

RaspiCar – Software für den Raspberry Pi

SLW 01-2026

1) Einleitung

Das RaspiCar ist ein kleines, batteriebetriebenes Fahrzeug, das über Motoren, einen LiDAR und eine Kamera verfügt. Das Fahrzeug wurde als Lern- und Entwicklungsumgebung für Themen im Bereich des autonomen Fahrens konzipiert. Das Fahrzeug wird mit einem Raspberry Pi gesteuert, der von einer Interface-Elektronik mit einem Raspberry Pi Pico Prozessor unterstützt wird. Der Raspberry Pi Pico kümmert sich um die Stromversorgung und die Steuerungen der Motoren.

Dieses Dokument beschreibt in Kürze die (minimal) notwendige Software auf dem Raspberry Pi, die für den Betrieb des RaspiCars notwendig ist.

2) Virtuelle Umgebung einrichten

Es wird empfohlen, die Software in einer virtuellen Umgebung aufzusetzen. Die Software sollte vollständig in einen separaten Ordner abgelegt werden, der normalerweise im Home-Verzeichnis des Users liegt. In diesem Dokument wird der Ordner RaspiCar genannt, kann aber auch anders heißen. Die Einrichtung erfolgt im Terminal:

Ordner einrichten und in den Ordner wechseln:

```
$ mkdir RaspiCar
$ cd RaspiCar
```

Dann wird die Umgebung mit `pip` eingerichtet. Die Software-Umgebung wird in dem Ordner `raspicar_venv` angelegt. Wichtig ist, dass die Pakete der System-Umgebung mitgenommen werden.

```
$ python -m venv raspicar_venv --system-site-packages
```

Anschließend wird die Umgebung aktiviert:

```
$ source raspicar_venv/bin/activate
```

Die aktive Umgebung wird in Klammern im Prompt angezeigt. Jetzt werden die notwendigen Module mit `pip` installiert:

```
(raspicar_venv)$ pip install gpiozero
(raspicar_venv)$ pip install pyserial
(raspicar_venv)$ pip install pandas
(raspicar_venv)$ pip install matplotlib
(raspicar_venv)$ pip install opencv-python
```

Damit sind die notwendigen Bibliotheken für die unten gezeigte Software eingerichtet. Bei Bedarf können natürlich noch andere Bibliotheken implementiert werden.

Um diese Umgebung in Thonny zu verwenden, wird dort unter ...

```
Tools → Options → Interpreter → Python executable
```

... die ausführbare Python-Datei in der neuen Umgebung ausgewählt. Sie liegt im Ordner:

```
raspicar_venv/bin/python3
```

Schließlich muss im Control Center → Interfaces (erreichbar aus dem Hauptmenü über Preferences) der Serial Port aktiviert werden.

Die Software-Module für das RaspiCar sind vollständig in Python programmiert. Alle Module enthalten ein kurzes Test-Script, das ausgeführt wird, wenn das Modul direkt als Hauptprogramm (z.B. in Thonny) gestartet wird. Zum Ausprobieren kann man jedes Modul explizit in Thonny öffnen und ausführen. Wenn die Module als Bibliothek in ein anderes Programm eingebunden werden, werden die Test-Script nicht ausgeführt.

3) Modul raspicar_ioctl.py

Dieses Modul ist der Treiber für das Interface Board mit dem Motor Controller. Es wird als Basis für alle weiteren Module benötigt und wird immer gebraucht.

Ein wesentlicher Teil von `ioctl` ist die ständige Überwachung der Batteriespannung. Wenn die Batteriespannung unter 10.5V fällt, wird ein Alarm gesetzt. Bei Spannungen unter 9.5V wird ein sofortiger Shutdown des Raspberry Pi ausgelöst, um die Batterie nicht zu beschädigen.

Weiterhin enthält das Modul Methoden, um mit dem Motor Controller zu kommunizieren, die zweifarbige LED und den LiDAR ein- bzw. auszuschalten. Und schließlich können Nachrichten auf das LCD-Display gesendet werden.

Abhängigkeiten

Das Modul benötigt `gpiozero` für die Steuerung der Ausgänge und `serial` für die serielle Schnittstelle.

