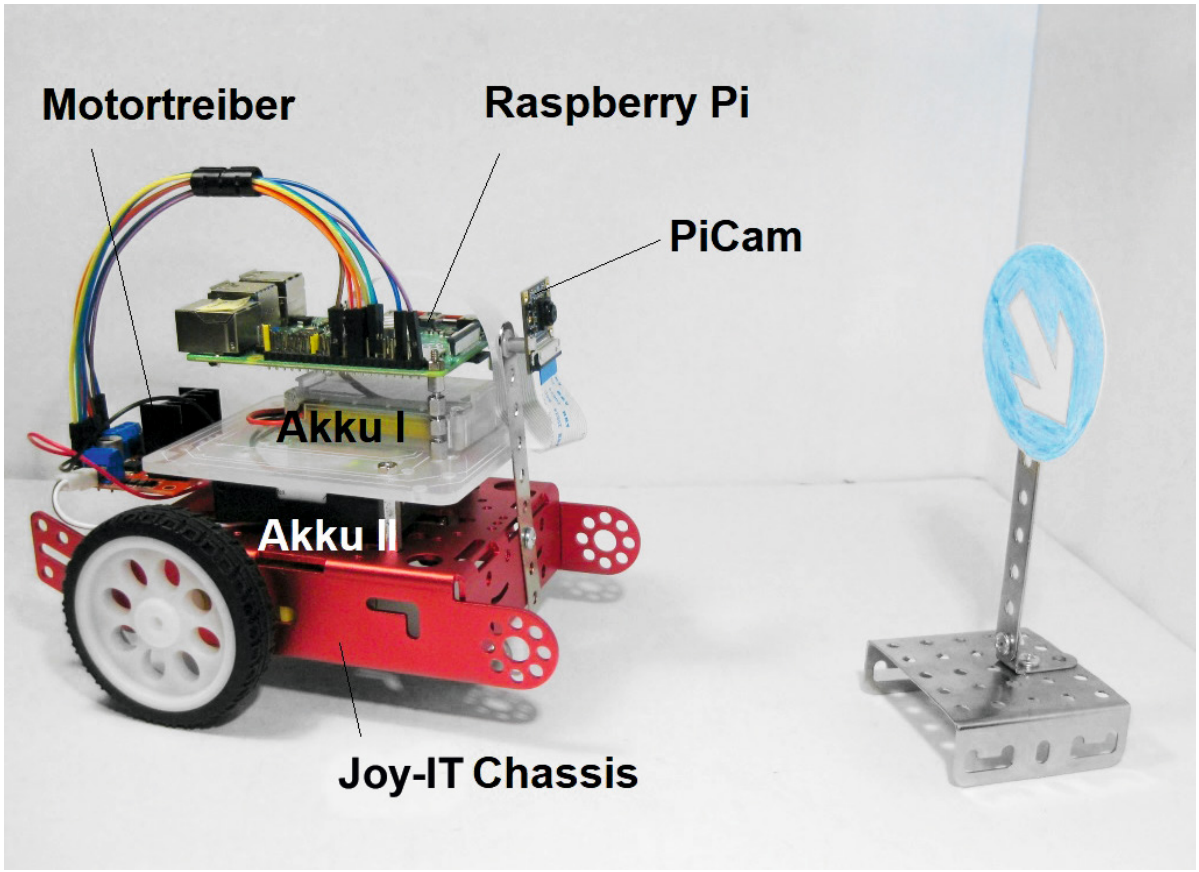
KI-Praxis VIII

Teil 8

Erste Schritte zu autonomen Systemen

Ein wichtiger Anwendungsbereich der Künstlichen Intelligenz ist der Einsatz sogenannter autonomer Systeme. Der Begriff wird meist mit selbstfahrenden Autos, humanoiden Robotern oder Drohnen verbunden. Die zugrunde liegende Technik entwickelt sich rasant und eröffnet unzählige neue Möglichkeiten.



Autonomie

Prinzipiell ist Autonomie die Fähigkeit, sich selbst unabhängig zu steuern und zu lenken. In der Technik ist insbesondere die Möglichkeit gemeint, unabhängig vom direkten Eingriff durch Menschen und unter zuvor nicht exakt eingeübten Bedingungen zu agieren. Damit wird auch der Unterschied zwischen automatisierten und autonomen Systemen klar. Ein automatisiertes Produktionssystem kann beispielsweise das Karosserieblech eines Autos unter immer gleichen Bedingungen korrekt montieren. Ein autonomes Robotersystem kann dagegen flexibel alle Aufgaben ausführen, für die es trainiert wurde. Auch wenn die Randbedingungen nicht immer exakt die gleichen sind, können auch unter variierenden Bedingungen und an unbekannten Orten Aktionen korrekt ausgeführt werden.

Dennoch existiert keine allgemeingültige Definition für autonome Systeme. So besteht in einigen Fällen die Zielsetzung darin, Menschen von gefährlichen oder körperlich extrem belastenden Aufgaben zu befreien. In anderen Fällen, wie etwa dem autonomen Fahren, geht es dagegen um die Verbesserung der Sicherheit oder die Entlastung von Routineaufgaben. Eins haben alle Szenarien jedoch gemeinsam: Der Nutzen autonomer Systeme ist durch die Datenmenge und die daraus extrahierbaren Informationen begrenzt.

Verkehrszeichenerkennung

Als Praxisbeispiel für ein autonomes System soll im Folgenden ein selbstfahrendes Robotersystem vorgestellt werden. Häufig werden für diese Projekte sogenannte Linienverfolger eingesetzt. Hierbei wird über Reflexionslichtschranken eine schwarze Linie auf dem Boden abgetastet. Dazu ist allerdings keine künstliche Intelligenz erforderlich. Diese Aufgabe kann problemlos mit den Verfahren der klassischen Regelungstechnik gelöst werden.

