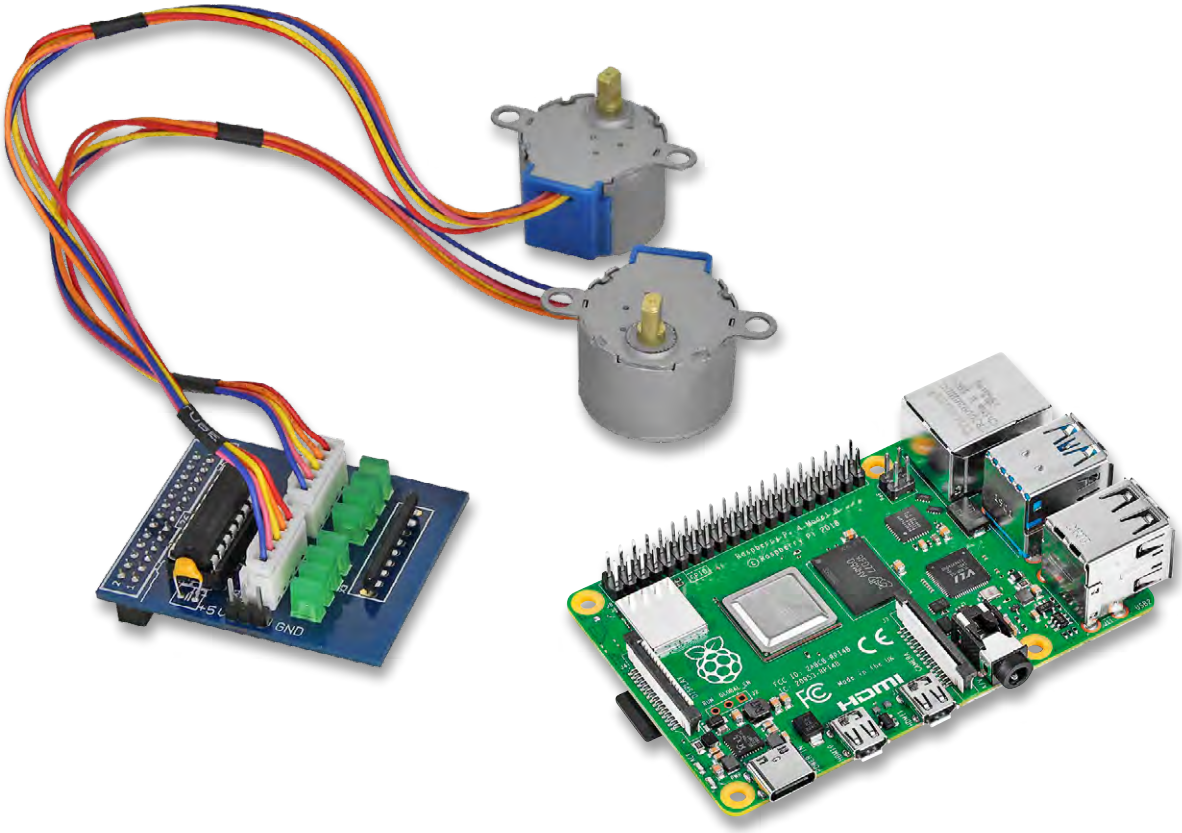
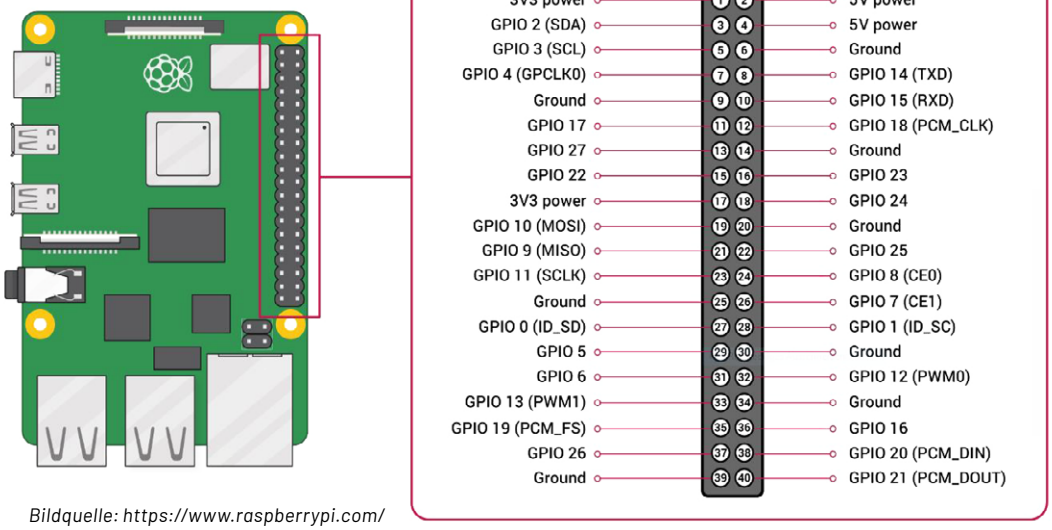
Schritt für Schritt

Der Raspberry Pi macht seine ersten Schritte

Nachdem im Grundlagenteil dieser Artikelserie die verschiedenen Arten von Schrittmotoren, ihr Aufbau, die Art der Wicklungsbestromung sowie ihre Vor- und Nachteile theoretisch erläutert wurden, geht es in diesem Artikel um den praktischen Betrieb eines Steppermotors am Raspberry Pi.





