KREATIV MIT TECHNIK

c't Make:

Make: ROBOTIK

SONDERHEFT

Technik • Tipps • Praxis

Anleitungen

- KI-fähige Roboter bauen
- Objekterkennung einsetzen
- Mensch-Maschine-Interaktion programmieren

Grundlagen

- Die beste Hardware für Sie
- Software-Frameworks nutzen
- Umgebung mit Laser scannen
- Daten per MQTT austauschen

Bausatz EVO 2: Balanciert auf zwei Rädern

Makey aus dem 3D-Drucker: Tanzt und läuft

JetBot: Findet seinen Weg dank Kamera und KI

So geht's: Roboter selber bauen

2019

CH CHF 18,00
AT, Benelux, IT, ES € 11,90

€ 10,90

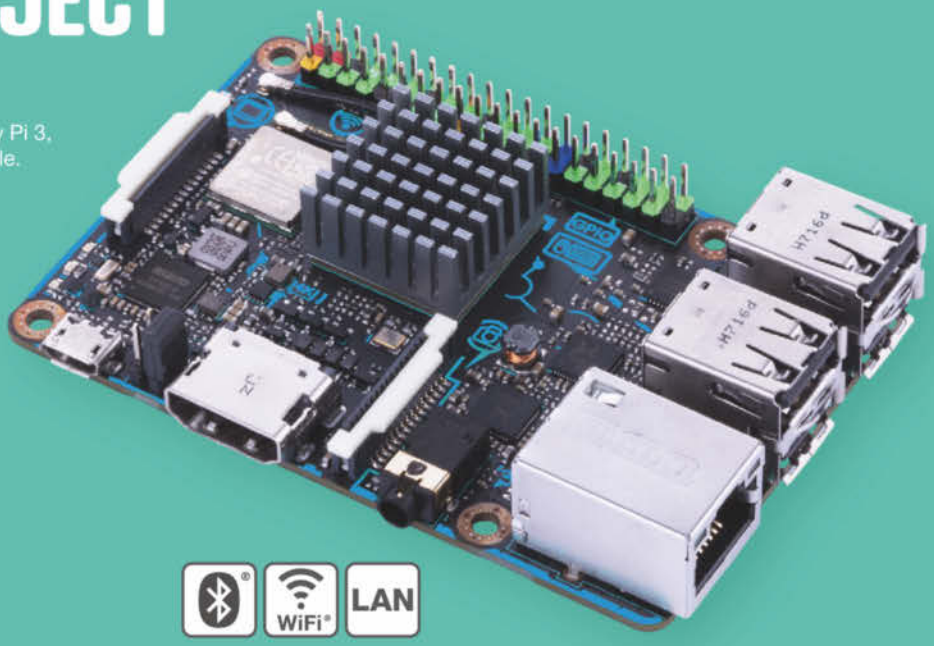
EINPLATINEN-COMPUTER „ASUS TINKER-BOARD S“

THE COMPATIBLE PART OF YOUR PROJECT

ASUS IN SEARCH OF INCREDIBLE

Das Tinker-Board S von Asus übernimmt wie sein Vorgänger nicht nur den Formfaktor des Raspberry Pi 3, sondern auch alle Anschlüsse an der gleichen Stelle.

- 1,8 GHz Quad-Core, 2 GB DDR3
- CPU: Rockchip RK3288
- GPU: ARM Mali-T764 MP2
- Gigabit-LAN, WLAN 802.11b/g/n, Bluetooth 4.0
- 16 GB eMMC Onboard-Speicher



BESTSELLER ~~89,90~~

Bestell-Nr.: **ASUS TINKER S**

83,50

Gehäuse für Raspberry Pi Zero im GB-Design

Dieses Gehäuse im GB-Design beinhaltet ein 2,8“ (7,11 cm) Display, Steuertasten, Adapterboards für den Raspberry Pi Zero W (nicht WH), Lautsprecher, B-Kabel und die Installationsanleitung.

Lieferung ohne Raspberry Pi Zero W, microSD-Karte und Batterien/Akkus

NEU

Bestell-Nr.: **RPI GPI CASE** 69,90

Raspberry Pi Zero W

Aufgrund seiner Größe und des geringen Stromverbrauchs kann er spielerisch für Projekte wie DIY-Drohne, Elektro-Skateboard, Gameboy, schaltbare Steckdosenleiste, und noch vieles mehr eingesetzt werden!

- BCM 2835 SOC @ 1GHz
- WiFi & Bluetooth onBoard
- mini-HDMI Typ C Anschluss
- je 1x micro-B USB für Daten und für Stromversorgung

Bestell-Nr.: **RASP PI ZERO W** 15,00

WiFi

RaspberryPi

Eneloop Pro Akkus, 2450 mAh

Perfekt für Projekte oder Geräte die viel Strom benötigen bzw. lange in Betrieb bleiben müssen.

- 4er-Pack: Mignon (AA) á 2450 mAh
- geringe Selbstentladung
- bis zu 500 Lade-/Entladezyklen

Bestell-Nr.: **ENELOOP P 4XAA** 17,30

Panasonic

Ultra MicroSDHC-Speicherkarte

Ideal für Android™-basierte Premium-Smartphones und -Tablets

- Kapazität: 32 GB
- Lesen bis zu: 98 MB/s
- Class 10 - UHS I

Bestell-Nr.: **SDSQUAR032GGN6MA** 7,95

SanDisk

TAGESPREISE! Preisstand: 22. 10. 2019

- Top-Preis-Leistungsverhältnis
- über 110.000 ausgesuchte Produkte
- Zuverlässige Lieferung – aus Deutschland in alle Welt.

Bestellservice: +49 (0)4422 955-333

www.reichelt.de

reichelt elektronik – The best part of your project

Paradedisziplin

Roboter üben eine besondere Faszination auf Maker aus. Das liegt zum einen daran, dass der Ablauf eines Computerprogramms eine konkrete physische Auswirkung hat und nicht nur in einer Bildschirmausgabe endet. Man kann an der Bewegung des Roboters sehen, wie ein Algorithmus zur Wegfindung, zum Balancieren oder zum Sortieren arbeitet. Man kann mit dem Programm sogar ohne Tastatur und Maus interagieren. Und man sieht sofort, wo Theorie und Praxis auseinanderdriften, wenn der Arm danebengreift oder der Roboter gegen die Wand fährt.

Zum anderen können Maker eines ihrer größten Talente beim Roboterbauen besonders gut zeigen: interdisziplinäres Arbeiten. Robotik ist die Verbindung von Informatik, Elektrotechnik und Maschinenbau. Und diese Kombination macht besonderen Spaß, weil sie viel Abwechslung bietet und man beim Bau zwischen verschiedenen Themen wechseln kann. Läuft's gerade bei der Programmierung nicht so gut, lötet man zur Abwechslung den neuen Motortreiber ein – und schon ist der Kopf und der Blick wieder frei.

Praktisch ist auch, dass man sich je nach eigenem Talent dem Roboterbau von verschiedenen Seiten nähern kann. Der Profi-Programmierer wird sich womöglich eher einen Bausatz kaufen, weil er noch nicht viel Erfahrung mit dem Selbstbau der Mechanik hat. Der Maschinenbauer wird eventuell auf fertige Arduino-Sketche zurückgreifen, dafür aber das Chassis selbst konstruieren und mit dem 3D-Drucker herstellen.

Egal von welcher Seite Sie sich dem Thema auch nähern: Noch nie war es so einfach und günstig, autonome, intelligente und ästhetisch eindrucksvolle Roboter zu bauen. 3D-Druck, Single-Board-Computer, fertige Elektronikmodule und freie, ausgereifte Software machen es möglich. Spracherkennung und -ausgabe, Gesichtserkennung, Navigation: Solche Funktionen stehen Ihnen heute ohne viel Programmieraufwand für eigene Roboter zur Verfügung. WLAN-, Bluetooth- und 4G-Netze machen den Datenaustausch einfach und ermöglichen es, bei doch zu komplexen Problemen Rechenleistung vom Roboter auf einen PC auszulagern.

Steigen Sie jetzt ein. Dieses Heft hilft Ihnen, sich einen Überblick zu verschaffen und erste Projekte anzugehen.

Viel Spaß!

Daniel Bachfeld

Sagen Sie uns Ihre Meinung!
mail@make-magazin.de

Grundlagen

Roboter sind so ziemlich das Faszinierendste, was man als Maker bauen kann – und eine komplexe Aufgabe. Bricht man die systematisch runter, sind die Probleme aber zu bewältigen. Dabei helfen unsere Übersichten zu Soft- und Hardware-Komponenten. Ausführlich erklären wir außerdem die Arbeitsweise von Distanzsensoren und die Kommunikation mit dem Roboter über das Protokoll MQTT.

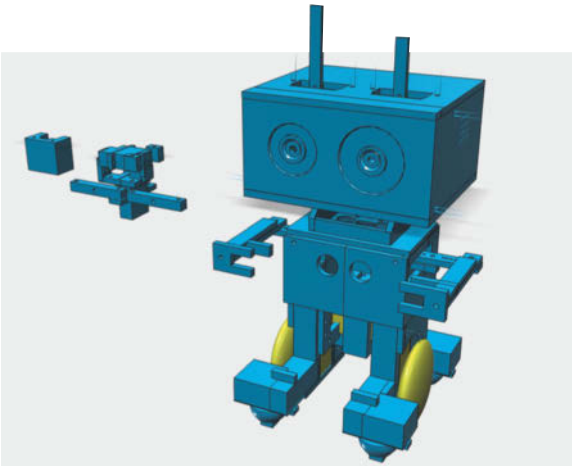
Inhalt

Roboter selber bauen

Der praktische Einstieg in die Robotik ist leichter, als man denkt – dank Bausätzen, 3D-Druckern und fertiger Software erwacht der eigene Roboter schon nach wenigen Stunden zum Leben.

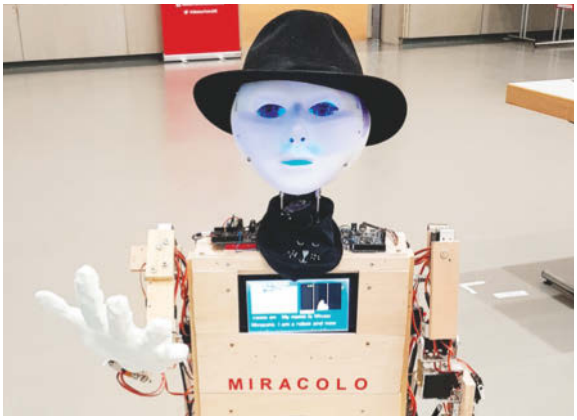
Anleitungen

Roboter können Flaschen von Äpfeln unterscheiden und sortieren, auf schädliche Sitzhaltung am Schreibtisch hinweisen und dank Bordkamera autonom Hindernissen ausweichen. Wir zeigen, wie es geht.



Projekte

Wer spielerisch an Roboter herangeht, hat mehr Spaß – setzen Sie sich an die Alpha-Position eines Maschinenrudels, plaudern Sie mit Miracolo oder schicken Sie echte Roboter auf die Felder des Brettspiels RoboRalley.



Themen von der Titelseite sind rot gesetzt.

Roboter für Kinder

Kinder lieben Roboter! Dank ihrer Fantasie machen sie schon Bürsten mit Motor und Wackelaugen zu ihren besten Freunden. Das kann man sich zunutze machen, um ihnen das Programmieren beizubringen – eine Übersicht.

Roboter für Kinder zwischen 4 und 100 Jahren

90

Der Weg zum eigenen Roboter

Roboter sind so ziemlich das Faszinierendste, was man als Maker bauen kann – und eine komplexe Aufgabe. Bricht man die systematisch runter, sind die Probleme aber zu bewältigen – und das Ergebnis begeistert.

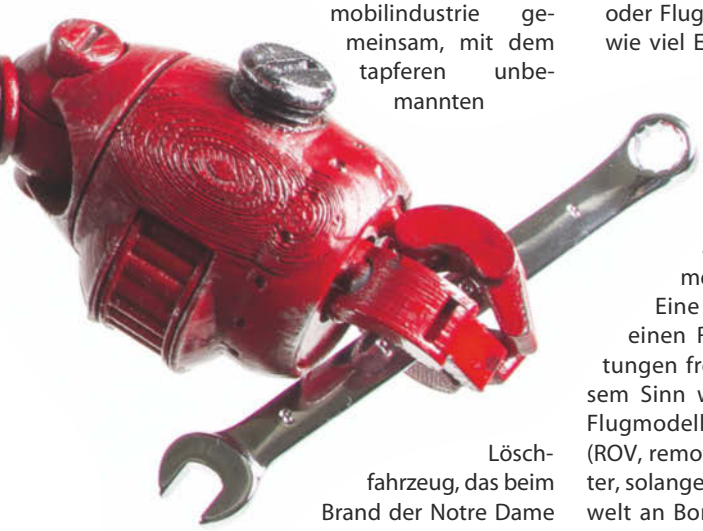
von Peter König

Sieht cool aus, bewegt sich aber nicht von selbst: Der Mega Makey entsprang der Fantasie des 3D-Designers Dan Spangler und wurde in 3D gedruckt. Die Vorlagen gibt es auf Thingiverse zum Download – als Puzzle aus 42 einzelnen Dateien …

Bild: Jeffery Braverman

Was ist ein Roboter? Ist doch klar: ein Maschinenwesen mit Kopf, Armen und Beinen, das sich autonom und souverän durch eine Umwelt bewegt, die mit ihren Treppenstufen, Türklinken und Drehtüren eigentlich auf Menschen zugeschnitten ist. Das präzise und schnell arbeitet, niemals murrt und schon gar nicht ermüdet. Das mit seinen menschlichen Zeitgenossen interagiert, obwohl es ihnen in so gut wie allem überlegen ist, vor allem bei der Intelligenz – und das gerade deshalb auf seine Mitmenschen ein klein wenig seltsam wirkt …

Was diese Beschreibung erfüllt, ist definitiv ein Roboter. Aber was haben *Marvin*, der depressive Androide aus *Per Anhalter durch die Galaxis* und seine Brüder und Schwestern aus der Science-Fiction mit einem Schweißautomaten an der Fertigungsstraße in der Automobilindustrie gemeinsam, mit dem tapferen unbemannten



Löschfahrzeug, das beim Brand der Notre Dame seine 15 Minuten Fernsehruhm hatte, oder mit *Hector 9000*, dem Cocktailmixer mit Raspi-Gehirn aus Make 3/19? All diese Maschinen werden ebenfalls als *Roboter* bezeichnet, ohne dass groß Protest anhebt.

Die Antwort lautet natürlich: Nicht viel. Aber ein paar Gemeinsamkeiten kommen dann doch zusammen:

– Es handelt sich in allen Fällen um technische Konstruktionen, die eine bestimmte **Aufgabe** erfüllen, auf die sie **spezialisiert** sind: Arbeiten, Löschen, Mixen. Die humanoiden Science-Fiction-Roboter hingegen sind so universell einsetzbar und handlungsfähig wie ein Mensch und dabei viel leistungsfähiger. Doch auch darin liegt eine Spezialisierung: In den großen Dramen aus der Zukunft oder einer Parallelwelt sind sie auf Übermenschliches abonniert, auf die Rolle des Superhelden, des Superschurken – oder die des überaus seltsamen Sidekicks.

– Sie **bewegen** sich und **interagieren** mit ihrer Umwelt. Dazu brauchen sie Sensoren und Aktoren. Mobile Roboter erforschen ihre Umgebung (und sei es durch Kollision mit Hindernissen), Industrieroboter wuchten Lasten von links nach rechts und schweißen, bohren oder lackieren, der Cocktailroboter macht aus Getränken Cocktails und bewegt dafür flüssige Zutaten durch Pumpen.

– Um sich zu bewegen und dadurch ihre Aufgaben zu erfüllen, sind Roboter **komplexe** und oft **hybride Systeme** aus elektrischen, elektronischen, mechanischen, pneumatischen, hydraulischen und noch exotischeren Komponenten. Dazu kommt oft (aber nicht immer) ein Schuss Informatik: eine Software und die dafür nötige Hardware für die Steuerung.

– Apropos Steuerung: Ein Roboter hat keinen Menschen an Bord, der ihn steuert – andernfalls handelt es sich um ein Fahr- oder Flugzeug, ein Luft- oder Raumschiff, wie viel Elektronik und Assistenzsysteme auch immer darin stecken. Ein Roboter folgt entweder einem festen, vorgegebenen **Programm** oder handelt **autonom** oder er wird **ferngelenkt**. Oft kommt auch von allem drei ein bisschen zusammen.

Eine wasserdichte Definition für einen Roboter bilden diese Beobachtungen freilich nicht. Zwar wären in diesem Sinn weder ein direkt ferngelenktes Flugmodell noch ein Forschungsfahrzeug (ROV, remotely operated vehicle) ein Roboter, solange es keine Sensoren für seine Umwelt an Bord hat. Montiert man hingegen eine Kamera drauf und überträgt ein Live-Video zum Piloten zurück, müsste man die Vehikel streng genommen direkt wieder in den Club der Roboter aufnehmen. Fordert man deshalb verschärfend, dass eine Maschine autonom auf seine Sensorsignale reagieren müsse, um zum echten Roboter zu werden, ist der Feuerwehrbot von Notre Dame wohl wieder draußen, aber ferngesteuerte FPV-Multikopter mit ihrer automatischen Lageregelung bleiben dann immer noch innerhalb der Roboter-Definition. Genauso ein 3D-Drucker, der per Thermosensor die Druckdüse und den Drucktisch überwacht und deren Heizung entsprechend regelt.

Wir betreiben keine Wissenschaft, wir wollen Roboter bauen. Als Eckpunkte für dieses Heft reichen uns deshalb die vier genannten Aspekte. Sie beschreiben auch grob die Konzeptionsschritte bei der Entwicklung eines eigenen Roboters, deshalb schauen wir sie uns im Folgenden genauer an – und streuen dabei ein, wo Sie im weiteren Verlauf des Hefts noch vertiefende Informationen zu bestimmten Themen finden.

Aufgabe und Spezialisierung

Solange die eingangs erwähnten humanoiden Super-Roboter noch Zukunftsmusik sind, kann ein Roboter in der Regel genau eine Aufgabe erfüllen, für die er konzipiert, optimiert und programmiert ist. Das gilt selbst für Systeme, die über Künstliche Intelligenz verfügen: So kann man eine Bilderkennung etwa darauf trainieren, Hindernisse zu erkennen (Seite 40), Go-Steine und Stellungen auf dem Spielbrett zu erfassen (Seite 70), Gesichter zu finden (Seite 76) oder Flaschen von Äpfeln zu unterscheiden (Seite 106), aber schlecht darauf, das alles gleichermaßen zu können wie der Mensch.

Als Maker hat man natürlich die Freiheit, absichtlich nicht zielstrebig vorzugehen und seinen Roboter als *work in progress* zu konzipieren, mit dem Ziel, dass der irgendwann „alles kann". Für alle anderen – insbesondere die Entwickler von Forschungsprototypen und Industrierobotern – steht hingegen am Anfang die Frage, welche Aufgabe der Roboter genau erfüllen soll und worauf er dabei spezialisiert ist.

Ihr Einsatzzweck ist eine mögliche Systematik, um Roboter einzuteilen. Am verbreitetsten dürften weltweit wohl die meist ortsfesten **Industrieroboter** verschiedener Bauarten sein. Bilder davon gibt es in diesem Heft etwa auf den Seiten 14 und 15 zu sehen. Sie sind dafür gemacht, beispielsweise Bauteile oder zu verpackende Waren aufzunehmen, zu transportieren und zu platzieren (pick and place). Oder sie sind mit unterschiedlichsten Werkzeugen ausgerüstet, um Material und Werkstücke zu bearbeiten. Mit der Technik solcher Roboterarme beschäftigt sich in diesem Heft ein eigener Artikel ab Seite 82.

Die zweite wichtige Gruppe stellen die **Serviceroboter** dar. Dazu zählen die üblichen Staubsaugerflundern, ihre Outdoor-Cousins für die Rasenpflege sowie die (noch) exotischen Fensterputzer und Solarpanel-Polierer. Zu den entfernteren Verwandten gehören Roboter, die Kanäle inspizieren, aber auch jene Multikopter, die dringend benötigte Medikamente übers Watt auf die Ostfriesischen Inseln fliegen. Nicht zu vergessen die Prototypen für Lieferroboter, die optisch meist zwischen selbstfahrender Sackkarre, rollender Kühlbox oder plattgedrücktem Auto changieren und schon in manchen Städten auf den Gehwegen unterwegs sein sollen. Eine spezielle Form stellen noch die **Assistenzroboter** dar, die Menschen mit körperlichen Einschränkungen unterstützen – entweder, indem sie ihnen bestimmte Handgriffe abnehmen und damit Autonomie und Privatsphäre erhöhen, oder indem sie menschlichen Pflegekräften aktiv zur Hand gehen.

Bild: DFKI GmbH, Foto: Florian Cordes

1 Der SherpaTT des DFKI ist ein hybrider Schreit-Fahrrover mit einem aktiven Fahrwerk für hohe Geländegängigkeit. Mit seinem erheblich kleineren Partner Coyote III (siehe Seite 16) bildet er ein Team.

In lebensfeindliche Umgebungen wie die Tiefsee, die dünne Atmosphäre ferner Planeten, in eingestürzte Häuser, Atomruinen oder Kriegsgebiete schickt man ebenfalls bevorzugt Roboter, die irgendwie mit den Servicerobotern verwandt sind, aber gefühlt eine eigene Gruppe bilden, die man **Rover** nennen könnte. Sie werden in der Regel ferngesteuert, liefern Live-Kamerabilder zurück und können dank Aktoren vor Ort Dinge tun – Bodenproben nehmen, Verletzte bergen oder Minen entschärfen, zum Beispiel. Oder neue legen, denn natürlich nutzt auch das Militär Roboter.

Strenggenommen könnte man die sogenannten **Telepräsenzroboter** ebenfalls unter die Rover einsortieren, denn auch sie verleihen dem fernlenkenden Menschen an einem anderen Ort virtuell Augen und Ohren. Dabei geht es aber (in der Regel) nicht um lebensfeindliche Umgebungen, sondern um eine spezielle Form von Telearbeit: Man kann per Telepräsenzroboter an einem Meeting teilnehmen, auch wenn man in einer anderen Stadt weilt. Der Roboter dient dabei als mobile Kamera und Mikrofon einerseits und als materieller Avatar des steuernden Menschen andererseits. Während Telepräsenzroboter oft betont technisch dargestellt werden und an ein Videotelefonie-Tablet auf einem rollenden Besenstiel erinnern, sorgt der japanische Robotik-Professor Hiroshi Ishiguro mit seinem äußerst realistischen, aber umso verstörenderen Androiden-Double immer mal wieder für Gesprächsstoff – er soll es bereits als Stellvertreter in seine Vorlesungen geschickt haben …

Durch seinen einzigen Zweck, mit Menschen in Dialog zu treten, kann man einen Telepräsenzroboter aber auch gut in die Kategorie der **Sozialen Roboter** stecken – die Grenzen der Kategorien sind ohnehin fließend. Andere soziale Roboter agieren hingegen autonom. Beispiele dafür sind in diesem Heft *Bobby*, der sein Gegenüber an eine gesunde Sitzhaltung ermahnt (Seite 76), und *Miracolo*, der Freunde für Unterhaltungen und zum Spielen sucht (Seite 64). Soziale Roboter befragen Besucher auf Veranstaltungen oder führen durch Ausstellungen (siehe auch Seite 119). Auch die ab Werk stubenreinen und antiallergischen Roboter-Haustier-Surrogate wie der „Hund“ *Aibo* oder der „Dinosaurier“ Pleo sind soziale Roboter, die der Unterhaltung dienen und den Spieltrieb befriedigen.

Bleiben noch die **Forschungsroboter** – also Roboter als Objekte der Wissenschaft oder der Maker-Wissbegier, mit denen etwa eine neue Antriebskonfiguration oder ein Einsatzszenario ausprobiert, ein Schwarmverhalten studiert oder bestimmte Aspekte der Mensch-Maschine-Kommunikation untersucht werden. Da sich die Forschungsziele

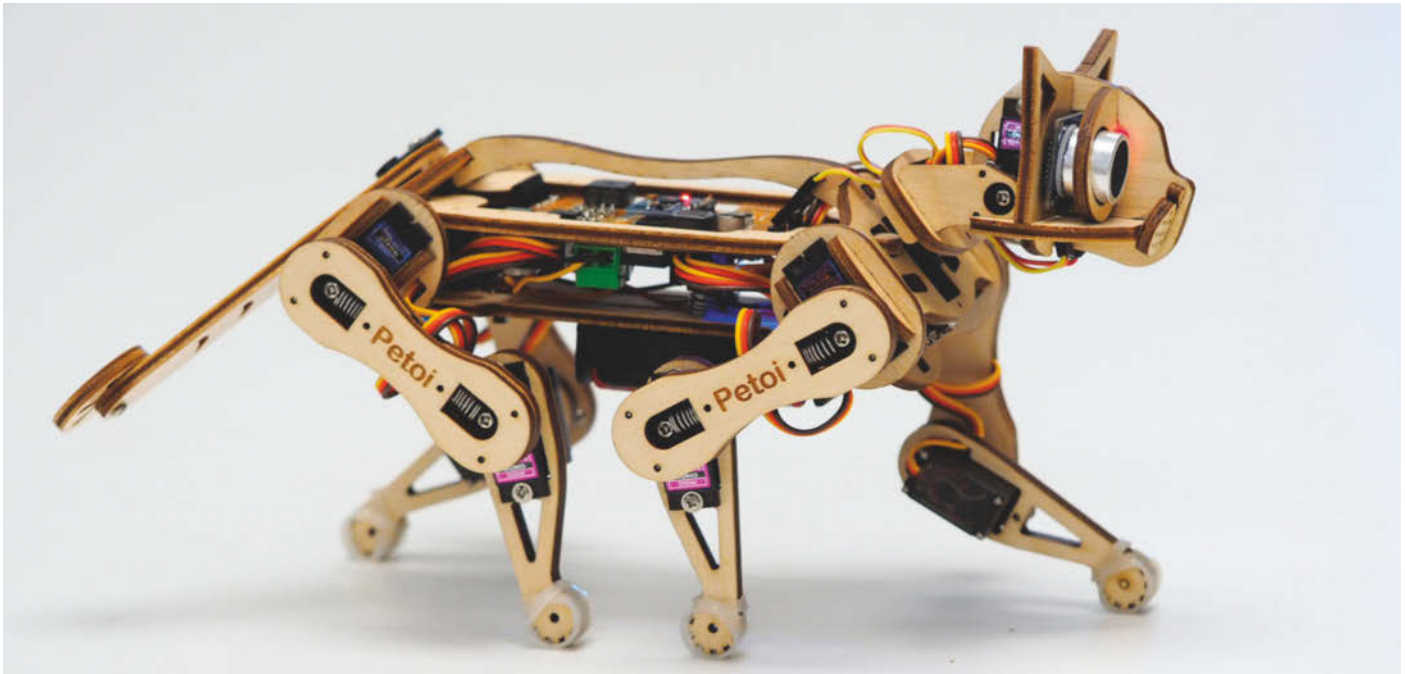


Bild: Rongzhong Li

2 In der Roboterkatze Nybble steckt ein Mikrocontroller für die Steuerung der Bewegungen im Detail – ein Raspberry Pi für mehr Künstliche Intelligenz ist optional.

stark unterscheiden, kommen hierfür die verschiedenartigsten Roboter in Frage.

Interaktion

Ist der Einsatzzweck des geplanten Roboters klar, ergeben sich daraus direkt die Anforderungen an den Bewegungsapparat. Wie unterschiedlich die Konzepte dafür und in der Folge auch die Roboter aussehen können, zeigt der Artikel ab Seite 14 anhand von vielen Beispielen. Auch die nötige Sensorausstattung unterscheidet sich nach Einsatzzweck: Ein Telepräsenzroboter braucht zwingend eine Kamera mit Live-Stream und möglichst geringer Latenz, sonst hat er seinen Zweck verfehlt. Wird er durchgehend ferngesteuert, benötigt er aber nicht unbedingt Distanzsensoren, um Kollisionen mit der Wand zu vermeiden. Ein Staubsaugerroboter hingegen kommt ohne solche Hinderniserkennung kaum zu einem sauberen Ergebnis.

Je konkreter man die zukünftige Umwelt des Roboters kennt, desto genauer lassen sich Sensor- und Aktor-Ausstattung darauf zuschneiden, sprich: Man kann sich Überflüssiges sparen, was nicht nur Geld kostet, sondern vom Roboter durch die Gegend getragen und mit Strom versorgt und auch sinnvoll von der Software ausgewertet werden muss. Ist ein autonomer Staubsauger ausschließlich in einer Etagenwohnung aktiv, sind Sensoren überflüssig, die vor drohendem Absturz an der Treppe warnen können. Ein fest montierter Portalroboter muss nicht nach jedem Anschalten aufs Neue seinen Arbeitsraum erforschen und braucht schon gar keine Räder. Ganz anders sieht das natürlich bei Rovern aus, die unbekanntes extraterrestrisches Terrain erkunden sollen, wie der *SherpaTT* des DFKI (Deutsches Forschungszentrum für Künstliche Intelligenz) in Bremen 1. Um dort mit allen Eventualitäten klarzukommen, kann der Roboter wahlweise mit seinen vier Beinen laufen oder auf den Rollen an deren Ende fahren. Der zentrale Manipulator-Arm hat in jedem Fall eine standfeste Basis.

Hybrides System

Steht der grobe Entwurf, kann man sich die konkreten elektronischen und mechanischen Module zusammensuchen. Unsere Übersicht ab Seite 22 hilft bei der Auswahl passender Boards, Motoren, Servos, Sensoren und dergleichen mehr; speziell mit der Sensorik zum Messen der Distanz zum nächsten Hindernis beschäftigt sich ein eigener Artikel ab Seite 98.

Dank der großen Auswahl muss hier niemand mehr das Rad neu erfinden, man kann sich fast wie bei einem großen Baukasten bedienen. Für Sensoren gibt es etablierte Pro-

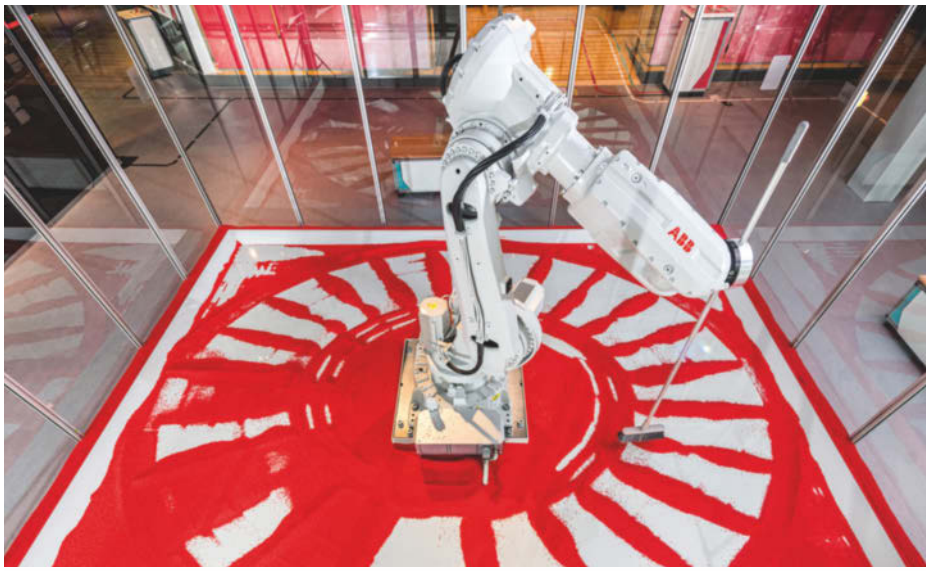


Bild: Heinz Nixdorf MuseumsForum

3 **Dieser Roboterarm fegt zur Zeit unermüdlich im Heinz Nixdorf MuseumsForum und hört auf den Namen Beppo.**

4 **R2-D2 hat mehr Platz und eine bessere Schwerpunktlage für eigene Einbauten als C-3PO in seinem goldenen Metropolis-Maßanzug.**

Bild: Lucasfilm Ltd. & TM

tokolle wie ISP, I²C und oft auch fertige Bibliotheken für Plattformen wie Arduino und Raspberry Pi. Schaut man sich die unterschiedlichen Roboterprojekte an, die wir in diesem Heft ab Seite 40 im Detail vorstellen, stößt man auch immer wieder auf dieselben Klassiker, etwa die Motortreiber L293 und L298 oder den Ultraschallsensor SRF05. Der Rückgriff auf fertige Komponenten und ganze Systeme ist übrigens auch in der Welt der großen Roboter üblich: So besteht die Kehrmaschinen-Installation *Beppo* 3, die zur Zeit in der Ausstellung „Mensch, Roboter" im Heinz Nixdorf MuseumsForum zu sehen ist (siehe auch Seite 119), aus einem Standard-Industrieroboter-Arm. Ähnliche Arme waren in anderen Ausstellungen auch schon damit befasst, die komplette Bibel in Fraktur auf eine große Papierbahn zu schreiben oder aus Hartschaumblöcken mit dem heißen Draht 3D-Kunstwerke zu schneiden.

Hat man seine Komponenten vollständig zusammen und auch schon eine fertige Software für die Steuerung (etwa, weil man ein Projekt aus dem Netz nachbaut), kann man das Innenleben seines Roboters probehalber zusammenstecken und in Betrieb nehmen. Das wird etwa in der Bauanleitung für den Makey-Roboter auf Seite 18 explizit so empfohlen, weil die einzelnen Servos später so tief in die Konstruktion eingebaut werden, dass man lange schrauben muss, um etwa ein defektes Exemplar wieder auszubauen.

Spätestens wenn die Motoren, Sensoren und Boards zur Zufriedenheit zusammenspielen, braucht man aber ein Chassis, auf dem alle Komponenten stabil montiert werden können. Gerade für mobile Roboter, aber auch für die Knochen und Streben von Armskeletten, für Verkleidungen und Abdeckkappen benutzen auffällig viele Maker einen 3D-Drucker. Das mag daran liegen, dass der Bau eines eigenen Roboters im Unterschied zu manchem anderen Projekt oft eine ästhetische und manchmal sogar eine emotionale Seite hat. Dank des menschlichen Drangs zum Animismus unterstellt man einem Roboter latent, eine Art Lebewesen zu sein; und eine solche Konstruktion ist desto faszinierender, je mehr einem ihr Äußeres gefällt. Egal, ob man Diesel- oder Steampunk bevorzugt, ob der eigene Bot postapokalyptisch, retrofuturistisch, martialisch, schick oder einfach nur niedlich aussehen soll – mit der passenden Vorlage ist ein 3D-Drucker mühelos in der Lage, dem Roboter jedes beliebige Outfit aufs Gerippe zu schneidern.

Apropos Gerippe: Deutlich seltener kommt für Roboter-Rahmenbauten der Lasercutter zum Einsatz. Eine Ausnahme und besonderer Hingucker ist die Roboterkatze *Nybble* 2, deren Bausatz man nach einer erfolgreichen Crowdfunding-Kampagne für gut 200 Euro bei Indiegogo bestellen kann. Alternativ gibt es auch eine Open-Source-Version aus 3D-gedruckten Teilen.

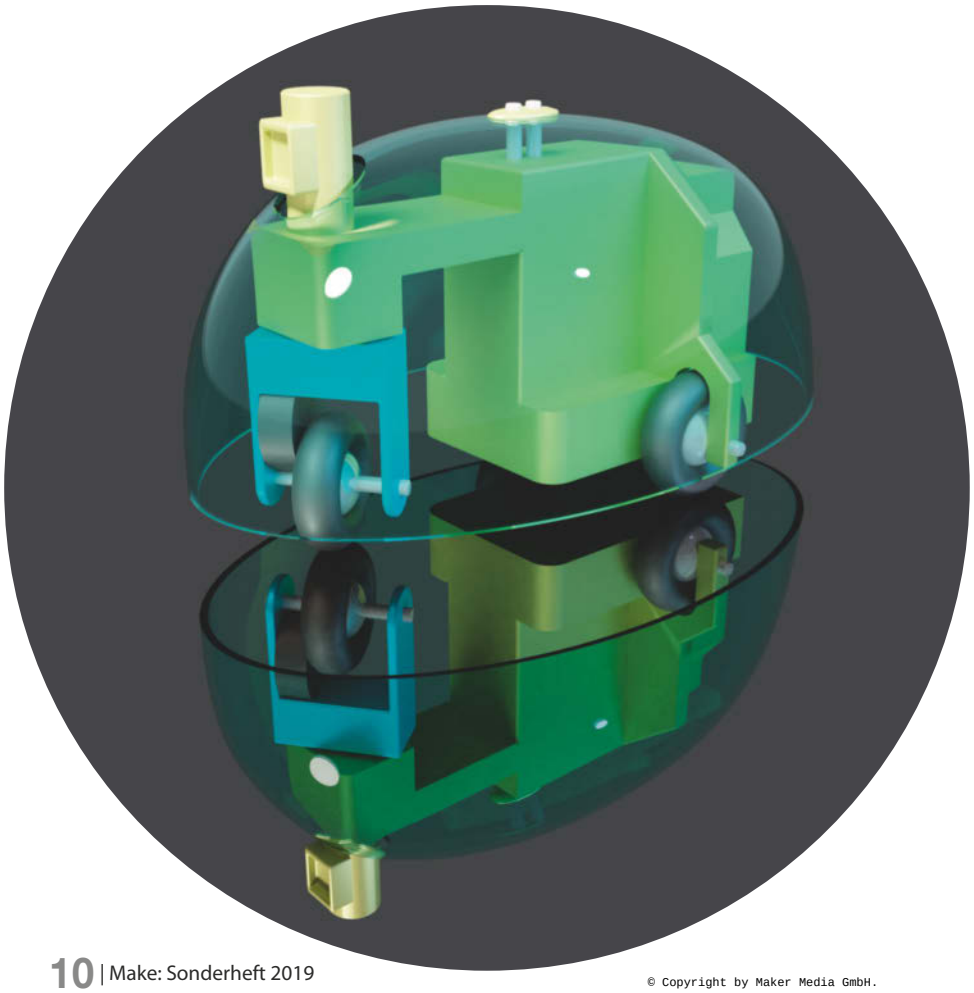
Darüber hinaus kann man natürlich auch Robotertechnik in beliebige Spielzeuge oder RC-Modelle einbauen, vielleicht sogar in solche, die Roboter darstellen. Das lohnt sich oft, denn manches davon sieht zwar gut aus, kann aber enttäuschend wenig. Bevor man loslegt, ist aber ein kritischer Blick auf das zu pimpende Objekt nötig: Ist es stabil genug und bietet sein Inneres genügend Platz für die geplante Technik? Lässt es sich zerstörungsfrei zerlegen? Ist es am Ende stabil genug für die zusätzlichen Innereien und stimmt der Schwerpunkt noch? Der fassförmige R2-D2 ist deshalb ein deutlich erfolgversprechenderes Objekt als der goldene Humanoide C-3PO 4.

Ähnliche Fragen stellen sich natürlich auch, wenn man selbst ein Chassis für seinen Roboter entwirft. Genügend Reserve in Sachen Platz und Stabilität einzuplanen lohnt sich – die nächste Erweiterung kommt bestimmt.

Steuerung

Beim Programmieren der Robotersteuerung helfen freie Frameworks, die der Artikel ab Seite 34 vorstellt. Wie man seine Steuerung prinzipiell aufzieht, ist je nach Einsatzszenario unterschiedlich. Beim Go-Roboter auf Seite 70 etwa genügt zur Modellierung des Ablaufs ein einfaches Flussdiagramm (oder im Informatikersprech: ein endlicher Automat), da die Go-Spielregeln eine genaue Abfolge von klaren Zuständen vorsehen: Der Mensch ist am Zug, setzt seinen Stein, bestätigt dies mit einem Druck auf einen Taster; der Roboter analysiert die Stellung, wählt seinen Zug, nimmt seinen Stein auf, setzt ihn und so weiter, bis die Partie vorüber ist. Eine viel weniger formalisierte Aufgabe hat *Bobby*, der sein Gegenüber beobachtet und dessen Sitzhaltung korrigiert – doch auch sein Verhalten lässt sich noch gut als Zustandsautomat modellieren, wie auf Seite 81 zu sehen.

Anders sieht das bei Robotern aus, die verschiedene Dinge parallel tun wie der mobile Roboter *Miracolo* (Seite 64), der gleichzeitig durch die Gegend fährt, den 3D-Raum analysiert und mit seinem menschlichen Gegenüber kommuniziert. Sein Erbauer Kai Schade realisiert das über verschiedene Subsysteme, die auf diversen Controller-Boards laufen. Miracolos Architektur kann man deshalb fast als eine Gemeinschaft von spezialisierten Robotern auf einer gemeinsamen



5 Drei Räder, zwei Sensoren, kein digitales Gehirn – diese Ausstattung reichte William Grey Walter vor 70 Jahren für die ersten Verhaltensstudien mit autonomen Robotern (schematische Darstellung).

Plattform auffassen. Bei Bedarf kann man zur Koordination einzelne Subsysteme über einen klassischen Bus oder auch über WLAN miteinander kommunizieren lassen.

Die sogenannte *Subsumtionsarchitektur* ist ein anderer, einstmals revolutionärer Ansatz, verschiedene Verhaltensweisen eines Roboters unter einen Hut zu kriegen. Sie entstand als Reaktion auf den klassischen Ansatz aus der Künstlichen Intelligenz, nach dem ein autonom handelnder Roboter in einer Endlosschleife folgende Schritte durchlief:

1. Sensoren auslesen
2. Daraus ein Modell der Welt konstruieren
3. Mit Hilfe dieses Modells die nächste Aktion planen
4. Die Aktion ausführen

Das konnte in den 70er Jahren durchaus pro Durchlauf mehrere Minuten bis Stunden dauern, je nach Hardware des Roboters und Komplexität der Umgebung. Die Folge: Sobald sich in der Zeit zwischen Sensor-Input und Aktion an der Umgebung etwas änderte, schlug der Plan unter Umständen fehl. Dem setzte Rodney Brooks 1986 mit der Subsumtionsarchitektur das Prinzip entgegen, einzelne Verhaltensweisen (*Behaviours* genannt) des Roboters unabhängig voneinander zu implementieren und in Schichten zu organisieren. Die einzelnen Behaviours ergeben sich dabei direkt aus dem Input durch die Sensoren, die dafür relevant sind: So vermeidet ein mobiler Roboter die Kollision mit Objekten, wenn er sich nach links wegdreht, sobald der Entfernungssensor ein Hindernis voraus meldet. Eine Schicht darüber kann man beispielsweise das Verhalten implementieren, durch die Gegend zu fahren. Solange kein Hindernis in Sicht ist, geht es voran, kommt dann ein Objekt in die Reichweite des Entfernungssensors, dreht der Roboter wieder nach links ab. So lassen sich durch übereinander geschichtete Verhaltensweisen verschiedene Ziele unter einen

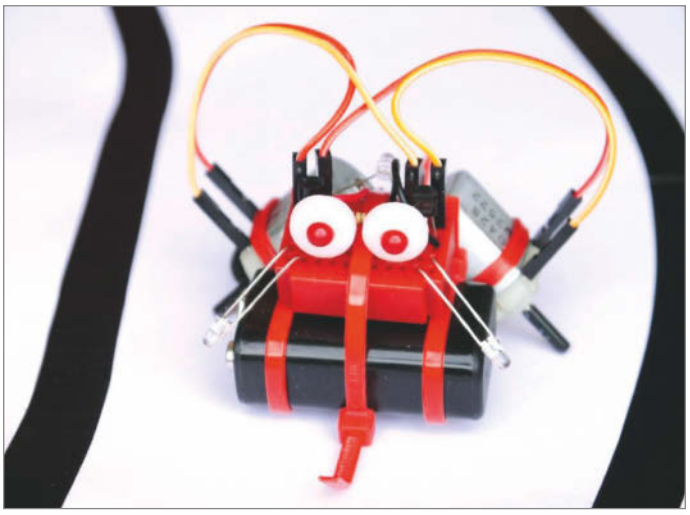


Bild: Tino Werner

6 Zwei Fototransistoren sind genug, damit ein analoger Roboter Linien folgt – ein dritter erlaubt schon unterschiedliche „Programme", je nach Ausrichtung der Lichtsensoren.

7 Der ZeroBot Pro ist als rollende Webcam mit LED-Scheinwerfern sicher schon praktisch, um den Hamster unterm Sofa hervorzuscheuchen – der Raspi Zero W in seinem Inneren schreit allerdings geradezu nach Erweiterungen und etwas Autonomie.

Hut bringen, die der Roboter verfolgen soll: Hindernissen ausweichen, vor Abgründen zurückschrecken, bei Spannungsabfall zur Ladestation zurückkehren, durch ein Labyrinth finden, die Umgebung kartieren …

Dabei kommt es nicht selten vor, dass für unterschiedliche Verhaltensweisen dieselben Sensoren und Aktoren angesprochen werden, wobei die Steuersignale aus unterschiedlichen Quellen sinnvoll integriert werden müssen. So hat der selbstbalancierende Roboter auf Seite 46 lediglich seine zwei Motoren zur Verfügung, um sich einerseits aktiv in der senkrechten Lage in der Balance zu halten und andererseits in die Richtung zu fahren, die man über die Fernsteuer-App oder über WLAN per visueller Programmierung vorgibt.

In diesem Roboter sind alle drei eingangs genannten Arten der Steuerung zu finden: Er behält die aufrechte Position aktiv bei und berechnet dazu **autonom** aus den Sensordaten Motorkommandos. Er fährt auf Wunsch **programmgesteuert** einen vorgegebenen Parcours ab und kann zudem per App auch von einem Menschen **ferngesteuert** werden. Dank des offenen Protokolls, mit dem die auf dem Rechner zusammengeklickten Programme wie auch die Befehle von der Fernsteuer-App übertragen werden, ist aber noch eine weitere Art der Steuerung denkbar: die **Computer-Fernsteuerung übers Netz**. Ein Mars-Rover muss zwingend alle Rechenarbeit mit Bordmitteln erledigen – die Netzabdeckung auf dem roten Planeten soll lausig sein, wie man hört, und Fernsteuerbefehle sind schlicht zu lange unterwegs, um drohenden Hindernissen ausweichen zu können. Die meisten Roboter aus der Maker-Werkstatt hingegen dürften in einem Umfeld zu Hause sein, wo es sich anbietet, zumindest einen Teil der aufwendigen Rechenoperationen auf einen Off-Board-Rechner auszulagern. So erledigt ein externer Laptop die Bildverarbeitung für Bobby und der sortierende Lego-EV3-Roboter auf Seite 106 ist strenggenommen als Handlanger für eine externe KI-Smartcam auf Raspi-Basis aktiv. Wickelt man die Kommunikation zwischen dem Roboter und der Steuerung beispielsweise wie auf Seite 112 gezeigt über ein Protokoll wie MQTT ab, eröffnen sich noch ganz neue Möglichkeiten: Der einzelne Roboter kann hier unter Umständen mit Artgenossen oder Smart-Home-Geräten kooperieren und koordiniert agieren.

Trotzdem: Völlig autonom agierende Roboter haben nach wie vor ihren Reiz, nicht zuletzt deshalb, weil sie auch in anderer Umgebung als der gewohnten funktionieren. Das geht sogar komplett ohne digitale Steuerung wie bei den rein analogen Robotern von Tino Werner (Make 3/16 und online) nach dem Vorbild der simplen kybernetischen Braitenberg-Vehikel 6. Durch Verschaltung von je zwei Motoren, LEDs, Fototransistoren und je einem PNP- und einem NPN-Transistor sind diese Mini-Bots in der Lage, dunklen Linien zu folgen und beispielsweise Licht zu suchen. Das taten auch die allerersten autonomen elektronischen Roboter *Elmer* und *Elsie*, die William Grey Walter 1948/49 baute und die aufgrund der Form ihrer Kunststoffschale und ihrer geringen Geschwindigkeit oft als Schildkröten bezeichnet wurden 5. Dank eines Helligkeitssensors fanden sie den Weg zu ihrer beleuchteten Ladestation, wenn der Füllstand des Akkus zu niedrig sank. Die zentral aufgehängte Glocke diente als Kollisionssensor, sodass die Roboter auf ihren vom Zufall bestimmten Wegen Hindernissen ausweichen konnten. Die rein analoge Verschaltung von zwei künstlichen Neuronen war aber nur begrenzt umprogrammierbar.

Einfach einsteigen

Roboter sind ein komplexes Thema, aber es geht auch einfach: Ab Seite 90 stellen wir Lernroboter-Bausätze (nicht nur) für Kinder und Jugendliche vor, mit denen man als Einsteiger die ersten praktischen Erfahrungen sammeln kann. Auch der selbstbalancierende B-Robot EVO 2 (Seite 46) ist ein Bausatz, der nach zwei Stunden bereits seine Runden dreht, aber dank transparentem Steuerprotokoll, offener Arduino-Firmware und reichlich zusätzlichen Ports auf dem Board eine gute Basis für eigene Erweiterungen ist. Außerdem gibt es ein Community-Forum auf der Webseite des Herstellers, das Fragen rund um den Roboter beantwortet.

Im Netz gibt es allerlei Bauanleitungen für weitere Roboter, etwa für den Makey auf dem Titelbild (Links zu allen hier erwähnten Projekten siehe Kurz-URL am Ende des Textes), den JetBot mit eigener Nvidia-GPU für die Bildverarbeitung (Seite 40) oder den *ZeroBot Pro* 7, in dem ein Raspi Zero steckt, auf dem ein Webserver läuft, der im Browser aufgerufen das Live-Bild von der Kamera zeigt und Steuerbefehle annimmt. Ein Roboter ist das im Grundzustand nur ein bisschen, viel eher eine rollende, ferngesteuerte Webcam (auch wenn die eingangs aufgestellte Roboterdefinition erfüllt ist). Da der Raspi aber genügend Reserven für weitere Sensoren und dann auch Programme für autonome Verhaltensweisen hat, kann der ZeroBot Pro irgendwann noch ein *echter* Roboter werden. Wir bleiben dran und wollen in den kommenden Make-Ausgaben auch mehr über Roboter schreiben – gerne auch über Ihren! Wenn Sie also selbst einen Roboter gebaut und programmiert haben, mailen Sie uns. Wir freuen uns drauf! —*pek*

Alles zum Artikel im Web unter
make-magazin.de/xhch

Die Maker Faires sind die wichtigsten Plattformen der Maker-Szene. Als Aussteller präsentieren Sie sich technologiebegeisterten Menschen, treffen potentielle Mitarbeiter und schwimmen in einem Ideen-Pool. Sprechen Sie uns an, um schon bald die Unternehmenskultur von morgen zu leben.

Werden Sie Aussteller!

Lernen Sie alle Vorteile kennen:

www.maker-faire.de/Aussteller **info@maker-faire.de** **Tel.: 0511 - 5352 133**

Bewegungsapparate

Damit eine Maschine als Roboter gelten kann, muss sie sich bewegen. Je nach Aufgabe und Umgebung sind dazu die unterschiedlichsten Konzepte geeignet. Vorbilder sind dabei oft herkömmliche Maschinen wie Autos oder Portalkräne, manchmal aber auch Tiere und Menschen.

von Peter König

Wenn das Wort *Roboter* fällt, stellen sich die meisten wohl spontan immer noch eine mindestens entfernt menschliche Gestalt aus Blech vor – Science-Fiction-Filme und Comics haben da ganze Arbeit geleistet. Und das, obwohl die meisten wissen, dass echte Roboter in der Regel so ähnlich aussehen, wie die Bilder auf dieser Seite zeigen: hoch spezialisierte Automaten, die sich in der Industrie nützlich machen.

Form follows function – das gilt auch ganz besonders für Roboter. Wie sein Bewegungsapparat gestaltet ist, ergibt sich direkt aus der Aufgabe, die er erfüllen soll. Je nach Arbeit, die die Maschine verrichtet, haben die unterschiedlichen Konstruktionen von **Roboterarmen** ihre jeweiligen Vorteile (auf dieser Doppelseite zu sehen). Mobile Roboter, die fremde Planeten oder dunkle Gegenden unterm Sofa erforschen sollen, die als Haushaltshelfer oder Unterhaltungsmaschinen geplant sind, bewegen sich je nach Terrain auf **Rädern** (folgende Doppelseite) oder besser als **Laufmaschinen** (anschließende Doppelseite) fort. Den Abschluss bilden einige **Spezialisten**, von rollenden Humanoiden über U-Boote bis zur Roboterband.

Wer als Maker seinen eigenen Roboter konstruieren will, kann seine Inspiration für den passenden Bewegungsapparat natürlich auch von teuren Forschungsprototypen beziehen – manche coole Idee aus der Wissenschaft lässt sich auch mit sehr simplen Mitteln in Eigenkonstruktionen nutzen. Auf den folgenden Seiten zeigen wir aber natürlich auch einige Maker- und Open-Source-Projekte zum Nachbauen. Links zu mehr Informationen dazu gibt es unter der URL in der Kurzinfo.

Roboterarm

Der klassische Industrieroboter 1 sieht alles andere als menschenähnlich aus. Doch von allen Robotern, die in der Industrie mit anpacken, ist er dem Prinzip nach derjenige, der am meisten dem menschlichen Vorbild folgt: Am frei beweglichen Ende sitzt ein Greifer oder ein anderes Werkzeug, wie beim Menschen die Hand. Am anderen Ende ist der Arm verankert – beim Roboter oft im Boden, beim Menschen an der Schulter am Rumpf. Dazwischen sorgen längliche Verbinder für Reichweite und eingefügte Dreh- und Schwenkgelenke für Beweglichkeit. Jede Drehebene (im Bild eingezeichnet) wird dabei als Achse gezählt, im Englischen auch als *DOF* bezeichnet (degree of freedom). Ihre Zahl ist eine wichtige Kenngröße für Roboterarme, ausgewachsene Industriemaschinen haben oft 7 bis 9 DOF.

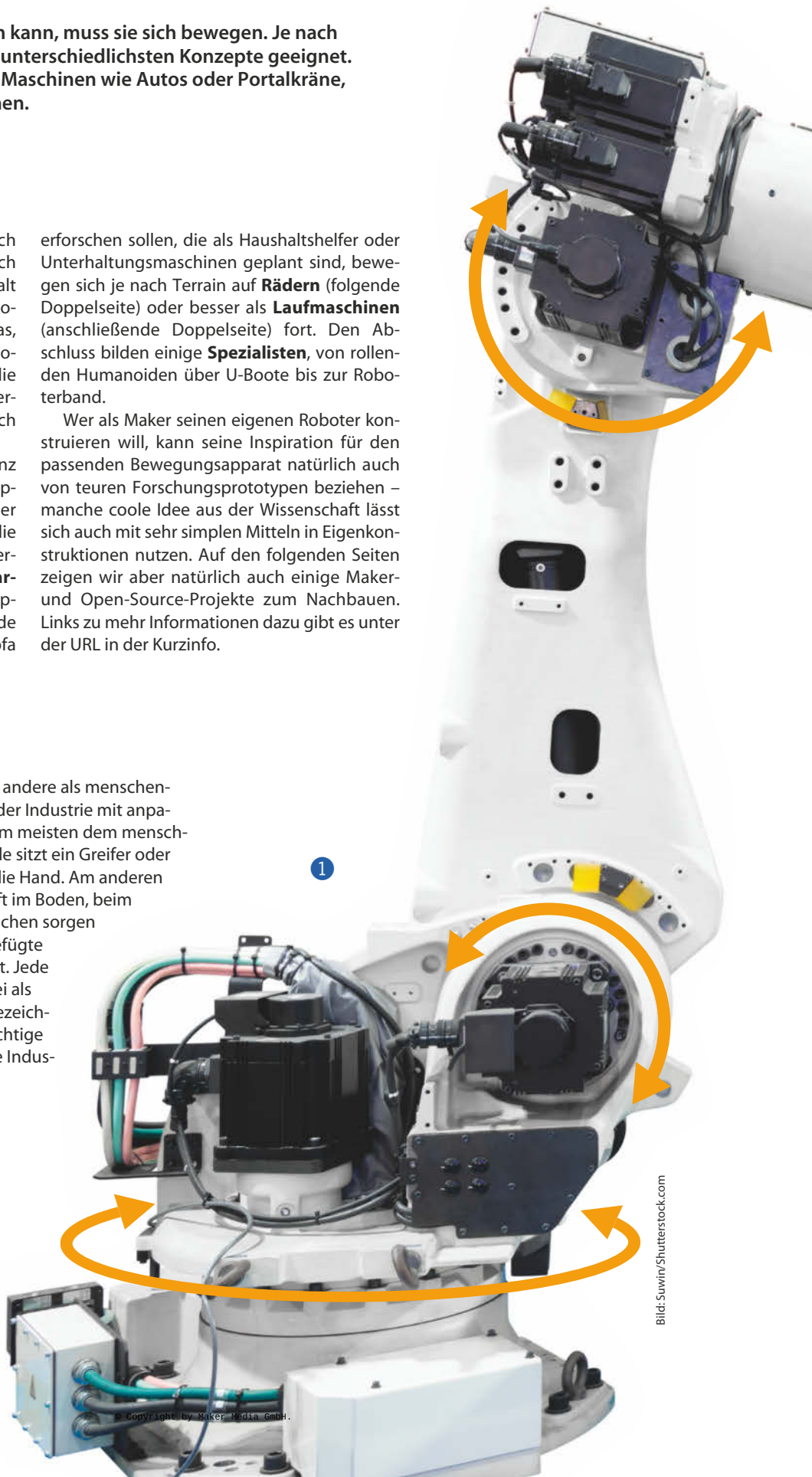
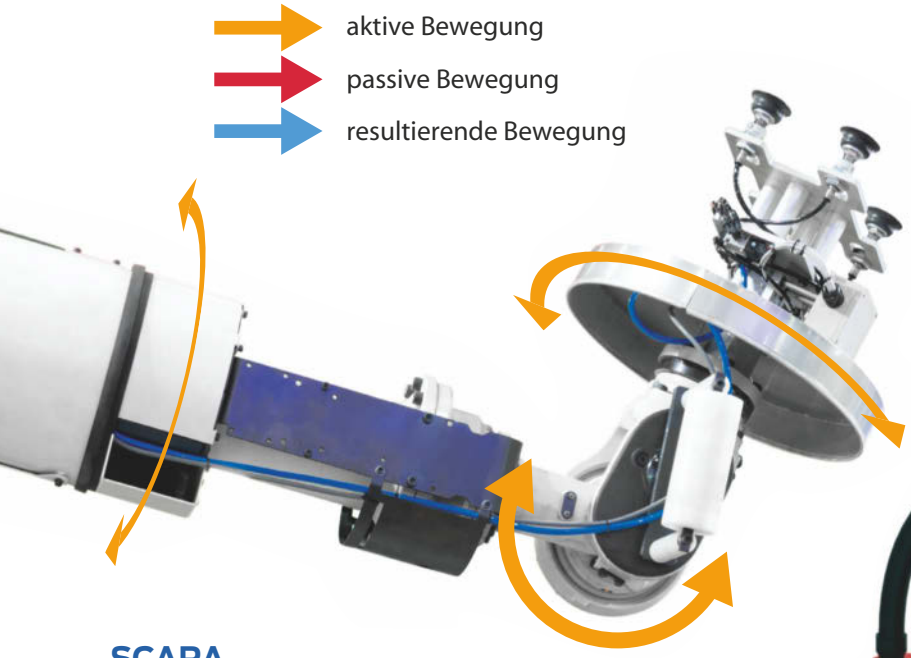


Bild: Suwin/Shutterstock.com

Kurzinfo

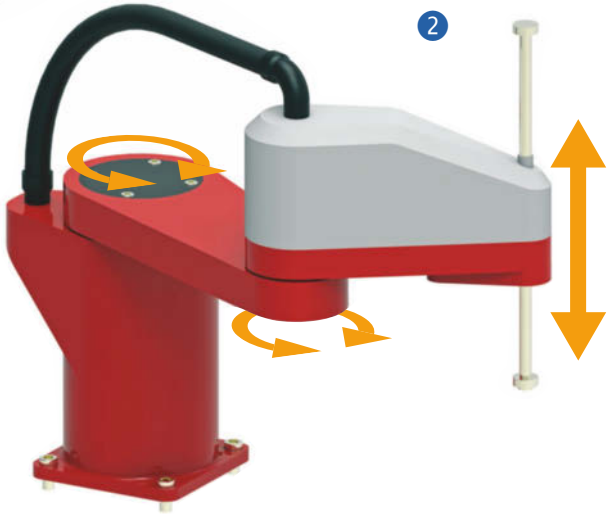
- Funktionsweise von Roboterarmen, Fahr- und Laufmaschinen
- Passende Bewegungsapparate für jedes Terrain und jede Aufgabe

Alles zum Artikel im Web unter *make-magazin.de/x2cz*



SCARA

Das Akronym SCARA steht für *Selective Compliance Assembly Robot Arm*, was zwar einiges über den Einsatzzweck solcher Systeme (2) aussagt, aber wenig über deren Besonderheit: Die ersten beiden Glieder dieser Arme bewegen sich ausschließlich in horizontalen Ebenen, wodurch sich die Position sehr leicht berechnen lässt. Am Ande sitzt meist eine Z-Achse für die senkrechte Bewegung. Wer einen Roboterarm schreiben oder Dinge platzieren lassen will, findet mit einem SCARA eine technisch einfache und platzsparende Lösung.

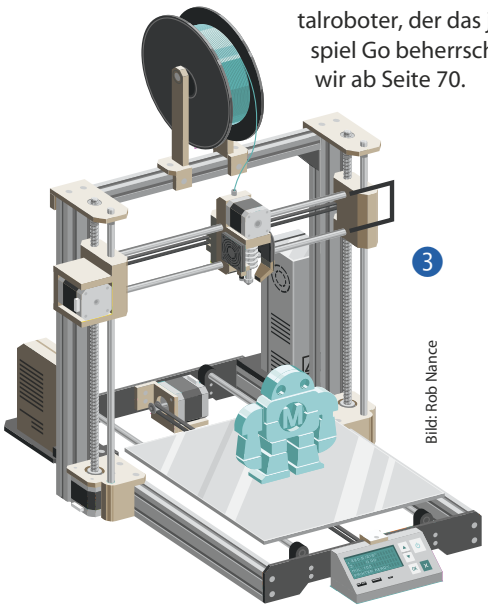


2

Bild: Boris15/Shutterstock.com

Portalroboter

Ja, auch eine XYZ-Positioniereinheit mit senkrecht aufeinander stehenden Achsen ist ein Roboter, sobald sie automatisch ein Programm ausführen kann, um zum Beispiel ein Objekt in 3D zu drucken. 3D-Drucker (3) sind somit ebenfalls Roboter, aber von ihnen ist in dieser Make-Ausgabe nur als Werkzeug für den Bau anderer Roboter die Rede. Einen Portalroboter, der das japanische Brettspiel Go beherrscht, beschreiben wir ab Seite 70.

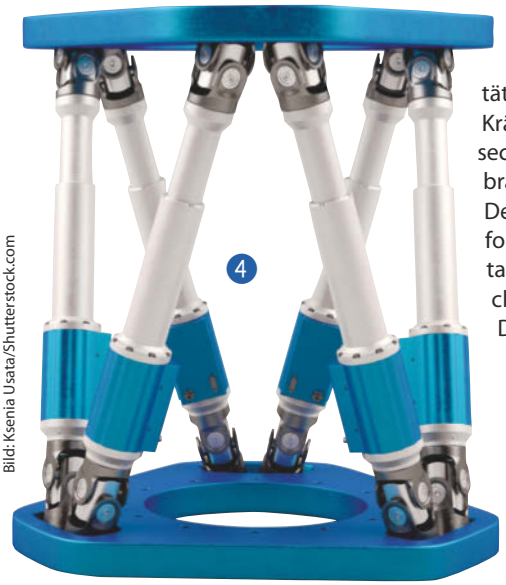


3

Bild: Rob Nance

Hexapod und Delta-Roboter

Wenn hier von *Hexapods* die Rede ist, sind keine sechsbeinigen Laufmaschinen wie auf Seite 21 gemeint, sondern Konstruktionen aus sechs pneumatischen oder hydraulischen Zylindern oder ähnlichen Aktoren, die in der Lage sind, eine Plattform gezielt zu schwenken und in alle drei Raumachsen zu bewegen (4). Sie kommen unter anderem in Flugsimulatoren zum Einsatz, um den Piloten auf dem Sitz auf der Plattform realitätsnahen Bewegungen und Kräften auszusetzen. Sind statt sechs nur drei Arme angebracht, spricht man von einem Delta-Roboter, der die Plattform stets parallel zur horizontalen Ebene bewegt. Diese Mechanik kennt man etwa aus Delta-3D-Druckern.



4

Bild: Ksenia Usata/Shutterstock.com

Roboter auf Rädern

Vier Räder

Roboter auf vier Rädern müssen – wie ein Auto – für Richtungsänderungen mindestens zwei Räder schwenken können und brauchen unter Umständen ein Differenzialgetriebe. Das bedeutet zwar einerseits einen gewissen mechanischen Aufwand, andererseits ist es mit vier Rädern recht einfach möglich, einen Roboter ohne zusätzliche Sensorik und Regelung wirklich geradeaus fahren zu lassen. Zudem kann man für eigene Konstruktionen auf Komponenten oder ganze Chassis aus der RC-Automodellszene zurückgreifen. Das wird zum Beispiel beim Bau eines *Donkey Cars* 5 gemacht – ein Raspberry Pi 3 samt Kamera und Servo-Shield verwandelt dabei ein Modellauto in ein autonomes Rennfahrzeug, das darauf trainiert wird, einer farbigen Linie entlang durch einen Rundkurs zu flitzen.

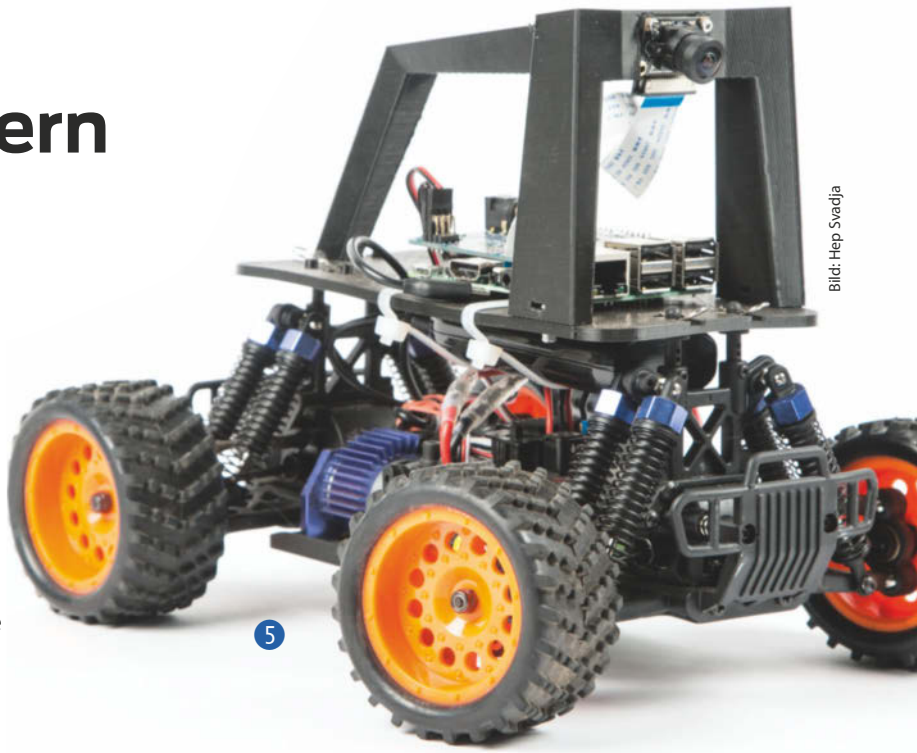


Bild: Hep Svadja

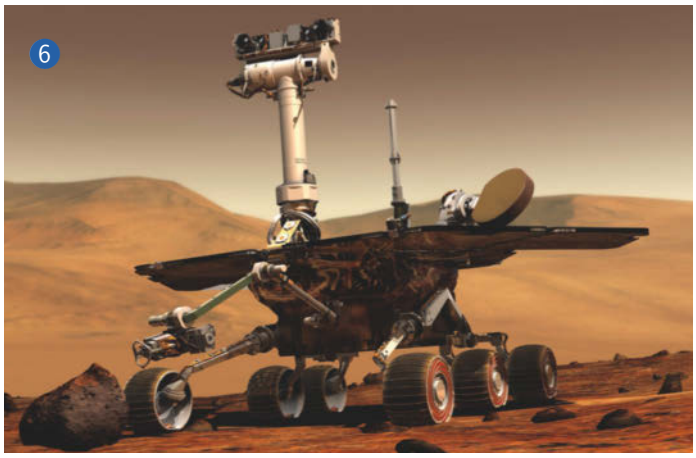


Bild: NASA

Bild: DFKI GmbH, Foto: Florian Cordes

Sechs oder mehr Räder

Roboter mit sechs oder noch mehr Rädern sind besonders geländegängig, wenn das Fahrgestell die dafür nötigen Finessen aufweist wie beim Mars-Rover *Opportunity* 6. Die Lenkung ist allerdings kniffelig, wenn man in den Kurven keine Räder quer über den Boden schleifen will oder für sie spezielle omnidirektionale Räder benutzt (siehe auch „Drei Räder"). Daher findet man solche vielräderigen Roboter bei Selbstbauten eher selten.

Rad-Alternativen

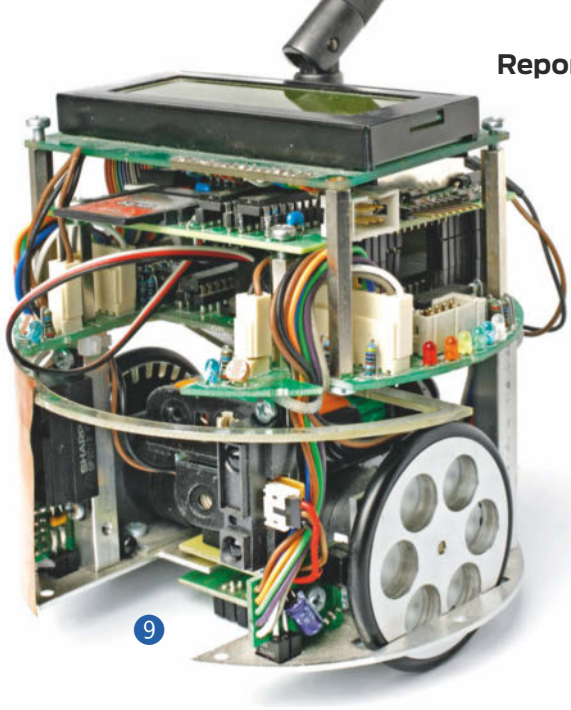
Stufen im Gelände oder steile Hindernisse sind für Roboter mit Rädern meist unüberwindbar – Laufmaschinen (siehe nächste Doppelseite) haben es da oft leichter, sind aber deutlich komplizierter zu bauen und zu programmieren. Ein pragmatischer Ansatz besteht darin, statt Räder auf die rotierenden Achsen einfach ringsherum simple „Beine" zu montieren, etwa in Sternform wie beim Forschungsroboter *Coyote III* vom DFKI in Bremen 7.

Raupen

Mit Raupen ausgerüstete Roboter wie der *Anki Vector* 8 überwinden Unebenheiten leichter. Für Richtungsänderungen lässt man die Raupe auf der einen Seite langsamer oder gar gegensinnig im Verhältnis zu der auf der anderen Seite drehen. Das funktioniert umso besser, je kürzer und breiter der Roboter ist. Raupenroboter sind in der Lage, auf dem Teller zu drehen – falls ihnen dabei die Querkräfte (rote Pfeile) an den Enden nicht die Raupen von den Rädern schälen.

Zweieinhalb Räder

Eine mehr oder weniger runde Bodenplatte, zwei einzeln angetriebene Räder und für die Balance ein Gleitknubbel oder eine Kugelrolle wie in einer alten PC-Maus – nach diesem Strickmuster sind wohl die meisten mobilen Roboter aufgebaut, angefangen von praktisch allen Staubsaugerrobotern bis hin zum c't-Bot aus dem Jahr 2005 (9). Dank zwei getrennt angetriebener Räder braucht man keine Mechanik für die Lenkung, der Roboter kann mühelos auf der Stelle drehen und da die beiden Räder den Boden praktisch nur an je einem Punkt berühren, tauchen die lästigen Scherkräfte wie beim Raupenantrieb nicht auf. Die Nachteile: Ohne Regelung und Odometrie ist die Geradeausfahrt Glückssache und bei mehrstöckigen Exemplaren wird die Schwerpunktlage manchmal kritisch.



