7 Motoren, Pumpen und Aktoren

Ein erstes Experiment mit einem Gleichstrommotor haben wir bereits in Kapitel 4 durchgeführt. Viele der Prinzipien, die Sie bei der Verwendung von Gleichstrommotoren lernen, lassen sich auch auf andere Dinge übertragen, die Sie mit einem Arduino oder Raspberry Pi steuern können.

Abbildung 7–1 zeigt eine Auswahl von Gleichstrommotoren. Wie Sie sehen, können diese Geräte alle verschiedenste Formen und Größen aufweisen.

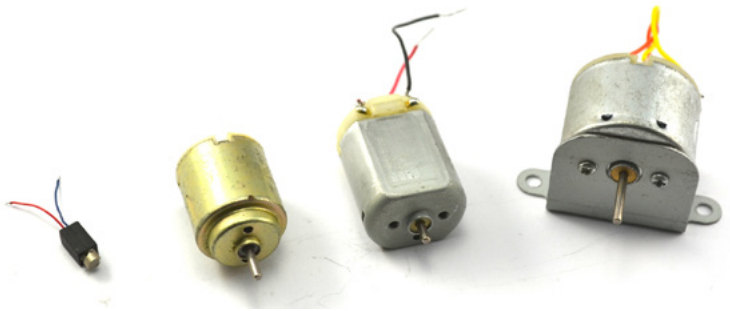


Abb. 7–1 *Verschiedene Gleichstrommotoren*

Motoren sorgen auch für den Antrieb vieler anderer nützlicher Geräte, wie z. B. Pumpen und Linearaktoren, die Sie weiter hinten in diesem Kapitel noch kennenlernen werden.

Wie Sie schon im Motorsteuerungsexperiment in Kapitel 4 erfahren haben, brauchen Gleichstrommotoren so viel Strom, dass Sie sie nicht direkt an den Ausgangspin eines Raspberry Pi oder Arduino anschließen können. Sie können sie allerdings mithilfe eines Transistors ein- und ausschalten.

In diesem Kapitel wollen wir als Erstes die Drehzahl eines Gleichstrommotors regeln.

Wie funktioniert ein Gleichstrommotor?

Gleichstrommotoren bestehen im Allgemeinen aus drei Hauptteilen, wie Sie in Abbildung 7–2 sehen. Sie verfügen über stationäre Magneten (Stator) entlang der Außenseite, einen Rotor (das Teil, das sich bewegt) und einen Kommutator.

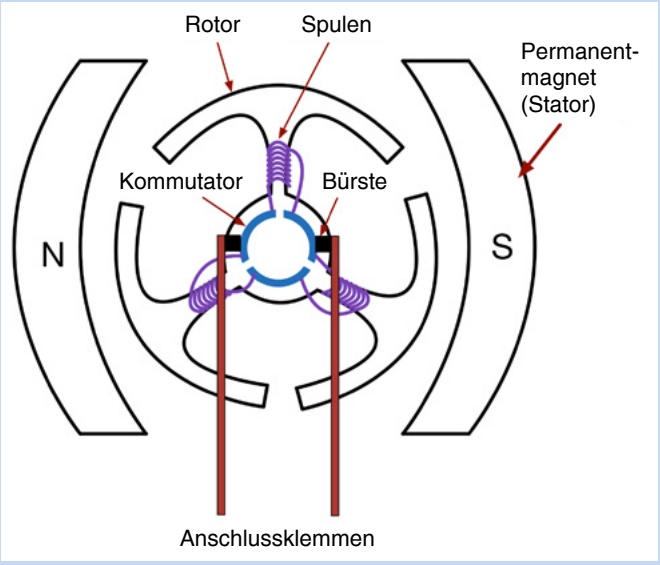


Abb. 7–2 *Innerer Aufbau eines Gleichstrommotors*

Um den Rotor ist Draht gewickelt. In Abbildung 7–2 sehen Sie drei Spulen an drei Formteilen des Rotors. Diese Spulen sind mit dem Kommutator verbunden. Dessen Aufgabe besteht darin, die Spulen nacheinander mit Strom zu versorgen, während sich der Rotor dreht, und zwar jeweils mit solcher Polarität, dass die aufeinander folgenden Spulen von den Permanentmagneten im Stator abgestoßen und angezogen werden. Insgesamt führt dies dazu, dass sich der Rotor dreht.

