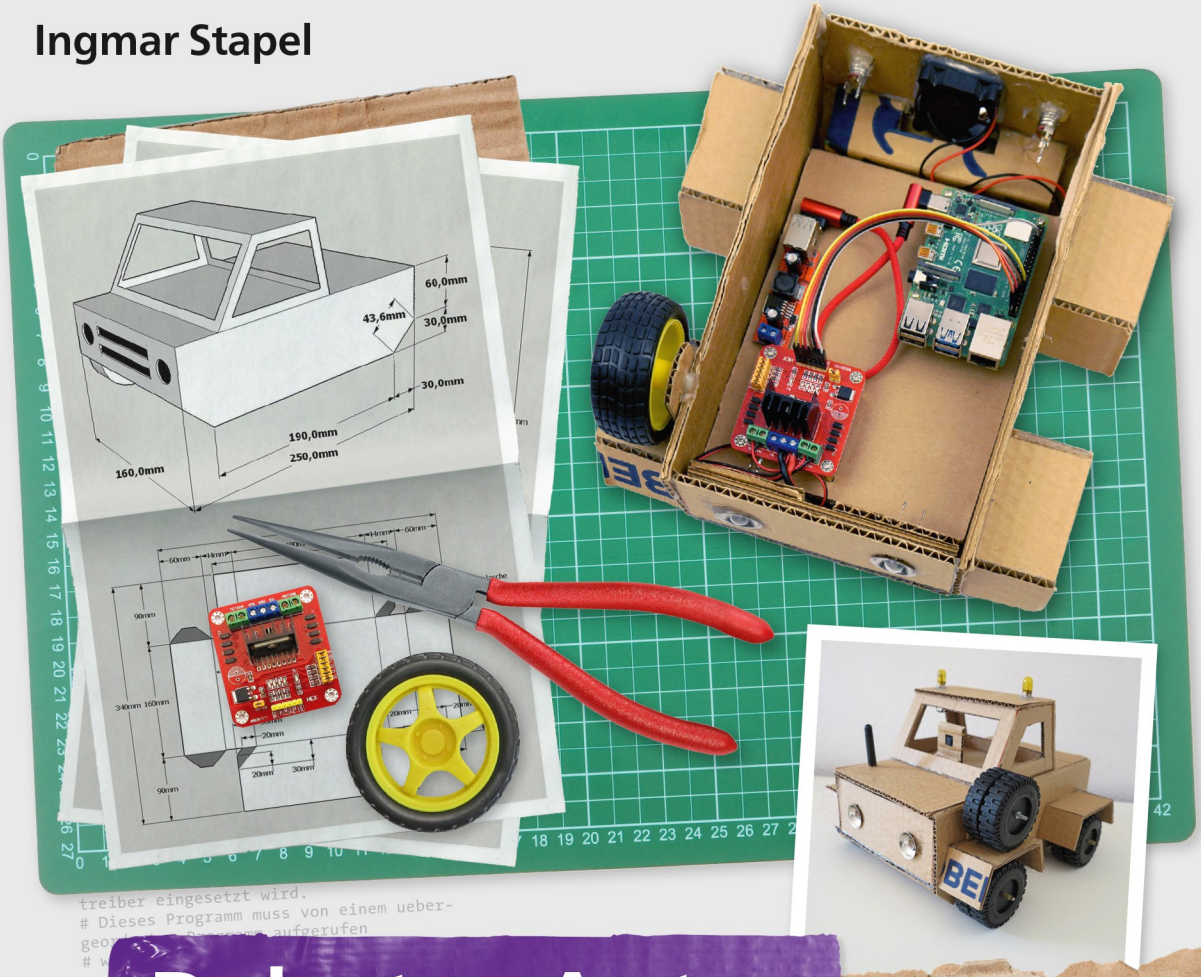
Ingmar Stapel



Roboter-Autos mit dem Raspberry Pi

Planen, bauen, programmieren

- Ferngesteuerte und autonome Modelle selbst bauen
- Alle relevanten Programmier- und Elektronikgrundlagen
- Ohne Vorwissen einfach einsteigen

Alle Codebeispiele und Vorlagen zum Download

Rheinwerk Technik

Kapitel 7

Lange Leitung? Manchmal besser! Verkabelung der elektronischen Komponenten

Nach der ganzen Theorie folgt nun die Verkabelung der elektronischen Komponenten des Roboter-Autos. Jetzt stellen Sie die Stromversorgung her, verkabeln den Raspberry Pi und schließen die Elektromotoren an den Motortreiber an.

In diesem Kapitel werde ich gemeinsam mit Ihnen detailliert die Verkabelung des Roboter-Autos durchgehen. Spätestens jetzt werden Sie alle elektronischen Komponenten in das Roboter-Auto einbauen und miteinander verbinden. Die Motoren werden dabei zusammen mit dem Motortreiber verkabelt, und der Motortreiber wird logisch am Raspberry Pi unter Verwendung der 40-poligen Stiftleiste angeschlossen.

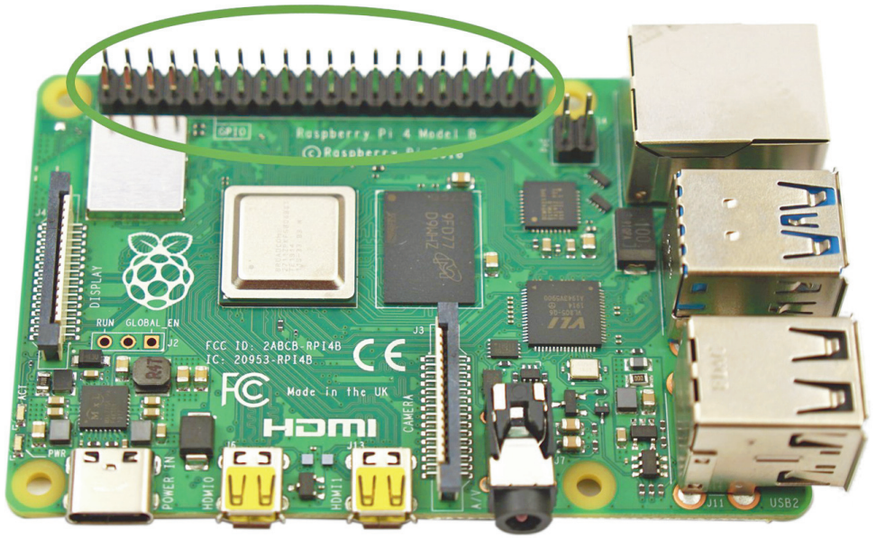


Abbildung 7.1 Raspberry Pi 4 Modell B

Diese Komponenten benötigen Sie

- 1 Raspberry Pi
- 4 Getriebemotoren
- 1 USB-C-Kabel
- 1 L298N-H-Brücke/Motortreiber
- 1 Step-down-Converter
- 5 × 20 cm Kupferkabel, zweiadrig
- 1 × RC-Akku 7,2 V 5.000 mAh
- 1 × Tamiya-Anschlusskabel für Akku (Stecker)
- 3 × Lüsterklemmen für je zwei Adern
- 6 Female-to-Female-Jumper-Kabel
- 2 × Mini-Tamiya-Kabelpaar (optional)

In vier Schritten zur fertigen Verkabelung

Die Verkabelung der elektronischen Komponenten ist in folgende vier Abschnitte unterteilt:

1. Anschließen der elektronischen Komponenten an die Batterie für die Stromversorgung
2. Motortreiber mit Raspberry Pi logisch verbinden
3. Verkabelung der Getriebemotoren
4. Getriebemotoren mit dem Motortreiber verbinden

7.1 Stromversorgung der elektronischen Komponenten

Der Motortreiber wird zusammen mit dem Step-down-Converter von der Batterie mit Strom versorgt. Die drei blauen Lüsterklemmen des Motortreibers sind mit **VMS**, **GND** und **5V** beschriftet. Die Beschriftung bedeutet im Einzelnen:

- **VMS**: Versorgungsspannung
- **GND**: Masse (*ground*)
- **5V**: Ausgangsspannung

Die 5-V-Ausgangsspannung des Motortreibers ist nicht stabil genug, um den Raspberry Pi direkt mit Strom zu versorgen. Der Grund hierfür ist, dass der Spannungswandler des Motortreibers nicht auf die hohen Ströme des Raspberry Pi ausgelegt ist und sich wegen

Überhitzung abschaltet. Deshalb wird im Roboter-Auto der Step-down-Converter für die Stromversorgung des Raspberry Pi verwendet.

An die beiden Lüsterklemmen **VMS** und **GND** des Motortreibers wird eines der 20 cm langen zweiadrigen Kupferkabel zusammen mit dem Tamiya-Steckerkabel des Batteriehalters angeschlossen. Bitte beachten Sie unbedingt, dass Sie die Pole **VMS** für Plus (+) und **GND** für Minus (–) richtig anschließen! Der Pluspol (**VMS**) wird mit dem roten Kabel verbunden und die Masse (**GND**) mit dem schwarzen Kabel.

Das freie Ende des zweiadrigen Kupferkabels am Motortreiber verbinden Sie nun mit den beiden Lüsterklemmen (+/–) des Step-down-Converters. Jetzt können Sie den RC-Akku mit dem Tamiya-Kabel das aus dem Motortreiber kommt verbinden.

In Abbildung 7.2 sehen Sie den bisherigen Aufbau, also den Step-down-Converter und die Batterie, die zusammen am Motortreiber angeschlossen sind.

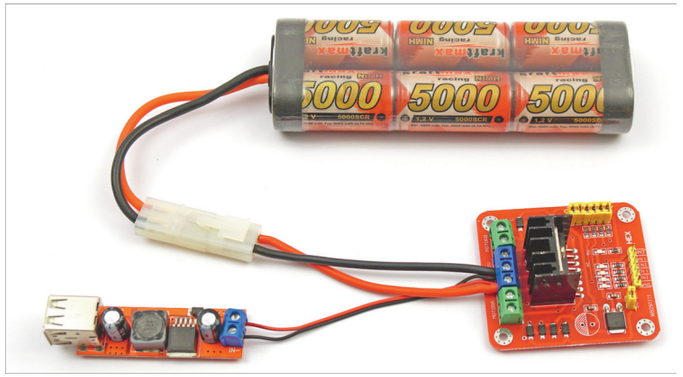


Abbildung 7.2 Stromversorgung, Motortreiber und Step-down-Converter

Batteriehalter mit sechs Mignon-Zellen

Sollten Sie statt des RC-Akkus sich für den Batteriehalter mit den sechs Mignon-Zellen entschieden haben, dann kann es zusammen mit dem Raspberry Pi 4 Modell B zu Problemen in der Stromversorgung kommen. Hier bietet der RC-Akku bei Spitzenlasten im Stromverbrauch – wenn z. B. das Roboter-Auto auf der Stelle dreht – ausreichende Reserven für einen stabilen Betrieb.

Der Raspberry Pi wird mit einem USB-C-Kabel an den Step-down-Converter angeschlossen. Da Sie erst später eine microSD-Karte mit dem benötigten Betriebssystem in den Raspberry Pi stecken werden, passiert jetzt noch nichts, wenn Sie die Stromversorgung über den Step-down-Converter aktivieren.

In Abbildung 7.3 sehen Sie den Raspberry Pi, der über den Step-down-Converter und das USB-C-Kabel mit Strom versorgt wird.

Sollten Sie zwei Batterien in ihrem Aufbau verwenden – wie eine Power-Bank, um den Raspberry Pi mit Energie zu versorgen, und einen weiteren Akku für die Motoren –, dann müssen Sie einen gemeinsamen Ground zwischen diesen beiden Batterien herstellen. Andernfalls kann es passieren, dass der Motortreiber logisch ein HIGH-Signal von einem LOW-Signal nicht unterscheiden kann.

Nachdem Sie die Energieversorgung der einzelnen Komponenten nun soweit sichergestellt haben, können Sie mit dem logischen Anschluss des Motortreibers an den Raspberry Pi beginnen.

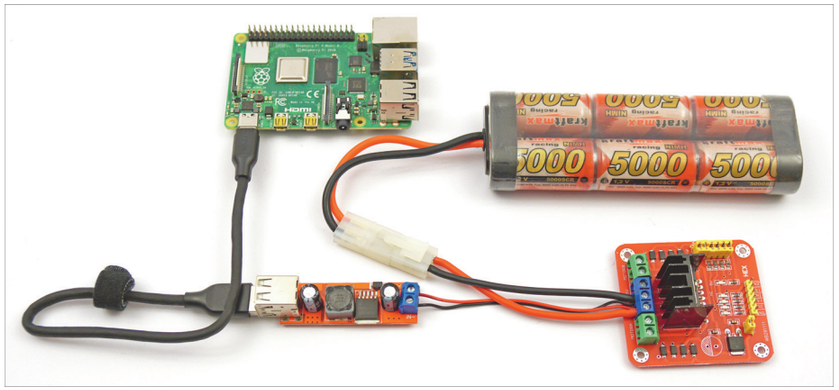


Abbildung 7.3 Die Stromversorgung für den Raspberry Pi

7.2 Motortreiber und Raspberry Pi logisch verbinden

Sie benötigen spezielle Kabel, um den Motortreiber logisch mit dem Raspberry Pi zu verbinden. Diese Kabel heißen *Female-to-Female-Jumper-Kabel*. Sie lassen sich einfach auf die Stifte des Raspberry Pi und die Stifte des Motortreibers stecken und wurden bereits in Abschnitt 2.8, »Kabel«, gezeigt.

