Wichtiger Hinweis zu Buch und Hardware

Dieses Arbeitsheft orientiert sich an praxisorientierten Aufgaben. Die passenden elektronischen Bauteile dazu sind nicht herstellergebunden und können bei beliebigen Elektronikhändlern erworben werden. Zur Erstellung dieser Anleitung wurde ein Lernset der Firma Funduino GmbH verwendet. Auf der Internetseite des Herstellers können passende Arduino-Lernsets bestellt werden, die exakt zu den Anleitungen in diesem Heft passen.

Abbildung: Lernset Funduino XXL UNO

Checkliste

Thema	Erledigt	Ungeklärte Fragen
Vorbereitung		
Theorieteil lesen	O	
Hardware / Lernset besorgen	O	
Praxisbezogene Anleitungen		
1 Eine blinkende LED	O	
2 Der Wechselblinker	O	
3 Eine LED pulsieren lassen	O	
4 Gleichzeitiges Licht- und Tonsignal	O	
5 Eine LED per Tastendruck aktivieren	O	
6 Eine Ampel programmieren	O	
7 Eine RGB-LED ansteuern	O	
8 Der Bewegungsmelder	O	
9 Licht messen - Fotowiderstand	O	
10 Drehregler – Drehpotentiometer	O	
11 Temperatur messen	O	
12 Entfernung messen per Ultraschall	O	
13 Infrarotfernbedienung	O	
14 Servomotor ansteuern	O	
15 LCD Display mit I^2C Schnittstelle	O	
16 Relais verwenden	O	
17 Schrittmotor	O	
18 Feuchtigkeitssensor	O	
19 Tropfensensor	O	
20 RFID Chipkarten verwenden	O	
21.1 Das 4x4 Tastenfeld	O	
21.2 Tastenfeld Teil2: Schließsystem	O	
22 Töne und Musik erzeugen	O	
23 Arduino Neigungssensor SW-520	O	
24 Vierstellige 7-Segment-Anzeige	O	
25 Bluetooth Modul HC-05 & HC-06	O	
26 WS2812 LED programmieren	O	

Inhaltsverzeichnis

1. Vorwort und didaktische Überlegungen

Aller Anfang ist schwer - jedoch nicht mit dieser Anleitung für Arduino. Sie soll als Grundlage zum Erlernen der Arduinoplattform dienen und Anfängern einen didaktisch fundierten, einfachen, interessanten und eng geleiteten Einstieg in die Arduinothematik geben. Der Kompetenzerwerb ist durch dieses Heft in diversen Bereichen des Mikrocontrollings so breit gefächert, dass der Leser befähigt wird, sich selbstständig in fortgeschrittene Themengebiete einzuarbeiten. Dazu gehört beispielsweise das Erlernen weiterer Programmiermöglichkeiten oder die Verwendung weiterer Module.
Dieses Werk ist in einem Zeitraum von fast zehn Jahren parallel zu einer Technikfortbildung für Lehrkräfte und dem Unterrichtseinsatz mit Schülern verschiedener Altersklassen entstanden. Mit den Jahren wurde es mehrfach evaluiert, aktualisiert und erweitert. Entstanden ist ein Arbeitsheft, das unterrichts- oder kursbegleitend eingesetzt werden kann.

Einsatz im Unterricht

Die **didaktische Konzeption** dieser Anleitung basiert darauf, dass die Lernenden weitgehend **selbstständig** mit der Anleitung und den entsprechenden Elektronikbauteilen **arbeiten**. Die Kursleitung gibt für jede Anleitung die notwendigen Arbeitsmaterialien aus und steht den Lernenden danach ausschließlich beratend zur Seite, beispielsweise um einzelne Teilnehmer oder Kleingruppen zu **fördern**. Gerade in der Anfangsphase kann es passieren, dass die Lernenden Fehler in der Elektronik, der Programmiersyntax oder der Programmlogik nicht selber finden oder erkennen können. Ein Gefühl für **Problemlösungen** dieser Art wird sich bei den Lernenden jedoch sehr schnell entwickeln. Wichtig ist in dieser Hinsicht, dass man vor den praktischen Übungen die **theoretische Einführung** liest, um bei den späteren Praxisaufgaben nicht an fehlendem Grundwissen zu scheitern.

Durch die **freie Arbeitsweise** erreicht man eine optimale und **effektive Lernzeit**, da **jeder in seinem eigenen Tempo** arbeiten und lernen kann. Darüber hinaus entwickelt sich mit der Erarbeitung der einzelnen Anleitungen schnell eine projektähnliche Gruppendynamik, da zwar jeder für sich arbeitet, aber dennoch alle das gleiche Ziel verfolgen. Das gegenseitige Erklären und Helfen **unterstützt den Lernprozess** aller Beteiligten und **fördert die Teamfähigkeit**.
In vielen Arbeitsphasen ergeben sich **Differenzierungsmöglichkeiten**, indem man Schüler dazu anleitet, die genannten Beispiele zu erweitern oder die eigene Idee **kreativ** in die Lösung der Aufgabe mit einfließen zu lassen.

1.1 Vorbereitung und Materialbeschaffung

Für die Übungen in diesem Heft werden folgende Materialien benötigt:

01. Mikrocontroller (UNO)	11. Relaiskarte	21. Bewegungsmelder
02. Breadboard	12. Infrarotfernbedienung	22. Temperatursensor TMP36
03. 65x Breadboardkabel	13. Schrittmotor mit Treiberplatine	23. Fotowiderstand
04. 10x Breadboardkabel w/w	14. Tropfensensor	24. Drehpotentiometer
05. 10x Breadboardkabel m/m	15. Feuchtigkeitssensor	25. Taster (groß)
06. Servo	16. RFID-KIT	26. Taster (klein)
07. je 20 LEDS (grün, rot, gelb, blau, weiß)	17. je 20 Widerstände (100,200,300,1K,10K Ohm)	27. Piezospeaker
08. RGB-LED	18. Ultraschallsensor	28. I²C LCD
09. Infrarotsender und -empfänger	19. Bluetooth Modul HC-05	29. Diode
10. Vierstellige 7-Segment Anzeige	20. Neigungssensor	30. Lautsprecher
Ohne Abb.: Ampelmodul	Ohne Abb.: WS2812 LED Modul mit 8 LEDs	

1.2 Was ist Arduino?

Arduino ist eine Open-Source-Elektronik-Prototyping-Plattform für flexible, einfach zu bedienende Hardware und Software im Bereich Mikrocontrolling. Es ist geeignet, um in kurzer Zeit spannende und spektakuläre Projekte zu verwirklichen. Viele davon lassen sich unter dem Begriff „Arduino" beispielsweise bei YouTube finden. Es wird vor allem von Künstlern, Designern, Tüftlern und Bastlern verwendet, um kreative Ideen zu verwirklichen. Aber auch in Schulen, Hochschulen und Universitäten wird die Arduino Entwicklungsumgebung zunehmend eingesetzt, um Lernenden einen kreativen und spannenden, aber vor allem auch einfachen Zugang zum Thema „Mikrocontrolling" zu ermöglichen. Auch Themengebiete wie „Automatisierungstechnik", „Robotik" etc. lassen sich mit der Arduino Entwicklungsumgebung erarbeiten.

1.3 Entwicklung – Arduino, Funduino, Genuino

Die Geschichte des Arduino Mikrocontrollings begann im Jahr 2005, als die beiden „Tüftler" Massimo Banzi und David Cuartielles das erste Mikrocontroller Board entwickelten und der Programmierer David Mellis die zugehörige Syntax schuf, welche auf den Programmiersprachen C++, C und Assembler beruht. Das Projekt untersteht einer Creative-Commons-Lizenz. Somit wurde Arduino weitgehend zu einer Open-Source Plattform, wodurch die Verbreitung und Entwicklung stark forciert wurde.
Im Lauf der Jahre gerieten die Gründer der Arduinoplattform (Arduino LLC) und die Produzenten der offiziellen Arduinoboards (Arduino S.r.l) in Streit, da beide Gruppen den Markennamen "Arduino" für sich beanspruchten. Zum damaligen Zeitpunkt konnte nicht eindeutig geklärt werden, welche Partei der rechtmäßige Inhaber der Marke sei - ein Rechtsstreit mit weitreichenden Folgen entstand. Die Onlineplattform spaltete sich in mehrere Internetpräsenzen auf (www.arduino.org, www.arduino.cc), beide Streitparteien vertrieben ihre eigenen Mikrocontroller, jedoch beide unter dem gleichen Markennamen „Arduino". Im Jahr 2015 wurde vom Gründungsmitglied M.Banzi die Marke „Genuino" vorgestellt. Genuino war fortan die Bezeichnung für Mikrocontroller Boards, die außerhalb der Vereinigten Staaten von Amerika verkauft werden sollten. Erst auf der World Maker Faire 2016 verkündeten Arduino LLC und Arduino S.r.l. den Zusammenschluss der streitenden Parteien unter einer neu gegründeten Arduino Holding. Während der Jahre entstanden aufgrund der Unsicherheit bzgl. der Namensrechte viele neue Marken und Namen anderer Hersteller von arduinokompatiblen Mikrocontrollerboards. Auf Basis der quelloffenen „Open Source" Grundlage sind die Boards zwar in den meisten Fällen technisch baugleich, dürfen jedoch nicht mit den Namen „Arduino" versehen werden. Diese Boards werden dann als „Arduino-Clon" oder „Arduino-kompatibel" bezeichnet. Als Beispiel ist in dem namensrechtlich-problematischen Zeitraum die deutsche Firma Funduino GmbH entstanden. Die Firma hat bereits in der Entstehungsphase im Jahr 2010 umfangreiche Unterrichtsmaterialien und Lernsets für den schulischen Bereich entwickelt. Da es nicht möglich war, eine konstruktive Zusammenarbeit mit Arduino herzustellen, produziert die Firma seither eigene arduino-kompatible Mikrocontrollerboards.

Heute stellt der Begriff Arduino die Definition für flexible, einfach zu bedienende Hard- und Software im Bereich Mikrocontrolling dar. Arduino ist für die Verwirklichung von spektakulären Projekten prädestiniert. Aufgrund der geringen Anschaffungskosten, der praxisnahen Entwicklungsumgebung und dem daraus resultierenden didaktischen Mehrwert, sowie den nahezu unendlichen Kombinationsmöglichkeiten der unzähligen Sensoren und Aktoren findet der Themenbereich Mikrocontrolling mehr und mehr Einzug in den globalen Bildungsbereich.

2. Hardware und Software

Der Begriff Arduino wird im allgemeinen Wortgebrauch gleichermaßen für die verschiedenen „Arduinoboards" (also die Hardware) als auch für die Programmierumgebung (Software) verwendet.

2.1 Hardware

Neben dem Mikrocontroller, Sensoren und Aktoren benötigt man als Basis für schnelle und flexible Versuchsaufbauten Steckkabel in Verbindung mit einem Breadboard. Dadurch erspart man sich zeitraubende Lötarbeiten. Des Weiteren eignen sich Leuchtdioden sehr gut, um die Signalausgabe des Boards zu überprüfen.

2.1.1 Der Mikrocontroller

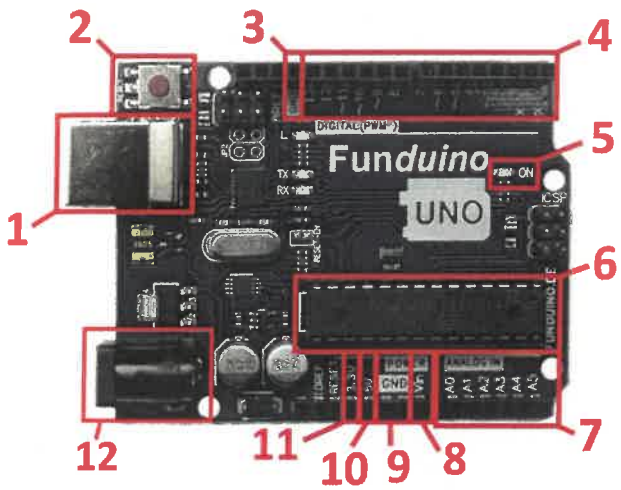
Der „Arduino" ist ein sogenanntes Mikrocontroller-Board (im weiteren Verlauf auch „Board" genannt). Es handelt sich dabei um eine Leiterplatte (PCB – *„Printed circuit board"*) mit einer Vielzahl von Elektronikbauteilen rund um den eigentlichen Mikrocontroller-Chip. Am Rand des Boards befinden sich viele Steckplätze (Pins genannt), an denen man die unterschiedlichsten Module wie Sensoren und Aktoren anschließen kann. Dazu gehören: Schalter, LEDs, Ultraschallsensoren, Temperatursensoren, Drehregler, Displays, Motoren, Servos usw.

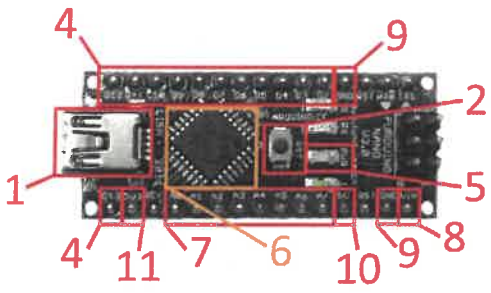
Es gibt sehr viele verschiedene Versionen von Mikrocontrollerboards, die mit der Arduino-Software verwendet werden können. Dazu gehören sowohl viele verschiedene große und kleine Boards mit der offiziellen „Arduino" Bezeichnung als auch eine Vielzahl von häufig günstigeren Arduino-kompatiblen Boards. Die Boards unterscheiden sich nur in kleinen Details, wie die Menge der digitalen Pins oder der Speicherkapazität.

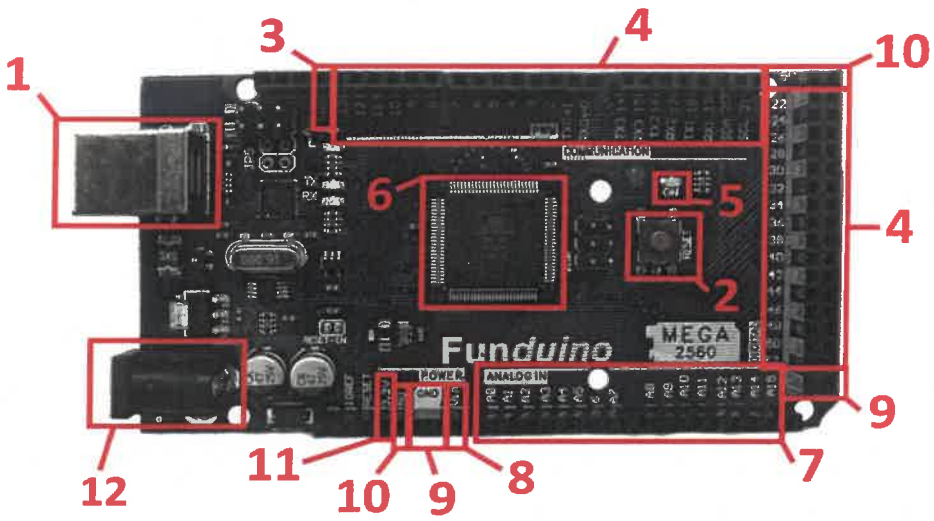
Diese Anleitung wurde mit einem Arduino-kompatiblem UNO-Board der Marke Funduino erstellt. Dennoch kann jeder beliebige Arduino-kompatibler Controller mit dieser Anleitung verwendet werden.

2.1.1 Der Mikrocontroller

Die folgenden drei Boards sind die bekanntesten Mikrocontrollerboards, die mit der Arduinosoftware verwendet werden: UNO, NANO und MEGA



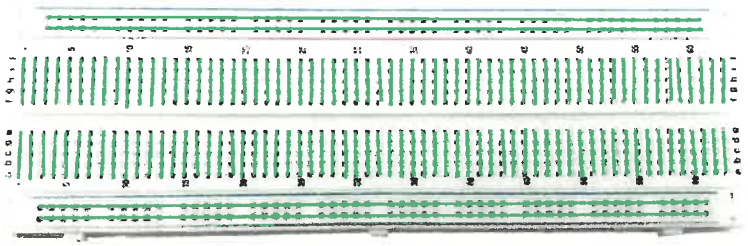




Legende:

1. USB-Anschluss	5. Power-ON Leuchte	9. GND (*Ground* bzw. „-“)
2. Reset-Knopf	6. Mikrocontroller	10. 5V Ausgang vom Spannungsregler
3. GND (*Ground* bzw. „-“)	7. Analoge Eingänge	11. 3,3V Ausgang vom Spannungsregler
4. Digitale Ein- und Ausgänge	8. Spannungsversorgung per Pin (*Vin*)	12. Externe Spannungsversorgung (*7-12V*)

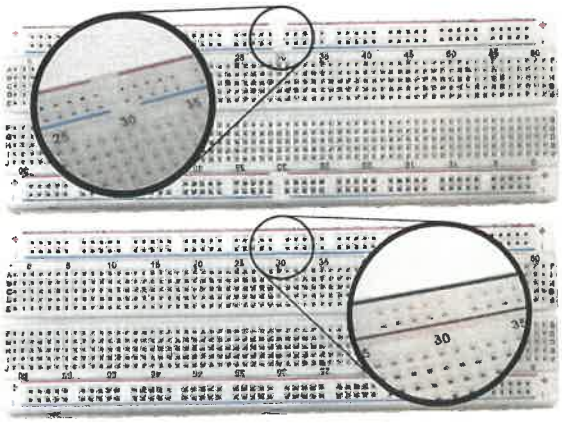
2.1.2 Zubehör - Das Breadboard



Ein Breadboard oder auch „Steckbrett“ ist ein gutes Hilfsmittel, um Schaltungen aufzubauen, ohne löten zu müssen. In einem Breadboard sind immer mehrere

Kontakte miteinander verbunden. Daher können an diesen Stellen viele Kontakte hergestellt werden, ohne dass sie verlötet oder verschraubt werden müssen.

Es gibt sehr viele verschiedene, große, kleine, farbige, transparente Breadboardversionen. Auch die Verdrahtung im Inneren des Breadboards kann unterschiedlich sein. In der Regel sind die Kontakte so miteinander verbunden, wie es auf dem Bild dargestellt ist. Die Äußeren Linien sind durchgehend miteinander verbunden und die kleinen Linien im Inneren Bereich sind jeweils mit fünf Steckplätzen untereinander verbunden. Die kleinen inneren Segmente eignen sich für detaillierte Aufbauten, während die äußeren Linien in der Regel für die Verteilung der Spannungsversorgung genutzt werden.



Es gibt auch Breadboards, bei denen die äußeren Kontakte in der Mitte noch einmal oder sogar mehrfach unterteilt sind. Daher sollte man vor dem Beginn mit der Arbeit prüfen, wie das verfügbare Breadboard kontaktiert ist. Häufig sind die äußeren Kontakte mit roten und blauen Linien versehen, an denen man erkennen kann, ob die Kontakte in einer durchgehenden Linie miteinander verbunden sind oder nicht.

2.1.3 Zubehör - Leuchtdioden

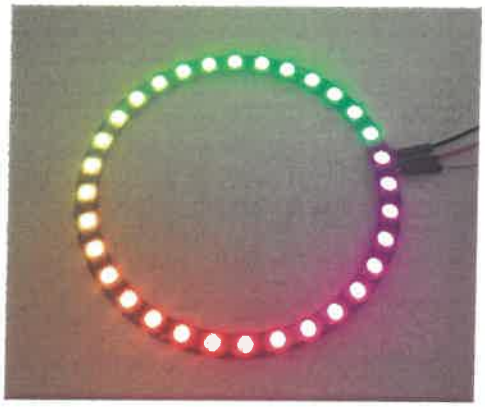
Mit LEDs kann man sehr schnell die Ergebnisse eines Projekts testen. Daher sind sie für nahezu alle Arduinoprojekte nützlich. **Die wichtigsten Informationen über Leuchtdioden:**

- Der Strom kann nur in einer Richtung durch die LED fließen. Daher muss sie richtig angeschlossen werden. Eine LED hat einen längeren und einen kürzeren Kontakt. Der längere Kontakt ist „+“ und der kürzere ist „-“.
- Eine LED ist für eine bestimmte Stromstärke ausgelegt. Wird diese Stromstärke unterschritten, leuchtet die LED weniger hell oder sie bleibt aus. Wird die Stromstärke jedoch überschritten brennt die LED sehr schnell durch und wird an den Kontakten heiß (Achtung, Verbrennungsgefahr!). Eine zu hohe Stromstärke kann auftreten, wenn die für die LED maximale Spannung überschritten wird. Schließt man bspw. eine LED direkt an den 5V Ausgang an, ist sie umgehend defekt. Daher nutzt man bei der Verwendung von LEDs an Arduinoboards immer einen Vorwiderstand.
- Typische Spannungswerte nach LED Farben: Blau: 3,1V, Weiß: 3,3V, Grün: 3,7V, Gelb: 2,2V, Rot: 2,1V. Die exakte vorgesehene Spannung kann man im Datenblatt zur jeweiligen LED nachlesen.

• Unverbindliche Empfehlung für Widerstände bei Verwendung der folgenden LED-Farben an den 5V Pins des Mikrocontrollers:

LED-Farbe:	Weiß	Rot	Gelb	Grün	Blau	Infrarot-LED
Widerstand:	100 Ohm	200 Ohm	200 Ohm	100 Ohm	100 Ohm	100 Ohm

Intelligente Leuchtdioden - NeoPixel



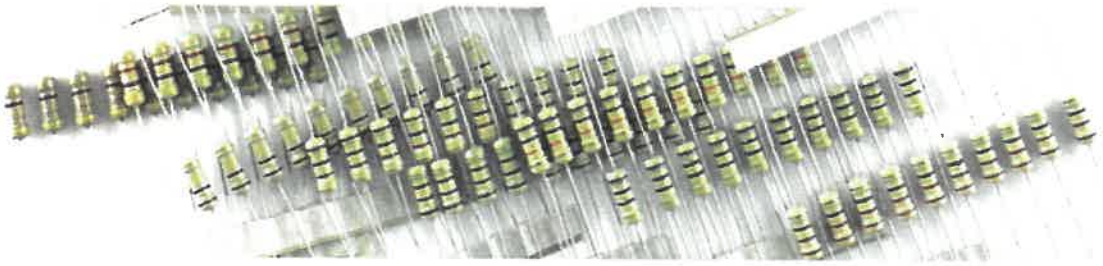
Neben den Standard-LEDs gibt es noch die "Luxusvariante" in Form von WS2811-, WS2812- und WS2812B-LEDs. Häufig werden diese LEDs auch als NeoPixel bezeichnet. Es handelt sich dabei um farbige (RGB) LEDs mit einem integrierten Chip, der dazu dient, die RGB-LEDs zu kontrollieren. Der Vorteil liegt darin, dass die LED über nur drei statt vier Kontakte angesteuert wird und es können sehr viele LEDs hintereinandergeschaltet werden, ohne dass weitere separate Kabel benötigt werden. Vorgefertigte WS2812-LED-Kombinationen findet man dann häufig in Form von LED-Ringen oder farbige LED-Lichterketten. Auf dem Bild ist ein WS2812-Ring zu sehen, bei dem die LEDs mit nur drei Kabeln in Regenbogenfarben angesteuert werden.

2.1.4 Zubehör – Widerstände

Ein elektrischer Widerstand ist ein passives elektrisches Bauelement und wird insbesondere im Bereich des Mikrocontrollings häufig in Stromkreisen integriert. Widerstände werden verwendet, um den elektrischen Strom zu begrenzen. Dadurch können Bauteile im Stromkreis geschützt werden. Typische Leuchtdioden haben z.B. eine vorgesehene Stromstärke von 20mA. Fließt in einem Stromkreis jedoch viel mehr Strom, so kann die Leuchtdiode kaputt gehen.

Widerstände können den elektrischen Strom in einer Schaltung auch aufteilen. Dadurch wird es z.B. möglich, Sensorwerte von Sensoren auszulesen, die in Abhängigkeit des zu messenden Wertes ihre elektrische Leitfähigkeit verändern.

Es gibt sehr viele verschiedene Widerstände, da je nach Anwendungszweck ein individueller Widerstandswert benötigt wird. Die Auswahl des passenden Widerstands ist nicht leicht. Daher wird in den Anleitungen der notwendige Widerstandswert vorgegeben.



2.1.4 Zubehör - Sensoren und Aktoren

Die Möglichkeiten, Sensoren oder Aktoren am Arduino-Mikrocontroller zu verwenden, sind nahezu unerschöpflich. Das liegt daran, dass die Arduino Entwicklungsumgebung kein in sich geschlossenes System ist, bei dem nur vorgefertigte Module verwendet werden können. Ganz im Gegenteil lassen sich unzählige elektronische Module aus dem Elektronikhandel in irgendeiner Weise mit Arduino verwenden, beispielsweise durch das auslesen digitaler oder analoger Werte. An dieser Stelle folgt beispielhaft eine (winzige) Auswahl an typischen Modulen, die in Kombination mit Arduino Mikrocontrollern verwendet werden können.

(S) = Sensor, (A) = Aktor, (K) = Kombiniertes Modul mit diversen Funktionen

1. Feuchtigkeit (S)
2. Strecke (Ultraschall) (S)
3. Bewegungen (S)
4. Luftfeuchtigkeit (S)
5. Luftdruck (S)
6. Stromstärke (S)
7. GPS-Empfänger (S)
8. Drehreglerposition (S)
9. Temperatur (S)
10. Tastendruck (S)
11. Tropfen (S)
12. Schieberegler (S)
13. Neigung (S)
14. Schallpegel (S)
15. Neigung / Beschleunigung (S)
16. Elektrische Spannung (S)
17. Farberkennung (S)
18. Infrarotsignale (S)
19. Leuchtstärke (S)
20. GSM-Mobilfunk (K)
21. pH-Wert (S)
22. Magnetfelder (S)
23. Flammen / Feuer (S)
24. Herzfrequenz (S)
25. RFID - Funkcode (K)
26. Wasserstand / Pegel (S)
27. Gas (CO, Alkohol etc.) (S)
28. Fingerabdruck (K)
29. Stöße/Erschütterung (S)
30. Mikroschalter (S)
31. Lichtschranken (S)
32. Vibrationen (S)
33. UV-Licht (S)
34. Feinstaub (S)
35. Abstand (Infrarot) (S)

2.2 Software

Die Software, mit welcher der Mikrocontroller programmiert wird, ist Open-Source-Software und kann auf www.arduino.cc kostenlos heruntergeladen werden. In dieser „Arduino-Software" schreibt man dann kleine Programme, die der Mikrocontroller später ausführen soll. Diese kleinen Programme werden „Sketch" genannt. Per USB-Kabel werden die fertigen Sketches dann auf den Mikrocontroller übertragen. Wie das funktioniert wird im Themengebiet „Programmieren" behandelt.

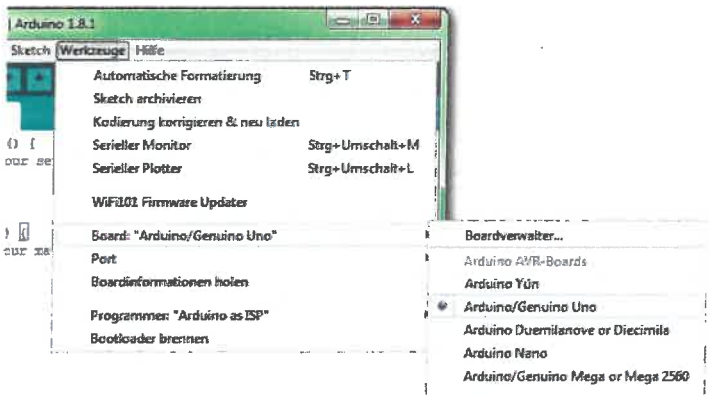
2.2.1 Installation

Nun muss nacheinander die Arduino-Software und der USB-Treiber für das Arduinoboard installiert werden.

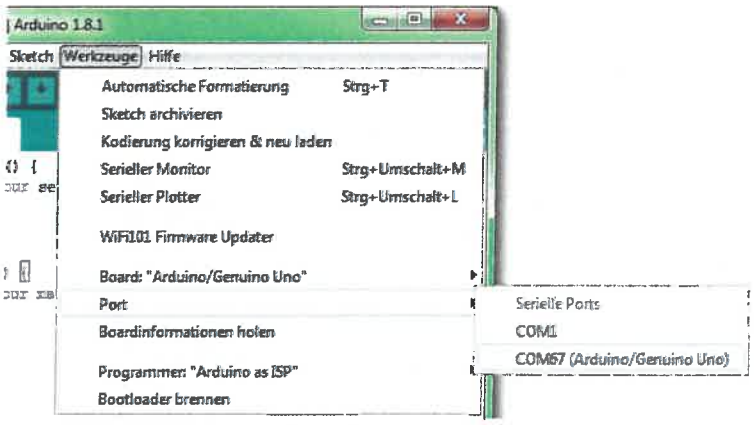
2.2.2 Einrichtung der Arduinosoftware

Die jeweils aktuellste Version der Arduinosoftware kann auf der Internetseite www.arduino.cc heruntergeladen werden. Nach dem download der Programmdatei beginnt die Installation. Sollte die Installation nicht automatisch beginnen, muss sie mit einem Doppelklick auf das heruntergeladene Programm gestartet werden. Während dieser Installation sollte noch kein Arduinoboard am Computer angeschlossen sein.
Nach der erfolgreichen Installation öffnet man den Arduinio-Softwareordner und startet das Programm mit der Datei „arduino.exe".

Zwei wichtige Einstellungen gibt es im Programm zu beachten.
a) Es muss das Board ausgewählt werden, das man am Computer anschließen möchte. Das „Funduino Uno" Board wird hier als „Arduino Uno" erkannt und das „Funduino MEGA2560" Board entsprechend als „Arduino MEGA 2560".



b) Es muss der richtige „Serielle Port" ausgewählt werden. Das ist wichtig, damit der PC zuordnen kann, an welchem USB Anschluss das Board angeschlossen ist. Dies ist jedoch nur möglich, wenn der Treiber richtig installiert wurde und der Mikrocontroller angeschlossen ist.



Wenn es nicht eindeutig ist, welcher Port zum jeweils angeschlossenen Mikrocontroller gehört, kann dies mit dem folgendem Ablauf geprüft werden. Ohne dass der Arduino-Mikrocontroller am PC angeschlossen ist, klickt man in der Software in dem Untermenü *Werkzeuge* auf „Port". Dort werden schon ein oder mehrere Ports zu sehen sein, wie „COM1", „COM4", „COM7" ... Die Anzahl der angezeigten Ports ist dabei unabhängig von der Anzahl der USB-Anschlüsse des verwendeten Computers. Wenn später das Board richtig installiert und angeschlossen ist, wird hier ein weiterer Port angezeigt.

2.2.3 USB-Treiberinstallation

Im Idealfall wird bei der Treiberinstallation von Arduino-Boards oder Arduino-kompatiblen Boards mit originalem ATMEL-Chip (bspw. UNO oder MEGA von Arduino oder Funduino) das Mikrocontrollerboard an den PC angeschlossen und automatisch installiert.
Der Treiber wird jedoch je nach System nicht immer automatisch erkannt und installiert. Man sollte in dem Fall im Verlauf der Installation den Treiber selber auswählen. Er befindet sich in dem Arduino-Programmordner in dem Unterordner „Drivers".
Kontrolle: In der Windows-Systemsteuerung des Computers findet man den *„Gerätemanager"*. Nach einer erfolgreichen Installation ist das Arduinoboard hier aufgelistet. Wenn die Installation nicht erfolgreich war, ist hier entweder nichts Besonderes zu entdecken oder es ist ein unbekanntes USB-Gerät mit einem gelben Ausrufezeichen vorhanden. In diesem Fall klickt man im Gerätemanager auf das unbekannte Gerät, wählt die Option „Treiber aktualisieren" und folgt dann den Anweisungen auf dem Bildschirm.

Neben den Arduino-Boards oder Arduino-kompatiblen Boards gibt es mittlerweile sehr viele weitere Mikrocontrollerboards, die mit der Arduino-Entwicklungsumgebung kompatibel sind, jedoch zum Teil auf völlig anderen Mikrocontrollern basieren. Daher sind diese Treiber auch nicht in der Arduinosoftware enthalten und müssen separat installiert werden.
Der bekannteste Chipsatz, der oft in den günstigsten Boards Verwendung findet, ist der „**CH340**" USB Chipsatz. Dieser wird in einer Vielzahl von UNO-, MEGA-, NANO- und vergleichbaren Mikrocontrollerboards verwendet.
Für die Installation lädt man sich vom Hersteller oder Händler die passenden Treiber herunter. Dabei muss in der Regel darauf geachtet werden, ob man Windows, Mac oder Linux verwendet. Bei der anschließenden Installation werden für gewöhnlich Administrationsrechte benötigt, damit der Treiber ordnungsgemäß installiert werden kann.
Praxistipp: Vor der Arbeit mit dem Mikrocontroller prüfen, ob die Installation von externen Treibern am gewünschten Computer möglich ist. Insbesondere in größeren EDV-Räumen mit Rechtevergaben etc. ist das wichtig, um unerwünschte Verzögerungen zu vermeiden.

2.2.4 Bibliotheken hinzufügen

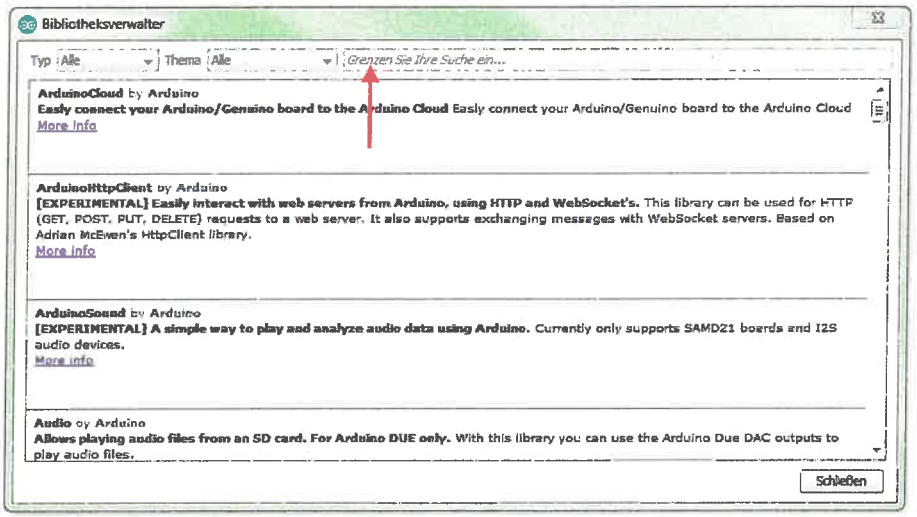
Eine Bibliothek (auch *Library* genannt) ist für einige Projekte sinnvoll, da diese die Programmierung vereinfachen kann. Durch die Verwendung einer Bibliothek kann in einem Sketch auf Funktionen zurückgegriffen werden, sodass diese nicht komplett im Sketch ausgeschrieben werden müssen. Im praktischen Teil dieser Anleitung wird mehrfach auf solche Bibliotheken verwiesen. Diese müssen dann erst zur Arduinosoftware hinzugefügt werden, damit sie verwendet werden können. Für das Hinzufügen einer Bibliothek zur Arduinosoftware gibt es verschiedene Möglichkeiten.

Die einfachste Möglichkeit bietet sich durch die Funktion „Bibliotheken verwalten...". Diese befindet sich in der Arduinosoftware im Bereich *„Sketch > Bibliothek einbinden > Bibliotheken verwalten..."* Dort kann über das Suchfeld die gewünschte Library gesucht und direkt installiert werden.

Nach der erfolgreichen Installation kann die Bibliothek direkt verwendet werden.

Mit der Installation von Programmbibliotheken werden häufig auch gleichzeitig Beispielsketche zur Arduinosoftware hinzugefügt. Diese Beispiele befinden sich unter „Datei > Beispiele" und können einen guten Einblick in die einzelnen Funktionen der jeweiligen Bibliothek geben.

Es gibt auch die Möglichkeit eine Bibliothek auf einer externen Seite herunterzuladen und über die „.ZIP Bibliothek hinzufügen..." Funktion einzubinden. Jedoch gestaltet sich diese Weise umständlicher als die zuvor beschriebene.



Wer sich einen detaillierten Überblick über die Funktionen einer Library verschaffen möchte, kann die entsprechenden Dateien auf der Festplatte im Arduinoverzeichnis suchen und dann mit einem Editor oder auch mit der Arduinosoftware öffnen.

Eine Library besteht in der Regel aus mindestens zwei Dateien, mit den Dateiendungen „ .h" und „ .cpp". Die Datei mit der Endung „ .h" enthält bzw. beschreibt die Funktionen, während sich in der Datei mit der Endung „ .cpp" der eigentliche Quellcode befindet.

Es ist auch möglich, selber Libraries zu erstellen. Dies ist aber nur für Fortgeschrittene Nutzer sinnvoll.