Der Kommutator besteht aus einem segmentierten Ring (in diesem Beispiel sehen Sie drei Segmente) und Bürsten, über die die Anschlussklemmen jeweils mit den einzelnen Segmenten des Kommutators verbunden werden, während sich der Kommutator mit dem Rotor dreht.

Die Konstruktion mit drei Spulen ist typisch für kleine Gleichstrommotoren der Art, wie Sie sie in Abbildung 7–1 gesehen haben.

Eine nützliche Eigenschaft von Gleichstrommotoren besteht darin, dass sie sich rückwärts drehen, wenn Sie die Polarität der Spannung an den Anschlussklemmen vertauschen.

Drehzahlregelung (PWM)

Im Experiment »Farben mischen« in Kapitel 6 haben Sie die Helligkeit von LEDs mithilfe der Pulsweitenmodulation geregelt. Mit demselben Prinzip können Sie auch die Drehzahl eines Motors regeln.

Experiment: Die Drehzahl eines Gleichstrommotors regeln

Für dieses Experiment verwenden Sie genau dieselbe Hardware wie in dem Experiment zur Motorsteuerung in Kapitel 4. Hier allerdings schalten Sie den Motor nicht einfach ein und aus, sondern regeln seine Drehzahl.

Die Hardware

Falls Sie das Experiment »Einen Motor steuern« aus Kapitel 4 noch nicht gebaut haben, dann sollten Sie das jetzt nachholen. Zur Erinnerung sehen Sie die Schaltung noch einmal in Abbildung 7–3.

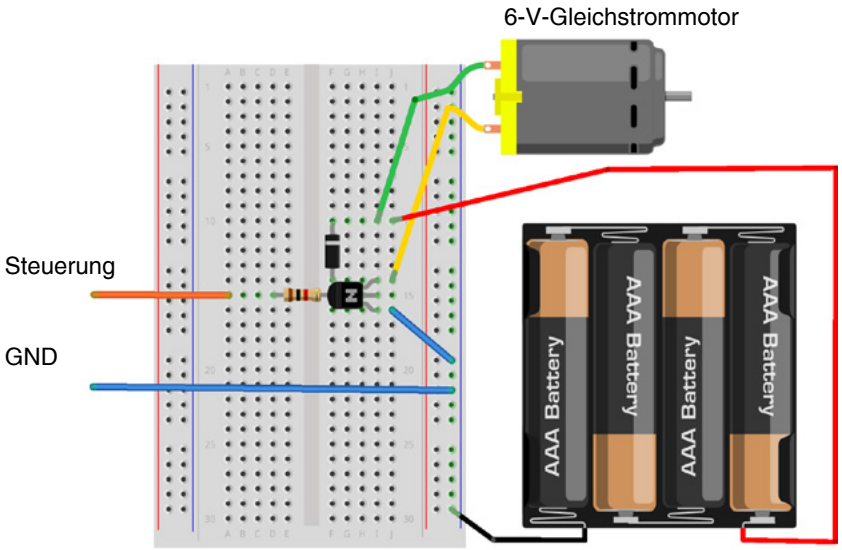


Abb. 7–3 *Schaltungsaufbau für die Drehzahlregelung*

Verbindungen mit dem Arduino

Schließen Sie den Arduino wie in Abbildung 7–4 gezeigt an. Verbinden Sie das Massekabel vom Steckbrett mit dem GND-Pin und die Steuerleitung mit Pin D9 des Arduino.