Anders liegen die Dinge, wenn das Fahrzeug in der Lage sein soll, Verkehrsschilder zu erkennen. Im realen Straßenverkehr ist diese Fähigkeit natürlich unabdingbar. Methoden zur maschinellen Erkennung von Verkehrszeichen gehören deshalb seit Längerem zu den zentralen Aufgaben bei der Entwicklung von autonomen Systemen bzw. Fahrzeugen. Über viele Jahre hinweg konnten hier kaum Fortschritte erzielt werden. Einige Forscher waren sogar der Ansicht, dass diese Aufgabe niemals mit ausreichender Zuverlässigkeit gelöst werden könne.

Ein zentrales Problem bei der maschinellen Verkehrszeichenerkennung ist, dass die verwendeten Symbole naturgemäß sehr anthropozentrisch, d. h. auf die Bedürfnisse der menschlichen Wahrnehmung ausgerichtet sind. Wie bereits in den letzten Beiträgen zu dieser Serie klar wurde, sind es aber genau die typisch menschlichen Fähigkeiten, die den Maschinen die größten Schwierigkeiten bereiten.

Ein Vorschlag zur Lösung des Problems war, alle Verkehrszeichen in doppelter Ausführung, also einmal in der bekannten Form und einmal in einer maschinell lesbaren Variante anzubringen. Die zweite Version könnte beispielsweise aus einem entsprechend optimierten Barcode oder einem QR-System (Quick Response) bestehen (Bild 1). Natürlich wäre die Lösung mit erheblichem Aufwand verbunden.

Glücklicherweise deutete sich mit dem zunehmenden Einsatz von Neuronalen Netzen eine wesentlich einfachere Lösung an. Nach ersten Erfolgen dieser Technologie wurde schnell klar, dass die Muster- bzw. Bilderkennung auch im Bereich des autonomen Fahrens eingesetzt werden könnte. Inzwischen ist das sichere Erkennen auch von klassischen Verkehrsschildern für moderne KI-Systeme kein Problem mehr.

Aktuell wird die Idee allerdings sogar wieder aufgegriffen. So tauchen an manchen Autobahnabschnitten sogenannte Landmarkenschilder auf (Bild 2). Auf diesen Strecken betreibt das Bundesministerium für Verkehr und digitale Infrastruktur ein Pilotprojekt namens „Digitales Testfeld Autobahn“. Entsprechend ausgerüsteten Fahrzeugen ermöglichen die schwarz-weißen Verkehrsschilder eine exakte Standortbestimmung und unterstützen bei der korrekten Erfassung der Umgebung. Die neuen Zeichen haben bewusst keinerlei Ähnlichkeit mit klassischen Verkehrsschildern und können von Fahrern nicht autonomer Autos vollständig ignoriert werden.

Aufgrund der hohen praktischen Relevanz wurde die Klassifizierung von Verkehrsschildern zu einem der wichtigsten Forschungsgebiete der KI. In der Automobilindustrie spielen Maschinelles Lernen und Deep Learning daher inzwischen eine zentrale Rolle bei der Entwicklung von Fahrassistenzsystemen. KI-Methoden haben daher schon vor Jahren Einzug in die Entwicklungszentren der Kfz-Industrie gehalten.

Um einen Einblick in diese hochinteressante Technologie zu gewinnen, soll hier ein Roboterfahrzeug entwickelt werden, das in der Lage ist, eigenständig einige Verkehrszeichen zu erkennen und entsprechend darauf zu reagieren.

Autonomes Roboterfahrzeug mit Verkehrszeichenerkennung

Das zentrale Thema dieses Praxisprojekts ist die Klassifizierung und Erkennung verschiedener Verkehrsschilder. Wie auch in den letzten Beiträgen (s. Artikelübersicht zur Serie KI-Praxis) soll wieder ein Raspberry Pi als Steuerrechner zum Einsatz kommen. Als Basis für das Roboterfahrzeug wird ein Metall-Chassis mit zweirädrigem Antrieb verwendet.

Bild 1: Verkehrsschilder mit QR-Code?

Bild 2: Landmarkenschild

Diese Hardwarekomponenten sind erforderlich:

- Raspberry Pi 4, idealerweise mit 8 GB RAM
- PiCam
- Roboter-Chassis mit Getriebemotoren
- Motortreiber
- Stromversorgung

Weitere Details dazu finden sich in der Materialliste am Ende dieses Beitrags.

Artikelübersicht zur Serie KI-Praxis

KI-Praxis I
Einstieg in die Künstliche Intelligenz
Artikel-Nr. 252090
KI-Praxis II
Neuronale Netze - Aufbau und Training
Artikel-Nr. 252174
KI-Praxis III
Handschrifterkennung
Artikel-Nr. 252233
KI-Praxis IV
Spracherkennung und Sprachsynthese
Artikel-Nr. 252343
KI-Praxis V
Objekterkennung mit einer See-and-Talk-Box
Artikel-Nr. 252461
KI-Praxis VI
Gesichtserkennung und -identifizierung
Artikel-Nr. 252589
KI-Praxis VII
Trainieren selbst erstellter KI-Modelle
Artikel-Nr. 252711

Um dieses Projekt erfolgreich abzuschließen, sind grundlegende Kenntnisse zum Aufbau Neuronaler Netze und zu den gängigen Bildverarbeitungstechniken erforderlich. Diese können bei Bedarf in den Beiträgen zu dieser Serie nachgelesen werden.