2.3 Alternative Software

Neben der Arduino-Software gibt es weitere Möglichkeiten, Arduinoboards zu programmieren. Diese basieren häufig auf die Programmierung mittels einer grafischen Programmieroberfläche, die insbesondere im Bildungsbereich durch „Scratch" oder die Programmierung von LEGO Mindstroms bekannt ist.

Bei dieser Art der Programmierung werden alle Elemente im Wesentlichen per Drag and Drop zusammengebaut. Die Elemente werden durch ihre intuitiv verständliche Darstellung repräsentiert, wie z. B. Programmierbefehle durch Bausteinbilder. Diese Art der Programmierung bewirkt gerade bei Schülern eine zusätzliche Motivation, da das Tippen von Code zunächst häufig als kompliziert erachtet wird. Ein weiterer Vorteil besteht darin, dass sich durch das Wegfallen des Eintippens von Programmcodes auch keine Syntaxfehler einschleichen können. Ein Nachteil besteht jedoch darin, dass das „echte" Schreiben von Programmcodes auf diesem Wege nicht gelernt werden kann.

2.3.1 Open Roberta (deutsch)

Im **deutschen Sprachraum** ist „Open Roberta" (https://lab.open-roberta.org) sehr bekannt. Mit Openroberta können nicht nur Arduinoboards, sondern auch viele andere Mikrocontroller und Roboter programmiert werden. Es ist also hardwareübergreifend funktional und dadurch besonders in Bildungseinrichtungen geeignet um mit Schülern das Programmieren zu erlernen.

Für die Programmierung von Arduinoboards gibt es im Openroberta-LAB den Bereich „NEPO4ARDUINO". Im Openroberta-Lab wird für die USB-Verbindung zwischen Browser und Mikrocontroller ein zusätzliches Programm benötigt. Die Informationen zur Installation sind etwas versteckt. Man findet sie, wenn man sich durch folgenden Menüpfad klickt:

Hilfe – Allgemeine Hilfe – Vorbereitung - NEPO4ARDUINO

Das Bild zeigt ein Programm, mit dem am Arduinoboard eine LED zum Blinken gebracht wird. In diesem Fall wurde als visuelle Programmieroberfläche Openroberta verwendet.

2.3.2 Englischsprachig

Im englischen Sprachraum sind die bekanntesten visuellen Programmierplattformen **Mblock** (http://www.mblock.cc) und **S4A** „Scratch for Arduino" (http://s4a.cat). S4A ist beliebt, da es dem regulären „Scratch" sehr ähnlich ist, welches an vielen Schulen bereits im Informatikunterricht etabliert ist.

3. Programmieren

Damit ein Arduino-Mikrocontroller das macht, was der Benutzer verlangt, wird mit Hilfe der Arduino-Software ein kleines Programm geschrieben. Dieses Programm wird in der Arduino Entwicklungsumgebung als „Sketch" bezeichnet. Ein fertiger „Sketch" wird nach der erfolgreichen Kontrolle in der Arduino-Software auf den Speicher des Mikrocontrollers geladen und dort unmittelbar ausgeführt.

3.1 Grundstruktur für einen Sketch

Ein Sketch kann zunächst in drei Bereiche eingeteilt werden, wie hier farblich dargestellt.

```
int LED=9;                                                    Bereich 1
int helligkeit=0;
int x=5;

void setup()                                                  Bereich 2
{
 pinMode(LED, OUTPUT);
}

void loop()                                                   Bereich 3
{
 analogWrite(LED, helligkeit);
 helligkeit=helligkeit + x;
 delay(25);
 if(helligkeit==0 || helligkeit== 255)
 {
 x = -x;
 }
}
```

3.1.1 Bereich 1 - Variablen benennen

Im ersten Bereich werden Elemente des Programms benannt. Zum Beispiel werden dort Variablen festgelegt oder sog. Programmbibliotheken geladen. Dieser Teil ist nicht für jeden Sketch zwingend erforderlich.

3.1.2 Bereich 2 - Setup

Der zweite Bereich wird „Setup" genannt. Das Setup wird vom Board nur einmal ausgeführt und ist zwingend erforderlich für jeden Sketch, selbst wenn in diesem Bereich keine Einträge erfolgen. Im Setup wird bspw. festgelegt, welcher Pin (Steckplatz für Kabel) am Mikrocontrollerboard ein Ausgang oder ein Eingang ist. Definiert als Ausgang, kann an dem jeweiligen Pin eine Spannung ausgegeben werden (bspw. um an diesem Pin eine LED leuchten zu lassen) und definiert als Eingang kann an dem Pin eine Spannung eingelesen werden (bspw. die Spannungswerte eines Sensors).

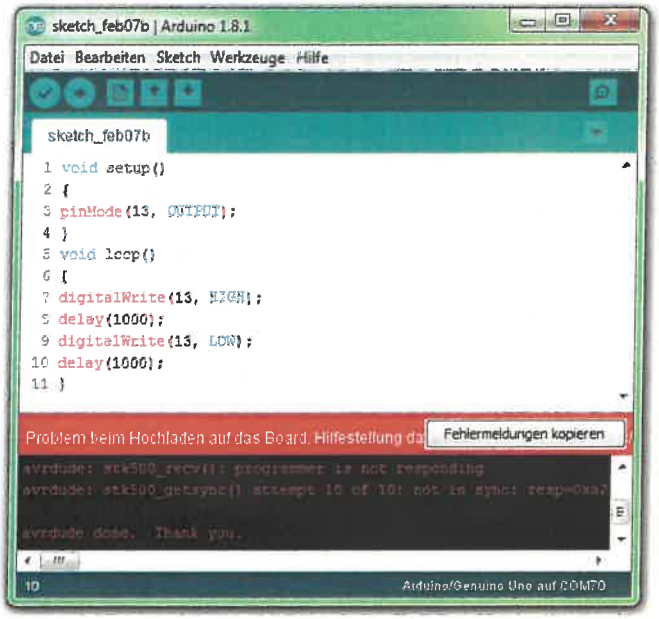
3.1.3 Bereich 3 - Loop

Der Bereich „Loop" wird vom Board kontinuierlich wiederholt und kann daher auch als Hauptteil des Sketches bezeichnet werden. Der Mikrocontroller verarbeitet den Sketch einmal komplett bis zum Ende und beginnt dann erneut am Anfang des Loop-Abschnitts. Fortgeschrittene Anwender lagern einzelne Programmabschnitte

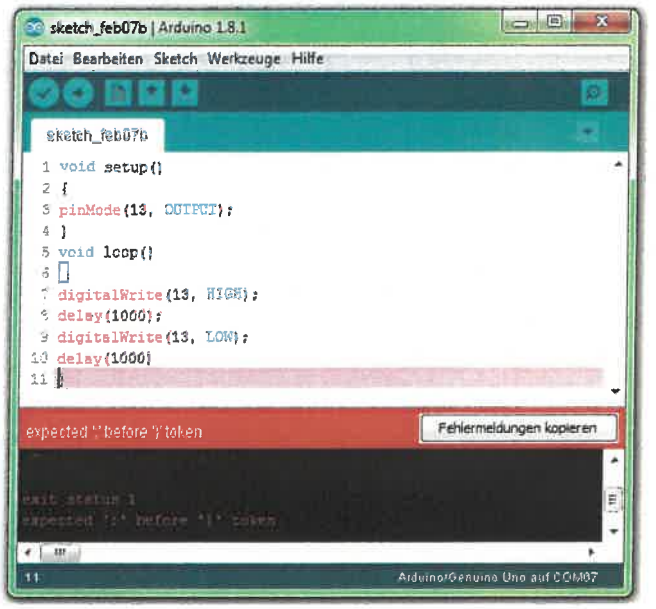
in Unterprogramme aus, die dann vom „Loop“ nur noch aufgerufen werden und ggf. auch Daten zur weiteren Verarbeitung übergeben. Diese Auslagerungen werden in dieser Anleitung nicht weiter behandelt.

3.2 Häufige Fehlerquellen

Bei der Programmierung von Mikrocontrollern können sich an vielen Stellen Fehler einschleichen. Die häufigsten „Anfängerfehler“ während der Arbeit mit der Arduino-Software sind die folgenden beiden.



1) Das Board ist nicht richtig installiert oder es ist ein falsches Board ausgewählt. Beim hochladen des Sketches wird im unteren Bereich der Software eine Fehlermeldung angezeigt, die in etwa so aussieht wie rechts abgebildet. Im Fehlertext befindet sich dann ein Vermerk „not in sync“.



2) Es gibt einen Fehler im Sketch. Beispielsweise ist ein Wort, eine Variable oder Befehl falsch geschrieben, oder es fehlt nur eine Klammer oder ein Semikolon. Im Beispiel links fehlt die geschweifte Klammer, die den Loop-Teil einleitet. Die Fehlermeldung beginnt dann häufig mit „expected...“. Das bedeutet, dass das Programm etwas erwartet, das noch nicht vorhanden ist.

3.3 Aufbau der Anleitungen

Die Struktur der folgenden Anleitungen ist jeweils sehr ähnlich.

1. Die Anleitungen beginnen mit einer Beschreibung der Aufgabe, die in der Anleitung abgearbeitet werden soll. Außerdem gibt es, falls erforderlich, eine Erklärung zu den verwendeten Bauteilen.
2. Vor dem jeweiligen Sketch befindet sich eine Skizze um die Verkabelung aller Module zu verdeutlichen.
3. Die abgedruckten Sketche in den folgenden Anleitungen bestehen gleichzeitig aus Programmcode und einer Beschreibung, welche die Wirkung des jeweiligen Programmcodes erklärt.

Im **linken Bereich ist der Programmcode** in schwarzer oder farbiger Schrift abgedruckt, während sich **im rechten Bereich** hinter dem Zeichen „//“ in grauer Schrift **die Erklärung** zum Programmcode befindet. Die Erklärungen in grauer Schrift dürfen mit in in die Arduino-Software eingegeben werden und haben keinen Einfluss auf den Ablauf des Sketches.

Beispiel:

Links: Programmcode in fetter Schrift **Rechts:** Erklärungen zum Programmcode

```
void setup()                //Hier beginnt das Setup.
{                           //Hier beginnt ein Programmabschnitt.
pinMode(13, OUTPUT);        //Pin 13 soll ein Ausgang sein.
}                            //Hier ist ein Programmabschnitt beendet.

void loop()                 //Hier beginnt das Hauptprogramm.
{                           //Programmabschnitt beginnt.
digitalWrite(13, HIGH);     //Schalte die Spannung an Pin13 ein. (LED an)
delay(1000);                //Warte 1000 Millisekunden (eine Sekunde).
digitalWrite(13, LOW);      //Schalte die Spannung an Pin13 aus.(LED aus)
delay(1000);                //Warte 1000 Millisekunden (eine Sekunde).
}                           //Programmabschnitt beendet.
```

Wer sich nach diesem System durch die Programme arbeitet, wird den Programmcode in kurzer Zeit selber durchschauen und anwenden können. Danach kann man sich selbst mit weiteren Funktionen oder Modulen vertraut machen. Diese Anleitung stellt einen Einstieg in die Arduino Entwicklungsumgebung dar und vermittelt einen Einblick in die Programmiermöglichkeiten. Einen Gesamtliste aller Programmcodes wird auf der Internetseite „www.arduino.cc“ unter dem Punkt „Reference“ aufgeführt und beschrieben.

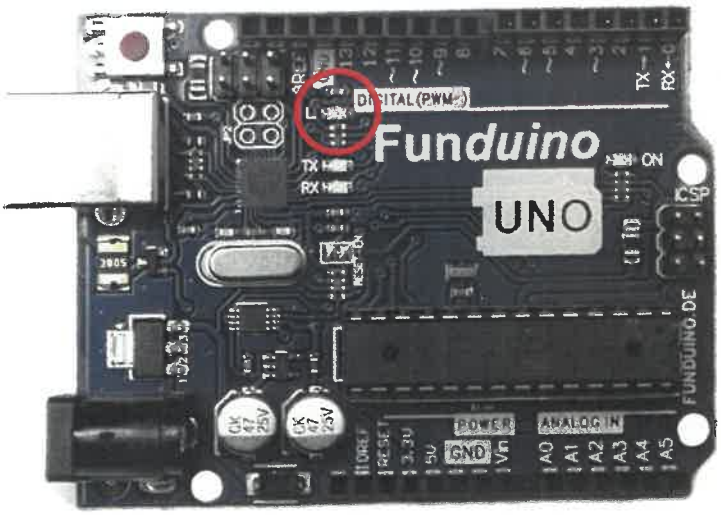
4. Praxisbezogene Anleitungen

Im folgenden Abschnitt wird anhand von praktischen Beispielen mit Bezug zu vorgegebener Hardware das Arbeiten mit der Arduino Entwicklungsumgebung erarbeitet.

4.1 Eine blinkende LED

Aufgabe: Eine Leuchtdiode soll blinken.

Materialbox
1x Arduino-Board



Auf dem Arduino Mikrocontrollerboard ist an Pin 13 bereits eine LED eingebaut (für Testzwecke). Häufig blinkt diese Lampe schon, wenn man ein neues Arduinoboard anschließt, da das Blink-Programm zum Testen des Boards je nach Hersteller bereits vorab installiert ist. Wir werden dieses Blinken jetzt selbst programmieren und die Blinkgeschwindigkeit verändern.

Schaltung:
Die auf dem Board vorhandene LED ist in dem Bild rot eingekreist. Es muss lediglich das Arduinoboard per USB-Kabel mit dem Computer verbunden werden.

Programmbereich 1 wird nicht benötigt.

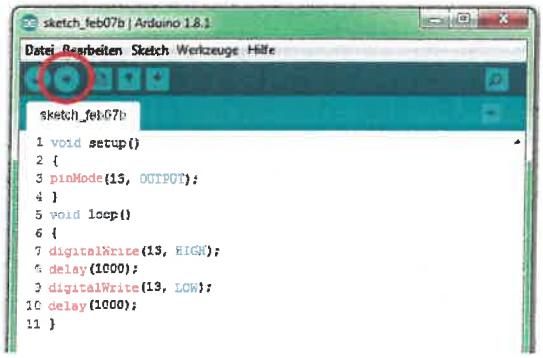
Programmbereich 2, Setup:
Wir benötigen für diese Aufgabe nur einen Pin des Mikrocontrollerboards, den Pin 13. An Pin13 soll eine Spannung ausgegeben werden, denn die LED soll schließlich leuchten. Daher muss im Setup angegeben werden, dass es sich bei dem Pin13 um einen Ausgang handelt. Die Erklärungen hinter dem doppelten Schrägstrich „//“ in grauer Schrift dürfen mit in die Arduinosoftware eingegeben werden, haben jedoch keinen Einfluss auf den Ablauf des Sketches. Das sogenannte Auskommentieren ist beim Programmieren sehr sinnvoll, um für sich selber Informationen zum Programm, oder einfach nur Geistesblitze zu hinterlassen.

Wir schreiben mitten in das weiße Eingabefeld der Arduinosoftware den folgenden Sketch. Dabei muss lediglich der fett gedruckte Programmcode auf der linken Seite abgetippt werden. Die *„auskommentierten“* Informationen zum Sketch können weggelassen werden.

```
void setup()          //Hier beginnt das Setup.
{                     //Hier beginnt ein Programmabschnitt.
pinMode(13, OUTPUT); //Pin 13 soll ein Ausgang sein.
}                     //Hier ist ein Programmabschnitt beendet.
void loop()           //Hier beginnt das Hauptprogramm.
{                      //Hier beginnt ein Programmabschnitt.
digitalWrite(13, HIGH);//Schalte die Spannung an Pin13 ein (LED an).
delay(1000);           //Warte 1000 Millisekunden (eine Sekunde).
digitalWrite(13, LOW);//Schalte die Spannung an Pin13 aus (LED aus).
delay(1000);           //Warte 1000 Millisekunden (eine Sekunde).
}                      //Programmabschnitt beendet.
```

Nach der letzten geschweiften Klammer im Loop-Teil ist der Sketch abgeschlossen. Wenn der Sketch innerhalb des *„Loop"* vom Mikrocontroller bis zur letzten Klammer abgearbeitet wurde, beginnt der Sketch wieder vorne im Loop-Teil. In diesem Beispiel geht dadurch die LED immer wieder an und aus.

Der Sketch sollte nun exakt so aussehen, wie er auf dem Bild rechts dargestellt ist. Er muss jetzt nur noch auf das Board hochgeladen werden. Das funktioniert mit der rot umkreisten Schaltfläche (Oben links in der Software).



Das Programm kann jetzt noch variiert werden. Beispiel: Die LED soll sehr schnell blinken. Dazu verkürzen wir die Wartezeiten (delay) von 1000 Millisekunden auf 200 Millisekunden.

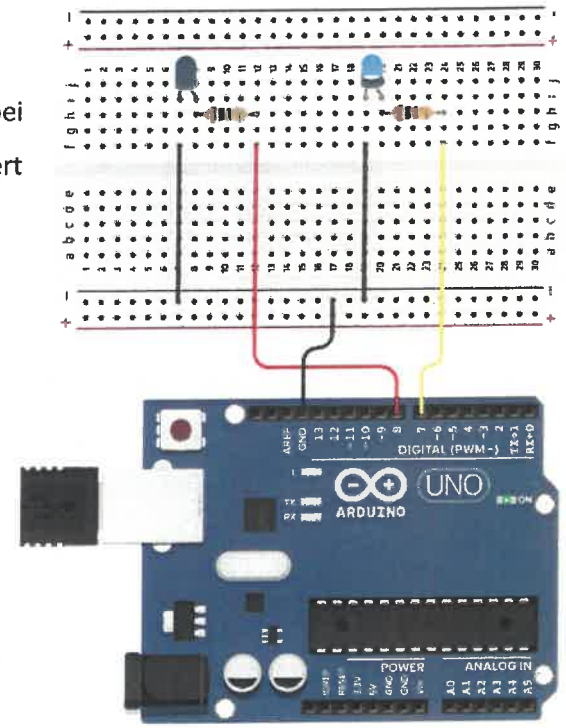
```
void setup()           //Hier beginnt das Setup
{                      //Hier beginnt ein Programmabschnitt.
pinMode(13, OUTPUT);   //Pin 13 soll ein Ausgang sein.
}                      //Hier ist ein Programmabschnitt beendet.
void loop()            //Hier beginnt das Hauptprogramm
{                      //Programmabschnitt beginnt.
digitalWrite(13, HIGH); //Schalte die Spannung an Pin13 ein (LED an).
delay(200);            //Warte 200 Millisekunden
digitalWrite(13, LOW); //Schalte die Spannung an Pin13 aus (LED aus).
delay(200);              //Warte 200 Millisekunden
}                      //Programmabschnitt beendet.
```

Der neue Sketch muss nun wieder auf das Board hochgeladen werden. Bei fehlerfreier Eingabe wird die LED nun schneller blinken.

4.2 Der Wechselblinker

Aufgabe: Zwei Leuchtdioden sollen abwechselnd blinken. Dabei kann die Blinkgeschwindigkeit und der Blinkrhythmus variiert werden.

Materialbox
1x Arduino-Board
2x Leuchtdioden (blau)
2x Widerstand 100 Ohm
1x Breadboard
Einige Steckkabel



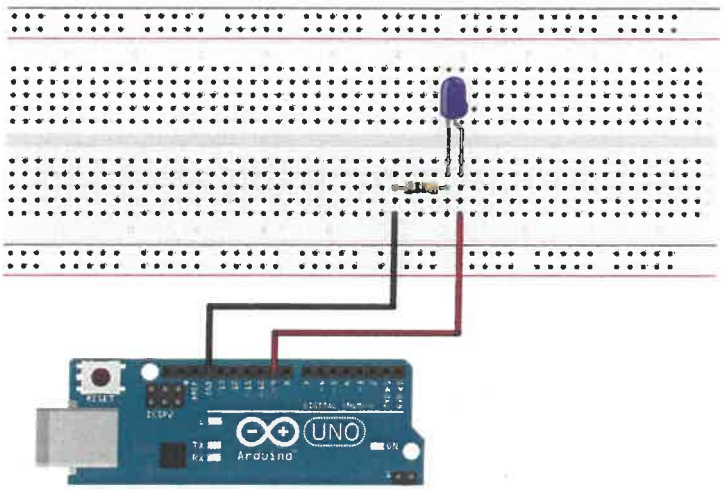
Sketch:

```
void setup()
{                    //Wir starten mit dem Setup
pinMode(7, OUTPUT); //Pin 7 ist ein Ausgang.
pinMode(8, OUTPUT); //Pin 8 ist ein Ausgang.
}
void loop()
{                          //Das Hauptprogramm beginnt.
digitalWrite(7, HIGH);     //Schalte die LED an Pin7 an.
delay(1000);               //Warte 1000 Millisekunden.
digitalWrite(7, LOW);      //Schalte die LED an Pin7 aus.
digitalWrite(8, HIGH);     //Schalte die LED an Pin8 ein.
delay(1000);               //Warte 1000 Millisekunden.
digitalWrite(8, LOW);      //Schalte die LED an Pin8 aus.
}                          //Hier am Ende springt das Programm an den Start des Loop-Teils.
Also: schalte die LED an Pin7 an… usw…
```

Je kürzer das Delay, also die Pause zwischen dem Wechsel der Leucht- und Ruhephasen der LEDs, gewählt wird desto schneller ist der Blinkrhythmus. Der Rhythmus kann dabei so schnell eingestellt werden, dass das menschliche Auge den Wechsel der Leucht- und Ruhephasen nicht mehr erkennen kann. Mit diesem Sketch kann man die Blinkfolge von Feuerwehr- und Polizeiwagen verschiedenster Länder als Modell nachbauen. Programmiere dieses Blaulicht:
Links-Pause-Links-Pause-Links-Pause-Rechts-Pause-Rechts-Pause-Rechts-Pause

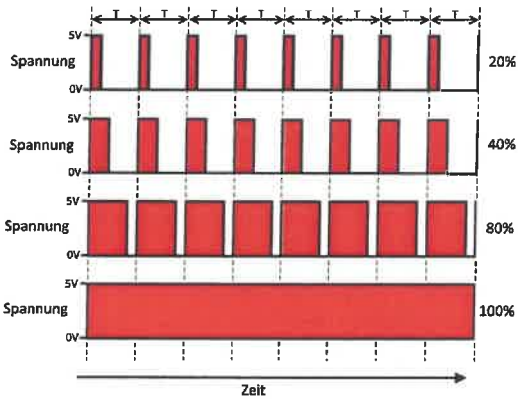
4.3 Eine LED pulsieren lassen

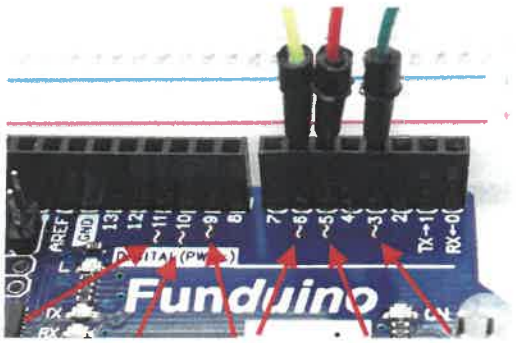
Aufgabe: Eine LED soll pulsierend heller und dunkler werden. (Auch als engl. „faden“ bezeichnet)



Materialbox
1x Arduino-Board
1x blaue LED
1x Widerstand mit 100 Ohm
1x Breadboard
Einige Steckkabel

Der *Arduino* ist ein digitaler Mikrocontroller. Er kennt an seinen Ausgängen nur „5 Volt an“ oder „5V aus“. Um die Helligkeit einer LED zu variieren, müsste man die Spannung jedoch variieren können. Zum Beispiel 5V wenn die LED hell leuchtet und 4 Volt, wenn sie etwas dunkler leuchtet usw. Das funktioniert an digitalen Pins nicht. Es gibt jedoch eine Alternative. Sie nennt sich Pulsweitenmodulation (PWM). Die PWM lässt die 5V Spannung pulsieren. Die Spannung wird also im Mikrosekundenbereich ein und ausgeschaltet. Bei einem geringen PWM-Wert ist das 5V Signal kaum noch vorhanden. Bei einem hohen PWM-Wert liegt das 5V Signal nahezu durchgehend am jeweiligen Pin an. (Da dies eine sehr kompakte Zusammenfassung ist, sollte man sich ggf. detaillierte Quellen zum Themengebiet PWM ansehen). Mit der PWM-Funktion kann man bei LEDs einen ähnlichen Effekt erreichen, als würde man die Spannung variieren. Nicht alle digitalen Pins am Board haben die PWM-Funktion. Die Pins an denen die PWM-Funktion am Arduino Mikrocontroller genutzt werden kann, sind auf der Platine durch eine kleine Welle vor der Zahl mit der Pinnummer gekennzeichnet.



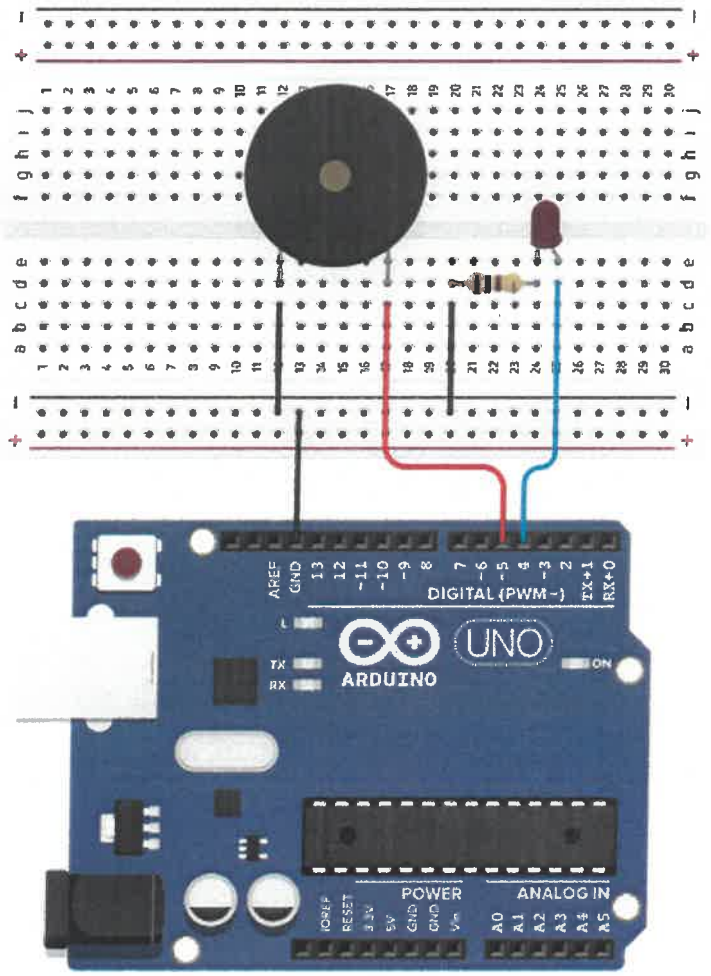


Der folgende Sketch erzeugt ein pulsieren der LED an Pin9.

```
int LED=9;          //Das Wort „LED“ steht jetzt für den Wert 9.
int helligkeit=0;   //Das Wort „helligkeit“ steht nun für den Wert, der bei der PWM
ausgegeben wird. Die Zahl 0 ist dabei nur ein beliebiger Startwert.
int x=5;      //x bestimmt die Geschwindigkeit des Pulsierens
void setup()        //Hier beginnt das Setup.
{
pinMode(LED, OUTPUT); //Der Pin mit der LED (Pin9) ist ein Ausgang
}
void loop()
{
analogWrite(LED, helligkeit);
//Mit der Funktion analogWrite wird hier an dem Pin mit der LED (Pin9) die PWM Ausgabe
aktiviert. Der PWM-Wert ist der Wert, der unter dem Namen „helligkeit“ gespeichert ist. In
diesem Fall „0“ (Siehe ersten Programmabschnitt)
helligkeit=helligkeit + x;
//Nun wird der Wert für die PWM-Ausgabe verändert. Zum Wert „helligkeit“ wird nun der Wert
x addiert. In diesem Fall: helligkeit=0+ 5. Der neue Wert für „helligkeit“ ist also nicht
mehr 0 sondern 5. Sobald der Loop-Teil einmal durchgelaufen ist, wiederholt er sich. Dann
beträgt der Wert für die Helligkeit 10. Im nächsten Durchlauf 15 usw…
delay(25);
//Die LED soll für 25ms (Millisekunden), also nur ganz kurz die Helligkeit beibehalten.
Verringert man diesen Wert, wird das Pulsieren ebenfalls schneller.
if(helligkeit==0 || helligkeit== 255)
{
//Bedeutung des Befehls: Wenn die Helligkeit den Wert 0 ODER 255 erreicht hat, wechselt der
Wert x von positiv zu negativ bzw. andersrum. Grund: Die LED wird zunächst bei jedem
Durchlauf des Loop-Teils immer ein bisschen heller. Allerdings ist irgendwann der
Maximalwert für die PWM-Ausgabe mit dem Wert 255 erreicht. Die LED soll dann wieder Schritt
für Schritt dunkler werden. Also muss der Wert x an dieser Stelle von positiv auf negativ,
bzw. von negativ auf positiv geändert werden, wenn der Wert für „helligkeit entweder 255
oder 0 erreicht hat.
x= -x;
//Das bedeutet für den nächsten Durchlauf, dass in der Zeile „helligkeit = helligkeit + x;“
die helligkeit abnimmt. Beispiel: „helligkeit=255+(-5)“. Der Wert für Helligkeit ist ab
dann 250. Im nächsten Durchlauf 245 usw. usw… Sobald der Wert für Helligkeit bei 0
angekommen ist, wechselt wieder das Vorzeichen. (Man bedenke die alte mathematische Regel:
„minus und minus ergibt plus“.)
}
}// Mit dieser letzten Klammer wird der Loop-Teil geschlossen.
```

4.4 Gleichzeitiges Licht- und Tonsignal

Aufgabe: Eine LED und ein Piezo-Lautsprecher sollen kontinuierlich blinken bzw. piepen. Zusätzlich werden in dieser Anleitung auch Variablen verwendet (Programmabschnitt 1).



Materialbox
1x Arduino-Board
1x blaue LED
1x Widerstand mit 100 Ohm
1x Piezo-Speaker
1x Breadboard
Einige Breadboardkabel

Dieses Mal nutzen wir auch den ersten Programmabschnitt. Hier werden Variablen eingetragen. Das bedeutet, dass sich nach der Festlegung in diesem Bereich hinter einem Buchstaben oder einem Wort für den gesamten folgenden Sketch eine Zahl verbirgt. Bei uns ist die LED an Pin4 angeschlossen und der Piezo-Speaker an Pin5. Damit man die beiden Pins später nicht verwechselt, benennen wir Pin4 und Pin5 einfach um.

Passiv Aktiv

Praxistipp: Die Bauform des aktiven und passiven Lautsprechers können nahezu identisch sein. Man erkennt den hier benötigten aktiven Lautsprecher (auch *„piezo Speaker"* oder *„active Speaker"*) anhand der geschlossenen, schwarzen Unterseite.

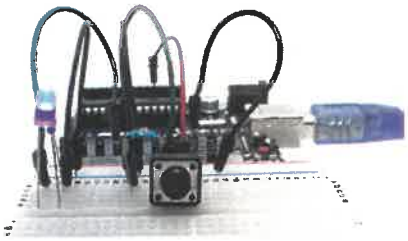
```
int LED=4;          //Das Wort „LED" steht jetzt für die Zahl „4".
int pieps=5;        //Das Wort „Pieps" steht jetzt für die Zahl „5".
void setup()
{                          //Wir starten mit dem Setup.
pinMode(LED, OUTPUT);      //Pin 4 (Pin „LED") ist ein Ausgang.
pinMode(pieps, OUTPUT);    //Pin 5 (Pin „Pieps") ist ein Ausgang.
}
void loop()
{                         //Das Hauptprogramm beginnt.
digitalWrite(LED, HIGH);  //Schalte die LED an.
```

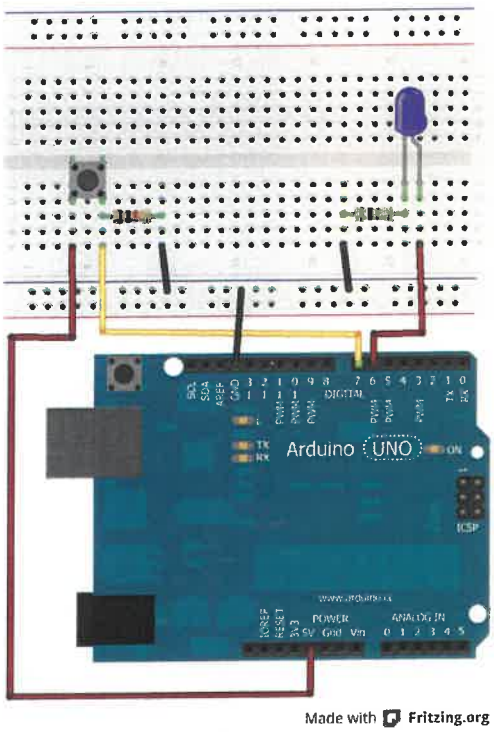
```
digitalWrite(pieps, HIGH);//Schalte den Piezo-Lautsprecher an.
delay(1000);    //Warte 1000 Millisekunden. (Es piepst und leuchtet)
digitalWrite(LED, LOW);   //Schalte die LED aus.
digitalWrite(pieps, LOW);  //Schalte den Piezo aus.
delay(1000); //Warte 1000 Millisekunden. (kein Lärm, kein Licht).
}
//Hier am Ende springt das Programm an den Start des Loop-Teils. Es wird somit gleich
wieder piepsen und leuchten. Wenn man die Pause (delay) verkleinert oder vergrößert,piepst
und leuchtet es schneller oder langsamer.
```

4.5 Eine LED per Taste aktivieren

Aufgabe: Eine LED soll für 5 Sekunden leuchten, wenn ein Taster betätigt wurde.

Materialbox
1x Arduino-Board
1x blaue LED
1x Taster
1x Widerstand mit 100 Ohm
1x Widerstand mit 1K Ohm (1000 Ohm)
1x Breadboard
einige Steckkabel





Der Mikrocontroller kann an seinen digitalen Pins nicht nur Spannungen ausgeben, sondern auch einlesen. Dies wollen wir in diesem Sketch ausprobieren.Bei dem Aufbau gibt es jedoch eine Besonderheit. Wenn man den Taster einfach nur mit dem Mikrocontroller verbindet, liegt an dem Pin des Mikrocontrollers eine Spannung an, sobald der Taster gedrückt wird. Lässt man den Taster aber los, hängt der Pin sozusagen "in der Luft" und ist mit gar nichts verbunden. Man kann sich das so vorstellen, als würden an dem besagten Pin elektrische Ladungen herumschwirren. Drückt man den Taster, schiebt die Spannung die Ladungsträger an und diese bewegen sich zum Arduino hin und er erkennt ein Signal. Wenn der Taster dann losgelassen wird, schiebt aber keine Spannung die Ladungsträger mehr und diese lassen sich leicht durch kleine Störungen aus der Umgebung beeinflussen. Das führt dazu, dass - obwohl der Taster geöffnet ist - der Arduino bewegte Ladungsträger wahrnimmt und ein Signal erkennt, wo keines ist. Der Mikrocontroller "denkt" dann also, dass der Taster gedrückt wird, weil sich die Ladungen bewegen. Dieses Problem lässt sich dadurch beheben, dass man den Pin über einen Widerstand (ca. 1000 Ohm bzw. 1 KOhm) mit GND verbindet. Dadurch hat der Pin immer eine definierte Spannung. Bei geöffneten Taster eine negative und bei geschlossenem Taster eine positive. Die Ladungsträger lassen sich dann nicht mehr von

außen stören, sondern werden immer korrekt, abhängig vom Taster, in die eine oder andere Richtung geschoben. Da der Widerstand die Spannung an dem Eingangspin immer auf 0V "herunter zieht", wird er auch als "PULLDOWN-" Widerstand bezeichnet. ACHTUNG: Wenn man dafür einen zu kleinen Widerstand verwendet, kann beim Drücken des Tasters ein Kurzschluss auf dem Mikrocontrollerboard entstehen.