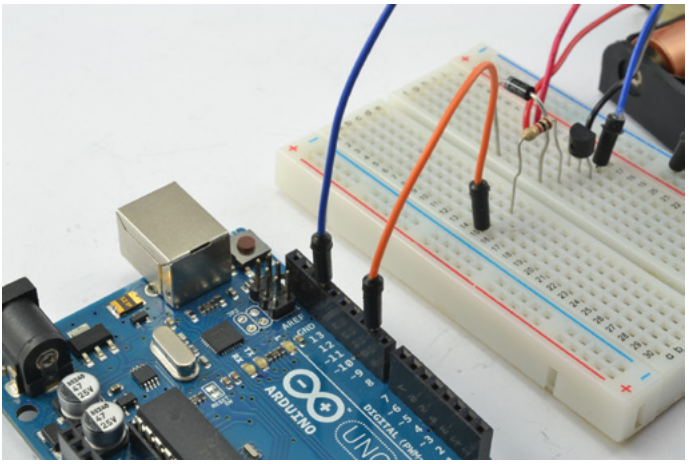


Abb. 7–4 *Anschluss des Steckbretts an den Arduino*

Die Software für den Arduino

Den Arduino-Sketch für dieses Projekt finden Sie im Verzeichnis */experiments/pwm_motor_control* an dem Speicherort, an dem Sie den Code zu diesem Buch abgelegt haben (siehe »Der Code zu diesem Buch« in Kapitel 2).

Dieses Programm verwendet den seriellen Monitor der Arduino-IDE. Dort können Sie einen Tastgrad eingeben, woraufhin die Motordrehzahl entsprechend geregelt wird.

```
const int controlPin = 9;

void setup() { // ❶
   pinMode(controlPin, OUTPUT);
   Serial.begin(9600);
   Serial.println("Enter Duty Cycle (0 to 100)");
}

void loop() { // ❷
   if (Serial.available()) { // ❸
      int duty = Serial.parseInt();
      if (duty < 0 || duty > 100) { // ❹
         Serial.println("0 to 100");
      }
      else {
         int pwm = duty * 255 / 100;
         analogWrite(controlPin, pwm);
         Serial.print("duty set to ");
         Serial.println(duty);
      }
   }
}
```

❶ Die `setup()`-Funktion definiert den Steuerpin (`controlPin`) als Ausgang und startet mithilfe von `Serial.begin()` die serielle Kommunikation, sodass Sie Tastgradwerte von Ihrem Computer an den Arduino senden können.

❷ Die Funktion `loop()` prüft mithilfe von `Serial.available()`, ob serielle Daten über die USB-Verbindung eingegangen sind.

❸ Wenn eine Nachricht in Form einer Zahl vorliegt, wird diese Zahl aus dem Zeichenstream gelesen und mithilfe von `parseInt()` in den Datentyp `int` umgewandelt.

❹ Es wird geprüft, ob sich der Wert im Bereich von 0 bis 100 befindet. Wenn nicht, wird über die USB-Verbindung ein Hinweis auf den erforderlichen Bereich an den seriellen Monitor gesendet.

Text und Zahlen

In diesem Buch werden wir noch häufig vom seriellen Monitor übermittelte Zahlen lesen. Daher lohnt es sich, zu erklären, was `parseInt()` macht und was hinter den Kulissen vor sich geht, wenn wir über die USB-Verbindung eine Nachricht an den Arduino senden.

Im Abschnitt »Serielle Kommunikation« am Ende von Kapitel 5 habe ich bereits angeführt, dass eine Kommunikation mit dem Arduino (und auch mit dem Raspberry Pi) über die Nutzung einer seriellen Schnittstelle möglich ist. Beim Arduino Uno ist die serielle Schnittstelle mit den Digitalpins D0 und D1 verbunden, weshalb Sie diese nicht als allgemeine digitale Ein- und Ausgangspins nutzen dürfen. Die Kommunikation zwischen dem Arduino und Ihrem Computer über diese serielle Schnittstelle erfolgt mithilfe eines USB-Schnittstellenchips, der zwischen der seriellen USB-Übertragung und der Art der direkten seriellen Kommunikation übersetzt, die der Arduino versteht.

Wenn Sie im seriellen Monitor eine Nachricht eingeben und an den Arduino senden, wird der Text in einen Bitstream umgewandelt (in `high`- und `low`-Signale), der beim Empfang wieder zu Gruppen aus je acht Bits (Bytes) rekonstruiert wird. Jedes dieser Bytes ist ein Zahlencode für einen Buchstaben des lateinischen Alphabets. Es gibt für alle Buchstaben und Ziffern einen eindeutigen Code, der im sogenannten ASCII-Standard definiert ist (American Standard Code for Information Interchange).

