elektrische Widerstand des NTC größer.



Anmerkung:

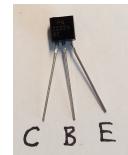
Es gibt auch Kaltleiter, kurz: **PTC** (*engl. positive temperature coefficient*), die ihren Widerstand verringern, wenn es kälter wird, und erhöhen, wenn es wärmer wird. Zusammen genommen bezeichnet man NTC's und PTC's auch als Thermistoren, also als temperaturabhängige Widerstände (*engl. thermally sensitive resistor*).

4.8. Schaltungen mit Transistoren steuern

Manche Projekte wie die **Straßenlampe** benötigt nur ein sehr simples Programm in der Form WENN - DANN - SONST. Für solche Fälle ist der Arduino eigentlich eine überdimensionierte Lösung - viel einfacher, jedenfalls in Bezug auf die Anzahl der Bauteile, ist die Umsetzung dieser Schaltung mithilfe eines Transistors. Dieser ist (unter anderem) ein elektronischer Schalter, mit dem sich das WENN - DANN - SONST - Verhalten ganz ohne Programm umsetzen lässt.

Frage: Wie verwendet man einen Transistor?

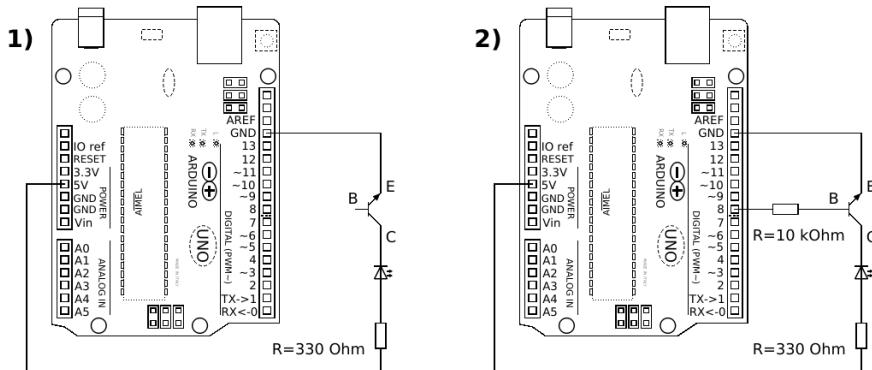
Ein Transistor hat drei Anschlüsse, die als Kollektor (**C** von engl. *collector*), Basis (**B**) und Emitter (**E**) bezeichnet werden. Wenn man auf die abgeflachte Seite des Transistors schaut, sind die drei Pins in der genannten Reihenfolge angeordnet. Im Folgenden geht es zunächst um deren Grundfunktionen.



Aufgabe 21: Digitalpins verstehen

Befolge die unten angegebenen Schritte und stelle Schlussfolgerungen über die Funktionsweise eines Transistors an. Erkenntst du Gemeinsamkeiten zu digitalen Pins?

- Baue die unten abgebildeten Schaltungen nacheinander auf. Spiele für die zweite Schaltung ein einfaches Blink-Programm auf den Arduino.

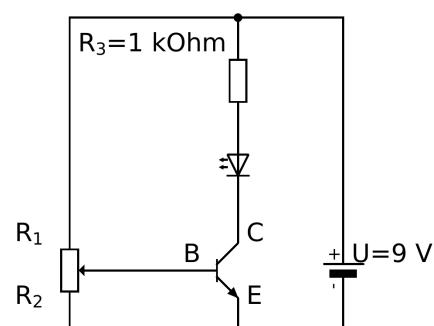


- Ersetze den $10\text{ k}\Omega$ Widerstand durch einen $100\text{ k}\Omega$ Widerstand.

Aufgabe 22: Vermessung

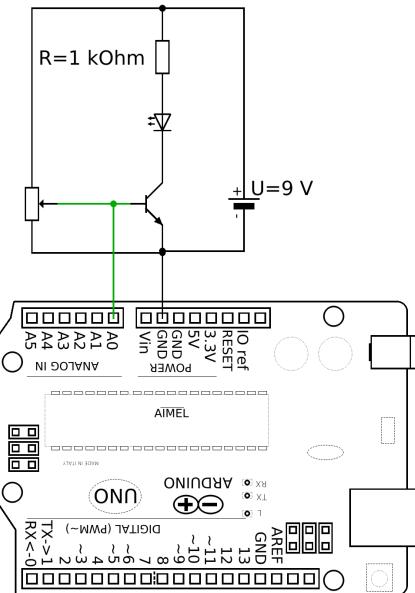
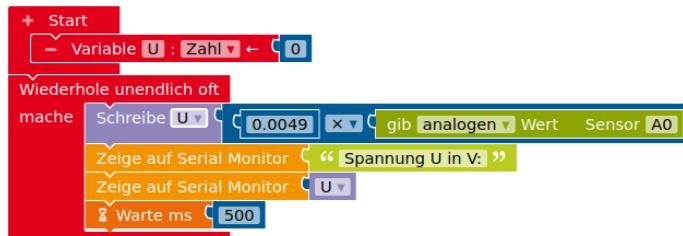
Um den Transistor zielgerichtet nutzen zu können, muss man die Spannung U_{BE} zwischen Basis und Emitter kennen, bei der der Transistor anfängt, durchzuschalten. Dazu dient die rechts abgebildete Schaltung.

Das Potentiometer lässt sich wieder in zwei Teilwiderstände R_1 und R_2 zerlegen, an denen die Spannung U_1 bzw. U_2 abfällt. Erkläre, wie der Widerstand R_2 und die Spannung U_{BE} zusammenhängen.



Baue die Schaltung nun auf. Um die Spannung U_{BE} messen zu können, wird ein Arduino ergänzt, der die Spannung in A0 ausliest und auf dem seriellen Monitor ausgibt.

Bestimme so die Grenzspannung U_{BE} , ab der der Transistor anfängt zu schalten, sodass die LED leuchtet.



Projekt 9: Straßenlampe ohne Mikrocontroller

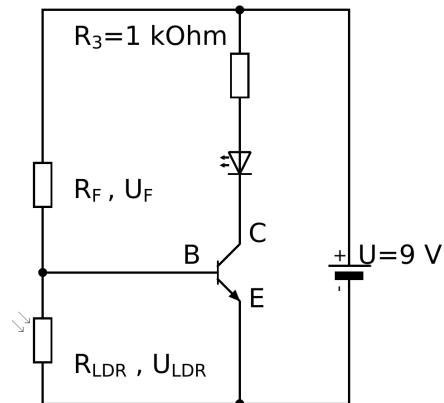
Nun kann die Straßenlampe auch ohne Arduino realisiert werden. Dazu wird statt des Potentiometers ein Spannungsteiler mit einem LDR und einem Festwiderstand aufgebaut (siehe Schaltplan rechts).

Bestimme die Größe des Festwiderstands R_F so, dass der Transistor schaltet, wenn die Größe des LDR $R_{LDR} = 7\text{ k}\Omega$ beträgt.

Tipps:

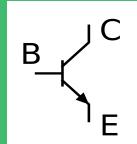
- Nutze die Spannungsteilerformel.
- Nutze $U_{LDR} = U_{BE}$. Ab welcher Spannung schaltet der Transistor?

Baue die Schaltung danach auf und teste sie.



Der Transistor

Ein Transistor hat drei Anschlüsse, die als Kollektor (**C** von engl. *collector*), Basis (**B**) und Emitter (**E**) bezeichnet werden. Wenn man auf die abgeflachte Seite des Transistors schaut, sind die drei Pins in der genannten Reihenfolge angeordnet.



Transistoren dienen als elektronische Schalter oder Verstärker (letzteres wird im Abschnitt 4.9 zu Motoren genutzt). Als Schalter lassen sie sich nutzen, weil die Strecke vom Kollektor zum Emitter ohne Weiteres nicht leitet. Erst wenn zwischen Basis und Emitter eine Spannung $U_{BE} \approx 0,6\text{ V}$ anliegt, fließt zwischen Basis und Emitter ein schwacher Strom, der den Transistor mit Elektronen flutet und es dadurch ermöglicht, dass zwischen Kollektor und Emitter ein starker Strom fließen kann.

Die Möglichkeit, mit Transistoren automatisierte Schalter herzustellen und dadurch Programme physikalisch abzubilden, macht Transistoren zur Grundlage von Mikrocontrollern und Computern und damit zu einer der wichtigsten Erfindungen des 20. Jahrhunderts. Schon auf dem kleinen integrierten Schaltkreis des Arduino, dem ATMEGA328P, sind Millionen von Transistoren verbaut. Wenn ein Digitalpin des Arduino auf HIGH gestellt wird, dann wird intern ein Transistor geschaltet.

Es gibt verschiedene Bauarten für Transistoren. Im hier verwendeten Starter Kit sind zwei npn-Transistoren (PN2222) vorhanden, was bedeutet, dass darin zwei n-dotierte und eine p-dotierte Schicht in der Mitte verbaut sind. npn-Transistoren müssen mit einer n-Schicht (normalerweise der Emitter) mit GND verbunden sein.

4.9. Elektromotor und Diode

Bei vielen Projekten soll sich etwas bewegen - dies lässt sich mit Elektromotoren realisieren. Die Ansteuerung eines Elektromotors erfordert auf der Hardware-Seite ein wenig Vorbereitung, denn aufgrund der hohen Ströme, die Elektromotoren ziehen, sollte man sie nicht direkt an den Digitalpins des Arduino anschließen. Für die Steuerung greift man meistens auf einen Transistor zurück; eine brauchbare Alternative ist aber auch das Relais. Beide Steuerungsmöglichkeiten werden im Folgenden erarbeitet.

Frage: Wie betreibt man einen Elektromotor am Arduino?

Aufgabe 23: *Motor und Diode - ein Paar, das zusammen gehört*

Erkläre die Funktion der Diode parallel zum Elektromotor in eigenen Worten. Lies dazu die Hintergrundinformationen zum Elektromotor und zur Diode.

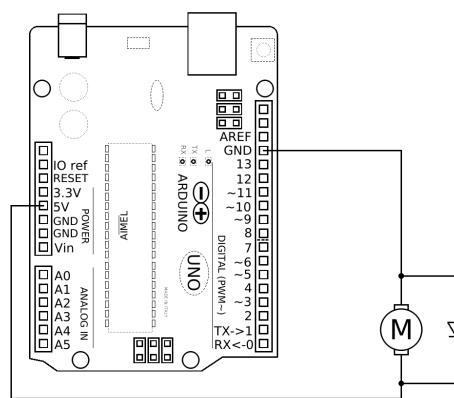
Baue die rechts abgebildete Schaltung zum Betrieb eines Gleichstrom-Elektromotors am Arduino auf.

! *Es ist sehr wichtig, dass die Diode richtig, also in Sperrrichtung, eingebaut wird, da sonst der Arduino zerstört werden könnte!*

Hintergrundinformationen:

Elektromotor

Ein **Elektromotor** besteht aus mehreren Spulen und Magneten. Wenn Strom durch die Spulen fließt, baut sich um die Spulen ein Magnetfeld auf, das mit dem Magnetfeld der eingebauten Magneten wechselwirkt (Anziehung/Abstoßung), sodass es zu einer Drehung des Motors kommt. Ein sogenannter Kommutator sorgt dafür, dass der Strom durch die Spulen ständig seine Richtung wechselt, sodass es immer wieder von Neuem zu Anziehung bzw. Abstoßung der Magnetfelder kommt und die Drehung nicht aufhört, solange eine Spannung anliegt.



B 4.11. Anschluss eines Gleichstrom-Elektromotors mit dem Arduino als Spannungsquelle.



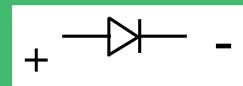
B 4.12. Gleichstrom-Elektromotor in real und als Schaltsymbol.

Wenn keine Spannung mehr am Motor anliegt, wird sich der Motor aufgrund seiner Trägheit immer noch ein wenig weiterdrehen. Durch das Drehen der Spulen im Magnetfeld der eingebauten

Permanentmagneten wird in den Spulen ein Strom induziert, dessen Richtung entgegengesetzt zur vorherigen Richtung ist. Dieser „falsch“ gerichtete Strom würde den Arduino zerstören. Aus diesem Grund schaltet man eine *Diode* parallel zum Motor.

Diode

Eine **Diode** ist wie ein elektrisches Ventil: Sie lässt den Strom nur in eine Richtung durch. Im Gegensatz zu Leuchtdioden wandeln „normale“ Dioden die elektrische Energie in Wärme um. In *Durchlassrichtung* wird der negative Pol (bzw. GND) mit der Seite verbunden, an der der Ring angebracht ist, und der positive Pol mit der anderen Seite.



B 4.13. Diode in real und als Schaltsymbol.

Die Diode wird jedoch *Sperrrichtung* eingebaut, also so, dass der Ring mit 5V und die andere Seite mit GND verbunden ist. Dadurch fließt im Normalbetrieb kein Strom durch die Diode. Wenn jedoch der entgegengerichtete Induktionsstrom des Motors auftritt, kann dieser durch die Diode abfließen, bis die verbleibende elektrische Energie vollständig in Wärme umgewandelt wurde.

🔍 Recherche: Verpolungsschutz

LEDs leuchten nicht, wenn man sie falsch herum anschließt. Andere Bauteile wie Elektrolytkondensatoren explodieren sogar, wenn man sie falsch herum anschließt. Um zu vermeiden, dass solche Schäden entstehen, wenn man eine Batterie falsch herum anschließt, werden in einigen Fällen Dioden genutzt. Recherchiere im [Elektronik-Kompendium](#),¹ wie dies funktioniert.

🔍 Recherche: Aufbau von Gleichstrom-Elektromotoren

Oben wurde die Funktionsweise von Gleichstrom-Elektromotoren bereits angedeutet. Recherchiere im Internet den genauen Aufbau und Ablauf der Drehbewegung.

¹<https://www.elektronik-kompendium.de/sites/slt/1206251.htm>

4.9.1. Elektromotor mit Transistor steuern

Der 5 V-Pin des Arduino liefert zwar in vielen Fällen genügend Strom für den Motor, jedoch lässt er sich nicht programmieren. Dazu lässt sich ein Transistor einbauen.

Frage: Wie steuert man einen Elektromotor mit einem Transistor am Arduino?

Die rechts abgebildete Schaltung zeigt, wie ein npn-Transistor eingebaut werden kann, um den Motor mithilfe von Digitalpin 9 zu schalten. Der Transistor lässt den Strom zwischen Emitter (E) und Kollektor (C) passieren, wenn die Spannung zwischen Basis (B) und Emitter (E) mehr als 0,7 V beträgt, anderenfalls sperrt er. Der Vorwiderstand mit $R = 330\Omega$ sorgt dafür, dass der Basisstrom nicht zu groß wird.

Es ist ratsam, die Basis mit einem PWM-Pin (gekennzeichnet durch \sim) zu verbinden, da sich dadurch die Geschwindigkeit des Motors steuern lässt.

Aufgabe 24:

Baue die oben abgebildete Schaltung auf und probiere die Steuerung des Motors mittels Pulsweitenmodulation aus.

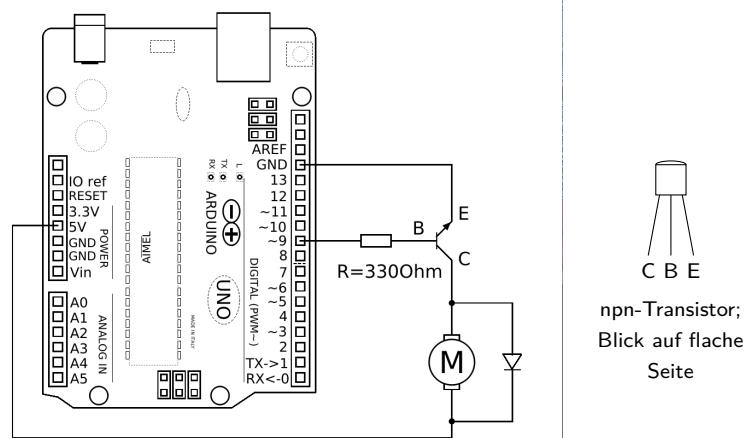
Schreibe analogen Wert Aktor M 135

Simuliere mit dem Motor eine konstant beschleunigende Bewegung (vgl. *Fading*), gefolgt von einer abrupten Bremsung.

Projekt 10: Automatischer Lüfter

Baue einen Lüfter, der anspringt, wenn die Temperatur größer als 30 °C wird. Probiere deine Schaltung durch Anfassen des NTC aus.

Erweiterung: Programmiere die Schaltung so, dass der Lüfter sich umso schneller dreht, je höher die Temperatur ist.

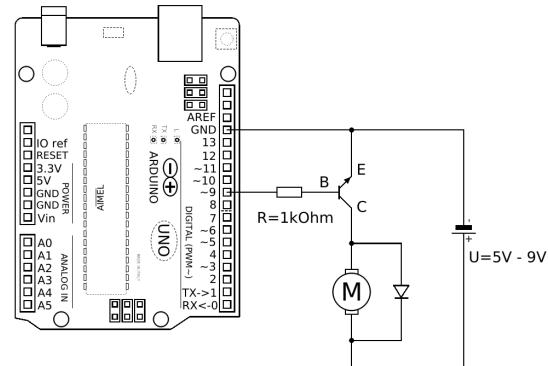


B 4.14. Anschluss eines Gleichstrom-Elektromotors am Arduino mit Steuerung über einen Transistor an Digitalpin 9.

Schaltung mit externer Spannungsquelle

Wenn der verwendete Elektromotor größer ist und mehr Strom zieht bzw. größere Spannungen benötigt, muss für den Elektromotor eine eigene Spannungsquelle verwendet werden, die genügend Spannung und Strom bieten kann. Der rechts abgebildete Schaltplan zeigt, wie der Aufbau dann vorzunehmen ist. Wichtig ist dabei, dass ein gemeinsames GND-Niveau hergestellt wird - vergleichbar einem „Normalnull“ für die Höhenangaben, hier allerdings als „Normalnull“ für das elektrische Potenzial.

Der Arduino kann über USB oder eine zweite Batterie mit Strom versorgt werden.



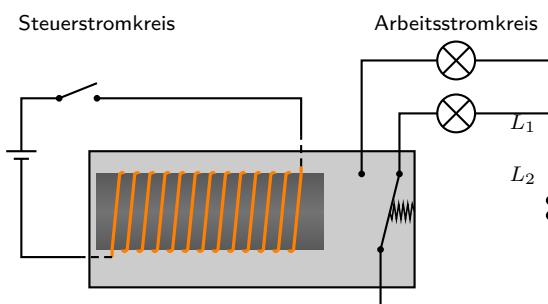
B 4.15. Anschluss eines Gleichstrom-Elektromotors am Arduino mit Steuerung über einen Transistor und mit externer Spannungsquelle für den Motor.

4.9.2. Elektromotor mit Relais steuern

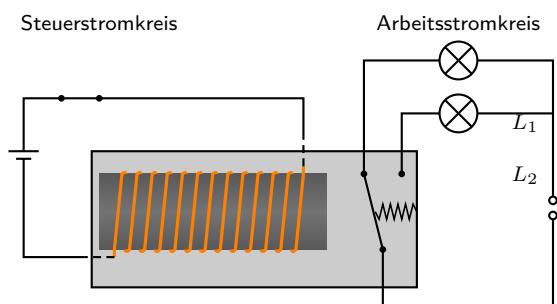
Wie oben zu sehen, muss der Arbeitsstromkreis mit dem Motor und der Steuerstromkreis mit dem Arduino bei Verwendung eines Transistors immer miteinander verbunden bleiben - auch bei sehr großen Stromstärken. Damit verbleibt immer eine gewisse Gefahr, dass eine Spannungsspitze auf den Arduino durchschlägt und ihn zerstört. Mit einem Relais lässt sich dieses Risiko vermeiden. Ein weiterer Vorteil ist, dass der Arbeitsstromkreis auch mit Wechselstrom betrieben werden kann, wenn ein Relais als Schalter genutzt wird.