9

10

Kugel

Mit dem Erscheinen der Star-Wars-Episode VII im Jahr 2015 war BB-8 gefühlt über Nacht der Roboter der Herzen. Neben einem kommerziellen Produkt von Sphero (10), einer Firma mit Erfahrung bei Kugelrobotern, gab es schon bald diverse Maker-Nachbauten – in Lebensgröße. Das Prinzip: Der eigentliche Roboter folgt dem zwei(einhalb)räderigen Schema, ist aber in seiner Form perfekt in die äußere Kugel eingepasst, auf deren Innenfläche er langfährt und durch sein Gewicht für die äußere Bewegung sorgt. Der Kopf rollt außen über die Oberfläche und wird von Magneten im Inneren an Position und in Richtung gehalten.

Bild: Hep Svadja

11

Zwei Räder

Roboter, die selbstständig auf zwei Rädern balancieren und dabei noch durch die Gegend manövrieren, haben ihre ganz eigene Faszination – sei es, weil sie durch die ständige Balancierbewegung besonders dynamisch wirken oder weil sie irgendwie immer an eine rennende Comic-Figur mit sprichwörtlich rotierenden Beinen erinnern … Ein klassischer Eigenbau-Robotern unter den *Self Balancern* war *Eddie* (11), benannt nach dem Edison-Board von Intel, das darin steckte. Eine Arduino-kompatible Konstruktion nach diesem Muster stellen wir ab Seite 46 vor.

12

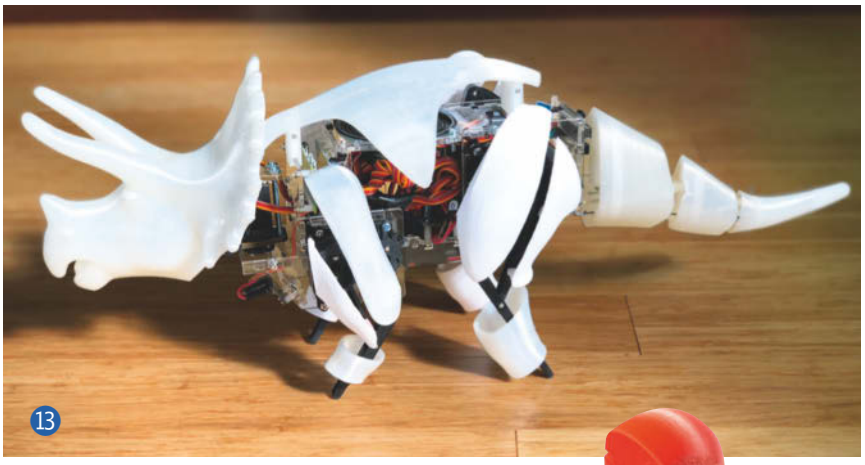
Drei Räder

Roboter mit drei Rädern können beispielsweise dem Muster des Kinder-Dreirads folgen: Ein Rad wird angetrieben und zum Lenken gedreht, die anderen beiden sind einzeln gelagert und drehen passiv mit – so hat es schon der Roboter-Pionier William Grey Walter in den späten vierziger Jahren bei seinen künstlichen Schildkröten namens *Elmer* und *Elsie* gemacht (siehe auch Seite 10). Ganz anders funktionieren die Dreiradroboter mit runder Bodenplatte, an deren Rand im gleichen Abstand drei sogenannte omnidirektionale Räder angebracht sind (siehe auch Seite 28). Solche Roboter können nicht nur auf der Stelle um die eigene senkrechte Achse rotieren wie die zweieinhalbräderigen Kollegen, sondern auch direkt in jede Richtung losfahren. Beim *Tribot* von Wowee sieht man diese Konfiguration sehr schön (12).

Laufmaschinen

Vierbeiner

Vierbeiner sind ein dankbares Konzept für Roboter – sie stehen mit einem gehobenen Bein meist noch sicher und erinnern fast automatisch an Haustiere. Wer von einem Miniatur-Triceratops als Mitbewohner träumt, kann sich den *Intellisaurus* 13 selbst in 3D drucken, einen Arduino Nano als Kleinhirn für die Bewegungssteuerung und ein Raspberry Pi Zero W als Großhirn für Künstliche Intelligenz einbauen. Der Entwurf stammt von Jacquin Buchanan, der sein Design gerade bei GitHub unter Open-Source-Lizenz veröffentlicht hat und den Zusammenbau in YouTube-Videos zeigt (siehe Link in der Kurzinfo). Wer lieber Katzen mag, findet in *Nybble* eine Alternative (siehe Seite 8).



13

Bild: Jacquin Buchanan

15

14

Bild: DeymosHR/Shutterstock

Zweibeiner

Hier kommen sie endlich, die zweibeinigen – um nicht zu sagen: humanoiden Roboter! Weil sie uns Menschen selbst so ähnlich sind, ist man unwillkürlich freigiebig, ihnen allerlei Fähigkeiten zuzuschreiben: *Makey*, das Maker-Faire-Maskottchen 14 scheint einen anzuschauen, dabei stecken nur LEDs in den Augenhöhlen. Und auch wenn er den Eindruck erweckt, er würde jeden Moment loslaufen, kann er in dieser Ausführung nur watscheln: Unter starkem Schwanken der Schultern arbeitet er sich in kleinen Schritten vorwärts, ohne die Zehenspitzen vom Boden zu nehmen. Als Erbe seiner Vorfahren, den Arduino-Servo-Konstruktionen *Bob*, *Zowi* und *Otto* (siehe auch Make 2/18, S. 90), verfügt er nur über zwei Servos pro unterer Extremität. Das reicht nicht für echtes Schreiten, aber allerlei Tanzschritte wie den *Moonwalk*. Ganz anders sieht das beim kommerziell hergestellten *Nao* 15 von SoftBank Robotics aus: Der ist so gelenkig, dass er bereits 2007 beim Roboterfußballwettbewerb Robocup den Vierbeiner *Aibo* als Standardspieler abgelöst hat. Jenseits des Platzes kommt der gut einen halben Meter große Humanoide mit einer auf Menschen zugeschnittenen Umgebung allerdings nicht zurecht: Treppenstufen sind zu hoch für ihn und an Türklinken kommt er nicht heran.

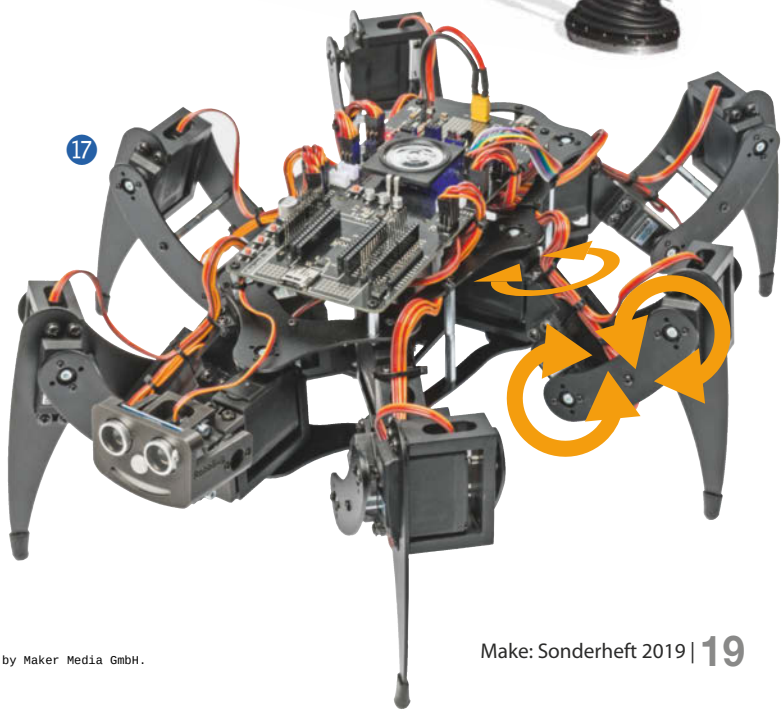
Strandbeest

Eine sehr animalisch anmutende Laufmaschine ist das sogenannte *Strandbeest*, das in der Version unseres Autors Joachim Haas den Titel der Make-Ausgabe 1/16 zierte 16. Strandbeester laufen wunderbar geradeaus – ihr Erfinder Theo Jansen baute die ersten Maschinen dieser Art, damit sie vom Wind angetrieben den Strand entlanglaufen, wie die Krabben, an die sie entfernt erinnern. Zwar kann man durch gegenläufiges Ansteuern der Laufmechaniken auf beiden Seiten ein solches Beest auf der Stelle drehen lassen, bekommt dabei allerdings ähnliche Probleme wie beim Raupenantrieb (siehe vorige Doppelseite), da manche Füße dabei zweitweise quer über den Boden geschleift werden können.

Bild: DFKI GmbH, Foto: Annemarie Popp

Sechs (und mehr) Beine

Vorbild für sechsbeinige Roboter sind meist Insekten. Der *Robobug* von Conrad 17 (siehe Make 6/18, S.128) steht hier stellvertretend für eine ganze Reihe von Konstruktionen und Bausätzen, bei denen sechs baugleiche Beine mit je drei Servos in Bewegung versetzt werden. Ein eigenes *Locomotion Board*, das die Firmware des freien Phoenix-Projekts des Herstellers Lynxmotion nutzt, entlastet beim Robobug den Nutzer davon, dem Roboter mühevoll die Bewegung und Koordination der Beine beizubringen. Der *Mantis* 18 des Deutschen Forschungszentrums für Künstliche Intelligenz (DFKI) in Bremen hat eine Gottesanbeterin zum Vorbild: Dieser Roboter mit sechs Extremitäten kann wahlweise auf allen sechs laufen oder den vorderen Teil des Körpers aufrichten, die vorderen Gliedmaßen als Arme benutzen und auf den vier hinteren gehen oder stehen.



Spezialisten

19

Fliegende Roboter

Mit eingebautem Autopiloten, der beispielsweise GPS-Wegpunkte abfliegt und davon Fotos schießt, wird jedes Flugmodell zum Flugroboter, sei es Luftschiff, herkömmlicher Tragflächenflieger (siehe Make 2/18 S. 42) oder Multikopter 19, von vielen pauschal „Drohne“ genannt. Aktuelle ferngesteuerte Kopter bleiben in der Luft stehen wie angenagelt, wenn man die Knüppel loslässt – dahinter steckt eine Menge Sensorauswertung und automatischer Regelung, sodass man solche Maschinen mit Fug und Recht als Roboter bezeichnen kann.

Autonome (U-)Boote

Natürlich kann man auch schwimmende Roboter bauen, speziell auf offenen Wasserflächen ist die Lokalisierung mit GPS schließlich kein Problem. Noch interessanter ist der Bau von autonomen Wasserfahrzeugen, wenn es unter die Oberfläche geht: Denn in der Tiefe ist weder die Orientierung per Satellit noch die Funkfernsteuerung möglich. Das Ein-Mann-Projekt *Beobachtung3D* von Andrej Gorodkov aus Prag hat mit *Amethyst* 20 ein modulares Design für verschiedene autonome U-Boote entwickelt, die man Stück für Stück auf einem üblichen Maker-3D-Drucker herstellen kann.

20

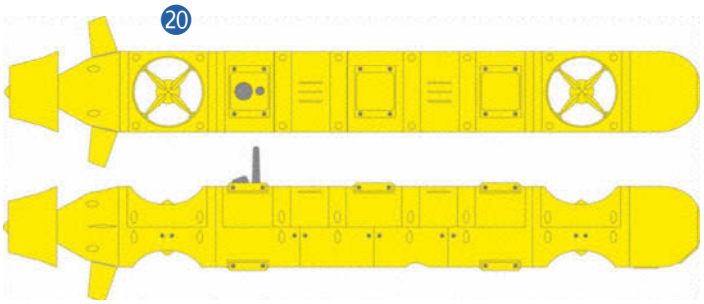


Bild: beobachtung3d.com

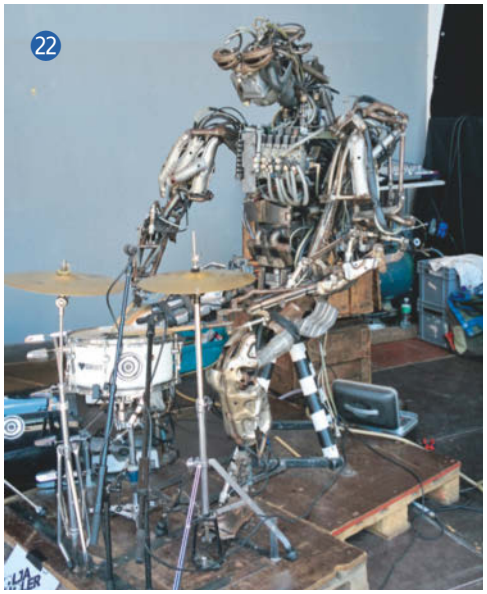
21

Bild: DFKI GmbH, Foto: Annemarie Popp

Mischkonzept

Man kann die auf den vorangegangenen Seiten vorgestellten Varianten von Bewegungsapparaten durchaus als Baukasten verstehen, die sich je nach Einsatzzweck kombinieren lassen. Beim Projekt *Aila* 21 ging es den Forschern vom DFKI darum, ein zweiarmiges Robotersystem zu bauen, das sich in für Menschen gemachten Umgebungen zurechtfinden soll. Um sich durch den Raum bewegen zu können, steht Aila allerdings auf einer sichtlich kippsicheren Plattform mit sechs Rädern statt auf zwei Beinen wie die Frau, die bei der Gestaltung des übrigen Körpers offensichtlich das Vorbild war.

22



Maschinenmusiker

Noch viel stärker als menschliche Musiker sind die Mitglieder der *One Love Machine Band* von Kolja Kugler auf ihre Instrumente spezialisiert: Was sie nicht zum Spielen brauchen, bewegt sich in der Regel nicht. So beherrscht der Drummer *Rubble Eindhoven* 22 nur eine Fußbewegung – die fürs Hi-Hat. Live macht er das aber durch seine Performance an den Sticks und Oberkörpereinsatz mehr als wett. —*pek*

Für Maker!

Zubehör und Lesestoff

shop.heise.de/gadgets

Waveshare Game HAT für Raspberry Pi

Ein Muss für jeden Retro Gamer! Verwandeln Sie Ihren Raspberry Pi in kürzester Zeit in eine Handheld-Konsole.
Mit Onboard-Speakern, 60 Frames/s, Auflösung von 480x320 und kompatibel mit allen gängigen Raspberrys.

shop.heise.de/game-hat **41,90 € ›**

BEST-SELLER

ODROID-GO

Mit diesem Bausatz emulieren Sie nicht nur Spiele-Klassiker, sondern programmieren auch in der Arduino-Entwicklungsumgebung.

shop.heise.de/odroid **49,90 € ›**

NEU

NVIDIA Jetson nano

Das Kraftpaket bietet mit 4 A57-Kernen und einem Grafikprozessor mit 128 Kernen ideale Voraussetzungen für die Programmierung neuronaler Netze, die ähnlich wie Gehirnzellen arbeiten.
Im Set mit Netzteil!

shop.heise.de/jetson **134,90 € ›**

Raspberry Pi-Kameras

Aufsteckbare Kameras, optimiert für verschiedene Raspberry Pi-Modelle mit 5 Megapixel und verschiedenen Aufsätzen wie z. B. Weitwinkel für scharfe Bilder und Videoaufnahmen.

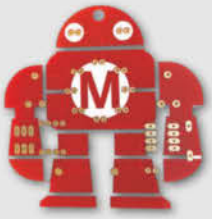
shop.heise.de/raspi-kameras **ab 18,50 € ›**

NEU

ArduiTouch-Set

Setzen Sie den ESP8266 oder ESP32 jetzt ganz einfach im Bereich der Hausautomation, Metering, Überwachung, Steuerung und anderen typischen IoT-Applikationen ein! Dieses Gehäuseset beinhaltet ein formschönes Wandgehäuse mit integrierter Leiterplatte.

shop.heise.de/arduitouch **69,90 € ›**



Makey Lötbausatz

Hingucker und idealer Löt-Einstieg: das Maskottchen der Maker Faire kommt als konturgefräste Platine mitsamt Leuchtdiodendie, die den Eindruck eines pulsierenden Herzens erwecken.

Jetzt neu mit Schalter!

shop.heise.de/makey-bausatz **ab 4,90 € ›**

NEU

Retro-Shirts von c't

„Never change a running system" - eine Weisheit, die seit Ewigkeiten Gültigkeit besitzt. Holen Sie sich den c't 86 - den ersten Selbstbau-16-Bit-Computer mit 8086-Prozessor von c't als hochwertiges schwarzes T-Shirt in den Größen S bis 4XL!
Fans der ersten Stunde tragen alternativ das erste c't-Logo von 1983 auf der Brust.

shop.heise.de/ct-shirts **je 15,00 € ›**

Stockschirm protec'ted

Innen ist Außen und umgekehrt.
Dieser etwas andere Regenschirm sorgt für interessierte Blicke auch bei grauem und nassem Wetter. Als Highlight kommt noch das stilvolle und dezente Design in Schwarz und Blau mit der mehr als passenden Aufschrift "Always protec'ted" daher.

shop.heise.de/ct-schirm **22,90 € ›**

c't Tassen

c't-Leser und -Fans trinken nicht einfach nur Kaffee, sie setzen Statements. Und zwar mit drei hochwertigen Blickfängern, individuell designt für Ihr Lieblings-Heißgetränk: „Kein Backup, kein Mitleid", „Deine Mudda programmiert in Basic" oder „Admin wider Willen". Perfekt für Büro und Frühstückstisch!

shop.heise.de/ct-tassen **ab 12,90 € ›**

NEU

„No Signal" Smartphone-Hülle

Passend für Smartphones aller Größen bis 23cm Länge blockt diese zusammenrollbare Hülle alle Signale von GPS, WLAN, 3G, LTE, 5G und Bluetooth, sowie jegliche Handy-Strahlung.

shop.heise.de/no-signal-sleeve **29,90 € ›**

› Bestellen Sie ganz einfach online unter **shop.heise.de** oder per E-Mail: **service@shop.heise.de**

› Generell portofreie Lieferung für Heise Medien- oder Maker Media Zeitschriften-Abonnenten oder ab einem Einkaufswert von 15 €. Nur solange der Vorrat reicht. Preisänderungen vorbehalten.

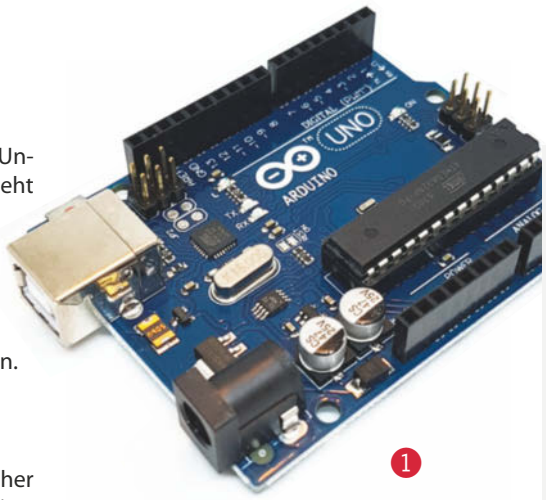
heise shop
shop.heise.de ›

Roboter-Hardware

Je nach Bauweise und geplantem Funktionsumfang benötigen Roboter unterschiedliche Komponenten. Unser Überblick erklärt die wichtigsten Teile und zu welchem Einsatzzweck sie am besten passen.

von Daniel Bachfeld

Was soll ich nehmen, wo liegen die Unterschiede, was ist einfacher, was geht schneller, was ist günstiger? Wer Roboter baut, sieht sich einer Fülle von Wahl- und Kombinationsmöglichkeiten bei der Hardware gegenüber. Dieser Artikel soll als Orientierungshilfe bei der Komponentenauswahl für eigene Projekte dienen.



1

Kurzinfo

- Welche Komponten wichtig sind
- Wozu sie gut sind
- Wie man sie miteinander kombiniert

Mehr zum Thema

- Daniel Bachfeld, Arduino-Alternativen, Make 4/2015, S. 114
- Heinz Behling, Wenn der Raspberry Pi nicht mehr reicht, Make 4/2015, S. 114
- Florian Schäffer, Modellbauservos, Make 4/2015, S. 114
- Carsten Meyer, Motoren und Antriebe, Make 4/2016, S. 102
- Ralf Stoffels, DC-Motoren steuern, Make 6/2016, S. 86

Alles zum Artikel im Web unter *make-magazin.de/xyxj*

Controller

Das Herz eines jeden Roboters, egal welcher Bauart er ist, ist die Steuerung. Sie arbeitet Befehle ab, steuert Motoren, fragt Sensoren ab und koordiniert das Zusammenspiel, um den Roboter eine Aufgabe erledigen zu lassen. Für einfache Robotersteuerungen reichen Mikrocontroller wie der Arduino Uno 1, Nano oder Mega und alle kompatiblen Produkte vollkommen aus. Wie der Name „Mikrocontroller" vermuten lässt, sind sie genau für solche Steuerungs- und Kontrollaufgaben entwickelt worden. Aus diesem Grund haben sie Ein- und Ausgabe-Pins, um weitere elektronische Bauteile direkt daran anzuschließen, etwa LEDs, Relais, Motoren, Infrarotsensoren und so weiter.

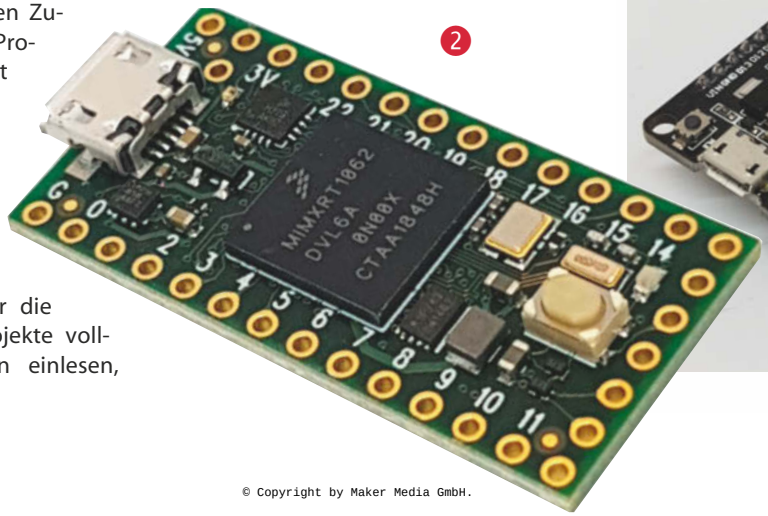
Da die Pins in ihrer Funktion umprogrammierbar sind und je nach Bedarf mal ein digitaler Ein- oder Ausgang oder ein analoger Ein- oder Ausgang oder gar der Pin eines Bus-Systems sein können, spricht man auch von General Purpose I/O (GPIO), also von einem allgemeinen Einsatzzweck. Arduinos mit 8-Bit-ATmega-Prozessoren bringen reichlich solch nützlicher und flexibler GPIOs mit: Analog-Digital-Wandler (ADC), Pulsweitenmodulation (PWM) für LEDs, Motoren und Lautsprecher, einen seriellen Port sowie einen I2C- und SPI-Bus für die Kommunikation mit Sensoren. 1

Die Pins lassen sich über einzelne Steuerregister Hardware-nah in Echtzeit programmieren. Das heißt, im Arduino-Programm greift man direkt auf einzelne Bits in Registern zu, um etwa einen Pin auf logisch High (3,3V/5V) oder logisch Low (0V) zu schalten. Andersherum kann man direkt Zustände an den Pins einlesen, sofern der benutzte GPIO als Eingang konfiguriert ist. Weil man im Programm unmittelbar auf alle Register zugreifen kann und entsprechend den Zuständen sofort den weiteren Programmfluss steuern kann, spricht man von Echtzeitverarbeitung. Mit ein bisschen Übung ist man beim Arduino mit den wichtigsten Registern schnell per Du und hat – zumindest subjektiv gefühlt – die volle Kontrolle über die Hardware.

Der Arduino Uno reicht für die meisten einfachen Robotik-Projekte vollkommen aus: Distanzsensoren einlesen, Gleichstrommotoren steuern, Servos bewegen und Hindernissen ausweichen oder Linien folgen. Dabei hat der Uno eine Stromaufnahme, die Batterien oder NiMH-Akkus problemlos befriedigen können.

Will man viel Hardware anschließen, sind die freien Pins jedoch bald erschöpft. Arduino Unos/Nanos haben zudem nur eine echte serielle Schnittstelle und die ist schon mit dem USB-Chip verbandelt. Zwar gibt es noch eine sogenannte SoftSerial-Funktion, die aber keine Hardwareunterstützung hat und insbesondere bei höheren Übertragungsraten gerne mal Daten verschluckt. Mit einem Wechsel auf einen Arduino Mega für knapp 35 Euro hat man vier serielle Schnittstellen, erheblich mehr PWM- und ADC-Ports und mehr Flash und RAM für Programm und Daten.

Für manche Anwendungen ist die geringe Taktrate der ATmega-Prozessoren von 16 MHz nicht ausreichend. Robotiker wechseln dann oft zur Teensy-Plattform, die zwar auf 32-bittigen ARM-Cortex-Controllern beruht, dank der Kompatibilität zur Arduino-IDE aber dennoch leicht zu programmieren ist. Der aktuelle Wurf „Teensy 4.0" 2 arbeitet mit 600MHz und hat trotz seines günstigen Preises von 20 US-Dollar sogar mehr PWM-Ports als der Arduino Mega. Laut Hersteller hat der Teensy bei 100mA Stromaufnahme genug Power für Aufgaben wie Gestenerkennung, Sprachausgaben und Echtzeitaudioanalysen. Auch komplexere Robotikprojekte sollten sich damit realisieren lassen, für Bildverarbeitung reicht es dann aber nicht mehr.

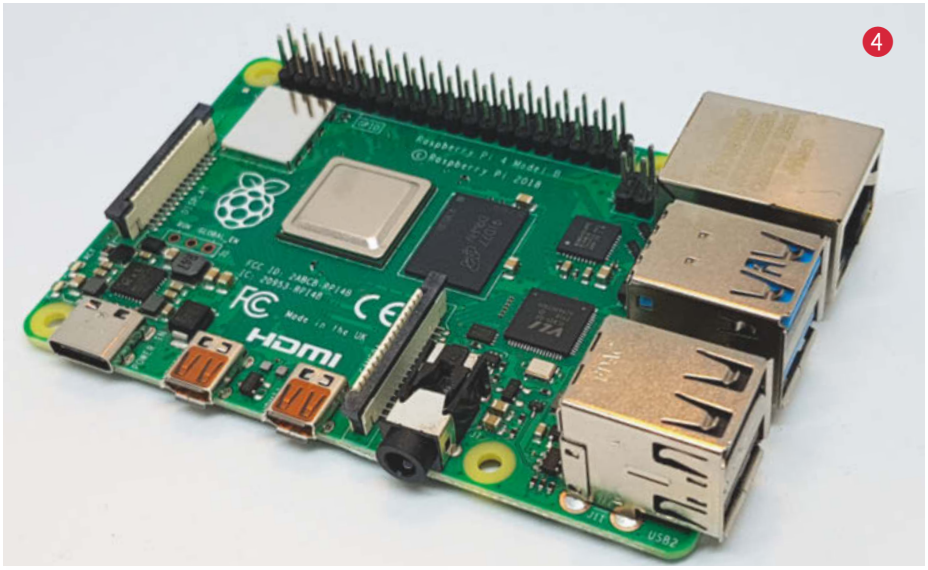


2

Aufgrund der sehr unterschiedlichen Hardware sind manche Funktionen leider doch nicht vollständig kompatibel zum Arduino-Ökosystem, sodass man Anpassungen an Bibliotheken oder dem eigenen Sketch vornehmen muss. Das ist insbesondere für Einsteiger nicht der Königsweg. Für Fortgeschrittene sind Teensy-Mikrocontroller aber eine prima Plattform zum Austoben, auf der man auch noch vergleichsweise einfach die Kontrolle über viele Bits und Register behalten kann.

Dank seiner WLAN-Fähigkeiten und des geringen Preises von unter 10 Euro findet


3

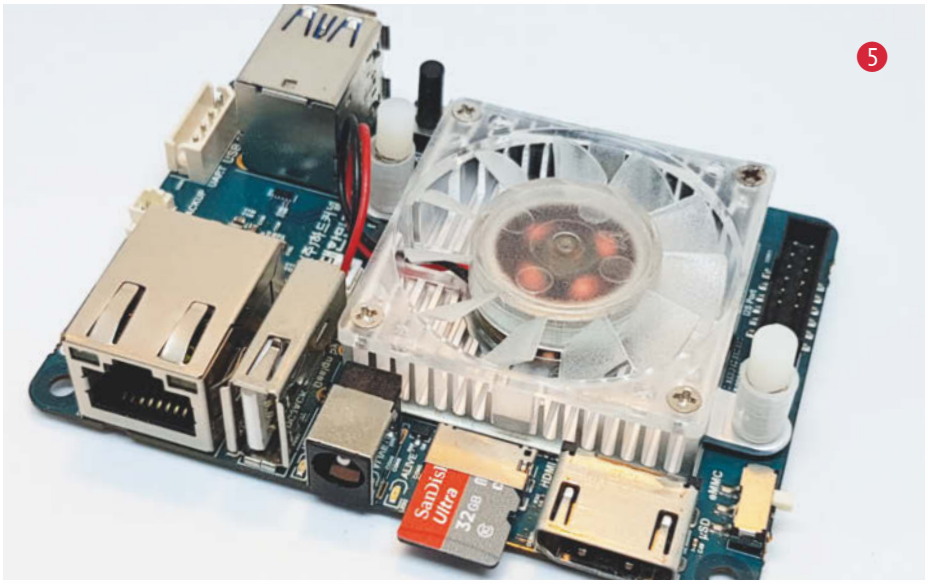


man zunehmend den ESP32 3 in Robotik-Projekten. Auch er lässt sich mit der Arduino-IDE programmieren und ist zu vielen Bibliotheken kompatibel. Zum Anschluss von Sensoren bietet er zahlreiche ADCs und Busse an, für Servo-Motoren können 16 PWM-Channel an beliebigen GPIOs dienen. Nutzt man WLAN, kann man seine Robotikprojekte leicht fernsteuern oder Daten zur Analyse übertragen. Mit seiner zweikernigen 32-Bit-CPU mit 240MHz Takt und 4MByte Flash stemmt der ESP32 anspruchsvollere Projekte. Auch er lässt sich bei einer mittleren Stromaufnahme von unter 300mA mit Batterien und Akkus betreiben, mitunter zieht er aber auch schon mal über 800mA, was bei schwachen Batterien zu einem Reset führen kann 3.

Soll der Roboter komplexe Aufgaben wie Bilderkennung, Spracherkennung, Navigation und Kartierung übernehmen, so greift man auf einen Single Board Computer (SBC) zurück. Der bekannteste Vertreter ist der Raspberry Pi 4, auch wenn er nicht unbedingt der geeignetste für Robotikprojekte ist, da er keinerlei ADCs und nur 2 PWM-Ausgänge hat (von denen einer bereits für die Sounderzeugung auf der Klinkenbuchse genutzt wird). Immerhin stellt er auf seinen GPIO-Pins Anschlüsse für I2C, SPI und serielle Schnittstellen bereit. Und er hat echte USB-Host-Anschlusse, um komplexere Hardware anschließen zu können.

Anders als auf den zuvor genannten Mikrocontrollern läuft auf dem Pi ein Betriebssystem, in der Regel ein Linux. Damit hat man zwar den Komfort eines Desktops, auf dem man direkt programmieren, testen, visualisieren und debuggen kann, allerdings zum Preis einer Abstraktion der Hardware, auf die man in der Regel keinen direkten Zugriff mehr hat. Das soll nicht heißen, dass man damit keine gut funktionierenden Steuerungen mehr bauen kann. Aber eben nicht mehr Hardware-nah und auch nicht mehr echtzeitfähig, weil eigentlich das Betriebssystem die Kontrolle hat und Programme anfragen müssen, ob sie dieses oder jenes Register lesen oder beschreiben dürfen.

Dafür kann man aus einer Fülle an Programmiersprachen und grafischen IDEs schöpfen und ist nicht auf C++ wie in der Arduino-IDE beschränkt. Zudem funktioniert Nebenläufigkeit von Hause aus, womit der Roboter mehrere Dinge in getrennten Programmen erledigen kann. Die populäre freie Robotik-Middleware ROS lässt sich leicht auf dem Pi installieren und bietet eine Fülle an fertigen Funktionen, an denen man alleine jahrelang frickeln würde (siehe auch Artikel auf Seite 34). Dazu gehören Machine Learning, Bilderkennung und Navigation, die sich in eigene Programme einbauen lassen. Der



8

Weg zum Bau eines heimischen Servierroboters war noch nie so kurz wie mit ROS.

Neben dem Pi gibt es andere Alternativen, von denen insbesondere die Odroids noch nennenswerte Verbreitung haben. Lange Zeit war der Odroid XU4 5 erste Wahl, weil der Pi bis zur Version 4 zu wenig Speicher und Rechenleistung bot. Der XU4 hat zwar GPIOs, die aber mit 1,8V arbeiten, was so gut wie kein Sensor, Motor und so weiter unterstützt. Das ist oft aber gar kein Problem, wenn man SBCs und Mikrocontroller kombiniert. Oft sieht man Odroids mit einem Teensy per USB gekoppelt bei der Zusammenarbeit, wobei der Teensy die Hardwaresteuerung übernimmt und der Odroid etwa mit Hilfe eines Laserscanners die Umwelt erfasst und Lenkbefehle an den Teensy sendet.

Ein Problem beim Einsatz von SBCs in mobilen Robotern war anfangs die Stromversorgung, weil SBCs bis zu 3A Strom erwarten, was sinnvoll nur mit teuren LiPo-Akkus oder schweren Blei-Gel-Akkus zu leisten ist – beide benötigen Ladegeräte oder -elektronik. Mittlerweile sind aber USB-Powerbanks günstig und leistungsfähig genug, mehr dazu später.

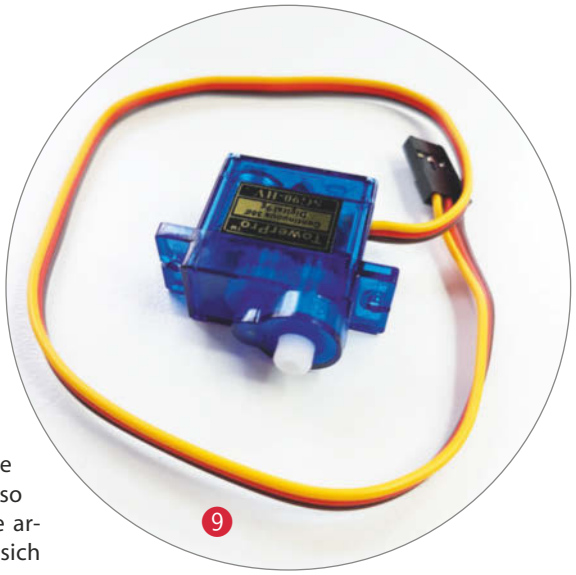
Motoren

Ob nun ein Roboterarm Objekte durch die Gegend wuchtet oder fahrende Roboter den Wegs durchs Labyrinth suchen: In der Regel drehen Elektromotoren die Gelenke oder Räder. Die einfachste Motorform ist der Gleichstrommotor (DC-Motor). Vereinfacht gesagt, dreht sich eine kontinuierlich umgepolte Spule (mit Eisenkern) innerhalb eines Permanentmagneten. Die Polarität der Spannung an den äußeren Anschlüssen der Spule entscheidet darüber, in welche Richtung sich die Spule und schließlich die nach außen geführte Welle dreht.

DC-Motoren gibt es in zahlreichen Größen, Leistungen, Stromaufnahmen und Spannungen sowie Rotationsgeschwindigkeiten. Weil kleine DC-Motoren erstens oft zu wenig Kraft besitzen und zweitens zu schnell drehen, flanscht man ein Getriebe an ihre Welle. Damit untersetzt man die Winkelgeschwindigkeit, womit ein Rad in einer akzeptablen Geschwindigkeit drehen kann. Zudem erhöht sich das Drehmoment. Das Ganze nennt sich dann Getriebemotor.

Populär sind die billigen gelben Getriebemotoren (1:48) mit Plastikgehäuse und -zahnrädern 6, über die man auf so gut wie jeder Robotikseite stolpert. Sie arbeiten zwischen 3 und 6V und drehen sich mit zwischen 90 und 240 Umdrehungen pro Minute, wobei sie je nach Last zwischen 100mA und 1A (bei Blockade) aufnehmen. Im Doppelpack mit zwei Rädern bekommt man sie auf Amazon für unter 8 Euro. Um kleinere, leichte Fahrroboter zu bewegen, reichen sie allemal. Ihr Vorteil ist auch die geringe Betriebsspannung, die man mit 4 Batterien oder NiMH-Akkus erreicht. Auch eine USB-Powerbank reicht aus. Mehr Kraft und eine längere Haltbarkeit versprechen Motoren wie in Bild 7. Der RB350050 arbeitet mit 12V, nimmt im Normalfall 300mA auf, rotiert 120Upm schnell und dreht mit 190Ncm. Der Preis liegt bei rund 20 Euro pro Stück – ohne Räder.

Praktisch sind die China-Motoren GM25-370 8, weil sie mit angeflanschtem Encoder zu haben sind. Damit lassen sich die getätigten Umdrehungen zählen und damit bei fahrenden Robotern Rückschlüsse auf die zurückgelegte Strecke (mehr dazu unter dem Punkt Odometrie). Es gibt sie für verschiedene Nennspannungen – 6V, 12V und 24V – und mit verschiedenen Geschwindigkeiten. Mit dem Quadraturencoder (mit Hall-Sensoren) lassen sich nicht nur die Umdrehungen des Motors zählen, sondern auch die Drehrichtung feststellen. Mit passenden Rädern bekommt man den Motor für unter 15 Euro.



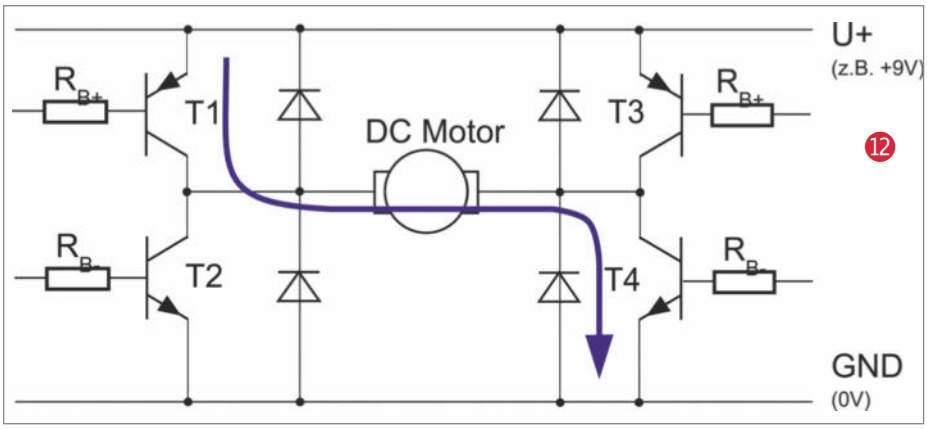
9

Servomotoren sind im Prinzip DC-Motoren mit einer integrierten Regelung, die eine Welle in eine gewünschte Position dreht und dort hält 9. Der Stellbefehl für den Sollwinkel wird als PWM-Signal übertragen, wobei in der Breite des Pulses die Information für den Stellwinkel steckt. Üblicherweise erwarten Servomotoren eine Versorgungsspannung zwischen 5V und 6V. Es gibt sie in verschiedenen Größen und für unterschiedliche Dreh- und Haltemomente, die Preise beginnen bei fünf Euro und gehen bis über hundert.

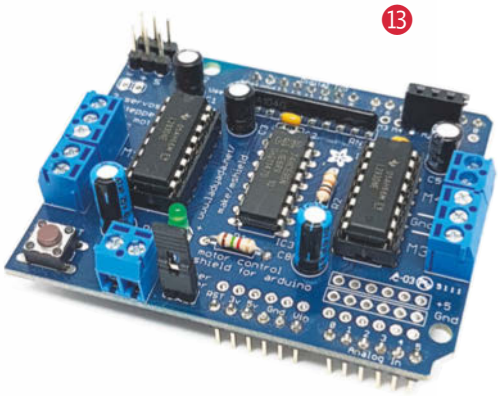
In der Robotik setzt man Servos gerne in die Gelenke von Roboterarmen oder von Lauf- und Krabbelrobotern. Zudem nutzt man sie für sogenannte Pan-Tilt-Halterungen, in denen sie Kameras oder Sensoren schwenken und neigen. Klassische Servomotoren aus dem Modellbau sind eher weniger für Fahrroboter geeignet, es sei denn, sie sind für den „Continuous Rotation"-Betrieb umgebaut worden, also der Anschlag in beide Richtungen fehlt. Dann fehlt dem Servo die Regelschleife (bzw. sie wurde „geöffnet" und der übertragene PWM-Wert legt nicht den Stellwinkel, sondern die Drehrichtung und Geschwindigkeit der Welle fest. Digitale Servos unterstützen „Continuous Rotation" ohne Modifikation.

10

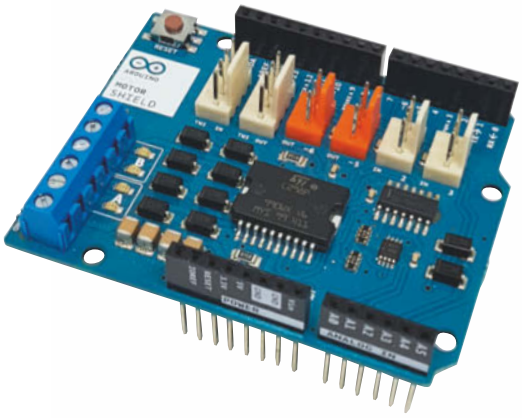
11



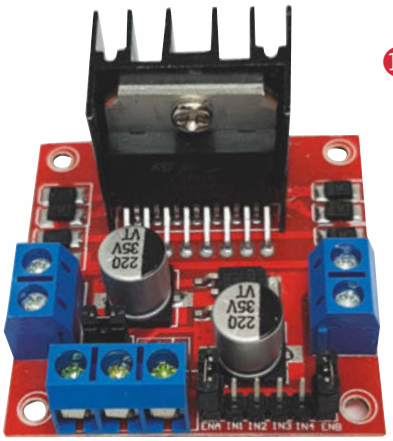
12



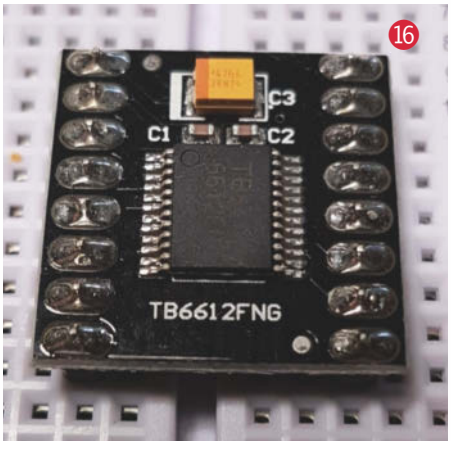
13



14



15



16

Die Königsklasse für Roboter-Servos stellen die Produkte von Dynamixel dar. Sie beherrschen sowohl das Drehen der Welle in den gewünschten Winkel als auch Continuous Rotation. Intern sind sie komplett anders aufgebaut: Ein Mikrocontroller steuert den Motor und kontrolliert den Winkel. Er erhält seine Befehle über einen seriellen Bus mit einem besonderen Protokoll. Der Bus ist kaskadierbar und jeder Servo adressierbar. Leider bedarf es eines zusätzlichen Adapters, um einen SBC an den Bus anzuschließen.

Der in Bild 10 zu sehende Servo AX-12A (50 Euro) hat ein (Stillstands-)Drehmoment von 1,5Nm. Damit lassen sich in Roboterarmen schon einige Gewichte heben. Der AX-12W hat weniger Drehmoment, dreht dafür schneller beim Einsatz als Radantrieb in mobilen Robotern wie dem Turtlebot 3. Das Praktische an allen Dynamixel-Servos ist die eingebaute Odometrie. Über den Bus lassen sich der eingestellte Winkel oder die Zahl der Rotationen abfragen. Die großen Modelle der Pro-Plus-Serie leisten sogar über 40Nm, liegen bei 2700 Euro pro Stück aber eher in drittmittelgeförderten Laboren rum als in der Hobbywerkstatt.

Meist in Portalfräsen und 3D-Druckern findet man Schrittmotoren 11 (aka Stepper), da ihre Stromaufnahme weit über der von DC-Motoren liegt. In mobilen Robotern wird die längerfristige Stromversorgung damit zum Problem. In Schrittmotoren dreht sich ein magnetischer Anker mit der Welle schrittweise in einem rotierenden, elektromagnetischen Feld äußerer Statorspulen. Der Aufbau von Anker und Stator bestimmt, welchen Winkel der Motor pro Schritt dreht, üblich sind 1,8 Grad. Die Spulen (2-phasig) des Stators sind meist bipolar (4 Anschlüsse, 2 pro Phase/Spule) ausgelegt, das heißt, durch Umpolen der Spannungen an den 2 Spulen kann man sowohl die Richtung als auch die Schrittweite steuern. Die Ansteuerung der Stepper ist sowohl hardwaretechnisch als auch softwaretechnisch aufwendiger als bei DC-Motoren, da man 2 Phasen beziehungsweise 4 Anschlüsse in koordinierter Weise schalten muss. Spezielle Treiber (siehe unten) und Bibliotheken nehmen dem Robotiker die Arbeit aber ab.

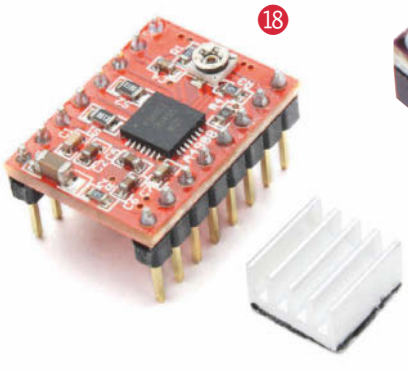
Die Geschwindigkeit der Drehung der Stepper wird allein von der Ansteuerungsgeschwindigkeit der Spulen bestimmt. Da die Motorsteuerung die Schritte selbst vorgibt, muss man die Schritte nicht mehr mit Encodern mitzählen. Der Vorteil von Schrittmotoren ist die Möglichkeit, sie sowohl für kontinuierliche Rotation als auch zum Einstellen und Halten von Positionen einsetzen zu können – ähnlich wie bei den Dynamixel-Servos. Mit 13 Euro für die oft eingesetzten NEMA-17-Stepper mit 1,8 Grad und 59Ncm liegt man preislich aber weit drunter, selbst wenn man noch die Motortreiber dazurechnen muss.

Motortreiber

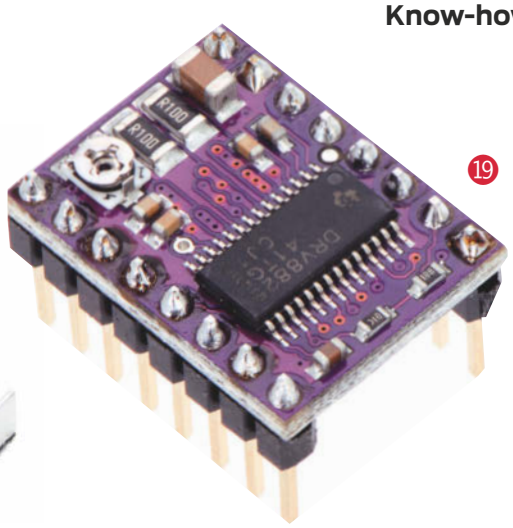
Abgesehen von Servomotoren kann man keine Motoren direkt an die Ausgänge von Mikrocontrollern oder SBCs anschließen – die Transistoren der Ausgänge würden wegen der zu hohen Ströme in Sekundenbruchteilen durchbrennen. Um einen DC-Motor mit einem 3,3V- oder 5V-Pegel eines Pins zu steuern, benutzt man Leistungs-Transistoren, entweder bipolare Transistoren oder MOSFETs. Sofern man den Motor in beiden Richtungen drehen lassen möchte, setzt man sogenannte H-Brücken ein 12. Sie heißen so, weil die Schaltung die Form eines H hat. In der Querverbindung sitzt der Motor und durch Schalten der Transistoren T1 und T4 wird der Motor mit Strom versorgt. Schaltet man stattdessen T2 und T3 an, so wird der Motor umgepolt und dreht in die andere Richtung.

Weil Motoren nicht immer „volle Pulle“ laufen müssen, schaltet man zur Geschwindigkeitsregulierung ein PWM-Signal auf die Eingänge der Transistoren. Im Unterschied zu dem PWM-Signal für Servos gibt die Breite eines Pulses keinen Winkel vor. Stattdessen bestimmt das Verhältnis der Dauer des Pul-

17



18

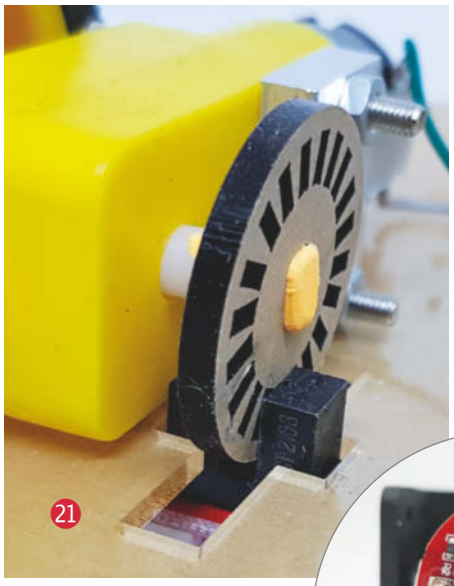


19

ses zur Dauer des Nicht-Pulses die effektive Spannung am Motor und damit dessen Geschwindigkeit.

Praktischerweise muss man sich seine H-Brücke nicht selbst mit diskreten Bauteilen aufbauen. Fertige ICs enthalten bis zu 2 H-Brücken inklusive Ansteuerungslogik, um 2 Motoren gleichzeitig kontrollieren zu können. Zu den Klassikern gehören der L293 (36V, 0,6A) und der L298 (40V, 2A), die beide mit bipolaren Transistoren arbeiten und jeweils Anschlüsse für zwei DC-Motoren haben. In Bild 13 ist ein Arduino-Shield von Adafruit zu sehen, das gleich zwei L293 enthält und somit 4 DC-Motoren oder 2 zweiphasige (bipolare) Schrittmotoren steuern kann. Die Bilder 14 und 15 zeigen einen L298 als Arduino-Shield und einen für den allgemeinen Einsatz (Kostenpunkt 12 Euro im Fünferpack). Die PWM-Signale erzeugt ein Mikrocontroller oder SBC. Daneben lassen sich die H-Brücken über spezielle Logik-Eingänge komplett stromlos schalten, auch wenn weiterhin PWM-Signale anliegen. Das erleichtert das Aus- und Anschalten der Motoren.

Modernere H-Brücken wie der TB6612 16 arbeiten mit MOSFETs, die dank geringerer Durchlasswiderstände eine kleinere Verlustleistung haben und deshalb auch kleiner ausgelegt sein können. Der TB6612 arbeitet mit 15V und verträgt einen Dauerstrom von 1,2A. Zwar ist er nur in einer SMD-Version erhältlich, diverse Hersteller bieten ihn aber auf Breakout-Boards für 10 Euro an. Nur einen DC-Motor, dafür aber mit bis zu 3,6A, unter-



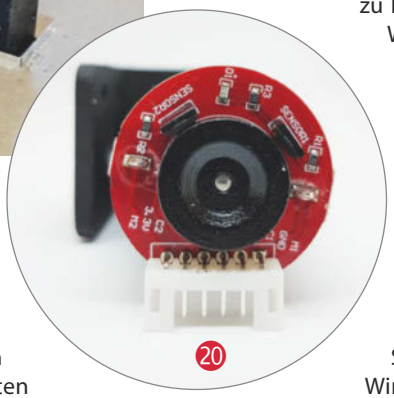
21

stützt der ebenfalls auf MOSFET-Technik beruhende DRV8871 17.

Spezielle Treiber erleichtern die Ansteuerung von Schrittmotoren. Sie enthalten neben den H-Brücken eine Logikeinheit, um die Stepper in neben Vollschritten noch 1/2-, 1/4-, 1/8-, und 1/16-Schritte machen zu lassen. Der IC A4988 18 versorgt die Stepper-Spulen mit maximal 35V und 2A. Der DRV8825 19 kann sogar 45V und 2,2A. Beide ICs bekommt man auf Breakout-Boards für unter 3 Euro.

Odometriesensoren

Um einen Fahrroboter eine bestimmte Strecke zurücklegen zu lassen oder ihn um einen gewünschten Winkel auf der Stelle drehen zu lassen, kann man zwei Wege gehen. Im einfachsten Fall misst man per Zollstock und Winkelmesser die Fahr- und Drehgeschwindigkeit pro Sekunde und berechnet daraus die erforderliche Zeit, um eine gewünschte Strecke oder einen Winkel zurückzulegen. Leider ändert sich abhängig von der Batterie- oder Akkuspannung (sofern sie nicht stabilisiert ist) die Geschwindigkeit, sodass ein Roboter mit halb leeren Batterien langsamer fährt als mit vollen. Eine bessere, aber aufwendigere Lösung ist Odometrie. Damit misst der Roboter selbst die zurückgelegte Strecke und den Drehwinkel.



20

23

22

Auch hier gibt es verschiedene Ansätze, von denen Radsensoren respektive Radencoder weit verbreitet sind. Die Sensoren messen entweder die Umdrehung der Räder, womit man bei bekanntem Raddurchmesser wieder die Strecke ableiten kann. Oder sie messen die Umdrehung des Motors, woraus sich mit Kenntnis der Getriebeuntersetzung ebenfalls die Strecke berechnen lässt. Der Drehwinkel eines Roboters berechnet sich aus den unterschiedlichen Umdrehungen bei zwei Motoren.

Um die Zahl der Umdrehungen zu zählen, dreht sich ein an der Motorwelle befestigter Magnet an zwei versetzt angebrachten Hall-Sensoren vorbei 20. Diese geben nacheinander Impulse aus, die ein Mikrocontroller zählen kann. Aus der zeitlichen Abfolge der Pulse kann der Mikrocontroller sogar ermitteln, in welche Richtung sich die Welle dreht. Bei einer Untersetzung eines Getriebes von 1:48 bedeutet jeder Impuls, dass die Getriebewelle das Rad um 7,5 Grad weiterdreht. Das entspricht bei einem Raddurchmesser von 65mm einer Strecke von 4,25mm. In der Praxis sind auf den Motorwellen aber mehrpolige Magnete montiert, sodass eine Umdrehung der Welle beispielsweise 36 Pulse verursacht. Damit erhöht sich die Auflösung auf theoretisch 0,11mm.

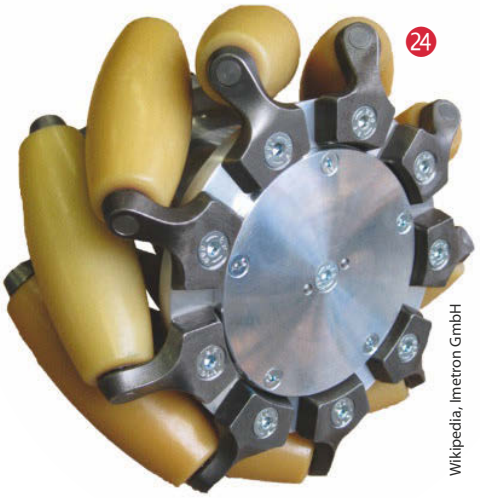
Einfachere Encoder arbeiten optisch: Eine an der Getriebewelle montierte Schlitzscheibe dreht sich in einer Gabellichtschranke 21. Das ist vergleichsweise günstig, hat aber den Nachteil, dass pro Radumdrehung nur 22 Pulse gezählt werden können. Die Genauigkeit pro Puls liegt bei den 65mm-Rädern dann nur noch bei 9mm. Für erste Robotikversuche kann das aber ausreichen. Die Gabellichtschranke ist eigentlich für 3D-Drucker als Endanschlagssensor gedacht und kostet 2 Euro.

Noch günstiger als Gabellichtschranken und Schlitzscheiben sind Reflexkoppler und Encoderscheiben. Auf der Scheibe wechseln sich helle und schwarze Felder ab, von denen das vom Reflexkopper ausgesandte IR-Licht reflektiert wird oder nicht 22. Im einfachsten Fall funktioniert das sogar mit selbst gemalten Pappscheiben. Allerdings benötigt man für die Reflexkoppler etwas Zusatzelektronik, um saubere Signale zu erhalten. Das Ganze ist sehr fehleranfällig – auch wegen Streulicht – und sollte nur bei extrem niedrigem Budget zum Einsatz kommen.

Räder

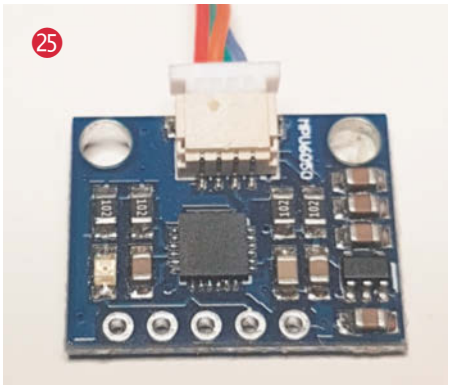
Nur mit Motoren würde sich ein Fahrroboter nicht von der Stelle bewegen, also braucht er Räder oder Ketten. Zu allgemeinen Rädern und Ketten gibt es wenig zu sagen, allerdings gibt es Spezialräder, mit denen ein Roboter sich auch quer zur Laufrichtung bewegen kann, ohne die Räder zu drehen. Bei den Omniwheels (auch Allseitenrad genannt) sind auf der Lauffläche des Rades Rollen angebracht, deren Drehachsen im rechten Winkel zur Drehachse des Rades 23. Damit kann sich das ganze Rad seitwärts bewegen. Der Preis liegt bei 6 Euro.

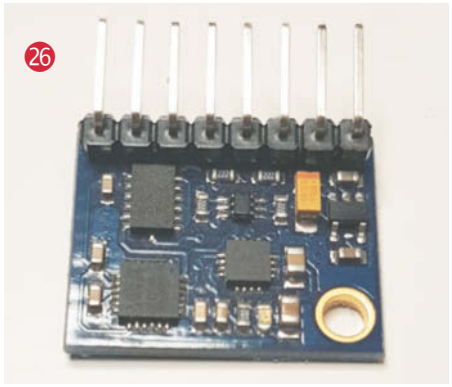
Bei Robotern kommen sie zusammen mit Motoren meist im Dreierpack zum Einsatz, deren Achsen sich im Mittelpunkt des Robo-



24

Wikipedia, Imetron GmbH





ters schneiden. Je nach Kombination verschiedener Geschwindigkeit der drei Motoren kann der Roboter ohne zusätzliche Drehung in jede Richtung fahren; Drehungen sind aber ebenfalls möglich. Der Vorteil der Konstruktion dieses omnidirektionalen Antriebs: Der Roboter muss nicht in eine Richtung wenden, um zu einem Ziel zu fahren. Dieser Antrieb ist beispielsweise bei Fußballrobotern Standard, weil Drehungen im Spiel Zeit kosten.

Ähnlich aufgebaut wie das Allseitenrad ist das Mecanum-Rad, bei dem die Rollen meist in einem Winkel von 45 Grad zur Hauptachse angebracht sind 24. In der Regel treiben vier solcher Rädern einen Roboter an, der sich durch unterschiedliche Geschwindigkeitskombinationen der Motoren in alle Richtungen bewegen kann, ohne dafür lenken oder sich drehen zu müssen. Die Räder sind allerdings erheblich aufwendiger in der Fertigung und deshalb doppelt so teuer wie Omniwheels.

Abstandssensoren

Zu den beliebtesten Übungen gehört es, Roboter irgendwelchen Hindernissen ausweichen zu lassen. Diesem Aspekt haben wir auf Seite 98 einen eigenen Artikel gewidmet, der die Funktionsweise von Infrarotsensoren, Ultraschall-Sensoren, 3D-Kameras und Laserscannern erklärt.

Lagesensoren

Um die Genauigkeit der Odometrie von Robotern zu erhöhen, baut man zusätzlich Lage- und Bewegungssensoren ein. Denn das Beobachten allein von Radumdrehungen ist wegen Schlupf beim Anfahren oder Rutschen beim Stoppen unzuverlässig. Deshalb beobachten Gyrosensoren mögliche Drehungen, Beschleunigungssensoren messen die Neigung der Ebene, auf der sie fahren, und Magnetometer ermitteln die Ausrichtung im Erdmagnetfeld.

Die Sensoren sind in MEMS-Technik (Micro-Electro-Mechanical Systems) gefertigt, für jede Messgröße wie Drehrate/Winkel (Gyro), Beschleunigung (Accel) und Magnetfeld (Magneto) gibt es eigene ICs. Da jeder Sensor in drei Raumachsen messen kann, spricht man von 3 Degrees of Freedom (3DoF). Kombiniert man zwei Sensoren, hat man 6DoF, bei drei Sensoren 9DoF. Diverse Anbieter kombinieren die ICs auf Breakout-Boards, von denen man die Sensordaten meist per I2C oder SPI abfragen kann.

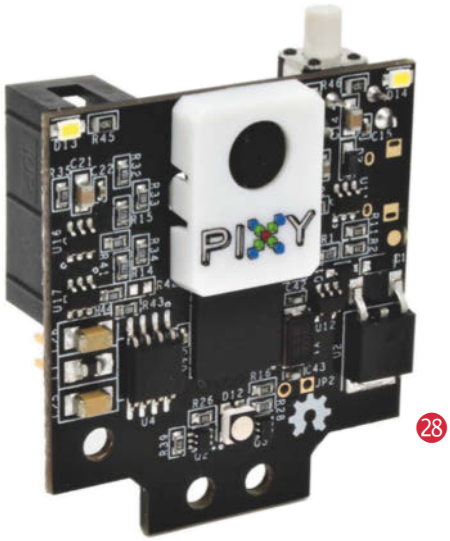
Das Sensorboard MPU6050 bringt einen Gyro und einen Beschleunigungssensor mit 25, das GY85 hat zusätzlich noch ein Magnetometer an Bord 26. Für beide Boards gibt es etwa für Arduino freie Bibliotheken, um die Daten auszulesen und in eigenen Projekten einzusetzen. Zusammen mit den Odometriedaten kann man bei mobilen Robotern auch nach längerer Fahrt recht genau die Lage und Orientierung im Raum verfolgen. In ROS gibt es spezielle Pakete, die beim Verwerten der Daten helfen.

In Self-Balancing-Robotern und Drohnen nutzt man die Sensoren, um die Lage zu halten und ein Umkippen oder einen Absturz zu verhindern.

Roboter im Außenbereich können ihre Position und Lage zudem mit GPS-Empfängern ermitteln und mit den anderen Odometriedaten fusionieren.

Kameras

Um die Außenwelt bildlich wahrzunehmen, bieten sich für SBCs normale Webcams an, wobei sich die Logitech C270 wegen ihrer Kompatibilität zu Linux großer Beliebtheit erfreut 27. Beim Raspberry Pi setzt man eher auf die PiCam, die dank des speziellen CSI-Anschlusses keinen USB-Port belegt. Sie ist auch in anderen SBCs mit CSI-Anschluss einsetzbar. Mit den Kameras können die Computer etwa per OpenCV Gesichter oder Objekte erkennen.



32

33

Doch auch für Roboter mit Mikrocontrollern gibt es eine Möglichkeit, Bildverarbeitung zu nutzen. Intelligente Kameras übernehmen die Aufgabe der Bilderkennung und senden per serieller Schnittstelle etwa Koordinaten, wo ein Objekt platziert ist, oder eine Zeichenkette, was gerade im Bild zu sehen ist.

Die Pixy2 Cam 28 (100 Euro) kann beispielsweise Linien erkennen und per USB, UART, SPI oder I2C als Vektor ausgeben. Für den Einsatz in Linienfolgern ist die Kamera wunderbar geeignet. Mit 16 verschiedenen Barcodes auf der Strecke kann man der Kamera bei der Fahrt Befehle übermitteln, etwa „abbiegen“ oder „langsamer fahren“.

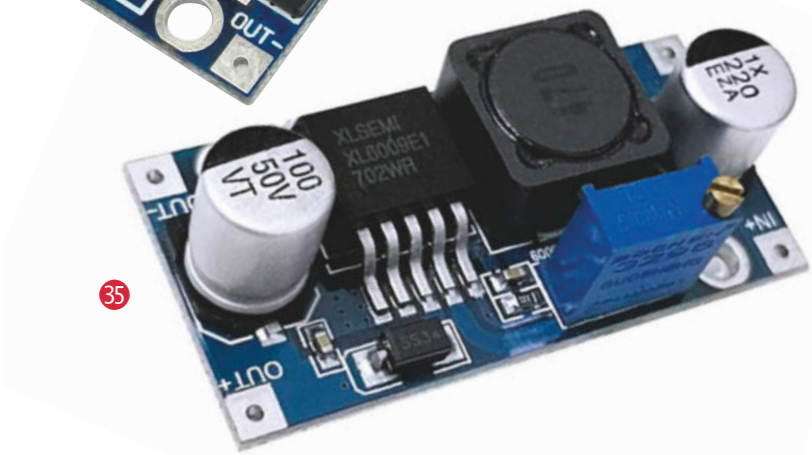
Darüber hinaus kann die Pixy2 bis zu 255 Objekte erkennen und tracken. Allerdings beruht das dafür benutzte Verfahren auf der Erkennung von 8 unterschiedlichen Farben (Color Connected Components). Eine echte Objekterkennung ist das nicht, eher ein Tracking einzelner Farbfelder im Kamerabild (Blob-Tracking) – das Ganze aber megaschnell.

Immerhin kann man Objekten anhand einer Farbkombination ein Label zuordnen. Klebt beispielsweise an einem Würfel ein grün-oranger Aufkleber, so kann man diese Farbkombi als „Würfel“ konfigurieren. Für Arduino (C++) und den Pi (Python) gibt es kostenlose Bibliotheken und Programmierbeispiele, die den Umgang mit der Kamera vereinfachen.

Die JeVois Smart Machine Vision Camera 29 kostet ebenfalls rund 100 Euro. Sie kombiniert im Prinzip einen ARM-Singleboard-Computer mit einer Kamera, Computer-Vision- und ML-Algorithmen (OpenCV, TensorFlow, Darknet) sowie einem USB-Port und einer seriellen Schnittstelle. Die Kamera nimmt Bilder auf, analysiert sie je nach gestellter Aufgabe, schreibt die Lösung in das Bild hinein und reicht es an den USB-Port weiter. Ein TTL-kompatibler serieller Port zum direkten Anschluss an Arduino & Co. ist zusätzlich vorhanden, damit Mikrocontroller-Projekte die Kamera nutzen können.

JeVois kann je nach Bedarf viele unterschiedliche Aufgaben übernehmen, die bereits in die Firmware auf der eingesteckten MicroSD-Karte integriert sind. Die jeweilige Anwendung (Gesichtserkennung, Objekterkennung, Tracking) startet man unter anderem durch Befehle über die serielle Schnittstelle. Ein eigenes Software-Framework erleichtert die Programmierung der Kamera in Python und C/C++.

34



35

Stromversorgung

Solange der Roboter mit all seinen Sensoren, Antrieben und Controllern auf dem Tisch steht und per Netzteil mit Strom versorgt wird, funktioniert meist alles problemlos. Soll er dann mit Akku oder Batterie durchs Zimmer fahren, beginnt oft das Drama: Reboots der Steuerung, zitternde Motoren, fehlerhafte Sensordaten. Schuld ist meist die Stromversorgung, die ungenügend Strom liefert oder deren Spannung einbricht.

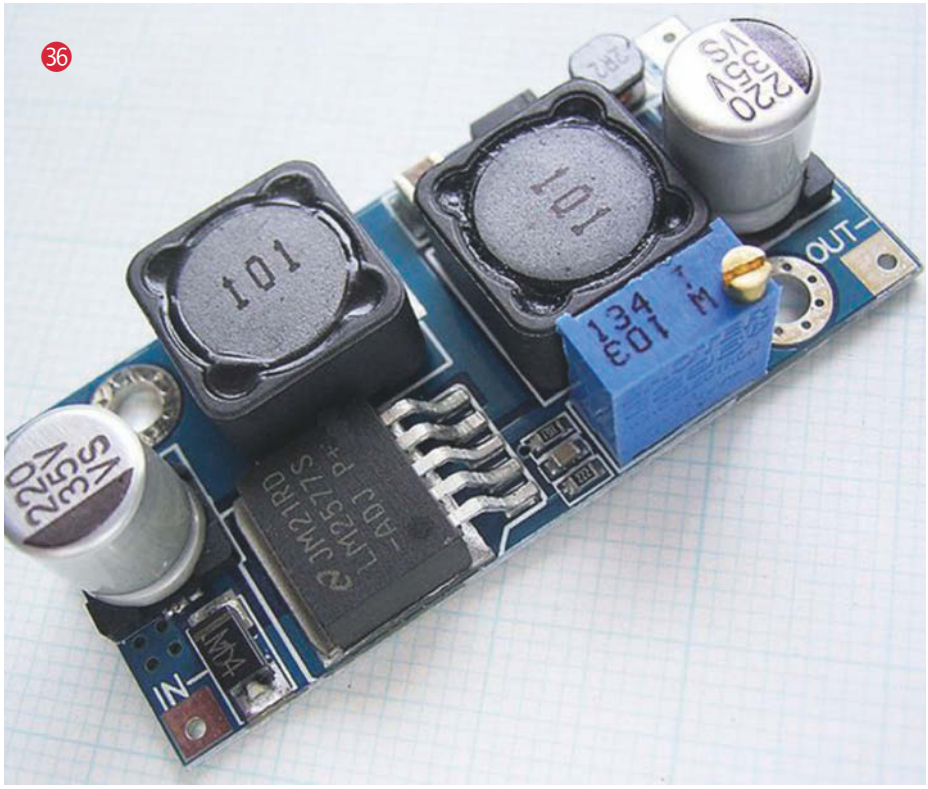
Bei batteriebetriebenen Robotern sollte man grundsätzlich die Finger von Zink-Kohle-Batterien lassen. Zum einen können sie nicht so viel Strom liefern wie Alkali-Mangan-Batterien und zum anderen sinkt ihre Spannung bei Belastung recht schnell ab. Das führt zu seltsamen Phänomenen in moderner Elektronik, die teilweise gar nicht mehr startet. Alkali-Mangan-Batterien 30 sind im Vergleich sogar auslaufsicherer und deshalb die bessere Wahl.

Alternativ greift man auf wiederaufladbare NiMH-Akkus 31 zurück, die allerdings eine pro Zelle um rund 0,2 bis 0,3 geringere Ladeschlussspannung haben als Batterien. Wo vier volle Batterien zum Betrieb bereits aus-

reichen, können 4 Akkus unter Umständen schon zu wenig sein. Bei der Konzeption eines Roboters sollte man besser vorher planen, welche Art der Spannungsversorgung man nutzen will. Als Drop-in-Ersatz funktionieren Akkus oft nicht gut und sie benötigen ein Ladegerät, die aber sehr günstig zu haben sind.

In größeren mobilen Robotern können auch Blei-Vlies-Akkus (32) als Stromversorgung dienen. Sie liefern hohe Ströme, und das vergleichsweise lang. Dafür sind sie aber erheblich schwerer, weshalb der Roboter stärkere Antriebe benötigt. Ein 12V-Akku mit 7,2Ah kostet rund 15 Euro und wiegt 2,3kg. Auch Blei-Vlies-Akkus benötigen ein Ladegerät, sind aber in der Wartung unproblematischer als NiMH-Akkus. Tiefentladungen vertragen sie aber genauso wenig, beispielsweise wenn man vergessen hat, den Roboter auszuschalten.

Viel Energie bei kleinem Gewicht erreicht man mit Lithium-Polymer-Akkus (33). Einzeln gibt es sie mit 3,7V, 7,4V, 11,1V und so weiter mit unterschiedlichen Kapazitäten. Ein Akku mit 7,4V und 7200 mAh (bei 320g) kostet jedoch schon 80 Euro, ist also deutlich teurer



37

als ein Blei-Vlies-Akku, aber auch deutlich leichter. LiPos benötigen spezielle Ladegeräte, insbesondere wenn mehrere Zellen in Reihe geschaltet sind. Sie sind auch nicht so robust und nehmen zu hohe Entladeströme übel, in dessen Folge sie sich aufblähen.