Das Programm auf dem Arduino, das die Nachricht über die serielle Verbindung empfängt, kann entweder ein Byte (Zeichen) nach dem anderen lesen oder die Funktion `parseInt()` verwenden, die fortlaufend Zeichen liest und zu einer Zahl zusammensetzt, solange es sich um Ziffern handelt. Nehmen wir an, die Zahl 154 wird übertragen. Dies geschieht in Form der drei Ziffern 1, 5 und 4. Stößt `parseInt()` dahinter auf einen Zeilenumbruch, ein Leerzeichen oder ein anderes Zeichen, das keine Ziffer ist, weiß die Funktion, dass sie alle Ziffern der Zahl empfangen hat, und gibt den Wert 154 als `int` zurück. Auch eine kurze Verzögerung nach der Übertragung einer Ziffer fasst `parseInt()` als das Ende der Zahl auf.

Fällt die Zahl dagegen in den Bereich zwischen 0 und 100, wird dieser Wert in eine Zahl zwischen 0 und 255 umgerechnet. Die Funktion `analogWrite()` stellt dann den entsprechenden PWM-Wert ein. Das ist notwendig, da die Arduino-Funktion `analogWrite()` Werte zwischen 0 und 255 entgegennimmt, wobei 0 einem Tastgrad von 0 % und 255 einem Tastgrad von 100 % entspricht.

Experimentieren mit dem Arduino

Klicken Sie auf dem Computer, von dem aus Sie den Arduino programmieren, oben rechts in der Arduino-IDE auf das Lupensymbol (eingekreist in Abb. 7–5), um den seriellen Monitor zu öffnen.

Darin werden Sie nun aufgefordert, einen Tastgrad zwischen 0 und 100 einzugeben. Probieren Sie verschiedene Werte aus und beobachten Sie, welche Auswirkungen das auf die Motordrehzahl hat.

Bei sehr niedrigen Werten, z. B. bei 10 oder vielleicht auch schon bei 20, kann es sein, dass sich der Motor nicht mehr dreht, sondern eher ein gequältes Jaulen von sich gibt, weil nicht mehr genug Leistung bei ihm ankommt, um die Reibung zu überwinden. Dieser Sketch bietet eine hervorragende Möglichkeit, den kleinsten noch brauchbaren Tastgrad für einen Motor, den Sie in einer praktischen Anwendung einsetzen wollen, herauszufinden.

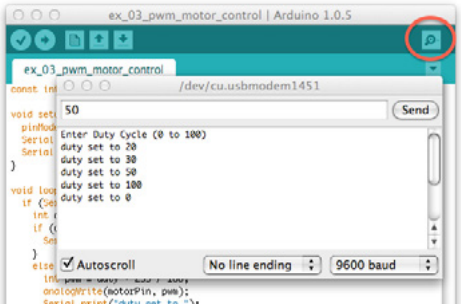


Abb. 7–5 *Drehzahlregelung über den seriellen Monitor*

Um den Motor anzuhalten, setzen Sie den Tastgrad einfach auf 0.

Verbindungen mit dem Raspberry Pi

Verbinden Sie das Steckbrett wie in Abbildung 7–6 gezeigt über Schaltdraht mit Buchse und Stecker mit dem Raspberry Pi. Schließen Sie das Massekabel vom Steckbrett an den GND-Pin an und die Steuerleitung an Pin 18 des Raspberry Pi.

Die Software für den Raspberry Pi

Die Software für den Raspberry Pi ist nach einem ähnlichen Muster aufgebaut wie die für den Arduino. Auch hier bittet Sie das Programm um Eingabe eines Werts für den Tastgrad und steuert Pin 18 dann entsprechend an.

Das Python-Programm für dieses Projekt finden Sie in der Datei *pwm_motor_control.py* im Verzeichnis *python/experiments/* (an dem Speicherort, an dem Sie den Code zu diesem Buch abgelegt haben – siehe den Abschnitt »Der Code zu diesem Buch« in Kapitel 3).