Bildquelle: https://www.raspberrypi.com/documentation/computers/raspberry-pi.html

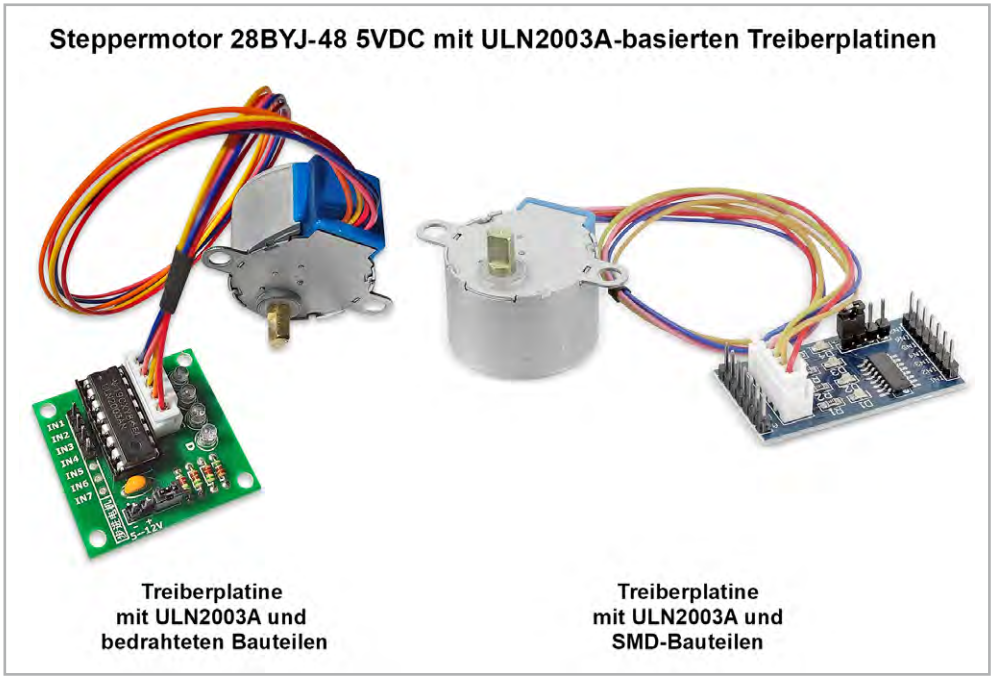


Bild 1: Dieses Set – bestehend aus einem Schrittmotor und einer Treiberplatine zu dessen Ansteuerung – ist für weniger als 5 € zu haben.

Kleine Kosten – großer Lerneffekt

Das erste praktische Beispiel zum Betrieb am Raspberry Pi sollte nicht an den Kosten scheitern. Wer bereits einen Raspberry Pi mit T-Cobbler und Steckbrett besitzt, muss nur je nach Einkaufsquelle zwischen 2 € und 5 € ausgeben, um lehrreiche Experimente ausführen zu können. Die Hardwaregrundlage dazu bildet der weitverbreitete Schrittmotor 28BYJ-48 5VDC, zu dessen Bestromung Breakoutplatinen mit dem Siebenfach-Transistorarray ULN2003A als Treiber dienen, die von mehreren Herstellern stammen und weitgehend funktionsgleich sind (Bild 1).

Der Motor verfügt über ein Untersetzungsgetriebe, das die Rotordrehzahl durch 64 teilt. Allerdings kann man sich auf den Untersetzungswert wohl nicht immer verlassen, denn es gibt Berichte im Internet, nach denen bei ein und derselben Typenbezeichnung verschiedene Getriebe verbaut wurden (Bild 2). Ähnliche Variationen sind bezüglich Wicklungswiderstand der Phasen anzutreffen. Im Internet finden sich interessante Ergebnisse zu einer Vielzahl untersuchter Exemplare. Am einfachsten ist es, mit einem kleinen Programm zu testen, wie viele Step-Impulse für eine Volldrehung der Abtriebsachse (also um 360°) erforderlich sind. Beim Exemplar des Autors waren es 512.

Das Schaltschema des Motors zeigt Bild 3. Man sieht, dass die Mittelanzapfungen beider Phasenwicklungen zu einem fünften Anschluss zusammengeführt sind, der mit +5 V verbunden ist (roter Draht). Somit ist nur der unipolare Betrieb möglich. Dabei werden die Phasenenden A, A', B, B' abwechselnd nach dem sogenannten Phasenschema vom Treiber-IC auf Masse gelegt, wodurch ein Magnetisierungsstrom von der Mittelanzapfung auf den geerdeten Phasenanschluss fließt. Das entstehende Spulenmagnetfeld bewirkt die schrittweise Drehung des Dauermagnetrotors. Die Wicklungswiderstände zwischen der Mittelanzapfung und den Enden betragen beim Exemplar des Autors ca. 21,5 Ω. Bei einer Spannung von 5 VDC ergibt sich also ein Strom 5/21,5 = 93 mA, was einer Leistung von 1,16 W im Stillstand entspricht.

Bild 2: Die Rotordrehung des Steppermotors wird durch ein mechanisches Getriebe auf 1/64 reduziert (meistens).

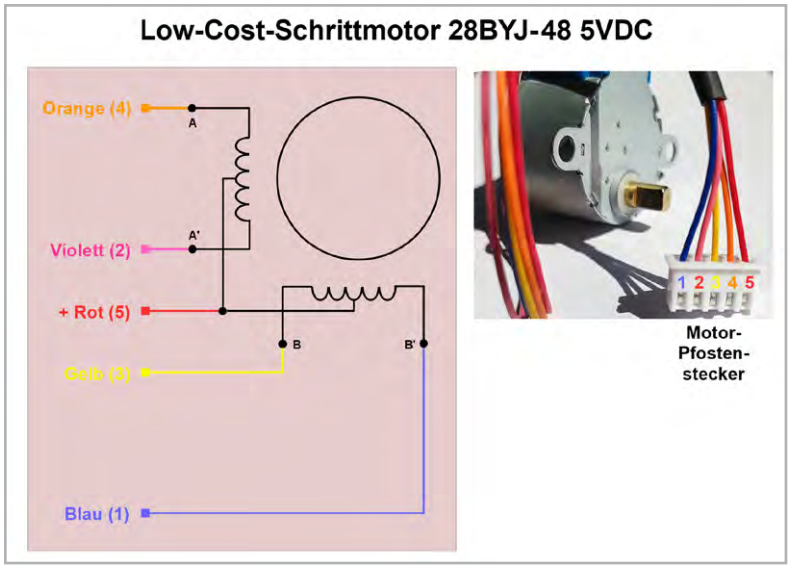


Bild 3: Durch die auf + VDD (positive Versorgungsspannung) gelegten Mittelanzapfungen ist nur eine unipolare Bestromung der Phasenspulen möglich.

Die Treiberschaltung beruht auf dem Seven-Darlington-Array ULN2003A. Sein Innenleben erläutert Bild 4. Jeder Darlington-npn-Transistor invertiert, d. h., wenn an seinem Eingang ein HIGH liegt, werden die Transistoren leitend und legen eine an die Kollektoren angeschlossene Last auf Masse (LOW). In unserem Fall die Spulenenden A, A', B, B', die über die Mittelanzapfung mit Plus verbunden sind. Man nennt einen solchen Schalter auch Low Side Switch. Als Schutz der Transistoren für das (Ab-)Schalten von induktiven Lasten sind bereits sieben Freilaufdioden (clamp diodes) integriert, deren Kathoden über IC-Pin 9 auf +5 V gelegt werden.