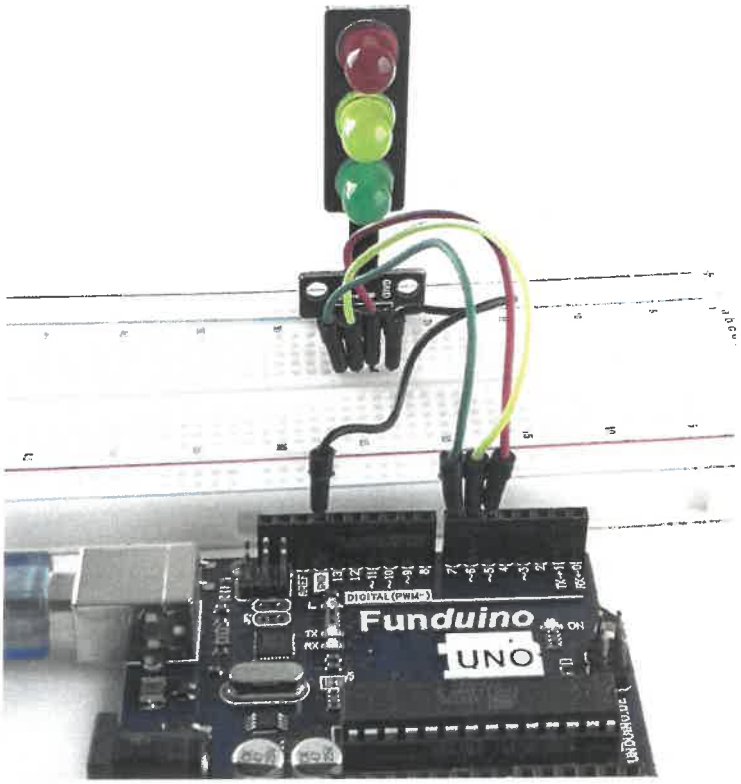
Sketch:

```
int LEDblau=6;    //Das Wort „LEDblau“ steht jetzt für den Wert 6.
int taster=7;     //Das Wort „taster“ steht jetzt für den Wert 7.
int tasterstatus=0;  //Das Wort „tasterstatus“ steht jetzt zunächst für den Wert 0. Später
wird unter dieser Variable gespeichert, ob der Taster gedrückt ist oder nicht.
void setup()
{                        //Hier beginnt das Setup.
pinMode(LEDblau, OUTPUT);//Der Pin mit der LED (Pin 6) ist jetzt ein Ausgang.
pinMode(taster, INPUT);  //Der Pin mit dem Taster (Pin 7) ist jetzt ein Eingang.
}
void loop()
{ //Mit dieser Klammer wird der Loop-Teil geöffnet
tasterstatus=digitalRead(taster); //Hier wird der Pin7 ausgelesen (Befehl:digitalRead). Das
Ergebnis wird unter der Variable „tasterstatus“ mit dem Wert „HIGH“ für 5Volt oder „LOW“
für 0Volt gespeichert.
if (tasterstatus == HIGH) //Verarbeitung: Wenn der taster gedrückt ist (Das Spannungssignal
ist hoch)..
{                         //Programmabschnitt des IF-Befehls öffnen.
digitalWrite(LEDblau, HIGH);//..dann soll die LED leuchten..
delay(5000);              //..und zwar für für 5 Sekunden (5000
                          Millisekunden).
digitalWrite(LEDblau, LOW); //danach soll die LED aus sein.
}                         //Programmabschnitt des IF-Befehls schließen.
else                      // ...ansonsten...
{                         //Programmabschnitt des else-Befehls öffnen
digitalWrite(LEDblau, LOW); // ...soll die LED aus sein.
}    //Programmabschnitt des else-Befehls schließen
}    //Mit dieser letzten Klammer wird der Loop-Teil  geschlossen.
```

4.6 Eine Ampel programmieren

Aufgabe: Ein Ampelmodul soll wie eine echte Ampel im Straßenverkehr programmiert werden.

Materialbox
1x Arduino-Board
1x Ampelmodul
(alternativ drei LEDs)
1x Breadboard
einige Steckkabel



Eine Ampel zu programmieren ist „der Klassiker" unter den Mikrocontrolling Projekten. Das liegt daran, dass es relativ einfach ist, aber auch einen starken Alltagsbezug hat. Denn jede Ampelanlage wird in irgendeiner Weise von einem Mikrocontroller angesteuert. Natürlich sind moderne Ampelanlagen in der Regel vernetzt und mit einer Reihe von Sensoren automatisiert, aber die grundlegende Funktion lässt sich mit einem Arduino-Mikrocontroller sehr einfach nachbauen.
Bei der Verwendung des Ampel-Moduls werden keine Vorwiederstände benötigt. Anstelle eines Ampelmoduls können auch drei einzelne LEDs verwendet werden. Damit lässt sich diese Anleitung ebenfalls durchführen, jedoch muss dann eine Schaltung wie in den vorherigen Anleitungen aufgebaut werden.

Ampel ohne Taster für Fußgänger

Die Programmierung einer Ampel mit der Arduino Software ist sehr leicht und kann direkt umgesetzt werden, wenn man bereits in der Lage ist, eine LED ein- und auszuschalten. Die Unterschiede bei der Ampel liegen lediglich darin, dass es drei anstatt einer LED gibt, und dass die Dauer der Ampelphasen programmiert werden muss.

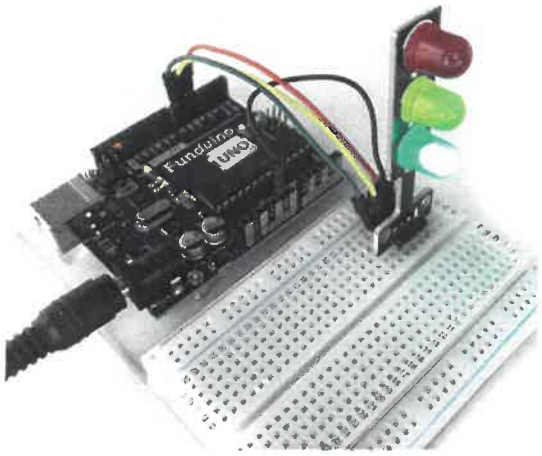
Verkabelung

Es ist empfehlenswert, das Ampelmodul aufrecht in einem Breadboard zu platzieren. Von vorne erkennt man an den Pins des Ampelmoduls, um welche Anschlüsse es sich handelt. GND= gemeinsames „-", also GND am Mikrocontroller. R= Anschluss der roten LED , Y= Anschluss der gelben LED und G= Anschluss der grünen LED. Die Verkabelung mit dem Mikrocontroller lässt sich aus dem ersten Foto zu dieser Anleitung entnehmen.

Sketch Nr.1: Eine typische Ampelphase

```
int ROT=5; //rote LED an Pin5
int GELB=6; //gelbe LED an Pin6
int GRUN=7; //grüne LED an Pin7
void setup() //Wir starten mit dem Setup
{
pinMode(ROT, OUTPUT); // Pin5 ist ein Ausgang.
pinMode(GELB, OUTPUT); // Pin6 ist ein Ausgang.
pinMode(GRUN,OUTPUT); // Pin7 ist ein Ausgang.
}
void loop() // Das Hauptprogramm beginnt.
{
digitalWrite(ROT, HIGH); // Schalte die LED an Pin5 an.
delay(2000); // Warte 1000 Millisekunden.
digitalWrite(GELB, HIGH); // Schalte die LED an Pin6 an.
delay(1000); // Warte 1000 Millisekunden.
digitalWrite(ROT, LOW); // Schalte die LED an Pin5 aus.
digitalWrite(GELB, LOW); // Schalte die LED an Pin6 aus.
digitalWrite(GRUN, HIGH); // Schalte die LED an Pin7 an.
delay(2000); // Warte 1000 Millisekunden.
digitalWrite(GRUN, LOW); // Schalte die LED an Pin7 aus.
digitalWrite(GELB, HIGH); // Schalte die LED an Pin6 an.
delay(1000); // Warte 1000 Millisekunden.
digitalWrite(GELB, LOW); // Schalte die LED an Pin6 aus.
} // Hier am Ende springt das Programm an den Start des Loop-Teils.
```

Nach dem Ausschalten der gelben LED beginnt der Loop-Teil von vorne und der Arduino Mikrocontroller schaltet die rote LED wieder ein. Die Ampel beginnt also von vorne mit der Rot-Phase. An dieser Ampelphase kann nun natürlich noch viel verbessert werden. Zum Beispiel kann man echte Ampelphasen im Straßenverkehr beobachten und im Sketch einsetzen. Für die Analyse ist übrigens die Wartezeit vor einer roten Ampel perfekt geeignet.



Fußgängerampel mit Tasterabfrage

Als nächstes soll eine Fußgängerampel programmiert werden. Fußgängerampeln geben den Autofahrern ein dauerhaftes Grünsignal und wechseln lediglich dann auf ein gelbes und rotes Licht, wenn ein Fußgänger oder Radfahrer den Taster an der Ampelanlage betätigt hat. Während der Grünphase wird vom Arduino Mikrocontroller permanent der Taster abgefragt und bei einem Tastendruck schaltet die Ampel das Signal nach einer gewissen Zeit für die Autos auf gelb und anschließend rot um. Nach einer weiteren kurzen Wartezeit wird dann die Ampel für die Fußgänger von rot auf grün geschaltet. Die Ampel für die Fußgänger ist in dem folgenden Sketch noch nicht enthalten. Es geht zunächst nur darum, die Ampel für die Fahrzeuge nach einem Tastendruck auf „rot“ umzustellen.

Sketch Nr.2 Fußgängerampel mit Taster

```
int Taster=4; // Wie in der vorherigen Anleitung wird ein Taster an Pin4 angeschlossen.
Auch hier muss ein Pulldown-Widerstand verwendet werden.
int tasterstatus=0; //Das Wort „tasterstatus“ steht jetzt zunächst für den Wert 0. Später
wird unter dieser Variable gespeichert, ob der Taster gedrückt ist oder nicht.
int ROT=5;
int GELB=6;
int GRUN=7;
void setup() //Wir starten mit dem Setup
{
pinMode(Taster, INPUT); // Pin4 ist ein Eingang, da hier der Taster abgefragt wird.
pinMode(ROT, OUTPUT); // Pin5 ist ein Ausgang.
pinMode(GELB, OUTPUT); // Pin6 ist ein Ausgang.
pinMode(GRUN,OUTPUT); // Pin7 ist ein Ausgang.
}
void loop() // Das Hauptprogramm beginnt.
{
digitalWrite(GRUN, HIGH); // Schalte die grüne LED an Pin7 an.
tasterstatus=digitalRead(Taster);  //Hier wird der Pin4 ausgelesen (Befehl:digitalRead).
Das Ergebnis wird unter der Variable „tasterstatus“ mit dem Wert „HIGH“ für 5 Volt oder
„LOW“ für 0 Volt gespeichert.
if (tasterstatus == HIGH) //Verarbeitung: Wenn der Taster gedrückt ist (das Spannungssignal
ist hoch), verarbeitet der Sketch den folgenden Programmblock:
{ //Programmabschnitt der IF-Bedingung öffnen.
 delay(2000); // Warte 2000 Millisekunden(2 Sekunden), da es bei einer echten Ampel aus
Sicherheitsgründen auch immer eine Weile dauert, bis die Ampel umspringt.
 digitalWrite(GRUN, LOW); // Schalte die grüne LED an Pin7 aus.
 digitalWrite(GELB, HIGH); // Schalte die gelbe LED an Pin6 an.
 delay(1000); // Warte 1000 Millisekunden.
 digitalWrite(GELB, LOW); // Schalte die gelbe LED an Pin6 aus.
 digitalWrite(ROT, HIGH); // Schalte die rote LED an Pin5 an.
 delay(5000); // Warte 5000 Millisekunden, die Ampel ist für die Autofahrer rot.
 digitalWrite(GELB, HIGH); // Schalte zusätzlich die gelbe LED an Pin6 an.
 delay(1000); // Warte 1000 Millisekunden.
 digitalWrite(ROT, LOW); // Schalte die rote LED an Pin5 aus.
 digitalWrite(GELB, LOW); // Schalte zusätzlich die gelbe LED an Pin6 aus.
} //Programmabschnitt des IF-Befehls schließen.
} // Hier am Ende springt das Programm an den Start des Loop-Teils. Da dort die grüne LED
direkt aktiviert wird, muss dies nicht zusätzlich am Ende der Schleife gemacht werden.
```

Was in diesem Sketch noch fehlt, ist ein Signal für die Fußgänger oder Radfahrer. Üblicherweise gibt es dafür eine weitere Ampel mit lediglich einem roten und einem grünen Signal. Diese zusätzliche Ampel bauen wir also auch noch auf. Wir schließen für die Fußgänger eine rote LED an Pin8 und eine grüne LED an Pin9 an.

Sketch Nr.3 Fußgängerampel mit Taster und erweiterter Ampelanlage

```
int Taster=4;
int tasterstatus=0;
int ROT=5;
int GELB=6;
int GRUN=7;
int ROTFuss=8; // rotes Ampelsignal für Fußgänger
int GRUNFuss=9; // grünes Ampelsignal für Fußgänger
void setup() //Wir starten mit dem Setup
{
```

```
pinMode(Taster, INPUT); // Pin4 ist ein Eingang.
pinMode(ROT, OUTPUT); // Pin5 ist ein Ausgang.
pinMode(GELB, OUTPUT); // Pin6 ist ein Ausgang.
pinMode(GRUN,OUTPUT); // Pin7 ist ein Ausgang.
pinMode(RotFuss, OUTPUT); // Pin8 ist ein Ausgang.
pinMode(GruenFuss,OUTPUT); // Pin9 ist ein Ausgang.
}
void loop() // Das Hauptprogramm beginnt.
{
digitalWrite(GRUN, HIGH); // Schalte die grüne LED an Pin7 an. Die Autos sehen also das
grüne Signal.
digitalWrite(RotFuss, HIGH); // Schalte die rote Fußgängerampel an Pin8 an.
tasterstatus=digitalRead(Taster);  //Hier wird der Pin4 ausgelesen (Befehl:digitalRead).
Das Ergebnis wird unter der Variable „tasterstatus“ mit dem Wert „HIGH“ für 5 Volt oder
„LOW“ für 0 Volt gespeichert.
if (tasterstatus == HIGH) //Verarbeitung: Wenn der Taster gedrückt ist (das Spannungssignal
ist hoch) verarbeitet der Sketch den folgenden Programmblock
{ //Programmabschnitt der IF-Bedingung öffnen.
  delay(2000); // Warte 2000 Millisekunden(2 Sekunden), da es bei einer echten Ampel aus
  Sicherheitsgründen auch immer eine Weile dauert, bis die Ampel umspringt.
  digitalWrite(GRUN, LOW); // Schalte die grüne LED an Pin7 aus.
  digitalWrite(GELB, HIGH); // Schalte die gelbe LED an Pin6 an.
  delay(1000); // Warte 1000 Millisekunden.
  digitalWrite(GELB, LOW); // Schalte die gelbe LED an Pin6 aus.
  digitalWrite(ROT, HIGH); // Schalte die rote LED an Pin5 an.
  delay(2000); // Warte 2000 Millisekunden(2 Sekunden), da es bei einer echten Ampel aus
  Sicherheitsgründen auch immer eine Weile dauert, bis die Ampel umspringt. Die Autofahrer
  haben jetzt "rot".
  digitalWrite(RotFuss, LOW); // Schalte die rote Fußgängerampel an Pin8 aus.
  digitalWrite(GruenFuss, HIGH); // Schalte die grüne Fußgängerampel an Pin9 an. Die
  Radfahrer und Fußgänger gehen los.
  delay(5000); // Warte weitere 5000 Millisekunden.
  digitalWrite(RotFuss, HIGH); // Schalte die rote Fußgängerampel an Pin8 ein.
  digitalWrite(GruenFuss, LOW); // Schalte die grüne Fußgängerampel an Pin9 aus. Die
  Radfahrer und Fußgänger müssen stehen bleiben.
  delay(2000); // Warte 2000 Millisekunden(2 Sekunden). Die Autofahrer haben immer noch rot
  und die Fußgänger müssen die Straße räumen.
  digitalWrite(GELB, HIGH); // Schalte zusätzlich für die Autofahrer die gelbe LED an Pin6
  an.
  delay(1000); // Warte 1000 Millisekunden.
  digitalWrite(ROT, LOW); // Schalte die rote LED an Pin5 aus.
  digitalWrite(GELB, LOW); // Schalte zusätzlich die gelbe LED an Pin6 aus.
} //Programmabschnitt des IF-Befehls schließen.
} // Hier am Ende springt das Programm an den Start des Loop-Teils. Da dort die grüne LED
direkt aktiviert wird, muss dies nicht zusätzlich am Ende der Schleife gemacht werden.
```

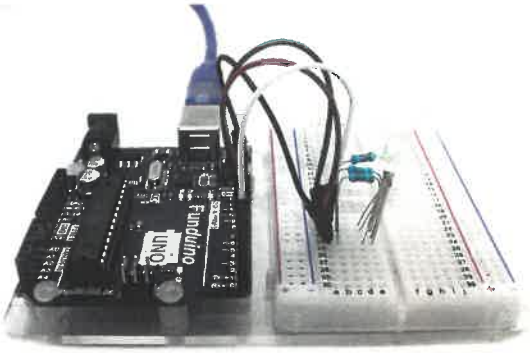
Achtung, nur für Fortgeschrittene

Schwieriger wird es, wenn ein Taster benötigt wird, um eine Ampel zu unterbrechen, die dauerhaft die Rot- und Grünphasen ändert. An Kreuzungen im Straßenverkehr ist das der Fall. Das größte Problem dabei ist die Verwendung des "delay" Befehls im letzten Sketch. Während einer Delay-Phase reagiert der Sketch auf keinerlei Eingaben, die bspw. an einem angeschlossenen Taster vorgenommen werden. Die Lösung für dieses Problem ist vergleichsweise aufwändig und daher an dieser Stelle noch nicht geeignet. Zu einem späteren Zeitpunkt kann das Problem mit der „Interrupt-“ Funktion gelöst werden.

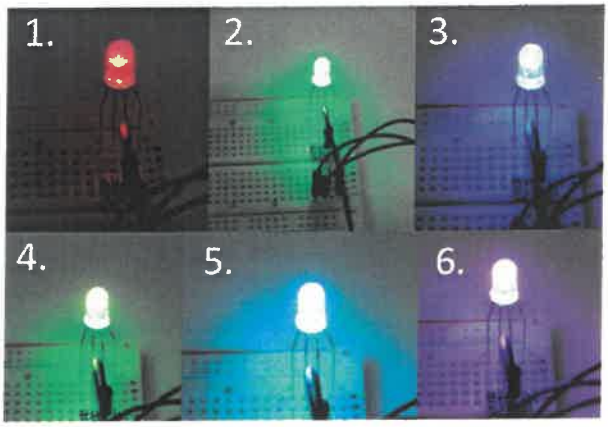
4.7 Eine farbige LED ansteuern (RGB)

Aufgabe: Eine RGB LED soll in verschiedenen Farben leuchten

Materialbox
1x Arduino-Board
1x RGB-LED
3x Widerstand 200 Ohm
1x Breadboard
Einige Steckkabel



Eine RGB-LED ist eine LED, die in verschiedenen Farben leuchten kann. Hinter der Bezeichnung RGB verbergen sich die Farbei „Rot", „Grün" und „Blau". Die LED besteht im Inneren aus drei einzeln ansteuerbaren LEDS, die in den drei Farben leuchten. Deswegen hat eine RGB-LED auch so viele Beinchen, nämlich genau vier. Das längste der vier Beinchen ist je nach Version die gemeinsame Anode (+) bzw. Kathode (-). Mit den drei kürzeren Beinchen werden die einzelnen Farben der RGB-LED angesteuert. Durch eine Mischung der Farben können noch sehr viele weitere Farben erzeugt werden. Zum Beispiel entsteht durch die Ansteuerung der Farben „Blau" und „Grün" die Farbe „Türkis".



Es gibt zwei Versionen von RGB-LEDs:

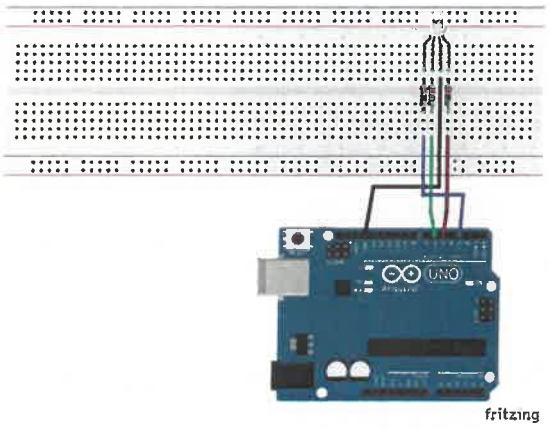
„Common anode" – Gemeinsame Anode. Das längste Beinchen der LED ist „+" und die drei kürzeren Beinchen werden über „-" (GND) angesteuert.

„Common cathode" – Gemeinsame Kathode. Das längste Beinchen der LED ist „-" und die drei kürzeren Beinchen werden über „+" (Spannung) angesteuert.

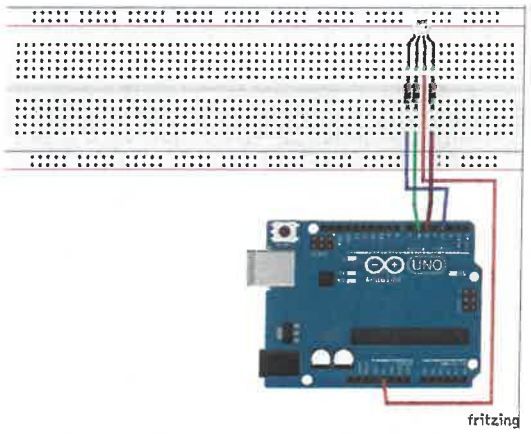
Welche Version man selber hat, findet man notfalls auch durch einfaches umstecken von „+“ und „-“ an der LED heraus. Die LED kann durch eine Verpolung nicht kaputt gehen, da der Strom nur fließen kann, wenn die LED richtig angeschlossen wurde.

Praxistipp: Beim Testen der LED niemals einen Kontakt der LED direkt an 5V anschließen! Man muss immer einen Widerstand verwenden (100-200 OHM), da die LED ansonsten sofort kaputt geht.

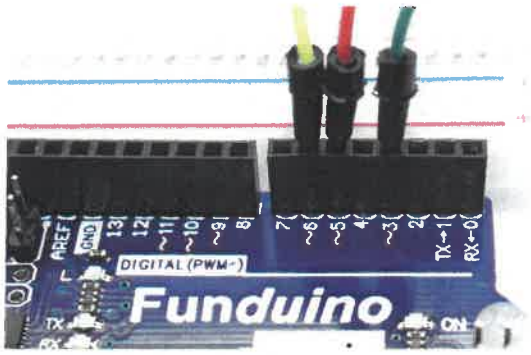
Aufbau für RGB LED - Gemeinsame Kathode:



Aufbau für RGB LED - Gemeinsame Anode:



Um mit einer RGB-LED viele verschiedenen Farben zu erzeugen, muss ähnlich wie in der Anleitung zur pulsierenden LED die PWM Funktion (Pulsweitenmodulation) der digitalen Pins verwendet werden. Die PWM kann an den digitalen Pins verwendet werden, an denen auf dem Board eine kleine Welle aufgedruckt ist.



Die PWM lässt die Spannung zwischen +5V und 0V pulsieren. Die Spannung wird dabei im Millisekundenbereich ein und ausgeschaltet. Bei einer hohen PWM liegt das 5V Signal nahezu durchgehend am jeweiligen Pin an. Bei einer geringen PWM ist das 5V Signal kaum noch vorhanden (Da dies eine sehr kompakte Zusammenfassung ist, sollte man sich im Internet nach weiteren Erläuterungen umsehen). Mit dieser PWM kann man bei LEDs einen ähnlichen Effekt erreichen, als würde man die Spannung variieren.

Die folgenden Sketche funktionieren für beide RGB-LED-Versionen gleichermaßen. Es muss nur eine Sache beachtet werden: Bei der LED Version b (Common anode) muss der Wert für „dunkel“ auf 255 gesetzt werden. Das hat zur Folge, dass dann nicht nur am gemeinsamen Pluspol der LED eine positive Spannung anliegt, sondern auch an der entsprechenden Farbe. Dann kann zwischen den beiden Kontakten der LED kein Strom mehr fließen und die jeweilige Farbe der LED bleibt aus. Aus diesem Grund ist auch für die Farbmischung zu

beachten, dass bei dieser Version der Leuchtdiode die Farbe heller wird, wenn der Wert kleiner wird. So leuchtet die Farbe blau an Pin 3 in diesem Sketch hell, wenn der Code für die blaue Farbe so gewählt wird:

int brightness1a = 0;

Sketch 1:

In diesem Code werden die drei einzelnen Farben nacheinander ein- und ausgeschaltet.

```
int LEDblau = 3;       //Farbe Blau an Pin 3
int LEDrot = 5;        //Farbe Rot an Pin 5
int LEDgruen=6;        //Farbe Gruen an Pin 6
int p=1000;            //p ist eine Pause mit 1000ms also 1 Sekunde
int brightness1a = 150; //Zahlenwert zwischen 0 und 255 - gibt die
                         Leuchtstärke der einzelnen Farbe an
int brightness1b = 150; //Zahlenwert zwischen 0 und 255 - gibt die
                         Leuchtstärke der einzelnen Farbe an
int brightness1c = 150; //Zahlenwert zwischen 0 und 255 - gibt die
                         Leuchtstärke der einzelnen Farbe an
int dunkel = 0; //Zahlenwer 0 bedeutet Spannung 0V - also LED aus
void setup()
{
pinMode(LEDblau, OUTPUT);
pinMode(LEDgruen, OUTPUT);
pinMode(LEDrot, OUTPUT);
}
void loop()
{
analogWrite(LEDblau, brightness1a);     //Blau einschalten
delay(p);                               //pause
analogWrite(LEDblau, dunkel);           //Blau ausschalten
analogWrite(LEDrot, brightness1b);      //Rot einschalten
delay(p);                               //pause
analogWrite(LEDrot, dunkel);            //Rot ausschalten
analogWrite(LEDgruen, brightness1c);    //Grün einschalten
delay(p);                               //pause
analogWrite(LEDgruen, dunkel);          //Grün ausschalten
}
```

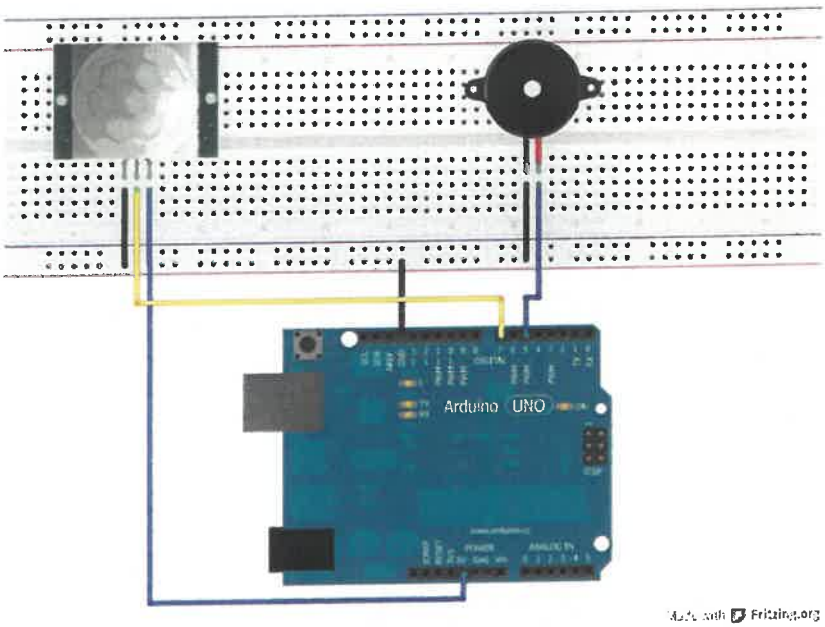
Sketch 2:

In diesem Code werden die drei einzelnen Farben jeweils paarweise nacheinander ein- und ausgeschaltet. Dadurch entstehen die Farbmischungen Gelb, Türkis und Lila.

```
int LEDblau = 3;           //Farbe blau an Pin 3
int LEDrot = 5;      //Farbe rot an Pin 5
int LEDgruen=6;      //Farbe gruen an Pin 6
int p=1000;          //p ist eine Pause mit 1000ms also 1 Sekunde
int brightness1a = 150;    //Zahlenwert zwischen 0 und 255 - gibt die
                           Leuchtstärke der einzelnen Farbe an
int brightness1b = 150;    //Zahlenwert zwischen 0 und 255 - gibt die
                           Leuchtstärke der einzelnen Farbe an
int brightness1c = 150;    //Zahlenwert zwischen 0 und 255 - gibt die
                           Leuchtstärke der einzelnen Farbe an
int dunkel = 0;      //Zahlenwert 0 bedeutet Spannung 0V - also LED aus
void setup()
{
pinMode(LEDblau, OUTPUT);
pinMode(LEDgruen, OUTPUT);
pinMode(LEDrot, OUTPUT);
}
void loop()
{
analogWrite(LEDgruen, brightness1c); //Grün und Rot ein = Gelb
analogWrite(LEDrot, brightness1b);
delay(p);
analogWrite(LEDgruen, dunkel);        //Grün und Rot aus = Gelb aus
analogWrite(LEDrot, dunkel);
analogWrite(LEDgruen, brightness1c); //Grün und Blau ein = Türkis
analogWrite(LEDblau, brightness1b);
delay(p);
analogWrite(LEDgruen, dunkel);        //Grün und Blau aus = Türkis aus
analogWrite(LEDblau, dunkel);
analogWrite(LEDrot, brightness1b);    //Rot und Blau ein = Lila
analogWrite(LEDblau, brightness1b);
delay(p);
analogWrite(LEDrot, dunkel);          //Rot und Blau aus = Lila aus
analogWrite(LEDblau, dunkel);
}
```

4.8 Der Bewegungsmelder

Materialbox
1x Arduino-Board
1x Bewegungsmelder
1x Piezo-Speaker
1x Breadboard
Einige Steckkabel



Aufgabe: Ein Piezo-Lautsprecher soll piepen, sobald eine Bewegung registriert wird.

*Praxistipp: Der Bewegungsmelder ist **sehr empfindlich** und kann bereits bei der Reflexion von Wärmestrahlung einer Person an Wand oder Decke reagieren. Daher ist es ratsam, den Bewegungsmelder bei den ersten Versuchen verdeckt in einer Schachtel zu Positionieren.*

Der Bewegungsmelder, auch PIR Sensor genannt, ist sehr einfach konstruiert. Sobald er eine Bewegung detektiert, gibt er auf einem Pin eine Spannung von 5 Volt aus. Diese muss nur ausgelesen und vom Mikrocontroller verarbeitet werden.

Bild1

Bild2

Bild3

Die Dauer des Ausgangssignals (linker Regler) und die Sensibilität (rechter Regler) kann über Drehregler eingestellt werden (Bild 1). Vor den ersten Versuchen sollten beide Drehregler in die Minimalposition gebracht werden.

Die Kunststofflinse ist nur leicht aufgesteckt und kann abgehoben werden. Darunter wird der Infrarotdetektor sichtbar, sowie die Beschriftung der Kontakte: GND (-) / OUT (Ausgang des Signals) / VCC (+). Auf Bild 2 sind die entsprechenden Bezeichnungen der Kontakte am oberen Bildrand erkennbar.

4.8 Der Bewegungsmelder

Es gibt Bewegungsmelder mit und ohne Jumper. Falls ein Jumper vorhanden ist, hat dieser folgende Funktionen.

1) Jumper ist wie auf dem Bild ganz außen: Das Ausgangssignal wird nachdem eine Bewegung detektiert wurde für eine gewisse Zeit aufrecht erhalten und danach auf jeden Fall wieder deaktiviert, auch wenn im Aktionsbereich des Bewegungsmelders noch eine Bewegung detektiert werden könnte. Nach einer gewissen Zeit wird das Ausgangssignal erneut erzeugt.

2) Der Jumper ist leicht nach innen versetzt. Das Ausgangssignal bleibt pausenlos aktiv, so lange vom Bewegungsmelder eine Bewegung detektiert wird.

Aufbau:

Der Aufbau und die Verkabelung zu dieser Anleitung ist in der ersten Skizze dargestellt.

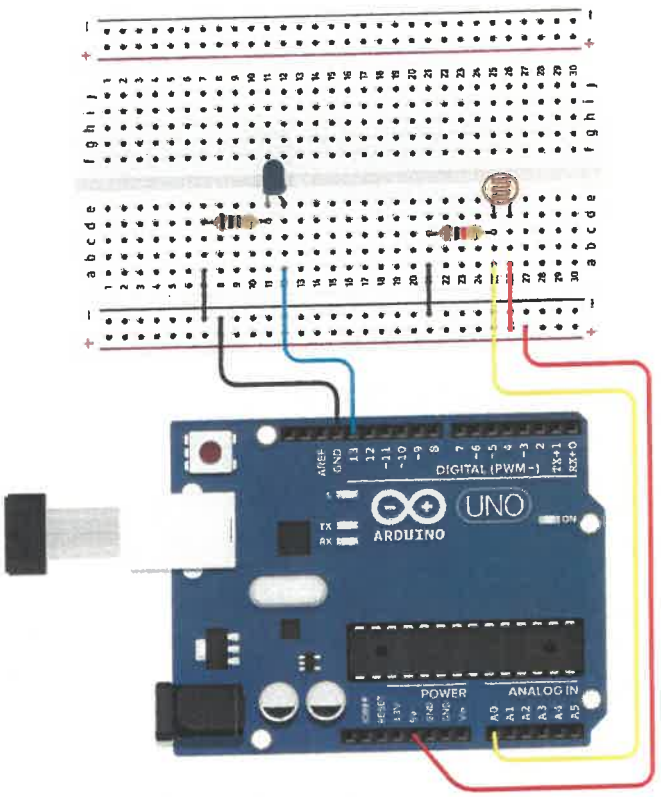
Sketch:

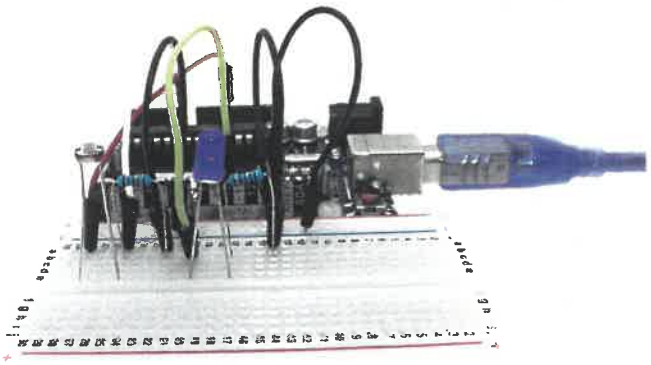
```
int piezo=5;     //Das Wort „piezo“ steht jetzt für den Wert 5.
int bewegung=7; //Das Wort „bewegung“ steht jetzt für den Wert 7.
int bewegungsstatus=0;     //Das Wort „bewegungsstatus“ steht jetzt zunächst für den Wert
0. Später wird unter dieser Variable gespeichert, ob eine Bewegung erkannt wird oder nicht.
void setup()           //Hier beginnt das Setup.
{
pinMode(piezo, OUTPUT);//Der Pin mit dem Piezo (Pin 5) ist jetzt ein Ausgang.

pinMode(bewegung, INPUT);//Der Pin mit dem Bewegungsmelder (Pin7) ist jetzt ein Eingang.
}
void loop()  //Der Loop-Teil beginnt
{            //Mit dieser Klammer wird der Loop-Teil geöffnet.
bewegungsstatus=digitalRead(bewegung); //Hier wird der Pin7 ausgelesen. Das Ergebnis wird
unter der Variablen „bewegungsstatus“ mit dem Wert „HIGH“ für 5Volt oder „LOW“ für 0 Volt
gespeichert.
if (bewegungsstatus == HIGH)     //Verarbeitung: Wenn eine Bewegung detektiert wird (Das
Spannungssignal ist hoch)
{                  //Programmabschnitt des IF-Befehls öffnen.
digitalWrite(piezo, HIGH); //dann soll der Piezo piepsen.
delay(5000);               //...und zwar für für 5 Sekunden.
digitalWrite(piezo, LOW);  //...danach soll er leise sein.
}                  //Programmabschnitt des IF-Befehls schließen.
else               //ansonsten...
{                   //Programmabschnitt des else-Befehls öffnen.
digitalWrite(piezo, LOW);  //...soll der Piezo-Lautsprecher aus sein.
}//Programmabschnitt des else-Befehls schließen.
}//Mit dieser letzten Klammer wird der Loop-Teil geschlossen.
```

4.9 Lichtstärke messen

Aufgabe: Eine LED soll leuchten, wenn es dunkel wird bzw. wenn ein Fotowiderstand abgedeckt wird.