Frage: Wie verwendet man ein Relais am Arduino?

Ein Relais (siehe unten, grau unterlegt) besteht im Wesentlichen aus einer Spule mit Eisenkern und einem Wechselschalter, an dem eine Feder angebracht ist.



B 4.16. Aufbau eines Relais (grau unterlegt) einschließlich der Stellung des Wechselschalters, wenn im Steuerstromkreis kein Strom fließt.



B 4.17. Aufbau eines Relais (grau unterlegt) einschließlich der Stellung des Wechselschalters, wenn im Steuerstromkreis Strom fließt.

Aufgabe 25: Physikalischer Hintergrund zum Relais

Erkläre die Stellung des Wechselschalters in Abhängigkeit des Steuerstromkreises in den beiden abgebildeten Situationen. Welche Lampe leuchtet?

Die Kontakte am Wechselschalter werden mit *NO* (*normally open*), *NC* (*normally closed*) und *C* (*common ground*) bezeichnet. Ordne die Bezeichnungen den Kontakten in der Skizze zu.

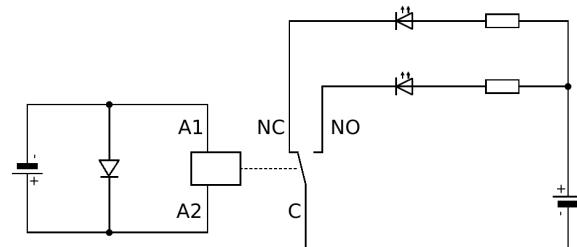
Aufgabe 26: Anschluss eines Relais

- a) Wie aus dem obigen Schaltplan ersichtlich wird, hat ein Relais fünf Anschlüsse, die jedoch nicht beschriftet sind. Suche im Internet nach „Datasheet <Gerätebezeichnung des Relais>“ (Bezeichnung vom Relais ablesen) und entnimm dem Datenblatt, welche Anschlüsse zum Steuer- bzw. Arbeitsstromkreis gehören.



Achtung: Auf dem Relais ist angegeben, dass damit bis zu 250 V Wechselspannung und 10 A geschaltet werden können. Das sollte man mit solch billigen Bastelmodulen aber **niemals machen!** Generell gilt: **Nur ausgebildete Fachleute sollten mit Spannungen von mehr als 24 V hantieren!**

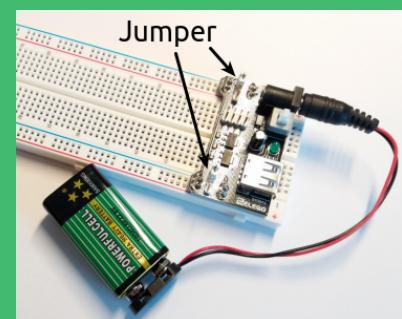
- b) Baue die Schaltung entsprechend des rechts abgebildeten Schaltplans auf. Nutze dazu das „Power Module“ auf dem Steckbrett (Erklärung unten). Probiere die Schaltung des Relais aus, indem du die Stromzufuhr der Spule unterbrichst und wieder herstelltst.



- c) Ordne in einer Skizze des Relais die Anschlüsse ihrer Bezeichnung (A1, A2, C, NO, NC) zu.

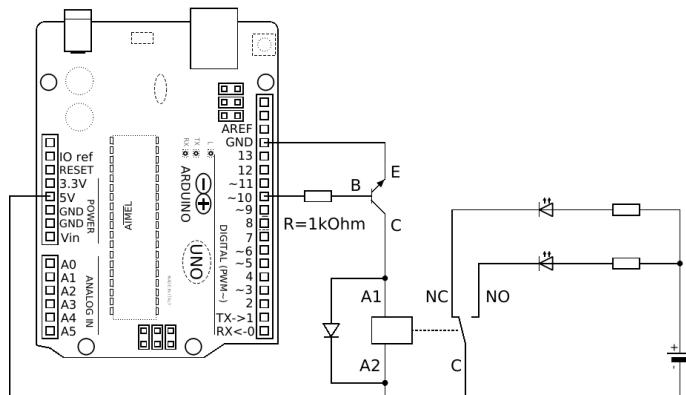
Das „Power Supply Module“

Das Power Supply Module dient zur Spannungsversorgung auf einem Steckbrett. Dazu kann eine Batterie mit 6,5 V bis 12 V oder ein USB-Kabel angeschlossen werden. Die Spannung wird auf dem Modul je nach Einstellung der *Jumper* auf 5 V oder 3,3 V heruntergeregt. Dazu verbindet man mithilfe der Jumper die Anschlüsse 5V und OFF bzw. 3.3 und OFF.



Die Spannung kann entlang der langen äußeren Leisten abgegriffen werden, wenn der Taster neben der Hohlbuchse gedrückt ist. Die Zuordnung zu Pluspol und Minuspol ist auf dem Power Supply Module mit + bzw. – markiert.

Aufgabe 27: Anschluss eines Relais am Arduino



npn-Transistor:
Blick auf flache Seite

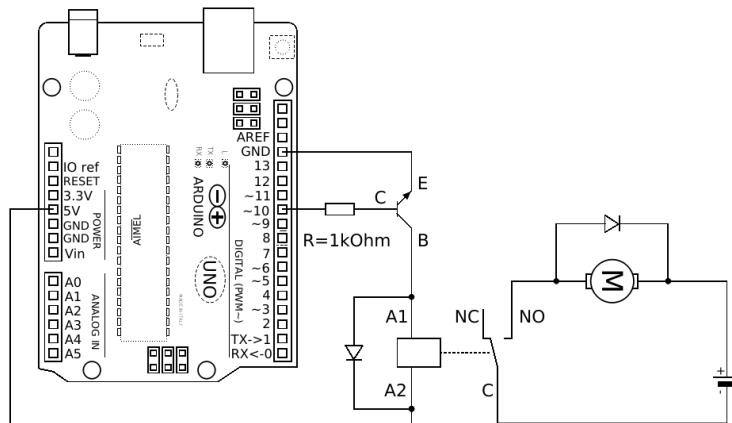
a) Erkläre die Funktion der in Sperrrichtung geschalteten Diode parallel zur Spule des Relais.

b) Erkläre die Funktion des Transistors.

Hinweis: Der 5V-Pin und der GND-Pin vertragen bis zu 200 mA. Die Digitalpins vertragen dagegen maximal 40 mA; normalerweise sollten 20 mA nicht überschritten werden (s. Pin Current Limitations).

c) Baue die Schaltung auf und teste sie mit einem Blink-Programm.

Projekt 11: Waschmaschinensteuerung



Baue die oben abgebildete Schaltung zur Steuerung eines Elektromotors mit einem Relais am Arduino auf. Achte auf die in Sperrrichtung geschaltete Diode parallel zum Motor.

Schließe dann drei Taster an (mit Widerstand! - vgl. Abschnitt 4.2).

Programmiere nun einen einfachen steuerbaren Waschmaschinenprototypen!

Dieser gibt solange auf dem seriellen Monitor die aktuell gesetzte Waschzeit aus, bis der mittlere Taster gedrückt wurde, was bedeutet, dass der Waschvorgang startet (der Motor dreht sich für die angegebene Zeit). Wenn der linke Taster gedrückt wird, wird die Waschzeit verringert (aber nicht niedriger als eine Sekunde). Wenn der rechte Taster gedrückt wird, wird die Waschzeit vergrößert

(aber nicht größer als 30 Sekunden). Nach dem Waschen fragt der Waschmaschinenprototyp wieder nach der Waschzeit.

🔍 Recherche: Anwendungen von Relais

Recherchiere einige Anwendungen von Relais. Einen guten Startpunkt bietet die [Seite von Leiphy-sik](#).

Aufgabe 28: Vergleich von Transistor und Relais

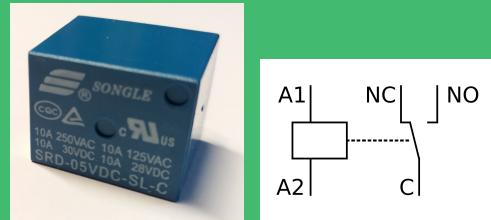
Transistoren und Relais erfüllen im Wesentlichen die gleiche Funktion: Sie sind elektronisch steuerbare Schalter, bei denen die zu steuernde Stromstärke größer sein kann als die Stromstärke im Steuerstromkreis. Dennoch gibt es einige Unterschiede, weshalb sie für unterschiedliche Aufgaben geeignet sind.

Vergleiche die Steuerung mit einem Transistor und mit einem Relais hinsichtlich der Vor- und Nachteile.

Relais

Relais sind elektronisch steuerbare Wechselschalter, bei denen Steuerstromkreis und Arbeitstromkreis komplett voneinander getrennt sind. Im Steuerstromkreis ist eine Spule, die ein Magnetfeld aufbaut, wenn der Strom eingeschaltet wird. Dadurch wird ein Wechselschalter im Arbeitsstromkreis angezogen, sodass er seine Position wechselt und einen anderen Teil des Arbeitsstromkreises anschaltet. Wenn der Strom durch die Spule im Steuerstromkreis abgeschaltet wird, baut sich das Magnetfeld ab und der Wechselschalter im Arbeitsstromkreis wird durch eine Feder zurück in die Standardstellung (NC) gezogen, sodass der erste Teil des Arbeitsstromkreises angeschaltet wird.

Die Anschlüsse der Spule werden in der Regel mit A1 und A2 bezeichnet; die Anschlüsse am Wechselschalter mit NO (*normally open*), NC (*normally closed*) und C (*common ground*) bezeichnet.



B 4.18. Relais in real und als Schaltsymbol.

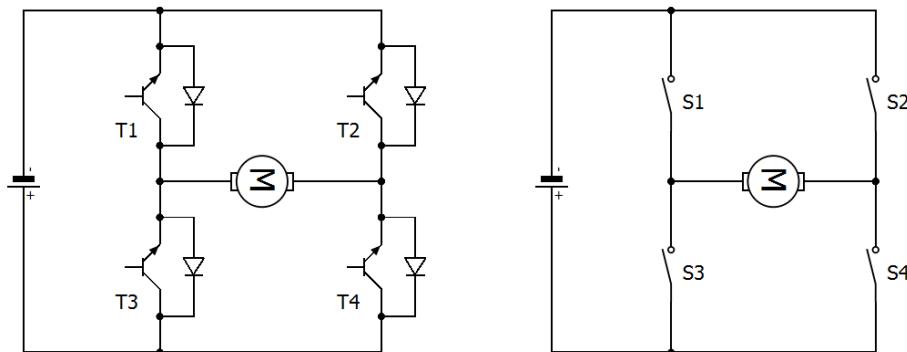
4.9.3. Elektromotoren mit dem Motortreiber-IC L293D steuern (inkl. Drehrichtung)

Die Steuerung von Motoren erfordert in den oben beschriebenen Fällen stets mehrere Bauteile und einige Überlegungen zum Aufbau der Schaltung. Außerdem kann dabei nicht die Drehrichtung geändert werden. Der integrierte Schaltkreis L293D vereinfacht den Aufbau der Schaltung für gleich zwei Motoren und ermöglicht zusätzlich die flexible Steuerung der Drehrichtung.

Frage: Wie steuert man einen Motor mit dem L293D?

Aufgabe 29: Aufbau des L293D - der Vierquadrantensteller

Um die Drehrichtung des Motors kontrollieren zu können, braucht man eine spezielle Anordnung von Transistoren, die als *H-Brücke* oder *Vierquadrantensteller* bezeichnet wird. Dieser Aufbau befindet sich auch im L293D.

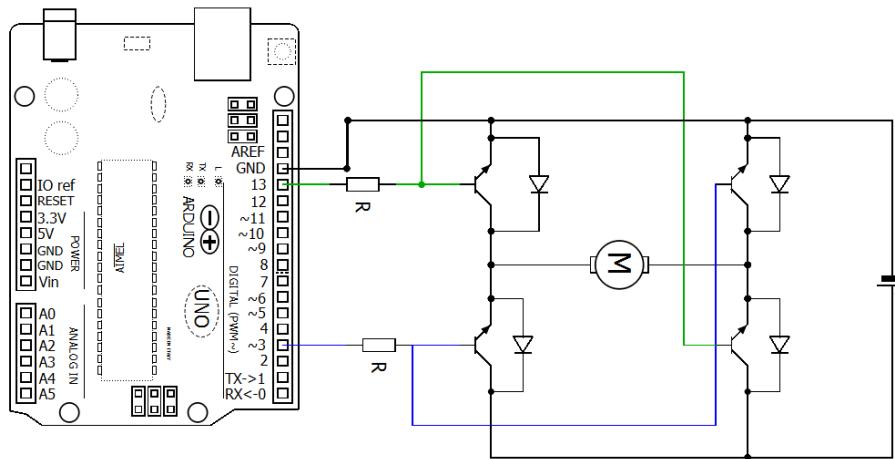


B 4.19. Vereinfachter Aufbau eines Vierquadrantenstellers mit Transistoren und zugehörigen Freilaufdioden (links) sowie die noch einmal vereinfachte Ersatzschaltung mit Schaltern.

- Die Drehrichtung des Motors hängt davon ab, in welcher Richtung der Strom durch den Motor fließt. Notiere, welche Transistoren / Schalter eingeschaltet und welche Transistoren / Schalter ausgeschaltet sein müssen, damit der Strom von links nach rechts durch den Motor fließt. Notiere danach die Kombination für die Stromrichtung von rechts nach links.
- Erkläre, wie sich der Motor mithilfe der vier Transistoren bzw. Schalter bremsen lässt.
- Welche Schaltkombinationen der Transistoren müssen unbedingt vermieden werden?

Hinweis: Die Freilaufdioden dienen dazu, die vom Motor induzierten Ströme abfließen zu lassen.

Da stets zwei Transistoren gemeinsam eingeschaltet werden müssen, könnten diese beim Anschluss an den Arduino über einen gemeinsamen Digitalpin gesteuert werden. Zudem ist es im Allgemeinen sinnvoll, für den Motor und den Arduino verschiedene Spannungsquellen zu verwenden, die über einen gemeinsamen GND-Anschluss geerdet werden, damit die möglicherweise hohen Ströme des Motors den Arduino nicht zerstören.



B 4.20. Steuerung eines Motors mit einem Vierquadrantensteller am Arduino.

Bei der oben dargestellten Schaltung muss jedoch immer noch darauf geachtet werden, dass nicht versehentlich alle vier Transistoren leitend geschaltet werden. Daher ist die Steuerung mit dem L293D noch ein wenig komplexer - die oben angestellten Überlegungen verdeutlichen aber gut den prinzipiellen Aufbau.

Der Motortreiber L293D

Der L293D ist ein integrierter Schaltkreis (*IC* von engl. *integrated circuit*), das heißt, in das schwarze Gehäuse sind Schaltkreise mit Transistoren, Widerständen, Dioden etc. integriert. Genauer gesagt, enthält der L293D zwei H-Brücken oder Vierquadrantensteller, die sich mit den Pins an beiden Seiten steuern lassen. Bei der Nummerierung der Pins ist darauf zu achten, dass die kleine Kerbe nach oben gehalten wird.



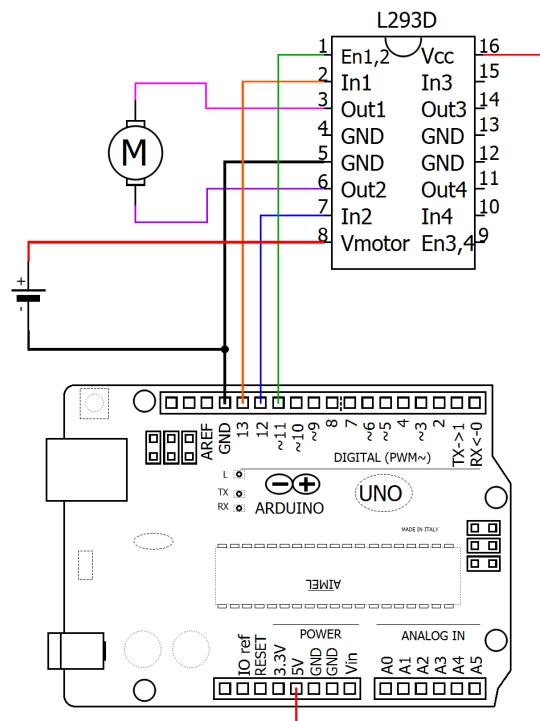
L293D	1	En1,2	Vcc	16
	2	In1	In3	15
	3	Out1	Out3	14
	4	GND	GND	13
	5	GND	GND	12
	6	Out2	Out4	11
	7	In2	In4	10
	8	Vmotor	En3,4	9

Im Folgenden wird die Belegung der Pins für die linke Seite beschrieben (vgl. Abbildung 4.21). Die Belegung auf der rechten Seite verläuft analog.

Der Motor wird an Pin 3 und 6 (Out1 und Out2) angeschlossen. Der jeweilige Zustand der Out-Pins kann über Pin 2 und 7 (In1 und In2) geregelt werden. Wenn an In1 der Zustand HIGH und an In2 LOW anliegt, wird das auf Out1 und Out2 übertragen, sodass durch den Motor ein Strom fließen kann. Diese Übertragung wird jedoch durch Pin 1 (En1,2 für enable pin 1, 2) gesteuert. Wenn an En1,2 HIGH anliegt, wird die Input-Konfiguration übertragen, bei LOW nicht. Durch ein PWM-Signal an En1,2 kann die Leistung des Motors entsprechend gedrosselt werden.

Die vier GND-Anschlüsse dienen zur Stromversorgung und zur Wärmeableitung, falls hohe Ströme auftreten. An Vmotor wird der Pluspol der Versorgungsspannung für den Motor angeschlossen; an

VCC der Logik-Pegel von 5V für die Schaltung des IC.

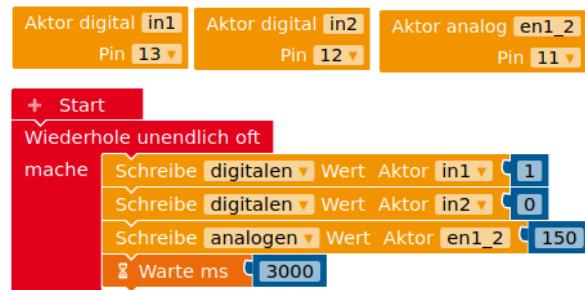


B 4.21. Steuerung eines Motors mit dem L293D.

Aufgabe 30: Betrieb des L293D

- Baue die oben beschriebene Schaltung auf. Nutze dazu das *Power Supply Module* (siehe S. 65).
- Experimentiere mit verschiedenen Input-Konfigurationen und PWM-Werten für den En1, 2-Pin.
- Halte die Wirkung auf den Motor tabellarisch fest. Hier genügt es, wenn für den En1, 2-Pin nur zwischen *ein / 1* und *aus / 0* unterschieden wird.

In1	In 2	En1,2	Wirkung
1	0	1	...



Recherche: Wie stark darf der L293D belastet werden?

Bei Motoren ist immer genau darauf zu achten, welche Stromstärken und Spannungen die verwendeten Bauteile aushalten. Suche nach dem Datenblatt (*data sheet*) des L293D und notiere die Maximalwerte zu Versorgungsspannung, Stromstärke und kurzfristige Spitzenstromstärke, die der IC aushält (*absolute maximum ratings*).

4.10. Vermischte Übungen

Aufgabe 31: Reihenschaltung

Eine rote LED soll an Pin 13 des Arduino betrieben werden. Durch die LED soll eine Stromstärke von 10 mA fließen, was bei einer Spannung von 2,1 V an der LED der Fall ist.

- Zeichne den zugehörigen Schaltplan.
- Berechne, wie groß der Vorwiderstand gewählt werden muss, damit diese Werte erreicht werden.