Spannungsregelung

Viele der oben beschriebenen Komponenten erwarten von ihrer Stromquelle eine Spannung von 5V, die keine Batterie und auch kein Akku liefert – egal in welcher Reihen- und Parallelschaltung man sie auch betreibt. 3,7V sind zu wenig, 12V sind zu viel. Um die Spannung herunterzuregeln, setzt man Step-down-Regler ein, um sie hochzuregeln, stattdessen Step-up-Regler.

Recht populär ist beispielsweise das Step-down-Modul DSN2596 (für unter zwei Euro auf Amazon) mit einem LM2596 34, das ohne weitere Kühlung 2A liefert. Der Eingangsbereich liegt zwischen 4 und 35 Volt, ausgangsseitig kann man das Modul zwischen 1 und 30 Volt einstellen. Dabei muss die Eingangsspannung immer mindestens um ein Volt höher als die Ausgangsspannung sein. In der Praxis betreiben wir damit einen Mini-PC vom Typ Odroid XU4 als Robotersteuerung an einem Bleiakku, dessen 12-Volt-Spannung wir über den Trimmer auf dem Wandler auf fünf Volt Ausgangsspannung eingestellt haben. Prinzipiell liefert das Wandler-Modul auch kontinuierlich 3-Ampere-Ausgangsstrom, allerdings empfehlen die Anbieter dann einen Kühlkörper für den LM2596.

Mitunter stößt man auf Module, die mit dem LM2576 oder dem LM2575 bestückt sind. Der Unterschied bei den beiden besteht in der wesentlich niedrigeren Schaltfrequenz von 52 kHz (150 kHz beim 2596), womit größere Spulen und Kondensatoren für den Ausgangsfilter notwendig sind. Wer mehr als 3A-Ausgangsstrom benötigt, sollte sich Module mit dem IC XL4001 oder dem XL4005 anschauen. Letzterer arbeitet mit einer Schaltfrequenz von 300kHz und liefert bis zu 5 Ampere Strom.

Benötigt man Step-up-Wandler, etwa um aus einer 3,7V-LiPo-Zelle fünf Volt oder eine höhere Spannung zu erzeugen, bieten sich Wandler-Module mit dem LM2577 an. Der erwartet Spannungen zwischen 3 und 34V, um daraus einen Output von 4 bis 35 Volt mit maximal 2,5 Ampere bereitzustellen. Auch hier gilt die Regel: Der Spannungsabstand zwischen Eingang und Ausgang sollte ein Volt betragen, hier allerdings in umgekehrter Richtung. Mehr Stromhunger befriedigen Modelle mit dem XL6009 35 für 3 Euro (Module auf eBay heißen dann meist DSN6009). Er liefert theoretisch bis zu 4 Ampere bei einer Schaltfrequenz von 400kHz.

Universell einsetzbar sind Step-up/Step-down-Module (auch Buck-Boost-Converter genannt), bei denen die Ausgangsspannung konstant bleibt, unabhängig davon, ob die Eingangsspannung nun darunter oder darüber liegt. Insbesondere bei Anwendungen für Akku-Betrieb, bei denen die Spannung des Akkus stärker variiert, trifft man häufig auf diese Anforderung. In der Regel beschaltet man einen normalen Step-up-Wandler (etwa LM2577/2587) mit einer zweiten zusätzlichen Induktivität (Speicherdrossel) für 3 Euro 36. Es gibt auch Step-up/down-Wander mit zwei Schaltreglern vom Typ LM2596S LM2577S mit 3A für 8 Euro.

Idealerweise verwendet man für Board und Sensoren einen eigenen Spannungsregler für 5V. Für Motoren verbaut man einen zweiten mit 6 bis 12V.

USB-Powerbanks

Will man sich das ganze Stromversorgungsgeraffel nicht antun, kauft man sich eine USB-Powerbank. Darin sind Akku, Ladeschaltung, Tiefentladungsschutz und Step-up-Konverter für 5V bereits enthalten. Gute Powerbanks 37 (ab 30 Euro) liefern auch 3A am Ausgang, womit sich locker ein Single Board Computer nebst Peripherie antreiben lässt. Unter Umständen reicht die Leistung noch, um die Motoren (bei 9–12V-Motoren mit zusätzlichem Step-up-Regler) zu versorgen. Unterstützt die Powerbank den Quick-Charge-3.0-Standard, kann sie ihren Ausgang von 5V auf 9V oder 12V umstellen. Spezielle Powerbanks für Laptops 38 können an separaten Ausgängen sogar noch höhere Spannungen liefern, kosten allerdings auch um die 100 Euro. —*dab*

38

secIT by Heise

HANNOVER 2020

25. – 26. März 2020
Hannover

Der Treffpunkt für Security-Anwender und -Anbieter!

Seien Sie dabei und profitieren Sie als Besucher von neuesten IT-Security Trends, Produkten oder Software-Lösungen.

HIGHLIGHT:
Krypto-Experte aus den USA

Bruce Schneier

Workshop und Keynote

Das erwartet Sie:

- Fachvorträge auf **2 Bühnen**
- bereits am Vortag **Schulungsseminare**
- Ausstellungsfläche auf rund **3.400 m²**
- unabhängige **redaktionelle Workshops**
- kostenfrei und fachbezogen **Partner-Workshops**
- kostenfrei und informativ **Partner-Expert-Talks**

© AdobeStock-kras99

Weitere Informationen und Anmeldung unter

sec-it.heise.de

Veranstalter: Heise Medien

organisiert von: heise Events – Conferences, Seminars, Workshops

Eventpartner: itsa 2019 – Die IT-Security Messe und Kongress

Roboter-Software und Frameworks

Mit Software haucht man seiner Roboter-Hardware Leben ein. Unterstützung bekommt man dabei von diversen freien Bibliotheken und Frameworks in verschiedenen Programmier-sprachen. Unser Artikel zeigt eine Auswahl der populärsten davon.

von Daniel Bachfeld

Kurzinfo

- »JavaScript-Frameworks für Roboter
- »Einführung in ROS
- »So programmiert man damit

Alles zum Artikel im Web unter *make-magazin.de/xc2f*

Wenn man in die Roboterprogrammierung einsteigt, sind die ersten Aufgaben noch überschaubar: Motoren steuern, Distanzsensoren lesen und gegebenenfalls ausweichen. Die Lösung ist auf einem Arduino in C++ oder einem Raspberry Pi in C++ oder Python nur wenige Zeilen lang. Doch bereits da nutzt man meist Bibliotheken, um die GPIOs anzusteuern. Der Aufruf `digitalWrite(pin, value)` ist beispielsweise eine Funktion der internen Wiring-Bibliothek der Arduino IDE. Sie ist immer automatisch eingebunden.

Daran angelehnt gibt es für den Raspberry Pi die Library „WiringPi", deren Funktionen – erwartungsgemäß – die gleichen Namen haben wir beim Arduino. Wer auf dem Raspberry Pi lieber in Python programmiert, der installiert sich WiringPi-Python und kann trotzdem die Namenskonventionen wie unter C++ beibehalten.

Bereits mit diesen Bibliotheken kann man seinem Roboter eine Menge Kunststücke und Verhalten respektive Interaktion beibringen. Zur Abfrage weiterer Sensoren oder der Steuerung anderer Aktoren bieten die Hersteller oder die Community Bibliotheken an, die man in seinen Programmen nutzen kann.

Sprachwechsel

So mancher kann sich jedoch weder mit C/C++ noch mit Python anfreunden. Alternativ kann man seine Projekte auch in JavaScript programmieren. Normalerweise kennt man JavaScript als Browsersprache für dynamische Webseiten. Doch dank des JavaScript-Frameworks Node.js kann man JavaScript ganz normal in einer Laufzeitumgebung auf dem PC (oder einem Pi) ausführen. Das Ganze ist so leistungsfähig, dass auch Desktop-Anwendungen wie die Editoren Atom und Visual Studio Code in JavaScript programmiert sind – beide auf Basis des GUI-Frameworks Electron.

Auch für Roboter gibt es spezielle JavaScript-Frameworks. Johnny Five – der Originalname des Roboters aus „Nummer 5 lebt" – ist das beliebteste, zumindest laut den Votings auf Github. Es unterstützt 40 Boards, darunter Arduino, den leider nicht mehr hergestellten Intel Edison, BeagleBone und Raspberry Pi. Johnny Five (J5) bringt viele verschiedene Module zur leichten Ansteuerung für diverse Motorarten, Umweltsensoren, Lagesensoren, LEDs, Displays, I/O, Abstandssensoren und Joysticks mit. Das Ganze ist jeweils abgestimmt auf die richtigen GPIOs des jeweiligen Boards, wobei nicht jedes Board jede Funktion unterstützt.

Das Listing „Johnny Five Motorsteuerung" zeigt exemplarisch, wie ein Skript aussieht, um einen Motor anzusteuern. Das sieht etwas ungewohnt aus, insbesondere weil JavaScript viel mit Events arbeitet. Der Befehl `motor.start()` triggert einen Event, der automatisch die Callback-Funktion `motor.on(start,…)` aufruft. Die ruft ihrerseits `motor.brake()` auf, die dann wieder `motor.on(brake,…)` triggert.

Auf Single-Board-Computern (SBC) wie dem Pi kann Johnny Five mit Nodes.js direkt laufen. Anders sieht es auf Mikrocontroller-Plattformen mit geringem Speicher aus. Node.js unterstützt diese nicht direkt. Mit Espruino und low.js gibt es zwar JavaScript-Interpreter für Embedded Systems, die bieten aber nicht den vollen Funktionsumfang wie Node.js. Um trotzdem einen Arduino mit JavaScript steuern zu können, geht J5 einen Umweg: Auf einem PC oder SBC läuft das JavaScript, das per serieller Schnittstelle dem Arduino (oder einem ähnlichen Board) Befehle sendet. Auf dem Arduino läuft eine zuvor installierte, sogenannte Firmata-Firmware, die die Befehle interpretiert und dementsprechend die GPIOs steuert oder abfragt. Die serielle Verbindung muss keine echte Kabelverbindung sein, auch per Bluetooth lassen sich Befehle senden und Daten abfragen. Damit wird es dann auch für mobile Roboter spannend.

Angriff der Zylonen

Eine interessante Alternative zu J5 stellt Cylon.js dar, das ebenfalls auf Node.js aufsetzt, reichlich Plattformen unterstützt und viele Module mitbringt. Daneben unterstützt es bereits fertige Roboter wie Rapiro sowie bluetoothfähiges Spielzeug wie die ARdrone, Bebop und Craziefly, den Kugelroboter

Johnny Five: Motorsteuerung

```
var five = require("johnny-five"),
    board = new five.Board();

board.on("ready", function() {

  var motor = new five.Motor({
    pins: {
      pwm: 3,
      dir: 12,
      brake: 9
    }
  });

  motor.on("forward", function(err, timestamp) {
    // demonstrate braking after 5 seconds
    board.wait(5000, function() {
      motor.brake();
    });
  });

  motor.on("brake", function(err, timestamp) {
    // Release the brake after .1 seconds
    board.wait(100, function() {
      motor.stop();
    });
  });

  // Start the motor at maximum speed
  motor.forward(255);

});
```

Cylon.js: Spherosteuerung

```
var Cylon = require('cylon');
Cylon.robot({
  connections: {
    sphero: { adaptor: 'sphero', port: '/dev/rfcomm0' }
  },
  devices: {
    sphero: { driver: 'sphero' }
  },
  work: function(my) {
    every((1).second(), function() {
      my.sphero.roll(60, Math.floor(Math.random() * 360));
    });
  }
}).start();
```

Gobot: BB-8

```
package main
import (
        "os"
        "time"
        "gobot.io/x/gobot"
        "gobot.io/x/gobot/platforms/ble"
        "gobot.io/x/gobot/platforms/sphero/bb8"
)
func main() {
        bleAdaptor := ble.NewClientAdaptor(os.Args[1])
        bb8 := bb8.NewDriver(bleAdaptor)
        work := func() {
                gobot.Every(1*time.Second, func() {
                        r := uint8(gobot.Rand(255))
                        g := uint8(gobot.Rand(255))
                        b := uint8(gobot.Rand(255))
                        bb8.SetRGB(r, g, b)
                })
        }
        robot := gobot.NewRobot("bb",
                []gobot.Connection{bleAdaptor},
                []gobot.Device{bb8},
                work,
        )
        robot.Start()
}
```

Artoo: Sphero

```
require 'artoo'

connection :sphero, :adaptor => :sphero, :port => '/dev/Sphero-WRW'
device :sphero, :driver => :sphero

work do
  @count = 1
  every(3.seconds) do
    sphero.set_color(@count % 2 == 0 ? :green : :blue)
    @count += 1
    sphero.roll 60, rand(360)
  end
end
```

Sphero und Smart-Home-Produkte wie Philips Hue und Nest.

Im Listing „Cylon.js – Spherosteuerung“ ist eine einfache Steuerung des Kugelroboters zu sehen, in der anonyme Funktionen (`function()`) zum Einsatz kommen. Für Arduino-Programmier ist das harter Tobak, an den man sich erst mal gewöhnen muss. Die Beschäftigung mit JavaScript lohnt aber, weil man damit nicht nur Roboter steuern, sondern auch browserbasierte Bedienoberflächen bauen kann. Besonderes Schmankerl von Cylon.js: Es unterstützt per Plug-in auch MQTT (siehe auch Artikel auf Seite 112) und Socket.io, womit sich interessante Netzwerkanwendungen entwickeln lassen.

Cylon.js hat noch zwei Schwesterprojekte: Gobot und Artoo, die beide eine ähnliche Zahl von Boards, Sensoren und fertige Roboter unterstützen. Statt JavaScript programmiert man bei Gobot jedoch in der von Google-Mitarbeitern erfundenen Sprache Go. Wie bei J5 kommuniziert Gobot mal per Firmata-Firmware, mal per Bluetooth, mal direkt mit den GPIOs.

Das Listing „Gobot: BB-8“ kugelt den Star-Wars-Roboter BB-8 in der Gegend rum und sieht aus wie eine Mischung zwischen C/C++ und Pascal. Ungewöhnlich sind die eckigen Klammern vor einigen Variablen. Sie kennzeichnen sogenannte Slices, eine Art Array, deren Länge aber zur Laufzeit veränderlich ist.

Das Projekt Artoo (R-Zwo) dreht sich hingegen um die Programmiersprache Ruby. Listing „Artoo: Sphero“ kommuniziert per Bluetooth mit einem Sphero, lässt ihn die Farbe wechseln und durch die Gegend rollen.

Level up!

Die bislang beschriebenen Frameworks bringen zwar Unterstützung für Hardwarekomponenten mit, wie dann aber die jeweiligen Sensordaten oder Fortbewegungsmöglichkeiten genutzt werden, bleibt immer noch dem Programmierer überlassen. Der muss dann mühselig selbst einzelne Aktionen implementieren. Spezielle Frameworks vereinfachen jedoch die Kontrolle über Hardware und helfen bei der Implementierung von Verhalten und Interaktion.

Das führende freie Framework ist das Robot Operating System, kurz ROS. Es ist der Standard in Industrie und Forschung schlechthin und hat auch unter Hobby-Robotikern eine große Community. Es ist kein Betriebssystem im eigentliche Sinne, sondern eher eine Middleware, die spezielle Tools, Dienste und Funktionen bereitstellt. ROS läuft grundsätzlich nur auf Systemen, auf dem bereits ein Betriebssystem wie Linux läuft. In Frage kommen also nur PC, NUCs oder SBCs mit Intel- oder ARM-Prozessoren.

ROS beruht auf einer Paketverwaltung, mit der sich Pakete für verschiedene Treiber und Funktionen installieren lassen und deren Paketabhängige dabei berücksichtigt werden. ROS bringt schon Treiber für viele spannende Hardwarekomponenten mit, darunter Laserscanner, 3D-Kameras, Dynamixel-Servos, inertiale Navigationssysteme, GPS, Motoren und so weiter.

Viele andere Pakete enthalten Programmmodule, die sich um die sinnvolle Verarbeitung der Daten kümmern. Beispielsweise können Kamerabilder direkt zur Erkennung von Gesichtern, von grafischen Tags, Umgebungen und Objekten genutzt werden. Pakete für Laserscanner können etwa die Daten

ROS-Talker

```
#!/usr/bin/env python
import rospy
from std_msgs.msg import Int32
rospy.init_node('topic_publisher')
pub = rospy.Publisher('counter', Int32)
rate = rospy.Rate(2)
count = 0
while not rospy.is_shutdown():
    pub.publish(count)
    count += 1
    rate.sleep()
```

ROS-Listener

```
#!/usr/bin/env python
import rospy
from std_msgs.msg import Int32
def callback(msg):
    print msg.data
rospy.init_node('topic_subscriber')
sub = rospy.Subscriber('counter', Int32, callback)
rospy.spin()
```

visualisieren oder in Kombination mit einem Mapping-Paket die Umrisse eines oder mehrerer Räume aufzeichnen, speichern und sofort für die weitere Navigation verwenden. Robotiker nennen das Simultaneous Localization and Mapping (SLAM), an deren Prinzipien Forscher jahrelang geforscht haben. Mit ROS bekommt man viele solcher coolen Pakete kostenlos frei Haus.

Dank SLAM kann man dem Roboter später auch den Befehl geben, an einen bestimmten Ort in der Karte zu fahren. Mit Pfadplanungspaketen berechnet er den günstigsten Weg und fährt los. Neuen, bislang nicht kartierten Hindernissen weicht er einfach aus und korrigiert seinen Pfad zum Ziel.

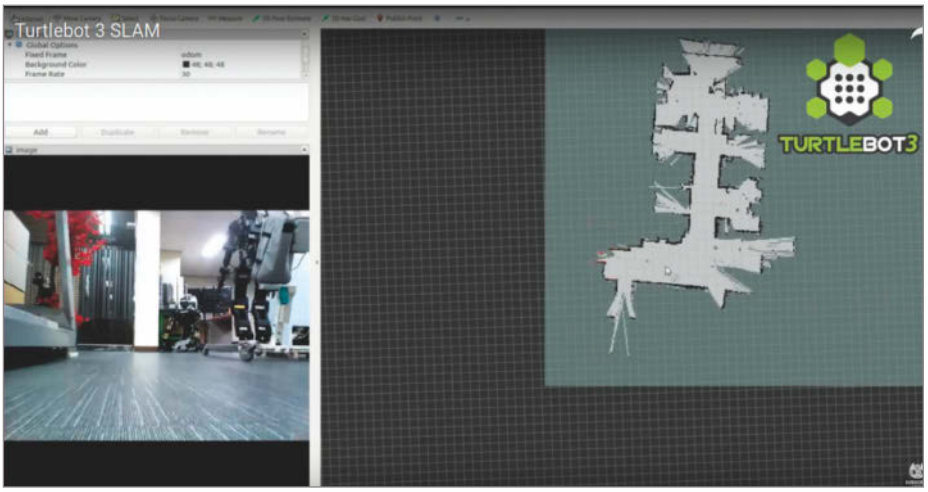
Listen up!

ROS-Module lesen die Daten nicht direkt aus der Hardware aus. Vielmehr stellen die Hardwaretreiber die Daten zur Verfügung. In ROS spricht man dabei von einem Publisher (Talker), der seine Daten in einem Channel veröffentlicht. Ein Subscriber (Listener), also ein Modul, abonniert den Channel und kann daraus die Daten lesen. Für die Kommunikation nutzt ROS sowohl lokal als auch im LAN TCP/IP.

Wer welchen Channel betreibt, erfahren Subscriber vom ROS Master. Er verwaltet alle Channels, die im ROS-Jargon Topics heißen, und gibt den Subscribern quasi die Kontaktdaten heraus, also IP-Adresse oder Name sowie den Netzwerkport. In weiteren kommunizieren Talker und Listener respektive Publisher und Subscriber direkt. ROS unterstützt C++ und Python und bietet Bibliotheken, die die Kommunikation zwischen beiden in wenige Befehle kapseln.

Das Listing ROS-Talker ist ein Proof of Concept eines Talkers in Python, der zweimal pro Sekunde den Wert `count` erhöht und mit `pub.publish(count)` unter dem Topic „count“ veröffentlicht. Listing ROS-Listener abonniert das Topic mit `sub = rospy.Subscriber('counter', Int32, callback)`. Die Callback-Funktion „callback“ wird bei jeder Veröffentlichung neuer Daten auf dem Topic aufgerufen.

Die jeweils laufenden Programme nennt man Nodes. Das ROS-Modell vereinfacht es ungemein, einzelne Funktionen in einzelnen Nodes zu implementieren. Die ganze Verwaltung, Koordination und Kommunikation er-



Ein Turtlebot 3 erstellt während der Fahrt mittels Laserscanner eine Karte seiner Umgebung.

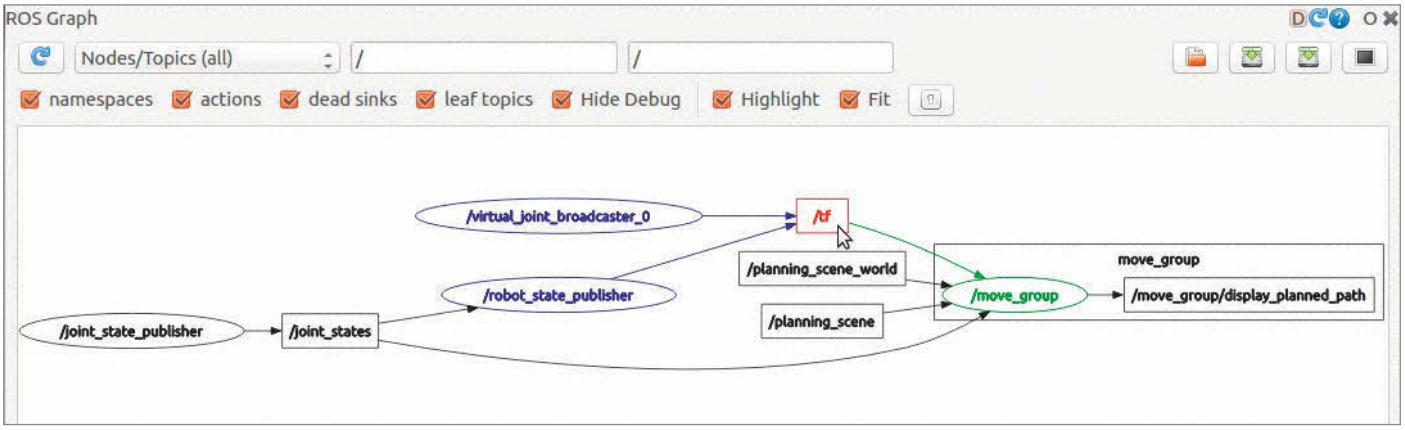


Bild: ROS Wiki

Das Tool rqt_graph zeigt alle Topics und die Zusammenhänge zwischen Publisher und Subscriber an.

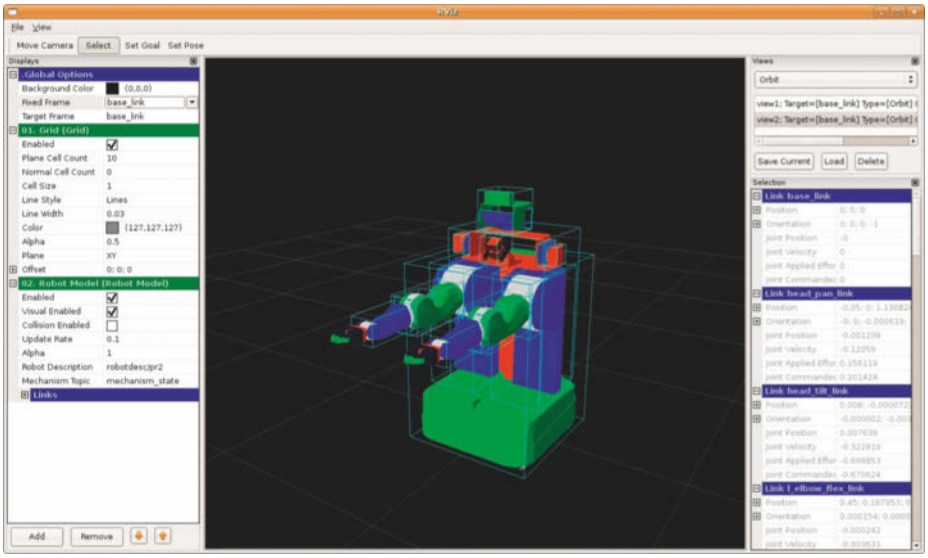


Bild: ROS Wiki

Rviz zeigt Sensordaten und Robotermodelle an.

fordert allerdings einiges an Ressourcen, was auch der Kritikpunkt an ROS ist. Es ist nicht echtzeitfähig und es läuft nicht ohne weiteres auf embedded Systemen. Das seit einigen Jahren entwickelte ROS2 soll diese Probleme lösen, doch so richtig kommt man dort nicht voran.

Tools

In der Praxis können ROS-Pakete sowohl Talker als auch Listener enthalten. Ein Modul zur Motorsteuerung kann als Listener auf Geschwindigkeitsbefehle anderer Talker horchen und als Talker die Odometriewerte der Encoder veröffentlichen. Letztere lassen sich mit den Daten von Lagesensoren beispielsweise über das Paket *robot_pose_ekf* fusionieren, um eine genauere Positionsbestimmung zu erhalten.

Die für den jeweiligen Roboter benötigten Nodes lassen sich samt Parameter praktischerweise in einer XML-Datei auflisten und per `roslaunch`-Befehl koordiniert starten. Vieles erledigt man bei ROS auf dem empfohlenen Ubuntu auf der Kommandozeile. Es stehen dabei eine Reihe an grafischen Tools zur Verfügung, mit denen sich die Arbeit des Roboters überwachen, steuern und simulieren lässt. *Rqt_graph* visualisiert alle Talker und welche Listener am jeweiligen Topic mithorchen.

Rviz (ROS visualization) ist ein 3D-Visualisierer, mit dem man sich Sensorendaten, etwa von Laserscannern oder 3D-Kameras, anzeigen lassen kann. Diverse Plug-ins unterstützen weitere Datenquellen. Zudem macht rviz die Position und Ausrichtung verschiedener Komponenten der Robotermodelle sichtbar, sofern man sich die Mühe gemacht hat, ein 3D-Modell des Roboters zu erstellen.

Apropos Modell: Wer sich keinen echten Roboter bauen kann oder will, kann mit einem 3D-Modell in dem Simulator Gazebo erste Roboter-Erfahrung sammeln. Gazebo ist zwar kein ROS-Projekt, aber im Framework enthalten. ROS kann die Roboter im Simulator steuern und simulierte Sensordaten abfragen und in rviz visualisieren.

Ausblick

Wer mit seinem Roboter mehr machen will, als nur Hindernissen auszuweichen, der sollte sich ROS näher ansehen. Das Framework hat zwar eine steile Lernkurve, dafür erhält man aber ein Füllhorn an fertigen Lösungen, die man alleine nie gestemmt hätte. Die Community ist recht groß und ähnlich wie auf Stackoverflow gerne bereit, bei Problemen zu helfen. *—dab*

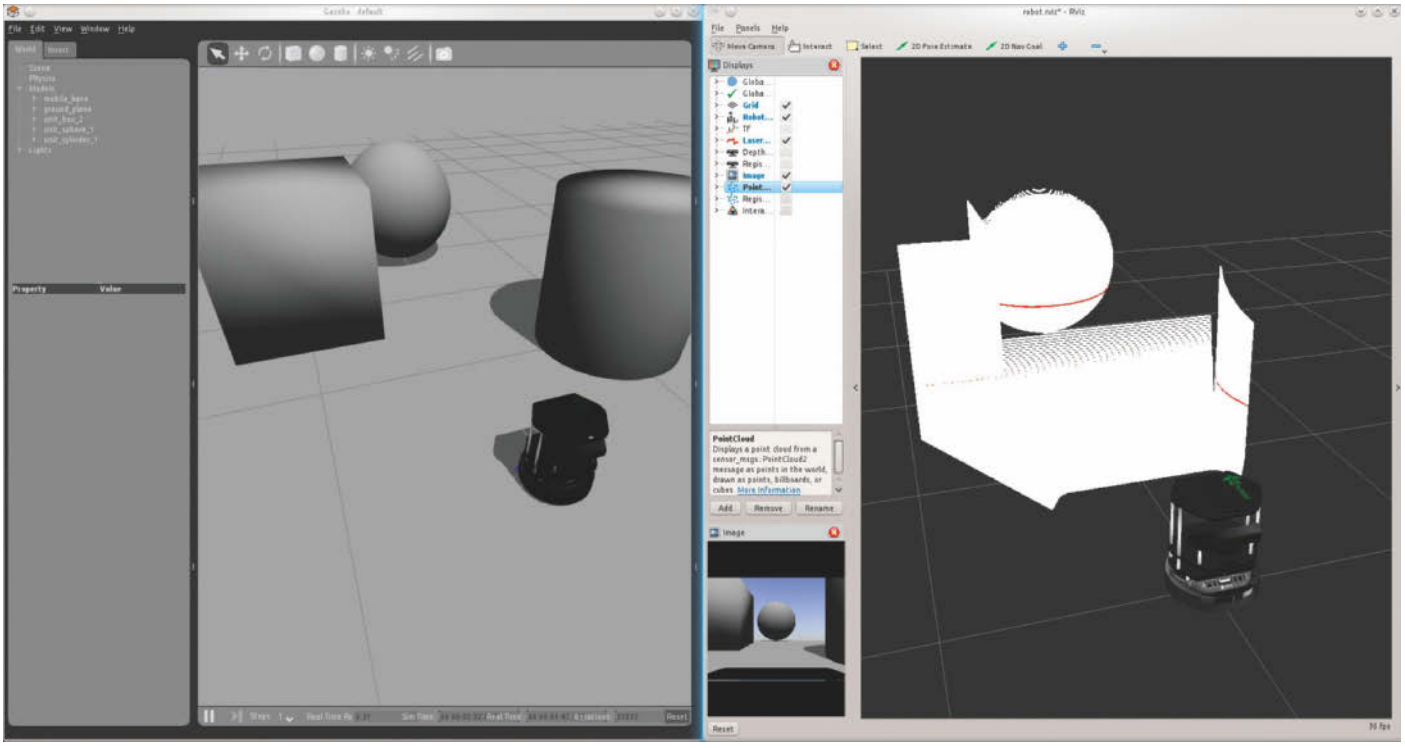


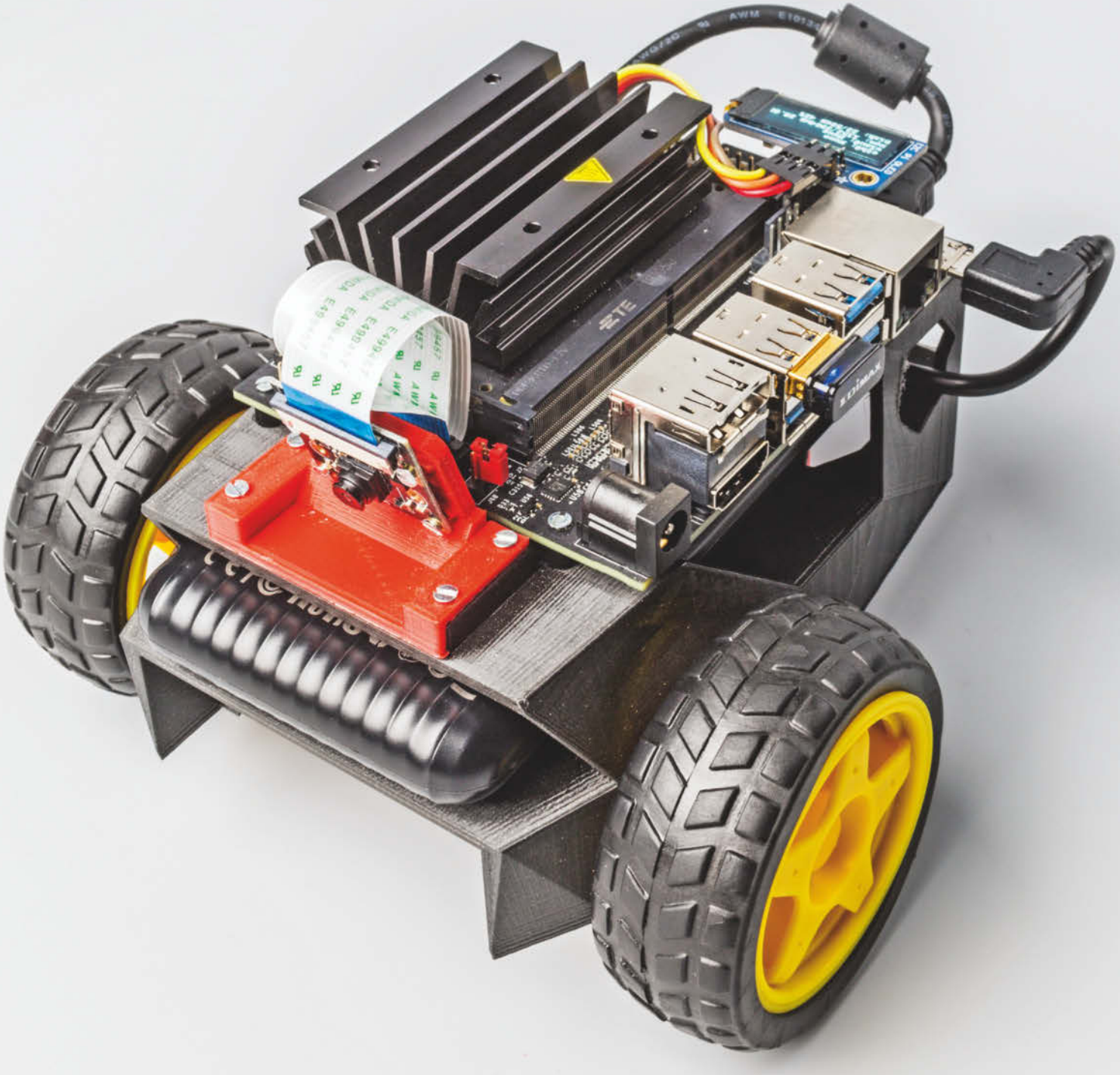
Bild: ROS Wiki

Links fährt ein Turtlebot durch eine simulierte Landschaft in Gazebo, rechts visualisiert rviz die vom Roboter gesehene Landschaft.

JetBot

Der JetBot ist dank seines leistungsfähigen Prozessors nicht nur ein KI-fähiger Roboter, sondern mit Hilfe seiner Software auch ein Lernsystem. Und zwar ein recht komfortables und leicht verständliches.

von Heinz Behling



Roboter-Programmierung gilt als schwierig, in Zusammenhang mit Künstlicher Intelligenz (KI) sogar als für Laien kaum beherrschbar. Der Selbstbau-Roboter JetBot macht Schluss mit diesen Vorurteilen. Er ist nämlich nicht nur einfach zusammenzubauen, sondern bringt ein Software-Paket mit, das auch Einsteigern richtig Freude macht und schnell erste Erfolge erzielen lässt.

Seinen Namen verdankt er übrigens dem Steuercomputer, einem Jetson nano (1). Dieser Computer enthält einen Grafik-Prozessor des Herstellers Nvidia, dessen zahlreichen Rechenkerne ideal sind für ein künstliches neuronales Netz. Das wird später benutzt, um Objekte zu erkennen. Der Jetson hat die Bauform eines Notebook-Speicherstreifens uns sitzt auf einer kleinen Platine, dem Jetson nano Developer Kit. Das stellt die notwendigen Anschlüsse für USB, Kamera und Display sowie die Spannungsversorgung zur Verfügung.

Die Steuerung des JetBot erfolgt über eine Web-Oberfläche, die vom lokal auf dem Jetson laufenden Webserver bereitgestellt wird. Die Programmierung erfolgt in der Programmiersprache Python. Mitgeliefert werden eine Reihe von Beispielen, die in sogenannten Notebook-Dateien (die haben nichts mit tragbaren Computern zu tun, sondern eher mit Notizblöcken) Schritt für Schritt jeden einzelnen Befehl erklären. Der Clou: Die in den Notebooks enthaltenen Programmteile können auch gleich aus dem Anleitungstext heraus an den JetBot geschickt und dort ausgeführt werden. Sogar Änderungen sind in den Befehlen möglich und deren Auswirkungen können sofort überprüft werden. Doch dazu später mehr.

Kurzinfo

- KI-fähigen mobilen Roboter selbst bauen
- Neuronales Netz zur Objekterkennung
- Weitere Netz-Modelle

Checkliste

Zeitaufwand:
zwei Stunden (ohne Zeit für 3D-Druck)

Kosten:
100 Euro zzgl. ca. 130 Euro für Jetson Nano Developer Kit

Löten:
einfache Lötarbeiten

Programmieren:
Python

3D-Druck:
downloadbare Dateien mit PLA drucken

Alles zum Artikel im Web unter
make-magazin.de/xbat

Material

- **Jetson Nano Developer Set**
- **Mikro-SD-Speicherkarte** mit mindestens 32GB
- **2 TT-Getriebemotoren** mit Rädern 65mm breit
- **Kameramodul** LI-IMX219-MIPI-FF-NANO
- **Motortreiber** ADA2927
- **Powerbank 10000mAh** Ausgang 5V/3A (Maße beachten: 138 × 68 × 15mm, siehe Bezugsquellen)
- **PiOLED-Display** I²C, 128 × 32 Pixel
- **2 USB-Kabel** USB-A- auf Mikro-USB-Stecker, gewinkelt, 22cm lang
- **PIN-Leiste** gewinkelt 90°
- **USB-Dongle** Edimax EW-7811Un
- **Schrauben und Muttern** M2, M3
- **Rollkugel** 1 Zoll
- **Druckfilament** für Chassis

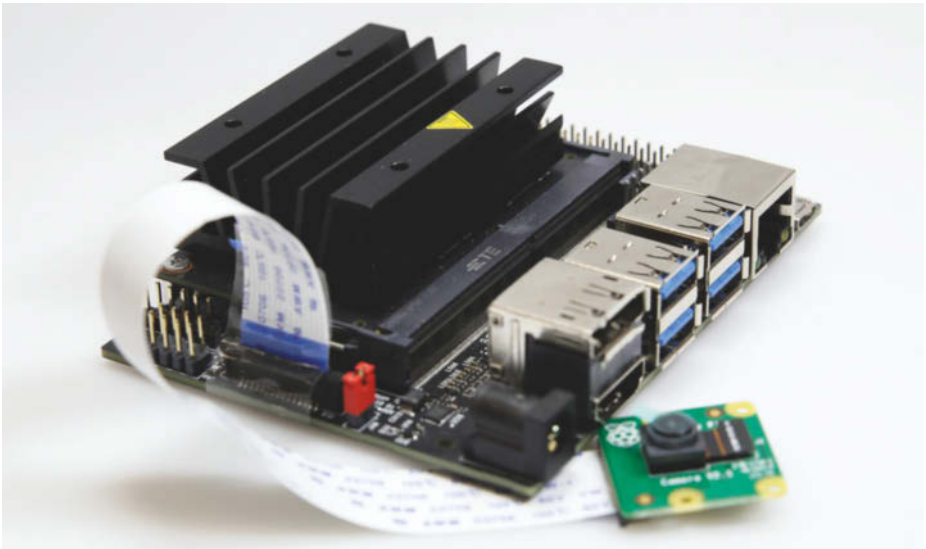
Aufbau

Zunächst müssen Sie die Hardware des kleinen Roboters zusammenbauen. Auf github (Adresse dazu und Bezugsquellen für alle Teile siehe Kurzinfo-Link) können Sie die Projektdateien downloaden. Sie enthalten auch die 3D-Druckdateien für das Chassis des JetBot. Falls Sie keinen Zugang zu einem 3D-Drucker haben, lesen Sie den Textkasten „Universal-Chassis als Grundlage“ auf Seite 45.

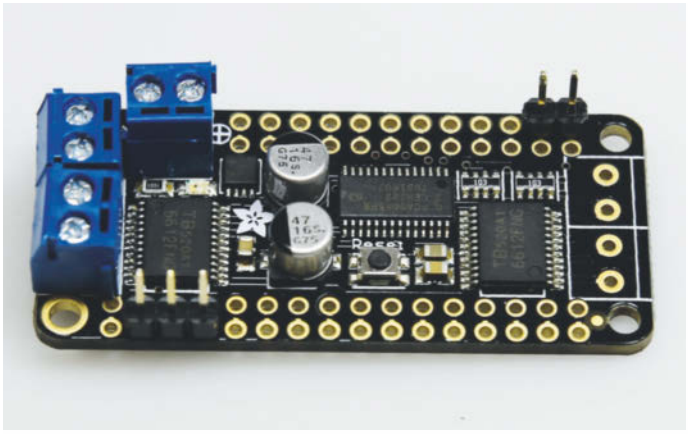
Wenn Sie das Chassis selbst drucken, müssen Sie zunächst den Durchmesser der Antriebsräder wissen. Das in den Bezugsquellen angegebene Rad-Motor-Set hat 65mm-Räder. Dementsprechend müssen Sie die passenden Teile für das vordere Stützlager und den Kamerahalter drucken. Die Dateinamen weisen jeweils auf den Raddurchmesser hin. Falls Sie die falschen Dateien nehmen, steht der Roboter später schief und die Kamera guckt zu tief nach unten.

Über den Kurzinfo-Link finden Sie auch die Adresse des JetBot-Wiki. Es enthält eine ausführliche Anleitung zum Aufbau der Hardware und zur Software-Installation sowie weitere nützliche Tipps. Das Wiki ist in leicht verständlichem Englisch und bebildert. Wir drucken daher hier keine komplette Bauanleitung, sondern geben nur einige Tipps zum Aufbau.

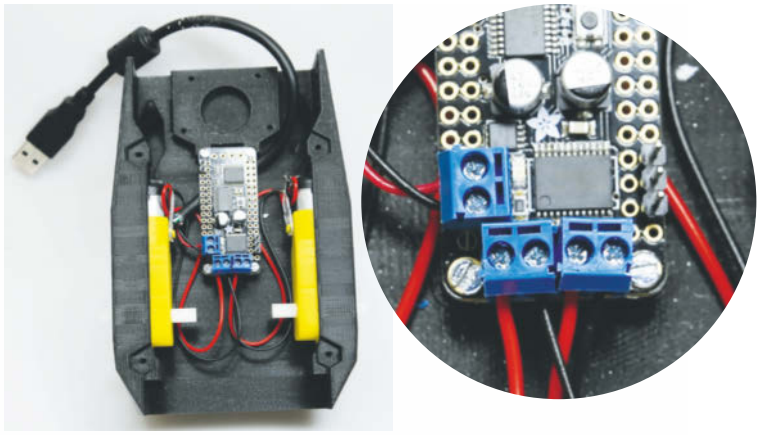
Falls die beiden Getriebemotoren ohne Anschlussdrähte geliefert werden, löten Sie an jede der beiden Lötfahnen der Triebwerke etwa 15cm Litze an, idealerweise in unterschiedlichen Farben, damit Sie beim Anschluss an die Treiberplatine die Polung unterscheiden können.



(1) Das Jetson Nano Developer Kit kann in diesem Roboter ohne Lüfter betrieben werden, da seine Leistungsaufnahme per Software gedrosselt wird.



2 Nur diese Anschlussklemmen und Pin-Leisten müssen auf der Motortreiber-Platine eingelötet werden. Lassen Sie neben der dreipoligen Leiste einen Anschluss frei.



3 Die Verkabelung von Motortreiber und Motoren ist einfach. Wichtig ist, dass die Motoren parallel zu den Gehäusekanten sitzen, sonst fährt der JetBot später nur Kurven.

In die Motortreiber-Platine müssen Sie Anschlussklemmen und Pin-Leisten einlöten (2). Da der JetBot mit nur zwei Motoren arbeitet, reichen die im Bild gezeigten Anschlüsse aus.

Die Anschlussklemmen an der kurzen Seite der Treiberplatine sind für die Motoranschlüsse vorgesehen. Motoren und Platine setzen Sie dann ins Chassis ein (3). Die Platine wird mit selbstschneidenden 2mm-Schrauben oder normalen 2,5mm-Schrauben befestigt. Die Motorenbefestigung erfolgt mit je zwei M3×30-Schrauben. Wichtig: Die Motoren müssen parallel zur Chassis-Kante sitzen. Falls nicht, säubern Sie das Chassis von eventuellen Resten einer Stützstruktur.

An die seitliche zweipolige Anschlussklemme der Treiberplatine werden die rote und schwarze Ader eines USB-Kabels angeschlossen wie im Bild zu sehen. Darüber erfolgt die Stromzufuhr. Alle Kabel verlegen Sie so, dass sie später nicht am Boden oder den Rädern schleifen können.

Dann stecken Sie die Teile des Kugellagers zusammen (4) und schrauben es ans Chassis (5). Achten Sie darauf, dass es nach dem Anschrauben leicht rollen kann. Andernfalls müssen Sie die Anlagefläche des Lagers nacharbeiten.

Das kleine Display, das später die IP-Adressen des JetBot anzeigt, wird über den I²C-Bus gesteuert. Der muss aber auch an die Motortreiber-Platine weitergeleitet werden. Daher ist noch eine gewinkelte PIN-Leiste (2 × 3 Pins) an das Display zu löten (6). Die Kabel für den I²C-Bus schließen Sie dann an die Treiberplatine (7) und das Display (8) an.

Zuletzt schrauben Sie das Jetson nano Developer Kit auf das Chassis, stecken das Display auf und schließen die Kamera an. Befestigen Sie die Kamera am Halter (9). Wichtig: Damit die Stromversorgung des Jetson über die USB-Buchse erfolgt, muss der Jumper neben dem Kameraanschluss entfernt werden. Jetzt fehlt nur noch die Powerbank. Sie kommt ins Chassis unter den Jetson. Schließen Sie die USB-Kabel des Jetson und der Treiberplatine aber noch nicht an, denn dem Jetson fehlt ja noch die Software.

Software-Installation

Auch die Software-Installation ist im Wiki gut beschrieben. Das Betriebssystem des Jetson (basierend auf Ubuntu) müssen Sie auf eine Mikro-SD-Karte speichern. Lediglich bei der Software-Aktualisierung im Punkt 5 sind Abweichungen möglich. Hier hilft vor dem `rsync`-Befehl ein

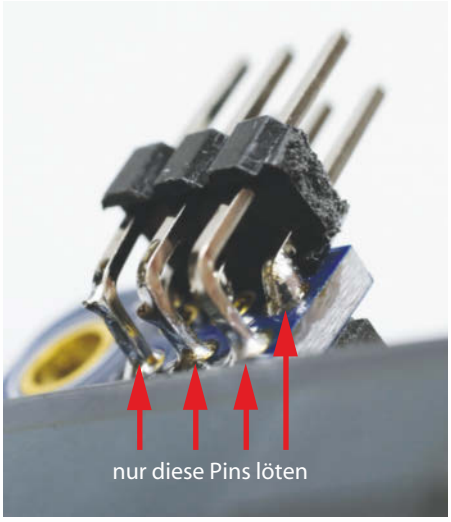
```
sudo apt-get update
```

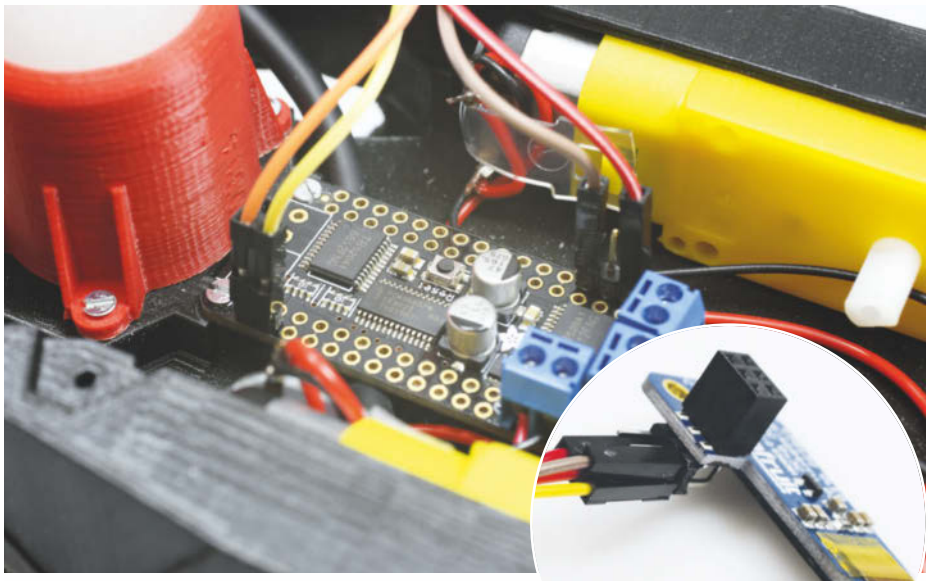
Vergessen Sie auch nicht, die WLAN-Verbindung und den Stromsparmodus einzustel-

4 In dieser Reihenfolge müssen Sie die Teile des vorderen Kugellagers zusammenstecken …

5 … und am Chassis anschrauben.

6 An den Display-Anschluss muss eine gewinkelte Pin-Leiste gelötet werden. Nur die markierten vier Pins müssen gelötet werden, sie sind die I²C-Schnittstelle für den Motortreiber.





7 Die vier Drähte für den I²C-Bus schließen Sie am Motortreiber so an …

8 … und am Display so.

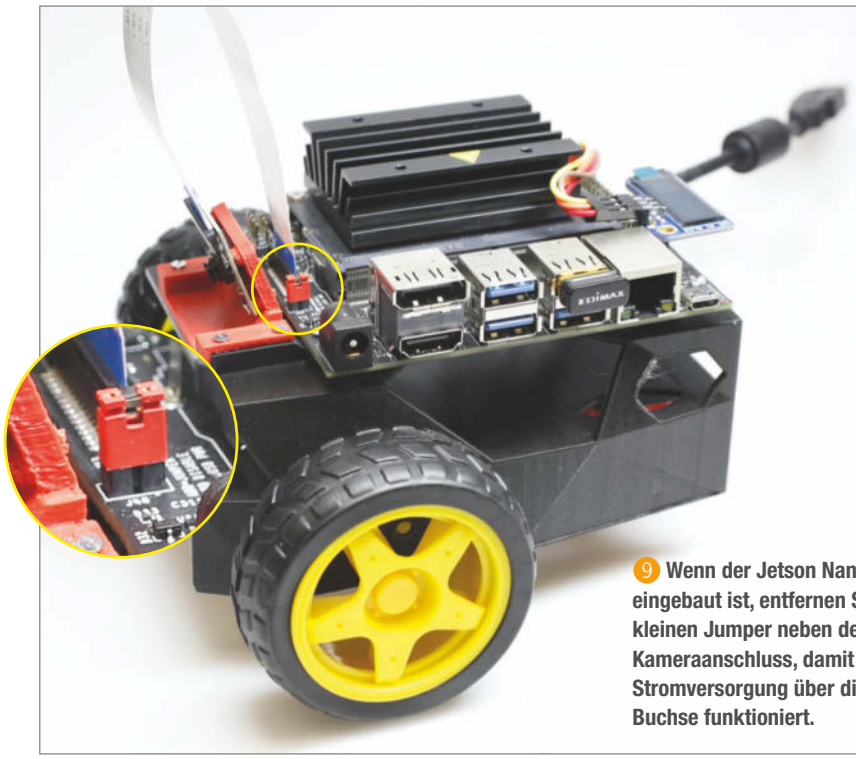
len, sonst wird die Powerbank schnell leer sein. Die WLAN-Einrichtung nehmen Sie am JetBot vor, daher müssen Sie Monitor, Tastatur und Maus anschließen.

Web-Oberfläche

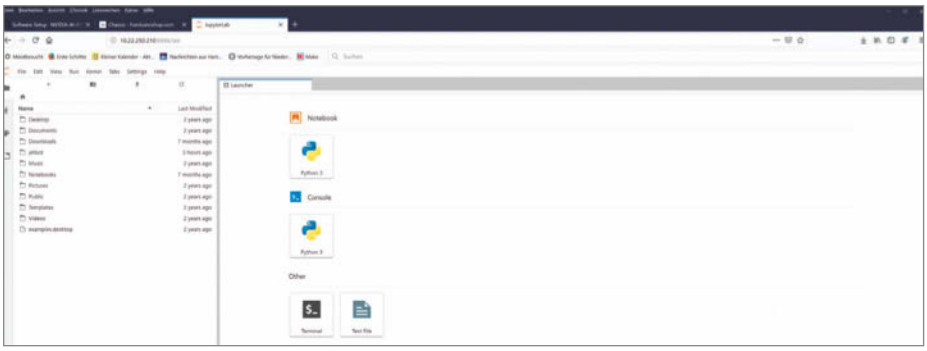
Damit ist der JetBot auch schon einsatzbereit für Ihre und seine ersten Schritte. Verbinden Sie sich von einem Web-fähigen Gerät im gleichen WLAN wie der JetBot. Geben Sie dazu im Browser diese Adresse ein:

```
http://XXX.XXX.XXX.XXX:8888
```

für `XXX.XXX.XXX.XXX` setzen Sie die IP-Adresse des JetBot ein, die er etwa ein bis zwei Minuten nach dem Start auf seinem Display anzeigt. Nach kurzer Zeit werden Sie nach dem Passwort gefragt. Es lautet `jetbot`. Anschließend erscheint die *JupyterLab* genannte



9 Wenn der Jetson Nano eingebaut ist, entfernen Sie den kleinen Jumper neben dem Kameraanschluss, damit die Stromversorgung über die USB-Buchse funktioniert.



⑩ Die sehr aufgeräumte Web-Oberfläche des JetBot

⑪ Nur wenn der blaue Markierungsbalken links neben einem Code-Block steht, kann der ausgeführt werden.

Oberfläche ⑩. Der linke Teil ist der File-Browser. Hier erscheinen Verzeichnisse und Dateien, die auf dem Jetson gespeichert sind. Im Verzeichnis *Notebooks* liegen die Beispielprojekte. Wählen Sie dort *basic_motion.ipynb*.

Im rechten Teil des JupyterLab wird nun das Dokument angezeigt. Es handelt sich nicht um ein einfaches Textdokument, sondern es enthält außerdem ausführbaren Programm-Code (grau hinterlegt). Wenn Sie mit der Maus oder den Cursortasten den blauen Markierungsbalken am linken Rand auf solch einen Code-Block setzen, können Sie die Befehle bei gleichzeitigem Drücken von STRG und Enter ausführen lassen ⑪

Beginnen Sie einfach: Lesen Sie die Anleitung und führen Sie die Code-Blocks aus. Achtung: Der dritte Block setzt den JetBot in Bewegung! Beim neunten Block wird Ihnen etwas auffallen: Auch sämtliche Anzeigen, die durch den Programmcode ausgelöst werden, erscheinen im Notebook, zum Beispiel die Schieberegler ⑫.

⑫ Alle Programmausgaben wie etwa die Schieberegler, aber auch Fehlermeldungen erscheinen direkt unter dem jeweiligen Programmblock.

Die Befehle der Code-Blöcke lassen sich auch ändern: Einfach hineinklicken und die Änderungen eingeben. So können Sie schnell kontrollieren, was das jeweils bewirkt. Spielen Sie einfach etwas herum, so lernen Sie schnell die Bedeutung der einzelnen Anweisungen.

Über das File-Menü können Sie auch selbst eigene Notebooks anlegen und darin Text und Programmcode einsetzen. Sie haben also Raum für viele Experimente.

Trainieren

Kommen wir nun aber einen Schritt weiter zum autonomen Roboter und bringen ihm das Ausweichen vor Hindernissen bei. Computer sind zwar dumm, aber heutzutage zumindest lernfähig. Deshalb kann und muss der JetBot zunächst trainiert werden, um Objekte zu erkennen, beispielsweise Hindernisse oder freien Bewegungsraum. Dieses Training erfolgt in zwei Teilen: der erste besteht im Anfertigen etlicher Fotos von Hindernissen und unbedenklichen Freiräumen. Es kommt hier vor allem auf eine möglichst hohe Anzahl von Fotos aus unterschiedlichen Perspektiven an. Diese Fleißarbeit müssen Sie mit Hilfe der JetBot-Software erledigen, und zwar mit dem *Notebook* namens *data_collection.ipynb*.

Bevor Sie den ersten Programm-Code darin aufrufen, starten Sie zunächst den dazugehörenden Kernel des JetBot erneut ⑬. Andernfalls kann es zu Fehlermeldungen kommen.

Lassen Sie nun die ersten fünf Code-Blöcke des Notebooks ausführen, also den Markierungsbalken daneben setzen, dann STRG und Enter drücken. Falls der jeweilige Code-Block eine längere Zeit zur Ausführung braucht, erscheint in der eckigen Klammer davor zunächst ein Sternchen. Ist der Code abgearbeitet, wechselt das zu einer Nummer, 1 für den ersten, 2 für den zweiten ausgeführten Code-Block und so weiter.

Unterhalb des fünften Code-Blocks sehen Sie dann das Kamerabild und je einen roten und grünen Button ⑭. Setzen Sie den JetBot nun auf dem Gelände aus, in dem er sich künftig zurechtfinden soll, beispielsweise einen Tisch mit verschiedenen Gegenständen. Anschließend nehmen Sie Fotos auf: Gegenstände und die Tischkanten nehmen sie mit einem Klick auf den roten Button, freie Flächen nehmen Sie mit dem grünen auf. Achten Sie darauf, alle Hindernisse aus möglichst vielen verschiedenen Richtungen aufzunehmen. Das erhöht die Erkennungsgenauigkeit. Sie sollten jeweils etwa 100 Bilder aufnehmen. Den JetBot bewegen Sie dabei von Hand, deswegen ist dies eine Fleißarbeit. Wenn Sie eine ausreichende Bilderzahl erreicht haben,

führen Sie den sechsten Code-Block aus. Damit werden alle Bilder in einem ZIP-Archiv gespeichert.

Der zweite Teil des Trainings beschäftigt vor allem den Jetson nano, denn nun muss das neuronale Netz dort die Bilder auswerten. Da der Leistungsbedarf des Jetson hierbei recht hoch ist, sollten sie ihn nicht über den Akku, sondern per Netzteil mit Energie versorgen (Jumper nicht vergessen!).

Das Training geschieht mit Hilfe des Notebooks *train_model.ipynb*. Falls das nicht angezeigt wird, finden Sie es im Verzeichnis jetbot/notebooks/collision_avoidance.

Lassen Sie hier alle neun Code-Blöcke ausführen. Es kann eine ganze Weile dauern, bis der Jetson fertig ist. Danach finden Sie eine Datei namens *best_model.pth* im Notebooks-Verzeichnis. Sie enthält die zur Objekterkennung notwendigen Daten.

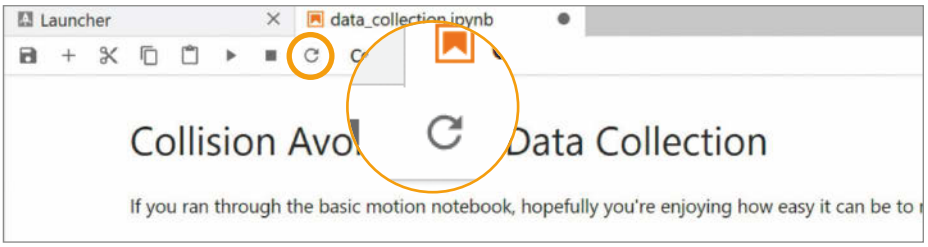
Autonome Bewegung

Richtig interessant ist der kleine JetBot aber erst, wenn er sich autonom bewegt. Er hat ja nun auch alles, was er dazu braucht. Entfernen Sie nun die Netzteilverbindung und den Jumper. Nach dem Neustart verbinden Sie sich wieder mit dem JetBot und rufen das Notebook *live_demo.ipynb* auf. Lassen Sie die ersten acht Code-Blöcke darin ausführen. Einige davon brauchen bis zu einer Minute, bis sie abgearbeitet sind. Der Grund: Jetzt wird auf dem Jetson nano das neuronale Netz für die Objekterkennung entsprechend dem vorherigen Training installiert. Da hierbei eine Menge Daten in den Speicher der GPU zu transportieren sind, dauert das eine Weile.

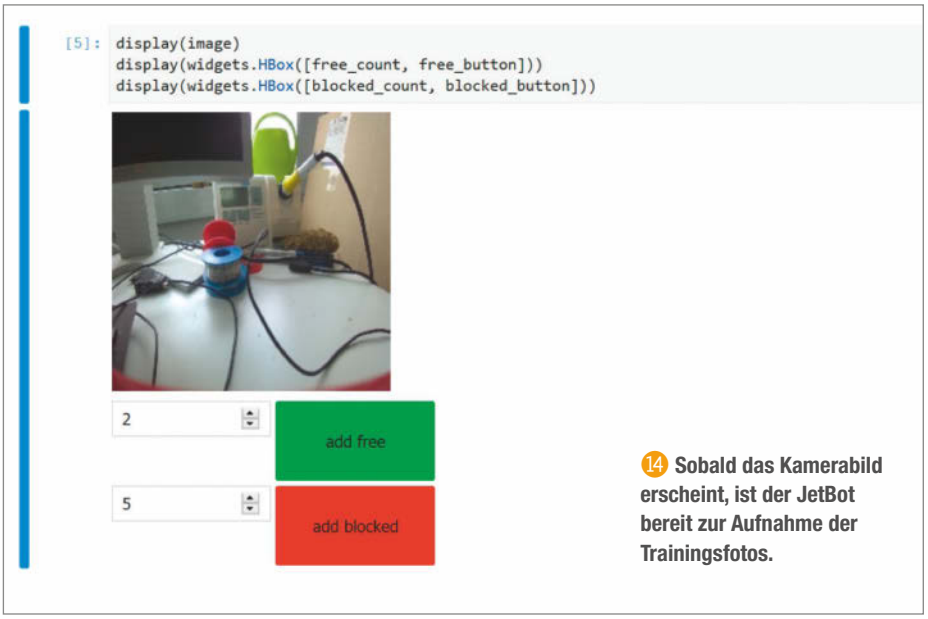
Schließlich aber wird sich der JetBot in Bewegung setzen. Falls er auf ein Hindernis trifft, weicht er ihm mit einer Links-Kurve aus. Übrigens: Sollte er sich nicht von der Stelle bewegen, weil seine Motoren nicht genug Kraft entwickeln und nur leise summen, können Sie mehr Gas geben. Setzen Sie im siebten Code-Block in den Klammern hinter `roboter.forward(0.x)` und `roboter.left(0.x)` für x größere Werte ein (etwa 6 bis 8).

Während der Roboterbewegung wird übrigens das Kamerabild laufend an den Browser übertragen. Das Kamerafenster finden Sie im angezeigten Notebook etwas weiter oben.

Dies ist nur ein kurzer Anriss der Möglichkeiten, die im JetBot stecken. Inzwischen gibt es im Internet schon eine ganze Anzahl an Jet Bot-Programmierern. Einige interessante Adressen dazu finden Sie über den Kurzinfo-Link. Ansonsten haben Sie jetzt das Grundwissen, um selber mit dem Roboter zu experimentieren. Viel Spaß dabei! *—hgb*



⑬ **Bevor Sie den Programm-Codes eines neuen Notebooks starten, setzen Sie den dazugehörenden Kernel mit einem Klick auf dieses Symbol zurück.**

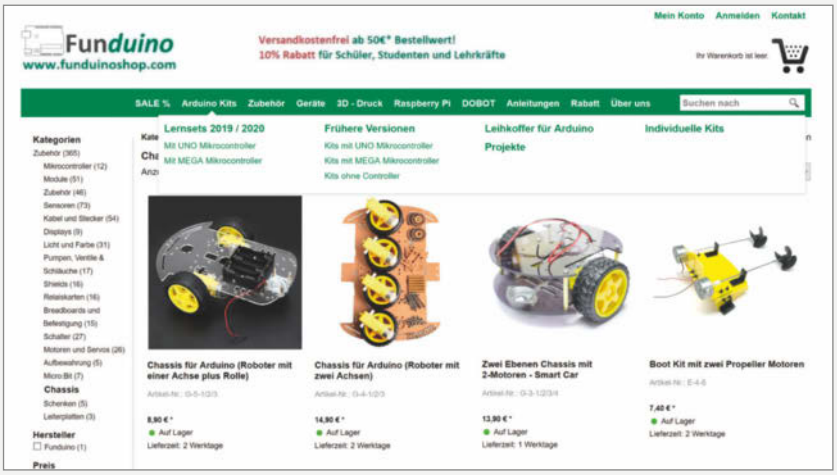


⑭ **Sobald das Kamerabild erscheint, ist der JetBot bereit zur Aufnahme der Trainingsfotos.**

Universal-Chassis als Grundlage

Falls Sie das Chassis des Roboters nicht selber drucken möchten, können Sie auch ein preiswertes Universalchassis verwenden.

Die gibt es im Set für unter zehn Euro bei diversen Internet-Shops. Zur Montage der einzelnen Komponenten des JetBots enthalten sie bereits zahlreiche Montagelöcher. Fehlende lassen sich aber auch schnell nachbohren.



Es muss nicht immer gleich 3D-Druck sein. Auch solche preiswerten Chassis sind brauchbar, erfordern aber eventuell etwas Bohrarbeit.

B-Robot EVO 2

Zwei Räder, aber nur ein Arm – verglichen mit dem rollenden JetBot von Seite 40 wirkt dieser selbstbalancierende Roboter geradezu exaltiert. Doch der Arm hat seinen Sinn, denn damit hilft sich der EVO 2 selbst wieder auf die Beine – pardon: Räder, sollte er sich trotz robuster Regelung mal auf den Bumper legen.

von Peter König

Erinnern Sie sich noch? Vor fast 20 Jahren hörte die Hoffnung auf die Zukunft der Elektromobilität auf den Namen „Segway“ und manch einer sah die Fußgänger der Zukunft schon in Scharen mit zwei Rädern unter den Füßen über die einstigen Gehwege flitzen. Klar, dass schon bald die ersten darüber nachdachten, wie das Konzept des selbstbalancierenden Zweirads mit den Mitteln von Makern zu nutzen sei – und dass immer billigere Lagesensoren zusammen mit immer schnelleren Bastelboards die Idee befeuerten, auch Eigenbau-Roboter auf zwei Rädern balancieren zu lassen. Eine bekannte Konstruktion vor ein paar Jahren war *Eddie*, bei dessen Namen das eingebaute Board *Intel Edison* Pate stand und dem sich unser Schwestermagazin in den USA ausführlich widmete (siehe auch Bild Seite 17).

Doch Intel hat das Edison-Programm vor zwei Jahren eingestellt, deshalb sahen wir uns nach einer Alternative um. Fündig wurden wir bei *jjrobots* und deren Konstruktion *B-Robot EVO 2*. Man kann diesen Roboter entweder als Bausatz kaufen, wahlweise mit und ohne Teile aus dem 3D-Drucker. Da mit Ausnahme des zentralen *Devia Robotics Control Board* im Bausatz nur Standard-Komponenten wie Schrittmotoren, Treiber und Servo enthalten sind, bekommen Maker mit gut sortierter Bastelkiste das Board für 36 Euro auch einzeln. Als Prozessor dient derselbe ARM Cortex M0 (ATSAMD21G18) wie im Arduino Zero, ein ESP12 sorgt für WLAN, außerdem sind neben einem 6-Achsen-Gyro- und Beschleunigungssensor (ICM 20600) noch Anschlüsse für drei Schrittmotoren und vier Servos sowie Schnittstellen für I²C, SPI, UART und einen Sensor mit wahlweise digitalem oder analoger Signalleitung an Bord. In der Arduino-IDE konfiguriert man das Board als *Arduino Zero*. Wer den Roboter bauen, aber das Board nicht kaufen will, findet beim Hersteller einen Schaltplan für den Eigenbau (siehe Link).

Wir waren etwas bequemer und haben den Bausatz gekauft 1, allerdings die Kunststoffteile für das Chassis selbst gedruckt 2. Die beiden Naben für die Räder 3 hätte man mit der Achsöffnung nach oben drucken können; aus optischen Gründen haben wir uns für die umgekehrte Orientierung entschieden und dafür in Kauf genommen, Stützen erst drucken (links im Bild) und dann entfernen zu müssen (rechts).

Aufbauen

Eine ausführliche Baubeschreibung erübrigt sich an dieser Stelle – zum einen ist die Montage nicht wirklich kniffelig, zum anderen gibt es vom Hersteller ein ausführliches Video zum Aufbau (siehe Link in der Kurzinfo). Deshalb im Folgenden nur ein paar Tipps.

Kurzinfo

- **Selbstbalancierender Zweirad-Roboter aus dem Bausatz**
- **Per App über WLAN fernsteuern und live mit Regelungsparametern experimentieren**
- **Erweiterbar dank Open Source, durch Arduino-Code und zusätzliche Board-Schnittstellen**

Checkliste

Zeitaufwand:
zwei Stunden (ohne 3D-Druckzeit und Programmierung)

Kosten:
79 Euro ohne 3D-Druck-Teile, sonst 120 Euro

Elektronik:
Komponenten einfach zusammenstecken

Programmieren:
Blockly, Arduino (optional)

3D-Druck:
Teile des Chassis selber drucken (optional)

Steuerung:
über kostenlose App für iOS und Android

Material

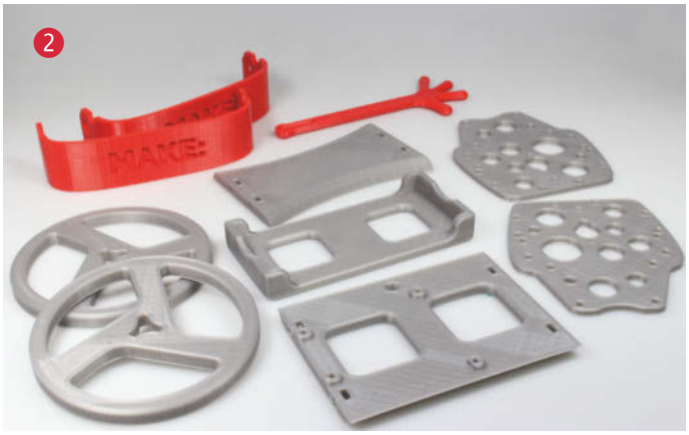
- **Devia Robotics Control Board** v1.0 von jjrobots.com, Alternative siehe Link
- **2 Schrittmotoren NEMA17** mit Kabeln, mindestens 14cm lang
- **2 Schrittmotortreiber** A4988 samt Kühlkörper
- **Servo** MG90S, mit Metallgetriebe
- **Batteriehalter 6 × AA** mit An/Aus-Schalter
- **Diverse Schrauben M3** Zylinderkopf, 6mm, 8mm und 16mm, mit Muttern
- **Zahnriemen** oder andere Gummiringe als Reifen
- **3D-gedruckte Teile** für das Chassis, aus PLA

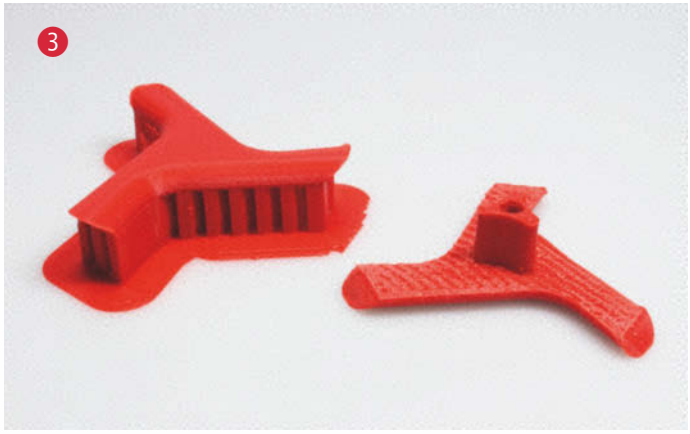
Alles zum Artikel im Web unter
make-magazin.de/xpvh

Das Chassis besteht aus zwei Seitenwänden und drei waagerechten Teilen, in der Anleitung *Shelves* (Regale) genannt, was ganz gut passt, weil diese die meisten technischen Komponenten tragen. Das Board schraubt man unter das mittlere Shelf. Bei unseren selbst gedruckten Teilen waren die Löcher für die M3-Schrauben etwas eng, weshalb man Kraft aufwenden muss – natürlich ohne mit dem Schraubendreher abzurutschen und das Board zu beschädigen.