Materialbox
1x Arduino-Board
1x blaue LED
1x Widerstand 100 Ohm
1x Widerstand 10K Ohm
1x Breadboard
1x Steckkabel
1x Fotowiderstand





Spannungen auslesen:

Der Mikrocontroller soll über einen Fotowiderstand auslesen, wie hell es ist. Dazu nutzt man ein einfaches physikalisches Prinzip. Wenn in einem Stromkreis zwei Verbraucher hintereinander angeschlossen sind (Reihenschaltung), dann „teilt" sich auch die gemeinsam anliegende Spannung. Ein Beispiel: Zwei gleiche Lampen sind in Reihe geschaltet und es wird eine Spannung von 6 Volt angelegt. Dann kann man mit einem Spannungsmessgerät feststellen, dass an den Lampen jeweils nur 3 Volt anliegen. Wenn zwei ungleiche Lampen angeschlossen werden (eine hat einen geringeren Widerstand), dann kann man zwei unterschiedliche Spannungen an den beiden Lampen messen, bspw. 1,5 Volt und 4,5 Volt. Ein Fotowiderstand ändert seinen Widerstand in Abhängigkeit der Lichtstärke. Diesen Effekt nutzt man, um anhand der an ihr anliegenden Spannung einen Wert für Helligkeit bzw. Dunkelheit in Form von verschiedenen Spannungen abzulesen. Damit man hier überhaupt eine Spannungsteilung erzeugen kann, schließt man den Fotowiderstand und einen Widerstand (1 – 10 K Ohm, je nach verwendetem Fotowiderstand. Der Widerstand sollte einen ähnlichen Widerstandswert wie der Fotowiderstand haben) in Reihe an und verbindet sie wie in der Skizze mit 5 Volt und der „Erdung" (Ground / GND). Das Mikrocontroller-Board ist in der Lage, analoge Signale (Spannung) zu messen und diese zu verarbeiten. Dies geschieht mit den analogen Eingängen auf dem Board. Dieser wandelt den gemessenen Spannungswert in eine Zahl um, die dann weiter verarbeitet werden kann. 0 Volt entspricht dabei dem Wert 0 und der höchste Messwert bei 5 Volt entspricht dem Wert 1023 (0 bis 1023

entspricht 1024 Zahlen = 10 Bit). Beispiel: Es wird eine Spannung von 2,5 Volt gemessen, dann liefert der Mikrocontroller den Wert 512 (1024 : 2).

Der serielle Monitor:

Der serielle Monitor oder auch engl. „serial monitor" ist ein wichtiger Bestandteil der Arduino-Software. Mit diesem Monitor kann man sich am PC Daten anzeigen lassen, die das Mikrocontroller-Board an den PC sendet (Zahlen oder Texte). Das ist sinnvoll, da man nicht immer ein LCD Display am Mikrocontroller angeschlossen hat, auf dem man bestimmte Werte ablesen könnte. In diesem Sketch wird der Monitor verwendet, um die Werte anzeigen zu lassen, die das Board von dem Fotowiderstand einliest. Wozu ist das sinnvoll? Mal angenommen, die LED soll erst bei beginnender Dunkelheit anfangen zu leuchten. Dann muss es im Sketch einen Bereich geben, der die Funktion hat: „Wenn der Wert des Fotowiderstandes den Wert x unterschreitet, dann soll die LED leuchten". Dazu müsste man wissen wie groß der Wert x bei beginnender Dämmerung ist.

Lösung: Ich sende den ausgelesenen Wert „x" der Spannung an dem Fotowiderstand bei entsprechender Helligkeit (bspw. Dämmerung) an den Monitor und lasse ihn mir dort anzeigen. Mit diesem Wissen kann ich später das Programm in der folgenden Form abändern. „Wenn der Spannungsausgabewert des Fotowiderstandes einen Wert von „x" unterschreitet, dann schalte die LED an."

Sketch

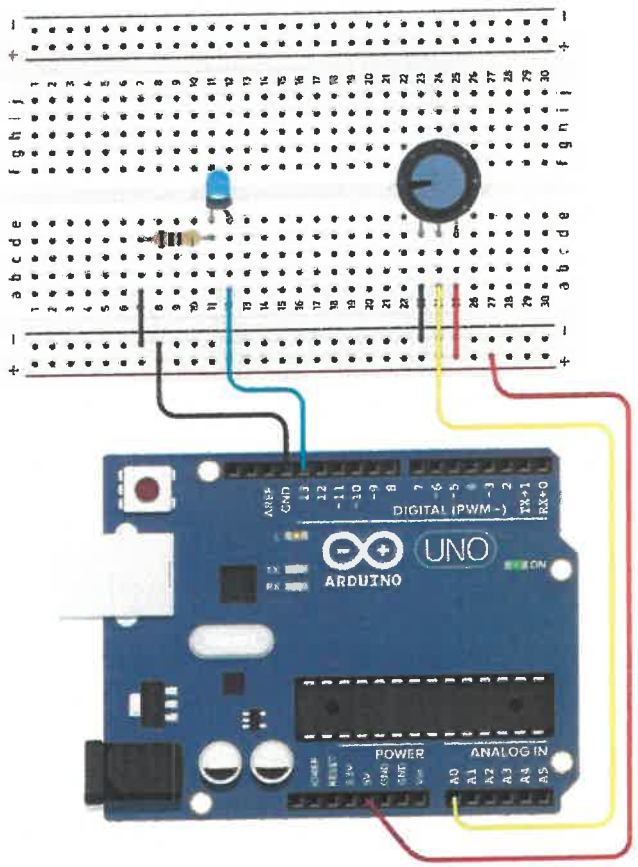
```
int eingang=A0; //Das Wort „eingang" steht jetzt für den Wert „A0" (Bezeichnung vom
Analogport 0)
int LED=13;          //Das Wort „LED" steht jetzt für den Wert 10
int sensorwert=0;    //Variable für den Sensorwert mit 0 als Startwert
void setup()         //Hier beginnt das Setup.
{
Serial.begin(9600); //Die Kommunikation mit dem seriellen Port wird gestartet, um den Wert
des Sensors im Seriellen Monitor darzustellen.
pinMode (LED,OUTPUT);//Der Pin mit der LED (Pin 10) ist jetzt ein
}                     Ausgang
void loop()
{ //Mit dieser Klammer wird der Loop-Teil geöffnet.
sensorwert=analogRead(eingang); //Die Spannung an dem Fotowiderstand auslesen und unter der
Variable „sensorWert" abspeichern.
Serial.println("Sensorwert=");    //Der Befehl „Seriel.print" erzeugt eine Ausgabe am
Seriellen Monitor.: Das Wort „Sensorwert:"
Serial.println(sensorwert); // Ausgabe am Seriellen Monitor. Mit dem Befehl
„Serial.println" wird der Sensorwert des Fotowiderstandes in Form einer Zahl zwischen 0 und
1023 an den Seriellen Monitor gesendet. Durch die Erweiterung um die beiden Buchstaben „LN"
wird aus dem Befehl „Seriel.println". Dadurch wird bei der Ausgabe am Seriellen Monitor ein
Seitenumbruch erzeugt und die Messwerte sind leichter ablesbar
if(sensorwert < 512)      //Wenn der Sensorwert unter dem Wert 512 liegt….
{
digitalWrite(LED,HIGH);    //…soll die LED leuchten…
}
else                       //…andernfalls
{
digitalWrite(LED,LOW);     //...soll sie nicht leuchten
}
```

```
delay(50);//Eine kurze Pause, in der die LED an oder aus ist
} //Mit dieser letzten Klammer wird der Loop-Teil geschlossen.
//Wenn nun der Sensorwert bei normaler Helligkeit bspw. nur den Wert 100 hat (der Wert ist
abhängig von den verwendeten Widerständen, von der Helligkeit und von der Stromrichtung),
dann nimmt man anstelle des Wertes 512 einen wesentlich kleineren Wert, bei dem die LED zu
leuchten beginnen soll. Bspw. nimmt man dann den Wert 90. Den aktuellen Sensorwert kann man
sich nun mit Hilfe des Seriellen Monitor anzeigen lassen. Dazu klickt man oben auf
„Werkzeuge“ und anschließend auf „Serieller Monitor“
```

4.10 Drehregler - Drehpotentiometer

Aufgabe: Eine LED soll blinken. Die Blinkgeschwindigkeit soll mit einem Drehregler eingestellt werden.

Materialbox
1x Arduino-Board
1x Drehregler (Potentiometer)
1x Breadboard
1x LED (optional)
1x Widerstand 100 Ohm (optional)
Einige Steckkabel



Ein Drehregler hat drei Anschlüsse. Außen wird + und – angeschlossen. Von dem mittleren Pin geht ein Kabel zu einem analogen Eingangspin am Mikrocontroller-Board. Wenn man den Drehregler dreht, dann gibt der mittlere Pin ein Spannung zwischen 0 und 5 Volt aus. Drehregler ganz links: 0V und Drehregler ganz rechts: 5V, bzw. seitenverkehrt, je nach Verkabelung. Der Drehregler agiert dabei als sogenannter „Spannungsteiler“.

Als LED, die blinken soll, verwenden wir wie im ersten Sketch die LED, die mit Pin13 am Mikrocontroller befestigt ist. Zusätzlich kann dort auch noch eine weitere LED angeschlossen werden, wie im Aufbau zu sehen ist.

Erweiterungsmöglichkeit:

Nach dem ersten Sketch ergeben sich viele Erweiterungsmöglichkeiten. So kann der Drehregler auch verwendet werden, um die Blinkgeschwindigkeit von zwei nacheinander leuchtenden LEDs zu variieren.

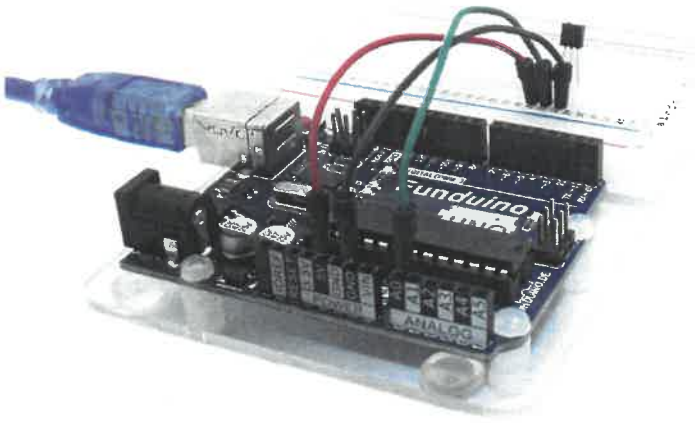
Sketch:

```
int eingang=A0;      //Das Wort „eingang“ steht jetzt für den Wert „A0“ (Bezeichnung vom
Analogport 0).
int LED=13;          //Das Wort „LED“ steht jetzt für den Wert 13.
int sensorwert=0;    //Variable für den Sensorwert mit 0 als Startwert.
void setup()
{                    //Hier beginnt das Setup.
pinMode (LED,OUTPUT);//Der Pin mit der LED (Pin 13) ist jetzt ein Ausgang.
}
void loop()
{            //Mit dieser Klammer wird der Loop-Teil geöffnet.
sensorwert= analogRead(eingang); //Die Spannung am Drehregler wird auslesen und wie im
vorherigen Sketch als Zahl zwischen 0 und 1023 unter der Variable „sensorwert“ gespeichert.
digitalWrite (LED,HIGH);   //Die LED wird eingeschaltet
delay(sensorwert);         //Die LED bleibt für so viele Millisekunden eingeschaltet, wie
der Wert von „sensorwert“ es gespeichert hat.
digitalWrite(LED, LOW);    //Die LED wird ausgeschaltet.
delay(sensorwert);         //Die LED bleibt für so viele Millisekunden ausgeschaltet, wie
der Wert von „sensorwert“ es gespeichert hat.
}            //Mit dieser Klammer wird der Loop-Teil geschlossen
//Der Loop-Teil wird nun erneut gestartet. Wenn sich der Wert des ausgelesenen Drehreglers
ändert, dann ändert sich auch die Zeit zwischen den Ein- und Aus-Phasen der LED. Das
Blinken wird dadurch schneller und langsamer. Das längste delay beträgt in diesem Sketch
1023ms (Millisekunden). Wenn man längere delays benötigt, dann baut man eine kleine
mathematische Zeile in den Code ein. Beispiel: Man ändert die Zeile „sensorwert
=analogRead(eingang);“ in „sensorwert =analogRead(eingang)*2;“ Damit wird der
abgespeicherte Sensorwert um den Faktor 2 vergrößert. Da längste delay wäre dann 2046ms
usw…
```

4.11 Temperatur messen

Aufgabe: Mit dem Temperatursensor TMP36 soll die Temperatur ausgelesen und mit dem Seriellen Monitor angezeigt werden.

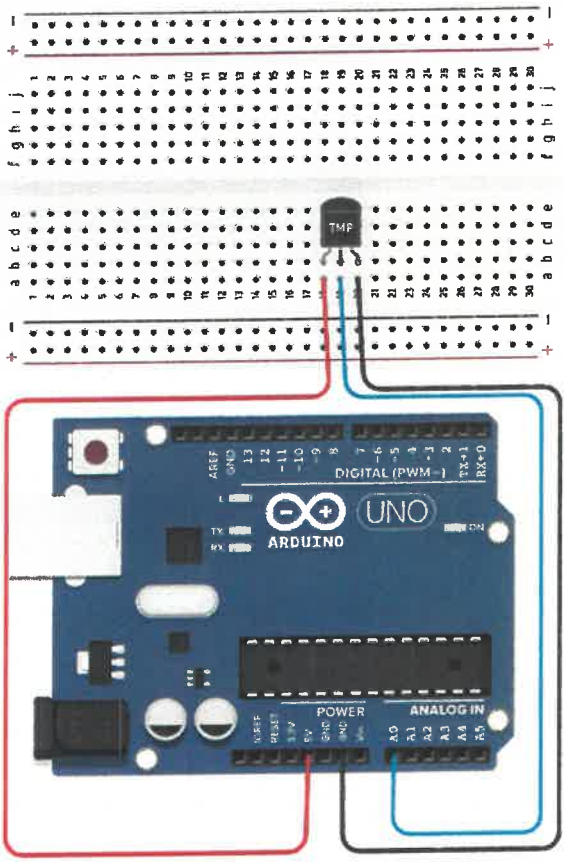
Materialbox
1x Arduino-Board
1x Temperatursensor TMP36
1x Breadboard
Einige Steckkabel



Der Sensor hat drei Anschlüsse. Beim Blick auf die flache Seite des Sensors: links 5V, rechts GND und in der Mitte den Pin für das Temperatursignal. Auf diesem Pin gibt der Sensor eine Spannung zwischen 0 und 2,0 Volt aus. Wobei 0V -50°C entsprechen und der Wert 2,0V entspricht 150°C. Laut Hersteller ist der Sensor zwischen -40°C und +125°C einigermaßen genau (±2°C). Die Spannung dieses Pins muss vom Mikrocontroller-Board ausgelesen und in einen Temperaturwert umgerechnet werden.

Praxistipp: Wenn der Sensor falsch angeschlossen wird, brennt er sofort durch. Bei schwankender Spannung am USB-Anschluss des Arduino Mikrocontrollerboards können die Messwerte verfälscht werden. In so einem Fall empfiehlt es sich, eine externe Spannungsversorgung zu verwenden.

Anstelle der Verwendung eines Breadboards kann der Sensor auch direkt mit Steckkabeln (Stecker/Fassung) verbunden werden. Dadurch lässt sich der Sensor flexibler zur Temperaturmessung einsetzen.



Für diesen Sketch wird der "map" Befehl benötigt. Dieser Befehl befindet sich in der Zeile: "temperatur= map(analogRead(TMP36), 0, 410, -50, 150);"

Anhand der allgemeinen Schreibweise „map (a, b, c, d, e)“ lässt sich die Funktion besser beschreiben. Ein Wert "a" (beispielsweise ein Messwert) wird in einem bestimmten Zahlenbereich zwischen den zwei Werten (b) und (c) erwartet. Der "map" Befehl wandelt dann den Wert "a" in einen anderen Wert um, der dem Zahlenbereich zwischen "d" und "e" entspricht. In unserem Sketch passiert dabei folgendes:

Der Temperatursensor TMP36 gibt an dem mittleren Pin den Messwert für die Temperatur in Form einer Spannung zwischen 0V und 2V aus. Dieser Spannungsbereich entspricht dem vom Sensor messbaren Temperaturbereich von -50°C bis +150°C. Bei 0°C gibt der Sensor am Ausgangspin eine Spannung von 0V aus und bei 150°C eine Spannung von 2V. Am analogen Eingangspin des Arduino Mikrocontrollerboards wird dieser Spannungsbereich mit Hilfe des Befehls "analogRead(TMP36)" als Zahlenwert zwischen 0 und 410 erkannt. Dieser Wert des Temperatursensors wird zunächst ausgelesen und unter der Variablen "sensorwert" gespeichert. Der "map" Befehl wird im Anschluss verwendet, um diesen Zahlenwert zwischen 0 und 410 in einen Zahlenwert zwischen -50 und +150 umzuwandeln. Dies entspricht dann am Ende dem Temperaturbereich zwischen -50°C und +150°C.

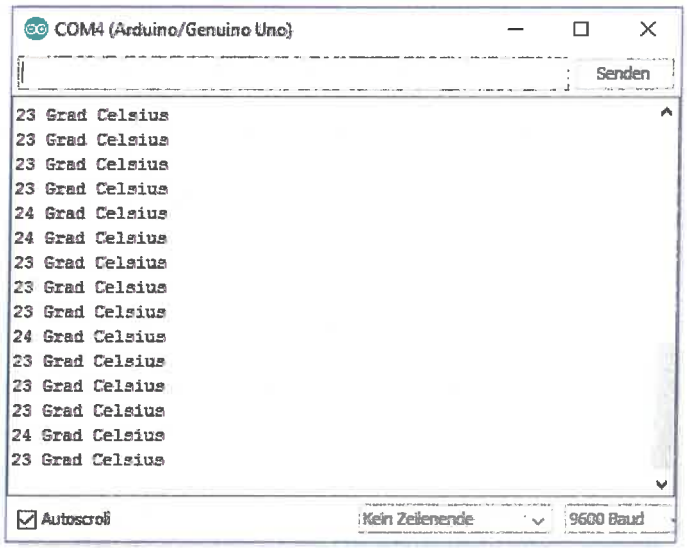
Map-Befehl allgemein: Variable = map (**a** , **b** , **c** , **d** , **e**) *Map-Befehl Beispiel:* temperatur = map(sensorwert, 0, 410, -50, 150);	**a**= umzuwandelnde Zahl **b**= Minimum Messbereich **c**= Maximum Messbereich **d**= Minimum Ausgabewert **e**= Maximum Ausgabewert

Nach der Umwandlung des analogen Messwertes in einen Temperaturwert, wird dieser mit dem Befehl "Serial.print(temperatur);" an den Seriellen Monitor gesendet und kann dann am PC abgelesen werden.

Sketch zur Temperaturmessung:

```
int TMP36 = A0;       //Der Sensor soll am analogen Pin A0 angeschlossen werden. Wir nennen
den Pin ab jetzt "TMP36"
int sensorwert;
int temperatur = 0;  //Unter der Variablen "temperatur" wird später der Temperaturwert
abgespeichert.
int t=500;            //Der Wert für „t“ gibt im Code die zeitlichen Abstände zwischen den
einzelnen Messungen vor.

void setup()
{
Serial.begin(9600);  //Im Setup beginnt die serielle Kommunikation,damit die Temperatur an
den Seriellen Monitor übertragen wird. Über die serielle Kommunikation sendet das Board die
Messwerte an den Computer. In der Arduino-Software kann man unter „Werkzeuge“ den
„Seriellen Monitor“ starten um die Messwerte zu sehen.
}

void loop()
{
sensorwert=analogRead(TMP36);     //Auslesen des Sensorwertes.
temperatur= map(sensorwert, 0, 410, -50, 150); //Umwandeln des Sensorwertes mit Hilfe des
"map" Befehls.
delay(t);// Nach jeder Messung ist je eine kleine Pause mit der Dauer „t“ in Millisekunden.
Serial.print(temperatur); //Nun wird der Wert „temperatur“ über die serielle Kommunikation
an den PC gesendet. Durch Öffnen des Seriellen Monitors in der Arduino-Software kann die
Temperatur abgelesen werden.
Serial.println(" Grad Celsius"); // Im Seriellen Monitor wird hinter der Temperatur die
Einheit eingeblendet.
}
```

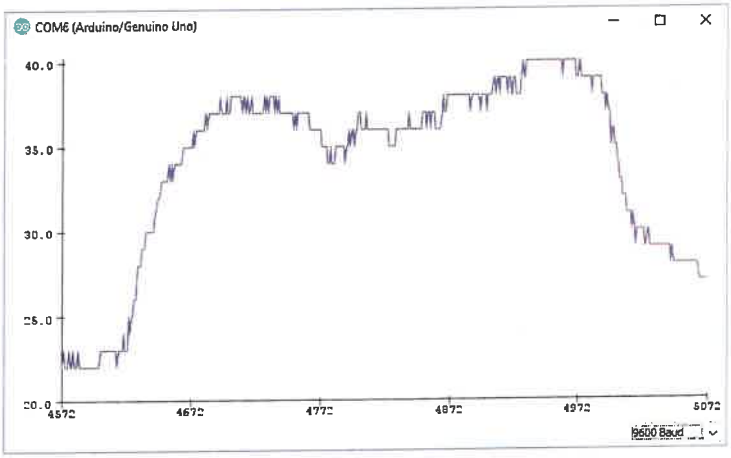
Nach dem Öffnen des Seriellen Monitors sollte das Ergebnis aussehen wie im folgenden Bild.



Der Serielle Plotter

Beim Thema Temperaturmessung ist auch der **„Serielle Plotter“** sehr nützlich. Mit dem Seriellen Plotter lassen sich Messwerte sofort visualisieren, ohne dass die Messwerte in einem anderen Programm (bspw. Tabellenkalkulation) aufbereitet werden müssen. Der Serielle Plotter kann im Bereich „Werkzeuge“ der Arduino-Software aufgerufen werden. Direkt nach dem öffnen werden die Daten visualisiert. Der Serielle Plotter fokussiert dabei automatisch den Messbereich, damit die Daten möglichst übersichtlich dargestellt werden. Die Y-Achse stellt dabei den Wert des gesendeten Datensatzes dar und die X-

Achse stellt aufsteigend die Anzahl der empfangenen Datensätze dar. In diesem Beispiel wird die Temperatur einer Teetasse dargestellt, wobei sich die Messwerte in einem Bereich zwischen 20°C und 40°C bewegen.



Erweiterung des Programms

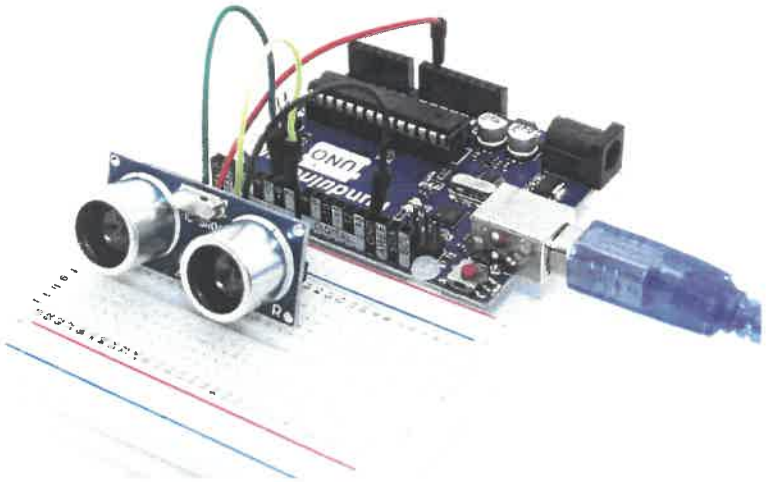
Sobald die Temperatur von 30°C erreicht ist, soll ein Warnsignal ertönen. Dazu wird ein Piezo-Lautsprecher mit dem „+" Pol an Pin5 des Arduino Mikrocontrollerboards angeschlossen. Der andere Pin des Lautsprechers wird mit GND verbunden.

```
int TMP36 = A0;
int sensorwert;
int temperatur = 0;
int t=500;
int piezo=5;//Das Wort „piezo" steht jetzt für die Zahl 5, also wird an Pin5 der Speaker
angeschlossen.
void setup()
{
Serial.begin(9600);
pinMode (piezo, OUTPUT);//Der Pin für den Piezo-Lautsprecher wird als Ausgang definiert, da
hier um zu piepsen eine Spannung benötigt wird.
}
void loop()
{
sensorwert=analogRead(TMP36);
temperatur= map(sensorwert, 0, 410, -50, 150);
delay(t);
Serial.print(temperatur);
Serial.println(" Grad Celsius");

if (temperatur>=30) //Es wird eine IF-Bedingung erstellt: Wenn der Wert für die Temperatur
über oder gleich 30 ist, dann…
{
digitalWrite(piezo,HIGH);  //…fange an zu piepsen.
}
else  //Und wenn das nicht so ist…
{
digitalWrite(piezo,LOW);  //…dann sei leise.
}
}
```

4.12 Entfernung messen (Ultraschall)

Aufgabe: Mit den Ultraschallsensor HC-SR04 und einem Arduino Mikrocontrollerboard soll eine Entfernung gemessen und mit dem Seriellen Monitor angezeigt werden.

Materialbox
1x Arduino-Board
1x HC-SR04 Ultraschallsensor
1x Breadboard
Einige Steckkabel

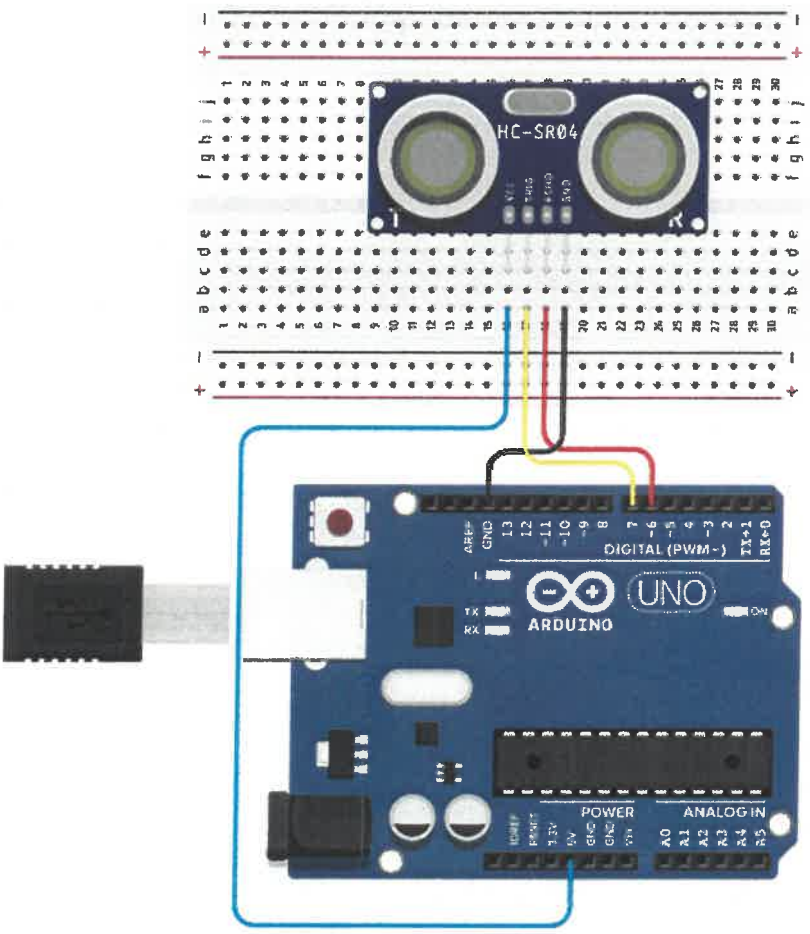


Wie funktioniert der Ultraschallsensor?

Der Sensor hat vier Kontakte:

1) 5V(+) 2) GND (–) 3) Echo 4) Trigger

Die Anschlüsse 5V und GND verstehen sich von selbst, sie versorgen den Sensor mit Energie. Der Pin „trigger" bekommt vom Mikrocontroller-Board ein kurzes Signal (5V), wodurch eine Schallwelle vom Ultraschallsensor ausgelöst wird. Sobald die Schallwelle gegen eine Wand oder sonstigen Gegenstand stößt, wird sie reflektiert und kommt irgendwann auch wieder zum Ultraschallsensor zurück. Sobald der Sensor diese zurückgekehrte Schallwelle erkennt, sendet der Sensor auf dem „echo" Pin ein 5V Signal an das Mikrocontroller-Board. Dieser misst dann lediglich die Zeit zwischen dem Aussenden und der Rückkehr der Schallwelle und rechnet diese Zeit dann in eine Entfernung um.



Sketch

```
int trigger=7;
//Trigger-Pin des Ultraschallsensors an Pin7 des Arduinoboards
int echo=6;
//Echo-Pim des Ultraschallsensors an Pin6 des Arduinoboards
long dauer=0;
//Das Wort dauer ist jetzt eine Variable, unter der die Zeit gespeichert wird, die eine Schallwelle bis zur Reflektion und zurück benötigt. Startwert ist hier 0.
long entfernung=0;  //Das Wort „entfernung“ ist jetzt die variable, unter der die berechnete Entfernung gespeichert wird. Info: Anstelle von „int“ steht hier vor den beiden Variablen „long“. Das hat den Vorteil, dass eine größere Zahl gespeichert werden kann. Nachteil: Die Variable benötigt mehr Platz im Speicher des Mikrocontrollers.
void setup()  //Beginn des Setups
{
Serial.begin (9600);  //Serielle Kommunikation starten, damit man sich später die Werte am Seriellen Monitor ansehen kann.
pinMode(trigger, OUTPUT);  //Trigger-Pin ist ein Ausgang
pinMode(echo, INPUT);  //Echo-Pin ist ein Eingang
}
void loop()
{
digitalWrite(trigger, HIGH);  //Jetzt sendet man eine Ultraschallwelle los.
delay(10);  //Dieser „Ton“ erklingt für 10 Millisekunden.
digitalWrite(trigger, LOW);  //Dann wird der „Ton“ abgeschaltet.
dauer = pulseIn(echo, HIGH);  //Mit dem Befehl „pulseIn“ zählt der  Mikrocontroller die Zeit in Mikrosekunden, bis der Schall zum Ultraschallsensor zurückkehrt.
entfernung = (dauer/2) * 0.03432; //Nun berechnet man die Entfernung in Zentimetern. Man teilt zunächst die Zeit durch zwei (weil man ja nur eine Strecke berechnen möchte und nicht die Strecke hin und zurück). Den Wert multipliziert man mit der Schallgeschwindigkeit in der Einheit Zentimeter/Mikrosekunde und erhält dann den Wert in Zentimetern.
if (entfernung >= 500 || entfernung <= 0)
//Wenn die gemessene Entfernung über 500cm oder unter 0cm liegt,…
{
Serial.println("Kein Messwert"); //dann soll der Serielle Monitor ausgeben „Kein Messwert“, weil Messwerte in diesen Bereichen falsch oder ungenau sind.
}
else  //Ansonsten…
{
Serial.print(entfernung);  //…soll der Wert der Entfernung am Seriellen Monitor ausgegeben werden.
Serial.println(" cm");  //Hinter dem Wert der Entfernung soll auch am Seriellen Monitor die Einheit "cm" angegeben werden.
}
delay(1000);  //Die Pause von einer Sekunde sorgt in ca. jeder neuen Sekunde für einen neuen Messwert.
}
```

Erweiterung des Programms

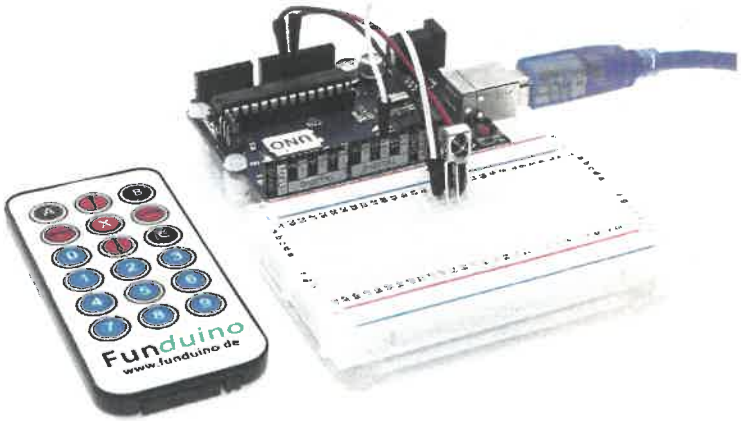
Wenn ein Abstand unter 80cm gemessen wird, soll ein Piezo-Lautsprecher piepsen.

```
int trigger=7;
int echo=6;
long dauer=0;
long entfernung=0;
int piezo=5;  //Das Wort piezo ist jetzt die Zahl 5
void setup()
{
Serial.begin (9600);
pinMode(trigger, OUTPUT);
pinMode(echo, INPUT);
pinMode(piezo, OUTPUT);  //Der Pin5 soll ein Ausgang sein, da der Lautsprecher eine
Spannung benötigt, um einen Ton zu erzeugen.
}
void loop()
{
digitalWrite(trigger, LOW);
delay(5);
digitalWrite(trigger, HIGH);
delay(10);
digitalWrite(trigger, LOW);
dauer = pulseIn(echo, HIGH);
entfernung = (dauer/2) * 0.03432;
if (entfernung >= 500 || entfernung <= 0)
{
Serial.println("Kein Messwert");
}
else
{
Serial.print(entfernung);
Serial.println(" cm");
}
//Es wird eine weitere IF-Bedingung erstellt:
if (entfernung <= 80)  //Wenn der Wert für die Entfernung unter oder gleich 80 ist, dann...
{
digitalWrite(piezo,HIGH);  //...fange an zu piepsen.
}
else  //Und wenn das nicht so ist,
{
digitalWrite(piezo,LOW);  //...dann sei leise.
}
delay(1000);
}
```

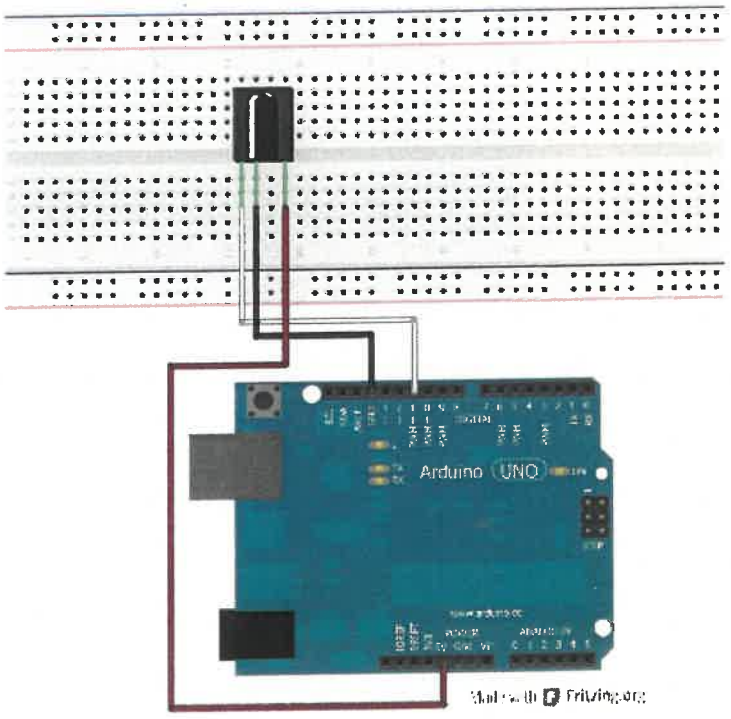
4.13 Infrarotfernbedienung

Aufgabe: Eine Infrarotfernbedienung zur Ansteuerung von Arduino Mikrocontrollern verwenden.

Materialbox
1x Arduino-Board
1x Infrarotfernbedienung
1x Infrarotsensor VS1838B
1x Breadboard
Einige Steckkabel



Mit Hilfe eines Infrarotempfängers kann ein Arduinoboard die Befehle einer Infrarotfernbedienung empfangen und auswerten. Die Daten werden dabei in Form von Infrarotlicht von der Fernbedienung zum Empfänger gesendet. Da unser Auge dieses Licht nicht wahrnehmen kann, können wir dieses Licht nicht sehen. Mit einem kleinen Trick kann man jedoch testen, ob bspw. eine Fernbedienung ein Infrarotsignal sendet. Dazu nimmt man eine Digitalkamera (zum Beispiel die vom Smartphone) und betrachtet über das Display die Infrarotdiode. Wenn die Fernbedienung betätigt wird, kann man die Infrarotdiode leuchten sehen. Das liegt daran, dass die Sensoren von Digitalkameras das Infrarotlicht wahrnehmen und darstellen können. Das Licht flackert leicht, da die Infrarotdiode sehr schnell an- und aus geht. Dahinter verbirgt sich ein ganz bestimmter Rhythmus, anhand dessen der Infrarotempfänger später auswerten kann, welche Taste an der Fernbedienung gedrückt wurde.



Für die Programmierung wird die Library „IRremote“ von Ken Shirriff in der **Version 2.2.3** benötigt. Die passende Version muss im Bibliotheksverwalter vor der Installation ausgewählt werden, da ansonsten die aktuellste Version installiert wird. Die Befehle in unserer Anleitung basieren jedoch auf die Version 2.2.3. Die Programmbibliothek kann im Bibliotheksverwalter mit dem Suchbegriff „irremote“ gefunden und installiert werden. Eine Anleitung zur Installation einer Bibliothek über den Bibliotheksverwalter befindet sich im Theorieteil zu dieser Anleitung. Dieser Sketch ist eine leichte Abwandlung des Sketches

„IRrecvDemo“, welcher sich nach Installation der Library auch in der Arduino Software unter den Beispielen zu dieser Library finden lässt.