Einige der 40 Anschlüsse der goldfarbigen Stiftleiste des Raspberry Pi sind sogenannte *GPIO-Pins*. Die Abkürzung *GPIO* leitet sich vom englischen Begriff *General Purpose Input/Output* ab und bedeutet übersetzt »Allzweckeingabe/-ausgabe«. GPIO-Pins sind also Stifte zur elektronischen Ein- und Ausgabe am Raspberry Pi.

Bevor Sie mit der Verkabelung beginnen, bitte ich Sie, die folgenden Hinweise genau zu lesen und dauerhaft zu beachten, um den Raspberry Pi nicht zu beschädigen.

Achtung: Spannungen größer 3,3 V können den Raspberry Pi zerstören!

- Der Raspberry Pi ist ein 3,3-V-Computer, an dessen GPIO-Anschlüsse Sie **keine höheren Spannungen als 3,3 V** anlegen dürfen!
- Die GPIO-Anschlüsse sind ungeschützt. Bei einem Fehler in der Verkabelung kann der Raspberry Pi durchbrennen.

Sie werden insgesamt sechs GPIO-Pins des Raspberry Pi benötigen. Davon sind zwei GPIO-Pins vorgesehen, die das sogenannte *PWM-Signal* übertragen (dazu gleich mehr), und weitere vier Pins, die die Drehrichtung der Motoren festlegen.

Die *Pulsweitenmodulation* (PWM) ist das Steuersignal für die Regelung der Geschwindigkeit der Motoren. Dieses Signal wird durch den Raspberry Pi erzeugt.

Die Periode, in der sich das Signal wiederholt, kann der Anwender mit z. B. 20 ms fest vorgeben. Die Pulsweite kann innerhalb der Periode variiert werden und – wie in Abbildung 7.4 gezeigt – beispielsweise 5 ms betragen. Je größer die Pulsweite ist, umso länger legt der Motortreiber Spannung an seinen Ausgängen an, und die Motoren können sich drehen.

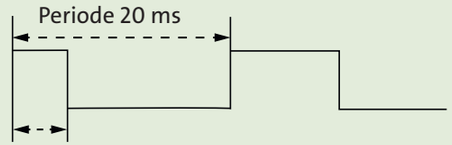


Abbildung 7.4 Schaubild zur Pulsweitenmodulation (PWM)

Der Motortreiber hat, wie Abbildung 7.5 zeigt, am unteren Rand eine Steckleiste mit sechs Kontakten mit den Bezeichnungen **ENA**, **ENB**, **IN1**, **IN2**, **IN3** und **IN4**. Über diese Kontakte wird der Motortreiber gesteuert.

PWM-GPIO-Pins am Raspberry Pi

Der Raspberry-Pi-Computer erzeugt mit seinem internen Taktgeber an den beiden GPIO-Pins 12 und 18 ein sehr stabiles und exaktes PWM-Signal. Es ist zwar möglich, an allen anderen GPIO-Pins ein über die CPU generiertes PWM-Signal auszugeben, aber dieses ist im Vergleich zu dem über den internen Taktgeber erzeugten PWM-Signal sehr ungenau.

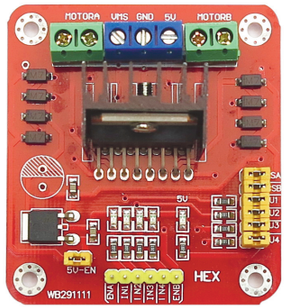


Abbildung 7.5 Motortreiber

Über die zwei Eingänge **ENA** und **ENB** des Motortreibers wird die Drehgeschwindigkeit der Motoren mithilfe des PWM-Signals gesteuert.

- Motor A (linke Seite)
  - **ENA**
- Motor B (rechte Seite)
  - **ENB**

Über die vier Eingänge **IN1**, **IN2**, **IN3** und **IN4** des Motortreibers wird die Drehrichtung der Motoren gesteuert. Für Motor A setzen Sie dazu die Pins **IN1** und **IN2** auf High bzw. Low; für Motor B setzen Sie entsprechend die Pins **IN3** und **IN4** auf High bzw. Low, und zwar jeweils direkt am Motortreiber.

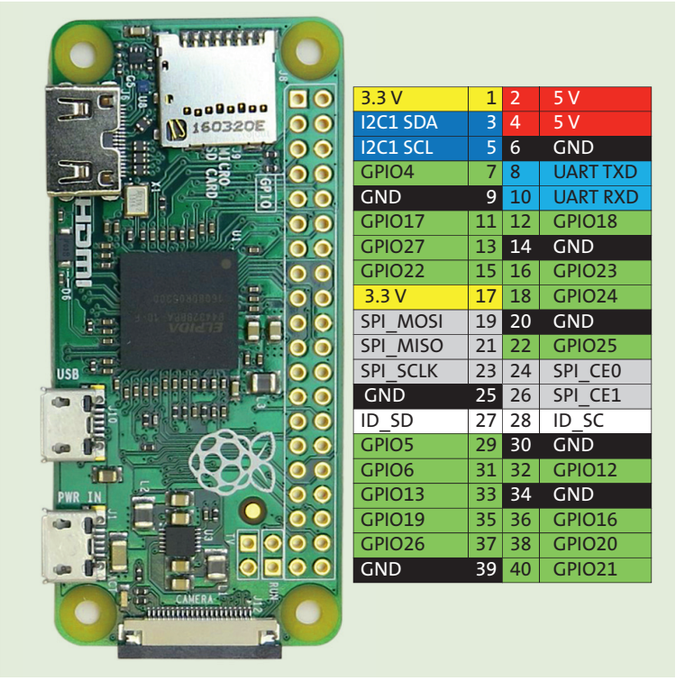
- Motor A
  - **IN1**
  - **IN2**
- Motor B
  - **IN3**
  - **IN4**

7.2.1 Die Ausrichtung der 40-Pin-Stiftleiste des Raspberry Pi

Damit Sie sich auf Ihrem Raspberry Pi orientieren können, habe ich eine Grafik erstellt, die für alle Raspberry-Pi-Modelle mit der 40-Pin-Stiftleiste gültig ist. Anhand dieser Grafik sehen Sie, wie die Ausrichtung der 40 Pins funktioniert und wo Pin 1 zu finden ist. Pin 1 finden Sie immer an der Stelle, an der der Rahmen, der um die Stiftleiste auf die Platine aufgedruckt ist, um 45° in der Ecke abgeschrägt ist.

In Abbildung 7.6 sehen Sie links den Raspberry Pi ZERO mit der 40-Pin-Stiftleiste und der um 45° abgeschrägten Ecke des weißen Rahmens, der auf die Platine gedruckt ist. Rechts sehen Sie die Tabelle mit der Pin-Belegung des Raspberry Pi. Auch bei dieser Platine ist die Ecke links oben abgeschrägt.

Die Tabelle aus Abbildung 7.6 werden Sie immer wieder brauchen, um die Pins im weiteren Verlauf des Buches richtig zu verkabeln. Ich empfehle Ihnen, die Pin-Belegung von meiner Webseite *https://custom-build-robots.com/roboter-auto-download* herunterzuladen und ausgedruckt griffbereit zu halten, wenn Sie mit der Verkabelung beginnen:



3.3 V	1	2	5 V
I2C1 SDA	3	4	5 V
I2C1 SCL	5	6	GND
GPIO4	7	8	UART TXD
GND	9	10	UART RXD
GPIO17	11	12	GPIO18
GPIO27	13	14	GND
GPIO22	15	16	GPIO23
3.3 V	17	18	GPIO24
SPI_MOSI	19	20	GND
SPI_MISO	21	22	GPIO25
SPI_SCLK	23	24	SPI_CE0
GND	25	26	SPI_CE1
ID_SD	27	28	ID_SC
GPIO5	29	30	GND
GPIO6	31	32	GPIO12
GPIO13	33	34	GND
GPIO19	35	36	GPIO16
GPIO26	37	38	GPIO20
GND	39	40	GPIO21

Abbildung 7.6 Ausrichtung und Belegung der 40-Pin-Stiftleiste am Raspberry Pi

Achten Sie bitte darauf, dass es die Nummerierung 1 bis 40 auf der Stiftleiste gibt und dass die Nummerierung der GPIO-Pins davon abweicht. So ist der GPIO-Pin 22 der Pin 15 der 40-Pin-Stiftleiste. Kommt ein anderer SBC als ein Raspberry Pi zum Einsatz, dann ändert sich die Pin-Belegung entsprechend dem von Ihnen alternativ verwendeten Modell. In diesem Fall müssen Sie die Pin-Belegung selbst recherchieren, z. B. auf der Hersteller-Website im Internet.

7.2.2 Übersicht über die Verkabelung des Motortreibers mit den GPIO-Pins

Tabelle 7.1 zeigt, welche der Pins des Raspberry Pi Sie mit dem Motortreiber verbinden müssen. Natürlich können Sie andere GPIO-Pins für die Ansteuerung des Motortreibers benutzen als die in Tabelle 7.1 angegebenen; in diesem Fall müssen Sie jedoch später das Roboter-Auto-Steuerprogramm entsprechend anpassen. Daher empfehle ich Ihnen, genau diese GPIO-Anschlüsse für die Verkabelung zu verwenden.

Anschlüsse	Funktion	Raspberry Pi Pin-Nummerierung		Funktion	Anschlüsse
	3.3 V	1	2	5 V	
	I2C1 SDA	3	4	5 V	
	I2C1 SCL	5	6	GND	
	GPIO 4	7	8	UART TXD	
	GND	9	10	UART RXD	
	GPIO 17	11	12	GPIO 18	PWM-Motor A (**ENA**)
	GPIO 27	13	14	GND	
	GPIO 22	15	16	GPIO 23	
	3.3 V	17	18	GPIO 24	
	SPI_MOSI	19	20	GND	
	SPI_MISO	21	22	GPIO 25	
	SPI_SCLK	23	24	SPI_CE0	
	GND	25	26	SPI_CE1	
	ID_SD	27	28	ID_SC	
	GPIO 5	29	30	GND	
Motor A (**IN1**)	GPIO 6	31	32	GPIO 12	PWM-Motor B (**ENB**)
Motor A (**IN2**)	GPIO 13	33	34	GND	
Motor B (**IN3**)	GPIO 19	35	36	GPIO 16	
Motor B (**IN4**)	GPIO 26	37	38	GPIO 20	
	GND	39	40	GPIO 21	

Tabelle 7.1 Raspberry-Pi-GPIO-Belegung

Das in Tabelle 7.1 verwendete Farbschema erklärt sich wie folgt:

	I2C-Interface
	UART-Interface
	ID-EEPROM-Interface
	Ground

	GPIO
	SPI-Interface
	5,0 V
	3,3 V

Tabelle 7.2 Erklärung des Farbschemas

Um Fehler in der Verkabelung zu vermeiden, empfehle ich Ihnen, sich die Kabelfarbe je Verbindung zwischen Raspberry Pi und Motortreiber in Tabelle 7.3 zu notieren. So können Sie immer wieder nachschlagen und wissen genau, wo welches Kabel hingeht, ohne sich die Verkabelung jedes Mal herleiten zu müssen.

Raspberry-Pi-Pins	Kabelfarbe	Motortreiber-Pins
GPIO 18		**ENA**
GPIO 6		**IN1**
GPIO 13		**IN2**
GPIO 19		**IN3**
GPIO 26		**IN4**
GPIO 12		**ENB**

Tabelle 7.3 Anschlussbelegung Ihres Raspberry Pi am Motortreiber mit Kabelfarben

In Abbildung 7.7 sehen Sie den Raspberry Pi und den Motortreiber. Beide sind über sechs Female-to-Female-Jumper-Kabel logisch miteinander verbunden. Die Kabel zwischen Raspberry Pi und dem Motortreiber müssen Sie exakt so setzen, wie in Tabelle 7.1 definiert. Nur so ist sichergestellt, dass die Ansteuerung klappt und keine Defekte an den beiden Komponenten entstehen.

Haben Sie die Kabel richtig gesteckt und das auch noch einmal überprüft? Na, dann machen wir doch gleich direkt weiter und schließen im nächsten Schritt die Motoren an den Motortreiber an.

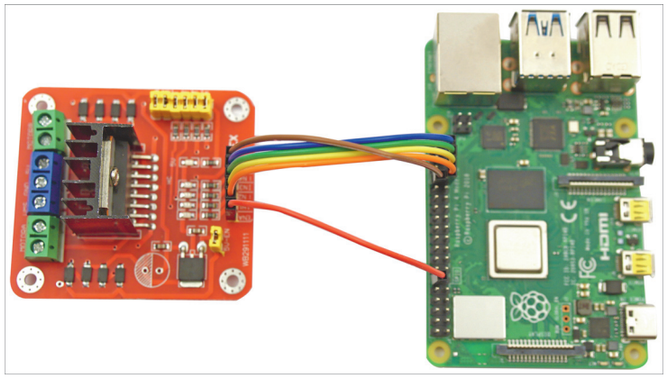


Abbildung 7.7 Logische Verkabelung des Raspberry Pi mit einem Motortreiber

7.3 Verkabelung der Getriebemotoren

Bevor Sie die Getriebemotoren einbauen, müssen Sie noch die Kabel für die Stromversorgung der Motoren an die Kupferlaschen der Getriebemotoren löten. Mit ein wenig Fingerspitzengefühl, der folgenden Komponenten- und Werkzeugliste sowie mit den Tipps zum Löten aus Abschnitt 3.2 dürfte das kein größeres Hindernis für Sie darstellen.