Die ULN2003A-Version im DIP-16L-Gehäuse (ca. 30 Cent) kann man direkt auf das Breadboard stecken und sich so die Treiberplatine ersparen. Diese ist allerdings nur unwesentlich teurer und bereits mit einer Steckverbindung für den Stecker an den fünf Motoranschlussdrähten versehen. In der Regel sind auf der Platine auch vier LEDs verbaut, die bei durchgeschaltetem Darlington 1 bis 4 leuchten. Es gibt Platinen mit bedrahteten Bauelementen und solche mit SMD-Bestückung aus verschiedenen Quellen, die jedoch fast alle funktionsgleich sind (Bild 4).

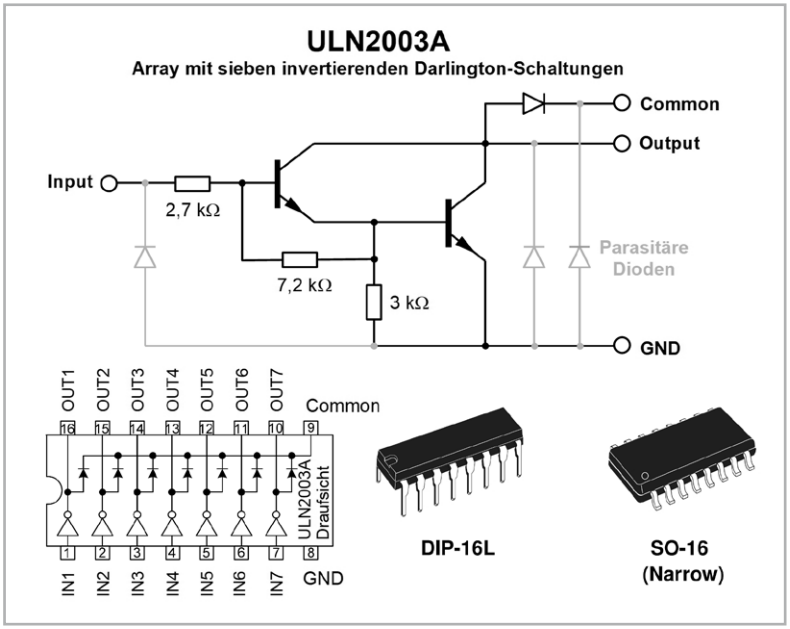


Bild 4: Das ULN2003A ist ein Chip, der sieben invertierende Darlingtonstufen nebst Freilaufdioden zum Schalten induktiver Lasten enthält.

Der Schaltplan der mit bedrahteten Bauelementen bestückten Treiberplatine ist in Bild 5 zu sehen. Die Bezeichnungen der Eingangspins (IN1 bis IN4) und der zur Stromversorgung (-, + 5–12 V) entsprechen der Platinenbedruckung.

Für die Bestromung der Statorspulen unterscheidet man drei Betriebsarten:

- Wave Drive (wellenförmige Bestromung)
- Full Step (Vollschritt)
- Half Step (Halbschritt)

Im **Wave-Drive-Betrieb** wird sukzessive jeweils nur eine Spule bestromt. Bild 6 fasst das beispielhaft an einem vierschrittigen Schrittmotor zusammen. Der Wave-Drive-Betrieb wird wegen seiner Nachteile in Bezug auf Drehmoment und Genauigkeit am seltensten verwendet. Wo es mehr auf eine geringe Stromaufnahme als auf Drehmoment ankommt, könnte er zum Einsatz kommen.

Der **Vollschrittbetrieb** zeichnet sich durch ein verbessertes Drehmoment gegenüber dem Wave-Drive-Betrieb aus. Das wird erzielt, indem immer zwei Spulen so bestromt werden, dass sie gemeinsam eine Anziehungskraft auf den magnetischen Rotor ausüben. Der Rotor bewegt sich dann in Positionen zwischen den Polen. In Bild 7 ist zu sehen, dass deshalb die Ausgangsposition des Rotors bei 45° liegt und es drei weitere Schrittwinkel 135°, 225°, 315° gibt, die den Drehzyklus vollenden.

Im **Halbschrittbetrieb** wird die Schrittzahl des Motors verdoppelt, indem der Schrittwinkel halbiert wird. Das geschieht, indem zwei Statorspulen wie im Halbschrittbetrieb gleichzeitig bestromt werden und im Folgeschritt wie im Wave-Drive-Betrieb nur eine einzelne Statorspule. Der Nachteil ist ein von Schritt zu Schritt alternierendes Drehmoment, was man jedoch durch Anheben des Spulenstroms in den Wave-Drive-Abschnitten egalisieren kann. Bild 8 gibt den Überblick.