Dieser Sketch ist sehr kurz gehalten und bietet sich daher sehr gut für die ersten Versuche an.

```
//Informationen über das ursprüngliche Programm „IrrecvDemo“.
* IRremote: IRrecvDemo - demonstrates
receiving IR codes with IRrecv
* An IR detector/demodulator must be connected to the input RECV_PIN.
* Version 0.1 July, 2009
* Copyright 2009 Ken Shirriff
* http://arcfn.com

#include <IRremote.h>  // Das Programm greift an dieser Stelle auf eine „Library“ zurück.
Das erleichtert einem viel Arbeit. Denn das Infrarotlicht wird mit einem Code verschlüsselt
gesendet. Um diesen Code selber auszulesen und in passende Werte umzuwandeln, wären sehr
viele Zeilen Code erforderlich.
int RECV_PIN = 11;  //Der Kontakt, der am Infrarotsensor die Daten ausgibt, wird mit Pin 11
des Arduinoboards verbunden.
IRrecv irrecv(RECV_PIN);  //An dieser Stelle wird ein Objekt definiert, dass den
Infrarotsensor an Pin 11 ausliest.
decode_results results;  //Dieser Befehl sorgt dafür, dass die Daten, die per Infrarot
eingelesen werden, unter „results“ abgespeichert werden.

void setup()
{
Serial.begin(9600);  //Im Setup wird die Serielle Verbindung gestartet, damit man sich die
Empfangenen Daten der Fernbedienung per seriellen Monitor ansehen kann.
irrecv.enableIRIn();  //Dieser Befehl initialisiert den Infrarotempfänger.
}

void loop()
{ //Der loop-Teil fällt durch den Rückgriff auf die „library“ sehr kurz aus.
if (irrecv.decode(&results)) { //Wenn Daten empfangen wurden,
Serial.println(results.value, DEC); //werden sie als Dezimalzahl (DEC) an den Seriellen
Monitor ausgegeben.
irrecv.resume(); //Der nächste Wert soll vom IR-Empfänger eingelesen werden.
}
}
```

Ein Druck auf die Taste „1“ der Infrarotfernbedienung bewirkt, dass am Seriellen Monitor die Zahl „16724175“ ausgegeben wird. Dies ist der entschlüsselte Zahlencode, der sich hinter dieser Taste verbirgt. Wenn man die Taste dauerhaft gedrückt hält, erscheint permanent die Zahl „4294967295“. Dies ist der Code der angibt, dass eine Taste dauerhaft gedrückt wird. Diese Zahl ist unabhängig davon, welche Taste gedrückt wird. Es können aber auch noch andere Zahlen auftauchen, falls eine Taste nur ganz kurz oder pulsierend gerückt wird. In dem Fall kann der Sensor keinen eindeutigen Wert auslesen.

Erweiterung des Programms

Beim Drücken der Taste“1“ soll eine LED an gehen und beim Drücken der Taste“2“ soll die LED aus gehen.

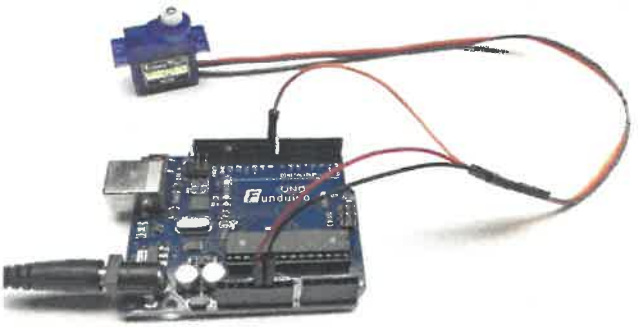
```
#include <IRremote.h>
int RECV_PIN = 11;
IRrecv irrecv(RECV_PIN);
decode_results results;

void setup()
```

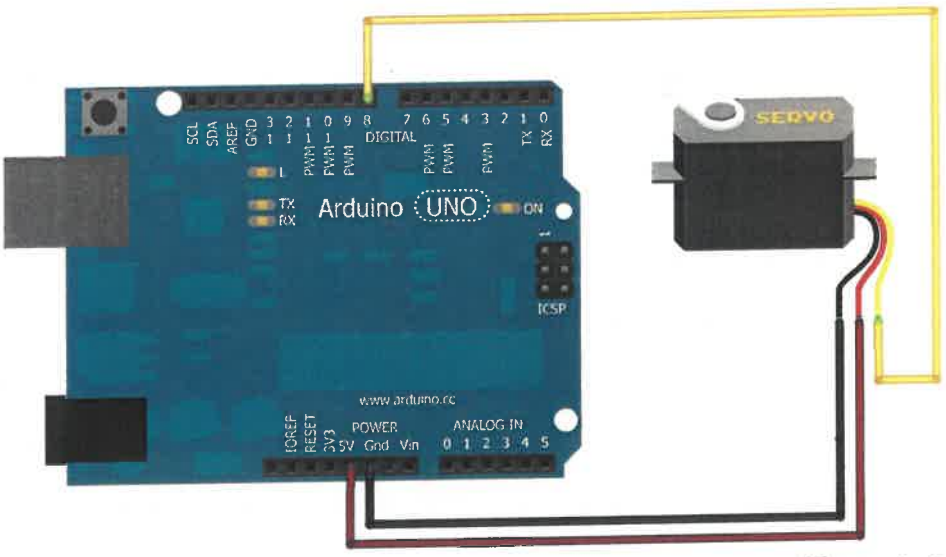
```
{
Serial.begin(9600);
pinMode (13, OUTPUT);  //An Pin 13 wird eine LED angeschlossen.
digitalWrite(13, LOW);  //Diese soll zunächst aus sein
irrecv.enableIRIn();
}
void loop() {
if (irrecv.decode(&results)) {
Serial.println(results.value, DEC);
if (results.value == 16724175)  //Wenn der Infrarotempfänger die Zahl „16724175“ ausgelesen
hat (entsprechend der Taste“1“ der Fernbedienung)
{digitalWrite (13, HIGH);}  //soll die LED an gehen.
if (results.value == 16718055)  //Wenn der Infrarotempfänger die Zahl  „16718055“
ausgelesen hat (entsprechend der Taste“2“ der Fernbedienung),
{digitalWrite (13, LOW);}  //soll die LED aus gehen.
irrecv.resume();
}
}
```

4.14 Servomotor ansteuern

Materialbox
1x Arduino-Board
1x Servo
Einige Steckkabel

Aufgabe: Ein Servomotor, auch *„Servo“* genannt, soll von einem Arduino-Mikrocontroller angesteuert werden. Der Servo soll dazu in diesem Beispiel drei verschiedene Positionen ansteuern und zwischen den Positionen eine kurze Zeit warten.





Servomotoren sind nicht mit regulären Elektromotoren vergleichbar, da es keine Drehachse gibt, die sich permanent drehen kann. Die Drehachse ist nur in einem kleinen Winkelbereich, meistens ca. 180°, beweglich. Im Inneren des Servos befindet sich ein Potentiometer, welches die durch den Mikrocontroller vorgegebene Winkelposition während des Betriebs permanent überwacht. Durch eine kleine Regelelektronik im

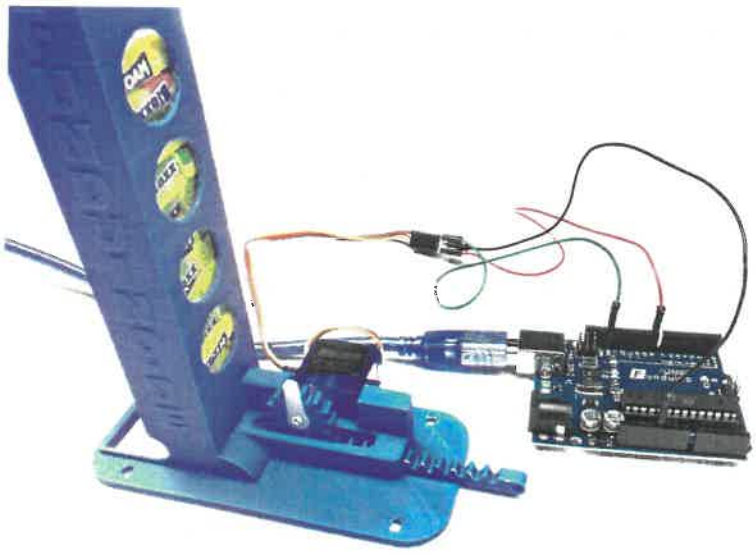
Inneren des Servos wird die Position auch bei äußeren mechanischen Einflüssen gehalten und ggf. korrigiert. Die Ansteuerung erfolgt über die Signalleitung in Form einer elektrischen Spannung, die vom Mikrocontroller ausgegeben wird.

Sketch

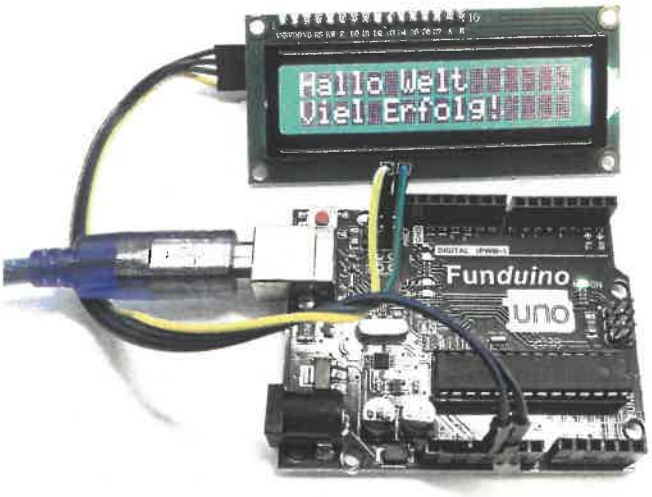
```
#include <Servo.h>  //Die Servobibliothek wird aufgerufen.
Servo servoblau;    //Erstellt für das Programm ein Servo mit dem Namen „servoblau“.
void setup()
{
servoblau.attach(8); //Servo wird mit Pin8 verbunden.
}
void loop()
{
servoblau.write(0);    //Position 1 ansteuern mit dem Winkel 0°.
delay(3000);           //Das Programm stoppt für 3 Sekunden.
servoblau.write(90);   //Position 2 ansteuern mit dem Winkel 90°.
delay(3000);           //Das Programm stoppt für 3 Sekunden.
servoblau.write(180);  //Position 3 ansteuern mit dem Winkel 180°.
delay(3000);           //Das Programm stoppt für 3 Sekunden.
}
```

Der Servo lässt sich in vielen Projekten einsetzen. So kann der Servo zum Beispiel zur Verriegelung einer Schachtel, als Materialvorschub in einem Süßigkeitenspender oder als Modell-Bahnschranke verwendet werden. In Kombination mit einer roten LED lässt sich mit wenigen Code-Zeilen ein Modell-Bahnübergang programmieren.

Beispiel: Ein Servomotor in einem Bonbon-Spender



4.15 LCD Display mit I²C Schnittstelle



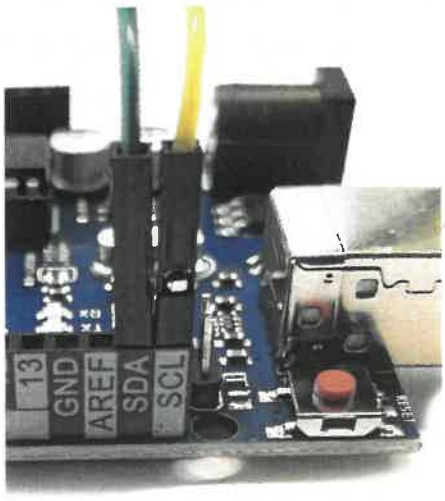
Aufgabe: Einen Text oder Messwerte auf einem LCD mit I²C Modul anzeigen.

Materialbox
1x Arduino-Board
1x I²C LCD Modul
Einige Steckkabel

Mit einem LCD-Display kann man Buchstaben und Ziffern darstellen. Dies ist in vielen Anwendungen nützlich, zum Beispiel um Messwerte oder auch Menüs darzustellen. Bisher war die Anzeige von Daten nur im Seriellen Monitor möglich. Mit Hilfe des LCD lassen sich aber auch Daten darstellen, wenn kein Computer am Arduino-Mikrocontroller angeschlossen ist.

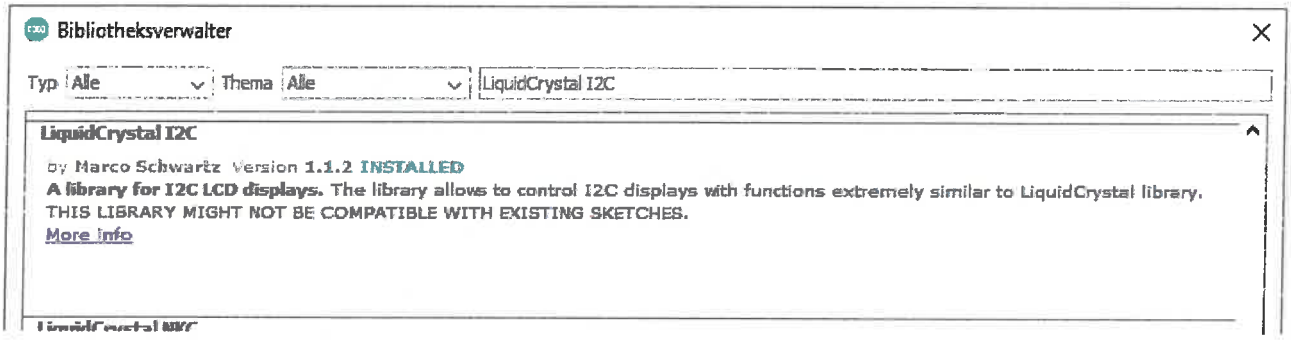
Das LCD mit angelötetem I²C-Modul ermöglicht die Verwendung eines LCD mit einer einfachen Verkabelung. Dies ist bei komplexeren Projekten besonders vorteilhaft. Ein weiterer großer Vorteil zum normalen LCD Display besteht darin, dass sich mehrere I²C-Module, wie Displays oder Sensoren, gleichzeitig über nur vier Leitungen verwenden lassen. Für die perfekte Kontrasteinstellung des Displays befindet sich auf der Rückseite ein Drehregler.

Praxistipp: Es gibt I²C-Module von verschiedenen Herstellern, die häufig unterschiedliche Ausgangs-I²C Adressen haben. Daher muss beim Programmieren darauf geachtet werden, dass die richtige I²C-Adresse im Sketch verwendet wird. Sollte die Adresse des jeweiligen I²C-Moduls nicht bekannt sein, kann diese jedoch auch mit einem I²C-Scanner-Sketch herausgefunden werden. Bei den Modulen der Funduino GmbH ist die Standartadresse in der Regel 0x3F oder 0x27.



Verkabelung: Am I2C LCD Modul sind nur vier Kontakte vorhanden.
GND am LCD wird mit dem GND Kontakt am Mikrocontroller verbunden.
VCC am LCD wird mit dem 5V Kontakt am Mikrocontroller verbunden.
SDA am LCD wird mit dem SDA Kontakt am Mikrocontroller verbunden
SCL am LCD wird mit dem SCL Kontakt am Mikrocontroller verbunden
Praxistipp: Wer die entsprechenden Pins am Board nicht findet, kann sich noch einmal die Beschriftung der Mikrocontroller im Bereich „Hardware" ansehen.

Programmieren: Um mit dem I²C LCD Modul zu arbeiten, benötigt man eine Library, welche noch nicht im Arduino Programm vorinstalliert ist. Wir verwenden in dieser Anleitung die „LiquidCrystal I2C" Library. Die Library kann über die Bibliothekenverwaltung der Arduino-Software hinzugefügt werden. Als Suchbegriff direkt den Namen „**LiquidCrystal I2C**" eingeben und dann die Library von „**Marco Schwarz**" auswählen.



Eine detaillierte Beschreibung, wie Bibliotheken eingefügt werden, findet man im Theorieteil zu dieser Anleitung im Unterpunkt „Bibliotheken zur Arduino Software hinzufügen".

Sketch:

```
#include <Wire.h> // Wire Bibliothek einbinden.
#include <LiquidCrystal_I2C.h> // Vorher hinzugefügte LiquidCrystal_I2C Bibliothek
einbinden.
LiquidCrystal_I2C lcd(0x27, 16, 2); //Hier wird festgelegt, um was für einen Display es
sich handelt. In diesem Fall eines mit 16 Zeichen in 2 Zeilen und der HEX-Adresse 0x27. Für
ein vierzeiliges I2C-LCD verwendet man den Code "LiquidCrystal_I2C lcd(0x27, 20, 4)".
void setup()
{
lcd.init(); //Im Setup wird der LCD gestartet.
lcd.backlight(); //Hintergrundbeleuchtung einschalten.
}                //(lcd.noBacklight(); schaltet die Beleuchtung aus).
void loop()
{
lcd.setCursor(0, 0);
//Hier wird die Position des ersten Zeichens festgelegt. In diesem Fall bedeutet (0,0) das
erste Zeichen in der ersten Zeile.
lcd.print("Hallo Welt"); // in der ersten Zeile erscheint der Text „Hallo Welt"
lcd.setCursor(0, 1);// In diesem Fall bedeutet (0,1) das erste Zeichen in der zweiten
Zeile.
lcd.print("Viel Erfolg!"); // in der zweiten Zeile erscheint der Text „Viel Erfolg!"
}
```

Anwendungsbeispiel:

Mit dem I2C LCD Modul können Messwerte angezeigt werden.

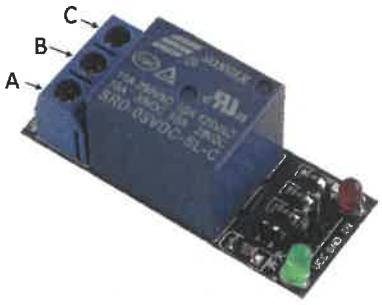
Es folgt ein Beispielcode, bei dem ein Feuchtigkeitssensor an Pin A0 angeschlossen wurde. Der Messwert des Sensors wird auf dem LCD dargestellt.

```
#include <Wire.h> // Wire Bibliothek einbinden
#include <LiquidCrystal_I2C.h> // Vorher hinzugefügte LiquidCrystal_I2C Bibliothek
einbinden.
```

```
LiquidCrystal_I2C lcd(0x27, 16, 2); //Hier wird festgelegt, um welches Display es sich handelt. Die HEX-Adresse 0x27 ist eine Standartadresse für LCD mit einem einfachen I²C-Modul auf der Rückseite. Wenn das I²C Modul Lötstellen zur Veränderung der HEX-Adresse aufweist, ist die Standartadresse "0x3F". In diesem Fall handelt es sich um ein LCD mit 16 Zeichen in 2 Zeilen (16,2). Für ein vierzeiliges I2C-LCD verwendet man den Code "LiquidCrystal_I2C lcd(0x27, 20, 4)".
int messwert=0;
void setup()
{
lcd.init(); //Im Setup wird der LCD gestartet.
lcd.backlight(); //Hintergrundbeleuchtung einschalten („lcd.noBacklight()“ schaltet die Beleuchtung aus).
}
void loop()
{
messwert=analogRead(A0); // Der Messwert vom Analogen Eingang A0 soll ausgelesen und unter der Variablen „messwert“ gespeichert werden. lcd.setCursor(0, 0); //Hier wird die Position des ersten Zeichens festgelegt. In diesem Fall bedeutet (0,0) das erste Zeichen in der ersten Zeile.
lcd.setCursor(0, 0); // Position des Textes auf dem LCD: In diesem Fall bedeutet (0,0) das erste Zeichen in der ersten Zeile.
lcd.print("Messwert: ");
lcd.setCursor(0, 1); // Position des Textes auf dem LCD: In diesem Fall bedeutet (0,1) das erste Zeichen in der zweiten Zeile.
lcd.print(messwert);
delay(500);
}
```

Das Ergebnis einer Messwertanzeige nach dem oben genannten Sketch ist auf diesem Bild dargestellt.

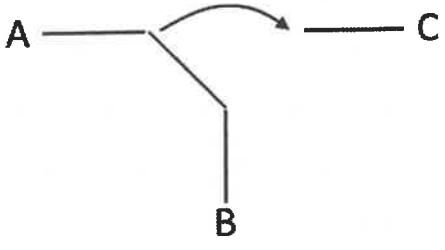
4.16 Relais verwenden



Aufgabe: Eine Relaiskarte im Sekundentakt schalten lassen.

Materialbox
1x Arduino-Board
1x Relaiskarte
Einige Steckkabel

Beim Arbeiten mit Arduino sind Relais sehr wichtig. Relais sind Schalter, die einen größeren Strom fließen lassen können, als es vom Mikrocontroller aus möglich wäre. Erreicht wird das dadurch, dass über den kleinen Ausgangsstrom eines digitalen Pins am Mikrocontrollerboard eine andere spannungsführende Leitung freigeschaltet werden kann.



Ein Anwendungsbeispiel wäre eine Wasserpumpe zur Pflanzenbewässerung, die je nach Bedarf vom Mikrocontroller ein- oder ausgeschaltet wird.

An den Kontaktpins, zwischen der roten und der grünen LED, wird die Relaiskarte mit dem Arduino verbunden. Die Karte ist im Betrieb dauerhaft mit 5V+ und GND (-) verbunden. Der Pin mit der Aufschrift „IN" wird mit einem digitalen Pin des Arduinoboards verbunden. Solange an dem Pin „IN" kein Signal anliegt (ausgegeben vom digitalen Pin des Arduino), sind die Schraubkontakte A und B miteinander verbunden. Sobald ein Signal anliegt, werden die Kontakte B und C miteinander verbunden.

Praxistipp: Es gibt Relaiskarten, die schalten, wenn an dem Pin „IN" GND angelegt wird (LOW trigger) und es gibt Relaiskarten, die schalten, wenn an dem Pin „IN" eine Spannung von 5V+ angelegt wird (HIGH trigger). Welche Version man hat, lässt sich leicht feststellen, indem man den „Signal-„Pin einmal mit GND und einmal mit 5V+ des Arduino Boards verbindet. Das Schalten der Relaiskarte ist durch ein lautes Knacken deutlich zu erkennen.

An den Kontakten (A, B, C) kann ein elektrisches Gerät angeschlossen werden, bei dem ein höherer elektrischer Strom fließt, als es der Arduino liefern könnte. Zum Beispiel einen großen Elektromotor, eine große Lampe usw. Als Beispielcode kann in diesem Fall der einfache „Blink-" Code verwendet werden. Anstelle der LED schließt man den Ausgabepin des Arduinoboards an den „Signal-" Pin der Relaiskarte an. Das Relais wird dann im Sekundentakt ein- und ausschalten, bzw. im Sekundentakt die Verbindung zwischen A und B bzw. B und C herstellen.

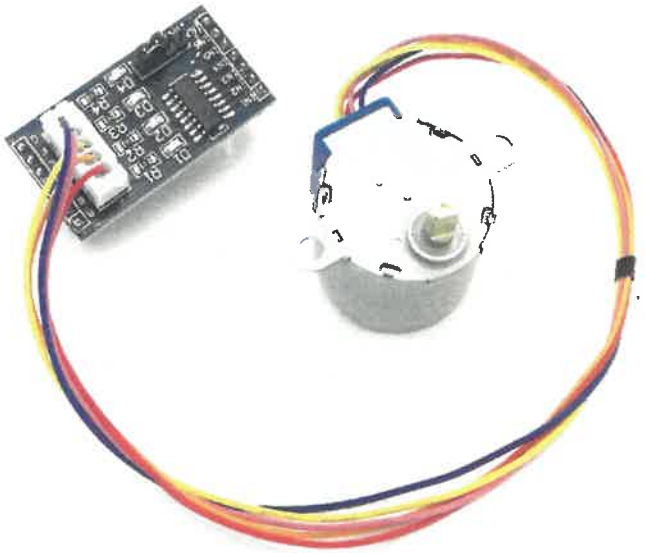
Sketch

```
void setup()
{
pinMode(6, OUTPUT);
}
void loop()
{
digitalWrite(6, HIGH);//An dieser Stelle wird das Relais einschalten
delay(1000);          //...eine Sekunde warten
digitalWrite(6, LOW); //...und wieder ausschalten
delay(1000);          //...und eine Sekunde warten.
}                     //Dann beginnt der „Loop" erneut.
```

4.17 Schrittmotor

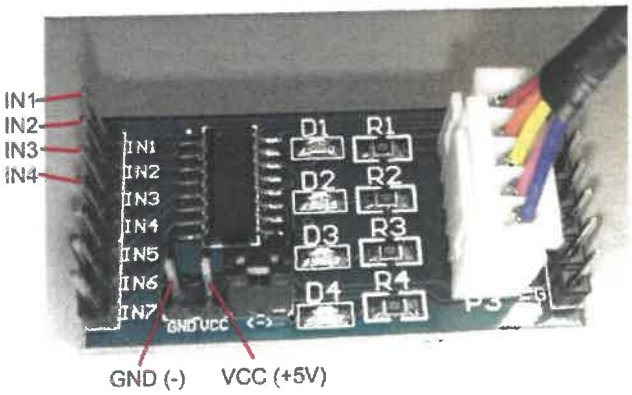
Aufgabe: Einen Schrittmotor ein ganze Umdrehung vor- und zurückdrehen lassen.

Materialbox
1x Arduino-Board
1x Schrittmotor mit Steuerplatine
Einige Steckkabel



Bei diesem Schrittmotor handelt es sich um einen Schrittmotor, der sich speziell für kleine Anwendungen mit Arduino Mikrocontrollerboards eignet. Die Besonderheit liegt darin, dass er ohne eine externe Spannungsversorgung betrieben werden kann. Der Motor entwickelt dabei ein relativ hohes Drehmoment. Dies wird durch ein Getriebe realisiert, welches innerhalb des Metallgehäuses vor dem eigentlichen Schrittmotor verbaut wurde. Dadurch wird es in dieser kompakten Bauweise überhaupt erst möglich, dass sich eine ganze Umdrehung der Antriebswelle auf 2048 Einzelschritte aufteilen lässt. Ein kleiner daraus resultierender Nachteil ist die langsame maximale Drehgeschwindigkeit.

Der Schrittmotor wird an eine Motorsteuerungsplatine angeschlossen. Diese versorgt den Motor mit ausreichend elektrischer Energie, damit die Leistung nicht von den digitalen Pins des Arduinoboards aufgebracht werden muss. Die Steuerungsplatine gibt es in verschiedenen Bauformen und Farben, die Anschlussbelegung ist jedoch gleich.



Verkabelung

IN1 der Motorsteuerplatine wird an Pin6 angeschlossen.

IN2 der Motorsteuerplatine wird an Pin5 angeschlossen.

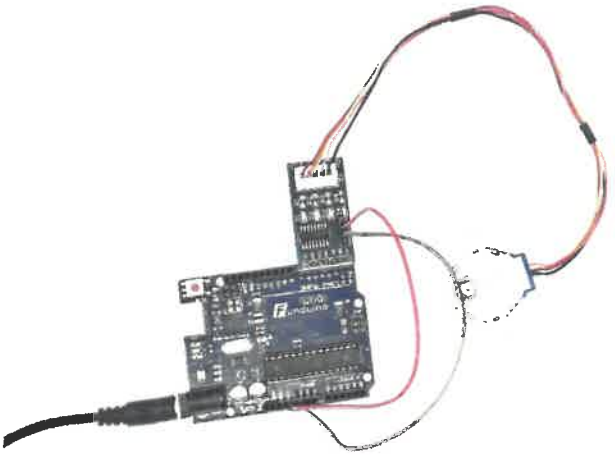
IN3 der Motorsteuerplatine wird an Pin4 angeschlossen.

IN4 der Motorsteuerplatine wird an Pin3 angeschlossen.

GND der Motorsteuerplatine wird an einem GND Pin am Arduinoboard angeschlossen.

VCC der Motorsteuerplatine wird an den 5V Pin am Arduinoboard angeschlossen.

Bei Motorsteuerplatinen mit nach unten ausgerichteten Pins kann die Steuerplatine auch direkt auf das Mikrocontrollerboard aufgesteckt werden.



Dies ist ein Beispielcode, der den Motor abwechselnd um 2048 Schritte (entspricht einer ganzen Umdrehung) vor- und zurückdrehen lässt:

```
#include <Stepper.h>        // Hinzufügen der Programmbibliothek.
int SPU = 2048;             // Schritte pro Umdrehung.
Stepper Motor(SPU, 3,5,4,6); // Der Schrittmotor erhält die Bezeichnung "Motor" und es wird
angegeben, an welchen Pins der Motor angeschlossen ist.
void setup()    //Hier beginnt das Setup.
{
Motor.setSpeed(5); //Angabe der Geschwindigkeit in Umdrehungen pro
}                  Minute.
void loop() {
Motor.step(2048);    // Der Motor macht 2048 Schritte, das entspricht einer Umdrehung.
delay(1000); // Durch diese Pause bleibt der Motor nach der Drehung für eine Sekunde
stehen.
Motor.step(-2048);  // Der Motor macht durch das Minuszeichen 2048 Schritte in die andere
Richtung.
delay(1000);        // Durch diese Pause bleibt der Motor nach der Drehung für eine Sekunde
stehen.
}
```

Drehung ohne große Schrittmengen

Die Verwendung von Befehlen mit einer großen Schrittmenge „x" (*Motor.step(x);*) ist in umfangreicheren Sketchen häufig störend, da während der Ausführung keine weiteren Signale vom Mikrocontroller empfangen oder gesendet werden können. Das ist ein ähnliches Problem wie bei langen Pausen („Delays"). Daher bietet es sich beim Schrittmotor an, die Drehbewegung in einer Schleife auszuführen.

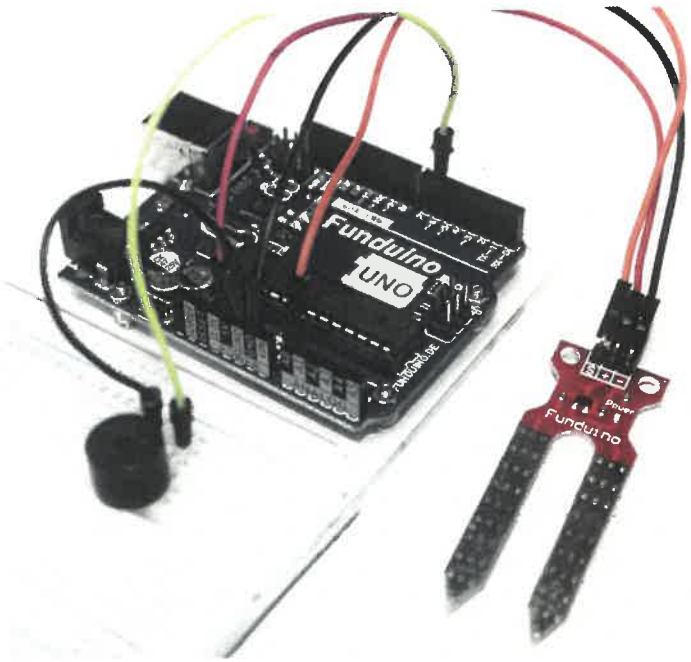
Hier ein kleines Beispiel:

In diesem Fall soll sich der Motor drehen, bis ein Taster gedrückt wird. Es könnte die Anhaltefunktion in einer Maschine sein. Natürlich soll bei einer solchen Funktion der Motor sofort stoppen und nicht erst seine Drehbewegung beenden. Mit einer großen Schrittmenge könnte man die Drehbewegung nicht an einem beliebigen Punkt stoppen, sondern erst wenn der Motor seine Drehbewegung beendet hat und der Mikrocontroller bereit für die nächsten Befehle ist.

```
#include <Stepper.h>
int SPU = 2048;
Stepper Motor(SPU, 3,5,4,6);
int Taster=10; // Taster an Pin10
int Tasterstatus=0; // Variable für den Status des Tasters
void setup()
{
Motor.setSpeed(4);
pinMode(Taster, INPUT);
}
void loop()
{
Tasterstatus=digitalRead(Taster); //Zunächst wird der Taster abgefragt.
while(Tasterstatus == HIGH) //Solange der Wert des Tasters HIGH, also gedrückt ist,...
 {
 delay(1);//...springt der Sketch in diese Klammer und verbleibt hier.
 Tasterstatus=digitalRead(Taster); //Innerhalb der Schleife muss der Taster immer wieder
abgefragt werden, damit eine Änderung des Status vom Mikrocontroller erkannt und
verarbeitet werden kann. Nur dadurch kann die Schleife beim loslassen des Tasters wieder
verlassen werden.
 } //Wenn der Taster nicht mehr gedrückt ist, springt der Sketch in den regulären Loop
zurück und führt dort weitere Befehle aus.
Motor.step(1); //...Drehe den Motor um einen einzigen Schritt.
} //Jetzt beginnt der Sketch von vorn. Solange der Taster nicht gedrückt ist, wird die
"While" Funktion nicht ausgeführt und der Motor dreht sich Schritt für Schritt weiter.
```

4.18 Feuchtigkeitssensor

Aufgabe: Der Messwert eines Feuchtigkeitssensors soll ausgelesen und auf dem Seriellen Monitor angezeigt werden.

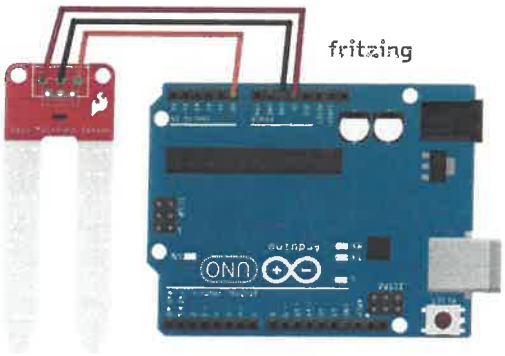


Materialbox
1x Arduino-Board
1x Feuchtigkeitssensor
Einige Steckkabel

Mit einem Feuchtigkeitssensor (Moisture Sensor) kann, wie es der Name schon sagt, die Feuchtigkeit gemessen werden. Dies bezieht sich jedoch auf die direkt anliegende Feuchtigkeit, wie z.B. Hautfeuchtigkeit oder Bodenfeuchtigkeit, nicht jedoch von Luftfeuchtigkeit. Er wird zum Beispiel verwendet, um die Feuchtigkeit im Boden von Pflanzen zu messen. Im

Falle von Trockenheit könnte dann ein Alarmsignal ertönen, oder eine elektrische Wasserpumpe würde aktiviert werden um die Pflanze automatisch mit Wasser zu versorgen. Der Sensor eignet sich ebenfalls, um den Wasserstand im Bereich des Sensors zu messen. Die Funktionsweise ist einfach. An den beiden Kontakten des Sensors liegt eine Spannung an. Je höher die Feuchtigkeit zwischen den beiden Kontakten ist, desto besser kann der Strom von einem Kontakt zum anderen fließen. Dieser Wert wird im Sensor elektronisch aufbereitet und in Form eines analogen Signals an einen analogen Eingang des Arduinoboards übermittelt. Da das Board, wie bereits in vorherigen Anleitungen beschrieben, keine elektrische Spannung als solche messen kann, wandelt es die am analogen Pin anliegende Spannung in einen Zahlenwert um. Die Messwerte von 0 bis 5 Volt entsprechen einem Zahlenwert von 0 bis 1023 (Das sind 1024 Zahlen, da die Null als erster Zahlenwert gezählt wird).

Bei dem Feuchtigkeitssensor liegt das obere Limit jedoch bei einem Zahlenwert von ca. 800, wenn der Sensor komplett im Wasser eingetaucht ist. Die genaue Kalibrierung ist jedoch abhängig vom Sensor und von der Art der Flüssigkeit, die gemessen wird (bspw. hat Salzwasser eine bessere Leitfähigkeit und der Messwert wäre entsprechend höher).