Dann steckt man die Motortreiber auf und klebt auf deren Leistungs-IC die Kühlkörper auf 4 (rechts im Bild). Die sind deutlich größer als das IC und dürfen **nicht** zentriert angebracht werden, weil sie sonst zu nah an


1





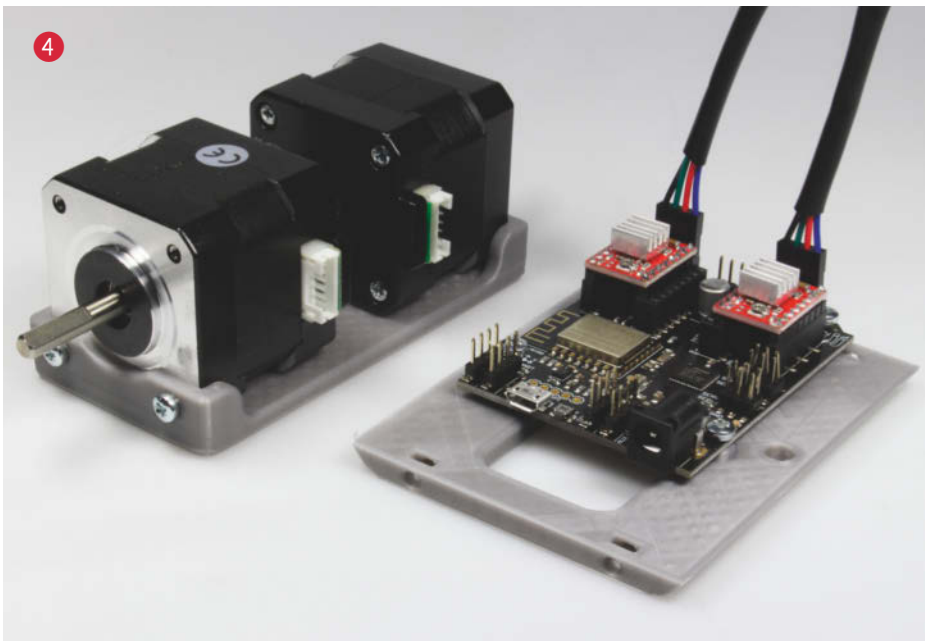
der einen Pin-Leiste liegen – bei ungewollten Berührungen kann hier ein Kurzschluss entstehen. Das Anleitungsvideo weist leider erst viel später auf diese Gefahr hin, wenn man die Kühlkörper schon längst geklebt hat.

Den Servo für den Arm haben wir – abweichend von der Anleitung – zuerst mit der Seitenwand verschraubt und erst dann diese an das Shelf mit der Platine montiert. So fällt es viel leichter, mit den winzigen M2-Muttern für die Servobefestigung zu hantieren. Die nötigen Schrauben hierfür schienen auf den ersten Blick im Bausatz zu fehlen – die Tüte mit den Schrauben enthielt zwar die passenden Muttern und zwei M2-Schrauben, die aber für die Servomontage die entscheidenden zwei Millimeter zu kurz sind. Die richtigen Schrauben liegen hingegen in der Tüte mit dem Servozubehör. Wie der Servo platziert wird 5 6, ist im sonst gut verständlichen Anleitungsvideo kaum zu erkennen – dort sind sowohl das Servogehäuse als auch die Seitenwand schwarz und nicht besonders gut beleuchtet.

Das Batteriefach soll man laut Anleitung mit einer Schraube fixieren – bei unserem Bot sitzt es so stramm im Rahmen, dass sich die erübrigt. Will man die Batterien wechseln, muss man die Schrauben am Rahmen lockern, sonst bekommt man das Fach überhaupt nicht bewegt.

Die serienmäßigen Bumper aus dem Bausatz bestehen aus fester Kunststofffolie; wir haben uns mit Hilfe des *Customizers* auf Thingiverse (siehe Link in der Kurzinfo) massivere Teile mit Make-Prägung gedruckt (und dafür auf die beiliegenden Wackelaugen verzichtet). Passende Schrauben lieferte die Bastelkiste, ebenso wie für die Montage des oberen Shelfs – hier erwiesen sich die Löcher im 3D-Druck-Teil als eine Idee zu wenig tief.

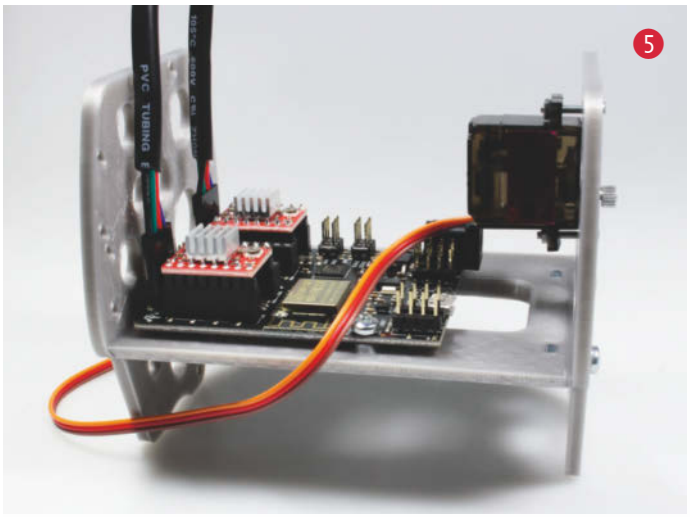
Die Naben sollen eigentlich nur in die Räder gesteckt werden und dann halten. Das war bei unseren Teilen nicht der Fall. Heißkleber erwies sich als gut, auf Dauer wären (Maden-)Schrauben besser. Die Löcher in den Naben für die Achsen der Schrittmotoren wiederum waren zu eng, um die Räder von Hand auf die Achsen zu pressen. Am Ende behalfen wir uns mit einer sogenannten Einhandzwinge 7. Dabei muss man sehr gefühlvoll vorgehen und den Druck nur langsam erhöhen. Die Zwingenbacken müssen außerdem genau mittig auf dem Rad und dem Motor platziert werden, damit nichts verkantet. Außerdem sollte man dafür sorgen, dass die Zwinge auf der Rückseite des Motors auf dessen Achse und nicht nur auf das Gehäuse wirkt, damit nicht das Kugellager der Achse den ganzen Druck abbekommt. Eine runde Zulage, die in das Loch auf der Rückseite des Schrittmotors passt, verhindert dies – ist nichts anderes zur Hand, passt dafür auch eine Mutter M4 8.



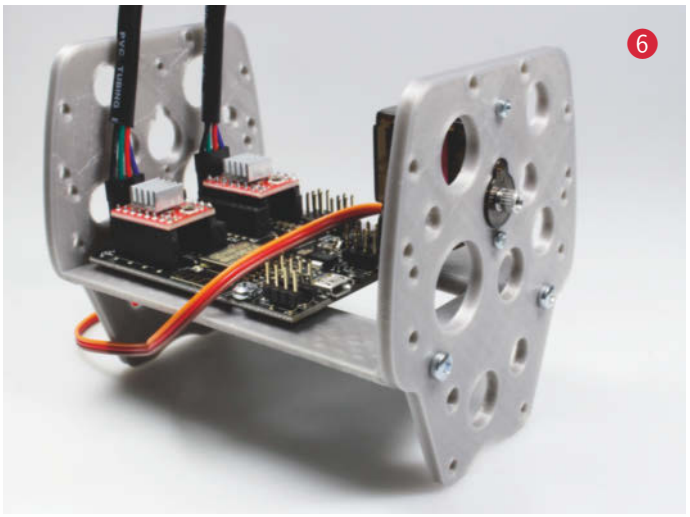
Balancieren

Nach gemütlich zwei Stunden Bauzeit kam der spannende Moment: Roboter zu einer Seite kippen, über den Schalter am Batteriefach anschalten und kurz warten – zum Zeichen, dass er fertig gebootet hat, winkt er dreimal kurz mit dem Arm. Einen Handgriff später stand er auf dem Boden des Labors – auf Anhieb. Nur ganz wenig schaukelte der Roboter hin und her, um die Balance zu halten.

Das war schon mal sehr beeindruckend. Nach ein bis zwei Minuten spätestens stand dann auch das WLAN des Roboters, mit dem man sich per Smartphone verbindet. Zum Steuern dient eine App – wir haben zuerst die Version für Android getestet. In der App bekommt man per Telemetrie (vom Roboter gesendete Sensordaten) die aktuelle Neigung 9 und den Batteriezustand 10 angezeigt und man kann den Roboter fernlenken. Dank der Schrittmotoren fährt er nicht nur wie am Lineal geführt exakt geradeaus, er bremst auch zackig ab, sobald man den Gashebel 11 wieder loslässt. Auf der Stelle dre-



5



6

hen ist dank der Zweiradkonfiguration kein Problem, der Roboter geht aber auch aus voller Fahrt heraus geschmeidig in die Kurve, wenn man etwas mit der Fernsteuerung übt – ein Druck auf die *Pro*-Taste (12) erhöht die Agilität. Zu stark sollte man bei hohem Tempo das Steuer allerdings nicht zur Seite reißen (13), sonst liegt der EVO 2 schnell mal auf der Nase beziehungsweise dem Bumper. Dann sollte ihm eigentlich ein Tipp auf die *Servo*-Schaltfläche (14) zum Aufstehen genügen. Reicht dieser Impuls nicht aus, kann man durch gefühlvolles Drehen im Uhrzeigersinn an den Trimmern der Motortreiber deren Referenzspannung erhöhen.

Noch zwei Tipps aus erster Hand: Die Zink-Kohle-Batterien, die für den schnellen ersten Test gerade zur Hand waren, erwiesen sich als ungeeignet – die App zeigte sie schon nach wenigen Minuten Balancieren als halb leer an. Mit hochwertigeren Batterien sollen Fahrzeiten von einer halben bis zu einer Stunde möglich sein.

Außerdem muss man je nach Android-Gerät unterschiedlich hartnäckig sein, um das Telefon oder Tablet dazu zu bewegen, die Verbindung zum Roboter-WLAN zu halten – unter Umständen muss man per Checkbox explizit darauf bestehen, dieses WLAN zu nutzen, obwohl es keinen Internetzugang bietet. Sonst kann es passieren, dass die Lage und der Batteriestand per Telemetrie korrekt angezeigt werden, aber der Roboter nicht auf Steuerbefehle reagiert. Unter iOS konnten wir dieses Problem nicht beobachten.

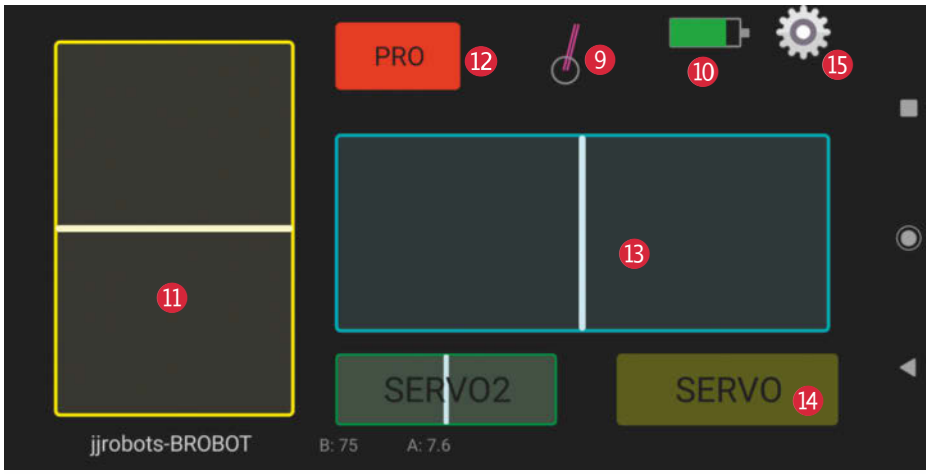
Invers pendeln

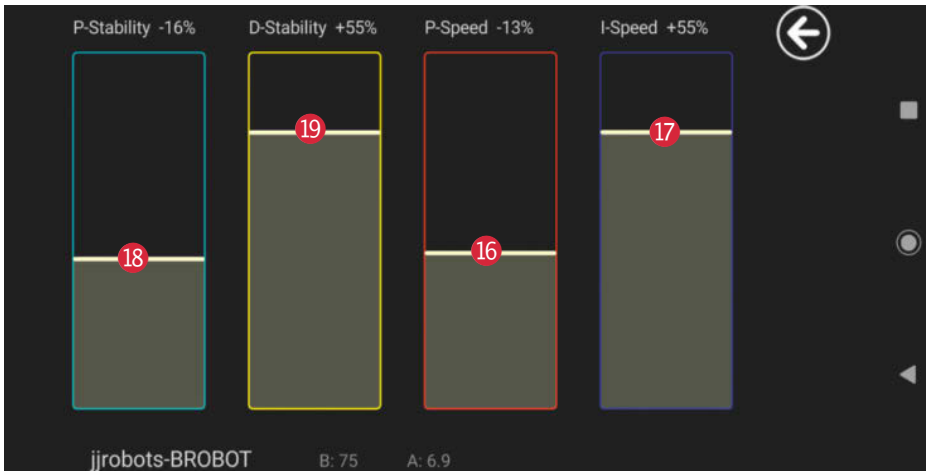
Physikalisch betrachtet stellt ein selbstbalancierender Roboter ein sogenanntes *inverses Pendel* dar – anders als beim normalen Pendel befindet sich hierbei der Drehpunkt unten und der Schwerpunkt oben. Zur Seite kann der Roboter dabei nicht kippen, er muss also lediglich durch wohldosierte Ansteuerung seiner Motoren dafür sorgen, dass er in Vor- und Rückrichtung sein labiles Gleichgewicht hält. Dabei hilft ihm sein Lagesensor, der 200mal pro Sekunde Informationen über die aktuelle Neigung liefert. Der Rest ist ein Regelungsproblem.

Ein Regler dient dazu, die Abweichung (auch *Fehler* genannt) zwischen *Sollwert* (beim Roboter ist das die gewünschten Lage) und *Istwert* (der gemessenen Lage) durch eine passende *Stellgröße* (Kommandos an die Motoren) auszugleichen. Damit das auf praktikable Art und Weise geschieht – also ohne dass sich das System aufschaukelt oder am Ende bei einem Zustand mit bestehen-


7

8





dem Fehler stehen bleibt –, kommen nicht nur bei selbstbalancierenden Robotern sogenannte PID-Regler zum Einsatz, echte Klassiker der Regelungstechnik.

PID ist dabei eine Abkürzung für die drei Komponenten, die in so einen Regler einfließen. Sie stehen für die englischen Begriffe *Proportional*, *Integral* und *Derivative*, zu deutsch: Proportionalität, Integral und Ableitung. Keine Angst, es folgt keine Wiederholung von Oberstufenmathematik und Kurvendiskussion – wer mehr Details nachlesen will, sei auf den Link in der Kurzinfo verwiesen. Deshalb nur in groben Zügen:

Der PID-Regler besteht aus drei Gliedern. Eines wird proportional zum aktuell ermittelten Fehler berechnet und sorgt dafür, den Fehler schnell zu minimieren. Das zweite berechnet das Integral des Fehlers über die Zeit, bildet also die zurückliegenden Abweichungen zwischen Soll- und Istwert ab – es sorgt dafür, dass der Fehler irgendwann auf null sinkt, weil sein Anteil am Regler desto größer wird, je länger ein Fehler besteht. Das dritte Glied bietet so etwas wie eine Hypothese auf die weitere Entwicklung des Fehlers, denn es bildet die Ableitung der Fehlerfunktion über die Zeit, was bildlich die Steigung der Tangente an dieser Kurve darstellt. Alle drei Glieder werden durch je einen Faktor gewichtet, dann alles zusammengerechnet und daraus die Stellgröße (sprich: die Motorkommandos) ermittelt.

Konkret wird das beim B-Robot in zwei Stufen abgewickelt: Ein PI-Regler sorgt für die Geschwindigkeitsregelung; hier fließt die vom Nutzer über die App eingestellte Geschwindigkeit als Sollwert und die ermittelte Geschwindigkeit des Roboters als Istwert ein. Die Gewichtungsfaktoren für das P- und das I-Glied kann man selbst tunen: Ein Fingertipp auf das Zahnrad-Symbol 15 in der App bringt dafür vier Schiebesteller zum Vorschein. Die beiden rechten davon beeinflussen die Faktoren für das P- 16 und das I-Glied 17 des Geschwindigkeitsreglers.

Daraus ergibt sich der gewünschte Kippwinkel – bei höherer Geschwindigkeit lehnt der Roboter sich weiter nach vorne. Der Kippwinkel geht als Sollwert in der nächsten Stufe in einen PD-Regler ein, der für die Stabilität zuständig ist. Der Lagesensor liefert hierfür den Istwert. Für diesen Regler kann man mit den Parametern für das P- 18 und das D-Glied 19 in der App herumspielen. Der maximale Wert ist dabei das Doppelte des Standard-Werts, der minimale ist 0, damit ist das Glied des Reglers faktisch deaktiviert. Zieht man beide Parameter links in der App auf 0, fällt der Roboter stumpf um.

Die optimalen Einstellungen für unterschiedliche Szenarien wie Wettrennen oder den Transport von Getränkedosen zu finden (für die es ein angepasstes oberes Shelf für den Roboter braucht) sind praktische Anwendungen für die Schiebesteller in der App. Setzt man etwa die *D-Stability* hoch, balanciert der Roboter mit deutlich kleineren Schaukelbewegungen, braucht aber dann mehr Gas, um sich nicht nur nach vorne zu neigen, sondern tatsächlich auch loszufahren.

Warnen müssen wir allerdings davor, in der iOS-App die Einstellungen für die Parameter zu öffnen: Reproduzierbar fing der Roboter daraufhin heftig an zu schaukeln, schlug dann hin und versuchte trotzdem, mit Vollgas über den Boden zu schlittern. Erst Abschalten machte diesem Spuk ein Ende.

Erweitern und programmieren

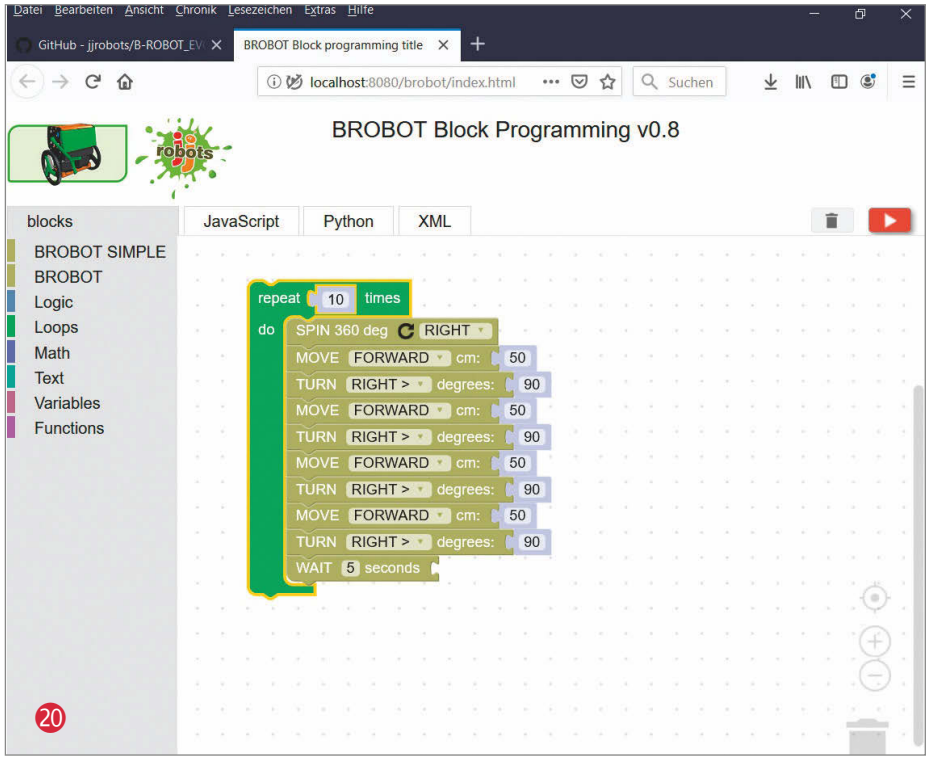
Die einfachste Erweiterung kann man rein mit Hardware und Mechanik umsetzen – die Steuer-App sieht bereits einen Schieberegler für einen zweiten Servo vor. Um den zu montieren, lädt man sich die 3D-Druckvorlagen für das Chassis von Thingiverse herunter und modifiziert die den eigenen Wünschen entsprechend mit dem 3D-Programm der eigenen Wahl.

Insgesamt kann das Board vier Servos ansteuern. Wer den Bewegungsapparat seines Roboters so stark erweitern will, braucht dafür aber eine andere App oder sonstige Steuerungssoftware. Der Hersteller hat als Protokoll für die Übertragung der Kommandos OSC (Open Sound Control) gewählt – ein Protokoll, das ursprünglich für die Vernetzung von Synthesizern entwickelt wurde. Die Kommunikation zwischen App und Roboter läuft über WLAN und UDP. Wer selbst ein Frontend für die Steuerung entwickeln will, findet OSC-Unterstützung in Programmierumgebungen wie OpenFrameworks, Processing oder Pure Data.

Spätestens wenn man die bisher vakanten Schnittstellen auf dem Board für zusätzliche Sensoren nutzen will, die auch autonomes Verhalten erlauben, muss man Hand an die Firmware des Roboters legen. Für diese bekommt man im Repository des Herstellers bei Github den Arduino-kompatiblen Quellcode. Einsteigerfreundlich ist der nicht gerade – und gut 2000 Zeilen lang, wenn auch mit Kommentaren und Debug-Ausgaben. Das ist definitiv nicht geeignet, um darüber in die Arduino-Programmierung einzusteigen, bietet versierten Makern aber viele Entfaltungsmöglichkeiten.

Für Einsteiger viel einfacher ist die Programmierung des Roboters mittels Blockly: Verbindet man seinen Laptop mit dem WLAN des Roboters und startet die aus dem Git-Repository heruntergeladene Exe-Datei, öffnet sich die visuelle Programmierumgebung im Browser 20. Für den B-Robot gibt es in der Seitenleiste spezielle Befehlsbausteine, etwa um eine bestimmte Strecke vorwärts zu fahren oder sich um einen bestimmten Winkel zu drehen. Ein Klick auf den *Run*-Button rechts (der ein wenig nach dem YouTube-Icon aussieht) schickt den Code dann Befehl für Befehl direkt zum Roboter, der sie ausführt. Wer will, kann zwischendrin kleine Korrekturen per App dazwischenmogeln.

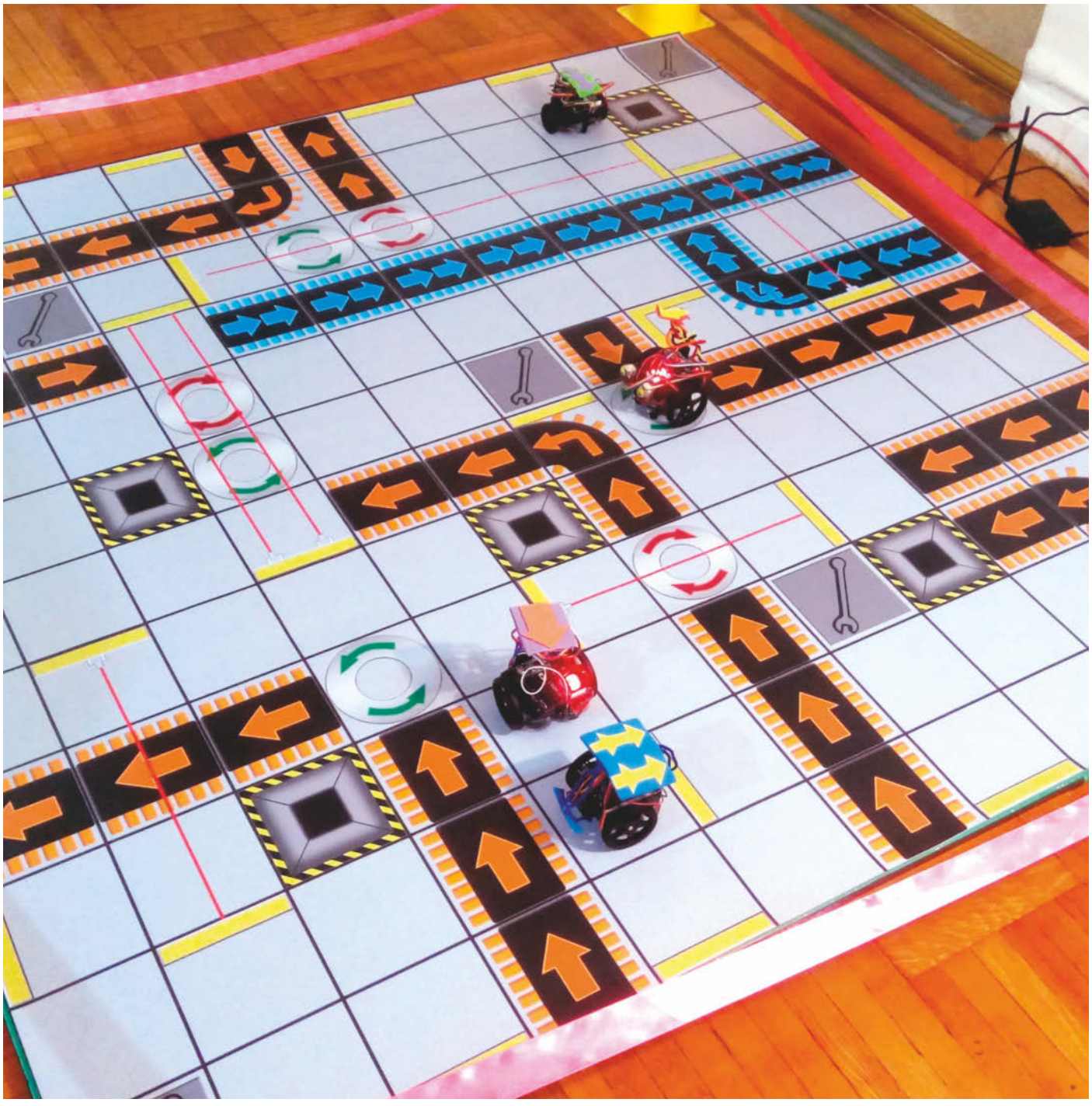
Der Aufbau des Roboters hat uns fast genauso viel Spaß gemacht wie das Rumheizen damit – und in Sachen Tuning und Erweiterungen haben wir sicher noch manches Vergnügen vor uns. Als Erstes wird aber noch ein neuer Bumper ohne Schrift gedruckt und darauf kommen dann die Wackelaugen – der Roboter steuert sich eben einfacher, wenn man genau sieht, wo vorne ist. *—pek*



RoboRally
mit echten Robotern

Im Spiel RoboRuckus navigieren die Spieler echte Roboter durch einen wilden Hinderniskurs. Das Open-Source-Projekt ist vom Brettspiel RoboRally inspiriert.

von Sam Groveman (Übersetzung: Niq Oltman)



Das Brettspiel RoboRally begeistert meine Freunde und mich schon lange. Irgendwann kam uns die Idee, das Spiel mit richtigen Robotern nachzubauen – und dieser Einfall war zu verlockend, um ihn nicht umzusetzen. Auch wenn die Idee simpel klang, war die Umsetzung unseres Open-Source-Projekts RoboRuckus im Endeffekt eine große Herausforderung.

Spielprinzip

Jeder Spieler steuert einen Roboter. Das Ziel ist es, den Parcours auf dem Spielbrett zu durchqueren und dabei bis zu vier Checkpoints in der richtigen Reihenfolge zu berühren. In jeder Runde bekommt der Spieler Karten, auf denen Anweisungen stehen, zum Beispiel *vorwärts fahren*, *links abbiegen* oder *rückwärts fahren*. Davon sucht die Person sich fünf Karten aus: Sie bestimmen, was der Roboter in diesem Zug tun soll (1). Sobald sich der Spieler für so ein „Programm" entschieden hat, darf er es nicht mehr verändern. Wenn alle Spieler ihre Programme festgelegt haben, fahren alle Roboter gleichzeitig los und versuchen die Checkpoints zu erreichen – während sie sich gegenseitig herumschubsen.

Navigation

Optische Streckenführung ist die einfachste und beliebteste Methode für Roboter-Navigation. Allerdings müssen dafür auf dem ganzen Spielbrett Linien mit hohem Kontrast abgebildet sein. Das passt leider nicht zum Spielbrett von RoboRally. Stattdessen kamen wir auf die Idee, mit einer magnetischen Streckenführung zu arbeiten. Dazu setzten wir magnetisches Klebeband sowie Hall-Effekt-Sensoren ein – als Gegenstück zu den Lichtsensoren bei einer optischen Führung. Später haben wir die Sensoren durch zuverlässigere Magnetometer (Digitalkompasse) ersetzt.

Roboter

Das Projekt sollte sowohl unkompliziert als auch kostengünstig sein. Daher entschieden wir uns dafür, einen Batteriehalter für zwei Mignonzellen als Rumpf für die Roboter und frei drehende Servos für den Antrieb zu nehmen. Eine Siebensegment-LED und ein Piezo-Summer sollten Statusinformationen liefern. Per WLAN können sich die Roboter mit dem Steuerungsserver austauschen.

Unser erster Prototyp (2) basierte auf einem Arduino Pro Mini. Schnell zeigte sich, dass die Hall-Effekt-Sensoren nicht empfindlich genug sind, um dem Magnetband auf der Rückseite des Papp-Spielbretts zu folgen. Wir tauschten die Sensoren gegen Digitalkompasse aus – einen für vorne und einen für hinten – damit der Roboter vorwärts und rückwärts navigieren kann. Dafür brauchten wir jetzt zwei I²C-Kanäle, also wechselten wir vom Arduino Pro Mini zu einem Teensy LC.

Unser zweiter Prototyp (3) konnte schon gut geradeaus fahren, vorwärts wie rückwärts. Allerdings konnte der Roboter beim Abbiegen dem schwankenden Magnetfeld nicht zuverlässig folgen. Zum Glück stieß ich auf ein sehr gutes Sensor-Breakout-Board mit neun Freiheitsgraden (9 DOF), auf dem der bisherige Digitalkompass mit einem digitalen Gyroskop kombiniert war. Nachdem ich das Board gewechselt hatte, konnten wir den Roboter durch Kurven lenken. Die Konstruktion gefiel uns gut, also legte ich in EAGLE einen Schaltplan an (4) und ließ damit eigene Platinen produzieren (5). Die sahen nicht nur gut aus, sie erlaubten uns auch, die Roboter schneller herzustellen. Wir haben

Kurzinfo

- Roboter fahren eigenständig auf dem Spielbrett
- Steuerung der Roboter mit einem Teensy über WLAN
- Wegfindung durch Magnetsensoren

Checkliste

Zeitaufwand:
ein Wochenende

Kosten:
circa 170 bis 200 Euro

Holzbearbeitung:
einfache Säge- und Fräsarbeiten

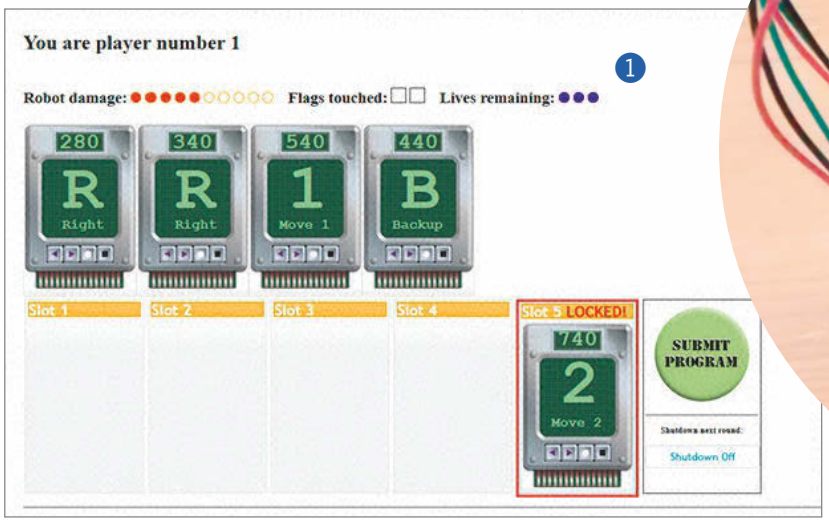
Elektronik:
Motoren installieren

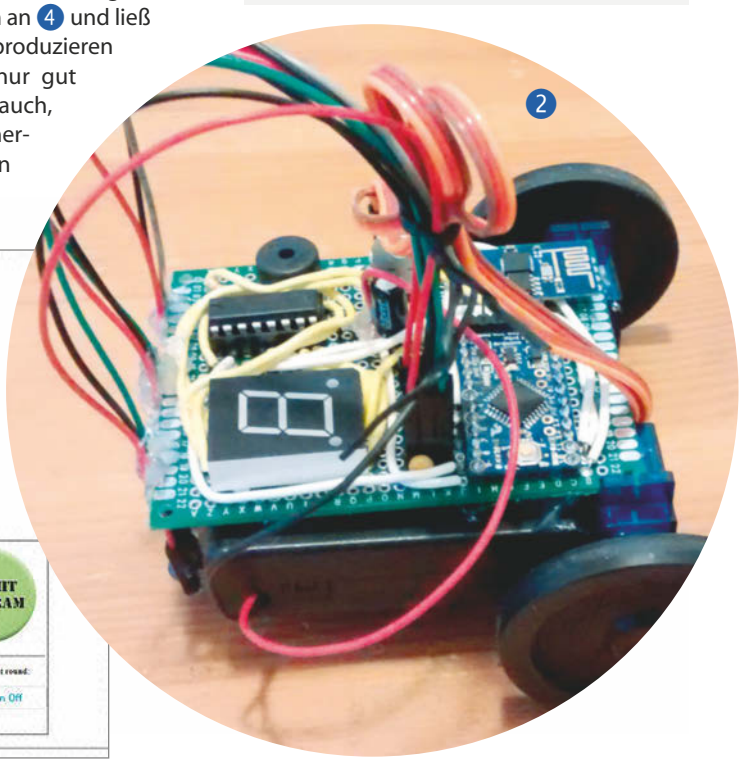
Löten:
Platine bestücken

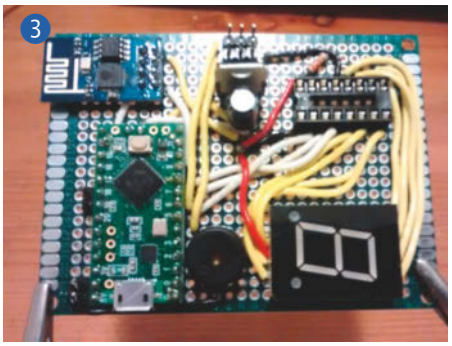
Material

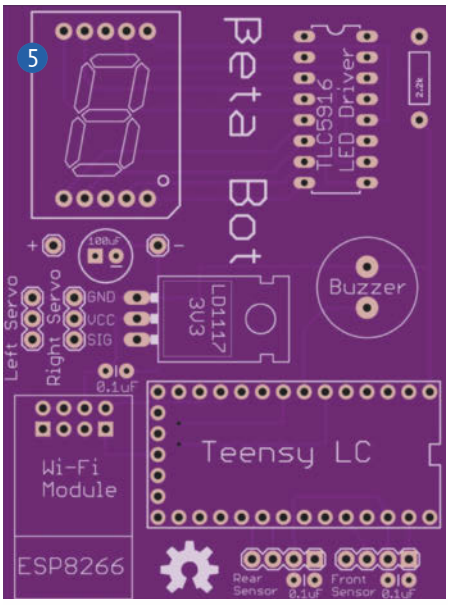
- Die Materialliste finden Sie unter dem Link.

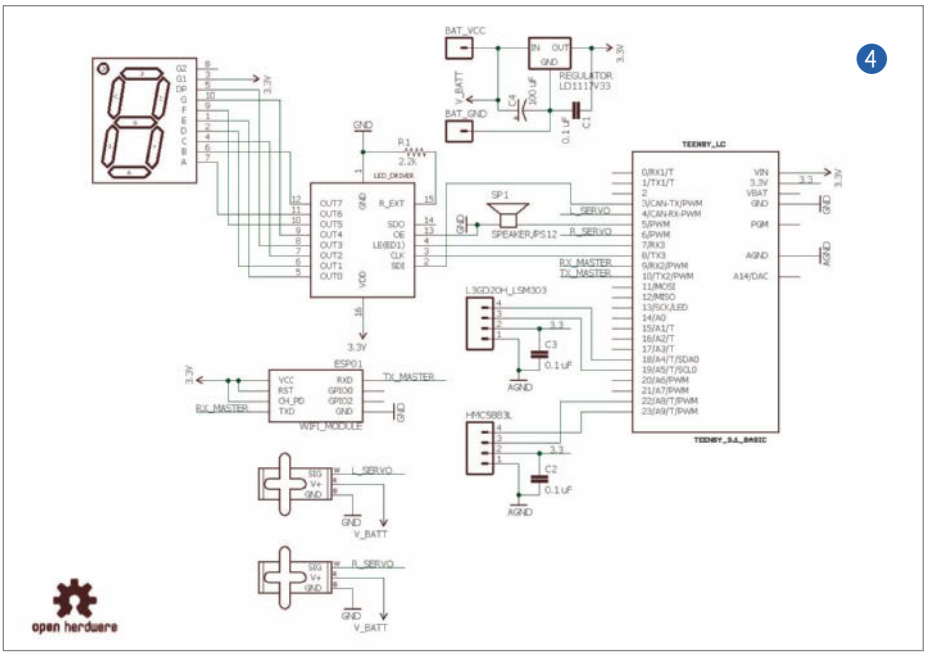
Alles zum Artikel im Web unter
make-magazin.de/xgmb











gleich eine ganze Armee aus Robotern gebaut – mit viel Charakter 6.

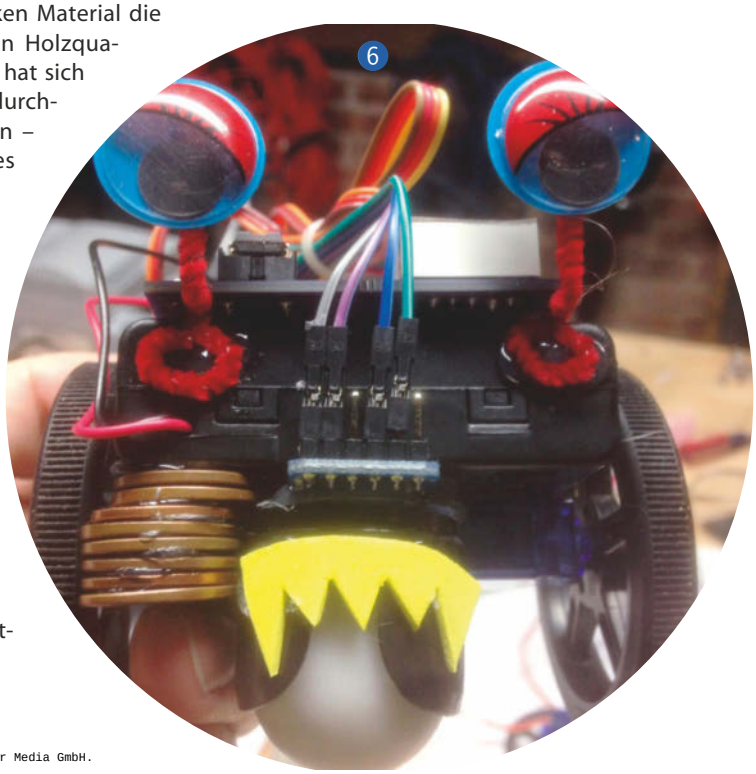
Spielbrett

Die längste Seite der Roboter maß etwa 10cm. Daraus schloss ich, dass das Spielbrett aus 12 × 12 Quadraten zu je 12,5cm Kantenlänge bestehen sollte. Wir haben die Quadrate auf einer 3mm dicken Sperrholzplatte angeordnet. Auf das Brett legten wir ein Raster aus Magnetband 7 und unterteilten es anschließend in neun Felder aus je 4 × 4 Quadraten. So ist es leichter auseinanderzunehmen und zu transportieren 8. Für den Aufdruck habe ich das klassische RoboRally-Brett in hoher Auflösung neu gezeichnet. Wir haben es auf Vinyl gedruckt, um mit dem dicken Material die Übergänge zwischen den Holzquadraten zu glätten. Leider hat sich schon nach einigen Testdurchläufen das Holz verzogen – vermutlich haben wir es falsch gelagert. Zusätzlich war das Magnetband unter dem Sperrholz für die Sensoren zu schwach, besonders an den Kreuzungen. Der Roboter fuhr häufig zu viele Felder geradeaus, bis er eine Kreuzung erkannte.

Am Ende mussten wir das ganze Spielbrett noch einmal neu entwerfen. Diesmal nahmen wir eine stabilere Platte aus 12mm dickem MDF. Zusätzlich verlegten wir diesmal das Magnetband auf der Oberseite, damit die Sensoren ein stärkeres Signal erhielten. Danach setzten wir in alle Kreuzungen Neodym-Magneten ein, um dort das Magnetfeld zu erhöhen 9. Mit diesen Änderungen erkannte der Roboter die Abzweigungen sehr viel besser. Die Vinylmatte mit dem Aufdruck konnten wir immerhin wiederverwenden.

Code

Die Spielregeln von RoboRally sind relativ einfach und algorithmisch, daher ließen sie sich prima in Computercode übertragen.



Der Gameserver läuft auf einem Raspberry Pi mit eigenem WLAN-Netzwerk. Der Pi hat ein Web-Interface, geschrieben in ASP.NET Core. Dort verbinden sich alle Spieler und Roboter. Der Gameserver koordiniert den Spielablauf, die Position der Roboter und die Eingaben der Spieler.

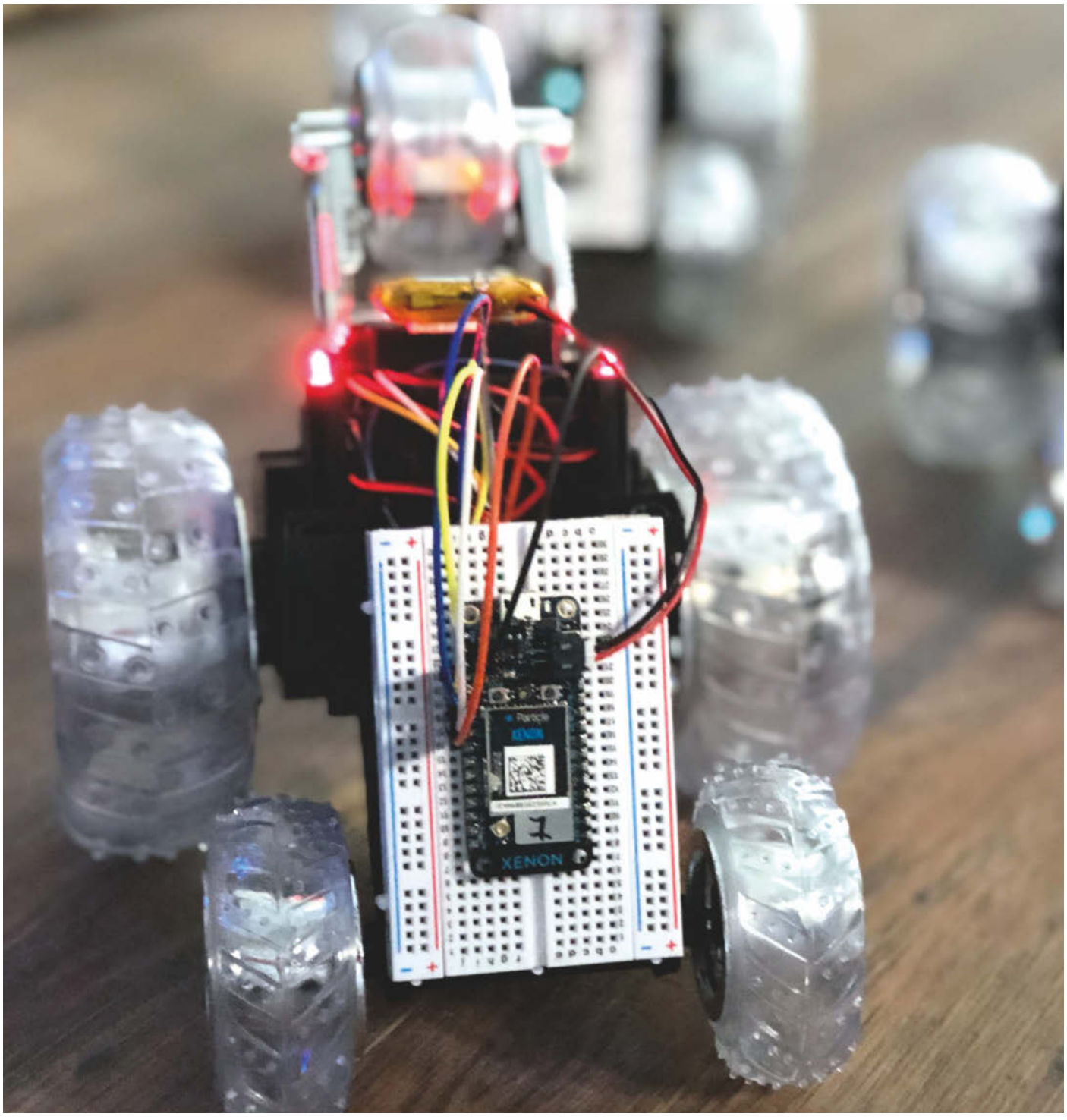
Fazit

Insgesamt sind wir mit dem Ergebnis sehr zufrieden. Allerdings wollen wir in der Zukunft noch einige Punkte verbessern. Ganz oben auf der Liste steht ein 3D-gedrucktes Fahrgestell für die Roboter. Die Stromversorgung sollten wir ebenfalls überarbeiten, damit die Servos eine stabilere Spannung haben – eventuell mit LiPoly-Akkus. Bis alles richtig lief, haben wir an diesem Projekt mit sporadischen monatlichen Bastelsessions etwas unter einem Jahr gesessen. Da wir nun alle wesentlichen Schwierigkeiten gelöst haben, glaube ich, dass eine engagierte kleine Gruppe das Projekt an einem Wochenende nachbauen könnte. *—rehu*

Swarmbots!

Dieser Schwarm von günstigen Modell-Autos kommuniziert mithilfe von Mesh-Netzwerken miteinander. Im Browser und per App kann man das Rudel synchron fernsteuern und Manöver ausführen lassen.

von Brandon Satrom (Übersetzung: Niq Oltman)



Spielzeuge zu hacken ist eine wunderbare Methode, um herauszufinden, wie Elektrogeräte konstruiert und aufgebaut sind. Man kann ungezwungen basteln, ohne Angst, dass man teure Ausrüstung ruiniert. Generell sind günstige Elektronikgeräte eine prima Lernplattform, mit der ganz neue Ideen entstehen können.

Fahrzeuge mit Funkfernsteuerung sind beliebte Bastelobjekte. Sie heißen auch RC-Fahrzeuge, dabei steht RC für radio-controlled. Eine erschwingliche und leicht zugängliche Plattform zum Hacken von RC-Autos sind die Autos der Marke Thunder Tumbler. Diese Autos wurden schon oft gehackt, aber meines Wissens noch nie für den Einsatz in einem Mesh-Netzwerk – zumindest bis jetzt. Mit der Mesh-fähigen Hardware von Particle habe ich ein Swarmbot-Netz aus Tumbler-Autos gebaut, die auf Kommando synchron herumfahren.

Mesh-Netze kurz erklärt

Wie funktionieren Mesh Networks? Die meisten vernetzten Systeme nutzen WLAN oder Mobilfunk zur Kommunikation untereinander. Hier hat üblicherweise jedes Gerät eine eigene Verbindung ins Internet. Das ist praktikabel, wenn man Zugang zur Cloud braucht, zum Beispiel zum Speichern oder Verarbeiten von Daten. Manchmal will man seine Maschinchen aber einfach nur lokal miteinander verbinden – unabhängig vom Internet. Das ist mit einem Mesh-Netzwerk möglich. Der Großteil des Netzwerks besteht hier aus „Endpunkten“ mit Sensoren oder Antriebselementen. Der andere Teil besteht aus Verteilern, sogenannten Repeatern. Sie leiten die Nachrichten von Gerät zu Gerät weiter und erhöhen damit die erreichbare Größe und Zuverlässigkeit des Netzwerks. Zusätzlich dient eine kleine Anzahl Geräte – oft nur ein einziges – als Gateway: Es ist der Vermittler für die Anbindung ans Internet. Als Besonderheit dieser lokalen Netzwerke können die Geräte sich auch dann weiter austauschen, wenn die Internetverbindung ausfällt. Die Microcontroller-Boards Argon, Boron und Xenon von Particle haben die Mesh-Netzwerk-Funktionalität für solche Anwendungen schon eingebaut.

Für dieses Projekt habe ich mit der Particle-Mesh-Plattform ein Netz aus ferngesteuerten Autos aufgebaut, jeweils gesteuert von einem Particle Xenon. In diesem Mesh-Netzwerk agieren die Xenon-bestückten RC-Autos als Endpunkte. Ein einzelner Particle Argon dient als Internet-Gateway. Sobald das Netzwerk eingerichtet ist, kann ich Kommandos mit geringer Verzögerung an alle Knoten – die vernetzten Geräte – absenden und meine RC-Autos tanzen lassen. Aber zunächst müssen die Tumbler gehackt werden!

Kurzinfo

- Mikrocontroller an RC-Auto-Motor anschließen
- Kommunikation über Mesh-Netzwerk programmieren
- Schwarm-Verhalten einrichten

Checkliste

Zeitaufwand: ein Wochenende

Kosten: circa 130 bis 150 Euro

Löten: Grundkenntnisse

Elektronik: Grundkenntnisse Motoren

Programmieren: C/C++

Steuerung: via App

Alles zum Artikel im Web unter *make-magazin.de/xf7c*

Material

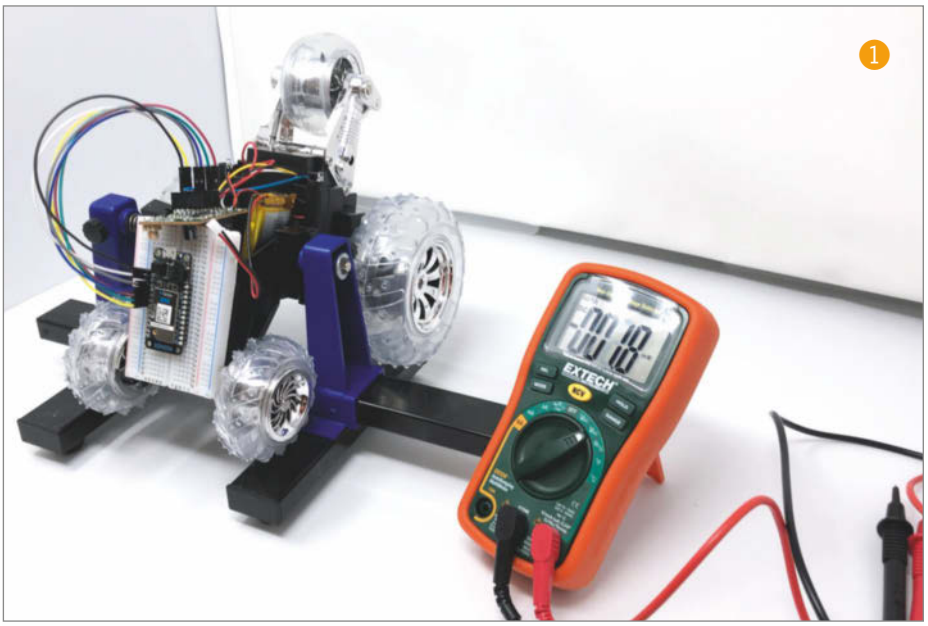
- **3 Thunder-Tumbler-RC-Autos**
- **Argon-Mikrocontroller** mit WLAN- und Mesh-Netzwerk-Gateway von Particle
- **3 Xenon-Mikrocontroller** mit Bluetooth- und Mesh-Netzwerk-Endpoint/Repeater von Particle
- **3 Breadboards** (sind auch in den Argon- und Xenon-Kits enthalten)
- **3 LiPo-Akkus** 3.7V und 1200mAh
- **Jumperkabel** zum Verbinden der Mikrocontroller mit den RC-Auto-Platinen

Werkzeug

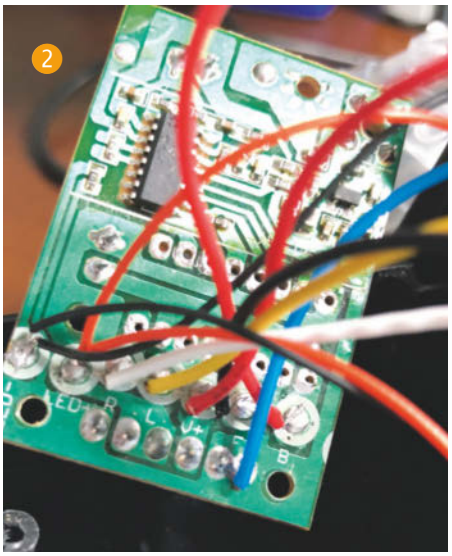
- **Multimeter** zum Testen der IC-Pin-Konfiguration
- **Lötkolben und Lötzinn**
- **Schraubendreher**
- **Computer**

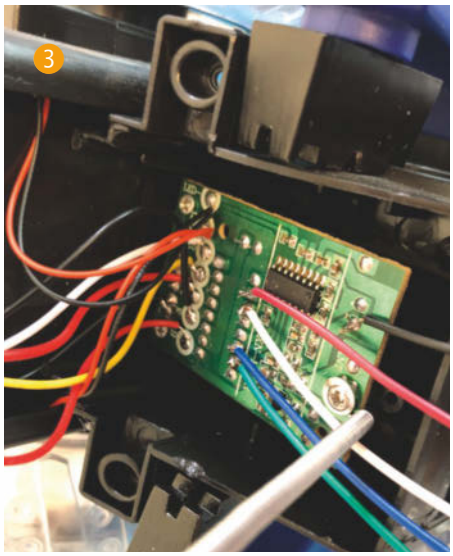
Hacken der Thunder Tumbler

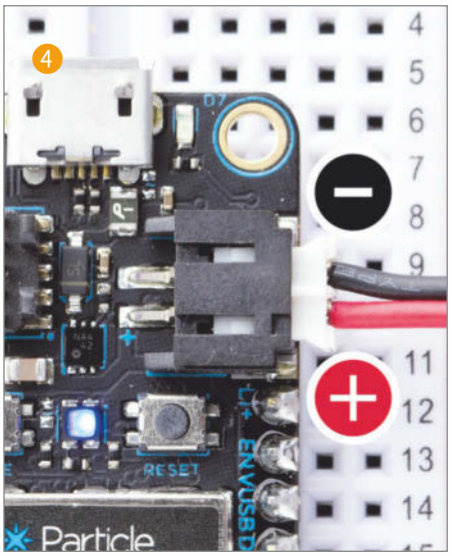
Im ersten Schritt bauen wir das RC-Auto so um, dass wir einen Xenon-Mikrocontroller einbauen können. Ganz egal, um was für ein ferngesteuertes Auto es sich handelt – zuerst schrauben wir es auf. Wir benötigen Zugang zur Platine im Innern, um herausfinden, über welche internen Signale die Motoren, und damit die Räder, angetrieben werden. Dann schließen wir unseren Xenon an die so ermittelten Motor-Antriebskontakte an. Bei so preisgünstigen Spielzeugautos müssen wir hier mit gewissen Abweichungen rechnen – auch zwischen den verschiedenen Modellen einer einzelnen Marke.



1







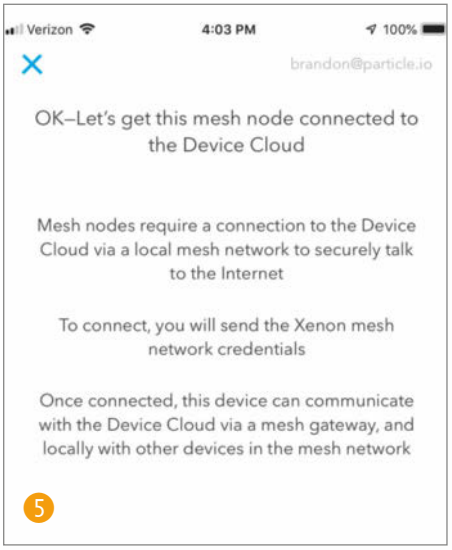
Für das Tumbler-Hacking gibt es zwar Online-Anleitungen, aber die sind zum Teil schon über neun Jahre alt. Das ist eine halbe Ewigkeit in der Elektronikwelt. Daher sollte man die genaue Funktion seiner RC-Autos gegenprüfen, wenn man diesen Anleitungen folgt. Über die Fernsteuerung lässt man die Räder laufen, während man mit dem Multimeter die Spannungen an verschiedenen Pins auf der Auto-Platine misst. Das Auto legt man so hin, dass die Räder sich ungehindert drehen können 1, sonst macht es sich während des Messvorgangs aus dem Staub.

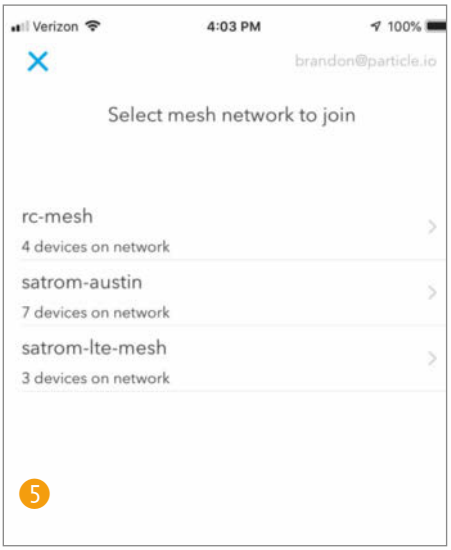
Die Karosserie wird mit zwei Schrauben zusammengehalten. Nachdem man sie gelöst hat, zeigt sich unter der Haube die Platine 2. Man sieht acht bis zehn Leitungen, die vom Auto auf die Platine führen, sowie Bauteile in Durchsteck- (THT) und Oberflächenmontagetechnik (SMD).

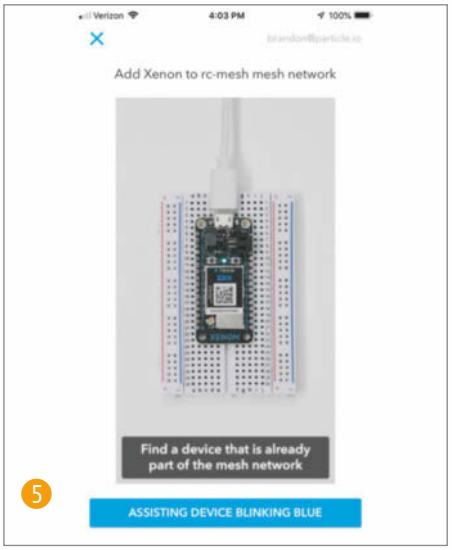
Das kleine SMD-Bauteil auf der Oberseite ist ein Funkfernsteuerungs-Empfänger, der RX-2B. Das Gegenstück, der Transceiver-Chip TX-2B, steckt im Sender der Fernsteuerung. Diese ICs sind in RC-Fahrzeugen weit verbreitet und die Datenblätter findet man leicht. Damit konnte ich zuordnen, welche Pins zu den Fernsteuerungs-Befehlen für vorwärts, rückwärts, links und rechts gehören und – wichtig für später – auch ein Masse-Pin identifizieren. Diese Zuordnung stellte sich als entscheidend heraus, denn für das andere, letztlich wichtigste IC auf der Platine (in Durchstecktechnik auf der Rückseite montiert) konnte ich trotz mehrstündiger Suche kein Datenblatt auftreiben. So ganz sicher bin ich noch immer nicht, was die genaue Funktion dieses Chips betrifft. Im Wesentlichen arbeitet er als Motorregler vom H-Brücken-Typ. Gibt man ein gepulstes Signal auf einen bestimmten Pin dieses Chips, entsteht ein Signal an einem der Motor-Antriebskontakte am Ausgang, mit dem sich das Rad dann vorwärts beziehungsweise rückwärts dreht. Durch Reverse Engineering habe ich herausgefunden, dass an diesem unbekannten IC nur vier Pins von Interesse sind, nämlich jeweils zwei für das linke und rechte Rad. Davon ist je einer zum Vorwärts- und Rückwärtsfahren. An diese vier Kontakte sowie den Masse-Pin habe ich je einen Draht gelötet 3. Damit alles richtig funktioniert, müssen der Xenon und das Auto sich eine Masse-Leitung teilen, obwohl sie getrennt mit Strom versorgt werden. Diese Leitungen wurden mit den Pins A0, A1, A2, A3 und GND am Xenon verbunden. Der Xenon bekam eine Stromversorgung in Form eines LiPo-Akkus 4.

Einrichten des Mesh-Netzwerks

Unser Mesh besteht aus einem Gateway sowie pro RC-Auto einem Xenon-basierten







Knoten. Wir wollen die Xenons drahtlos programmieren, sie also nicht jeweils an einen Computer anschließen müssen. Dazu richten wir das Netzwerk schon vorher ein. Das geht mit der Particle-Mobil-App 5. Alternativ kann man der Online-Anleitung folgen – Link in der Infobox.

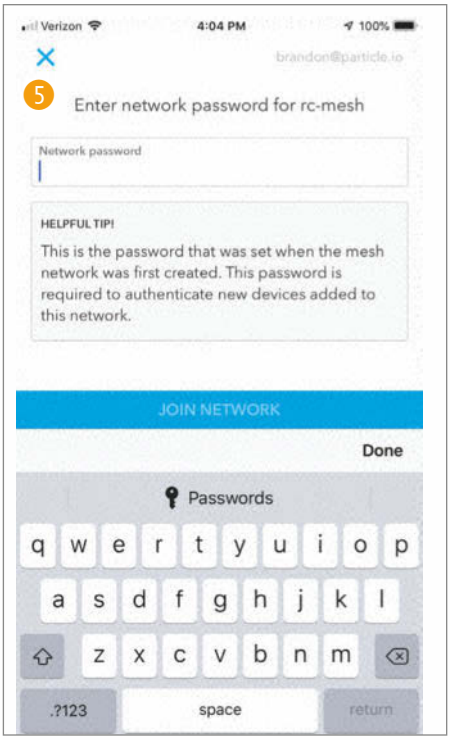
Programmierung der Tumbler

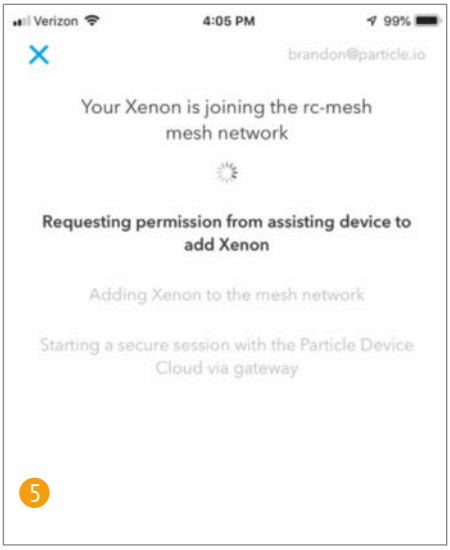
Nachdem die Xenons im Netzwerk eingerichtet sind, installieren wir die Firmware zum Ansteuern der Autos. Wir fangen mit einer einfachen Testsequenz an, die unsere Autos vorwärts, rückwärts, nach links und rechts fahren lässt. Der Upload auf den Xenon funktioniert genau wie beim Arduino.

Wie man sieht, besteht der Code hauptsächlich aus `analogWrite`-Anweisungen, mit denen eine Spannung auf die passenden Pins gegeben wird, um die Räder vorwärts oder rückwärts laufen zu lassen 6. Durch Ausprobieren habe ich entdeckt, dass meine Autos PWM-fähig sind: Ich kann pro Steuerungs-Pin auch eine geringere Spannung als digital-`HIGH` einstellen. Im Fall des Xenon sind es 3,3 Volt. Dann dreht sich das Rad entsprechend langsamer. Man beachte, dass ich fürs Links- und Rechtslenken andere Werte eingesetzt habe als fürs Vorwärts- und Rückwärtsfahren. Mit PWM (pulse-width modulation) können wir die unterschiedlichsten Geschwindigkeiten einstellen und unsere Spielzeugautos ziemlich komplexe Muster fahren lassen.

Mit dem fertigen Demo-Code, verpackt in eine Funktion namens `runDemo`, fehlt unseren RC-Autos jetzt noch Folgendes: ein Weg, wie wir diese Funktion über unser Mesh-Netzwerk aufrufen. Für den Austausch von Nachrichten im Netz bietet Particle eine API mit den Aufrufen `Mesh.publish` und `Mesh.subscribe`. Mit `Mesh.publish` kann man eine Broadcast-Nachricht, bestehend aus einem Namen und den eigentlichen Nutzdaten, an alle Knoten im Netzwerk senden. Mit `Mesh.subscribe` richtet man für die zu empfangenden Nachrichten (identifiziert durch ihren Namen) sogenannte Handler ein, die die gewünschte Aktion ausführen. Für unsere Xenons haben wir einen `Mesh.subscribe`-Aufruf, der bei Empfang einer Netzwerk-Nachricht unsere Demo-Sequenz abspielt 7.

Der fertige Code wird auf die Xenons in unseren RC-Autos geflasht, und dann kommen wir zum letzten Part: Wir programmieren das Gateway, um die Bewegungen unseres Auto-Schwarms zu koordinieren.





Listing 1

```
// Wheel pin mappings
int leftReverse = A0;
int leftForward = A1;
int rightForward = A2;
int rightReverse = A3;

// Speed and delay variables
int speed = 85;
int turnSpeed = 255;
int forwardDelay = 1000;
int backDelay = 1000;
int turnDelay = 2000;

void setup()
{
      // Set motor pins to outputs
      pinMode(leftReverse, OUTPUT);
      pinMode(leftForward, OUTPUT);
      pinMode(rightForward, OUTPUT);
      pinMode(rightReverse, OUTPUT);

      // Make sure each motor is off
      digitalWrite(leftReverse, LOW);
      digitalWrite(leftForward, LOW);
      digitalWrite(rightForward, LOW);
      digitalWrite(rightReverse, LOW);
}

void runDemo(const char *event, const char *data)
{
      allOff();

      goForward(speed);
      delay(forwardDelay);

      goBack(speed);
      delay(backDelay);

      // Max spin to raise up on the back tires
      turnLeft(turnSpeed);
      delay(turnDelay);

      allOff();
}

void allOff()
{
      analogWrite(leftReverse, 0);
      analogWrite(leftForward, 0);
      analogWrite(rightForward, 0);
      analogWrite(rightReverse, 0);
      delay(50);
}

void goForward(int speed)
{
      allOff();

      analogWrite(rightForward, speed);
      analogWrite(leftForward, speed);
}

void goBack(int speed)
{
      allOff();

      analogWrite(rightReverse, speed);
      analogWrite(leftReverse, speed);
}

void turnLeft(int speed)
{
      allOff();

      analogWrite(rightForward, speed);
}

void turnRight(int speed)
{
      allOff();

      analogWrite(leftForward, speed);
}

void loop()
{
      // Nothing needed here.
}
```

6

Programmieren des Mesh-Gateways

Der Gateway-Code für den Argon ist weniger komplex als der Code für die Xenons, besteht aber dennoch aus zwei Teilen. Per `Mesh.publish` starten wir die Demo-Sequenz für unsere RC-Autos. Dann müssen wir noch festlegen, wann das Gateway diese Nachricht an das Netzwerk senden soll. Wir könnten die Nachricht natürlich einfach senden, sobald das Gateway online ist oder nach einer bestimmten Verzögerungszeit – aber das wäre ja langweilig. Stattdessen machen wir uns eine weitere Methode aus der Particle-API zunutze – `Particle.function`.

`Particle.function` benötigt zwei Argumente: einen Namen und eine Handler-Funktion, die dann aufgerufen wird, wenn wir den Namen aus der jeweiligen Oberfläche – zum Beispiel Mobil- oder Web-App – auswählen. Für unseren Demo-Code enthält die Handler-Funktion einfach den `Mesh.publish`-Aufruf (8). Nachdem dieser Code auf dem Gateway installiert ist, sind unsere Swarmbots klar zum Start (9).

Listing 2

7

```
// Add to setup function Mesh.subscribe("run-demo", runDemo);
```

Steuerung per App

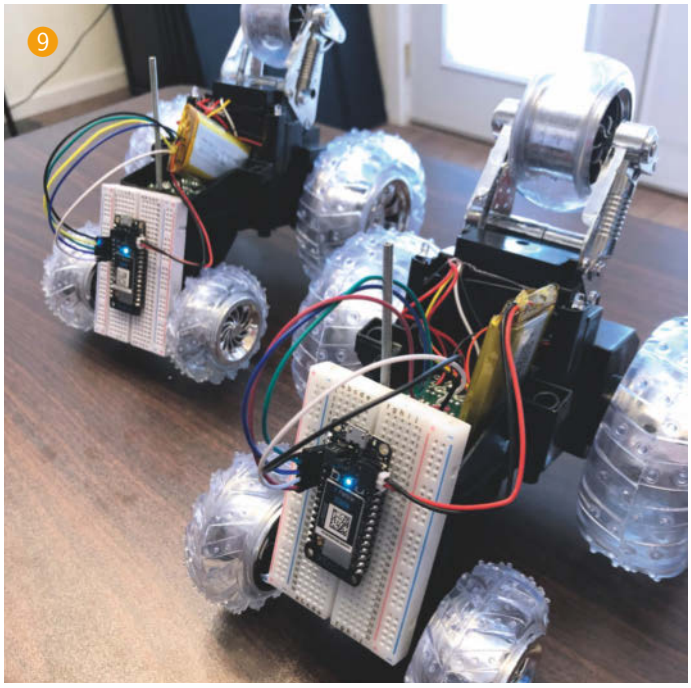
Particle-Funktionen können von jedem Gerät aufgerufen werden, das eine gesicherte Verbindung in die Particle Device Cloud hat – zum Beispiel die browserbasierte Particle Console (10), die Particle-Kommandozeilen-Schnittstelle (CLI) (11) oder eine eigene Anwendung. Es geht aber auch mit der Particle-App; (12) diese Option fanden wir am passendsten für unser Projekt.

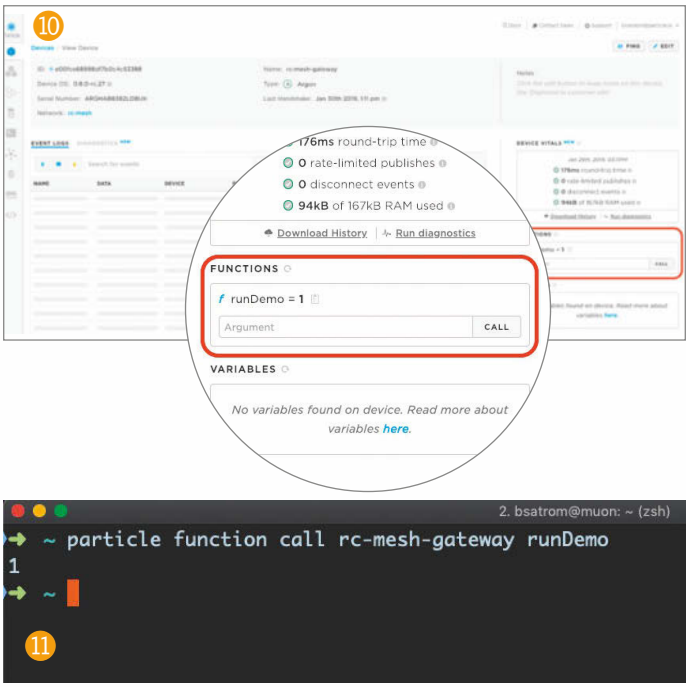
In der Mobil-App sehen wir eine Liste unserer Particle-Geräte. Hier klicken wir auf unser Gateway und dann auf den Data-Tab, wo wir unsere `runDemo`-Funktion wiederfinden. Nachdem wir die Xenons in Stellung gebracht haben, klicken wir auf die Funktion – und der RC-Schwarm fährt los.

Listing 3

8

```
void setup() {
      Particle.function("run
            Demo", runDemo);
}
int runDemo (String command)
{
      Mesh.publish("run-demo",
            NULL);
      return 1;
}
void loop() {
      // Nothing needed here
}
```





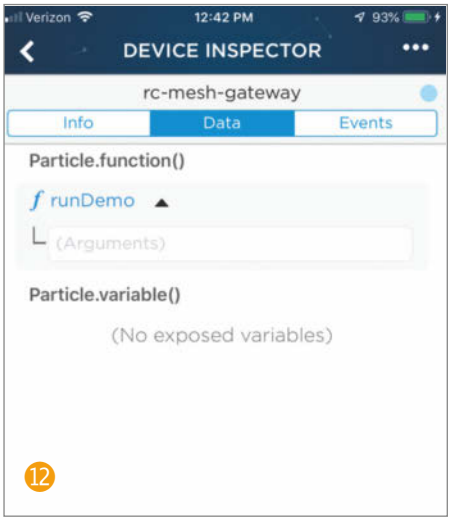
Der Demo-Code zeigt nur eine einfache, koordinierte Bewegungssequenz. Aber mit PWM und etwas Ausprobieren kann man Mesh-vernetzte RC-Autos in jeder erdenklichen Weise synchron zum Tanzen bringen.

