ARDUINO PROJEKTBUCH

ARDUINO – DAS PROJEKTBUCH

AUTOREN
Die Projekte und die Texte sind von Scott Fitzgerald und Michael Shiloh, Mirco Piccin. Mit einem Beitrag von Tom Igoe, Arturo Guadalupi

DESIGN UND ART DIRECTION
TODO
Giorgio Olivero, Vanessa Poli, Michelle Nebiolo, Stefania Vulpi
Todo.to.it

DIGITAL HERSTELLUNG UND PROJECT MANAGEMENT
Officine Arduino Torino, Katia De Coi, Enzico Bassi, Stefano Paradiso, Andrea Richetta

ADVISOR UND SUPPORTER
Massimo Banzi

TEST DER PROJEKTEN UND REVISION DER TEXTE
Michael Shiloh, Michelle Nebiolo, Katia De Coi, Alessandro Buat, Federico Vanzati, David Mellis, Arturo Guadalupi, Luisa Castiglioni

ÜBERSETZUNG UND LAYOUT DER ITALIENISCHEN VERSION
Luisa Castiglioni

DANKESAGUNG
Großer Dank geht an die ganze Community für die fortwährenden Beiträge, ihre Unterstützung und Feedback. Ein besonderer Dank geht an Fritzing's Team: Einige Illustrationen mit elektronische Komponenten, die im Buch verwendet wurden, wurden übernommen oder vom Open-Source-Fritzing-Projekt (www.fritzing.org) geändert.
Ein herzliches Dankeschön geht an Paul Badger für den Library CapacitiveSensor, das im Projekt 13 verwendet wurde.

DANKE FÜR DIE ERSTE AUSGABE
Mario Ciazdulli, Gianluca Martino

INHALTSVERZEICHNIS

00

Jeder von uns verwendet täglich Technologien. Die meisten von uns überlassen jedoch die Programmierung und Elektronik den Ingenieuren, weil wir denken das sei zu kompliziert und mühselig. Tatsächlich können dies aber spaßige und aufregende Aktivitäten sein. Dank Arduino lernen Designer, Künstler, Bastler und Studenten jeden Alters, Dinge zu erschaffen, die aufleuchten, sich bewegen und mit Menschen, Tieren, Pflanzen und dem Rest der Welt interagieren.

Im Verlauf der Jahre wurde Arduino als „Gehirn" in tausenden Projekten eingesetzt, eines kreativer als das andere. Eine weltweite Gemeinschaft von schöpferischen Menschen hat sich um diese Open-Source-Plattform versammelt, um sich von Personal Computing zu Personal Fabrication weiter zu entwickeln und in eine neue Welt der Beteiligung, Zusammenarbeit und des Austauschs einzutauchen.

Arduino ist offen und einfach. Es basiert auf Erkenntnissen, die wir beim Unterrichten unserer Klassen gewonnen haben, wenn Du davon ausgehst, dass das Erlernen von digitalen Technologien einfach und zugänglich ist, dann wirst Du es auch so empfinden. Plötzlich werden aus Elektronik und Programmcode kreative Werkzeuge, die jeder benutzen kann – wie Pinsel und Farbe. Dieses Buch führt Dich spielerisch in die Grundlagen ein, indem Du beim Lernen kreative Projekte zum Leben erweckst. Sobald Du die Grundlagen gemeistert hast, steht Dir eine ganze Palette von Software und Schaltkreisen zur Verfügung, um etwas Bewundernswertes zu erschaffen und anderen mit Deinen Erfindungen ein Lächeln auf die Lippen zu zaubern.

WILLKOMMEN BEI ARDUINO!

ARDUINO MACHT ES SO EINFACH WIE MÖGLICH, WINZIGE COMPUTER, SOGENANNTE MIKROCONTROLLER, ZU PROGRAMMIEREN, UM OBJEKTE INTERAKTIV ZU MACHEN.

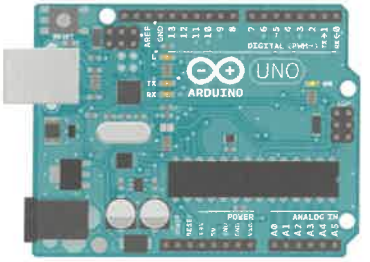
Du bist täglich von Dutzenden von ihnen umgeben: sie sind eingebettet in Weckern, Thermostaten, Spielzeugen, Fernbedienungen, Mikrowellen, und sogar in einigen Zahnbürsten. Sie erledigen nur eine ganz bestimmte Aufgabe, und dass Du sie kaum beachtest – was meistens der Fall ist – liegt daran, dass sie ihre Aufgabe gut machen. Sie wurden programmiert mit Sensoren und Aktoren Aktivitäten zu erfassen und zu steuern.

Sensoren nehmen die physikalische Welt wahr. Sie wandeln die Energie, die Du abgibst, wenn Du eine Taste drückst, mit Deinen Armen winkst oder schreist, in elektrische Signale um. Tasten und Drehknöpfe sind solche Sensoren, die Du mit Deinen Fingern berühren kannst, aber es gibt noch viele weitere Arten von Sensoren. Aktoren agieren in der physischen Welt. Sie wandeln elektrische Energie wieder in physische Energie, wie Licht, Wärme und Bewegung, um.

Mikrocontroller hören auf Sensoren und sprechen mit Aktoren. Sie entscheiden basierend auf dem von Dir geschriebenen Programm, was zu tun ist.
Mikrocontroller und die Elektronik, die Du an sie anschließt, bilden allerdings nur das Grundgerüst für Deine Projekte. Um dem Ganzen Leben einzuhauchen, werden Deine eigenen Fähigkeiten gefordert.
Zum Beispiel schlagen wir in einem Projekt vor, dass Du einen Pfeil bastelst, ihn an einem Motor befestigst und zusammen mit einem Drehknopf in eine Schachtel einbaust, um so anderen Leuten anzeigen zu können, wie beschäftigt Du gerade bist. In einem anderen Projekt setzt Du Lichter und einen Kippschalter auf einen Papprahmen zur Erschaffung einer Sanduhr.
Mit Arduino werden Deine Projekte zwar reaktionsfähig, doch es liegt in Deiner Hand sie zu verschönern. Auf dem Weg dorthin geben wir Dir ein paar Ratschläge, wie Du das bewerkstelligen könntest.

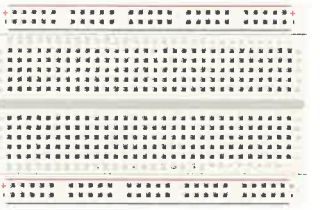
Arduino wurde konzipiert, um Dir die Umsetzung Deiner Projekte so weit wie möglich zu erleichtern. Um das zu erreichen, haben wir zusätzliche Informationen und Materialien zu Programmierung und Elektronik auf ein Minimum beschränkt. Wenn Du mehr über gewisse Aspekte wissen möchtest, gibt es viele hervorragende Anleitungen dafür. Wir stellen einige solcher online unter:
arduino.cc/starterkit zur Verfügung.

TEILE IN DEINEM KIT

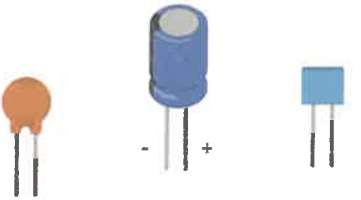


Arduino Uno - Das Mikroprozessor-Entwicklungsboard ist das Herzstück Deiner Projekte. Es ist ein einfacher Computer, der momentan noch nicht mit Dir interagieren kann. Dazu musst Du erst die Schaltkreise und Schnittstellen einrichten und dem Mikrocontroller erklären, wie er die weiteren Komponenten ansprechen soll.

Batterieclip - Wird verwendet, um eine 9-V-Batterie mit Stromkabeln zu verbinden, um diese einfach an eine Steckplatine oder Deinen Arduino anzuschließen.



Steckplatine - Eine Platine auf der Du elektronische Schaltkreise aufbauen kannst. Es ist wie eine Stecktafel mit Reihen von Buchsen, mithilfe derer Du Leitungen und Bauteile verbinden kannst. Es gibt auch Varianten, bei denen man löten muss, hier ist das jedoch nicht erforderlich.



Kondensatoren - Diese Bauteile speichern Energie oder geben sie in den Schaltkreis zurück. Wenn die Spannung im Schaltkreis höher ist als die gespeicherte im Kondensator, fließt der Strom hinein und lädt den Kondensator auf.
Wenn die Spannung im Schaltkreis hingegen niedriger ist, wird die gespeicherte Ladung abgegeben. Kondensatoren werden oft zwischen Strom und Erdung in der Nähe eines Sensors oder Motors platziert, um Spannungsschwankungen auszugleichen.

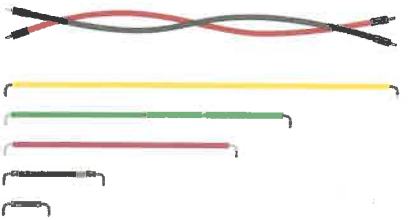
Gleichstrommotor - Wandelt elektrische Energie in mechanische Energie um, wenn Elektrizität an seinen Leitungen angelegt wird. Drahtspulen im Inneren des Motors werden magnetisiert, sobald Strom durch sie fließt. Diese Magnetfelder ziehen Magneten an und stoßen sie wieder ab und

bringen so die Welle zum Rotieren. Wenn sich die Richtung der Elektrizität umkehrt, dreht sich auch der Motor in die entgegengesetzte Richtung.

Diode - Stellt sicher, dass Elektrizität nur in eine Richtung fließt. Das ist nützlich, wenn Du einen Motor oder eine andere starke Strom-/ Spannungslast in Deinem Schaltkreis hast. Dioden sind polarisiert, d.h. die Richtung, in der sie platziert werden, spielt eine Rolle. In der einen Richtung lassen sie Strom durchfließen, in der anderen blockieren sie ihn. Für gewöhnlich schließt man die Anode an der Position im Schaltkreis mit höherer Energie an, wohingegen die Kathode an der Position mit niedrigerer Energie oder der Erdung angeschlossen wird. Die Kathode ist üblicherweise mit einem Streifen auf einer Seite des Bauteils gekennzeichnet.

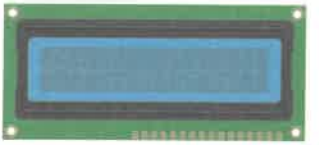
Farbfilter (rot, grün, blau) - Sie filtern unterschiedliche Wellenlängen des Lichts heraus. Wenn sie in Verbindung mit Photowiderständen verwendet werden, sorgen sie dafür, dass die Sensoren nur auf die Lichtstärke der gefilterten Farbe reagieren.

H-Brücke - Ein Schaltkreis, in dem man nicht nur die Höhe, sondern auch die Polarität der Spannung auf einer Last, in der Regel einen Motor, steuern kann. Die H-Brücke ist im Kit als Schaltung integriert, aber man könnte sie auch mit einzelnen Bauteilen herstellen.

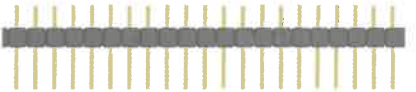


Verbindungskabel - Werden dazu verwendet, Bauteile auf der Steckplatine miteinander oder dem Arduino zu verbinden.

Leuchtdioden (LEDs) - Eine Diode, die aufleuchtet, sobald Elektrizität durch sie fließt. Wie bei allen Arten von Dioden fließt Elektrizität in diesen Bauteilen nur in eine Richtung. Du kennst sie wahrscheinlich als Anzeigen auf einer Vielzahl von elektronischen Geräten. Die Anode, wo für gewöhnlich der Strom angelegt wird, ist in der Regel das längere Bein, die Kathode das kürzere.



Flüssigkristallanzeige (LCD) - Eine auf Flüssigkristallen basierende alphanumerische oder grafische Anzeige. LCDs gibt es in vielen Größen, Formen und Arten. Deine LCD hat zwei Zeilen mit je 16 Zeichen.



Stiftleiste - Diese Pins passen in Buchsen, wie die auf der Steckplatine. Sie vereinfachen die Verbindung zwischen Teilen erheblich.

Optokoppler - Hiermit kann man zwei Schaltkreise verbinden, die keine gemeinsame Stromversorgung haben. Innen ist eine kleine LED eingebaut, die beim Aufleuchten einen Photorezeptor anstrahlt, der daraufhin einen internen Schalter schließt. Wenn Du die Spannung an + (Plus) anschließt, leuchtet die LED und der interne Schalter schließt. Durch die zwei Ausgänge ersetzt man einen Schalter im zweiten Stromkreis.

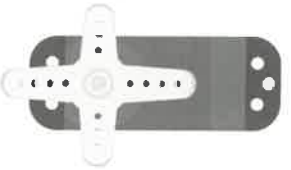
Piezo - Ein elektrisches Bauteil zur Ermittlung von Erschütterungen und Geräuscherzeugung.

Photowiderstand - (auch Fotozelle oder lichtabhängiger Widerstand genannt). Ein variabler Widerstand, der sich in Abhängigkeit von der auf ihn eintreffenden Lichtstärke verändert.

Potentiometer - Ein variabler Widerstand mit drei Pins. Zwei der Pins werden an die Enden eines festen Widerstandes angeschlossen. Der mittlere Pin, auch Schleifer genannt, bewegt sich über den Widerstand und teilt ihn in zwei Hälften. Wenn die beiden äußeren Seiten des Potentiometers an Spannung und Erdung angeschlossen werden, gibt der mittlere Pin, je nach Position des Drehknopfs, die Differenz der Spannung aus. Wird häufig auch Poti genannt.

Drucktasten - Tasten, die einen Schaltkreis schließen, wenn sie gedrückt werden. Sie eignen sich gut für die Ermittlung von Ein/Aus-Signalen.

Widerstände - Behindern den Fluss der elektrischen Energie in einem Schaltkreis und ändern so die Spannung und den Strom. Widerstandswerte werden in Ohm gemessen (dargestellt durch den griechischen Buchstaben Omega: Ω). Die farbigen Streifen an den Widerständen zeigen ihren Widerstandswert an (siehe die Widerstands-Farbtabelle auf Seite 41).



Servomotor - Ein Getriebemotor, der sich nur um 180 Grad drehen kann. Er wird durch elektrische Impulse von Deinem Arduino gesteuert, die ihm sagen, an welche Position er sich bewegen soll.

Temperaturfühler - Ändert seine ausgehende Spannung in Abhängigkeit von der Temperatur des Bauteils. Die äußeren Pins werden an Strom und Erdung angeschlossen. Die Spannung auf dem mittleren Pin ändert sich, wenn er wärmer oder kälter wird.

Kippschalter - Ein Schalter, der in Abhängigkeit von seiner Ausrichtung öffnet oder schließt. Üblicherweise sind dies Hohlzylinder mit einer Metallkugel im Innern, die eine Verbindung zwischen zwei Leitungen herstellen, wenn sie in die richtige Richtung gekippt werden.

Transistor - Ein Bauteil mit drei Pins, das als elektronischer Schalter eingesetzt werden kann, der sich dazu eignet, Bauteile mit hohem Stromfluss bzw. hoher Spannung, wie z.B. Motoren, zu steuern. Ein Pin wird an die Erdung angeschlossen, ein weiterer an die zu steuernde Komponente, der dritte an den Arduino. Wenn der Transistor Spannung auf dem mit dem Arduino verbundenen Pin empfängt, schließt er den Schaltkreis zwischen der Erdung und der zu steuernden Komponente.

USB-Kabel - Hiermit kannst Du Deinen Arduino Uno zur Programmierung mit Deinem PC verbinden. Außerdem versorgt es für die meisten Projekte in Deinem Kit den Arduino mit ausreichend Strom.

TABELLE DER SCHEMATISCHEN SYMBOLE

VERBUNDENE LEITUNGEN

NICHT VERBUNDENE LEITUNGEN

KIPPSCHALTER

WIDERSTAND

LED

POLARISIERTER KONDENSATOR

ERDUNG

TRANSISTOR

MOSFET

PHOTOWIDERSTAND

DIODE

KONDENSATOR

DRUCKTASTER

MOTOR

POTENTIOMETER

PIEZO

BATTERIE

In diesem Buch zeigen wir Dir Schaltkreise sowohl in realistischer Abbildung, als auch mit schematischen Diagrammen (sogenannte Schaltpläne). Die Abbildungen geben Dir eine Idee davon, wie die Steckplatine in einer möglichen Umsetzung des Projektes aussehen könnte. Schaltpläne hingegen benutzen Symbole, um das wesentliche des Schaltkreises darzustellen: sie zeigen die Bauteile und ihre Anschlüsse klar, präzise und eindeutig, aber nicht deren physische Anordnung. Wir benutzen Schaltpläne und schematische Symbole, um Schaltkreise zu beschreiben. Du wirst bei Deiner Erforschung der Welt der Elektronik feststellen, dass einige Bücher und Webseiten ausschließlich schematische Darstellungen verwenden. Zu lernen, wie man diese Schaltpläne lesen kann, ist daher eine wertvolle Fähigkeit. Hier sind die Symbole, die wir in diesem Buch verwenden werden.

DIE ARDUINO-PLATINE

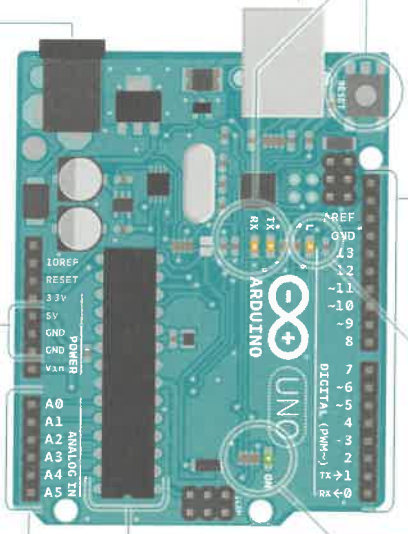
Stromanschluss
Hier wird der Arduino mit Strom versorgt, wenn dieser nicht über einen USB-Anschluss verbunden ist. Der Anschluss ist für Spannungen von 7-12 V ausgelegt.

USB-Anschluss
Wird verwendet, um Deinen Arduino mit Strom zu versorgen, um Entwürfe auf Deinen Arduino hochzuladen und um mit Deinem Arduino zu kommunizieren (über `Serial println()` usw.)

Rücksetztaste (Reset)
Setzt den ATmega Mikrocontroller zurück.

TX- und RX-LEDs
Diese LEDs zeigen Dir durch ihr Leuchten, wenn Dein Arduino mit Deinem Computer kommuniziert. Du kannst davon ausgehen, dass sie während des Uploads eines Sketches und während der seriellen Kommunikation schnell blinken. Sie können bei der Fehlersuche von Nutzen sein.

Digitale Pins
Benutze diese Pins mit `digitalRead()`, `digitalWrite()` und `analogWrite()`. `analogWrite()` funktioniert nur an Pins mit dem PWM-Symbol.



Pin 13 LED
Der einzige eingebaute Aktor auf Deinem Arduino. Abgesehen davon, dass er für Deinen ersten Blink-Sketch zum Einsatz kommt, ist diese LED für die Fehlersuche sehr hilfreich.

ATmega Mikrocontroller
Das Herzstück Deines Arduino Uno.

Power LED
Zeigt an, dass Dein Arduino mit Strom versorgt wird. Nützlich zur Fehlersuche.

GND- und 5-V Pins
und 5V-Pins - Benutze diese Pins zur Bereitstellung von Energie und Erdung (GND) in Deinen Schaltkreisen.

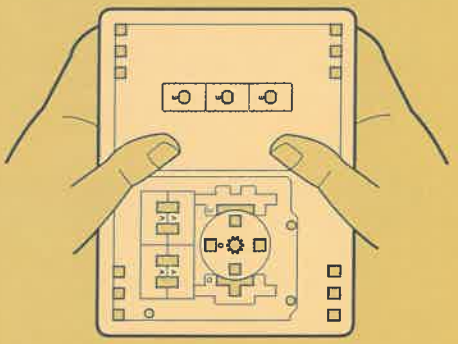
Analogeingang
Benutze diese Pins mit `analogRead()`

Dein Starter-Kit enthält eine vorgestanzte, einfach zusammenzubauende hölzerne Unterlage, mit der die Umsetzung all Deiner Projekte - ob aus diesem Buch oder daran anschließende - sogar noch einfacher wird.

Um sie zusammenzubauen, nimm die hölzerne Platte aus der Schachtel und befolge die Anweisungen rechts.

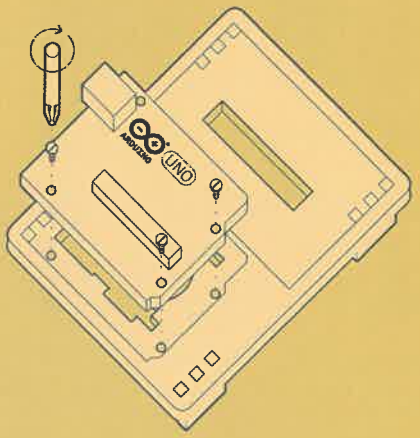
Gib acht nur die gezeigten Teile zu benutzen und bewahre die anderen Teile gut auf: Du benötigst sie für einige der späteren Projekte.

Los geht's!



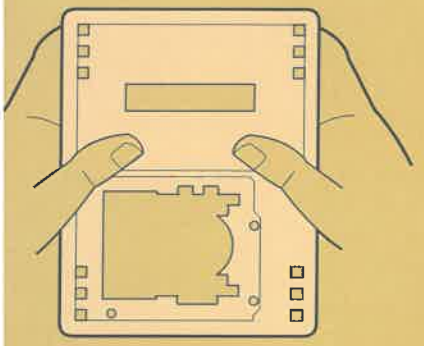
1

Nimm die hölzerne Platte und trenne sorgfältig die Teile ab.



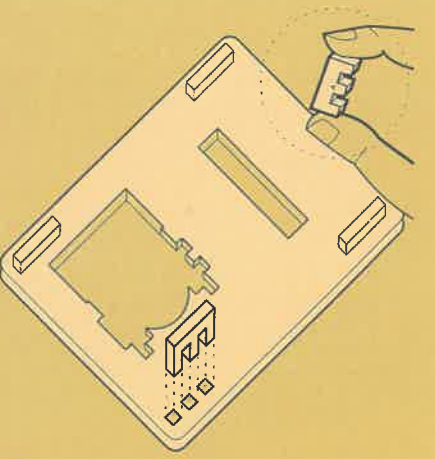
4

Befestige Deinen Arduino Uno mit 3 Schrauben an der Platte.
Gib acht, dass Du die Schrauben nicht zu fest anziehst.



2

Fahre fort bis Du alle Teile abgetrennt hast.

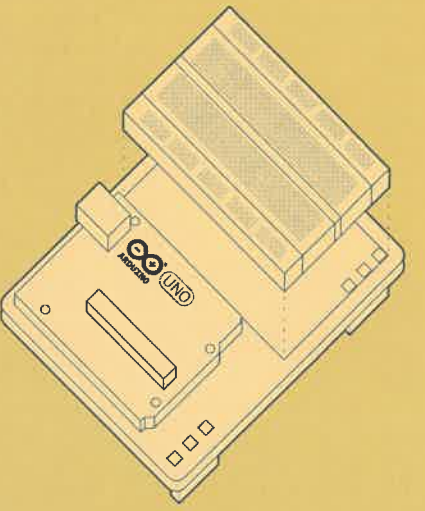


3

Platziere alle Teile mit einem „A" in die Löcher in den Ecken dies sind die Füße der Platte.

5

Ziehe sorgfältig die Schutzfolie von der Steckplatine ab



6

Klebe die Steckplatine auf die Platte, neben den Arduino Uno.

SACHEN, DIE DU BESCHAFFEN MUSST

9-V-Batterie

Kleine Lichtquelle, wie eine Taschenlampe

Leitendes Material wie Aluminiumfolie oder eine Kupfermatte

Farbiges Papier

Schere

Eine alte CD oder DVD

Klebeband und Kleber

Eine Schachtel, in die Du Löcher bohren kannst

Grundlegende Werkzeuge wie einen Schraubendreher

9-V-batteriebetriebenes Gerät
Ein beliebiges 9V-Batterie betriebenes elektronisches Gerät mit mindestens einem Schalter, das du gewillt bist, auseinander zu nehmen.

Lötkolben und Lötzinn
(Wird nur für Projekt 15 benötigt)

INSTALLATION

BEVOR DU ANFÄNGST, DIE WELT UM DICH HERUM ZU STEUERN, MUSST DU ZUR PROGRAMMIERUNG DEINER PLATINE DIE IDE DOWNLOADEN

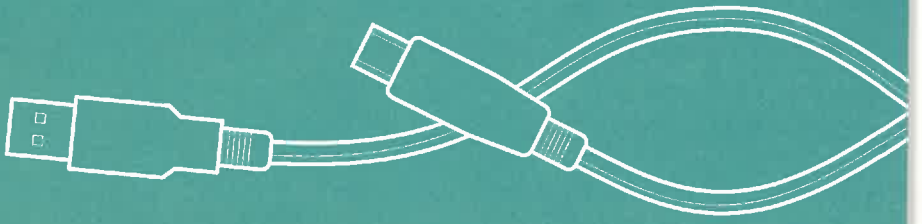
Mit der Arduino-IDE kannst Du Programme schreiben und sie auf Deinen Arduino hochladen.

Lade die neueste Version der IDE von:
arduino.cc/download herunter.

Halte Deine Arduino-Platine und das USB-Kabel griffbereit, schließe sie aber noch nicht an.

Folge den entsprechenden Anweisungen auf den folgenden Seiten für Windows, Mac OS X oder Linux.

Die Online-Version dieser Anleitung ist unter
arduino.cc/guide abrufbar.



INSTALLATION UNTER WINDOWS

Online-Version arduino.cc/windows

WINDOWS 7, VISTA UND XP

1. Nach Fertigstellung des Downloads doppelklicke auf die Datei „Arduino installieren". Falls eine Sicherheitswarnung erscheint, klicke auf „Starten" oder „Zulassen" und akzeptiere die Lizenzvereinbarung. Klicke auf „Weiter", wähle den Ordner, in dem die IDE installiert werden soll, und klicke auf „Installieren".

2. Verbinde den Arduino und Deinen Computer mit dem USB-Kabel. Das Board bezieht automatisch Strom vom USB-Anschluss des Computers und die grüne LED (mit ON beschriftet) wird aufleuchten.

3. Windows sollte den Vorgang der Treiberinstallation automatisch starten, sobald das Board angeschlossen wird. Allerdings wird Dein Computer den Treiber nicht selbständig finden können; Du wirst ihm den richtigen Order zeigen müssen.
   — Windows XP: Wenn Windows Update nach dem Pfad der Software fragt, wähle „Ja, nur dieses eine Mal" und dann „Software von einer Liste oder bestimmten Quelle installieren".
   — Vista oder Windows 7: Wenn unter Windows 7 ein Popup-Fenster nachfragt, ob der Treiber automatisch installiert werden oder ob er auf dem Computer gesucht werden soll, wähle Treiber auf dem Computer suchen. Unter Vista mache direkt mit dem nächsten Schritt weiter, indem Du die empfohlene Auswahl übernimmst.

4. Falls die Installation nicht automatisch startet, klicke auf das Startmenü und öffne die Systemsteuerung. Dann starte den Geräte-Manager und folge den nachstehenden Schritten:
   — Windows XP: Schalte zur klassischen Ansicht und wähle System - Hardware - Geräte-Manager
   — Windows Vista: Klassische Ansicht - Geräte-Manager
   — Windows 7: System und Sicherheit - System - Geräte-Manager.

5. Suche die Arduino-Platine unter der Kategorie „Andere Geräte" oder „Unbekannte Geräte" und wähle mit der rechten Maustaste „Treiber aktualisieren" oder „Treibersoftware aktualisieren".

6. Klicke auf „Durchsuchen" und wähle den Ordner „Drivers" (nicht „FTDI USB Drivers") im Arduino-Verzeichnis. Klicke „OK" und „Weiter". Wenn ein Dialogfeld über einen Test mit dem Windows Logo erscheint, klicke auf „Trotzdem fortfahren". Windows installiert jetzt den Treiber.

7. Im Geräte-Manager, unter „Anschlüsse (COM & LPT)", solltest Du jetzt etwas wie „Arduino UNO (COM4)" sehen.

Glückwunsch! Du hast die Arduino IDE auf Deinem Computer installiert.

INSTALLATION UNTER MAC-OS X

OS X 10.5 UND NEUER

Online-Version **arduino.cc/mac**

1. Wenn Du 10.8 (Mountain Lion) oder neuer verwendest, navigiere zu „System Preferences" und öffne „Security & Privacy". Im Tab „General" unter dem Titel „Allow applications downloaded from" aktiviere „Anywhere".

2. Sobald die Arduino IDE heruntergeladen ist, doppelklicke auf die .zip-Datei, um die Anwendung zu entpacken.

3. Kopiere die Arduino-Anwendung in den „Applications"-Ordner, oder wo immer sont Du die Software installieren möchtest.

4. Verbinde Computer und Arduino mit dem USB-Kabel. Das Board bezieht automatisch Strom vom USB-Anschluss des Computers und die grüne LED (mit ON beschriftet) wird aufleuchten.

5. Du brauchst keine Treiber zu installieren, um mit dem Board zu arbeiten.

6. Abhängig von der Version von OS X, die Du benutzt, könnte eine Dialogbox erfragen, ob Du die „System Preferences" öffnen möchtest. Klicke auf die Schaltfläche „Network Preferences" und dann auf „Apply".

7. Der Uno wird als „Not Configured" angezeigt, aber er funktioniert bereits. Du kannst die „System Preferences" verlassen.

Glückwunsch! Du hast den Arduino installiert und bist bereit für Deine Projekte.

LINUX INSTALLIEREN

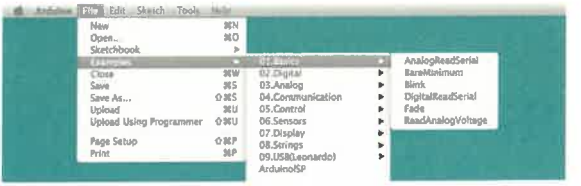
Wenn Du Linux verwendest, gehe bitte für eine Anleitung auf unsere Homepage.
arduino.cc/linux

KOMMUNIKATION MIT DEM ARDUINO

Jetzt, da Du die Arduino-IDE installiert hast und das Board mit dem Computer kommunizieren kann, ist es Zeit zu überprüfen, ob Du ein Programm hochladen kannst.

1. Doppelklicke die Arduino-Anwendung, um sie zu öffnen. Falls die IDE in der falschen Sprache startet, kannst Du dies in den Anwendungseinstellungen ändern. Schaue unter „Language Support" auf der Seite *arduino.cc/ide* nach Details.

2. Navigiere zum LED-Blink-Beispiel-Sketch („Sketch" werden Arduino-Programme genannt). Du findest ihn unter:
DATEI > BEISPIELE > 01.BASICS > BLINK



3. Ein Fenster mit etwas Text sollte sich geöffnet haben. Ignoriere es erstmal und wähle Dein Board unter:
WERKZEUGE > PLATINE

4. Wähle die serielle Schnittstelle, an der Dein Arduino angeschlossen ist im Menü ***WERKZEUGE-SERIELLE SCHNITTSTELLE.***.

— *Unter Windows.* Es ist wahrscheinlich der COM mit der höchsten Nummer. Es kann nichts passieren, wenn man den falschen erwischt, also probiere sie ruhig der Reihe nach aus, bis es funktioniert. Um es gezielt herauszufinden, entferne Deine Arduino-Platine und öffne das Menü erneut; der Eintrag, der jetzt verschwunden ist, sollte Deine Arduino-Platine sein. Schließe das Board wieder an und wähle diesen seriellen Port.
— *Unter Mac.* Es sollte etwas in der Art /dev/tty.usbmodem zu finden sein. Es gibt normalerweise zwei von diesen; wählen einen der beiden.

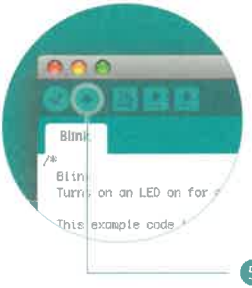


Abbildung 1

5. Um den Blink-Sketch auf Deinen Arduino hochzuladen, klicke auf die Schaltfläche ***HOCHLADEN*** in der oberen linken Ecke des Fensters. Siehe Abb. 1

6. Du solltest in der linken unteren Ecke einen Balken sehen, der den Hochladefortschritt anzeigt, und die TX- und RX-LEDs auf der Arduino-Platine sollten blinken. Wenn das Hochladen erfolgreich war, zeigt die IDE die Meldung ***„HOCHLADEN ABGESCHLOSSEN"*** an.

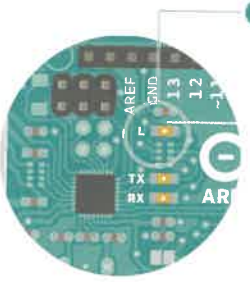


Abbildung 2

7. Ein paar Sekunden später solltest Du sehen, wie die mit L markierte LED anfängt zu blinken. Siehe Abb. 2
Wenn das der Fall ist, Glückwunsch! Du hast erfolgreich den Arduino programmiert, seine integrierte LED blinken zu lassen!

Manchmal ist Dein nagelneuer Arduino bereits mit dem Blink-Sketch programmiert, so dass Du nicht sicher sein kannst, ob Du wirklich die volle Kontrolle hast. Um völlig sicherzugehen, ändere die Verzögerungszeit zwischen den Blinks, indem Du die Zahl in Klammern auf 100 setzt und den Blink-Sketch erneut hochlädst. Jetzt sollte die LED viel schneller blinken.

Glückwunsch! Du hast jetzt wirklich die volle Kontrolle!
Nun ist es Zeit, zu Projekt 1 zu gehen.
(Du brauchst keine Änderungen zu speichern, die Du vorgenommen hast.)

ZUSÄTZLICHE INFORMATIONEN

Wenn Du Probleme mit einem der Schritte hast, die oben beschrieben wurden, schau bitte bei unseren Fehlerbehebungsvorschlägen nach:
arduino.cc/trouble

Neben der Vorbereitung Deiner eigenen Projekte, kannst Du auf den folgenden Seiten zusätzliche Informationen über die Arduino-Programmierumgebung erhalten:
arduino.cc/ide

Du kannst auch hier reinschauen:

— Beispiele für die Verwendung verschiedener Sensoren und Aktoren
arduino.cc/tutorial

— Referenz zur Arduino-Programmiersprache
arduino.cc/examples

01

SCHALTER

LED

220-OHM-WIDERSTAND

LERNE DEINE WERKZEUGE KENNEN

DU BAUST EINEN EINFACHEN SCHALTKREIS MIT EINIGEN SCHALTERN, EINER LED UND EINEM WIDERSTAND

Entdecke grundlegende elektrische Theorien, wie eine Steckplatine funktioniert, Bauteile in serieller und Parallelschaltung.

Zeit: **30 MINUTEN**
Niveau: ■ □ □ □ □

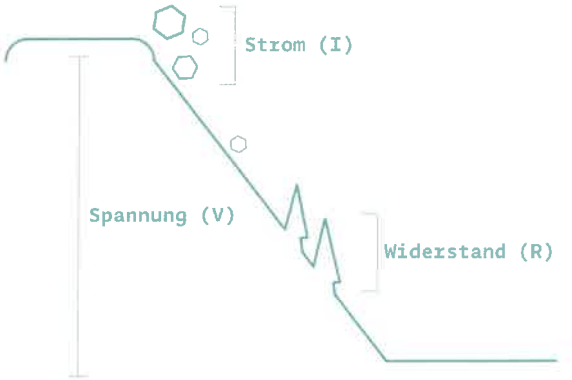
Elektrizität ist eine Form von Energie, wie Wärme, Schwerkraft oder Licht. Elektrische Energie fließt durch Leiter, wie z.B. Draht. Du kannst elektrische Energie in andere Energieformen umwandeln, um etwas Interessantes zu bewirken, wie z.B. ein Licht einzuschalten oder Geräusche mit einem Lautsprecher zu erzeugen.

Bauteile, die man dazu benutzen könnte, wie Lautsprecher oder Glühlampen, sind elektrische „Wandler". Wandler transformieren andere Arten von Energie in elektrische Energie und umgekehrt. In der Regel nennt man jene Dinge, die andere Energieformen in elektrische Energie umwandeln, Sensoren, und solche Dinge, die umgekehrt elektrische Energie in andere Formen umwandeln, „Aktoren". Du wirst „Schaltkreise" bauen, um Elektrizität durch verschiedene Bauteile zu leiten. Schaltkreise sind in sich geschlossen, bestehen aus Draht und verfügen über eine Energiequelle (wie z.B. eine Batterie), sowie eine sogenannte Last, die etwas Nützliches mit dieser Energie macht.

In einem Schaltkreis fließt Elektrizität von einem Punkt höherer potenzieller Energie (normalerweise gekennzeichnet als Power oder +) zu einem Punkt niedrigerer potenzieller Energie. Die Erdung (häufig dargestellt als - oder GND) ist im Allgemeinen der Punkt mit der niedrigsten potenziellen Energie in einem Schaltkreis. In den Schaltkreisen, die Du erstellen wirst, fließt die Elektrizität nur in eine Richtung, was „Gleichstrom" oder DC genannt wird. Bei Wechselstrom (AC) wechselt die Elektrizität 50 oder 60 mal pro Sekunde die Richtung (abhängig davon, wo Du lebst). Dies ist die Art von Elektrizität, die aus einer Steckdose kommt.

Es gibt einige Begriffe die Du kennen solltest, wenn Du mit elektrischen Schaltungen arbeitest. Strom (gemessen in Ampere; mit dem Symbol „**A**") ist die Menge an elektrischer Ladung, welche an einem bestimmten Punkt in Deinem Schaltkreis fließt. „Spannung" (gemessen in Volt; mit dem Symbol „**V**") ist die Energiedifferenz zwischen zwei Punkten in einem Schaltkreis. Widerstand (gemessen in Ohm; mit dem Symbol „Ω") kennzeichnet, wie sehr ein Bauteil dem Fluss der elektrischen Energie widersteht.

Zur Veranschaulichung kann man hierbei an Felsen denken, die gerade einen Abhang herunterstürzen, wie in Abb. 1 dargestellt. Je höher der Abhang, desto mehr Energie haben die Felsen, wenn sie unten aufschlagen. Die Höhe des Abhangs ist also wie die Spannung in einem Schaltkreis: und je höher die Spannung an der Energiequelle, desto mehr Energie muss verbraucht werden. Die Anzahl der Felsen hingegen entspricht dem Strom in einem Schaltkreis: je mehr Felsen man hat, desto mehr Energie wird den Abhang herunter getragen. Büsche oder andere Hindernisse, welche die Felsen beim Herunterrollen überwinden müssen, sind wie Wiederstände im Schaltkreis zu verstehen. Die Überwindung verbraucht Energie, die somit in andere Energieformen umgewandelt wird.