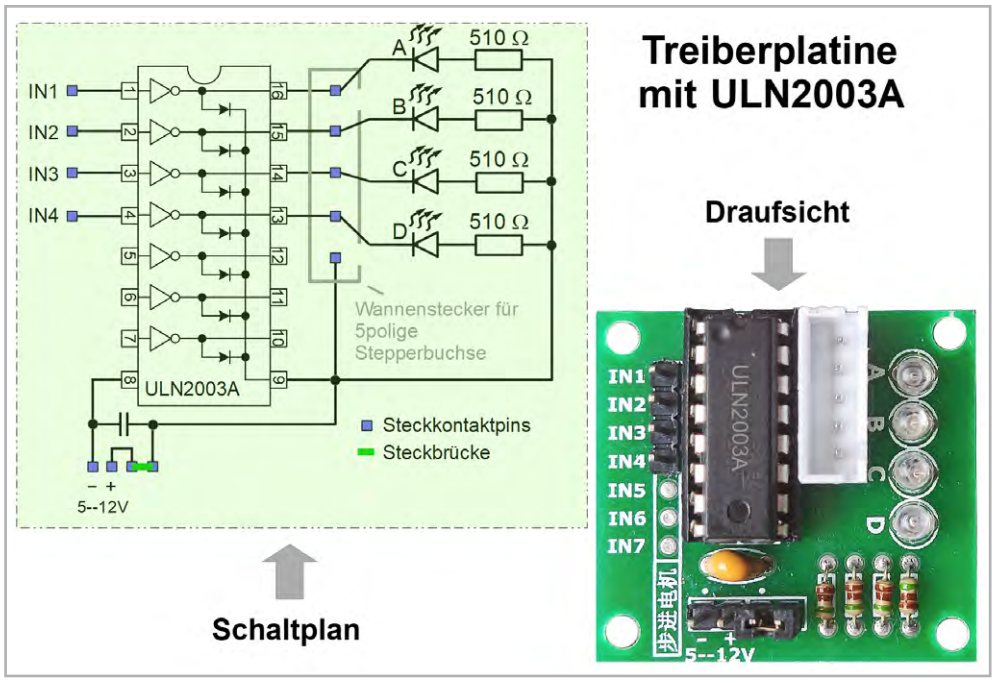


Bild 5: Die Treiberplatine mit ULN2003A: Schaltplan und Draufsicht

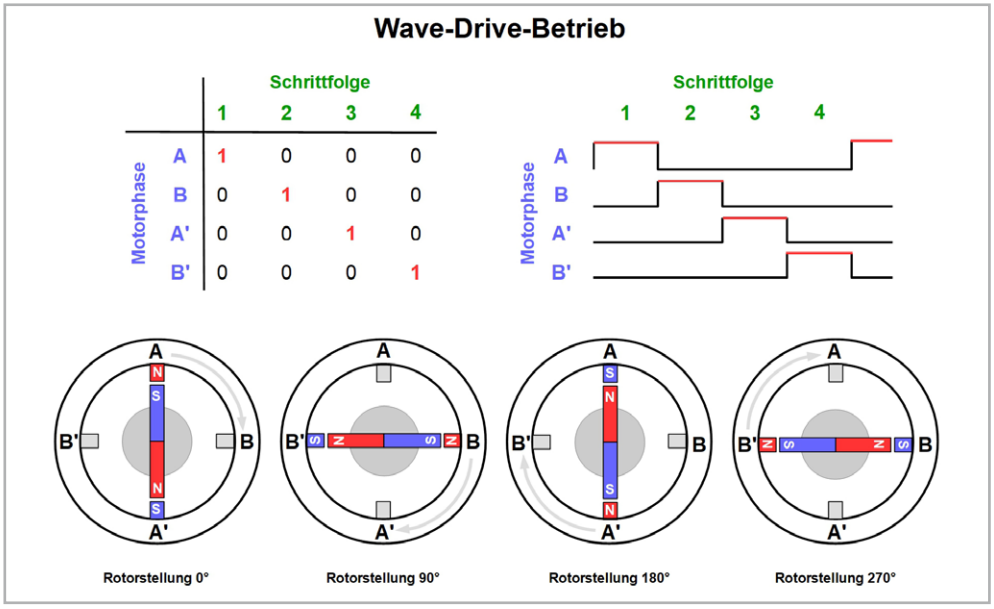


Bild 6: Im Wave-Drive-Betrieb werden die vier Phasenspulen einzeln zyklisch bestromt, was vier Schritte pro Umdrehung ergibt.

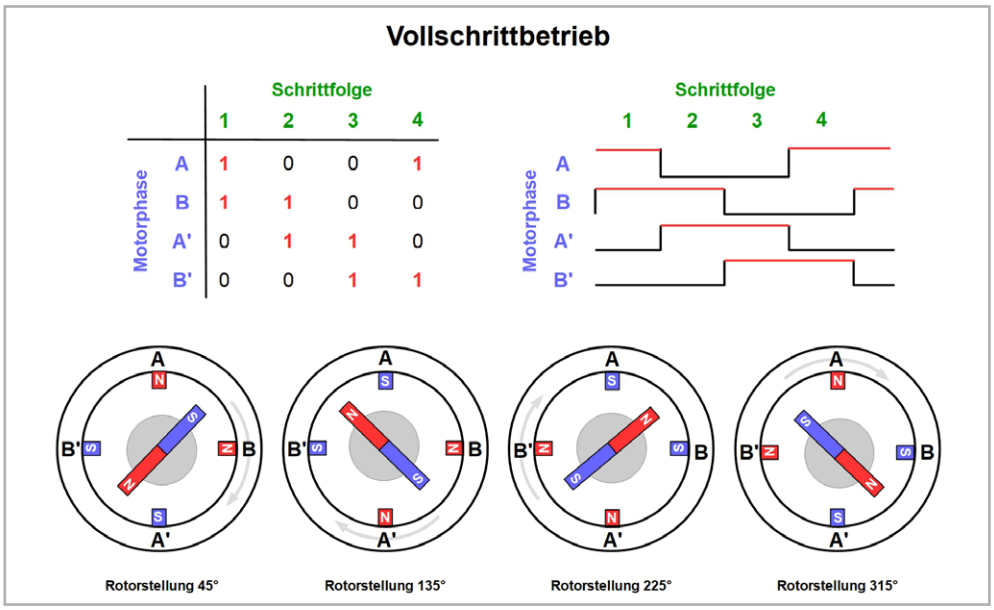


Bild 7: Im Vollschrittbetrieb werden immer zwei Spulen gleichzeitig bestromt, wodurch der Rotor vier stabile Schrittpositionen zwischen den Polen einnimmt.