Verwendung

Zuerst muss ein `io`-Objekt angelegt werden:

```
import raspicar_ioctl
io = raspicar_ioctl.IoCtrl()
```

Das `io`-Objekt verfügt über die folgenden Methoden:

<code>set_lidar_pwr(<pwr: bool>)</code>	Schaltet die Stromversorgung für den LiDAR ein oder aus.
<code>set_led_red(<status: bool>)</code>	Schaltet die rote LED an oder aus
<code>set_led_green(<status: bool>)</code>	Schaltet die grüne LED an oder aus
<code>send_ser(<msg: str>) → str</code>	Sendet einen Command-String an das Interface-Board
<code>send_msg(<msg: str>)</code>	Zeigt einen Text auf den Display-Zeilen 2 und 3 (max. 40 Zeichen)
<code>send_titel(<msg: str>)</code>	Zeigt einen Text auf der ersten Display-Zeile (max. 20 Zeichen)
<code>clear_display()</code>	Löscht den aktuellen Display-Inhalt
<code>get_status() → str</code>	Gibt den aktuellen Batterie-Status zurück
<code>close()</code>	Beendet die serielle Kommunikation und gibt die Ressourcen frei

Beispiel-Programm

Der Batteriestatus wird ausgelesen und ausgegeben. Der LiDAR wird für zwei Sekunden eingeschaltet.

```
1 import raspicar_ioctl
2 import time
3
4 io = raspicar_ioctl.IoCtrl()
5
6 print(io.get_status())
7
8 io.set_lidar_pwr(True)
9 time.sleep(2)
10 io.set_lidar_pwr(False)
```

4) Modul `raspicar_motors.py`

Dieses Modul dient der einfachen Ansteuerung der Motoren. Das Modul kommuniziert mit dem Motor-Treiber auf der Interface-Platine. Es können alle Bewegungen im zweidimensionalen Raum ausgeführt werden.

Abhängigkeiten

Das Modul benötigt `raspicar_ioctrl` für die Ansteuerung des Motor-Treibers auf der Interface-Platine.

Verwendung

Zuerst muss ein `mot`-Objekt angelegt werden. Dabei wird der Klasse `Motors` das vorher angelegte `io`-Objekt übergeben :

```
import raspicar_ioctrl
import raspicar_motors
io = raspicar_ioctrl.IoCtrl()
mot = raspicar_motors.Motors(io)
```

Das `mot`-Objekt ermöglicht jetzt die Steuerung der Motoren mit den folgenden Methoden:

`run(angle: int, speed: int)` Lässt die Motoren laufen. `speed` bestimmt die Geschwindigkeit und darf zwischen -100 und +100 liegen. Negative Werte bedeuten Rückwärtsfahrt. `angle` bestimmt die Drehrichtung. Der Wertebereich ist ebenfalls -100 bis +100. Negative Werte bewirken Drehungen nach links und positive Werte Drehungen nach rechts. Die Werte 0 bedeuten Stillstand. Das Fahrzeug kann sich auch auf der Stelle drehen, indem `angle` auf einen positive oder negativen Wert gesetzt wird, während `speed` auf 0 bleibt.

`stop()` Stoppt beide Motoren

Beispiel-Programm

Hier beschreibt das Fahrzeug eine Linkskurve.

```
1 import raspicar_ioctrl
2 import raspicar_motors
3 import time
4
5 # Create objects
6 io = raspicar_ioctrl.IoCtrl()
7 mot = raspicar_motors.Motors(io)
8
9 # Turning left ...
10 mot.run(-50, +30)
11 time.sleep(2)
12 mot.run(0, 0)
13
14 # Close everything
15 mot.stop()
16 io.close()
```

5) Modul `ylidar_x2.py`

Dieses Modul dient zur Verarbeitung der Daten des LiDARs. Es liest in einem eigenen Thread ständig den Datenstrom, der über die serielle Schnittstelle kommt. Daraus werden Sektoren berechnet. Für jeden Sektor wird der Abstand zu dem nächstliegenden Objekt ausgegeben.

Abhängigkeiten

Die Stromversorgung des LiDARs wird beim RaspiCar explizit ein- oder ausgeschaltet. Da der LiDAR einen relativ hohen Strombedarf hat (etwa 500 mA), kann er so bei Bedarf aktiviert werden. Die Schaltung geschieht bei RaspiCar über das Interface-Board (Pin 21). Am einfachsten geschieht die Steuerung über das oben beschriebene Modul `raspicar_ioctrl`.

Die Berechnung wird aus Performance-Gründen mit dem Modul `numpy` durchgeführt.