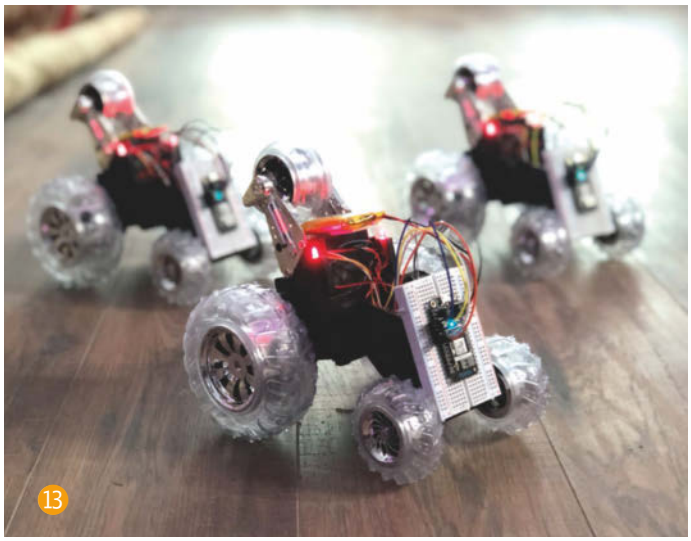
Herrsche über deinen Schwarm!

Da unsere RC-Auto-Roboter nun vernetzt sind, verwandeln wir sie doch gleich in richtige Swarmbots, die miteinander kommunizieren können. Ein typisches Schwarmverhalten ist das Leader-Follower-Prinzip. Der Anführer legt fest, wie sich der Schwarm verhalten soll, und kommuniziert dies direkt an alle Nachfolger 13 14. Zum Beispiel können wir den Anführer mit der normalen Fernsteuerung lenken, seine Pin-Spannungen auslesen und diese als Anweisung an alle Nachfolger senden 15.

Hierfür habe ich eines meiner drei Autos zum Anführer auserkoren und die Flotte mit neuer Firmware versehen. Anstatt nur die Motor-Pins anzusteuern, liest der Anführer einfach die von der Fernsteuerung gesendeten Analogwerte aus und versendet sie an die Empfänger im Netzwerk – mittels eines `Mesh.publish-Aufrufs`. Er enthält als Nutzdaten das zu bewegende Rad, die Drehrichtung und den entsprechenden Analogwert.

Auf die Nachfolger-Autos habe ich eine Firmware installiert, die für jeden Pin auf ein Mesh-Ereignis mit einem `analogWrite` und dem empfangenen Wert reagiert 16.





Listing 4

```
int32_t lastLeftRVal = 0;
int32_t lastLeftFVal = 0;
int32_t lastRightRVal = 0;
int32_t lastRightFVal = 0;

#define MIN_PIN_VAL 150
#define DRIVE_VAL 200

void setup()
{
      pinMode(leftReverse, INPUT);
      pinMode(leftForward, INPUT);
      pinMode(rightForward, INPUT);
      pinMode(rightReverse, INPUT);
}

void loop()
{
      checkPin(leftReverse, &lastLeftRVal,
"leftR");
      checkPin(leftForward, &lastLeftFVal,
"leftF");
      checkPin(rightReverse, &lastRightRVal,
"rightR");
      checkPin(rightForward, &lastRightFVal,
"rightF");
}

void checkPin(int pin, int32_t *lastVal, const
char *event)
{
      int32_t pinVal = analogRead(pin) / 16;

      if (pinVal > MIN_PIN_VAL)
            pinVal = DRIVE_VAL;
      else
            pinVal = 0;

      if (pinVal != *lastVal &&
            pinVal == DRIVE_VAL)
      {
            *lastVal = pinVal;

            Mesh.publish(event,
String(DRIVE_VAL));
      } else if (pinVal == 0 &&
*lastVal != 0) {
      *lastVal = 0;

      Mesh.publish(event,
String(0));
}
}
```

Bei den Autos habe ich die Antennen ausgebaut und die Leiterbahnen vom RX-IC durchtrennt, um sicherzustellen, dass Bewegungskommandos nur vom Anführer kommen können.

Auf dieser Grundlage haben wir dann noch ein paar andere Schwarm-Sequenzen gebaut: *Folge dem Anführer* ist eine einfache Vorwärts-Rückwärts-Demo mit End-Spin. Beim *Zersplittern* trennen sich die Autos in drei unterschiedliche Richtungen auf und treffen wieder zusammen. Eine weitere Variation ist *Folge dem Anführer und schiebe*. Hier gibt der Anführer den Nachfolgern ein Stopp-Kommando, fährt zwei Sekunden vorwärts, dreht sich um und fährt zwei Sekunden rückwärts. Dann befiehlt er seinen Nachfolgern, rückwärts zu fahren, während er weiter vorwärts fährt. Im *Wachposten-Modus* fahren die Autos einen rechteckigen Pfad mit 90-Grad-Drehungen. Und im *Orbit* umkreisen die Nachfolger einen stehenden Anführer.

Der Link zum vollständigen Projekt-Quellcode befindet sich im Infokasten.

Wir sind sehr gespannt auf Maker, die das Projekt nachbauen – vielleicht sogar mit anderen Sensoren und neuen Funktionen? Dem Anführer könnte man zum Beispiel mit einem PIR- oder Ultraschallsensor eine Kollisionserkennung spendieren. Oder man zeichnet über die Fernsteuerung bestimmte Bewegungsmuster auf, die man dann automatisch abspielen kann. *—rehu*

Listing 5

```
void setup()
{
      pinMode(leftReverse, OUTPUT);
      pinMode(leftForward, OUTPUT);
      pinMode(rightForward, OUTPUT);
      pinMode(rightReverse, OUTPUT);

      digitalWrite(leftReverse, LOW);
      digitalWrite(leftForward, LOW);
      digitalWrite(rightForward, LOW);
      digitalWrite(rightReverse, LOW);

      Mesh.subscribe("leftR", leftR);
      Mesh.subscribe("leftF", leftF);
      Mesh.subscribe("rightR", rightR);
      Mesh.subscribe("rightF", rightF);
}

void leftR(const char *event, const char *data)
{
      move(leftReverse, data);
}

void leftF(const char *event, const char *data)
{
      move(leftForward, data);
}

void rightR(const char *event, const char *data)
{
      move(rightReverse, data);
}

void rightF(const char *event, const char *data)
{
      move(rightForward, data);
}

void move(int pin, const char *speed)
{
      int32_t speedVal = atoi(speed);

      if (speedVal > 16) // Filter out noise from
the leader
      {
            analogWrite(pin, speedVal);
      }
      else
      {
            analogWrite(pin, 0);
      }
}

void loop() {}
```

BUSINESS 3D-DRUCKER „ULTIMAKER S5“

THE MOST VERSATILE PART OF YOUR PROJECT

Ultimaker

Entdecken Sie den einfach zu bedienenden Desktop-3D-Drucker mit einem großen Bauvolumen, das immer wieder präzise Teile in Industriequalität liefert.

- Technologie: Schmelzschichtungs-Verfahren (FDM/FFF)
- druckt Objekte bis zu 330 x 240 x 300 mm
- Material: 2,85 mm Filamente (z.B. Nylon, PLA, ABS, CPE, PVA)
- Schichtdicke: 0,02 - 0,60 mm
- Dual-Extrusion

BESTSELLER

Bestell-Nr.: **ULTIMAKER S5**

6.526,00

Der ideale 3D-Drucker für Einsteiger und Schulen

Der Select Mini Pro, mit einfach zu bedienenden Touchscreen, ist der perfekte 3D Drucker für anspruchsvolle Einsteiger. Die automatische Nivellierung des beheizten Druckbettes ermöglicht Ausdrucke ohne aufwändiges Einstellen.

- komplett montiert und druckfertig
- Technologie: Schmelzschichtungs-Verfahren (FDM/FFF)
- druckt Objekte bis zu 120 x 120 x 120 mm
- Material: 1,75 mm Filamente (ABS, PLA, PETG, Wood Fill, Copper Fill)
- Schichtdicke: 0,10 - 0,30 mm
- abnehmbare, magnetische Druckplatte

PREIS-TIPP

Bestell-Nr.: **MONOPRICE 133286** **199,95**

MONOPRICE

Desktop-Vakuumformer „FormBox“

Gestalten Sie Ihr Design mit dem 3D-Drucker und die Formbox formt in Sekundenschnelle eine 3D-Form als Vorlage für weitere Objekte.

- 200 x 200 mm Formfläche und 130 mm Ziehtiefe
- 1000 W Keramikheizkörper
- Temperatur: 160 - 340 °C
- Materialstärke: 0,5 - 1,5 mm

Bestell-Nr.: **MAYKU FORMBOX** **675,00**

MAYKU

Über 500 Filamente für jeden Anwendungszweck finden Sie online!

Gleich entdecken ▶ www.reichelt.de/filamente

TAGESPREISE! Preisstand: 22. 10. 2019

- Top-Preis-Leistungsverhältnis
- über 110.000 ausgesuchte Produkte
- Zuverlässige Lieferung – aus Deutschland in alle Welt.

Bestellservice: +49 (0)4422 955-333

www.reichelt.de

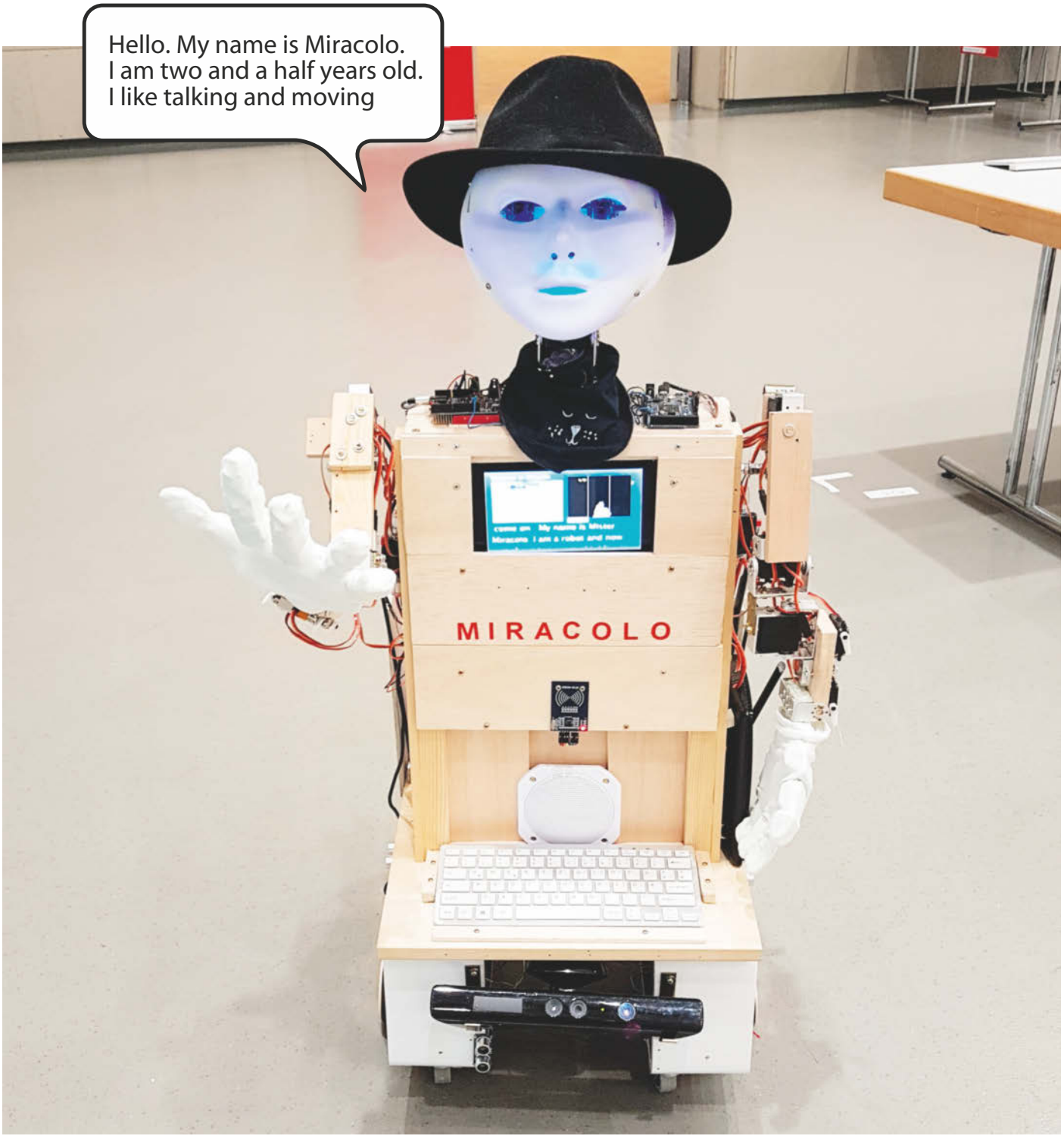
reichelt elektronik – The best part of your project

Es gelten die gesetzlichen Widerrufsregelungen. Alle angegebenen Preise in € inkl. gesetzl. MwSt. zzgl. Versandspesen für den gesamten Warenkorb. Es gelten ausschließlich unsere AGB (unter www.reichelt.de/agb, im Katalog oder auf Anforderung). Abbildungen ähnlich. Druckfehler, Irrtümer und Preisänderungen vorbehalten. reichelt elektronik GmbH & Co. KG, Elektronikring 1, 26452 Sande, Tel.:+49 (0)4422 955-333

Roboter Miracolo

Miracolo durfte sich schon zweimal auf der Maker Faire in Hannover vorstellen. Mit Sätzen wie *„Das ist ein guter Tag, ich freue mich, heute hier zu sein“* oder *„keep on smiling“* versuchte der Roboter dort, unter den Messegästen gute Laune zu verbreiten.

von Kai Schade



Miracolo kann sowohl deutsch mit zwei verschiedenen Stimmen als auch englisch sprechen. Während er spricht, lässt Miracolo sein blaues Licht in seinem Mund erstrahlen. Das Gesprochene zeigt er zusätzlich auf seinem eigenen Display an.

Für die Fortbewegung stehen Miracolo zwei Antriebsräder zur Verfügung, womit er vor- und rückwärts und sogar Kurven fahren kann. Seine Arme können unabhängig voneinander bewegt werden, wodurch es ihm sogar gelingt zu klatschen. Seine Hände sind optisch denen eines Menschen nachgebaut. Besonders stolz ist Miracolo darauf, dass er jeweils vier von fünf Fingern jeder Hand einzeln bewegen kann. Auch seinen Kopf kann Miracolo drehen sowie hoch und runter bewegen. Diese Bewegungsmöglichkeiten benutzt der Roboter ebenfalls, um seine Freunde mit Winken und Gesten zu begrüßen.

Gerne mag es Miracolo, mit eleganten Bewegungen zu zeigen, was er kann. Am liebsten jedoch dreht er sich auf der Stelle. Um zu sprechen, sich zu bewegen und zu fahren, braucht Miracolo keine Fernsteuerung. Das macht er viel lieber autonom. So ist es doch viel spannender, gerade nicht zu wissen, was Miracolo als Nächstes sagt oder macht.

Mit seinen Augen hält er nach Freunden Ausschau. Hierzu dient ihm seine Kamera. Alles, was er sieht, wird auf seinem Display angezeigt. Wenn jemand vor ihm steht, verwandelt er dessen Gesicht auf seinem Display in einen strahlenden Smiley. Auch hiermit möchte Miracolo gute Laune ausstrahlen. Die 3D-Kamera Kinect unterstützt seine Wahrnehmung. Hierdurch kann Miracolo die Bewegungen seiner Freunde ebenfalls gleich auf seinem Display anzeigen. Solange er niemanden erkennt, schweigt Miracolo. Umso größer ist die Freude, wenn er Personen bemerkt und sie dann plötzlich freundlich anspricht.

Der Kleine interessiert sich aber auch für die Fragen seiner Freude, die man ihm über die Tastatur stellen kann. Informationen wie zum Beispiel Alter, Hobbys und Lieblingsfarbe beantwortet Miracolo am liebsten.

Kurzinfo

- Sprechender autonomer Roboter selbst gebaut
- Bedienung per Tablet oder Gesten
- Zukunftsaussicht: Roboter lernt Lernen durch Python-Programm

Alles zum Artikel im Web unter *make-magazin.de/x97r*

1 Im Hauptmenü des Roboters stehen all seine Funktionen zur Verfügung.

Funktionsweise

Es war mir wichtig, nicht auf einen externen Computer angewiesen zu sein. Daher befinden sich alle Programme, die Miracolo zum Funktionieren braucht, in ihm. Als Hauptprogramm, das die Funktionen von Miracolo kontrolliert, habe ich *intern control* in der Programmiersprache Processing 3 geschrieben. Außerdem enthält Miracolo mittlerweile zusätzlich Programme in der Programmiersprache Python 3, die mit *intern control* zusammenarbeiten – dazu später mehr.

Sobald *intern control* startet, wird ein Menü auf Miracolos Display angezeigt 1. Die Darstellung erfolgt in Form von Kästchen. Die Menüs stehen teilweise für Funktionen, die noch in Entwicklung sind, denn mein Projekt Miracolo ist ein „Work in Progress“. Daher bekommt es hin und wieder Verbesserungen und neue Funktionen.

Der wichtigste Menüpunkt ist *self control*. Hier zeigt Miracolo links im Display sein Kamerabild und rechts seine Auswertung der Kinectdaten. Darunter schreibt Miracolo den Text, falls er gerade etwas gesprochen hat.

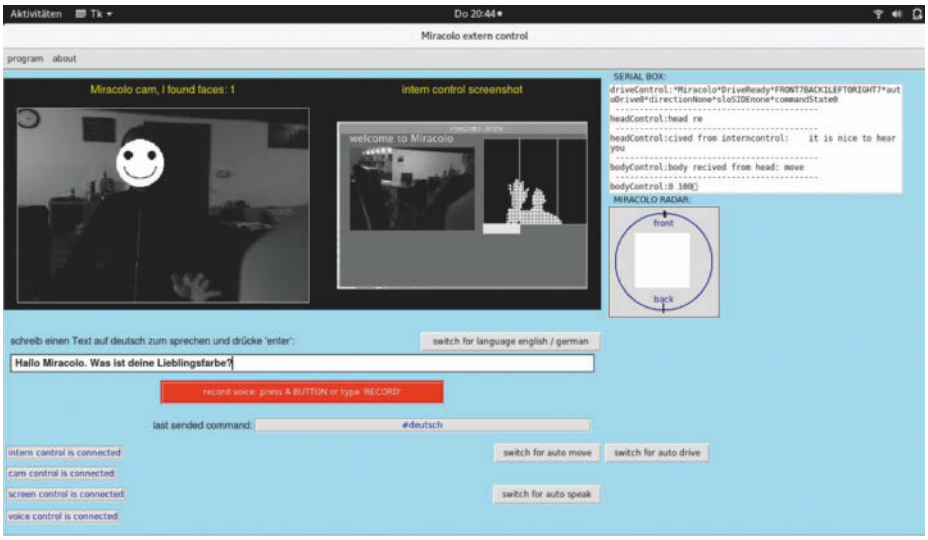
Sobald Miracolos Gesichtserkennung ein Gesicht wahrnimmt, zeichnet er einen Smiley über das Gesicht. Miracolo wird etwas Nettes sagen und sich dazu auch bewegen 2. Miracolo sagt auch gerne mal: „Ich sehe drei freundliche Gesichter.“ In der rechten Displayhälfte kann man seine eigenen Bewegungen sehen. Die Bewegungen werden aus mehreren weißen Punkten dargestellt. Wenn nun jemand vor Miracolo steht und seine Hand zu ihm ausstreckt, dann zeichnet Miracolo diese Hand oder Unterarm als gelbe Punkte in sein Display. Wer jetzt mit seiner Hand langsam und deutlich hin und her winkt, bekommt schon mal gesagt: „Danke, dass du mir winkst.“

Ein anderer Menüpunkt nennt sich *Find Card*. Hier zeigt Miracolo eines seiner Lieb-

2 Wenn Miracolo ein Gesicht erkennt, beginnt er freundlich zu sprechen und sich dabei zu bewegen.

3 Das Spiel *Find Card* erinnert an Hütchenspiele.

4 Durch das Programm *small control* kann man mit Miracolo auch per Tablet kommunizieren.



5 Das Programm *extern control* bietet Interaktion und Fernsteuerung zu Miracolo.

lingsspiele 3. Auf dem Display erscheinen drei Spielkarten, zwei schwarze und eine rote. Nun werden die Spielkarten verdeckt und die Karten gemischt. Danach muss man die rote Karte erraten. Dieses Spiel ist besonders bei Kindern beliebt. Dank der Kinect kann die Karte per Handbewegung gewählt werden.

Interaktion

Auf der vorigen Maker Faire hatte ich die Idee, das Programm *small control* ebenfalls in der Programmiersprache Processing 3 zu schreiben, das die Interaktion über ein Tablet ermöglicht. Auf dem Tablet erscheint ein Display mit einer Eingabezeile im oberen Bereich 4. Darunter befinden sich Bildsymbole und die Touchscreen-Tastatur. Damit kann man nun einen Text eintippen, den Miracolo nachspricht. Mit einem Fragezeichen am Ende signalisiert man Miracolo, dass er nicht nachsprechen, sondern antworten soll.

Man kann Miracolo fragen, was er gerne macht, ob er Tiere mag, wie alt er ist und wo er herkommt. Er gibt immer eine Antwort, auch wenn er die Frage nicht kennt. Antworten wie „Ich trinke gerne Kaffee“ gehören genauso dazu wie „Meine Name ist Miracolo. Ich bin jetzt zweieinhalb Jahre alt“. Wenn ich ihn frage, wie spät es ist, antwortet er: „Jetzt sind es 10 Stunden 9 Minuten, heute haben wir den 28. September 2019.“ Miracolo versteht auch englisch. Auf „What are your hobbies?“ könnte er antworten: „I like talking and moving.“ Miracolo gibt nicht immer dieselben Antworten. So bleibt es spannender.

Besonders beliebt war Miracolo auf den Maker Faires bei Kindern. Da die aber noch nicht alle schreiben können, habe ich zusätzlich die Bildsymbole in *small control* programmiert. Über acht Bildsymbole können unterschiedliche Anweisungen gesendet werden. Der *Smiley* veranlasst Miracolo, einen Witz zu erzählen. Drückt man auf das *Uhrsymbol*, sagt er die Uhrzeit. Mein persönlicher Favorit ist das Kästchensymbol. Hier beginnt Miracolo ein Spiel auf seinem Display, in dem drei Smileys in eine Reihe gebracht werden müssen. Über die Tablet-Steuerung via *small control* kann man seinen Smiley setzen. Aber aufgepasst: Miracolo spielt gut. Während des Spiels spricht Miracolo ein paar passende Kommentare.

Das Spielkartensymbol startet *findCard*, das ich bereits beschrieben habe. Auch hier spricht Miracolo seine Kommentare. Die Auswahl der Karte ist auch übers Tablet möglich.

Externe Kontrolle

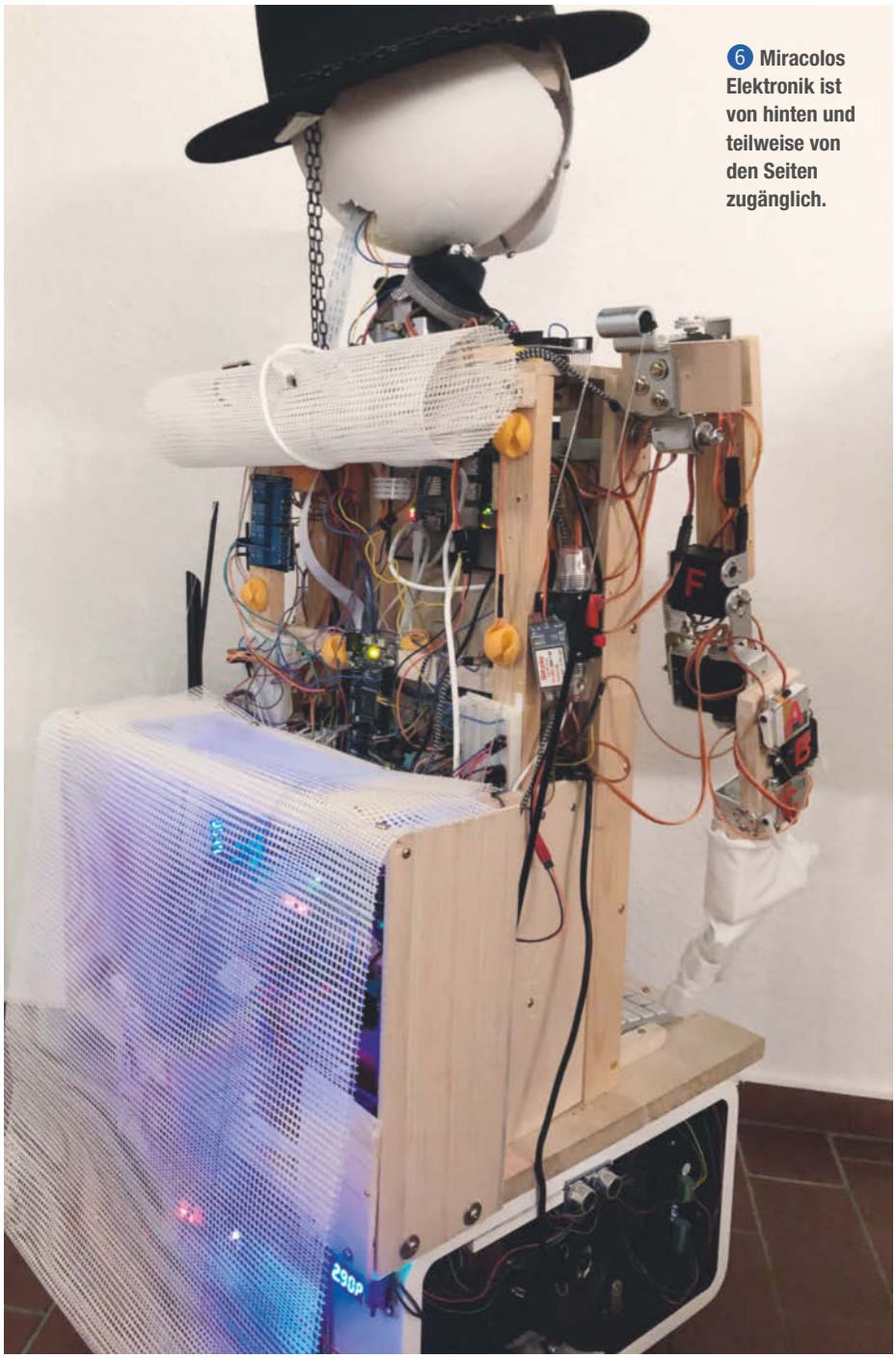
Auch wenn Miracolo vermutlich am liebsten autonom funktioniert, ist es nützlich, ihn kontrollieren und fernsteuern zu können.

Dazu dient das von mir geschriebene Python-3-Programm *extern control*. Es läuft unter Linux auf meinem Laptop. Damit kann ich Miracolos Funktionen überwachen und teilweise fernsteuern. Das Programm öffnet auf dem Display des Laptops ein eigenes Fenster. Darin wird Miracolos Kamerabild übertragen mitsamt der Smileys für Gesichter 5. Daneben wird ein Screenshot von Miracolos eigenem Display angezeigt. Dadurch brauche ich für die reine Überwachung keine zusätzliche VNC-Verbindung aufrechtzuerhalten. Rechts im Laptop-Display werden Informationen über die aktuellen Arbeitsabläufe Miracolos angezeigt. Dies ermöglicht, Fehler schneller zu erkennen. Ein kleines Radarfeld zeigt an, ob die Ultraschallsensoren etwas erkennen. Über *extern control* habe ich die Möglichkeit, die autonomen Funktionen für Sprache, Bewegung sowie Fahren ein- und auszuschalten.

Etwa in der Mitte des Laptop-Displays liegt die Eingabezeile. Hier kann man Texte eintippen, die von Miracolo nachgesprochen werden. Über einen Button kann man ihm mitteilen, ob der Text in Englisch oder Deutsch eingetippt wird. Wie bei *small control* können auch Fragen eingetippt werden. Das Fragezeichen signalisiert Miracolo, dass er etwas beantworten soll. Daraufhin spricht er seine Antwort, so wie im Abschnitt Interaktion beschrieben.

An meinem Laptop habe ich ein Gamepad angeschlossen. Über die Richtungstasten kann ich Miracolo vor- und rückwärts fahren sowie ihn auf der Stelle drehen lassen. Mit Hilfe der hinteren Tasten am Gamepad lasse ich Miracolo in Kurven fahren.

Derzeit noch in Arbeit befindet sich eine Funktion für Sprachübertragung. Mit Drücken der A-Taste am Gamepad beginnt das Programm *extern control* für 5 Sekunden, die Stimme aufzuzeichnen und anschließend an Miracolo zu übertragen. Daraufhin gibt Miracolo das Übertragene aus seinen Lautsprechern wieder. Das Programm *extern control* bedarf noch ein paar Korrekturen. Spaß macht es jetzt schon.



6 Miracolos Elektronik ist von hinten und teilweise von den Seiten zugänglich.

Technischer Aufbau

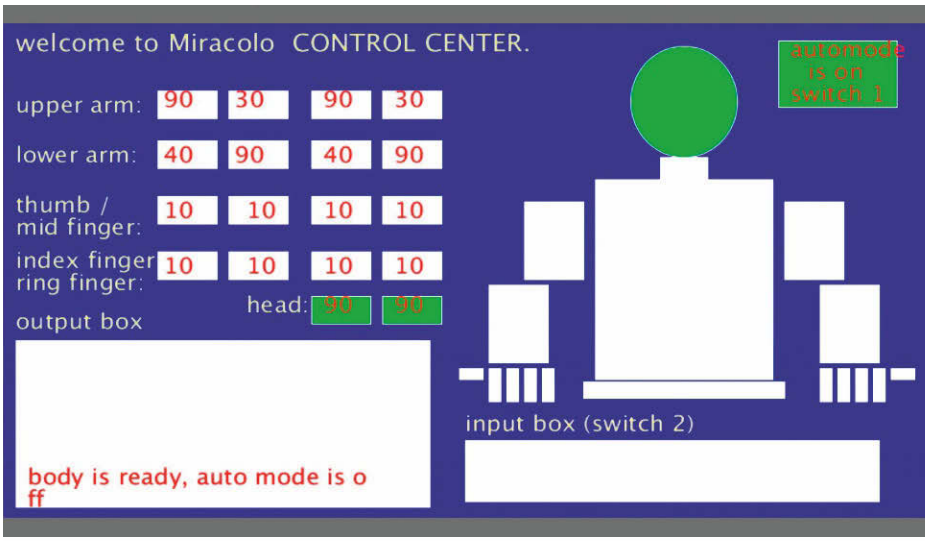
Mit dem Bau von Miracolo habe ich vor etwa zweieinhalb Jahren angefangen. Zu Beginn hatte Miracolo zwei Arduino-Due-Mikrocontroller. Ein Arduino ist seither zuständig, die Bewegungen zu steuern. Dessen Programm habe ich den Namen *body control* gegeben.

Der zweite Arduino steuert die Sprachausgabe sowie die Spracherkennung. Hier läuft mein Programm *head control*. Die Spracherkennung funktioniert, wenn überhaupt, nur in einem ruhigen Zimmer. Deshalb arbeite ich für die Interaktion lieber mit *small control*. Zunächst erhielt Miracolo seine zwei Arme mit Händen, die ich mit insgesamt 16 Servomotoren beweglich gemacht habe. Der Kopf wird mit zwei Servomotoren bewegt. Alle Servomotoren werden vom Arduino durch *body control* angesteuert. In Miracolos Kopf befindet sich ein Arduino Nano, der nur die Aufgabe hat, das blaue Licht beim Sprechen zu steuern.

Dann erhielt Miracolo einen Raspberry Pi 3, der mit meinem Programm *intern control* die zentrale Steuerung übernahm. Das autonome Sprechen und Bewegen funktioniert bereits allein mit den Arduinos. Vor einem Jahr fing ich an, das Fahrwerk zu bauen. Hierfür habe ich wieder einen eigenen Arduino Due eingesetzt und das Programm *drive control* geschrieben. Später kamen ein zweiter Raspberry Pi 3 sowie weitere Hilfsprogramme zur Unterstützung dazu.

Die Kommunikation zwischen *intern control* und den Arduinos erfolgt über serielle Verbindungen. Dagegen kommunizieren Miracolos Programme untereinander über Miracolos eigenes WLAN. Darüber erfolgt auch die Verbindung zu den externen Programmen *small control* und *extern control*.

Die Elektronik ist von hinten und teilweise von den Seiten zugänglich 6. Miracolo



7 **Bewegungen können im *control center* eingestellt werden.**

verfügt über drei Lautsprecher mit zwei verschiedenen Verstärkermodulen, eine Kamera, ein Kinectmodul, zwei Steppermotoren und diverse Sensoren. Die Grundkonstruktion besteht aus Holz. Zweifelsohne sind Miracolos Hut und Halstuch seine Erkennungszeichen.

Damit Miracolo nicht hungrig wird, bekommt er seine Energie aus drei Akkus, die über separate Stromkreise jeweils mehrere Komponenten in Parallelschaltungen mit Strom versorgen.

Mechanik und Bewegungen

Als ich vor über zweieinhalb Jahren angefangen habe, mich für Technik zu interessieren, hatte ich fast keine Vorkenntnisse. Ich hatte keine technische Ausbildung oder Studium. Erste Erfahrungen sammelte ich mit einem Experimentierkasten. Schnell hatte ich die Idee, einen eigenen Roboter zu bauen. Ein Grundgerüst sowie die Arme und Hände gehörten zu den ersten Entwicklungsschritten für Miracolo. So habe ich bei den Armen auch gleich mehrmals neu angesetzt, bevor ich mit dem Ergebnis zufrieden war. Auch die Hände sind bereits die zweite Version. Um Gewicht an den Armen zu sparen, habe ich zwei Servomotoren mit je etwa 25 Kilo Zugkraft an Miracolos Rückseite verbaut. Die Servomotoren für die Hände befinden sich an den Unterarmen. Bei Armen und Händen strebe ich noch Verbesserungen an.

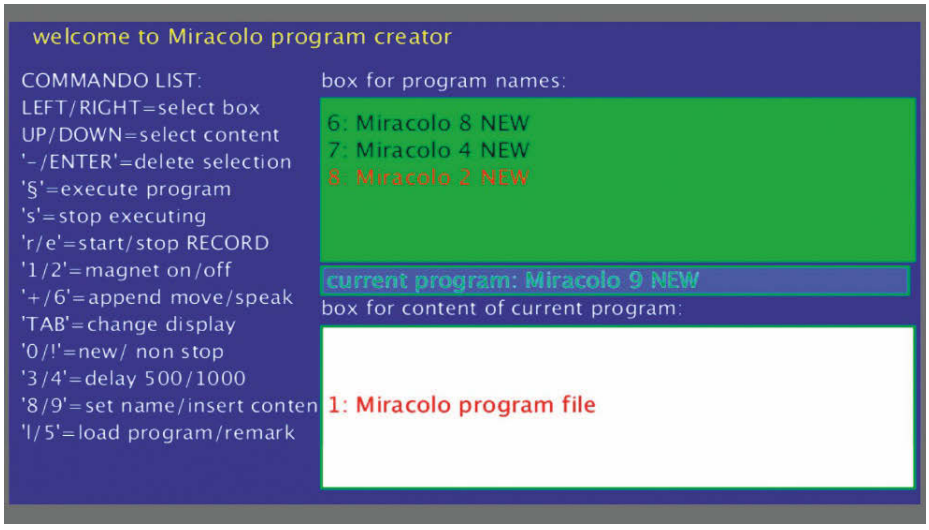
Die Bewegungen habe ich im Programm *body control* programmiert. Das Programm habe ich so konzipiert, dass Miracolo für eine Bewegung zuerst die Namen der Gelenke benötigt, um sie dem Servomotor zuzuordnen. Des Weiteren benötigt er die Angaben in Grad, wonach das Gelenk gestellt werden soll. So bedeutet die Anweisung `Schulter-links120` eine Drehung des linken Armes nach innen. Miracolo benötigt nur die Angaben für die Endpositionen. Die Bewegung von der Anfangsposition bis zur Endposition führt er selbständig aus. Er steuert seine Servomotoren so an, dass alle Gelenke scheinbar gleichzeitig bewegt werden.

Um die Positionen von Armen, Händen und Kopf zu testen, habe ich eine Art Fernsteuerung für die Bewegungen in Miracolos Programm *intern control* programmiert. Da sich dieses Programm auf dem Raspberry Pi 3 befindet, muss ich über eine VNC-Verbindung vom Laptop aus auf Miracolo zugreifen. Innerhalb von *intern control* führt nun der Menüpunkt *control center* zu einem Screen, der für jedes Gelenk ein Eingabekästchen vorsieht 7. So kann ich jedes einzelne Gelenk stellen, um zu testen, ob die Bewegung den Vorstellungen entspricht.

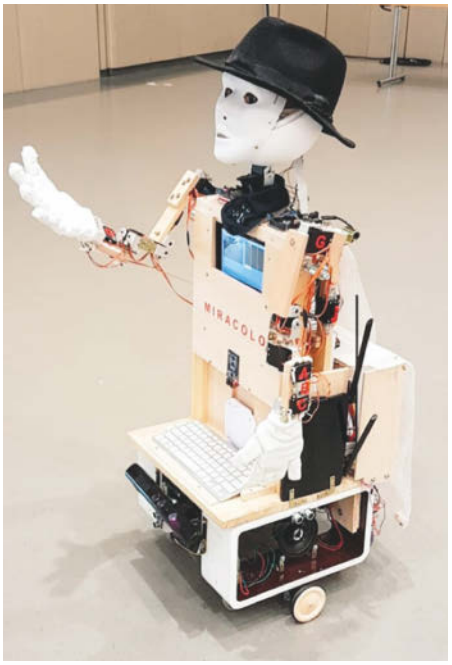
Miracolo wird auf dem Screen grafisch vereinfacht angezeigt. Das gerade angesteuerte Gelenk wird farblich angezeigt, um Missverständnisse zu vermeiden. Ist die gewünschte Position erreicht, kann diese per Knopfdruck in einem neuen Miracolo-Arbeitsprogramm gespeichert werden.

Arbeitsprogramme erstellen

Neben der Fähigkeit, autonom zu arbeiten sowie ferngesteuert zu werden, kann Miracolo auch frei programmierbare Arbeitsab-



8 **Mit dem *program creator* können eigene Arbeitsprogramme erstellt werden.**



9 **Das Kinect-Modul zur Bewegungserkennung sitzt im Sockel des Roboters.**

⑩ Miracolo sagt, dass er ein Gesicht gesehen hat. Das Gesicht verwandelt er in einen Smiley.

läufe abarbeiten. Diese Programmierung geschieht im *control center*, und dort im Untermenü *program creator* ⑧. Ich nenne diese Abläufe Arbeitsprogramme. Miracolo speichert sie als Textdateien auf seinem Raspberry Pi 3, wodurch diese auch ganz einfach über Texteditoren überarbeitet werden können.

Um Anweisungen in ein Arbeitsprogramm zu speichern, wird das entsprechende Arbeitsprogramm innerhalb des *program creator* ausgewählt und `record` betätigt. Für die Eingaben der Arbeitsanweisung steht innerhalb des *control center* eine Eingabezeile zur Verfügung. Hier kann Text eingetippt werden, der für das Arbeitsprogramm als Sprachanweisung gespeichert wird.

Eine Bewegung kann direkt als Anweisung in die Eingabezeile eingetippt werden. Zum Beispiel steht die Arbeitsanweisung `Finger1links65` für eine Bewegung des Ringfingers an Miracolos linker Hand. Bequemer geht es über die grafische Oberfläche: Hier genügt die Eingabe der Endposition der Bewegung im linken Eingabekästchen für *ring finger*. Außerdem ist es möglich, eine Arbeitsanweisung zu speichern, wodurch Miracolo eine Zufallsbewegung aus seinem eigenen Programm *body control* ausführt. Dies ist ganz nützlich und zeitsparend, wenn der Schwerpunkt des Arbeitsprogramms in den Sprachanweisungen liegt. So übernimmt Miracolo die Bewegungen wieder selbstständig. Beispielsweise kann man nette Grußbotschaften als Arbeitsprogramme speichern.

Fertige Arbeitsprogramme können im *program creator* ausgewählt werden. Das ausgewählte Arbeitsprogramm kann einmal oder in einer Schleife immer wieder durchlaufen werden. Derzeit ist es möglich, Bewegungen sowie Sprachanweisungen und Pausen zu programmieren. Künftig möchte ich die Programmierfähigkeiten noch um Anweisungen für das Fahrwerk des Roboters ergänzen.

Kamera und Kinect

In Miracolos Auge befindet sich seine Kamera, die mit dem zweiten Raspberry Pi 3 verbunden ist. Dort läuft mein Python-3-Programm *cam control*. Es ist dafür zuständig, ein Foto nach dem anderen zu machen und jeweils durch die Gesichtserkennungs-Funktion laufen zu lassen. Anschließend werden die gefundenen Gesichter mit einer Programmfunktion durch einen Smiley ersetzt. Die Fotos werden in Grautönen verarbeitet und über Miracolos eigenes WLAN ans Programm *intern control* gesendet. Sofern das externe Steuerungsprogramm *extern control* mit dem WLAN verbunden ist, werden die Fotos auch direkt an *extern control* gesendet. Durch die Übertragung in Grautönen bestehen die Fotos aus kleineren Datenmengen, wodurch Übertragungszeit gespart wird.

An Miracolos Fahrwerk sitzt das Kinectmodul ⑨, das an Miracolos ersten Raspberry PI 3 angeschlossen ist. Das nötige Programm in der Programmiersprache Processing 3 läuft als eigener Thread innerhalb des Programms *intern control*. Ich nenne diesen Thread, man kann es sich schon denken, *kinect control*. Zur Erkennung des Kinectmoduls benötigt Processing das Einfügen der Library `org.openkinect.processing.*`. Die Library sowie wichtige Informationen dazu finden Sie über den Link in der Kurzinfo.

Ich arbeite gerne mit der Programmiersprache Processing 3. Sie ist verständlich und arbeitet gut auf dem Raspberry Pi 3. Nicht alle Kinectmodule funktionieren mit dem Raspberry Pi 3, da er hierbei schnell an seine Leistungsgrenzen kommt. Das von mir verwendete Kinectmodul arbeitet mit einer Auflösung von 640 × 480 Punkten. Das Programm bekommt vom Kinectmodul zu jedem Bildpunkt die Entfernung mitgeteilt. Hierdurch können auch Bewegungen wahrgenommen werden. Um Ressourcen zu sparen, nutze ich zur Auswertung in meinem Programm nur jeden fünften Punkt. Das reicht für Miracolos Zwecke schon aus. Bisher hat Miracolo gelernt, winkende Handbewegungen zu erkennen. Das Erkannte zeigt Miracolo in seinem Display mit weißen und gelben Punkten ⑩, gelb für nahe und weiß für ferne Punkte.

Lernen lernen

Als Work-in-progress-Projekt arbeite ich bei Miracolo immer wieder an Verbesserungen, Korrekturen und neuen Funktionen. Dies geschieht ausschließlich in meiner Freizeit, da Miracolo mein Hobby ist. Deshalb ist Miracolo kein fertiges Projekt, sondern ein ständiger Arbeitsprozess. So werde ich künftig Miracolos Fahrverhalten nachbessern und weiter an der Spracherkennung arbeiten. Ich teste noch aus, ob Miracolo seine Steppermotoren behalten wird oder ob er künftig Gleichstrom-Antriebsmotoren erhält.

Das neueste Projekt für Miracolo ist mein Programm *Miracolo learning*. Damit soll Miracolo befähigt werden, seine Datenbanken selbstständig zu erweitern. Die Interaktion mit Miracolo soll dadurch künftig verbessert werden.

Wenn Miracolo etwas gefragt wird, greift er bisher auf seine internen Datenbanken zu. Miracolo ist aber sehr wissbegierig und möchte gerne lernen, wie er seine Datenbanken selber erweitern kann. *Miracolo learning* habe ich in der Programmiersprache Python 3 geschrieben. Das Programm steht noch ganz am Anfang, doch ein erster Schritt ist bereits gelungen. Es fordert dazu auf, eine Eingabe zu machen. Ich tippe zum Beispiel die Frage ein: „Was ist ein Flugzeug?“ Daraufhin kommt die Antwort: „Zu Flugzeug kann ich leider nichts sagen.“ Als Nächstes tippe ich ein: „Ein Flugzeug fliegt hoch.“ *Miracolo learning* antwortet: „Ok, du schreibst, ein Flugzeug fliegt hoch.“ Nun tippe ich erneut ein: „Was ist ein Flugzeug?“ und *Miracolo learning* gibt nun als Antwort: „Zu Flugzeug fällt mir ein, fliegt hoch“.

Dies mag ein sehr kleiner Schritt sein. Für Miracolo bedeutet es die Fähigkeit, in Zukunft seine Datenbanken selber erweitern zu können. Damit wird er in die Lage versetzt, selbstständig Neues zu lernen. Das Programm wird mir noch sehr viel Arbeit, aber auch Spaß bereiten.

Miracolo freut sich schon, auf der nächsten Maker Faire in Hannover neue Freunde kennenzulernen. —*hgb*

iGoBot

Der iGoBot ist ein Roboter, der eigenständig das asiatische Brettspiel Go spielt. Er erkennt mit einer Kamera die Züge des menschlichen Gegners und setzt eigene Spielsteine mit einem Roboterarm.

von Daniel Springwald

Kurzinfo

» Stationärer Roboter mit drei Linear-Achsen
» Spielsteinspender mit Vereinzelungsfunktion
» OpenCV-Bilderkennung mit Haar-Cascade

Alles zum Artikel im Web unter *make-magazin.de/xbvy*

Mehr zum Thema

» Carsten Meyer, MaXYposi, Make 1/17, S. 12

Beim Go-Spiel legen zwei Spieler jeweils abwechselnd linsenförmige Spielsteine auf ein Brett. Ziel ist, am Ende möglichst große Gebiete des Spielbrettes umzäunt zu haben, wobei die gesetzten Steine die Zäune bilden. Die genauen Regeln des Spiels können etwa beim Deutschen Go-Bund angeschaut werden. Sie sind so simpel, dass selbst Kinder innerhalb von Minuten eine erste Partie spielen können. Gleichzeitig fordert das Spiel von erfahrenen Spielern höchst anspruchsvolle und strategische Überlegungen. Dass man Go zudem auf großen wie kleinen Spielbrettern mit unterschiedlicher Anzahl Spielfelder spielen kann, macht den Einstieg noch einmal einfacher.

Das Spiel begleitet mich bereits seit meiner Jugend und die Idee zu einem Go-spielenden Roboter hatte ich erstmals in den 1990er Jahren. Durch den Raspberry Pi und die günstig verfügbaren Bauteile ist dies nun in der Preislage eines Hobby-Projekts möglich geworden. Der Name iGoBot ist übrigens nicht an Apple Produkte angelehnt: In Japan heißt das Spiel „iGo“ – das „i“ wird dabei als „i“ und nicht als „ei“ gesprochen.

Ziel und Aufbau

Das Ziel beim Bau des Roboters war, dass er möglichst authentisch Go spielt und der menschliche Spieler nichts tun muss, was er bei einer Partie gegen einen anderen Menschen nicht auch tun müsste. Der Spieler sollte also keine Spielfeld-Koordinaten auf einer Tastatur eingeben und auch keine Spielsteine für den Roboter setzen oder vom Brett nehmen müssen.

Damit ein Go-Roboter selbstständig am Spiel teilnehmen kann, benötigt er folgende Fähigkeiten:

- Erkennen, wohin der Spieler einen Stein gesetzt hat
- Den nächsten sinnvollen Spielzug berechnen
- Einen Spielstein aufnehmen
- Den Stein auf das Spielbrett setzen
- Geschlagene Steine vom Spielbrett nehmen

Die Grundlage des Go-Roboters ist der IKEA-Tisch „Lack“. Da Go ursprünglich auf dem Boden sitzend gespielt wird, sind die Beine des Tisches gekürzt und mit 3D-gedruckten Füßen von Thingiverse (siehe Link in der Kurzinfo) versehen. Das Go-Spielbrett kann später einfach auf den Tisch gestellt werden und wird von zwei kleinen Metallwinkeln in Position gehalten.

Der fertige Aufbau besteht aus drei Achsen, die an den Seiten des Tisches sowie darüber angebracht sind. Ein eigens konstruierter Steinspender aus 3D-gedruckten Teilen legt dem Roboter die Go-Steine einzeln zurecht. Sein Greifer platziert sie passend auf dem Spielfeld – das der Roboter dank der OpenCV-Bilderkennung überblickt. Auf einem Raspberry Pi läuft die Steuerung aller Systeme, von den Motoren über die Bilderkennung bis zum Spielalgorithmus.

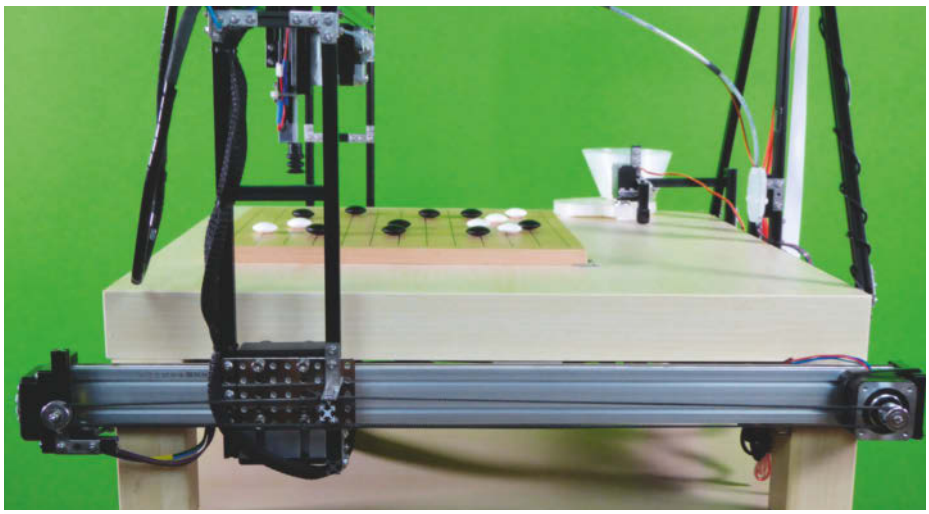
Spindeln aus China?

Ursprünglich wollte ich für die Z-Achse links und rechts vom Tisch jeweils eine Linearführung mit Spindelachsen verwenden. Diese konnte ich sehr günstig in China bestellen, sie waren aber leider nicht zu gebrauchen: Zum einen waren beide Spindelachsen leicht verbogen – vermutlich, weil sie für die lange Reise lediglich in einem Plastiktüten-Versandumschlag verpackt wurden. Zum anderen war die Untersetzung durch die Spindel so groß, dass sich die Achse nur sehr langsam bewegt hätte. Das Setzen eines Spielzugs hätte selbst bei maximaler Schrittmotorgeschwindigkeit bis zu 20 Sekunden gedauert. Immerhin konnte ich die Führung und den Schlitten nutzen.

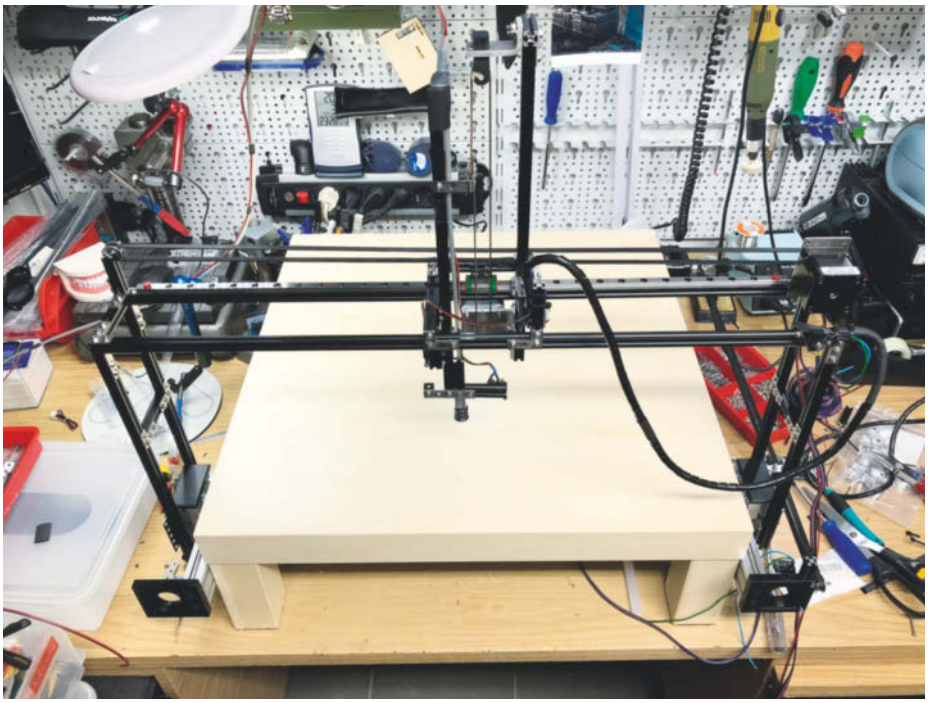
X- und Y-Achsen

Die Führung der X-Achse und die Y-Achse (zum Anheben der Spielsteine) habe ich aus Makerbeam-Elementen konstruiert. Diese T-Profile aus Aluminium habe ich auf der Maker Faire 2013 in Hannover entdeckt und war direkt begeistert. Sie finden seitdem in fast jedem meiner Projekte Verwendung. Angetrieben werden auch diese beiden Achsen durch einen Schrittmotor per Zahnriemen.

Den elektronischen Kern des Roboters bildet ein Raspberry Pi 3 mit einem aufgesetzten GrovePi+ Shield von Seeed. Über die Aufsteckplatine können bis zu 15 Bauteile mit Grove-Steckern angeschlossen werden. Die Verwendung des I²C-Bus vereinfacht den Aufwand für das Anschließen und Ansteuern der Schrittmotoren sowie das Abfragen der Endstopp-Schalter deutlich. Die drei Motorsteuerungen stammen ebenfalls von Seeed. Für die Stromversorgung nutze ich ein Schaltnetzteil mit 9 Volt und 5 Ampere; für die Schrittmotoren sowie für den Raspberry Pi und die Beleuchtung ein USB-Netzteil mit 5 Volt.



Die Mechanik der Z-Achse links und rechts vom Tisch basiert auf kugelgelagerten Schlitten und wird durch einen Zahnriemen per Schrittmotor angetrieben.



Alle drei Achsen auf dem IKEA-Tisch

Vakuum mit Lego und Miniatur-Servo

Pneumatik lite

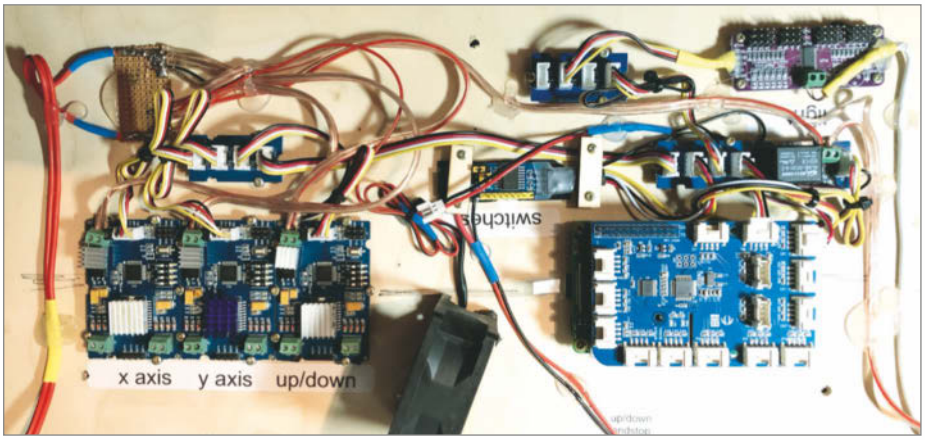
Zum Aufnehmen eines Steines befindet sich an der senkrechten Achse ein Sauggreifer mit angeschlossenem Luftschlauch. Meine erste Idee war, das Ansaugen mit pneumatischen Industriekomponenten zu realisieren. Durch den notwendigen Kompressor und Venturi-Vakuum-Erzeuger stellte sich diese Lösung aber schnell als ziemlich sperrig und laut heraus.

Glücklicherweise sind Go-Steine sehr glatt und dichten den Sauggreifer beim Ansaugen nahezu vollständig ab. Dadurch konnte ich eine deutlich simplere Lösung verwenden: Diese besteht aus einem Lego-Pneumatik-Kolben und einem Modellbau-Servo. Der Servo zieht oder schiebt den Kolben und erzeugt damit den nötigen Unterdruck, um den Stein sicher zu halten.

Steinspender

Das Entnehmen von Go-Steinen aus einer traditionellen Go-Dose wäre schwierig zu lösen, da der Sauggreifer den Stein möglichst flach treffen muss, um ihn halten zu können. In der Dose liegen die Steine aber kreuz und quer. Stattdessen sollte jeweils ein Stein an der immer gleichen Stelle bereitliegen. Damit der Spieler diesen Stein nicht bei jedem Zug platzieren muss, habe ich an einer Vorratsmöglichkeit gearbeitet. Zunächst versuchte ich eine Art Rutsche. Dabei verkanteten sich die Steine aber regelmäßig oder stockten. Zudem müsste die Rutschbahn unhandlich lang sein, sofern sie eine angemessene Anzahl Steine fassen sollte.

Die jetzige Lösung ist ein 3D-gedruckter Stein-Vereinzeler, der von einem Modellbau-Servo angetrieben wird. Der obere Teil ist ein Trichter, in welchen ich die Steine gebe. Abgeschlossen wird er unten von einer beweglichen Platte, die von einem Servo vor- und zurückgeschwenkt werden kann. In der Platte befindet sich ein Loch, in das exakt ein Go-Stein hinein passt. Bewegt sich die Platte per Servo einmal unter dem Trichter hin und her, liegt anschließend in der Vertiefung ein Go-Stein bereit. Den Stein-Vereinzeler habe ich mit dem kostenlosen Konstruktionsprogramm OpenSCAD entworfen. Die Druckdateien liegen auf Github zum Download bereit (siehe Link in der Kurzinfo).



Die elektronischen Komponenten sind unter dem Tisch angebracht.

Kamera-Erkennung

Um zu erkennen, welchen Spielzug der menschliche Spieler gesetzt hat, beobachtet der Roboter das Brett senkrecht von oben. Dazu nutze ich ein Raspberry-Pi-Kameramodul – allerdings mit einem Anschlusskabel von einem Meter Länge. Trotz fehlender Abschirmung funktioniert das Kabel bei dieser Länge noch problemlos. Da das Kamerabild ausgewertet wird, wenn die Schrittmotoren stillstehen, kommt es auch nicht zu Störungen durch deren Antriebsströme.

Im Trend läge es natürlich, Deep Learning und neuronale Netze zum Training und Er-

kennen der Go-Steine zu verwenden. Es geht aber auch simpler mit einem Machine-Learning-Algorithmus für Objekterkennung, der *Haar Cascade*. Haar Cascades kann man über die bereits 2006 erschienene freie Bibliothek zur Bildverarbeitung OpenCV einbinden. Der Begriff Haar hat hier nichts mit Haaren zu tun, sondern rührt von seinem Erfinder her, dem ungarischen Mathematiker Alfréd Haar.

Der häufigste Anwendungsfall für Haar Cascades ist wohl Gesichterkennung – dafür liefert OpenCV bereits fertige Beispiele mit. Die Erkennung von Go-Steinen musste ich dem System hingegen erst noch beibringen. Dazu habe ich Beispielbilder geschossen und daraus ein Modell in OpenCV erstellt. Für das Training habe ich drei Arten von Fotos benötigt:

1. Bilder, auf denen ein weißer Spielstein zu sehen ist.
2. Bilder, auf denen ein schwarzer Stein abgebildet ist.
3. Bilder, die keine Spielsteine enthalten.

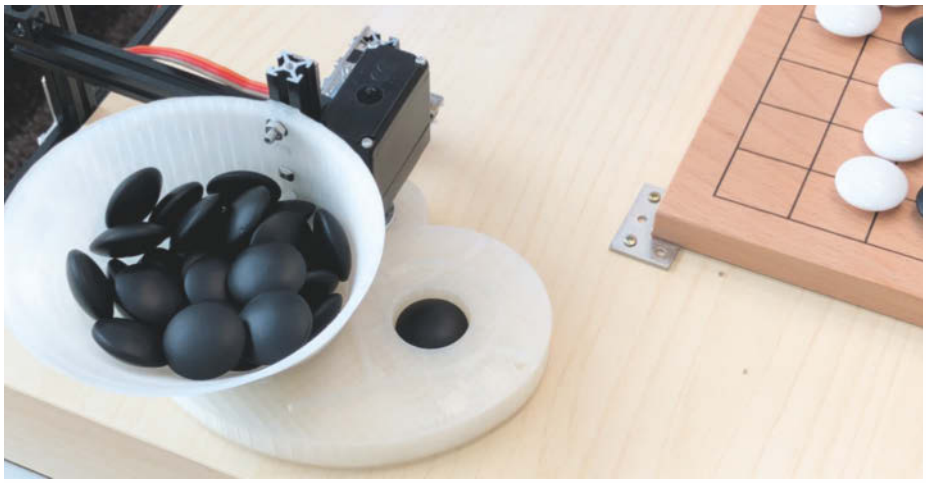
Um etwa das Modell zur Erkennung weißer Steine zu trainieren, lädt man die Bilder der weißen Steine als positive Beispiele in das Training und die restlichen Bilder als negative Beispiele. Ebenso wie beim Trainieren anderer Deep-Learning-Algorithmen gilt hier für die Anzahl der verwendeten Bilder: Viel hilft viel.

Nachdem ich für jede der drei Kategorien mehrere Hundert Bilder geschossen hatte, konnte ich das Modell trainieren. Das macht man üblicherweise von Hand auf der Kommandozeile. Das wird schnell recht aufwendig, da es meist einige Durchläufe zum Herausfinden der optimalen Parameter braucht und die Ergebnisse immer wieder visuell überprüft werden müssen. Deutlich komfortabler klappt dies mit dem freien Werkzeug *Cascade Trainer GUI* (siehe Kurzinfo). Es ergänzt das Training um eine grafische Oberfläche, in der man die Qualität der Ergebnisse direkt überprüfen kann.

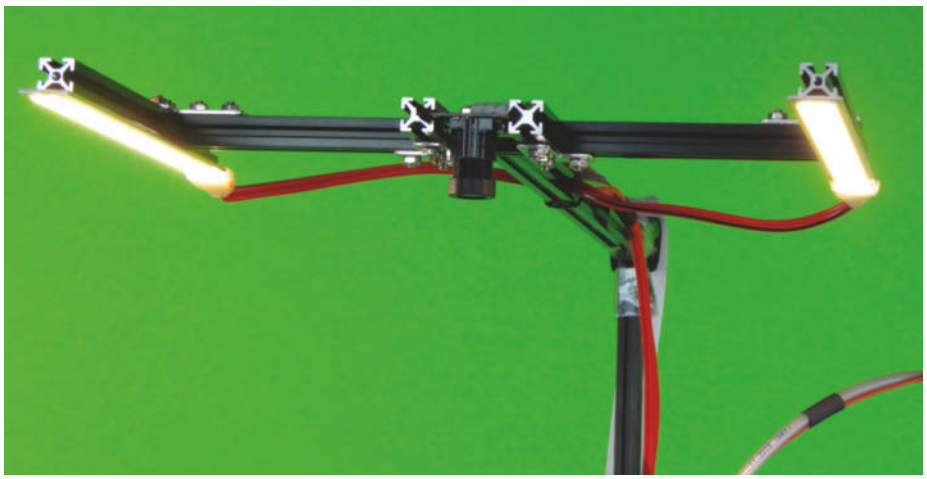
Die Erkennung funktionierte unter halbwegs gleichmäßigen Lichtverhältnissen auf Anhieb sehr gut. Deutliche Schatten auf dem Brett können das Modell aber durchaus verwirren. Um dieses Problem zu minimieren, habe ich neben der Kamera zwei größere LED-Streifen montiert, die das Spielbrett gleichmäßig von oben beleuchten. Nur bei wirklich starkem, senkrechtem Licht – wie direktem Sonnenlicht – muss der Roboter noch von oben mit einer halbtransparenten Folie abgedeckt werden, damit die Kameraaufhängung keinen Schlagschatten auf das Spielbrett wirft.

OpenCV

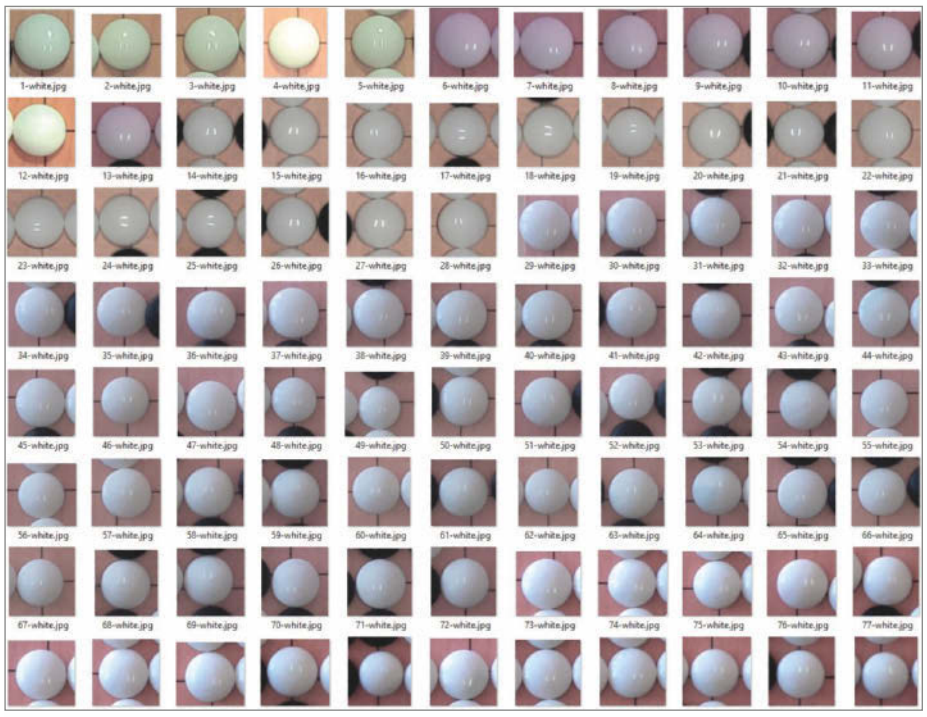
Die Einbindung von OpenCV in ein Python-Programm ist recht einfach und benötigt nur wenig Programmcode. Zuvor muss aller-



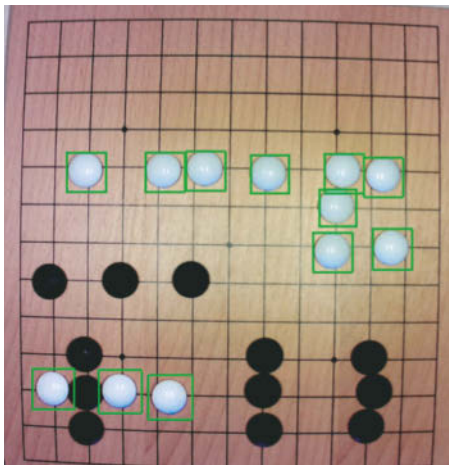
Der Steinvereinzeler



Neben der Kamera ist die Beleuchtung angebracht.



Eine kleine Auswahl der Trainingsbilder



Von OpenCV erkannte Rechtecke für das Modell „Weiße Steine“

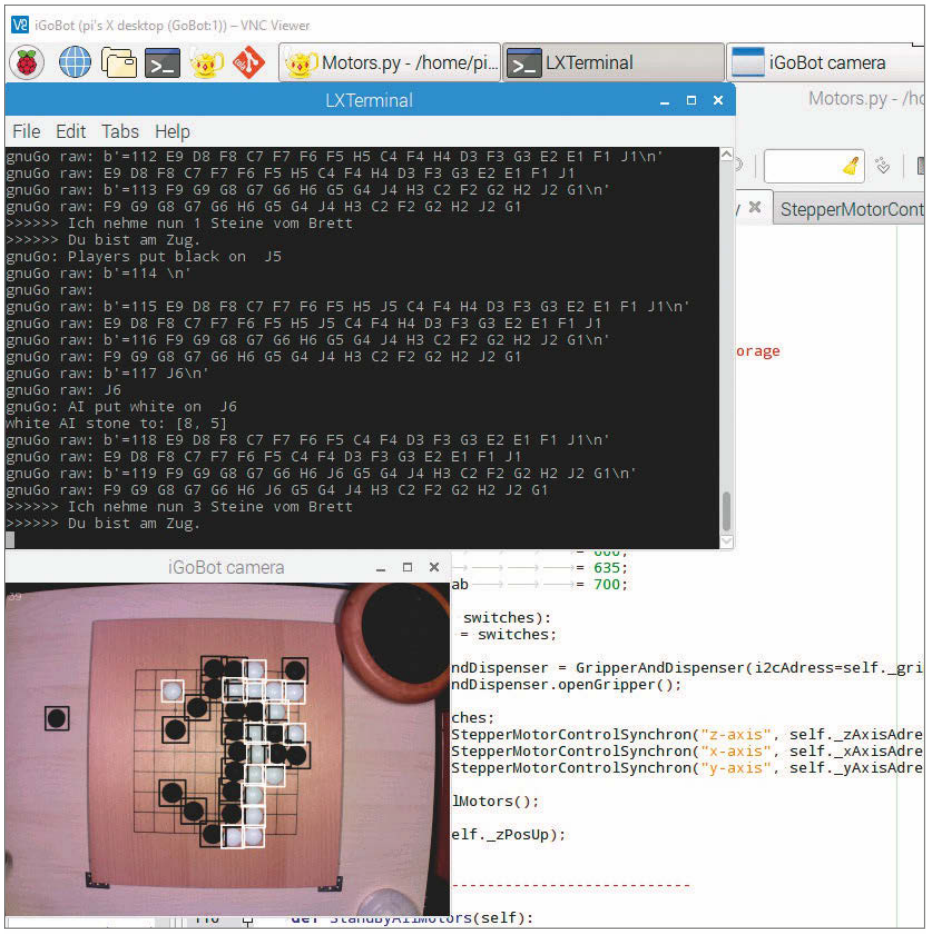
dings OpenCV für Python installiert werden, was beim Raspberry Pi durchaus knifflig sein kann. Hilfreiche Schritt-für-Schritt-Anleitungen finden Sie über den Link in der Kurzinfo.

In OpenCV können die trainierten Haar-Modelle unter Angabe des Dateinamens mit der Methode `cv2.CascadeClassifier(filename)` geladen werden. Zur Überprüfung des Kamerabilds wird dann im Spiel nacheinander für jedes Modell die Methode `detectMultiScale` aufgerufen. Als Ergebnis erhält man eine Datenstruktur, die die erkannten Spielsteine als Rechtecke mit Angaben für Größe und Position enthält, gemessen in Pixel.

Künstliche Spiel-Intelligenz

Die Schwerigkeit eines Go-Spiels wird in Schüler (Kyu), Meister (Dan) und Profis (ebenfalls Dan) unterteilt. iGoBot spielt mit einer Spielstärke von etwa 8 Kyu – und damit deutlich besser als ich, weshalb ich von ihm Vorgabesteine gewährt bekomme. Seine Intelligenz stammt aus der freien Software GNU Go, die es für zahlreiche Betriebssysteme zum Herunterladen gibt, so auch für den Raspberry Pi (siehe Kurzinfo).

GNU Go ist eine reine Konsolenanwendung, sie kommuniziert also primär über Textein- und -ausgabe. Um dies für den Roboter nutzbar zu machen, kann man GNU Go aus Python heraus als neuen Prozess starten und die Ein- und Ausgaben entsprechend umleiten. Die Syntax des auszutauschenden Textes ist sehr übersichtlich und kann in Python leicht erzeugt und ausgewertet werden. Der Ablauf und die übertragenen GNU-Go-Befehle werden dabei zu Anschauungs- und Testzwecken auf dem Raspberry-Pi-Bildschirm ausgegeben.