Felssturz als Gleichnis für den elektrischen Stromfluss.
Abbildung 1

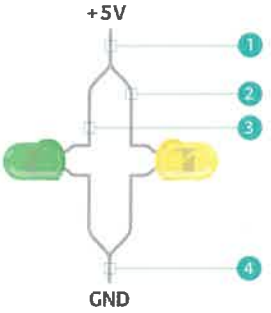
EIN PAAR DETAILS ZU SCHALTKREISEN

— Damit ein Schaltkreis funktioniert, muss es einen vollständigen Pfad von der Energiequelle (Strom) zum Punkt der geringsten Energie (Erdung) geben, über den der Strom fließen kann.

— Die gesamte elektrische Energie in einem Schaltkreis wird durch die Bauteile in ihm verbraucht. Jedes Bauteil wandelt etwas von der Energie in eine andere um. In jedem Schaltkreis wird die gesamte Spannung in eine andere Form von Energie umgewandelt (Licht, Wärme, Ton usw.).

— Der Stromfluss an einem spezifischen Punkt in einem Schaltkreis ist immer der gleiche, eingehend und ausgehend.

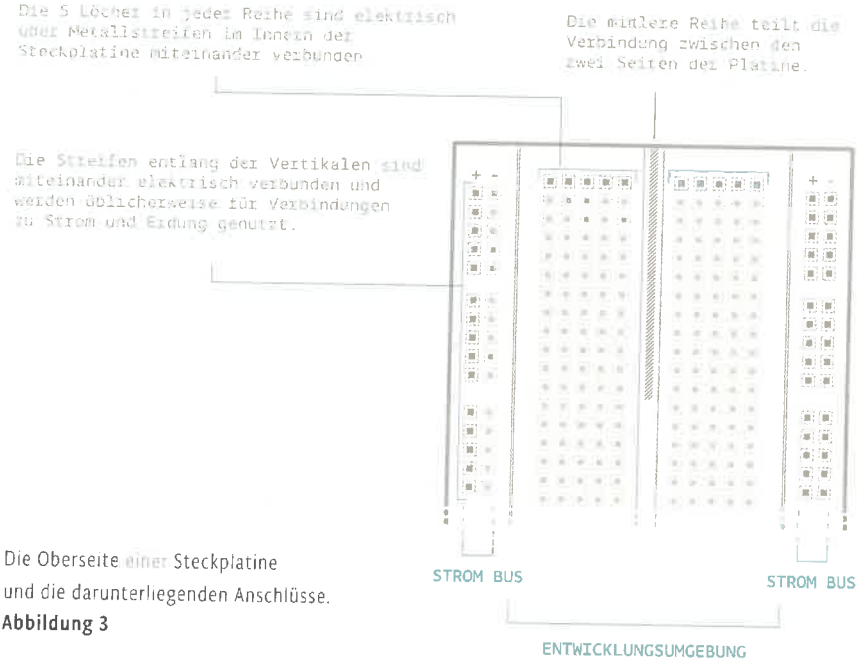
— Elektrischer Strom bevorzugt den Pfad des geringsten Widerstandes zur Erdung, d.h. bei zwei möglichen Pfaden fließt der größere Anteil des elektrischen Stroms entlang des Wegs mit geringerem Widerstand. Wenn Du eine Verbindung hast, welche Strom und Erdung ohne Widerstand zusammen verbindet, verursachst Du einen Kurzschluss und der Strom versucht, diesem Pfad zu folgen. Bei einem Kurzschluss wandeln die Energiequelle und die Leitungen die elektrische Energie in Licht und Wärme um, was sich als Funken äußern oder sogar zu einer Explosion führen kann. Wenn Du schon einmal eine Batterie kurzgeschlossen hast und die Funken gesehen hast, dann weißt Du, wie gefährlich ein Kurzschluss sein kann.



Der Strom bei (1) = Strom bei (2) + Strom bei (3) = Strom bei (4).
Abbildung 2

WAS IST EINE STECKPLATINE?

Die Steckplatine ist der primäre Ort, an dem Du Schaltkreise aufbauen wirst. Die Steckplatine in Deinem Kit erfordert kein Löten, sondern ist eher wie eine Art elektronisches LEGO. Die horizontalen und vertikalen Reihen der Steckplatine, wie in Abb. 3 gezeigt, leiten Strom durch dünne Metallstecker, die sich unter dem Kunststoff mit den Bohrungen befinden.

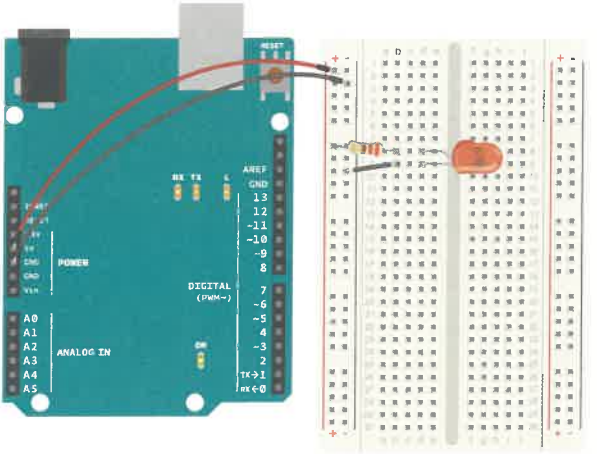


Die Oberseite einer Steckplatine und die darunterliegenden Anschlüsse.
Abbildung 3

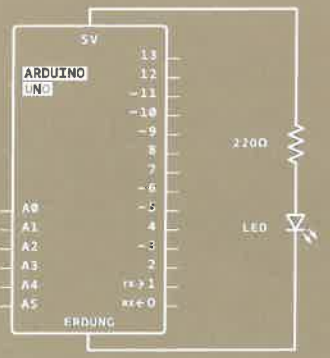
Die Leiterbahnen innerhalb einer Steckplatine.
Abbildung 4

SCHALTKREIS-ILLUSTRATION

Während Deiner Projekte werden Dir zwei Darstellungsformen von Schaltkreisen begegnen: eine Ansicht zeigt die Steckplatine, wie sie in Deinem Kit enthalten ist (siehe Abb. 5). Die andere ist eine schematische Darstellungsform (siehe Abb. 6), d.h. eine abstraktere Art Verbindungen zwischen den Bauteilen in einem Schaltkreis abzubilden. Diese sogenannten Schaltbilder zeigen nicht wie Bauteile im Verhältnis zu einander gesetzt, sondern wie sie verbunden werden.



Schaltkreis-illustration
Abbildung 5



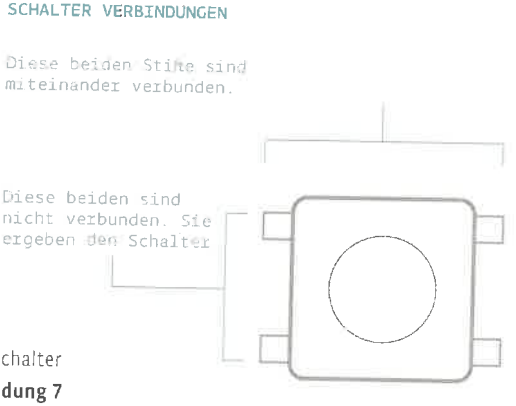
Diagrammansicht
Abbildung 6

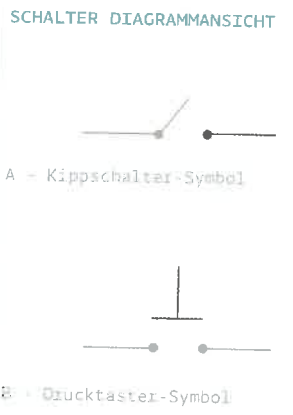
DEINE ERSTEN BAUTEILE

Eine ***LED*** oder Leuchtdiode ist ein Bauteil, das elektrische Energie in Licht umwandelt. LEDs sind polarisiert, d.h. sie lassen Elektrizität nur in eine Richtung durchfließen. Der längere Stift der LED wird Anode genannt und an Strom angeschlossen. Der kürzere Stift wird Kathode genannt und an die Erdung angeschlossen. Wenn Spannung an der Anode der LED angelegt wird und die Kathode an Erdung angeschlossen ist, leuchtet die LED auf.

Ein ***Widerstand*** ist ein Bauteil, das dem Fluss der elektrischen Energie widersteht (für eine Erklärung zu den farbigen Streifen auf der Seite siehe Bauteilverzeichnis). Er wandelt einen Teil der elektrischen Energie in Wärme um, so dass andere nachgeschaltete Bauteile, wie z.B. eine LED, entsprechend weniger Energie empfangen. Das ermöglicht Dir, Bauteilen genau die Menge an Energie zuzuführen, die sie benötigen. Durch den sequentiellen Einsatz eines Widerstandes mit einer LED, schützt man sie davor, zu viel Spannung zu empfangen. Ohne den Widerstand würde die LED zwar einige Momente heller leuchten, jedoch auch schnell ausbrennen.

Ein geöffneter ***Schalter*** unterbricht den Fluss der Elektrizität und somit den Schaltkreis. Wenn ein Schalter geschlossen wird, vervollständigt er den Schaltkreis. Es gibt viele Arten von Schaltern. Die in Deinem Kit werden Taster oder Drucktaster genannt, weil sie nur geschlossen sind, d.h. den Stromfluss zulassen, wenn Kraft auf sie ausgeübt wird.





Schalter
bildung 7

BAUE DEN SCHALTKREIS

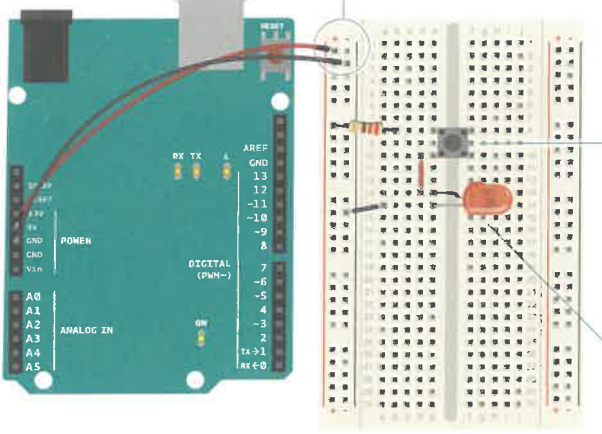


Abbildung 8

Dein erster interaktiver Schaltkreis mit einem Taster, einem Widerstand und einer LED.
Der Arduino ist in diesem Schaltkreis nur die Energiequelle; in späteren Projekten wirst Du seine Ein- und Ausgänge für den Aufbau komplexerer Schaltkreise einsetzen.

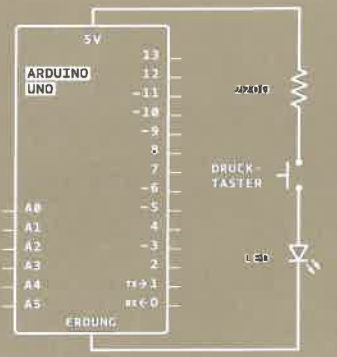


Abbildung 9

Du verwendest den Arduino in diesem Projekt nur als Stromquelle. Über eine Verbindung zu USB oder einer 9-Volt-Batterie stellt der Arduino 5 Volt zwischen seinem 5V-Pin und GND-Pin zur Verfügung. 5V steht für 5 Volt und wird häufig auf diese Weise geschrieben.

1. Wenn Dein Arduino über eine Batterie oder USB angeschlossen ist, trenne die Stromverbindung, bevor Du mit dem Bau des Schaltkreises anfängst! Stecke ein rotes Kabel in den 5V-Pin des Arduino und das andere Ende in eine der langen Busleitungen auf der Steckplatine.

2. Schließe die Erdung (GND) des Arduino mit einem schwarzen Kabel an der danebenliegenden Busleitung an. Es ist sinnvoll die Kabelfarben in einem Schaltkreis einheitlich zu halten (rot für Strom, Schwarz für Erdung).

3. Nun, da die Platine mit Strom versorgt wird, platziere den Taster mittig über der Steckplatine, so dass er über dem Zentrum der Platine in einer Richtung sitzt und die Krümmung des Stifts zur Mitte der Platine zeigt.

4. Benutze einen 220-Ohm-Widerstand, um eine Seite des Tasters am Strom anzuschließen. Die Abbildungen in diesem Buch zeigen 4 Streifen, aber in Deinem Kit gibt es auch Widerstände mit 5 Streifen. Benutze die Abbildung auf der Seite, um zu überprüfen, ob Du den richtigen verwendest. Beachte Seite 47 für eine ausführliche Erklärung der Farbcodes von Widerständen. Schließe auf der anderen Seite des Tasters die Anode (langer Stift) der LED an und verbinde mit einem Kabel die Kathode (kurzer Stift) der LED mit der Erdung. Wenn Du so weit bist, schließe das USB-Kabel am Arduino an.

VERWENDUNG

Sobald alles fertig aufgebaut ist, drücke die Taste. Du solltest die LED aufleuchten sehen. Glückwunsch, Du hast gerade einen Schaltkreis gebaut! Sobald Du vom Drücken des Lichtschalters genug hast, wird es Zeit, das Ganze durch einen zweiten Taster aufzupeppen.

Du wirst Bauteile auf der Steckplatine sowohl nacheinander (in Reihe), als auch nebeneinander (parallel) einsetzen.

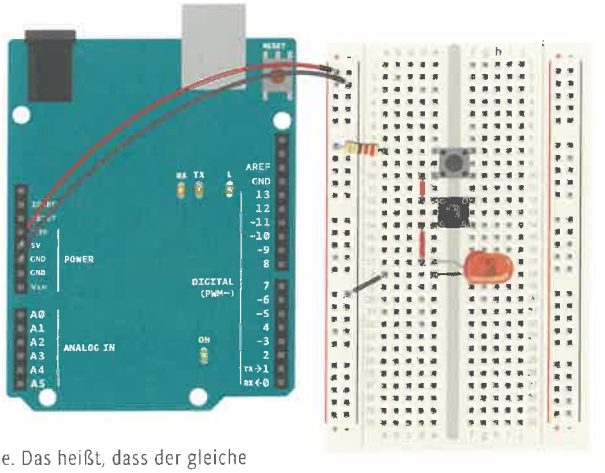
Reihenschaltung

SERIELLE, D.H. IN REIHE GESCHALTETE BAUTEILE KOMMEN NACHEINANDER DRAN

Sobald Du die Verbindung zur Energiequelle getrennt hast, füge einen Taster neben dem bereits vorhandenen Taster auf Deiner Steckplatine hinzu. Verdrahte sie in Reihe, wie in Abb. 10 gezeigt. Schließe diesmal die Anode (langer Stift) der LED am zweiten Schalter an und verbinde wieder die LED-Kathode mit der Erdung. Nun verbinde den Arduino wieder mit Strom; um die LED jetzt einzuschalten, musst Du beide Taster betätigen. Da diese in Reihe geschaltet sind, müssen auch beide geschlossen werden, damit im Schaltkreis Strom fließen kann.

Diese beiden Elemente sind in Reihe

TRENNE IMMER DIE STROMVERBINDUNG, BEVOR DU ETWAS IN DEINEM SCHALTKREIS VERÄNDERST.



Die beiden Schalter sind in Reihe. Das heißt, dass der gleiche elektrische Strom durch beide fließt, weshalb auch beide gedrückt werden müssen, um die LED einzuschalten.

Abbildung 10

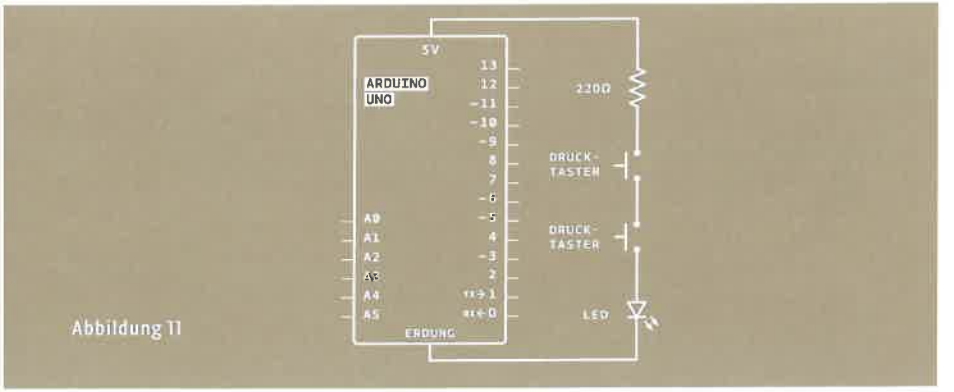
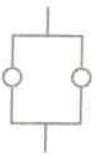


Abbildung 11



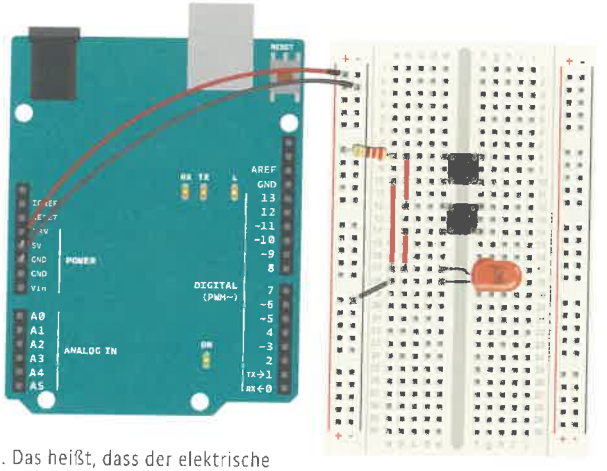
Diese beiden Elemente sind parallel

Parallelschaltung

PARALLEL GESCHALTETE BAUTEILE LAUFEN NEBENEINANDER

Jetzt, da Du die Reihenschaltung beherrschst, ist es an der Zeit, die Taster parallel zu schalten.

Belasse die Schalter und die LED an ihren Positionen, aber entferne die Verbindung zwischen den beiden Tastern. Verdrahte beide Schalter mit dem Widerstand und schließe jeweils das andere Ende an der LED an, wie in Abb. 12 gezeigt. Wenn Du jetzt einen der beiden Taster drückst, wird der Schaltkreis geschlossen und das Licht leuchtet auf.



Die beiden Schalter sind parallel. Das heißt, dass der elektrische Strom zwischen ihnen aufgeteilt wird, weshalb es ausreicht einen der Taster zu drücken, um die LED einzuschalten.

Abbildung 12

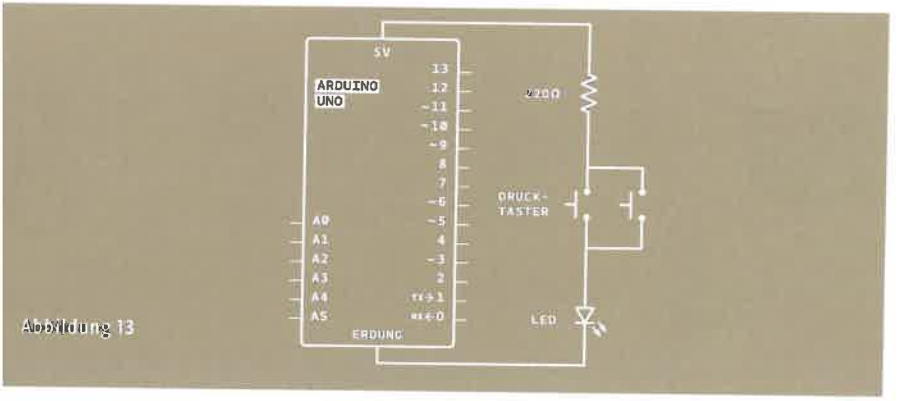
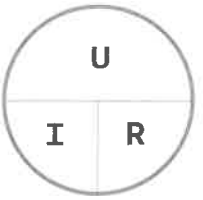


Abbildung 13

DAS OHMSCHE GESETZ VERSTEHEN



Du kannst diesen Kreis verwenden, um Dir das Verhältnis zwischen Spannung, Strom und Widerstand einzuprägen.
Lege Deinen Finger auf einen der drei Teile, um zu sehen, wie er sich zu den anderen beiden verhält.

`U = I * R`

`I = U / R`

`R = U / I`

Strom, Spannung und Widerstand sind miteinander verwandt. Wenn Du eine dieser Größen in einem Schaltkreis änderst, beeinflusst das automatisch die anderen. Das Verhältnis zwischen ihnen ist als Ohmsches Gesetz bekannt, benannt nach seinem Entdecker Georg Simon Ohm.

`SPANNUNG (U) = STROM (I) X WIDERSTAND (R)`

Die messbare Stromstärke der Schaltungen, die Du bauen wirst, liegt bei Werten im Milliampere-Bereich. Das ist ein Tausendstel eines Amperes.

Im Schaltkreis von Abb. 5 besteht eine Spannung von 5 Volt. Der Widerstand beträgt 220 Ohm. Um die Stromstärke zu berechnen, die bei der LED eintrifft, ersetze die Werte in der Gleichung. Eingesetzt ergibt das 5 = I x 220. Wenn Du beide Seiten durch 220 teilst, erhältst Du I = 0,023. Das heißt die LED hat 23 Tausendstel eines Amperes, oder 23 Milliampere (23 mA), verbraucht. Dieser Wert entspricht ungefähr dem Maximum, mit welchem diese LEDs noch sicher betrieben werden können. Deshalb haben wir einen 220-Ohm-Widerstand benutzt.

Du hast eine Reihe von Möglichkeiten, dieses Projekt zu erweitern. Zum Beispiel indem Du Deinen eigenen Taster baust (zwei Stücke Folie mit einer Leitung funktionieren gut), oder indem Du eine Kombination von Tastern und LED parallel und in Reihe schaltest. Was passiert, wenn Du drei oder vier LEDs in Reihe schaltest? Was geschieht, wenn sie parallel angeordnet werden? Wie kommt dieser Unterschied zustande?

Ein ***Multimeter*** ist ein Werkzeug, das die Menge des Widerstandes, des Stromes und der Spannung in Deinem Schaltkreis prüfen kann. Auch wenn Du es für diese Projekte nicht unbedingt benötigst, ist es ein nützlicher Bestandteil der Werkzeugkiste jedes Ingenieurs. Eine gute Beschreibung zur Verwendung eines Multimeters findest Du online unter ***arduino.cc/multimeter***

Du hast die elektrischen Eigenschaften der Spannung, des Stroms und des Widerstands beim Aufbau eines Schaltkreises auf einer Steckplatine kennengelernt. Mit Bauteilen wie LEDs, Widerständen und Tastern hast Du eines der einfachsten interaktiven Systeme gebaut: ein Benutzer betätigt den Taster, die Lichter gehen an. Diese Grundlagen der Elektronik werden in die kommenden Projekte einbezogen und erweitert.

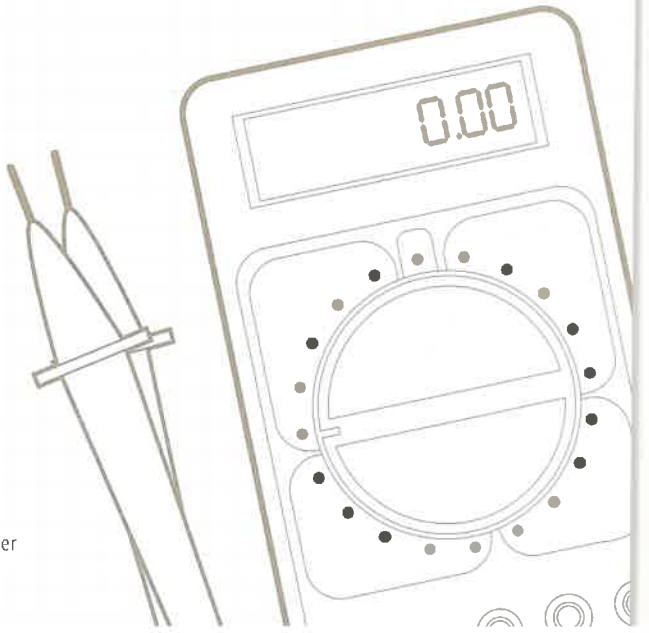


Abb. 14 - Multimeter

TASTER

LED

220-OHM-WIDERSTAND

10-KOHM-WIDERSTAND

RAUMSCHIFF-STEUERZENTRALE

DEIN ARDUINO WIRD ZUM MITTELPUNKT IN EINEM SCIENCE-FICTION-FILM

Entdecke: digitaler Eingang und Ausgang, Dein erstes Programm, Variablen

Zeit: **45 MINUTEN**
Niveau: ■■■■■

Basierend auf Projekten: **1**

Nun, da Du die Grundlagen der Elektrizität beherrschst, wird es Zeit, mit Deinem Arduino Dinge zu steuern. Das Ergebnis dieses Projekts wird ein Bedienelement mit Schaltern und Lichtern sein, das genauso gut das zentrale Steuerpult in einem Science-Fiction-Film der 70er hätte sein können. Hier bist Du derjenige der entscheidet, ob die Lichter den „Hyperdrive aktivieren" oder „Laser abfeuern!" bedeuten. Auf der Konsole wird eine grüne LED leuchten, bis Du eine Taste drückst. Das sendet dem Arduino ein Signal, das grüne Licht auszuschalten und 2 andere Lichter blinken zu lassen.

Die digitalen Pins des Arduino können nur zwei eingehende Zustände lesen: Spannung liegt an, oder nicht. Üblicherweise werden diese Art von Eingang digital (oder manchmal binär, für zwei Zustände) und diese Zustände **HIGH** und **LOW** genannt. **HIGH** entspricht dabei zu sagen „hier liegt Spannung an!" und **LOW** bedeutet „auf diesem Pin liegt keine Spannung an". Wenn Du einen **AUSGANG** mit dem Befehl `digitalWrite()` auf **HIGH** setzt, schaltest Du ihn ein. Wenn Du nun die Spannung zwischen dem Pin und der Erdung misst, sollte sie 5 Volt betragen. Wenn Du einen **AUSGANG** auf **LOW** setzt, schaltest Du ihn aus.
Die digitalen Pins des Arduino können sowohl als Eingänge, als auch als Ausgänge dienen, was durch die Konfiguration Deines Programms festgelegt wird. Konfigurierst Du die Pins als Ausgänge, können Sie Bauteile wie LEDs einschalten. Wenn Du die Pins als Eingänge konfigurierst, können sie überprüfen, ob ein Schalter betätigt wurde oder nicht. Da die Pins 0 und 1 für die Kommunikation mit dem Computer verwendet werden, ist es am besten, mit Pin 2 zu beginnen.

BAUE DEN SCHALTKREIS

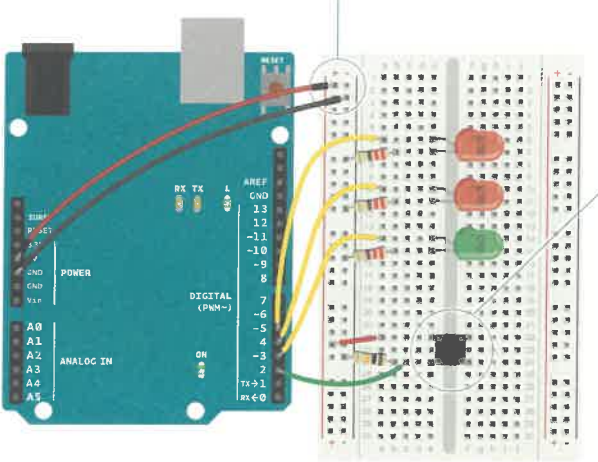


Abbildung 1

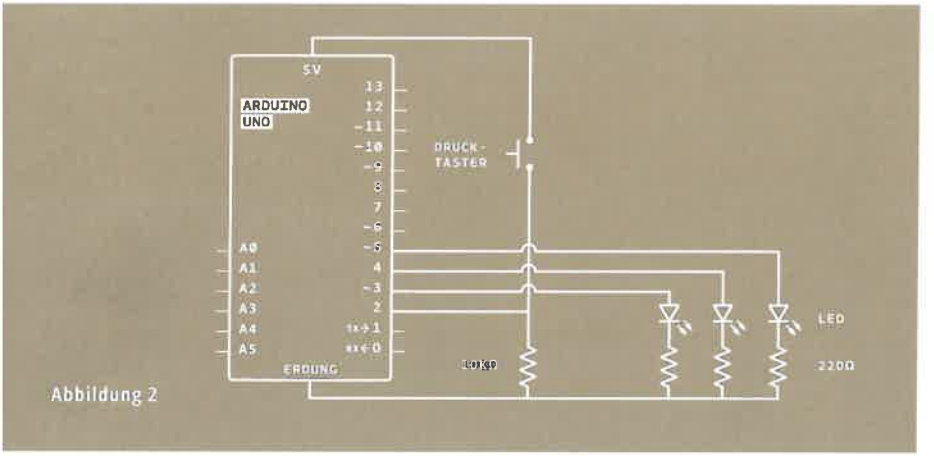
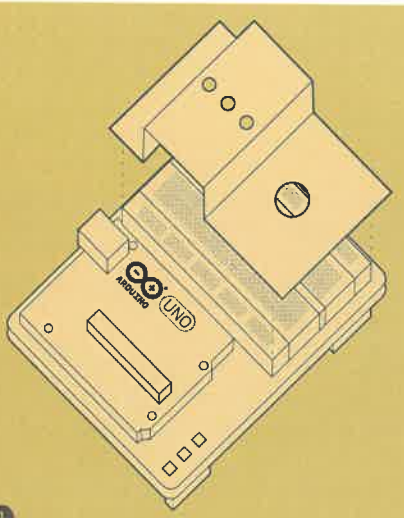


Abbildung 2

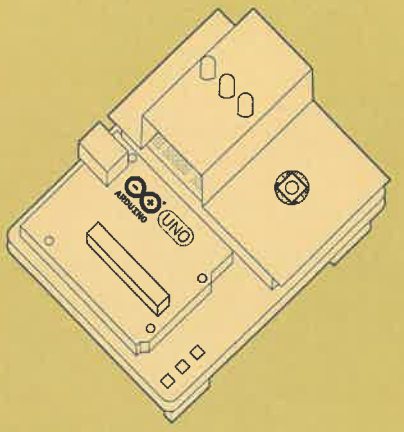
1. Verdrahte Deine Steckplatine mit 5V und der Erdung des Arduino, genau wie beim vorhergehenden Projekt. Setze zwei rote LEDs und eine grüne LED auf die Steckplatine und verbinde jeweils die Kathode (kurzer Stift) der LEDs über einen 220-Ohm-Widerstand mit der Erdung. Schließe die Anode (langer Stift) der grünen LED an Pin 3 an. Schließe die Anoden der roten LEDs an Pin 4 bzw. Pin 5 an.

2. Setze den Taster auf die Steckplatine, so wie Du es zuvor auch getan hast. Schließe eine Seite am Strom und die andere Seite am digitalen Pin 2 des Arduino an. Außerdem musst Du einen 10-kOhm-Widerstand zwischen der Erdung und dem Pin des Tasters, der mit dem Arduino verbunden ist, einsetzen. Dieser Pull-down-Widerstand verbindet den Pin mit der Erdung solange der Taster geöffnet ist, damit er **LOW** liest, wenn keine Spannung durch den Taster hindurch kommt.

Du kannst die Steckplatine mit der im Kit enthaltenen Schablone abdecken, oder sie selbst dekorieren, um Dein eigenes Abschusssystem zu bauen. Erst durch das Design Deiner Steuerkonsole und durch Deine Beschriftungen erhalten die Lichter Bedeutung. Wofür soll die grüne LED stehen? Was bedeuten die blinkenden roten LEDs? Du entscheidest!



1. Falte das vorgeschnittene Papier wie abgebildet.



2. Lege das gefaltete Papier auf die Steckplatine. Die drei LEDs und der Taster helfen, es an der richtigen Stelle zu halten.

DAS PROGRAMM

Einige Hinweise bevor Du beginnst

Jedes Arduino-Programm hat zwei Hauptfunktionen. Funktionen sind Teile eines Computerprogramms mit eindeutigen Namen, über die sie „aufgerufen" werden, und die spezifischen Befehle ablaufen lassen. Die notwendigen Funktionen in einem Arduino-Programm werden `setup()` und `loop()` genannt. Diese Funktionen müssen deklariert werden, d.h. Du musst Deinem Arduino erst erklären, was diese Funktionen tun sollen. `setup()` und `loop()` wurden deklariert, wie Du auf der rechten Seite siehst. Bevor Du mit dem Hauptteil des Programms beginnst, wirst Du eine Variable erstellen. Variablen sind von Dir zugewiesene Namen für Orte im Speicher des Arduino, um verfolgen zu können, was geschieht. Diese Werte können sich ändern, abhängig von Deinen Anweisungen im Programm. Variablennamen sollten den Wert beschreiben, den sie speichern. Beispielsweise sagt Dir eine Variable mit dem Namen `switchState`, was sie speichert: den Zustand (state) eines Schalters (switch). Hingegen sagt eine Variable mit dem Namen „X" nicht viel darüber aus, was sie enthält.

Beginnen wir zu programmieren

Um eine Variable zu erstellen, musst Du zuerst deklarieren, welchen *Typ* sie hat. Der *Datentyp* `int` speichert eine Ganzzahl (oder *Integer*); dies ist jede mögliche Zahl ohne ein Dezimalkomma. In der Regel gibst Du Variablen auch einen Anfangswert. Die Deklaration der Variablen muss, wie jeder Befehl, mit einem Semikolon (;) abgeschlossen werden.

Konfiguriere die Pin-Funktionsweise

`setup()` wird einmalig beim Einschalten des Arduino ausgeführt. Dort konfigurierst Du mit der Funktion `pinMode()`, ob die digitalen Pins Ein- oder Ausgänge sind. Die Pins, die an die LEDs angeschlossen werden, sind AUSGÄNGE, und der Pin am Taster ist ein EINGANG.

Schreibe die Schleife

`loop()` wird wiederholt ausgeführt, nachdem `setup()` abgeschlossen ist. In der Funktion `loop()` überprüfst Du die Spannung auf den Eingängen und schaltest Ausgänge ein und aus. Um den Spannungspegel auf einem digitalen Eingang zu überprüfen, benutzt Du die Funktion `digitalRead()`, welche die Spannung ausliest. Um zu wissen, welcher Pin ausgelesen werden muss, erwartet `digitalRead()` ein *Argument*. Argumente sind Informationen, die Du an Funktionen weitergibst, um ihnen zu erklären, was sie zu tun haben. Beispielsweise benötigt `digitalRead()` ein Argument: welcher Pin soll geprüft werden. In Deinem Programm wird `digitalRead()` den Zustand von Pin 2

```
void setup(){
}

void loop(){
}
```

{ Geschweifte Klammern }
Code innerhalb der geschweiften Klammern wird bei Funktionsaufruf ausgeführt.

```
int switchState = 0;
```

```
void setup(){
  pinMode(3,OUTPUT);
  pinMode(4,OUTPUT);
  pinMode(5,OUTPUT);
  pinMode(2,INPUT);
}
```

Groß- und Kleinschreibung (case sensitivity)
Beachte die Groß- und Kleinschreibung in Deinem Code. Z.B. ist `pinMode` der Name eines Befehls, wohingegen pinmode einen Fehler verursachen wird.

```
void loop(){
  switchState = digitalRead(2);
  // das ist ein Kommentar
```

Kommentare
Wann immer Du Anmerkungen in Deinem Code machen möchtest, kannst Du einen Kommentar setzen.
Kommentare sind nur für Dich und werden vom Computer ignoriert. Um einen Kommentar zu setzen, füge zwei Schrägstriche `//` ein. Alles was danach kommt in der Zeile, wird vom Computer nicht ausgeführt.

überprüfen und den Wert in der Variable switchState speichern. Wenn Spannung auf dem Pin beim Aufruf von `digitalRead()` anliegt, erhält die Variable `switchState` den Wert HIGH (oder 1). Wenn keine Spannung auf dem Stift anliegt, erhält switchState den Wert LOW (oder 0).

if-Statement

Oben hast Du das Wort wenn (if) verwendet, um den Status von etwas zu überprüfen, und zwar den des Tasters. Bei der Programmierung vergleicht ein `if()`-Statement zwei Dinge miteinander, um festzustellen, ob der Vergleich wahr (TRUE) oder falsch (FALSE) ist. Je nach Ausgang des Vergleichs werden dann von Dir festgelegte Aktionen durchgeführt. Wenn Du zwei Dinge in der Programmierung vergleichst, benutzt Du zwei Gleichheitszeichen ==. Wenn Du nur ein Zeichen benutzt, weist Du einen Wert zu, anstatt ihn zu vergleichen.

Baue Dein Raumschiff

`digitalWrite()` ist der Befehl mit dem Du 5V oder 0V an einen Ausgang schicken kannst. `digitalWrite()` braucht dazu zwei Argumente: welcher Pin zu steuern ist, und welchen Wert der Pin annehmen soll, also HIGH oder LOW. Wenn Du innerhalb des `if()`-Statements die roten LEDs ein- und die grüne LED ausschalten möchtest, sollte Dein Programm so aussehen.

Wenn Du Dein Programm jetzt ablaufen lässt, ändern sich die Lichter, sobald Du den Taster betätigst. Das ist schon recht ordentlich, aber Du kannst dem Programm ein wenig mehr Komplexität hinzufügen, für eine noch interessantere Ausgabe.

Du hast dem Arduino gesagt, was er tun soll, wenn der Schalter geöffnet ist. Jetzt definiere was geschieht, wenn der Schalter geschlossen ist. Das `if()`-Statement hat eine optionale `else`-Komponente, in der man etwas ausführen lassen kann, wenn die ursprüngliche Bedingung nicht erfüllt wurde. Da Du überprüft hattest, ob der Schalter auf LOW stand, schreibe die Befehle für den Zustand HIGH nach dem `else`-Statement.

Um die roten LEDs blinken zu lassen, sobald der Taster betätigt wurde, musst Du die LEDs innerhalb des `else`-Statements ein- und ausschalten. Um dies zu tun, ändere das Programm so.

Jetzt blinkt Dein Programm die roten LEDs, wenn der Taster betätigt wird.