Wie bereits in den letzten Beiträgen klar wurde, ist für das Training eines Neuronalen Netzes immer ein möglichst umfangreicher Datensatz erforderlich. Für das hier vorliegende Problem bietet sich der Datensatz „German Traffic Sign Recognition Benchmark" (GTSRB) an, der 43 verschiedene Klassen von Verkehrszeichen der deutschen Straßenverkehrsordnung enthält. Der Datensatz kann kostenlos unter [1] oder [2] geladen werden.

Für das Training ist der Datensatz „GTSRB_Final_Training_Images.zip" erforderlich. Dieser enthält 43 Kategorien mit jeweils mehreren Hundert Bildern von Verkehrszeichen im ppm-Format.

Das zip-Archiv enthält im Unterordner „Images" die in 43 Kategorien eingeteilten Bilder (s. Bild 3). Die mit 00000 bis 00042 bezeichneten Ordner enthalten die jeweiligen Abbildungen. Die Zahlen entsprechen dabei den folgenden Verkehrszeichen (s. road_signs_names.csv):

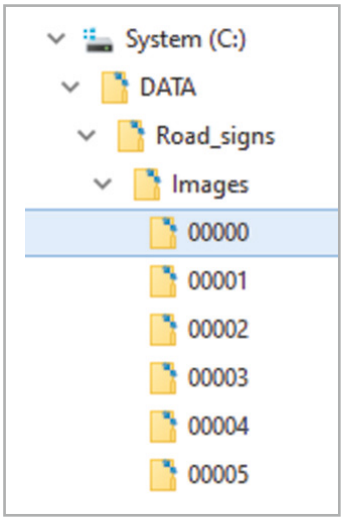


Bild 3: Datenstruktur des GTSRB-Satzes

Id	SignNameD	SignNameE
0	Zulässige Höchstgeschwindigkeit (20 km/h)	Speed limit (20 km/h)
1	Zulässige Höchstgeschwindigkeit (30 km/h)	Speed limit (30 km/h)
2	Zulässige Höchstgeschwindigkeit (50 km/h)	Speed limit (50 km/h)
3	Zulässige Höchstgeschwindigkeit (60 km/h)	Speed limit (60 km/h)
4	Zulässige Höchstgeschwindigkeit (70 km/h)	Speed limit (70 km/h)
5	Zulässige Höchstgeschwindigkeit (80 km/h)	Speed limit (80 km/h)
6	Ende der Höchstgeschwindigkeit (80 km/h)	End of speed limit (80 km/h)
7	Zulässige Höchstgeschwindigkeit (100 km/h)	Speed limit (100 km/h)
8	Zulässige Höchstgeschwindigkeit (120 km/h)	Speed limit (120 km/h)
9	Überholverbot	No passing
10	Überholverbot für Kraftfahrzeuge über 3,5 t	No passing for vehicles over 3.5 metric tons
11	Verbot für Fahrzeuge aller Art	Right-of-way at the next intersection
12	Vorfahrtstraße	Priority road
13	Vorfahrt gewähren	Yield
14	Stop	Stop
15	Verbot für Fahrzeuge aller Art	No vehicles
16	Verbot für Kraftfahrzeuge	Vehicles over 3.5 metric tons prohibited
17	Verbot der Einfahrt	No entry
18	Gefahrzeichen	General caution
19	Kurve (links)	Dangerous curve to the left
20	Kurve (rechts)	Dangerous curve to the right
21	Doppelkurve	Double curve
22	Unebene Fahrbahn	Bumpy road
23	Schleudergefahr	Slippery road
24	Einseitig (rechts) verengte Fahrbahn	Road narrows on the right
25	Baustelle	Road work
26	Lichtzeichenanlage	Traffic signals
27	Fußgänger	Pedestrians
28	Kinder	Children crossing
29	Radverkehr	Bicycles crossing
30	Schnee- oder Eisglätte	Beware of ice/snow
31	Wildwechsel	Wild animals crossing
32	Ende aller Streckenverbote	End of all speed and passing limits
33	Vorgeschriebene Fahrtrichtung (rechts)	Go right
34	Vorgeschriebene Fahrtrichtung (links)	Go left
35	Fahrtrichtung nur geradeaus	Ahead only
36	Fahrtrichtung nur geradeaus und rechts	Go straight or right
37	Fahrtrichtung nur geradeaus und links	Go straight or left
38	Vorbeifahrt rechts	Keep right
39	Vorbeifahrt links	Keep left
40	Kreisverkehr	Roundabout mandatory
41	Ende Überholverbot	End of no passing
42	Ende des Überholverbotes für Lkw	End of no passing by vehicles over 3.5 metric tons

Nach dem Download sollten die Daten in ein eigenes Verzeichnis z. B.

c:\DATA\Road_signs\Images

kopiert werden.

Danach kann man sich mit einem kurzen Jupyter-Notebook-Skript (Roadsign_plot_1V1.ipynb, s. Downloadpaket [3]) einen ersten anschaulichen Überblick über die Daten verschaffen (Bild 4). Man erkennt, dass die Bilder von sehr unterschiedlicher Qualität sind. Während einige, wie beispielsweise in Kategorie 14 („Stop"), sehr gut zu erkennen sind, sind andere auch für den menschlichen Betrachter kaum zu identifizieren (z. B. 40 „Kreisverkehr").