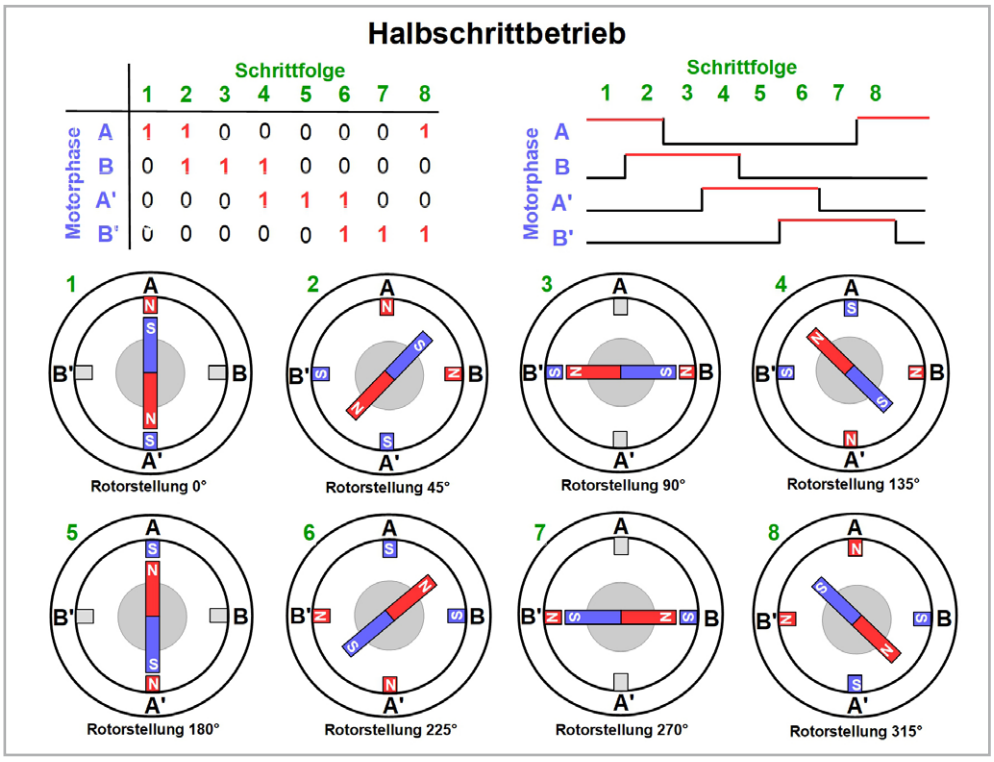


Bild 8: Im Halbschrittbetrieb wird die Zahl der stabilen Rotorpositionen auf acht verdoppelt.

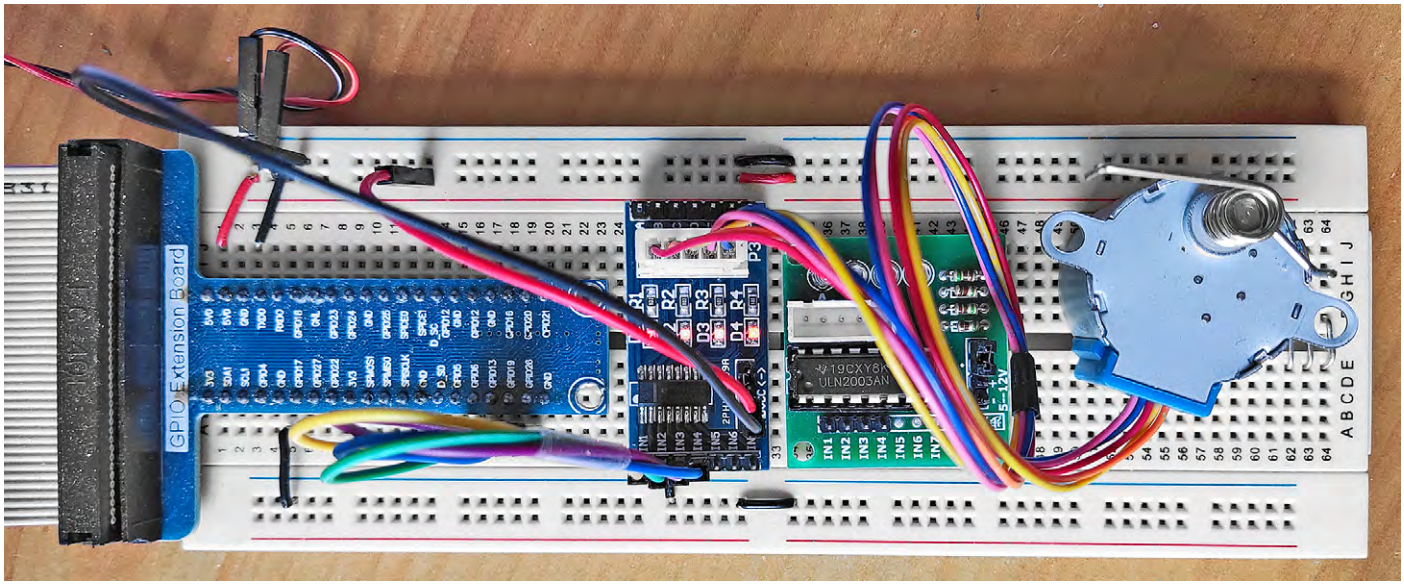


Bild 9: Der Versuchsaufbau. Das Breadboard wird nur benötigt, um die Eingangssignale der Treiberplatine von den als Ausgang geschalteten vier GPIO-Ports des Raspberry Pi 4B und die Versorgungsspannung des Motors +5VDC abzugreifen.

Betrieb des Schrittmotors

Zunächst zur Hardware. Hier sind, wie in Bild 9 zu sehen, vier als Output geschaltete IO-Ports der 40-poligen GPIO-Leiste des Raspberry Pi 4B mit den vier Eingängen der Treiberplatine IN1 bis IN4 zu verbinden. Die Wahl fiel auf die nach Broadcom mit GPIO6, GPIO13, GPIO19 und GPIO26 bezeichneten Raspi-Ports, weil sie physisch unmittelbar benachbart sind. Nun sind noch die Eingänge für +5 VDC und GND (Ground, Masse) der Treiberplatine mit den entsprechenden PINs der GPIO-Leiste zu verbinden und der Pfostenstecker am Motor ist in die Buchse auf der Treiberplatine zu stecken. Damit ist die Verdrahtung abgeschlossen.

In Bild 9 sind zwei funktionsgleiche Treiberplatinen zu sehen: Die angeschlossene ist mit SMD-Bauteilen ausgeführt, die nicht angeschlossene mit bedrahteten Bauteilen. Die Fritzing-Darstellung mit der zweitgenannten Treiberplatine verdeutlicht das Gesagte (Bild 10).

Die Software wurde ohne Effizienzehrgeiz direkt aus den Phasenschemata in Bild 6, Bild 7 und Bild 8 abgeleitet. Die Python-Scripts für die drei Betriebsarten sind dementsprechend einfach nachzuvollziehen.