Nachdem Du die LEDs in einen bestimmen Zustand versetzt hast, solltest Du den Arduino eine Pause machen lassen, bevor er sie wieder zurücksetzt. Wenn Du nicht wartest, gehen die Lichter so schnell hin und her, dass es scheint als wären die LEDs gedimmt und nicht ein- und ausgeschaltet. Das kommt daher, weil der Arduino seine `loop()`-Schleife tausendfach jede Sekunde durchläuft und die LEDs schneller ein- und ausgeschaltet werden, als wir wahrnehmen können. Die `delay()`-Funktion lässt den Arduino die Ausführung des Programms für eine gewisse Zeitspanne anhalten. `delay()` akzeptiert ein Argument, welches die Anzahl der Millisekunden festlegt, bis im Programm fortgefahren wird. Eine Sekunde hat 1000 Millisekunden. `delay(250)` unterbricht also für eine Viertelsekunde.

```
11  if (switchState == LOW) {
12   // Der Schalter wird nicht gedrückt
13     digitalWrite(3, HIGH); // grüne LED
14     digitalWrite(4, LOW);  // rote LED
15     digitalWrite(5, LOW);  // rote LED
16   }
17   else {  // Der Schalter wird gedrückt
18     digitalWrite(3, LOW);
19     digitalWrite(4, LOW);
20     digitalWrite(5, HIGH);
21     delay(250);  // Warte eine Viertelsekunde
22     // Schalte die LEDs um
23     digitalWrite(4, HIGH);
24     digitalWrite(5, LOW);
25     delay(250); // Warte eine Viertelsekunde
26   }
27 } // Beginne von vorne mit der Schleife
```

Es kann hilfreich sein, den Programmfluss in Pseudocode zu schreiben: eine Beschreibung des Ablaufs in normaler Sprache, jedoch so strukturiert, dass es einfach ist, ein tatsächliches Programm daraus zu schreiben.
In diesem Fall stellst Du fest, ob `switchState` auf `HIGH` ist (was bedeutet, dass der Taster gedrückt wird) oder nicht.
Wenn der Schalter betätigt wird, lässt Du die grüne LED erlöschen und die roten leuchten.
In Pseudocode könnte das Statement wie folgt aussehen:

```
Wenn switchState LOW ist:
  grüne LED einschalten und
  rote LEDs ausschalten;

wenn switchState HIGH ist:
  grüne LED ausschalten und
  rote LEDs einschalten.
```

BENUTZ ES

Sobald Dein Arduino programmiert ist, solltest Du das grüne Licht leuchten sehen. Wenn Du den Taster betätigst, fangen die roten Lichter an zu blinken, und das grüne Licht erlischt. Probiere unterschiedliche Zeiten für die zwei `delay()`-Funktionen aus und betrachte, was mit den Lichtern passiert und wie sich die Antwort des Systems abhängig von der Geschwindigkeit des Blinkens ändert. Wenn Du `delay()` in Deinem Programm aufrufst, wird alle andere Funktionalität gestoppt. Keine Sensor-Messwerte werden gelesen, bis die Zeit abgelaufen ist. Verzögerungen sind häufig nützlich, pass jedoch beim Entwurf Deiner Projekte auf, dass diese nicht unnötig Deine Schnittstelle behindern.

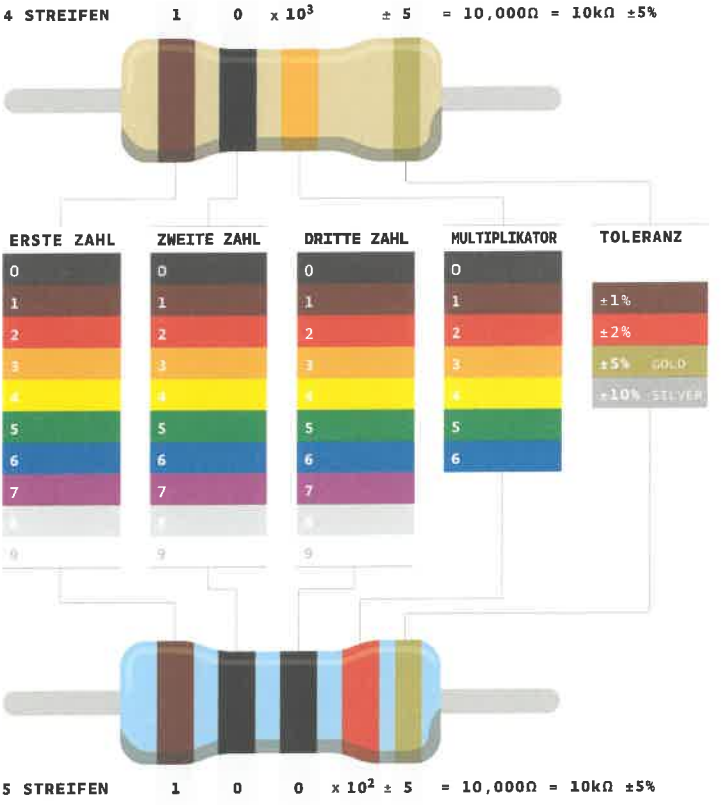
Wie könntest Du Deine roten LEDs blinken lassen, sobald das Programm startet? Wie könntest Du mit Schaltern und LEDs eine größere, komplexere Steuerkonsole für Deine interstellaren Abenteuer bauen?

Wenn Du mit der Umsetzung Deiner Steuerkonsole beginnst, versuche auch immer die Erwartungen zur Bedienung, die andere Personen haben könnten, mit einzubeziehen. Wenn man eine Taste drückt, möchte man dann ein unmittelbares Feedback, oder sollte es eine Verzögerung geben? Versuche Dich also bei der Entwicklung Deines Designs in die Situation eines anderen Benutzers hinein zu versetzen und schaue, ob Deine Erwartungen der Realität entsprechen.

In diesem Projekt hast Du Dein erstes Arduino-Programm erstellt, um das Verhalten von LEDs mit einem Schalter zu steuern. Du hast den Einsatz von Variablen, if()...else-Statements, sowie Funktionen zum Lesen und Schreiben von Ein- und Ausgängen kennengelernt.

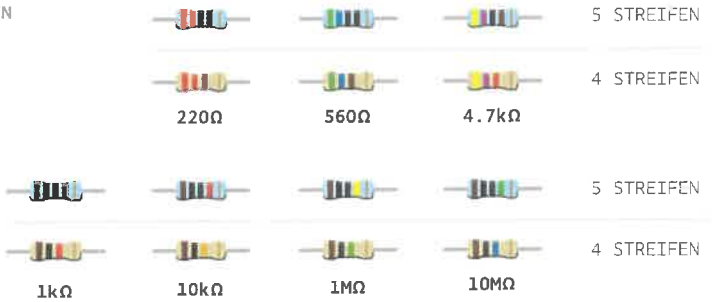
FARBCODES FÜR WIDERSTÄNDE LESEN

Widerstandswerte werden mit farbigen Streifen gekennzeichnet, deren Codierung in den 1920er Jahren entwickelt wurde, als es zu schwierig war, Zahlen auf solch winzige Gegenstände zu schreiben. Jede Farbe entspricht einer Zahl, wie Du unten in der Tabelle siehst. Jeder Widerstand hat entweder 4 oder 5 Streifen. Bei Widerständen mit 4 Streifen repräsentieren die ersten zwei Streifen die ersten zwei Ziffern des Wiederstandwertes und der dritte die Anzahl der folgenden Nullen (d.h. er steht er für die Zehnerpotenz). Der letzte Streifen spezifiziert die Toleranz: im Beispiel unten zeigt Gold an, dass der Widerstandwert 10 kOhm plus oder minus 5 % sein kann.



WIDERSTÄNDE IN DEINEM KIT

Du findest welche mit 4 oder 5 Streifen



03

LED

220-OHM-WIDERSTAND

TEMPERATURFÜHLER

BESTANDTEILE

LOVE-O-METER

MACH AUS DEINEM ARDUINO EIN LIEBESTHERMOMETER. MIT EINEM ANALOGEN EINGANG MISST DU, WIE HEISS DU WIRKLICH BIST!

Entdecke: Analoge Eingänge, Verwendung des seriellen Monitors

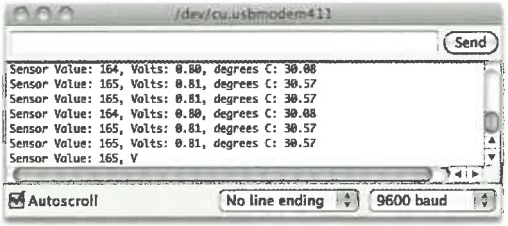
Zeit: **45 MINUTEN**
Niveau: ■□□□□

Basierend auf Projekten: **1, 2**

Obwohl Schalter und Taster großartig sind, gibt es viel mehr in der physikalischen Welt als nur ein und aus. Obwohl der Arduino ein digitales Werkzeug ist, kann er Informationen von analogen Sensoren erfassen, um Dinge wie Temperatur oder Licht zu messen. Dazu verwendest Du den im Arduino integrierten Analog-Digital-Konverter (ADC). Die analogen Eingänge A0-A5 liefern Werte zwischen 0 - 1023, was 0 Volt bis 5 Volt Spannung entspricht.

Du verwendest einen Temperaturfühler zur Messung, wie warm Deine Haut ist. Dieses Bauteil gibt abhängig von der gefühlten Temperatur eine sich ändernde Spannung aus. Es hat drei Pins: einer wird mit der Erdung verbunden, ein weiterer mit Strom, und ein Dritter, der die variable Spannung ausgibt, mit Deinem Arduino. Im Sketch dieses Projekts wirst Du die Messwerte des Sensors auslesen und dazu benutzen, LEDs als Anzeige Deiner Temperatur an- und auszuschalten. Es gibt unterschiedliche Modelle von Temperaturfühlern. Dieses Modell, der TMP36, ist praktisch, weil sich seine ausgehende Spannung direkt proportional zur Temperatur in Grad Celsius verhält.

Die Arduino-IDE verfügt über einen sogenannten ***seriellen Monitor***, mit dem man Ergebnisse vom Mikrocontroller während der Programmausführung überwachen kann. Dadurch erhältst Du fortwährend einen Überblick über den Status Deiner Sensoren und kannst nachvollziehen, was in Deinem Schaltkreis geschieht.



Serieller Monitor
Abbildung 1

BAUE DEN SCHALTKREIS

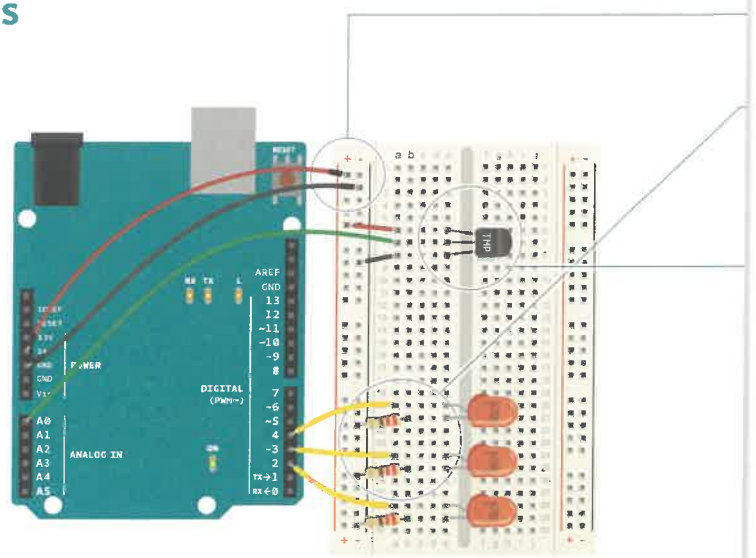


Abbildung 2

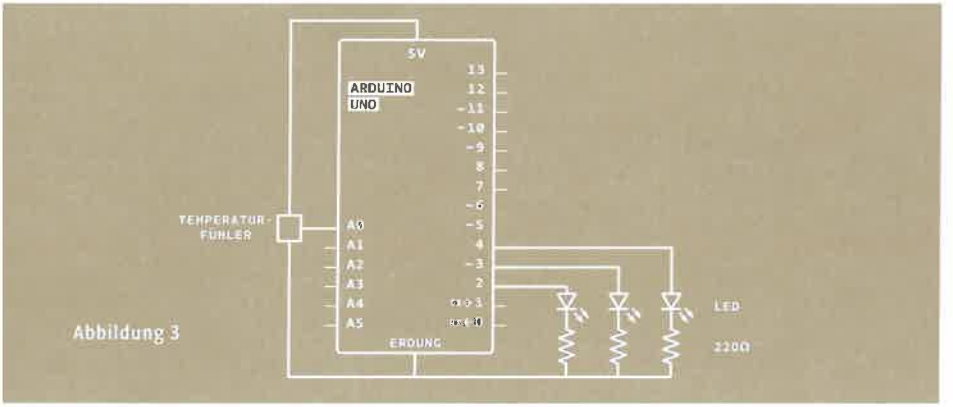


Abbildung 3

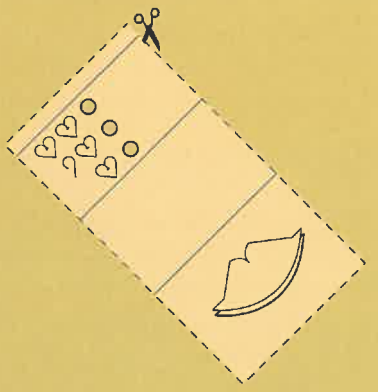
In diesem Projekt musst Du zuerst die Umgebungstemperatur messen, bevor Du fortfahren kannst. Du überprüfst das im Moment noch manuell, aber das ließe sich auch durch eine Kalibrierung erreichen. Man könnte z.B. einen Schalter dazu einsetzen, auf Knopfdruck die Ausgangstemperatur festzuhalten, oder man könnte den Arduino vor der `loop()` ein paar Probemessungen durchführen lassen, um diese als Referenz zu verwenden. Projekt 6 geht hierzu mehr ins Detail. Du kannst Dir auch das Kalibrierungsbeispiel anschauen, das mit der Arduino Software mitgeliefert wird: *arduino.cc/en/calibration*

1. Verbinde Deine Steckplatine mit Strom und Erdung, wie Du es bereits in den vorigen Projekten getan hast.

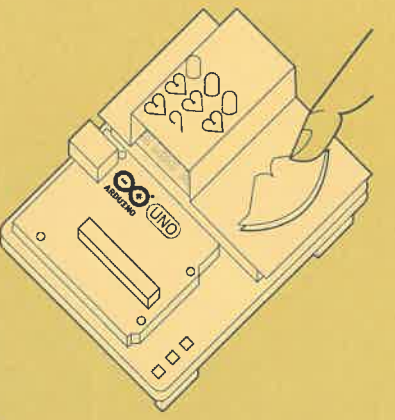
2. Verbinde jeweils die Kathode (kurzer Stift) der verwendeten LEDs über einen 220-Ohm-Widerstand mit der Erdung. Schließe jeweils die Anoden der LEDs an den Pins 2 bis 4 an. Das sind die Anzeigen für das Projekt.

3. Setze den TMP36, wie in Abb. 2 gezeigt, mit der abgerundeten Fläche vom Arduino abgewandt auf die Steckplatine (die Reihenfolge der Pins ist wichtig!). Schließe den linken Pin der flachen Seite am Storm, den rechten an der Erdung an. Schließe den mittleren Pin an Pin A0 vom Arduino an. Das ist der analoge Eingang 0.

Baue eine Schnittstelle für Deinen Sensor, damit Personen damit interagieren können. Ein Papierausschnitt in der Form einer Hand ist ein guter Hinweisgeber. Wenn Du Dein Glück herausfordern möchtest, bastle ein paar Lippen, die man küssen kann, und schau wie dies die Lichter anfeuert! Du kannst die LEDs auch beschriften, um ihnen eine Bedeutung zu geben. Vielleicht bedeutet eine LED, dass Du eiskalt bist, zwei LEDs, dass Du warmherzig und freundlich bist, und drei LEDs, dass Du einen zum Schmelzen bringst!



1. Schneide ein Stück Papier aus, das über die Steckplatine passt. Zeichne ein paar Lippen dort, wo der Sensor sein wird, und schneide Kreise für die LEDs aus.



2. Platziere das ausgeschnittene Papier so auf der Steckplatine, dass die Lippen den Sensor bedecken und die LEDs durch die ausgeschnittenen Kreise heraus gucken. Drück auf die Lippen um zu sehen, wie heiß Du bist!

DAS PROGRAMM

Einige nützliche Konstanten

Konstanten erlauben Dir ähnlich wie Variablen, Dinge im Programm eindeutig zu benennen. Im Gegensatz zu Variablen können sie sich allerdings nicht ändern. Benenne den analogen Eingang, um leicht auf ihn zugreifen zu können, und erstelle eine zweite benannte Konstante, welche die Ausgangstemperatur speichert. Für jede 2 Grad oberhalb dieser Ausgangstemperatur wird eine LED eingeschaltet. Der Dir bereits bekannte Datentyp int wird hier verwendet, um zu wissen, an welchem Pin der Sensor angeschlossen ist. Die Temperatur wird als float Datentyp, d.h. als Gleitkommazahl gespeichert. Zahlen dieser Art haben ein Dezimalkomma und werden verwendet, um Brüche auszudrücken.

Initialisiere die serielle Schnittstelle mit der gewünschten Geschwindigkeit

In den Einstellungen verwendest Du einen neuen Befehl, `Serial.begin()`. Dieser öffnet eine Verbindung zwischen dem Arduino und dem Computer, damit Du die Werte des analogen Eingangs auf dem Computerbildschirm verfolgen kannst. Das Argument `9600` ist die Geschwindigkeit, in der der Arduino kommunizieren wird, nämlich 9600 Bits pro Sekunde. Du wirst den seriellen Monitor der Arduino-IDE dazu verwenden, die von Dir über Deinen Mikrocontroller gesendeten Informationen anzeigen zu lassen. Wenn Du den seriellen Monitor der IDE öffnest, vergewissere Dich, dass die Baudrate auf 9600 eingestellt ist.

Initialisiere die Richtung der digitalen Pins und schalte aus

Als Nächstes kommt eine `for()`-Schleife, um einige Pins als Ausgänge zu setzen. Dies sind die Pins, an denen Du zuvor die LEDs angeschlossen hast. Anstatt jedem einen eigenen Namen zu geben und einzeln über die `pinMode()`-Funktion anzusprechen, kannst Du eine `for()`-Schleife benutzen, um sie alle schnell zu durchlaufen. Das ist ein praktischer Trick, wenn Du bestimmte Befehle mehrmals in einem Programm wiederholen möchtest. Sage Deiner `for()`-Schleife, sie soll der Reihe nach die Pins 2 bis 4 durchlaufen.

Temperaturfühler ablesen

In der `loop()` verwendest Du eine lokale Variable `sensorVal`, um den gelesenen Messwert Deines Sensors zu speichern. Um den Wert des Sensors zu erhalten, rufst Du die Funktion `analogRead()` auf, die ein Argument annimmt: von welchem Pin die Spannung gelesen werden soll. Der Wert, der zwischen 0 und 1023 liegt, repräsentiert die Spannung auf dem Stift.

Schicke die Werte des Temperaturfühlers zum Computer

Die Funktion `Serial.print()` schickt Informationen vom Arduino zum verbundenen Computer. Du kannst diese Informationen in Deinem seriellen Monitor sehen. Wenn Du `Serial.print()` ein Argument in Anführungszeichen gibst, wird der Text angezeigt, den Du eingegeben hast. Wenn Du eine Variable als Argument eingibst, wird ihr Wert ausgegeben.

```
const int sensorPin = A0;
const float baselineTemp = 20.0;
```

```
void setup(){
  Serial.begin(9600); // öffnet eine serielle Schnittstelle
```

```
  for(int pinNumber = 2; pinNumber<5; pinNumber++){
    pinMode(pinNumber,OUTPUT);
    digitalWrite(pinNumber, LOW);
  }
}
```

Tutorial zur for() loop
arduino.cc/for

```
void loop(){
  int sensorVal = analogRead(sensorPin);
```

```
  Serial.print("Sensor Value: ");
  Serial.print(sensorVal);
```

Rechne Sensorwerte in Spannung um

Mit ein wenig Mathematik ist es möglich festzustellen, wie die tatsächliche Spannung auf dem Pin ist. Der Spannungswert liegt zwischen 0 und 5 Volt und kann einen Bruchteil haben (z.B. könnte er 2,5 Volt sein), also musst Du die Werte in einer `float` Variable speichern. Zur Umrechnung teile `sensorVal` durch 1024,0 und multipliziere mit 5,0. Das Ergebnis speicherst Du direkt in einer Variable mit der Bezeichnung Spannung . Der errechnete Wert entspricht der Spannung auf dem Pin.

Wie mit dem Sensorwert lässt Du auch diesen im seriellen Monitor ausgeben.

Rechne die Spannung in Temperatur um und sende den Wert an den Computer

Wenn Du das Datenblatt des Sensors studierst, findest Du Informationen über den Wertebereich der ausgehenden Spannung. Datenblätter sind wie Handbücher für elektronische Bauteile. Sie werden von Ingenieuren für andere Ingenieure geschrieben. Das Datenblatt dieses Sensors besagt, dass 10 Millivolt Veränderung vom Sensor einer Temperaturänderung von einem Grad Celsius entspricht. Es erklärt weiterhin, dass der Sensor auch Temperaturen unterhalb von 0 Grad lesen kann. Deswegen musst Du eine Anpassung vornehmen, um Werte unter dem Gefrierpunkt (0 Grad) zu berücksichtigen. Um also die genaue Temperatur in Grad Celsius zu bestimmen, subtrahiere zunächst 0,5 vom Spannungswert und multipliziere dann mit 100. Speichere diesen errechneten Wert in einer float-Variable mit der Bezeichnung Temperatur.
Nun, da Du die tatsächliche Temperatur kennst, lasse auch sie vom seriellen Monitor ausgeben. Da die Temperaturvariable die Ausgaben in dieser Schleife abschließt, wirst Du einen etwas anderen Befehl verwenden: `Serial.println()`. Dadurch wird nach der Ausgabe im seriellen Monitor eine neue Zeile begonnen, was die Lesbarkeit deutlich verbessert.

Schalte die LEDs bei niedriger Temperatur aus

Mit dem Temperaturwert kannst Du ein `if()...else`-Statement erstellen, um die LEDs ein- oder auszuschalten. Schalte eine LED für jede 2 Grad Temperaturanstieg, relativ zur Ausgangstemperatur, ein. Du wirst nach einem Wertebereich für die Temperaturskala Ausschau halten.

```
14   // rechne den gelesenen ADC-Wert in Spannung um
15   float voltage = (sensorVal/1024.0) * 5.0;
```

```
16   Serial.print(", Volts: ");
17   Serial.print(voltage);
```

```
18   Serial.print(", degrees C: ");
19   // rechne die Spannung in Temperatur in Grad um
20   float temperature = (voltage - .5) * 100;
21   Serial.println(temperature);
```

Starter Kit Datenblätter
arduino.cc/kitdatasheets

```
22   if(temperature < baselineTemp){
23     digitalWrite(2, LOW);
24     digitalWrite(3, LOW);
25     digitalWrite(4, LOW);
```

Schalte eine LED bei niedriger Temperatur ein

Der &&-Operator bedeutet logisches „**und**". Du kannst auf mehrere Zustände prüfen: „wenn die Temperatur 2 Grad höher als die Ausgangstemperatur und weniger als 4 Grad über der Ausgangstemperatur ist."

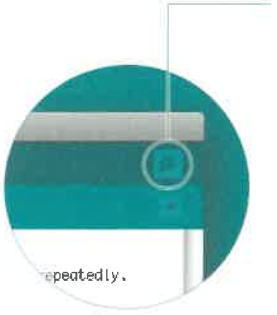
Schalte zwei LEDs bei mittlerer Temperatur ein

Wenn die Temperatur zwischen 4 und 6 Grad über der Ausgangstemperatur ist, schaltet dieser Programmblock auch die LED auf Pin 3 ein.

Schalte drei LEDs bei einer hohen Temperatur ein

Der Analog-Digital-Konverter liest nur mit einer bestimmten Geschwindigkeit, also musst Du eine kleine Verzögerung ganz am Ende der `loop()` einbauen. Wenn Du zu häufig abliest, erscheinen die Werte unregelmäßig.

BENUTZ ES



Nachdem das Programm auf den Arduino hochgeladen wurde, klicke auf das Symbol vom seriellen Monitor. Du solltest eine Abfolge von Werten sehen, die wie folgt formatiert ist:

```
Sensor: 200, Volt: 0,70, Grad C: 17
```

Leg Deine Finger um den Sensor, während er in der Steckplatine steckt, und betrachte, was mit den Werten im seriellen Monitor passiert. Notiere Dir die Temperatur, wenn der Sensor frei an der Außenluft ist.

Schließe den seriellen Monitor und ändere die baselineTemp-Kontstante in Deinem Programm zu dem Wert, den Du gerade aufgeschrieben hast. Lade das Programm nochmals hoch und halte den Sensor in Deinen Fingern. Während die Temperatur steigt, solltest Du sehen, wie eine LED nach der anderen aufleuchtet. Glückwunsch, heiße Sache!

```
  }else if(temperature >= baselineTemp+2 &&
    temperature < baselineTemp+4){
    digitalWrite(2, HIGH);
    digitalWrite(3, LOW);
    digitalWrite(4, LOW);

  }else if(temperature >= baselineTemp+4 &&
    temperature < baselineTemp+6){
    digitalWrite(2, HIGH);
    digitalWrite(3, HIGH);
    digitalWrite(4, LOW);

  }else if(temperature >= baselineTemp+6){
    digitalWrite(2, HIGH);
    digitalWrite(3, HIGH);
    digitalWrite(4, HIGH);

  }
  delay(1);
}
```

Baue eine Schnittstelle, damit zwei Leute ihre Vereinbarkeit miteinander prüfen können. Dabei entscheidest Du, was Vereinbarkeit bedeutet und wie sie zu messen ist. Vielleicht müssen sie Händchen halten und Wärme erzeugen? Vielleicht müssen sie sich umarmen? Was denkst Du?

Du hast gelernt weitere Arten von Eingangswerten auszulesen und dabei analogRead() und den seriellen Monitor verwendet, um Änderungen in Deinem Arduino zu verfolgen. Nun kannst Du viele verschiedene analoge Sensoren und Eingänge lesen.

LED

220-OHM-WIDERSTAND

10-KOHM-WIDERSTAND

PHOTOWIDERSTAND

FARBFILTER

FARBMISCHENDE LAMPE

MIT EINER DREIFARBEN-LED UND DREI PHOTOWIDERSTÄNDEN WIRST DU EINE LAMPE BAUEN, DIE IHRE FARBE ANHAND DER UMGEBUNGSBELEUCHTUNG FLIESSEND VERÄNDERT

Entdecke: Analoge Ausgänge, Werte umwandeln

Zeit: **45 MINUTEN**
Niveau: ■■□□□

Basierend auf Projekten: **1, 2, 3**

Blinkende LEDs sind schön und gut, aber wie wäre es mit Dimmen oder Farben mischen? Man könnte annehmen, dass nur weniger Spannung an einer LED angelegt werden muss, um sie zu dimmen.

Da die Spannung der Ausgänge vom Arduino ausschließlich bei 5V liegt und nicht variiert werden kann, musst Du eine Technik verwenden, die Pulsbreitenmodulation *(PWM)* genannt wird, um LEDs zu dimmen. PWM schaltet einen Ausgang über einen festgelegten Zeitraum sehr schnell zwischen `HIGH` und `LOW` hin und her. Dieser Wechsel geschieht schneller, als es das menschliche Auge wahrnehmen kann. Dieser Vorgang ist vergleichbar zu Filmen, die eine schnelle Abfolge von Einzelbildern zeigen, um so eine Bewegung vorzutäuschen.

Der schnelle Wechsel des Pins zwischen `HIGH` und `LOW` führt also sozusagen zu einer Veränderung der Spannung.
Der Anteil der Zeit, in dem der Pin auf `HIGH` steht, wird Einschaltdauer (englisch: duty cycle) genannt.

Der Arduino Uno hat sechs Pins, die für PWM genutzt werden und durch die Tilde (~) neben ihrer Nummer auf dem Board erkannt werden können (*digitale Pins 3, 5, 6, 9, 10 und 11*).

Als Eingänge in diesem Projekt verwendest Du *Photowiderstände* (Sensoren, die ihren Widerstand abhängig von der auf sie eintreffenden Lichtstärke ändern, auch bekannt als Fotozellen oder lichtabhängige Widerstände). Wenn Du ein Ende des Photowiderstandes an Deinen Arduino anschließt, kannst Du die sich ändernden Werte messen, indem Du die Spannung auf dem Pin ausliest.

BAUE DEN SCHALTKREIS

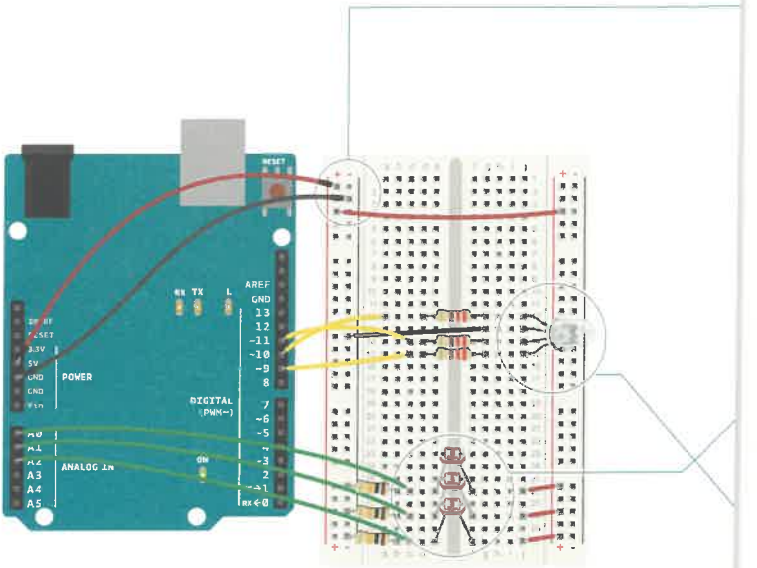


Abbildung 1

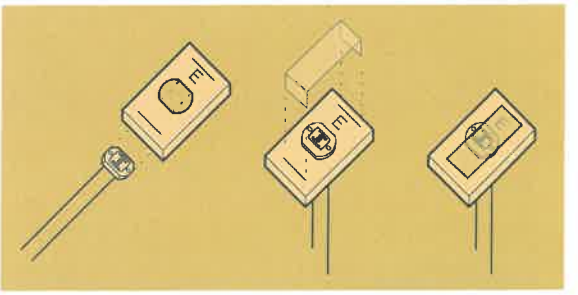


Abbildung 2

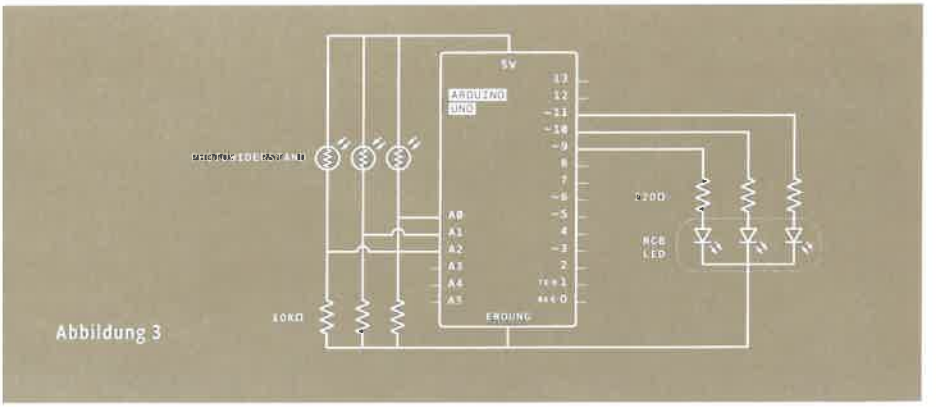


Abbildung 3

1. Verbinde Deine Steckplatine mit Strom und Erdung auf beiden Seiten, genau wie in den vorangegangenen Projekten.

2. Platziere drei Photowiderstände, wie in Abb. 1 dargestellt, über der Mitte der Steckplatine, so dass sie den mittleren Bereich überbrücken. Schließe ein Ende jedes Photowiderstandes am Strom an. Auf der anderen Seite schließt Du sie über einen 10-kOhm-Widerstand an Erdung an. Dieser Widerstand ist in Reihe mit dem Photowiderstand geschaltet, und zusammen bilden sie einen Spannungsteiler. Am Punkt ihres Zusammentreffens ist die Spannung proportional zum Verhältnis ihrer Widerstände, entsprechend dem Ohmschen Gesetz (siehe Projekt 1, um mehr über das Ohmsche Gesetz zu erfahren). Wenn sich der Wert des Photowiderstandes durch das eintreffende Licht ändert, verändert sich auch die Spannung an diesen Knotenpunkten. Verbinde auf der Seite der 10-kOhm-Widerstände die Photowiderstände mit Anschlusskabel mit den analogen Pins 0, 1 und 2.

3. Nimm die drei Farbfilter und setze jeweils einen über einen der Photowiderstände. Setze den roten Farbfilter über den Photowiderstand, der bei A0 angeschlossen ist, den grünen über den, der an A1 angeschlossen ist, und den blauen über den, der an A2 angeschlossen ist. Jeder dieser Filter lässt nur Licht einer spezifischen Wellenlänge zum abgedeckten Sensor, durch. Der rote Filter lässt nur rotes, der grüne nur grünes, und der blaue nur blaues Licht durch. Dadurch kannst Du die relative Farbintensität ermitteln, die auf die Sensoren trifft.

4. Die LED mit 4 Stiften ist eine RGB-LED mit gemeinsamer Kathode. Die LED hat getrennte rote, grüne und blaue Elemente integriert, sowie eine gemeinsame Erdung (die Kathode). Durch die Verursachung eines Spannungsunterschieds zwischen der Kathode und der ausgehenden Spannung der PWM-Pins des Arduino (die über 220-Ohm-Widerstände an den Anoden angeschlossen sind), bringt man die LED dazu, zwischen ihren drei Farben zu wechseln. Wenn Du die LED auf die Steckplatine setzt, schließe den längsten Stift an die Erdung an. Schließe die anderen drei Pins an die digitalen Pins 9, 10 und 11 in Reihe mit 220-Ohm-Widerständen an. Stelle sicher, dass du jeden LED-Stift am richtigen PWM-Pin anschließt, entsprechend der Abbildung links.

DAS PROGRAMM

Nützliche Konstanten

Initialisiere die Konstanten für die Pins, welche Du als Ein- und Ausgänge nutzt, damit Du den Überblick behältst, welcher Sensor zu welcher Farbe der LED gehört. Verwende als Datentyp const int.

Variable zur Speicherung der Sensor-Messwerte sowie der Lichtstärke der LEDs

Erstelle Variablen sowohl für die eingehenden Sensor-Werte, als auch für die Ausgangswerte, die Du benutzen wirst, um die LED zu dimmen. Du kannst den Datentyp int für alle diese Variablen benutzen.

Richtung der digitalen Pins und serielle Schnittstelle einstellen

Starte die serielle Kommunikation mit 9600 bps in `setup()`. Genau wie im vorhergehenden Beispiel verwendest Du sie dazu, die Sensorwerte im seriellen Monitor mit zu verfolgen. Zusätzlich kannst Du die abgebildeten Werte sehen, die Du zur Dimmung der LED verwenden wirst. Setze mit `pinMode()` alle LED-Pins als Ausgänge.

Auslesen der Werte der Lichtsensoren

In der Funktion `loop()` lies die Sensorwerte von A0, A1 und A2 mit `analogRead()` und speichere sie in den entsprechenden Variablen. Setze ein kurzes `delay()` zwischen den einzelnen `analogRead()`, da der ADC eine Millisekunde benötigt, um seine Arbeit zu erledigen.

Sensor-Messwerte dem Computer melden

Gib die Sensorwerte in einer Zeile aus. „`\t`" ist gleichbedeutend mit dem Betätigen der „TAB"-Taste auf der Tastatur.

```
const int greenLEDPin = 9;
const int redLEDPin = 11;
const int blueLEDPin = 10;

const int redSensorPin = A0;
const int greenSensorPin = A1;
const int blueSensorPin = A2;


int redValue = 0;
int greenValue = 0;
int blueValue = 0;

int redSensorValue = 0;
int greenSensorValue = 0;
int blueSensorValue = 0;


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

  pinMode(greenLEDPin,OUTPUT);
  pinMode(redLEDPin,OUTPUT);
  pinMode(blueLEDPin,OUTPUT);
}


void loop() {
  redSensorValue = analogRead(redSensorPin);
  delay(5);
  greenSensorValue = analogRead(greenSensorPin);
  delay(5);
  blueSensorValue = analogRead(blueSensorPin);

  Serial.print("Raw Sensor Values \t Red: ");
  Serial.print(redSensorValue);
  Serial.print("\t Green: ");
  Serial.print(greenSensorValue);
  Serial.print("\t Blue: ");
  Serial.println(blueSensorValue);
```

Umwandeln der Sensor-Messwerte

Die Funktion zur Veränderung der Helligkeit der LEDs mittels PWM wird `analogRead()` genannt. Sie benötigt zwei Argumente: der Pin, auf den geschrieben werden soll, sowie einen Wert zwischen 0-255, der die Einschaltdauer des Pins festlegt. Ein Wert von 255 setzt den Pin durchgehend auf `HIGH`, was die angeschlossene LED so hell wie möglich leuchten lässt. Ein Wert von 127 schaltet den Pin zur Hälfte der Zeit auf `HIGH`, was die LED dimmt. 0 würde den Pin durchgehend auf `LOW` setzen, so dass auch die LED ausgeschaltet wäre. Um den Sensor-Messwert von einem Wert zwischen 0-1023 in einen Wert zwischen 0-255 für `analogRead()` umzuwandeln, teile den Sensor-Messwert durch 4.

Errechnete LED-Stärken melden

Gib die umgewandelten Werte in einer Zeile aus.

LED-Lichtstärken festlegen

BENUTZ ES

Öffne den seriellen Monitor sobald Du Deinen Arduino programmiert und angeschlossen hast. Die LED wird vermutlich weißgrau leuchten, je nach vorherrschender Lichtfarbe in Deinem Raum. Betrachte die Werte, die von den Sensoren im seriellen Monitor angezeigt werden. Sofern Du in einer Umgebung mit stabiler Beleuchtung bist, sollte die Zahl ziemlich gleichbleibend sein.

Schalte das Licht im Raum aus und schau, was mit den Werten der Sensoren passiert. Beleuchte jeden der Sensoren einzeln mit einer Taschenlampe und beobachte, wie sich die Werte im seriellen Monitor und die Farbe der LED ändern. Wenn die Photowiderstände mit einem Farbfilter abgedeckt werden, reagieren sie nur auf Licht einer bestimmten Wellenlänge. Das gibt Dir die Möglichkeit, jede der Farben unabhängig voneinander zu verändern.

```
  redValue = redSensorValue/4;
  greenValue = greenSensorValue/4;
  blueValue = blueSensorValue/4;
```

```
  Serial.print("Mapped Sensor Values \t Red: ");
  Serial.print(redValue);
  Serial.print("\t Green: ");
  Serial.print(greenValue);
  Serial.print("\t Blue: ");
  Serial.println(blueValue);
```

```
  analogWrite(redLEDPin, redValue);
  analogWrite(greenLEDPin, greenValue);
  analogWrite(blueLEDPin, blueValue);
}
```

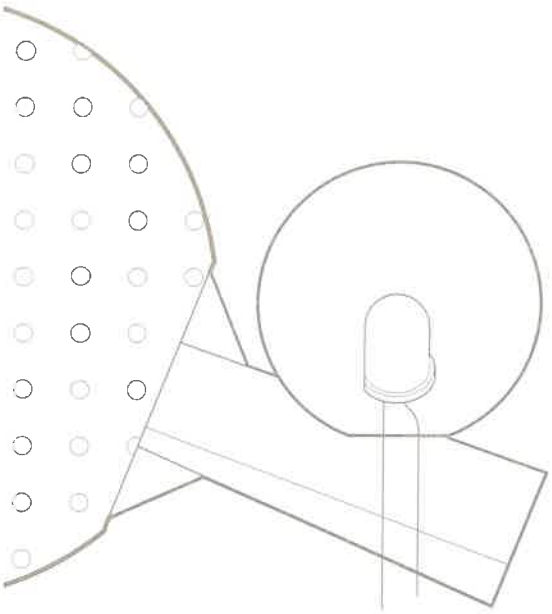
Vielleicht hast Du bemerkt, dass die Ausgabe der Photowiderstände nicht den ganzen Bereich von 0 bis 1023 abdeckt. Das ist für dieses Projekt in Ordnung, aber für eine ausführlichere Erklärung, wie man einen Auslesebereich kalibriert, schau bei Projekt 6 nach.