Aufgabe 32: Parallelschaltung

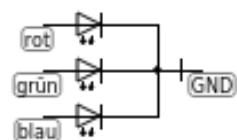
Drei grüne LEDs sollen parallel geschaltet an Pin 13 des Arduino angeschlossen und mit einem gemeinsamen Vorwiderstand betrieben werden. Die LEDs halten eine Stromstärke von maximal 20 mA bei einer Spannung von 3,3 V aus.

- Zeichne den zugehörigen Schaltplan.
- Ein Digitalpin am Arduino darf maximal mit einer Stromstärke von 40 mA belastet werden. Berechne, welche Stromstärke dann maximal durch die einzelnen LEDs fließen darf.
- Der Tabelle unten kannst du den zugehörigen Spannungswert an den LEDs entnehmen. Berechne, wie groß der gemeinsame Vorwiderstand der LEDs sein muss, damit die in b) berechnete Stromstärke eingehalten wird.

Spannung U	3,03 V	3,07 V	3,1 V	3,13 V	3,16 V	3,19 V
Stromstärke I	10 mA	11 mA	12 mA	13 mA	14 mA	15 mA

Aufgabe 33: Schaltung einer RGB-LED

Eine RGB-LED besteht aus drei einzelnen LEDs (rot, grün, blau), die jeweils über einen eigenen Digitalpin angesteuert werden (vgl. Schaltplan rechts). Am gemeinsamen GND-Anschluss soll ein gemeinsamer Vorwiderstand für alle LEDs angebracht werden, um die Stromstärke auf maximal 15 mA zu begrenzen. Die Spannung an den LEDs sollte dann 2,25 V nicht überschreiten.



B 4.22. Verschaltung der RGB-LED.

- Erkläre, welche Unterschiede zur Parallelschaltung von drei LEDs an *einem* Digitalpin zu beachten sind.
- Berechne, wie groß der gemeinsame Vorwiderstand mindestens sein muss.

4.11. Ausblick

Offene Fragen:

- Wie werden weitere Bauteile angeschlossen und im Programm angesprochen?
- Wie wird die Programmlogik physikalisch abgebildet?
- Wie funktionieren solche Bauteile wie LDR, NTC, Dioden, Transistoren?

Motivationsquellen

> [Laser-Game](#)

Ein kleines Spiel, das sich auf einfache Weise nachbauen lässt.

> [Arduino Garden Controller](#)

Gartenarbeit muss heute nicht mehr aufwendig sein: Mit einem Arduino lassen sich die Pflanzen automatisch bewässern, wenn die Erde nicht mehr feucht genug ist. Die erhobenen Daten lassen sich außerdem schön visualisieren.

> [Wetterstation von bitluni](#)

[Das Problem mit Wettervorhersagen \(Dr. Whatson, Youtube\)](#)

Selbst gebaute Wetterstationen sind beliebte Anfängerprojekte, bei denen meist ein WLAN-fähiger Mikrocontroller auf Basis des ESP8266 zum Einsatz kommt. Dieser lässt sich ebenfalls über die Arduino IDE programmieren. Wer etwas mehr Hintergrundwissen dazu haben will, schaut sich das Video von *Dr. Whatson* an, der außerdem das Projekt [SenseBox](#) vorstellt.

> „[Use the force or your brainwaves“ \(Youtube\)](#)

[„Use the force or your brainwaves“ \(Projektseite\)](#)

Der Schüler Imets Tamás hat es mithilfe mehrerer Arduinos geschafft, seine Gehirnwellen einzulesen und zu nutzen, um einen Roboter zu steuern!

> [Autonomes Auto](#)

Der Bastler hinter diesem Projekt hat einen Arduino-basierten Prototypen für ein autonomes Auto entworfen.

> [Pong-Bot](#)

Ein kleines witziges Spiel hat dieser Bastler mit einem Arduino automatisiert.

> [Snack-Automat](#)

Ein Arduino-basierter Snack-Automat!

5. Erweiterung des Werkzeugkastens: Bauteilkunde

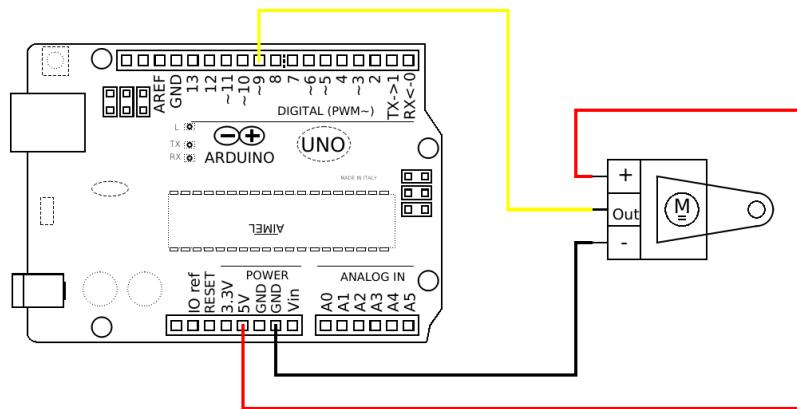
Die folgenden Seiten bieten jeweils eine Einführung in neue Bauteile, für die unterschiedliche Grundlagen aus dem Kapitel „Bausteine von Algorithmen“ benötigt werden. Das Kapitel „Elektrische Grundlagen“ ist zum tieferen Verständnis sicherlich hilfreich, aber für diese vorkonfigurierten Bauteile nicht unbedingt notwendig. Mit den hier vorgestellten Bauteilen lässt sich bereits eine Vielzahl an größeren Projekten umsetzen. Vielleicht hast du eine tolle Idee?

Projekte in diesem Kapitel:

> Schranke	74	> Wetterstation	88
> Sekundenzeiger	76	> Präzises Thermometer	89
> Alarmanlage	81	> Automatischer Scheibenwischer ..	91
> Automatische Tür	83	> Pulsmesser	93
> Joystick-Motor-Steuerung	85	> Einparkhilfe für ein Auto	94
> Fernsteuerung v. LED-Streifen ..	87	> Katzentür	96

5.1. Servo

Ein Servo ist in der Regel ein kleiner Elektromotor zusammen mit einer elektronischen Steuereinheit, die dazu dient, den Motor auf einen bestimmten Winkel einzustellen. Häufig wird beides zusammen als Servomotor bezeichnet. Angewendet werden Servos in vielen Bereichen - zum Beispiel im Modellbau.



B 5.1. Verschaltung eines Servo am Arduino.

Der Servo wird mit drei Anschlüssen an den Arduino angeschlossen:

- VCC (rot): Die Stromversorgung des Servo wird mit dem 5 V-Pin des Arduino verbunden. Dabei ist zu beachten, dass ein Servomotor relativ große Stromstärken „ziehen“ kann. Der 5 V-Pin des Arduino kann bis zu 200 mA ausgeben, bevor er durchbrennt. Das ist für den Servo genug. Ein normaler Digitalpin verträgt dagegen nur 40 mA, was deutlich zu wenig für den Servo ist. Die Stromversorgung des Servo kann also nicht über einen normalen Digitalpin sichergestellt werden.
- GND (schwarz/braun): Die Stromversorgung ist nur komplett, wenn auch das GND-Niveau auf das GND-Niveau des Arduino festgelegt wird.
- Signalleitung (gelb): Die Einstellung des Winkels erfolgt über ein Pulsweltenmodulation, allerdings wird diese von einer zusätzlichen Bibliothek bereitgestellt, sodass das gelbe Kabel mit jedem Digitalpin am Arduino verbunden werden kann.

Projekt 1: Schranke

Baue mit einem Servo eine Schranke, die auf Knopfdruck geöffnet und wieder geschlossen werden kann.

 Setze Servomotor SG90  S auf ° 

B 5.2. Die Servo-Steuerung erfolgt über Angabe eines Winkels zwischen 0° und 180°.

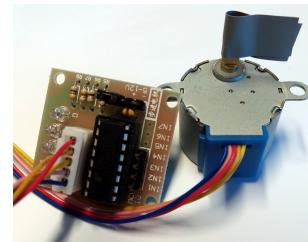
🔍 Recherche: Wie funktioniert die Steuerung eines Servos?

Der Winkel, auf den sich die Ausgangswelle des Servo drehen soll, wird über ein PWM-Signal geregelt. Recherchiere im Internet, wie dies realisiert wird und fasste es zusammen.

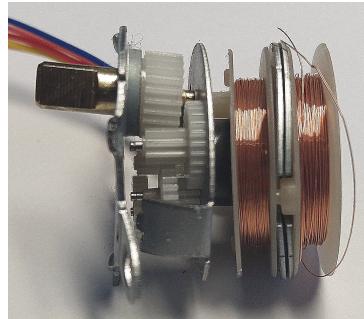
5.2. Schrittmotor

Während ein Servo darauf ausgelegt ist, einen Winkel möglichst präzise anzusteuern, dient ein Schrittmotor dazu, möglichst präzise Drehungen zu realisieren. Damit können zum Beispiel 3D-Drucker oder Roboterarme, aber auch DVD-Laufwerke sehr genau gesteuert werden.

Herzstück des Motors sind zwei Spulen, die jeweils in der Mitte durch eine 5 V-Spannungsversorgung in zwei Teile geteilt werden. Weil der Pol mit dem positiven 5 V-Potential dadurch festgelegt ist und nicht flexibel an beiden Enden angelegt werden kann, nennt man den Motor auch *unipolaren* Schrittmotor.



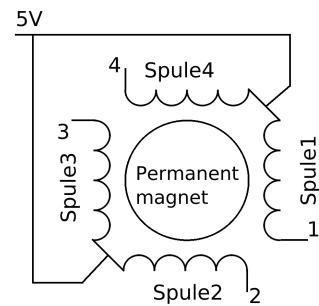
B 5.3. Schrittmotor mit Motorsteuerungschip ULN2003.



B 5.4. Innenansicht eines Schrittmotors.



B 5.5. Sicht auf Permanentmagnet mit Zahnrad in den Spulen.

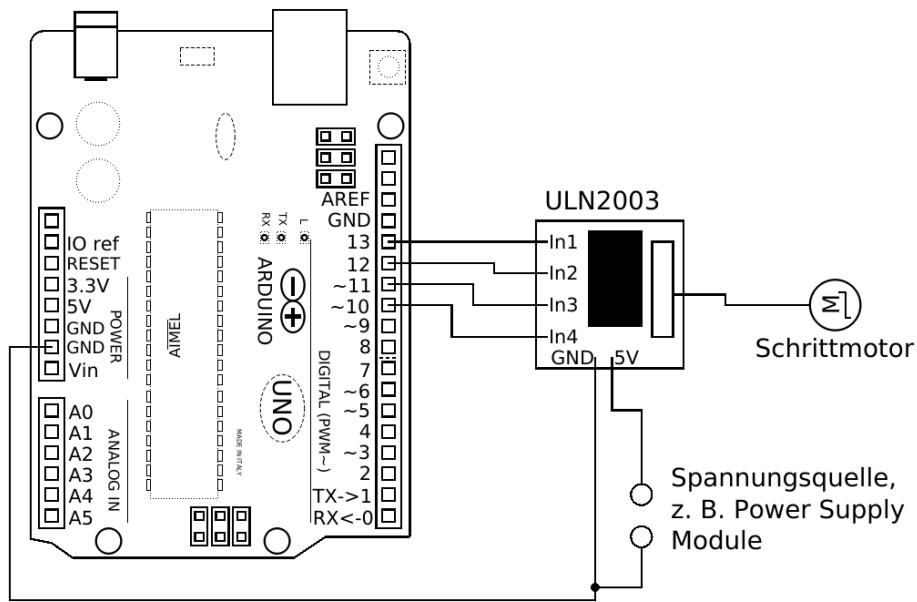


B 5.6. Schaltplan zu den Spulen.

In der Mitte der Spulen befindet sich ein Permanentmagnet, an dessen Ende ein Zahnrad angebracht ist, das wiederum weitere Zahnräder in Bewegung versetzt. Die Spulen werden nun abwechselnd an und aus geschaltet und wirken dabei als Elektromagnet. Durch die wechselnde Anziehung und Abstoßung des Permanentmagneten in der Mitte beginnt dieser, sich zu drehen. Für das Ein- und Ausschalten der Spulen wird ein Motortreiber genutzt, der den Arduino vor zu hohen und rückläufigen Strömen schützt (vgl. Abschnitt 4.9). Aufgrund der Stromaufnahme von ca. 240 mA ist es empfehlenswert, eine externe Spannungsversorgung am Motortreiber anzuschließen (zum Beispiel mit Hilfe des *Power Supply Module*, vgl. S. 65 - denke an die gemeinsame Erdung mit dem Arduino!). Da die Spulen auch beim Halten der Position dauerhaft unter Strom stehen müssen, ist der Stromverbrauch konstant hoch.

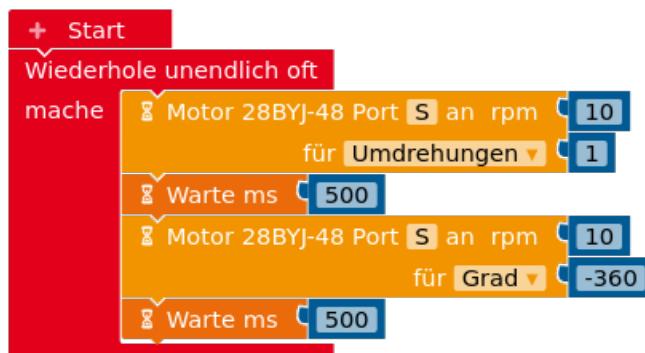
Die korrekte Reihenfolge des Ein- und Ausschaltens der Spulen ist bereits in Nepo implementiert, sodass die Steuerung sich einfach bewerkstelligen lässt. Es ist aber wichtig zu verstehen, dass jede Spuleneinstellung den Permanentmagneten in der Mitte dazu bringt, sich um einen kleinen Winkel zu drehen. Dieser Winkel wird auch Schritt genannt und dies gibt dem Schrittmotor seinen Namen. Dieser Schritt oder Winkel ist auch die kleinste Schrittweite, die der Motor ansteuern kann und gibt somit die Präzision des Motors an. Beim hier verwendeten Modell 28BYJ-48 beträgt der Schrittwinkel 5,625° (vgl. Datenblatt), woraus sich ergibt, dass der Motor 64 Schritte für eine Umdrehung benötigt. Durch die eingebauten Zahnräder wird die Drehung des Schafts aber weiter verlangsamt,

Funktionsweise eines Schrittmotors



B 5.7. Anschluss des Schrittmotors an den Motortreiber ULN2003 und den Arduino mit externer Spannungsquelle.

sodass ein Motorschritt nur einer Schaftdrehung von etwa $0,176^\circ$ (1/32 der Motorschrittweite) entspricht. Dies bedeutet, dass der Motor 2048 Schritte für eine Umdrehung braucht. Wenn du in den textbasierten Quellcode von Nepo schaust, nachdem du einen Schrittmotor konfiguriert hast, wirst du diese Zahl wiederfinden! In Nepo lässt sich einfach angeben, mit welcher Geschwindigkeit (in



Umdrehungen pro Minute, engl. *revolutions per minute, rpm*) und um wie viele Umdrehungen sich der Schaft drehen soll. Alternativ kann angegeben werden, um wie viel Grad sich der Schaft drehen soll.

Projekt 2: Sekundenzeiger

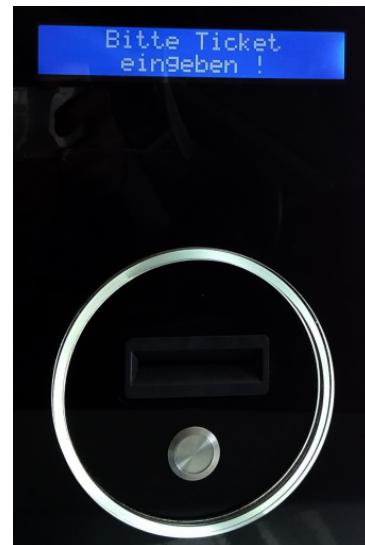
Programmiere einen möglichst präzisen Sekundenzeiger. Überprüfe ihn mit der Stoppuhr deines Smartphones.

Tipp: Nutze Klebeband, um eine kleine Fahne als Zeiger zu basteln.

5.3. Liquid Crystal Display (LCD)

In vielen Projekten genügt es nicht, Messwerte, Statusanzeigen oder Menüs über den seriellen Monitor am Computer anzeigen zu lassen - man benötigt stattdessen ein Display, das sich direkt an den Arduino anschließen und mit ihm verbauen lässt. Ein günstige Möglichkeit dafür bieten sogenannte **Liquid Crystal Displays (LCD)**, die man zum Beispiel in Kaffeemaschinen oder Parkautomaten finden kann (siehe B5.8). Modernere LCD werden in Laptops, Fernsehern und Tablets verbaut.

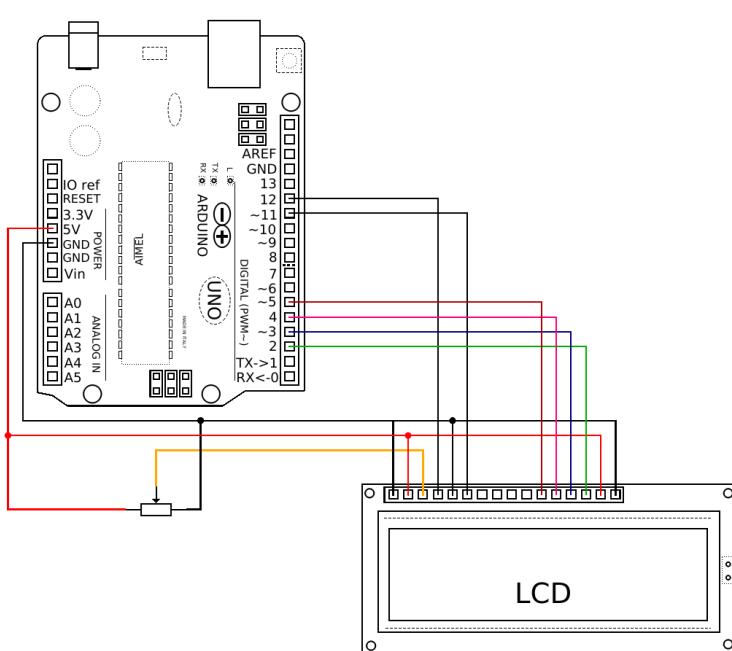
Um ein LC-Display anzusteuern, werden ziemlich viele Kabel benötigt. Daher gibt es neben den normalen LC-Displays häufig auch eine Variante, bei der ein I2C-Modul am LC-Display angebracht ist, was die benötigten Kabel deutlich reduziert. Im Folgenden werden beide Varianten beschrieben.



B 5.8. LC-Display an einem Parkhaus-Automaten.

LC-Display ohne I2C-Modul

Im Schaltplan sind die Kabel so mit dem Arduino verbunden, wie es die Standardkonfiguration für das LCD 1602 in Nepo angibt. Es empfiehlt sich, die zahlreichen 5V- und GND-Anschlüsse auf den Längsseiten des Steckbretts zu sammeln.

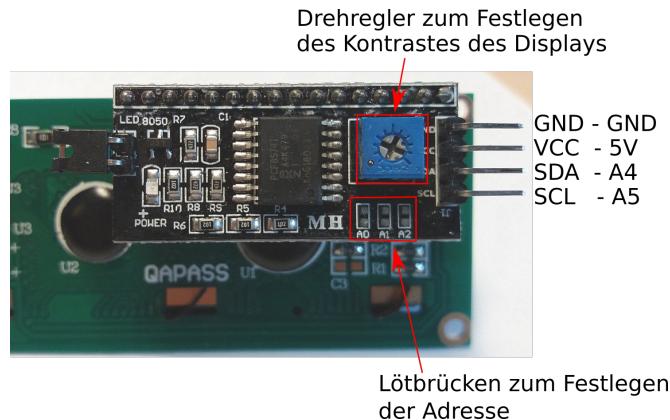


LCD	Arduino
VSS	GND
VDD	5V
V0	Drehregler (Mitte)
RS	12
RW	GND
E	11
D0 - D3	-
D4	5
D5	4
D6	3
D7	2
A	5V
K	GND

Die Anschlüsse A und K stehen für die Anode und Kathode der LEDs, die die Hintergrundbeleuchtung ausmachen. Mit dem Drehregler wird der Kontrast des Bildschirms eingestellt.