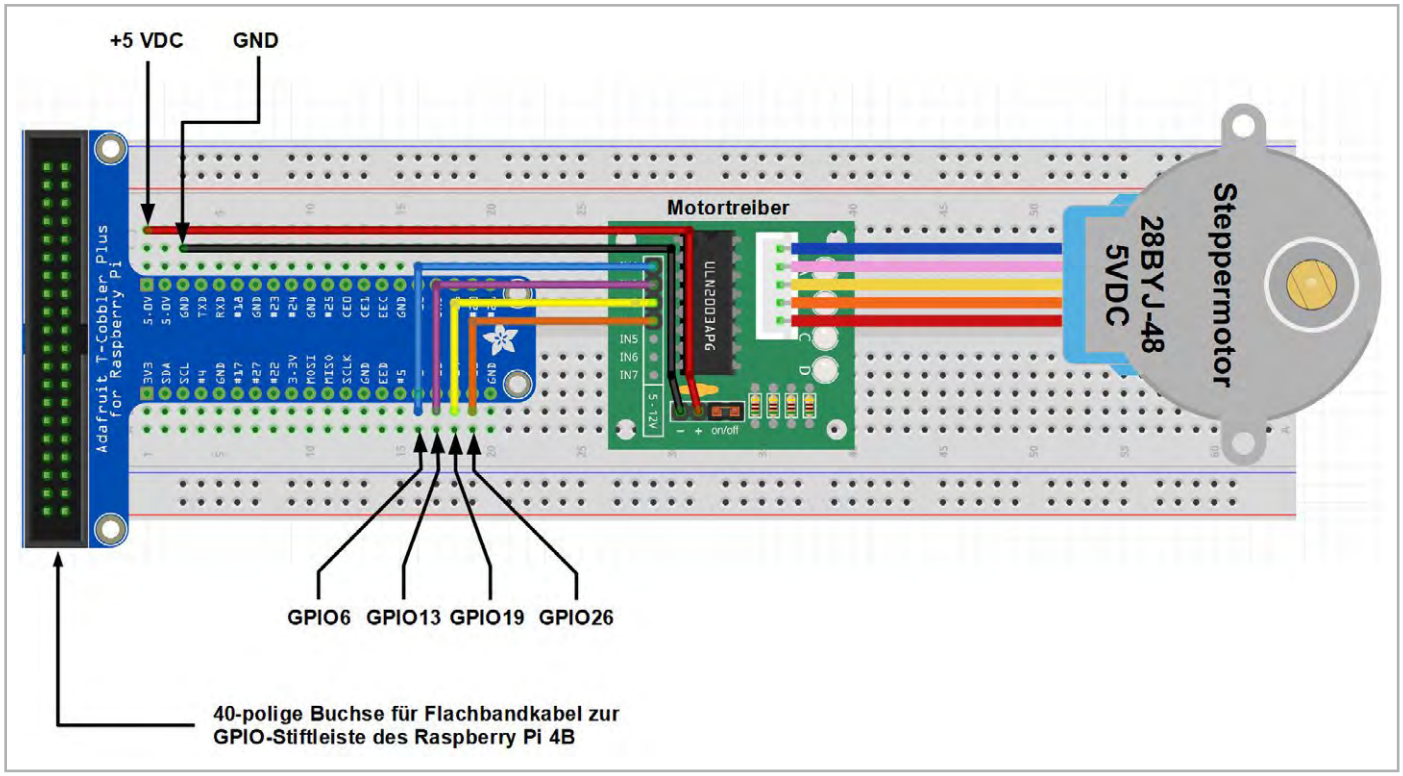


Bild 10: Mit dem Programm Fritzing lässt sich die Verdrahtung der Versuchsanordnung noch deutlicher als in Bild 9 darstellen.

SBC-Moto1_Wave.py *

```
from time import sleep
import RPi.GPIO as GPIO
import random

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
time = 0.002

IN1 = 6
IN2 = 13
IN3 = 19
IN4 = 26

GPIO.setup(IN1, GPIO.OUT)
GPIO.setup(IN2, GPIO.OUT)
GPIO.setup(IN3, GPIO.OUT)
GPIO.setup(IN4, GPIO.OUT)

# Phasenschema: Wave-Drive-Betrieb

def Step1():
    GPIO.output(IN1, 1)
    GPIO.output(IN2, 0)
    GPIO.output(IN3, 0)
    GPIO.output(IN4, 0)
    sleep(time)

def Step2():
    GPIO.output(IN1, 0)
    GPIO.output(IN2, 1)
    GPIO.output(IN3, 0)
    GPIO.output(IN4, 0)
    sleep(time)

def Step3():
    GPIO.output(IN1, 0)
    GPIO.output(IN2, 0)
    GPIO.output(IN3, 1)
    GPIO.output(IN4, 0)
    sleep(time)

def Step4():
    GPIO.output(IN1, 0)
    GPIO.output(IN2, 0)
    GPIO.output(IN3, 0)
    GPIO.output(IN4, 1)
    sleep(time)

def right(step):
    for i in range(step):
        Step1()
        Step2()
        Step3()
        Step4()

def left(step):
    for i in range(step):
        Step4()
        Step3()
        Step2()
        Step1()

while True:
    leftsteps = 512
    rightsteps = 512
    left(leftsteps)
    print(leftsteps, "Rotorschritte links", " =>", round((leftsteps/4)*0.5*5.625/360, 3), "Wellenumdrehung links,")
    right(rightsteps)
    print(rightsteps, "Rotorschritte rechts", " =>", round((rightsteps/4)*0.5*5.625/360, 3), "Wellenumdrehung rechts,")
```

Bild 11: Dieses Python-Script bewegt den Rotor im Wave-Drive-Betrieb in vier Steps/Umdrehung.

Bild 11 zeigt das Script, das eine Bestromung nach dem Wave-Drive-Schema bewirkt. Das Script für eine Bestromung der Phasen gemäß dem Vollschrittbetrieb ist in Bild 12 abgebildet, das für eine Halbschrittbestromung in Bild 13. Den Konsolenoutput der drei Scripts fasst Bild 14 zusammen.

In allen drei Scripts werden in den Zeilen 1 bis 3 die erforderlichen Bibliotheken geladen. In Zeile 5 wird die BCM-Benennung der Pins gewählt (BCM steht für Broadcom, den Hersteller der Raspi-CPU). Die GPIO-Pin-Belegung ist im Titelbild des Beitrags zu sehen.

In Zeile 7 wird eine Konstante **`time`** definiert, die für die Rotationsgeschwindigkeit zuständig ist. Je größer, umso langsamer – ruhig einmal etwas experimentieren. Die Drehgeschwindigkeit bestimmt der Parameter **`time`**. Je nach Betriebsart variiert sein kleinstmöglicher Wert, bei dem es zu Schrittaussetzern (missing steps) kommt. Meist kann **`time = 0,001`** nicht deutlich unterschritten werden.