Fahrtraining mit Jupyter

Nachdem der Datensatz in maschinenlesbarer Form bereitsteht, kann man anfangen ein Neuronales Netz aufzubauen, das in der Lage ist, Verkehrszeichen zu erkennen. „Erkennen" bedeutet in diesem Zusammenhang, dass die folgende Sequenz ausgeführt wird:

- Erfassen des Zeichens mit einer Kamera
- Einspeisen des aufgenommenen Bildes in ein entsprechend trainiertes Neuronales Netz
- Kategorisierung in eine der 43 Zeichenklassen

Die Arbeit mit Jupyter wurde in den letzten Beiträgen zum Thema Künstliche Intelligenz ausführlich dargelegt. Bei Bedarf können Details in dieser Artikelserie nachgelesen werden. Da das Notebook bereits recht umfangreich ist, sollen hier nur die wesentlichen Schritte dargelegt werden. Zunächst werden in der ersten Zelle wie üblich alle erforderlichen Libraries geladen. Dann folgt die Festlegung der wichtigsten Parameter:

```
noClasses =  2
epoch     = 12
steps     = 32
imgSize   = 32
batchSize =  4
opt       = 'adam'
ModelName = 'road_signs_model_1_1_2classes.h5'
```

Die Anzahl der Klassen legt fest, wie viele verschiedene Verkehrszeichen unterschieden werden sollen. Maximal können hier alle 43 Zeichen verwendet werden. Im einfachsten Fall kann man das Training auf zwei Klassen reduzieren. Für die Praxisanwendung genügen zunächst die Klassen 38 „Vorbeifahrt rechts" und 39 „Vorbeifahrt links".

Dann werden die im letzten Abschnitt bereitgestellten Bilddaten geladen. Es folgt die Definition des Neuronalen Netzes, danach kann bereits mit dessen Training begonnen werden.

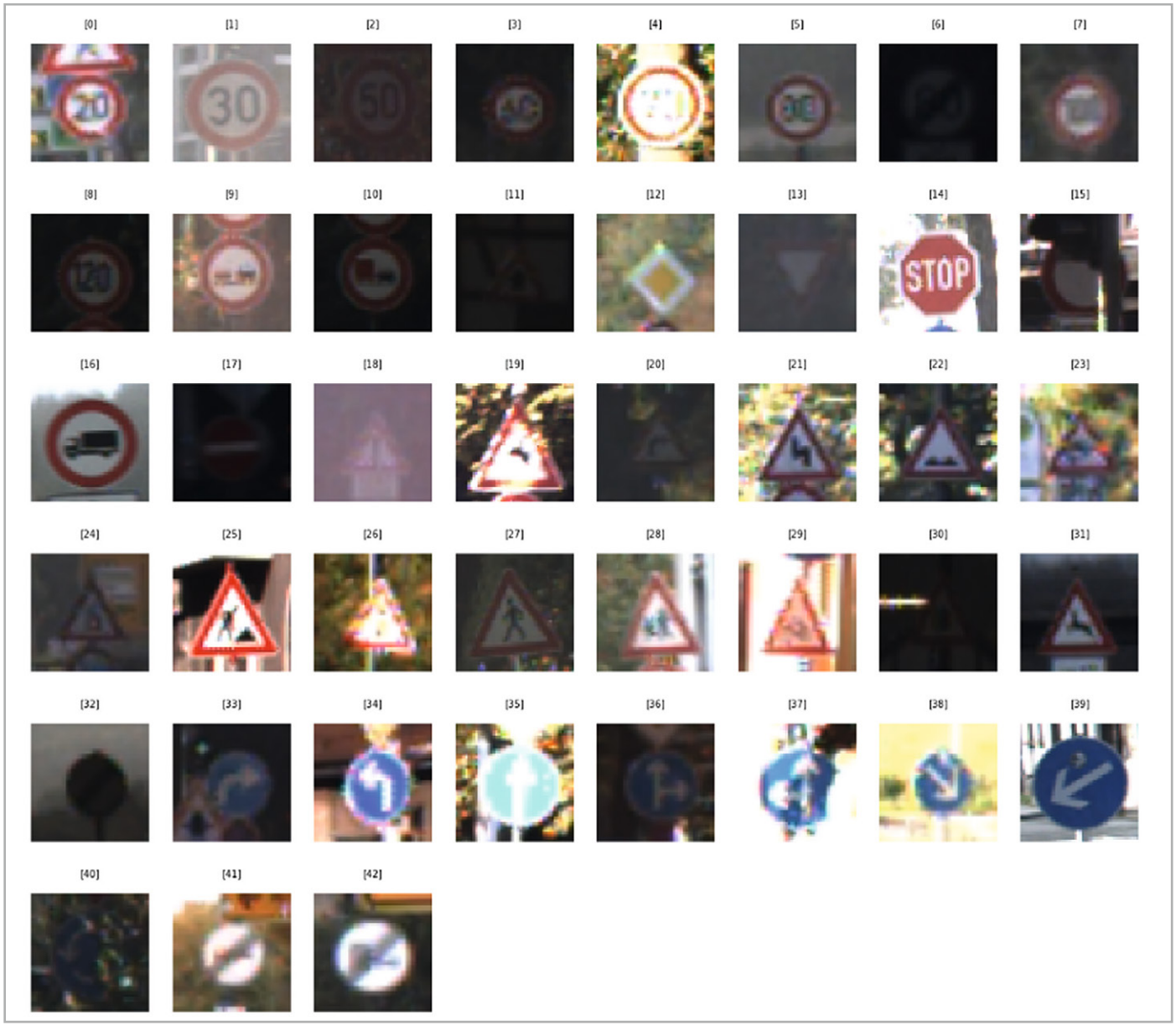


Bild 4: Auszug aus dem GTSRB-Datensatz

Bild 5: Lernkurve zum Verkehrszeichentraining

Nach Abschluss des Trainings wird das erstellte Modell unter dem in der zweiten Zelle angegebenen Namen (z. B. road_signs_model_1_1_2classes.h5) abgespeichert.

Die nächste Zelle liefert die Trainingsergebnisse. Unter anderem wird hier wieder eine Lernkurve ausgegeben. Bild 5 zeigt ein typisches Ergebnis. Man erkennt, dass nach etwa zehn Epochen kein weiterer Lernfortschritt mehr erzielt werden kann. Die letzte Zelle liefert schließlich noch die beispielhafte Klassifizierung einiger Bilder (Bild 6).