LCD mit I2C-Modul

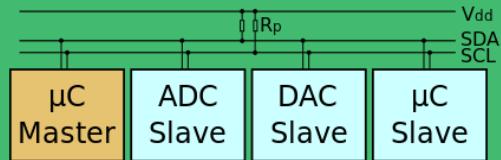
Die I2C-Schnittstelle erlaubt es, ein LC-Display mit nur vier Kabel anzusteuern. Der Anschluss an den Arduino erfolgt wie in der Abbildung dargestellt.



I2C oder IIC: Inter-Integrated Circuit

I2C steht für *Inter-Integrated-Circuit*. Dies ist ein sogenannter Datenbus, also ein System zur Übertragung von Daten zwischen mehreren Teilnehmern. Die Datenübertragung funktioniert über ein getaktetes An- und Ausstellen der Datenleitung, um die Daten in Binärform (1 und 0) zu übertragen. Neben der Spannungsversorgung (GND und VCC) wird dazu ein Kabel für die serielle Datenübertragung (SDA - Serial Data) und ein Kabel für die Abstimmung des Taktes (SCL - Serial Clock) benötigt.

Da auch mehrere I2C-kompatible Geräte an denselben Datenbus angeschlossen werden können, bekommt jedes Gerät eine Adresse, damit klar ist, welches Gerät die Daten bekommen soll. Die Adresse wird bei der Konfiguration als Hexadezimalzahl angegeben und kann prinzipiell zwischen 0 und 127 liegen. Typischerweise ist die voreingestellte Adresse 0x27. Dabei bedeutet 0x, dass die folgenden beiden Ziffern als Hexadezimalzahl zu interpretieren sind. Falls bereits ein anderes Gerät auf dem gleichen Datenbus dieselbe Adresse hat, kann die Adresse über die Lötbrücken verändert werden (siehe [bastelgarage.ch](#)).



B 5.9. I2C-Bus mit einem Master- und drei Slave-Geräten (Quelle: [Wikipedia](#), CC BY-SA 3.0, Urheber: [Colin Burnett](#)).

Aufgabe 1: Funktionstest

Schließe das LC-Display wie beschrieben an den Arduino an und erstelle ein Programm, das „Hello World!“ auf dem LC-Display anzeigt.

Hinweis: Falls alle Pixel weiß oder blau bleiben, kann es sein, dass der Drehregler falsch eingestellt ist. Drehe in diesem Fall an dem Drehregler, um den Kontrast zu verbessern. Für den Drehregler auf dem I2C-Modul (blauer Kasten) brauchst du einen Schraubenzieher o. ä.

Aufgabe 2: Messwertanzeige

In vielen Anwendungen soll auf dem LC-Display ein Messwert o. ä. angezeigt werden, der sich mit der Zeit ändern kann. Diese Anzeige soll aber schön formatiert sein.

Erstelle ein Programm, das alle drei Sekunden eine Zufallszahl z zwischen 0 und 200 erzeugt und auf dem Display folgende Anzeige ausgibt:

Messwert: z E

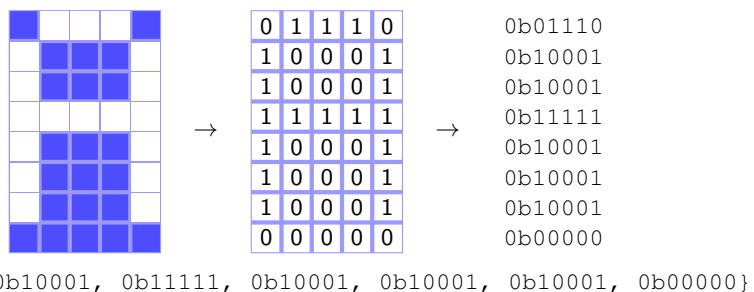
Hinweise:

- E soll für eine beliebige Einheit stehen.
 - Achte darauf, dass der vorherige Wert von z gelöscht wird (`lcd clear`).
 - Die Ausgabe der Zahl z soll immer rechtsbündig erfolgen, sodass zwischen den Einern von z und der Einheit genau ein Leerzeichen steht.
 - Du benötigst den Befehl `LCD set cursor (line <_> position <_>)`. Darin steht line für die Zeile, deren Nummer entweder 0 (oben) oder 1 (unten) ist. position steht für die Spalte, deren Nummer sich von 0 bis 15 erstrecken kann. (In der Informatik beginnt das Zählen stets mit der Null!)

Aufgabe 3: Codierung von Zeichen auf dem LCD

Für die Darstellung von Zeichen auf dem LCD muss im Hintergrund geklärt werden, welche Pixel an- und ausgestellt werden müssen. Dazu lohnt ein genauer Blick auf die Zellen des LCD.

Jede Zelle besteht aus 5×8 Pixeln, von denen manche weiß und manche blau sind. Wenn man für jedes Pixel eine 1 (weiß) oder eine 0 (blau) notiert, dann erhält man Bitfolgen (gekennzeichnet durch das vorstehende 0b), die sich in einer Reihe notieren lassen.



B 5.10. Codierung des Buchstabens A auf einem LC-Display.

Man könnte die Reihung von Bitfolgen auch als Reihung von Dezimalzahlen notieren und käme auf das gleiche Ergebnis. Das macht den Code zwar kürzer, jedoch leidet die Lesbarkeit des Codes deutlich.

Entwerfe einen Smiley und ein eigenes Symbol auf 5×8 Pixeln und notiere die zugehörige Reihung von Bitfolgen, die dieses Zeichen codiert.

Hinweis: Mit der textbasierten Arduino-IDE lassen sich nach dem oben beschriebenen Prinzip auch eigene Zeichen für das LC-Display codieren. Ein Beispiel findet sich unter Datei → Beispiele → LiquidCrystal → CustomCharacter.

5.4. Neigungsschalter

Mit sogenannten Neigungsschaltern (engl. *tilt switch*) lässt sich eine Neigung, aber auch eine Erschütterung oder der Beginn einer Beschleunigung messen. So lässt sich zum Beispiel feststellen, ob ein Gegenstand angehoben wird.



Ziel: Es soll eine Alarmanlage gebaut werden, die auslöst, wenn das Steckbrett angehoben wird.

B 5.11. Neigungsschalter.

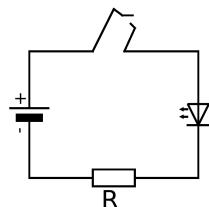
Aufgabe 1:

Die Abbildung rechts zeigt den Aufbau eines Neigungsschalters im geschlossenen und geöffneten Fall. Beschreibe den Aufbau des Schalters und erkläre, wie es in Abhängigkeit der Neigung des Neigungsschalters zum Leuchten der LED in Abbildung 5.12 kommt. Handelt es sich um einen digitalen oder analogen Sensor?



B 5.13. Neigungsschalter (geöffnet).

Neigungsschalter



B 5.14. Neigungsschalter (geschlossen).

B 5.12. Einfacher Aufbau zum Test eines Neigungsschalters ohne Arduino.

Programmierung: Der Neigungsschalter ist in Nepo nicht vorkonfiguriert, aber dies ist auch gar nicht nötig. Er lässt sich als digitaler Sensor konfigurieren. Der Rückgabewert eines digitalen Sensors ist in Nepo vom Typ *Zahl* statt vom Typ *Wahrheitswert*. Dabei bedeutet die Zahl 0 so viel wie *false* und die Zahl 1 bedeutet *true*.

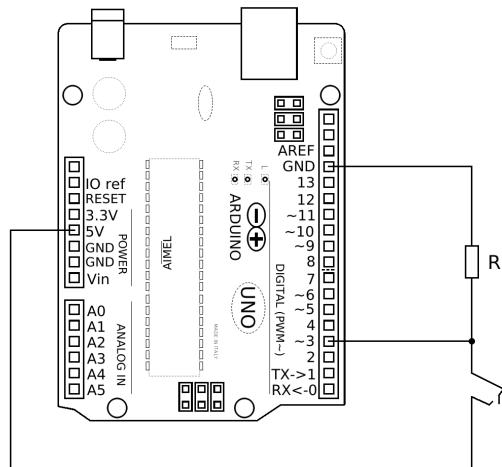


Projekt 1: Alarmanlage

Baue eine Alarmanlage, die auslöst, wenn das Steckbrett angehoben wird.

Hinweis: Wenn der Neigungsschalter wie rechts abgebildet am Arduino angeschlossen wird, kann sein Zustand in Digitalpin 3 ausgelesen werden (vgl. das Auslesen von Tastern).

Zusatz: Erkläre, warum es sinnvoll ist, den Piezo-Summer nicht so wie die LED in Abb. 5.12 direkt mit dem Neigungsschalter zu verbinden, sondern das Auslösen des Tons im Programm zu regeln.



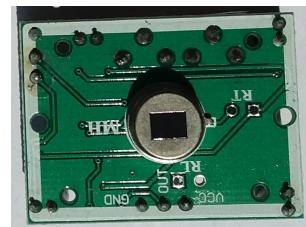
5.5. Bewegungsmelder

Bewegungsmelder wurden bereits in Abschnitt 3.3 erklärt und genutzt, um Wahrheitswerte einzuführen. In diesem Abschnitt steht das Bauteil im Vordergrund. Dazu werden noch einmal alle Informationen zusammengefasst und eine alternative Programmierung vorgestellt.

Bewegungsmelder verfügen über drei Pins, deren Beschriftung man lesen kann, wenn man die Kunststofflinse vorsichtig abzieht (*Vorsicht: Nach Abziehen der Linse nicht den Sensor berühren!*). VCC und GND dienen der Stromversorgung der elektronischen Komponenten und müssen mit 5 V und GND am Arduino verbunden werden.



Der mittlere OUT-Pin ist der Signal-Pin: Wenn eine Bewegung registriert wurde, liegt er auf einem hohen elektrischen Potential (HIGH); wenn keine Bewegung registriert wurde, liegt er auf einem niedrigen elektrischen Potential (LOW). Zum Einlesen des Signals wird dieser Pin mit einem digitalen Pin des Arduino verbunden, der dann digitaler Eingang heißt.



Hinter befinden sich zwei Drehregler („Potentiometer“), mit denen sich die Dauer des Bewegungssignals (links) und die Empfindlichkeit (rechts) einstellen lassen. Zusätzlich befindet sich auf der rechten Seite ein sogenannter Jumper, mit dem auf einfache Weise eine Steckverbindung zwischen benachbarten Pins hergestellt werden kann. Wenn sich der Jumper ganz außen befindet, dann bleibt das Bewegungssignal nach dem Erkennen einer Bewegung eine Weile aktiv und wird dann auf jeden Fall deaktiviert. Eine neue Bewegung kann erst nach einer gewissen Zeit wieder registriert werden. Wenn der Jumper hingegen leicht nach innen versetzt ist, bleibt das Bewegungssignal so lange erhalten, wie eine Bewegung erkannt wird (siehe [Funduino.de](#)).



Programmierung: Der Bewegungsmelder ist in Nepo bereits vorkonfiguriert und lässt sich damit einfach auslesen. Aufgrund der Ausgabe von HIGH und LOW bzw. true und false lässt er sich aber auch als digitaler Sensor konfigurieren. Der Rückgabewert eines digitalen Sensors ist in Nepo vom Typ Zahl statt vom Typ Wahrheitswert. Dabei bedeutet die Zahl 0 so viel wie false und die Zahl 1 bedeutet true.



B 5.15. Zwei Varianten zum Auslesen eines Bewegungsmelders. Oben wurde die Konfiguration als Bewegungsmelder vorgenommen, unten als digitaler Sensor.

Projekt 2: Automatische Tür

Baue und programmiere eine automatische Tür, die sich öffnet, wenn eine Bewegung registriert wird. Der Bewegungsmelder soll als digitaler Sensor konfiguriert werden. Experimentiere mit den Drehreglern, um die Empfindlichkeit und Dauer des Signals richtig einzustellen.

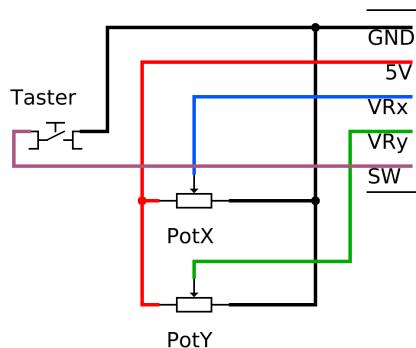
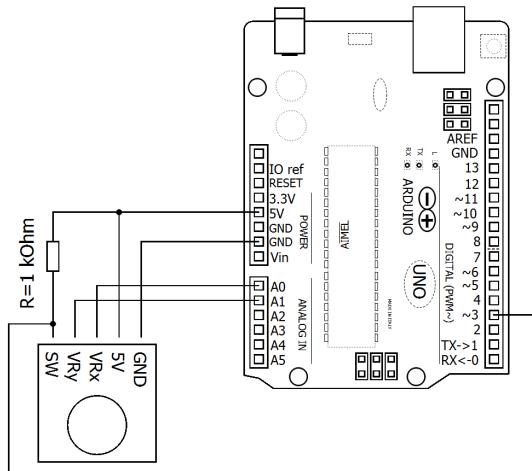
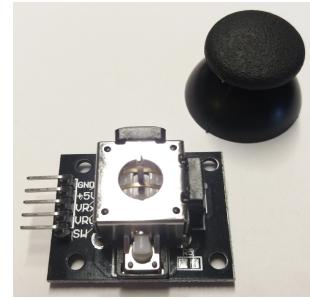
🔍 Recherche: Wie funktioniert eigentlich ein Bewegungsmelder?

Das zentrale Bauteil eines Bewegungsmelders ist ein sogenannter *Passiver Infrarot Sensor (PIR)*, auch *Pyroelektrischer Sensor*. Recherchiere im Internet, wie solche Sensoren funktionieren und fasse zusammen, wie es zur Registrierung einer Bewegung kommt.

5.6. Joystick

Joysticks werden bekanntermaßen für Spielecontroller oder auch zur Steuerung von Maschinen genutzt. Mit dem Arduino lassen sich einfache Versionen davon nachbauen.

Ein Joystick besteht im Wesentlichen aus zwei Potentiometern, die über einen gemeinsamen Hebel variiert werden können. Wie im Schaltbild zu sehen, teilen sich beide den 5V- und GND-Anschluss; der mittlere Anschluss muss natürlich jeweils einzeln ausgelesen werden. Zusätzlich wird durch Drücken des Joysticks ein angebrachter Taster gedrückt, dessen Status am SW-Pin ausgelesen werden kann (*sw* von engl. „switch“). Da das elektrische Potential am SW-Pin normalerweise schwankt, sollte ein *Pullup*-Widerstand mit $R = 1\text{ k}\Omega$ angebracht werden (vgl. Schaltbild).



B 5.17. Ersatzschaltplan für das Joystick-Modul.

B 5.16. Anschluss des Joystick-Moduls an den Arduino.

Programmierung: Das Joystick-Modul ist in Nepo nicht vorkonfiguriert. Die Bestandteile, also die zwei Potentiometer und der Taster, lassen sich aber einzeln konfigurieren. Dies geht wahlweise mit den vorkonfigurierten Potentiometer- und Taster-Blöcken oder als analoger und digitaler Sensor.

Aufgabe 2: Erste Experimente

- Bewege den Hebel des Joystick-Moduls und beobachte, wie sich dabei die Potentiometer an den Seiten mitbewegen. Bringe auch den Plastikdeckel an, drücke den Joystick herunter und beobachte dabei das Verhalten des Tasters.
- Schließe das Joystick-Modul wie oben beschrieben an den Arduino an. Lies die Werte der Potentiometer aus, während du sie bewegst. Notiere, welche Bewegungsrichtung die X-Richtung und welche die Y-Richtung darstellt. Notiere außerdem, welches der beiden Potentiometer (ggü. von Taster oder ggü. der Pins) für die X-Richtung bzw. Y-Richtung verantwortlich ist.

- c) Mit dem Pullup-Widerstand wird eine sogenannte Active-Low Schaltung aufgebaut. Teste die Funktionsweise des Tasters, indem du das elektrische Potential in D3 ausliest und beschreibe, was mit dem Begriff Active-Low gemeint ist.

Projekt 3: Joystick-Motor-Steuerung

Steuere mit dem Joystick-Modul einen Schrittmotor!

Achtung bei Verwendung von Motoren: Spätestens, wenn mehr als ein Motor am Arduino betrieben werden soll, muss eine externe Spannungsquelle genutzt werden, zum Beispiel durch Anschluss einer 9 V-Batterie an das Power-Supply-Module. Schaue dir dazu noch einmal Abschnitt 5.2 an.



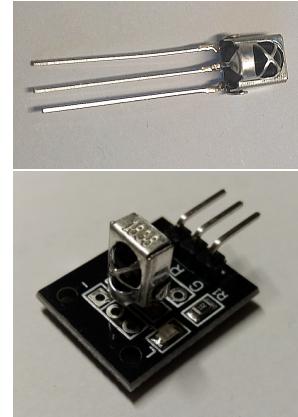
5.7. Infrarot-Sensor mit Fernbedienung

Jeder weiß, wie angenehm es ist, wenn man ein Gerät fernsteuern kann statt aufzustehen zu müssen, um die angebrachten Knöpfe zu drücken. Eine einfache Möglichkeit dafür bietet eine Infrarot(IR)-Fernbedienung.

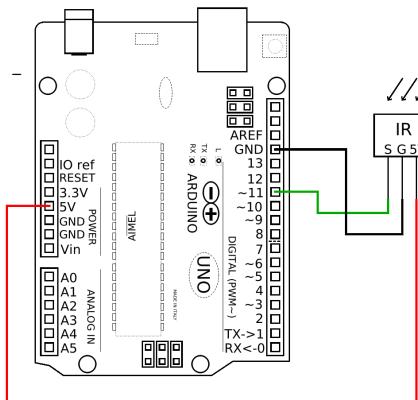
Wie am Namen zu erkennen, verwendet eine IR-Fernbedienung Infrarotstrahlen, die mit dem bloßen Auge nicht sichtbar sind. Hält man jedoch eine Digitalkamera, z. B. vom Smartphone, auf die Infrarot-LED der Fernbedienung und drückt eine Taste, dann kann man ein schnelles Aufblitzen erkennen. Am besten probierst du es selbst einmal aus oder schaust dir ein kurzes  [Video der IR-Strahlen](#) an. Das Aufblitzen zeigt, dass die Strahlen in einem bestimmten Rhythmus gesendet werden, aus dem sich entschlüsseln lässt, welche Taste gedrückt wurde.

Empfangen werden die Infrarotstrahlen von einem Infrarotsensor, der im Wesentlichen aus einer **Photodiode** besteht. Diese ist sehr ähnlich wie eine Leuchtdiode aufgebaut, allerdings funktioniert sie genau umgekehrt: Anhand eintreffender (infraroter) Lichtstrahlen wird ein Stromfluss ausgelöst, der dann registriert und weiter verarbeitet werden kann. Die Photodiode reagiert zwar am empfindlichsten auf infrarotes Licht bei einer Frequenz von 38 kHz, allerdings auch (weniger stark) auf sichtbares Licht. Um dieses sichtbare Licht, insbesondere die Umgebungshelligkeit, wegzufiltern, befindet sich die Photodiode in einer schwarzen Kunstharzsicht.