- benötigte Komponenten
  - 4 Getriebemotoren
  - 4 × 20 cm Kupferkabel zweiadrig
- Werkzeugliste
  - Zange
  - Lötstation und Lötzinn

Für jeden Motor benötigen Sie jeweils ein 20 cm langes, zweiadriges Kupferkabel mit einem Querschnitt von mindestens 0,14 mm^2. Bitte entfernen Sie an beiden Enden der Kabel 3 bis 4 mm der Kunststoffisolierung.

Die Motoren sollten Sie, bevor Sie die beiden Adern des Kabels anlöten, aus dem Getriebe ausbauen. Das ist leicht erledigt: Lösen Sie einfach mit einer Zange die Kunststofflasche hinten an den Getriebemotoren, um den Motor aus dem Getriebe herauszuziehen. Auf diese Weise vermeiden Sie es, die Kunststofflasche und das Gehäuse des Getriebemotors mit dem heißen Lötkolben zu beschädigen. Im ausgebauten Zustand lässt sich das Kupferkabel an den beiden Kontakten des Motors auch wesentlich einfacher anlöten. Bitte achten Sie darauf, das Kupferkabel fest an die Kontakte des Getriebemotors anzulöten. In

Abbildung 7.8 sehen Sie einen Elektromotor mit angelöteten Anschlusskabeln für die Stromversorgung.

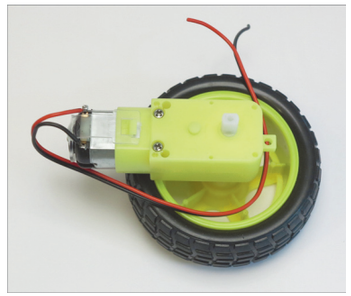


Abbildung 7.8 Elektromotor mit angelöteten Stromkabeln

Nach Abschluss der Lötarbeiten schieben Sie die Motoren einfach wieder zurück in die Getriebe. Fixieren Sie die Motoren mit der Plastiklasche. Damit sind Ihre Arbeiten an den Getriebemotoren bereits abgeschlossen.

Die fertigen Motoren sollten bei Ihnen jetzt so wie in Abbildung 7.9 aussehen.

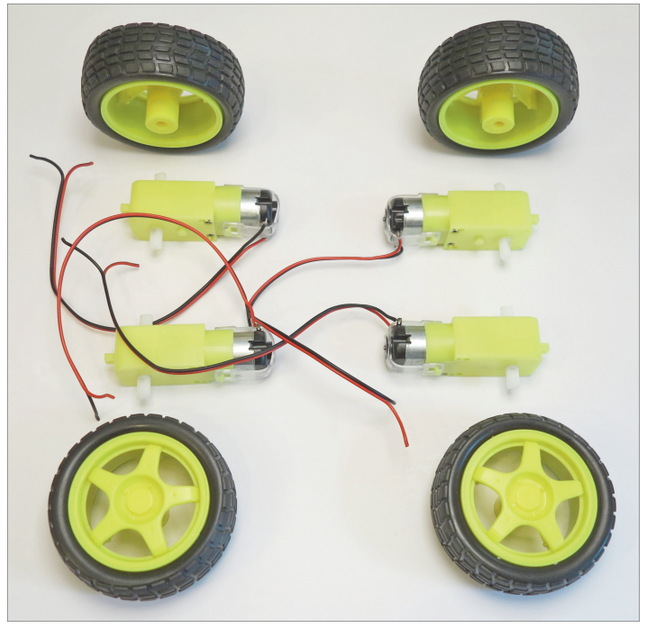


Abbildung 7.9 Getriebemotoren mit Kabeln und Rädern

7.4 Getriebemotoren mit dem Motortreiber verbinden

Die beiden Getriebemotoren einer Seite werden parallel an den Motortreiber angeschlossen. Schließen Sie dazu die vier Kabel der beiden linken Motoren in einer Lüsterklemme an, und schließen Sie die vier Kabel der beiden rechten Motoren ebenfalls in einer eigenen Lüsterklemme zusammen. Anschließend müssen Sie nur jeweils die beiden Kabel pro Lüsterklemme am Motortreiber anschließen.

Die grünen Anschlussklemmen (siehe Abbildung 7.10) des Motortreibers sind für die linke Seite (Motor A) und die rechte Seite (Motor B) vorgesehen. Dazu schließen Sie die beiden linken und die beiden rechten Motoren, wie nachfolgend beschrieben, über Kreuz zusammen erst an der Lüsterklemme und dann am Motortreiber an.

Sie müssen die Motoren über Kreuz anschließen, da sie sich durch den Einbau und die damit verbundene 180°-Drehung der Motoren ansonsten entgegengesetzt zueinander drehen würden.

Abbildung 7.10 zeigt, wie Sie die beiden Motoren einer Seite des Roboter-Autos über Kreuz verkabeln.

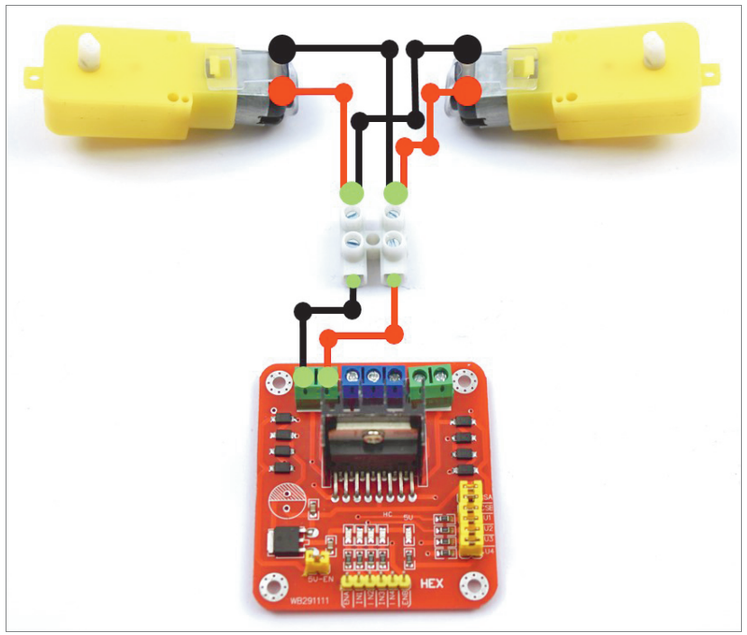


Abbildung 7.10 Schemabild der Motorverkabelung

Falsche Verkabelung ist eine häufige Fehlerquelle. Sie kann beispielsweise dazu führen, dass die Motoren sich gegenläufig drehen oder dass die linke und rechte Seite der Motoren vertauscht ist. Sollte Ihnen das einmal passieren, dann können und müssen Sie genau an diesem Punkt der Verkabelung die Ursache des Fehlers beheben. In aller Regel ist der Fehler – sollte er einmal auftreten – mit etwas Ausprobieren schnell gefunden und behoben.

Abbildung 7.11 zeigt Ihnen die Verkabelung des Roboter-Autos insgesamt. Links im Bild ist die Bodenwanne mit den Motoren zu erkennen. Rechts daneben im Bild sehen Sie den Elektronikträger mit den elektronischen Komponenten. Die Raspberry-Pi-Kamera fehlt noch auf dem Bild, da sie erst später angeschlossen wird. Wie Sie die Kamera anschließen, konfigurieren und in Betrieb nehmen, erfahren Sie in Kapitel 13, »Geisterfahrer aufgepasst! Wir sorgen für Durchblick«.

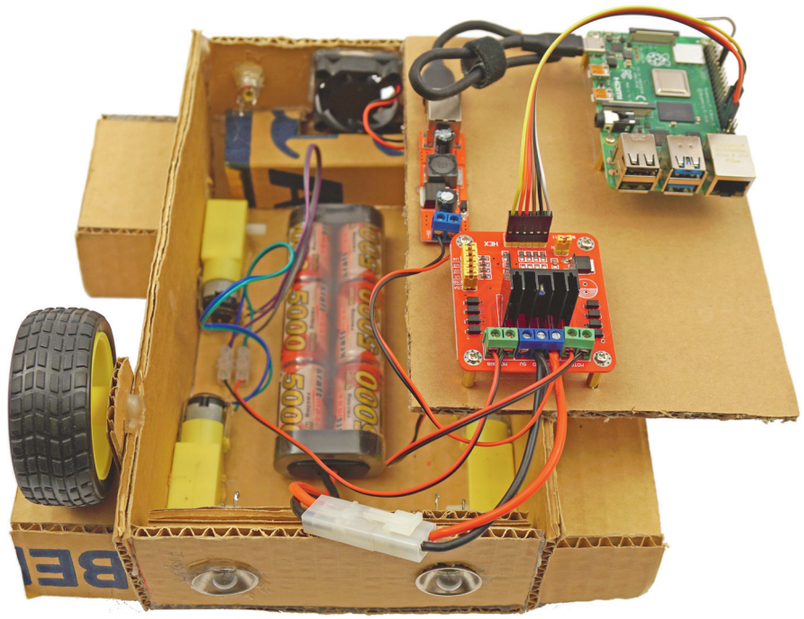


Abbildung 7.11 Gesamtbild der Verkabelung

Kapitel 19

Auslesen, verstehen und programmieren: Bringen Sie die Sensoren und Aktoren zum Laufen

Nachdem Sie in den vorigen Kapiteln die Sensoren in das Roboter-Auto eingebaut, angeschlossen und die Software installiert haben, erkläre ich Ihnen in diesem Kapitel, wie die Sensoren ausgelesen werden. Ich gebe Ihnen zu jedem Sensor ein Python-Beispielprogramm an die Hand.

Das Kapitel beginnt mit der Programmierung des Raspberry Pi Sense HAT. Nachdem Sie gelernt haben, alle für das Roboter-Auto relevanten Sensoren des Sense HAT anzusprechen und auszulesen, geht es weiter mit den Time-of-Flight-Sensoren und mit der Frage, wie Sie mit einem eigenen Python-Programm die aktuelle GPS-Position bestimmen können.

Sense-HAT zeigt Regenbogenfarben

Wenn Ihr Sense-HAT nach dem Einschalten des Roboter-Autos nur Regenbogenfarben anzeigt und nicht wie gewohnt die Anzeige erlischt, dann booted der Raspberry Pi auch nicht. Das liegt in dem mir bekannten Fall daran, dass aktuell kein Monitor angeschlossen ist. Um das Problem zu beheben, muss wie in Abschnitt 10.3, »Real VNC Server konfigurieren und Viewer installieren«, beschrieben die Konfiguration »*hdmi_force_hotplug*=1« aktiviert werden.

19.1 Raspberry Pi Sense HAT auswerten und programmieren

Damit Sie die verschiedenen Sensoren und die LED-Matrix des Raspberry Pi Sense HAT ansprechen und auswerten können, hat die Raspberry Pi Foundation eine eigene Python-API (*Application Programming Interface*) für das Sense HAT zur Verfügung gestellt. Mit dieser API können Sie aus einem selbstprogrammierten Python-Programm heraus komfortabel auf die einzelnen Komponenten des Raspberry Pi Sense HAT

zugreifen. Sie können sich diese API als das Fundament beinahe jedes Python-Programms vorstellen, das für den Raspberry Pi geschrieben worden ist.

Die vollständige Dokumentation der Python-API für das Raspberry Pi Sense HAT finden Sie mit diversen Verwendungsbeispielen unter *http://pythonhosted.org/sense-hat*.

Wollen Sie die Sense-HAT-API in Ihrem Python-Programm verwenden, binden Sie sie mit den folgenden Programmzeilen am Beginn des jeweiligen Python-Programms ein:

```
from sense_hat import SenseHat
sense = SenseHat()
```

Möchten Sie mit einem Python-Programm auf einen der drei IMU-(Inertial Measurement Unit-)Sensoren zugreifen – also auf das Magnetometer, den Beschleunigungssensor oder das Gyroskop des Raspberry Pi Sense HAT –, so müssen Sie den entsprechenden Sensor im Programm zuvor aktivieren. Es kann immer nur einer der Sensoren aktiviert sein.

Indem Sie `True` in der Klammer entsprechend setzen, aktivieren Sie den jeweiligen IMU-Sensor des Sense HAT. Mi dem Setzen von `False` deaktivieren Sie diesen Sensor wieder. Sie können allerdings im jeweiligen Programmaufruf immer nur einen der drei IMU-Sensoren aktivieren und somit auslesen. Die einzelnen Kombinationen von `True` und `False` sind in der folgenden Übersicht aufgeführt:

Magnetometer aktivieren: `set_imu_config(True, False, False)`

- Gyroskop aktivieren: `set_imu_config(False, True, False)`
- Beschleunigungssensor aktivieren: `set_imu_config(False, False, True)`

Wenn Sie in Ihrem Programm `sense.set_imu_config(False, False, False)` setzen, geben Sie den zuvor verwendeten IMU-Sensor wieder explizit frei. Anschließend können Sie einen anderen der drei Sensoren innerhalb Ihres Python-Programms aktivieren und aufrufen.

Für das autonome Fahren des Roboter-Autos werden Sie die beiden Sensoren Gyroskop und Magnetometer sowie die LED-Matrix verwenden. In den folgenden Abschnitten erkläre ich Ihnen daher die Grundlagen zur Verwendung der beiden Sensoren und der LED-Matrix anhand von Beispielprogrammen.