Hardware-Basis

Die eigentliche KI-Aufgabe wäre mit der erfolgreichen Erkennung und Kategorisierung der Verkehrszeichen bereits abgeschlossen. Aber natürlich ist es wesentlich reizvoller, die Methoden auch in der Praxis anzuwenden. Auf diese Weise können die Grundlagen des autonomen Fahrens auch in Schulen, Makerspaces oder in FabLabs eindrucksvoll präsentiert werden.

Mithilfe eines Roboter-Chassis (Bild 7, s. Materialliste) kann ein solches Projekt leicht umgesetzt

Bild 6: Klassifizierung von Beispielbildern im Jupyter-Notebook

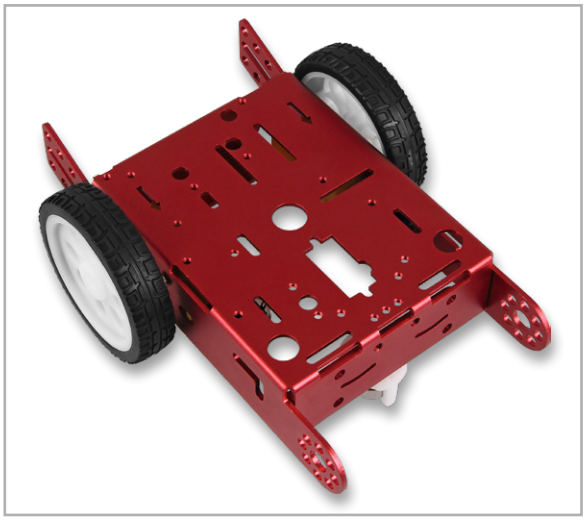


Bild 7: Roboter-Chassis „Joy-IT Robot Car Kit“

werden. Das Kit enthält neben einem stabilen Metall-Chassis auch zwei passende Motoren, die zugehörigen Räder sowie ein Bugrad in Form einer Metallkugel mit Halterung. Diese Fahrzeugversion kommt häufig in Roboterprojekten zum Einsatz, da sie den Vorteil einer besonders einfachen Fahrzeugsteuerung aufweist. Richtungsänderungen des Fahrzeugs können einfach über unterschiedliche Motordrehzahlen erfolgen. Auf einen aufwendigen Lenkmechanismus kann also verzichtet werden. Die Ansteuerung der Motoren erfolgt über ein Motortreibermodul (z. B. Joy-IT Motormodul mit L298N, s. Materialliste). Bild 8 zeigt den Schaltplan dazu.

Ein Aufbaubeispiel zum fertigen Roboter ist im Titelbild zu sehen. Zur Kontrolle sind die Steuerungsverbindungen zwischen dem Raspberry Pi und dem Motortreiber in der folgenden Tabelle nochmals aufgeführt.

Motortreiber-Pin	in1	in2	enA	in3	in4	enB
Raspberry-Pi-Pin	24	23	25	17	18	27

Intelligenz braucht Power

Obwohl in dieser Artikelserie die Methoden der KI und ihre praktischen Anwendungen im Vordergrund stehen, soll hier in aller Kürze auf ein eher basistechnisches Problem eingegangen werden. Für die effiziente Anwendung der in den letzten Abschnitten entwickelten Modelle ist mindestens ein Raspberry Pi 3 erforderlich. Allerdings erreicht dieser zusammen mit Kamera, Bildverarbeitung und Motorsteuerung rasch das Ende seiner Leistungsfähigkeit. Ein Pi 4 kommt mit den hier gestellten Aufgaben dagegen problemlos klar. Allerdings ist der Pi 4 – wenig überraschend – auch die Variante mit der höchsten Leistungsaufnahme. Die Tabelle 1 liefert einen Überblick dazu.

Andererseits ist bekannt, dass der Raspberry Pi hinsichtlich der Stromversorgung etwas kritisch ist. So führen bereits Spannungseinbrüche von etwas unter 5 V zu unerwünschten Reboot-Sequenzen. In einigen Fällen wurden sogar SD-Karten im Pi beschädigt, als dieser mit einer unzureichenden Stromquelle betrieben wurde. Es ist daher immer eine gute Idee, die Spannungsversorgung des Pi im Auge zu behalten. Für das Roboterfahrzeug wurden aus diesem Grund zwei getrennte Stromversorgungen verwendet:

1. eine hochwertige Powerbank mit geregeltem 5-V-Ausgang für den USB-C-Anschluss des Raspberry Pi (Akku I)
2. ein Satz Batterien oder Akkus für die Motorversorgung (Akku II)

Die Motorversorgung kann dabei mit bis zu 12 V erfolgen. Allerdings zeigt sich, dass auch 6 V (z. B. 4x 1,5 V/AA) für ein zügiges Vorankommen des Roboterfahrzeugs vollkommen ausreichen.

Robotersteuerung, Test und Justierung

Die Steuerung des Roboters kann über ein Python-Programm erfolgen (Autonomous_robot_TEST.py). Einsatz und Anwendung dieser Programmiersprache wurden in früheren Beiträgen ausführlich erläutert.