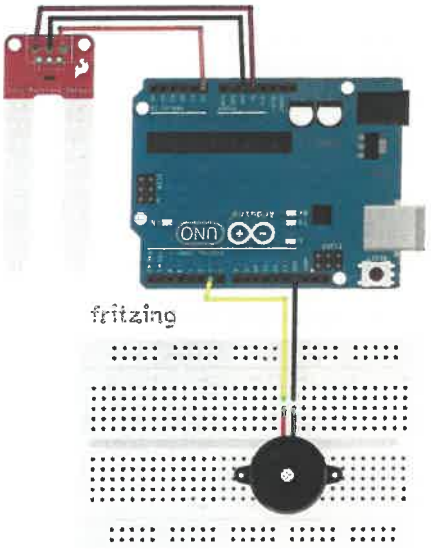


Die Programmierung ist sehr einfach und ähnelt sehr stark dem Auslesen von Potentiometern, da lediglich ein analoger Spannungswert ausgelesen wird.

```
int messwert=0;  //Unter der Variablen „messwert“ wird später der Messwert des Sensors
gespeichert.
void setup()
{                    //Hier beginnt das Setup.
Serial.begin(9600);  //Die Kommunikation mit dem seriellen Port wird gestartet. Das
benötigt man, um sich den ausgelesenen Wert im Seriellen Monitor anzeigen zu lassen.
}
void loop()
{                     //Mit dieser Klammer wird der Loop-Teil geöffnet.
messwert=analogRead(A0);//Die Spannung an dem Sensor wird ausgelesen und unter der Variable
„messwert“ gespeichert.
Serial.print("Feuchtigkeits-Messwert:");  //Ausgabe am Seriellen Monitor: Das Wort
„Feuchtigkeits-Messwert:“
Serial.println(messwert);  //und im Anschluss der eigentliche Messwert
delay(500);  //Zum Schluss noch eine kleine Pause, damit nicht zu viele Zahlenwerte in
kurzer Zeit am Seriellen Monitor erscheinen.
}
```

Praxistipp: Wir empfehlen, aufgrund der Elektrolyse durch das Wasser bzw. die Feuchtigkeit an den Sensoren, die Messungen nicht im Sekundentakt durchzuführen. Aus Erfahrungen durch Tests direkt im Wasser, empfehlen wir einen Abstand von 15 Minuten zwischen jeder Messung. Bei dauerhaften Messungen im Sekundenbereich,

bei direktem und durchgängigen Wasserkontakt, entstehen durch die Elektrolyse nach ca. 24 Stunden Schäden am Sensor. Für dauerhafte Messungen werden kontaktlose „kapazitive" Sensoren benötigt.



Erweiterung des Programms

Ein Alarmsignal soll mit Hilfe eines Piezo-Speakers ertönen, sobald die gemessene Feuchtigkeit einen bestimmten Grenzwert unterschreitet. Als Grenzwert wird in diesem Fall der Wert „200" festgelegt.

```
int messwert=0;
int PIEPS=6;   //Mit dem Namen „PIEPS" wird jetzt der Pin6 bezeichnet, an dem ein Piezo-
Speaker angeschlossen wird.
void setup()
{
Serial.begin(9600);
pinMode (6,OUTPUT);  //Im Setup wird der Pin6 als Ausgang festgelegt
}
void loop()
{
messwert=analogRead(A0);
Serial.print("Feuchtigkeits-Messwert:");
Serial.println(messwert);
delay(500);
if ( messwert <200)          //Hier beginnt die Abfrage: Wenn der Sensorwert kleiner als
„200" ist, dann…
  {
  digitalWrite(PIEPS, HIGH);  //…soll der Piezo-Speaker piepsen,
  }
  else                        //…ansonsten…
  {
  digitalWrite(PIEPS, LOW);   //…soll er leise sein.
  }
}
```

4.19 Tropfensensor

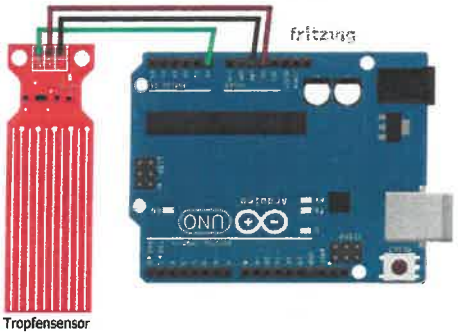
Aufgabe: Den Messwert eines Tropfensensors auslesen und auf dem Seriellen Monitor anzeigen.

Materialbox
1x Arduino-Board
1x Tropfensensor
Einige Steckkabel

Mit einem Tropfensensor oder auch Flüssigkeitssensor kann man wie es der Name schon sagt, die Tropfen einer Flüssigkeit detektieren. Dazu muss sich die Flüssigkeit direkt auf dem Sensor befinden. Es reicht bereits ein kleiner Tropfen aus, um einen eindeutigen Messwert zu erhalten.

Man kann den Sensor zum Beispiel als Regensensor verwenden. Sobald ein Tropfen auf den Sensor trifft, kann das Arduinoboard eine Aktion ausführen wie z.B. eine Markise einrollen, Jalousien schließen, einen Alarm auslösen oder einen Scheibenwischer betätigen.

Die Funktionsweise ist einfach. An den langen Kontaktstellen, die den Sensor durchziehen, liegt eine Spannung an. Sobald eine Flüssigkeit bspw. durch einen Tropfen zwei Kontakte verbindet, fließt ein kleiner Strom von einem Kontakt zum anderen. Dieser Wert wird im Sensor elektronisch aufbereitet und in Form eines analogen Signals an einen analogen Eingang des Boards übermittelt. Da das Board, wie bereits in vorherigen Tutorials beschrieben, keine elektrische Spannung als solche messen kann, wandelt es die am analogen Pin anliegende Spannung in einen Zahlenwert um. 0 bis 5 Volt entspricht einem Zahlenwert von 0 bis 1023 (Das sind 1024 Zahlen, da die Null als erster Zahlenwert gezählt wird).



Tropfensensor

Bei dem Flüssigkeitssensor liegt der Wert im Trockenen bei Null „0". Sobald ein Tropfen Wasser auf die Kontakte des Sensors trifft, liegt der Wert bei ca „480". Je mehr Tropfen sich auf dem Sensor befinden, desto höher ist der Wert.

Im ersten Code geht es nur darum, den Sensorwert mit dem Arduinoboard auszulesen und mit dem Seriellen Monitor darzustellen.

Die Programmierung ist sehr einfach und ähnelt sehr stark dem Auslesen von Potentiometern oder dem Auslesen des Feuchtigkeitssensors, da einfach nur ein analoger Wert ausgelesen wird.

Sketch

```
int messwert=0;   //Unter der Variablen „messwert“ wird später der Messwert des Sensors gespeichert.
void setup()  // Hier beginnt das Setup.
{
Serial.begin(9600);  //Die Kommunikation mit dem seriellen Port wird gestartet. Diese benötigt man, um sich den ausgelesenen Wert im Seriellen Monitor anzeigen zu lassen.
}
void loop()
{   // Mit dieser Klammer wird der Loop-Teil geöffnet.
messwert=analogRead(A0);   //Die Spannung an dem Sensor wird ausgelesen und unter der Variable „messwert“ gespeichert.
Serial.print("Feuchtigkeits-Messwert:");  //Ausgabe am Seriellen Monitor: Das Wort „Feuchtigkeits-Messwert:“
Serial.println(messwert);  //und im Anschluss der eigentliche Messwert
delay(500); //Zum Schluss noch eine kleine Pause, damit nicht zu viele Zahlenwerte ausgegeben werden.
}
```

Praxistipp: Wir empfehlen, aufgrund der Elektrolyse durch das Wasser bzw. die Feuchtigkeit an den Sensoren, die Messungen nicht in einem Sekundenabstand durchzuführen. Aus Erfahrungen durch Tests direkt im Wasser, empfehlen wir einen Abstand von 15 Minuten zwischen jeder Messung. Bei Messungen im Sekundenbereich, bei direktem und durchgängigen Wasserkontakt, entstehen durch die Elektrolyse bereits nach ca. 24 Stunden Schäden am Sensor.

Erweiterung des Programms:

Nun soll ein Alarmsignal mit Hilfe eines Piezo-Speakers ertönen, sobald ein Regentropfen auf den Sensor trifft. Als Grenzwert wird in diesem Fall der Messwert 400 festgelegt, da bei einem Tropfen auf dem Sensor ein Messwert von ca. 480 zu erwarten ist. Der Aufbau und die Verkabelung wird mit einem Piezo-Lautsprecher an Pin6 erweitert.

Sketch

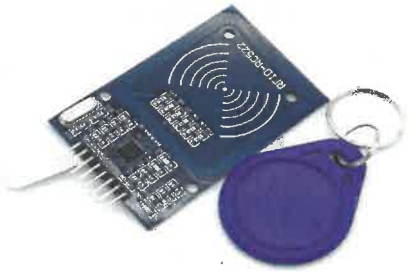
```
int messwert=0;
int PIEPS=6;  //Mit dem Namen „PIEPS“ wird jetzt der Pin6 bezeichnet, an dem ein Piezo-Speaker angeschlossen wird.
void setup()
{
Serial.begin(9600);
pinMode (6,OUTPUT);  //Im Setup wird der Pin6 als Ausgang festgelegt
}
void loop()
```

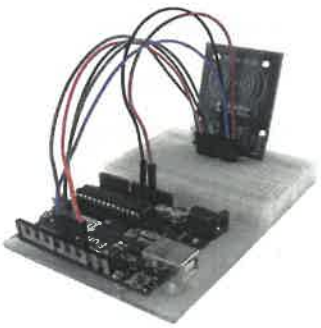
```
{
messwert=analogRead(A0);
Serial.print("Feuchtigkeits-Messwert:");
Serial.println(messwert);
delay(500);
if ( messwert >400)  //Hier beginnt die Abfrage: Wenn der Sensorwert kleiner als „400“ ist,
dann…
{
digitalWrite(PIEPS, HIGH);  //…soll der Piezo-Speaker piepsen
}
else                        //…ansonsten…
{
digitalWrite(PIEPS, LOW);  //…soll er leise sein.
}
}
```

4.20 RFID Chipkarten verwenden

Aufgabe: Mit Hilfe eines Arduino-Mikrocontrollers soll ein RFID-TAG ausgelesen werden. Sofern es sich um den richtigen TAG handelt, soll eine Leuchtdiode leuchten.

Materialbox
1x Arduino-Board
1x RFID-Reader
1x Mindestens einen RFID-TAG
1x LED (wird erst später benötigt)
1x Breadboard
1x 100 Ohm Widerstand
Einige Steckkabel





Auf dem Foto vom Aufbau ist an dem RFID-Empfänger die um 90° geboge Stiftleiste angelötet, damit der Empfänger senkrecht in ein Breadboard gesteckt werden kann.

Der RFID („radio-frequency identification“) Reader wird verwendet, um von RFID Sendern (auch „RFID Tags“ genannt) per Funk einen bestimmten Code auszulesen. Jeder Sender hat dabei nur einen einmaligen ganz individuellen Code. Damit lassen sich mit dem Arduino Schließanlagen oder ähnliche Projekte verwirklichen, bei denen sich eine Person mit einem Sender identifizieren soll.

RFID-TAGs können verschiedene Formen haben, wie z.B. Schlüsselanhänger oder Karten im Kreditkartenformat. Auf dem Bild des RFID-Kits sieht man Links oben und rechts unten zwei RFID-TAGs und in der Mitte den RFID-Empfänger RFID-RC522 mit bereits angelöteter 90° Stiftleiste. Es gibt auch Versionen, bei denen die Stiftleiste

selbst an den RFID-Empfänger gelötet werden muss. Wie funktioniert das Ganze? Ein RFID-Empfänger enthält eine kleine Kupferspule, die ein magnetisches Feld erzeugt. Ein RFID-Sender enthält ebenfalls eine Kupferspule, die das magnetische Feld aufgreift und in dem Sender eine elektrische Spannung erzeugt. Diese Spannung wird dann verwendet, um einen kleinen elektronischen Chip dazu zu bringen, per Funk einen elektrischen Code auszusenden. Dieser Code wird dann direkt vom Sender empfangen und so verarbeitet, dass der Arduino-Mikrocontroller den empfangenen Code weiterverarbeiten kann.

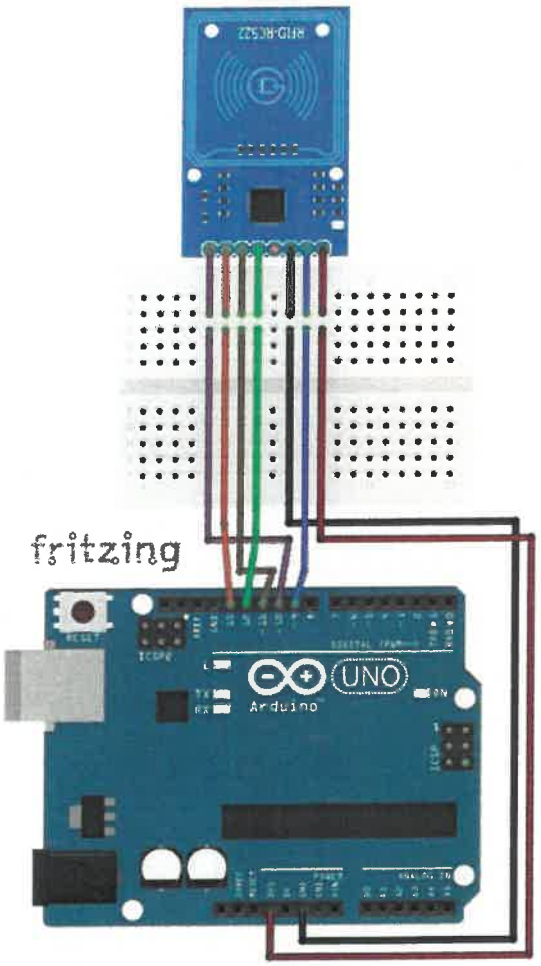
Es ist auch möglich, RFID-TAGs zu mit einem Code zu beschreiben. Dies wird aufgrund der Komplexität in dieser Anleitung nicht behandelt. Weiterführende Tutorials zu dem Themengebiet gibt es im Internet.

Einen RFID-TAG mit Arduino auslesen und die Daten verarbeiten

Aufgabe: Mit Hilfe eines Arduino-Mikrocontrollers soll ein RFID-TAG ausgelesen werden. Sofern es sich um den richtigen TAG handelt, soll eine Leuchtdiode für 5 Sekunden leuchten.

Verkabelung des Arduinoboards mit dem RFD-Empfänger:

Board:	Arduino Uno	Arduino Mega	MFRC522-READER
Pin:	10	53	SDA
Pin:	13	52	SCK
Pin:	11	51	MOSI
Pin:	12	50	MISO
Pin:	unbelegt	unbelegt	IRQ
Pin:	GND	GND	GND
Pin:	9	5	RST
Pin:	3,3V	3,3V	3,3V



Progammierung

Beim Auslesen und verarbeiten der Daten eines RFID-Empfängers wären wie auch bei anderen komplexen Aufgaben sehr viele Zeilen Quellcode erforderlich. Daher bedienen wir uns einer vorgefertigten Library. Die „MFRC522" Library von GithubCommunity kann über den Bibliothekenverwalter mit dem Suchbegriff „RFID" gefunden und installiert werden. Eine Anleitung zur Installation einer Bibliothek über den Bibliotheksverwalter findet sich unter 2.2.2 Bibliotheken zur Arduino Software hinzufügen.

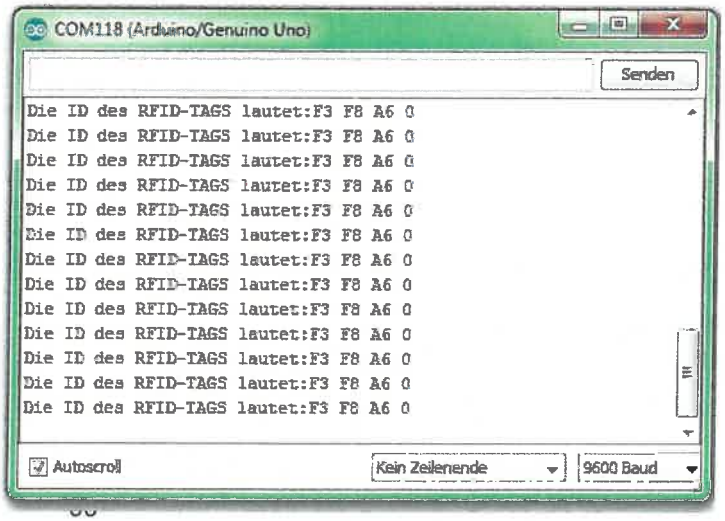
Vorbereitung – Der erste Sketch mit dem RFID-READER

4.20 RFID Chipkarten verwenden

Zunächst werden wir die UID („Unique Identification Number"), also die individuelle Bezeichnung eines RFID-TAGs auslesen. Dazu verwenden wir den folgenden Sketch (Achtung, der Sketch funktioniert nur, wenn die Programmbibliothek wie oben beschrieben zur Arduinosoftware hinzugefügt wurde). Das Programm ist für den UNO R3 Mikrocontroller vorgesehen. Bei MEGA2560 und anderen Controllern müssen die Pins entsprechend der zuvor in dieser Anleitung dargestellten Tabelle angepasst werden

```
#include <SPI.h> // SPI-Bibiothek hinzufügen
#include <MFRC522.h> // RFID-Bibiothek hinzufügen
#define SS_PIN 10 // SDA an Pin 10 (bei MEGA anders)
#define RST_PIN 9 // RST an Pin 9 (bei MEGA anders)
MFRC522 mfrc522(SS_PIN, RST_PIN); // RFID-Empfänger benennen
void setup() // Beginn des Setups:
{
Serial.begin(9600); // Serielle Verbindung starten (Monitor)
SPI.begin(); // SPI-Verbindung aufbauen
mfrc522.PCD_Init(); // Initialisierung des RFID-Empfängers
}
void loop() // Hier beginnt der Loop-Teil
{
if ( ! mfrc522.PICC_IsNewCardPresent()) // Wenn eine Karte in Reichweite ist,...
{
return; //...springt das Programm zurück vor die if-Abfrage, womit sich die Abfrage
wiederholt.
}
if ( ! mfrc522.PICC_ReadCardSerial()) // Wenn ein RFID-Sender ausgewählt wurde,
{
return; //...springt das Programm zurück vor die if-Abfrage, womit sich die Abfrage
wiederholt.
}
Serial.print("Die ID des RFID-TAGS lautet:"); // "Die ID des RFID-TAGS lautet:" wird auf
den Serial Monitor geschrieben
for (byte i = 0; i < mfrc522.uid.size; i++)
{
Serial.print(mfrc522.uid.uidByte[i], HEX); // Dann wird die UID ausgelesen, die aus vier
einzelnen Blöcken besteht und der Reihe nach an den Seriellen Monitor gesendet. Die Endung
Hex bedeutet, dass die vier Blöcke der UID als HEX-Zahl (also auch mit Buchstaben)
ausgegeben wird
Serial.print(" "); // Der Befehl „Serial.print(" ");" sorgt dafür, dass zwischen den
einzelnen ausgelesenen Blöcken ein Leerzeichen steht.
}
Serial.println(); // Mit dieser Zeile wird auf dem Seriellen Monitor nur ein Zeilenumbruch
gemacht.
}
```

Wenn alles funktioniert hat, sieht das Ergebnis am Seriellen Monitor (Abgesehen von der eigenen UID) aus wie in diesem Bild.



Mit dieser hintereinander geschriebenen „HEX-Zahl" lässt sich nicht gut arbeiten. Also ändern wir die Zeile „Serial.print(mfrc522.uid.uidByte[i], **HEX**);" in

„Serial.print(mfrc522.uid.uidByte[i], **DEC**;" um. Dadurch werden die einzelnen Blöcke des UID-Codes als Dezimalzahl ausgegeben.

RFID Sketch 2

Jetzt wird der UID-Code zwar als Dezimalzahl ausgegeben, aber er ist immer noch in vier Blöcke aufgeteilt. Wir verändern den Code jetzt mit ein wenig Mathematik dahingehend, dass wir für die UID eine einzige zusammenhängende normale Zahl erhalten (Dezimalzahl).

Warum machen wir das? Wenn wir später den Sketch verwenden wollen um etwas in Abhängigkeit eines richtig ausgelesenen RFID-TAGs auszulösen (Z.B. eine LED soll leuchten oder eine Servomotor soll sich um 90 Grad drehen...), können wir mit einer zusammenhängenden Zahl besser einen IF-Befehl verwenden. Zum Beispiel: „Wenn der RFID-Code=1031720 ist, dann soll eine LED für 5 Sekunden leuchten".

Schwerer wäre es dagegen mit dem Befehl: „Wenn der erste Block 195 lautet und der zweite Block 765 lautet und der dritte Block 770 lautet und der vierte Block 233 lautet ... Dann schalte eine LED für 5 Sekunden an."

Nachteil ist jedoch, dass der Sketch dadurch etwas unsicherer wird, weil nicht alle Zahlen der vier Blöcke (Max. 12 Ziffern) in einer Zusammenhängenden Zahl dargestellt werden können. Wenn es sicherer sein soll, müsste man jeden einzelnen Block separat abfragen.

```
#include <SPI.h>
#include <MFRC522.h>
#define SS_PIN 10
#define RST_PIN 9
MFRC522 mfrc522(SS_PIN, RST_PIN);
void setup()
{
Serial.begin(9600);
SPI.begin();
mfrc522.PCD_Init();
}
void loop()
{
if ( ! mfrc522.PICC_IsNewCardPresent())
{
return;
}
if ( ! mfrc522.PICC_ReadCardSerial())
{
return;
}
long code=0; // Als neue Variable fügen wir „code" hinzu, unter welcher später die UID als
zusammenhängende Zahl ausgegeben wird. Statt int benutzen wir jetzt den Zahlenbereich
„long", weil sich dann eine größere Zahl speichern lässt.
for (byte i = 0; i < mfrc522.uid.size; i++)
{
code=((code+mfrc522.uid.uidByte[i])*10); // Nun werden wie auch vorher die vier Blöcke
ausgelesen und in jedem Durchlauf wird der Code mit dem Faktor 10 „gestreckt". (Eigentlich
müsste man hier den Wert 1000 verwenden, jedoch würde die Zahl dann zu groß werden.
}
Serial.print("Die Kartennummer lautet::"); //Zum Schluss wird der Zahlencode (man kann ihn
nicht mehr als UID bezeichnen) ausgegeben.
Serial.println(code);
}
```

4.20 RFID Chipkarten verwenden

Jetzt kann von einem RFID-TAG die eindeutige Identifikationsnummer ausgelesen werden. Sie wird am Seriellen Monitor angezeigt. In unserem Fall lautet die Identifikationsnummer dieses individuellen RFID-TAGs „1031720". Und wie geht es weiter? Jetzt wollen wir eine LED für 5 Sekunden einschalten, wenn genau dieser RFID-TAG mit der Nummer „1031720" vor den RFID-Reader gehalten wird.

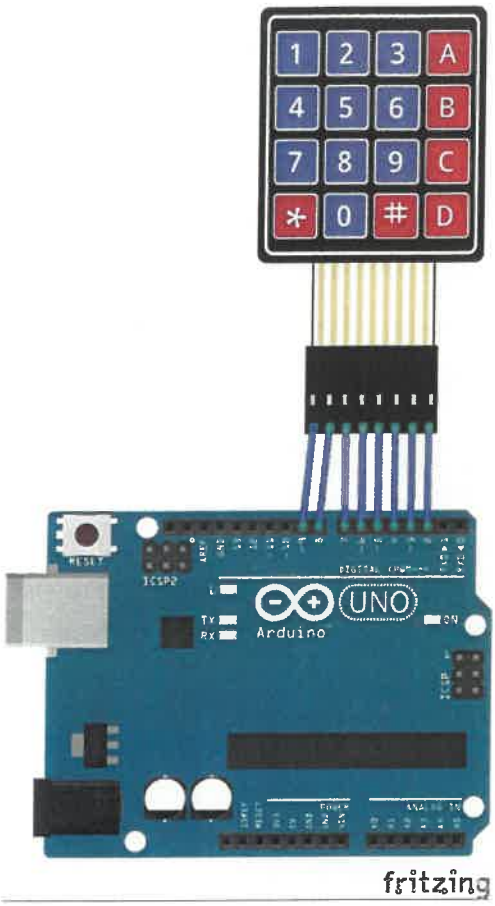
Für diesen Sketch muss der Aufbau um eine LED an Pin2 erweitert werden.

Sketch 3

```
#include <SPI.h>
#include <MFRC522.h>
#define SS_PIN 10
#define RST_PIN 9
MFRC522 mfrc522(SS_PIN, RST_PIN);
void setup()
{
Serial.begin(9600);
SPI.begin();
mfrc522.PCD_Init();
pinMode (2, OUTPUT); // Der Pin 2 ist jetzt ein Ausgang (Hier wird nun zusätzlich eine LED
angeschlossen)
}
void loop()
{
if ( ! mfrc522.PICC_IsNewCardPresent())
{
return;
}
if ( ! mfrc522.PICC_ReadCardSerial())
{
return;
}
long code=0;
for (byte i = 0; i < mfrc522.uid.size; i++)
{
code=((code+mfrc522.uid.uidByte[i])*10);
}
Serial.print("Die Kartennummer lautet:");
Serial.println(code);

// Ab hier erfolgt die Erweiterung des Programms.

if (code==1031720) //Wenn der Zahlencode 1031720 lautet,...
{ //Programmabschniss öffnen
digitalWrite (2, HIGH); //...dann soll die LED an Pin 2 leuchten...
delay (5000); //für 5 Sekunden
digitalWrite (2, LOW); // … und danach wieder aus gehen.
} // Programmabschnitt schließen
} //Sketch abschließen
```

In weiterführenden Aufgaben könnte anstelle der LED nun ein Relais oder ein 12V Türöffner angesteuert werden, um per RFID eine Aktion auszulösen.

4.21.1 Das Tastenfeld

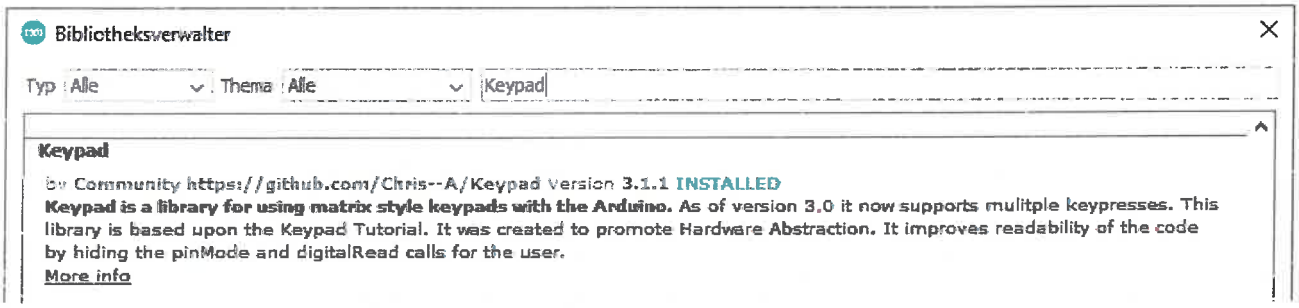


Aufgabe: Mit dem Arduino soll ein Keypad (Tastenfeld) ausgelesen und die gedrückte Taste am Seriellen Monitor angezeigt werden.

Materialbox
1x Arduino-Board
1x Tastenfeld 4x4 oder 3x4
Einige Steckkabel

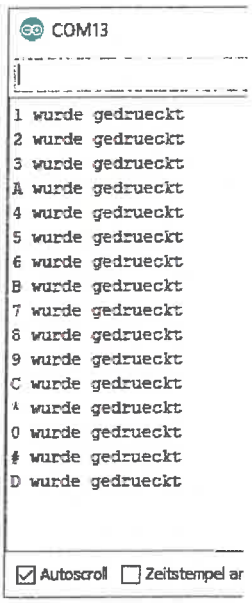
Bei dieser Anleitung wird ein 4×4 Tastenfeld benutzt. Alternativ kann auch ein 3×4 Tastenfeld verwendet werden,hierbei müssen die Pins und die definierten Tasten (COLS/hexaKeys) angepasst werden (s.u.).

Um mit dem Keypad arbeiten zu können, benötigt man eine Library, welche noch nicht im Arduino Programm vorinstalliert ist. Für diese Anleitung verwenden wir die Library „Keypad". Die Library kann über die Bibliothekenverwaltung der Arduino-Software hinzugefügt werden. Als Suchbegriff verwendet man den Namen „Keypad". Eine detaillierte Beschreibung, wie Bibliotheken eingefügt werden, findet man im Theorieteil zu dieser Anleitung im Unterpunkt „Bibliotheken zur Arduinosoftware hinzufügen".



Das Tastenfeld besteht aus einem 4×4 Tasten großem Feld mit 16 Tasten oder einem 3×4 Tasten großem Feld mit 12 Tasten. Um die Funktionsweise zu verstehen, muss man sich das Keypad wie ein Koordinatensystem vorstellen. Jede Taste ist mit zwei Pins verbunden; ein Pin für die Spalten(COLS) und einer für die Zeilen (ROWS). Wird eine Taste gedrückt, werden die entsprechenden Pins miteinander verbunden. Diese Pins werden vom Arduino-Mikrocontroller mit der digitalRead() Funktion auslesen, jedoch findet diese Abfrage durch die Nutzung der Library versteckt statt und muss nicht separat vorgenommen werden. Wie auch bei der Nutzung anderer Libraries, bleibt der Sketch dadurch übersichtlicher.

Auf dem Bild erkennt man die Darstellung der vom Mikrocontroller ausgelesenen Tasten am Seriellen Monitor.



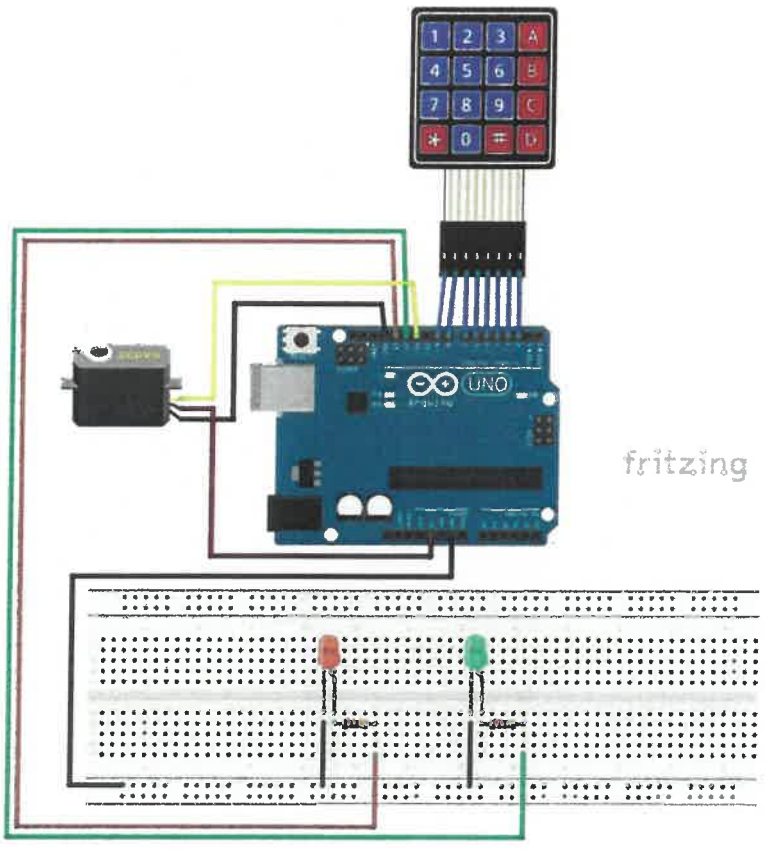
```
#include <Keypad.h> //Hier wird die größe des Keypads definiert
const byte COLS = 4; //4 Spalten
const byte ROWS = 4; //4 Zeilen //Die Ziffern und Zeichen des Keypads werden eingegeben:
char hexaKeys[ROWS][COLS]={
{'D','#','0','*'},
{'C','9','8','7'},
{'B','6','5','4'},
{'A','3','2','1'}
};
byte colPins[COLS] = {2,3,4,5}; //Definition der Pins für die 4 Spalten
byte rowPins[ROWS] = {6,7,8,9}; //Definition der Pins für die 4 Zeilen
char Taste; //Taste ist die Variable für die jeweils gedrückte Taste.
Keypad Tastenfeld = Keypad(makeKeymap(hexaKeys), rowPins, colPins, ROWS, COLS); //Das
Keypad kann absofort mit "Tastenfeld" angesprochen werden.

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

void loop()
{
Taste = Tastenfeld.getKey(); //Unter der Variablen „Taste“ wird die gedrückte Taste
gespeichert.
if (Taste)
{ //Wenn eine Taste gedrückt wurde Serial.print("Die Taste ");
Serial.print(Taste);
Serial.print(" wurde gedrueckt");
Serial.println(); //Teile uns am Serial Monitor die gedrückte Taste mit
}
}
```

Folgende Abschnitte werden bei der Verwendung eines 4×3 Tastenfeldes geändert:

```
const byte COLS = 3;  //3 Spalten
byte colPins[COLS] = { 8, 7, 6 };  //Definition der Pins für die 3 Spalten
char hexaKeys[ROWS][COLS] = {
{'#','0','*'},
{'9','8','7'},
{'6','5','4'},
{'3','2','1'}
};
```

4.21.2 Tastenfeld mit Schließsystem

Materialbox
1x Arduino-Board
1x Breadboard
1x Servo
2x 100 Ohm Widerstand
1x Tastenfeld
1x Grüne und 1x rote LED
1x Tastenfeld 4x4 oder 3x4
Einige Steckkabel



Aufgabe: Mit der Eingabe eines 4-stelligen Codes auf dem Tastenfeld soll eine LED aufleuchten und ein Servo eine bestimmte Position einnehmen.

Am Servo könnte z.B. ein Riegel befestigt werden, der eine Tür ver- und entriegelt.

Wir wollen mit der Eingabe eines 4-stelligen Passworts auf dem Tastenfeld ein grüne LED aufleuchten lassen und einen Servo eine bestimmte Position einnehmen lassen. Wenn das „Schloss" gesperrt ist soll eine rote LED leuchten und der Servo eine andere Position einnehmen. Dieses Tutorial dient als Anregung und Beispiel dazu, wie aus diesen einfachen Bauteilen z.B. ein Tresor gebaut werden kann.

Praxistipp: An den Außenseiten des schwarzen Kontaktsteckers des Tastenfeldes sind die Zahlen 1 und 8 (1 bis 7 beim 3×4 Tastenfeld) zu sehen. Der Kontakt 1 am Tastenfeld wird mit dem Pin 2 am Arduino verbunden. Aufsteigend geht es dann weiter bis zu Kontakt 8 am Tastenfeld welcher mit dem Pin9 am Arduino verbunden wird.