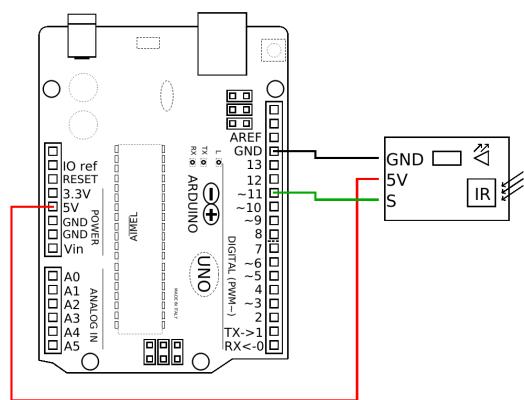
Häufig wird der Infrarotsensor zusammen mit einer LED und einem zugehörigen Vorwiderstand auf einer kleinen Platine ausgeliefert, damit das Empfangen eines Signals durch die LED angezeigt werden kann. Es sind aber auch Infrarotsensoren ohne weitere Anzeige im Umlauf.



B 5.18. Infrarotsensor (oben) und Infrarotsensor-Modul mit LED-Anzeige (unten).



B 5.19. Schaltplan zum Anschluss eines Infrarotsensors am Arduino.



B 5.20. Schaltplan zum Anschluss eines Infrarotsensor-moduls mit LED-Anzeige am Arduino.

Der Anschluss an den Arduino ist einfach: GND und 5V dienen wie üblich der Stromversorgung. Der Signal-Pin S muss mit einem beliebigen PWM-Pin (mit \sim) verbunden werden. Die Pin-Belegung

ist aber leider unterschiedlich; je nach dem, ob man ein Modul mit LED-Anzeige oder nur den Infrarotsensor anschließt.

Nachdem der IR-Sensor mit dem Arduino verbunden und in Nepo konfiguriert wurde, können die empfangenen Werte in Nepo abgefragt werden. Abbildung 5.21 zeigt ein einfaches Beispiel, wie mit den Tasten 0 und 1 auf einer Infrarot-Fernbedienung die Board-LED des Arduino an- und ausgestellt werden kann. Wenn die Taste mehrere Aktionen auslösen soll (wie die Ausgabe des Codes auf dem seriellen Monitor und das Anschalten der LED), dann muss der Wert in einer Variable gespeichert werden.



B 5.21. Einfaches Beispielprogramm zur Verwendung einer IR-Fernbedienung.

Aufgabe 3: Codes kennen lernen

- Übertrage das oben abgebildete Programm auf den Arduino und probiere es aus.
- Erstelle eine Tabelle, in der du den Zahlencode für jede Taste festhältst. Probiere auch aus, was passiert, wenn du die Tasten länger gedrückt hältst.



Lichterkette-Beispielvideo

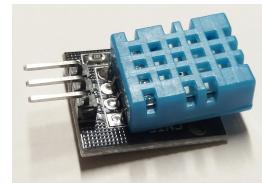
Projekt 4: Fernsteuerung eines LED-Streifens

In vielen Bereichen werden LED-Streifen genutzt, um einen Raum mit passendem, indirektem Licht auszustatten. Die meisten LED-Streifen lassen sich über eine kleine Infrarot-Fernbedienung steuern, wodurch sich die Farbe, aber auch der Modus einstellen lässt - zum Beispiel eine einzelne Farbe, Fading, Strobe, ...

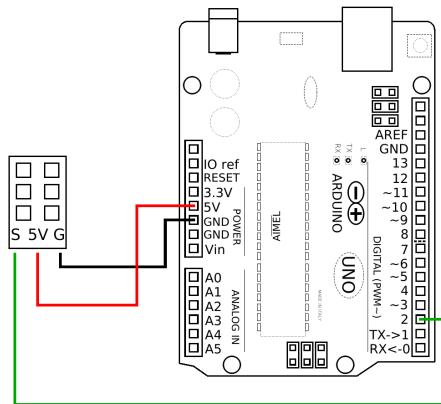
Das Prinzip lässt sich auf ein Lauflicht übertragen. Baue ein Lauflicht und programmiere verschiedene Lauflicht-Modi, die sich mit der Fernbedienung einstellen lassen.

5.8. Temperatur- und Luftfeuchtigkeitssensor DHT-11

Bei vielen Umweltmessungen interessiert nicht nur die Temperatur, sondern auch die Luftfeuchtigkeit. Der Sensor DHT-11 ist ein einfaches, kleines Bauteil, mit dem sich beides messen lässt.



Der DHT-11 verfügt über drei Pins - 5V und GND dienen der Stromversorgung, während das Signal zu den Messdaten über den Signalpin ausgegeben wird. Für die Temperaturmessung ist auf dem DHT-11 ein NTC verbaut.



Das Auslesen des Signalpins ist einfach, weil der DHT-11 in der Konfiguration bereits implementiert wurde, sodass man nur noch den Signalpin angeben muss. Das Signal wird dann automatisch analysiert und decodiert, um anhand der entsprechenden Befehle auf die übermittelte Temperatur bzw. Luftfeuchtigkeit zuzugreifen. Die Größe kann über das Drop-Down-Menü ausgewählt werden.

gib Temperatur ▾ ° Luftfeuchtigkeitssensor DHT11 L2

gib Luftfeuchtigkeit ▾ % Luftfeuchtigkeitssensor DHT11 L2

Projekt 5: Wetterstation

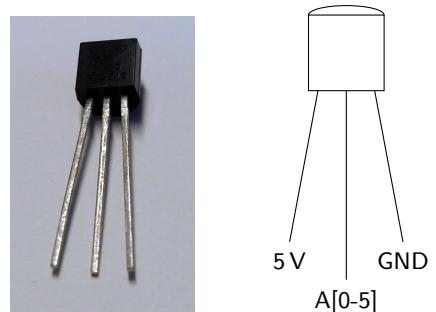
Baue eine kleine Wetterstation, die alle zehn Minuten Temperatur und Luftfeuchtigkeit misst und auf dem seriellen Monitor ausgibt.

⌚ Recherche: Wie wird die Luftfeuchtigkeit gemessen?

Mit dem DHT-11 lässt sich die relative Luftfeuchtigkeit bestimmen. Recherchiere, was darunter zu verstehen ist, und wie diese durch ein elektrisches Bauteil gemessen wird.

5.9. Temperatursensor TMP36

Bei deinen Temperaturmessungen mit einem NTC oder dem DHT-11 (in dem auch ein NTC verbaut ist), hast du vielleicht festgestellt, dass die Bauteile nicht besonders genau arbeiten. Für professionellere Anwendungen benötigt man eine wesentlich höhere Genauigkeit. Hier kann der TMP36 helfen: Er hat eine Genauigkeit von $\pm 1^\circ\text{C}$ und kann Temperaturen in einem Bereich von -40°C bis 125°C zuverlässig messen. Die Messung der Temperatur erfolgt über die Messung einer temperaturabhängigen Spannung. Bei 0°C beträgt die Spannung 500 mV (ein sogenannter *Offset*).



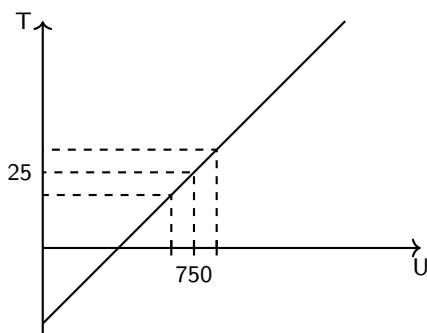
B 5.22. Temperatursensor TMP36 (links) mit Pinbelegung (rechts, Blick auf flache Seite).
Achtung: Verwechslungsgefahr mit Transistor (Aufschrift beachten)!

Ein weiterer Vorteil des TMP36 ist, dass die Abhängigkeit von ausgegebener Spannung und Temperatur linear verläuft: Eine Temperaturänderung von 1°C entspricht immer einer Spannungsänderung von 10 mV.

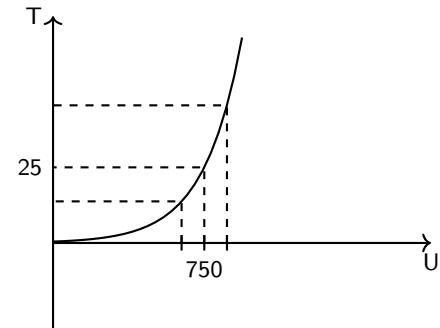
Aufgabe 4: Vergleich von Kennlinien

Ein linearer Zusammenhang zwischen der gemessenen Größe (oft eine Spannung) und der gesuchten Größe (hier die Temperatur) gilt als vorteilhaft, ein exponentieller Zusammenhang dagegen als nachteilig. Begründe anhand der unten abgebildeten Skizzen von Kennlinien, warum ein linearer Zusammenhang besser ist.

Beachte: Jede Messung ist mit einem Fehler versehen!



B 5.23. Lineare Kennlinie.



B 5.24. Exponentielle Kennlinie.

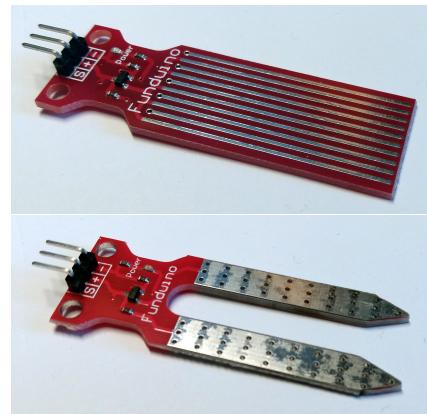
Projekt 6: Präzises Thermometer

- Baue ein präzises Thermometer mit den vorkonfigurierten Nepo-Blöcken für den TMP36.
- Baue ein präzises Thermometer und konfiguriere den TMP36 als analogen Sensor.

Tipp: Zur Berechnung der Temperatur musst du zuerst den eingelesenen Analogwert in eine Spannung umrechnen. Um aus der Spannung die Temperatur zu berechnen, musst du die oben angegebenen Informationen noch einmal genau lesen. Zur Kontrolle: Die maximal mögliche Spannung am Signalpin beträgt 2 V, was einer Temperatur von 150°C entspricht.

5.10. Tropfensor und Feuchtigkeitssensor

Mit einem Tropfensor lässt sich die Feuchtigkeit auf dem Sensorblatt messen. Solche Sensoren werden zum Beispiel in Windschutzscheiben von Autos eingesetzt, um die Scheibenwischer und ihre Geschwindigkeit zu steuern. Feuchtigkeitssensoren funktionieren im Wesentlichen genauso, allerdings sind die Kontakte dabei so weit auseinander, dass durch Tropfen noch kein Stromfluss entsteht, sondern erst durch zum Beispiel die feuchte Erde eines Blumentopfes, der automatisch bewässert werden soll. Der Anschluss an den Arduino ist in beiden Fällen einfach (siehe rechts). Am Signalpin wird eine Spannung (als Analogwert im analogen Eingang) gemessen, die im trockenen Zustand bei 0 V (0) liegt und bis zu 5 V (1023) steigen kann.



B 5.25. Tropfensor (oben) und Feuchtigkeitssensor (unten). Anschluss am Arduino: S > A[0-5] + > 5V - > GND.

Aufgabe 5: Theorie: Wie kann man Feuchtigkeit messen?

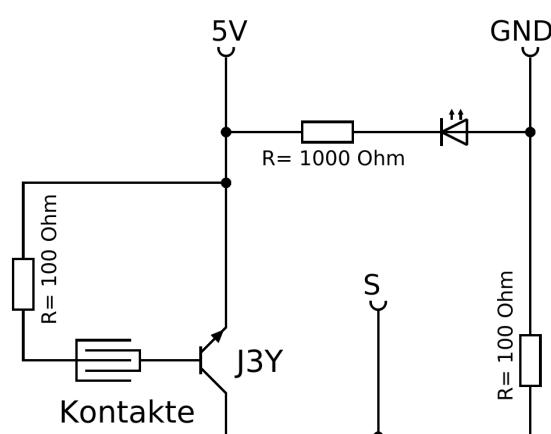
Das grundlegende Prinzip eines Tropfensors/Feuchtigkeitssensors ist einfach: Je nasser/feuchter es ist, desto besser der Strom durch die feuchte Stelle fließen. Der Arduino kann aber nur eine Spannung messen (als Analogwert) - wie bekommt man diese Umwandlung hin?

Erkläre anhand des unten abgebildeten Schaltplans eines Tropfensors, wie die Feuchtigkeit auf den Kontakten des Sensorblatts am Signalpin S (analoger Eingang zur Spannungsmessung) gemessen werden kann.

Tipp: Zentral ist der Transistor J3Y, der mit seinen drei Kontakten auf dem Sensor gut erkennbar ist. Die Status-LED mit Vorwiderstand kann dagegen vernachlässigt werden.

C
J3Y
E
B

Pinbelegung
des Transistors
J3Y auf dem
Tropfen- und
Feuchtigkeits-
sensor.



B 5.26. Schaltplan eines Tropfensors.

Projekt 7: Automatischer Scheibenwischer

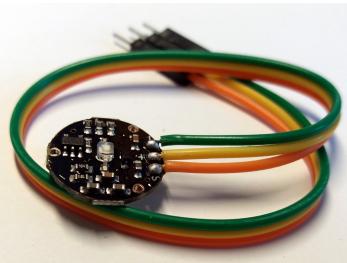
Baue einen Scheibenwischer, der automatisch startet, wenn Feuchtigkeit registriert wird. Je nach Feuchtigkeitslevel soll eine von drei Geschwindigkeiten ausgewählt werden.

Hinweis: Du kannst die vorkonfigurierten Blöcke von Nepo für den Tropfensor verwenden oder einen analogen Sensor konfigurieren, dessen Analogwert du selbst in einen Prozentwert umrechnest (0 entspricht 0 %, 1023 entspricht 100 %).

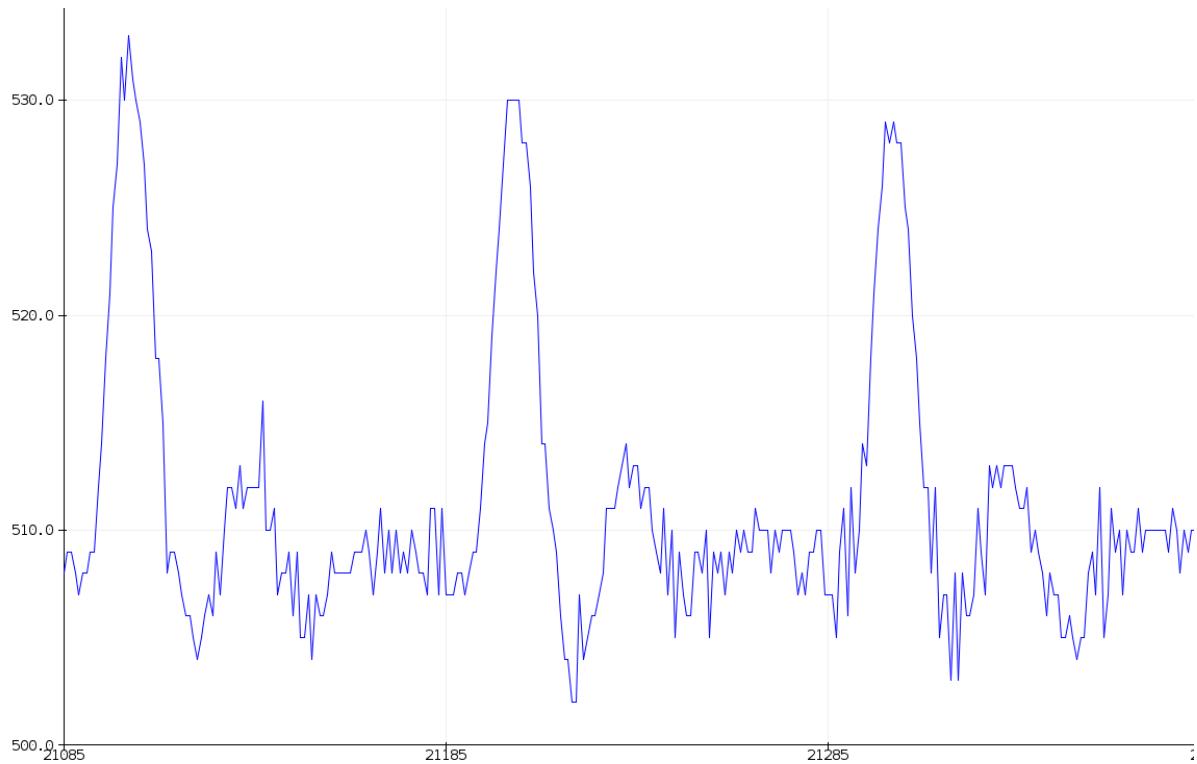
5.11. Pulssensor

Aktivitätstracker mit Pulsmessung liegen voll im Trend – aber wie funktioniert so ein Pulssensor eigentlich? Das lässt sich am einfachsten verstehen, wenn man selber einen nachbaut. Weitere Anwendungsmöglichkeiten wären übrigens Lügen-/Angstdetektoren, Schlafanalyse oder ein Alarmsystem für Risikopatienten.

Der Anschluss an den Arduino ist einfach (siehe rechts). Am Signalpin liegt eine Spannung an, die sich im Rhythmus des Herzschlags verändert und am analogen Eingang des Arduino gemessen werden kann. Dies lässt sich mit dem seriellen Plotter der Arduino IDE veranschaulichen (siehe Abb. 5.28). Man erkennt, dass die gemessenen Analogwerte zwischen ca. 500 und ca. 535, also in einem relativ kleinen Bereich, schwanken (35 bzw. 0,171 V). Als Kriterium für einen Herzschlag könnte man festlegen, dass der Analogwert über 520 liegt. Diese Werte sind jedoch wenig stabil und schwanken je nach Person und Umgebung! Man sollte unbedingt darauf achten, dass die Haut nicht verschwitzt ist und keine Bauteile auf dem Sensor berührt werden (insbesondere auf der Rückseite), damit die Ergebnisse einigermaßen zuverlässig sind. Wenn sich auf dem Arm keine brauchbaren Werte einstellen, lohnt sich ein Versuch auf dem Ringfinger oder dem Ohrläppchen.



B 5.27. Rückseite eines Pulssensors. Anschluss am Arduino:
S > A[0-5] + > 5 V - > GND.

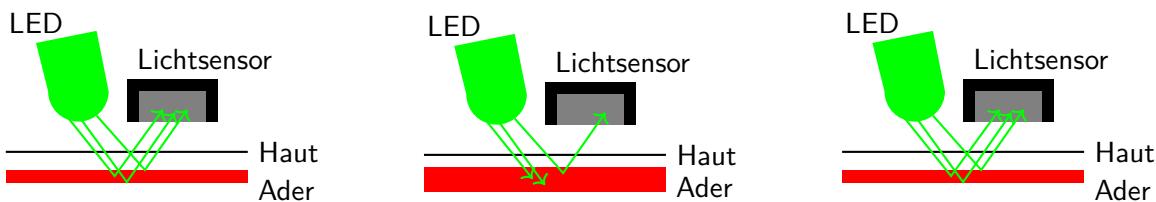


B 5.28. Visualisierung von gemessenen Analogwerten zur Bestimmung des Pulses.

Der Pulssensor kann mit den vorkonfigurierten Blöcken von Nepo oder direkt als analoger Sensor eingelesen werden. Im Hintergrund passiert das Gleiche.

Aufgabe 6: Theorie: Wie wird der Puls gemessen?

Für die Messung des Pulses ist die grüne LED und ein Lichtsensor zentral. Erkläre anhand der Abbildungen unten das Prinzip der optischen Pulsmessung.



B 5.29. Schematische Darstellung eines Pulses und seiner optischen Vermessung.