19.1.1 Das Python-Programm für das Gyroskop

In diesem Abschnitt erfahren Sie, wie Sie auf die Daten des Gyroskops zugreifen können. Mit dem Gyroskop wird die Lage des Raspberry Pi Sense HAT bzw. des Roboter-

Autos ermittelt. Später brauchen Sie diese Informationen, um z. B. zu überprüfen, wie weit sich das Roboter-Auto gedreht hat. Um die Orientierung zu messen, stehen Ihnen die drei Achsen des Gyroskops zur Verfügung:

- **Kippen** (englisch *pitch*): Das Roboter-Auto hebt oder senkt die Front.
- **Rollen** (englisch *roll*): Das Roboter-Auto neigt sich zur linken bzw. rechten Seite oder überschlägt sich.
- **Drehen** (englisch *yaw*): Das Roboter-Auto dreht sich auf der Stelle nach links oder rechts.

Abbildung 19.1 erklärt die Lage der drei Achsen anhand des Raspberry Pi Sense HAT. Sie müssen sich dabei vorstellen, dass sich das Roboter-Auto um diese drei Achsen drehen kann.

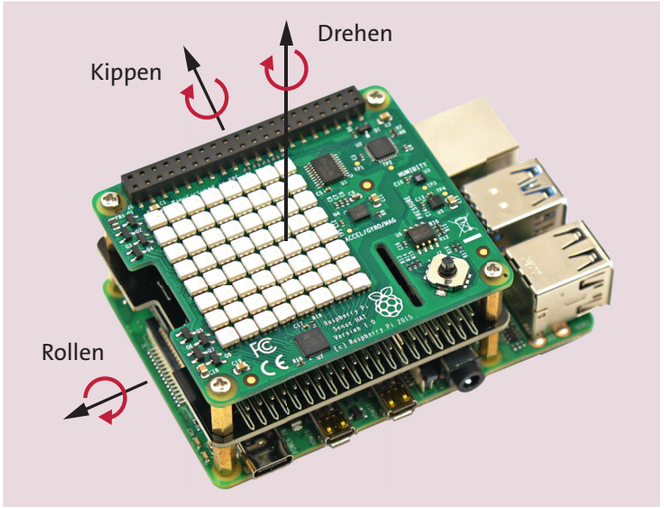


Abbildung 19.1 Die Drehachsen für das Gyroskop im Raspberry Pi Sense HAT

Mit dem Python-Programm aus Listing 19.1 können Sie die aktuelle Lage des Gyroskops auslesen und sich diese im Terminal-Fenster anzeigen lassen.

Das hierfür benötigte Python-Programm *sense-hat-gyroscope.py* können Sie von der folgenden Webseite herunterladen und auf dem Raspberry Pi im Ordner */home/pi/robot* abspeichern:

https://custom-build-robots.com/roboter-auto-download

Möchten Sie das Programm selbst erstellen, so legen Sie im Editor Notepad++ die Datei *sense-hat-gyroscope.py* an, geben den Programmcode in die Datei ein und speichern diese im Ordner */home/pi/robot* ab.

Um das Programm zu starten, geben Sie den folgenden Befehl im Terminal-Fenster ein:

```
python sense-hat-gyroscope.py
```

Im Folgenden der Quellcode des Programms *sense-hat-gyroscope.py* mit ausführlichen Kommentaren:

```
# !/usr/bin/env python
# coding:   latin-1
# Autor:    Ingmar Stapel
# Datum:    20190824
# Version:  2.1
# Homepage: https://custom-build-robots.com

# Dieses Programm greift auf das Gyroskop des Raspberry Pi
# Sense HAT zu und zeigt die Lage in Grad an.

import sys, tty, termios, os, time

# Es wird die Python-Klasse SenseHat importiert.
from sense_hat import SenseHat

# Hier werden die Variablen der drei Achsen des Gyroskops auf 0 gesetzt.
# So wird ein eventueller Fehler bei der Ausgabe vermieden,
# falls die Variablen leer sein sollten.
x = 0
y = 0
z = 0

# Ein Objekt "sense" wird von der Klasse SenseHat() erzeugt,
# um auf die Sensoren des Sense HAT zugreifen zu koennen.
sense = SenseHat()

# Jetzt wird der Sensor Gyroskop aktiviert.
sense.set_imu_config(False, True, False)

try:
   while True:
      # Hier werden die Werte der drei Achsen x, y und z
```



```
      # ausgelesen.
      x, y, z = sense.get_orientation().values()

      # Der Bildschirminhalt wird pro Schleifendurchlauf
      # geloescht.
      os.system('clear')

      # Die Anzeige wird im Terminal-Fenster ausgegeben.
      # Die Werte werden auf eine Nachkommastelle genau gerundet.
      print "Orientierung mit dem Gyroskop:"
      print "Kippen x (pitch): ",round(x,2)
      print "Rollen y (roll):  ",round(y,2)
      print "Drehen z (yaw):   ",round(z,2)

      # Damit die Schleife geregelt laeuft, wird eine sleep-
      # Phase von 0,1 Sekunden eingelegt.
      time.sleep(0.1)

# Tippt der Anwender im Terminal-Fenster Strg+C, wird das Programm
# beendet und der Sensor des Gyroskops freigegeben.
except KeyboardInterrupt:
   sense.set_imu_config(False, False, False)
# Ende des Programms
```

Listing 19.1 Das Python-Programm »sense-hat-gyroscope.py« für das Gyroskop

Die Anzeige des Programms aktualisiert sich alle 0,1 Sekunden im Terminal-Fenster und zeigt die aktuelle Lage des Roboter-Autos in Bogengrad an. Wird das Roboter-Auto z. B. an der Nase angehoben, so steigt der Wert der x-Achse (Kippen) an.

Für die in Abbildung 19.2 gezeigten Werte im Programm habe ich das Roboter-Auto auf die Seite gedreht und angehoben.

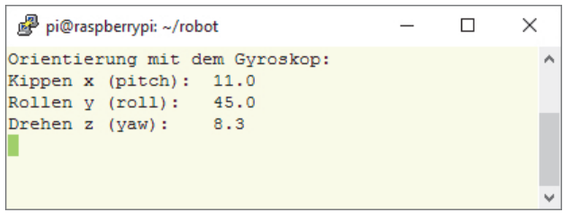


Abbildung 19.2 Anzeige der Achsenwerte mit dem Gyroskop

Sie beenden das Programm durch Drücken der Tasten Strg + C.

Mit diesem Programm haben Sie die Grundlagen zur Verwendung des Gyroskops gelernt. Ab jetzt sind Sie in der Lage, die Ausrichtung des Roboter-Autos zu ermitteln. Dieses Wissen ist notwendig, um später das autonome Fahren zu ermöglichen und mit der Unterstützung des Gyroskops das Roboter-Auto gezielt um einen bestimmten Winkel drehen zu können.

19.1.2 Das Python-Programm für das Magnetometer

Das Magnetometer ist Ihr elektronischer Kompass im Roboter-Auto. Mit dem Magnetometer können Sie feststellen, in welche Himmelsrichtung die Vorderseite des Roboter-Autos aktuell ausgerichtet ist. Abhängig davon, wie Sie das Sense HAT am Roboter-Auto befestigt haben, zeigt die Nordrichtung des Sense HAT z. B. nach vorn oder nach hinten am Roboter-Auto. Entsprechend der Ausrichtung des Sense HAT müssen Sie in Ihrer Software auf die gemessenen Werte reagieren. Das folgende Beispielprogramm geht davon aus, dass die Nordrichtung zum Heck des Roboter-Autos zeigt.

Bevor Sie das Magnetometer verwenden können, müssen Sie es kalibrieren, damit es Ihnen zuverlässig die Himmelsrichtung anzeigt.

Kalibrierung des Magnetometers

Das Magnetometer muss für den jeweiligen Verwendungsort kalibriert werden, da das Magnetfeld der Erde ortsabhängig ist und es zusätzlich zu Störungen durch die Umgebung kommen kann. Im Roboter-Auto sind solche Störquellen die Akkus, die Motoren und die weiteren elektronischen Komponenten. Ohne diese Kalibrierung wäre die Abweichung in Richtung Norden zu groß, und die Navigation mit dem Magnetometer wäre nicht möglich. Daher empfehle ich Ihnen, den Sense HAT idealerweise direkt im Roboter-Auto zu kalibrieren, da dieses mit seinen Motoren und Kabeln eigene Magnetfelder erzeugt. Die einzelnen Schritte, die Sie für die Kalibrierung benötigen, erkläre ich Ihnen im folgenden Abschnitt.

Die Kalibrierung durchführen

Wechseln Sie im Terminal-Fenster des Raspberry Pi in den Ordner */usr/share/librtimulib-utils/RTEllipsoidFit*. Starten Sie das Programm `RTIMULibCal` mit dem folgenden Befehl im Terminal-Fenster, um mit der Kalibrierung des Magnetometers zu beginnen:

```
sudo RTIMULibCal
```


Nachdem Sie den Befehl ausgeführt haben, erscheint im Terminal-Fenster das Textmenü, das Sie in Abbildung 19.3 sehen.

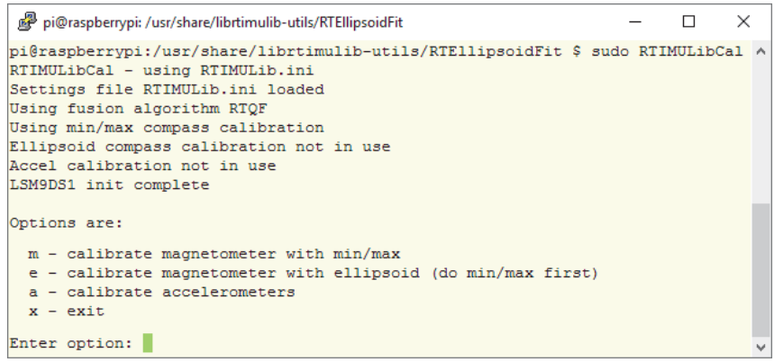


Abbildung 19.3 Das Menü des Programms »RTIMULibCal«

Sie starten die Kalibrierung, indem Sie die Taste M drücken. Nun erscheint ein kurzer Text mit einer Erläuterung, wie der Raspberry Pi nebst montiertem Sense HAT zu bewegen ist. Diesen Text bestätigen Sie mit einer beliebigen Taste, um die Kalibrierung zu starten. Jetzt erscheinen viele Zahlenreihen im Terminal-Fenster. Die Anzeige sollte bei Ihnen ähnlich wie in Abbildung 19.4 aussehen.

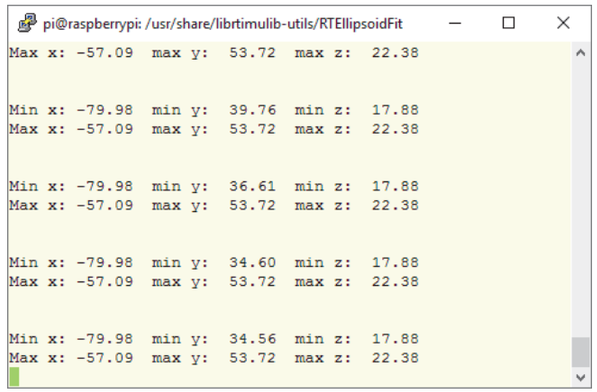


Abbildung 19.4 Anzeige der Magnetometer-Kalibrierung

Bewegen Sie das Roboter-Auto mit dem Raspberry Pi Sense HAT wie in Abbildung 19.5 dargestellt, indem Sie es rollen, drehen und kippen. Diese Bewegungen sind notwendig,

damit das Magnetometer mit der Unterstützung des Programms *RTIMULibCal* kalibriert werden kann.

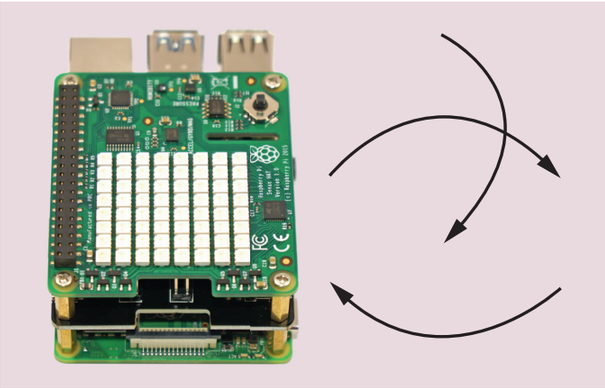


Abbildung 19.5 Drehrichtung zur Kalibrierung des Magnetometers

Führen Sie die verschiedenen Bewegungen so lange abwechselnd durch, bis sich die Zahlen im Terminal-Fenster nicht mehr verändern. Dabei spielt die Reihenfolge der Dreh-, Kipp- und Rollbewegungen keine Rolle. Bleiben die Zahlen gleich, ist die Kalibrierung abgeschlossen, und durch die Eingabe des Buchstabens [S] speichern Sie die erfassten Daten in der Konfigurationsdatei *RTIMULib.ini*. Drücken Sie dann [X], um das Programm *RTIMULibCal* zu beenden.

Wie Sie in dem Menü in Abbildung 19.3 sehen, können Sie auch den Beschleunigungssensor (Accelerometer) kalibrieren. Drücken Sie dazu die Taste [A], folgen Sie der angezeigten Beschreibung, und kalibrieren Sie wie zuvor das Magnetometer jetzt in den gleichen Schritten den Beschleunigungssensor.