In den Zeilen 9 bis 17 werden die bereits genannten GPIO-Ports GPIO6, GPIO13, GPIO19 und GPIO26 als Ausgänge konfiguriert. Die jetzt folgenden vier oder acht Unterprogramme bilden nur Spalten der Schrittfolge aus den Tabellen in Bild 6, Bild 7 und Bild 8 ab. Das Unterprogramm **`right(step)`** ruft nacheinander die Unterprogramme **`step1()`** bis **`step4()`**

beziehungsweise **step1()** bis **step8()** in aufsteigender Reihenfolge auf, was eine Rechtsdrehung bewirkt. Das Unterprogramm **left(step)** macht das entsprechend in absteigender Reihenfolge, wodurch eine Linksdrehung entsteht.

Das eigentliche Hauptprogramm startet mit der Einrichtung einer Endlosschleife durch **while True:**. Sicherlich kann man sich auf Dauer nicht nur an einer ständigen Hin-und-her-Drehung der Abtriebswelle um 360° erfreuen. Weil das doch gar zu fad ist, wurden drei die Drehung charakterisierende Parameter durch Zufallsentscheidungen bestimmt: Richtung, Rotorschrittanzahl (und damit starr verknüpft die Anzahl der Umdrehungen der Abtriebswelle) und Geschwindigkeit.

Dabei spielt die Funktion **random.randint(A, B)** die entscheidende Rolle. Sie sucht in dem Intervall B–A eine ganze Zufallszahl (A und B eingeschlossen) aus, die dann den entsprechenden Parameter beeinflusst. Mit wenigen Codezeilen ist das erledigt, wie in Bild 15 zu sehen ist. Zwei print-Zeilen dokumentieren in der Konsole fortlaufend die Rotationen. Übrigens hat die Funktion **random.randrange(A, B+1)** den gleichen Effekt wie **random.randint(A, B)**, beide sind also gegeneinander austauschbar.

Jetzt sind Drehrichtung, Drehwinkel und Drehgeschwindigkeit in den vorgegebenen Intervallen zufällig.

SBC-Moto1_Full_Step.py *

```
from time import sleep
import RPi.GPIO as GPIO
import random

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
time = 0.002

IN1 = 6
IN2 = 13
IN3 = 19
IN4 = 26

GPIO.setup(IN1, GPIO.OUT)
GPIO.setup(IN2, GPIO.OUT)
GPIO.setup(IN3, GPIO.OUT)
GPIO.setup(IN4, GPIO.OUT)

# Phasenschema: Full-Step-Betrieb

def Step1():
    GPIO.output(IN1, 1)
    GPIO.output(IN2, 1)
    GPIO.output(IN3, 0)
    GPIO.output(IN4, 0)
    sleep(time)

def Step2():
    GPIO.output(IN1, 0)
    GPIO.output(IN2, 1)
    GPIO.output(IN3, 1)
    GPIO.output(IN4, 0)
    sleep(time)

def Step3():
    GPIO.output(IN1, 0)
    GPIO.output(IN2, 0)
    GPIO.output(IN3, 1)
    GPIO.output(IN4, 1)
    sleep(time)

def Step4():
    GPIO.output(IN1, 1)
    GPIO.output(IN2, 0)
    GPIO.output(IN3, 0)
    GPIO.output(IN4, 1)
    sleep(time)

def right(step):
    for i in range(step):
        Step1()
        Step2()
        Step3()
        Step4()

def left(step):
    for i in range(step):
        Step4()
        Step3()
        Step2()
        Step1()

while True:
    leftsteps = 512
    rightsteps = 512
    left(leftsteps)
    print(leftsteps, "Rotorschritte links", " =>", round((leftsteps/4)*0.5*5.625/360, 3), "Wellenumdrehung links,")
    right(rightsteps)
    print(rightsteps, "Rotorschritte rechts", " =>", round((rightsteps/4)*0.5*5.625/360, 3), "Wellenumdrehung rechts,")
```

Bild 12: Im Vollschrittbetrieb ist das Drehverhalten wie im Wave-Betrieb, aber um 45° versetzt und mit höherem Drehmoment und Stromverbrauch.

SBC-Moto1_Half_Step.py

```python
from time import sleep
import RPi.GPIO as GPIO
import random

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
time = 0.0007

IN1 = 6
IN2 = 13
IN3 = 19
IN4 = 26

GPIO.setup(IN1, GPIO.OUT)
GPIO.setup(IN2, GPIO.OUT)
GPIO.setup(IN3, GPIO.OUT)
GPIO.setup(IN4, GPIO.OUT)

# Phasenschema: Half-Step-Betrieb

def Step1():
    GPIO.output(IN1, 1)
    GPIO.output(IN2, 0)
    GPIO.output(IN3, 0)
    GPIO.output(IN4, 0)
    sleep(time)

def Step2():
    GPIO.output(IN1, 1)
    GPIO.output(IN2, 1)
    GPIO.output(IN3, 0)
    GPIO.output(IN4, 0)
    sleep(time)

def Step3():
    GPIO.output(IN1, 0)
    GPIO.output(IN2, 1)
    GPIO.output(IN3, 0)
    GPIO.output(IN4, 0)
    sleep(time)

def Step4():
    GPIO.output(IN1, 0)
    GPIO.output(IN2, 1)
    GPIO.output(IN3, 1)
    GPIO.output(IN4, 0)
    sleep(time)

def Step5():
    GPIO.output(IN1, 0)
    GPIO.output(IN2, 0)
    GPIO.output(IN3, 1)
    GPIO.output(IN4, 0)
    sleep(time)

def Step6():
    GPIO.output(IN1, 0)
    GPIO.output(IN2, 0)
    GPIO.output(IN3, 1)
    GPIO.output(IN4, 1)
    sleep(time)

def Step7():
    GPIO.output(IN1, 0)
    GPIO.output(IN2, 0)
    GPIO.output(IN3, 0)
    GPIO.output(IN4, 1)
    sleep(time)

def Step8():
    GPIO.output(IN1, 1)
    GPIO.output(IN2, 0)
    GPIO.output(IN3, 0)
    GPIO.output(IN4, 1)
    sleep(time)

def right(step):
    for i in range(step):
        Step1()
        Step2()
        Step3()
        Step4()
        Step5()
        Step6()
        Step7()
        Step8()

def left(step):
    for i in range(step):
        Step8()
        Step7()
        Step6()
        Step5()
        Step4()
        Step3()
        Step2()
        Step1()

while True:
    leftsteps = 512
    rightsteps = 512
    left(leftsteps)
    print(leftsteps, "Rotorschritte links", " =>", round((leftsteps/8)*5.625/360, 3), "Wellenumdrehung links")
    right(rightsteps)
    print(rightsteps, "Rotorschritte rechts", " =>", round((rightsteps/8)*5.625/360, 3), "Wellenumdrehung rechts")
```