Statusanzeige während des Spiels

Kalibrierung

Nach dem Start des Programmes müssen zuerst die Achsen und dann die Kamera kalibriert werden. Zur Justierung der Achsen fahren die Schrittmotoren einmal bis zu den Endstopp-Schaltern. Der Abstand des Spielfeldes relativ zu den Endstopp-Schaltern muss zuvor einmal experimentell ermittelt und im Programmcode hinterlegt werden. Gleiches gilt für die Position, an welcher der Steinspender jeweils einen Spielstein bereitstellt. Die Entfernungen – gemessen in Schrittmotor-Schritten – können dann im Programmcode fest hinterlegt werden. Gleiches gilt für die Position, an welcher der Steinspender jeweils einen Spielstein bereitstellt.

Die Kamera muss hingegen vor jedem Spiel kalibriert werden. Dazu erwartet der iGoBot fünf Spielsteine in einem bestimmten Muster auf dem Brett: Vier schwarze Steine jeweils in den Ecken des Brettes und ein weißer Stein in der Mitte.

Die OpenCV-Erkennung wartet so lange, bis der Spieler diese fünf Spielsteine auf das Brett gelegt hat. Aus den erkannten Steinen berechnet das Programm anschließend die durchschnittliche Größe eines Spielsteins und die maximalen Außenpositionen, auf welchen Steine liegen können. Diese Werte sind wichtige Korrekturinformationen, denn manchmal reagiert die OpenCV-Erkennung über und erkennt Spielsteine, wo gar keine sind. So sortiert das Programm die fehlerhaften Treffer aufgrund ihrer ungewöhnlichen Größe oder Position aus.

Außerdem werden die möglichen Positionen der Spielsteine bei der Umrechnung von Pixeln in Brett-Koordinaten genutzt, was ich mit simpler Division erledige. Die Anzahl der Spielfelder ist im Programm hinterlegt und es muss nur noch ermittelt werden, in welchem Quadranten des Spielfeldes der Stein erkannt wurde.

Ursprünglich hatte ich vor, noch eine Transformationsmatrix zu erzeugen, um die Pixelpositionen der erkannten Spielsteine auf die echten Koordinaten der Spielfelder abzubilden. Da die Raspi-Kamera aber keine nennenswerte Linsenverzerrung hat und durch die mittige Position kein Trapezeffekt auftritt, habe ich darauf verzichtet. Falls das etwa durch den Einsatz einer anderen Kamera notwendig würde, gibt es dazu eine Anleitung inklusive Quellcode von Carlos E. Vidales (siehe Kurzinfo). Diese habe ich in einem früheren Projekt erfolgreich umgesetzt.

Spielablauf

Nach der Kalibrierung bittet iGoBot per Sprachausgabe darum, Vorgabesteine zu legen und dann „die Taste“ zu drücken. Vorgabesteine werden beim Go als Handicap verwendet, um Unterschiede in der Spielstärke der Spieler auszugleichen.

Bei der genannten „Taste“ handelt es sich um einen großen Taster. Er ist umgeben von 12 RGB-LEDs, die grün leuchten, sobald der Spieler an der Reihe ist. Dadurch wird die Bilderkennung der Spielsteine nicht permanent durchgeführt und man kann sich darauf verlassen, dass bei der Aufnahme keine Hände oder Arme im Bild sind.

Damit der Roboter etwas persönlicher wirkt, setze ich bei der Sprachausgabe weitere RGB-LEDs ein. Diese sind in Form eines Gesichtes angeordnet, das den Mund bewegt, während Sprache ausgegeben wird. Die Platine McRoboFace ist ein Projekt von Robin Newman und ist fertig bestückt beim britischen Elektronikhändler 4tronix erhältlich.

Wenn der Spieler seinen Stein gesetzt und die Kamera diesen erkannt hat, sendet das Programm eine simulierte Texteingabe an GNU Go. Dieses antwortet nach kurzer Bedenkzeit mit den Koordinaten des gegnerischen Spielzugs. Anschließend ist der Spieler wieder am Zug.

Das Spiel ist zu Ende, wenn entweder der Spieler oder der Roboter keinen sinnvollen Zug mehr sehen und passen.

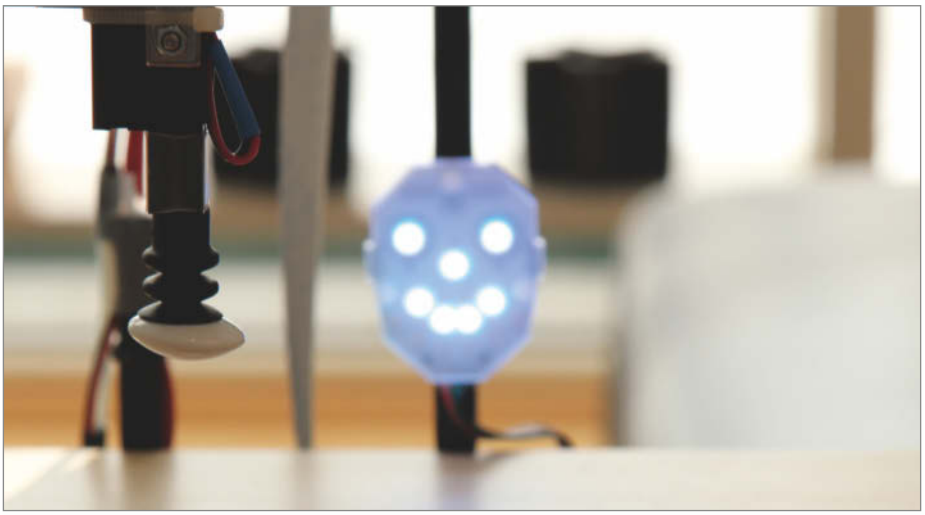
Ausblick

Der Programmcode, die 3D-Druck-Dateien und auch die bereits trainierten OpenCV-Haar-Modelle sind auf Github (siehe Kurzinfo) zu finden. Einen eigenen iGoBot zu bauen oder das Projekt selbst weiterzuentwickeln, sollte daher nichts im Wege stehen.

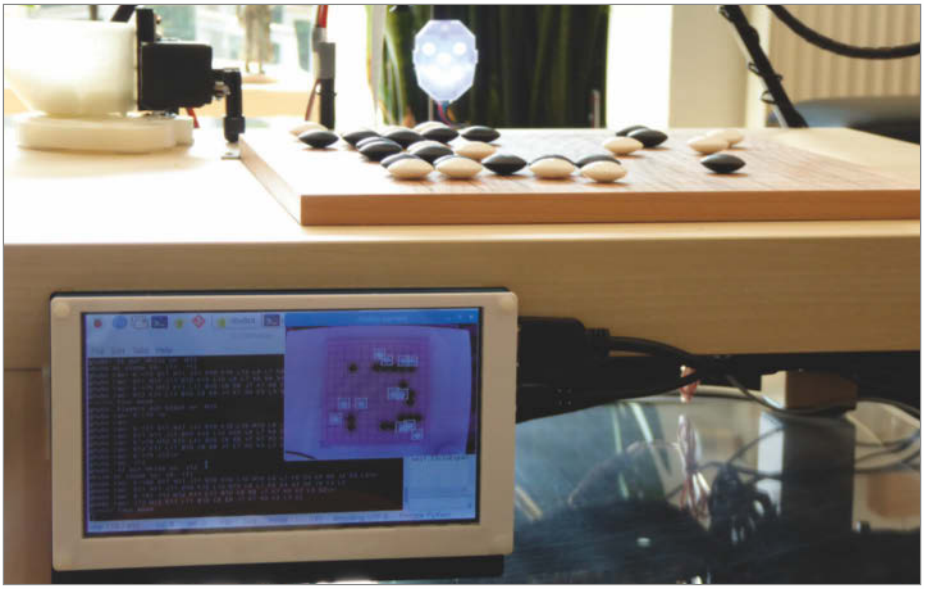
Die Software des Roboters ist in Python 3 geschrieben und pro Thema objektorientiert in einzelne Klassen unterteilt. Wer also nicht den iGoBot nachbauen möchte, kann zum Beispiel immer noch die Routinen zur Motorsteuerung oder Bilderkennung in eigenen Projekten nutzen.

Auf meinem Blog (siehe Kurzinfo) gibt es darüber hinaus noch weitere Infos und Videos, die den iGoBot zum Beispiel während eines Spieles zeigen.

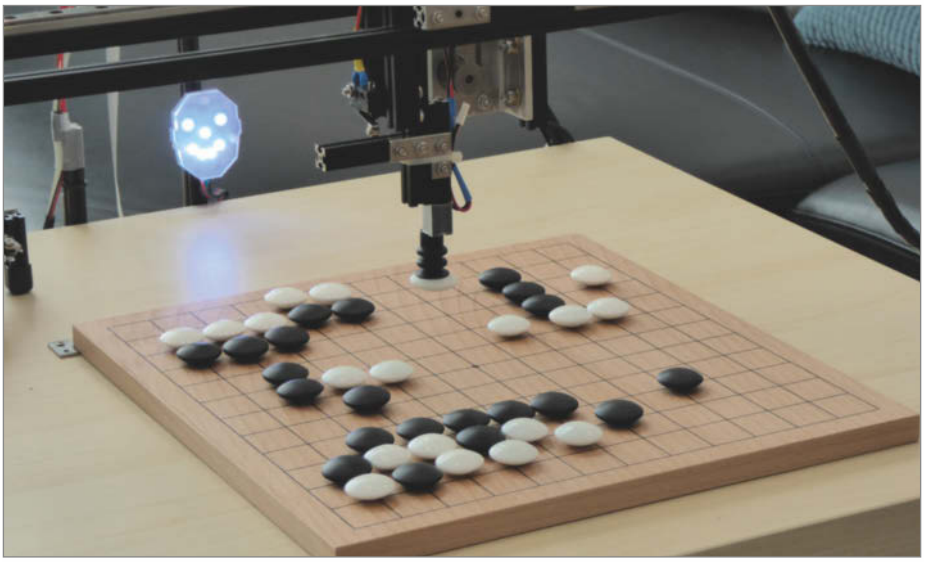
Es gibt auch ein paar Punkte, die ich gern noch für iGoBot realisieren möchte. Praktisch wäre zum Beispiel ein zweiter Steinspender, damit der Roboter auch aufgezeichnete Partien nachspielen kann. Am meisten Spaß macht mir iGoBot aber, wenn ich ihm auf der Maker Faire beim Spiel gegen menschliche Gegner zuschauen kann. *—hch*



iGoBot kann sprechen und Gesichter machen.



Mit einem Bildschirm sehe ich, was die GNU-Go-KI gerade „denkt“.

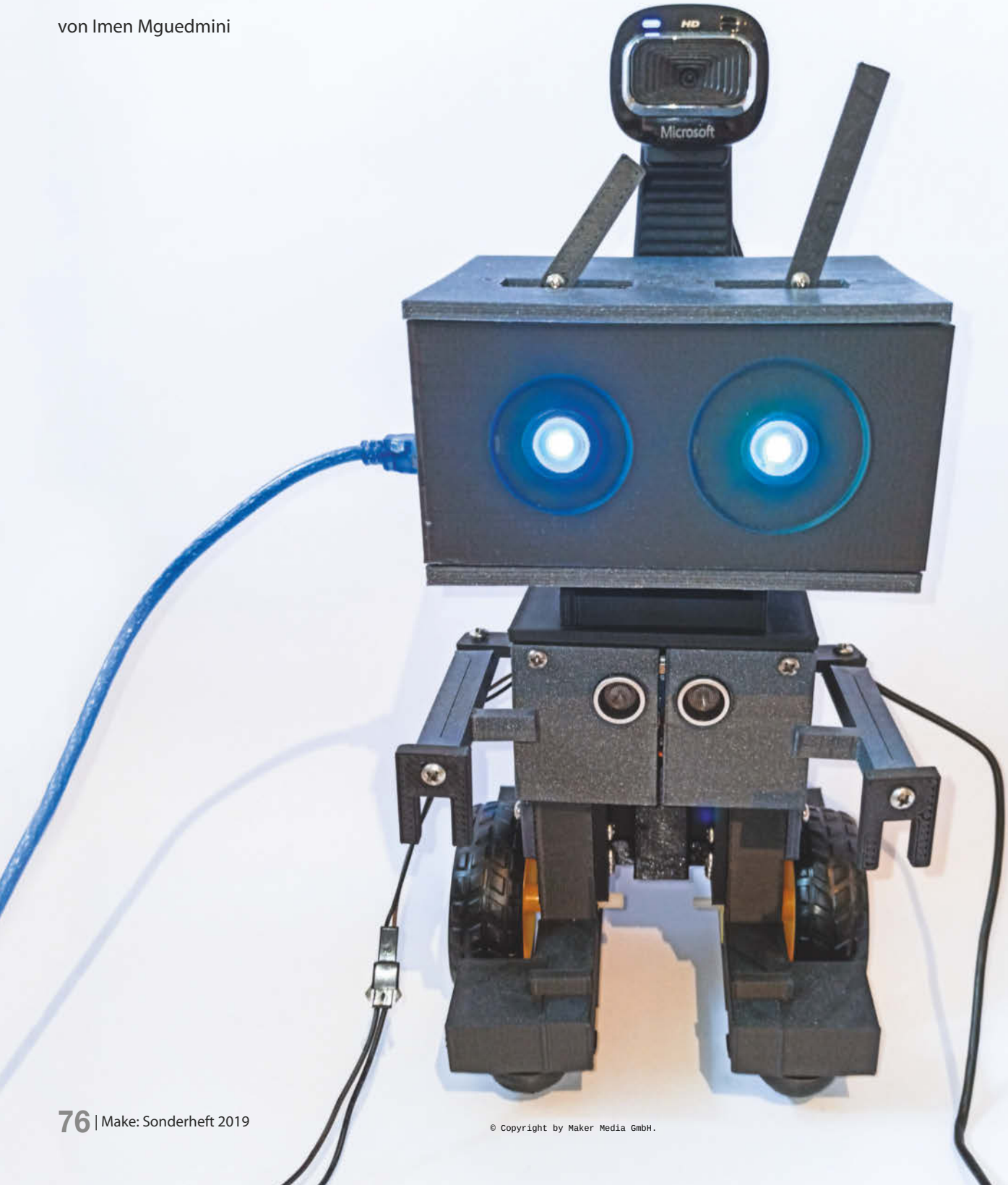


iGoBot während des Spieles

Bobby – Sitzhaltungsbot mit Persönlichkeit

Es gibt ein großes Angebot an Geräten, die uns dabei unterstützen sollen, unser Verhalten zu verbessern. Leider machen wir häufig die Erfahrung, dass wir diese tollen Gadgets auch schnell wieder ignorieren. Mit diesem Roboter passiert das garantiert nicht mehr!

von Imen Mguedmini



Ändern Menschen ihr Verhalten leichter, wenn sie sich beobachtet fühlen? Die Ergebnisse aktueller Forschung deuten darauf hin. So beteiligen sich Menschen beispielsweise ehrlicher an einer Kaffeekasse, wenn sie in der Küche ein Aufkleber von einem menschlichen Auge beobachtet. Man kann annehmen, dass sich der Effekt verstärkt, je realistischer das Gefühl des „Beobachtet-Werdens" ist. Humanoide Roboter wie zum Beispiel *Nao* werden dank ihrer Gestalt und Interaktion schnell als Individuen mit Persönlichkeit ins Herz geschlossen. Warum also nicht einen Roboter als Unterstützung einsetzen, wenn man sein Verhalten ändern möchte? Er könnte „ein Auge" auf einen werfen und notfalls sogar ermahnen – beispielsweise mit einem strengen: „Du wolltest doch nicht mehr um 2 Uhr nachts den Kühlschrank plündern?". Derselbe Effekt ist auch in der Arbeitswelt anwendbar: „Hey, warum trägst du keinen Schutzhelm?". So entstand bei mir die Idee für *Bobby*, den Roboter-Wachtmeister. Im Rahmen meines Studiums Mensch-Computer-Systeme an der JMU in Würzburg habe ich Bobby als Plattform für Experimente umgesetzt.

Als passendes „Problem" für Bobby habe ich das Thema ungesunde Sitzhaltung gewählt. Sitzen wird zunehmend als „das neue Rauchen" bezeichnet. Entsprechend gibt es viele Produkte, die den Benutzer bei einer guten Sitzhaltung unterstützen sollen. Sie reichen von der einfachen Erinnerungsfunktion für Sitz-Pausen bis hin zu Bauch- und Rückengurten. Diese Produkte führen oft nicht zum Erfolg, weil die Benutzer sie irgendwann als lästig und nutzlos empfinden. Dann werden die Produkte ausgeschaltet oder ignoriert. Bobby ist da anders: Er steht auf dem Schreibtisch und beobachtet den Nutzer und nimmt mittels einer einfachen Webcam Positionsveränderungen des Gesichts wahr. Gegebenenfalls fordert er mit Nachdruck eine aufrechte Körperhaltung ein. Durch sein Verhalten soll der Effekt des „Beobachtet-Werdens" möglichst verstärkt werden. So fängt Bobby bei „Langeweile" beispielsweise an, mit seinen Antennen „Yoga" zu machen. Ich habe verschiedene Verhaltensmodelle umgesetzt, um Bobby mit einer neutraleren oder frecheren Persönlichkeit auszustatten. Hier stelle ich die neutrale Variante vor.

3D-Druck und Aufbau

Ich entschied mich für ein Open-Source-Design namens *WireBeings* als Grundlage für meinen Roboter: ein simples, aber robustes und druckbares Roboter-Design für den 3D-Druck mit einer Höhe von rund 35cm (1). Bei dieser Größe passt Bobby gut auf einen Schreibtisch.

Kurzinfo

- **Mit Roboter eigene Sitzhaltung verbessern**
- **Dem Roboter Gesichtserkennung beibringen**
- **Robotergehäuse 3D-drucken**

Checkliste

Zeitaufwand:
für den Aufbau vier bis sechs Stunden und für den 3D-Druckvorgang circa 40 Stunden

Kosten:
60 bis 100 Euro

3D-Druck:
600g Filament

Programmieren:
Python, Arduino IDE

Elektronik:
Grundkenntnisse

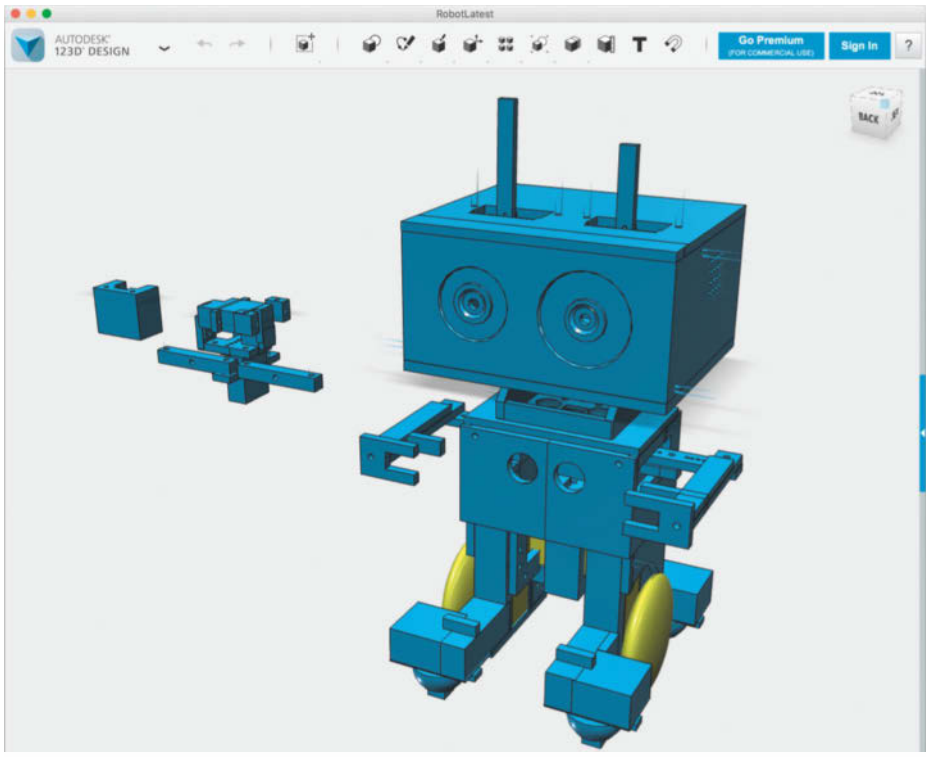
Werkzeug

- **Zugang zu einem 3D-Drucker** mit Bauraum von mindestens 20 × 20 × 20cm
- **Kreuzschraubendreher**
- **Feinschraubendreher**
- **Akkuschrauber**
- **Seitenschneider**
- **Feinlötkolben** mit Lötzinn

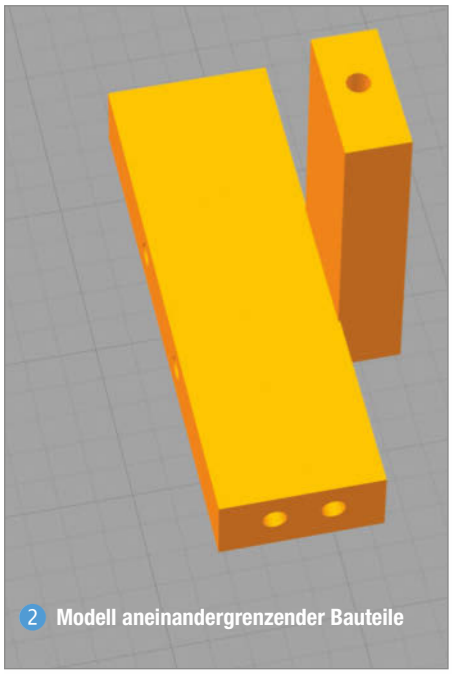
Material

- **2 DC-Motoren** 3V–6V mit Plastikreifen
- **Arduino Uno**
- **L298N Motortreiber-Board**
- **20cm-Dupont-Kabel** in verschiedenen Male/Female-Kombinationen
- **24AWG Kabel** 10cm lang
- **9V-Batteriegehäuse**
- **HC-SR04 Ultraschall-Abstandssensor** optional
- **2 Mini-Breadboards** 170 Punkte
- **2 LEDs 5mm mehrfarbig**
- **3 220-Widerstände** 250mW
- **1 RS-232-Bluetooth-Modul HC 06 RF** optional
- **Microsoft LifeCam HD-3000** oder ähnliches Produkt
- **40 Schrauben 6mm × 32mm**
- **9V-Batterie,** alternativ Netzteil mit 12V und 1,5A
- **600g PLA-Filament**
- **2 MG90S-Micro-Servos**

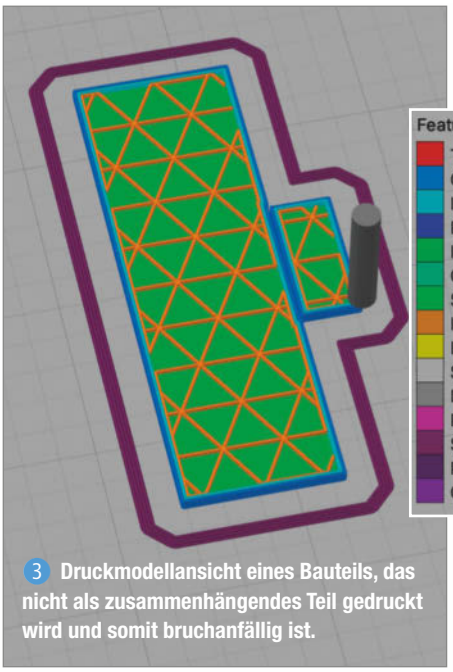
Alles zum Artikel im Web unter
make-magazin.de/xmqv



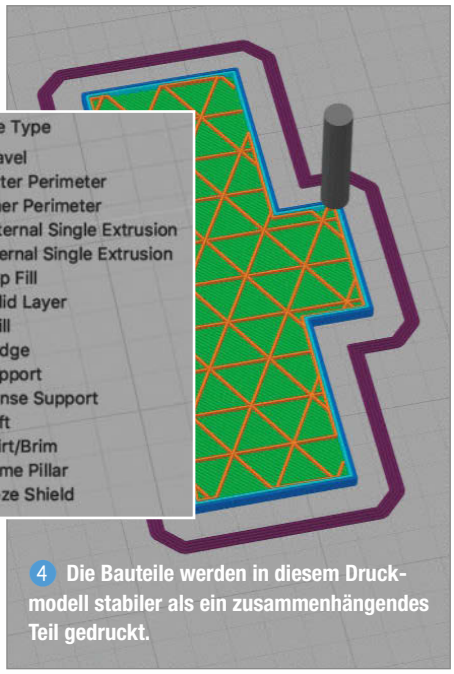
(1) WireBeings 3D-Modell von Matthew Hallberg



2 Modell aneinandergrenzender Bauteile



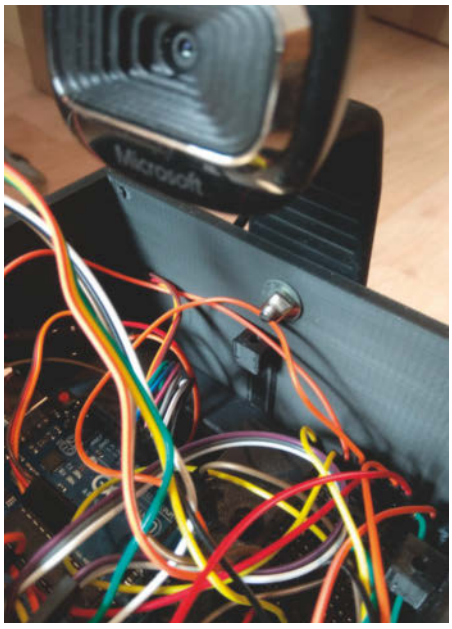
3 Druckmodellansicht eines Bauteils, das nicht als zusammenhängendes Teil gedruckt wird und somit bruchanfällig ist.



4 Die Bauteile werden in diesem Druckmodell stabiler als ein zusammenhängendes Teil gedruckt.

Das WireBeings-Modell habe ich in PLA auf einem Creality-CR-10-3D-Drucker gedruckt. Das 3D-Druck-Modell findet man auf der Plattform thingiverse.com. Die vorgeschlagenen 10 Prozent Infill haben sich als ausreichend erwiesen. Insgesamt dauerte der Druck circa 40 Stunden und erforderte 600g Filament. Bei Thingiverse stehen verschiedene Varianten des Roboters als STL-Dateien zur Verfügung. Dort gibt es zum Beispiel eine Kopf-Variante, die sich auf- und abbewegen lässt. Leider sind diese teilweise fehlerhaft. Schwierig ist der Druck von Bauteilen, die aus mehreren verbundenen Elementen bestehen. Bei diesen sind die einzelnen Elemente des Bauteils nicht als ein zusammenhängendes „solid" Modell definiert 2. Dadurch entstehen beim Druck Sollbruchstellen 3. Ich habe die einzelnen Baugruppen als solide Objekte erneut exportiert und stelle diese bei Thingiverse und Github zur Verfügung 4. Der Downloadlink befindet sich in der Kurzinfo. Die Bauteile konnte ich nach dieser Modifikation deutlich stabiler drucken.

Für den Zusammenbau steht bei Instructables eine gut bebilderte und nachvollziehbare Anleitung bereit. Der Link zur Anleitung befindet sich ebenfalls in der Kurzinfo. Ich habe einige Veränderungen an der Hardware-Ausstattung vorgenommen. Der zum Zeitpunkt meines Projekts aktuellste Raspberry Pi war nicht schnell genug für die Gesichtserkennung. Daher entschied ich mich relativ früh dafür, die Bild-Auswertung und die Ablaufsteuerung auf einem Laptop durchzuführen. Ich habe aus diesem Grund auf das Bluetooth-Modul verzichtet. Mit dem Nvidia Jetson Nano gibt es zwar sogar einen spezialisierten Single-Board-Computer für entsprechende Aufgaben (siehe auch Seite 40), aber das hätte die Kosten mehr als verdoppelt. Zusätzlich brachte ich eine Webcam am Kopf an. Dazu habe ich einfach mit einem Akkuschrauber ein Loch gebohrt und die Halterung mit dem Gehäuse verschraubt 5, 6.



5 Befestigung der Webcam an der Innenseite des Kopfs

6 Befestigung der Webcam an der Außenseite des Kopfs

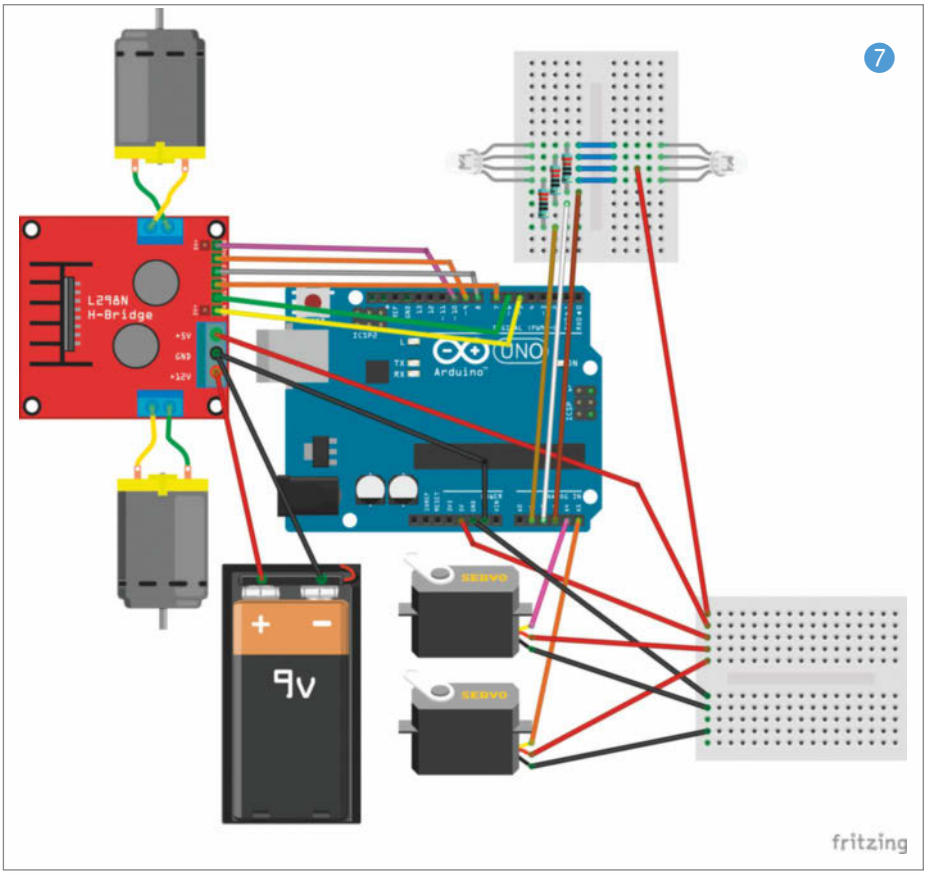
Der Schaltplan auf Instructables beinhaltete zwei kleinere Fehler, die ich nach einiger Suche beheben konnte. Ich habe den Schaltplan 7 um ein zusätzliches Breadboard zwischen den Augen ergänzt 8. Zusätzlich habe ich das Konzept um zwei Servos erweitert. Unser Motor-Treiber L298N erwartet eine Spannung von 12 Volt. Der im Entwurf vorgesehene Betrieb mit 9 Volt funktioniert zwar mit einer frischen Batterie kurzfristig, ich habe jedoch auf ein externes 12-Volt-Netzteil umgestellt 9.

Arduino

Im Kopf ist ein Arduino verbaut. Er ist für die Ansteuerung des Motor-Treibers und der Servos für die Ohrenbewegungen verantwortlich. Dazu steuert er die mehrfarbigen LEDs in den Augen von Bobby. Er bezieht in meinem Entwurf seine Stromversorgung via USB vom Laptop. Für den Arduino habe ich

einen Sketch programmiert, den ich auf Github zur Verfügung stelle. Dieser kann regulär mit der Arduino IDE aufgespielt werden. Man sendet dann über die serielle Schnittstelle einzelne Zeichen an den Arduino, der daraufhin Aktionen durchführt.

Das Gesamtsystem arbeitet dabei wie folgt: Die Webcam filmt den Nutzer. Auf dem Laptop läuft ein Python-Programm, das in den eingehenden Bildern Gesichter erkennt. Anhand der Position eines Gesichts und einer Zustandsmaschine wird im Programm entschieden, wie sich der Roboter verhalten soll. Die Reaktionen reichen von Signalfarben der Augen über Bewegung der Ohren bis zur Sprachausgabe. Das Programm stößt die entsprechenden Aktionen an, indem es zum Beispiel Audio-Dateien abspielt oder die passenden Zeichen an den Arduino sendet. Das geschieht über die auf dem USB-Kabel vom Arduino bereitgestellte virtuelle serielle Schnittstelle. Der Arduino wiederum steuert über seine GPIO-Ports die Servos und die LEDs an. Er nutzt gegebenenfalls den Motor-Treiber, um den Roboter zu bewegen.



7

OpenCV

Jetzt will ich Bobby beibringen, Gesichter von anderen Objekten zu unterscheiden. Dabei geht es mir nicht darum, einzelne Gesichter zu identifizieren oder zuzuordnen. Wie wertet man also Bilder überhaupt aus und erkennt, ob und wo sie Gesichter abbilden?

Die Open-Source-Bibliothek OpenCV implementiert für diese Aufgabe frei verfügbare Algorithmen, sogenannte Klassifizierer. Eine gut verständliche Erklärung und einen Vergleich von Klassifizierern für die Gesichtserkennung findet man auf dem Blog von SuperDataScience (Link in der Kurzinfo). Python wird häufig in der Roboterprogrammierung und im maschinellen Lernen eingesetzt, weshalb im Bereich Computervision viele Bibliotheken zur Verfügung stehen. Dazu gehört auch eine Integration von OpenCV. Die OpenCV-Python-Bibliothek lässt sich mittels `pip`-Konsolenbefehl installieren 10.

Sie beinhaltet statisch gebundene OpenCV-Binaries für die jeweilige Plattform. OpenCV muss also nicht gesondert installiert werden. Innerhalb meines Python-Programms muss ich die angeschlossene Webcam initialisieren 11.

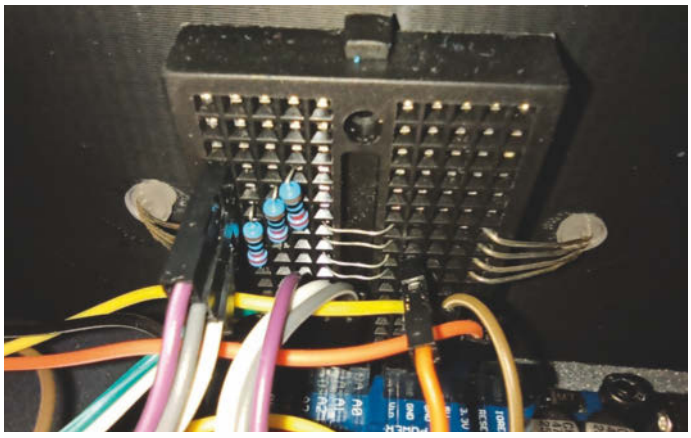
Der Parameter ist dabei die sequentielle ID der Kamera. Gegebenenfalls muss man hier die ID anpassen: Anders als erwartet hat auf meinem MacBook die externe Webcam die ID „0“. Den Klassifizierer instanziiere ich dann 12.

Listing 1 10

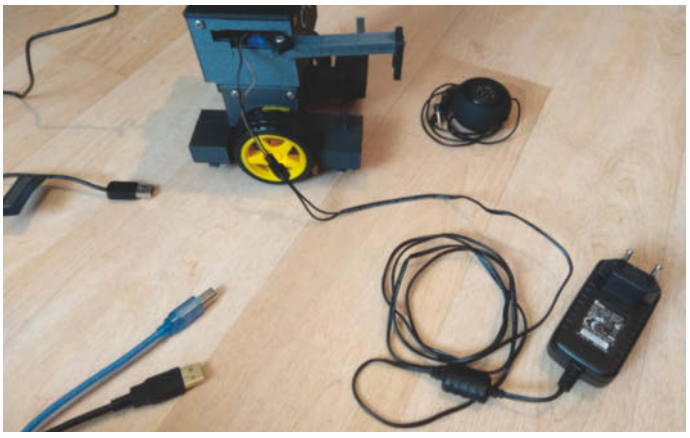
```
1 pip3 install opencv-python
```

Listing 2 11

```
1 cam = cv2.VideoCapture(0)
```



8 **Mit einem zweiten Breadboard lassen sich die LEDs leichter anbringen.**



9 **Die Stromversorgung des Motor-Treibers erfolgt über ein 12V-Netzteil anstelle einer 9V-Batterie.**

Listing 3 (12)

```
1 haar_face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + "haarcascade_frontalface_default.xml")
```

Listing 4 (13)

```
1 faces = haar_face_cascade.detectMultiScale(gray_img, scaleFactor=1.1, minNeighbors=5)
```

Danach wenden wir die Gesichtserkennung an und finden alle Gesichter auf einem Bild (13).

Diese Methode liefert für jedes erkannte Gesicht die Position – kleinste X-Koordinate und Y-Koordinate – sowie die Breite und Höhe des Rechtecks, das das Gesicht umrandet (14).

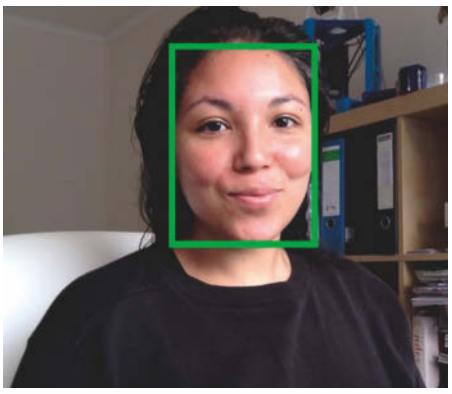
Wir nutzen die so ermittelten Daten nun, um die Gesichtsposition zu tracken. Für meinen Prototyp gehe ich dabei davon aus, dass sich immer nur eine Person im Sichtfeld der Webcam befindet. In (15) steht der Code für die Gesichtserkennung.

Zustandsmaschine

Eine Zustandsmaschine (16) bildet dann mögliches Verhalten des Nutzers sowie Zustandsübergänge und Aktionen des Roboters als zyklischen Graphen ab und nutzt die Daten des Trackings. In einem typischen Szenario möchte ein Nutzer während seiner Schreibtischarbeit eine gute Sitzhaltung beibehalten. Er nähert sich dem Schreibtisch und wird mit einem „Hallo“ begrüßt. Nachdem er einige Zeit seine Position nicht verändert – sich also vermutlich gesetzt hat –, beginnt der Roboter mit der Kalibrierung. Dazu fordert er den Nutzer auf, die angestrebte Haltung einzunehmen. Der Nutzer erhält eine Bestätigung der erfolgreichen Kalibrierung. Wenn der Nutzer länger von dieser Haltung abweicht, weist der Roboter freundlich – „Sei nicht dumm, sitz nicht krumm rum“ – darauf hin, die Sitzhaltung zu überprüfen. Behält der Nutzer seine Haltung trotzdem bei, so wird dies als bewusste Entscheidung wahrgenommen und es findet eine erneute Kalibrierung statt. Auch andere Fälle, wie zum Beispiel das Verlassen des Bildes, das Aufstehen und länger andauernde Bewegung werden in der Zustandsmaschine berücksichtigt.

Das Verhalten des Nutzers wird dabei je nach aktuellem Zustand unterschiedlich interpretiert. Es löst anhand vorgegebener Bedingungen Reaktionen des Roboters aus, zum Beispiel Audioausgaben. Die Audioausgabe erfolgt über das Open-Source-Programm SoX, das für alle gängigen Plattformen verfügbar ist. Es spielt über die Kommandozeile mp3-Dateien ab. In (17) steht der Code für die Audioausgabe.

Die Sprachausgaben habe ich mithilfe des say-Kommandos in macOS mit der Stimme „Markus“ erstellt. Alle mp3-Dateien stehen auf Github zur Verfügung.



(14) **Der Bildverarbeitungsalgorithmus ist darauf spezialisiert, Gesichter von anderen Objekten zu unterscheiden, und liefert Koordinaten sowie Breite und Höhe des Gesichts.**

Listing 5 (15)

```
 1 import cv2
 2 import datetime
 3 from time import sleep
 4
 5 cam = cv2.VideoCapture(0)
 6 sleep(1)
 7
 8 letztePositionGesicht = []
 9 anzahlLetzerPositionen = 5
10
11 def erkenneGesicht():
12     global letztePositionGesicht
13     ret_val, img = cam.read()
14     gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
15     haar_face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
       "haarcascade_frontalface_default.xml")
16     faces = haar_face_cascade.detectMultiScale(gray_img,
       scaleFactor=1.1, minNeighbors=5)
17
18     if len(faces) == 0:
19         if (len(letztePositionGesicht) > anzahlLetzerPositionen):
20             del letztePositionGesicht[0]
21         letztePositionGesicht.append((None, None))
22         return None,None
23
24     for (x, y, w, h) in faces:
25         if (len(letztePositionGesicht) > anzahlLetzerPositionen):
26             del letztePositionGesicht[0]
27         letztePositionGesicht.append((y,y+h))
28         return y,y+h
29
```

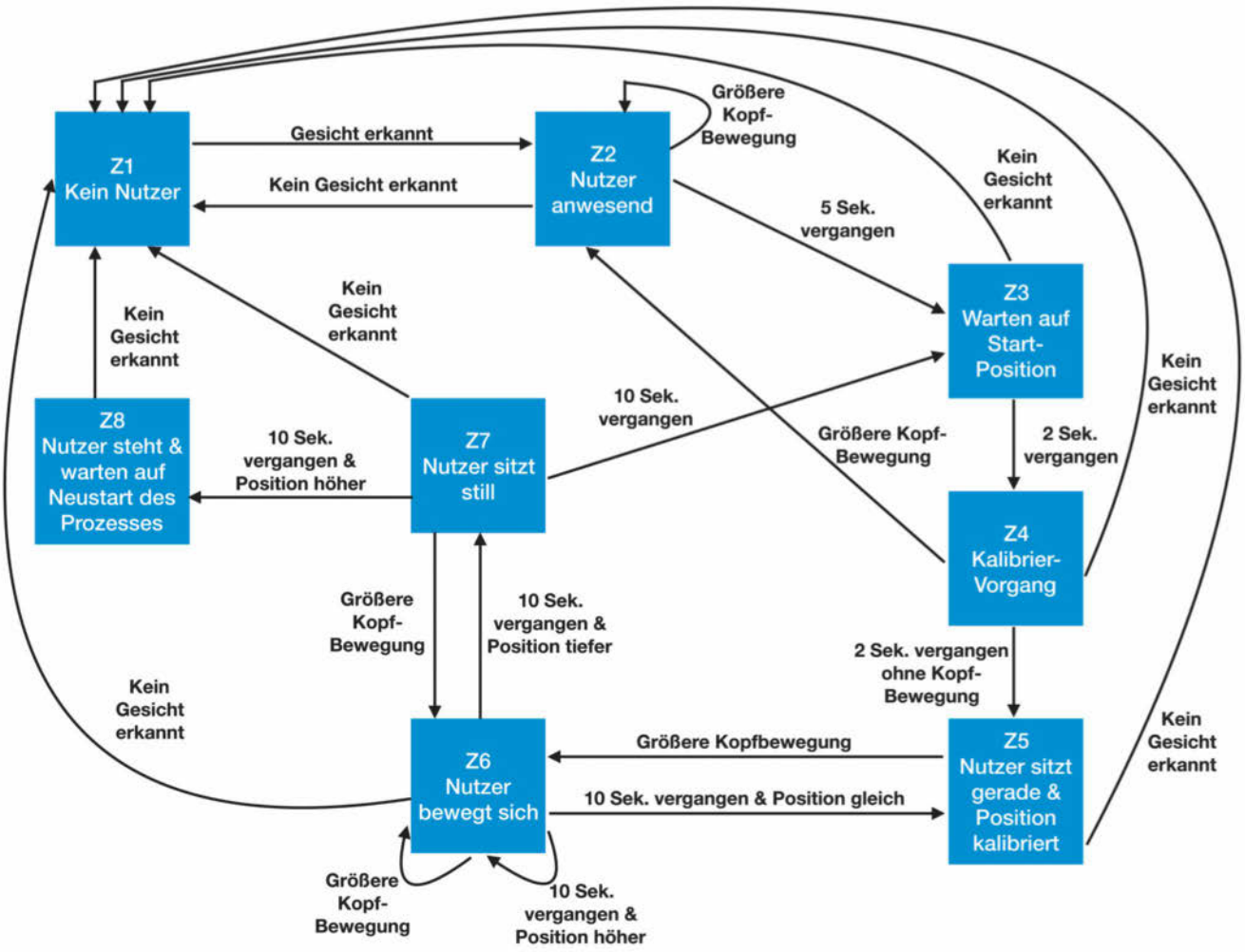
Listing 6 (17)

```
1 import subprocess
2
3 subprocess.run(["sox",  "../assets/sounds/"+ file +".mp3", "-d", "-q"])
4
```

Weiterentwicklung

Besonders die Sprachausgaben prägen die Mensch-Roboter-Interaktion. Bobby drückt sich umgangssprachlich aus, reimt und erklärt eigene Pläne. Im frechen Modell flucht er manchmal, wenn das Kalibrieren nicht klappt, oder plappert einfach vor sich hin. Die Nutzer finden das meist sehr niedlich. Für meine Experimente ist es nützlich, wenn Bobby Persönlichkeit zugesprochen wird. Aktuell arbeite ich daran, die Mobilität von Bobby nutzbar zu machen. Der ursprüngliche Entwurf sieht auch die Möglichkeit vor, weitere Servos (für die Auf- und Abbewegung des Kopfes und zur Bewegung der Arme) zu verbauen. Es sind jedochweitere Anwendungen nur mit den hier beschriebenen Features denkbar: So könnte Bobby den Nutzer in der Grippesaison mittels OpenCV unterstützen, sich nicht ins Gesicht zu fassen. —*rehu*

16 Zustandsmaschine: Mensch-Roboter-Interaktion basierend auf Haltungssituation



Roboterarme

Vom Investitionsgut zum Maker-Spielzeug: Manipulatoren und Roboterarme sind nicht mehr nur großen Produktionsbetrieben vorbehalten, sie erobern dank eines massiven Preisverfalls nun auch Bastelkeller, Klassenzimmer und Fablabs. Wir zeigen das Funktionsprinzip und eine Auswahl käuflicher Roboterarme.

von Carsten Meyer

Bild: KUKA AG

Kurzinfo

»Von kinematischen Ketten und Freiheitsgraden
»Antriebe und Gelenke
»Marktübersicht

Alles zum Artikel im Web unter *make-magazin.de/xehx*

Von SCARA-, Delta- und Portalrobotern unterscheiden sich Roboterarme (oder genauer: Gelenkarmroboter) durch ihre dem menschlichen Arm angenäherte Form – was der Name schon verrät. Die eher schlanke Bauweise erfordert im Vergleich zu den Erstgenannten eine komplett andere Mechanik: Während zum Beispiel Portalroboter ihre Arbeitslast auf mehrere Stützpunkte verteilen können, müssen sich Gelenkarmroboter komplett auf die Spielfreiheit der Gelenke und eine möglichst geringe Durchbiegung und Torsion der tragenden Teile verlassen.

Nicht umsonst bestehen die Gliedmaßen eines Industrieroboters aus geschmiedetem Stahl oder Stahlguss, was maßgeblich zum – gemessen an der Tragkraft – enormen Gesamtgewicht beiträgt. Der kleine Kuka-Industrieroboter Cybertech Nano KR10 trägt zum Beispiel 10kg, wiegt aber selbst 165kg – ohne Steuerungseinheit. Zur Traglast wird auch der sogenannte Effektor gezählt, das ist die „Hand" des Roboters (zum Beispiel Greifer, Fräswerkzeug oder Schweißzange). Bei 1,4m Reichweite erreicht das Gerät immerhin eine Wiederholgenauigkeit von 4/100mm, man könnte damit also ohne weiteres einen überdimensionalen 3D-Drucker bauen.

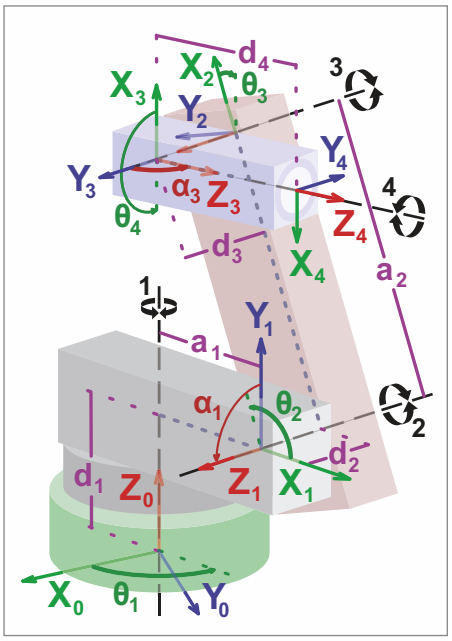
Kennzeichnend für einen Gelenkarmroboter sind seine Freiheitsgrade (DOF, Degrees of Freedom) – im Prinzip die Anzahl seiner Gelenke. Mit 5 oder 6 Gelenken/Rotationsgraden erreicht man schon die „Gelenkigkeit" des menschlichen Arms. Aktuelle Roboter besitzen bis zu 7 Freiheitsgrade, um mit dem Effektor das Werkstück (zum Beispiel eine Automobilkarosse) von überall und aus jeder Richtung erreichen zu können.

Der heutige Gelenkarmroboter geht auf eine Entwicklung von Victor Scheinman an der Stanford University zurück. Scheinman baute 1969 mit dem „Stanford Arm" einen komplett elektrisch angetriebenen Roboterarm mit sechs Freiheitsgraden und trug auch zu den Grundlagen der seriellen Kinematik bei, die bei Gelenkarmrobotern eine wichtige Rolle spielt. Seine Entwürfe, etwa der am Massachusetts Institute entwickelte MIT-Arm, wurden von der Industrie (vor allem der japanischen) dankbar angenommen.

Kinematische Ketten

Die Ansteuerung eines Gelenkarmroboters ist keineswegs trivial, und die mathematischen Grundlagen würden den Rahmen dieses Artikels – wenn nicht des ganzen Heftes – sprengen. Der Roboterarm besteht aus einer offenen „kinematischen Kette" aus motorisch betriebenen Dreh- und manchmal auch Schubgelenken – am einen Ende das Podest, am anderen Ende der Effektor. Um mit dem Effektor eine beliebige Raumposition zu erreichen, muss die Steuerungssoftware die Winkel der Gelenke nur passend einstellen.

Doch was ist „passend"? Schon der umgekehrte Fall – bekannte Winkel, zu errechnende Endposition des offenen Kettenendes – geht an die Grenzen der Mittelstufen-Mathematik. Mit Trigonometrie (Sinus/Cosinus-Satz, Winkel im Dreieck) lässt sich die Endposition in zwei Raumrichtungen zwar noch vollständig berechnen, bei drei Dimensionen stößt der Ansatz aber schnell an seine Grenzen – und dann würden die ermittelten Formeln auch nur für diese eine Maschine mit genau diesen Dimensionen gelten.



Freiheitsgrade eines Gelenkarmroboters: Aus den Gelenkwinkeln die Position am Ende der kinematischen Kette zu berechnen ist schon kompliziert genug. Der umgekehrte Weg gelingt numerisch nur mit Näherungen und Iterationen.

Bilder: Wikipedia

Vorläufer des weit verbreiteten PUMA-Roboters (Programmable Universal Machine for Assembly) war der MIT-Arm, den Victor Scheinman 1972 vorstellte.

Schrittmotor-Gelenkantrieb eines älteren SCARA-Roboters. Durch die niedrige Drehzahl des Schrittmotors genügt eine mäßig hohe Untersetzung, hier realisiert durch ein präzises Stirnradgetriebe.

Zum Glück fanden Jacques Denavit und Richard Hartenberg 1955 ein Verfahren, um beliebige kinematische Ketten in eine Matrizenform zu überführen, sodass sich schließlich standardisierte Rechenschritte zur Ermittlung der Endposition ergeben – unabhängig von den Maschinendimensionen. Dass ihre eigentlich für Koppelgetriebe (Hebelgestänge) erdachte Lösung auch der Robotik zu Gute kommt, konnten die beiden Ingenieure nicht ahnen: Die Denavit-Hartenberg-Transformation (DH-Transformation) ist heute das Standardverfahren zur Bestimmung der Effektor-Position bei Industrieroboter.

Inverse Kinematik

Damit ist dem Roboter aber noch nicht geholfen: Schließlich gibt man die Effektor-Endposition vor und nicht die Winkel der Gelenke – die möge die Gerätesteuerung doch bitte selbst berechnen. Hier kommt nun die „inverse Kinematik" ins Spiel, das Gegenstück zur gerade beschriebenen „direkten Kinematik".

Jetzt wird es richtig heftig: Eigentlich gibt es dafür kein direktes Lösungsverfahren; die Rücktransformation der DH-Matrix liefert in der Regel neben ungültigen (Position durch Beschränkungen der Gelenkwinkel nicht erreichbar) auch überraschend viele gültige Lösungen. Anschauliches Beispiel: Ein Glas auf dem Tisch erreichen Sie sowohl mit am Körper angelehnten wie auch mit angehobenem Ellenbogen. Gelenkarmroboter nutzen daher in der Regel iterative Verfahren, um den günstigsten (also schnellsten) Weg zur gewünschten Effektor-Position zu finden.

Kein Wunder, dass frühe Industrieroboter recht heftige Steuerungsrechner-Kaliber benötigten – eine komplette PDP-11 (ein Mainframe-Rechner der mittleren Datentechnik) an der Seite jedes einzelnen Roboterarms war keine Seltenheit. Heute hat man es leichter: Mit einem Raspberry Pi erhält man die Rechenleistung einer Cray-2-Anlage im Zigarettenschachtelformat.

Lernfähig

Bevor es ausreichend Rechenleistung oder gar 3D-visualisierte Roboterarm-Simulationen gab, war das Teach-In-Verfahren direkt vor Ort die Regel; es ist aber in der Industrie auch heute noch gängig. Mit einer Fernsteuerung bewegt man die einzelnen Achsen so lange manuell, bis die gewünschte Position erreicht ist, und speichert diese ab. Der Roboter muss nun nur noch die gespeicherten Gelenkwinkel „abfahren". Das Verfahren eignet sich vor allem für statische Situationen, etwa das Ansteuern verschiedener Schweißpunkte an immer gleichen Blechteilen. Auch die billigen Spielzeug-Roboterarme arbeiten mit dem leicht zu beherrschenden Teach-In-Verfahren.

Bei modernen Industrierobotern muss der Roboter-Operator dafür noch nicht einmal eine Fernsteuerung in die Hand nehmen: Er führt den mit Kraft- und Winkelsensoren ausgerüsteten Roboterarm „an der Hand" zu den abzuspeichernden Werkstückpositionen, ähnlich einem Lehrer, der dem Erstklässler die Hand mit dem Stift führt. Lackierroboter zum Beispiel bekommen als „Lehrer" erfahrene Lackierer, die genau wissen, wie schnell und aus welcher Richtung und Entfernung man die Spritzpistole führen muss, um einen optimalen Lackauftrag zu erreichen.

Sind die benötigten Winkeleinstellungen erst einmal gefunden, ist der Rest ein Klacks: Die Steuerung hat nun die Servo- oder Schrittmotoren der Gelenke nur noch nach Vorgabe einzustellen. Hierbei muss sie allerdings berücksichtigen, dass Beschleunigungs-Grenzwerte nicht überschritten werden – einerseits, um die dynamischen Kräfte am Effektor oder einer Nutzlast nicht zu überschreiten, anderseits, um die Motoren und Getriebe nicht zu überlasten. Je nach Trägheit der zu bewegenden Massen werden die Motoren also mehr oder weniger sanft beschleunigt oder abgebremst.

Servomotor mit Regeleinheit für kleinere Industrieroboter. Für den Bahn-Controller, der die Bewegungen koordiniert, verhält sich die Servo-Steuerung ähnlich wie ein Schrittmotor, angesteuert mit Schritt- und Richtungsimpulsen.

Motoren

Der Standardantrieb von Industrieroboter-Gelenken ist ein Servomotor, also ein DC-, Brushless- oder Drehstrommotor mit angeflanschtem Winkelgeber (Resolver) und einem stark untersetzenden Getriebe. Der Resolver liefert neben der Drehrichtung eine bestimmte Anzahl von Impulsen pro Umdrehung an die Steuerung zurück. Aus Getriebeübersetzung und Pulsanzahl kann die Steuerung jederzeit den aktuellen Gelenkwinkel

berechnen; weicht er vom Sollwert ab, etwa durch Belastung, kann sie den Motor gegensteuern, bis der Winkel wieder stimmt. Es handelt sich hierbei um einen geschlossenen Regelkreis, deshalb auch die Bezeichnung Servomotor.

Die aus 3D-Druckern und CNC-Maschinen bekannten Schrittmotoren findet man nur bei kleineren Geräten, im Industrie-Bereich sind sie eher selten. Schrittmotoren kommen ohne Resolver und Regelkreis aus, stattdessen verlässt man sich darauf, dass der Motor bedingungslos den Schritt-Impulsen und der Richtungs-Vorgabe folgt; die Steuerung zählt die Schritte ausgehend von einem beim Maschinenstart angefahrenen Nullpunkt intern mit.

Die Lösung ist kostengünstig, weist aber einige Nachteile auf: Hat der Motor aufgrund einer plötzlichen Belastung einen Schritt „verloren“, stimmen sämtliche folgenden Positionen nicht mehr, was zu Unfällen oder zu Schäden am Werkstück führen kann. Zudem sind Schrittmotoren eher langsam und gemessen an ihrer vergleichsweise bescheidenen Leistung recht groß und schwer. Sie erreichen nur selten mehr als 1000Upm, zudem fällt ihr Drehmoment bei steigender Drehzahl stark ab.

Servomotoren sind viel schneller, sie erreichen mühelos 3000Upm und mehr, wobei die maximale Drehzahl eher von den Fähigkeiten des Resolvers begrenzt wird. Sie sind kurzzeitig überlastbar, ohne dass man sich um die erreichte Position sorgen muss. Und schließlich ist ihre Dynamik besser, weil sie wegen des leichteren Ankers weniger Masse in Drehung versetzen müssen als ein Schrittmotor.

Nachteilig ist der erhöhte Steuerungsaufwand: Nötig ist neben Frequenzumrichter (Drehstrommotoren) oder DC-Spannungssteller mit Umpolung natürlich auch die Auswerteschaltung für den Drehwinkel-Resolver und ein Controller, der den angeforderten mit dem aktuellen Drehwinkel vergleicht. In modernen Steuerungen erledigt ein Signalprozessor die Regelaufgaben samt Drehfeld- oder PWM-Erzeugung. Er optimiert auch die Parameter des Servo-Regelkreises für ein möglichst rasches Arbeiten, ohne dass Resonanzen oder Überschwinger auftreten.

Zum Glück hat der Anwender damit nicht allzu viel zu tun, kommerzielle Roboterarme sind bereits fertig vom Hersteller konfiguriert. Wer den Selbstbau eines Roboterarms samt Steuerung plant, sollte sich allerdings über die zu erwartenden Probleme Gedanken machen.

Getriebe

Eine besondere Herausforderung bei Roboterarm-Gelenken stellt das Getriebe dar: Die Untersetzung ist mit 40:1 bis 200:1 recht hoch, weil die Anwendung ein hohes Ausgangsdrehmoment erfordert. Das Getriebe muss kompakt, leicht und vor allem spielfrei sein, außerdem sollte es einen hohen Wirkungsgrad aufweisen. Kompakte Abmessungen lassen sich eigentlich nur mit einem einstufigen Getriebe erreichen.

Normale Stirnradgetriebe müssten bei der geforderten Untersetzung mindestens dreistufig sein, wobei sich das zu vermeidende Spiel mit jeder Zahnradpaarung um den Faktor des jeweiligen Übersetzungsverhältnisses vergrößert. Eine Spielfreiheit ist mit einem erheblichen konstruktiven Aufwand verbunden, etwa durch mit Federkraft gegeneinander verspannte Zahnräder. In der Praxis findet man Stirnradgetriebe vorrangig bei Kleinst- und Spielzeugroboterarmen.

Auch die im Lowest-Cost-Bereich gern verwendeten Modellbau-Servos arbeiten überwiegend mit Stirnradgetrieben. Hier nimmt man das Getriebespiel in Kauf; es wird dadurch kompensiert, dass der Resolver (hier ein einfaches, den Drehwinkel „messendes“ Potentiometer) am Getriebe-Ausgang statt am Motor platziert ist. Der Motor im Servo muss das Spiel des Getriebes somit aktiv über die Regelschleife ausgleichen. Das klappt, wie man bei besseren Modellbau-Servos sieht, ganz gut, solange keine allzu schnellen dynamischen Lasten auftreten. In so einem Fall benötigt der Motor einige Zeit, bis er das Getriebespiel überwunden hat und gegensteuern kann.

Als Roboter-Getriebe besser geeignet sind Schnecken- und Zykloidgetriebe sowie das raffinierte Spannungswellengetriebe (Harmonic Drive). Schneckengetriebe erlauben große Untersetzungen, allerdings sind An- und Abtriebswelle um 90° versetzt und liegen nicht auf einer Höhe. Schneckengetriebe werden gern bei den im Podest untergebrachten Drehtellern zum Schwenken des Arms verwendet, bei den Armgelenken ist der Versatz von An- und Abtriebswelle dagegen oft ein konstruktives Hindernis.

Bei großen professionellen Robotern kommen als Gelenkantrieb fast durchweg Zykloidgetriebe zum Einsatz (siehe Bild), die laut Marktführer Nabtesco Drehmomente bis 28.000 Nm und Wirkungsgrade bis 95 Prozent erreichen können. Eine Selbsthemmung (wie sie beim Schneckengetriebe auftritt) ist nicht vorhanden – bei nicht angetriebenen Motoren würde man den Roboterarm durch eine äußere Kraft bewegen können.

Bei mittleren und kleinen Robotern verwendet man gern das Spannungswellengetriebe: Hier presst ein elliptischer Rotor eine zahnbewehrte, elastische „Tasse“ an einen feststehenden Außenring mit leicht unterschiedlicher Zähnezahl. Die Untersetzung ergibt sich aus der Differenz der Zähneanzahl im Verhältnis zum Innenläufer: Wenn der Au-

Zykloidgetriebe (hier ein Modell) erreichen ihre hohe Untersetzung (bis 200:1) über den von einer Exzenterscheibe bewegten Rotor-Ring.

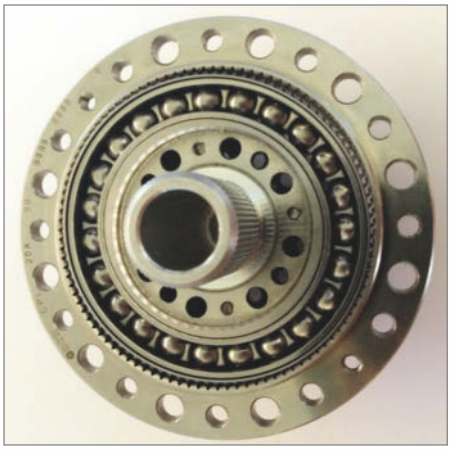


Bild: Harmonic Drive AG, Wikipedia

Wellgetriebe: Angetrieben wird der elliptische Rotor, der den elastischen Innenläufer (Abtriebsseite) an die Verzahnung des feststehenden Außenrings presst. Durch die hohe Zahl ineinandergreifender Zähne ist das Wellgetriebe nahezu spielfrei.

Bild: igus

Kleines Wellgetriebe vom Kunststoff-Spezialisten igus. Der Hersteller bietet auch passende Arm-Glieder und Motoren an.

Bild: igus

Mit dem Apiro-Modularsystem von igus kann man sich Roboter nach Maß selbst bauen.

Typischer Spielzeug-Roboterarm: Wegen fehlender Positionsgeber ist die Wiederholgenauigkeit stark belastungsabhängig.

Angebot von Banggood.com: Kräftige Digitalservos gibt es schon bei Roboterarmen in der Preisklasse um 170 Euro, wenn man eine China-Direktbestellung nicht scheut.

ßenring 202 Zähne aufweist, der Innenläufer dagegen 200, beträgt die Untersetzung 200:2 = 100:1. Der Wirkungsgrad reicht mit bis zu 85 Prozent nicht an das Zykloidgetriebe heran, außerdem ist das Wellgetriebe immer selbsthemmend. Bei besonders leichten Ausführungen fertigt man die „Tasse“ und den Außenläufer-Ring aus einem reibungsarmen Kunststoff; so macht es etwa igus bei seinen schmierstofffreien Gelenken.

In den mit „for education purposes“ etwas euphemistisch beworbenen Billig-Roboterarmen kommen Modellbau-Servos als Gelenk-Antrieb zum Einsatz, preiswerte Standard-Teile. Mit der Wiederholgenauigkeit ist es bei ihnen nicht so weit her, weil hier wie oben erwähnt als Resolver einfache Potentiometer verwendet werden. Unter Last geben auch bessere Modellbau-Servos um einige Winkelgrad nach, was sich mit vier oder fünf Gelenken dann auf einige Millimeter am Effektor summiert.

Preisklassen

Genug der Theorie und hinein in die Praxis der Maker-gerechten Roboterarme. In den letzten Monaten haben sich hier drei Preisklassen etabliert, die wir hier genauer betrachten wollen: Geräte bis 50 Euro, bis 250 Euro und bis 1500 Euro. Danach klafft eine recht große Lücke, die erst bei preiswerten professionellen Robotern ab 4000 Euro endet.

Getrost von Spielzeug kann man in der Preisklasse bis 50 Euro sprechen: Zwar gibt es hier manchmal einen USB-Anschluss zur Fernsteuerung über den Rechner, der Antrieb besteht aber nur aus Spielzeugmotoren (keinen Servos!) mit wackeligem Stirnradgetriebe, denen jegliche Rückkopplung zur

aktuellen Position fehlt. Eine programmierte Abfolge von Bewegungen – also das, was einen Roboter eigentlich auszeichnet – ist nur über die Laufzeit und die geschätzte Drehzahl der Motoren möglich und deshalb äußerst ungenau.

Geräte dieser Art taugen noch nicht einmal für Anschauungs- oder Lehrzwecke, bestenfalls machen sie einem Drittklässler noch Freude. Immerhin sind in einigen Billigst-Armen bereits Modellbau-Servos eingebaut, die eine Wiederholgenauigkeit im Zentimeter-Bereich versprechen. Damit kann man zumindest schon einmal das Zusammenspiel der einzelnen Achsen üben.

Ein breitgefächertes Angebot findet sich in der Preisklasse bis 250 Euro. Hier kommen zumindest passable Modellbau-Servos in kräftiger Ausführung zum Einsatz, bei den teureren Geräten auch schon die genauer arbeitenden Dynamixel-Servos des koreanischen Herstellers Robotis oder ähnliche digitale Servos.

Mit herkömmlichen RC-Servos sind beispielsweise die Bausatz-Arme RA1-Pro und RA2-Hobby des niederländischen Herstellers Arexx ausgerüstet. Die mitgelieferte Steuerelektronik findet in der Bodenplatte Platz. Die Software-Installation ist nicht ganz so einfach wie der mechanische Zusammenbau: Zum Kompilieren der Steuerungs-Firmware gilt es zunächst, WinAVR (Windows) oder AVR-GCC (Linux) zu installieren – das wird den Arduino-verwöhnten Anwender etwas verwundern. Die eigentliche Steuerung erfolgt über ein Java-Programm, eventuell sind hierfür weitere Installationen (Java JRE 1.6, Treiber) erforderlich.

Dynamixel-Servos lösen typischerweise 4096 Schritte pro 360°-Drehung auf, was einer Schrittweite von 0,088° entspricht. Die kräftigeren Ausführungen kosten aber auch schon mal 300 Euro und mehr – pro Stück, wohlgemerkt.

Bild: Robotis

Gute Wiederholgenauigkeiten lassen sich bei Mini-Roboterarmen nur mit digitalen Servos erreichen. Es gibt hierbei zwei Klassen: Die einen arbeiten intern mit einem digitalen Resolver und mit digitaler Positionsdatenverarbeitung, werden aber „klassisch“ mit variablen Impulsbreiten angesteuert; sie sind somit mit „analogen“ RC-Servos kompatibel. Der aktuelle Trend geht allerdings zu voll digitalen Servos, die nicht über Impulsbreiten angesteuert werden, sondern mit Befehlen

Fertigroboter Robolink von igus in der 5000-Euro-Preisklasse.

über ihre serielle RS-485-Schnittstelle. So arbeiten beispielsweise die Dynamixel-Servos von Robotis; es gibt in dieser Familie hochwertige Ausführungen mit durchgehender Achse, die speziell für Roboter entwickelt wurden.

Während die Pulsbreitensteuerung nur Winkelauflösungen im 1-Grad-Bereich zulässt, arbeiten Digitalservos viel genauer: Üblich sind 4096 Schritte für eine volle Umdrehung, womit sich eine theoretische Auflösung von nur 0,088° ergibt. Außerdem lassen sich Beschleunigungs- und Drehmomentprofile erstellen und abspeichern. Für Roboterarme ist dies ideal, da man so materialschonende Belastungs- und Beschleunigungsgrenzen vorgeben kann.

Bestellen Sie Geräte mit digitalen Servos immer mit dem passenden Controller (sofern im Paket nicht enthalten). Von außen sieht man den Servos nicht an, ob sie im Serial-TTL- oder RS-485-Halbduplex-Betrieb mit dem Controller kommunizieren. Serial-TTL-Servos haben nur eine Signalleitung und demzufolge drei Anschlussleitungen, was leicht zu Verwechslungen mit ihren „analogen" Brüdern führen kann. Die Ausführungen mit vollwertiger RS-485-Schnittstelle haben symmetrische Datenleitungen und demzufolge vier Anschlussleitungen.

Digitalservos sind in gewisser Weise netzwerkfähig – alle hängen an der gleichen Datenleitung, und sie sind über Befehls-Präambel einzeln adressierbar. Zum Anschluss an den PC (zum Beispiel über USB) benötigt man einen speziellen Controller, der selbsttätig das Servo-Befehlsprotokoll abwickelt (z. B. Robotis U2D2). Arduino-Shields für autarke Dynamixel-Roboter sind ebenso verfügbar.

Robotis hat spezielle Robotik-Servos mit universellen Befestigungsmöglichkeiten im Angebot, darunter auch Servos mit durchgehender oder zentraler Achse, was den Aufbau von Roboterarmen erleichtert. Es war nur eine Frage der Zeit, bis auch andere Hersteller die Idee digitaler Roboter-Servos aufgriffen.

Im Zusammenspiel mit einem Greifer oder einem pneumatischen Sauger können solche Geräte bereits leichte Sortier- oder Fügearbeiten erledigen, auch wenn sie noch in der Klasse „pädagogische Roboter" laufen. Roboterarme dieser Art sieht man oft auf Messen, wo sie zum Beispiel am Miniatur-Förderband die Leistungsfähigkeit von Machine-Vision-Systemen demonstrieren und Schokoladendragees sortieren. Die im unteren Millimeterbereich anzusiedelnde Wiederholgenauigkeit macht sie aber auch im Labor interessant, wo sie zum Beispiel Proben ordnen und einlegen können.

Semiprofis

Zwischen 250 und 700 Euro ist das Angebot eher dünn, erst in der Preisspanne 700 bis 1500 Euro wird es wieder dichter. So viel Geld wird man nicht zum Spaß (im Wortsinn) ausgeben; dem potenziellen Anwender wird also ein konkreter Anwendungsfall vorliegen, etwa die Bauteilebestückung (Pick and Place), das Auftragen von Kleber auf ein Werkstück oder die Vereinzelung von Kleinteilen.

Für rund 870 Euro erhält man beispielsweise bei Reichelt den UFactory Swift Pro, der bei einem Arbeitsradius von 320mm immerhin schon eine Wiederholgenauigkeit von 0,2mm bietet und 500g tragen kann. Das kostenpflichtige Zubehör umfasst unter anderem einen Greifer, eine 3D-Druck-Düse oder ein OpenMV-kompatibles Machine-Vision-Kit.