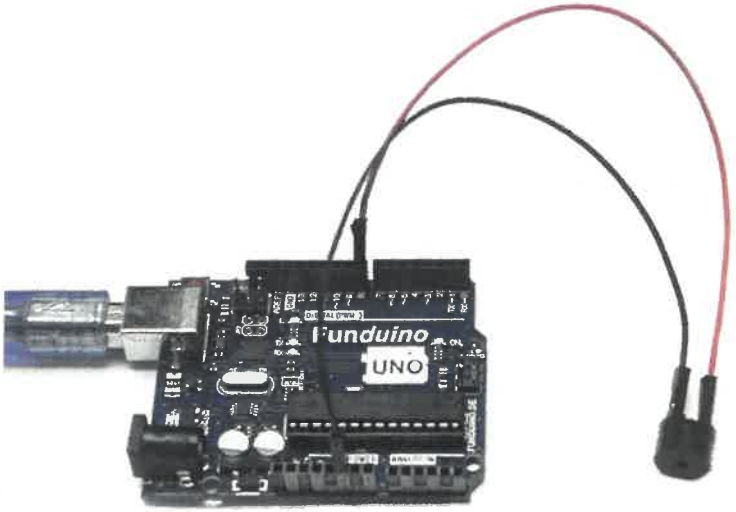
```
#include <Keypad.h>
#include <Servo.h>
char P1='1';char P2='2';char P3='3';char P4='A'; // Hier werden die vier Zeichen des Passwortes
eingegeben Hier: "123A"
char C1, C2, C3, C4; // Unter C1 bis C4 werden im Loop die vier eingegebenen Zeichen gespeichert.
Servo servoblau; //Servo wird ab jetzt mit „servoblau" angesprochen.
int roteLED = 12; //Die rote LED ist an Pin 12 angeschlossen.
int grueneLED = 13; //Die grüne LED wird an Pin 13 angeschlossen.
//Ab hier wird die größe des Keypads definiert.
const byte COLS = 4; //4 Spalten
const byte ROWS = 4; //4 Zeilen
int z1=0, z2, z3, z4; // Diese Variablen werden verwendet, um für die einzelnen Zahlencodes die
Eingabe freizuschalten. Damit wird im Sketch verhindert, dass eine einzene Codeziffer einer falschen
Position zugeordnet wird.
//Die Ziffern und Zeichen des Keypads werden eingegeben.
char hexaKeys[ROWS][COLS]={
{'D','#','0','*'},
{'C','9','8','7'},
{'B','6','5','4'},
{'A','3','2','1'}
};
byte colPins[COLS] = {2,3,4,5}; //Definition der Pins für die 4 Spalten
byte rowPins[ROWS] = {6,7,8,9}; //Definition der Pins für die 4 Zeilen
char Taste; //Taste ist die Variable für die jeweils gedrückte Taste.
Keypad Tastenfeld = Keypad(makeKeymap(hexaKeys), rowPins, colPins, ROWS, COLS); //Das Keypad kann ab
sofort mit "Tastenfeld" angesprochen werden.

void setup()
{
Serial.begin(9600);
pinMode(roteLED, OUTPUT); //Die LEDs werden als Ausgang festgelegt.
pinMode(grueneLED, OUTPUT);
servoblau.attach(11); //Der Servo ist an Pin 11 angeschlossen.
}

void loop()
{
Anfang: // Dies ist eine Markierung, zu der per "goto-"Befehl gesprungen werden kann.
Taste = Tastenfeld.getKey(); //Unter „Taste" wird die gedrückte Taste gespeichert.
if (Taste) //Wenn eine Taste gedrückt wurde...
//Ab hier werden die Eingaben des Tastenfeldes verarbeitet. Zunächst die "*"Taste, da diese eine
Sonderfunktion für die Verriegelung besitzt und danach die #-Taste, nach deren Eingabe der zuvor
eingegebene Code auf Richtigkeit geprüft wird.
 {
  if (Taste=='*') // Wenn die "*" Taste gedrückt wurde,...
   {
   Serial.println("Tuer verriegelt");
   delay(3000);
   servoblau.write(90); //Servo zum verriegeln auf 90 Grad ansteuern.
   digitalWrite(roteLED, HIGH); //..die rote LED einschalten.
   digitalWrite(grueneLED, LOW); //..die grüne LED ausschalten.
   z1=0; z2=1; z3=1; z4=1; // Zugang zur ersten Zeicheneingabe freischalten.
   goto Anfang; //An dieser Stelle springt der Sketch zur Eingabe der Taste zurück, damit das Zeichen
```

```
 "*" nicht im folgenden Abschnitt als Codeeingabe gewertet wird.
    }
   if (Taste=='#') // Wenn die Rautetaste gedrückt wurde,...
    {
    if (C1==P1&&C2==P2&&C3==P3&&C4==P4) //wird geprüft, ob die eingaben Codezeichen (C1 bis C4) mit den
 Zeichen des Passwortes (P1 bis P4) übereinstimmen. Falls der eingegebene Code richtig ist...
    {
    Serial.println ("Code korrekt, Schloss offen");
    servoblau.write(0); //Servo zum öffnen auf 0 Grad ansteuern...
    digitalWrite(roteLED, LOW); //...die rote LED nicht leuchten...
    digitalWrite(grueneLED, HIGH); //...die grüne LED leuchten...
    }
    else // ist das nicht der Fall, bleibt das Schloss gesperrt.
    {
    Serial.println ("Code falsch, Schloss gesperrt");
    digitalWrite(roteLED, HIGH); // Die rote LED leuchtet.
    digitalWrite(grueneLED, LOW); // Die grüne LED ist aus.
    delay(3000);
    z1=0; z2=1; z3=1; z4=1; // Der Zugang für die erste Zeicheneingabe wird wieder freigeschaltet.
    goto Anfang; //An dieser Stelle springt der Sketch zur Eingabe der Taste zurück, damit das Zeichen
"#" nicht im folgenden Abschnitt als Codeeingabe gewertet wird.
    }
   }
// Ab hier werden die vier Code-positionen unter den Variablen C1 bis C4 abgespeichert. Damit die
eingegebenen Zeichen auch an der richtigen Position des Passwortes gespeichert werden, wird mit den
Variablen z1 bis z4 der Zugang zu den einzelnen Positionen freigegeben oder gesperrt.
  if (z1==0) // Wenn das erste Zeichen noch nicht gespeichert wurde,...
  {
  C1=Taste; //Unter der Variablen "C1" wird nun die aktuell gedr. Taste gespeichert.
  Serial.print("Die Taste "); //Teile uns am Seriellen Monitor die gedr. Taste mit.
  Serial.print(C1);
  Serial.println(" wurde gedrueckt");
  z1=1; z2=0; z3=1; z4=1; // Zugang zur zweiten Zeicheneingabe freischalten
  goto Anfang;
  }

  if (z2==0) // Wenn das zweite Zeichen noch nicht gespeichert wurde,...
  {
  C2=Taste; //Unter der Variablen "C2" wird nun die aktuell gedr. Taste gespeichert,
  Serial.print("Die Taste "); //Teile uns am Seriellen Monitor die gedr. Taste mit,
  Serial.print(C2);
  Serial.println(" wurde gedrueckt");
  z1=1; z2=1; z3=0; z4=1; // Zugang zur dritten Zeicheneingabe freischalten
  goto Anfang;
  }

  if (z3==0) // Wenn das dritte Zeichen noch nicht gespeichert wurde,...
  {
  C3=Taste; //Unter der Variablen "C3" wird nun die aktuell gedr. Taste gespeichert
  Serial.print("Die Taste "); //Teile uns am Serial Monitor die gedrückte Taste mit.
  Serial.print(C3);
  Serial.println(" wurde gedrueckt");
  z1=1; z2=1; z3=1; z4=0; // Zugang zur vierten Zeicheneingabe freischalten.
  goto Anfang;
  }

  if (z4==0) // Wenn das vierte Zeichen noch nicht gespeichert wurde,...
  {
  C4=Taste; //Unter der Variablen "C1" wird nun die aktuell gedr. Taste gespeichert.
  Serial.print("Die Taste "); //Teile uns am Serial Monitor die gedrückte Taste mit.
  Serial.print(C4);
  Serial.println(" wurde gedrueckt");
  z1=1; z2=1; z3=1; z4=1; // Zugang zur weiteren Zeicheneingabe sperren.
  }
 }
}
```

4.22 Töne und Musik erzeugen

Aufgabe: Mit einem passiven Lautsprecher sollen unterschiedliche Töne und eine Melodie erzeugt werden.

Materialbox
1x Arduino-Board
1x Breadboard
1x passiver Lautsprecher
Einige Steckkabel



Mit dem Arduino lassen sich auf verschiedene Art und Weise Töne erzeugen. Die einfachste Möglichkeit ist die Tonerzeugung mit einem aktiven Lautsprecher (auch active Buzzer oder Piezo-speaker), der lediglich an die Spannung von 5V angeschlossen wird. Die Tonerzeugung übernimmt eine im Inneren eingebaute Elektronik. Der Nachteil liegt jedoch darin, dass ein „active buzzer" nur einen einzigen Ton erzeugen kann – Melodien oder Sirenengeräusche sind damit nicht möglich.

Mit einem passiven Lautsprecher (passive buzzer) hat man die Möglichkeit, mit Hilfe des Arduino Mikrocontrollers verschiedene Töne, Melodien oder Sirenensignale zu generieren, da im passive Buzzer keine Elektronik vorhanden ist, die einen Ton vorgibt.

Passiv Aktiv

Praxistipp: Die Bauform des aktiven und passiven Lautsprechers können nahezu identisch sein. Man erkennt den hier benötigten aktiven Lautsprecher (auch *„piezo Speaker"* oder *„active Speaker"*) anhand der geschlossenen, schwarzen Unterseite.

Die Erzeugung des Tones basiert maßgeblich auf dem Befehl „tone(x, y)", wobei der x-Wert den Pin angibt, an dem der Lautsprecher mit der positiven Seite angeschlossen ist und der y-Wert der die Tonhöhe angibt.

Ein Beispiel:

```
tone(8, 100); // Der Lautsprecher an Pin 8 wird mit der Tonhöhe „100" aktiviert
delay(1000); // Pause mit 1000 Millisekunden, also 1 Sekunde - Der Lautsprecher bleibt für
diese Zeit aktiv.
noTone(8); // Der Lautsprecher an Pin 8 wird deaktiviert
```

Code1: Einfache Tonerzeugung

Für dieses Beispiel wird lediglich ein Lautsprecher an Pin8 und GND angeschlossen.

```
void setup(){} //Im Setup werden keine Informationen benötigt. Die Spannungsausgabe für den Pin an dem
der Lautsprecher eine Spannungsausgabe benötigt (PinMode OUTPUT), wird im Sketch durch den Befehl
"tone" automatisch festgelegt.
void loop()
{
tone(8, 100);          //Im Hauptteil wird nun mit dem Befehl "tone ( x , y )" ein Ton abgegeben...
delay(1000);           //...mit einer Dauer von 1 Sekunde.
noTone(8);             //Der Ton wird abgeschaltet.
delay(1000);           //Der Lautsprecher bleibt eine Sekunde aus.
}
```

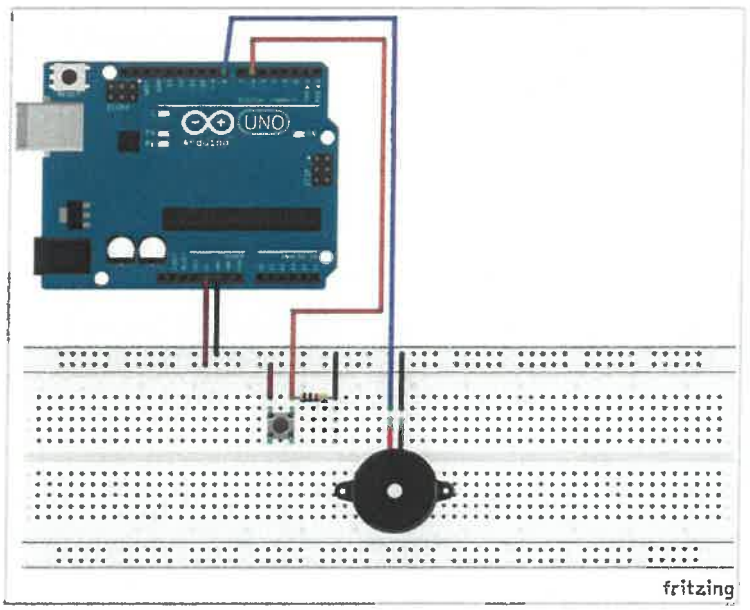
Code2: Abwechselnde Tonhöhen

Im zweiten Beispiel werden im Loop nur ein paar Zeilen ergänzt. Dadurch ertönen zwei Töne abwechselnd mit der Tonhöhe "100" bzw. "200", wie bei einer Sirene. Eine pause zwischen den Tönen gibt es nicht.

```
void setup(){}
void loop()
{
tone(8, 100);
delay(1000);
noTone(8);      //An dieser Stelle geht der erste Ton aus.
Tone(8, 200);   // Der zweite Ton mit der neuen Tonhöhe "200" ertönt.
delay(1000);    //… und zwar für eine Sekunde...
noTone(8);      // An dieser Stelle geht der zweite Ton aus und der Loop-Teil des Sketches beginnt von
vorn.
}
```

Code3: Ton erzeugen durch Tastendruck

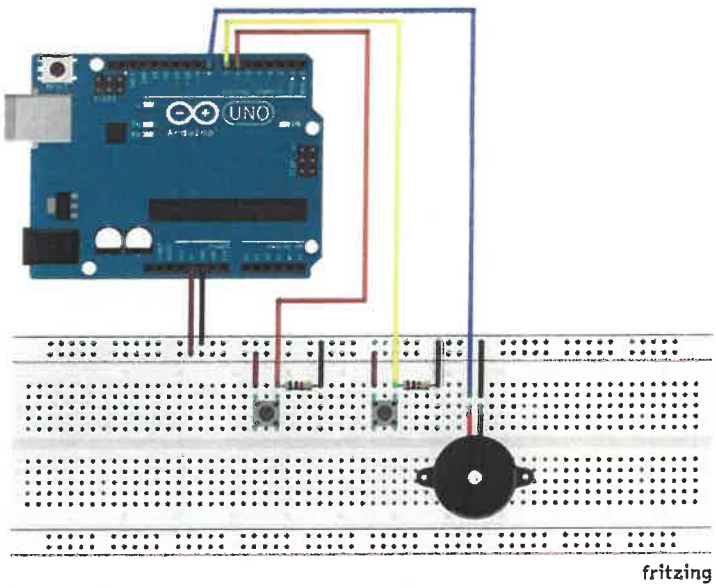
Im dritten Beispiel wird ein Taster an Pin6 zum Aufbau hinzugefügt. Im Hauptteil wird durch eine "IF-Abfrage" der Taster ausgelesen. Falls der Taster gedrückt ist, ertönt für eine Sekunde ein Ton mit der Tonhöhe "300".



```
int Taster=6;
int Tasterstatus=0;
void setup()
{
pinMode(Taster, INPUT);
}
void loop()
{
Tasterstatus=digitalRead(Taster);
if (Tasterstatus == HIGH)
{
tone(8, 300);
delay(1000); //… und zwar für eine Sekunde...
noTone(8);
}
}
```

Code4: Töne in Abhängigkeit von verschiedenen Tasten

Materialbox
1x Arduino-Board
1x Breadboard
1x passiver Lautsprecher
2x Taster
2x 1K Widerstand
Einige Steckkabel



Im vierten Beispiel soll je nach Betätigung unterschiedlicher Tasten entschieden werden, welcher Ton vom Lautsprecher abgegeben wird. Dazu wird jeweils ein Taster an den Pins 6 und 7 angeschlossen. Im Hauptteil wird durch zwei "IF-Abfragen" entschieden, welcher Ton abgegeben wird.

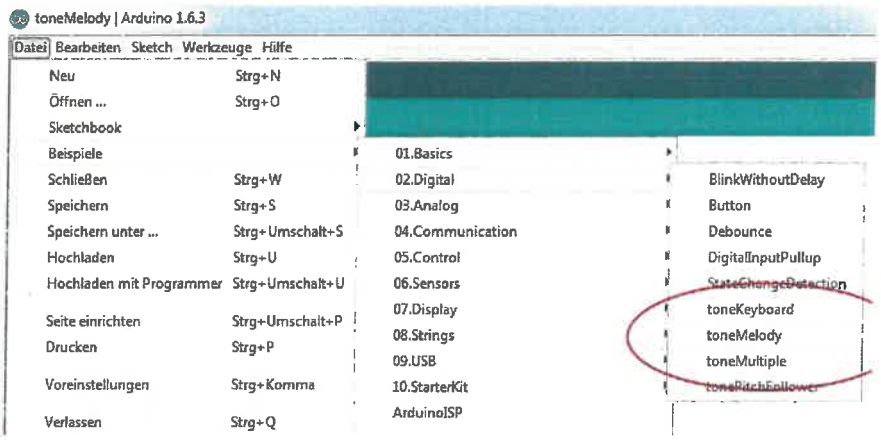
Sketch:

```
int Taster1=6;              // Taster1 an Pin6 angeschlossen
int Taster2=7;              // Taster2 an Pin7 angeschlossen
int Tasterstatus1=0;        // Variable um den Status des Tasters 1 zu speichern.
int Tasterstatus2=0;        // Variable um den Status des Tasters 2 zu speichern.

void setup()
{
pinMode(Taster1, INPUT);    // Taster1 als Eingang festlegen
pinMode(Taster2, INPUT);    // Taster2 als Eingang festlegen
}
void loop()
{
Tasterstatus1 = digitalRead(Taster1);    // Status von Taster1 auslesen(HIGH oder LOW)
```

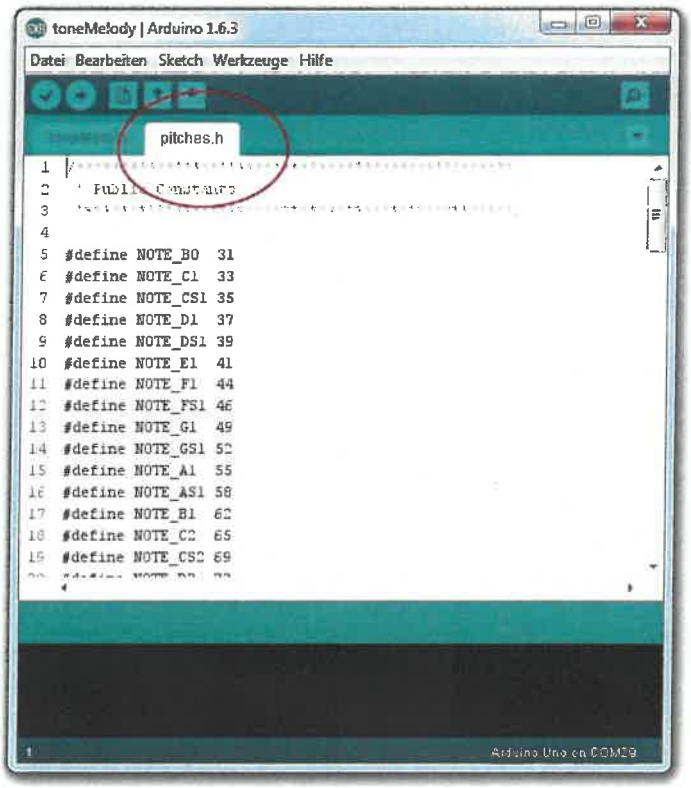
```
Tasterstatus2 = digitalRead(Taster2);   // Status von Taster2 auslesen (HIGH oder LOW)

if (Tasterstatus1 == HIGH) // Wenn der Taster1 gedrückt ist, dann...
{
tone(8, 100);// Ausgabe eines Tons mit der Tonhöhe 100...
delay (1000);// ...mit der Dauer von einer Sekunde.
noTone(8);    // Ausschalten des Tons.
}

if (Tasterstatus2 == HIGH)
{
tone(8, 200);// Ausgabe eines Tons mit der Tonhöhe 200...
delay (1000);// ...mit der Dauer von einer Sekunde.
noTone(8);    // Ausschalten des Tons.
}
}
```

Melodien erzeugen

In der Arduino-Software ist eine Datei speziell zum Generieren von Tönen mit Hilfe eines Lautsprechers (passive Buzzer) enthalten. Anstelle von Tonhöhen in Form von Zahlen wie bspw. "tone(8, 200)", können nun auch Töne aus der Tonleiter ausgewählt werden.



Als Beispiel gibt es dafür in der Arduino-Software den Sketch "toneMelody". Zu finden unter "Beispiele" -> "02.Digital" -> "toneMelody". In dem Beispiel ist bereits die Datei hinterlegt, in der sich die Zuordnung von Tonhöhe und Zahlenwert befindet. Sie hat den Namen "pitches.h". Den Inhalt kann man sich ansehen, wenn man in dem geöffneten Beispielprogramm auf den zweiten Kartenreiter innerhalb des Sketches klickt, wie im folgenden Bild dargestellt.



Um auf die Datei zurückgreifen zu können, muss im Sketch lediglich der Befehl #include „pitches.h“ eingebaut werden und die Datei selber muss als Tab geöffnet sein. Am besten geht das, indem man den Sketch “toneMelody” aus den Beispielen in der Arduino-Software öffnet und dann bearbeitet. Dadurch wird die Datei “pitches.h” automatisch als Tab aufgerufen. Das wird im folgenden Beispiel deutlich.

Dies ist ein ganz kleiner Sketch, der Abwechselnd zwei Töne aus der Datei “pitches.h” abspielt.

```
#include "pitches.h"
void setup()
{
pinMode (8,OUTPUT);  // Lautsprecher an Pin8
}
void loop()
{
tone(8, NOTE_C4, 1000);  // An Pin8 wird die Note C4 für 1000ms gespielt
delay(3000);  //Nachdem die Note ertönt, pausiert der Sketch für 3 Sekunden. Das bedeutet,
dass nachdem der Ton zu Ende gespielt wurde, noch zwei Sekunden Pause ohne Ton verbleiben.
tone(8, NOTE_G3, 1000);  // An Pin8 wird die Note G3 für 1000ms gespielt
delay(3000);  //Nachdem die Note ertönt, pausiert der Sketch für 3 Sekunden. Das bedeutet,
dass nachdem der Ton zu Ende gespielt wurde, noch zwei Sekunden Pause ohne Ton verbleiben.
}
```

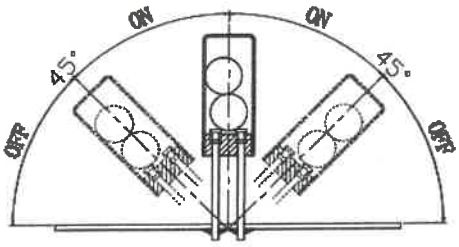
In diesem Sketch wird durch den erweiterten “tone-” Befehl die Beendigung des Tones nicht mehr benötigt.

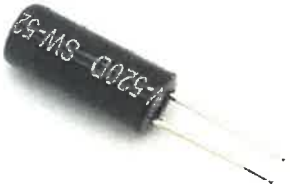
Befehl: “tone(x, y, z)” x= Pin des Lautsprechers, y= Tonhöhe, z= Dauer des Tons in Millisekunden.

4.23 Neigungssensor verwenden

Aufgabe: Mit einem Neigungssensor soll eine LED eingeschaltet werden, sobald eine Neigung von mehr als 45° vorliegt

Materialbox
1x Arduino-Board
1x Breadboard
1x Eine rote LED
1x 100 Ohm Widerstand
1x 1K Ohm Widerstand
1x Neigungssensor SW-520D
Einige Steckkabel



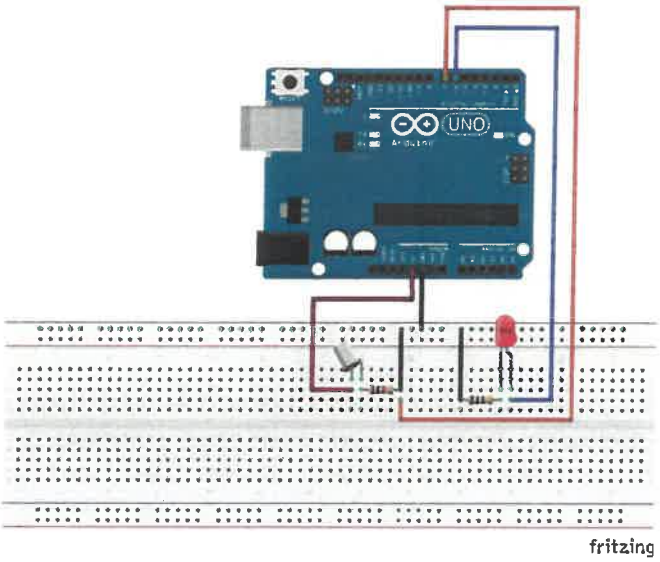


Mit dem Arduino lassen sich auf verschiedene Art und Weise Neigungen messen. Sehr exakte Messungen bekommt man zum Beispiel mit einem Beschleunigungssensor, bei dem man nahezu jeden Winkel genau messen kann. Es geht jedoch auch ganz einfach mit dem Neigungssensor (engl. Tilt sensor oder Tiltswitch). Dieser Sensor erkennt jedoch nur zwei Stufen, nämlich “geneigt” oder “nicht geneigt”. Dazu rollen im Inneren

des Sensors zwei frei bewegliche Metallkugeln umher. Sobald der Sensor so gekippt ist, dass eine der Kugeln im Inneren gegen die zwei Kontakte stößt, werden diese beiden Kontakte miteinander verbunden. Das ist dann so, als wäre ein Taster gedrückt und der elektrische Strom kann von einem Kontakt zum anderen Kontakt fließen. Daher ist der Sketch und auch der Aufbau sehr ähnlich, wie bei der Verwendung eines einfachen Tasters.
Auf der Skizze erkennt man die Funktionsweise des Sensors mit den beiden Kugeln.

Wenn der Sensor wie auf diesem Bild senkrecht verwendet wird, beispielsweise wenn er einfach im Breadboard eingesteckt wird, ist der Stromkreis zwischen den beiden Kontakten geschlossen und der Strom kann von einem Kontakt zum anderen fließen. Sobald der Sensor jedoch um mehr als 45° geneigt wird, rollen die Kugeln von den beiden Kontakten weg und die Verbindung ist unterbrochen.

Aufbauskizze:



Der Sketch:

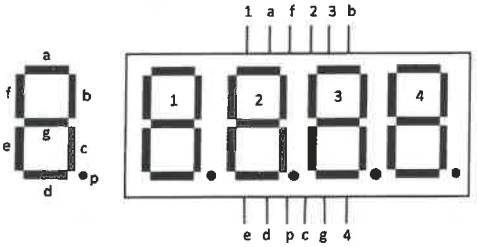
```
int LED=6;          //Das Wort „LED“ steht jetzt für den Wert 6.
int TILT=7;         //Das Wort „TILT“ steht jetzt für den Wert 7 - Der Neigungssensor wird
also an Pin7 angeschlossen.
int NEIGUNG=0;      //Das Wort „NEIGUNG“ steht jetzt zunächst für den Wert 0. Später wird
unter dieser Variablen gespeichert, ob eine Neigung von mehr als 45 Grad vorliegt oder
nicht.

void setup() //Hier beginnt das Setup.
{
pinMode(LED, OUTPUT); //Der Pin mit der LED (Pin 6) ist ein Ausgang.
pinMode(TILT, INPUT); //Der Pin mit dem Neigungssensor (Pin 7) ist jetzt ein Eingang.
}

void loop()
{//Mit dieser Klammer wird der Loop-Teil geöffnet.
NEIGUNG=digitalRead(TILT); //Hier wird der Pin7, also der Neigungssensor, ausgelesen
(Befehl:digitalRead). Das Ergebnis wird unter der Variablen „NEIGUNG“ mit dem Wert „HIGH“
für 5Volt oder „LOW“ für 0 Volt gespeichert. Wenn der Sensor gerade steht, kann der Strom
zwischen beiden Kontakten des Sensors fließen. Es liegt dann am Pin7 eine Spannung an. Der
Status wäre dann also “HIGH”. Wenn der Sensor 45 oder mehr Grad geneigt ist, kann kein
Strom fließen und der Status wäre dann “LOW”.

if (NEIGUNG == HIGH) //Verarbeitung: Wenn der Sensor weniger als 45 Grad geneigt ist...
{                     //Programmabschnitt des IF-Befehls öffnen.
digitalWrite(LED, LOW);   //...dann soll die LED nicht leuchten
}                     //Abschnitt des IF-Befehls schließen.
else                  //...ansonsten...
{                     //Abschnitt des else-Befehls öffnen.
digitalWrite(LED, HIGH);  //...soll die LED leuchten.
}     //Programmabschnitt des else-Befehls schließen.
}     //Mit dieser letzten Klammer wird der Loop-Teil geschlossen.
```

4.24 Vierstellige 7 Segment Anzeige

Aufgabe: Wir wollen eine Zahl auf einer vierstelligen 7-Segment-Anzeige darstellen.

Materialbox
1x Arduino-Board
1x Breadboard
4x 1K Ohm Widerstand
1x Vierstellige 7 Segment Anzeige
Einige Steckkabel

Die 7-Segment-Anzeige besteht aus vielen kleinen Segmenten, die jeweils mit einer kleinen LED im Inneren ausgeleuchtet werden. Die LEDs in dem Modul können entweder von der Sorte „Common Cathode" oder „Common Anode" sein. Der Unterschied besteht darin, dass die einzelnen LEDs im Modul entweder mit einer positiven Spannung oder mit GND angesteuert werden. Im Sketch wird dieser Teil durch eine Einstellung der Library erledigt.

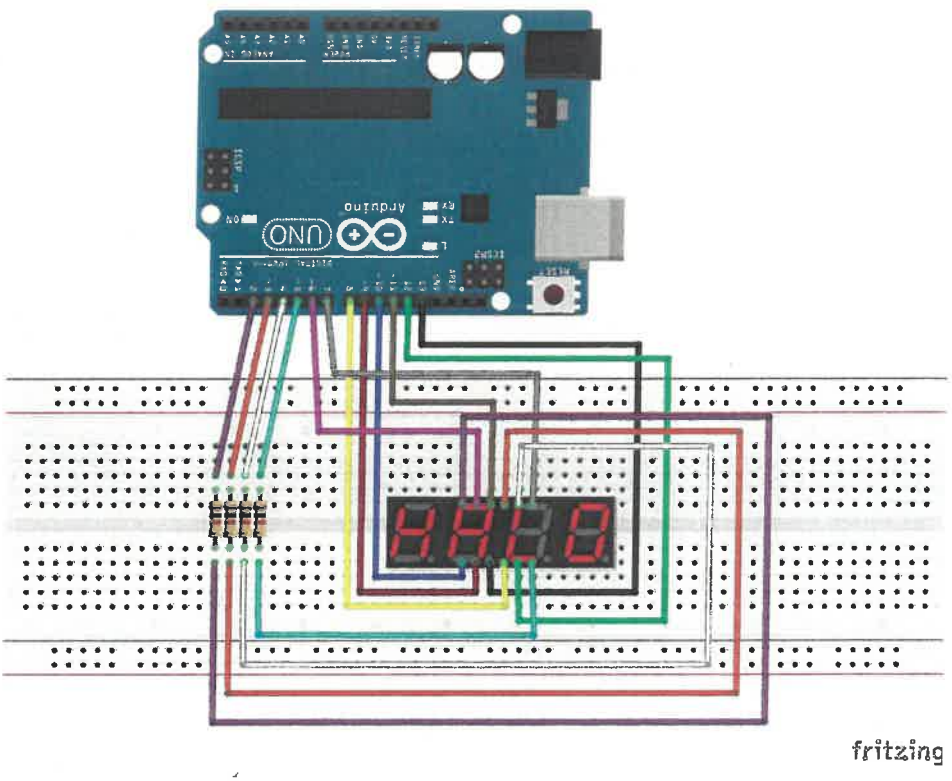


Inklusive des leuchtenden Punktes besteht jede Ziffer aus acht einzeln anzusteuernden Segmenten (LEDs). Bei 7-Segment-Modulen mit nur einer oder zwei Ziffern werden die Segmente jeder Ziffer einzeln angesteuert. Das vorliegende Modul fasst vier dieser Ziffern zusammen. Daher müsste es eigentlich mit 32 einzelnen Kabeln angesteuert werden. Da das bei vier Ziffern aber ein noch größeres Kabeldurcheinander wäre, als es bereits ist, funktionieren diese Displays mit „Multiplexing". Dabei werden die einzelnen Ziffern vom Mikrocontroller hintereinander angesteuert. Dies geschieht so schnell, dass es für das menschliche Auge aussieht, als würden alle vier Ziffern gleichzeitig angezeigt werden.

Programmierung

Um eine 7-Segment-Anzeige ohne endlos langen Code zu programmieren, benötigt man eine Library, die noch nicht in der Arduino Software enthalten ist. In unserem Beispiel verwenden wir die „SevenSeg" Library von „Dean Reading". Die Library muss, wie schon aus vorherigen Anleitungen bekannt, zur Arduino Software hinzugefügt werden. Dies geht am leichtesten in der Bibliothekenverwaltung der Arduino Software. Unter dem **Suchbegriff „sevseg"** lässt sich die passende Bibliothek schnell finden.

Praxistipp: Da die 7-Segment-Anzeige sehr schnell angesteuert wird, darf in einem Sketch mit diesem Modul kein Delay verwendet werden. Dies würde dazu führen, dass auf dem Modul keine Zahlen zu erkennen sind. Man muss also ggf. erforderliche Delays durch mathematische Funktionen ersetzen. Alternativ gibt es 7-Segment-Module bspw. mit einer I²C Ansteuerung. Diese Module speichern einen gesendeten Zahlenwert in einer separaten Speichereinheit, bis eine neue Zahl empfangen wird. Daher sind solche I²C Module für größere Projekte wesentlich besser geeignet, jedoch auch wesentlich teurer in der Anschaffung.



Sketch

```
#include "SevSeg.h"          //Die vorher hinzugefügte Library laden.
SevSeg sevseg;               //Ein sieben Segment Objekt initialisieren.
void setup()
{
byte numDigits = 4;      //Hier wird die Anzahl der Ziffern angegeben.
byte digitPins[] = {2, 3, 4, 5};//Die Pins zu den Ziffern werden festgelegt.
byte segmentPins[] = {6, 7, 8, 9, 10, 11, 12, 13};
//Die Pins zu den Segmenten werden festgelegt.
sevseg.begin(COMMON_CATHODE, numDigits, digitPins, segmentPins);
//In diesem Abschnitt kann man nun entweder testen, welche Art von Display man besitzt oder
wenn man es schon weiß, angeben, ob es sich um ein COMMON_CATHODE oder COMMON_ANODE Display
handelt. Das Display funktioniert nur, wenn die richtige Art eingetragen ist, ansonsten
werden alle Segmente gleichzeitig leuchten.
}
void loop() {
sevseg.setNumber(1234,3);  //Hier können wir nun die gewünschte Zahl eintragen. Wir haben
als Beispiel 1234 angegeben. Die Zahl hinter dem Komma steht für den Punkt hinter einer
Ziffer. Hierbei ist 3 der Punkt neben der ersten Ziffer und 0 wäre der Punkt ganz rechts
neben der letzten Ziffer. Wenn man keinen Punkt mit angezeigt haben möchte, kann man z.B. 4
angeben.
```

```
sevseg.refreshDisplay();   // Dieser Teil lässt die Nummer auf dem Display erscheinen.
sevseg.setBrightness(90); //Hier kann die Helligkeit des Displays angepasst werden. In
einem Bereich von 0-100 wobei 100 das Hellste ist. 0 bedeutet jedoch nicht, dass das
Display komplett dunkel ist. Für die Anzeige einer Zahl ist allein die
"sevseg.refreshDisplay();" Zeile verantwortlich
}
```

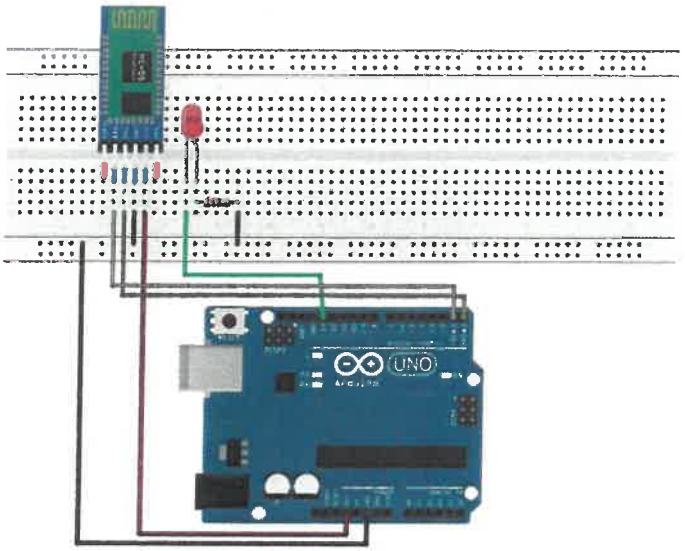
Praxistipp: In der „sevseg library" sind weitere interessante Beispielcodes vorhanden.

Diese können in der Arduino Software unter: Datei > Beispiele > SevSeg-master

aufgerufen werden.

4.25 Bluetooth Modul HC-05 & HC-06

Materialbox
1x Arduino-Board
1x Breadboard
1x 200 Ohm Widerstand
1x Bluetooth Modul HC-05 oder HC-06
1x rote LED
einige Steckkabel



Mit einem Bluetoothmodul kann eine drahtlose Kommunikation zwischen einem Arduino Mikrocontroller und einem Smartphone oder Laptop hergestellt werden. Es können Daten zum Mikrocontroller gesendet, aber auch vom Mikrocontroller empfangen werden. Für die Arduino Entwicklungsumgebung gibt es diverse Bluetoothmodule. Sehr verbreitet sind die Module „HC-05" und „HC-06". Der Unterschied zwischen den beiden Modulen ist für diese Anleitung unrelevant. Daher kann diese Anleitung mit beiden Modulen verwendet werden.

Info: Aufgrund der wesentlich stärkeren Verbreitung von Android-Betriebssystemen bei Smartphones, bezieht sich diese Anleitung nur auf die Nutzung des Bluetooth-Moduls mit diesem System. Für andere Systeme, wie bspw. „Apple" werden ggf. andere Module für die Bluetoothverbindung benötigt.

Aufgabe: Mit Hilfe eines Bluetooth Moduls, soll eine LED mit einem Android-Smartphone an- und ausgeschaltet werden.

Verkabelung

Die Verkabelung wird wie in der Skizze aufgebaut und ergibt sich aus folgendem Schema:

HC-05/HC-06	+5V	GND	TX	RX
UNO R3	3,3 V	GND	RX	TX

Hinweis: Am Signaleingangsport „RX" des Bluetooth-Moduls sollte nach Vorgabe des Chipherstellers nur eine Spannung von 3,3V statt 5V angeschlossen werden. Diese Spannung lässt sich über einen Spannungsteiler realisieren, der bspw. aus einem 1K Ohm und einem 2K Ohm Widerstand aufgebaut wird. In vielen Stunden der Nutzung ohne diese Spannungsteilerschaltung haben wir keine Nachteile feststellen können. Daher verwenden wir das Modul in dieser Anleitung zur Vereinfachung ohne einen Spannungsteiler.
Zusätzlich zum Bluetoothmodul kann eine LED an PIN13 des Mikrocontrollerboards angeschlossen werden, um das Ergebnis deutlicher erkennen zu können.

Programmierung:

Der grundlegende Datentransfer zwischen Smartphone und Arduino-Mikrocontroller erfolgt über die Serielle Schnittstelle. Die App auf dem Handy sendet die eingegebenen Ziffern oder Buchstaben an das Bluetoothmodul und dieses stellt die Daten an der Seriellen Schnittstelle des Mikrocontrollers bereit. In Abhängigkeit der empfangenen Elemente führt der Sketch dann weitere Aktionen aus.

Achtung!!! Beim Hochladen auf den Mikrocontroller **muss das Bluetooth Modul herausgenommen werden.** Sonst erscheint die Fehlermeldung, dass der Code nicht hochgeladen werden kann. Nach dem Hochladen wird das Modul wieder eingesteckt.