Projekt 8: Pulsmesser

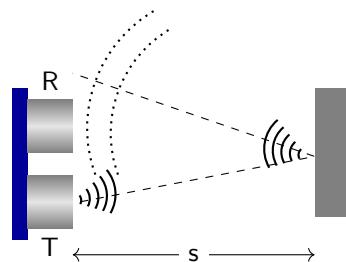
Baue einen Pulsmesser, der anhand der Messwerte von 10 Sekunden den Puls (Herzschläge pro Minute) berechnet.

Hinweis: Es kann nötig sein, sich die Werte von einem seriellen Plotter visualisieren zu lassen, um einen Eindruck vom Wertebereich und von der Grenze für die Herzschlagerkennung zu bekommen. Lasse dir ggf. von deinem Lehrer zeigen, wie man die Arduino IDE dafür einstellen muss.

5.12. Ultraschallsensor

Ultraschallsensoren ermöglichen die berührungslose Messung eines Abstands zwischen dem Sensor und dem nächstgelegenen Gegenstand. Dies macht sie zu einer interessanten Ausrüstung für Staubsaugerroboter, die nicht gegen die Wand fahren sollen, oder Einparkhilfen im Auto, die dem Fahrer anzeigen sollen, wie viel Platz er noch hat.

Die wichtigsten Bestandteile des Ultraschallsensors sind der „Transducer“ (**T**) und der „Receiver“ (**R**). Der Transducer ist praktisch ein Lautsprecher, der für uns nicht hörbare Schallwellen aussendet. Der Receiver entspricht einem Mikrofon für Schallwellen. Die Schallwellen werden also vom Transducer ausgesendet, an einem Hindernis reflektiert und vom Receiver empfangen.



Der Ultraschallsensor verfügt über vier Pins. GND und VCC (5V) sind wie üblich zu belegen und dienen der Energieversorgung. Der Trigger-Pin dient dazu, einen Ultraschallpuls auszusenden - wird er für 10 Mikrosekunden auf ein HIGH-Potential gebracht, wird der Ultraschallpuls getriggert. Wenn dies geschieht, wird der Echo-Pin von der Elektronik des Sensors auf ein HIGH-Potential gebracht, das so lange anhält, bis der Receiver die reflektierte Schallwelle empfängt.

Die Zeit, die der Echo-Pin auf HIGH liegt, gibt also an, wie lange der Schall braucht, um vom Sensor zum Hindernis und zurück zu gelangen. Mit Hilfe der Schallgeschwindigkeit wird dann berechnet, welche Strecke der Schall zurückgelegt hat. Diese Berechnungen werden praktischerweise von den vorkonfigurierten Nepo-Blöcken übernommen, sodass man direkt die Strecke erhält.

Projekt 9: Einparkhilfe für ein Auto

Baue eine Einparkhilfe für ein Auto, die umso schneller piepst, je näher man dem Hindernis kommt. Ab einer Entfernung von 30 cm soll der Ton durchgängig ertönen.

Recherche: Wie wird Ultraschall erzeugt und gemessen?

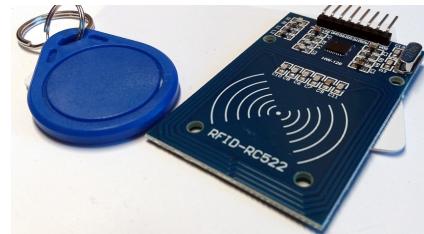
Die Erzeugung des Ultraschalls beruht wie beim Piezo-Summer auf dem inversen piezo-elektrischen Effekt (vgl. S. 13); die Messung des Ultraschalls beruht auf dem piezo-elektrischen Effekt. Recherchiere im Internet die Hintergründe dieser Effekte und fasse sie zusammen.

5.13. RFID

Mit RFID-Chips (*radio frequency identification*) lässt sich ein digitaler Schlüssel bauen, um zum Beispiel Türen abzusichern. Der RFID-Empfänger sendet über die rechteckige Spule Radiowellen, welche Energie transportieren. Diese empfängt der RFID-Sender (blauer Chip oder auch weiße Karte), woraufhin er seine eizigartige ID zurücksendet. Diese wird wiederum von der Rechteckspule empfangen und elektronisch so aufbereitet, dass man am Arduino die ID lesen kann.

Die Verbindung mit dem Arduino erfolgt über ein *Serial Peripheral Interface (SPI)* (weitere Informationen unten), weshalb die meisten Pins festgelegt sind (siehe rechts). Der RST-Pin und der SDA-Pin lassen sich ggf. noch ändern. Wichtig: Der Mikrocontroller auf dem RFID-Chip arbeitet mit einem Logiklevel von 3,3 V und würde durchbrennen, wenn man ihn an 5 V anschließt. Der Arduino verfügt direkt neben dem 5 V-Anschluss auch über einen 3,3 V-Anschluss. Der IRQ-Pin des RFID-Empfängers wird nicht benötigt und kann ignoriert werden.

Zur Programmierung kann man die bloße Anwesenheit eines RFID-Senders abfragen oder die ID eines RFID-Senders. Die Anwesenheitsabfrage unterscheidet nicht, welcher RFID-Sender anwesend ist, sie liefert also bei *jedem* RFID-Sender `true` zurück. Sinnvoller ist also in den meisten Fällen die Abfrage der einzigartigen ID des RFID-Senders, um diese abspeichern und im nächsten Programm darauf reagieren zu können. Ein Beispielprogramm zeigt Abbildung 5.31.



B 5.30. RFID-Empfänger (rechts) mit blauem RFID-Sender (links).

RFID-RC522 Reader	R
RST	9
SDA	10
GND	GND
3,3V	3,3V
SCK	13
MOSI	11
MISO	12

gib Anwesenheit RFID-RC522 Reader R

gib ID RFID-RC522 Reader R



B 5.31. Beispielprogramm zum Auslesen eines RFID-Senders in Form einer weißen Karte.

Fehlerquellen: Bei der Programmierung gibt es zwei Fehlerquellen zu beachten, die man möglichst vermeiden sollte:

- Werden die Befehle `gib ID` und `gib Anwesenheit` kurz hintereinander verwendet, wird der zweite Befehl nicht funktionieren, weil dazu (durch die Implementierung im Hintergrund) immer eine neue Karte erkannt werden muss.
- Für den Vergleich von vorgegebener ID und eingelesener ID muss für beide eine Variable vom Typ Zeichenkette angelegt werden, damit Nepo keinen Fehler anzeigt. Das unten abgebildete Beispiel erzeugt also einen Fehler. (Hintergrund: Wenn die vorgegebene ID nicht als Zeichenkette (String) definiert wird, wird sie als sogenanntes Character-Array behandelt. Da die eingelesene ID aber vom Typ String ist, sollen sozusagen Äpfel mit Birnen verglichen werden.)



B 5.32. Der Vergleich von vorgegebener ID und eingelesener ID erzeugt hier einen Fehler.

Projekt 10: Katzentür

Baue und programmiere einen Prototypen für eine Katzentür, die sich automatisch öffnet, wenn die Katze (mit RFID-Chip am Halsband) sich der Tür nähert.

Erweiterung: Schließe einen weiteren RFID-Empfänger am Arduino an (siehe Informationen unten). Einer der RFID-Empfänger soll sich draußen, der andere drinnen befinden. Programmiere nun drei Modi, zwischen denen sich hin- und herschalten lässt:

- Standard: Tür geht immer auf,
- Tag: Tür geht nur von innen nach außen auf (alle Katzen sollen raus),
- Nacht: Tür geht nur von außen nach innen auf (alle Katzen sollen rein).

Serial Peripheral Interface (SPI)

Das *Serial Peripheral Interface* (engl. für serielle periphere Schnittstelle) ist ein Bus-System, um Daten zwischen Mikrocontrollern und integrierten Schaltkreisen nach dem Master-Slave-Prinzip auszutauschen. Der Master ist hier immer der Arduino, die Slaves sind die angeschlossenen Bauteile. Für die Kommunikation werden neben zwei Leitungen zur Spannungsversorgung (5V bzw. 3,3V und GND) vier weitere Leitungen benötigt:

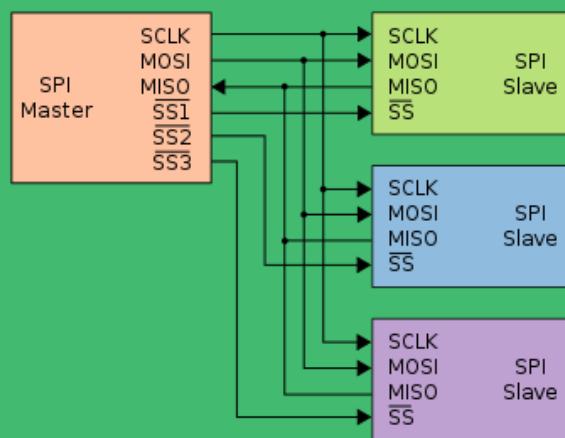
SCLK: *Serial Clock*. Auf dieser Leitung gibt der Master den Takt an, in dem die Bits gesendet werden.

MOSI: *Master Out, Slave In*. Auf dieser Leitung sendet der Master Informationen, die die Slaves empfangen.

MISO: *Master In, Slave Out*. Auf dieser Leitung empfängt der Master Informationen, die die Slaves senden.

SS: *Slave Select*. Diese Leitung wird vom Master auf ein GND-Potential (logisch 0) gebracht, um dem damit verbundenen Slave mitzuteilen, dass die folgenden Informationen an ihn gehen sollen. Im Gegensatz zu den ersten drei Leitungen erhält jeder Slave einen eigenen SS-Pin am Master, damit dieser sie einzeln ansprechen kann (siehe Abb. XX).

Da das RFID-Modul aus diesem Abschnitt auch über den I2C-Bus angesteuert werden kann, haben zwei Pins eine mehrfache Funktion. Insbesondere entspricht der SDA-Pin (im I2C-Bus die serielle Datenleitung) bei der Ansteuerung über den SPI-Bus dem SS-Pin (siehe [components101.com](#)).



B 5.33. SPI-Verbindung mit einem Master und drei Slaves. Quelle: [Wikipedia](#), Lizenz: CC-BY-SA 3.0, Colin Burnett.

5.14. Projektarbeit

Überlegt euch ein Arduino-Projekt, das ihr gerne umsetzen würdet, und erarbeitet euch ggf. die fehlenden Grundlagen dazu. Ihr könnt alle Bauteile aus eurem Arduino-Kit nutzen. Die Abschnitte aus diesem Kapitel geben euch eine Einführung zu einigen Bauteilen und ggf. bieten sie euch auch Ideen, was man mit dem Arduino umsetzen kann.

Am Ende solltet ihr euer Projekt mit einem funktionierenden Prototypen auf einem Steckbrett vorstellen können. Zudem sollt ihr eine kurze Ausarbeitung mit folgenden Abschnitten anfertigen:

- Projektziel
- Materialliste
- Schaltplan
- Programm (als Screenshot)
- Erläuterung des Schaltplans und des Programms
- Ausblick: Mögliche Erweiterungen oder Anwendungen

Mögliche übergeordnete Themen für Projekte:

- Smart Home
- Smart Garden
- Verkehrsmanagement / E-Mobilität
- Smart Classroom
- ...

6. Steuerung komplexer Informatik-Systeme

Mit den vielen Bauteilen, die nun zur Verfügung stehen, lassen sich bereits große und komplexe Projekte realisieren. Um dabei nicht den Überblick zu verlieren, hilft es, sich mit einigen weiteren Konzepten aus der Informatik zu beschäftigen, die das Programm und die Vorgehensweise zur Erstellung des Programms besser strukturieren.

In diesem Kapitel lernst du...

- ... die Funktionsweise von Programmen mit einem Zustandsdiagramm zu planen und zu veranschaulichen,
- ... Programme durch eine zustandsbasierte Modellierung als endlicher Automat zu flexibilisieren.

Projekte in diesem Kapitel:

> Fußgängerampel 101

> Parkplatzschanke 103

6.1. Automaten

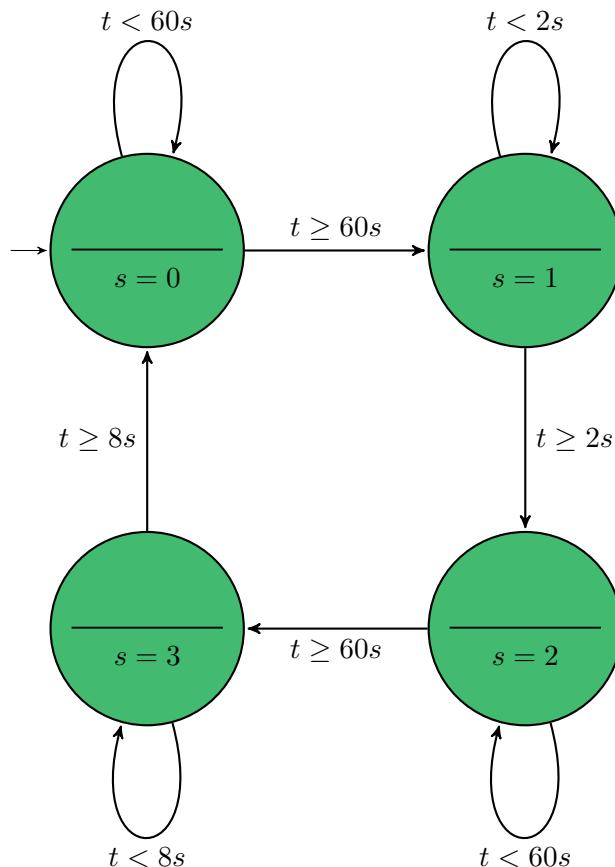
Bisher wurden Algorithmen rezeptartig als feste Handlungsabfolgen beschrieben, die nacheinander durchlaufen werden und dabei ggf. einfache Fallunterscheidungen berücksichtigen. In der Automatentheorie steht weniger das Befolgen eines Rezeptes im Mittelpunkt als das Einnehmen verschiedener Zustände, zwischen denen unter festgelegten Bedingungen Übergänge stattfinden. Dies macht die Algorithmen flexibler im Umgang mit Eingaben aus der Umwelt, wodurch sie zudem leichter zu erweitern sind.

Ziel: Durch eine zustandsbasierte Modellierung sollen Algorithmen flexibler werden und einfacher zu erweitern.

Aufgabe 1: Ampelzustände

- Nenne die vier Zustände (*engl. states*) einer Ampel. Beschreibe die Bedingung(en), unter denen der Wechsel vom einen in den nächsten Zustand stattfindet.
- Unten ist ein sogenanntes Zustandsdiagramm einer Ampel zu sehen. Erläutere, wie es aufgebaut ist und ordne ihnen die vier Ampelzustände zu.

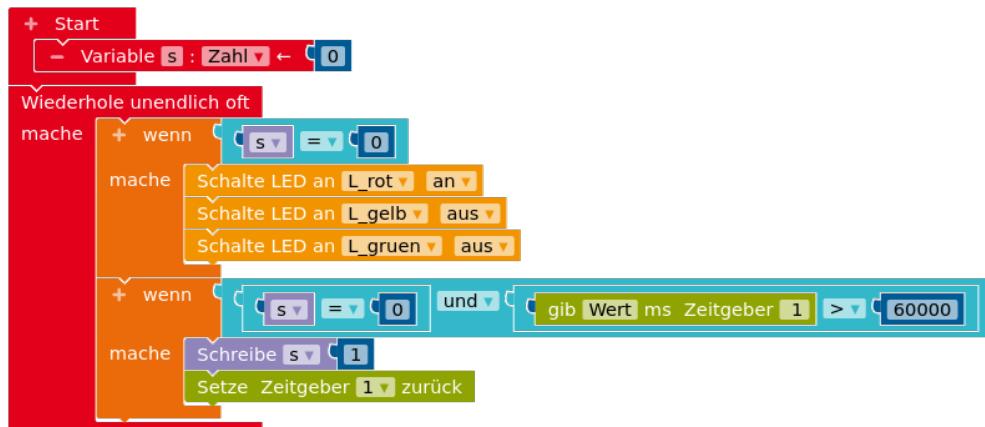
Hinweis: s : state (*engl. für Zustand*), t : time (*engl. für Zeit*)



- Unten ist abgebildet, wie ein Programm aussehen könnte, das das Zustandsdiagramm modelliert. Jeder Übergang (jede Bedingung) wird durch eine `falls`-Abfrage in das Programm

integriert. Vervollständige das Programm zur Modellierung einer einfachen Ampel und bau die Ampel auf.

Erweiterung: Programmiere jeden Ampelzustand als eigene Funktion, sodass das Programm lesbarer wird.



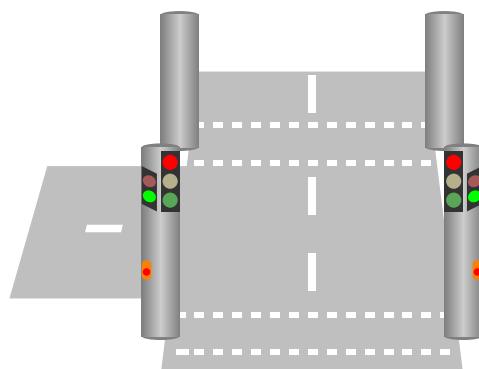
Idee: Materialien zum Kerncurriculum Informatik im Sekundarbereich I, Niedersächsisches Kultusministerium

Bei einer klassischen Programmierung mit `warte`-Blöcken wäre eine Erweiterung um eine Fußgängerampel, die zu jedem beliebigen Zeitpunkt aktiviert werden kann, unmöglich, da das Programm während des Wartens das Drücken der Fußgängerampel gar nicht mitbekommen würde. Durch die zustandsbasierte Modellierung mithilfe einer Stoppuhr lässt sich die Fußgängerampel jedoch integrieren.

Projekt 1:

Erweiterung um eine Fußgängerampel

Eine Fußgängerampel ist zunächst aus oder deaktiviert. Sie kann jederzeit durch einen Taster aktiviert werden, das heißt, sie zeigt rot. Sie bleibt solange rot, bis die normale Ampel ebenfalls rot zeigt und eine gewisse Mindestdauer vergangen ist. Die normale Ampel bleibt nun rot, während die Fußgängerampel auf grün springt. Nach einer gewissen Zeit schaltet die Fußgängerampel wieder auf rot. Nach ein paar Sekunden Rotphase für die letzten Fußgänger, die ihren Weg über die Straße noch beenden müssen, ist die Fußgängerampel wieder aus und die normale Ampel springt in die Rot-Gelb-Phase, mit der sie ihren normalen Rhythmus fortsetzt.



- Entnimm der obigen Beschreibung die vier Zustände einer Fußgängerampel und die Bedingungen, unter denen vom einen Zustand in den nächsten gewechselt wird. Entwickle daraus ein Zustandsdiagramm für die Fußgängerampel. Ergänze die Übergänge von der Rotphase der normalen Ampel zur Rotphase der Fußgängerampel und von der (wieder) deaktivierten

Fußgängerampel zur Rot-Gelb-Phase der normalen Ampel.

Achtung: Hier liegen zwei Automaten vor, nämlich die normale Ampel und die Fußgängerampel, daher gibt es auch zwei Startzustände.

- b) Ergänze die Fußgängerampel inkl. Taster auf dem Steckbrett und ergänze das Programm um die Zustände der Fußgängerampel.

Tipp: Nutze eine Variable, um den Tasterstatus zu speichern, nachdem der Taster einmal gedrückt wurde.