Du hast vermutlich auch festgestellt, dass das Dimmen der LED nicht linear ist. Wenn die LED ungefähr bei halber Helligkeit ist, scheint sie nicht mehr viel heller zu werden. Das liegt daran, dass unsere Augen Helligkeit nicht linear wahrnehmen. Die Helligkeit des Lichts hängt nicht nur vom Pegel ab, den Du mit `analogRead()` festsetzt, sondern auch vom Abstand des Lichts zum Diffusor, vom Abstand Deines Auges zum Licht, und von der Helligkeit des Lichts relativ zu anderen Lichtquellen im Raum.

Wie könnten wir das verwenden, um zu wissen, ob es draußen schön ist, während Du drinnen arbeitest? Welche anderen Sensoren könntest Du sonst noch verwenden, um die Farbe der LED zu steuern?

Die einzelne LED ist zwar ganz ordentlich, aber für eine richtige Lampe fehlt noch ein bisschen. Es gibt eine Vielzahl von Möglichkeiten, mit denen Du Licht zerstreuen kannst, damit es herkömmlichen Glühlampen ähnlicher wird. Ein Pingpong-Ball mit einem Loch, durch das Du die LED hinein stecken kannst, wäre ein guter Diffusor. Weitere Optionen wären die LED mit lichtdurchlässigem Klebstoff zu bedecken oder das Abschmirgeln der LED-Oberfläche. Egal welche Variante Du wählst, durch die Zerstreuung wirst Du immer ein wenig Helligkeit einbüßen, aber dafür wird es vermutlich deutlich schöner aussehen.

Nun kannst Du kontrollieren, wie hell oder dunkel etwas sein soll, und bist nicht länger darauf beschränkt, Lichter nur ein- und auszuschalten. Das Konzept der PWM hast Du mit der Funktion analogWrite() umgesetzt, mit der die Einschaltdauer der Bauteile an den Pins 3, 5, 6, 9, 10 oder 11 gesteuert werden kann.

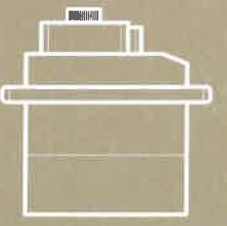


Der zugeschnittene Pingpong-Ball um die LED unterzubringen.

Abbildung 4

05

POTENTIOMETER



SERVOMOTOR

BEWEGUNGSARM

100-µF-KONDENSATOR

STIFTLEISTE (3 PINS)

BESTANDTEILE

STIMMUNGSBAROMETER

BENUTZE EINEN SERVOMOTOR UM EIN MECHANISCHES MESSINSTRUMENT ZU BAUEN, WELCHES ANZEIGT, IN WELCHER STIMMUNG DU HEUTE BIST

Entdecke: Werte umwandeln, Servomotoren, mitgelieferte Bibliotheken benutzen

Zeit: **1 STUNDE**
Niveau: ■■□□□

Basierend auf Projekten: **1, 2, 3, 4**

Servomotoren sind solche Motoren, die sich nicht im Kreis drehen, sondern an eine bestimmte Position bewegen und dort solange bleiben, bis ein neuer Befehl eingeht. Servos rotieren für gewöhnlich nur um 180 Grad (die Hälfte eines Kreises). Wenn Du einen dieser Motoren mit einer kleinen Kartonbastelei kombinierst, kannst Du andere Personen wissen lassen, ob sie Dich gerade um Hilfe für ihr nächstes Projekt bitten können, oder ob sie das besser sein lassen sollten.

Ähnlich wie bei den PWM-Impulsen, die Du im vorigen Projekt für die Lampe benutzt hast, erwarten Servomotoren eine Anzahl von Impulsen, die ihnen mitteilen, zu welchem Winkel sie sich bewegen sollen. Die Impulse haben immer den gleichen zeitlichen Abstand, aber die Breite schwankt zwischen 1000 und 2000 Mikrosekunden. Auch wenn es natürlich möglich ist, ein Programm zur Erzeugung dieser Impulse zu schreiben, wird die Arduino Software mit einer Bibliothek geliefert, welche die Steuerung des Motors erheblich vereinfacht.

Weil der Servo nur 180 Grad dreht und Dein analoger Eingang von 0-1023 liefert, musst Du eine Funktion verwenden, die `map()` genannt wird, um die Grösse der Werte zu ändern, die vom Potentiometer kommen.

Einer der tollen Vorzüge der Arduino-Gemeinschaft sind die begabten Leute, die durch zusätzliche Software die Funktionalität des Arduino erweitern. Es ist jedem möglich, solche sogenannten Bibliotheken zu schreiben. Es gibt Bibliotheken für eine Vielzahl von Sensoren, Aktoren und weiterer Teile, die Benutzer der Community zur Verfügung gestellt haben. Eine Software-Bibliothek erweitert die Funktionalität einer Programmierumgebung. Die Arduino-Software kommt mit einer Vielzahl von Bibliotheken, die für das Arbeiten mit Hardware oder Daten nützlich sind. Eine dieser mitgelieferten Bibliotheken wurde für die Verwendung von Servomotoren erstellt. Um auf diese in Deinem Programm zugreifen zu können, wirst Du die Bibliothek mit all ihren Funktionen importieren.

BAUE DEN SCHALTKREIS

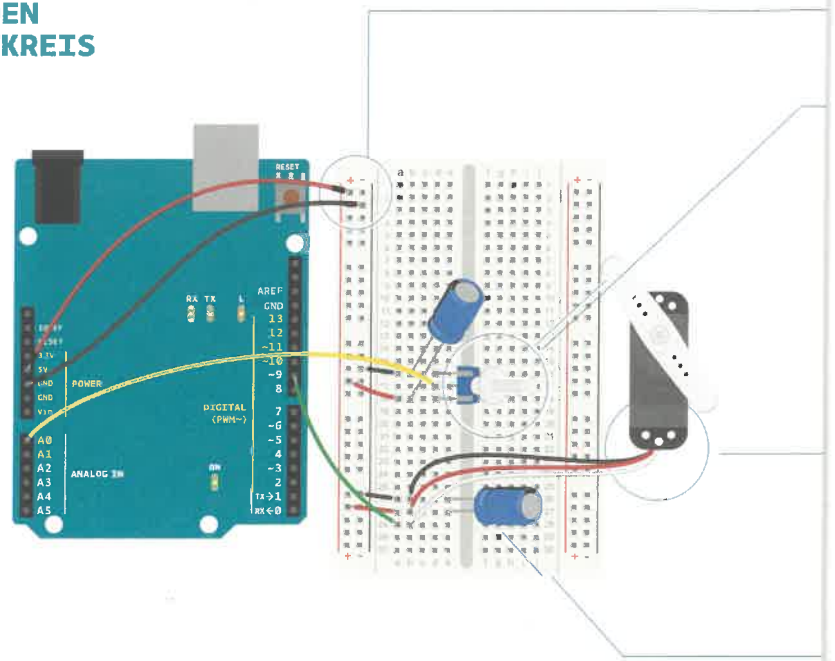


Abbildung 1

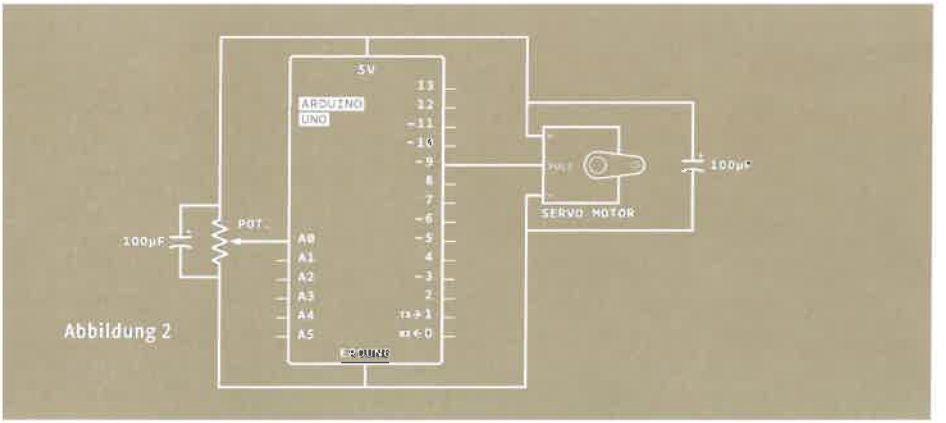


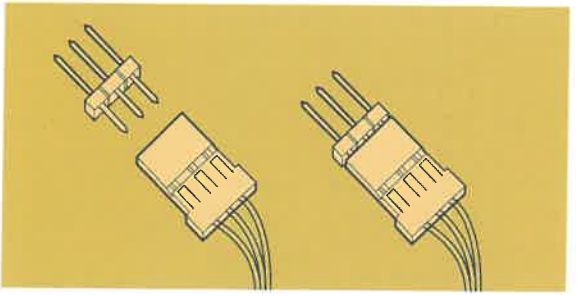
Abbildung 2

1. Schließe 5V und Erdung vom Arduino an einer Seite der Steckplatine an.

2. Setze ein Potentiometer auf die Steckplatine und verbinde eine Seite mit 5V und die andere mit Erdung. Ein Potentiometer ist ein Spannungsteiler. Durch Drehen des Knopfes änderst Du das Spannungverhältnis zwischen dem mittleren Pin und der Stromquelle. Diese Veränderung kann man an einem analogen Eingang auslesen. Verbinde dazu den mittleren Pin mit dem analogen Pin 0. Das wird die Position Deines Servomotors steuern.

3. Aus dem Servo kommen drei Kabel. Eines ist für Strom (rot), eines ist für Erdung (schwarz) und das Dritte (weiß) ist das Steuerkabel, das die Informationen vom Arduino erhält. Stecke drei Pins in die Buchsen der Servoleitungen (siehe Abb. 3). Stecke die Stiftleiste in Deine Steckplatine, sodass jeder Pin in einer anderen Reihe ist. Schließe 5V am roten Kabel, die Erdung am schwarzen Kabel und Pin 9 am weißen Kabel an.

4. Wenn ein Servomotor sich zu bewegen beginnt, verbraucht er mehr Strom, als wenn er bereits in Bewegung wäre. Dies verursacht einen Spannungsabfall auf Deiner Platine. Durch die Platzierung eines 100-µF-Kondensators zwischen Strom und Erdung direkt neben den Pins, wie in Abb. 1 gezeigt, kannst Du alle auftretenden Spannungsänderungen glätten. Du kannst auch einen Kondensator zwischen Strom und Erdung bei Deinem Potentiometer platzieren. Diese werden Entkopplungskondensatoren genannt, weil sie die durch andere Bauteile verursachten Veränderungen im Schaltkreis, verringern oder entkoppeln. Gib acht, dass Du die Kathode an die Erdung (die Seite mit dem schwarzen Streifen) und die Anode am Strom anschließt. Wenn Du die Kondensatoren verkehrt herum setzt, können sie explodieren.



Weil Dein Servomotor Buchsen hat, musst Du erst eine Stiftleiste einstecken, um ihn an der Steckplatine anschließen zu können

Abbildung 3

DAS PROGRAMM

Bibliothek importieren

Um die Servo-Bibliothek nutzen zu können, musst Du sie zuerst importieren. Dadurch werden die zusätzlichen Funktionen der Bibliothek für Deinen Sketch verfügbar gemacht.

Servo-Objekt erstellen

Um Dich auf den Servo beziehen zu können, musst Du eine benannte Instanz der Servo-Bibliothek in einer Variable erzeugen. Das bezeichnet man als ***Objekt***. Diese eindeutig benannte Instanz verfügt über alle Funktionen und Fähigkeiten, die die Servo-Bibliothek bereitstellt. Wann immer Du von nun an im Programm `myServo` referenzierst, sprichst Du das Servo-Objekt an.

Variablendeklaration

Erstelle eine benannte Konstante für den mit dem Potentiometer verbundenen Pin, sowie Variablen zur Speicherung der Werte des Analogeingangs und des Winkels, zu dem der Servo bewegt werden soll.

Servo-Objekt mit dem Arduino-Pin verknüpfen, serielle Schnittstelle initialisieren

In der Funktion `setup()` musst Du dem Arduino sagen, an welchem Pin der Servomotor angeschlossen ist.

Richte eine serielle Verbindung ein, um die Werte des Potentiometers und ihre Umwandlung in Winkel vom Servomotor mit verfolgen zu können.

Potentiometer-Werte auslesen

In der Funktion `loop()` liest Du den Analogeingang und gibst die Werte im seriellen Monitor aus.

Umwandlung von Werten vom Potentiometer zu Werten für den Servo

Um einen für den Servomotor verwertbaren Wert aus Deinem Analogeingang zu erhalten, lässt sich am einfachsten die Funktion `map()` einsetzen. Diese praktische Funktion skaliert Zahlen, d.h. sie wandelt Werte von einem Wertebereich in einen anderen um. In diesem Fall werden die Werte von 0-1023 zu Werten von 0-179 umgewandelt. Sie benötigt fünf Argumente: der Wert, der skaliert werden soll (hier potVal), der Mindestwert der Eingabe (0), der Höchstwert des Eingabe (1023), der Mindestwert des Ausgabe (0) und der Höchstwert des Ausgabe (179). Speichere diesen neuen Wert in der angle Variable und lasse ihn am seriellen Monitor ausgeben.

Den Servo drehen

Jetzt ist es Zeit, den Servo zu bewegen. Der Befehl `myServo.write()` bewegt den Motor zu dem von dir spezifizierten Winkel. Am Ende der Funktion `loop()` setzt Du eine Pause, damit der Servo Zeit hat, sich an seine neue Position zu bewegen.

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

Beachte, dass #include Befehle kein Semikolon am Ende der Zeile haben.

```
Servo myServo;
```

```
int const potPin = A0;
int potVal;
int angle;
```

```
void setup() {
  myServo.attach(9);

  Serial.begin(9600);
}
```

```
void loop() {
  potVal = analogRead(potPin);
  Serial.print("potVal: ");
  Serial.print(potVal);
```

```
  angle = map(potVal, 0, 1023, 0, 179);
  Serial.print(", angle: ");
  Serial.println(angle);
```

```
  myServo.write(angle);
  delay(15);
}
```

VERWENDE ES

Sobald Du Deinen Arduino programmiert und eingeschaltet hast, öffne den seriellen Monitor.
Du solltest eine Abfolge von Werten sehen, die diesen ähnlich sind:

```
potVal : 1023, Winkel : 179
potVal : 1023, Winkel : 179
```

Wenn Du das Potentiometer drehst, solltest Du eine Veränderung der Zahlen sehen können. Vor allem aber sollte sich Dein Servomotor zu einer neuen Position bewegen. Beachte dabei das Verhältnis zwischen dem Wert von potVal, dem Winkel im seriellen Monitor und der Position des Servos. Du solltest beim Drehen des Knopfes konsistente Ergebnisse erhalten.

Ein Vorteil von Potentiometern als Analogeingänge ist, dass sie Dir einen breiten Wertebereich zwischen 0 und 1023 liefern. Dadurch eignen sie sich besonders für die Testung von Projekten, die Analogeingänge verwenden.

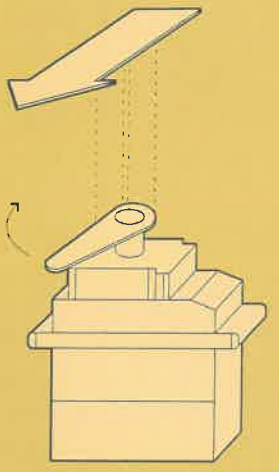
Servomotoren sind gewöhnliche Motoren mit ein paar Zahnrädern und Schaltkreisen im Innern. Die interne Mechanik liefert ein Feedback an den Schaltkreis, so dass er sich fortwährend seiner Position bewusst ist. Auch wenn es scheint als wäre ein Servo in seiner Bewegung stark eingeschränkt, kann man dennoch eine Vielzahl unterschiedlichster Bewegungen durch ein paar zusätzliche Mechanismen hervorrufen. Es gibt etliche Ressourcen, die diese Mechanismen detailliert beschreiben, wie z.B. ***robives.com/mechs*** oder das Buch ***Making Things Move*** von ***Dustyn Roberts***.

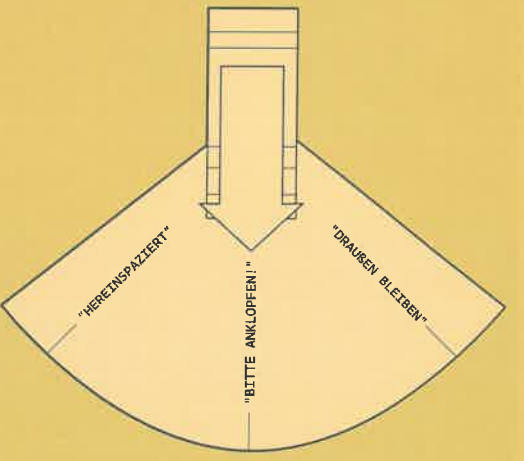
Das Potentiometer ist nicht der einzige Sensor, den Du zur Steuerung eines Servos benutzen kannst. Fällt Dir ein weiterer Sensor ein, mit dem Du so eine Anzeige bauen könntest (d.h. mit einem Pfeil, der auf unterschiedliche Hinweise zeigt)? Wie würde dies mit Temperaturen funktionieren (wie im Love-O-Meter)? Könntest Du die Tageszeit mit einem Photowiderstand erkennen? Welche Relevanz hätte die Umwandlung von Werten mit diesen Sensoren?

Servomotoren können leicht vom Arduino mit einer Bibliothek gesteuert werden. Eine Bibliothek ist eine Sammlung von Funktionen, die eine Programmierumgebung erweitern. Je nach Anwendung ist es manchmal nötig, Werte von einer Skala zu einer anderen umzuwandeln.

Nachdem Du nun Bewegung kennen gelernt hast, wird es Zeit, andere Personen wissen zu lassen, ob sie Dich mit ihren Projekten behelligen dürfen, oder ob man Dich bei der Planung Deiner nächsten Kreation besser in Frieden lässt.

Schneide mit einer Schere ein Stück Pappe in der Form eines Pfeils aus. Bewege Deinen Servo auf 90 Grad (überprüfe den Winkelwert im seriellen Monitor, wenn Du Dir nicht sicher bist). Klebe den Pfeil auf, sodass er in die gleiche Richtung wie der Motor zeigt. Durch Drehen des Potentiometers solltest Du den Pfeil nun um 180 Grad rotieren können. Nimm ein Stück Papier, das größer als der Pfeil und der Servo ist, und zeichne einen Halbkreis darauf. An einem Ende des Halbkreises schreibst Du „Draußen bleiben". Am anderen Ende schreibst Du „Hereinspaziert". Schreibe „Bitte anklopfen!" in die Mitte des Kreisbogens. Setze den Servo mit dem Pfeil auf das Papier. Glückwunsch, jetzt kannst Du anderen Leuten zeigen, wie ausgelastet Du gerade durch Deine Projekte bist!





❶

Befestige einen Papppfeil am Servo-Arm.

❷

Entwerfe eine Papierunterlage und lege sie unter den Servo.

06

PIEZO

PHOTOWIDERSTAND

10-KOHM-WIDERSTAND

BESTANDTEILE

LICHT-THEREMIN

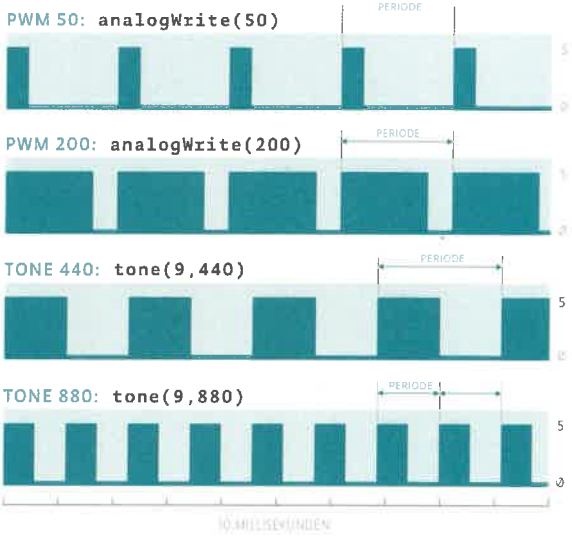
ZEIT, UM ETWAS LÄRM ZU MACHEN!
MIT EINEM PHOTOWIDERSTAND UND EINEM PIEZOELEMENT BAUST DU EIN LICHTBASIERTES THEREMIN

Entdecke: Klänge mit der Funktion tone() erstellen. Kalibrierung von analogen Sensoren.

Zeit: **45 MINUTEN**
Niveau: ■■■■□

Basierend auf Projekten: **1, 2, 3, 4**

Ein *Theremin* ist ein Instrument, das Töne basierend auf Handbewegungen in seiner Nähe formt.Vielleicht hast Du es sogar schon mal in einem Horrorfilm gehört. In Relation zu zwei Antennen erkennt das Theremin die Händeposition des Musikers über Kapazitätsveränderung. Diese Antennen werden an analoge Schaltkreise angeschlossen, die den Ton erzeugen. Eine Antenne steuert die Frequenz des Tons, die andere die Lautstärke. Auch wenn der Arduino die geheimnisvollen Töne dieses Instruments nicht exakt reproduzieren kann, ist es dennoch möglich sie mit der Funktion `tone()` nachzuahmen. Abb. 1 verdeutlicht den Unterschied zwischen `analogWrite()` und `tone()` bei der Generierung von Impulsen. Dadurch kann sich ein Wandler, wie z.B. ein Lautsprecher oder ein Piezoelement, mit unterschiedlichen Geschwindigkeiten hin und her bewegen.



Das Signal ist die meiste Zeit aus, aber die Frequenz ist die gleiche wie bei PWM 200.

Die Spannung liegt die meiste Zeit an, aber die Frequenz ist die gleiche wie bei PWM 50.

Die Einschaltdauer ist 50% (an die halbe Zeit, aus die andere), aber die Frequenz ändert sich.

Die gleiche Einschaltdauer wie bei Tone 440, aber die doppelte Frequenz.

Abbildung 1

Anstatt die Kapazität mit dem Arduino zu messen, verwendest Du einen Photowiderstand, um die Lichtstärke zu ermitteln. Indem Du Deine Hände über den Sensor bewegst, veränderst Du den Lichteinfall auf den Photowiderstand, so wie in Projekt 4. Die Spannungsänderung auf dem analogen Pin wird die zu spielende Frequenz bestimmen.

Wie schon in Projekt 4 baust Du für die Verbindung zwischen Photowiderstand und Arduino einen Schaltkreis mit Spannungsteiler. Vermutlich hattest Du zuvor bemerkt, dass bei dieser Art von Schaltkreis die mit `analogRead()` ausgelesenen Werte nicht den ganzen Bereich von 0 bis 1023 abgedeckt haben. Der an Erdung angeschlossene, festgelegte Widerstand beschränkt das untere Ende des Wertebereichs, und die Helligkeit Deines Lichts begrenzt das obere Ende. Um dieses Mal auf den größtmöglichen Wertebereich für den Theremin zugreifen zu können, kalibrierst Du den Sensor auch die niedrigsten und höchsten Messwerte auszugeben und wandelst sie dann mit der Funktion `map()` zu Tonfrequenzen um. Das hat außerdem den Vorteil, dass die Sensor-Messwerte justiert werden, wann immer Du Deinen Schaltkreis in eine neue Umgebung mit veränderter Beleuchtung bringst.

Ein *Piezo* ist ein kleines Element, das vibriert, wenn es Elektrizität empfängt. Durch seine Bewegung verdrängt es Luft, und es entstehen Schallwellen.

BAUE DEN SCHALTKREIS

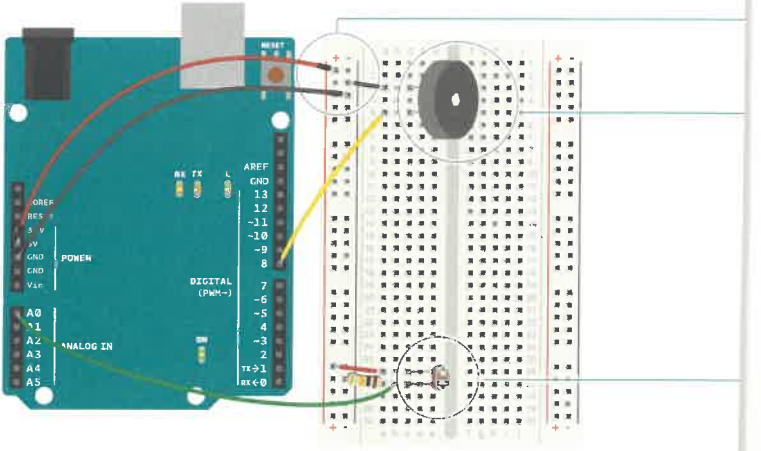


Abbildung 2

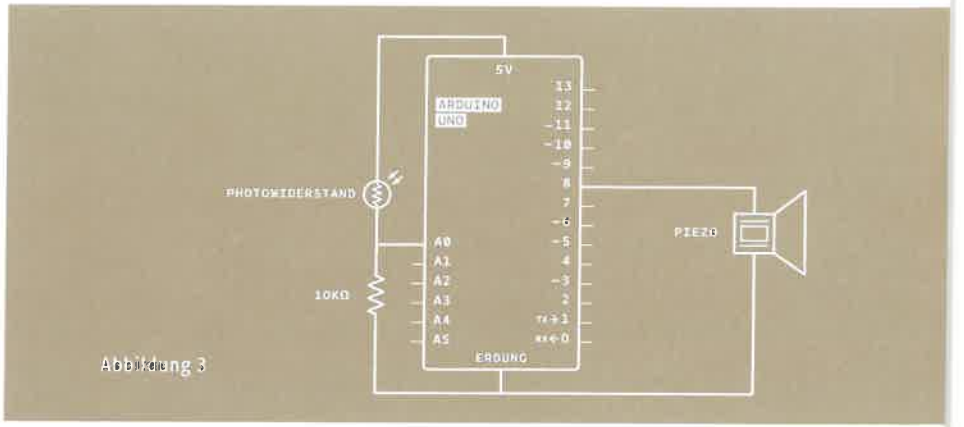


Abbildung 3

Traditionelle Theremine können die Frequenz und die Lautstärke des Tons steuern. In diesem Beispiel wirst Du aber nur die Frequenz steuern. Auch wenn Du die Lautstärke mit dem Arduino nicht steuern kannst, könnte man die am Lautsprecher eingehende Spannung manuell einstellen. Was geschieht, wenn Du ein Potentiometer mit Pin 8 und dem Piezo in Reihe schaltest? Wie wäre es mit einem weiteren Photowiderstand?

1. Auf Deiner Steckplatine schließt Du wieder die äußeren Buslinien an Strom und Erdung an.

2. Nimm das Piezoelement und schließe ein Ende an Erdung, das andere am digitalen Pin 8 des Arduino an.

3. Setze den Photowiderstand auf die Steckplatine und verbinde eine Seite mit 5V, und die andere Seite mit dem analogen Pin 0 des Arduino, sowie mit Erdung über einen 10-kOhm-Widerstand. Dieser Schaltkreis ist der gleiche, wie der Spannungsteiler in Projekt 4.

DAS PROGRAMM

Variablen für die Sensor-Kalibrierung

Erstelle eine Variable, um den mit `analogRead()` abgerufenen Messwert vom Photowiderstand zu speichern. Als Nächstes erstelle Variablen für die theoretisch höchsten bzw. niedrigsten Sensorwerte. Der Ausgangswert der `sensorLow`-Variable beträgt 1023, der der `sensorHigh`-Variable 0. Bei der ersten Ausführung des Programms werden diese theoretischen Werte mit den Sensor-Messwerten abgeglichen, um so das tatsächliche Maximum bzw. Minimum zu finden.

Konstante für Deine Kalibrierungsanzeige

Erstelle eine Konstante mit der Bezeichnung `ledPin`. Sie wird als Hinweis für das Ende der Sensor-Kalibrierung dienen. Für dieses Projekt benutzen wir die in der Platine integrierte LED, die an Pin 13 angeschlossen ist.

Richtung und Zustand des digitalen Pins einstellen.

Ändere im `setup()` den `pinMode()` von ledPin zu `OUTPUT` und schalte die LED ein, indem Du sie auf `HIGH` setzt.

Kalibrierung mit while()-Schleife

Die folgenden Schritte kalibrieren den höchsten und niedrigsten Wert des Sensors. Du wirst ein `while()`-Statement aufrufen, das für 5 Sekunden eine Schleife ausführt. `while()`-Schleifen laufen solange, bis eine bestimmte Bedingung erfüllt ist. Hier verwenden wir die Funktion `millis()`, um die Zeit abzufragen. `millis()` meldet, wie viel Zeit seit dem letzten Einschalten oder Neustart des Arduino verstrichen ist.

Abgleich der Sensor-Werte zur Kalibrierung

In der Schleife wird wiederholt der Sensor-Messwert abgerufen; wenn der Wert kleiner als `sensorLow` ist (anfangs 1023), wird `sensorLow` mit dem derzeitigen Messwert aktualisiert. Wenn der Messwert unabhängig davongrößer als `sensorHigh` ist (anfangs 0), wird auch sensorHigh mit diesem Wert aktualisiert.

Anzeige des Endes der Kalibrierung

Nach 5 Sekunden endet die while()-Schleife. Schalte die an Pin 13 angeschlossene LED aus, indem Du sie auf `LOW` setzt. Die soeben ermittelten sensorHigh und sensorLow Werte wirst Du zur Umrechnung des Frequenzbereichs im Hauptprogramm einsetzen.

```
int sensorValue;
int sensorLow = 1023;
int sensorHigh = 0;
```

```
const int ledPin = 13;
```

```
void setup() {

  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, HIGH);
```

```
  while (millis() < 5000) {
```

while()
arduino.cc/while

```
    sensorValue = analogRead(A0);
    if (sensorValue > sensorHigh) {
      sensorHigh = sensorValue;
    }
    if (sensorValue < sensorLow) {
      sensorLow = sensorValue;
    }
  }
```

```
  digitalWrite(ledPin, LOW);
}
```

Sensor-Werte lesen und speichern

In `loop()` speicherst Du den von A0 abgerufenen Messwert in `sensorValue`.

Sensor-Wert in eine Frequenz umwandeln

Erstelle eine Variable mit der Benennung `pitch`, auf die der umgerechnete Wert der Variable `sensorValue` abgelegt wird. `sensorLow` und `sensorHigh` bilden die Begrenzung der eingehenden Werte. Als Startpunkt für die ausgehenden Werte probieren wir 50 bis 4000, was den Frequenzbereich der vom Arduino erzeugten Töne festlegt.

Frequenz abspielen

Nun rufe die Funktion `tone()` auf, um einen Ton zu erzeugen. Sie verlangt drei Argumente: an welchem Pin der Ton zu spielen ist (in diesem Fall Pin 8), welche Frequenz gespielt wird (festgelegt durch die `pitch`-Variable) und wie lange der Ton zu spielen ist (probiere erstmal 20 Millisekunden).

Mit `delay()` setzen wir eine kurze Pause von 10 ms, um den Tönen etwas Zeit zum Klingen zu geben.

BENUTZ ES

Für die Sensor-Kalibrierung hast Du ein Zeitfenster von 5 Sekunden nach dem Start des Arduinos. Dazu bewege Deine Hand über dem Photowiderstand hin und her, um die einfallende Lichtmenge zu ändern. Je genauer die Bewegungen denen entsprechen, die Du beim Spielen des Instruments machen wirst, desto besser ist die Kalibrierung.

Nach 5 Sekunden ist die Kalibrierung abgeschlossen und die LED auf dem Arduino erlischt. Sobald das geschieht, solltest Du Geräusche vom Piezo hören! In Abhängigkeit von der auf den Photowiderstand einfallenden Lichtmenge wird sich die vom Piezoelement gespielte Frequenz ändern.

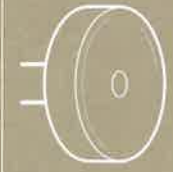
```
19 void loop() {
20   sensorValue = analogRead(A0);

21   int pitch =
       map(sensorValue,sensorLow,sensorHigh, 50, 4000);

22   tone(8,pitch,20);

23   delay(10);
24 }
```

Zur Ermittlung der Tonhöhe (pitch Variable) haben wir den Bereich der ausgehenden Werte in der Funktion `map()` ziemlich breit festgelegt. Um Deinen eigenen Musikstil zu finden, kannst Du mit diesen Frequenzwerten herumspielen.

Die Funktion `tone()` ähnelt in ihrer Funktionsweise dem PWM von `analogWrite()`, jedoch mit einem wichtigen Unterschied. Bei `analogWrite()` ist die Frequenz festgelegt; Du veränderst nur das Verhältnis der Impulse, um die Einschaltdauer zu variieren. Mit `tone()` sendest Du auch Impulse, aber Du änderst ihre Frequenz. `tone()` pulsiert immer mit einer Einschaltdauer von 50% (die Hälfte der Zeit ist der Pin auf HIGH, die andere Hälfte auf LOW).

Mithilfe von tone() kannst Du unterschiedliche Frequenzen erzeugen, wenn ein Lautsprecher oder ein Piezoelement pulsiert wird. Wenn Du Sensoren in einem Spannungsteiler verwendest, wirst Du vermutlich nicht den vollen Bereich der möglichen Messwerte von 0-1023 erhalten. Durch Kalibrierung ist es möglich, die eingehenden Sensorwerte in einen geeigneten Wertebereich zu übertragen.

SCHALTER



PIEZO

10-KOHM-WIDERSTAND

1-MOHM-WIDERSTAND

220-OHM-WIDERSTAND

KEYBOARD

MIT EIN PAAR WIDERSTÄNDEN UND TASTEN WIRST DU EIN KLEINES KEYBOARD BAUEN

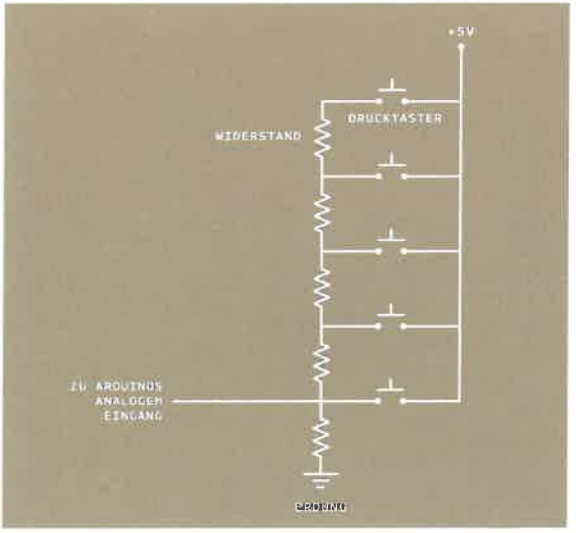
Entdecke: Widerstandsmatrix, *Arrays*

Zeit: **45 MINUTEN**
Niveau: ■■■■■

Basierend auf Projekten: **1, 2, 3, 4, 6**

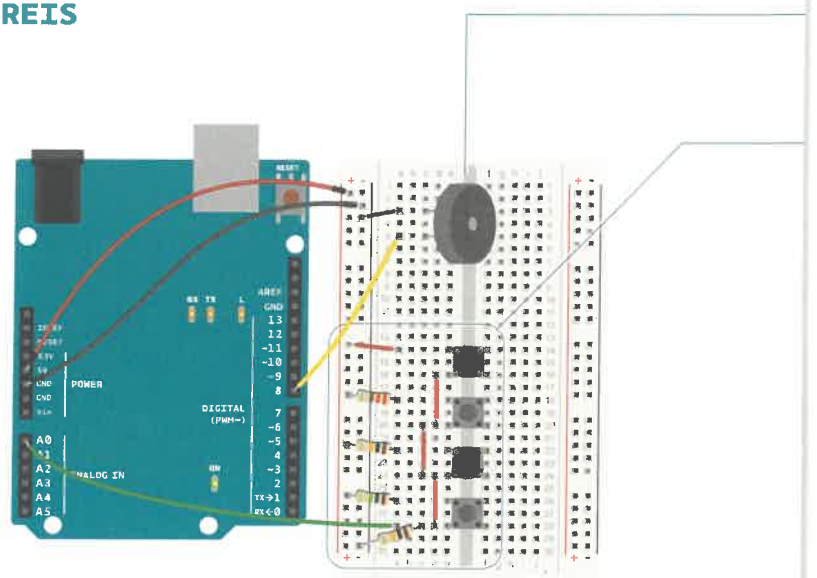
Anstatt etliche Taster für unterschiedliche Töne an ebenso vielen digitalen Eingängen anzuschließen, werden wir in diesem Projekt eine sogenannte Widerstandsmatrix bauen.

Dadurch kann man mehrere Schalter mit einem einzelnen analogen Eingang auslesen, so dass man eine ganze Menge an digitalen Eingängen spart. Du wirst mehrere parallel geschaltete Taster mit dem Analogeingang 0 verbinden. Die meisten von ihnen werden über einen Widerstand am Strom angeschlossen. Je nach gedrückter Taste wird eine andere Spannung an den Eingangspin geleitet. Wenn Du zwei Tasten gleichzeitig drückst, erhältst Du eine eindeutige Spannungssignatur durch das Verhältnis zwischen den zwei parallelen Widerständen.