Die Konfigurationsdatei RTIMULib.ini verwenden

Nachdem Sie die Konfigurationsdatei *RTIMULib.ini* gespeichert haben, liegt sie in einer aktualisierten Form im Ordner */usr/share/librtimulib-utils/RTEllipsoidFit*. Kopieren Sie diese Datei mit dem Midnight Commander in das Verzeichnis */etc/*. Falls dort schon eine ältere Version der *RTIMULib.ini*-Datei vorhanden ist, überschreiben Sie sie mit der aktuellen Version.

Die *RTIMULib.ini*-Datei wird von der Sense-HAT-API im Ordner */etc/* erwartet. Wenn Sie die Sense-HAT-API in einem Python-Programm einsetzen, das auch das Magnetometer und der Beschleunigungssensor verwendet, wird die *RTIMULib.ini*-Datei von dort eingelesen.

Die Raspberry-Pi-Sense-HAT-API legt eine Kopie der Datei *RTIMULib.ini* im Ordner */home/pi/.config/sense_hat* an. Löschen Sie dort die Datei *RTIMULib.ini* immer, wenn Sie die Kalibrierung neu durchgeführt haben. Andernfalls kann es passieren, dass die neue Datei *RTIMULib.ini* aus dem Ordner */etc/* nicht von der Sense-HAT-API verwendet wird. Ich habe bei meinen ersten Kalibrierungen stundenlang den Fehler gesucht, bis ich die Kopie der alten Datei fand und sie als Fehlerquelle ausmachen konnte. Wenn Sie die Datei nicht überschreiben können, liegt das eventuell an den Rechten Ihres aktuellen Users. Starten Sie den Midnight Commander mit Adminrechten, und überschreiben Sie anschließend die Datei.

```
sudo mc
```

Sie haben die Kalibrierung des Magnetometers und des Beschleunigungssensors erfolgreich abgeschlossen, die *RTIMULib.ini*-Datei aktualisiert, die alte *RTIMULib.ini*-Datei gelöscht (falls vorhanden) und so alles für die Verwendung des Magnetometers im Roboter-Auto vorbereitet. Im folgenden Abschnitt erkläre ich Ihnen, wie Sie in einem Python-Programm das Magnetometer auslesen können.

Das Python-Programm für das Magnetometer

Das Python-Programm in Listing 19.2 liest die Abweichung des Magnetometers zum Nordpol in Grad aus und zeigt Ihnen die entsprechende Gradzahl im Terminal-Fenster an. Das hierfür benötigte Python-Programm *sense-hat-magnetometer.py* können Sie von der folgenden Webseite herunterladen und auf dem Raspberry Pi im Ordner */home/pi/robot* abspeichern:

https://custom-build-robots.com/roboter-auto-download

Möchten Sie das Programm eigenständig erstellen, so legen Sie im Editor Notepad++ die Datei *sense-hat-magnetometer.py* an, geben den Programmcode in die Datei ein und speichern diese im Ordner */home/pi/robot* ab.

Um das Programm zu starten, geben Sie den folgenden Befehl im Terminal-Fenster ein:

```
python sense-hat-magnetometer.py
```

Im folgenden Quellcode erkläre ich Ihnen mit Kommentierungen die Funktionsweise des Programms *sense-hat-magnetometer.py*:

```
# !/usr/bin/env python
# coding:   latin-1
# Autor:    Ingmar Stapel
# Datum:    20190810
# Version:  2.1
```

```
# Homepage: https://custom-build-robots.com

# Dieses Programm greift auf das Magnetometer des Raspberry Pi
# Sense HAT zu und zeigt die Abweichung nach Norden in Grad an.

import sys, tty, termios, os, time

# Die Python-Klasse "SenseHat" wird importiert.
from sense_hat import SenseHat

# Hier wird die Variable "degree initial" gesetzt, damit ein Fehler
# bei der Ausgabe der Variablen vermieden wird, wenn diese leer ist.
degree = 0

# Ein Objekt "sense" wird von der Klasse "SenseHat()" erzeugt,
# um auf die Sensoren des Sense HAT zugreifen zu koennen.
sense = SenseHat()

# Jetzt wird der Sensor Magnetometer aktiviert.
sense.set_imu_config(True, False, False)

# Die Endlosschleife liest so lange das Magenetometer aus, bis der
# Anwender im Terminal-Fenster Strg+C tippt.

# Mit dieser Variable wird festgelegt, um wie viel Grad die Nordrichtung
# des Sense HAT zur Vorderseite des Roboter-Autos abweicht.
s_h_direction = 180

# Der Variablen new_degree wird initial der Wert degree uebergeben.
# Sie dient dazu, bei einer eventuellen Abweichung von s_h_direction
# die neu berechnete Nordrichtung des Roboter-Autos zu speichern.
new_degree = 0

try:
   while True:
        # Es wird die Gradzahl Richtung Norden ausgelesen.
        north = sense.get_compass()

        # Die Gradzahl soll als Float-Wert angezeigt werden.
```



```
        degree = float(north)

        # Abhaengig von der Ausrichtung des Sense HAT am
        # Roboter-Auto muss die Nordrichtung korrigiert werden.
        # In diesem Beispiel zeigt die Nordrichtung des Sense
        # HAT nach hinten am Roboter-Auto.
        new_degree = degree
        if degree <= s_h_direction and s_h_direction != 0:
            new_degree = degree + s_h_direction
        else:
            new_degree = degree - s_h_direction

        # Der Bildschirminhalt wird pro Schleifendurchlauf
        # geloescht.
        os.system('clear')

        # Die Anzeige wird im Terminal-Fenster ausgegeben.
        print "Orientierung mit dem Kompass:"
        print("Nordrichtung: %s" % round(new_degree,1))

        # Damit die Schleife geregelt laeuft, wird eine sleep-
        # Phase von 0,1 Sekunden eingelegt.
        time.sleep(0.1)

# Tippt der Anwender im Terminal-Fenster Strg+C, wird das Programm
# beendet und der Sensor des Magnetometers freigegeben.
except KeyboardInterrupt:
   sense.set_imu_config(False, False, False)
# Ende des Programms
```

Listing 19.2 Das Python-Programm »sense-hat-magnetometer.py« für das Magnetometer

Die Anzeige des Programms aktualisiert sich alle 0,1 Sekunden und zeigt Ihnen, wie stark die Ausrichtung des Roboter-Autos von der Richtung Norden abweicht. Indem Sie das Roboter-Auto drehen, können Sie es in Richtung Süden ausrichten. Dazu drehen Sie das Roboter-Auto so lange, bis die Gradzahl 180° im Terminal-Fenster erscheint.

Abbildung 19.6 zeigt eine Abweichung von 121,6° in Richtung Norden an.

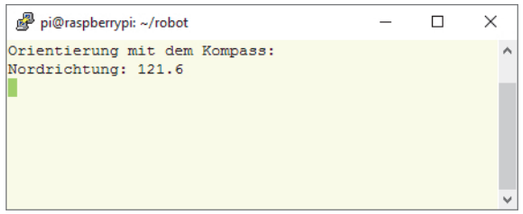


Abbildung 19.6 Anzeige der Nordrichtung mit dem Magnetometer

Sie beenden das Programm durch Drücken der Tasten Strg + C.

Norden ist doch ganz woanders!

Ihre Erfahrung sagt Ihnen, dass die Anzeige des Sense HAT für Norden nicht stimmt? Dann empfehle ich Ihnen, das Sense HAT erneut zu kalibrieren.

Mit dem Kalibrieren des Magnetometers und dem Programm *sense-hat-magnetometer.py* haben Sie das Grundwissen dazu erworben, wie Sie das Roboter-Auto gezielt in eine bestimmte Himmelsrichtung ausrichten.

19.1.3 Das Python-Programm für die LED-Matrix

In diesem Abschnitt erkläre ich Ihnen, wie Sie die LED-Matrix des Raspberry Pi Sense HAT mit einem Python-Programm ansprechen können. Dieses Python-Programm ist als gekapseltes Modul aufgebaut, wie Sie es vom Motortreiber-Modul bereits kennen. So können Sie es ohne wiederholenden Programmieraufwand in Ihre Programme einbinden und die LED-Matrix ansprechen.

Die LED-Matrix des Sense HAT mit ihren 8 × 8-RGB-LEDs eignet sich z. B. dazu, Statusinformationen des Roboter-Autos auszugeben. Über die LED-Matrix können Sie die Entfernung des Autos zu einem Hindernis mit einem Farbcode oder auch als Laufschrift in Zentimeter ausgeben. Im weiteren Verlauf dieses Abschnitts gehe ich zuerst auf das Modul *ledmatrix.py* ein, in dem die Funktionen der LED-Matrix gekapselt sind. Anschließend folgt ein Python-Testprogramm, das auf der LED-Matrix die Temperatur, den Luftdruck und die Luftfeuchtigkeit als Lauftext ausgibt.

Python-Modul für die LED-Matrix

Ich habe in dem gekapselten Modul *ledmatrix.py* die Funktionen abgebildet, die die Raspberry-Pi-Sense-HAT-API mitbringt und die für das Roboter-Auto relevant sind. Das

Modul verfügt über sieben Funktionen, mit denen Sie Text, Bilder oder Icons auf der LED-Matrix anzeigen lassen können:

Python-Funktion	Beschreibung
`display_letter`	Zeigt ein Zeichen auf der LED-Matrix an.
`display_message`	Zeit einen Text in Form einer Laufschrift an.
`display_pixels`	Zeigt einzelne Pixel (z. B. einen Pfeil) an.
`display_rotate`	Lässt die Anzeige um 90°, 180° oder 270° rotieren.
`display_image`	Zeigt ein Bild mit 8 × 8 Pixeln in RGB-Farben an.
`display_clear`	Löscht die Anzeige.
`display_flip`	Spiegelt das Bild horizontal oder vertikal.

Tabelle 19.1 Python-Funktionen zum Ansteuern der LED-Matrix

Mit diesem Set an Funktionen werden Sie in den weiteren Programmen in diesem Buch die Anzeige auf der LED-Matrix realisieren.

Das Python Modul *ledmatrix.py* können Sie von der folgenden Webseite herunterladen und auf dem Raspberry Pi im Ordner */home/pi/robot* abspeichern:

https://custom-build-robots.com/roboter-auto-download

Möchten Sie das Programm selbst erstellen, so legen Sie im Editor Notepad++ die Datei *ledmatrix.py* an, geben den Programmcode in die Datei ein und speichern diese im Ordner */home/pi/robot* ab.

Das Modul *ledmatrix.py* kann nicht allein ausgeführt werden. Sie binden es in Ihr Python-Programm ein, mit dem Sie auf der LED-Matrix etwas anzeigen wollen. Wie Sie das machen, erfahren Sie in Abschnitt 19.1.3

```
#!/usr/bin/env python
# coding:   latin-1
# Autor:    Ingmar Stapel
# Datum:    20190810
# Version:  2.0
# Homepage: https://custom-build-robots.com
# Modul, das die Anzeige von Informationen auf der
# LED-Matrix des Raspberry Pi Sense HAT umsetzt
```

```
import time, os

# Sense-HAT-Bibliothek importieren
from sense_hat import SenseHat

# SenseHat initialisieren
sense = SenseHat()

# Farben für die Symbole auf der LED-Anzeige festlegen
g = [0, 255, 0]
r = [255, 0, 0]
e = [0, 0, 0]

# Ende-Symbol fuer die LED-Matrix
icon_finish = [
e,e,e,r,r,e,e,e,
e,e,r,e,e,r,e,e,
e,r,e,e,e,e,r,e,
r,e,e,r,r,e,e,r,
r,e,e,r,r,e,e,r,
e,r,e,e,e,e,r,e,
e,e,r,e,e,r,e,e,
e,e,e,r,r,e,e,e
]

# Pfeil als Symbol fuer die LED-Matrix
icon_arrow = [
e,e,e,g,g,e,e,e,
e,e,g,g,g,g,e,e,
e,g,e,g,g,e,g,e,
g,e,e,g,g,e,e,g,
e,e,e,g,g,e,e,e,
e,e,e,g,g,e,e,e,
e,e,e,g,g,e,e,e,
e,e,e,g,g,e,e,e
]

def display_letter(text,txt_colour,bg_color):
    if bg_color =="":
        bg_color = [255,255,255]
    if txt_colour =="":
```



```
        txt_colour = [0,0,0]
    # Der Inhalt der Variablen "text" wird explizit in einen String
    # umgewandelt, damit Fehler bei der Anzeige vermieden werden.
    text = str(text)
    # Anzeige des Textes auf der LED-Matrix
    sense.show_letter(text,text_colour=txt_colour,\
    back_colour=bg_color)
    return -1

def display_message(text,txt_colour,bg_color, speed):
    if bg_color =="":
        bg_color = [255,255,255]
    if txt_colour =="":
        txt_colour = [0,0,0]
    # Der Inhalt der Variablen "text" wird explizit in einen String
    # umgewandelt, damit Fehler bei der Anzeige vermieden werden.
    text = str(text)
    # Anzeige der Laufschrift auf der LED-Matrix
    sense.show_message(text,scroll_speed=speed,\
    text_colour=txt_colour,back_colour=bg_color)
    return -1

def display_pixels(icon):
    if icon == "arrow":
        sense.set_pixels(icon_arrow)
    elif icon == "finish":
        sense.set_pixels(icon_finish)
    return -1

def display_rotate(value):
    # Die Sense-HAT-API akzeptiert nur Winkel von 0, 90, 180
    # und 270 Grad, um die die Anzeige gedreht werden kann.
    if value == 0:
        sense.set_rotation(value)
    elif value == 90:
        sense.set_rotation(value)
    elif value == 180:
        sense.set_rotation(value)
    elif value == 270:
        sense.set_rotation(value)
    return -1
```

```
def display_image(file_path):
    # Pruefen, ob es die Datei ueberhaupt gibt
    if os.path.isfile(file_path)== True:
        sense.load_image(file_path,redraw=True)
    return -1

def display_clear():
    # Loescht die aktuelle Anzeige auf der LED-Matrix.
    sense.clear(255, 255, 255)
    return -1

def display_flip(flip):
    # Spiegelt die aktuelle Anzeige auf der LED-Matrix vertikal.
    if flip == "v":
        sense.flip_v()
    # Spiegelt die aktuelle Anzeige auf der LED-Matrix horizontal.
    elif flip =="h":
        sense.flip_h()
    return -1
# Ende des Programmes
```

Listing 19.3 Das Python-Modul »ledmatrix.py« für Anzeigen auf der LED-Matrix

Python-Beispielprogramm für die LED-Matrix

Nachdem Sie das Python-Modul *ledmatrix.py* programmiert oder heruntergeladen haben, werden Sie mit dem Programm *sense-hat-led-matrix.py* dieses Modul einbinden und aufrufen. Mit dem folgenden Import-Befehl können Sie das Modul *ledmatrix.py* in den von Ihnen erstellten Python-Programmen zu Beginn importieren:

```
import ledmatrix as matrix
```

Mit diesem Import stehen Ihnen die im vorigen Abschnitt beschriebenen sieben Funktionen für die Anzeige auf der LED-Matrix zur Verfügung.