Tabelle 1

Leistungsaufnahme Raspberry Pi 3 im Vergleich zum Raspberry Pi 4

Pi Version	Betriebszustand	Stromaufnahme	Leistungsaufnahme
Raspberry Pi 4 B	Leerlauf	600 mA	3 W
	Maximale CPU-Auslastung	1400 mA	7 W
Raspberry Pi 3 B+	Leerlauf	350 mA	2 W
	Maximale CPU-Auslastung	1000 mA	5 W

Für einen ersten Hardware-Test ist der folgende Code bestens geeignet:

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

in1 = 24
in2 = 23
enA = 25

in3 = 17
in4 = 18
enB = 27


GPIO.setmode(GPIO.BCM)

GPIO.setup(in1,GPIO.OUT);GPIO.setup(in2,GPIO.OUT)
GPIO.setup(enA,GPIO.OUT)
GPIO.output(in1,GPIO.LOW);GPIO.output(in2,GPIO.LOW)

GPIO.setup(in3,GPIO.OUT);GPIO.setup(in4,GPIO.OUT)
GPIO.setup(enB,GPIO.OUT)
GPIO.output(in3,GPIO.LOW);GPIO.output(in4,GPIO.LOW)
```

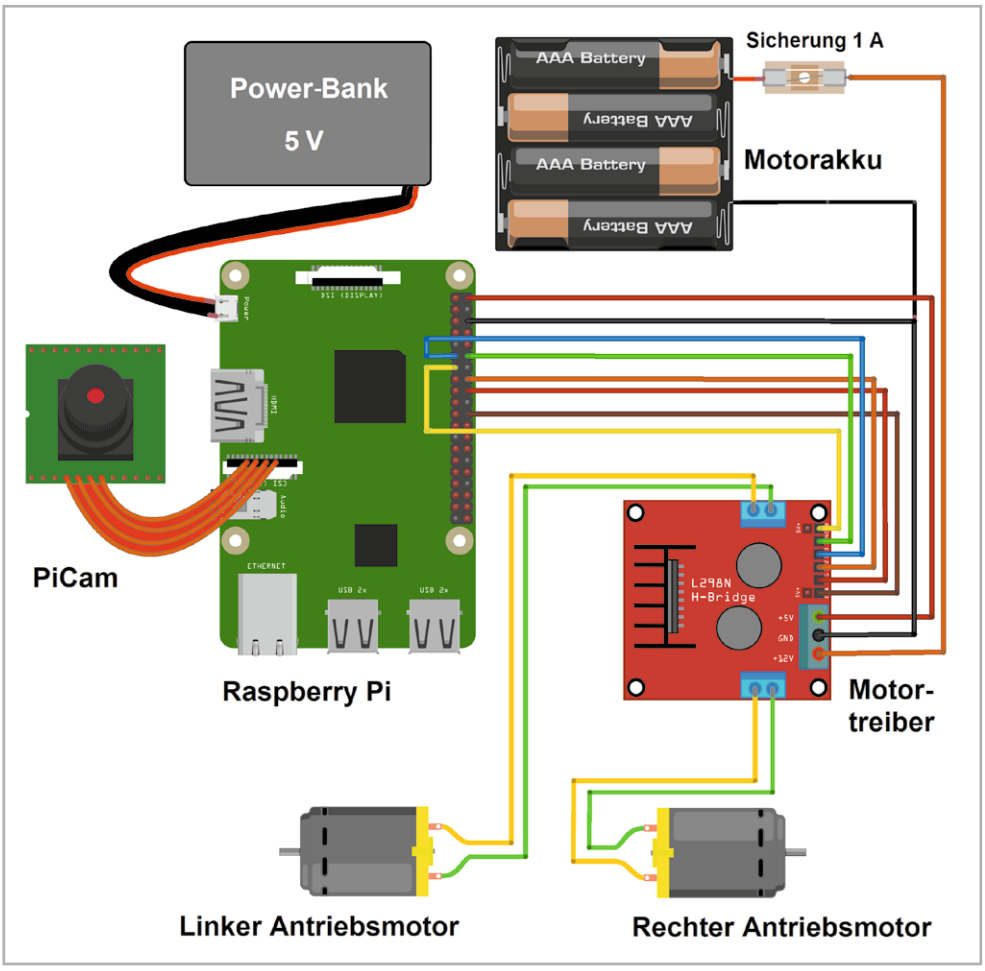


Bild 8: Schaltplan zum Roboterfahrzeug

```
p1=GPIO.PWM(enA,1000);p2=GPIO.PWM(enB,1000)
p1.start(25)
p2.start(27)
p1.ChangeDutyCycle(30);p2.ChangeDutyCycle(30)

while(1):
    x=input()
    print(x)
    if x=='s':
        print("stop")
        GPIO.output(in1,GPIO.LOW);GPIO.output(in2,GPIO.LOW)
        GPIO.output(in3,GPIO.LOW);GPIO.output(in4,GPIO.LOW)
    elif x=='f':
        print("forward")
        GPIO.output(in1,GPIO.HIGH);GPIO.output(in2,GPIO.LOW)
        GPIO.output(in3,GPIO.HIGH);GPIO.output(in4,GPIO.LOW)
    elif x=='b':
        print("backward")
        GPIO.output(in1,GPIO.LOW);GPIO.output(in2,GPIO.HIGH)
        GPIO.output(in3,GPIO.LOW);GPIO.output(in4,GPIO.HIGH)
    elif x=='tl':
        print("turning left")
        GPIO.output(in1,GPIO.LOW);GPIO.output(in2,GPIO.HIGH)
        GPIO.output(in3,GPIO.HIGH);GPIO.output(in4,GPIO.LOW)
        sleep(1)
        GPIO.output(in1,GPIO.LOW);GPIO.output(in2,GPIO.LOW)
        GPIO.output(in3,GPIO.LOW);GPIO.output(in4,GPIO.LOW)
    elif x=='tr':
        print("turning right")
        GPIO.output(in1,GPIO.HIGH);GPIO.output(in2,GPIO.LOW)
        GPIO.output(in3,GPIO.LOW);GPIO.output(in4,GPIO.HIGH)
        sleep(1)
        GPIO.output(in1,GPIO.LOW);GPIO.output(in2,GPIO.LOW)
        GPIO.output(in3,GPIO.LOW);GPIO.output(in4,GPIO.LOW)
    elif x=='e':
        GPIO.cleanup();print("bye...");break