Sketch

```
char Daten; //"Char" ist ein Datentyp zum Speichern eines ZeichenwertsWerte. In diesem Fall die Zeichen, die per Bluetooth empfangen werden.
char LEDstatus; //Speichert den letzten Status der LED (on/off).

void setup()
{
Serial.begin(9600);  //Serieller Monitor wird gestartet, Baudrate auf 9600 festgelegt.
pinMode(13,OUTPUT);  //PIN 13 wird als Ausgang festgelegt.
}

void loop()
{
 if(Serial.available()) //wenn Daten verfügbar sind,...
 {
 Daten=Serial.read(); //..sollen diese ausgelesen werden. Die Daten werden unter der Variablen "Daten" abgespeichert.
 }

if (Daten=='1') //Wenn das Bluetooth Modul eine „1“ empfängt,...
 {
 digitalWrite(13,HIGH); //...soll die LED leuchten.
 }

if (Daten=='0') //Wenn das Bluetooth Modul „0“ empfängt,...
 {
 digitalWrite(13,LOW); //..soll die LED nicht leuchten.
 }
}
```

Nach dem Hochladen des Sketches wird das Bluetooth Modul wieder eingesteckt. Die blinkende LED am Modul zeigt an, dass die Verbindung zwischen Smartphone und Bluetooth-Modul hergestellt werden muss.

Vorbereitung des Smartphone

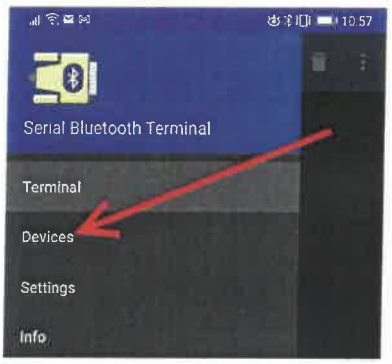
Das Bluetoothmodul muss in den Bluetooth-Einstellungen des Smartphones zunächst mit dem Smartphone „gekoppelt“ werden. Für den Verbindungsaufbau ist die Eingabe einer PIN-Nummer erforderlich. Diese PIN lautet standardmäßig 1234. Erst danach wird das Modul auch in den Verbindungseinstellungen der App angezeigt. Das Bluetooth Modul wird unter dem Namen HC-05 oder HC-06 gefunden.

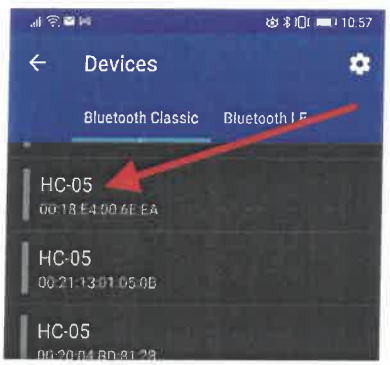
App auf dem Smartphone

Für die Verwendung eines Bluetooth-Moduls mit dem Smartphone gibt es viele verschiedene Apps. Wir verwenden für diese Anleitung die APP „Serial Bluetooth Terminal“, die im „Play-Store“ kostenfrei heruntergeladen werden kann.

Die Entwickler der Apps ändern gelegentlich das Layout und den Funktionsumfang. Im Falle einer Änderung muss ggf. intuitiv agiert werden.

Zu Beginn wird in der App die Verbindung zum Bluetoothmodul hergestellt. Dazu klickt man im App-Menü auf „Devices“ und wählt dort das Bluetoothmodul aus (z.B. HC-05).



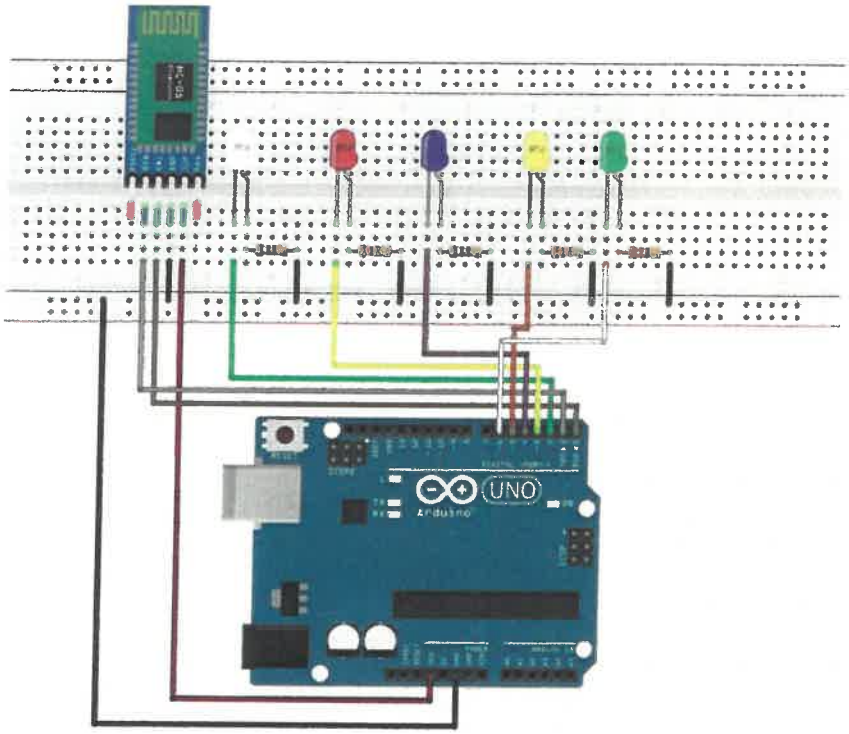


Nachdem das Smartphone mit dem Bluetoothmodul gekoppelt ist, können die ersten Daten versendet werden. In diesem Fall senden wir die Ziffern „1“ und „0“. Der Sketch ist so geschrieben, dass er auf diese beiden Zeichen reagiert. Sobald am Arduino-Microcontroller über die Serielle Schnittstelle eine „1“ empfangen wird, wird die LED an Pin13 aktiviert. Beim Empfang einer „0“ wird die LED wieder deaktiviert.

Nach diesem Prinzip lassen sich viele weitere Anwendungen programmieren. Man könnte zum Beispiel eine Ampel oder ein Auto per Bluetooth steuern, oder auch Musik machen. Es muss im Sketch lediglich festgelegt werden, auf welches durch die App versendete Zeichen der Arduino-Mikrocontroller reagiert.

Sprachsteuerung mit dem Bluetooth Modul

Materialbox
1x Arduino-Board
1x Breadboard
2x 200 Ohm Widerstand
3x 100 Ohm Widerstand
1x Bluetooth Modul HC-05
(Alternativ HC-06)
1x rote LED
1x weiße LED
1x blaue LED
1x grüne LED
einige Steckkabel



fritzing

Um die vielen Möglichkeiten des Bluetoothmoduls aufzuzeigen, erweitern wir diese Anleitung beispielhaft durch das Ein-und Ausschalten mehrerer LED per Sprachbefehl.

Verkabelung: Die Verkabelung unterscheidet sich nicht maßgeblich von der Verkabelung im Beispiel zuvor. Es werden jedoch vier weitere LEDs hinzugefügt. Um Verwechselungen zu vermeiden, sollten die LEDs wie in diesem Beispiel aufgebaut werden.

Praxistipp: Beim Hochladen auf den Mikrocontroller muss das Bluetooth Modul herausgenommen werden. Sonst erscheint die Fehlermeldung, dass der Code nicht hochgeladen werden kann. Nach dem Hochladen wird das Modul wieder eingesetzt.

Sketch

```
String voice;
int
ledweiss = 2,        //Die weiße LED mit Pin2 verbinden.
ledrot = 3,          //Die rote LED mit Pin3 verbinden.
ledblau = 4,         //usw….
ledgelb = 5,
ledgruen = 6;

void allon(){  //Die Funktion für den Befehl „alle an".
  digitalWrite(ledweiss, HIGH);
  digitalWrite(ledrot, HIGH);
  digitalWrite(ledblau, HIGH);
  digitalWrite(ledgelb, HIGH);
  digitalWrite(ledgruen, HIGH);
}
void alloff(){      //Die Funktion für den Befehl „alle aus".
  digitalWrite(ledweiss, LOW);
  digitalWrite(ledrot, LOW);
  digitalWrite(ledblau, LOW);
  digitalWrite(ledgelb, LOW);
  digitalWrite(ledgruen, LOW);
}
```

```
void setup() {
  Serial.begin(9600);
  pinMode(ledweiss, OUTPUT);      //Die Pins mit den LEDs werden als Ausgänge festgelegt.
  pinMode(ledrot, OUTPUT);
  pinMode(ledblau, OUTPUT);
  pinMode(ledgelb, OUTPUT);
  pinMode(ledgruen, OUTPUT);
}
void loop() {
  while (Serial.available()){ //überprüfen, ob lesbare Werte vorhanden sind.
  delay(10);
  char c = Serial.read();
  if (c == '#') {break;}  //"#" zeigt das Ende eines Befehls an, deshalb soll der Loop
verlassen werden wenn ein ''#''vorkommt.
  voice += c;
  }
  if (voice.length() > 0) {
     Serial.println(voice);

//In diesem Abschnitt werden die Befehle den einzelnen Funktionen zugeordnet.
 if(voice == "*alle an") {allon();} //Alle Pins sollen angeschaltet werden (vorher
festgelegte Funktion „allon")
 else if(voice == "*alle aus"){alloff();} //Alle Pins sollen ausgeschaltet werden (vorher
festgelegte Funktion „alloff").

//In diesem Abschnitt wird das Einschalten der einzelnen LEDs festgelegt
else if(voice == "*weiß an") {digitalWrite(ledweiss, HIGH);}
else if(voice == "*rot an") {digitalWrite(ledrot, HIGH);}
else if(voice == "*blau an") {digitalWrite(ledblau, HIGH);}
else if(voice == "*gelb an") {digitalWrite(ledgelb, HIGH);}
else if(voice == "*grün an") {digitalWrite(ledgruen, HIGH);}

//In diesem Abschnitt wird das Ausschalten der einzelnen LEDs festgelegt
else if(voice == "*weiß aus") {digitalWrite(ledweiss, LOW);}
else if(voice == "*rot aus") {digitalWrite(ledrot, LOW);}
else if(voice == "*blau aus") {digitalWrite(ledblau, LOW);}
else if(voice == "*gelb aus") {digitalWrite(ledgelb, LOW);}
else if(voice == "*grün aus") {digitalWrite(ledgruen, LOW);}
voice="";
}
}
```

Für die Funktion der Sprachsteuerung gibt es wieder eine spezielle App. Wir haben die kostenlose Android App „BT Voice Control für Arduino" von SimpleLabsIN verwendet.

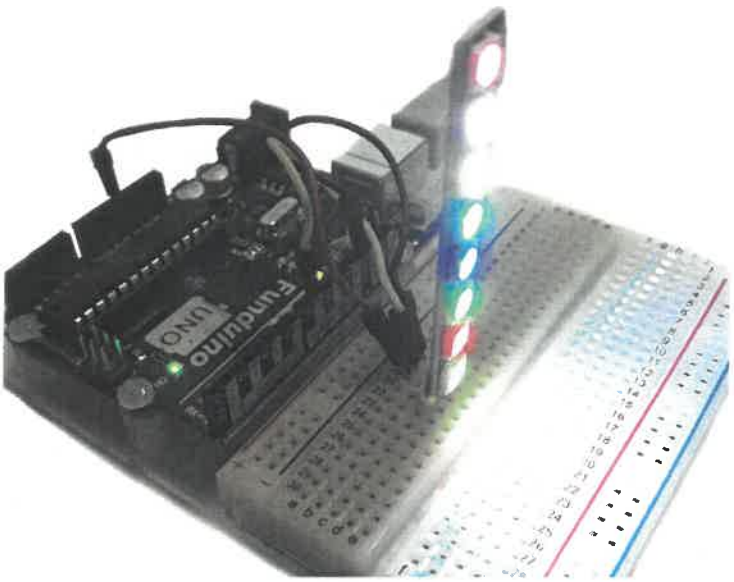
In der App muss zunächst eine Verbindung zum Bluetoothmodul hergestellt werden. In dieser App geschieht das oben rechts im Menü „Connect Robot".

Jetzt können auch schon die Befehle durch Sprache eingegeben werden.

Achten Sie dabei auf eine deutliche Aussprache. Natürlich können die Befehle unterschiedlich variiert und auch für andere spannende Projekte angepasst werden.

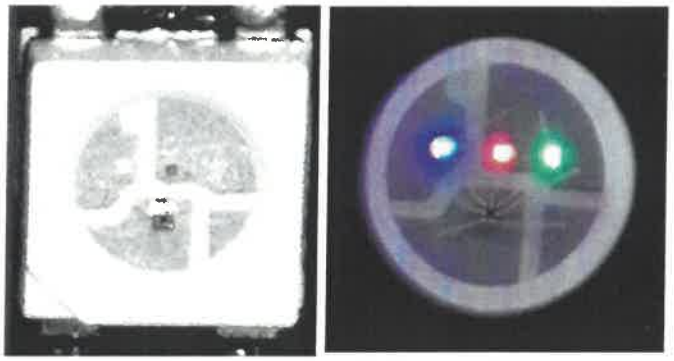
4.26 WS2812 LED programmieren

Materialbox
1x Arduino-Board
1x Breadboard
1x WS2812 Streifen
einige Steckkabel



Aufgabe: Programmiere Regenbogenfarben mit WS2812 LEDs.

Hinter dem sperrigen Namen WS2812 LED verbirgt sich eine sehr interessante und intelligente Leuchtdiode mit integriertem Schaltkreis („IC"). Diese neuere Form der LED ist je nach Hersteller auch unter anderen Namen wie „NeoPixel" bekannt und wird mittlerweile in sehr vielen Anwendungen wie LED-Streifen und Displays verwendet. Bei Lichtdesignern, Spieleentwicklern, Raumausstattern oder auch Möbeldesignern werden die LEDs eingesetzt, um farbenfrohe Lichtspektakel in allen Regenbogenfarben zu erzeugen. Das besondere daran ist, dass die einzelnen LEDs miteinander vernetzt sind. Jede einzelne LED kann beispielsweise in einem LED-Streifen individuell angesteuert werden, obwohl insgesamt nur drei Leitungen benötigt werden. Die Anzahl der hintereinander gekoppelten LEDs kann dabei frei variiert werden. Jede LED kann in 256 Helligkeitsstufen leuchten und dabei über 16 Millionen Farben darstellen.



Auf dem Foto sieht man eine stark vergrößerte WS2812 LED, einmal ohne Spannungsversorgung und einmal mit den schwach aktivierten Farben Blau, Rot und Grün. Der winzige dunkle Fleck in der Mitte ist der integrierte Schaltkreis (IC).

Mit einem Mikrocontroller, in diesem Falle natürlich aus der Arduino-Reihe, werden über die Datenleitung alle LEDs angesteuert. Damit der Sketch überschaubar bleibt, verwenden wir auch in dieser Anleitung eine Programmbibliothek. Eine gute Library zu den WS2812 LEDs gibt es von Adafruit. Dort werden die LEDS als „NeoPixel" bezeichnet. Außerdem bietet Adafruit auch eine sehr umfangreiche Informationsseite und einige Sketches rund um das Thema Neopixel an. Damit die Sketche in dieser Anleitung funktionieren, muss vorab in der Arduino-Software die Library „Adafruit NeoPixel" installiert werden. Dies geht am leichtesten in der

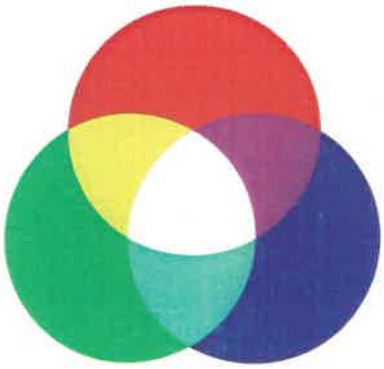
Bibliothekenverwaltung der Arduino-Software. Unter dem Suchbegriff „Adafruit NeoPixel" lässt sich die passende Bibliothek schnell finden.

In unserer Anleitung zu diesem Thema starten wir nicht sofort mit farbenfrohen Regenbogeneffekten, sondern mit der Ansteuerung von einzelnen Pixeln in unterschiedlichen Helligkeitsstufen. Regenbogeneffekte werden später mit sogenannten Programmschleifen erzeugt, die wiederum auf andere Schleifen für die Farbänderung zurückgreifen. Um diese verschiedenen Inhalte zu entzerren, beginnen wir möglichst unkompliziert.

In unserem Beispiel verwenden wir einen Streifen mit WS2812 LEDs. Diese Anleitung lässt sich jedoch auch mit jeglichen anderen WS2812 bzw. NeoPixel Modulen durchführen. Wichtig ist dabei nur, dass im jeweiligen Sketch die Gesamtanzahl der LEDs des jeweiligen Moduls angegeben wird (Zeile: „#define NUMPIXELS 8„). Die Zahl (hier 8) steht für die Anzahl der vorhandenen LEDs.

Der wichtigste Befehl in den folgenden Sketches lautet „**pixels.setPixelColor(x , pixels.Color(0,255,0));**" . Der Befehl beinhaltet in der Klammer zwei Informationen. Das erste ist eine Zahl (hier als „x" bezeichnet) und gibt die Nummer der LED an, die leuchten soll. Hier muss beachtet werden, dass die erste LED die Nummer 0 ist. Wenn man zum Beispiel einen WS2812 LED Streifen mit 8 LEDs verwendet, haben die einzelnen LEDs die Zahlen 0 bis 7.

Die zweite Information „pixels.Color(0,255,0)" beinhaltet gleichzeitig die Farbe und Helligkeit, in der die LED leuchten soll. Dazu gibt es in der Klammer drei Zahlen, jeweils durch ein Komma getrennt. Die erste Zahl steht für die Farbe „**Rot**", die Zweite für die Farbe „**Grün**" und die Dritte für die Farbe „**Blau**". Für die Zahlen können Werte zwischen 0 und 255 gewählt werden. Je kleiner die Zahl ist, desto dunkler leuchtet die entsprechende Farbe. Bei dem Wert 0 bleibt die Farbe komplett dunkel, der Wert 255 lässt die LED in maximaler Stärke leuchten. Wenn zwei oder drei Farben gleichzeitig aktiviert werden, ergibt sich eine Farbmischung. In dem Schaubild zur additiven Farbmischung sieht man, welche Farben bei einer Farbmischung entstehen.



Beispiele für Farben:

pixels.setPixelColor(x,pixels.Color(255,0,0))=Rot

pixels.setPixelColor(x,pixels.Color(0,255,0))=Grün

pixels.setPixelColor(x,pixels.Color(0,0,255))=Blau

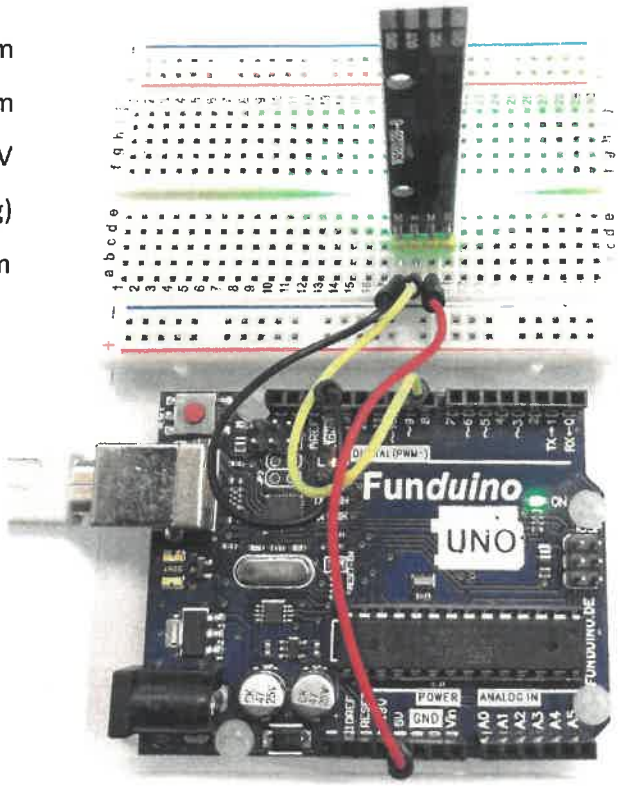
pixels.setPixelColor(x,pixels.Color(255,255,0))=Gelb (eine additive Mischung aus rotem und grünem Licht)

pixels.setPixelColor(x,pixels.Color(0,255,255))=Türkis (eine additive Mischung aus grünem und blauem Licht)

pixels.setPixelColor(x,pixels.Color(255,255,255))=Weiß (eine additive Mischung der Farben Rot, Grün und Blau)

Aufbau und Verkabelung

WS2812-Module werden mit lediglich drei Kabeln am Mikrocontroller angeschlossen. *GND* und *VCC* sind am WS2812-Modul die Kontakte für die Spannung (VCC=5V und GND=GND) und am Kontakt *DIN* (Digitaler Eingang) wird die Datenleitung angeschlossen. In unserem Beispiel ist das der Pin9.

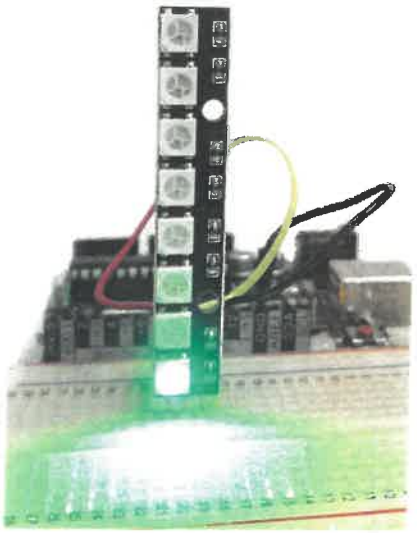


Programmierung

Sketch Nr.1: Eine einzelne LED grün leuchten lassen

```
#include <Adafruit_NeoPixel.h>
#ifdef __AVR__
#include <avr/power.h>
#endif
#define PIN 9 // Hier wird angegeben, an welchem digitalen Pin die WS2812 LEDs bzw.
NeoPixel angeschlossen sind
#define NUMPIXELS 8 // Hier wird die Anzahl der angeschlossenen WS2812 LEDs bzw. NeoPixel
angegeben.
Adafruit_NeoPixel pixels = Adafruit_NeoPixel(NUMPIXELS, PIN, NEO_GRB + NEO_KHZ800);

void setup()
{
pixels.begin(); // Initialisierung der WS2812LEDs
}

void loop()
{
pixels.setPixelColor(0, pixels.Color(0,255,0)); // Die erste LED (Nr.0) leuchtet in der
Farbe Grün.
pixels.show(); // Durchführen der LED-Ansteuerung
delay (1000); // Eine Sekunde Pause. In dieser Zeit wird nichts verändert.

// Zurücksetzen der WS2812 LED auf Stufe "0" (Die LED wird abgeschaltet).
pixels.setPixelColor(0, pixels.Color(0,0,0));
pixels.show(); // Durchführen der LED-Ansteuerung.
delay (1000); // Eine Sekunde Pause. Die LED bleibt während dieser Zeit aus.
}
```

Das Ergebnis sollte so aussehen:



Schaffst du es, eine andere LED in einer anderen Farbe leuchten zu lassen? Wenn nicht, dann versuche es im Sketch mit dieser Codezeile:

```
pixels.setPixelColor(5, pixels.Color(255,0,0));
```

Sketch Nr.2: Mit dem folgenden Sketch werden wir alle acht LEDs des WS2812 Streifens in unterschiedlichen Farben nacheinander aktivieren.

```
#include <Adafruit_NeoPixel.h>
#ifdef __AVR__
#include <avr/power.h>
#endif
#define PIN 9 // Hier wird angegeben, an welchem digitalen Pin die WS2812 LEDs bzw. NeoPixel angeschlossen sind.
#define NUMPIXELS 8 // Hier wird die Anzahl der angeschlossenen WS2812 LEDs bzw. NeoPixel angegeben
Adafruit_NeoPixel pixels = Adafruit_NeoPixel(NUMPIXELS, PIN, NEO_GRB + NEO_KHZ800);
int pause=200; // 200 Millisekunden Pause bis zur Ansteuerung der nächsten LED.

void setup()
{
pixels.begin(); // Initialisierung der WS2812LEDs
}

void loop()
{
pixels.setPixelColor(0, pixels.Color(20,100,20)); // LED1 leuchtet in der Farbe Hellgün
pixels.show(); // Durchführen der LED-Ansteuerung
delay (pause); // Pause, in dieser Zeit wird nichts verändert
pixels.setPixelColor(1, pixels.Color(255,0,0)); // LED2 leuchtet in der Farbe Rot
pixels.show(); // Durchführen der LED-Ansteuerung
delay (pause); // Pause, in dieser Zeit wird nichts verändert
pixels.setPixelColor(2, pixels.Color(0,255,0)); // LED3 leuchtet in der Farbe Grün
pixels.show(); // Durchführen der LED-Ansteuerung
delay (pause); // Pause, in dieser Zeit wird nichts verändert.
pixels.setPixelColor(3, pixels.Color(0,0,255)); // LED4 leuchtet in der Farbe Blau
pixels.show(); // Durchführen der LED-Ansteuerung
delay (pause); // Pause, in dieser Zeit wird nichts verändert.
pixels.setPixelColor(4, pixels.Color(0,255,255)); // LED5 leuchtet in der Farbe Türkis
pixels.show(); // Durchführen der LED-Ansteuerung
delay (pause); // Pause, in dieser Zeit wird nichts verändert.
pixels.setPixelColor(5, pixels.Color(255,255,0)); // LED6 leuchtet in der Farbe Gelb
pixels.show(); // Durchführen der LED-Ansteuerung
delay (pause); // Pause, in dieser Zeit wird nichts verändert.
pixels.setPixelColor(6, pixels.Color(255,255,255)); // LED7 leuchtet in der Farbe Weiß
pixels.show(); // Durchführen der LED-Ansteuerung
pixels.setPixelColor(7, pixels.Color(100,20,20)); // LED8 leuchtet in der Farbe Rosa
pixels.show(); // Durchführen der LED-Ansteuerung
delay (4000); // Am Ende setzen wir eine etwas längere Pause (4 Sekunden). In dieser Zeit leuchten alle acht LEDs.
```

```
// Zurücksetzen aller LED-farben auf Stufe "0" (Alle LEDs werden abgeschaltet)
pixels.setPixelColor(1, pixels.Color(0,0,0));
pixels.setPixelColor(2, pixels.Color(0,0,0));
pixels.setPixelColor(3, pixels.Color(0,0,0));
pixels.setPixelColor(4, pixels.Color(0,0,0));
pixels.setPixelColor(5, pixels.Color(0,0,0));
pixels.setPixelColor(6, pixels.Color(0,0,0));
pixels.show(); // Durchführen der LED-Ansteuerung
delay (pause); // Pause, die LEDs bleiben in dieser Zeit aus.
}
```

Das Abschalten aller LEDs kann alternativ über den folgenden Befehl erfolgen:

pixels.clear(); // Befehl zum Deaktivieren aller LEDs.

pixels.show(); // Senden der aktualisierten Daten an die WS2812 LED.

Das Ergebnis sollte so aussehen wie auf diesem Bild.

Erweiterungsaufgabe

Schaffst du es, mit dem LED-Streifen eine Ampel zu programmieren?

5. Code Referenz

Arduino Programmiersprache – Code reference

Die Arduino Programmiersprache basiert auf den Programmiersprachen „C" und „C++" und verfügt über eine Vielzahl von Befehlen. Um mit den vielen Codes effektiv arbeiten zu können, sollte man sie idealerweise alle kennen, genauso wie man beim Erlernen einer Fremdsprache möglichst viele Wörter kennenlernen muss, um bei der Kommunikation nicht unnötig viele Umwege machen zu müssen. Natürlich ist dieses Erlernen schwer und langwierig. Es werden geeignete Quellen benötigt, die beim Erlernen der Programmiersprache als Hilfestellung dienen.

Englischsprachige Dokumentation

Die jeweils aktuellsten Befehle und Informationen zu Programmcodes findet man auf der englischsprachigen Internetseite „www.arduino.cc" im Bereich „Reference" **https://www.arduino.cc/reference/en/**

Deutsche Dokumentation

Im deutschsprachigen Raum hat es bezüglich einer **deutschen Übersetzung** zwei nennenswerte Entwicklungen gegeben. Zum einen hat eine deutsche Foren-Community an einer sehr ausführlichen Übersetzung samt Beispielen und zusätzlichen Erklärungen gearbeitet. Zeitgleich wurde von den Gründern von Arduino über die Internetseite www.arduino.cc in Zusammenarbeit mit GitHub.com eine kollaborative Möglichkeit geschaffen, sich an der Übersetzung der Dokumentation der Programmiersprache in sehr vielen Sprachen zu beteiligen. Im Jahr 2019 wurde eine nahezu vollständige deutsche Dokumentation fertiggestellt und ist seitdem über den folgenden Link erreichbar:

https://www.arduino.cc/reference/de

Eine Dokumentation aller Programmcodes mit Erklärungen benötigt nach einem aktuellen Formatierungstest durch den Autor **mehr als 200 DinA4 Seiten** und ist damit für ein praxisbezogenes Lernheft viel zu umfangreich. Daher folgt an dieser Stelle eine Kurzübersicht über Programmelemente, die für den Beginn mit der Arbeit mit Arduino wichtig sein können. Eine solche Übersicht ist insbesondere im Bildungsbereich für den ersten Umgang mit Arduino üblich und wird daher von Schülern länderübergreifend auch „Cheatsheet", also **Spickzettel** genannt. In Klassenräumen findet man solche „Cheatsheets" häufig in Form von Postern an den Pinnwänden der Fachräume.

Arduino Programmiersprache – Spickzettel

Die Arduino-Programmiersprache kann in drei Hauptteile unterteilt werden: Funktionen, Werte (Variablen und Konstanten) und Strukturen.

5.1 Funktionen

Zur Steuerung des Arduinoboards und zur Durchführung von Berechnungen.

Mathematik
map(); Wandelt einen Zahlenbereich in einen anderen Zahlenbereich um
max(x,y); Gibt den höheren Wert zurück
min(x,y); Gibt den kleineren Wert zurück

Analoge Ein- und Ausgänge I/O
analogWrite(pin, var);
Gibt einen analogen Wert am Pin aus, wobei „var" einen ganzzahligen Wert zwischen 0 und 255 annehmen kann.
int var = analogRead(pin);
Liest den Wert vom analogen Pin. Eine Spannung zwischen 0 und 5V wird als Zahlenwert zwischen 0 und 255 abgespeichert. Beispiel: 2,5 Volt entspricht dem Zahlenwert 128.

Erweiterte I/O Funktionen
tone(pin, freq);
Generiert eine Rechteckwelle (bzw. mit angeschlossenem Lautsprecher einen Ton) der vorgegebenen Frequenz. Dafür muss ein Pin mit mit PWM-Funktion verwendet werden.
tone(pin, freq, zeit);
Generiert eine Rechteckwelle (bzw. mit angeschlossenem Lautsprecher einen Ton) der vorgegebenen Frequenz für eine vorgegebene Zeit in Millisekunden. Dafür muss ein Pin mit mit PWM-Funktion verwendet werden.
noTone(pin);
Der Befehl beendet die Abgabe einer Frequenz.

Digitale Ein- und Ausgänge
pinMode(pin, [INPUT \ OUTPUT \ INPUT_PULLUP]);
Setzt den Modus der digitalen Ein- und Ausgänge. Sie können als Eingang, als Ausgang, oder als Eingang mit internem Pull-Up Widerstand gesetzt werden.
digitalWrite(Pin, HIGH or LOW);
Pin= Nummer des digitalen Pins. HIGH bzw. LOW schaltet den Pin auf 5V (HIGH) bzw. auf 0V (LOW)
int var = digitalRead(Pin);
Liest den Status des entsprechenden Pins und speichert HIGH oder LOW.

Zeit
delay(Zeit);
Pausiert den Sketch für die angegebenen Zeit in Millisekunden
delayMicroseconds(Zeit);
Pausiert den Sketch für die angegebenen Zeit in Mikrosekunden
millis();
Speichert die Zeit in Millisekunden ab dem Beginn des aktuellen Programms
micros();
Speichert die Zeit in Mikrosekunden ab dem Beginn des aktuellen Programms

5.2 Variablen

Datentypen und Konstanten

Datentypen

boolean	0, 1, false (unwahr), true (wahr)
char	8 Bits: ASCII Zeichen
byte	8 Bits: 0 bis 255, ohne Vorzeichen
int	16 Bits: -32,768 bis 32,767 mit Vorzeichen
long	32 Bits: 2.147.483.648 bis 2.147.483.647 mit Vorzeichen
float	32 Bits, mit Vorzeichen und Komma (Dezimalstellen)

Konstanten
HIGH \ LOW
INPUT \ OUTPUT
true \ false
LED_BUILTIN Viele Arduino-Boards haben einen Pin, der mit einer integrierten LED verbunden ist. Die Konstante ist die Nummer des Pins, an den die LED angeschlossen ist. Bei den meisten Boards (bspw. UNO und MEGA) ist diese LED an den digitalen Pin 13 angeschlossen.

5.3 Struktur

Elemente des Arduinocodes

Vergleichsoperatoren
== Gleich
!= Nicht gleich
< Kleiner als
> Größer als
<= Kleiner oder gleich
>= größer oder gleich

Boolesche Operatoren
&& „und"
|| „oder"
! „nicht"

Kommentare
// Kommentare zum Sketch werden hinter die beiden Schrägstriche geschrieben. Sie haben keinen Einfluss auf den Sketch und werden nicht auf das Board geladen.
/*Schrägstrich und Sternchen verwendet man für umfangreiche Kommentare, die über die Länge einer Zeile hinaus gehen*/

Struktur
Jeder Arduino-Sketch muss die beiden folgenden Funktionen besitzen.

```
void setup()
{
Dieser Code läuft nur ein einziges Mal zu Beginn der Sketch-Ausführung
}

void loop()
{
Dieser Code wiederholt sich immer wieder, solange das Arduinoboard mit Spannung versorgt wird.
}
```

Rechenzeichen
= Zuordnung
+ Addition
- Subtraktion
* Multiplikation
/ Division
% Modulo (ist eine mathematische Funktion, die den Rest aus einer Division zweier ganzer Zahlen benennt)

Kontrollstrukturen

```
if(Bedingung)
{
Wenn die Bedingung wahr ist, mache dies
}
else
{
ansonsten mache dies
}

for(Initialisierung; Bedingung;Zuwachs)
{
mache dies
}
```

Die „for" Anweisung wird verwendet, um einen Anweisungsblock mit einer festgelegten Häufigkeit zu wiederholen.
Beispiel: Der folgende Programmblock wird zehnmal wiederholt. Der Zählvorgang startet bei 0. Die Bedingung lautet, dass die Wiederholung so lange stattfindet, wie x kleiner als 10 ist. Nach jeder Ausführung wird x durch das „++" um eine Zahl höher. Alternativ kann auch „--" verwendet werden, um einen „negativen" Zuwachs zu erreichen.

```
for (x=0; x<10; x++) {Programmblock}
```

goto name; Der Befehl „goto" verursacht einen Sprung im Sketch zur Stelle mit dem entsprechenden Namen

```
name:
```

5.4 Programmbibliotheken (Libraries)

#include <libraryname.h>

Das Einbinden von Bibliotheken ermöglicht den Zugriff auf spezielle Zusatzfunktionen und vereinfacht den Aufbau des Sketches. Anwendungsmöglichkeiten sind zum Beispiel Servomotoren, Schrittmotoren, Sensoren oder Module für Funkverbindungen wie bspw. Bluetooth. In der Arduinosoftware ist eine Reihe von Bibliotheken vorab enthalten. Man kann jedoch auch eigene Bibliotheken herunterladen oder erstellen. Die Informationen darüber, welche Bibliothek benötigt wird, gibt es in der Regel zu den verwendeten Bauteilen dazu oder lassen sich in den entsprechenden Anleitungen zur Verwendung der Bauteile finden.
Zusammen mit den heruntergeladenen Bibliotheken stehen wieder neue Programmcodes zur Verfügung, die sich ganz konkret auf die heruntergeladene Bibliothek beziehen. Um einen Einblick in die jeweils neuen Möglichkeiten zu bekommen, lohnt sich immer ein Blick in den Bibliotheks-Ordner. In den meisten Fällen sind dort Beispielsketche vorhanden, in denen die neuen Funktionen verwendet und beschrieben werden.

Und jetzt?

Wer dieses Heft von Beginn an bis zu dieser letzten Seite durchgearbeitet hat, verfügt über eine gute Grundlage, kreativ mit Arduino-Mikrocontrollern und den zahlreichen Sensoren und Aktoren zu arbeiten.
Dies ist der Startpunkt für spannende Projekte mit der Arduino-Entwicklungsumgebung.

„MACH DOCH EINFACH!“ *(Motto der Ideenexpo Hannover)*

6. Skizzen und Notizen