Eine Widerstandsmatrix und fünf Schalter als Analogeingang.
Abbildung 1

BAUE DEN SCHALTKREIS



Die Anordnung von Widerständen und Schaltern, die einen gemeinsamen Analogeingang ansprechen, wird Widerstandsmatrix genannt.
Abbildung 2

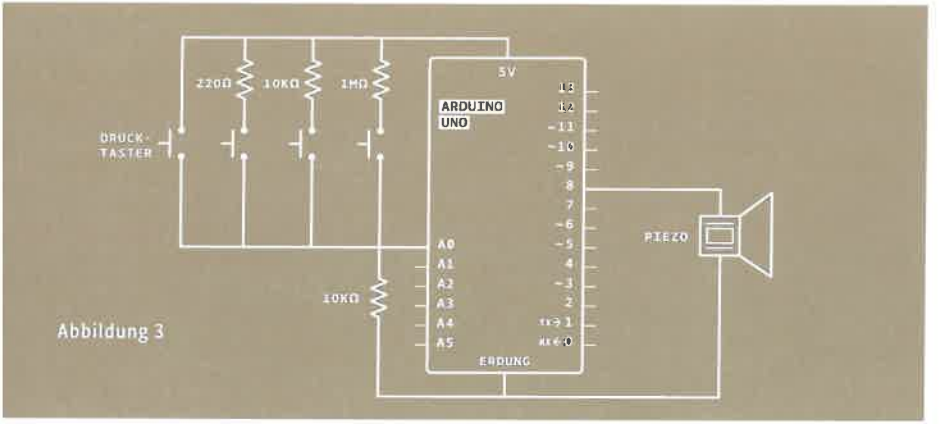
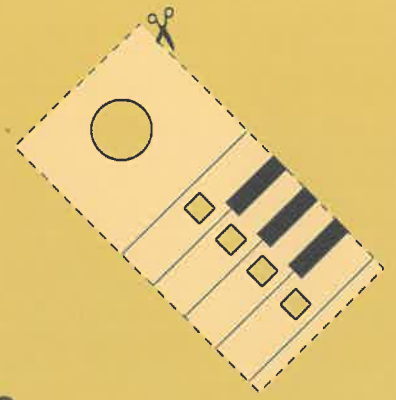


Abbildung 3

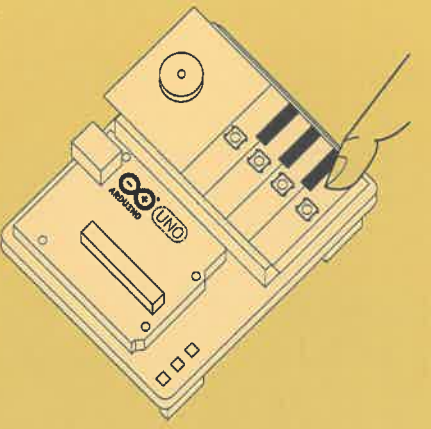
1. Schließe Deine Steckplatine an Strom und Erdung wie in den vorigen Projekten an. Schließe ein Ende des Piezoelements an Erdung, und das andere an Pin 8 Deines Arduino an.

2. Arrangiere die Schalter auf der Steckplatine nach dem Vorbild des Schaltplans. Die Anordnung von Widerständen und Schaltern, die einen gemeinsamen Analogeingang ansprechen, wird Widerstandsmatrix genannt. Schließe den ersten Schalter direkt am Strom an. Schließe den zweiten, dritten und vierten Schalter über einen 220-Ohm-, 10-kOhm- bzw. 1-mOhm-Widerstand am Strom an. Verbinde die Ausgänge aller Schalter an einem Knotenpunkt und schließe diesen über einen 10-kOhm-Widerstand an Erdung, sowie am Analogeingang 0an. Jeder davon dient als Spannungsteiler.

Überlege Dir ein Gehäuse für das Keyboard. Alte analoge Synthesizer hatten überall herausguckende Drahte, aber Dein Keyboard ist schnittig und digital. Bereite ein kleines Stück Pappe vor, an dem Du herumschneiden kannst, um Deine Tasten unterzubringen. Beschrifte die Tasten, damit Du jeweils weißt, welche Töne durch sie ausgelöst werden.



1. Schneide Löcher in ein Stück Pappe für die vier Tasten und das Piezoelement. Bemale es, damit es wie eine Klaviertastatur aussieht



2. Lege die Pappe über die Tasten und das Piezoelement. Erfreue Dich an Deiner Kreation!

DAS PROGRAMM

Das Array

In diesem Programm wirst Du in einer Liste die Frequenzen vermerken, die beim Drücken der jeweiligen Tasten gespielt werden sollen. Du kannst zunächst mit den Frequenzen für das mittlere C, D, E und F (262 Hz, 294 Hz, 330 Hz und 349 Hz) beginnen.
Hierzu benötigst Du einen neuen Variablen-Typ mit der Bezeichnung Array. Ein Array verwendet man, um verschiedene zusammenhängende Werte, wie z.B. die Frequenzen einer Tonleiter, unter einem gemeinsamen Namen zu speichern. Sie sind ein nützliches Werkzeug, um effizient Informationen abzurufen. Ein Array deklariert man wie auch andere Variablen mit einem Namen, jedoch zusätzlich gefolgt von einem Paar eckiger Klammern: []. Nach dem Gleichheitszeichen setzt Du die Werte dann in geschweifte Klammern. Um die Werte eines Arrays abzurufen oder zu verändern, wird auf das einzelne Element mit dem Namen des Arrays und seinem Index verwiesen. Der Index bezieht sich auf die Reihenfolge der Werte, als das Array erstellt wurde. Das erste Element der Eingabe erhält im Array den Index 0, das zweite den Index 1, und so weiter.

Frequenz-Array erstellen

Erstelle ein Array mit den oben genannten Frequenzwerten für die vier Noten. Weil Du es vor `setup()` deklarierst, ist das Array eine globale Variable, d.h. sie steht allen Funktionen zur Verfügung.

Serielle Kommunikation starten

In `setup()` starte die serielle Kommunikation mit dem Computer.

Analogen Wert lesen und an den seriellen Monitor senden

In `loop()` deklariere eine lokale Variable, welche die von Pin A0 gelesenen Werte speichert. Weil jeder Schalter mit einem anderen Widerstandswert am Strom angeschlossen ist, wird auch jeder einen anderen zugehörigen Wert besitzen. Um die Werte zum Computer zu senden, füge die Zeile `Serial.println(keyVal)` ein.

if()...else-Statement für die Ermittlung der zu spielenden Note

Mit einem `if()...else`-Statement kannst Du jedem Wert einen anderen Ton zuweisen. Die Werte in diesem Beispielprogramm sind grob geschätzt für diese Widerstandsgrößen. Da alle Widerstände eine gewisse Fehlertoleranz aufweisen, könnten diese Werte für Dich unter Umständen nicht ideal sein. Verwende die Informationen vom seriellen Monitor für etwaige Anpassungen.

```
int buttons[6];
// Erstelle ein Array mit 6 Integer

int buttons[0] = 2;
// Gib dem ersten Element des Arrays den Wert 2
```

```
int notes[] = {262,294,330,349};
```

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

```
void loop() {
  int keyVal = analogRead(A0);
  Serial.println(keyVal);
```

```
  if(keyVal == 1023){
    tone(8, notes[0]);
  }
```

Noten entsprechend des analogen Wertes abspielen

Rufe die Funktion tone() nach jedem `if()`-Statement auf. Der Verweis zum Array bestimmt, welche Frequenz gespielt werden soll. Wenn der von A0 ausgelesene Wert eines Deiner `if()`-Statements erfüllt, wird der Arduino einen Ton abspielen. Durch leichtes „Rauschen" im Schaltkreis können die Werte beim Betätigen einer Taste ein wenig schwanken. Um diesen Fluktuationen Rechnung zu tragen, kann man einen kleinen Wertebereich statt einzelner Werte abfragen. Mithilfe des „&&"-Operators kannst Du mehrere Aussagen überprüfen.

Wenn Du die erste Taste drückst, wird notes[0] gespielt. Beim Drücken der zweiten notes[1], und bei der dritten notes[2]. In Situationen wie diesen sind Arrays wirklich praktisch.

Ton-Wiedergabe beenden, wenn keine Taste gedrückt wird

Da an einem einzelnen Pin nicht mehrere Frequenzen gleichzeitig gespielt werden können, hörst Du trotz Drückens mehrerer Tasten nur einen Ton.
Um die Wiedergabe zu beenden, wenn keine Taste gedrückt wird, rufe die Funktion `noTone()` mit der entsprechenden Pin-Nummer auf.

BENUTZ ES

Wenn die Werte Deiner Widerstände gut mit denen im Beispielprogramm übereinstimmen, solltest Du Töne vom Piezoelement als Antwort auf Deinen Tastendruck zu hören bekommen. Falls nicht, kannst Du im seriellen Monitor überprüfen, ob die jeweiligen Tasten Werte erzeugen, die einem der Bereiche in den `if()...else`-Statements entsprechen. Erhöhe die Wertebereiche ein wenig, wenn die Töne zu stottern scheinen.
Betrachte die Werte im seriellen Monitor, wenn Du mehrere Tasten gleichzeitig drückst. Du kannst diese neuen Werte dazu verwenden, noch mehr Töne zu spielen. Probiere verschiedene Frequenzen aus, um Dein musikalisches Spektrum zu erweitern. Du kannst die Frequenzen der Tonleiter auf dieser Seite nachschlagen: *arduino.cc/en/frequencies*

Wärest Du in der Lage ein dynamischeres Instrument zu bauen, wenn Du die Schalter und Widerstandsmatrix mit analogen Sensoren ersetzen würdest? Du könntest z.B. die zusätzliche Information einsetzen, um die Notendauer zu verändern oder, wie im Theremin-Projekt, fließende Töne zu erzeugen.

```
  else if(keyVal >= 990 && keyVal <= 1010){
    tone(8, notes[1]);
  }
  else if(keyVal >= 505 && keyVal <= 515){
    tone(8, notes[2]);
  }
  else if(keyVal >= 5 && keyVal <= 10){
    tone(8, notes[3]);
  }
```

```
  else{
    noTone(8);
  }
}
```

Soviel Spaß das Erzeugen von Tönen mit der Funktion `tone()` auch macht, so hat sie doch einige Einschränkungen. Sie kann z.B. nur Rechteckwellen generieren, keine glatten Sinuswellen oder Dreiecke. Und Rechteckwellen sehen nun nicht gerade wie Wellen aus. Wenn Du Dich an Abbildung 1 in Projekt 6 erinnerst, ist das nur eine Reihe von Ein- und Aus-Impulsen.

Bedenke bei der Gründung Deiner Band folgendes: nur ein Ton kann gleichzeitig gespielt werden und `tone()` behindert `analogWrite()` auf den Pins 3 und 11.

Arrays sind nützlich, um ähnliche Informationen zusammen zu gruppieren; der Zugriff geschieht über Indexe, die sich auf ein bestimmtes Element beziehen. Eine Widerstandsmatrix mit einem analogen Eingang zu verbinden, ist eine einfache Möglichkeit, die Anzahl verfügbarer digitaler Eingänge in einem System zu erhöhen.

SCHALTER

LED

10-KOHM-WIDERSTAND

220-OHM-WIDERSTAND

DIGITALE SANDUHR

IN DIESEM PROJEKT BAUST DU EINE DIGITALE SANDUHR, DIE ALLE 10 MINUTEN EINE LED EINSCHALTET. ERFAHRE, WIE LANGE DU AN DEINEN PROJEKTEN ARBEITEST, INDEM DU DEN INTEGRIERTEN TIMER DES ARDUINO VERWENDEST

Entdecke: *Datentyp Long, Timer erstellen*

Zeit: **30 MINUTEN**
Niveau: ■■■□□

Basierend auf Projekten: **1, 2, 3, 4**

Bislang hast Du für die Erzeugung von Zeitintervallen die Funktion delay() verwendet. Die ist praktisch, aber leider auch ein wenig limitiert, denn wenn der Arduino delay() aufruft, pausiert er für die Dauer der Verzögerung alle Vorgänge. D.h. also auch, dass während der Wartezeit keine Aus- oder Eingänge angesprochen werden. Außerdem eignen sich Verzögerungen nicht dazu, die Zeit zu messen. Wenn Du eine bestimmte Aktion z.B. alle 10 Sekunden ausführen möchtest, wäre eine alles unterbrechende Verzögerung für diesen Zeitraum ziemlich lästig.

Die `millis()`-Funktion schafft hier Abhilfe. Sie merkt sich die Zeit in Millisekunden, die Dein Arduino seit seinem letzten Start in Betrieb war. Du hast sie schon zuvor in Projekt 6 verwendet, als Du den Zeitraum für die Kalibrierung festgelegt hast.

Bis jetzt hast Du Variablen als `int` deklariert. Ein `int` (Integer) ist eine 16-Bit-Zahl; sie speichert Werte zwischen -32.768 und 32.767. Das sind zwar große Zahlen, aber wenn der Arduino 1000 Mal pro Sekunde mit `millis()` zählt, wäre nach weniger als einer Minute der Speicher voll. Der Datentyp `Long` speichert eine 32-Bit-Zahl (zwischen -2.147.483.648 und 2.147.483.647). Da Zeit nicht rückwärts läuft und wir somit keine negativen Zahlen erhalten können, wird die Variable für die Werte von `millis()` als `unsigned Long` deklariert. Wenn ein Datentyp unsigned ist (d.h. ohne Vorzeichen), akzeptiert er nur positive Werte. Dadurch kann man doppelt so hoch zählen, d.h. ein `unsigned Long` kann bis zu 4.294.967.295 groß sein. Das wäre genug Platz für `millis()`, um einen Zeitraum von fast 50 Tagen zu speichern. Durch den Vergleich des derzeitigen Wertes von `millis()` mit einem zuvor gespeicherten Wert, kannst Du die verstrichene Zeit berechnen. Wenn Du Deine Sanduhr umdrehst, wird ein Kippschalter umgelegt und ein neuer LED-Zyklus beginnt. Der Kippschalter arbeitet wie auch andere Schalter als ein Ein-/Aus-Sensor. Hier wirst Du ihn als Digitaleingang einsetzen. Das Besondere an Neigungsschaltern ist, dass sie ihre Lage feststellen

können. Für gewöhnlich haben sie eine Metallkugel in einer kleinen Vertiefung innerhalb des Gehäuses. Wenn sie auf eine Seite gekippt werden, rollt die Kugel zu dieser Seite des Hohlraums und schließt den Schalter, indem sie die zwei Kabel verbindet, die zu Deiner Steckplatine führen. Mit sechs LEDs wird Deine Sanduhr eine Stunde laufen.

BAUE DEN SCHALTKREIS

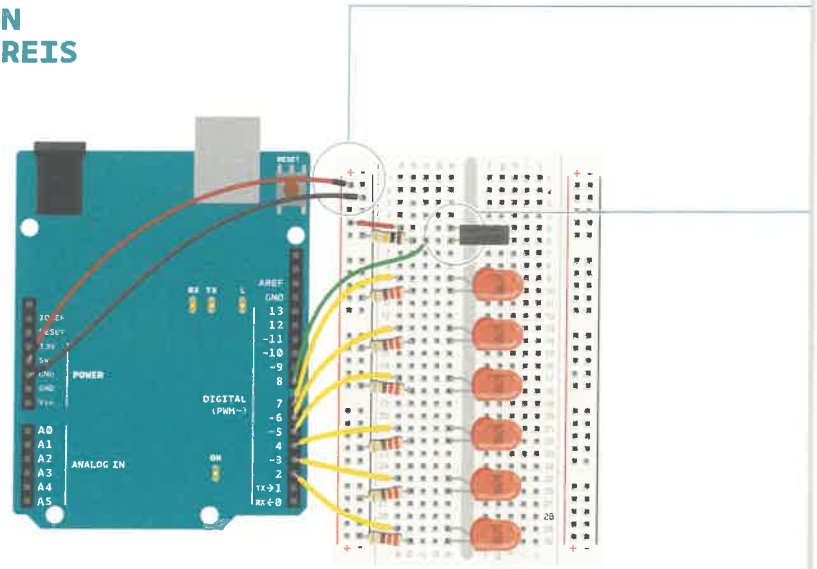


Abbildung 1

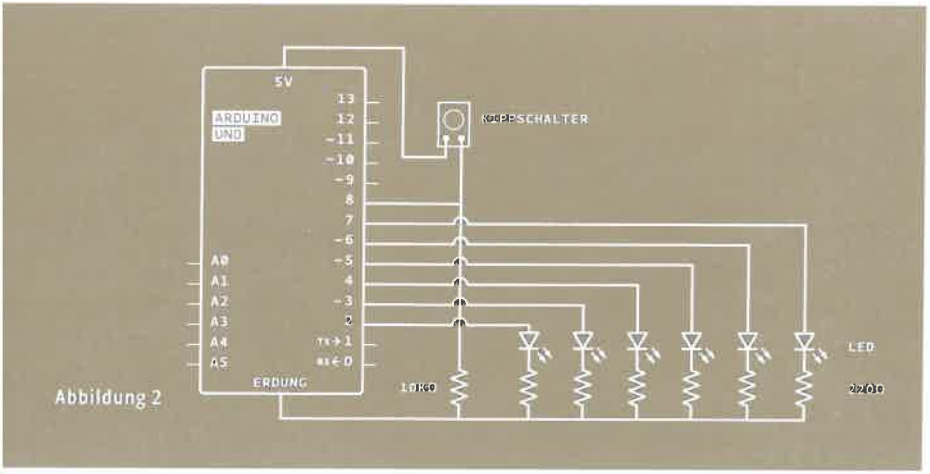


Abbildung 2

1. Schließe Strom und Erdung an Deiner Steckplatine an.

2. Verbinde die Anoden (längerer Stift) der sechs LEDs mit den digitalen Pins 2-7. Schließe die LEDs über 220-Ohm-Widerstände an Erdung an.

3. Schließe einen Stift des Kippschalters an 5V und den anderen Stift über einen 10-kOhm-Widerstand an Erdung an. Schließe ihren Knotenpunkt am digitalen Pin 8 an.

Du brauchst hierfür nicht unbedingt eine PC-Verbindung. Du könntest eine tragbare Version erstellen, indem Du eine Halterung aus Pappe oder Styropor baust und den Arduino mit einer Batterie versorgst. Als Abdeckung bietet sich eine Zahlenanzeige neben den Lichtern an.

Kippschalter sind tolle und günstige Bauteile zur Bestimmung der Lage. Eine andere Art von Neigungssensor wäre z.B. ein Beschleunigungsmesser. Der liefert wesentlich mehr Informationen, ist aber auch deutlich teurer. Wenn man nur wissen möchte, ob etwas oben oder unten ist, reichen Kippschalter völlig.

DAS PROGRAMM

Konstante für Kippschalter

Du wirst in Deinem Programm einige globale Variablen brauchen. Beginne mit einer Konstante switchPin für den mit dem Kippschalter verbundenen Pin.

Variable für Zeit

Ersteile eine Variable vom Typ `unsigned Long`. Diese wird sich den Zeitpunkt merken, an dem eine LED zuletzt geändert wurde.

Variablen für Ein- und Ausgänge

Erstelle eine Variable für den aktuellen, und eine weitere für den vorangegangenen Zustand des Schalters. Du wirst sie dazu verwenden, die Schalterposition von einem Durchgang der Schleife zum nächsten zu vergleichen.
Erstelle eine Variable mit der Bezeichnung led um mitzuzählen, welche LED als Nächste eingeschaltet werden soll. Den Anfang macht Pin 2.

Variable für das Zeitintervall zwischen den Ereignissen

Abschließend erstellst Du eine Variable für die Pause zwischen den LED-Einschaltungen vom Datentyp `Long`. In 10 Minuten (die Zeit zwischen den LED-Einschaltungen) vergehen 600.000 Millisekunden. Wenn Du die Verzögerung zwischen den LEDs anpassen möchtest, musst Du diese Zahl ändern.

Richtung der digitalen Pins

Mithilfe einer `for()`-Schleife werden in der Funktion `setup()` die LED-Pins 2-7 mit nur 3 Programmzeilen als Ausgänge definiert. Darüber hinaus legst Du switchPin als Eingang fest.

Überprüfung der Startzeit

Zu Beginn von `loop()` rufst Du mit der Funktion `millis()` die seit dem Einschalten verstrichene Zeit ab und speicherst sie in einer lokalen Variablen mit der Bezeichnung `currentTime`.

Auswertung der verstrichenen Zeit

Mit einem `if()`-Statement überprüfst Du, ob es an der Zeit ist, die nächste LED einzuschalten. Dazu subtrahierst Du `currentTime` von `previousTime` und vergleichst das Resultat mit dem festgelegten Zeitintervall. Sobald 600.000 Millisekunden (10 Minuten) verstrichen sind, überschreibst Du previousTime mit dem Wert von `currentTime`.

```
const int switchPin = 8;

unsigned long previousTime = 0;

int switchState = 0;
int prevSwitchState = 0;

int led = 2;

long interval = 600000;

void setup() {
  for(int x = 2;x<8;x++){
    pinMode(x, OUTPUT);
  }

  pinMode(switchPin, INPUT);
}

void loop(){
  unsigned long currentTime = millis();

  if(currentTime - previousTime > interval) {
    previousTime = currentTime;
```

LED einschalten und weiterzählen

`previousTime` zeigt an, wann zuletzt eine LED eingeschaltet wurde. Nachdem Du `previousTime` überschrieben hast, schaltest Du eine neue LED ein und erhöhst die Variable led auf die nächste Pin-Nummer, so dass nach Ablauf des folgenden Intervalls die nächste LED eingeschaltet wird.

Überprüfen, ob alle LEDs eingeschaltet sind

Ein weiteres `if()`-Statement an dieser Stelle soll den Einschaltstatus der letzten LED an Pin 7 überprüfen. Was geschehen soll, sobald eine volle Stunde verstrichen ist, wirst Du später festlegen.

Zustand des Schalters abrufen

Nachdem Du die Zeit überprüft hast, bringst Du den Zustand des Schalters in Erfahrung. Dazu schreibst Du den abgerufenen Wert in die Variable `switchState`.

Zurücksetzen der Variablen auf ihre Standardwerte

Überprüfe mit einem `if()`-Statement, ob sich die Position des Schalters verändert hat. Hier wird mit `!=` (ungleich) auf einen Unterschied zwischen `switchState` und `prevSwitchState` geprüft. Wenn sich die Werte nicht gleichen, schaltest Du alle LEDs aus, setzt die Variable led auf den ersten Pin zurück und beginnst den Timer von vorne, indem Du `previousTime` auf `currentTime` setzt.

Aktueller Zustand wird zu vorigem Zustand

Am Ende von `loop()` speicherst Du den Schalterzustand in `prevSwitchState`, damit Du ihn im nächsten Durchlauf mit dem dann neu ausgelesenen Wert von `switchState` vergleichen kannst.

BENUTZ ES

Sobald Du die Platine programmiert hast, überprüfe die Zeit auf einer Uhr. Sobald 10 Minuten vergangen sind, sollte sich die erste LED einschalten. Das sollte sich alle 10 Minuten fortsetzen, bis nach einer Stunde alle 6 LEDs eingeschaltet sind. Wenn Du den Schaltkreis umdrehst und so den Zustand des Neigungsschalters änderst, werden die LEDs ausgeschaltet und der Timer beginnt von neuem.

```
    digitalWrite(led, HIGH);
    led++;


    if(led == 7){
    }
  }

  switchState = digitalRead(switchPin);


  if(switchState != prevSwitchState){
    for(int x = 2;x<8;x++){
      digitalWrite(x, LOW);
    }

    led = 2;
    previousTime = currentTime;
  }

  prevSwitchState = switchState;
}
```

Nach Ablauf einer Stunde sind alle sechs LEDs eingeschaltet und bleiben an, sofern der Schalter nicht betätigt wird. Fällt Dir vielleicht eine Möglichkeit ein, Dich auf diesen Umstand hinzuweisen? Geräusche oder blinkende Lichter wären z.B. gute Indikatoren. Mit der Variable led könnte man überprüfen, ob alle Lichter eingeschaltet sind. Die zuvor freigelassene Bedingung wäre eine gute Stelle, um einen Hinweis zu programmieren.. Anders als eine richtige Sanduhr gehen die LEDs je nach Lage des Schalters entweder hoch oder runter. Wie könntest Du die switchState-Variable zur Festlegung der Richtung die Lichter einsetzen?

Die Funktion millis() kannst Du zur Messung des Zeitabstands zwischen zwei Ereignissen verwenden. Da die von der Funktion generierten Zahlen schnell zu groß werden, um sie in einem int zu speichern, solltest Du stattdessen den Datentyp unsigned Long wählen.

09

BESTANDTEILE

MOTORISIERTES WINDRAD

BRINGE DEN ARDUINO DAZU, MIT EINEM MOTOR EIN BUNTES WINDRAD ZU DREHEN

Entdecke: *Transistoren, hohe Strom-/Spannungslasten*

Zeit: **45 MINUTEN**
Niveau: ■■■□□

Basierend auf Projekten: **1, 2, 3, 4**

Für den Arduino ist die Steuerung eines Motors eine größere Herausforderung als LEDs. Erstens benötigen Motoren mehr Strom als die Ausgänge des Arduino liefern können. Zweitens können Motoren durch sogenannte Induktion ihren eigenen Strom erzeugen, was ohne ausreichende Vorkehrungen Deinen Schaltkreis beschädigen könnte. Da Motoren durch die Bewegung von Objekten Deinen Projekten erst die richtige Würze geben, sind sie jedoch die Komplikationen wert!

Objekte in Bewegung zu setzen benötigt eine Menge Energie. Motoren erfordern für gewöhnlich mehr Strom, als der Arduino zur Verfügung stellen kann. Einige Motoren benötigen außerdem eine höhere Spannung. Ein Motor zieht so viel Strom wie nur möglich, um sich in Bewegung zu setzen, insbesondere wenn eine schwere Last angehängt ist. Der Arduino kann nur 40 Milliampere (mA) von seinen digitalen Pins liefern, viel weniger als für die meisten Motoren erforderlich wäre.

Transistoren sind Bauteile, die einem trotz des niedrigen Ausgangsstroms des Arduino die Steuerung hoher Strom- und Spannungsquellen ermöglichen. Es gibt viele unterschiedliche Typen, aber sie arbeiten alle nach dem gleichen Prinzip. Du kannst Dir Transistoren als digitale Schalter vorstellen. Sie verfügen über drei Anschlüsse, genannt Gate (engl. für Tor), Source (engl. für Quelle oder Zufluss) und Drain (engl. für Abfluss). Wenn Spannung auf dem Gate des Transistors eingeht, wird zur Kontrolle des Stromflusses der Schaltkreis zwischen den anderen beiden Pins geschlossen. Dadurch kann man selbst Motoren mit höheren Strom- bzw. Spannungsanforderungen mit dem Arduino ein- und ausschalten.

Motoren folgen dem Prinzip der Induktion. Induktion ist ein Prozess, bei dem in einer Leitung durch eine Änderung des elektrischen Stromflusses ein Magnetfeld um die Leitung herum entsteht. Im Falle eines Motors erzeugt eine eng gewundene Kupferspule innerhalb des Gehäuses das Magnetfeld, sobald Strom angelegt wird. Dieses Feld bringt die Welle (das Teil, welches aus dem Gehäuse herausschaut) zum Drehen.

Das Gleiche gilt auch umgekehrt: ein Motor kann durch Rotation der Welle Elektrizität erzeugen. Für eine kleine Demonstration schließe eine LED an den zwei Leitungen Deines Motors an und drehe die Welle mit der Hand. Falls nichts geschieht, drehe die Welle in die andere Richtung. Die LED sollte beginnen zu leuchten. Du hast Deinen Motor soeben zu einem kleinen Generator gemacht.

Selbst wenn die Energiezufuhr an einen Motor unterbrochen wird, dreht er sich noch aufgrund seiner Schwungkraft weiter. Durch die Drehung erzeugt er Spannung in die entgegengesetzte Richtung des gelieferten Stroms. Du hast diesen Effekt gerade beim Aufleuchten der LED beobachten können. Diese Rückspannung oder Sperrspannung kann Deinen Transistor beschädigen. Deshalb solltest Du eine Diode parallel zum Motor schalten, damit die Rückspannung durch sie geht. Die Diode lässt Elektrizität nur in einer Richtung fließen und schützt so den Rest des Schaltkreises.

BAUE DEN SCHALTKREIS

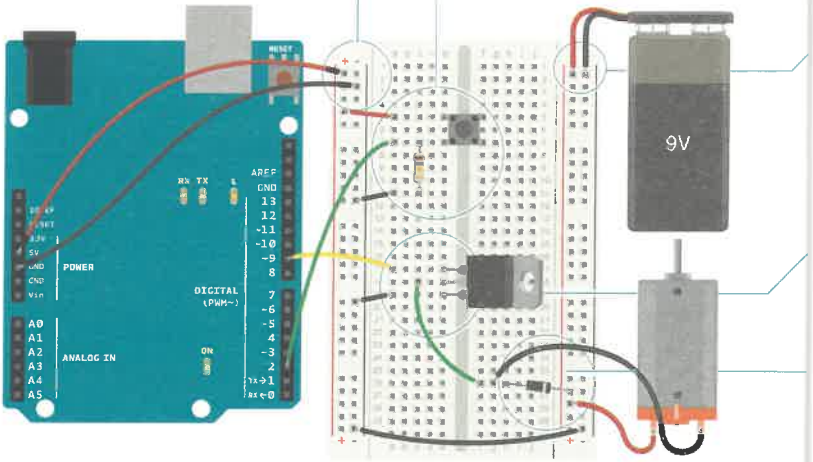


Abbildung 1

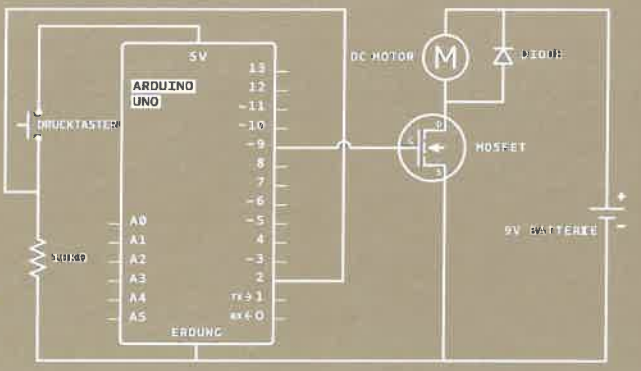


Abbildung 2

1. Schließe Strom und Erdung vom Arduino an der Steckplatine an.

2. Setze einen Drucktaster auf die Platine und schließe eine Seite am Strom und die andere Seite am digitalen Pin 2 des Arduino an. Füge einen 10-kOhm-Pull-down-Widerstand zwischen Erdung und Ausgang des Tasters ein.

3. Wenn Du Schaltkreise mit unterschiedlichen Spannungen verwendest, musst Du die jeweiligen Erdungen zu einer gemeinsamen verbinden. Schließe den 9V-Batterieclip an der Steckplatine an. Verkable die beiden Erdungen von Batterie und Arduino auf der Steckplatine, wie in Abb. 1 gezeigt. Verbinde das freie Kabel des Motors mit dem 9V-Strom.

4. Setze den Transistor auf die Platine. Schaue so auf das Bauteil, dass der Metallanhänger von Dir weg zeigt. Schließe den digitalen Pin 9 am linken Pin des Transistors an. Dieser Pin wird **Gate** genannt. Durch eine Änderung der Spannung am Gate wird eine Verbindung zwischen den anderen beiden Pins hergestellt. Schließe ein Ende des Motors am mittleren Pin des Transistors an. Dieser Pin wird **Drain** genannt. Wenn der Arduino den Transistor mit Spannung am Gate aktiviert, wird dieser Pin mit dem dritten Pin, genannt **Source**, verbunden. Verbinde Source mit der Erdung.

5. Als Nächstes verbinde die Spannungsquelle mit Motor und Steckplatine. Das letzte einzufügende Bauteil ist die Diode. Die Diode ist ein polarisiertes Bauteil, d.h. sie erlaubt den Stromfluss nur in eine Richtung. Sie ist mit einem Streifen versehen, der das negative Ende oder die Kathode der Diode kennzeichnet. Die andere Seite ist das positive Ende oder die Anode. Schließe die Anode an der Erdung des Motors und die Kathode am Strom des Motors an (siehe Abb. 1). Das scheint verkehrt herum zu sein, und genau genommen ist es das auch. Die Diode wird dafür sorgen, dass die vom Motor erzeugte Rückspannung nicht in Deinen Schaltkreis zurück gelangen kann. Wie Du Dich sicherlich erinnerst, fließt die Rückspannung nämlich in die entgegengesetzte Richtung der eingehenden Spannung.

LEDs sind übrigens auch Dioden, falls Du Dich gewundert hast, dass die Anschlüsse auch Anoden und Kathoden genannt werden. Es gibt viele Arten von Dioden, aber sie alle teilen sich ein Merkmal. Sie lassen Strom von der Anode zur Kathode fließen, aber nicht umgekehrt.

DAS PROGRAMM

Konstanten und Variablen deklarieren

Der Code gleicht jenem, den Du für das Einschalten einer LED verwendet hast. Erstelle zunächst Konstanten für die Pins des Tasters und des Motors, sowie eine Variable mit der Bezeichnung `switchState`, um den Schalterzustand zu speichern.

Richtung der Pins

Setze in `setup()` die Pin-Richtung mittels `pinMode()` vom Motor auf Ausgang `(OUTPUT)` und vom Taster auf Eingang `(INPUT)`.

Eingang lesen und Ausgang bei Druck auf HIGH ändern

Die `loop()` ist einfach gehalten. Überprüfe den Zustand des `switchPin` mit `digitalRead()`.

Wenn der Taster gedrückt wird, setze den motorPin auf `HIGH`. Wenn nicht, setze ihn auf `LOW`. Wenn der Pin auf `HIGH` steht, wird der Transistor aktiviert und der Schaltkreis des Motors geschlossen. Auf `LOW` wird sich der Motor nicht drehen.

Motoren verfügen über eine optimale Betriebsspannung. Sie verrichten ihre Arbeit noch bis zu 50 % unterhalb bzw. 50 % oberhalb dieser Betriebsspannung. Durch Variation der Spannung kannst Du die Geschwindigkeit ändern, mit der sich der Motor dreht. Halte Dich jedoch etwas dabei zurück, oder Du riskierst Deinen Motor durchzubrennen.

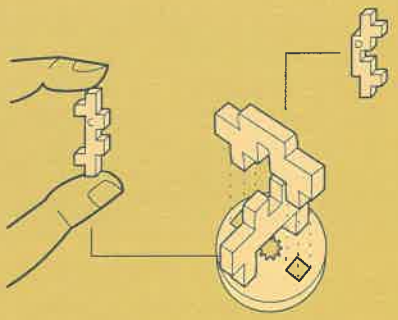
Motoren erfordern besondere Vorkehrungen, wenn sie durch einen Mikrocontroller gesteuert werden. Normalerweise kann der Mikrocontroller nicht genügend Strom und/oder Spannung liefern, um einen Motor anzutreiben. Deshalb benutzt Du Transistoren, um die Verbindung zwischen den beiden herzustellen. Es empfiehlt sich Dioden zur Vermeidung von Beschädigungen an Deinem Schaltkreis zu verwenden.

```
const int switchPin = 2;
const int motorPin = 9;
int switchState = 0;

void setup() {
  pinMode(motorPin, OUTPUT);
  pinMode(switchPin, INPUT);
}

void loop(){
  switchState = digitalRead(switchPin);

  if (switchState == HIGH) {
    digitalWrite(motorPin, HIGH);
  }
  else {
    digitalWrite(motorPin, LOW);
  }
}
```

Transistoren sind sogenannte Festkörperbauteile, d.h. sie haben keine beweglichen Teile. Aus diesem Grund kannst Du sie sehr schnell ein- und ausschalten. Schließe ein Potentiometer an einem Analogeingang an, um mittels PWM den Pin des Transistors zu steuern. Was denkst Du wird mit der Geschwindigkeit des Motors geschehen, wenn Du die ihm zugeführte Spannung veränderst? Könntest Du vielleicht mit den Mustern auf Deinem Windrad unterschiedliche visuelle Effekte erzeugen?

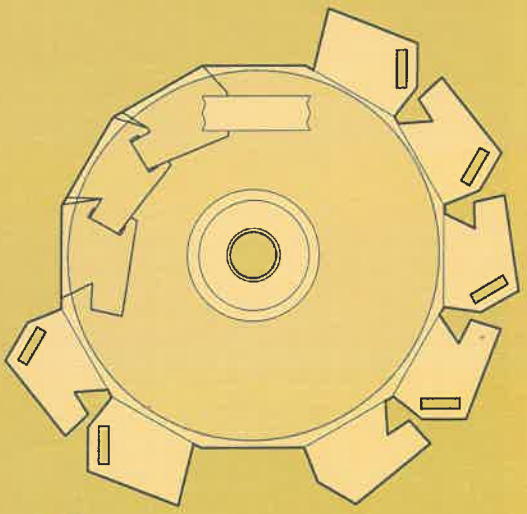
BENUTZ ES

Montiere die CD-Nabe (siehe Schritt 1) und bringe sie am Motor an (siehe Schritt 2). Bringe das vorgeschnittene Papier an einer CD an (siehe Schritt 3). Stecke die CD auf die Nabe und befestige sie mit einem Tropfen Klebstoff. Gib dem Klebstoff etwas Zeit zum Trocknen, bevor Du fortfährst. Schließe eine 9V-Batterie im Batterieclip an. Versorge Deinen Arduino über USB mit Strom. Wenn Du den Taster auf der Steckplatine betätigst, wird sich der Motor sehr schnell drehen.



1

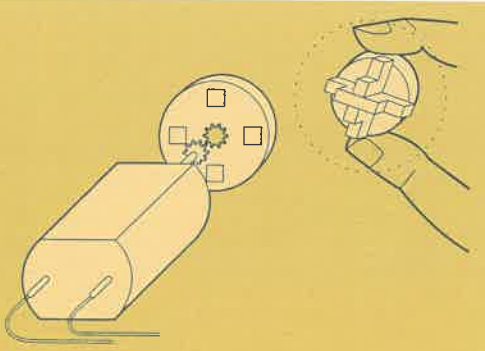
Raste Teil C in Teil B ein und stecke Teil D vorsichtig darauf.



3

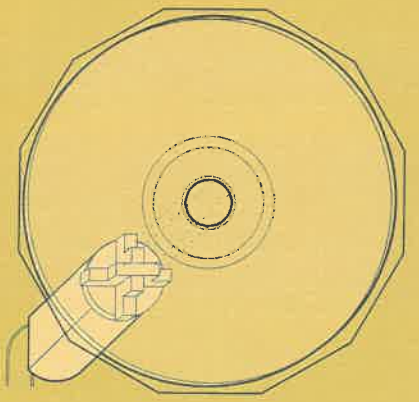
Setze die Papierscheibe auf die CD und befestige sie mit den Laschen auf der Rückseite.

Da der Motor schnell dreht, kannst Du wahrscheinlich ein recht großes Windrad bauen. Aber gib Acht, dass es nicht wegfliegt und bei jemandem im Auge landet. Experimentiere mit verschiedenen Mustern auf der Außenseite, um visuelle Effekte zu erzeugen.



2

Drücke die Welle des Motors vorsichtig in die Bohrung der Rückseite von Teil B.



4

Stecke die CD auf das Kreuz, das durch die Teile B und D geformt wird. Befestige die CD mit einem Tropfen Klebstoff.