    else:
        print("<<<  wrong data  >>>")
        print("please enter the defined data to continue.....")
```

Im Programm werden nach der Zuweisung der Pins die verwendeten Ports als Ausgänge definiert. In der Hauptschleife kann dann eine Anweisung eingegeben werden. Folgende Befehle sind zugelassen:

- s: Stop
- f: Vorwärts (forward)
- b: Zurück (backward)
- tl: Drehung nach links (turn left)
- tr: Drehung nach rechts (turn right)
- e: Programm beenden (exit)

Damit kann die Funktion der Roboterhardware ausführlich getestet werden. Erst wenn der Roboter alle Anweisungen korrekt ausführt, ist er bereit für den Einsatz der KI-Programme.

Falls die Anweisungen nicht richtig ausgeführt werden, kann man das Programm auch zur Fehlersuche nutzen. Bewegt sich der Roboter beispielsweise auf die Anweisung „f" hin rückwärts, müssen die Motoranschlüsse umgepolt werden. Werden die Drehungen in die falsche Richtung ausgeführt, deutet dies ebenfalls auf eine fehlerhafte Polung eines Motors hin.

Zudem kann man den Drehwinkel des Roboters justieren. Über die Sleep-Anweisung in den Drehbefehlen kann der Winkel z. B auf exakt 90° eingestellt werden. Ist die Drehung zu gering, muss das Sleep-Intervall vergrößert werden. Bei zu großen Drehwinkeln ist es entsprechend zu reduzieren. Über die Anweisungen

```
p1.ChangeDutyCycle(30);p2.ChangeDutyCycle(30)
```

ist die Motorgeschwindigkeit der entsprechenden Motoren einstellbar. Hier können auch Differenzen in den individuellen Motordrehzahlen korrigiert werden. Bei optimaler Einstellung sollte der Roboter auf den Befehl „f" hin exakt geradeaus fahren.

Wenn die Hardware-Basis des Roboterfahrzeugs fertiggestellt, justiert und geprüft ist, kann man im nächsten Schritt die Erkennung von Verkehrszeichen und eine entsprechende Reaktion des Fahrzeugs in Angriff nehmen.

Autonomes Fahren

Sind die Vorarbeiten abgeschlossen, kann das eigentliche Steuerprogramm für den Roboter (RobotMove_2V0.py) geladen werden. Über die Anweisung

```
model = tf.keras.models.load_model ↩
(, road_signs_model_1_0.h5')
```

wird das im Abschnitt „Fahrtraining mit Jupyter" erstellte Modell geladen. In der Hauptschleife wird dieses über

```
result=model.predict(img)
n=np.argmax(result)
```

ausgewertet. Anhand der Kennzahl n erfolgt die Steuerung des Fahrzeugs:

```
if (n!=0 and n!=1):
 print("forward")
 GPIO.output(in1,GPIO.HIGH);GPIO.output(in2,GPIO.LOW)
 GPIO.output(in3,GPIO.HIGH);GPIO.output(in4,GPIO.LOW)
 elif n==1:
 print("turning left")
 GPIO.output(in1,GPIO.LOW);GPIO.output(in2,GPIO.HIGH)
 GPIO.output(in3,GPIO.HIGH);GPIO.output(in4,GPIO.LOW)
 time.sleep(1)
 GPIO.output(in1,GPIO.LOW);GPIO.output(in2,GPIO.LOW)
 GPIO.output(in3,GPIO.LOW);GPIO.output(in4,GPIO.LOW)
 elif n==0:
 print("turning right")
 GPIO.output(in1,GPIO.HIGH);GPIO.output(in2,GPIO.LOW)
 GPIO.output(in3,GPIO.LOW);GPIO.output(in4,GPIO.HIGH)
 time.sleep(1)
        GPIO.output(in1,GPIO.LOW);GPIO.
output(in2,GPIO.LOW)
        GPIO.output(in3,GPIO.LOW);GPIO.
output(in4,GPIO.LOW)
```

Wird also ein „Vorbeifahrt links"-Zeichen erkannt (n = 1), wird eine Wende nach links eingeleitet. Entsprechend erfolgt bei „Vorbeifahrt rechts", also n = 0, die Wende nach rechts. Alle anderen Werte (n!=0 and n!=1) führen zu einer Geradeaus-Fahrt. Natürlich könnte man auch weitere Verkehrszeichen mit einbeziehen und das Fahrzeug beim Erkennen z. B. eines Stoppschilds anhalten oder vor einem Einbahnstraßenschild umkehren lassen usw.

Alternatives Training mit Lobe

Das Trainieren über ein selbst erstelltes Jupyter-Notebook ist vergleichsweise aufwendig. Wie in den letzten Abschnitten klar wurde, erfordert dieser Weg bereits tiefgehende Kenntnisse in Python und auch das Beherrschen der Jupyter-Umgebung. Zudem sind eine Vielzahl von Libraries erforderlich.

Diese verschiedenen Softwarekomponenten arbeiten nicht immer perfekt zusammen, da sie schnellen Update-Folgen unterworfen sind und deshalb immer wieder Inkompatibilitäten auftreten. Die Arbeit mit Libraries und Jupyter erfordert also bereits ein hohes Maß an Fachkenntnissen und Erfahrungen.

Natürlich bietet die Erstellung eines eigenen Trainingsprogramms die größte Flexibilität. Im professionellen Bereich kann man daher meist den entsprechenden Aufwand nicht vermeiden. Für einfachere Anwendungen in der Lehre oder im Hobbybereich gibt es jedoch eine Alternative. Das bereits im letzten Beitrag ausführlich vor-

gestellte Lobe-Trainingssystem kann auch hier erfolgreich eingesetzt werden.

Das Training kann hier wieder mit dem bereits bekannten GTSRB-Bildsatz durchgeführt werden. Alternativ können aber auch eigene Bilder zum Einsatz kommen. Die Verwendung eigener Bilder hat sich sogar als besonders effizient erwiesen. Idealerweise werden die Trainingsbilder bereits mit der PiCam aufgenommen und zeigen genau die Schilder, die später zum Einsatz kommen.

Das Training mit Lobe ist sehr einfach und das erforderliche Vorgehen ist weitgehend intuitiv. Bei Bedarf können Details dazu im letzten Beitrag nachgelesen werden.

Ist das Training abgeschlossen, wird das Modell als tfLite-Datei exportiert. Nach dem Übertragen der Datei auf den Raspberry Pi kann das Modell mit dem Python-Programm „classifier_live.py" getestet werden. Dabei wird wieder ein Livebild der PiCam angezeigt. Sobald ein Zeichen erkannt wurde, erscheint die zugehörige Bezeichnung links oben im Bild (Bild 9).

Falls die Erkennung noch nicht zufriedenstellend arbeitet, kann das Modell mit Lobe und seinen interaktiven Optimierungsalgorithmen weiter trainiert und angepasst werden. Schließlich kann man mit dem Programm „Classifier_live_robot_control_1V0.py" die Funktion in der Praxis überprüfen. Das Programm ist im Wesentlichen identisch mit dem Code für das Jupyter-generierte KI-Modell mit dem Unterschied, dass nun das tfLite-Modell geladen und verwendet wird.

Solange die PiCam kein Zeichen erkennt (resultControl=='bckgrnd'), fährt der Roboter weiter geradeaus. Wird ein Zeichen erkannt (resultControl=='GoLeft' / resultControl=='GoRight'), werden die entsprechenden Richtungsänderungen veranlasst. Der Roboter dreht dann vor einem entsprechenden Schild in die korrekte Richtung ab (s. Titelbild).

Optimierungen und Versuchsanregungen

Das hier vorgestellte Demonstrationsprojekt zum autonomen Fahren zeigt einerseits, dass auch praxisrelevante Lösungen schon mit einfachen Mitteln umzusetzen sind. So genügt bereits die Rechenleistung eines Einplatinen-Minicomputers wie dem Raspberry Pi, um eine effektive Verkehrszeichenerkennung zu realisieren. Andererseits zeigt sich auch sehr schnell, welche Probleme in der Realität auftreten. Neben den rein elektrischen Fragestellungen zu Stromversorgung und Motorcontroller spielen nun auch Themen wie die Bildauflösung, die Frame Rate, also die Anzahl der erfassbaren Bilder pro Zeiteinheit (fps: frames per second), und der Kamerawinkel eine wesentliche Rolle. So wird die maximale Geschwindigkeit des Fahrzeugs von der Frame Rate begrenzt. Bewegt sich das Fahrzeug zu schnell, werden die Zeichen nicht rechtzeitig erkannt und es kommt zu Kollisionen.

Beim Vergleich der beiden Modelle – Keras und Lobe – zeigen sich jeweils klare Vor- und Nachteile. So ist das Lobe-Modell bei vergleichbarem Trainingsaufwand wesentlich treffsicherer, allerdings nimmt die Modellauswertung deutlich mehr Zeit in Anspruch. Während sich beim Keras-Modell noch Frame Rates von bis zu 10 fps erreichen lassen, liegt der Wert bei einem Lobe-Modell häufig bei unter 3 fps.

Bild 9: Zeichen erkannt!

Fazit und Ausblick

Mit diesem Beitrag findet die Artikelserie zum Thema KI-Praxis ihr Ende. Ausgehend von einfachen Neuronalen Netzwerken zur Erkennung von handgeschriebenen Zahlen, über Spracherkennung und -erzeugung bis hin zu Objekt- und Personenerfassung wurden weite Bereiche der modernen KI-Forschung vorgestellt.

Dabei zeigte sich, dass bereits ein kostengünstiger Einplatinen-Computer wie der Raspberry Pi eine erstaunliche Leistungsfähigkeit aufweist. Moderne Algorithmen und Bibliotheken ermöglichen es auch dem noch nicht so erfahrenem Python-Programmierer, hochinteressante Projekte umzusetzen.

All dies wurde bereits mit einer einfachen klassischen CPU erreicht. Welche Möglichkeiten zukünftige Neuromorphe Chips bieten, die bereits auf Hardware-Ebene für KI-Aufgaben zugeschnitten sind, lässt sich heute noch kaum erahnen. Für diese neue Technologie wurden bereits 10000-fache Geschwindigkeitssteigerungen prognostiziert. Zudem sollen die neuartigen Chips mit einem Bruchteil der Leistung ihrer klassischen Konkurrenten auskommen.

Ob allerdings diese so fantastisch anmutenden Entwicklungen der Menschheit immer zum Vorteil gereichen, muss die Zukunft erst noch zeigen! ELV

Material	Artikel-Nr.
Raspberry Pi 4 Model B, 8 GB RAM	250567
Raspberry Pi Kamera-Modul v2	125073
Joy-IT Bausatz Roboter-Auto-Kit	250539
Motortreiber Joy-IT Motormodul	145162

i Weitere Infos

[1] https://www.kaggle.com/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign
[2] https://sid.erda.dk/public/archives/daaeac0d7ce1152aea9b61d9f1e19370/published-archive.html
[3] Downloadpaket: Artikel-Nr. 252783

Alle Links finden Sie auch online unter: de.elv.com/elvjournal-links