Die gleichen Leistungsdaten bietet der MakerFactory CCR-45, erhältlich zum Beispiel bei Conrad. Bemerkenswert ist bei beiden Geräten die gemischte Bestückung mit Getriebe-Schrittmotoren und Modellbau-Servos. Die Gelenkhebel-Konstruktion erlaubt es, zwei Gelenkantriebe in der Nähe der Basis zu montieren, so dass der Arm nur sich selbst, aber nicht den Antrieb tragen muss.

Wer kein Freund von Open-Frame-Konstruktionen ist, wird eher zu den Dobot-Roboterarmen von Variobotic greifen, die in Form und Funktion schon sehr an ihre großen Brüder aus der Industrie angelehnt sind. In den Leistungsdaten ist der 1350 Euro teure Dobot Magician Basic mit dem UFactory Swift Pro vergleichbar, bietet aber etwas Luxus obendrauf. Neben dem gefälligen Äußeren ist die „Teach and Playback"-Funktion erwähnenswert: Man kann dem Roboter bestimmte Abläufe beibringen, indem man den Effektor von Hand führt; so etwas kennt man sonst nur von den Industrie-Pendants. Die 200 Euro teurere „Advanced"-Ausführung des Dobot Magician bietet einige Schnittstellen (WLAN, Bluetooth) obendrauf, außerdem wird ein Gamepad zur Fernsteuerung mitgeliefert.

Wer noch mehr Geld ausgeben kann (oder muss), wird erst wieder in der Gegend um 5000 Euro fündig. Hier gibt es Geräte mit 600 bis 700mm Reichweite und 500g Traglast (etwa beim Robolink-Komplettpaket von igus) oder aber kräftige Geräte mit kleinerer Reichweite, aber deutlich höherer Tragfähigkeit (z. B. 1,5kg beim Dobot M1). —*cm*

MakerFactory CCR-45 von Conrad: Hybride Bestückung mit Getriebe-Schrittmotoren und Modellbau-Servos. Eine sehr ähnliche Konstruktion findet man beim UFactory Swift Pro.

Bild: Conrad

Semiprofi-Klasse: Der Dobot Magician kommt mit einem Rundum-Sorglos-Softwarepaket.

Das DIY-Kompendium!

Richtig loslegen mit Raspi, ESP & Co.

Jetzt für nur **12,90 €** bestellen.

JETZT NEU! c't Projekte 2019

c't Projekte 2019 führt in die Welt der Einplatinencomputer ein, stellt Plattformen vor und vermittelt Know-How für anspruchsvollere Projekte. In zahlreichen Bau- und Programmiervorschlägen finden Einsteiger wie Fortgeschrittene Anleitungen zum Nachbau und Anregungen für eigene Ideen.

Auch komplett digital erhältlich!

shop.heise.de/ct-projekte19

12,90 € ›

Auch als Bundle erhältlich

Mit 2GByte Raspberry Pi 4-Starterset Classic Edition

- Inkl. c't Projekte Sonderheft
- Raspberry Pi 4 Model B 2GB RAM
- Offizielles Netzteil USB-C
- Offizielles Gehäuse (rot/weiss)
- Sandisk microSDHC 16GB inkl. NOOBS
- Micro HDMI - HDMI Kabel

shop.heise.de/raspi4-bundle

84,90 € ›

› Generell portofreie Lieferung für Heise Medien- oder Maker Media Zeitschriften-Abonnenten oder ab einem Einkaufswert von 15 €. Nur solange der Vorrat reicht. Preisänderungen vorbehalten.

heise shop

shop.heise.de/ct-projekte19 ›

Roboter für Kinder

Kinder lieben Roboter! Dank ihrer Fantasie machen sie schon Bürsten mit Motor und Wackelaugen zu ihren besten Freunden. Das kann man sich zunutze machen, um ihnen das Programmieren beizubringen. Wir geben hier eine Übersicht über Lernroboter – und decken dabei die Altersgruppen von 4 bis 100 Jahren ab.

von Elke Schick

Grundschule

My first Robot

Funktionsweise: „My first Robot“ besteht aus von Tinkerbot entwickelten Bauteilen. Zum einen sind das die Cubies, Bausteine aus Plastik, die nach dem gleichen Prinzip wie Legosteine aufgebaut und auch mit diesen kompatibel sind. Das Herzstück des Roboters ist das sogenannte „Powerbrain“ – ein quadratisches Plastikgehäuse, das den Lithium-Ionen-Akku und Mikrocontroller enthält. Das Powerbrain hat auf einer Seite mehrere Taster zur Steuerung und auf den anderen Seiten Anschlüsse für Motoren oder Sensoren. Mit den mitgelieferten Motoren kann man die Räder des Roboters steuern.

Programmierung und Lehrmethode: Die Kinder lernen das Programmieren bei diesem Roboter über eine App. Zuerst lösen sie im Game-Modus der App verschiedene Aufgaben. Dabei müssen die Kinder die Bewegungen des Roboters in Codeblöcke übersetzen. Mit jeder gelösten Aufgabe schalten die Kinder neue Programmier-Blöcke frei, die sie dann im Programmier-Modus der App verwenden können. Sie können Bewegungen des Motors, Geschwindigkeiten und Geräusche miteinander kombinieren. Die App kommt allerdings komplett ohne Erklärungen aus, sodass die Kinder nur lernen, die Blöcke mit Bewegungen und Geräuschen des Roboters zu verknüpfen. Allerdings kann man mit dem Roboter auch die Tinkerbots Blockly App benutzen, in der grundlegende Programmierkonzepte vermittelt werden. Der Roboter ist nicht mit den Arduino-Sets des Herstellers kompatibel.

Anleitungen und Community: Die Aufbauanleitung für den Roboter kommt komplett ohne Text aus, ist sehr kleinschrittig und für Kinder sicherlich gut nachvollziehbar. Auch die App ist übersichtlich. Tinkerbots hat auf seiner Webseite keinen Community-Bereich, ist aber immer sehr bemüht, selbst für Fragen und Antworten zur Verfügung zu stehen.

Hersteller	Tinkerbots
Preis	129,95 Euro
Altersempfehlung	ab 5 Jahren
Voraussetzungen	Smartphone oder Tablet mit Bluetooth 4.0 und Android 5.0+ beziehungsweise iOS 8.2+, keine Kenntnisse vorausgesetzt
Programmiersprache	Blockly (JavaScript- oder Python-Code-Export)

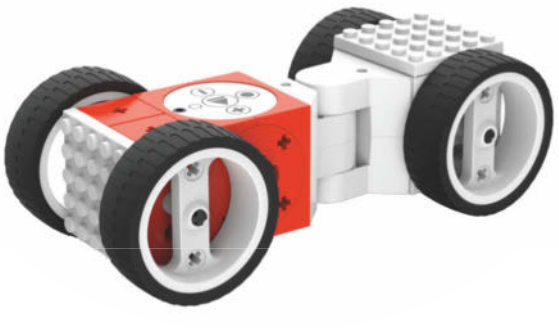
Robotics Starter Set

Funktionsweise: Das Starter Set besteht aus dem beim vorigen Roboter schon erläuterten Powerbrain, einem Motormodul, einem Gelenkmodul, genannt Pivot, und den Cubies. Im Gegensatz zum „My first Robot“ kann man hier verschiedene Modelle bauen, die von diesen Komponenten angetrieben werden.

Programmierung und Lehrmethode: Mit der Tinkerbots World App kann man die Roboter steuern. Entweder legt man dort neue Bewegungsabläufe fest oder man nutzt die App als Fernsteuerung. Außerdem ist es möglich, im Aufnahmemodus des Powerbrains Bewegungsabläufe zu speichern. Für das Erlernen von grundlegenden Prinzipien des Programmierens kann man die Tinkerbots Blockly App mit den Robotern verwenden. Diese App ist für Windows und macOS erhältlich, läuft also nicht auf dem Smartphone, sondern auf dem PC. Der Roboter ist nicht mit den Arduino-Sets des Herstellers kompatibel.

Anleitungen und Community: Die Anleitung für den Aufbau der verschiedenen Modelle arbeitet fast nur mit Zeichnungen. Kinder können mit ihr und der App sicherlich relativ selbstständig arbeiten. Eine Community gibt es nicht, der Hersteller hilft aber selbst gerne bei Problemen.

Hersteller	Tinkerbots
Preis	189,95 Euro
Altersempfehlung	ab 6 Jahren
Voraussetzungen	Smartphone oder Tablet mit Bluetooth 4.0 und Android 5.0+ beziehungsweise iOS 8.2+, keine Kenntnisse vorausgesetzt
Programmiersprache	Blockly (JavaScript- oder Python-Code-Export)



Ozobot Bit 2.0

Funktionsweise: Die nur 3cm großen Roboter enthalten einen Mikrocontroller, einen Li-Po-Akku, Mikromotoren, Helligkeits- und Farbsensoren. Man kann mit den passenden Filzstiften Strecken zeichnen, die die kleinen Roboter abfahren. In diese Strecken kann man Befehle für das Verhalten der Roboter einbauen, indem man vorgegebene Farbcodes einbettet. Fügt man zum Beispiel in eine Strecke einen Abschnitt ein, in dem ein blauer, ein grüner und ein roter Strich aufeinander folgen, so wird der „Super Turbo" ausgelöst. Hierbei fährt der Ozobot für einen kurzen Moment sehr schnell. Für den Ozobot Bit gibt es 33 Befehle dieser Art, die Kinder auf verschiedene Arten und Weisen kombinieren können.

Programmierung und Lehrmethode: Die Programmierung über Farbcodes ist der Einstieg in die Nutzung der Ozobots. Später können die Roboter über verschiedene spielerische Apps auf dem Tablet oder Smartphone und auch am PC mit Blockly programmiert werden. In Ozobot Blockly kann man komplette Programme für den Ozobot zusammenstellen und diese auf den Roboter übertragen.

Anleitungen und Community: Ozobots bietet viel Lehrmaterial für den direkten Einsatz im Unterricht an. Von der großen Menge an Materialien, die es auf der US-amerikanischen Seite des Herstellers gibt, ist ein Teil auch auf Deutsch übersetzt. Für den Ozobot Bit gibt es zurzeit insgesamt 5 Lektionen, an weiteren Übersetzungen wird gearbeitet.

Hersteller	Ozobot
Preis	69,95 Euro
Altersempfehlung	ab 6 Jahren
Voraussetzungen	Smartphone oder Tablet mit Bluetooth 4.0 und Android 6.0+ beziehungsweise iOS 10.0+, keine Kenntnisse vorausgesetzt
Programmiersprache	Blockly (JavaScript- oder Python-Code-Export)

Bausatz Cubelets Curiosity

Funktionsweise: Die Cubelets beruhen auf dem Prinzip der Bauklötze – der Unterschied ist, dass hier die Klötze aus Plastik sind und sowohl Hardware als auch Software enthalten. Die Klötze können magnetisch miteinander verbunden werden und zeigen in unterschiedlichen Kombinationen verschiedene Verhaltensweisen. Das Curiosity-Set besteht aus 4 Action-Cubelets zur Fortbewegung und für Lichtsignale, 3 Sensor-Cubelets (Entfernungsmesser und Lichtsensor) und 4 Think-Cubelets, die die Funktionen von anderen Bausteinen verändern können oder für die Stromversorgung oder die Bluetooth-Verbindung zum PC da sind.

Programmierung und Lehrmethode: Das Curiosity-Set ist vor allem dazu da, Kindern zu ermöglichen, erste Programmier- und Robotik-Erfahrungen ohne Computer zu machen. Das erlaubt es, die Blöcke auch für jüngere Kinder einzusetzen. Später können die Roboter auch per Bluetooth mit der Cubelets-App auf dem Tablet oder Smartphone oder der Blockly-App auf dem PC verbunden und selbst programmiert werden.

Anleitungen und Community: Auf der Webseite des Herstellers gibt es viele Materialien für Eltern und Pädagogen. Auf der Grundlage dieser Materialien kann man die Cubelets im Unterricht einsetzen. Neben konkreten Anwendungsbeispielen erhält man hier auch viel pädagogischen Hintergrund für die Anwendung der Robotik-Bauklötze. Die Knowledge Base des Herstellers ist besonders für Lehrende interessant, die pädagogische Grundlagen für die digitale Bildung suchen. Außerdem bietet die Webseite Tipps für den Download der Apps oder der anderen Software auf Netzwerken, die starken Sicherheitseinschränkungen unterliegen.

Hersteller	Modrobotics
Preis	249,90 Euro
Altersempfehlung	ab 4 Jahren
Voraussetzungen	Smartphone oder Tablet mit Bluetooth 4.0 und Android 6.0+ beziehungsweise iOS 10.0+, keine Kenntnisse vorausgesetzt
Programmiersprache	Blockly (JavaScript- oder Python-Code-Export), Cubelets Flash (C), kann um Komponenten erweitert werden, die eine Programmierung mit Scratch erlauben

Bee-Bot

Funktionsweise: Alle Komponenten des Bee-Bot befinden sich im Plastik-Bienenkörper. Der Bee-Bot enthält einen Mikrocontroller, der die Motoren, Distanzsensoren, LEDs und Geräusche des Roboters steuert. Als Stromquelle enthält der Bee-Bot einen Akku.

Programmierung und Lehrmethode: Dieser Roboter dient vor allem dazu, Kindern logisches Denken, Zählen und Raumwahrnehmung zu vermitteln. Mit den Tasten auf dem Roboter kann man bestimmte Bewegungsabläufe wie zum Beispiel „drei Schritte nach vorne, eine Drehung nach links“ eingeben. Der Bee-Bot speichert diese Eingaben und setzt sie nach einem Druck auf die mittlere Go-Taste in Bewegungsabläufe um. Lehrmaterial für den Bee-Bot gibt es in Form von Spielmatten und Karten.

Anleitungen und Community: Eine Anleitung für den Bee-Bot findet man online auf der Webseite des US-amerikanischen Herstellers. Dort gibt es auch einen ausführlichen „Teacher's Guide“, aber keinen Community-Bereich. Der „Teacher's Guide“ist für die Lehrpläne in Großbritannien konzipiert, aber die Unterrichtsvorschläge lassen sich in allen Kindergärten und Grundschulen gut anwenden.

Hersteller	TTS Group Ltd.
Preis	84,95 Euro
Altersempfehlung	ab 4 Jahren
Voraussetzungen	keine technischen Voraussetzungen, keine Vorkenntnisse notwendig
Programmiersprache	kann nur über Tasten programmiert werden

Dash

Funktionsweise: Der mit Motoren, zahlreichen LEDs, Abstandssensoren, Mikrofonen und einem Lautsprecher ausgestattete Dash muss nicht zusammengesetzt oder eingerichtet werden. Kinder, die dazu neigen, ihren Spielzeugen eine eigene Persönlichkeit zuzuschreiben, werden dieses als ersten echten Roboterfreund vermarktete Produkt lieben. Mit der Vielzahl an vorprogrammierten Geräuschen, seiner Mimik und den voreingestellten Bewegungsabläufen wirkt er niedlich. Für Dash gibt es 5 verschiedene Apps, mit denen er programmiert werden kann.

Programmierung und Lehrmethode: In jeder der verschiedenen Apps gibt es eine eigene Geschichte, in die die Programmieraufgaben eingebunden sind. Dabei kommt an verschiedenen Stellen die Programmiersprache „Wonder“ zum Einsatz, in der die einzelnen Funktionen des Roboters als Elemente am unteren Bildschirmrand angezeigt werden. Sie müssen in die Mitte des Bildschirms gezogen und in der richtigen Reihenfolge miteinander verbunden werden. In anderen Apps gibt es Steuerungsoberflächen, die wie ein Set aus Reglern aufgebaut sind. Außerdem gibt es eine Dash-Blockly-App, die in allen Unterrichtsvorschlägen und Workshops eingesetzt wird.

Anleitungen und Community: Für Dash gibt es im deutschsprachigen Bereich keine Materialien. Auf der Webseite des US-amerikanischen Herstellers hingegen findet man eine Vielzahl von Ressourcen. Darunter finden sich Unterrichtskonzepte, Webinars, Lektüreempfehlungen und zahlreiche Aufrufe zu Wettbewerben.

Hersteller	Wonder Workshop
Preis	179,99 Euro
Altersempfehlung	ab 6 Jahren
Voraussetzungen	Smartphone oder Tablet mit Bluetooth 4.0 und Android 4.4.2+ beziehungsweise iOS 9.3+, keine Kenntnisse vorausgesetzt
Programmiersprache	Blockly (JavaScript- oder Python-Code-Export)

Lego Boost

Funktionsweise: Der Controller des Lego Boost nennt sich Lego Move Hub. Er enthält 2 Positionsmotoren, einen Neigungssensor, das Batteriegehäuse und Anschlüsse für die Verbindung mit weiteren Motoren oder Sensoren aus der „Powered Up"-Reihe des Herstellers. Außerdem gehören noch ein Farb- und Entfernungssensor und ein externer Motor zum Basisset. Die Bestandteile des Move Hub sind, wie beim EV3 oder den Tinkerbot-Robotern auch, in ein Plastikgehäuse integriert. Alle Bauanleitungen und die Programmierung des Controllers sind in der Lego-Boost-App enthalten. Die Verbindung zwischen App und Roboter stellt man über Bluetooth her. Mit dem Basis-Set kann man fünf verschiedene Modelle bauen – darunter sind eine Katze, eine Gitarre und eine automatische Fertigungsmaschine, die Legosteine zusammensetzen kann.

Programmierung und Lehrmethode: Zum Boost gehört nur eine App für Android, Mac- oder Windows-Tablets. Smartphones werden bisher noch nicht unterstützt, weil die App für die Bildschirmgröße von Tablets entwickelt wurde. Für die Programmierung fügen die Kinder Blöcke mit verschiedenen Funktionen zusammen. Jedes Modell hat eigene Blöcke, die zwar technisch immer das Gleiche ansteuern (nämlich die Motoren und Sensoren), aber optisch an das jeweilige Computermodell angepasst sind. Für jedes Modell gibt es damit eigene Programme – die Katze schnurrt und folgt mit ihren Augen einem Lego-Fisch, der Roboter fährt auf zwei Rädern und kann mit einer integrierten Zwille schießen und für die Gitarre kann man verschiedene Klangarten auswählen. Zusätzlich können sich die Kinder eigene Modelle ausdenken und deren Programmierung in der App speichern. Da der Boost im Vergleich zum Mindstorms EV3 eher einfach und spielerisch gehalten ist, tauchen hier keine Programmier-Fachbegriffe auf.

Anleitungen und Community: Alle Anleitungen sind in die App integriert und können außerdem am PC heruntergeladen werden. Sie sind übersichtlich und nachvollziehbar. Auch die Bedienung der App ist einfach. Da der Boost nicht so viele Funktionen und Schnittstellen hat wie der Mindstorms EV3, gibt es für ihn nur eine eher kleine, von Lego unabhängige Community. Diese findet man zum Beispiel in Form einer Facebook-Gruppe.

Hersteller	Lego
Preis	159,99 Euro
Altersempfehlung	ab 7 Jahren
Voraussetzungen	Smartphone oder Tablet mit Bluetooth 4.1+ und Android 5.3+ beziehungsweise iOS 10.3+ oder Windows 10, keine Kenntnisse vorausgesetzt
Programmiersprache	eigene Block-Programmierung, die Scratch ähnelt

Weiterführende Schule

Robotics TXT Controller

Funktionsweise: Fischertechnik bietet 6 verschiedene Robotik-Bausätze an. Für die größeren Kinder sind die Sets aus der Robotics-TXT-Serie interessant. Gesteuert werden die Roboter aller Sets, ob es sich jetzt um fahrende Modelle, Industrieroboter oder eine Umweltdaten-Messstation handelt, vom sogenannten Robotics TXT Controller. Da der Controller das Kernstück aller Sets ist, eigentlich auch ohne diese verwendet werden kann und ordentlich Geld kostet, stellen wir hier nur ihn vor. Der Robotics TXT wird über ein Touch-Display bedient und läuft mit einem Arm Cortex A8 und einem Cortex M3. Er hat 8 Universaleingänge, 4 digitale Eingänge, 4 Motorausgänge, ein Bluetooth- und WLAN-Funkmodul. Außerdem verfügt er über einen MikroSD-Karten-Slot, eine Mini-USB-Buchse, eine USB-A-Buchse, einen Lautsprecher und eine Echtzeituhr. Das Betriebssystem des Controllers ist Linux-basiert und Open Source. Mit der Software „Robo Pro" – die man zusätzlich erwerben muss – kann man ihn programmieren.

Programmierung und Lehrmethode: In der visuellen Programmierumgebung von Robo Pro ziehen die Kinder die verschiedenen Komponenten eines Projekts auf einen Arbeitsbildschirm und verbinden sie dort miteinander. Die Bedienoberfläche der Software ist im Vergleich zu den anderen hier gezeigten Produkten eher altmodisch und von Gamification kann bei Fischertechnik wirklich nicht die Rede sein. Dennoch: Kein anderes Produkt hier bietet Open-Source-Software auf Linux-Basis an. Und die mitgelieferte Dokumentation ist wirklich ausführlich.

Anleitungen und Community: Für alle Sets gibt es ein didaktisches Begleitheft, das online zum Download zur Verfügung steht. Auch dieses Material wirkt im Vergleich zu den anderen Anleitungen hier sehr trocken und altmodisch. Es ist aber solide gemacht und erlaubt es den Kindern, die Roboter wirklich zu nutzen.

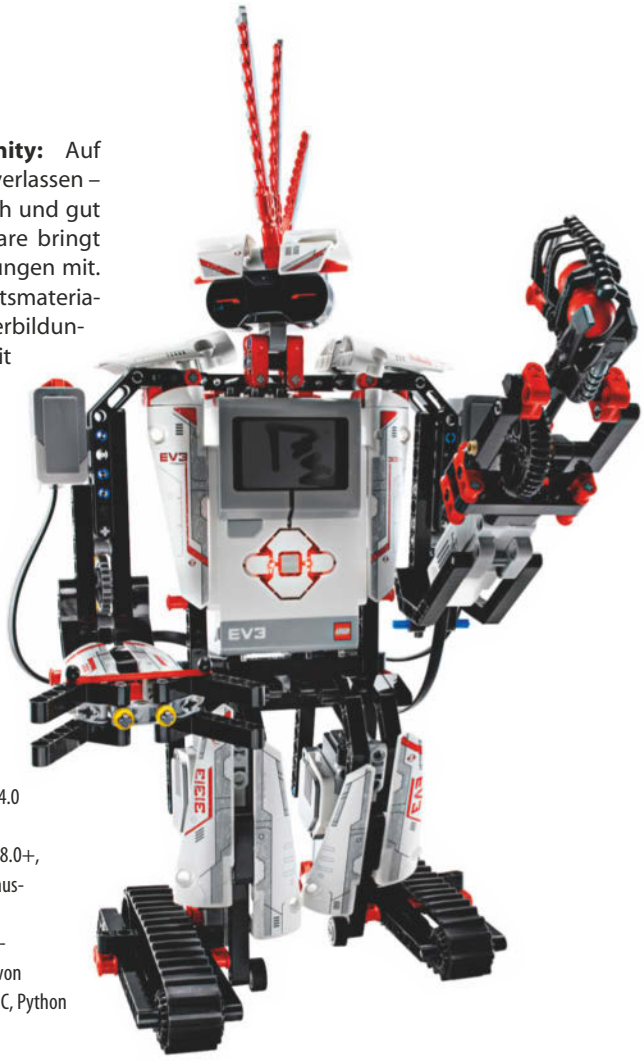
Hersteller	Fischertechnik
Preis	349,95 Euro
Altersempfehlung	ab 10 Jahren
Voraussetzungen	Windows 7, 8 oder 10 und ein Kind, das sich wirklich für Technik interessiert
Programmiersprache	ROBO Pro, visuelle Programmierumgebung

Lego Mindstorms EV3

Funktionsweise: Das Kernstück des Mindstorms EV3 ist der EV3-Stein, ein Plastikgehäuse, in das ein ARM9-Prozessor, ein Display, Taster, ein USB-Port, ein MicroSD-Kartenleser und 4 Motoranschlüsse integriert sind. Man kann den Roboter über WLAN und per Infrarot mit einer App steuern. Er enthält außerdem drei Servomotoren, einen Farbsensor, einen Berührungs- und einen Infrarotsensor. Mit den mitgelieferten 601 Lego-Bausteinen können die Kinder verschiedenste Modelle bauen. Über PC-Software und eine App kann der Roboter programmiert und gesteuert werden.

Programmierung und Lehrmethode: Für den Mindstorms stehen eine PC- beziehungsweise Mac-Software und eine App für die Programmierung, eine App für die Steuerung und eine Spiel-App zur Verfügung. Die verwendete Programmier-Sprache ist von Lego selbst entwickelt und basiert auf Labview. Die Kinder fügen für die Programmierung Blöcke zusammen, mit denen sie das Verhalten der Motoren und Sensoren steuern. Die Funktionen der verschiedenen Programmierblöcke sind gut nachvollziehbar. Im der Programmier-Umgebung werden typische Begriffe wie „Schleife“ verwendet und man kann zum Beispiel auch logische Verknüpfungen oder Arrays in seine Programme einfügen.

Anleitungen und Community: Auf Lego-Anleitungen kann man sich verlassen – sie sind grundsätzlich übersichtlich und gut nachvollziehbar. Auch die Software bringt viele Erklärungen und Unterstützungen mit. Lego bietet ausführliche Unterrichtsmaterialien für Lehrende und auch Weiterbildungen an. Außerdem gibt es weltweit eine aktive Community, die den Mindstorms für die verschiedensten Projekte verwendet.



Hersteller	Lego
Preis	349,99 Euro
Altersempfehlung	ab 10 Jahren
Voraussetzungen	Smartphone oder Tablet mit Bluetooth 4.0 und Android 4.2.2+ beziehungsweise iOS 8.0+, keine Kenntnisse vorausgesetzt
Programmiersprache	visuelle Programmierumgebung auf Basis von Labview, zudem sind C, Python und Java möglich

Kodi

Funktionsweise: Dieser Roboter wird rein mechanisch programmiert. Kodi ist in Form eines Bausatzes erhältlich und enthält ein sogenanntes Coding-Wheel, in das Pins eingesetzt werden, um die verschiedenen Programme zu aktivieren. Mit den passenden Umbauten am Roboter kann Kodi greifen, vorwärts, rückwärts und in Kurven fahren, Fußball spielen oder zeichnen.

Programmierung und Lehrmethode: Im Rahmen einer Geschichte, die als Comic erzählt wird, lernen die Kinder die Grundlagen der mechanischen Programmierung. Sie erfahren, was Lochkarten sind und wie sich die mechanische Programmierung zur Computerprogrammierung verhält.

Anleitungen und Community: Wie die meisten Anleitungen von Kosmos ist die zu diesem Baukasten sehr gut. Die Kinder können die einzelnen Arbeitsschritte gut nachvollziehen – ganz ohne Hilfe werden sie den Roboter nicht aufbauen, aber das gilt auch für die anderen Produkte in diesem Artikel.

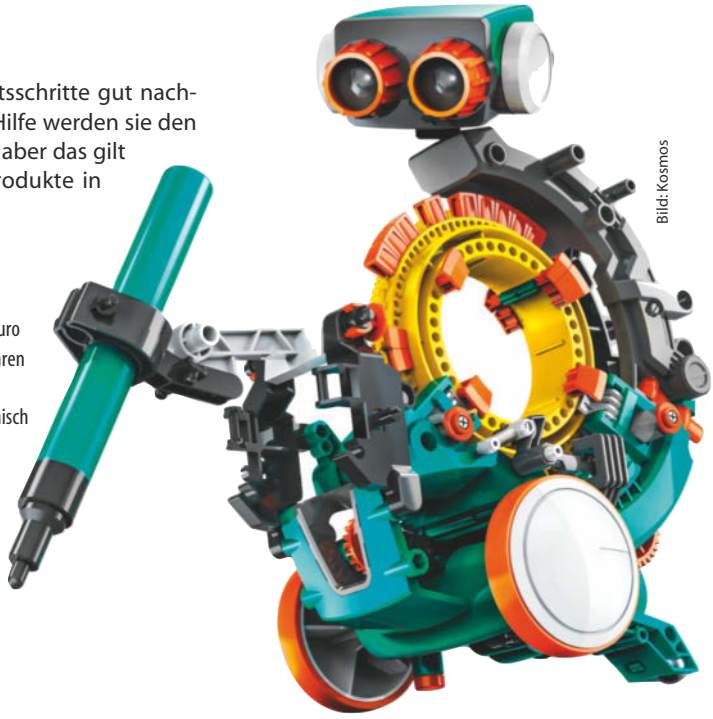


Bild: Kosmos

Hersteller	Kosmos
Preis	44,99 Euro
Altersempfehlung	ab 8 Jahren
Voraussetzungen	keine
Programmiersprache	mechanisch

mBot Explorer Kit

Funktionsweise: Der mBot ist explizit für den Einsatz im Unterricht konzipiert. Er funktioniert aber auch prima als Spielzeug. Diesen Roboter müssen die Kinder selbst zusammenbauen. Dabei setzen sie alle Elemente, wie das Mainboard, die Motoren, Sensoren und die Räder, auf ein Chassis aus Metall. Der mBot verfügt über einen Lichtsensor, einen IR-Sensor, einen Ultraschallsensor, zwei Linienfolge-Sensoren und Bluetooth. Er ist mit Bausteinen wie denen von Lego kompatibel und kann so erweitert werden. Im Kit ist außerdem eine LED-Matrix enthalten, die man auf die Front des Roboters montiert. Sobald der mBot aufgebaut ist, kann er direkt losfahren. Dafür kann man zwischen drei Modi wählen. Im Hindernis-Erkennungs-Modus fährt der Roboter selbstständig durch die Gegend, im Linienfolge-Modus fährt er auf vorgegebenen Strecken und im Fernsteuerungsmodus kann man ihn über eine Fernbedienung oder App lenken.

Programmierung und Lehrmethode: Für den mBot gibt es mehrere Apps. Die Makeblock-App kann für die Fernsteuerung und Programmierung des Roboters genutzt werden. Die mBlock Blockly App ist als Einführung in die Programmierung gedacht. Hier werden durch Spiele erste Programmierschritte und -techniken erklärt. Außerdem gibt es die mBlock-Software für den PC. Diese Software basiert auf Scratch 3.0, der Code kann in Arduino-C exportiert werden.

Anleitungen und Community: Sowohl die Aufbau- als auch die Programmieranleitungen sind gut aufgebaut und übersichtlich. Kinder ab 8 Jahren können weitgehend selbstständig mit diesen Materialien arbeiten. Makeblock bietet eine Vielzahl von Materialien für Lehrende an – bisher allerdings hauptsächlich auf Englisch. Der Hersteller selbst bietet für die Community ein Forum, in dem hauptsächlich Probleme geklärt werden, und ein OpenLab für Projekte mit den Produkten. Außerdem gibt es eine Reihe von Lehrbüchern für den mBot.

Hersteller	Makeblock
Preis	99,95 Euro
Altersempfehlung	ab 8 Jahren
Voraussetzungen	Smartphone oder Tablet mit Bluetooth 4.0 und Android 5.0+ beziehungsweise iOS 9.0+, PC mit Win7 oder Win10 (64-Bit) oder macOS 10.10+, keine Kenntnisse vorausgesetzt
Programmiersprache	Blockly (Apps), Scratch 3.0 (PC)

Metabot

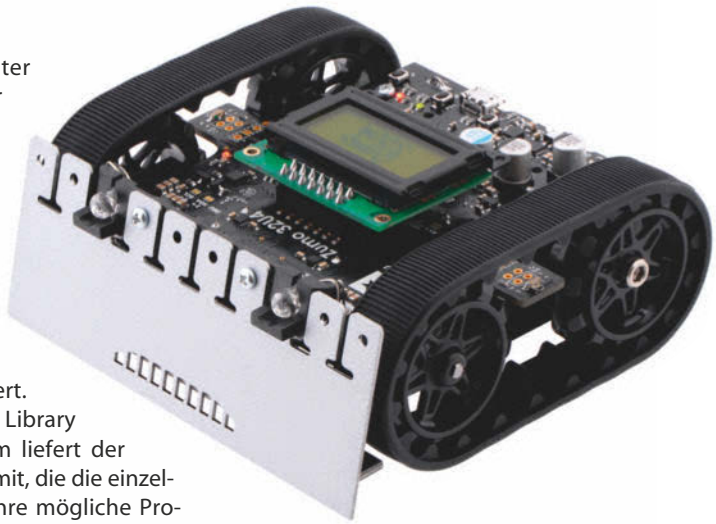
Funktionsweise: Der Metabot ist kein käuflich erhältliches Produkt, sondern ein Open-Source-Projekt. Die mechanischen Teile muss man sich selbst mit dem 3D-Drucker oder Lasercutter herstellen. Die Elektronik ist im Einzelhandel erhältlich. Auf der Metabot-Webseite findet man alle notwendigen Dateien und Teilelisten. Der vierbeinige Roboter kann nach dem Zusammensetzen über Bluetooth mit einer PC-Software oder auch dem Terminal programmiert und mit einer Android-App ferngesteuert werden. Jedes der vier Beine wird dabei von drei Motoren bewegt. Als Controller kommen ein Xbee und ein Maple Mini zum Einsatz, die mit einer inertialen Messeinheit ergänzt werden. Außerdem verfügt der Metabot über einen Distanzsensor.

Programmierung und Lehrmethode: Die „Metabot Blocks"-Software arbeitet mit der visuellen Block-Programmierung. Die Programmierumgebung ist nur auf Englisch oder Französisch verfügbar. Aber da sie auf Scratch basiert und daher auf wiedererkennbaren Prinzipien beruht, macht das die Nutzung nicht schwieriger. Trotzdem sollte man sich für die Verwendung des Metabots bereits mit Roboterprogrammierung auskennen. Der Hilfe-Bereich der Software ist nämlich noch komplett auf Französisch.

Anleitungen und Community: Auf der Webseite des Projekts findet man die Anleitungen zum Zusammenbau – sie sind reich bebildert und gut nachvollziehbar. Die Anleitung für die Nutzung der Programmierumgebung besteht aus einem französischsprachigen YouTube-Video. Hat man bereits etwas Programmiererfahrung, versteht man die Grundlagen auch dann, wenn man die Sprache nicht spricht. Für den Metabot gibt es auch einen Community-Bereich auf der Webseite, der allerdings nicht genutzt wird.

Hersteller	keiner
Preis	Kosten für die Anschaffung und den 3D-Druck der Teile
Altersempfehlung	ab 16 Jahren
Voraussetzungen	Linux oder Windows, Erfahrungen mit Scratch und Roboterprogrammierung
Programmiersprache	Block-Programmierung auf Scratch-Basis

Zumo 32U4



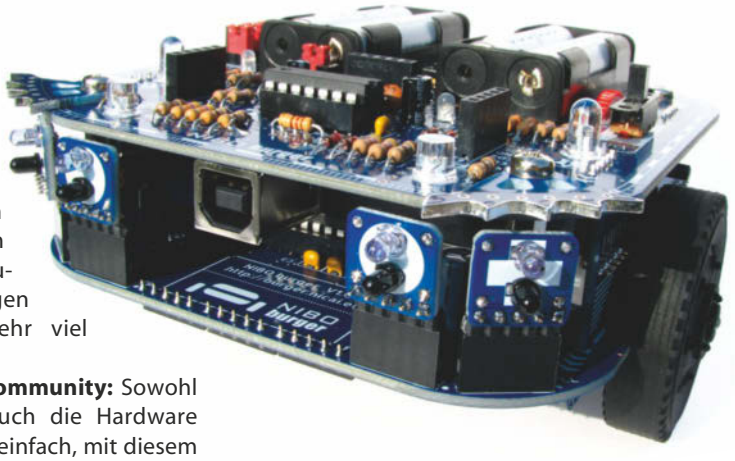
Funktionsweise: Okay, okay, mit diesem Roboter bewegen wir uns jetzt eindeutig im Oberstufen- und Hochschulbereich, aber Kinder sollen auch nicht den ganzen Spaß nur für sich haben. Der Zumo 32U4 ist fürs Kämpfen im Robotersumo in der Klasse Mini-Sumo konzipiert. Wie der Name schon sagt, basiert er auf dem Arduino-kompatiblen ATmega32U4. Zusammengebaut ist er nur knapp 10 × 10cm groß und wiegt mit Batterien 275g. Beim Robotersumo geht es darum, dass zwei Roboter versuchen, einander aus einer Kampfarena zu schieben. Dafür werden die Roboter häufig mit Schaufeln ausgestattet. Hier dient dazu das Zumo Shield, das man an die Front anbauen kann. Der Roboter verfügt über zwei 50:1-HP-Mikro-Metall-Getriebemotoren mit integrierten Dual-Motortreibern, ein LC-Display, Quadratur-Encoder (Drehgeber), Linienfolger und 5 Distanzsensoren vorne und an den Seiten. Außerdem ist eine komplette inertiale Messeinheit, die aus einem 3-Achsen-Beschleunigungsmesser, einem Kreisel und einem Magnetometer besteht, in den Roboter integriert. Sie dient dazu, Zusammenstöße zu erkennen und Orientierung bei der Verfolgung des Gegners zu bieten.

Programmierung und Lehrmethode: Der Zumo wird in der Arduino-IDE programmiert. Dafür steht eine eigene Library zur Verfügung. Außerdem liefert der Hersteller eine Anleitung mit, die die einzelnen Komponenten und ihre mögliche Programmierung erklärt.

Anleitungen und Community: Die Anleitung ist zwar recht textlastig, aber der Komplexität des Roboters angemessen. Gleichzeitig werden die Dinge hier nicht unnötig verkompliziert. Außerdem gibt es viele Hinweise auf Ressourcen bei Github und weitere Dokumentationen des Herstellers.

Hersteller	Pololu
Preis	139,94 Euro
Altersempfehlung	ab 16 Jahren
Voraussetzungen	Arduino-IDE, Programmiererfahrung
Programmiersprache	C, C++, Arduino, Java

Nibo Burger



Funktionsweise: Der als Bausatz erhältliche Nibo Burger verfügt über 2 AVR-Mikrocontroller, 2 Motoren und Antriebsräder, 3 Farbsensoren zur Farbanalyse von Oberflächen, 4 IR-Sensorbricks zur Kollisionsvermeidung und 2 IR-Sensoren mit Schmitt-Trigger, die zur Drehzahlmessung dienen. Die Drehzahlmessung wird für die Odometrie, also die Orientierung des Roboters im Raum, verwendet. Außerdem hat der Nibo Burger noch 10 Slots, in die man weitere Sensoren einsetzen kann. Der Hauptprozessor ist ein Atmega 16A, der für die Programmierung über USB um einen Attiny 44A ergänzt wurde. Außerdem gibt es einen Steckplatz für Arduino-Shields.

Programmierung und Lehrmethode: Der komplett frei programmierbare Roboter kann über die Arduino-IDE, AVR-Studio oder Roboter.cc, eine Web-basierte Entwicklungsumgebung mit Compiler für C, C++, Java und Arduino, programmiert werden. Für Anfänger bietet die Herstellerfirma ein Online-Tutorial an. Dieses Tutorial erklärt die Programmierung in sehr kleinen Schritten. Um zu überprüfen, ob man die Erklärungen verstanden hat, kann man je Arbeitsschritt ein Quiz ausfüllen. Die in das Tutorial integrierten kleinen Gimmicks wie Jubelschreie bei richtigen Antworten machen sehr viel Spaß.

Anleitungen und Community: Sowohl die Anleitungen als auch die Hardware selbst machen es Laien einfach, mit diesem Bausatz in die Robotik einzusteigen. Die Anleitungen sind sehr ausführlich und erklären wirklich alle Schritte, etwa auch das Herausbrechen der Platinen und Sensor-Bricks aus dem Rahmen. Durch das zahlreiche und genaue Bildmaterial wird man hier trotzdem nicht überfrachtet. Die Nibo-Burger-Community findet man in einem Forum auf Roboter.cc. Hier werden Fragen zum Roboter besprochen und Projekte vorgestellt. *—esk*

Hersteller	Nicai
Preis	75,95 Euro
Altersempfehlung	ab 16 Jahren
Voraussetzungen	ein PC mit einem einigermaßen aktuellen Betriebssystem und Interesse an der Robotik
Programmiersprache	C, C++, Arduino, Java

Distanzsensoren

Während normale Distanzsensoren nur sehr grob erfassen können, ob ein Hindernis vor ihnen liegt, bieten Laserscanner eine Rundumsicht. Auf diese Weise kann man nicht nur Hindernissen ausweichen, sondern von der Umgebung sogar Karten für die Navigation erstellen.

von Markus Knapp

Wer mal einem Staubsauger-Roboter bei der Arbeit zugesehen hat, fragte sich vielleicht, wie sich dieser in der Wohnung eigentlich zurechtfindet, ohne dabei ständig gegen Möbel oder Wände zu fahren. Damit die kleinen Helfer unfallfrei arbeiten, haben sie – je nach Preisklasse – unterschiedliche Sensoren eingebaut.

Infrarotsensoren sind für diesen Zweck die mit Abstand günstigsten Sensoren. Sie enthalten üblicherweise eine Infrarotdiode, die (unsichtbares) Infrarotlicht aussendet, und einen passenden Empfänger, der dieses Licht empfangen kann. Wie funktioniert das nun genau? Das ausgesendete Infrarotlicht wird von Gegenständen vor dem Sensor reflektiert. Jeder, der mal eine TV-Fernbedienung im richtigen Winkel auf eine Wand richtet , wird feststellen, dass das unsichtbare Licht von der Wand reflektiert wird und noch immer den Fernseher erreichen kann und dieser reagiert.

So ähnlich ist es beim Infrarotempfänger einiger Sensoren: Er misst die „Menge" des reflektierten Lichts und erzeugt an seinem Ausgang ein messbares Signal – zum Beispiel eine analoge Spannung von 0,5 bis 3 Volt. Diese Spannung kann beispielsweise ein Arduino an einem seiner analogen Eingänge messen und in eine Entfernung zum Objekt vor dem Sensor umrechnen.

Eine alternative Technik verwenden die von Robotikern oft eingesetzten Sensoren von Sharp. Der Empfänger misst nicht die Intensität, sondern an welcher Stelle seines Sensors das reflektierte Licht auftrifft. Per Triangulation leitet er dann den Abstand des Objektes ab. Der GP2Y0A21YK0F von Sharp ist solch ein Sensor und liegt preislich bei rund 11 Euro.

Solche einfachen Infrarotsensoren haben jedoch den Nachteil, dass sie empfindlich auf direktes Sonnenlicht reagieren können, da dieses einen hohen Infrarotanteil enthält. In der Folge liefert der Sensor falsche Werte und weicht von seinem Pfad ab, obwohl gar kein Hindernis vor ihm im Weg steht. Um diesem Problem aus dem Weg zu gehen, verwendet man stattdessen Ultraschallsensoren.

Ultraschall

Ultraschallsensoren arbeiten nach einem ganz anderen Prinzip als Infrarotsensoren. Sie senden für den Menschen nicht hörbare Schallimpulse aus. Der an einem Objekt reflektierte Schall wird von einem Empfänger erkannt. Aus der Zeitspanne zwischen Senden und Empfangen eines Impulses lässt sich bei bekannter Schallgeschwindigkeit die Entfernung berechnen. Ein typischer, von Bastlern häufig eingesetzter Ultraschallsensor ist beispielsweise der SRF05 von Devantech. Er kostet knapp 17 Euro.

Infrarot- und Ultraschallsensoren haben allerdings den Nachteil, dass sie immer nur in einem sehr schmalen Bereich die Entfernung eines Objektes messen können. Prinzipiell kann man den Sensor mit einem Servomotor winkelweise verstellen, das Ergebnis messen, speichern und abermals den Winkel verstellen, bis man beispielsweise einen Bereich von 180 Grad abgedeckt hat. Das nimmt jedoch viel Zeit in Anspruch, währenddessen etwa ein autonomer Roboter schon längst gegen die Wand gefahren sein kann.

Die Microsoft-Kamera Kinect

Lasersensoren arbeiten erheblich schneller, weil sich zum einen Licht erheblich schneller ausbreitet und sie zum anderen schnelle Prozessoren zur Auswertung einsetzen. Grundsätzlich gibt es zwei Verfahren, nach denen Laserscanner arbeiten: Structured Light und Time of Flight.

Kurzinfo

- So funktionieren Laserscanner
- Überblick typischer Modelle
- So setzt man sie unter Arduino und Pi ein

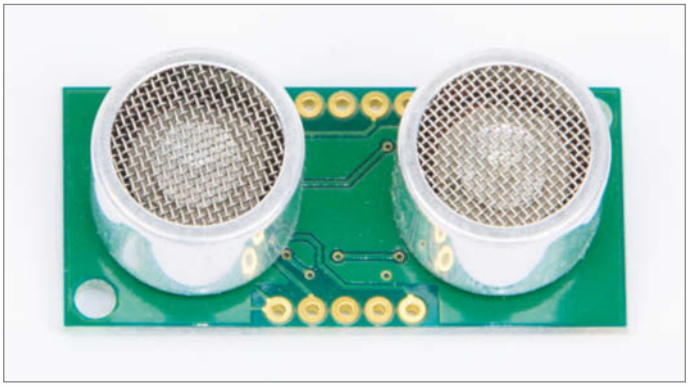
Alles zum Artikel im Web unter
make-magazin.de/x8p2

Microsoft Kinect

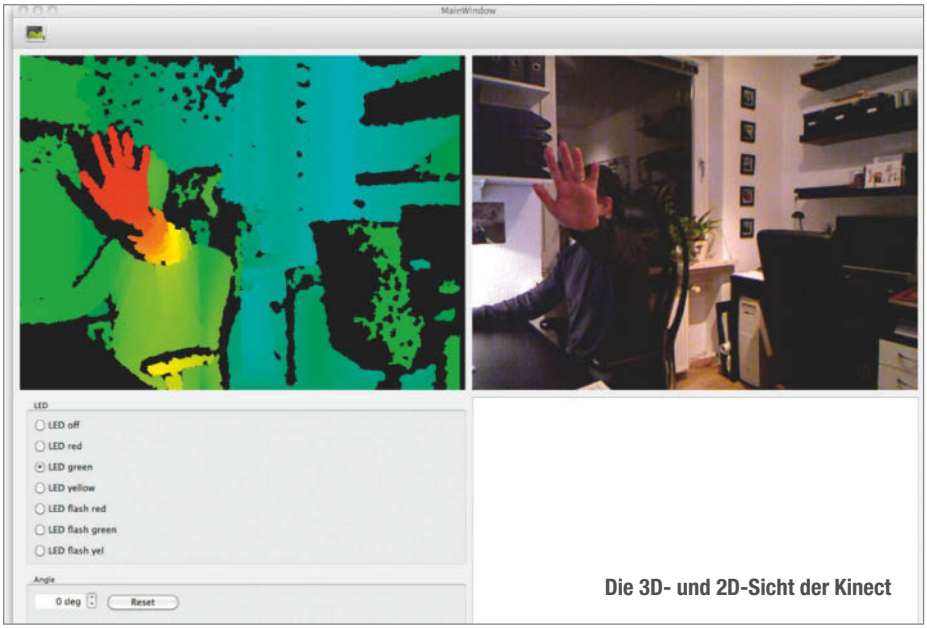
Die nach dem Structured-Light-Prinzip arbeitende 3D-Kamera Kinect war ursprünglich für die Gestensteuerung für die Xbox360 entwickelt worden, um Spiele ohne Controller und stattdessen nur mit Armen und Beinen zu steuern. Als Microsoft 2010 die Kinect für nur 149 Euro auf den Markt brachte, fand sich sehr schnell eine Community im Netz, die das Produkt „hackte" und einen Open-Source-Treiber für Linux entwickelte. Was war der Grund?

Die Kinect war die erste 3D-Kamera (manchmal auch Tiefenkamera genannt) mit einem so bisher nicht dagewesenen Preis-/Leistungsverhältnis. Sie liefert nämlich nicht nur ein normales Kamera-Livebild, sondern gleichzeitig auch Entfernungsinformationen im dreidimensionalen Raum. Ein Infrarotlaserstrahl projiziert auf die Szenerie vor ihm im Raum ein bestimmtes Punktmuster, auch Structured Light genannt, und nimmt mit einer Kamera das entstehende Bild auf. Aus

Infrarotsensor GP2Y0A21YK0F von Sharp



Ultraschallsensor SRF05 von Devantech



Die 3D- und 2D-Sicht der Kinect

Ein klassischer LIDAR zur Entfernungsmessung. Eine Optik dient dem Senden, die andere dem Empfang des Laserstrahls.

der Abweichung, die sich im Vergleich zur Projektion auf einer (gedachten) planen Leinwand ergibt, lässt sich die Entfernung der einzelnen Punkte berechnen. Die Kinect gibt dies als Bild mit Entfernungsinformation für jede Koordinate aus.

Leider wird die Kamera seit 2017 so nicht mehr hergestellt. Eine vollständig überarbeitete und verkleinerte Version 4 der Kinect soll in Kürze von Microsoft unter dem Namen „Kinect for Azure" ausgeliefert werden – vorerst aber nur in China und in den USA. Alternativ gibt es 3D-Kameras von Intel, die mit der RealSense-Produktlinie weiterhin auf Gestensteuerung für PCs und Laptops setzen.

Ein Nachteil der alten Kinect und der RealSense im Einsatz auf einem Roboter ist, dass die Kameras immer noch keine Rundumsicht ermöglichen. Der Aufnahmewinkel, also das Sehen von links bis rechts, ist immer noch eingeschränkt. Auch ist die Genauigkeit vergleichsweise schlecht. Und hier kommen nun die Laserscanner ins Spiel.

Laserscanner

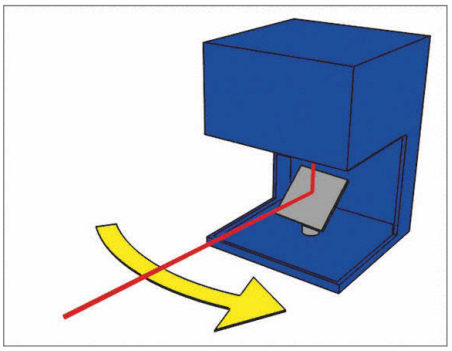
Prinzipiell arbeiten Laserscanner mit einer Laserdiode und einem Empfänger. Um die Entfernung zu einem vor ihm liegenden Objekt zu messen, schickt der Laser einen Lichtimpuls aus und ein Prozessor misst die Zeit, bis er den reflektierten Impuls wieder empfängt (Time of Flight, ToF). Anhand der Zeit und der bekannten Lichtgeschwindigkeit berechnet der Prozessor nun die Entfernung. Man spricht bei diesem Verfahren auch von LIDAR (light detection and ranging). Verglichen mit den zuvor genannten Infrarot- und Ultraschallsensoren sind Laser wesentlich präziser. Die meisten Lasermodule können über den Dopplereffekt auch die Geschwindigkeit der Objekte berechnen, von denen der Lichtstrahl reflektiert wird.

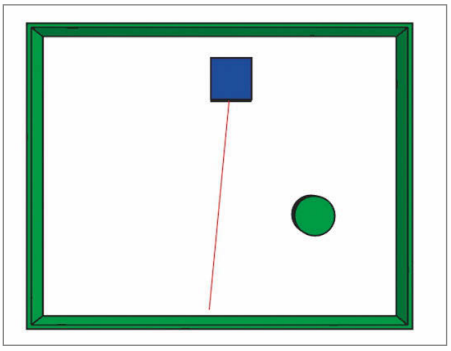
Indem man nun den Laserstrahl nicht nur starr nach vorne schauen lässt, sondern kontinuierlich hin- und herschwenkt, erhält man einen Laserscanner.

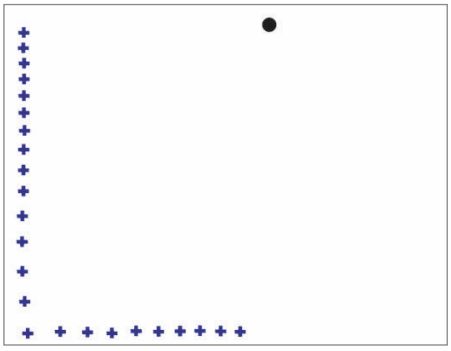
Rundumsicht

Laserscanner bieten einen wesentlichen Vorteil: Sie haben wie ein Leuchtturm eine maximale 360-Grad-Sicht und erkennen dank ihres vergleichsweise dünnen (unsichtbaren) Infrarotlaserstrahls auch kleinere Hindernisse im Zentimeterbereich. Die Bewegung des Laserstrahls kann auf zwei Wegen erfolgen. Ein rotierender Spiegel vor dem Laser lenkt den Strahl ab und fängt auch den zurückkommenden Strahl wieder ein, um ihn auf den Empfänger zu lenken. Das ist jedoch optisch vergleichsweise aufwendig. Einfacher und günstiger geht es, indem man einfach das komplette LIDAR-Modul per Motor dreht.

Damit dabei keine Kabel im Weg sind, sind das sich drehende Modul und die feste Basis per elektrische Schleifringe gekoppelt, über die Strom und Daten übertragen werden. Mitunter geschieht die Datenübertragung auch optisch. Die Messdaten für jeden Winkel wer-







Funktionsprinzip eines Laserscanners mit Ablenkspiegel

Ein SICKS300 (rechts) und sein Vorgängergerät (links)

SICK-S300-Spezifikationen

Anschlussspannung	24V (16,8–30V), DC
Schnittstelle	RS-422 oder RS-232 über das Wartungsinterface
Messbereich	30–30.000mm
Scanwinkel	270°
Auflösung	0,5°
Genauigkeit	30mm
Maße	106 × 102 × 162mm
Gewicht	1,2kg
Sonstiges	Robuste und langlebige Industrie-Hardware

den üblicherweise im Sensor verarbeitet und als Entfernungen über eine einfache serielle Schnittstelle (RS-232/COM-Port) oder per USB-Port ausgeliefert.

Es gibt mittlerweile eine Vielzahl dieser Scanner auf dem Markt, daher wird hier exemplarisch nur auf vier verschiedene eingegangen. Alle im Folgenden erwähnten Datenblätter, Softwarepakete und Beispielprojekte haben wir unter unserem Link in der Kurzinfo bereitgestellt.

SICK

Im industriellen Bereich werden bereits sehr lange Laserscanner des deutschen Herstellers SICK eingesetzt – allerdings anders, als man vielleicht zunächst vermuten würde. Wer schon einmal gesehen hat, wie Roboter in der Automobil-Produktion vollautomatisch eine Karosserie zusammenschweißen, der bekommt schnell einen Eindruck davon, dass hier Arbeitssicherheit sehr wichtig ist: Einfache Roboter nehmen Menschen in ihrer Umgebung nicht wahr. Aus diesem Grund hat man diese Bereiche früher mittels einfacher Lichtschranken abgesichert. Sobald ein Mensch diesen Arbeitsbereich betritt und dabei eine Lichtschranke durchkreuzt, stoppen die Roboter ihre Arbeit.

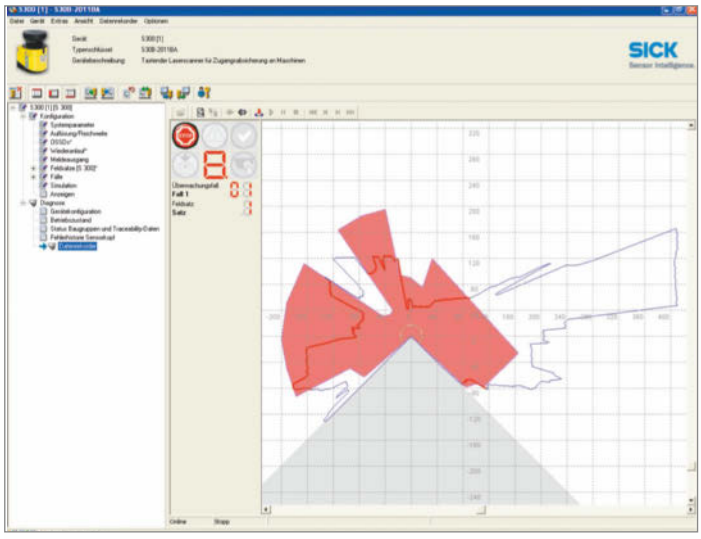
Anstatt nun viele individuelle Lichtschranken zu montieren, die bei jeder Änderung einer Produktionsstraße vielleicht auch noch umgebaut werden müssen, kommt hier ein Laserscanner zum Einsatz. Dieser scannt eine ganze Zone und kann per Software an einem Computer per serielle Schnittstelle so programmiert werden, dass er nur anspricht, wenn bestimmte Bereiche betreten werden.

Für einige SICK-Laserscanner gibt es auch Open-Source-Bibliotheken, um deren Daten per serielle Schnittstelle auszulesen und auf einem selbst gebauten Roboter einzusetzen. Beispielhaft sei hier auf den Roboter direcs1 des Autors hingewiesen, der die Daten eines SICK-Lasers S300 Standard (genauer der S30B-2011BA) auslas und zur Navigation nutzte.

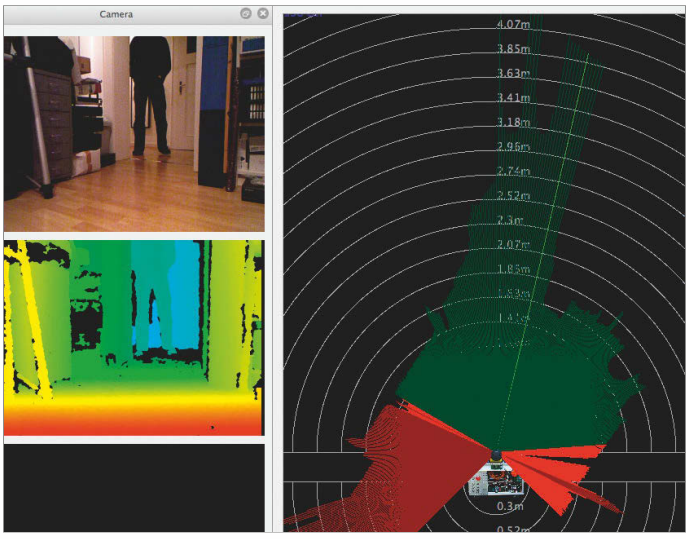
Um eine Idee zu bekommen, was der Laser genau „sieht“, ist in der Abbildung unten der Blick des Laserscanners (und einer Kinect-Kamera) auf eine offene Tür mit einem Menschen zu sehen. In den hier grün dargestellten Laserlinien sieht man deutlich die „Lücken“, wo die Beine des Menschen erkannt werden. Unterhalb des normalen Kamerabildes ist hier auch noch das 3D-Bild der Kinect zu erkennen.

Der Vorteil der Industrie-Laser ist, dass sie äußerst robust und für den harten und lang anhaltenden Industrieeinsatz gebaut werden – das Datenblatt spricht hier von 20 Jahren(!) Lebensdauer (EN ISO 13849). Dementsprechend haben diese Sensoren für den Einsatz im Hobbybereich einen gravierenden Nachteil: Sie sind alles andere als preiswert. Der hier gezeigte Laserscanner S300 hatte seinerzeit einen Neupreis von ca. 3.000 Euro und wurde 2009 gebraucht bei eBay für 957 Euro verhältnismäßig günstig, weil neuwertig, erworben.

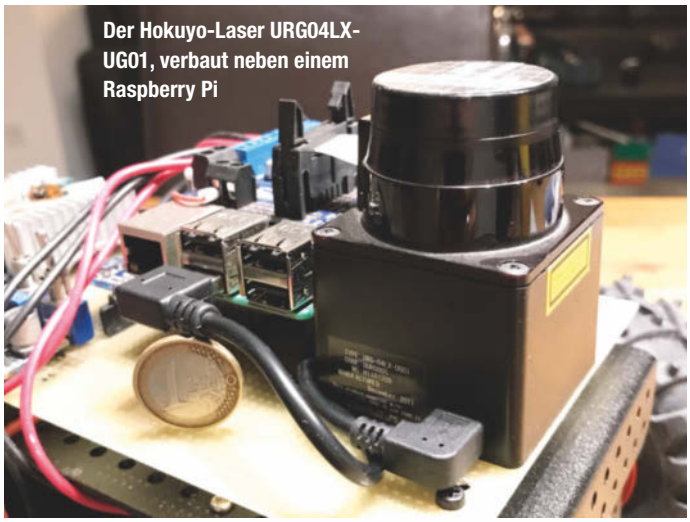
Wer bereits ROS (Robot Operating System) nutzt, wird sich freuen: Um die Daten dieses Laserscanners auszulesen, bedarf es lediglich der Installation des Paketes cob_sick_s300.



Die Konfigurationssoftware für SICK-Laserscanner im Einsatz (Windows)



So sehen die Kinect (links) und der SICK-Laserscanner (rechts) ihre Umwelt.



Der Hokuyo-Laser URG04LX-UG01, verbaut neben einem Raspberry Pi

Hokuyo-URG04LX-UG01-Spezifikationen

Anschlussspannung	5V, DC (USB-Bus)
Schnittstelle	USB 2.0
Scanzeit	100ms pro Scan
Messbereich	20–4000mm
Scanwinkel	240°
Auflösung	0,36°
Genauigkeit	60–1000mm: +/–30mm. Ab 1000mm: +/– 3% des Messwertes
Maße	50 × 50 × 70mm
Gewicht	160g
Sonstiges	Leichte Bauweise, Stromversorgung und Daten via USB (2.0)

Hokuyo

Wer damit leben kann, dass ein Laserscanner eine etwas geringere Lebensdauer hat und zudem nicht in fordernden industriellen Umgebungen eingesetzt werden soll, der wird beim japanischen Hersteller Hokuyo fündig.

Das derzeit preiswerteste Modell von Hokuyo ist der URG04LX-UG01. Er ist in Online-Shops für knapp 1000 Euro erhältlich. Das ist sicherlich nicht gerade ein Schnäppchen (und es gibt später im Artikel noch ein wesentlich günstigeres Gerät). Aber verglichen mit SICK ist der Hokuyo noch im niedrigeren bis mittleren Preissegment angesiedelt; und dabei noch sehr gut verarbeitet: Es gibt äußerlich keine beweglichen Teile, er ist sehr handlich und klein und wiegt dabei nur 160 Gramm. Ein weiterer Vorteil ist, dass man ihn ebenfalls sehr leicht in ROS einbinden kann (siehe Artikel Software auf Seite 34). Hierzu wird lediglich das Paket urg_node benötigt.

Wer die Daten in seiner eigenen Software auslesen will, findet auf der Website des Herstellers Hokuyo fertige Code-Beispiele (nach formloser Registrierung).

Apropos „geringere“ Lebensdauer: Der Hokuyo gibt sicherlich nicht nach einem halben Jahr den Geist auf, nur weil es kein SICK ist. Der Hersteller gibt die Lebensdauer mit 5 Jahren an – vermutlich, wenn der Scanner ununterbrochen durchlaufen würde. Das dürfte im Bastel- oder Robotikumfeld eher nicht die Regel sein. Geht man also nur von einem Einsatz von beispielsweise 4 Stunden am Tag an 5 Tagen die Woche aus, so kommt man rechnerisch bereits auf eine Lebensdauer von 42 Jahren.

RPLIDAR

Nun kommen wir endlich in den bezahlbaren Hobbybereich: Der RPLIDAR A1M8 von SLAMTEC ist ein 360-Grad-Laserscanner mit USB-Anschluss, der bereits für unter 100 Euro in Deutschland online erhältlich ist.

Er gewinnt vielleicht keinen Schönheitspreis und man muss in Kauf nehmen, dass er außen zugängliche, bewegliche Teile hat, die wie bei einem Plattenspieler über einen einfachen Riemen angetrieben werden. Dafür ist er aber aktuell unschlagbar günstig und mit 190g auch noch sehr leicht.

Wie steuert man den RPLIDAR an?

Für Bastler erfreulich: Der RPLIDAR lässt sich sowohl direkt über eine serielle sogenannte UART-Schnittstelle mit 5V-Pegel ansprechen als auch über einen (mitgelieferten!) Adapter per USB. Für die ROS-Profis gibt es auch hier ein fertiges Paket namens rplidar. An einem Raspberry Pi empfiehlt sich der Anschluss mittels Adapter an einen der USB-Ports. Hierbei darauf achten, dass der

Der sehr günstige Laserscanner RPLIDAR A1M8

RPLIDAR-A1M8-Spezifikationen

Anschlussspannung	5–10V Motorspannung, 5V für die Schnittstelle, DC
Schnittstelle	UART oder USB
Scannzeit	180ms pro Scan
Messbereich	15–6000mm
Scanwinkel	360°
Auflösung	1°
Genauigkeit	<1% des Messwertes
Maße	97 × 71 × 51mm
Gewicht	190g
Sonstiges	Bewegliche Teile sind zugänglich.

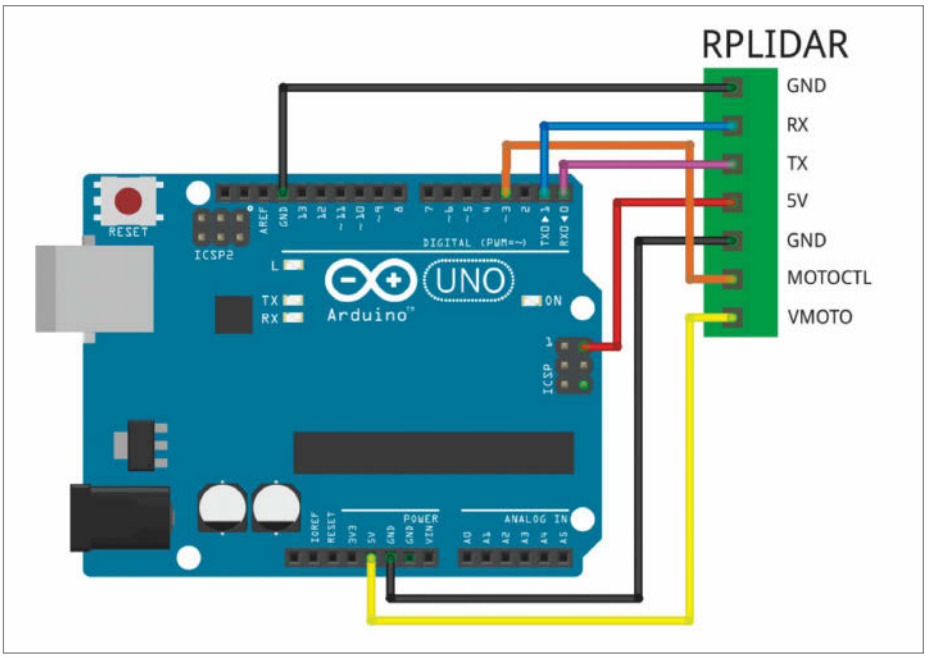
Adapter auch einwandfrei mit dem Laserscanner verbunden ist. Der Stecker scheint hier gelegentlich etwas locker zu sitzen.

Zum Anschluss an einen Arduino werden die Pins RX/TX/GND vom Laserscanner mit dem Arduino verbunden – wie üblich TX mit RX und RX mit TX sowie GND mit GND der jeweils anderen Platine. Der Pin MOTOCTL des RPLIDAR wird mit einem analogen Ausgang (PWM) des Arduino verbunden; er steuert später die Geschwindigkeit des Laserscanner-Motors. Bei einem Arduino Uno sähen diese Verbindungen wie in der Abbildung rechts aus.

Praktischerweise gibt es bereits eine fertige Arduino-Bibliothek. Diese kann bei GitHub heruntergeladen oder mit git geklont werden (siehe Link in den Kurzinfos) und in das Library-Verzeichnis im Arduino-Ordner kopiert werden; dabei sollte lediglich das Unterverzeichnis RPLidarDriver kopiert werden.

Wer mag, nennt es dabei noch nach RLIDAR um. Danach gibt es, wie in der Arduino-IDE üblich, unter dem Menüpunkt *Datei > Beispiele* einen neuen Eintrag *RPLidar > simple_connect* zum ersten Testen.

Dieses Beispiel stellt die generelle Verbindung zum Laserscanner her, startet den Motor und scannt dann die Umgebung. Als Ergebnis liefert es für den jeweiligen Winkel (angle) die gemessene Distanz (distance).



Direkter Anschluss des RPLIDAR A1M8 an einen Arduino Uno

Visualisieren

An der Stelle, wo im Source Code `//perform data processing here...` steht, kann nun eigener Code eingefügt werden, um die gelesenen Daten anzuzeigen oder auszuwerten. Hierfür könnte zum Beispiel eines von zahlreichen verfügbaren Displays mit fertigen Bibliotheken verwendet werden – eine gute Quelle sind hier Produkte von Adafruit. Apropos Anzeigen: Die rplidar_arduino-Bibliothek unterstützt leider nur serielle Hardware-Ports am Arduino und kein SoftSerial. Wer also über den Serial-Monitor der Arduino IDE sich auch Daten ausgeben lassen möchte, braucht einen Arduino mit mehr als einer seriellen Schnittstelle (z. B. Arduino M0/Zero, Mega etc.).

Aber es gibt auch noch ein zweites Beispiel in der Arduino IDE: distance_to_color. Hier wird an den Arduino eine RGB-LED angeschlossen. Diese ändert beim Scannen dann ihren Farbton (hue) in Abhängigkeit zur Richtung des dichtesten Objektes, welches der Scanner erkannt hat. Zusätzlich verändert sich die Helligkeit der LED zur Distanz des Objektes.

Basierend auf letzterem Beispiel hat ein Bastler auf instructables.com eine schöne Visualisierung mittels RGB-LED-Ring realisiert. Hierauf lässt sich gut aufbauen, wer den RPLIDAR mit einem Arduino einsetzen möchte.

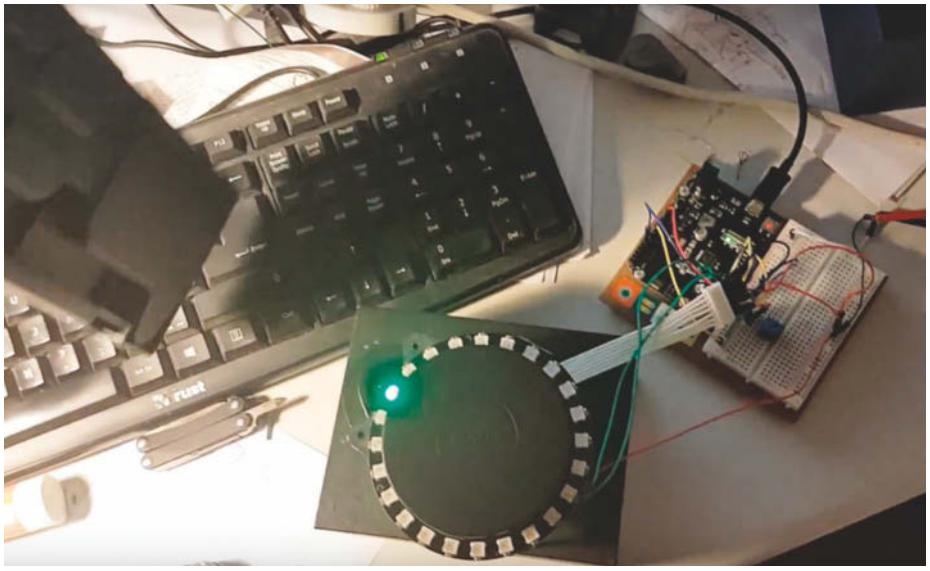


Bild: YouTube, „geobruce"

Der LED-Ring, platziert um den Laserscanner, signalisiert das dunkle Objekt links.

Ergänzend sollte noch erwähnt werden, dass die Arduino-Beispiele der Bibliothek rplidar_arduino auf dem SDK des Herstellers SLAMTEC basieren. Dieses findet sich bei GitHub. Hier gibt es auch weitere Infos, wie man zum Beispiel die Seriennummer oder den „Gesundheitsstatus" des Scanners auslesen kann.

macOS

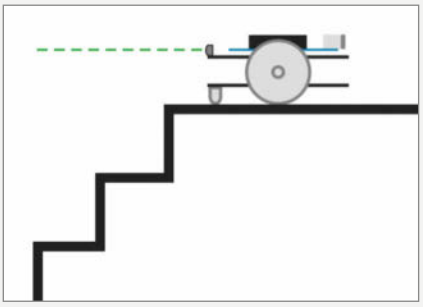
Wer auf dem Mac dem RPLIDAR seine Daten entlocken will, muss erst einen Treiber für den USB-Wandler-Chip CP2102 installieren.

Tipp

Falls der USB-Adapter partout nicht erkannt wird, einfach mal das USB-Kabel tauschen! Der Autor hat hier viel Zeit in Fehlersuche investiert, obwohl das zuvor verwendete Kabel mit anderen Geräten einwandfrei funktionierte – aber eben nicht mit dem Laserscanner beziehungsweise Adapter.

Wichtig!