10

BESTANDTEILE

ZOETROP

ERZEUGE VORWÄRTS WIE RÜCKWÄRTS SICH BEWEGENDE BILDER MIT DEINEM ARDUINO, INDEM DU EINEN MOTOR MIT EINER H-BRÜCKE SCHALTEST

Entdecke: *H-Brücken*

Zeit: **30 MINUTEN**
Niveau: ■■■■■

Basierend auf Projekten **1, 2, 3, 4, 9**

Noch bevor es Internet, Fernsehen oder Kinos gab, wurden einige der ersten beweglichen Bilder mit einem Gerät namens ***Zoetrop*** *erzeugt. Zoetrope schaffen die Illusion einer Bewegung aus Sammlungen von Standbildern, die voneinander ein wenig abweichen. Für gewöhnlich haben sie die Form eines Zylinders mit eingeschnittenen Schlitzen an der Seite. Wenn sich der Zylinder dreht und Du durch die Schlitze schaust, nehmen Deine Augen die einzelnen Bilder auf der anderen Seite bewegt wahr. Durch die Schlitze werden die Bilder nicht unscharf und die Geschwindigkeit der Bildabfolge lässt das Ganze beweglich erscheinen. Ursprünglich wurden diese Sensationen von Hand oder mit einer Kurbel gedreht.*

In diesem Projekt baust Du Dein eigenes Zoetrop, das eine fleischfressende Pflanze animiert. Die Bewegung erzeugst Du mit einem Motor. Für ein wenig mehr Raffinesse im System fügst Du einen Schalter zur Bestimmung der Drehrichtung hinzu, einen weiteren zum Ein- und Ausschalten, sowie ein Potentiometer zur Geschwindigkeitssteuerung.

Im Projekt mit dem motorisierten Windrad hast Du einen Motor in eine Richtung drehen lassen. Durch Umkehrung von Strom und Erdung am Motor veranlasst man einen Richtungswechsel der Drehung. Es ist nicht besonders praktisch das jedes Mal zu tun, wenn man eine Richtungsänderung wünscht, deswegen wirst Du eine sogenannte H-Brücke zur Umkehrung der Polarität des Motors verwenden.

H-Brücken sind integrierte Schaltungen (*IC* für engl. *„integrated circuit"*), die wie Bauteile eingesetzt werden. Sie enthalten umfangreiche Schaltkreise in einem kleinen Gehäuse, mithilfe derer der Aufbau komplexer Schaltkreise vereinfacht werden kann. Die H-Brücke in diesem Projekt verfügt zum Beispiel über eine Reihe eingebauter Transistoren. Um den Schaltkreis innerhalb der H-Brücke nachzubauen, würdest Du vermutlich eine zusätzliche Steckplatine benötigen.

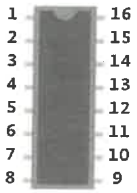


Abbildung 1

Man erhält über die an der Seite befindlichen Pins Zugriff auf den Schaltkreis. ICs können dabei über unterschiedlich viele Pins verfügen und nicht alle werden in jedem Schaltkreis eingesetzt. Es ist in der Regel übersichtlicher sich auf die Pins durch Zahlen anstelle von ihrer Funktion zu beziehen. Die Oberseite ist mit einer kleinen Kerbe gekennzeichnet und die Nummerierung der Pins erfolgt U-förmig, angefangen von oben links (siehe Abb. 1).

BAUE DEN SCHALTKREIS

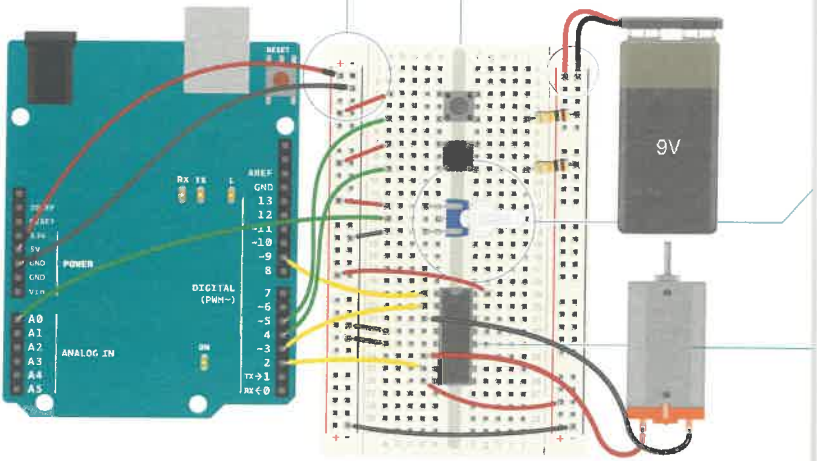


Abbildung. 2

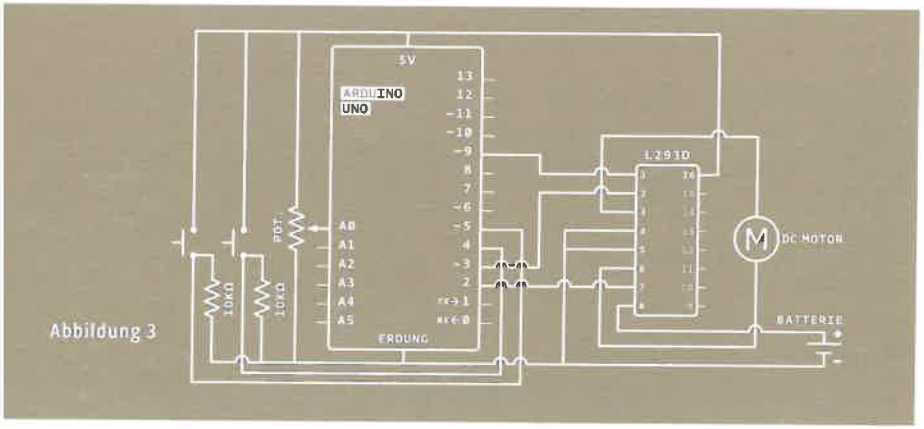


Abbildung 3

1. Schließe Strom und Erdung vom Arduino an der Steckplatine an.

2. Platziere zwei Taster auf der Steckplatine und verbinde sie jeweils mit Strom. Schalte einen 10-kOhm-Pull-down-Widerstand in Reihe mit der Erdung und den Ausgangs-Pins beider Schalter. Der Taster auf Pin 4 steuert die Richtung, der Taster auf Pin 5 schaltet den Motor ein und aus.

3. Setze das Potentiometer auf die Steckplatine. Schließe 5V auf einer Seite und Erdung auf der anderen an. Schließe den mittleren Pin am Analogeingang 0 des Arduino an. Dieser wird die Geschwindigkeit des Motors steuern.

4. Platziere die H-Brücke entlang der Mitte Deiner Steckplatine (siehe Abb. 2). Schließe Pin 1 der H-Brücke am digitalen Pin 9 des Arduino an. Das ist der Aktivierungs-Pin der H-Brücke. Wenn 5V anliegen, schaltet sich der Motor ein, wenn 0V anliegen, schaltet sich der Motor aus. Mittels PWM wirst Du an diesem Pin der H-Brücke die Drehzahl des Motors steuern.

5. 5 Schließe Pin 2 der H-Brücke am digitalen Pin 3 des Arduino an. Schließe Pin 7 am digitalen Pin 2 an. Mit diesen Pins teilst Du der H-Brücke mit, in welche Richtung sie den Motor drehen soll. Wenn Pin 3 auf `LOW` und Pin 2 auf `HIGH` stehen, dreht sich der Motor in eine Richtung. Wenn Pin 2 auf `LOW` und Pin 3 auf `HIGH` stehen, dreht sich der Motor in entgegengesetzter Richtung. Wenn beide Pins gleichzeitig auf `HIGH` oder `LOW` stehen, hört der Motor auf, sich zu drehen.

6. Die H-Brücke erhält ihren Strom über Pin 16, verbinde diesen mit 5V. Pin 4 und 5 werden an Erdung angeschlossen.

7. Schließe Deinen Motor an Pin 3 und 6 der H-Brücke an. Diese zwei Pins schalten ein oder aus, je nachdem welches Signal an Pin 2 und 7 gesendet wird.

8. Schließe die Batteriehalterung (noch ohne Batterie) an der anderen Stromschiene auf Deiner Steckplatine an. Verbinde die Erdung von Deinem Arduino mit der Erdung der Batterie. Schließe Pin 8 der H-Brücke am Batteriestrom an. Über diesen Pin betreibt die H-Brücke den Motor. Vergewissere Dich, dass die 9V- und 5V-Kabel nicht kurzgeschlossen sind, d.h. es darf keine Verbindung zwischen ihnen bestehen. Lediglich die Erdungen sollten verbunden sein.

DAS PROGRAMM

Konstanten deklarieren

Erstelle Konstanten für die Ausgangs- und Eingangs-Pins.

Variablen für den Programmstatus

Speichere die Werte der Eingänge in Variablen. Du wirst für beide Schalter Zustandsänderungen in jedem Durchlauf der Schleife ermitteln, ähnlich wie beim Sanduhr-Projekt. D.h. es muss sowohl der momentane, als auch der vorige Zustand beider Schalter festgehalten werden.

Variablen für die Motorsteuerung

`motorDirection` merkt sich die Drehrichtung des Motors, und motorEnabled speichert, ob der Motor in Betrieb ist oder nicht.

Digitale Ein- und Ausgänge festlegen

Im `setup()` definierst Du die Richtung der Pins als Ein- bzw. Ausgang.

Motor ausschalten

Setze zu Beginn den Aktivierungs-Pin (enablePin) auf `LOW`, damit der Motor nicht gleich zu drehen anfängt.

Informationen abrufen

In `loop()` liest Du zunächst den Zustand des Ein-/Ausschalters und speicherst ihn in der `onOffSwitchState`-Variable.

```
const int controlPin1 = 2;
const int controlPin2 = 3;
const int enablePin = 9;
const int directionSwitchPin = 4;
const int onOffSwitchStateSwitchPin = 5;
const int potPin = A0;

int onOffSwitchState = 0;
int previousOnOffSwitchState = 0;
int directionSwitchState = 0;
int previousDirectionSwitchState = 0;

int motorEnabled = 0;
int motorSpeed = 0;
int motorDirection = 1;

void setup(){
  pinMode(directionSwitchPin, INPUT);
  pinMode(onOffSwitchStateSwitchPin, INPUT);
  pinMode(controlPin1, OUTPUT);
  pinMode(controlPin2, OUTPUT);
  pinMode(enablePin, OUTPUT);

  digitalWrite(enablePin, LOW);
}

void loop(){
  onOffSwitchState =
    digitalRead(onOffSwitchStateSwitchPin);
  delay(1);
  directionSwitchState =
    digitalRead(directionSwitchPin);
  motorSpeed = analogRead(potPin)/4;
```

Veränderung des Ein/Aus-Schalters überprüfen	Wenn sich der gegenwärtige und der vorhergehende Schalterzustand unterscheiden, wird der Wert von motorEnabled umgekehrt. Lese die Werte des Richtungsschalters und des Potentiometers aus und speichere sie in den entsprechenden Variablen.
Änderung der Richtung überprüfen	Überprüfe, ob sich die Position des Richtungsschalters geändert hat. Falls ja, kehre auch hier die motorDirection-Variable um, denn es gibt nur zwei Richtungen, in die sich der Motor drehen kann. Dazu kann man den Ungleich-Operator wie folgt einsetzen: `motorDirection =!motorDirection.`
Pins für den Richtungswechsel setzen	Die motorDirection-Variable bestimmt, in welche Richtung sich der Motor drehen soll. Zur Umsetzung dieser Vorgabe muss ein Kontroll-Pin auf **HIGH** und der andere auf **LOW** gesetzt werden. Entsprechend des neuen Werts von motorDirection setzt Du also den Zustand der Kontrollpins. Dadurch wird bei Betätigung der Richtungstaste die Drehrichtung des Motors umgekehrt.
Geschwindigkeit mit PWM steuern	Wenn die motorEnabled-Variable auf 1 steht, steuere die Geschwindigkeit des Motors mittels PWM am Aktivierungs-Pin über `analogWrite()`. Wenn `motorEnabled` auf 0 steht, schalte den Motor aus, indem Du den `analogWrite()`-Wert auf 0 setzt.
Aktuelle Zustände speichern	Bevor Du dieser Durchgang von `loop()` abgeschlossen wird, speichere den aktuellen Zustand der Schalter als vorhergehenden Zustand für den nächsten Durchlauf.

```
 if(onOffSwitchState != previousOnOffSwitchState){
   if(onOffSwitchState == HIGH){
     motorEnabled = !motorEnabled;
   }
 }
```

```
  if (directionSwitchState !=
 previousDirectionSwitchState) {
  if (directionSwitchState == HIGH) {
    motorDirection = !motorDirection;
   }
 }
```

```
 if (motorDirection == 1) {
   digitalWrite(controlPin1, HIGH);
   digitalWrite(controlPin2, LOW);
 }

 else {
   digitalWrite(controlPin1, LOW);
   digitalWrite(controlPin2, HIGH);
 }
```

```
 if (motorEnabled == 1) {
   analogWrite(enablePin, motorSpeed);
 }
 else {
   analogWrite(enablePin, 0);
 }
```

```
 previousDirectionSwitchState =
   directionSwitchState;
 previousOnOffSwitchState = onOffSwitchState;
}
```

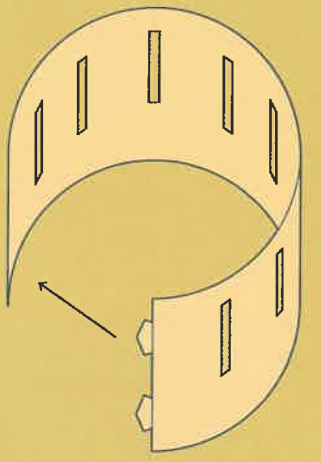
BENUTZ ES

Sobald Du überprüft hast, ob Dein Schaltkreis wie erwartet funktioniert, trenne die Batterie und das USB-Kabel vom Schaltkreis.

Schließe Deinen Arduino am Computer an. Schließe die Batterie an der Batteriehalterung an. Wenn Du den Ein-/Ausschalter betätigst, sollte der Motor sich zu drehen beginnen. Wenn Du das Potentiometer drehst, sollte er schneller bzw. langsamer werden. Wenn Du die Ein-/Austaste nochmals drückst, stoppt der Motor. Drücke die Richtungstaste und überprüfe, ob sich der Motor in beide Richtungen dreht. Wenn Du den Knopf des Potentiometers drehst, solltest Du anhand der gesendeten Werte die Geschwindigkeitsänderung beobachten können.

1

Befestige die CD auf der hölzernen Unterlage. Füge einen Tropfen Klebstoff hinzu, damit sie sich nicht löst, wenn der Motor anläuft.



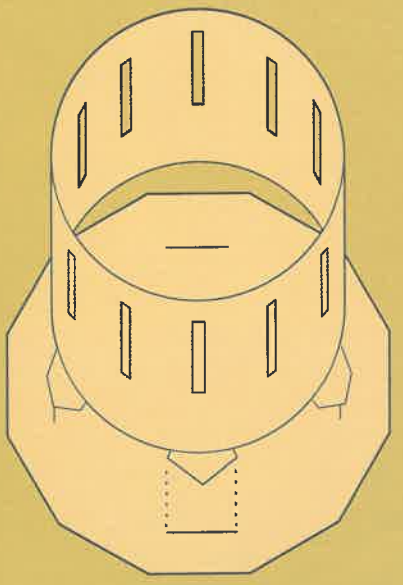
2

Benutze die Laschen, um einen Zylinder zu formen.

Um Dein Zoetrop zu bauen, benötigst Du das Windrad aus Projekt 9 und den Ausschnitt mit den vertikalen Schlitzen aus Deinem Kit. Sobald die CD sicher an der Welle des Motors angebracht ist, stelle alle Verbindungen wieder her. Halte Dein Werk in die Luft, sodass Du durch die Schlitze schauen kannst (vergewissere Dich erst, dass die CD gut am Motor befestigt ist und gehe nicht zu nahe heran). Du solltest die Abfolge der Standbilder sich „bewegen" sehen. Wenn es zu schnell oder zu langsam läuft, drehe am Knopf des Potentiometers, um die Geschwindigkeit der Animation anzupassen.

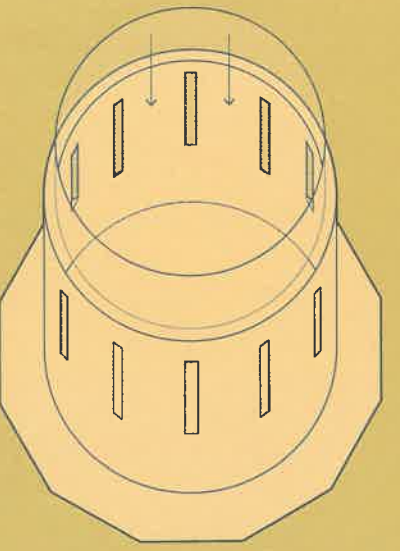
Drücke den Richtungsschalter, um die Animation rückwärts zu sehen. Das Zoetrop und die beiliegenden Bilder im Kit sind nur eine Starthilfe: nutze den Ausschnitt aus dem Kit als Vorlage für Deine eigenen Animationen.

Beginne mit einem einfachen Bild. Kennzeichne darin einen Fixpunkt, und nimm kleine Änderungen an jedem Bild vor. Versuche, schrittweise zum ursprünglichen Bild zurückzukommen, damit Du die Animation in einer ununterbrochenen Schleife abspielen kannst.



3

Stecke die vier Laschen in das Unterteil des Zoetrops.



4

Setze den Papierstreifen mit den Bildern in das Zoetrop ein.

Zoetrope funktionieren aufgrund des Phänomens, das als „Augenträgheit" bekannt ist. Augenträgheit beschreibt die Illusion von Bewegung, die erzeugt wird, wenn unsere Augen Standbilder mit kleinen Abweichungen in schneller Abfolge sehen. Wenn Du online nach „POV display" (POV ist engl. für „persistence of vision") suchst, wirst Du viele Projekte von Leuten finden, die diesen Effekt wirksam einsetzen, häufig mit LEDs und einem Arduino.

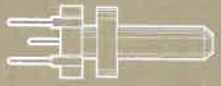
Baue eine Halterung für den Motor. Eine kleine Pappschachtel mit einem ausgeschnittenen Loch sollte funktionieren. So hast Du die Hände frei, um mit den Schaltern und dem Drehknopf zu spielen. Das wird es einfacher machen, Dein Werk jedem zu zeigen.

Mit ein wenig Arbeit kannst Du ein Zoetrop bauen, das auch bei geringem Licht funktioniert. Schließe eine LED und einen Widerstand an einem der freien digitalen Ausgänge an. Nimm ein zweites Potentiometer und schließe es an einen Analogeingang an. Platziere die LED so, dass sie die Bilder anstrahlt. Verwende den Analogeingang, um das Aufblitzen der LED zu steuern. Versuche, es zeitlich so abzustimmen, dass die LED immer dann aufblitzt, wenn der Schlitz vor Deinen Augen ist. Vielleicht musst Du dazu ein wenig mit den Drehknöpfen herumtüfteln, aber das Ergebnis ist wahrlich spektakulär!

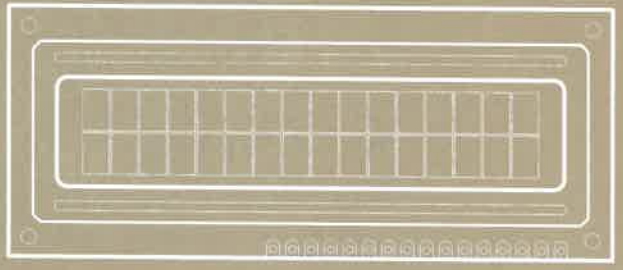
SCHALTER

10-KOHM-WIDERSTAND

220-OHM-WIDERSTAND



POTENTIOMETER



LCD-ANZEIGE

KRISTALLKUGEL

ERSCHAFFE EINE KRISTALLKUGEL, UM DEINE ZUKUNFT VORAUSZUSAGEN

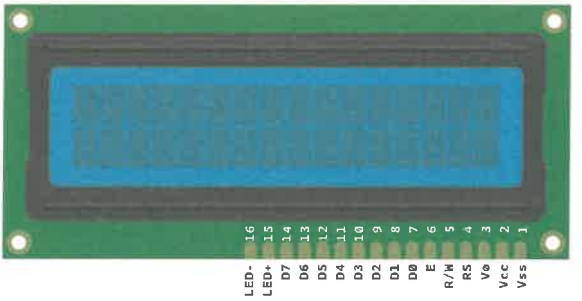
Entdecke: LCD-Anzeigen, switch/case-Statements, random()

Zeit: **1 STUNDE**
Niveau: ■■■■■

Basierend auf Projekten: **1, 2, 3**

Kristallkugeln erfreuen sich allgemeiner Beliebtheit für die „Vorhersage" der Zukunft. Wenn Du der allwissenden Kugel eine Frage stellst, musst Du sie nur noch umdrehen, um eine Antwort zu erhalten. Die Antworten werden vorher festgelegt, aber ihr Inhalt obliegt ganz allein Dir. Du wirst mit Deinem Arduino aus insgesamt 8 Antworten wählen. Der Kippschalter in Deinem Kit wird dabei behilflich sein, die Rüttelbewegung der Kugel bei der Antwortfindung nachzustellen.

Die LCD wird zur Darstellung alphanumerischer Zeichen verwendet. Die in Deinem Kit hat 16 Spalten und 2 Zeilen für insgesamt 32 solcher Zeichen. Sie verfügt über eine Vielzahl von Anschlüssen. Diese Pins dienen der Stromversorgung und der Kommunikation dessen, was auf der Anzeige geschrieben werden soll. Du musst jedoch nicht alle von ihnen anschließen. Siehe Abb. 1 für die relevanten Pins.



Die LCD-Pins, die für das Projekt und die Beschriftung verwendet werden
Abbildung 1

BAUE DEN SCHALTKREIS

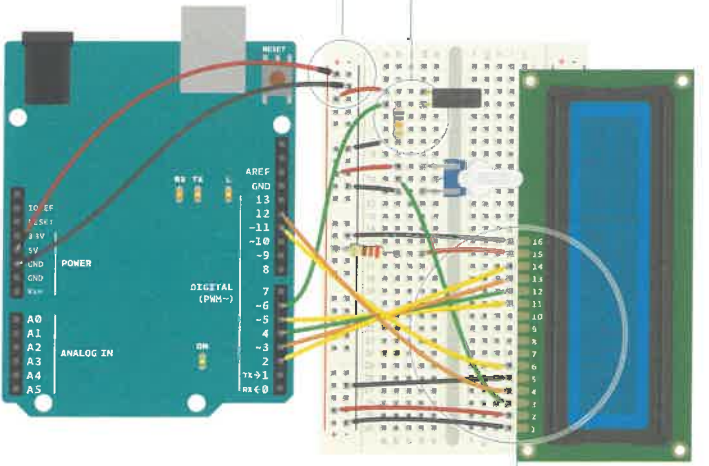


Abbildung 2

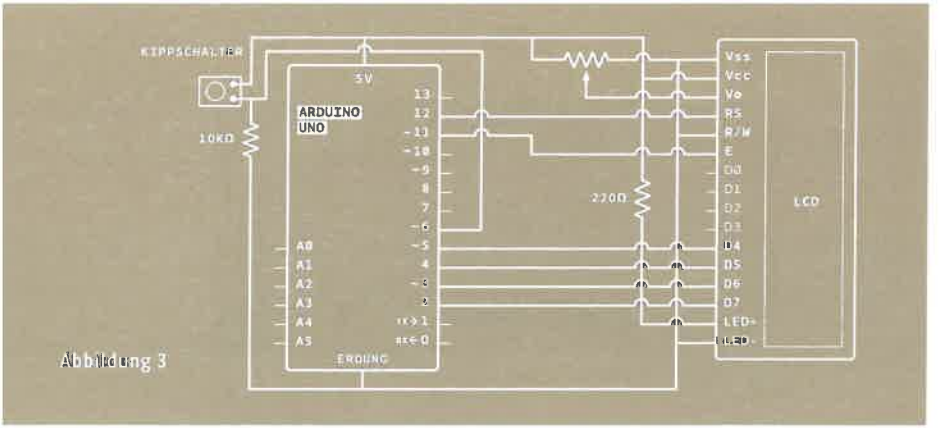


Abbildung 3

In diesem schematischen Diagramm entspricht die Anordnung der Pins nicht der tatsächlichen Anordnung auf der LCD wie in Abb. 2. In einem Diagramm werden die Pins ihrem Zweck entsprechend logisch geordnet, um das Diagramm so klar wie möglich darstellen zu können.
Das benötigt für Neulinge erst ein wenig Gewöhnung.

Der Schaltkreis ist nicht übermäßig komplex, hat aber eine ganze Menge Kabel. Pass also gut bei der Verkabelung auf, damit alles seine Richtigkeit hat.

1. Schließe Strom und Erdung vom Arduino an der Steckplatine an.

2. Setze den Kippschalter auf die Steckplatine und schließe einen Stift an 5V an. Schließe den anderen mit einem 10-kOhm-Widerstand an Erdung und an Pin 6 Deines Arduino an. Die Verbindung erfolgt als digitaler Eingang wie schon bei einigen vorigen Projekten.

3. Der Registerauswahl-Pin (**RS**) steuert, wo die Zeichen auf der Anzeige erscheinen werden. Der Lese-/Schreib-Pin (**R/W**) setzt die Anzeige in den Lese- oder Schreibmodus. In diesem Projekt verwendest Du letzteren. Der Aktivierungs-Pin (**EN**) teilt der Anzeige Befehlseingänge mit. Die Daten-Pins (**D0-D7**) werden zur Zeichenübertragung genutzt. Du wirst nur 4 dieser Pins verwenden (**D4-D7**). Schließlich gibt es einen Anschluss zur Kontrasteinstellung der Anzeige. Dazu wirst Du ein Potentiometer einsetzen.

4. Die mitgelieferte LiquidCrystal-Bibliothek erledigt alle Schreibvorgänge auf diesen Pins und vereinfacht so die Programmierung der Zeichenanzeige. Die beiden äußeren Pins (**Vss** und LED-) müssen an Erdung angeschlossen sein. Schließe auch den R/W-Pin an Erdung an. Dies versetzt die Anzeige in den Schreibmodus. Die Stromversorgung der LCD (**Vcc**) muss direkt an 5V angeschlossen werden. Der Pin LED+ wird über einen 220-Ohm-Widerstand am Strom angeschlossen.

5. Verbinde: Arduino digitaler Pin 2 mit LCD **D7**, Arduino digitaler Pin 3 mit LCD **D6**, Arduino digitaler Pin 4 mit LCD **D5**, Arduino digitaler Pin 5 mit LCD **D4**. Das sind die Daten-Pins, die der Anzeige die darzustellenden Zeichen übermitteln.

6. Schließe **EN** der LCD an Pin 11 des Arduino an. **RS** der LCD wird an Pin 12 angeschlossen. Dieser Pin erlaubt das Schreiben zur LCD.

7. Setze das Potentiometer auf die Steckplatine und schließe ein Ende am Strom, das andere an Erdung an. Der mittlere Pin wird mit **VØ** der LCD verbunden. Damit kann man den Kontrast des Displays anpassen.

LiquidCrystal-Bibliothek einrichten

Zuerst musst Du die `LiquidCrystal`-Bibliothek importieren. Ähnlich wie schon zuvor bei der Servo-Bibliothek initialisiert der nächste Schritt die Bibliothek und zusätzlich die Pins zur Kommunikation.

Nun wird es wieder Zeit Variablen und Konstanten zu deklarieren. Erstelle eine Konstante für den Pin des Schalters (switchPin), eine Variable um den aktuellen Zustand des Schalters zu speichern (switchState), eine um den vorhergehenden Zustand des Schalters zu speichern (prevSwitchState), sowie eine für die ausgewählte Antwortmöglichkeit (reply).

Erste Zeile ausgeben

Setze switchPin als Eingang mit dem Befehl `pinMode()` in `setup()`. Starte die LCD-Bibliothek und teile ihr die Größe der Anzeige mit.

Cursor bewegen

Du beginnst mit einem Begrüßungstext beim Start der Kristallkugel. Mit `print()` kannst Du auf die LCD-Anzeige schreiben. In der oberen Zeile lässt Du die Worte „Frage die" erscheinen. Der Cursor ist automatisch am Anfang der ersten Zeile.

Die Koordinaten der ersten Spalte auf der zweiten Zeile sind 0,1 (wie Du Dich vielleicht entsinnst, beginnt die Indizierung bei Computern bei null; 0,0 wäre also die erste Spalte der ersten Zeile). Benutze die Funktion `lcd.setCursor()`, um den Cursor an seine neue Position zu verschieben, und schreibe auf die Anzeige „Kristallkugel!".

Wenn Du jetzt das Programm startest, wird „Frage die Kristallkugel!" auf deiner Anzeige dargestellt.
In `loop()` rufst Du zunächst den Zustand des Schalters ab und schreibst den Wert in die switchState-Variable.

Zufällige Antwort wählen

Überprüfe mit einem `if()`-Statement, ob der Schalter in einer anderen Position als vorher ist. Wenn das der Fall ist und er derzeit auf LOW steht, soll eine zufällige Antwort ausgewählt werden.
Die Funktion `random()` gibt eine Zahl basierend auf dem ihr übergebenen Argument aus. Anfangs legst Du insgesamt 8 verschiedene Antworten für die Kugel fest. Wann immer random() mit dem Argument 8 aufgerufen wird, generiert sie eine Zufallszahl zwischen 0-7. Speichere diese Zahl in der reply-Variable.

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

const int switchPin = 6;
int switchState = 0;
int prevSwitchState = 0;
int reply;

void setup() {
  lcd.begin(16, 2);
  pinMode(switchPin,INPUT);

  lcd.print(„Frage die“);

  lcd.setCursor(0, 1);
  lcd.print(„Kristallkugel!“);
}

void loop() {
  switchState = digitalRead(switchPin);

  if (switchState != prevSwitchState) {
    if (switchState == LOW) {
      reply = random(8);
```

LCD-Bibliothek-Referenz
arduino.cc/lcdlibrary

Zufalls-Referenz
arduino.cc/random

Lösche den angezeigten Text mit der Funktion `lcd.clear()`. Dadurch wird auch der Cursor zurück an die Position 0,0 bewegt, d.h. die erste Spalte in der ersten Zeile der LCD. Schreibe auf die LCD „Die Kugel sagt:" und verschiebe den Cursor in die nächste Zeile für die ausgewählte Antwort. Abhängig vom übergebenen.

Wert führt das `switch()`-Statement unterschiedliche Teile des Programms aus. Jeder dieser unterschiedlichen Programmteile wird Case (engl. Fall) genannt. `switch()` entscheidet anhand des Werts der reply-Variable welcher Fall gilt und somit ausgeführt werden soll.

Innerhalb der case-Statements ist der Code gleich, aber die Textmeldungen unterscheiden sich. Zum Beispiel wird bei case 0 der Code `lcd.print("Ja")` ausgeführt. Nach der `lcd.print()`-Funktion gibt es einen weiteren Befehl: `break`. Dadurch wird das Ende des case-Statements definiert. Wenn das Programm bei `break` ankommt, springt es an das Ende des switch-Statements. Vorerst benötigst Du insgesamt 8 case-Statements, also für jede „Antwortmöglichkeit eines. Vier der Antworten sind positiv, zwei negativ, und zwei fordern Dich auf es nochmals zu versuchen.

Den Abschluss von `loop()` macht die Zuweisung des switchState-Wertes zu `prevSwitchState`. Dadurch kannst Du Veränderungen vom Schalter in den loop() Durchgängen mit verfolgen.

```
      lcd.clear();
      lcd.setCursor(0, 0);
      lcd.print(„Die Kugel sagt:“);
      lcd.setCursor(0, 1);

      switch(reply){
        case 0:
        lcd.print(„Ja“);
        break;
        case 1:
        lcd.print(„Ganz bestimmt“);
        break;
        case 2:
        lcd.print(„Gewiss“);
        break;
        case 3:
        lcd.print(„Gute Aussichten“);
        break;
        case 4:
        lcd.print(„Nicht sicher“);
        break;
        case 5:
        lcd.print(„Frag nochmal“);
        break;
        case 6:
        lcd.print(„Zweifelhaft“);
        break;
        case 7:
        lcd.print(„Nein“);
        break;
      }
    }
  }

  prevSwitchState = switchState;
}
```

Switch Case Referenz
arduino.cc/switchcase

BENUTZ ES

Schalte den Arduino ein, um die Kristallkugel zu verwenden. Überprüfe, ob „Frage die Kristallkugel!" angezeigt wird. Wenn Du keine Zeichen sehen kannst, drehe am Potentiometer zur Kontrastanpassung der Anzeige.

Stelle Deiner Kristallkugel eine Frage und kippe den Schalter vor und zurück. Nun solltest Du Deine Antwort erhalten. Sollte sie Dir nicht gefallen, frage erneut.

Füge Deine eigene Aussagen mit `print()` hinzu, aber bedenke, dass Du nur Platz für 16 Zeichen pro Zeile hast. Du kannst auch die Zahl der Antwortmöglichkeiten erhöhen. Achte neben der Erweiterung des switch/case-Statements darauf, dass Du auch das Argument der random()-Funktion für die Generierung des Werts der reply-Variable anpasst.

LCDs funktionieren über die Veränderung der elektrischen Eigenschaften einer Flüssigkeit zwischen polarisiertem Glas. Das Glas lässt nur bestimmte Arten von Licht durchscheinen. Durch Aufladung nimmt die Flüssigkeit zwischen dem Glas einen halbfesten Zustand an. Dieser neue Zustand läuft in eine andere Richtung als das polarisierte Glas und verhindert so das Licht durchdringt, wodurch die Zeichen auf der Anzeige erscheinen.

Die hier verwendeten Funktionen für die Textanzeige auf der LCD sind ziemlich simpel. Sobald Du diese Funktionen gut beherrschst, schaue Dir ruhig noch ein paar weitere aus der Bibliothek an. Versuche z.B. Text zu scrollen oder kontinuierlich zu aktualisieren. Um mehr über die LiquidCrystal-Bibliothek zu erfahren, besuche: ***arduino.cc/lcd***

Mithilfe einer LCD und der LiquidCrystal-Bibliothek lässt sich Text auf einer Anzeige darstellen. Mit switch/case-Statements kannst Du den Programmablauf über den Vergleich einer Variablen mit festgelegten Werten steuern.

12

BESTANDTEILE

KLOPF-SCHLOSS

BAUE DEINE EIGENE GEHEIME SCHLIESSVORRICHTUNG, UM UNERWÜNSCHTE GÄSTE FERNZUHALTEN!

Entdecke: Piezo als Eingang verwenden, eigene Funktionen schreiben

Zeit: **1 STUNDE**
Niveau: ■■■■■

Basierend auf Projekten: **1, 2, 3, 4, 5**

Der Piezo zur Musikwiedergabe im Theremin- und Keyboard-Projekt kann auch als Eingabegerät verwendet werden. Sofern er mit 5V versorgt wird, kann er zur Messung von Vibrationen eingesetzt werden, die über die Analogeingänge des Arduino ausgelesen werden können. Damit das gut gelingt, wird ein Widerstand mit hohem Wert (z.B. 1 MOhm) als Referenz zur Erdung notwendig sein.

Wenn der Piezo flach gegen eine feste, vibrationsfähige Oberfläche gedrückt wird, wie z.B. einen Holztisch, kann der Arduino die Intensität des Klopfens messen. Mit dieser Information kannst Du die Anzahl der erfolgten Klopfzeichen abschätzen und mit einem akzeptierten Bereich vergleichen. Du verfolgst im Programm also die Anzahl von Klopfzeichen und ihre Übereinstimmung mit Deiner Vorgabe.

Mit einem Schalter verriegelst Du den Motor. LEDs wirst Du zur Statusmeldung einsetzen: eine rote LED für verschlossen, eine grüne LED für entriegelt, und eine gelbe LED, wenn ein gültiges Klopfzeichen empfangen wurde.

Außerdem wirst Du eine eigene Funktion schreiben, um die Lautstärke eines Klopfzeichens zu überprüfen. Eigene Funktionen können Dir durch die Wiederverwendung von mehrfach erforderlichem Code viel Zeit sparen. Funktionen nehmen Argumente an und geben Werte zurück. In diesem Fall wirst Du einer Funktion die Lautstärke des Klopfens übergeben und, sofern der Wert in den festgelegten Bereich fällt, eine Variable erhöhen.

Man könnte natürlich nur den Schaltkreis an sich bauen, aber ein selbstgebastelter Verriegelungsmechanismus bringt doch viel mehr Freude. Mit einer hölzernen Kiste oder Pappschachtel als Verkleidung und einem Loch an der Seite, kann der Servo-Motor als Sperre eingesetzt werden, um andere von Deinen Sachen fernzuhalten.

BAUE DEN SCHALTKREIS

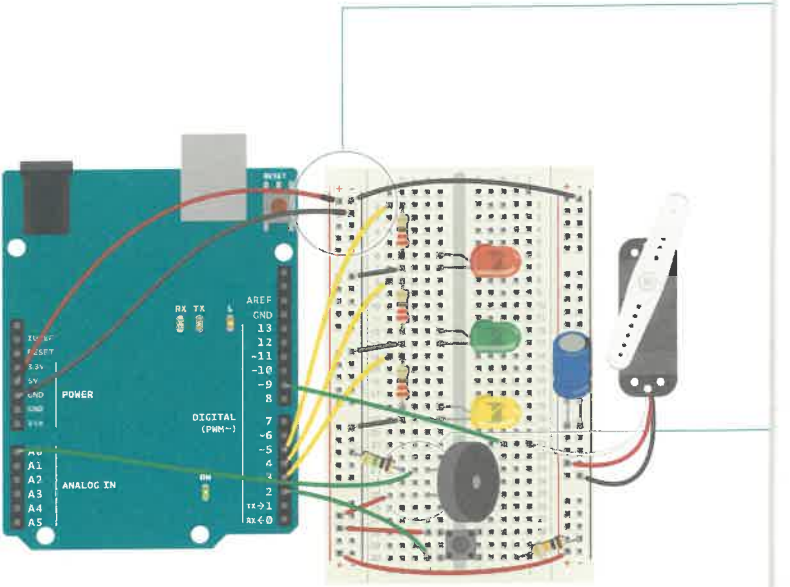


Abbildung 1

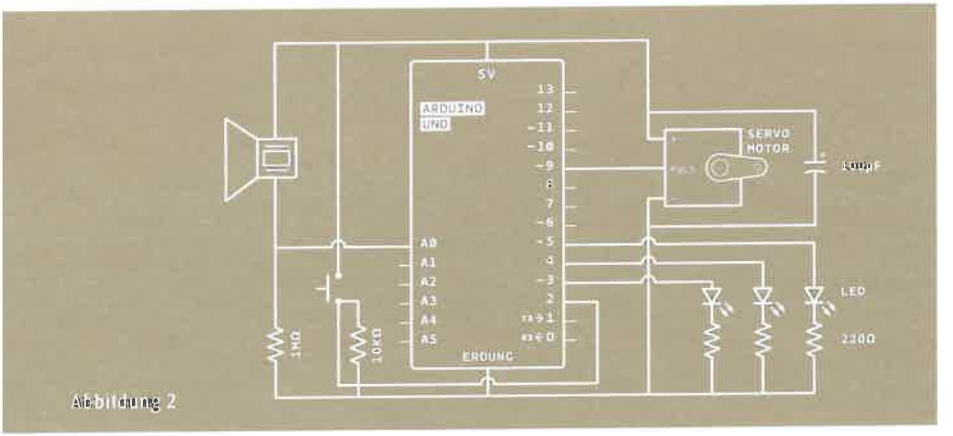


Abbildung 2

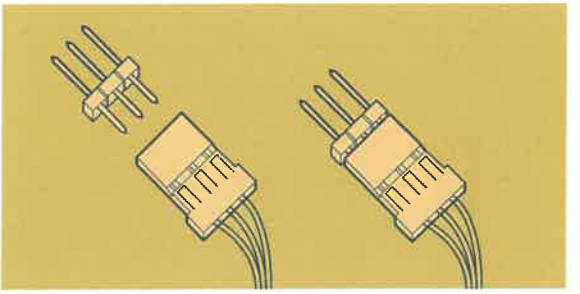
Hier gibt es diesmal eine ganze Menge von Verbindungen auf der Platine, also pass gut bei der Verkabelung auf.

1. Schließe Strom und Erdung vom Arduino an der Steckplatine an. Setze einen Drucktaster auf die Steckplatine und schließe eines seiner Enden an 5V, und das andere über einen 10-kOhm-Widerstand an Erdung an. Verbinde den Knotenpunkt mit dem digitalen Pin 2 des Arduino.

2. Schließe die Piezo-Kabel an der Steckplatine an. Schließe ein Kabel am Strom an. Falls Dein Piezo eine rote oder mit einem „+" gekennzeichnete Leitung hat, ist diese für den Stromanschluss. Wenn Dein Piezo keinen Hinweis auf Polarität anzeigt, kannst Du ihn auf beide Arten anschließen. Verbinde das andere Ende des Piezo mit dem analogen Pin 0 des Arduino. Schalte einen 1-MOhm-Widerstand zwischen Erdung und dem anderen Kabel. Niedrigere Widerstandswerte machen den Piezo weniger empfindlich für Erschütterungen.

3. Verbinde die Kathoden (kurzer Stift) der LEDs mit Erdung an und schalte einen 220-Ohm-Widerstand in Reihe mit den Anoden. Schließe über die jeweiligen Widerstände die gelbe LED am digitalen Pin 3, die grüne LED am digitalen Pin 4 und die rote LED am digitalen Pin 5 des Arduino an.

4. Stecke die Stiftleiste in die Buchse des Servo-Motors (siehe Abb. 3). Schließe das rote Kabel am Strom, und das schwarze Kabel an Erdung an. Setze einen 100-µF-Elektrolytkondensator zwischen Strom und Erdung, um Spannungsschwankungen auszugleichen. Achte auf die richtige Polarisierung des Kondensators. Schließe das Datenkabel des Servos an Pin 9 auf Deinem Arduino an.



Da Dein Servomotor Buchsen hat, benötigst Du eine Stiftleiste, um ihn an der Steckplatine anschließen zu können.
Abbildung 3

DAS PROGRAMM

Servo-Bibliothek

Wie im vorangegangenen Stimmungsbarometer-Projekt musst Du die `Servo`-Bibliothek zunächst importieren und eine Instanz für die Verwendung des Servomotors initialisieren.

Nützliche Konstanten

Erstelle Konstanten, um Deine Ein- und Ausgänge zu benennen.

Variablen für Schalter- und Piezo-Werte

Erstelle Variablen, um die Werte vom Schalter und vom Piezo zu speichern.

Klopfschwellen

Deklariere Konstanten für die Schwellwerte minimal bzw. maximal zulässiger Klopfmessungen.

Pin-Richtung, Servo-Objekt und serielle Schnittstelle initialisieren.

Die Variable mit der Bezeichnung locked (engl. verschlossen) wird speichern, ob die Vorrichtung verriegelt ist oder nicht. Ein `Boolean` ist ein Datentyp, der nur wahr (true, 1) oder falsch (false, 0) annehmen kann. Zunächst solltest Du mit einem entriegelten Zustand beginnen. Die letzte globale Variable speichert die Anzahl empfangener gültiger Klopfzeichen.

Richtung der digitalen Pins festlegen und Servo-Objekt und serielle Schnittstelle initialisieren.

Im `setup()` teilst Du den Servo Pin 9zu. Definiere die LED-Pins als Ausgänge, und die Pins des Schalters als Eingänge.

Entriegeln

Initialisiere die serielle Kommunikation mit dem Computer, um die gemessene Lautstärke des Klopfens, den aktuellen Status des Schlosses und die Anzahl der noch erforderlichen Klopfzeichen einzusehen. Schalte die grüne LED ein, bewege den Servo auf die entriegelte Position und sende den aktuellen Status zum seriellen Monitor.

Schalter überprüfen

In `loop()` überprüfst Du als erstes, ob das Schloss verriegelt ist oder nicht. Der weitere Ablauf des Programms wird hier bestimmt. Wenn es offen ist, lies den Wert des Schalters aus.

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

const int piezo = A0;
const int switchPin = 2;
const int yellowLed = 3;
const int greenLed = 4;
const int redLed = 5;

int knockVal;
int switchVal;

const int quietKnock = 10;
const int loudKnock = 100;

boolean locked = false;
int numberOfKnocks = 0;

void setup(){
  myServo.attach(9);
  pinMode(yellowLed, OUTPUT);
  pinMode(redLed, OUTPUT);
  pinMode(greenLed, OUTPUT);
  pinMode(switchPin, INPUT);
  Serial.begin(9600);

  digitalWrite(greenLed, HIGH);
  myServo.write(0);
  Serial.println("Das Schloss ist offen!");
}

void loop(){
  if(locked == false){
    switchVal = digitalRead(switchPin);
```