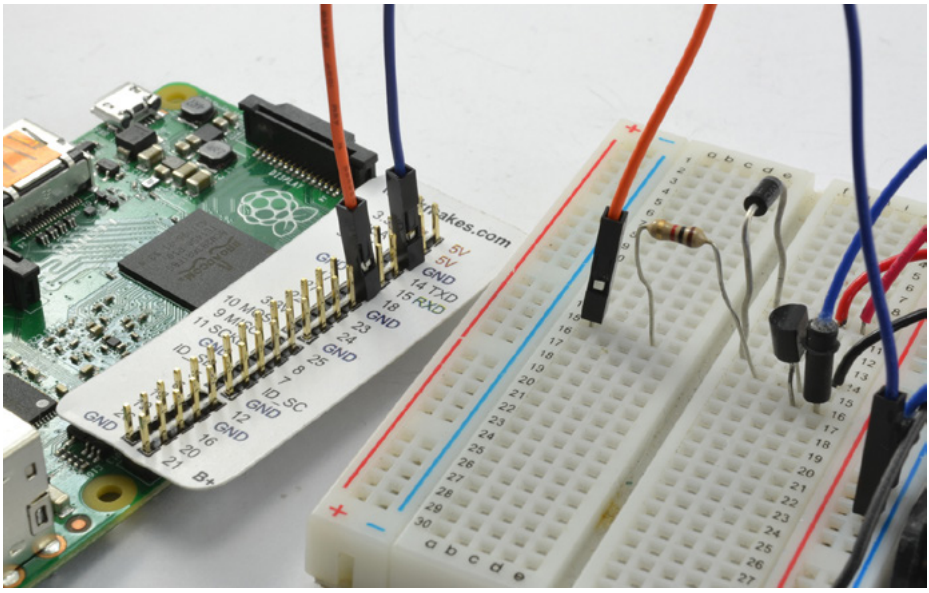


Abb. 7–6 *Anschluss des Steckbretts mit der Motorsteuerung an den Raspberry Pi*

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

control_pin = 18 # ❶

GPIO.setup(control_pin, GPIO.OUT)
motor_pwm = GPIO.PWM(control_pin, 500) # ❷
motor_pwm.start(0) # ❸

try:
   while True: # ❹
      duty = input('Enter Duty Cycle (0 to 100): ')
      if duty < 0 or duty > 100:
         print('0 to 100')
      else:
         motor_pwm.ChangeDutyCycle(duty)

finally:
   print("Cleaning up")
   GPIO.cleanup()
```

❶ Der erste Teil des Programms bis zur Definition von Pin 18 als Ausgabe ist identisch mit dem Code für das Motorsteuerungsexperiment aus Kapitel 4.

❷ In dieser Zeile wird der Pin als PWM-Ausgang festgelegt. Mithilfe der Bibliothek `RPi.GPIO` können Sie dazu jeden der GPIO-Pins verwenden. Der Parameter `500` legt die PWM-Frequenz auf 500 Hz fest.

❸ Die Ausgabe am PWM-Ausgang beginnt erst, wenn `start` aufgerufen wird. Der Parameter dieser Funktion ist der anfängliche Tastgrad. Da der Motor zu Beginn ausgeschaltet sein soll, setzen wir ihn auf `0`.

❹ In der Hauptschleife wird der Tastgrad, also der Wert von `duty`, mithilfe von `input` angefordert und anschließend überprüft. Liegt der Wert zwischen 0 und 100, so wird er mithilfe der Funktion `ChangeDutyCycle` als Tastgrad festgelegt.

Experimentieren mit dem Raspberry Pi

Führen Sie das Programm mithilfe von `sudo` als Superuser aus und versuchen Sie dann, verschiedene Werte für den Tastgrad einzugeben. Die Motordrehzahl ändert sich entsprechend, wie Sie es auch schon beim Arduino gesehen haben.