Mit dem folgenden *sense-hat-led-matrix.py*-Programm werden die Temperatur-, Luftdruck- und Luftfeuchtigkeitswerte auf der LED-Matrix als Laufschrift dargestellt. Dazu verwenden Sie die Funktion `display_message()` aus dem *ledmatrix.py*-Modul, deren Verwendung ich Ihnen im Folgenden erkläre.

Mit dem Aufruf von

```
matrix.display_message(text,txt_colour,bg_color, speed)
```


übergeben Sie in der Variablen `text` die Werte für Temperatur, Luftdruck und Luftfeuchtigkeit an das Modul. Mit der Variablen `txt_colour` legen Sie die Farbe des Textes auf der LED-Matrix fest. In der Variablen `bg_color` legen Sie die Hintergrundfarbe der LED-Matrix fest. Fällt die Temperatur unter 20 °C oder steigt sie über 27 °C, färbt sich der Hintergrund des Displays rot. Andernfalls ist der Hintergrund grün. Mit der Variablen `speed` können Sie bestimmen, wie schnell der Text durch die LED-Matrix läuft.

Das hierfür benötigte Python-Programm *sense-hat-led-matrix.py* können Sie von der folgenden Webseite herunterladen und auf dem Raspberry Pi im Ordner */home/pi/robot* abspeichern:

https://custom-build-robots.com/roboter-auto-download

Möchten Sie das Programm selbst erstellen, so legen Sie im Editor Notepad++ die Datei *sense-hat-led-matrix.py* an, geben den Programmcode in die Datei ein und speichern diese im Ordner */home/pi/robot* ab.

Geben Sie den folgenden Befehl im Terminal-Fenster ein, um das Programm zu starten:

```
python sense-hat-led-matrix.py
```

In Listing 19.3 erkläre ich Ihnen den Quellcode des Programms *sense-hat-led-matrix.py* wie gewohnt mit Kommentierungen der einzelnen Funktionen.

```
# !/usr/bin/env python
# coding:   latin-1
# Autor:    Ingmar Stapel
# Datum:    20190824
# Version:  2.1
# Homepage: https://custom-build-robots.com

# Dieses Programm greift auf die Sensoren fuer Luftdruck, Temperatur
# und Luftfeuchtigkeit des Raspberry Pi Sense HAT zu und zeigt
# die Werte auf der LED-Matrix des Raspberry Pi Sense HAT an.

import sys, tty, termios, os, time

# Die Python-Klasse "SenseHat" wird importiert.
from sense_hat import SenseHat

# Die Python-Klasse "ledmatrix" wird importiert.
import ledmatrix as matrix
```

```
# Ein Objekt "sense" wird von der Klasse "SenseHat()" erzeugt,
# um auf die Sensoren des Sense HAT zugreifen zu koennen.
sense = SenseHat()

try:
   while True:
      # Es werden die Werte fuer die Temperatur, den Luftdruck
      # und die Luftfeuchtigkeit aus dem Sensor ausgelesen.
      temp = sense.get_temperature()
      pres = sense.get_pressure()
      humi = sense.get_humidity()

      # Die ausgelesenen Werte werden auf eine Stelle nach
      # dem Komma gerundet.
      temp = round(temp, 1)
      pres = round(pres, 1)
      humi = round(humi, 1)

      # Die LED-Matrix faerbt sich gruen oder rot, je nachdem,
      # ob die gemessene Temperatur im definierten gueltigen
      # Intervall liegt.
      if temp > 20 and temp < 27:
         bg = [0, 100, 0]  # Gruen
      else:
         bg = [100, 0, 0]  # Rot

      # Hier wird in der Variablen "msg" der Text zusammen-
      # gesetzt, der auf der LED-Matrix ausgegeben
      # werden soll.
      msg = "Temp=%s, Press=%s, Humi=%s"%(temp, pres, humi)

      # In Form einer Laufschrift werden die einzelnen
      # gemessenen Werte Temperatur, Luftdruck und
      # Luftfeuchtigkeit auf der LED-Matrix ausgegeben.
      matrix.display_message(msg,[250,250,50],bg,0.06)

# Tippt der Anwender im Terminal-Fenster Strg+c, wird das Programm
# beendet und der Sensor des Magnetometers freigegeben.
```



```
except KeyboardInterrupt:
    sense.set_imu_config(False, False, False)
# Ende des Programms
```

Listing 19.4 Das Python-Programm »sense-hat-led-matrix.py« für eine Anzeige auf der LED-Matrix

Ab jetzt sind Sie in der Lage, aus Ihren Python-Programmen heraus komfortabel Anzeigen auf der LED-Matrix des Raspberry Pi Sense HAT zu realisieren.

19.2 Python-Programm zur Verarbeitung der GPS-Koordinaten

In Abschnitt 18.3 haben Sie die Python-GPS-Bibliothek auf dem Raspberry Pi installiert und den gpsd-Daemon kennengelernt. Das folgende Python-Programm *gps-test-read.py* greift über die Python-GPS-Bibliothek auf den gpsd-Daemon zu. Dieser stellt dem Programm die GPS-Informationen des GPS-Empfängers zur Verfügung.

Damit das Python-Programm *gps-test-read.py* die GPS-Informationen einlesen kann, muss der gpsd-Daemon aktiv laufen. Das Programm startet diesen selbstständig. Wenn es zu Problemen kommen sollte, können Sie den gpsd-Daemon manuell starten wie in Abschnitt 18.3 erklärt.

Das Python-Programm *gps-test-read.py* können Sie von der folgenden Webseite herunterladen und auf dem Raspberry Pi im Ordner */home/pi/robot* abspeichern:

https://custom-build-robots.com/roboter-auto-download

Möchten Sie das Programm eigenständig erstellen, so legen Sie im Editor Notepad++ die Datei *gps-test-read.py* an, geben den Programmcode in die Datei ein und speichern diese im Ordner */home/pi/robot* ab.

Um das Programm zu starten, geben Sie den folgenden Befehl im Terminal-Fenster ein:

```
sudo python gps-test-read.py
```

In Listing 19.5 sehen Sie den Quellcode des Programms *gps-test-read.py*. Ich habe innerhalb des Programms Erklärungen eingefügt, die Ihnen beim Verstehen des Codes helfen.

```
#!/usr/bin/env python
# coding:   latin-1
# Autor:    Ingmar Stapel
# Datum:    20190810
# Version:  2.1
# Homepage: https://custom-build-robots.com
```

```
# Mit diesem Programm kann der GPS-Empfaenger ausgelesen werden.

import os
from gps import *
import time
import subprocess

# Der Zaehler zeigt dem Anwender, wie oft versucht wurde,
# die Koordinaten auszulesen.
count = 0

# Der gpsd-Daemon wird im Hintergrund gestartet.
print("Der gpsd-Daemon wird mit dem USB-Geraet /dev/ttyACM0 gestartet.")
process = subprocess.Popen(["sudo", "gpsd", "-b", "/dev/ttyACM0", "-F", "/var/ ↩
run/gpsd.sock", "-G"])
print("GPSD PID: ", process.pid)

print("Der gpsd-Daemon wurde gestartet.")
time.sleep(2)

# Erzeugen der Session fuer den Zugriff auf den Empfaenger
session = gps(mode=WATCH_ENABLE)

try:
   while True:
      # Warten, um die while-Schleife etwas zu verzoegern
      time.sleep(0.5)

      # Loeschen des Terminal-Fensters
      os.system('clear')

      # Ausgabe der Statuszeile, dass das Programm laeuft
      print 'Auslesen der GPS-Koordinaten'

      # Der Counter wird hochgesetzt.
      count = count + 1

      # Der Anwender wird ueber den Lesezugriff informiert.
      print 'Zaehler:', count ,' Lesezugriffe.'
      session.next()
```



```
      # Wenn der GPS-Empfaenger keinen Fixpunkt findet,
      # wird der Anwender darueber informiert.
      if session.fix.latitude == 0.0:
         print '----------------------------------------'
         print 'Es wurden keine GPS-Informationen empfangen'
         print 'Wurde der gpsd-Daemon schon gestartet?'
         print '----------------------------------------'
      # Werden GPS-Koordinaten empfangen, so werden diese
      # ausgegeben.
      else:
         print '------------------ GPS-Informationen --------------'
         print 'Breitengrad:  ' , session.fix.latitude
         print 'Laengengrad:  ' , session.fix.longitude
         print 'Zeit utc:     ' , session.utc, \
         session.fix.time
         print '---------------------------------------------------'
# Tippt der Anwender im Terminal-Fenster Strg+c, wird das Programm
# beendet und der GPSD Daemon ebenfalls.
except KeyboardInterrupt:
    print("Programm beendet: ")

# Ende des Programms
```

Listing 19.5 Das Python-Programm »gps-test-read.py« zum Auslesen des GPS-Empfängers

In Abbildung 19.7 sehen Sie das Programm bei der Arbeit. Es zeigt die GPS-Koordinaten an, die der GPS-Empfänger erhalten hat, und den dazugehörigen Zeitstempel des GPS-Signals.

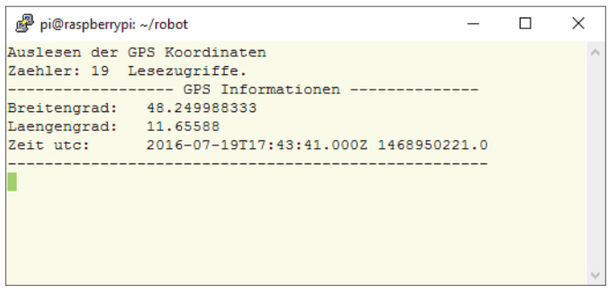


Abbildung 19.7 Anzeige der GPS-Koordinaten

Ab jetzt können Sie die GPS-Koordinaten in eigenen Python-Programmen verarbeiten. Im nächsten Schritt soll das Roboter-Auto in die Richtung einer bestimmten Koordinate gesteuert werden. Wie das funktioniert, erkläre ich Ihnen in Abschnitt 21.2, »GPS-Wegpunkte abfahren«, noch ausführlich.

19.3 Python-Programme für den Time-of-Flight-Sensor

Damit Sie die Entfernung zu einem Hindernis vor und hinter dem Roboter-Auto messen können, brauchen Sie ein Python-Programm, mit dem Sie den Time-of-Flight-Sensor auslesen können. Die hier vorgestellte Programmierung wurde so aufgebaut, dass die Logik in zwei Teile aufgeteilt ist. Das Programm *read_VL53L1X.py* greift auf den ToF-Sensor zu und liest die Messwerte über den I^2C-Bus aus. Das Programm *display_VL53L1X.py* zeigt die vom Programm *read_VL53L1X.py* ausgelesenen Messwerte im Terminal-Fenster an. Mit diesem modularen Aufbau wird eine große Flexibilität erreicht, was die spätere Verwendung der ToF-Sensoren in anderen Programmen vereinfacht.

19.3.1 Programm für den Lesezugriff auf den ToF-Sensor

Das Python-Programm *read_VL53L1X.py* können Sie von der folgenden Webseite herunterladen und auf dem Raspberry Pi im Ordner */home/pi/robot* abspeichern:

https://custom-build-robots.com/roboter-auto-download

Möchten Sie das Programm eigenständig erstellen, so legen Sie im Editor Notepad++ die Datei *read_VL53L1X.py* an, geben den Programmcode in die Datei ein und speichern diese im Ordner */home/pi/robot* ab.