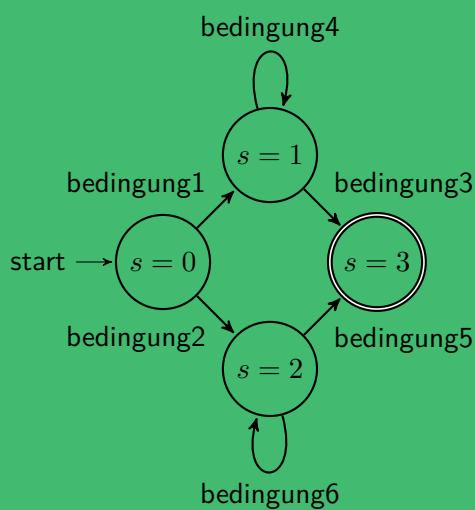
Idee: Materialien zum Kerncurriculum Informatik im Sekundarbereich I, Niedersächsisches Kultusministerium

↶ Taster
auslesen

Endlicher Automat und Zustandsdiagramm

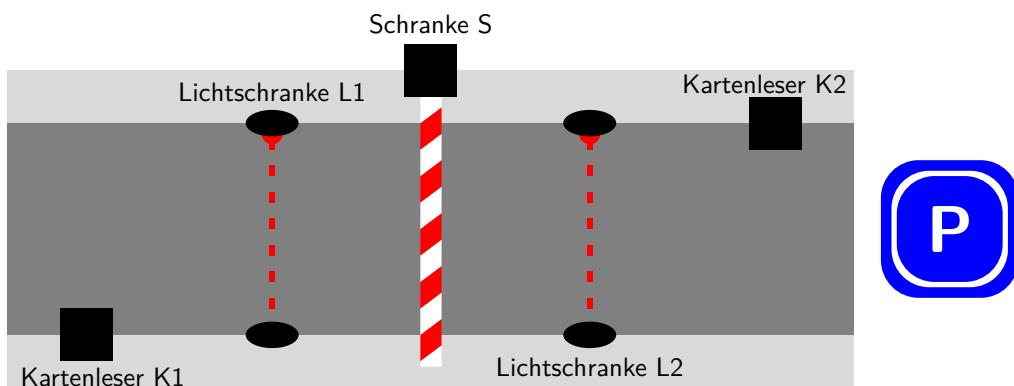
Ein Automat (auch: abstrakte Maschine) ist in der Informatik ein Modell zur Beschreibung einer Datenverarbeitung. Er befindet sich anfangs in seinem Startzustand. Abhängig von der nächsten Eingabe (z. B. das Ablaufen einer Zeit oder das Drücken eines Tasters) und des aktuellen Zustands erfolgt der Übergang in einen Folgezustand. Auch zu diesem gibt es wiederum einen Folgezustand, der abhängig von der nächsten Eingabe und dem aktuellen Zustand erreicht wird. Es kann einen oder mehrere Endzustände geben, die keinen Folgezustand haben, wenn der Ablauf des Automaten vollständig ausgeführt wurde. Wenn der Automat endlich viele Zustände hat, spricht man von einem **endlichen Automaten**.

Ein Zustandsdiagramm veranschaulicht das Verhalten eines Automaten. Die Zustände werden mit einem Kreis oder abgerundeten Rechteck dargestellt. Die Übergänge werden mit Pfeilen dargestellt, an denen die Bedingung für den Übergang steht. Der Startzustand wird mit einem zusätzlichen Pfeil markiert, an den manchmal zusätzlich `start` geschrieben wird. Der ggf. vorhandene Endzustand wird durch einen doppelten Rahmen markiert.



Projekt 2:**Parkplatzschranke**

Auf einen Parkplatz kommt man häufig erst, wenn man einen Parkschein gezogen hat. Danach öffnet sich die Schranke und man kann auf den Parkplatz fahren. Die Schranke schließt sich automatisch wieder, nachdem das Auto hindurchgefahren ist. Beim Verlassen des Parkplatzes muss man wiederum zuerst den bezahlten Parkschein einlesen lassen, bevor sich die Schranke öffnet und automatisch wieder schließt, nachdem ein Auto hindurchgefahren ist.



Das Verhalten der Parkplatzschranke soll auf dem Steckbrett simuliert werden. Dazu wird aus Pappe ein „Auto“ geschnitten, das durch die mittlere Spalte des Steckbretts „fährt“.

- Erläutere, wie man die Kartenleser und das Durchfahren der Schranke mithilfe von zwei Tastern und zwei Lichtschranken simulieren kann. Notiere alle benötigten Bauteile.
- Baue die Parkplatzschranke mit allen benötigten Teilen auf dem Steckbrett auf.
- Entwickle ein Automatenmodell für die Parkplatzschranke. Überlege dazu, in welchen Zuständen sich die einzelnen Elemente beim Einfahren und beim Ausfahren befinden.
- Implementiere das Automatenmodell.

Idee: Materialien zum Kerncurriculum Informatik im Sekundarbereich I, Niedersächsisches Kultusministerium

↶ Taster
↶ LDR
↶ Servo

Motivationsquellen

> [Cocktail Mixer](#)

Dieser Automat mixt automatisch einen aus sechs wählbaren Drinks!

> [Roboterhand](#)

Mit diesem Aufbau wird die eigene Hand auf eine Roboterhand gespiegelt, wodurch sich der Roboter viel besser steuern lässt.

> [One | Aerospace](#)

Diese Gruppe von Studenten versucht u. a. mit Hilfe eines Arduino eine Rakete zu bauen!

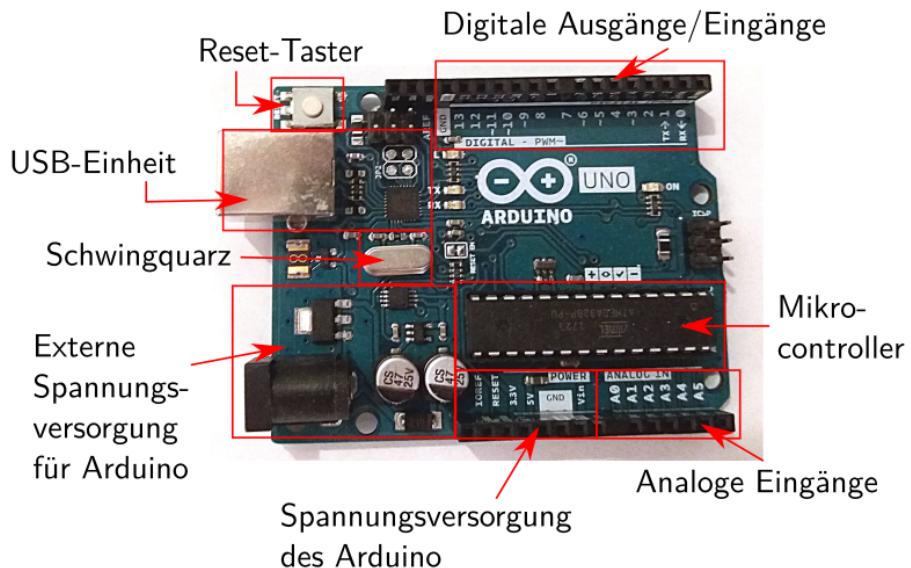
> [Ziegen-Tracker](#)

Mit dieser Kombination aus Arduino und GPS-Antenne wird es Ziegen unmöglich von ihrer Farm auszubrechen.

A. Anhang

A.1. Ein kurzer Überblick - die Funktionseinheiten des Arduino Uno

Im Folgenden wird ein vertiefter Überblick über die wichtigsten Komponenten des Arduino Uno, den sogenannten Funktionseinheiten, gegeben (vgl. Abb. A.1).



B A.1. Der Arduino Uno lässt sich in wenige Funktionseinheiten aufteilen (weitere Erklärungen im Text).

Mikrocontroller: Wenn der Arduino Uno als Mikrocontroller bezeichnet wird, dann stimmt das eigentlich nicht ganz. Der eigentliche Mikrocontroller, auf dem das Programm läuft, ist das markierte schwarze Bauteil, bei dem es sich um einen Atmega328 handelt. Die Arbeit von Massimo Banzi und David Cuartielles unter anderem bestand darin, die restlichen Bauteile, die zur Verwendung des Mikrocontrollers benötigt werden, zusammen mit dem Atmega328 auf einem Board zu befestigen. Dadurch wurde die Handhabung wesentlich vereinfacht! Auf dem Mikrocontroller befindet sich ein sogenannter Bootloader, der das Programm vom Computer in den Speicher des Mikrocontrollers lädt.

Schwingquarz: Der Atmega328P arbeitet mit einer Taktfrequenz von 16 MHz, die von dem Schwingquarz vorgegeben wird. Das heißt, es werden pro Sekunde 16 000 000 „Arbeitsschritte“ vollzogen! Natürlich sind moderne Computer noch viel schneller, aber für unsere Zwecke reicht das völlig aus.

Stromversorgungseinheit: Natürlich benötigt ein Mikrocontroller Strom oder besser gesagt eine gewisse Spannung, um funktionieren zu können. An der schwarzen Buchse können Gleichspan-

nungsquellen zwischen 7 V und 12 V angeschlossen werden, um den Mikrocontroller mit Energie zu versorgen. Die anderen Bauteile regeln diese Spannung dann auf die 5 V herunter, die der Mikrocontroller zum Arbeiten benötigt.

USB-Einheit: An der silbernen Buchse kann das USB-Kabel angeschlossen und der Arduino Uno somit mit dem Computer verbunden werden. Dies ermöglicht die Übertragung eines Programms auf den Arduino Uno. Außerdem lassen sich dadurch weitere Daten, z.B. von Sensoren am Arduino Uno, mit dem Computer austauschen. Der USB-Anschluss dient außerdem bereits als Spannungsversorgung. Solange der Arduino Uno am Computer angeschlossen ist, braucht man in der Regel keine Batterie als zusätzliche Spannungsversorgung. Eine Ausnahme sind Motoren, denn der USB-Anschluss stellt *maximal 500 mA* bereit und Motoren benötigen häufig mehr. Glücklicherweise ist die USB-Buchse gegen Überstrom geschützt und schaltet sich bei einem Kurzschluss automatisch ab.

Digitale Eingänge und Ausgänge: Die digitalen Pins lassen sich entweder als Ausgänge oder als Eingänge benutzen. Wenn sie als Ausgang festgelegt werden, dann kann dort eine Spannung von 0 V oder 5 V angelegt werden. Wenn sie als Eingang festgelegt werden, können dort Spannungen von 0 V oder 5 V eingelesen werden. Einige Pins sind mit einer Tilde (~) gekennzeichnet, was bedeutet, dass sie über eine Pulsweitenmodulation verfügen. Der GND-Pin stellt den negativen Pol dar (auch Erdung oder GrouND genannt).

Hinweise:

- Die **maximale Stromstärke**, die die Digitalpins vertragen, ist **40 mA**. Empfohlen sind eher 20 mA.¹
- Pin 0 und Pin 1 sind bei angeschlossenem USB-Kabel für die Kommunikation mit dem Computer belegt und können daher nicht benutzt werden, wenn der Arduino Uno mit dem Computer kommunizieren soll!

Spannungsversorgung: Hiermit ist die Spannungsversorgung für die Bauteile gemeint, die man an den Arduino anschließt. Diese Pins lassen sich nicht über das Programm an- und ausstellen, sondern liefern permanent eine gewisse Spannung (5 V bzw. 3,3 V), die mit den GND-Pins geerdet wird. Der RESET-Pin wird in diesem Skript nicht genutzt, weil man den Reset über die Reset-Taste durchführen kann (siehe unten). Mit VIN kann eine Referenzspannung (Vergleichsspannung) eingelesen werden. Der IOREF-Pin legt fest, ob der Mikrocontroller mit 5 V oder mit 3,3 V operiert. Standardmäßig sind 5 V eingestellt.

Hinweis: Die **maximale Stromstärke**, die der 5V-Pin sowie die GND-Pins vertragen, ist 200 mA.

Analoge Eingänge: An analogen Eingängen kann eine Spannung eingelesen werden. Dabei sind nicht nur zwei Werte möglich (wie bei den digitalen Eingängen), sondern auch viele Werte zwischen 0 V und 5 V.

Reset-Taste: Mit der Reset-Taste lässt sich der Arduino neu starten. Dabei wird das Programm, das auf den Arduino geladen wurde, nicht gelöscht, sondern lediglich neu gestartet.

¹Weitere Infos finden sich unter <https://playground.arduino.cc/Main/ArduinoPinCurrentLimitations>.

A.2. Installation der integrierten Entwicklungsumgebung (IDE) für den Arduino

Üblicherweise wird der Arduino mit der integrierten Entwicklungsumgebung (IDE - engl. *integrated development environment*), die ebenfalls Arduino heißt, programmiert. Diese verfügt über einen Texteditor, in dem das Programm geschrieben werden kann. Zusätzlich lassen sich hier einfach die Bibliotheken und weiteren Boards verwalten und einbinden. Wichtig ist zudem die Syntax-Prüfung sowie letztendlich die Kompilation des Programms. Das Programm wird in der maschinennahen Programmiersprache C++ geschrieben, für die jedoch einige vorgefertigte Befehle für den Arduino bereitgestellt werden, damit der Einstieg leichter fällt. Das Kompilieren des Programms macht aus dem Programmcode einen maschinenlesbaren Code (von Einsen und Nullen), der dann auf den Arduino übertragen wird.

Die IDE lässt sich im Internet auf der Projekthomepage für jedes Betriebssystem herunterladen:

<https://www.arduino.cc/en/Main/Software>.

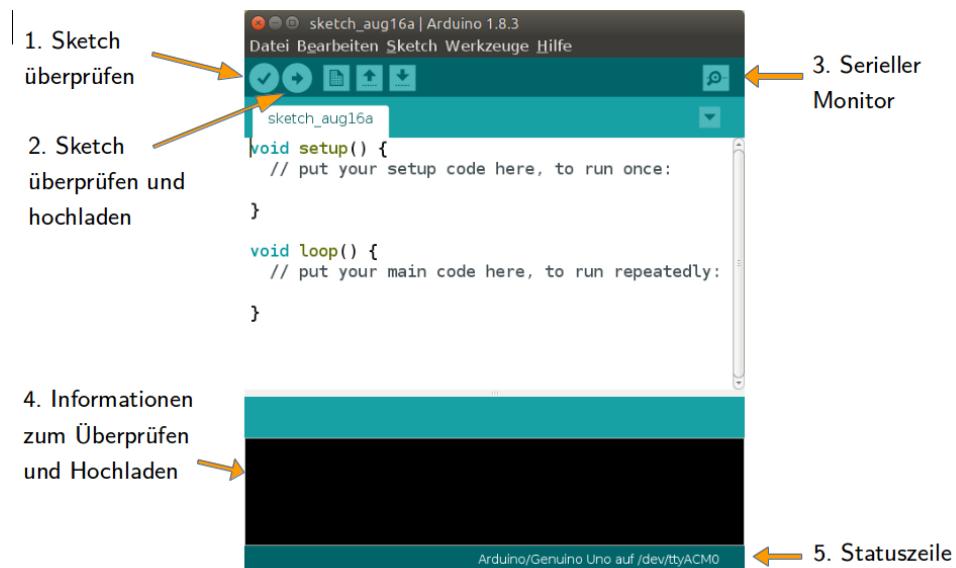
Auf derselben Seite gibt es zudem ausführliche Anleitungen zur Installation (englisch):

<https://www.arduino.cc/en/Guide/HomePage>.

Eine ebenfalls ausführliche und deutschsprachige Installationsanleitung wird auf der CD des Starter Kits mitgeliefert.

A.3. Mit der Arduino IDE ein Programm auf den Arduino Uno übertragen

Nach Installation der Entwicklungsumgebung kann das erste, noch leere Programm auf den Arduino Uno übertragen werden, um zu testen, ob und wie das Ganze funktioniert. Dazu wird der Arduino Uno mit dem USB-Kabel an den PC angeschlossen und die Arduino-Entwicklungsumgebung geöffnet. Wenn der Arduino Uno zum ersten Mal an den PC angeschlossen wird, muss evtl. kurz gewartet werden, damit die Treiber für die Kommunikation von PC und Arduino Uno installiert werden. Das passiert aber vollautomatisch.



B A.2. Ein Überblick über die Arduino-Entwicklungsumgebung.

In der Mitte des Fensters der Entwicklungsumgebung wird das Programm geschrieben. Innerhalb der geschweiften Klammern von `void setup()` werden Anweisungen geschrieben, die ganz am Anfang genau einmal ausgeführt werden. Im Setup befindet sich bereits ein Kommentar, der sich daran erkennen lässt, dass er eine graue Farbe hat. Kommentare werden im Programm dadurch gekennzeichnet, dass am Anfang zwei Schrägstriche stehen. Sie sind nützlich, um in komplizierteren Programmen auch später noch den Überblick zu behalten und um anderen, die das Programm nutzen wollen, die Möglichkeit zu geben, es zu verstehen.

Innerhalb der geschweiften Klammern von `void loop()` stehen Anweisungen, die immer wieder wiederholt werden. Die Anweisungen werden der Reihe nach abgearbeitet und nach dem letzten Befehl beginnt der Arduino Uno wieder mit dem ersten Befehl innerhalb des Loops, der auf Deutsch entsprechend Schleife heißt. In dieser unendlichen Schleife wird der Hauptteil des Programms stehen.

! Für den Einstieg ist es am einfachsten, den Quelltext von Nepo zu kopieren, nachzuvollziehen und anzupassen. Um den Quelltext von Nepo in der Arduino IDE nutzen zu können muss lediglich die Zeile `#include <NEPODefs.h>` entfernt werden.

Die wesentlichen anderen Bestandteile der Entwicklungsumgebung werden im Folgenden erklärt.

1. Sketch überprüfen: Mit dem Haken kann überprüft werden, ob das Programm, das in der Arduino-Welt auch Sketch genannt wird, richtig geschrieben ist. Damit ist natürlich nicht der Inhalt gemeint, denn die Entwicklungsumgebung weiß natürlich nicht, was das Programm bewirken soll. Es wird nur überprüft, ob die „Grammatik“ (Fachwort: die Syntax) des Programms richtig ist. So wie in einem Satz die Wörter in einer bestimmten Reihenfolge stehen und ggf. mit Kommata getrennt werden müssen, müssen in einer Programmiersprache Befehle korrekt geschrieben, geöffnete Klammern wieder geschlossen und das Ende von Befehlen mit einem Semikolon deutlich gemacht werden. Genau das wird mit dem Haken überprüft.

2. Sketch überprüfen und hochladen: Mit dem Pfeil wird der Sketch nicht nur überprüft, sondern

auch auf den Arduino Uno hochgeladen.

3. *Serieller Monitor*: Mit dem seriellen Monitor kann man Daten vom Arduino Uno abfragen und am Computer anzeigen lassen. Man kann auch umgekehrt Daten am Computer eingeben und vom Arduino auslesen lassen. Wenn man ihn entsprechend programmiert, kann man dem Arduino auf diese Weise auch während der Durchführung des Programms noch Befehle geben.

4. *Informationen zum Überprüfen und Hochladen*: Beim Überprüfen und Hochladen zeigt die Entwicklungsumgebung in dem schwarzen Bereich Informationen zum Fortschritt und ggf. zu Fehlern an.

5. *Statuszeile*: In der Statuszeile sieht man, welcher Arduino-Typ ausgewählt wurde (in diesem Skript ist das immer der Arduino Uno) und an welchem USB-Port er sich befindet. Dargestellt ist der Pfad unter Linux - unter Windows werden die Ports mit COM1, COM2, ... bezeichnet.

Fehlerursachen

Eine häufige Fehlerursache, die das Hochladen verhindert, ist, dass kein oder ein falscher USB-Port ausgewählt wurde. Der USB-Port muss unter Werkzeuge -> Port ausgewählt werden. In der Regel ist dort nur der eine Port auswählbar, an dem sich der Arduino Uno befindet. Falls kein Port auswählbar ist, fehlt der Treiber für die Kommunikation von Arduino und Computer. Er muss dann manuell nachinstalliert werden.