```
pi@raspberrypi ~/make_action/python $ sudo python pwm_motor_control.py
Enter Duty Cycle (0 to 100): 50
Enter Duty Cycle (0 to 100): 10
Enter Duty Cycle (0 to 100): 100
Enter Duty Cycle (0 to 100): 0
Enter Duty Cycle (0 to 100):
```

Gleichstrommotoren über ein Relais steuern

Wenn Sie mit dem Arduino oder Raspberry Pi nur gelegentlich einen Motor ein- und ausschalten müssen, können Sie dazu auch ein Relais verwenden. Das wird zwar meistens als eine ziemlich altmodische Vorgehensweise abgetan, bietet aber eine Reihe von Vorteilen:

- Es lässt sich leicht realisieren und erfordert nur wenige Bauteile.
- Es bietet eine sehr gute Isolierung zwischen dem empfindlichen Arduino oder Raspberry Pi und dem Motor mit seinen hohen Strömen und seinem starken elektrischen Rauschen.
- Mit dem richtigen Relais können Sie hohe Ströme schalten.
- Vorgefertigte Relaismodule lassen sich unmittelbar an den Raspberry Pi oder Arduino anschließen.

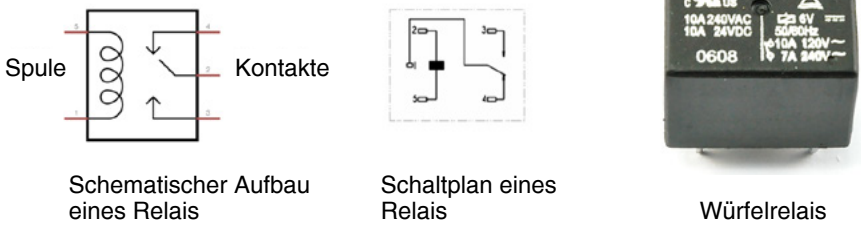
Allerdings gibt es auch Nachteile:

- Die Bauteile sind relativ groß.
- Mit einem Relais können Sie einen Motor nur ein- und ausschalten, aber nicht die Drehzahl regeln.
- Relais sind elektromechanische Geräte mit einer typischen Lebensdauer von 10.000.000 Schaltvorgängen.

Elektromechanische Relais

Abbildung 7–7 zeigt den wahrscheinlich am häufigsten verwendeten Typ von Relais, das sogenannte *Würfelrelais*.

Das allgemeine Funktionsprinzip solcher elektromechanischen Relais besteht darin, dass die Spule im Innern wie ein Elektromagnet wirkt, sobald ein Strom (von ca. 50 mA) hindurchfließt, und die Schalterkontakte schließt. Diese Kontakte können hohe Ströme und hohe Spannungen handhaben und sind in der Lage, Stromstärken im zweistelligen Amperebereich zu schalten.



Schematischer Aufbau eines Relais

Schaltplan eines Relais

Würfelrelais

Abb. 7–7 *Relais*

In diesem Kapitel verwenden Sie ein Relais, um einen Motor zu steuern. Da es aber praktisch wie ein Schalter funktioniert, können Sie es für so ziemlich alle Geräte einsetzen.

Relais wie diese werden als SPCO (Single Pole Change Over) bezeichnet, da sie nicht nur zwei Kontakte haben, die entweder miteinander verbunden sind oder nicht, sondern drei. Neben dem gemeinsamen Kontakt, der gewöhnlich mit COM beschriftet ist, gibt es noch die Kontakte NO (normalerweise geöffnet) und NC (normalerweise geschlossen), wobei »normalerweise« in diesem Zusammenhang bedeutet, solange die Spule nicht bestromt ist. Das heißt, die Verbindung zwischen NO und COM ist offen, bis Strom durch die Spule fließt. Beim NC-Kontakt ist es

genau anders herum, d.h., NC und COM sind »normalerweise« verbunden und werden getrennt, sobald die Spule mit Strom versorgt wird.