Für das Interface der seriellen Schnittstelle wird das Modul `serial` benötigt.

Verwendung

Zuerst wird ein `lid`-Objekt angelegt:

```
import raspicar_ioctrl
import ydlidar_x2
io = raspicar_ioctrl.IoCtrl()
lid = ydlidar_x2.YDLidarX2()
```

Der LiDAR wird durch Einschalten der Stromversorgung aktiviert. Anschließend wird die Verbindung zur seriellen Interface hergestellt und der Scan gestartet.

```
io.set_lidar_pwr(True)
lid.connect()
lid.start_scan()
time.sleep(0.25)
```

Nach einer kurzen Wartepause von etwa 0.25 Sekunden sind die Daten des LiDAR bereit zum Auslesen. Es gibt zwei Funktionen zum Auslesen:

`sectors = lid.get_sectors40() → np.ndarray` Zeigt die Werte in 9-Grad Schritten. So ergeben sich 40 Werte für den 360-Grad-Kreis. Die Werte beim Index 19 und 20 befinden sich links und rechts von der 0-Grad-Linie nach vorne.

`sectors = lid.get_sectors20() → np.ndarray` Zeigt die Werte in 18-Grad Schritten. So ergeben sich 20 Werte für den 360-Grad-Kreis. Die Werte beim Index 9 und 10 befinden sich links und rechts von der 0-Grad-Linie nach vorne.

Wenn der LiDAR nicht mehr benötigt wird,

sollte er abgeschaltet werden, um Batteriestrom zu sparen. Dazu dient die folgende Sequenz:

```
lid.stop_scan()
lid.disconnect()
io.set_lidar_pwr(False)
```

Der LiDAR verfügt über eine ganze Reihe von Hilfsfunktionen zum Verarbeiten und zur Anzeige der Daten. Eine vollständige Dokumentation des LiDAR-Moduls ist auf GitHub verfügbar.

Beispiel-Programm

Hier wird der LiDAR aktiviert, 50 mal ausgelesen und anschließend wieder abgeschaltet.

```
2 import ydlidar_x2
3 import time
4
5 # Create objects
6 io = raspicar_ioctrl.IoCtrl()
7 mot = raspicar_motors.Motors(io)
8 lid = ydlidar_x2.YDLidarX2()
9
10 # Activate the LiDAR
11 io.set_lidar_pwr(True)
12 lid.connect()
13 lid.start_scan()
14 time.sleep(0.25)
15 print("LiDAR started")
16
17 # Read LiDAR values
18 for idx in range(50):
19     sectors = lid.get_sectors20()
20     print(sectors[6:15])
21     time.sleep(0.25)
22
23 # Close everything
24 lid.stop_scan()
25 time.sleep(0.2)
26 lid.disconnect()
27 io.set_lidar_pwr(False)
28 io.close()
```

6) Modul raspicar_camera.py

Dieses einfache Modul erlaubt das Auslesen der Camera.

Abhängigkeiten

Das Modul benötigt OpenCV für den Raspberry Pi. Im Skript wird OpenCV als cv2 referenziert.

Verwendung

Zuerst muss ein Objekt cma angelegt werden. Die Bildgröße kann bei der Initialisierung angepasst werden. Die Default-Einstellung ist 800 x 600 Pixel. Andere Werte können z.B. 1280 x 800, etc.

```
import raspicar_camera
cam = raspicar_camera.Camera()
```

Anschließend können die Daten eines Kamerabildes mit `get_frame()` gelesen oder mit `show_frame()` in einem eigenen Fenster angezeigt werden.

`get_frame()` → `numpy.ndarray` Liest die aktuellen Daten der Kamera aus und stellt sie als Numpy-Array zur Verfügung. Die Bilder können dann von eigener Software (z.B. Objekterkennung) weiter verarbeitet werden.

`show_frame()` → `bool` Liest die aktuellen Daten der Kamera aus und zeigt sie in einem eigenen Fenster an. Anschließend wird die Tastatur eingelesen. Wenn die Tasten <q> oder <ESC> gedrückt werden, wird der boolsche Wert True zurück gegeben, andernfalls False. So lässt sich eine Bildschirm-Show mit der Tastatur beenden.

`close()` Beendet den Camera-Stream und gibt die Ressourcen frei.