Nachfolgend der Programmcode des Programms *read_VL53L1X.py*.

```
#!/usr/bin/env python
# coding:  latin-1
# Autor:   Ingmar Stapel
# Datum:   20190810
# Version: 1.0
# Homepage: https://custom-build-robots.com
# Dieses Programm liest die Werte eines VL53L1X-ToF-
# Sensors aus.
# Bei Aufruf muss immer der GPIO-Pin mit uebergeben
# werden, an dem der ToF-Sensor mit seinem XSHUT-Pin
# angeschlossen ist.
```



```
# Es werden verschiedene Python-Klassen importiert, deren Funktionen
# fuer die Programmverarbeitung benoetigt werden.
import time
import VL53L1X

# Es wird die Klasse RPi.GPIO importiert, die die Ansteuerung
# der GPIO-Pins des Raspberry Pi ermoeglicht.
import RPi.GPIO as io
io.setmode(io.BCM)

# Hier wird ein Objekt tof angelegt, mit dem der Zugriff via I2C-Bus
# auf den ToF-Sensor erfolgt.
tof = VL53L1X.VL53L1X(i2c_bus=1, i2c_address=0x29)

# Der Sensor wird mit dieser Funktion aktiv gesetzt. Dazu
# muss die GPIO-Pin-Nummer mit uebergeben werden.
def start_sensor(number):
    # Der XSHUT-Pin wird auf HIGH gesetzt, um den Sensor zu aktivieren.
    io.setup(number, io.OUT)
    io.output(number, True)
    time.sleep(0.2)

    # Hier erfolgt der Zugriff auf den ToF-Sensor.
    tof.open() # initialisieren des I2C Bus und starten des Sensors
    # Die Messung am gewaehlten Sensor wird gestartet.
    # Es gibt drei Modi fuer die Messung:
    # 1 = Short Range,
    # 2 = Medium Range,
    # 3 = Long Range
    tof.start_ranging(1)

# Diese Funktion haelt die Entfernungsmessung an. Die GPIO-Pin-
# Nummer muss mit uebergeben werden, um den ToF-Sensor inaktiv zu
# setzen.
def stop_sensor(number):
    # Die Messung am gewaehlten Sensor wird angehalten.
    tof.stop_ranging()
    # Der XSHUT-Pin wird auf LOW gesetzt, um den Sensor zu deaktivieren.
    io.output(number, False)
```

```
# Die Funktion liest die gemessene Entfernung aus.
def get_distance():
    distance_in_mm = 0
    distance_in_mm = tof.get_distance()
    return distance_in_mm
# Programmende
```

Listing 19.6 Das Python-Programm »read_VL53L1X.py« misst die Entfernung mit dem VL53L1X-ToF-Sensor.

19.3.2 Programm für die Anzeige der gemessenen Entfernungen mit dem ToF-Sensor

Für die Ausgabe der Messwerte im Terminal-Fenster müssen Sie noch das Programm *display_VL53L1X.py* anlegen. Das Python-Programm *display_VL53L1X.py* können Sie von der folgenden Webseite herunterladen und auf dem Raspberry Pi im Ordner */home/pi/robot* abspeichern:

https://custom-build-robots.com/roboter-auto-download

Möchten Sie das Programm eigenständig erstellen, so legen Sie im Editor Notepad++ die Datei *display_VL53L1X.py* an, geben den Programmcode in die Datei ein und speichern diese im Ordner */home/pi/robot* ab.

Um das Programm zu starten, geben Sie den folgenden Befehl im Terminal-Fenster ein:

```
python display_VL53L1X.py
```

Nachfolgend der Programmcode des Programms *display_VL53L1X.py*.

```
#!/usr/bin/env python
# coding:  latin-1
# Autor:   Ingmar Stapel
# Datum:   20190810
# Version: 1.0
# Homepage: https://custom-build-robots.com
# Dieses Programm gibt die Werte eines VL53L1X-ToF-Sensors
# auf der Konsole aus.

# Es werden verschiedene Python-Klassen importiert, deren Funktionen
# fuer die Programmverarbeitung benoetigt werden.
import time
```



```
# Das Programm read_VL53L1X wird als Modul geladen und liest einen
# VL53L1X-ToF-Sensor aus.
import read_VL53L1X as dist

# Der gewaehlte ToF-Sensor am GPIO-Pin 23 wird aktiviert.
dist.start_sensor(23)

# Es werden 30 Messwerte mit der For-Schleife ausgegeben.
for x in range(30):
    # Hier erfolgt der Zugriff auf den aktiven ToF-Sensor.
    dist_mm = dist.get_distance()
    print("ToF-Sensor hinten - GPIO-Pin 23:")
    print("Entfernung: {}mm".format(dist_mm))

# Der gewahlte ToF-Sensor am GPIO-Pin 23 wird inaktiv geschaltet.
dist.stop_sensor(23)

time.sleep(0.5)
# Der gewaehlte ToF-Sensor am GPIO-Pin 24 wird aktiviert.
dist.start_sensor(24)

# Es werden 30 Messwerte mit der For-Schleife ausgegeben.
for x in range(30):
    # Hier erfolgt der Zugriff auf den aktiven ToF-Sensor.
    dist_mm = dist.get_distance()
    print("ToF-Sensor hinten - GPIO-Pin 24:")
    print("Entfernung: {}mm".format(dist_mm))

# Der gewahlte ToF-Sensor am GPIO-Pin 24 wird inaktiv geschaltet.
dist.stop_sensor(24)
# Programmende
```

Listing 19.7 Das Python-Programm »display_VL53L1X.py« gibt die gemessene Entfernung eines ToF-Sensors im Terminal-Fenster aus.

Die Ausgabe des Programms *display_VL53L1X.py* sieht im Terminal-Fenster wie in Abbildung 19.8 gezeigt aus.

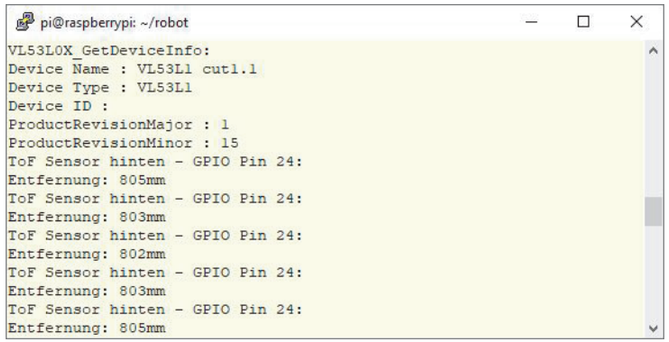


Abbildung 19.8 Programmausgabe der Entfernungsmessung mit dem VL53L1X-ToF-Sensor

19.4 Python-Programme für den Servocontroller

In diesem Kapitel werde ich Ihnen zwei Programme vorstellen, die den PCA9685-Servocontroller verwenden. Das erste Programm ist das Ihnen bereits bekannte Steuerungsprogramm für die Drehgeschwindigkeit der Motoren. Das zweite Programm ist neu, und Sie können es dazu verwenden zwei Servomotoren anzusteuern oder ein Pan-Tilt-Kit mit einer Kamera nach Ihren Wünschen zu drehen und zu schwenken.

19.4.1 Steuerungsprogramm für den L298N-Motortreiber mit dem Servocontroller

Sie werden im folgenden Abschnitt das Ihnen schon bekannte Programm *L298NHBridge.py* aus Abschnitt 12.2.1, »Das Steuerungsprogramm für den Motortreiber in Python«, so erweitern, dass das PWM-Signal für die Regelung der Drehgeschwindigkeit der Motoren nicht mehr vom Raspberry Pi generiert wird, sondern vom PCA9685-Servocontroller.

Das Python-Programm *L298NHBridgePCA9685.py* können Sie von der folgenden Webseite herunterladen und auf dem Raspberry Pi im Ordner */home/pi/robot* abspeichern:

https://custom-build-robots.com/roboter-auto-download

Möchten Sie das Programm eigenständig erstellen, so legen Sie im Editor Notepad++ die Datei *L298NHBridgePCA9685.py* an, geben den Programmcode in die Datei ein und speichern diese im Ordner */home/pi/robot* ab.

Nachfolgend der Programmcode des Programms *L298NHBridgePCA9685.py*.


```
#!/usr/bin/env python
# coding:   latin-1
# Autor:    Ingmar Stapel
# Datum:    20190810
# Version:  1.0
# Homepage: https://custom-build-robots.com

# Dieses Programm wurde fuer die Ansteuerung der linken und rechten
# Motoren des Roboter-Autos entwickelt. Es geht dabei davon aus,
# dass eine L298N-H-Bruecke als Motortreiber eingesetzt wird.
# Das PWM-Signal fuer die Steuerung der Drehgeschwindikgkeit der
# Motoren wird von einem Adafruit-Servocontroller generiert.

# Dieses Programm muss von einem uebergeordneten Programm aufgerufen
# werden, das die Steuerung des Programms L298NHBridge übernimmt.

# Es wird die Klasse RPi.GPIO importiert, die die Ansteuerung
# der GPIO-Pins des Raspberry Pi ermoeglicht.

from __future__ import division
import RPi.GPIO as io
io.setmode(io.BCM)

import time

# Importiere die Adafruit-PCA9685-Bibliothek
import Adafruit_PCA9685

# initialisieren des PCA9685 mit der standard Adresse (0x40).
PCA9685_pwm = Adafruit_PCA9685.PCA9685()

# Hier wird die Frequenz des PWM-Signals gesetzt.
# Gute Erfahrungen fuer die L298N-H-Bruecke habe ich mit 60 Hz gemacht.
# Auch arbeiten die meisten Servomotoren mit 60 Hz.
PCA9685_pwm.set_pwm_freq(60)

# Die Variable duty_cycle gibt die maximale Einschaltdauer der
# Motoren pro 100 Hertz vor. Sie liegt zwischen 0 und 4095.
# Für die Geschwindigkeit der Motoren beginnt die Einschaltdauer
# immer bei 0 und endet bei einem Wert ]0, 4095[.
```

```
duty_cycle = 4095

# Mit dem folgenden Aufruf werden eventuelle Warnungen, die die
# Klasse RPi.GPIO ausgibt, deaktiviert.
io.setwarnings(False)

# Im folgenden Programmabschnitt wird die logische Verkabelung des
# Raspberry Pi im Programm abgebildet. Dazu werden den vom Motor-
# treiber bekannten Pins die GPIO-Adressen zugewiesen.

# --- START KONFIGURATION GPIO-Adressen ---
IN1 = 6
IN2 = 13
IN3 = 19
IN4 = 26
# --- ENDE KONFIGURATION GPIO-Adressen ---

# ENA und ENB der L298N-H-Bruecke sind mit dem Kanal 0 und 1
# des Servocontrollers verbunden.
# ENA = Kanal 0
# ENB = Kanal 1

# Der Variablen leftmotor_in1_pin wird die Varibale IN1 zugeordnet.
# Der Variablen leftmotor_in2_pin wird die Varibale IN2 zugeordnet.
leftmotor_in1_pin = IN1
leftmotor_in2_pin = IN2
# Beide Variablen, leftmotor_in1_pin und leftmotor_in2_pin, werden als
# Ausgaenge "OUT" definiert. Mit den beiden Variablen wird die
# Drehrichtung der Motoren gesteuert.
io.setup(leftmotor_in1_pin, io.OUT)
io.setup(leftmotor_in2_pin, io.OUT)

# Der Variablen rightmotor_in1_pin wird die Varibale IN1 zugeordnet.
# Der Variablen rightmotor_in2_pin wird die Varibale IN2 zugeordnet.
rightmotor_in1_pin = IN3
rightmotor_in2_pin = IN4
# Beide Variablen, rightmotor_in1_pin und rightmotor_in2_pin, werden
# als Ausgaenge "OUT" definiert. Mit den beiden Variablen wird die
# Drehrichtung der Motoren gesteuert.
io.setup(rightmotor_in1_pin, io.OUT)
```



```
io.setup(rightmotor_in2_pin, io.OUT)

# Die GPIO-Pins des Raspberry Pi werden initial auf False gesetzt.
# So ist sichergestellt, dass kein HIGH-Signal anliegt und der
# Motortreiber nicht unbeabsichtigt aktiviert wird.
io.output(leftmotor_in1_pin, False)
io.output(leftmotor_in2_pin, False)
io.output(rightmotor_in1_pin, False)
io.output(rightmotor_in2_pin, False)

# Die Funktion setMotorMode(motor, mode) legt die Drehrichtung der
# Motoren fest. Die Funktion verfügt über zwei Eingabevariablen.
# motor     -> Diese Variable legt fest, ob der linke oder der rechte
#              Motor ausgewaehlt wird.
# mode      -> Diese Variable legt fest, welcher Modus gewaehlt ist.
# Beispiel:
# setMotorMode(leftmotor, forward)  Der linke Motor ist gewaehlt
#                                   und dreht vorwaerts.
# setMotorMode(rightmotor, reverse) Der rechte Motor wird ausgewaehlt
#                                   und wird rueckwaerts gedreht.
# setMotorMode(rightmotor, stopp)   Der rechte Motor wird ausgewaehlt,
#                                   und wird gestoppt.

def setMotorMode(motor, mode):
   if motor == "leftmotor":
      if mode == "reverse":
         io.output(leftmotor_in1_pin, True)
         io.output(leftmotor_in2_pin, False)
      elif  mode == "forward":
         io.output(leftmotor_in1_pin, False)
         io.output(leftmotor_in2_pin, True)
      else:
         io.output(leftmotor_in1_pin, False)
         io.output(leftmotor_in2_pin, False)
   elif motor == "rightmotor":
      if mode == "reverse":
         io.output(rightmotor_in1_pin, False)
         io.output(rightmotor_in2_pin, True)
      elif  mode == "forward":
         io.output(rightmotor_in1_pin, True)
```

```
            io.output(rightmotor_in2_pin, False)
        else:
            io.output(rightmotor_in1_pin, False)
            io.output(rightmotor_in2_pin, False)
    else:
        io.output(leftmotor_in1_pin, False)
        io.output(leftmotor_in2_pin, False)
        io.output(rightmotor_in1_pin, False)
        io.output(rightmotor_in2_pin, False)

# Die Funktion setMotorLeft(power) setzt die Geschwindigkeit der
# linken Motoren. Die Geschwindigkeit wird als Wert zwischen -1
# und 1 uebergeben. Bei einem negativen Wert sollen sich die Motoren
# rueckwaerts drehen, ansonsten vorwaerts.
# Anschliessend werden aus den uebergebenen Werten die notwendigen
# %-Werte fuer das PWM-Signal berechnet.

# Beispiel:
# Die Geschwindigkeit kann mit +1 (max) und -1 (min) gesetzt werden.
# Das Beispiel erklaert, wie die Geschwindigkeit berechnet wird.
# SetMotorLeft(0)     -> Der linke Motor dreht mit 0 %, er ist gestoppt.
# SetMotorLeft(0.75)  -> Der linke Motor dreht mit 75 % vorwaerts.
# SetMotorLeft(-0.5)  -> Der linke Motor dreht mit 50 % rueckwaerts.
# SetMotorLeft(1)     -> Der linke Motor dreht mit 100 % vorwaerts.
def setMotorLeft(power):
    int(power)
    if power < 0:
        # Rueckwaertsmodus fuer den linken Motor
        setMotorMode("leftmotor", "reverse")
        pwm = -int(duty_cycle * power)
        if pwm > duty_cycle:
            pwm = duty_cycle
    elif power > 0:
        # Vorwaertsmodus fuer den linken Motor
        setMotorMode("leftmotor", "forward")
        pwm = int(duty_cycle * power)
        if pwm > duty_cycle:
            pwm = duty_cycle
    else:
        # Stoppmodus fuer den linken Motor
```



```
        setMotorMode("leftmotor", "stopp")
        pwm = 0
    PCA9685_pwm.set_pwm(0, 0, pwm)
# Die Funktion setMotorRight(power) setzt die Geschwindigkeit der
# rechten Motoren. Die Geschwindigkeit wird als Wert zwischen -1
# und 1 uebergeben. Bei einem negativen Wert sollen sich die Motoren
# rueckwaerts drehen, ansonsten vorwaerts.
# Anschliessend werden aus den uebergebenen Werten die notwendigen
# %-Werte fuer das PWM-Signal berechnet.

# Beispiel:
# Die Geschwindigkeit kann mit +1 (max) und -1 (min) gesetzt werden.
# Das Beispiel erklaert, wie die Geschwindigkeit berechnet wird.
# setMotorRight(0)     -> Der linke Motor dreht mit 0%, er ist gestoppt.
# setMotorRight(0.75)  -> Der linke Motor dreht mit 75% vorwaerts.
# setMotorRight(-0.5)  -> Der linke Motor dreht mit 50% rueckwaerts.
# setMotorRight(1)     -> Der linke Motor dreht mit 100% vorwaerts.

def setMotorRight(power):
    int(power)
    if power < 0:
        # Rueckwaertsmodus fuer den rechten Motor
        setMotorMode("rightmotor", "reverse")
        pwm = -int(duty_cycle * power)
        if pwm > duty_cycle:
            pwm = duty_cycle
    elif power > 0:
        # Vorwaertsmodus fuer den rechten Motor
        setMotorMode("rightmotor", "forward")
        pwm = int(duty_cycle * power)
        if pwm > duty_cycle:
            pwm = duty_cycle
    else:
        # Stoppmodus fuer den rechten Motor
        setMotorMode("rightmotor", "stopp")
        pwm = 0
    PCA9685_pwm.set_pwm(1, 0, pwm)
# Die Funktion exit() setzt die Ausgaenge, die den Motortreiber
# steuern, auf False. So befindet sich der Motortreiber nach dem
# Aufruf der Funktion in einem gesicherten Zustand, und die Motoren
```

```
# sind gestoppt.
def exit():
   io.output(leftmotor_in1_pin, False)
   io.output(leftmotor_in2_pin, False)
   io.output(rightmotor_in1_pin, False)
   io.output(rightmotor_in2_pin, False)
   io.cleanup()

# Ende des Programmes
```