Verriegeln

Sofern der Schalter gedrückt wird, setzt Du die locked-Variable auf true, um den Zustand des Schlosses als verriegelt zu vermerken. Schalte die grüne LED aus, und die rote LED ein. Wenn Du den seriellen Monitor nicht in Betrieb hast, hilft dieses visuelle Feedback den aktuellen Status zu erfahren. Bewege den Servo an die Position zur Verriegelung und melde an den seriellen Monitor zurück. Füge eine Verzögerung hinzu, damit das Schloss genug Zeit für die Bewegung zur entsprechenden Position hat.

Klopf-Sensor abrufen

Wenn die Variable locked den Wert true hat und somit die Verriegelung aktiv ist, lese die vom Piezo gemessene Vibration aus und speichere sie in der Variable `knockVal`.

Nur gültige Klopfzeichen zählen

Das nächste if-Statement überprüft, ob bislang weniger als drei gültige Klopfzeichen eingegangen sind und ob der Sensor Vibrationen empfängt. Sofern beides zutrifft, überprüfe das aktuelle Klopfzeichen auf seine Gültigkeit und, falls ja, erhöhe die Variable `numberOfKnocks`. An dieser Stelle rufst Du die von dir erstellte Funktion `checkForKnocks()` auf. Du wirst sie zwar erst nach `loop()` schreiben, aber da Du bereits weißt, dass hier die Klopfzeichen überprüft werden müssen, kannst Du den Funktionsaufruf mit knockVal als Argument schon einfügen. Nach Abschluss der Überprüfung gebe die noch benötigten Klopfzeichen aus.

Entriegeln

Überprüfe, ob Du drei oder mehr gültige Klopfzeichen erhalten hast. Falls ja, ändere die Variable locked zu false und bewege den Servo an die Position zur Entriegelung. Warte einige Millisekunden, damit die Bewegung beginnen kann, und ändere den Status der grünen und roten LEDs. Sende eine Statusmeldung an den seriellen Monitor, dass das Schloss offen ist.

Schließe das `else`-Statement und die `loop()` mit zwei geschweiften Klammern ab.

Funktion zur Gültigkeitsermittlung eines Klopfzeichens schreiben

Jetzt ist es Zeit für die Funktion `checkForKnock()`. Bei der Erstellung einer Funktion musst Du festlegen, ob sie einen Wert zurück geben wird oder nicht. Falls nicht, deklarierst Du sie mit dem Typ `void`, wie bei den Funktionen `loop()` und `setup()`. Andernfalls musst Du den Datentyp des Rückgabewertes deklarieren (`int`, `long`, `float` usw.) In diesem Fall überprüfst Du, ob ein Klopfzeichen gültig ist (true) oder nicht (false), d.h. der Rückgabewert ist ein Boolean.

```
    if(switchVal == HIGH){
      locked = true;
      digitalWrite(greenLed,LOW);
      digitalWrite(redLed,HIGH);
      myServo.write(90);
      Serial.println(„Das Schloss ist verriegelt!“);
      delay (1000);
    }
  }

  if(locked == true){
    knockVal = analogRead(piezo);

    if(numberOfKnocks < 3 && knockVal > 0){
      if(checkForKnock(knockVal) == true){
        numberOfKnocks++;
      }
      Serial.print(3-numberOfKnocks);
      Serial.println(“weitere Klopfzeichen nötig“);
    }

    if(numberOfKnocks >= 3){
      locked = false;
      myServo.write(0);
      delay(20);
      digitalWrite(greenLed,HIGH);
      digitalWrite(redLed,LOW);
      Serial.println(“Der Kasten ist entriegelt!“);
    }
  }
}

boolean checkForKnock(int value){
```

Diese Funktion wird anhand des Werts von `knockVal` das Klopfzeichen als gültig bzw. ungültig einstufen. Um den Wert der Variable der Funktion zu übergeben, erstellst Du zu Beginn einen Parameter mit der Bezeichnung value.

Klopf-Gültigkeit überprüfen

Wann immer Du in Deiner Funktion auf den `value` Parameter zugreifst, wird auf das im Hauptprogramm eingegebene Argument verwiesen. An dieser Stelle entspricht `value` `knockVal`. Überprüfe nun, ob value größer als quietKnock und kleiner als loudKnock ist.

Gültigkeit anzeigen

Wenn value zwischen diesen beiden Schwellen liegt, ist das Klopfzeichen gültig. Blinke die gelbe LED einmal und sende den Wert des Klopfzeichens an den seriellen Monitor.

True zurückgeben

Um den Ausgang des Vergleichs dem Hauptprogramm mitzuteilen, benutzt Du den Befehl `return`. Durch Verwendung von `return` wird auch die Funktion beendet und Du kehrst zum Hauptprogramm zurück.

Ungültiges Klopfen zurückmelden; Funktion gibt false zurück

Wenn `value` entweder zu leise oder zu laut ist, sende diesen Wert mit entsprechender Nachricht zum seriellen Monitor und gib false zurück.

Beende Deine Funktion mit einer weiteren Klammer.

BENUTZ ES

Verbinde den Schaltkreis mit Deinem Arduino und öffne den seriellen Monitor. Die grüne LED sollte sich einschalten und der Servo sich in die geöffnete Position bewegen.

Im seriellen Monitor sollte „Das Schloss ist offen!" zu lesen sein. Vermutlich wirst Du vom Piezo ein kurzes Klicken bei der ersten Inbetriebnahme hören.

Um ein Gefühl für Deine Klopffunktion zu bekommen, klopfe unterschiedlich stark und beobachte was passiert. Ein gültiges

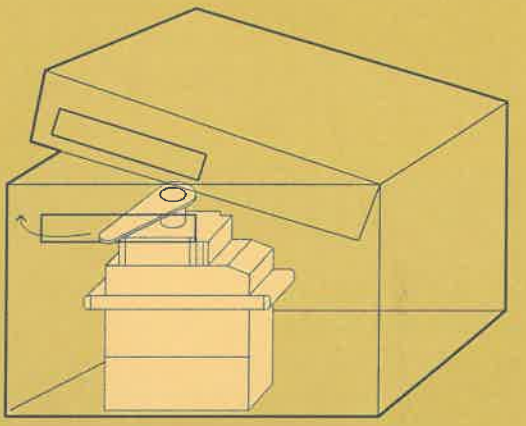
```
57  if(value > quietKnock && value < loudKnock){
58    digitalWrite(yellowLed, HIGH);
59    delay(50);
60    digitalWrite(yellowLed, LOW);
61    Serial.print(„Gültiges Klopfen mit Wert“);
62    Serial.println(value);
63    return true;
64  }
65  else {
66    Serial.print(„Ungültiges Klopfen mit Wert“);
67    Serial.println(value);
68    return false;
69  }
70 }
```

Klopfzeichen wird Dir durch das Blinken der gelben LED und im seriellen Monitor mit dem entsprechenden Wert anzeigt. Dort wirst Du auch eine Rückmeldung bzgl. der verbleibenden Anzahl noch erforderlicher Klopfzeichen zur Öffnung des Schlosses erhalten.

Sobald die richtige Anzahl erreicht ist, wird die rote LED aus- und die grüne LED angehen, der Servo wird sich um 90 Grad bewegen und der serielle Monitor wird die Entriegelung melden.

Sinnvolle gültige Werte Deines Klopfens könnten möglicherweise von denen im Beispiel abweichen. Das kann von mehreren Umständen abhängen, wie z.B. der Art der Oberfläche, auf welcher der Sensor angebracht ist, oder auch wie stabil er befestigt wurde. Benutze den seriellen Monitor und das AnalogInSerialOut-Beispiel der Arduino-IDE, um passende Klopfwerte für Deinen Aufbau zu finden. Du kannst eine ausführliche Erklärung dieses Beispiels hier finden:
arduino.cc/analogtoserial

Wenn Du dieses Projekt in einem Kasten verbauen willst, wirst Du Löcher für die LEDs und den Schalter machen müssen. Außerdem wird ein Schlitz zur Verriegelung nötig sein, in den sich der Servo hineindrehen kann. Ein weiteres Loch für das USB-Kabel sollte sich ebenfalls als nützlich erweisen, um herauszufinden wie empfindlich diese neue Umgebung für Klopfgeräusche ist.
Vielleicht wirst Du auch den Arduino und die Steckplatine neu anordnen, oder die LEDs und den Schalter löten müssen, um sie gut von außen zugänglich zu machen. Löten ist ein Prozess bei dem zwei oder mehr Metallteile mit einem Haftmittel verschmolzen werden. Falls Du noch nie zuvor gelötet hast, bitte jemanden mit Erfahrung um Hilfe oder experimentiere erstmal mit Kabelresten, bevor Du Dich an Teile aus dem Projekt heranwagst. Da eine gelötete Verbindung nicht wieder getrennt werden kann, solltest Du Dich vorher vergewissern, dass die von Dir ausgewählten Teile nicht anderweitig benötigt werden.
Gehe auf ***arduino.cc/soldering*** für eine gute Anleitung zum Löten.



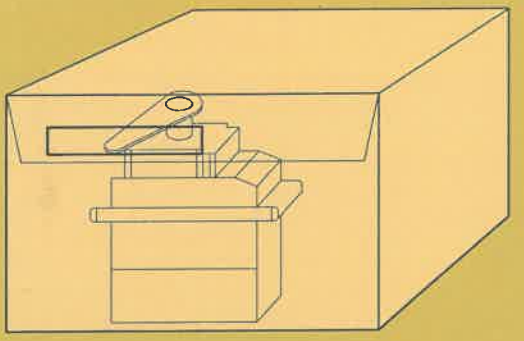
1

Schneide zwei Schlitze in Deinen Kasten, einen auf der Seite, einen zweiten durch den Deckel. Setze den Servo so in dem Kasten ein, dass sich der Arm in den Schlitz hinein- und wieder heraus bewegen kann.

Deine eigenen Funktionen zu schreiben vereinfacht nicht nur die Steuerung des Programmflusses, sondern trägt auch zur Lesbarkeit des Codes bei, wenn Deine Projekte immer größer und größer werden. Im Laufe der Zeit wirst Du beim Schreiben Deiner Programme immer häufiger feststellen, dass Du zuvor erstellte Funktionen in verschiedenen Projekten wiederverwenden kannst, so dass sich der Entwicklungsprozess beschleunigt und Deinem Programmierstil anpasst.

In diesem Beispiel wird lediglich die Anzahl gültiger Klopfzeichen gezählt, egal wie viel Zeit dazwischen verstreicht. Du könntest dem Programm Komplexität hinzufügen, indem Du einen Timer mit `millis()` erstellst. Mithilfe des Timers kannst Du feststellen, ob die Klopfzeichen in einem bestimmten Zeitraum erfolgt sind. Falls Du eine Auffrischung zu der Funktionsweise von Timern benötigst, kannst Du im Sanduhr-Projekt nachschauen. Du bist natürlich auch nicht nur darauf beschränkt, Klopfzeichen in einem gewissen Lautstärkebereich zu detektieren. Du könntest komplexe Klopfmuster abfragen, die auf der Stärke der Erschütterung und dem Timing basieren. Es gibt etliche Beispiele online, die zur Umsetzung Hilfe bieten. Suche nach „Arduino knock lock", um mehr Projekte dieser Art zu entdecken.

Piezo-Elemente können als Eingänge benutzt werden, wenn Sie als Spannungsteiler mit einem starken Widerstand verwendet werden. Eine Funktion zu erstellen ist eine elegante Möglichkeit, wiederverwendbaren Code zu schreiben.



2

Befestige den Servo mit etwas Klebeband und achte darauf, dass sich der Arm leicht durch die Aussparung drehen kann.

LED

220-OHM-WIDERSTAND

1-MOHM-WIDERSTAND

METALLFOLIE

BERÜHRUNGSEMPFINDLICHE LAMPE

DU WIRST EINE LAMPE BAUEN, DIE SICH DURCH BERÜHRUNG EINES LEITENDEN MATERIALS EIN- UND AUSSCHALTET

Entdecke: Installation externer Bibliotheken, Berührungssensor bauen

Zeit: **45 MINUTEN**
Niveau: ■■■■□

Basierend auf Projekten: **1, 2, 5**

Du wirst die CapacitiveSensor-Bibliothek von Paul Badger für dieses Projekt verwenden. Mithilfe dieser Bibliothek kannst Du die elektrische Kapazität Deines Körpers messen.

Kapazität beschreibt die Aufnahmefähigkeit, d.h. wie viel elektrische Ladung etwas speichern kann. Die Bibliothek überprüft zwei Pins auf Deinem Arduino (einer ist ein Sender, der andere ein Empfänger) und misst die benötigte Zeit, bis beide den gleichen Zustand erreicht haben. Diese Pins werden an einen Metallgegenstand wie z.B. Alufolie angeschlossen. Wenn Du dem Metall näher kommst, absorbiert Dein Körper etwas dieser Aufladung, wodurch die Zustandsangleichung der zwei Pins länger dauert.

Die Bibliothek installieren

Die neueste Version der CapacitiveSensor-Bibliothek findest Du unter: ***arduino.cc/capacitive***. Lade die Datei auf Deinen Computer herunter und entpacke sie. Öffne Deinen Arduino Sketch-Ordner (standardmäßig in Deinem Ordner „Dokumente"). Erstelle in diesem Ordner ein neues Verzeichnis mit dem Namen „Bibliotheken ". Kopiere den zuvor entpackten CapacitiveSensor-Ordner in dieses Verzeichnis und starte die Arduino-Software neu. Klicke auf Datei -> Beispiele, um den neuen Eintrag „CapacitiveSensor" anzuzeigen. Die hinzugefügte Bibliothek enthält ein Beispiel-Projekt. Öffne das CapacitiveSensorSketch-Beispiel und kompiliere es. Sofern keine Fehlermeldungen erscheinen, war die Installation der Bibliothek erfolgreich.

Für mehr Information zu Bibliotheken:
arduino.cc/en/Reference/Libraries

BAUE DEN SCHALTKREIS

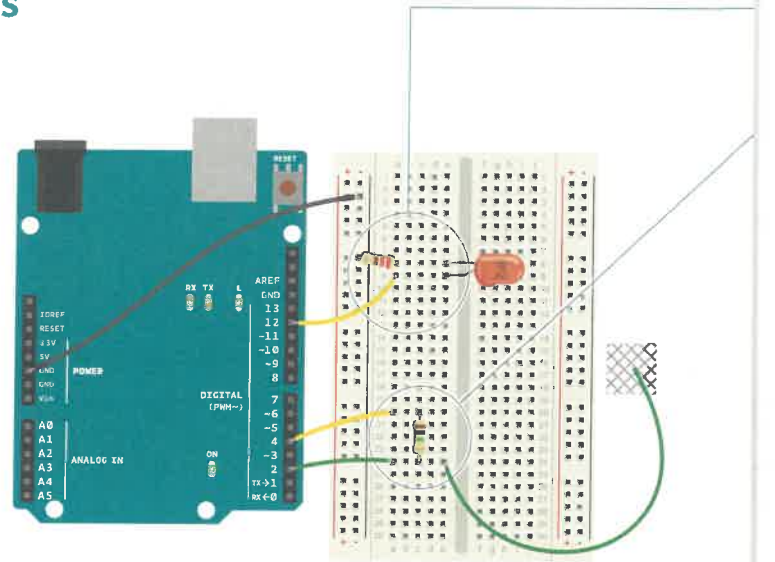


Abbildung 1

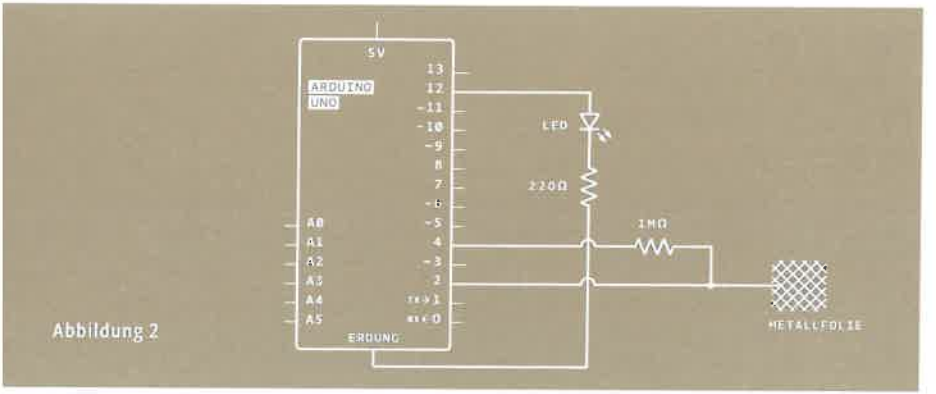


Abbildung 2

1. Schließe eine LED an Pin 12 an und verbinde entsprechend der Abbildung die Kathode über einen 220-Ohm-Widerstand mit der Erdung.

2. Schließe die digitalen Pins 2 und 4 an Deiner Steckplatine an. Schließe die beiden Pins mit einem 1-MOhm-Widerstand an. Setze ein langes Kabel (mindestens 8-10 cm) in der gleichen Reihe wie Pin 2 ein und führe es von der Steckplatine weg; am anderen Ende ist nichts angeschlossen. Das wird Dein Berührungssensor.

In diesem Projekt muss die Steckplatine nicht mit 5V versorgt werden, denn der digitale Pin 4 liefert den Strom an den Sensor.

Wie schon bei den vorigen LED-Projekten wird das Licht durch Zerstreuung wesentlich schöner. Pingpong-Bälle oder kleine Lampenschirme aus Papier oder Kunststoff können sich dazu eignen, je nachdem was Du zur Hand hast.

Du kannst den Sensor auch in etwas festem einhüllen und er funktioniert trotzdem noch. Kapazität kann durch nicht leitfähige Materialien wie Holz und Kunststoff gemessen werden. Durch Vergrößerung der leitenden Oberfläche des Sensors wird er empfindlicher; probiere z.B. Aluminiumfolie oder eine Kupfermatte an Dein Kabel anzuschließen. Du könntest einen Sockel für die Lampe aus Pappe, dünnem Holz oder Stoff bauen und die Innenseite mit Folie auskleiden, die an das Sensorkabel angeschlossen wird. Der gesamte Sockel der Lampe würde so als Berührungssensor agieren. Aktualisiere die Variable für die Schwelle im Code, falls Du diese Erweiterung vornimmst, damit Du weiterhin zuverlässige Resultate erhältst.

DAS PROGRAMM

CapacitiveSensor-Bibliothek importieren

Am Anfang Deines Programms lade die CapacitiveSensor-Bibliothek. Das geschieht auf die gleiche Weise wie bei einer normalen Arduino-Bibliothek, z.B. der `Servo`-Bibliothek aus einem früheren Projekt. Erstelle eine Instanz der Bibliothek. Hier legst Du fest, welche Pins für das Senden und Empfangen von Informationen benutzt werden sollen. In unserem Fall sendet Pin 4 durch den Widerstand zum leitenden Sensormaterial, und Pin 2 ist der Empfangs-Pin.

Schwelle definieren

Erstelle eine Variable für den Schwellwert, über dem sich die Lampe einschaltet. Du wirst diese Zahl nach dem Funktionalitätstest des Sensors noch anpassen.
Dann deklariere den Pin, an dem die LED angeschlossen ist.

Öffne eine serielle Schnittstelle mit 9600 bps im `setup()`, um die Werte zu sehen, die der Sensor ausliest. Definiere Deinen ledPin außerdem als Ausgang.

Berührung wahrnehmen

In `loop()` erstelle eine Variable des Typs Long, um den Sensorwert zu speichern. Die Bibliothek liefert mit dem Befehl `CapacitveSensor()` den Sensorwert zurück. Das Argument legt die Anzahl der Messungen fest, die Du abrufen willst. Bei einer zu geringen Anzahl werden größere Abweichungen zwischen den Auslesungen wahrscheinlich. Eine zu hohe Anzahl könnte hingegen zu Verzögerungen führen, da der Sensor wiederholt ausgelesen werden muss. 30 Messungen sind ein guter Anfangswert. Sende den Sensorwert an den seriellen Monitor.

Lampensteuerung

Überprüfe mit einem `if()...else`-Statement, ob der Sensorwert höher als der Schwellwert ist. Falls ja, schalte die LED ein. Falls nicht, schalte sie aus.

Dann füge eine kleine Verzögerung mit `delay()` vor dem Ende der Schleife ein.

```
#include <CapacitiveSensor.h>
CapacitiveSensor capSensor = CapacitiveSensor(4,2);
```

```
int threshold = 1000;
const int ledPin = 12;
```

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

```
void loop() {
  long sensorValue = capSensor.capacitiveSensor(30);
  Serial.println(sensorValue);
```

```
  if(sensorValue > threshold) {
    digitalWrite(ledPin, HIGH);
  }
  else {
    digitalWrite(ledPin, LOW);
  }
```

```
  delay(10);
}
```

BENUTZ ES

Nun, da der Arduino programmiert ist, wirst Du herausfinden wollen, welche Sensorwerte bei Berührung gemessen werden. Öffne den seriellen Monitor und vermerke den Sensorwert, wenn Du ihn nicht berührst. Drücke leicht auf das blanke Kabel, das von der Steckplatine kommt. Die Zahl sollte sich erhöhen. Drücke fester und schaue, ob sich der Wert ändert.

Sobald Du ein Gefühl für den Wertebereich des Sensors hast, gehe zurück zum Sketch und ändere den Schwellwert auf einen Wert der höher ist als der Sensorwert ohne Berührung, aber niedriger als der Wert bei Berührung. Lade den Sketch mit dem neuen Wert hoch. Das Licht sollte zuverlässig bei Berührung der Leitung angehen, und sich wieder ausschalten, wenn die Berührung aufhört. Wenn sich das Licht nicht einschaltet, versuche den Schwellwert ein wenig zu senken.

Vielleicht hast Du bemerkt, dass sich die Sensorwerte in Abhängigkeit von der Kontaktfläche mit Deinem Finger ändern. Könntest Du Dir das zunutze machen, um andere Interaktionsmöglichkeiten mit der LED einzurichten? Wie wäre es z.B. mit mehreren Sensoren, um die Lichtstärke zu variieren? Durch den Einsatz eines anderen Widerstands zwischen den Pins 2 und 4 wird die Empfindlichkeit beeinflusst. Könnte das für Deine Schnittstelle relevant sein?

Externe Bibliotheken wie die CapacitiveSensor-Bibliothek von Paul Badger sind nützliche Werkzeuge zur Erweiterung der Fähigkeiten des Arduino. Einmal installiert, verhalten sie sich ähnlich wie integrierte Bibliotheken, die mit der Arduino-Software geliefert wurden.

14

POTENTIOMETER

ARDUINO-LOGO UMGESTALTEN

MITTELS SERIELLER KOMMUNIKATION KANNST DU ÜBER DEINEN ARDUINO EIN PROGRAMM AUF DEINEM COMPUTER STEUERN

Entdecke Serielle Kommunikation mit einem Computerprogramm, Processing

Zeit: **45 MINUTEN**
Niveau: ■■■■■

Basierend auf Projekten: **1, 2, 3**

Nachdem Du viele coole Dinge in der physischen Welt angestellt hast, wird es Zeit Deinen Computer über den Arduino zu lenken. Wenn Du Deinen Arduino programmierst, öffnest Du eine Verbindung zwischen dem Computer und dem Mikrocontroller. Diese Verbindung kann auch dazu genutzt werden, Daten mit anderen Anwendungen auszutauschen.

Der Arduino verfügt über einen Chip, der die USB-basierte Kommunikation des Computers in die serielle Kommunikation des Arduino überträgt. Serielle Kommunikation bedeutet nichts weiter, als dass die zwei Computer, Dein Arduino und Dein PC, Informationen in Form von Bits seriell austauschen, d.h. nacheinander. Bei dieser Art der Kommunikation müssen sich die Computer derselben Geschwindigkeit des Datenaustauschs bedienen. Du hast sicherlich bei der Verwendung des seriellen Monitors unten rechts in der Ecke des Fensters eine Zahl bemerkt. Diese Zahl, 9600 Bits pro Sekunde oder Baud, ist identisch mit dem Wert, den Du bei `Serial.begin()` angegeben hast. Mit dieser Geschwindigkeit tauschen der Arduino und der Computer Daten aus. Ein Bit ist die kleinste Informationseinheit, die ein Computer verstehen kann. Bislang hast Du den seriellen Monitor dazu benutzt Werte von den analogen Eingängen anzuzeigen; Du wirst jetzt eine ähnliche Methode anwenden, um Werte an ein Programm zu senden, das Du in einer Programmierumgebung namens **„Processing"** schreiben wirst. Processing basiert auf Java und die Programmierumgebung des Arduino basiert auf Processing. Sie sind sich ziemlich ähnlich, also solltest Du keine Schwierigkeiten damit haben.

Bevor Du mit dem Projekt beginnst, lade die neueste Version von Processing von *processing.org* herunter. Möglicherweise könnten Dir die Tutorials „Getting Started" und „Overview" bei *processing.org/learning* den Einstieg erleichtern. Dadurch würdest Du Dich zunächst mit Processing vertraut machen, bevor Du beginnst, Software zur Kommunikation mit Deinem Arduino zu schreiben.

Die effizienteste Möglichkeit zum Datenaustausch zwischen dem Arduino und Processing bietet die Funktion `Serial.write()`. Sie ähnelt der dir bereits bekannten Funktion `Serial.print()`, mit der man Daten an einen verbundenen Computer senden kann. Anstelle von lesbaren Informationen wie Zahlen und Buchstaben sendet diese Funktion jedoch Byte-Werte zwischen 0-255. Das schränkt zwar die Werte ein, die der Arduino senden kann, aber ermöglicht dafür eine schnellere Übertragung von Informationen.

Auf Deinem Computer und dem Arduino gibt es einen seriellen Puffer, der die empfangenen Informationen zwischenspeichert, bis sie von einem Programm ausgelesen werden. Du wirst Bytes vom Arduino zum seriellen Puffer von Processing senden. Processing liest dann die Bytes aus dem Puffer. Sobald das Programm die Informationen vom Puffer ausgelesen hat, wird Platz für neue Daten geschaffen.

Bei der Einrichtung serieller Kommunikation zwischen Geräten und Programmen muss nicht nur die Geschwindigkeit des Austauschs festgelegt werden, sondern auch die Form der zu erwartenden Informationen. Wenn Du jemandem begegnest, erwartest Du vermutlich ein „Hallo!"; würde die Begrüßung stattdessen „Die Katze ist undeutlich" lauten, wärst Du wahrscheinlich verwirrt. Bei Software muss zunächst geklärt werden, was gesendet und empfangen wird.

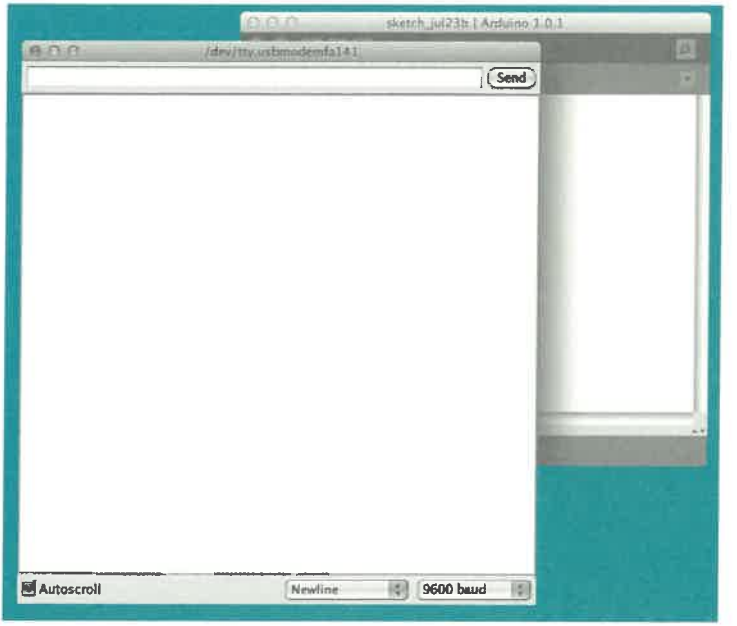


Abbildung 1

BAUE DEN SCHALTKREIS

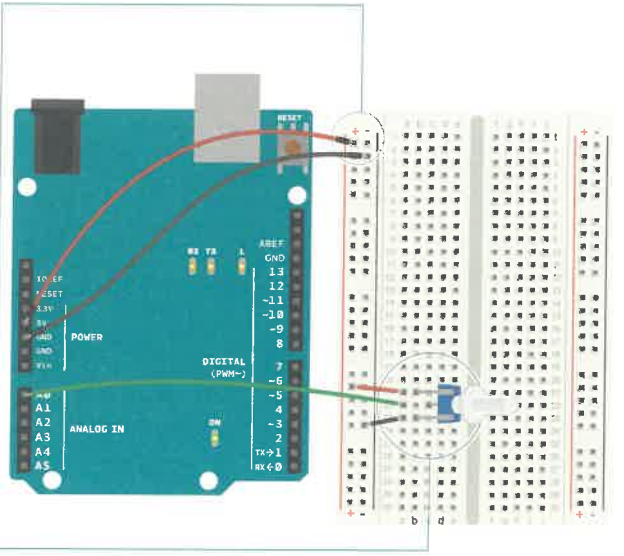


Abbildung 2

1. Schließe Strom und Erdung an Deiner Steckplatine an.

2. Schließe ein Ende Deines Potentiometers an Strom und das andere an Erdung an. Schließe den mittleren Stift am analogen Pin 0 an.

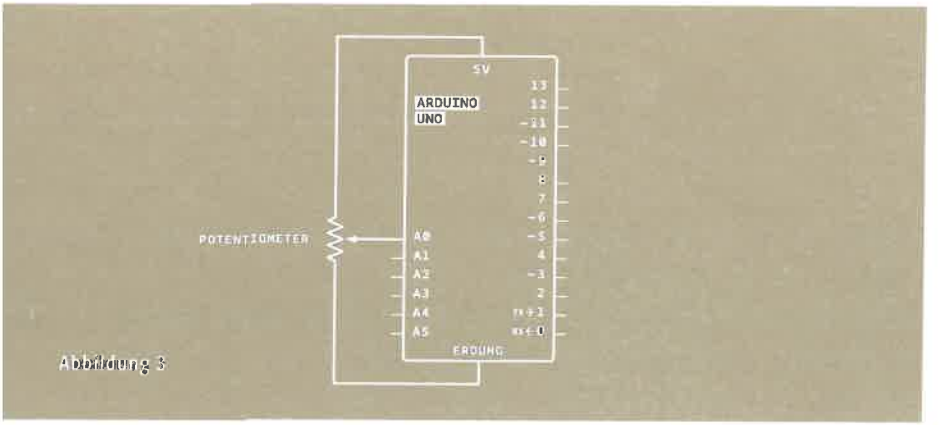


Abbildung 3

DER ARDUINO-CODE

Serielle Schnittstelle öffnen

Zuerst programmierst Du Deinen Arduino. Im `setup()` startest Du wieder die serielle Kommunikation, genauso wie beim Auslesen der Werte von einem angeschlossenen Sensor. Das Processing-Programm, das Du schreiben wirst, hat die gleiche Baudrate, wie Dein Arduino.

Sensorwert senden

In `loop()` verwendest Du den Befehl `Serial.write()`, um Informationen über die serielle Schnittstelle zu senden. `Serial.write()` kann nur einen Wert zwischen 0 und 255 senden. Um sicherzustellen, dass die ausgelesenen und weiterverschickten Messwerte in diesem Bereich liegen, teile sie durch 4.

ADC einpendeln lassen

Nach Versendung des Bytes geben wir dem ADC eine Millisekunde, um sich einzupendeln. Lade das Programm auf den Arduino hoch und widme Dich dem Processing-Sketch.

DER PROCESSING-CODE

Import und Einrichten des seriellen Objekts

Obgleich die Sprache von Processing der vom Arduino ähnelt, gibt es dennoch ausreichend Unterschiede, um einen Blick in die zuvor erwähnten Tutorials und die „Getting Started"-Anleitung zu rechtfertigen, damit Du mit der Sprache vertraut wirst.
Öffne einen neuen Processing-Sketch. Im Gegensatz zum Arduino kennt Processing keine seriellen Schnittstellen, ohne eine externe Bibliothek einzubinden. Importiere daher die serielle Bibliothek.
Wie auch zuvor bei der Servo-Bibliothek vom Arduino, musst Du das serielle Objekt instanziieren. Zur Verwendung der seriellen Schnittstelle wirst Du dieses benannte Objekt aufrufen.

Objekt für das Bild erstellen

Um Bilder in Processing zu verwenden, musst Du ihnen ein benanntes Objekt zuweisen.

```
1 void setup() {
2   Serial.begin(9600);
3 }
```

```
4 void loop() {
5   Serial.write(analogRead(A0)/4);
```

SPEICHERE UND SCHLIESSE DIE ARDUINO-IDE. LASS UNS BEGINNEN

```
6   delay(1);
7 }
```

```
1 import processing.serial.*;
2 Serial myPort;
```

```
3 PImage logo;
```

Variable für die Hintergrundfarbe

Erstelle eine Variable für die Hintergrundfarbe des Arduino-Logos. Das Logo ist eine .png-Datei mit transparentem Hintergrund, daher kann man durch Einstellung eines Farbtons eine Veränderung beobachten.