Beispiel-Programm

Hier wird die Kamera gestartet und so lange in einem Fenster angezeigt, bis der Anwender die Show mit <q> oder <ESC> beendet

```
1 import raspicar_camera
2
3 # Create a cam object
4 cam = raspicar_camera.Camera()
5
6 # Run the display show ...
7 while True:
8     if cam.show_frame():
9         break
10
11 # Close everything
12 cam.close()
```

7) Modul `raspicar_socket.py`

Dieses Modul wird nur für die Kommunikation mit dem RaspiCar-Controller benötigt. Damit lässt sich das Fahrzeug fernsteuern.

Abhängigkeiten

Das Modul benötigt die Bibliotheken `socket`, `subprocess` und `os`, die standardmäßig auf dem Raspberry Pi implementiert sind.

Außerdem wird eine Datei `ip_addr.dat` benötigt, in der für die jeweiligen Netzwerke die IP-Adressen des Raspi-Controllers abgelegt sind. Das Modul geht davon aus, dass der Controller unter der dort eingetragenen IP-Adresse bereit ist und auf Anfragen antwortet. Wenn der Controller nicht gefunden werden kann, bricht das Programm ab.

Verwendung

Zuerst muss die Bibliothek importiert und ein Socket-Objekt `sock` angelegt werden.

```
import raspicar_socket
sock = raspicar_socket.RaspiCarSocket()
```

Das `sock`-Objekt verfügt über das Attribut `sock.okay`, das als Booleschen Wert anzeigt, ob das Objekt erfolgreich initialisiert wurde.

```
if not sock.okay:
    print("Socket initialization failed!")
```

Wenn die Verbindung steht, dann kann mit der Methode `get_data()` der Status des Joysticks und der Tasten abgefragt werden. `get_data()` liefert ein Tuppel mit vier Werte:

[0] ein boolescher Wert, der anzeigt, ob die Datenübertragung erfolgreich war

[1] Ein Integer-Wert, der den Status der 4 Buttons auf dem Controller anzeigt. Die vier Tasten belegen jeweils ein Bit: Blau → Bit 0, grün → Bit 1, gelb → Bit 2, rot → Bit 3

[2] Ein Integer-Wert, der die Vor/Rück-Position des Joysticks angibt. Der Mittelwert ist 0. Negative Werte kennzeichnen Rückwärts- bzw. positive Werte Vorwärts-Positionen. Der Wertebereich ist etwa -100 bis + 100.

[3] Ein Integer-Wert, der die Links/Rechts-Position des Joysticks angibt. Der Mittelwert ist 0. Negative Werte kennzeichnen Links und positive Werte Rechts-Positionen. Der Wertebereich ist etwa -100 bis + 100.

```
print(sock.get_data())
(True, 0, 45, 59)
```

Die Methode `send_msg()` erlaubt es, einen Text auf das Display des Controllers zu übertragen.

```
sock.send_msg("Hello from RaspiCar")
```

Wenn die Verbindung nicht mehr benötigt wird, sollte sie mit `close()` geschlossen werden.

Beispiel-Programm

Hier wird der Controller 20-mal abgefragt und das Ergebnis ausgegeben.

```
1 import time
2 import raspicar_socket
3
4 sock = raspicar_socket.RaspiCarSocket()
5 if not sock.okay:
6     print("Socket initiation failed!")
7 else:
8     sock.send_msg("Hello from RaspiCar")
9     for idx in range(20):
10         print(sock.get_data())
11         time.sleep(0.5)
12
13     sock.close()
14
```

8) Skript raspicar_terminal.py

Dieses ist ein einfaches Skript, das zum Austesten des Interface zwischen dem Raspberry Pi und dem Interface Board dient. Es wird nur zu Testzwecken benötigt.

Es nimmt Anweisungen über die Tastatur entgegen, die an den Motortreiber auf dem Interface Board gesendet werden. Nachfolgend werden die Antworten vom Interface Board ausgegeben. Eine Liste der verfügbaren Kommandos ist auf GitHub verfügbar: [RaspiCar-rp2040-motor_driver/CommandList.pdf](#).

Beispiel-Session

```
Shell x
>>> %Run raspicar_terminal.py
Connection: OK

Quit with 'quit'<return>
> mp1,1
OK

> mr200,300
OK

> i
RaspiCar Motor Driver (by SLW)
Software Version: 0.96

> quit
Serial interface closed ...

>>> |
```