Listing 19.8 Das Python-Programm »L298NHBridgePCA9685.py« dient der Ansteuerung des Motortreibers über den Servocontroller.

Wie in Abschnitt 12.2.2, »Steuerungsprogramm für das Roboter-Auto in Python«, bereits erläutert, ist es auch mit dem neu erstellten Programm *L298NHBridgePCA9685.py* nicht möglich, das Roboter-Auto über die Tastatur direkt zu steuern. Hierfür wird wieder das Programm *robot-control.py* benötigt, das die Tastaturbefehle an das *L298NHBridge-PCA9685.py* -Programm weitergibt.

Sie müssen das Programm *L298NHBridgePCA9685.py* anstelle des Programms *L298NH-Bridge.py* im Programm *robot-control.py* einbinden.

Um diese Anpassung jetzt vorzunehmen, öffnen Sie das Programm *robot-control.py* und suchen nach der folgenden Zeile.

```
import L298NHBridge as HBridge
```

Kommentieren Sie diese Zeile mit einem vorgestellten # aus, da wir das Modul *L298NH-Bridge.py* nicht weiterverwenden werden.

Suchen Sie jetzt ein paar Zeilen tiefer nach der folgenden Zeile:

```
# import L298NHBridgePCA9685 as HBridge
```

Entfernen Sie in dieser Zeile das vorgestellte #, um das im vorherigen Abschnitt neu erstellte Modul *L298NHBridgePCA9685.py* einzubinden.

Nachdem Sie beide Python-Programme erstellt bzw. angepasst haben, können Sie das Hauptprogramm *robot-control.py* starten. Dazu wechseln Sie einfach in Ihrem aktiven Terminal-Fenster in den Ordner */home/pi/robot* des Raspberry Pi und rufen den folgenden Befehl auf:

```
python robot-control.py
```


Ab jetzt erfolgt die Steuerung der Geschwindigkeit der Motoren über den Servocontroller.

Denken Sie auch daran, dass Sie die Websteuerung *robot-control-web.py* Ihres Roboter-Autos ebenfalls anpassen müssen. Hier gehen Sie exakt so vor wie schon im Programm *robot-control.py*.

19.4.2 Steuerungsprogramm für zwei Servomotoren oder ein Pan-Tilt-Kit

Das folgende Programm ist für das in Abschnitt 17.1.5, »Raspberry-Pi-Kamera befestigen«, erwähnte Pan-Tilt-Kit geschrieben. Wenn Sie dieses Pan-Tilt-Kit nicht verbaut haben, dann schließen Sie für das Ausprobieren des Programms bitte zwei Servomotoren an. Den ersten Servomotor schließen an Kanal 2 und den zweiten Servomotor an Kanal 3 des Servocontrollers an.

Über die Tastatureingabe des Programms *pan-tilt-kit.py* können Sie die Kamera drehen und schwenken. Um die Kamera nach oben zu schwenken, drücken Sie I und K; um die Kamera nach links oder rechts zu drehen, drücken Sie die Tasten J und L.

Das Python-Programm *pan-tilt-kit.py* können Sie von der folgenden Webseite herunterladen und auf dem Raspberry Pi im Ordner */home/pi/robot* abspeichern:

https://custom-build-robots.com/roboter-auto-download

Möchten Sie das Programm eigenständig erstellen, so legen Sie im Editor Notepad++ die Datei *pan-tilt-kit.py* an, geben den Programmcode in die Datei ein und speichern diese im Ordner */home/pi/robot* ab.

Um das Programm zu starten, geben Sie den folgenden Befehl im Terminal-Fenster ein:

```
sudo python pan-tilt-kit.py
```

Nachfolgend der Programmcode des Programms *pan-tilt-kit.py*.

```
#!/usr/bin/env python
# coding: latin-1
# Autor:   Ingmar Stapel
# Datum:   20190810
# Version:   1.0
# Homepage:   https://custom-build-robots.com
# Dieses Programm ermoeglicht die Steuerung des Pan-Tilt-Kits.

# Es werden verschiedene Python-Klassen importiert, deren Funktionen
# im Programm benoetigt werden fuer die Programmverarbeitung.
import sys, tty, termios, os, readchar
```

```
# Importieren des PCA9685 Moduls fuer den Servocontroller.
import Adafruit_PCA9685

# Initialisierung des Servocontrollers
pwm = Adafruit_PCA9685.PCA9685()

# Mit einer Frequenz von 60 Hz arbeiten die Servomotoren sehr gut.
pwm.set_pwm_freq(60)

# Hier werden die initialen Werte fuer die beiden Servomotoren festgelegt.
servo1_pwm = 480
servo2_pwm = 350

# Hier werden die Min- und Max-Werte der beiden Servomotoren definiert.
servo1_min = 260
servo1_max = 640

servo2_min = 160
servo2_max = 640

# Die Variable intervall legt die Schrittweite fuer die Servomotoren fest.
intervall = 5

# Das Menue fuer den Anwender, wenn er das Programm ausfuehrt.
# Das Menue erklaert, mit welchen Tasten das Pan-Tilt-Kit gesteuert wird.
print("w/s: Kamera hoch / runter")
print("a/d: Kamera links / rechts")
print("x: Programm beenden")

# Die Funktion getch() nimmt die Tastatureingabe des Anwenders
# entgegen. Die gedrueckten Buchstaben werden eingelesen. Sie werden
# benoetigt, um die Servomotoren zu steuern.
def getch():
   ch = readchar.readchar()
   return ch

# Die Funktion printscreen() gibt immer das aktuelle Menue aus
# sowie die PWM-Werte der Servomotoren.
```



```
def printscreen():
   # Der Befehl os.system('clear') leert den Bildschirminhalt vor
   # jeder Aktualisierung der Anzeige. So bleibt das Menue stehen,
   # und die Bildschirmanzeige im Terminal-Fenster steht still.
   os.system('clear')
   print("w/s: Kamera hoch / runter")
   print("a/d: Kamera links / rechts")
   print("x:   Programm beenden")
   print("========== Geschwindigkeitsanzeige ==========")
   print("PWM Wert Servomotor 1:  ", servo1_pwm)
   print("PWM Wert Servomotor 2: ", servo2_pwm)

# Diese Endlosschleife wird erst dann beendet, wenn der Anwender
# die Taste X betaetigt. Solange das Programm laeuft, wird ueber diese
# Schleife die Eingabe der Tastatur eingelesen.
while True:
   # Mit dem Aufruf der Funktion getch() wird die Tastatureingabe
   # des Anwenders eingelesen. Die Funktion getch() liest den
   # gedrueckten Buchstaben ein und uebergibt diesen an die
   # Variable char. So kann mit der Variablen char weiter-
   # gearbeitet werden.
   char = getch()

   # Die Kamera schwenkt nach oben.
   if(char == "w"):
      if servo1_max >= servo1_pwm:
         servo1_pwm = servo1_pwm + intervall
         pwm.set_pwm(3, 0, servo1_pwm)
      printscreen()

   # Die Kamera schwenkt nach unten.
   if(char == "s"):
      if servo1_pwm >= servo1_min:
         servo1_pwm = servo1_pwm - intervall
         pwm.set_pwm(3, 0, servo1_pwm)
      printscreen()

   # Die Kamera schwenkt nach rechts.
   if(char == "d"):
```

```
            if servo2_pwm > servo2_min:
                servo2_pwm = servo2_pwm - intervall
                pwm.set_pwm(2, 0, servo2_pwm)
            printscreen()

        # Die Kamera schwenkt nach links.
        if(char == "a"):
            if servo2_max > servo2_pwm:
                servo2_pwm = servo2_pwm + intervall
                pwm.set_pwm(2, 0, servo2_pwm)
            printscreen()

        # Mit der Taste "x" wird die Endlosschleife beendet,
        # und das Programm wird ebenfalls beendet.
        if(char == "x"):
            print("Program Ended")
            break

        # Die Variable char wird pro Schleifendurchlauf geleert.
        # Das ist notwendig, um weitere Eingaben sauber zu übernehmen.
        char = ""
# Ende des Programmes
```

Listing 19.9 Das Python-Programm »pan-tilt-kit.py« zur Steuerung des Pan-Tilt-Kits mit zwei Servomotoren

Abbildung 19.9 zeigt die Ausgabe des Programms für die Steuerung des Pan-Tilt-Kits im Terminal-Fenster.

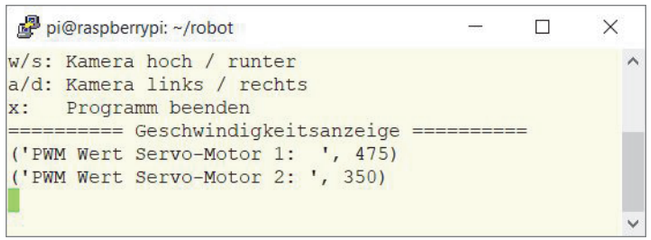


Abbildung 19.9 Programmausgabe der Pan-Tilt-Kit Servomotoren

Auf einen Blick

Inhalt

TEIL II Hände weg vom Steuer: Lassen Sie Ihr Roboter-Auto autonom fahren