Bild 13: Der Halbschrittbetrieb kann als Mischung aus Wave-Drive-Betrieb und Vollschrittbetrieb verstanden werden. Weil wechselweise zwei oder eine Phasenspule bestromt werden, springt das Drehmoment von Schritt zu Schritt zwischen zwei Werten hin und her.

```
while True:
    leftsteps = 512
    rightsteps = 512
    left(leftsteps)
    print(leftsteps, "Rotorschritte links", " =>", round((leftsteps/4)*0.5*5.625/360, 3), "Wellenumdrehung links")
    right(rightsteps)
    print(rightsteps, "Rotorschritte rechts", " =>", round((rightsteps/4)*0.5*5.625/360, 3), "Wellenumdrehung rechts")
```

Kommandozeile

```
Python 3.7.3 (/usr/bin/python3)
>>> %Run SBC-Moto1_Wave.py
 512 Rotorschritte links  => 1.0 Wellenumdrehung links
 512 Rotorschritte rechts  => 1.0 Wellenumdrehung rechts
 512 Rotorschritte links  => 1.0 Wellenumdrehung links
 512 Rotorschritte rechts  => 1.0 Wellenumdrehung rechts
 512 Rotorschritte links  => 1.0 Wellenumdrehung links
 512 Rotorschritte rechts  => 1.0 Wellenumdrehung rechts
```

```
while True:
    leftsteps = 512
    rightsteps = 512
    left(leftsteps)
    print(leftsteps, "Rotorschritte links", " =>", round((leftsteps/4)*0.5*5.625/360, 3), "Wellenumdrehung links,")
    right(rightsteps)
    print(rightsteps, "Rotorschritte rechts", " =>", round((rightsteps/4)*0.5*5.625/360, 3), "Wellenumdrehung rechts,")
```

Kommandozeile

```
>>> %Run SBC-Moto1_Full_Step.py
 512 Rotorschritte links  => 1.0 Wellenumdrehung links,
 512 Rotorschritte rechts  => 1.0 Wellenumdrehung rechts,
 512 Rotorschritte links  => 1.0 Wellenumdrehung links,
 512 Rotorschritte rechts  => 1.0 Wellenumdrehung rechts,
 512 Rotorschritte links  => 1.0 Wellenumdrehung links,
```

```
while True:
    leftsteps = 512
    rightsteps = 512
    left(leftsteps)
    print(leftsteps, "Rotorschritte links", " =>", round((leftsteps/8)*5.625/360, 3), "Wellenumdrehung links")
    right(rightsteps)
    print(rightsteps, "Rotorschritte rechts", " =>", round((rightsteps/8)*5.625/360, 3), "Wellenumdrehung rechts")
```

Kommandozeile

```
Python 3.7.3 (/usr/bin/python3)
>>> %Run SBC-Moto1_Half_Step.py
 512 Rotorschritte links  => 1.0 Wellenumdrehung links
```

Bild 14: Diese Ausgaben in der Konsole (Kommandozeile) erzeugen die drei Python-Scripts.

```
while True:
    if random.randint(0, 1) >= 0.5:          # zufällige Drehrichtung
        leftsteps = random.randint(10, 300)   # zufälliger Drehwinkel links
        time = random.randint(8, 40)/10000    # zufällige Drehgeschwindigkeit links
        left(leftsteps)
        print(leftsteps, "Rotorschritte links", " =>", round((leftsteps/8)*5.625/360, 3), "Wellenumdrehungen links,", "time =", time)
    else:
        rightsteps = random.randint(10, 300)  # zufälliger Drehwinkel rechts
        time = random.randint(8, 40)/10000    # zufällige Drehgeschwindigkeit rechts
        right(rightsteps)
        print(rightsteps, "Rotorschritte rechts", "=>", round((rightsteps/8)*5.625/360, 3), "Wellenumdrehungen rechts,", "time =", time)
```

Kommandozeile

```
 53 Rotorschritte links  => 0.104 Wellenumdrehungen links, time = 0.0026
 35 Rotorschritte links  => 0.068 Wellenumdrehungen links, time = 0.0012
178 Rotorschritte links  => 0.348 Wellenumdrehungen links, time = 0.0026
 69 Rotorschritte rechts => 0.135 Wellenumdrehungen rechts, time = 0.0014
289 Rotorschritte rechts => 0.564 Wellenumdrehungen rechts, time = 0.0033
210 Rotorschritte rechts => 0.41 Wellenumdrehungen rechts, time = 0.0009
 21 Rotorschritte links  => 0.041 Wellenumdrehungen links, time = 0.0015
209 Rotorschritte rechts => 0.408 Wellenumdrehungen rechts, time = 0.0031
126 Rotorschritte links  => 0.246 Wellenumdrehungen links, time = 0.0011
 78 Rotorschritte rechts => 0.152 Wellenumdrehungen rechts, time = 0.0038
249 Rotorschritte links  => 0.486 Wellenumdrehungen links, time = 0.0038
```

Bild 15: Überlässt man die drei Parameter der Drehung einem Zufallsgenerator, ergibt sich ein interessantes Drehverhalten der Abtriebswelle. Weder Richtung und Geschwindigkeit noch Winkel der Drehung sind vorhersagbar.

Zusammenfassung

In diesem Artikel wurde gezeigt, wie die prinzipielle Ansteuerung eines Low-Cost-Schrittmotors mit nachgeschaltetem Getriebe mittels eines Raspberry Pi erfolgt. Das dabei verwendete Array aus sieben Darlington-Transistoren hat keine weiteren Funktionen, wie sie ein echter Controllerbaustein für Steppermotoren aufweist. Damit wird sich die nächste Folge der Artikelreihe beschäftigen. ELV