Processing hat wie auch der Arduino eine `setup()` Funktions. Hier wirst Du die serielle Kommunikation öffnen und dem Programm ein paar Parameter übergeben, die es zur Ausführung benötigen wird.

Farbmodus einstellen

Du kannst einstellen, wie Processing mit Farbinformationen umgehen soll. Normalerweise arbeitet es im Rot-Grün-Blau (RGB) Modus. Das entspricht der Farbmischung aus Projekt 4, als Du Werte zwischen 0 und 255 zur Änderung der Farbe einer RGB-LED eingesetzt hast. In diesem Programm wirst Du den HSB genannten Farbmodus verwenden, was für Hue (Farbton), Saturation (Farbsättigung) und Brightness (Helligkeit) steht. Den Farbton änderst Du mithilfe des Potentiometers.
`colorMode()` erwartet zwei Argumente: den Farbmodus und den höchsten zu erwartenden Wert.

Bild laden

Um das Arduino-Bild in den Sketch zu laden, lese es in das zuvor erstellte Logo-Objekt ein. Wenn Du die URL eines Bildes angibst, lädt Processing es herunter, sobald Du das Programm startest.
Mit der Funktion `size()` sagst Du Processing, wie groß das Anzeigefenster sein soll. Durch Verwendung von `logo.width` und `logo.height` als Argumente, passt der Sketch die Größe automatisch auf das eingelesene Bild an.

Verfügbare serielle Schnittstellen ausgeben

Processing kann Statusmeldungen mit dem Befehl `println()` ausgeben. Wenn Du ihn zusammen mit der `Serial.list()`-Funktion aufrufst, werden Dir alle vom Computer gefundenen seriellen Schnittstellen beim Programmstart angezeigt. Sobald die Programmierung abgeschlossen ist, kannst Du dadurch erfahren, an welchem Port Dein Arduino ist.

Serielles Objekt erstellen

Processing benötigt noch Informationen zur seriellen Verbindung. Um diese in das `myPort` Objekt einzuspeisen, muss dem Programm zunächst mitgeteilt werden, dass es sich um eine neue serielle Instanz handelt. Die erwarteten Parameter beschreiben, mit welcher Anwendung kommuniziert wird, der Port der seriellen Schnittstelle und die Geschwindigkeit.

```
int bgcolor = 0;

void setup() {

  colorMode(HSB, 255);

  logo = loadImage("http://arduino.cc/logo.png");
  size(logo.width, logo.height);

  println("Available serial ports:");
  println(Serial.list());

  myPort =
    new Serial(this, Serial.list()[0], 9600);
}
```

Das Attribut this sagt Processing, dass es die serielle Verbindung in dieser spezifischen Anwendung benutzen soll. Das Argument `Serial.list()[0]` spezifiziert, welche serielle Schnittstelle benutzt wird. `Serial.list()` enthält ein Array aller angeschlossenen seriellen Geräte. Das Argument `9600` sollte vertraut sein, es definiert die Geschwindigkeit, in der das Programm kommuniziert.
Die Funktion `draw()` erzeugt eine Endlosschleife analog zu der Funktion `loop()` beim Arduino. Hier wird auf das Programmfenster gezeichnet.

Arduino-Daten von der seriellen Schnittstelle auslesen

Überprüfe, ob Informationen vom Arduino eingegangen sind. Über den Befehl `myPort.available()` kannst Du herausfinden, ob sich etwas im seriellen Puffer befindet. Wenn dort Bytes eingetroffen sind, lies den Wert in die Variable `bgcolor` und gebe sie im Debug-Fenster aus.

Bildhintergrund einstellen und Bild anzeigen

Die Funktion `background()` stellt die Farbe des Fensters ein. Sie erwartet drei Argumente. Das erste Argument ist der Farbton, das zweite die Helligkeit, und das dritte die Farbsättigung. Verwende die Variable bgcolor als den Wert für den Farbton, und setze Helligkeit und Farbsättigung auf den Höchstwert von 255.
Du zeichnest das Logo mit dem Befehl `image()`. Du musst `image()` mitteilen, was es zeichnen und an welchen Koordinaten im Fenster begonnen werden soll. 0,0 ist der erste Punkt oben links, also starte dort.

BENUTZ ES

Verbinde Deinen Arduino und öffne den seriellen Monitor. Drehe den Knopf des Potentiometers auf Deiner Steckplatine. Du solltest verschiedene Schriftzeichen sehen, während Du am Knopf drehst. Der serielle Monitor erwartet ASCII-Schriftzeichen, nicht rohe Bytes. ASCII ist eine Zeichencodierung, um Text in Computern darzustellen. Was Du im Fenster siehst, ist der Versuch des seriellen Monitors die Bytes als ASCII zu interpretieren.

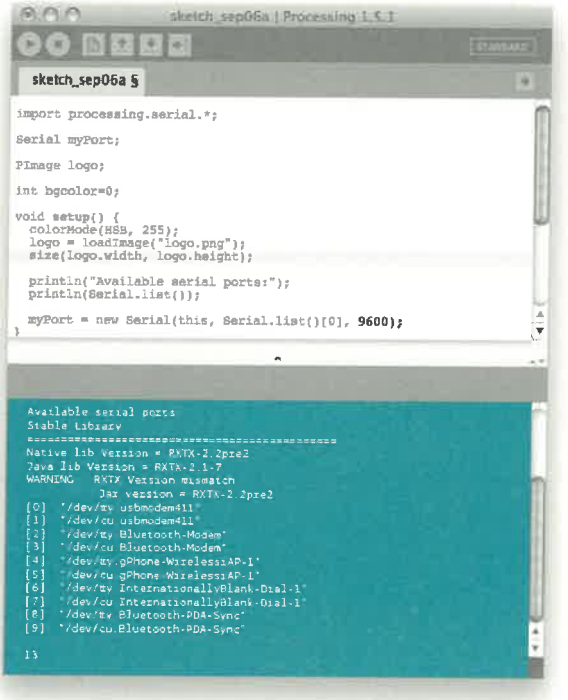
Mit `Serial.println()` werden die gesendeten Informationen für den seriellen Monitor transformiert. Mit `Serial.write()` hingegen sendest Du rohe, unveränderte Informationen.
Programme wie Processing können diese Bytes interpretieren.

```
13 void draw() {
```

```
14   if (myPort.available() > 0) {
15     bgcolor = myPort.read();
16     println(bgcolor);
17   }
```

```
18   background(bgcolor, 255, 255);
19 image(logo, 0, 0);
20 }
```

Schließe den seriellen Monitor. Starte den Processing-Sketch durch Drücken des Knopfs mit dem Pfeil in der Processing-IDE. Betrachte das Ausgabefenster von Processing. Du solltest eine Liste sehen, die jener in der folgenden Abbildung ähnelt.



Dies ist eine Liste aller seriellen Schnittstellen auf Deinem Computer. Falls Du OSX verwendest, suche nach einer Zeile in der Art „/dev/tty.usbmodem411", höchstwahrscheinlich das erste Element in der Liste. Unter Linux kann sie „/dev/ttyUSB0" oder ähnlich heißen. Unter Windows erscheint es als COM-Port, derselbe wie bei der Programmierung der Platine. Die Zahl vor jeder Zeile ist der Index im `Serial.list()[]`-Array. Ändere die Zahl in Deinem Processing-Sketch auf den entsprechenden Port auf Deinem Computer.

Starte den Processing-Sketch erneut. Sobald das Programm gestartet hat, drehe am mit dem Arduino verbundenen Potentiometer. Beim Drehen sollte sich die Farbe hinter dem Arduino-Logo ändern. Außerdem sollten die Messwerte im Processing-Fenster ausgegeben werden. Diese Zahlen entsprechen den Bytes, die vom Arduino gesendet werden.

Nachdem Du nach Herzenslust gedreht hast, ersetze das Potentiometer durch einen analogen Sensor. Suche Dir etwas Interessantes zur Steuerung der Farbe. Fühlt sich diese Interaktion für Dich natürlich an? Es ist sicherlich anders als mit einer Maus oder einer Tastatur.

Wenn Du serielle Kommunikation verwendest, kann nur eine Anwendung mit dem Arduino gleichzeitig sprechen. Das heißt solange ein Processing-Sketch mit dem Arduino kommuniziert, kannst Du keinen neuen Arduino-Sketch hochladen oder den seriellen Monitor benutzten. Dazu musst Du erst die laufende Anwendung schließen.

Mit Processing und anderen Programmierumgebungen kannst Du Medien auf Deinem Computer auf bemerkenswerte und neuartige Weisen kontrollieren. Wenn Du von den Möglichkeiten begeistert bist, Inhalte auf Deinem Computer zu kontrollieren, nimm Dir etwas Zeit, um mit Processing herum zu spielen. Es gibt etliche Beispiele zur seriellen Kommunikation sowohl in Processing wie auch in der Arduino-IDE, die Dir beim weiteren Erkunden helfen.

Serielle Kommunikation ermöglicht es dem Arduino, mit Programmen auf einen Computer zu sprechen. Processing ist eine Open-Source-Programmierumgebung, auf der die Arduino-IDE basiert. Es ist möglich, einen Processing-Sketch mit dem Arduino über serielle Kommunikation zu steuern.

15

OPTOKOPPLER

220-OHM-WIDERSTAND

TASTEN HACKEN

STEUERE ANDERE BAUTEILE UM DICH HERUM.
MIT EIN PAAR ZUSÄTZLICHEN SCHALTKREISEN KANNST
DU TASTEN MIT DEINEM ARDUINO „DRÜCKEN"

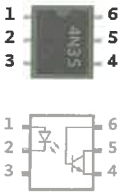
Entdecke: Optokoppler, Verbindung mit anderen Bauteilen

Zeit: **45 MINUTEN**
Niveau: ■■■■■

Basierend auf Projekten: **1, 2, 9**

Warnung: Wenn Du mit diesem Projekt beginnst, bist Du nicht länger ein Anfänger. Du wirst ein elektronisches Gerät öffnen und modifizieren. Dadurch wirst Du die Garantie auf dieses Gerät verlieren und, wenn Du nicht vorsichtig bist, könntest Du es beschädigen. Bevor Du Dich an dieses Projekt wagst, solltest Du mit allen elektronischen Konzepten der vorherigen Projekte vertraut sein. Wir empfehlen Dir für Deine ersten Versuche keine teuren Geräte zu benutzen, bei denen eine Beschädigung egal wäre, damit Du Erfahrung sammeln und Vertrauen gewinnen kannst.

Auch wenn der Arduino die verschiedensten Dinge steuern kann, ist es manchmal schlichtweg einfacher, sich für spezifische Zwecke hergestellter Hilfsmittel zu bedienen. Denke z.B. an die Steuerung eines Fernsehers, Musikwiedergabegeräts oder eines ferngesteuerten Autos. Die meisten elektronischen Geräte verfügen bereits über Bedienelemente mit Tasten, und viele dieser Tasten können gehackt werden, sodass Du sie mit dem Arduino „drücken" kannst. Die Steuerung von Audio-Aufnahme ist ein gutes Beispiel. Wenn Du Geräusche aufnehmen und abspielen möchtest, erfordert die Bewerkstelligung mit dem Arduino einigen Aufwand. Stattdessen ist es viel einfacher, die Tasten eines kleinen Geräts, das speziell für diese Aufgabe geschaffen wurde, mit den Ausgängen des Arduino zu ersetzen.



Optokoppler sind integrierte Schaltungen, mit denen man einen Schaltkreis über einen anderen steuern kann, ohne dass eine elektrische Verbindung zwischen ihnen besteht. Innerhalb eines Optokopplers befinden sich eine LED und ein Helligkeitssensor. Wenn die LED im Optokoppler durch Deinen Arduino eingeschaltet wird, schließt der Helligkeitssensor einen internen Schalter. Der Schalter ist an die zwei Ausgänge (4 und 5) des Optokopplers angeschlossen. Sobald der interne Schalter geschlossen wird, werden die zwei Ausgänge verbunden. Solange der Schalter geöffnet ist, sind sie nicht verbunden. Dadurch lassen sich Schalter von anderen Geräten schließen, ohne sie mit Deinem Arduino zu verbinden.

DAS PROGRAMM

Konstante deklarieren

Das Spannendste an diesem Projekt steckt im Schaltkreis und im Optokoppler. Der Code hingegen ist Deinem ersten Arduino-Projekt sehr ähnlich. Du wirst einen Ton alle 20 Sekunden spielen, indem Du Pin 2 auf `HIGH` schaltest.
Erstelle eine Konstante für den Optokoppler-Pin.

Pin-Richtung konfigurieren

Setze den Optokoppler-Pin als Ausgang im `setup()`.

Pin auf HIGH und LOW setzen

In `loop()` wird optoPin für einige Millisekunden auf `HIGH` gesetzt, damit der Optokoppler den Schalter im Gerät schließen kann. Danach wird optoPin wieder auf `LOW` gestellt.

Wartezeit

Warte für 21 Sekunden, damit die vollständige Aufnahme abgespielt werden kann, bevor `loop()` wieder von vorne beginnt.

BENUTZ ES

Schließe die Batterie am Audio-Aufnahmegerät an. Halte die Aufnahmetaste auf dem Gerät gedrückt. Währenddessen kannst Du über das Mikrofon aufnehmen. Nutze Deine Stimme, die Katze, oder Töpfe und Pfannen in der Küche, um Geräusche zu machen (aber sei behutsam mit der Katze).

Sobald Du mit der Aufnahme zufrieden bist, schließe Deinen Arduino über USB-Kabel an. Deine Aufnahme sollte jetzt abgespielt werden. Sofern Du die vollen 20 Sekunden aufgezeichnet hast, sollte die Wiedergabe gleich nach dem Ende wieder von vorne beginnen.

Probiere unterschiedliche Geräusche und Wartezeiten mit `delay()` in Deinem Programm aus.
Wenn Du den Schalter während der Wiedergabe betätigst, wird sie gestoppt. Wie könnte man das dazu nutzen, einzigartige Klangabfolgen zu erzeugen?

```
const int optoPin = 2;

void setup(){
  pinMode(optoPin, OUTPUT);
}

void loop(){
  digitalWrite(optoPin, HIGH);
  delay(15);
  digitalWrite(optoPin, LOW);

  delay(21000);
}
```

Integrierte Schaltungen (IC) sind in praktisch jedem elektronischen Gerät vorhanden. Der große 28-Pin Chip auf dem Arduino ist ein IC, der das Gehirn der Platine beherbergt. Es gibt noch weitere ICs, die diesen durch Kommunikation und Energie unterstützen. Der Optokoppler und der Hauptchip auf dem Arduino sind *Dual In-Line Package (DIP)* Chips. Diese DIP-Chips werden von den meisten Hobbyisten bevorzugt, da sie leicht auf eine Steckplatine passen und für die Verwendung nicht festgelötet werden müssen.

Dieses Beispielprojekt hat Audio lediglich in einem regelmäßigen Intervall abgespielt. Wie könntest Du die Eingänge aus früheren Projekten dazu verwenden, diese Klänge auszulösen? Welche anderen batteriebetriebenen Geräte hast Du noch zu Hause, die dringend einen Arduino zur Steuerung benötigen? Diese Technik, die Steuerung eines elektronischen Gerätes durch Anschluss eines Optokopplers an beiden Seiten eines Schalters, kann für viele andere Geräte genutzt werden. Welche anderen Geräte möchtest Du steuern?

Optokoppler können Geräte auf einem anderen Schaltkreis steuern. Die zwei Schaltkreise sind innerhalb des Bauteils elektrisch voneinander getrennt.

A/Z

(ADC) -
Aktor -
Analog
Analog-zu-Digital-Konverter -
Anode -
Argument -
Array -
Baud -
Beschleunigungsmesser -
Bibliothek -
Binär -
Bit -
Boolean -
Byte -
Datenblatt -
Datentyp -
Digital -
Drain (Transistor) -
Einschaltdauer -
Elektrizität -
Entkopplungskondensator -
Erdung -
Fehlersuche -
Fototransistor -
Fotozelle -
Gate -
Gleichstrom -
Globale Variable -
IDE -
Index -
Induktion -
Int -
Integrierte Schaltung (IC) -
Isolierung -
Kalibrierung -
Kapazität -
Kathode -
Konstante -
Kurzschluss -
Last -
LED mit gemeinsamer Kathode -
Leiter -
Lokale Variable -
Long -
Löten -
Mikrocontroller -
Millisekunde -
Objekt -
Ohm -
Ohmsches Gesetz -
Optokoppler -
Parallel -
Parameter -
Photowiderstand -
Polarisiert -
Processing -
Pseudocode -
Puffer -
Pulsbreitenmodulation (PWM) -
Rechteckwelle -
Reihe -
Rückspannung -
Schalter -
Schaltkreis -
Sensor -
Serielle Kommunikation -
Serieller Monitor -
Serieller Puffer -
Signalumformer -
Sketch -
Source (Transistor) -
Spannung -
Spannungsteiler -
Strom -
Stromstärke (Ampere) -
Stromversorgung -
Transistor -
Unsigned -
USB -
Variable -
Vorgang -
Wechselstrom -
Widerstand -
Zeitraum -

GLOSSAR

ES GIBT EINE VIELZAHL NEUER BEGRIFFE, DIE DU WÄHREND DIESER PROJEKTE GELERNT HAST.
WIR HABEN SIE HIER ALLE ZUR REFERENZ GESAMMELT.

A

Aktor - ein Bauteil, dblas elektrische Energie in Bewegung umwandelt. Motoren sind Aktoren.

Analog - etwas, das sich über einen Zeitraum kontinuierlich ändern kann.

Analog-Digital-Konverter (ADC) - eine Schaltung, die eine Analogspannung in eine entsprechende digitale Zahl umwandelt. Dieser Schaltkreis ist im Mikrocontroller integriert und ist an den Analogeingängen A0-A5 angeschlossen. Diese Umwandlung benötigt ein ganz bisschen Zeit, deswegen warten wir nach einem analogRead() immer mit einem kurzen delay().

Anode - das positive Ende eines Kondensators oder einer Diode (eine LED ist z.B. eine Diode).

Argument - Informationen, die einer Funktion übergeben werden. Um beispielsweise der Funktion digitalRead() mitzuteilen, welcher Pin von ihr abgerufen werden soll, erwartet sie als Argument die Pin-Nummer.

Array - in der Programmierung bezeichnet dies eine Gruppe von Variablen, die durch einen Namen gekennzeichnet ist und über einen Index den Zugriff auf einzelne Variablen erlaubt.

B

Baud - Kürzel für „Bits pro Sekunde", kennzeichnet die Geschwindigkeit, mit der Computer untereinander kommunizieren.

Beschleunigungsmesser - ein Sensor, der die Beschleunigung misst. Manchmal werden sie zur Richtungs- und Neigebestimmung benutzt.

Bibliothek/Library - Code, der die Funktionalität eines Programms erweitert. Im Falle der Arduino-Bibliotheken ermöglichen sie entweder die Kommunikation mit einer bestimmten Hardware oder die Transformation von Daten.

Binär - ein System mit nur zwei Zuständen, wie wahr/falsch oder aus/ein.

Bit - die kleinste Informationseinheit, die ein Computer senden oder empfangen kann. Es kann die Werte 0 und 1 annehmen.

Boolean - ein Datentyp zur Überprüfung, ob etwas wahr oder falsch ist.

Byte - 8 Bit Informationen. Ein Byte kann eine Zahl zwischen 0 und 255 speichern.

D

Datenblatt - ein Dokument, das von Ingenieuren für andere Ingenieure geschrieben wurde und das Design und die Funktionalität der elektrischen Bauteile beschreibt. Typische Informationen in einem Datenblatt sind die maximale Spannung und Strom, die ein Bauteil benötigt, sowie eine Beschreibung der Funktionsweisen der Pins.

Datentyp - Klassifizierungssystem, das festlegt, welche Werte eine bestimmte Konstante, Variable oder ein Array speichern kann. Int, float, Long und Boolean sind Typen, die im Arduino verwendet werden.

Digital - ein System mit diskreten (schrittweisen) Werten. Der Arduino ist z.B. ein digitales Gerät, er kennt nur zwei diskrete Werte, aus und ein, nichts dazwischen.

Drain (Transistor) - der Pin, der an die höhere Strom-/Spannungslast angeschlossen wird.

Dual In-Line Package (DIP) - eine Hülle für integrierte Schaltungen, die es erlaubt Bauteile leicht in eine Steckplatine einzusetzen.

E

Einschaltdauer - das Verhältnis über einen bestimmten Zeitraum, in dem ein Bauteil eingeschaltet ist. Wenn Du einen PWM-Wert von 127 (von insgesamt 256) verwendest, erhältst Du eine Einschaltdauer von 50%.

Elektrizität - Energie, die durch elektrische Ladung erzeugt wird. Du kannst elektronische Bauteile nutzen, um Elektrizität in andere Formen von Energie umzuwandeln, wie z.B. Licht und Wärme.

Entkopplungskondensatoren - Kondensatoren, die zur Glättung von Spannungsspitzen und -abfällen genutzt werden, oft in der Nähe von Sensoren und Aktoren platziert.

Erdung - der Punkt in einem Schaltkreis, an dem es 0 potenzielle elektrische Energie gibt. Ohne die Erdung kann Elektrizität im Schaltkreis nirgendwohin fließen.

F

Fehlersuche (Debugging) - Prozess des

schrittweisen Testens einer Schaltung oder eines Programms, um Fehler zu finden (auch „Bugs" genannt), bis das gewünschte Verhalten gezeigt wird.

Float - ein Datentyp, der auch Brüche (bzw. Gleitkommazahlen) speichern kann. Das heißt Zahlen werden mit einem Dezimalkomma gespeichert.

Fototransistor - Transistor, der durch Licht anstelle von Strom gesteuert wird.

Fotozelle - ein Bauteil, das Licht in elektrische Energie umwandelt.

Funktion - ein Abschnitt des Codes, der eine spezifische Aufgabe wiederholt durchführt.

G

Gate (Transistor) - der Pin, der am Arduino angeschlossen wird. Wenn das Gate mit 5V aktiviert wird, schließt es den Knotenpunkt zwischen Drain und Source und somit den angeschlossenen Schaltkreis.

Gleichstrom - Strom, der immer in die gleiche Richtung fließt. Alle Projekte in diesem Kit verwenden Gleichstrom.

Globale Variable - eine Variable, auf die überall innerhalb Deines Programms zugegriffen werden kann. Sie wird vor der Funktion setup() deklariert.

I

IDE - steht für „Integrated Development Environment" (integrierte Entwicklungsumgebung). In der Arduino-IDE schreibst Du die Software, die auf den Arduino hochgeladen wird. Sie enthält alle Funktionen, die der Arduino verstehen kann. Andere Programmierumgebungen, wie z.B. Processing, haben ihre eigene IDE.

Index - Kennzeichnung für ein Array, auf welches Element Du Dich beziehst. Computer sind null-basiert, d.h. sie fangen bei 0 anstatt bei 1 an zu zählen. Um z.B. auf das dritte Element des Arrays tones zuzugreifen, würdest Du tones[2] schreiben.

Induktion - ein Prozess, bei dem elektrische Energie zur Erzeugung eines Magnetfelds genutzt wird. Bewirkt in Motoren die Drehung.

Instanz - Kopie eines Software-Objektes. Du hast z.B. zur Verwendung der Servo-Bibliothek in den Projekten 5 und 12 benannte Instanzen von ihr erstellt.

Int - ein Datentyp, der eine ganze Zahl zwischen -32.768 und 32.767 speichert.

Integrierte Schaltung (IC) - ein Schaltkreis, der auf einem kleinen Stück Silikon hergestellt wurde und in Kunststoff oder Epoxid eingebettet ist. Herausführende Pins oder Stifte ermöglichen Dir mit dem enthaltenen Schaltkreis zu interagieren. Häufig können wir einen IC bereits gut gebrauchen, wenn wir nur wissen, was an die Pins angeschlossen wird, ohne dabei das Innere des ICs verstehen zu müssen.

Isolierung - etwas, das die Elektrizität am Fließen hindert. Leitende Materialien, wie z.B. Drähte, werden häufig mit Isolierungen, wie z.B. Gummi, ummantelt.

K

Kalibrierung - der Prozess der Einstellung von bestimmten Zahlen oder Bauteilen, um die besten

Resultate von einem Schaltkreis oder Programm zu erhalten. In Arduino-Projekten wird dies häufig gemacht, wenn Sensoren unterschiedliche Werte unter verschiedenen Bedingungen liefern können, beispielsweise bei der Messung der Lichtstärke, die auf einen Photowiderstand trifft. Kalibrierung kann automatisch (wie in Projekt 6) oder manuell (wie in Projekt 3) erfolgen.

Kapazität - Fähigkeit, elektrische Ladung zu speichern. Diese Ladung kann mit der Capacitive-Sensor-Bibliothek gemessen werden, wie in Projekt 13.

Kathode - das Ende eines Kondensators oder einer LED, das normalerweise an Erdung angeschlossen wird.

Konstante - eine Variable, die ihren Wert in einem Programm nicht ändern kann.

Kurzschluss - ein Kurzschluss zwischen Strom und Erdung stoppt Deinen Schaltkreis und sollte deshalb vermieden werden. In einigen Fällen kann er Deine Stromversorgung oder Teile Deines Schaltkreises beschädigen, und in seltenen Fällen kann sogar ein Feuer ausbrechen.

L

Last - ein Gerät, das elektrische Energie in eine andere Form umwandelt, wie Licht, Wärme oder Töne.

LED mit gemeinsamer Kathode - eine LED, die mehrere Farben im selben Gehäuse hat, mit einer Kathode und mehreren Anoden.

Leiter - etwas, durch das Elektrizität fließen kann, wie z.B. ein Kabel.

Lokale Variable - eine Variable, die nur für eine kurze Zeit gebraucht und anschließend vergessen wird. Eine innerhalb der Funktion setup() deklarierte Variable wäre lokal: nachdem setup() abgeschlossen ist, würde der Arduino die Variable aus dem Speicher löschen.

Long - ein Datentyp, der eine sehr große Zahl speichern kann, von -2.147.483.648 bis 2.147.483.647.

Löten - der Prozess der Herstellung einer elektrischen Verbindung, bei dem Lötzinn über den elektrischen Bauteilen oder Kabeln geschmolzen wird. Dadurch entsteht eine feste Verbindung zwischen den Bauteilen.

M

Mikrocontroller - das Gehirn des Arduino, ein kleiner Computer, den Du programmieren kannst, um auf Informationen zu hören, zu verarbeiten und anzuzeigen.

Millisekunde - 1/1000 Teil einer Sekunde. Der Arduino arbeitet seine Programme so schnell ab, dass delay() und andere zeitbasierte Funktionen in Millisekunden angegeben werden.

O

Objekt - Instanz einer Bibliothek. Zur Verwendung der Servo-Bibliothek hast Du eine Instanz mit der Bezeichnung myServo erstellt. myServo wäre hier das Objekt.

Ohm - Einheit zum Messen des Widerstands. Dargestellt mit dem Omega-Symbol (Ω).

Ohmsches Gesetz - mathematische Gleichung, die das Verhältnis zwischen Widerstand, Strom und Spannung darstellt. Gewöhnlich dargestellt als U (Spannung) = I (Strom) x R (Widerstand).

Optokoppler - Auch Optoisolator, Fotokoppler, Fotoisolator, Fotoschalter oder Optoschalter genannt. Eine LED wird in einem versiegelten Gehäuse mit einem Fototransistor kombiniert. Die LED wird so platziert, dass sie den Fototransistor anstrahlt, sobald sie eingeschaltet wird, woraufhin der Fototransistor schließt und der Strom fließen kann. Bietet einen hohen Grad der Isolierung, da keine gemeinsame elektrische Verbindung zwischen Ein- und Ausgang besteht.

Parallel - Bauteile, die an zwei gleichen Punkten im Schaltkreis platziert werden, sind parallel geschaltet. Sie haben immer den gleichen Spannungsabfall.

Parameter - bei der Deklaration einer Funktion dient der Parameter als Brücke zwischen den lokalen Variablen innerhalb der Funktion und den Argumenten, die sie beim Aufruf erhält.

Periode - eine bestimmter Zeitraum, in dem etwas geschieht. Die Periode bestimmt die Frequenz, d.h. die Häufigkeit, mit der etwas passiert.

Photowiderstand - ein Bauteil, das seinen Widerstand abhängig vom einfallenden Licht verändert.

Polarisiert - die Enden polarisierter Bauteilen (z.B. LEDs oder Kondensatoren) haben unterschiedliche Funktionen und müssen daher auf die richtige Weise angeschlossen werden. Polarisierte Bauteile können falsch herum angeschlossen nicht funktionieren, beschädigt werden oder andere Teile Deiner Schaltung beschädigen. Nicht polarisierte Bauteile (z.B. Widerstände) können beliebig angeschlossen werden.

Processing - eine auf Java basierende Programmiersprache. Wird häufig zur Einführung in die Konzepte der Programmierung und in Produktionsumgebungen verwendet. Die Arduino-IDE ist in Processing geschrieben und wird daher vertraut aussehen. Außerdem teilen sich Processing und Arduino die gleiche Philosophie und das gleiche Motiv, nämlich Open-Source-Hilfsmittel bereitzustellen, die auch Nicht-Technikern die Arbeit mit Hardware und Software ermöglichen.

Pseudocode - ein Bindeglied zwischen dem Schreiben in einer Computer-Programmiersprache und der natürlichen Sprache. Pseudocode sollte sich aus kurzen, expliziten Aussagen zusammensetzen.

Pulsbreitenmodulation (PWM) - eine Möglichkeit zur Simulation eines Analogausgangs mit einem Digitalgerät. PWM beinhaltet einen Pin sehr schnell ein- und auszuschalten. Das Verhältnis zwischen den EIN- und AUS-Zeiträumen bestimmt den simulierten Analogwert.

Rechteckwelle - eine Wellenform, die nur zwei Zustände hat, ein und aus. Wenn sie zur Erzeugung von Tönen verwendet wird, kann es „summend" klingen.

Reihe - Bauteile sind in Reihe geschaltet, wenn Strom nacheinander von einem in das nächste Bauteil fließt. Der durchfließende Strom ist bei beiden derselbe und die Spannung fällt jeweils ab.

Rückspannung - Spannung, die entgegen dem Strom zurückfließt, der sie erzeugte. Sie kann durch auslaufende Motoren verursacht werden und Schaltkreise beschädigen, deshalb werden Dioden oft in Verbindung mit Motoren verwendet.

S

Schalter - ein Bauteil, das einen elektrischen Schaltkreis öffnen oder schließen kann. Es gibt viele unterschiedliche Arten von Schaltern: die im Kit sind „tastend", d.h. sie schließen nur während sie gedrückt werden.

Schaltkreis - ein zirkulärer Pfad, der von einer Stromquelle über eine Last zurück zur anderen Seite der Stromversorgung führt. Strom fließt in einen Schaltkreis nur, wenn er geschlossen ist, d.h. der abgehende und eingehende Teil des Pfads wird nicht unterbrochen. Wenn ein Pfad unterbrochen oder offen ist, kann kein Strom durch den Schaltkreis fließen.

Sensor - ein Bauteil, das eine Form der Energie (wie Licht, Wärme oder mechanische Energie) misst und in elektrische Energie umwandelt, die der Arduino verstehen kann.

Serielle Kommunikation - die Art des Datenaustauschs zwischen dem Arduino und Computern bzw. anderen Geräten. Sie beinhaltet 1-Bit-Information nacheinander zu senden. Der Arduino hat einen USB-to-serial-Konverter auf der Platine, um mit Geräten zu kommunizieren, die keine dedizierte serielle Schnittstelle haben.

Serieller Monitor - integriertes Hilfsmittel der Arduino-IDE, der serielle Daten von einem angeschlossenen Arduino empfangen und senden kann. Siehe die Serial()-Funktionen.

Serieller Puffer - Platz im Speicher des Computers und des Mikrocontrollers, an dem über serielle Kommunikation empfangene Informationen gespeichert werden, bis sie von einem Programm ausgelesen werden.

Signalumformer - etwas, das eine Form von Energie in eine andere wandelt.

Sketch - Bezeichnung von Programmen, die in der Arduino-IDE geschrieben werden.

Source (Transistor) - der Pin eines Transistors, der an der Erdung angeschlossen wird. Wenn Strom am Gate eintrifft, werden Source und Drain miteinander verbunden und schließen die Schaltung.

Spannung - ein Maß potenzieller Energie, mit der eine Ladung in einem geschlossenen Schaltkreis bewegt wird.

Spannungsteiler - ein Schaltkreis, dessen Spannung an seinem Ausgang einen Bruchteil der Spannung seines Eingangs beträgt. Du baust z.B. einen Spannungsteiler, wenn Du einen Photowiderstand und einen festen Widerstand kombinierst, um einen Analogausgang zu erhalten. Ein Potentiometer ist ein weiteres Beispiel für einen Spannungsteiler.

Strom - Fluss der elektrischen Ladung durch einen geschlossenen Schaltkreis. Gemessen in Ampere.

Stromstärke (Ampere) - die Menge elektrischer Ladung, die an einem bestimmten Punkt in Deinem Schaltkreis fließt. Beschreibt den Stromfluss durch einen Leiter, wie z.B. ein Kabel.

Stromversorgung - Energiequelle, normalerweise eine Batterie, ein Transformator oder der USB-Anschluss Deines Computers. Existiert in vielen Variationen wie geregelt oder ungeregelt, Wechselstrom oder Gleichstrom. Für gewöhnlich wird die Spannung zusammen mit der maximalen Stromstärke angegeben, welche die Versorgung ohne Überlastung liefern kann.

T

Transistor - eine zumeist elektronische Vorrichtung mit 3 Anschlüssen, die entweder

als Verstärker oder als Schalter dienen kann. Eine Steuerspannung oder -strom zwischen zwei Anschlüssen steuert eine (üblicherweise) höhere Spannung oder Strom zwischen zwei anderen Anschlüssen. Gängige Typen von Transistoren sind der Bipolartransistor (BJT) und der Metall-Oxid-Halbleiter-Feldeffekttransistor (MOSFET). Sie werden häufig dazu eingesetzt, mit der niedrigen Stromstärke des Arduino (begrenzt auf 40 mA) einen wesentlich höheren Stromfluss zu steuern, wie ihn z.B. Motoren, Relais oder Glühlampen benötigen. Abhängig vom Design sind Transistoren entweder N-Kanal oder P-Kanal, was die Art ihres Anschlusses bestimmt.

Unsigned - ohne Vorzeichen. Eine Bezeichnung für Datentypen, die keine negativen Werte annehmen können. Wenn die möglichen Werte nur in eine Richtung gehen können, erhält man durch diese Kennzeichnung die doppelte Speicherkapazität. Wenn Du beispielsweise die Zeit mit millis() misst, empfiehlt es sich den Datentyp unsigned Long zu verwenden.

USB - Universal Serial Bus. Standardanschluss der meisten modernen Computer. Mithilfe eines USB-Kabels kann man einen Arduino über eine USB-Verbindung programmieren und mit Strom versorgen.

Variable - ein Platz im Speicher des Computers oder des Mikrocontrollers, um vom Programm benötigte Informationen zu speichern. Variablen speichern Werte, die sich während der Programmausführung wahrscheinlich ändern. Der Datentyp einer Variable hängt von der Art der Informationen ab, die Du speichern möchtest, sowie von der maximalen Größe der Informationen; z.B. kann ein Byte bis zu 256, aber ein int 65.536 unterschiedliche Werte speichern. Variablen können lokal in einem bestimmten Programmteil, oder global für das gesamte Programm deklariert werden (siehe globale Variable, lokale Variable).

Widerstand - Maß, wie effizient ein Material Elektrizität leitet. Der Widerstand kann mit dem Ohmschen Gesetz berechnet werden: R = U/I.

WEITERFÜHRENDE LITERATUR

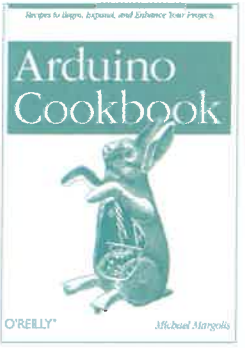
Getting Started with Arduino von **Massimo Banzi** [O'Reilly Media / Make, 2011]. Die maßgebliche Einführung in Arduino.

Getting Started with Processing von **Casey Reas** und **Ben Fry** [O'Reilly Media / Make, 2010]. Dieses kurze Handbuch zur Processing-Programmierumgebung liefert eine Einführung zur Programmierung von Grafiken, Klängen und Multimedia mit Deinem Computer.

Making Things Talk Auflage von **Tom Igoe** [O'Reilly Media / Make, 2011]. Geschrieben für erfahrene Arduino-Benutzer, zeigt Dir dieses Buch viele Techniken, um zwischen den Arduino-Mikrocontrollern und anderen Geräten im Internet und darüber hinaus zu kommunizieren.

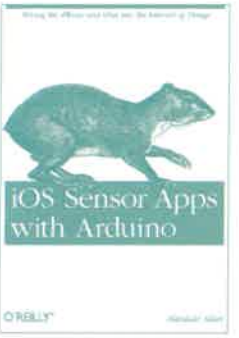
Learning Processing: A Beginner's Guide to Programming Images, Animation, and Interaction von **Daniel Shiffman** [Morgan Kaufman, 2009]. Eine detaillierte Einführung in die Programmierung mit Processing für Einsteiger jeden Alters.

Getting Started with RFID von **Tom Igoe** [O'Reilly Media / Make, 2012]. Eine kurze Einführung in die Verwendung von Radiofrequenz-Identifikation mit Arduino und Processing.



The Arduino Cookbook, 2. Auflage von **Michael Margolis** [O'Reilly Media / Make, 2011]. Dieses Buch liefert viele großartige Ideen, wie der Arduino auf fortgeschrittene Weise eingesetzt werden kann.

Making Things Move: DIY Mechanisms for Inventors, Hobbyists, and Artists von **Dustyn Roberts** [McGraw-Hill, 2010]. Ein großartige Quelle der Inspiration für die Erschaffung beweglicher Vorrichtungen und deren Integration mit Deinen Projekten.



Make: Electronics, von **Charles Platt** [O'Reilly Media / Make, 2009]. Clever geschriebene Einführung in die Elektronik, für jedermann geeignet. In diesem Buch werden zwar keine Arduinos verwendet, aber es ist ein wertvoller Text zum besseren Verständnis von Elektronik.

iOS Sensor Apps with Arduino, von **Alasdair Allan** [O'Reilly Media / Make, 2011]. Mit dieser kompakten Einführung lernst Du, wie man einen externen Sensor an ein iOS-Gerät anschließt und sie über den Arduino kommunizieren lässt. Du wirst auch eine iOS-App erstellen, welche die ausgelesenen Sensor-Werte in Echtzeit ausgibt und grafisch darstellt.