B. Didaktisch-Methodische Bemerkungen

Im Jahr 2016 bin ich das erste Mal in einem Youtube-Video auf den Arduino aufmerksam geworden. Darin wurde er für ein Heimwerker-Projekt genutzt und als einfache Möglichkeit vorgestellt, programmieren zu lernen und Projekte umzusetzen. Ich war schnell begeistert und schaffte mir selbst ein Arduino-Starter-Kit an. Ich arbeitete mich durch einige Tutorials und lernte schnell, immer komplexere Projekte aufzubauen und zu programmieren - dabei war mein Studium in Mathematik und Physik sicherlich hilfreich. Aber endlich war das ganze Wissen, das ich mir angeeignet hatte, nicht nur theoretisch wertvoll, sondern auch ganz praktisch anwendbar! Diese Mischung aus theoretischem Wissen, praktischem Aufbau, Verständnis für den technischen Alltag sowie insbesondere der Möglichkeit, *eigene Projekte anzugehen und kreativ zu werden*, war es, die mich tief begeisterte und die ich bald weitergeben wollte.

Der Bezug zur Schule ist offensichtlich. Nachdem man ein wenig in die Arduino-Welt eingetaucht ist, erscheint es fast fahrlässig, dass man im Physikunterricht Halbleiterbauelemente behandelt, ohne gleichzeitig zu thematisieren, wie man sie über ein Programm anspricht und ausliest, sodass sie zu etwas Praktischem beitragen können.¹ Zudem werden sie in Plastikkästen versteckt, die zwar verhindern, dass etwas kaputt geht und für wenige Cent neu beschafft werden muss, aber auch dazu führt, dass die meisten Schülerinnen und Schüler kaum wissen, wie die elektrischen Bauteile, die sie im Bändermodell erklären sollen, aussehen. Auf diese Art und Weise kann die Elektronik nur furchtbar theoretisch und dementsprechend wenig motivierend werden. Und so versuchte ich mich schon bald an der Integration des Arduino in den Unterricht - zunächst ein kleines Projekt im Physikunterricht, dann eine Arduino-AG und schließlich ein Arduino-Wahlpflichtfach im MINT-Profil.

Was mir für den Unterricht noch fehlte, war ein Skript wie dieses, welches das theoretische Wissen, das sich rund um den Arduino vermitteln lässt, didaktisch aufbereitet und strukturiert, sodass es sich im Unterricht anwenden lässt. Im Gegensatz zu den vielen Internet-Tutorials sollte zum Einen eine graphische Programmiersprache im Vordergrund stehen, weil meine Erfahrungen im Unterricht zeigten, dass die Schüler bei ihrer ersten Begegnung mit dem Programmieren größere Probleme mit der komplexen Syntax von C++ bekamen. Zum Anderen sollte das Wissen mit den Schülern erarbeitet werden statt es einmal zu erklären und dann nachmachen zu lassen.

So verwendete ich im Schuljahr 2018/19 sehr viel Zeit und Arbeit in dieses Skript. Mein Ziel ist, meine Begeisterung für die Arduino-Welt mit der beschriebenen Mischung aus theoretischem Wissen, praktischem Aufbau, Verständnis für den technischen Alltag sowie der Möglichkeit eigene Projekte umzusetzen an möglichst viele Schülerinnen und Schüler weiter zu geben. Aus diesem

¹Dass man stattdessen auch eine Transistorschaltung o. ä. aufbauen kann, ist mir bewusst, aber da diese recht kompliziert werden, werden Transistorschaltungen wenn überhaupt erst ganz am Ende der Halbleiterelektronik behandelt.

Grund veröffentliche ich dieses Skript auch unter einer CC-Lizenz ([BY-NC-SA 4.0](#)), die es jedem erlaubt, dieses Skript zu vervielfältigen und anzupassen, solange mein Name genannt und mögliche Anpassungen unter der gleichen Lizenz weitergegeben werden, die zudem enthält, dass dieses Skript (von Verlagen etc.) nicht kommerziell vertrieben werden darf.

Im Jahr 2020 hatte ich dann durch den Shutdown während der Corona-Pandemie noch einmal genug Zeit, um eine große Umstellung vorzunehmen. Statt wie bisher auf das Programm mBlock zu setzen, das eigentlich auf die Programmierung von firmeneigenen „mBots“ ausgelegt ist und in der neuesten Version über keinen seriellen Monitor mehr verfügte, probierte ich nun Neko4Arduino von der Fraunhofer-Gesellschaft aus und merkte, dass sich damit noch viel mehr Projekte und Inhalte umsetzen lassen würden. Zudem ist der Aufbau an einigen Stellen intuitiver und macht die Anbindung von Bauteilen an den Arduino leichter. Die andere Vorgehensweise machte es aber auch notwendig, das Skript noch einmal komplett zu überarbeiten. So gut wie jede einzelne Seite wurde noch einmal angefasst und umgearbeitet; außerdem sollte eine ganz andere Struktur entstehen. Und so bin ich froh und stolz, dieses große Projekt zu einer ersten fertigen Version gebracht zu haben.

B.1. Zielgruppe

Das Skript wurde für das MINT-Profil im Jahrgang 10 geschrieben. Die Schülerinnen und Schüler haben bis dahin den gewöhnlichen Physikunterricht zu Stromstärke, Spannung und Widerstand gehabt. Die Behandlung von Halbleitern erfolgt bei uns ebenfalls im Jahrgang 10, also parallel zu diesem Kurs. Dies ist sicherlich hilfreich, aber nicht notwendig für den Einsatz dieses Skripts. Dieses Skript behandelt nur die elektrotechnische Anwendung von Halbleiterelementen, nicht jedoch die Funktionsweise im Bänder- oder Teilchenmodell, die im Physikunterricht erläutert wird. Viele der teilnehmenden Schülerinnen und Schüler haben vorher bereits im MINT-Profil mit LegoMindstorms-Robotern gearbeitet und dadurch etwas Programmiererfahrung gewonnen. Dies ist ebenfalls hilfreich, aber nicht notwendig für den Einsatz dieses Skripts. Es sind keine vorherigen informatischen Kenntnisse notwendig!

Neben Jahrgang 10 erscheint es auch denkbar, dieses Skript in Jahrgang 9 oder höheren Jahrgangsstufen einzusetzen, da dort bereits (fast) alle physikalischen und mathematischen Grundlagen gelegt sind - ggf. ist bei der Regression zum Messen von Helligkeit oder Temperatur eine Anpassung notwendig.

B.2. Organisatorisches: Arduino Starter Kits, Raum, Zusammenarbeit

Für den Wahlpflichtkurs wurden 15 Arduino Starter Kits angeschafft, die neben einem Arduino-Klon auch fast alle hier beschriebenen Bauteile enthielten. Zusätzlich wurden Bewegungsmelder und USB-Kabel mit 2 m Länge angeschafft. Da die mitgelieferten Boxen sehr eng gepackt sind und nie wieder ordentlich zusammengepackt werden, wurden zusätzlich größere Aufbewahrungsboxen angeschafft, in die später auch die aufgebauten Schaltungen gesteckt werden konnten, sodass sie

am Anfang der nächsten Stunde direkt wieder zur Verfügung standen.

Die Schüler bekamen jeweils zu zweit eine Box zugewiesen, auf die eine Nummer geschrieben wurde, sodass die Namen der Schüler und die zugehörige Box festgehalten werden konnten. Damit waren die Schüler auch für den Inhalt der Box verantwortlich. Im Unterricht arbeiteten die Schüler in den Praxisübungen dann auch stets zu zweit zusammen. Die Arbeit zu zweit kann gut dazu genutzt werden, dass einer hauptsächlich für die Schaltung und einer hauptsächlich für das Programmieren zuständig ist. Der Unterricht fand stets im Computerraum statt, da Computer für die Arbeit mit dem Arduino unerlässlich sind. Aus diesem Grund ist dieses Skript auch nicht zum Ausdrucken gedacht.

B.3. Zum Aufbau dieses Skripts

B.3.1. Aufbau der Kapitel

Im Einführungskapitel stehen die Grundlagen im Vordergrund: Nach etwas Theorie zum Aufbau des Arduino, die auch später thematisiert werden könnte, wird die Programmierumgebung Open Roberta vorgestellt, um dann möglichst schnell mit dem praktischen Programmieren der Board-LED loslegen zu können. Erst im zweiten Schritt geht es dann um eine externe LED, weil dazu auch das Steckbrett und die Bedeutung der Beinlänge der LED verstanden werden muss. Während die Farben des Widerstands hier noch genau vorgegeben werden, geht es danach um die Bedeutung der Farbringe. Damit sind die wichtigsten Grundlagen gelegt, um Schaltungen nach Anleitung aufzubauen zu können.

Im Kapitel zu Bausteinen von Algorithmen geht es dann um die schrittweise Einführung von Blöcken zum Ansteuern von Bauteilen oder algorithmischen Strukturen wie Sequenz, Verzweigung, Schleifen und schließlich Funktionen. Dabei wird der physikalische Aspekt der Schaltungen kaum thematisiert, damit das Skript ggf. auch in einem reinen Informatikunterricht eingesetzt werden kann, bei dem die Anbindung an den Arduino eine Black Box bleiben darf. So kann man sich im Algorithmen-Kapitel voll auf die Programmierung konzentrieren und schnelle Fortschritte machen, die der Motivation hilfreich sein dürften. Der ausgewählten Schaltungen sind darauf ausgelegt, dass sie einfach zu handhaben sind und die Funktionsweise des Algorithmus interaktiv erlebbar machen (z. B. bei boolschen Ausdrücken) oder visualisieren (z. B. bei Lauflichtern). Zum Abschluss des Kapitels wird das EVA-Prinzip angebahnt, um von den bisherigen Erfahrungen auf eine verallgemeinernde Sichtweise zu abstrahieren, die auch für das physikalische Verständnis hilfreich ist.

Im nächsten Kapitel geht es dann um die elektrischen Grundlagen, die im Algorithmenkapitel vernachlässigt wurden. *Wenn eine tiefergehende Thematisierung der elektrischen Grundlagen nicht gewünscht ist, kann dieses Kapitel auch ohne Nachteile für die nachfolgenden Kapitel übersprungen werden.* Hier werden noch einmal die elektrischen Grundlagen zu Spannung, Widerstand und Stromstärke wiederholt, bevor sie um das elektrische Potential erweitert werden. In den folgenden Abschnitten liegt der Schwerpunkt auf dem Verständnis des Spannungsteilers, der in verschiedenster

Form immer wieder auftaucht. Die Formel zum Spannungsteiler wird dabei immer in der Form

$$\frac{U_1}{U_2} = \frac{R_1}{R_2} \quad \text{oder} \quad \frac{U_1}{R_1} = \frac{U_2}{R_2} = \frac{U_{ges}}{R_{ges}}$$

belassen, ohne bereits $U_1 = U_{ges} - U_2$ o. ä. einzusetzen, weil sich gezeigt hat, dass die Schüler dies mit konkreten Zahlen sehr gut schrittweise hinkriegen und flexibel an die Situation anpassen können, was bei einer allgemeinen Formel nicht der Fall war. Die einhergehende Thematisierung von Potentiometer, LDR, NTC und Transistor sind auch klassische Themen des Physikunterrichts, allerdings wird hier der Fokus auf die elektrische Schaltung und praktische Anwendung gelegt und nicht auf die interne Funktionsweise und abstrakte quantenphysikalische Modelle.

Weiterhin wird der Elektromotor mit verschiedenen Ansteuerungsmöglichkeiten (Transistor, Relais, Motortreiber) ausführlich thematisiert, um einen umfassenden Einblick in verschiedene Bauteile sowie deren Vor- und Nachteile für ein bestimmtes Ziel zu erreichen.

Das Kapitel zur Erweiterung des Werkzeugkastens nimmt eine Sonderrolle ein. Hierin werden alle fehlenden Bauteile eingeführt, die bisher nicht auftraten und die sich mit Neko4Arduino ansprechen lassen. Es ist nicht notwendig, die vorherigen Kapitel komplett durchgearbeitet zu haben, um dieses Kapitel bearbeiten zu können. Im Gegenteil kann es sich immer wieder zwischendurch anbieten, auf ein neues Bauteil aufmerksam zu machen, weil diese Abschnitte weniger Theorie, aber dafür fast immer ein neues Projekt enthalten. Alternativ oder ergänzend kann das Kapitel unter der Vorgabe ein eigenes Projekt umzusetzen, von den Schülern eigenständig nach Bedarf bearbeitet werden, um sich die Grundlagen für die von ihnen benötigten Bauteile zu verschaffen (vgl. Abschnitt 5.14).

Als (bisher) letztes Kapitel folgt die Thematisierung von Automaten zur Steuerung von komplexeren Informatik-Systemen. Hier wurde versucht, das Informatik-KC (in Niedersachsen) zu berücksichtigen, das die Thematisierung von Automaten fordert. Es ist geplant, einen weiteren Abschnitt zum Steuern und Regeln hinzuzufügen.

B.3.2. Erklärung zu Links, Symbolen und eingebetteten Arbeitsblättern

Dieses Skript enthält verschiedenfarbige Links, die teilweise mit Symbolen ergänzt sind und teilweise zu eingebetteten Arbeitsblättern führen.

Links

Dunkelroter Link	Link innerhalb des Dokuments, z. B. zu einem anderen Abschnitt
Dunkelvioletter Link	Link ins Internet
Dunkelblauer Link	Link zu einer eingebetteten Datei, die als Arbeitsblatt oder Folie verwendet werden kann

Symbole

- 🔗 kennzeichnet Links, die an eine frühere Stelle im Dokument führen
- 🔧 kennzeichnet Links, die zu einem späteren Abschnitt führen, an denen ein Bauteil eingeführt wird, für das nun alle Grundlagen gelegt wurden
- 🖨️ kennzeichnet Links zu eingebetteten Arbeitsblättern, die ausgedruckt werden können
- ▶️ kennzeichnet Internet-Links, die zu einem passenden Video führen
- 🎞️ kennzeichnet Links zu eingebetteten Folien, die per Beamer gezeigt werden können und die Aufmerksamkeit auf das Wesentliche richten sollen
- ❗ Hinweis auf ein Arbeitsblatt, das die ganze oder Teile der Lösung enthält. Daher ist das Ausrufezeichen selbst der Link zu dem Arbeitsblatt, sodass man den Link nicht so leicht als solchen erkennt. Wenn kein Link vorliegt, handelt es um einen wichtigen Hinweis zum Text.

B.4. Erfahrungsbericht

Der folgende Erfahrungsbericht bezieht sich auf eine erste Version dieses Skriptes. Zur aktuellen, vollständig überarbeiteten Version liegen noch keine Erfahrungen vor, weil Corona-bedingt kein jahrgangsübergreifender Wahlpflichtunterricht angeboten werden konnte (Stand 20.08.20).

Ich habe dieses Skript im Laufe des Schuljahres 2018/19 geschrieben und in meinem Kurs mit 16 Schülern eingesetzt und ausprobiert. Dadurch sind bereits viele kleine Anpassungen in das Skript eingeflossen, um kleine Fehler zu beseitigen, Missverständnisse zu umgehen oder Hinweise besser zu platzieren.

Im ersten Halbjahr habe ich Kapitel 1 bis 4 (Bausteine von Algorithmen) behandelt und darüber auch eine Arbeit schreiben lassen. Dies hat insgesamt gut geklappt. Die vermischten Übungen am Ende von Kapitel 3 und 4 dienten zur Vorbereitung auf die Arbeit. Die Lösungen dazu, die auch in diesem Paket enthalten sind, habe ich den Schülern vor der Arbeit ebenfalls zur Selbstkontrolle zur Verfügung gestellt. Auf die Verweise zu zusätzlichen Bauteilen in Kapitel 6 habe ich in der letzten Doppelstunde vor den Weihnachtsferien zurückgegriffen.

Ich bin bereits im ersten Halbjahr mit Kapitel 5 zu analogen Ausgängen und Eingängen angefangen, welches sich über einen recht langen Zeitraum bis Anfang März zog. Über Kapitel 5 wurde auch die Klausur geschrieben, weshalb hier wiederum vermischte Übungen (und eine Extradatei mit Lösungen) bereitstehen. Weitere Bauteile aus Kapitel 6 habe ich selbstgesteuert erarbeiten lassen, um Freiraum zum Experimentieren zu geben. Die Recherche-Aufträge kamen dabei letztlich nicht zum Einsatz. Erst in Kapitel 7 zu Elektromotoren habe ich den Unterricht wieder stärker kontrolliert und für einen gemeinsamen Vergleich gesorgt. Zum Ende des Schuljahres hin haben wir in einem gemeinsamen Projekt eine Wasserrakete mit Sensoren bestückt, sodass wir letztlich nur noch eine kurze Einführung zu Funktionen hatten. Den Abschnitt zu Automaten konnte ich dadurch noch nicht im Unterricht testen.

Insgesamt haben die teilnehmenden Schüler den Unterricht zum Arduino interessiert mitgemacht. Es kommen immer wieder Fragen auf, ob man dies oder das nachbauen könnte oder wie etwas genau funktioniert. Zudem ist immer wieder zu bemerken, dass sich einzelne Schüler selbst ein Arduino-Kit zulegen und damit zu Hause experimentieren. So macht Unterricht Spaß!

B.5. Ausblick

Ein Schuljahr ist mit diesem Skript zwar schon gut gefüllt, aber ich habe schon ein paar Ideen zur Erweiterung des Skripts, die entweder von interessierten Schülern zu Hause gelesen oder im Unterricht genutzt werden können, wenn vorherige Abschnitte übersprungen oder kurz gehalten werden. Dazu gehört ein Abschnitt zu „Steuern und Regeln“ im letzten Kapitel „Steuerung komplexer Informatik-Systeme“ sowie ein Kapitel zur Spannungsversorgung des Arduino (Batterie, Spannungsregler, Akku, Laderegelung, Solarzellen) und schließlich ein Kapitel zu verschiedenen Codes und evtl. zur Kommunikation von Rechnern mittels Codes (Morse, Binärcode, Hexadezimalcode, ASCII-Code).

B.6. Anpassung / Weiterentwicklung

Wer das Skript für seine Zwecke anpassen oder gar weiterentwickeln will, ist herzlich dazu eingeladen, benötigt allerdings einige Kenntnisse zu \LaTeX . Dazu muss man sich außerdem die Entwicklerversion des Arduino-Skripts, das alle tex-Dateien, Bilddateien etc enthält, herunterladen.

Ich habe dieses Skript mit der TeX Live - Distribution und Texstudio als Editor verfasst. Es ist in mehrere tex-Dateien aufgeteilt, die sich im Hauptordner sowie im Ordner `src` befinden. Als Erstes muss die Präampel vorkompiliert werden. Dazu wählt man in Texstudio `Options - Configure Texstudio`. Beim Untermenü Commands gibt man folgenden Befehl für pdflatex ein:

```
pdflatex -ini -jobname="preamble" "&pdflatex preamble.tex\dump"
```

Nun öffnet man die Datei `preamble.tex` und kompiliert diese mit pdflatex (standardmäßig drückt man dazu F5). Danach ändert man den Befehl von pdflatex wieder zurück auf:

```
pdflatex -synctex=1 -interaction=nonstopmode --shell-escape %.tex
```

Nun kann die Hauptdatei `arduino-skript.tex` oder auch jede beliebige Teildatei mit pdflatex kompiliert werden (standardmäßig F5). Nach dem Vorkompilieren der Präambel muss das Hauptdokument ggf. zwei Mal kompiliert werden, damit alles funktioniert.

Weitere Programme:

- Zeichnungen habe ich in der Regel mit TikZ erstellt, für das es ein praktisches kleines Programm namens *KTikz* (Linux) bzw. *QTikz* (Windows) gibt, mit dem man die Zeichnung programmieren kann und dabei stets sieht, wie der aktuelle Code aussieht.
- Schaltpläne habe ich mit *QEelectroTech* erstellt. Die vorhandenen Schaltpläne können damit geöffnet und bearbeitet werden. Dazu sollten auch die selbst erstellten Bauteile im Ordner QET-Bauteile importiert werden.