Im Allgemeinen werden nur die Kontakte NO und COM zum Schalten verwendet.

Ein Relais mit dem Arduino oder dem Raspberry Pi schalten

Für Projekte mit Raspberry Pi oder Arduino sollten Sie Relais mit einer Spulenspannung von 5 V auswählen. Die Relaisspulen ziehen zu viel Strom (ca. 50 mA), um sie direkt an einem Raspberry Pi oder Arduino zu betreiben, weshalb wir in beiden Fällen einen kleinen Transistor einsetzen, um die Relaisspulen zu schalten.

Den Schaltplan dafür sehen Sie in Abbildung 7–8.

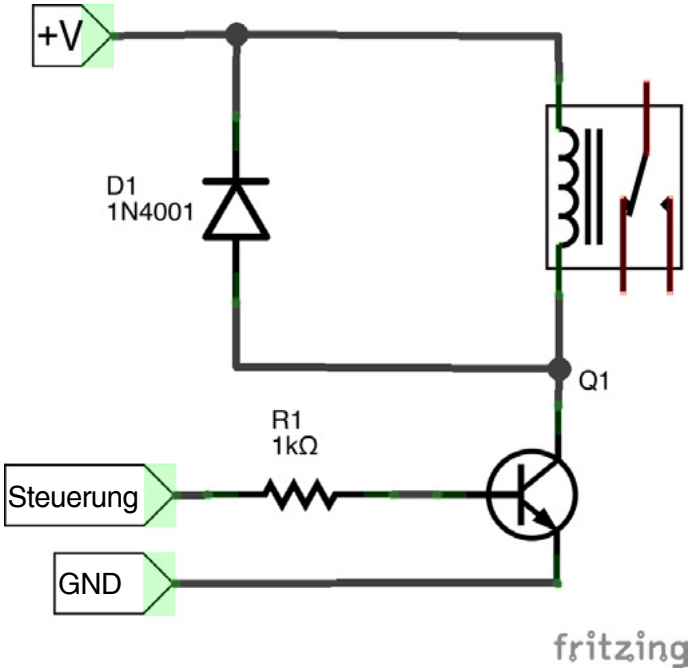


Abb. 7–8 *Schalten eines Relais mit einem kleinen Transistor*

Die Spule des Relais soll bei 5 V arbeiten und ungefähr 50 mA Strom ziehen. Das ist ein kleines bisschen zu viel für die GPIO-Pins des Arduino und viel zu viel für die des Raspberry Pi. Ebenso wie beim Motorsteuerungsexperiment in Kapitel 4 verwenden Sie daher einen Transistor, um die Last zu steuern (in diesem Falle die Relaisspule anstelle des Motors).

Dieser Aufbau ist nur dann sinnvoll, wenn die zu steuernde Last einen so hohen Stromverbrauch hat, dass sie nicht direkt vom Arduino bzw. Raspberry Pi gesteuert werden kann.

Ebenso wie ein Motor kann auch ein Relais beim Ein- und Ausschalten Spannungsspitzen erzeugen, weshalb wir in der Schaltung auch eine Diode benötigen.

In dem Schaltplan in Abbildung 7–8 können Sie gut erkennen, dass der Schalter des Relais elektrisch komplett von der Spule isoliert ist. Dadurch ist die Wahrscheinlichkeit geringer, dass elektrisches Rauschen, Spannungsspitzen oder andere schlechte elektrische Einflüsse ihren Weg zum Arduino bzw. Raspberry Pi finden.

Da das Relais nur etwa 50 mA benötigt, reicht ein schwacher 2N3904-Transistor, den Sie schon für Centbeträge bekommen, völlig aus.

Relaismodule

Wenn Sie mehrere Geräte steuern wollen und bei allen die bereits erwähnten Einschränkungen hinnehmen können, die sich bei der Verwendung von Relais ergeben (nur Ein-/Ausschalten möglich), können Sie zur Vereinfachung auch ein Relaismodul wie das in Abbildung 7–9 kaufen.

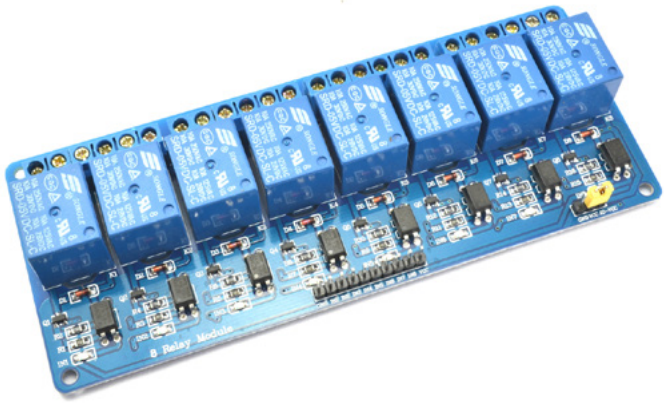


Abb. 7–9 *Ein 8-Kanal-Relaismodul*

Solche Module können Sie kostengünstig bei Amazon oder eBay erwerben. Neben den Relais enthalten sie auch die erforderlichen Transistoren sowie kleine LEDs, die Ihnen mitteilen, ob die einzelnen Relais eingeschaltet sind oder nicht. Da die Transistoren bereits eingebaut sind, können Sie diese Module direkt an den Raspberry Pi oder Arduino anschließen.

Es gibt Module mit einem einzigen Relais, aber auch solche, die noch mehr als die acht Relais steuern, die das Modul in der Abbildung aufweist.