Alle in diesem Artikel vorgestellten Sensoren können nicht durch feste Gegenstände durchsehen. Es ist also für den Scanner nicht erkennbar, ob ihn hinter einem Sessel noch eine Wand erwartet. Wichtig für Hindernisse wie z.B. Treppen: Ein fest installierter Laserscanner/Infrarotsensor/Ultraschallsensor scannt nur zweidimensional; er misst also genau nur auf der Höhe, wo das Signal aus dem Sensor austritt. Ein Roboter würde also eine Treppe, für die er „nach unten blicken müsste", nie erkennen!



Für eine „Treppenerkennung" reicht ein fest montierter Laserscanner o. ä. nicht aus!

Danach können die Beispiele aus dem zuvor erwähnten GitHub-Repository geklont und mit `make` erstellt werden. Der serielle Adapter identifiziert sich unter macOS im Dateisystem als /dev/tty.SLAB_USBtoUART.

Für den Mac gibt es ebenfalls zwei Beispiele im Sourcecode. Das erste lautet ultra_simple, das zweite ist der simple_grabber. Letzterer benötigt noch den USB-Port und die Baudrate als Argumente. Der Aufruf lautet im Terminal also

```
simple_grabber
/dev/tty.SLAB_USBtoUART 115200
```

Unter Windows kann man sich das jeweils aktuellste Release herunterladen; es enthält sogar fertige exe-Dateien. Im Bild unten sieht man sehr gut, wie für den RPLIDAR die gescannte Welt aussieht.

Raspberry Pi

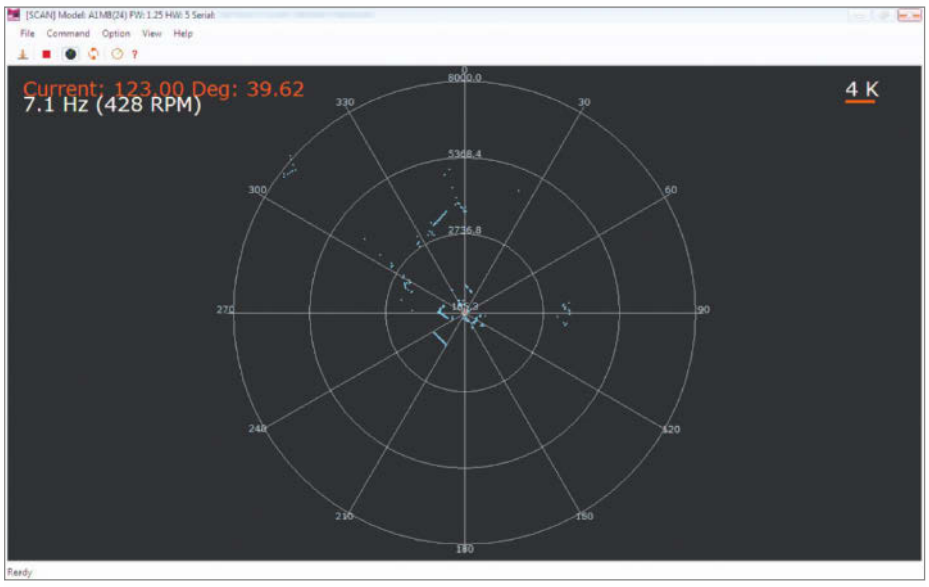
Für den Raspberry Pi hat mal wieder Adafruit ganze Arbeit geleistet und ein recht einfaches Modul für Python zur Verfügung gestellt, welches die Daten zusätzlich auf einem kleinen touchfähigen TFT für den Pi visualisiert. Die benötigten Pakete werden auf dem Pi wie folgt im Terminal installiert:

```
sudo apt-get install python-pip
python-pygame
```

Die Installation dauert ein paar Minuten. Danach wird das eigentliche Paket adafruit_rplidar für den Laserscanner installiert:

```
python3 -m pip install -no-deps
adafruit_rplidar
```

Der konkrete Source Code, der die Laserdaten ausliest und grafisch darstellt, ist zum Download im Adafruit-Manual verlinkt und soll daher hier nicht detailliert aufgeführt werden. Als Ergebnis sieht man ein Livebild der Scandaten auf einem kleinen TFT auf dem Raspberry Pi. Praktisch zur Visualisierung, und es macht auf einem selbst gebauten Roboter schwer was her.



RPLIDARs Sicht auf die Welt (unter Windows)

Eigene Projekte

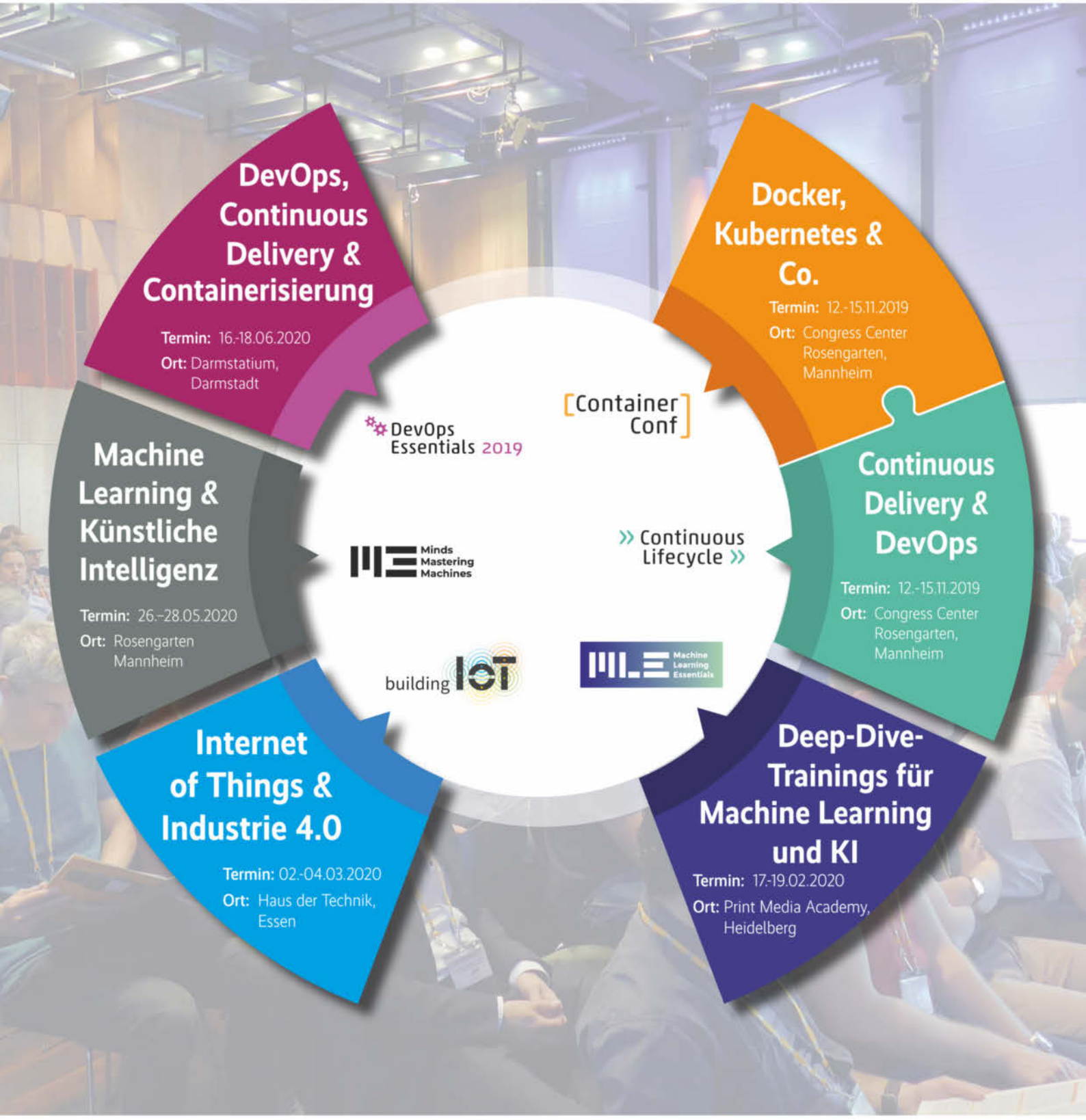
Wie setzt man nun einen Laserscanner in eigenen Projekten ein? Die Scanner liefern zu jedem Winkel eine Entfernung zurück. Im einfachsten Fall wertet man nur die Entfernung für Winkel 0 aus, also beispielsweise das, was vor dem Roboter liegt. Dann hat man aber im Grunde keine wesentlichen Vorteile gegenüber den günstigeren Lösungen mit Infrarot- oder Ultraschallscannern. Dank der 360-Grad-Umsicht kann der Roboter nun aber vor einem Ausweichmanöver entscheiden, in welche Richtung er lenkt. Nähert sich ein Hindernis, weicht er aus – und zwar in die Richtung, die laut den Daten des Scanners hindernisfrei ist.

Eine weitere Möglichkeit, die der Autor bei einem Roboter nutzte, wäre als erstes festzustellen, an welcher Stelle der Scandaten kein Hindernis in Entfernung X ist. Als Nächstes kann dann mittels Kosinussatz errechnet werden, ob dort eine ausreichende „Durchfahrbreite" für den Roboter vorhanden ist, und diese dann nutzen – es könnten ja schließlich mehrere Hindernisse erkannt worden sein.

Sobald man mehr als ein simples „Hindernis-rechts-links-Ausweichen" mit seinem Roboter umsetzen will, kommt man um das bereits erwähnte ROS eigentlich nicht herum. Dieses lässt sich sogar auf einem Raspberry Pi installieren. Der Autor hat hierzu die Distribution Ubuntu Mate genutzt. ROS bietet fertige Pakete für autonome Navigation und Kartenerstellung an, die aber für den eigenen Roboter und dessen Hardware angepasst werden müssen. In der Regel muss man bei üblicher Hardware aber nur wenige Zeilen oder Parameter ändern.

Der Nachteil von ROS ist ganz klar die nicht gerade flache Lernkurve. Deshalb soll der Fairness halber an dieser Stelle erwähnt werden, dass genau diese autonome Navigation mit Kartenerstellung der nächste Schritt ist, den der Autor mit seinem Roboter minibot mittels Laserscanner noch selbst absolvieren muss. Aber wie heißt es so schön als Maker: Der Weg ist das Ziel. *—dab*

DEVELOPER-KONFERENZEN + -WORKSHOPS 2019 / 2020



Veranstalter: iX, heise Developer, dpunkt.verlag

Weitere Informationen unter:

www.heise.de/developer/

Pi steuert EV3-Roboter

Objekterkennung mit einem Raspberry Pi ist eine Geduldsprobe und für die Robotersteuerung nur eingeschränkt zu gebrauchen. Mit der Unterstützung durch eine Edge TPU wird der Raspi jedoch erheblich fixer – schnell genug, um einem Lego-Roboter das Sortieren von Gegenständen beizubringen.

von Detlef Heinze

Sensoren spielen eine große Rolle für Roboter bei der Wahrnehmung ihrer Umwelt. Sie liefern grundlegende Daten für die Algorithmen, die ein Roboter nutzt, um sich in seiner Umwelt zurechtzufinden, sich zu bewegen und Objekte in seiner Umgebung zu manipulieren. Die wichtigsten sind neben Infrarot-, Ultraschall- und Tastsensoren auch Kameras.

In diesem Projekt verwenden wir die Pi-Kamera zusammen mit der Edge TPU und einem neuronalen Netz, um Objekte schnell maschinell zu erkennen und objektabhängig einen EV3-Roboter zu steuern. Mit Hilfe der Edge TPU kann der Raspberry Pi mit unserem Programm sechs Bilder pro Sekunde auswerten. Das ermöglicht, schnell auf Ereignisse zu reagieren, wie zum Beispiel das Hinstellen von bestimmten Objekten im Blickfeld der Kamera. Basis für dieses Projekt ist die Software der *SmartPiCam* (siehe Artikel „Objekterkennung mit Pi-Kamera und Edge-TPU“ in Make 3/19), die wir in diesem Artikel um die Steuerung eines EV3-Roboters erweitern.

Als Roboter verwenden wir den Gripp3r aus dem Lego-Mindstorms-EV3-Baukasten. Er besitzt eine Hand, mit der er Objekte greifen, transportieren und wieder loslassen kann. Eine Bauanleitung ist im Netz verfügbar (siehe Links zum Projekt). Wir nennen das Projekt *RoboPiCam*.

Kurzinfo

- **Offline-Smartcam als Roboterauge**
- **Raspberry Pi steuert Lego EV3-Roboter über Bluetooth**
- **Programmierung mit Python und Lego Mindstorms**

Checkliste

Zeitaufwand:
ab etwa 6 Stunden

Kosten:
etwa 130 Euro
(ohne Lego-Baukasten)

Programmieren:
Grundkenntnisse in Python und Linux

Alles zum Artikel im Web unter
make-magazin.de/xbxr

Material

- **Raspberry Pi 2/3** Model B/B+
- **MikroSD-Speicherkarte** mit installiertem Raspbian Stretch (Python 3.5.3) oder Raspbian Buster (Python 3.7.3)
- **Edge TPU: Coral USB Accelerator**
  Mouser (212-842776110077)
- **Raspberry Pi Kamera V2**
- **Lego Mindstorms EV3 Baukasten**

Mehr zum Thema

- Felix Pfeifer, Raspberry Pi einrichten (online)
- Daniel Bachfeld, KI-Komponenten für Maker, Make 6/18, S. 42
- Detlef Heinze, Objekterkennung mit Pi-Kamera und Edge-TPU, Make 3/19, S. 58
- Detlef Heinze, KI für Lego-Roboter, Make 6/18, S. 48

Aufgabe

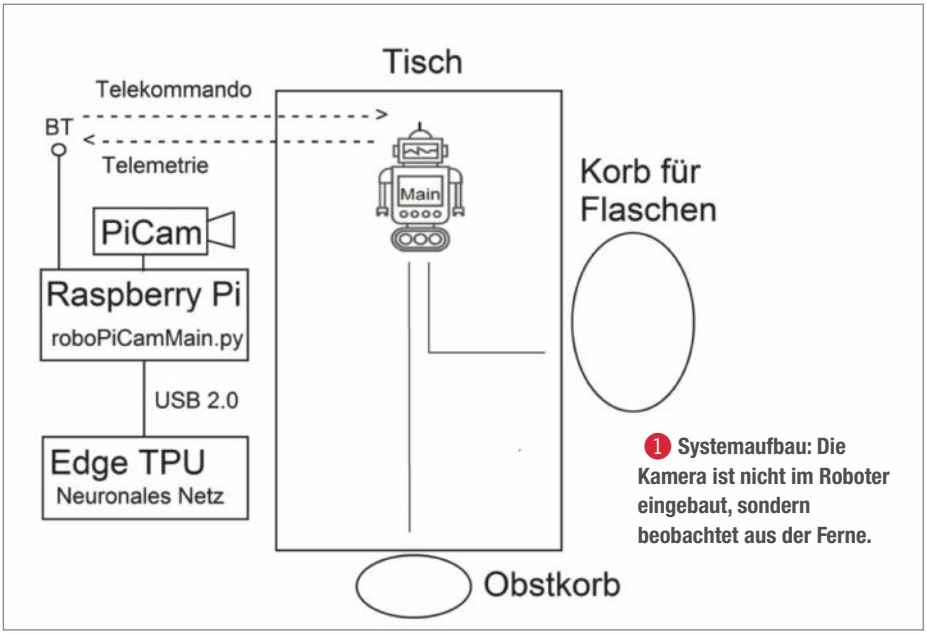
Zusammen mit der *SmartPiCam* soll der Roboter folgende Aufgabe lösen: Ein Benutzer stellt entweder eine Flasche oder legt einen Apfel vor die Greifhand des Roboters. Im Falle eines Apfels soll der Roboter den Apfel in einen Obstkorb bringen. Stellt der Benutzer aber eine Flasche hin, soll der Roboter sie in einen Einkaufskorb legen.

Die Aufgabe der *SmartPiCam* ist es dabei, Objekte vor der Greifhand des Roboters mit Hilfe eines neuronalen Netzes zu erkennen und dem Roboter einen entsprechenden Auftrag zu erteilen. Der Roboter nimmt lediglich den Auftrag an und bringt dann selbstständig, ohne weitere Befehle von der *SmartPiCam* zu erhalten, den Apfel oder die Flasche zum richtigen Korb. Während der Roboter arbeitet, wartet die *SmartPiCam* auf die Erledigung der Aufgabe. In dieser Zeit macht sie keine weiteren Fotos. Erst nach getaner Arbeit, wenn der Roboter wieder auf neue Aufträge wartet, beginnt die *SmartPiCam* wieder, Fotos zu schießen und auszuwerten.

Das GitHub-Repository zum Projekt (über die Kurz-URL in der Kurzinfo zu finden) enthält ein Video, das den obigen Ablauf gleichzeitig aus zwei Perspektiven zeigt: Die eine zeigt die Aufnahmen der Pi-Kamera mit den erkannten Objekten. Die andere Perspektive einer zweiten Kamera gibt einen Überblick über den gesamten Aufbau mit dem arbeitenden Roboter. Beide Perspektiven sind im Video zeitlich synchronisiert.

Systemaufbau

Der Aufbau unseres kleinen Systems 1 besteht aus der *SmartPiCam* mit dem Raspberry Pi, seiner Kamera und dem Coral USB Accelerator, den wir hier verkürzt mit seiner technischen Basis *Edge TPU* bezeichnen. Letzterer enthält das neuronale Netz für die Objekterkennung. Die *SmartPiCam* ist nicht auf dem Roboter montiert, sondern beobachtet von Ferne von einem festen Ort aus die Fläche auf einem Tisch vor dem Roboter. Dazu führt sie das Programm `roboPiCamMain.py` aus.

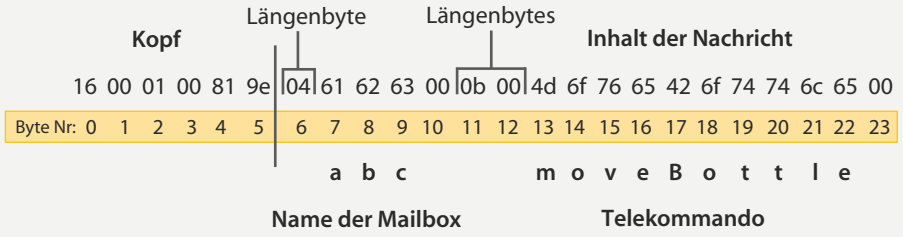


1 **Systemaufbau: Die Kamera ist nicht im Roboter eingebaut, sondern beobachtet aus der Ferne.**

Mailbox

2

Der Aufbau des Telekommandos „MoveBottle“ als kodierte EV3-Mailbox-Nachricht



Der Roboter selbst steht am Rande des Tisches und führt ebenfalls ein Programm aus (`Main`). Es erwartet einen Auftrag der SmartPiCam. Den erhält der Roboter über Bluetooth in Form eines Telekommandos, sobald ein Objekt erkannt wurde. Der Roboter quittiert die Ankunft eines Telekommandos sowie dessen Erledigung mit der Telemetrie.

Telekommandos und Telemetrie sind in diesem Projekt kurze Nachrichten, die die Zusammenarbeit beider Geräte koordinieren und auch der Fehlererkennung dienen. Es sind kurze Textnachrichten und im Falle der Telemetrie zusätzlich boolesche Nachrichten (`True` oder `False`).

Für die Realisierung des Systemaufbaus enthält das GitHub-Repository für die *SmartPiCam* und den Roboter jeweils eine Installationsanleitung sowie alle benötigten Quellen. Im Folgenden schauen wir uns nun die einzelnen Elemente des Systems näher an.

Das neuronale Netz

Unser neuronale Netz verwendet ein bereits trainiertes Modell, das für die Edge TPU erzeugt wurde. Es ist im GitHub-Repository enthalten und wurde bei Google mit dem COCO-Bilddatensatz (**C**ommon **O**bjects in **Co**ntext) trainiert. Dieser Bilddatensatz enthält 90 Objekte des täglichen Lebens wie Äpfel und Flaschen, Personen, Autos, Hunde sowie weitere Objekte. Diese Objekte kann die *SmartPiCam* somit prinzipiell erkennen.

Das trainierte Modell legt den Aufbau (Architektur) und die Initialisierung des neuronalen Netzes mit den maschinell gelernten Parametern fest. Die Architektur heißt in unserem Fall *MobileNet SSD v2* und ist eine häufig verwendete Architektur für die Objekterkennung. Das Kürzel *SSD* im Namen bezeichnet den Algorithmus ***S**ingle **S**hot Multibox **D**etector*, der bei diesem neuronalen Netz zur Anwendung kommt. Er kann in nur einem Schritt bei der Analyse eines Fotos auch mehrere unterschiedliche Objekte darin erkennen und deren Ort im Foto durch ein umgebendes Rechteck angeben.

Das Modell erwartet Farbfotos in einer Auflösung von 300 Pixel × 300 Pixel. Wir laden das Modell zum Start des Programms auf dem Raspberry Pi über USB in die Edge TPU. Dort läuft die Objekterkennung dann lokal. Aus diesem Grund benötigt das Programm auch keinen Zugriff in die Cloud auf KI-Dienste, da alle notwendigen Daten für das Inferencing, also die Klassifizierung von Objekten, lokal vorhanden sind.

Die Edge TPU basiert auf dem weit verbreiteten Framework *TensorFlow* von Google für das maschinelle Lernen. Sie benutzt hier die spezielle Variante *TensorFlow Lite* für kleine eingebettete Systeme. Auch unser Modell wurde für diese Variante erzeugt. Da die Edge TPU die notwendigen Rechenschritte für das Inferencing mit Hardwarekomponenten ausführt, ist sie sehr schnell bei gleichzeitig sehr geringem Stromverbrauch.

Bluetooth-Schnittstelle

EV3-Steine können untereinander Nachrichten über Bluetooth austauschen. Diese Fähigkeit nutzen wir, um die *Telekommandos* und die *Telemetrie* zwischen dem Raspberry Pi und dem EV3-Roboter zu übertragen. Dazu realisieren wir folgenden einfach gehaltenen Kommunikationsablauf.

Die Kommunikation initiiert in unserem Projekt immer das Programm auf dem Raspberry Pi. Der EV3-Roboter reagiert entsprechend. Beide haben ein Master-Slave-Verhältnis. Der Raspberry Pi (Master) beginnt damit, ein Telekommando zu senden, zum Beispiel, weil er eine Flasche erkannt hat. Der EV3-Roboter (Slave) muss innerhalb von zwei Sekunden den Erhalt des Telekommandos mit einer *Telemetrie* quittieren. Das ist eine Kurznachricht mit dem booleschen Wert `True`.

Quittiert der EV3-Roboter den Eingang der Nachricht nicht oder nicht rechtzeitig, geht der Raspberry Pi von einer Fehlersituation aus, meldet diese und das Programm beendet sich. Andernfalls führt der EV3-Roboter das gesendete *Telekommando* aus. Dazu gibt der Raspberry Pi ebenfalls einen Zeitraum vor, in dem das Kommando bearbeitet sein muss. Wird die Zeit nicht eingehalten, geht der Raspberry Pi wieder von einer Fehlersituation aus, und das Programm auf dem Raspberry Pi terminiert. Im positiven Fall sendet der EV3-Roboter nach Ausführung des *Telekommandos* eine Textnachricht als Fertigmeldung (*Telemetrie*).

Folgende *Telekommandos* und *Telemetrie* verwendet das Programm *roboPiCamMain.py*. Das *Telekommando* `Heartbeat` prüft zum Start des Programms auf dem Raspberry Pi, ob eine Bluetooth-Verbindung besteht und das Programm auf dem EV3-Roboter läuft. Es benötigt keine Fertigmeldung, da der Roboter keine Aktion ausführen muss.

Telekommandos und *Telemetrie* sind einfache Strings, die als EV3-Mailbox-Nachrichten hier an die Standard-Mailbox `abc` übermittelt werden. Damit der Raspberry Daten mit einem EV3-Stein austauschen kann, muss er sich wie ein EV3-Stein bei der Kommunikation verhalten. Dazu müssen wir *Telekommandos* und *Telemetrie* gemäß dem EV3-Kommunikationsprotokoll als Mailbox-Nachricht kodieren und dekodieren und eine serielle Bluetooth-Schnittstelle aufbauen.

Die Kodierung einer EV3-Mailbox-Nachricht ist in der Dokumentation des EV3-Developer-Communication-Kit (siehe Links zum Artikel) im Detail aufgeführt. Bild 2 zeigt, wo der Name der Mailbox und das *Telekommando* im Falle von `MoveBottle` kodiert sind. Die anderen Bytes in der kodierten Nachricht geben folgende Informationen an: Länge der Nachricht minus 2 (Byte 0 und 1), Nachrichtenzähler (Byte 2 und 3), Befehlstyp (Byte 4), und Byte 5 `9e` bezeichnet den Befehl `Write`

Telekommandos (TC) und Telemetrie (TM)

Telekommando	Quittierung	Telemetrie	Fertigmeldung	max. Zeit
Heartbeat	ja	nein	entfällt	entfällt
MoveApple	ja	ja	MoveAppleok	19s
MoveBottle	ja	ja	MoveBottleok	19s

`Mailbox`. Die Bytes vor den beiden Strings geben ihre jeweiligen Längen an: Für den Mailboxnamen werden ein Längenbyte, für den darauf folgenden Nachrichtenstring zwei Byte benutzt. Die abschließenden Nullbytes werden jeweils mitgerechnet. Alle Bytes sind hexadezimal angegeben.

Die serielle Bluetooth-Schnittstelle bauen wir nach dem einmaligen Pairing der beiden Geräte nach jedem Booten mit Hilfe eines Bluetooth-Managers auf. Er muss zusätzlich auf dem Raspberry Pi installiert werden (siehe Installationsanleitung im GitHub-Repository). Die Kommunikation erfolgt dann über den seriellen Port */dev/rfcomm0* des Raspberry Pi. Dazu später mehr.

Programmierung des Pi

Die Programmierung des Raspberry Pi erfolgt mit Python in der Thonny IDE, die in der Raspbian-Installation (Stretch oder Buster) enthalten ist. Das Programm ist auf vier Dateien verteilt, damit es änderungsfreundlich und besser zu verstehen ist. Die Datei *roboPiCamMain* ist das Hauptprogramm der Anwendung und realisiert auch die Anzeige der Fotos auf dem Anwendungsfenster. Die Datei *roboPiCamContr* implementiert die Klasse `RoboPiCamContr`, die die Pi-Kamera und die Edge TPU initialisiert und verwendet. Darüber hinaus realisiert sie die Beauftragung des EV3-Roboters.

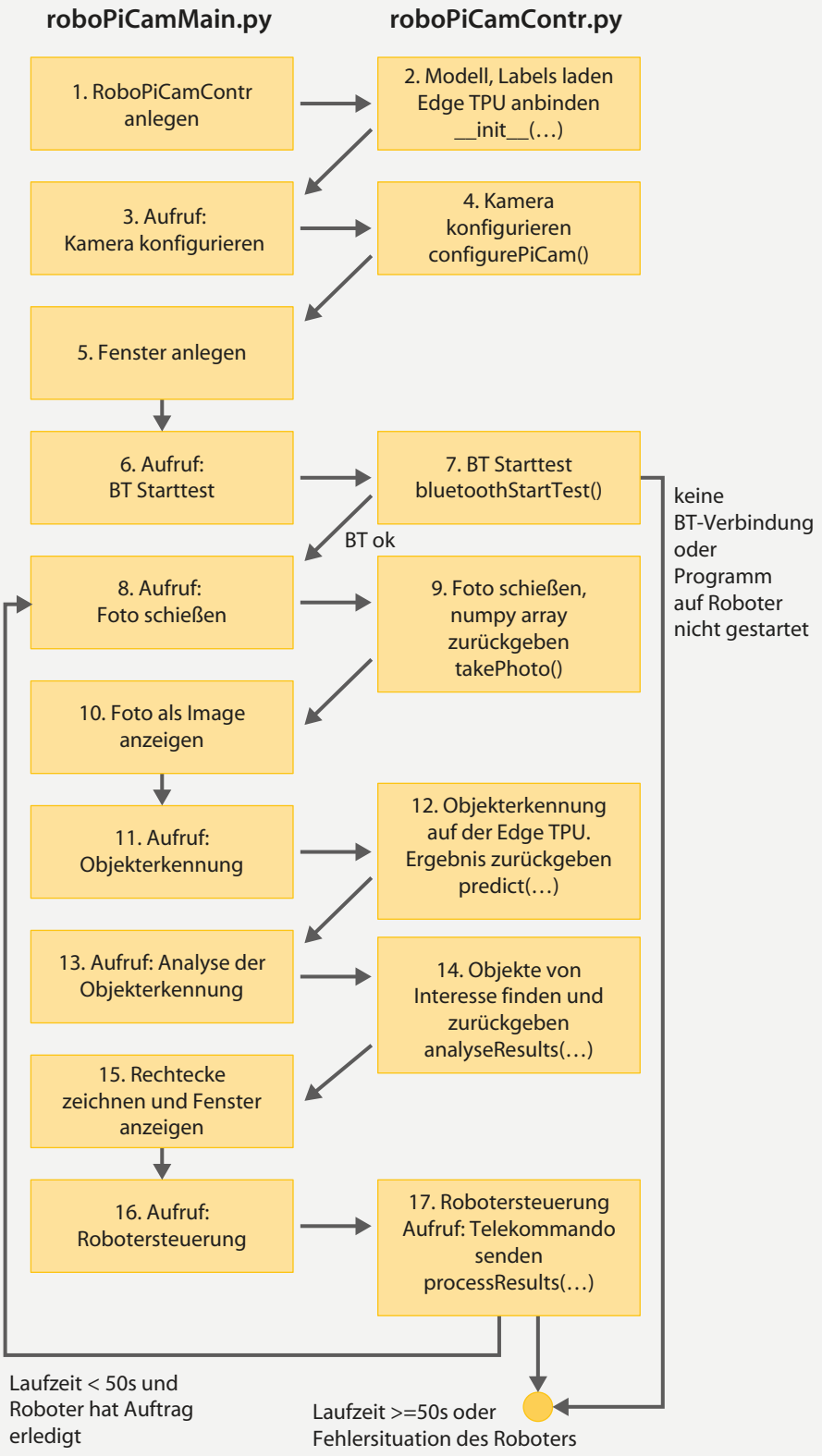
Die dritte Datei *TMTCpi2EV3* implementiert den oben geschilderten Kommunikationsablauf beim Senden eines Telekommandos und Empfangen der Telemetrie über die Bluetooth-Verbindung. Die Kodierung (2) und Dekodierung der Mailbox-Nachricht übernimmt die vierte Datei *EV3mailbox*.

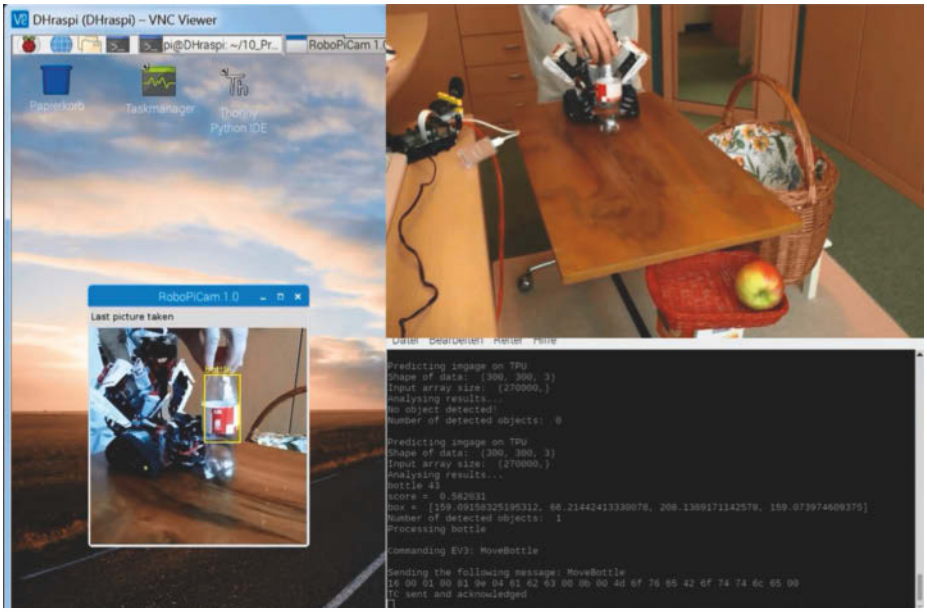
Das Flussdiagramm (3) der RoboPiCam gibt einen Überblick über den Algorithmus mit 17 Hauptschritten. Sie sind in den beiden Dateien *roboPiCamMain* und *roboPiCamContr* implementiert. Die entsprechenden Stellen im Code sind mit den Schrittnummern gekennzeichnet. Auf einen Abdruck der Listings haben wir aus Platzgründen verzichtet. Sie können sie online im GitHub-Repository einsehen und herunterladen.

Die ersten fünf Schritte des Algorithmus haben im Wesentlichen Initialisierungs- und Konfigurationsaufgaben für die Edge TPU, für die Kamera und für das Anwendungsfenster. Die Startphase des Programms wird mit Schritt 6 und Schritt 7 abgeschlossen – dem Bluetooth Starttest. Dabei versendet das Programm das oben erwähnte Telekommando `Heartbeat`. Quittiert der EV3-Roboter das Telekommando, kann die Hauptschleife des Programms ausgeführt werden. Im anderen Fall beendet sich das Hauptprogramm mit entsprechender Fehlermeldung, da der EV3-Roboter nicht erreichbar ist.

Funktionsweise der Roboter-Steuerung (3)

So kommunizieren die beiden auf dem Raspberry ausgeführten Python-Programme miteinander.





4 **Der Moment, in dem die smarte Pi-Kamera eine Flasche erkennt und den EV3-Roboter beauftragt.**

Die Schritte 8 bis 17 gehören zur Hauptschleife des Programms. Sie läuft mindestens 50s und wiederholt die Schritte Fotoaufnahme, Fotoanzeige, Übertragung der Fotodaten zur Edge TPU, Objekterkennung, Ergebnisübertragung, Analyse der Ergebnisse, Rechtecke und Label einzeichnen sowie Fenster aktualisieren. Für Details zur Implementierung der *SmartPiCam* von Schritt 8 bis 15 sei auf den Code und auf den Artikel „Objekterkennung mit Pi-Kamera und Edge-TPU" aus Make 3/19 verwiesen.

In Schritt 16 ruft das Hauptprogramm dann die Robotersteuerung in Schritt 17 mit der Funktion `processResults()` auf. Hat die *SmartPiCam* zum Beispiel eine Flasche erkannt 4, sendet diese Funktion ein Telekommando an den EV3-Roboter mit folgendem Befehl:

```
ack, reply= self.ev3.sendTC(
    'MoveBottle', True, 19)
```

In dieser Codezeile ist `ev3` ein Objekt der Klasse `TMTCpi2EV3` (siehe *TMTCpi2EV3*). Der erste Parameter von `sendTC()` ist das Telekommando an den Roboter. Der zweite gibt an, dass eine Telemetrie erwartet wird, und zwar innerhalb von 19s (dritter Parameter).

Die sendTC()-Funktion realisiert den oben im Abschnitt Bluetooth-Schnittstelle beschriebenen Kommunikationsablauf. Sie gibt einen Datensatz (Tupel) aus zwei Werten zurück: In die Variable `ack` wird der boolesche Wert `True` gespeichert, falls die Operation rechtzeitig ausgeführt wurde, und in `reply` gelangt der String, der sagt, welche Operation das war, im Beispiel `MoveBottleok`. Innerhalb dieser Funktion wird auch das Kodieren (und Dekodieren) der EV3-Nachrichten aus der Datei *EV3mailbox* aufgerufen.

Ist die Nachricht kodiert, sendet `sendTC()` sie mit folgendem Befehl an den EV3-Roboter über den seriellen Port */dev/rfcomm0* des Raspberry Pi. Dazu verwendet sie ein Objekt der Klasse `Serial`:

```
self.EV3serial.write(s)
```

Danach wartet `sendTC()` maximal zwei Sekunden auf die Quittierung des Telekommandos durch den Roboter. Dazu wiederholt sie die folgende Programmzeile alle 100ms, bis entweder die Quittierung eingetroffen ist oder die zwei Sekunden abgelaufen sind:

```
n = self.EV3serial.in_waiting
```

Der Rückgabewert `n` liefert beim Eintreffen der Nachricht die Anzahl der vom Roboter gesendeten Bytes. Diese können wir mit

```
s = self.EV3serial.read(n)
```

auslesen und danach dekodieren. Das LXTerminal 4 zeigt die Ausgaben bis zu dem Moment, in dem die Quittierung des EV3-Roboters empfangen und dekodiert wurde (Meldung: `TC sent and acknowledged`). Anschließend bringt der EV3-Roboter die Flasche in den Flaschenkorb, während `sendTC()` auf die Telemetrie auf die gleiche Weise maximal 19s wartet, damit der EV3-Roboter die Aufgabe erledigen kann.

Ist die *Telemetrie* gesendet (das *Telekommando* ist erfolgreich ausgeführt), wiederholt sich die Hauptschleife, wenn die Laufzeit von 50s noch nicht abgelaufen ist.

Roboter-Programmierung

Auf der Seite des EV3-Roboters ist die Programmierung einfacher. Sie befindet sich in der Datei *AB_Gripp3r_V1.ev3*. Mit dem Block `Bluetooth-Verbindung` nimmt das Programm zunächst eine Verbindung zum Raspberry Pi auf, der hier mit dem Namen `pi4robo` identifiziert wird. Wenn Sie diesen Namen verwenden wollen, müssen Sie ihn auf dem Raspberry Pi erst definieren. Dazu editieren Sie die Datei */etc/machine-info* mit:

```
sudo nano /etc/machine-info
```

und fügen die folgende Zeile ein:

```
PRETTY_HOSTNAME=pi4robo
```

Die Hauptschleife des Programms beginnt damit, auf ein Telekommando vom Raspberry in der Mailbox `abc` zu warten. Sobald eine Nachricht eingetroffen ist, sendet das Programm sofort eine boolesche Nachricht mit dem Wert `True` an den Raspberry Pi, der ja auf die Quittierung nur 2 Sekunden wartet.

Danach zeigt der EV3-Roboter das Telekommando auf dem Display inklusive Tonausgabe an. Mit dem Schalter-Block kommt es zur Auswahl und Ausführung des Telekommandos mit einem eigenen Block, der die Bewegung des EV3-Roboters realisiert (`MoveApple` oder `MoveBottle`). Im Anschluss sendet ein Nachrichtenblock die dazugehörige Telemetrie

`MoveAppleok` oder `MoveBottleok`, auf die der Raspberry Pi maximal 19s wartet.

Das Telekommando `Heartbeat` wird über den Standardfall des Schalter-Blocks bearbeitet (siehe ersten Reiter 5), der keinen Code enthält. Der letzte Reiter `Abort` implementiert den Abbruch der Schleife, indem er die Variable `abort` auf `True` setzt und die Telemetrie `Abort` sendet. Das Telekommando ist auf dem Raspberry nicht implementiert, da ein automatischer Abbruch des Programms durch den Raspberry zu wiederholtem Start des Programms auf dem EV3 führt. Es kann für Erweiterungen genutzt werden.

Tipps zur BT-Verbindung

Der Aufbau der Bluetooth-Verbindung beginnt mit einem einmaligen Pairing von Raspberry Pi und dem EV3-Roboter. Nach

jedem Einschalten beider Geräte muss noch eine serielle Bluetooth-Verbindung eingerichtet werden. Zu beiden Schritten beachten Sie bitte die Installationsanleitung im GitHub-Repository und die folgenden Tipps.

Schalten Sie für beide Schritte immer zuerst den Raspberry ein und melden sich an. Danach fahren Sie den Roboter hoch. Vor dem einmaligen Pairing und vor jedem Einrichten der Bluetooth-Verbindung klicken Sie auf den Menüeintrag `Make Discoverable` des Standard Bluetooth-Dialogs.

Beim Pairing muss der 4-stellige Code `1234` möglichst schnell eingegeben werden. Ist man zu langsam, schlägt das Pairing fehl. Hat das Pairing und danach auch die Einrichtung der seriellen Bluetooth-Verbindung geklappt, zeigt das Display des EV3-Roboters in der linken oberen Ecke ein Bluetooth-Symbol und die Zeichen <> dahinter an. Dann starten Sie das Programm *Main* auf dem EV3-Roboter und anschließend *roboPiCamMain* auf dem Raspberry Pi.

Erweiterungsmöglichkeiten

Eine einfache Erweiterung besteht darin, die Bewegungen des Roboters von `MoveApple` oder `MoveBottle` an andere örtliche Gegebenheiten anzupassen. Dazu verändern Sie die gleichnamigen eigenen Blöcke mit der Lego-Mindstorms-Software. Auf die gleiche Weise lassen sich Anpassungen für einen anderen Roboter, zum Beispiel mit einem anderen Greifmechanismus oder Fahrverhalten, realisieren. Falls der Roboter nun länger oder deutlich kürzer als 19s für die Aufgabe benötigt, muss auch die Zeitangabe des `sendTC()`-Aufrufs in Schritt 17 (`processResults()`) auf dem Raspberry Pi angepasst werden.

Soll der Roboter ganz andere Aufgaben erledigen, erweitern Sie die Software auf dem Roboter und dem Raspberry Pi durch neue Telekommandos und Telemetrie. Auch hier ist der Ort im Code auf dem Raspberry Pi die Funktion von Schritt 17 (`processResults()`). Auf der Seite des EV3-Roboters fügen Sie dann einen neuen Fall in den Schalter-Block nach dem Muster von `MoveApple` ein.

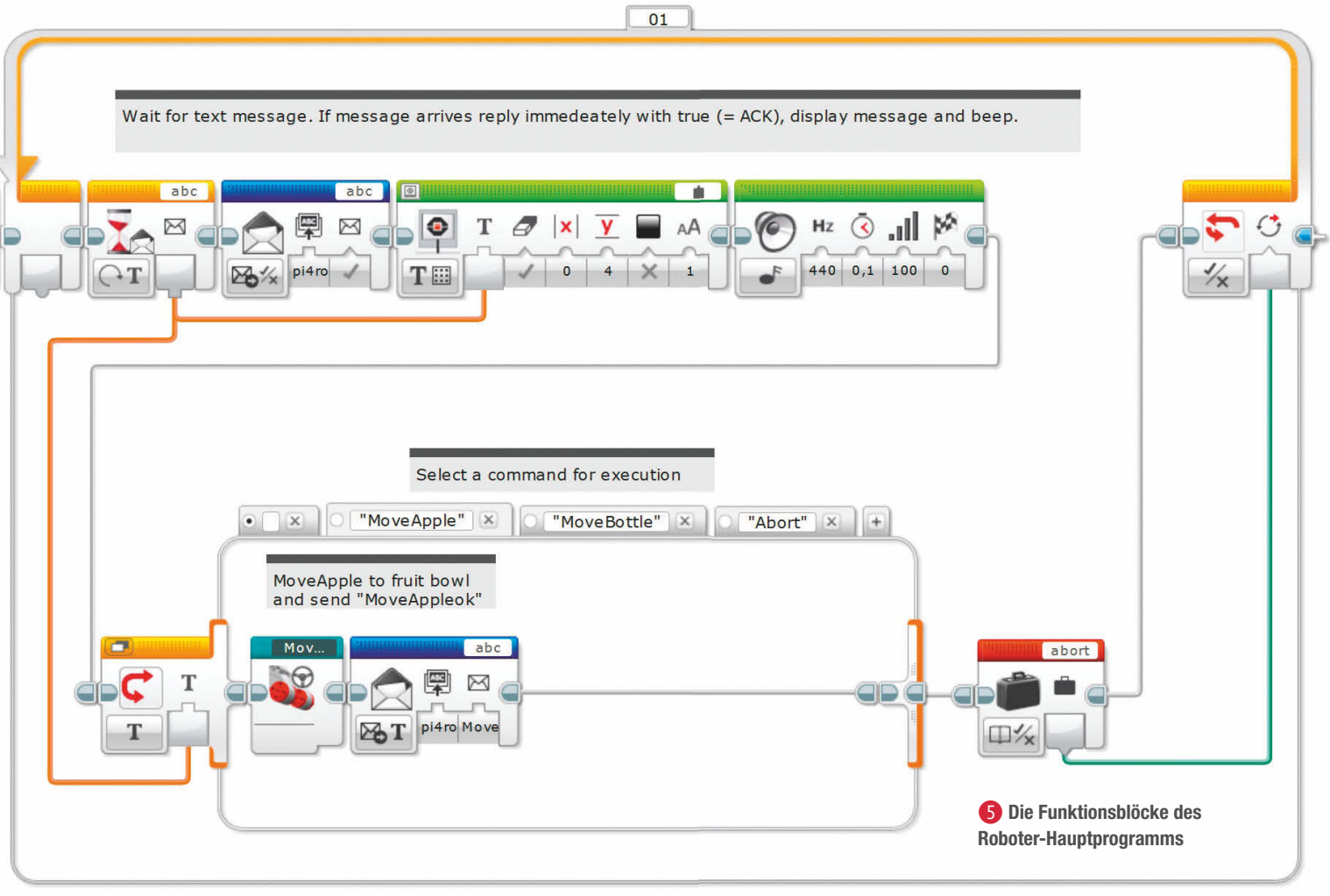
Die *SmartPiCam* kann aber auch andere Objekte als Äpfel und Flaschen erkennen. Das trainierte Modell des neuronalen Netzes kann 90 verschiedene Objekte unterscheiden. Sie sind in der Datei *coco_labels.txt* aufgelistet, die im GitHub-Repository enthalten ist. Soll die *SmartPiCam* andere Objekte erkennen, editieren Sie in der Datei *roboPiCamMain* die Zeile

```
objectIdsOfInterest = { 43, 52}
```

Sie definiert die Menge (Set) der erkennbaren Objekte durch deren Objektnummern, hier für Flasche und Apfel. Nachdem Sie die Zeile geändert haben, passen Sie wiederum die Funktion `processResults()` für die neuen Objekte an.

Schließlich ist es auch möglich, das Verhalten des Roboters vom Ort des erkannten Objekts auf den Fotos abhängig zu machen. Steht die Flasche zum Beispiel links im Bild, so könnte der Roboter mit einem neuen Telekommando und Telemetrie einen Schwenk fahren. Die Ortsinformation entnehmen Sie dem umgebenden Rechteck des Objektes auf dem Bild. Die Koordinaten des Rechtecks sind im Schritt 17 (`processResults()`) im Objekt `detectedObjects[0][1]` abgelegt.

Experimentieren Sie ein wenig mit Anpassungen und Erweiterungen an den Programmen und finden Sie neue Möglichkeiten, Ihren Roboter mit einer smarten offline Pi-Kamera zu steuern. —*hgb*



5 Die Funktionsblöcke des Roboter-Hauptprogramms

Krabbelroboter sendet Telemetrie

Dank des schlanken Protokolls MQTT können auch einfache Roboter Daten über das Netzwerk verschicken. Wir nutzen den Robotling, den wir in der Make 3/19 gebaut haben.

von Thomas Euler

Oft wünscht man sich die Möglichkeit, mit einem Mikrocontroller-Projekt drahtlos und ohne großen Aufwand Nachrichten austauschen zu können. Hier bietet sich das MQTT-Protokoll an, das speziell für die Maschine-zu-Maschine-Kommunikation entworfen wurde. Es kann daher leicht auf Mikrocontrollern mit Netzwerkzugang eingesetzt werden. Auf dem ESP32 wird es von MicroPython unterstützt.

Im Folgenden zeigen wir, wie man das Protokoll zunächst auf dem heimischen PC testet, und erläutern anschließend an Python-Code-Beispielen den Austausch von Nachrichten zwischen PC und ESP32. Zum Schluss werden wir Telemetriedaten vom „Robotling", unserem MicroPython-betriebenen Laufroboter aus Make 3/19, empfangen und visualisieren.

MQTT steht für „Message Queuing Telemetry Transport" und wurde 1999 von Andy Standford-Clark und Arlen Nipper für die einfache Kommunikation zwischen Maschinen konzeptioniert. Seitdem hat MQTT mehrere Überarbeitungen erfahren und ist heute ein weit verbreitetes, ISO-standardisiertes Nachrichtenprotokoll, das im Internet der Dinge eine wichtige Rolle spielt. Ganz frisch ist die Version 5.0, die aber noch nicht von allen MQTT-Distributionen unterstützt wird. Das Protokoll benutzt für die Datenübertragung das Netzwerkprotokoll TCP und funktioniert damit unabhängig von der Art der Netzwerkverbindung. Anstatt Nachrichten direkt zwischen zwei Teilnehmern (Clients) auszutauschen, benutzt MQTT einen zentralen Server. Dieser sogenannte Broker übernimmt die Verwaltung der Nachrichten, was die Implementation von MQTT auf der Seite des Clients relativ einfach macht. Dabei kann ein MQTT-Broker ein Server im Internet sein (z. B. test.mosquitto.org) oder, wenn die Kommunikation nur innerhalb des Heimnetzwerkes stattfinden soll, der lokale PC.

MQTT-Nachrichten sind hierarchisch in Themen (Topics) und Inhalte organisiert, ähnlich wie Ordner, Unterordner und Dateien auf der Festplatte. Die Information, dass der Antriebsmotor 1 eines Roboters ausgeschaltet ist, könnte man zum Beispiel mit dem Topic */mein_roboter/antrieb/motor1/* und dem Inhalt `0` kodieren – wobei nicht festgelegt ist, ob die Nachricht den aktuellen oder gewünschten Zustand („schalte Motor 1 ein") enthält. Dies muss der Anwender festlegen. Das Topic ist ein UTF-8-kodierter String von maximal 64KB Länge, während der Nachrichteninhalt ein bis zu 256MB großes Bytepaket („byte array") ist, dessen Struktur vom Anwender definiert wird (zum Beispiel ein einfacher String). Aus Effizienzgründen sind MQTT-Nachrichten in der Regel eher kurz.

Clients kommunizieren mit dem Broker, indem sie Nachrichten unter bestimmten Topics veröffentlichen („publish") oder abonnieren („subscribe"). Dabei kann ein Client auch beide Rollen gleichzeitig einnehmen. Der Broker kümmert sich darum, die Nachrichten entgegenzunehmen und an die Clients weiterzureichen, die das entsprechende Topic abonniert haben. Dank dieser Architektur kann im Prinzip jeder jedes Topic abonnieren und damit mitlesen. Dies kann man ausprobieren, indem man

```
.\mosquitto_sub -h
   test.mosquitto.org -t    "#"    -v
```

in die Kommandozeile eingibt (die Befehle werden meist in einer Zeile eingegeben, was wir jeweils mit Einrücken kennzeichnen). Für das Beispiel muss Mosquitto installiert sein; siehe dazu Kasten „Mosquitto testen".

Die daraufhin angezeigte Flut an Nachrichten erklärt sich dadurch, dass man mit dem Wildcard-Symbol „#" alle Topics und Untertopics abonniert, die dem Broker be-

Kurzinfo

- MQTT-Architektur verstehen
- MQTT einrichten
- Roboterdaten mit MQTT versenden

Checkliste

Zeitaufwand:
1–2 Stunden

Kosten:
100 bis 150 Euro (je nach Ausführung)

Programmieren:
Grundkenntnisse in Python

Alles zum Artikel im Web unter
make-magazin.de/xvxb

Material

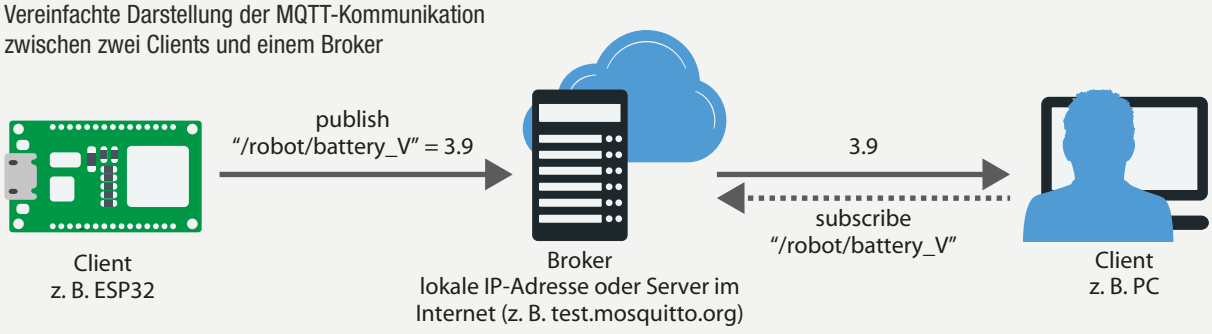
- ESP32-Modul
- PC
- Spinnenroboter „Robotling"

Mehr zum Thema

- Thomas Euler, Einstieg in MicroPython, Make 2/19, S. 104
- Thomas Euler, Krabbeltier mit Python-Antrieb, Make 3/19, S. 88

So funktioniert MQTT

Vereinfachte Darstellung der MQTT-Kommunikation zwischen zwei Clients und einem Broker



kannt sind. Durch die Angabe bestimmter Topics kann man so gezielt fremde Nachrichten mitschneiden. Dies ist kein Sicherheitsleck, sondern Teil des MQTT-Konzepts, möglichst einfachen, transparenten Nachrichtenaustausch zu ermöglichen. Ist man auf öffentliche Server angewiesen, sollte man erwägen, eine TLS-gesicherte Verbindung zum Broker aufzubauen, was MQTT unterstützt, und die Nachrichten zu verschlüsseln. Bei Letzterem hat der Anwender alle Freiheiten, da MQTT den Nachrichteninhalt selbst nur als Bytepaket betrachtet.

Obwohl MQTT verschlüsselte und gesicherte Kommunikation unterstützt, werden in den folgenden Beispielen die Daten der Einfachheit halber unverschlüsselt und ungesichert übertragen. Dessen sollte man sich bewusst sein, wenn man nicht innerhalb des privaten WLAN mit einem lokalen MQTT-Broker arbeitet, sondern öffentliche MQTT-Broker benutzt.

Neben der Übermittlung von Nachrichten bietet MQTT auch Mechanismen, um die Verlässlichkeit der Nachrichtenübermittlung zu beeinflussen („Quality of Service“, QoS). Der Standard QoS=0 bedeutet, dass jede Nachricht nur einmal verschickt wird. Der Broker stellt sie möglichen Abonnenten (Empfängern) zur Verfügung, übernimmt aber keine weiteren Maßnahmen, um den Empfang der Nachricht sicherzustellen. Der sendende Client (Sender) erfährt also nicht, ob die Nachricht angekommen ist. Bei QoS=1 speichert der Broker die Nachricht und bestätigt dem Sender den Empfang, sobald der Empfänger sie erhalten und dies bestätigt hat. Hier kann es vorkommen, dass eine Nachricht den Empfänger mehrfach erreicht.

Der meiste Aufwand wird bei QoS=2 betrieben: Hier vermittelt der Broker zwischen Sender und Empfänger, um sicherzustellen, dass der Empfänger die Nachricht genau ein einziges Mal erhält und der Sender den Empfang bestätigt bekommt. Welches QoS-Level benötigt wird, entscheidet der Anwender: Bei sich langsam ändernden Informationen (wie etwa dem Ladezustand einer Batterie) reicht QoS=0 mit dem geringsten Overhead, da es unbedeutend ist, wenn mal eine Nachricht verloren geht. Soll sichergestellt werden, dass der Empfänger handelt, weil zum Beispiel ein Motor in einer kritischen Situation abgestellt werden muss, wählt man eine höhere QoS-Stufe. Sowohl Clients wie auch Broker müssen dazu die gewählte Stufe unterstützen. Mit MicroPython ist derzeit maximal QoS=1 möglich.

Was MQTT so einfach für die Clients macht, ist, dass sie nur den Broker kennen und nichts über die möglichen Abonnenten wissen müssen. MQTT besitzt aber Mechanismen, mit Hilfe derer Clients etwas über den Zustand der Abonnenten (und umgekehrt) erfahren können. Gibt es etwa keine Abonnenten für ein Topic, verwirft der Broker die Nachricht. Ein Abonnent, der sich später einklinkt, verpasst frühere Ansagen. Für diesen Fall kann der Sender den Broker anweisen, eine Nachricht für zukünftige Abonnenten vorzuhalten („retain“). Auch kann ein Client, wenn er sich zum ersten Mal mit dem Broker verbindet, eine Nachricht hinterlegen („last will“), die der Broker verschickt, sobald die Verbindung mit dem Client abbricht. So erfahren Abonnenten, dass der Client wegen Verbindungsproblemen schweigt und nicht, weil er gerade nichts mitzuteilen hat.

Mosquitto testen

Die folgenden Anweisungen zur Installation von Client und Broker wurden unter Windows 10 getestet, sollten sich aber leicht auf andere Betriebssysteme übertragen lassen. Die Liste von MQTT-Distributionen und -Werkzeugen ist lang (siehe Link in der Kurzinfo). Voraussetzung ist eine Python-Distribution – in Make 2/19 haben wir die Installation von Anaconda erklärt.

Da wir später mit Python arbeiten wollen, verwenden wir mit *paho-mqtt* eine Python-Client-Bibliothek für MQTT des Open-Source-Projekts Eclipse Paho. Diese Bibliothek ermöglicht es, auf dem PC einen Client zu erstellen, der sich mit einem MQTT-Broker verbindet, um Nachrichten zu versenden und Nachrichten zu bestimmten abonnierten Topics zu empfangen. Das Paket installiert man von der Python-Kommandozeile aus mit:

```
pip install paho-mqtt
```

Mosquitto ist ein Open-Source-MQTT-Broker, der ebenfalls von der Eclipse Foundation gepflegt wird und den wir hier zum Testen von mosquitto.org/download herunterladen und auf dem lokalen PC installieren. Die Installation testet man am einfachsten, indem man drei Kommandozeilen-Fenster öffnet: eines für den MQTT-Broker und zwei für MQTT-Clients. In allen Fenstern muss man zunächst in das Installationsverzeichnis von Mosquitto wechseln:

```
cd "C:\Program Files\mosquitto\"
```

Im ersten Fenster startet man den Broker mit:

```
.\mosquitto.exe -v
```

Die Option `-v` („verbose“) sorgt dafür, dass der Broker über alle Ereignisse berichtet. Das hilft, wenn nötig, beim Debuggen. Im zweiten Fenster starten wir einen Client, der Nachrichten zu einem bestimmten Topic abonniert:

```
.\mosquitto_sub.exe -h
   192.168.4.150 -t robotling/test
```

Die Option `-h` gibt die IP-Adresse des Brokers an. In diesem Fall ist das die IPv4-Adresse unseres PC, die wie mit `ipconfig` ermittelt haben. Die Option `-t` gibt das zu abonnierende Topic an (hier *robotling/test*). Damit weiß der Broker, dass der Client auf Nachrichten zu diesem Topic wartet, und leitet sie entsprechend weiter. Im dritten Fenster erfolgt der Nachrichtenversand:

```
.\mosquitto_pub.exe -h
   192.168.4.150 -t robotling/
   test -m "123"
```

Dieser Befehl sendet die Nachricht `123` unter dem Topic *robotling/test* an den MQTT-Broker mit der IP-Adresse 192.168.4.150 (wieder unseren PC). Sobald der Befehl ausgeführt wird, erscheinen Protokolleinträge im Broker-Fenster und die Nachricht `123` im Client-Fenster.

MQTT mit MicroPython

MicroPython unterstützt das MQTT-Protokoll in einer etwas abgespeckten Form von Haus aus. Die folgenden Beispiele sollten mit jedem EPS32-Modul funktionieren, auf dem MicroPython läuft. Nur eine LED muss man eventuell nachrüsten. Wir verwenden das HUZZAH32-Board von Adafruit, auf dem ein aktuelles MicroPython installiert ist. Dazu benötigt man eine Test- und Entwicklungsumgebung auf dem PC (etwa Atom mit dem Plugin pymakr). In Make 2/19 haben wir die Installationen auf PC und Controller näher erläutert sowie den Umgang mit dem interaktiven Eingabefenster REPL (Read-eval-print loop). Zusätzlich nutzen wir nun den öffentlichen MQTT-Broker (mqtt.eclipse.org), aber ein lokaler Broker ginge auch (siehe Kasten). Außerdem müssen beide Beispielprogramme an das heimatliche WLAN angepasst werden, bevor sie auf den ESP32 geladen werden (siehe Anweisungen in den Listings).

Im ersten Beispielprogramm veröffentlich der Mikrocontroller einen „Countdown“. Dafür benötigen wir einen Client, der diese Nachrichten abonniert. Das funktioniert – wie im Kasten 1 gezeigt – am einfachsten über die Kommandozeile:

```
.\mosquitto_sub.exe -h
   mqtt.eclipse.org -t
   "b4e62da1dccd/countdown" -v
```

Gerade bei einem öffentlichen Broker ist es wichtig, nur die eigenen Topics zu abonnieren, denn bei sehr generischen Topics erhält man sonst eine Flut von Nachrichten. Daher stellen wir vor das Topic *countdown* eine GUID („globally unique identifier“), in diesem Fall die „Seriennummer“ des ESP32-Chips. Diese liest man mit den folgenden Anweisungen in der REPL als String aus:

```
import machine
import binascii
binascii.hexlify(machine.unique_id()
   ).decode()
```

Das Ergebnis ist ein String wie `b4e62da1dccd`, den wir in das oben genannte Kommando einfügen, bevor wir es ausführen. Nun wartet der Abonnent auf Nachrichten zum Topic *countdown*. Um das Beispielprogramm 1 auf dem ESP32 laufen zu lassen, lädt man es in den Atom-Editor, verbindet den ESP32 über USB mit der REPL des Atom-Plug-ins pymakr und startet es mit „run“. Sobald das Programm läuft, erscheinen die Ziffern 10 bis 1 als Nachrichten im Fenster des Abonnenten.

In Listing 1 werden anfangs einige Module importiert (Zeilen 1–5), darunter die Klasse `MQTTClient`, die die gesamte MQTT-Kommunikation übernimmt. Dann werden Netzwerkzugangsdaten, Broker und Topic definiert (7–11). Die Funktion `do_connect` (13–20) stellt eine WLAN-Verbindung her; sie wird im Hauptprogramm aufgerufen (24). Nachdem ein MQTTClient-Objekt erzeugt wurde (25), versucht das Programm eine Verbindung mit dem Broker herzustellen (28). Sobald das gelungen ist, zählt das Programm von 10 bis 1 und veröffentlicht im Sekundenabstand die Ziffern als Nachrichten (32–36). Dann endet es. Das `try-except`-Konstrukt (26–41) kümmert sich um eine rudimentäre Fehlerbehandlung, während das `try-finally`-Konstrukt (30-40) dafür sorgt, dass die Verbindung zum Broker in jedem Fall kontrolliert beendet wird.

Nachrichten mit dem ESP32 empfangen

Das zweite Programm (Listing 2) demonstriert, wie man auf einem Mikrocontroller mittels MQTT-Nachrichten Aktionen auslösen kann. In diesem Beispiel wartet der Mikrocontroller auf Nachrichten unter dem Topic *b4e62da1dccd/led*, wobei die Zeichenfolge vor dem Schrägstrich wieder durch die GUID des eigenen ESP32 ersetzt werden muss. Je nach Inhalt der Nachricht schaltet der Mikrocontroller eine LED an (`1`) oder aus (`0`). Hierzu startet man das Programm auf dem Mikrocontroller und schickt (genauer, publiziert) anschließend über ein Kommandozeilenfenster entsprechende Nachrichten, wobei der Inhalt (Option `-m`) `0` oder `1` sein kann:

```
.\mosquitto_pub.exe -h
   mqtt.eclipse.org -t
   "b4e62da1dccd/led" -m "1"
```

Die ersten Zeilen (1–20) entsprechen weitgehend denen vom ersten Listing. In den Zeilen 23–28 wird eine Funktion namens `on_message` definiert. Sie schaltet je nach Nachrichteninhalt (Argument `msg`) die rote LED auf dem HUZZAH-Board an oder aus. Dies passiert mittels eines Objekts der Klasse *Pin*, das in Zeile 37 als Ausgangspin an Anschluss #13 angelegt wird. `on_message` ist eine sogenannte „callback“-Funktion. Sie wird an das MQTTClient-Objekt übergeben (35) und anschließend automatisch immer dann aufgerufen, wenn eine Nachricht unter einem abonnierten Topic eintrifft. Damit MQTTClient seinen Job machen kann, muss regelmäßig dessen Funktion `check_msg` aufgerufen werden. Dies passiert in einer `while`-Schleife (44–48), die startet, sobald die Verbindung mit dem Broker steht und das relevante Topic abonniert wurde (40–41). Hierbei wartet `check_msg` nicht auf Nachrichten, sondern kehrt sofort wieder zurück, sodass in der Schleife auch andere Dinge erledigt werden können.

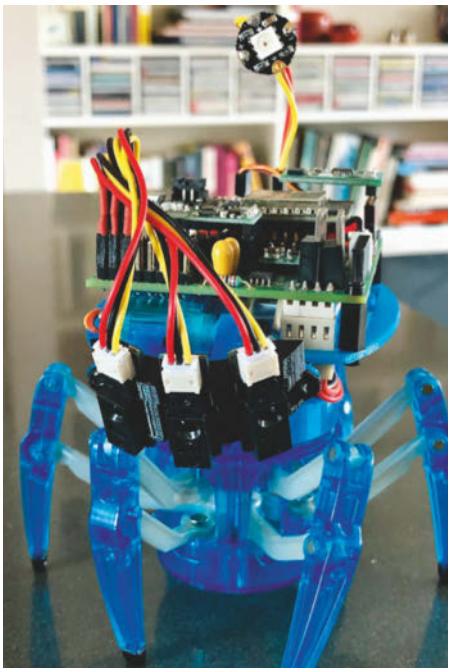
Robotling sendet Telemetrie

Da der Robotling mit einem WLAN-fähigen Mikrocontrollermodul (HUZZAH32) ausgestattet ist, kann man ihn relativ einfach dazu

Listing 1

```
 1 import network
 2 import time
 3 from machine import Pin, unique_id
 4 from binascii import hexlify
 5 from umqtt.robust import MQTTClient
 6
 7 WLAN_SSID   = "SSID"                # WLAN-Name
 8 WLAN_PWD    = "PASSWORT"            # WLAN-Passwort
 9 MQTT_BROKER = "mqtt.eclipse.org"    # IP-Adresse/URL des MQTT-Brokers
10 TOPIC       = b"countdown"          # Topic für Veröffentlichungen
11 CLIENT_ID   = hexlify(unique_id())  # eindeutige Modul-ID bestimmen
12
13 # Funktion, um eine WLAN-Verbindung herzustellen
14 def do_connect():
15   wlan = network.WLAN(network.STA_IF)
16   if not wlan.isconnected():
17     wlan.active(True)
18     wlan.connect(WLAN_SSID, WLAN_PWD)
19     while not wlan.isconnected():
20       pass
21
22 if __name__ == "__main__":
23   # Mit dem WLAN verbinden und einen MQTT-Client erzeugen
24   do_connect()
25   client = MQTTClient(CLIENT_ID, MQTT_BROKER)
26   try:
27     # Mit dem MQTT-Client verbinden
28     client.connect()
29     print("Connected to `{0}`".format(MQTT_BROKER))
30     try:
31       # Zahlen von 10 bis 1 unter festgelegtem Topic veröffentlichen
32       for i in range(10, 0, -1):
33         msg = str(i)
34         print("Publishing `{0}` under topic `{1}`".format(msg, TOPIC))
35         client.publish(CLIENT_ID +"/" +TOPIC, msg)
36         time.sleep(1.0)
37     finally:
38       # Verbindung zum MQTT-Broker beenden
39       client.disconnect()
40       print("Done.")
41   except OSError as err:
42     print("Error: Cannot connect to MQTT broker ({0})".format(err))
```

Der ESP32 publiziert MQTT-Nachrichten unter dem Topic *countdown*. Damit sich der Mikrocontroller mit dem heimischen WLAN verbindet, müssen die Strings `SSID` und `PASSWORT` angepasst werden. Anstatt eines öffentlichen Brokers wie mqtt.eclipse.org kann man auch die IP-Adresse des lokalen PC einsetzen, auf dem der Mosquitto-Broker läuft.



Ein „aufgebohrter“ Robotling mit drei Infrarot-Distanzsensoren

bringen, seinen aktuellen Zustand über MQTT bekannt zu geben. MicroPython unterstützt standardmäßig MQTT. Das entsprechende Modul *umqtt* bietet zwei verschiedene Client-Klassen an, „simple“ und „robust“, wobei der wesentliche Unterschied darin besteht, dass die „robuste“ Version automatisch versucht, eine unterbrochene Verbindung zum Broker wiederherzustellen.

Damit der Roboter Daten über MQTT veröffentlicht, muss man die aktuelle Version der Robotling-Software von Github beziehen (siehe Link in der Kurzinfo) und die Konfigurationsdatei *hexbug_config.py* anpassen. Unter anderem müssen die MQTT-Funktionen (`SEND_TELEMETRY = 1`) und WLAN aktiviert werden; Letzteres erreicht man, wenn man den String `wlan` zur Liste `MORE_DEVICES` hinzufügt. Außerdem benötigt der Roboter die Zugangsdaten zum lokalen WLAN (in *NETWORK.py*). Detaillierte Anweisungen finden sich im Robotling-Wiki auf Github.

Ist alles vorbereitet, kann man beim Programmstart der Robotling-Software (etwa im REPL-Fenster von Atom) anhand der roten Onboard-LED verfolgen, wie sich das HUZZAH32-Modul mit dem WLAN (schnelles Blinken der LED) und dann mit dem MQTT-Broker verbindet (LED leuchtet dauerhaft). Dauert das Blinken länger als ein paar Sekunden oder geht die LED nicht aus, so weist dies auf ein Problem hin, zu dessen Eingrenzung man sich die Ausgabe in der REPL ansehen muss.

Der Robotling-Code benötigt für das Zusammensetzen und Verschicken einer Telemetrie-Nachricht im Schnitt etwa sieben Millisekunden. Die Nachrichten werden in `housekeeper` verschickt, das heißt in der Funktion, die Sensordaten aktualisiert und den Neopixel des Robotlings blinken lässt. Weil `housekeeper` 20-mal pro Sekunde aufgerufen wird, reduziert die Telemetrie-Option so auch die Rechenzeit, die dem Roboter für andere Aufgaben zur Verfügung stehen. Ohne Telemetrie verbringt der Robotling etwa 20 Prozent seiner Rechenzeit mit „Hausarbeit“, mit Telemetrie sind es gut 35 Prozent. Man sollte also Telemetrie nur aktivieren, wenn man sie nutzt.

Um anzuzeigen, was Robotling publiziert, kann man das oben eingeführte Kommandozeilenprogramm *mosquitto_sub* verwenden. Nach dem Wechsel in das Mosquitto-Programmverzeichnis auf dem PC abonniert man die Nachrichten mit:

```
.\mosquitto_sub.exe -h
   mqtt.eclipse.org -t
   "robotling_b4e62da1dccd/raw" -v
```

Listing 2

```
 1 import network
 2 import time
 3 from machine import Pin, unique_id
 4 from binascii import hexlify
 5 from umqtt.robust import MQTTClient
 6
 7 WLAN_SSID   = "Elchland3"           # WLAN-Name
 8 WLAN_PWD    = "LaufenImWald5519"    # WLAN-Passwort
 9 MQTT_BROKER = "mqtt.eclipse.org"    # IP-Adresse/URL des MQTT-Brokers
10 TOPIC       = b"led"                # Abonniertes Topic
11 CLIENT_ID   = hexlify(unique_id())  # eindeutige Modul-ID bestimmen
12
13 # Funktion, um eine WLAN-Verbindung herzustellen
14 def do_connect():
15   wlan = network.WLAN(network.STA_IF)
16   if not wlan.isconnected():
17     wlan.active(True)
18     wlan.connect(WLAN_SSID, WLAN_PWD)
19     while not wlan.isconnected():
20           pass
21
22 # Funktion, die beim Eintreffen von Nachrichten aufgerufen wird
23 def on_message(topic, msg):
24   print("Received message `{0}` under topic {1}".format(msg, topic))
25   if topic == CLIENT_ID +"/" +TOPIC:
26     # LED je nach dem Nachrichteninhalt an- oder ausschalten
27     led.value(int(msg))
28     print("LED switched {0}".format("0tFF" if int(msg) == 0 else "0N"))
29
30 if __name__ == "__main__":
31   # Mit  WLAN verbinden, MQTT-Client erzeugen und diesem die Funktion
32   # mitteilen, die beim Eintreffen von Nachrichten aufgerufen wird
33   do_connect()
34   client = MQTTClient(CLIENT_ID, MQTT_BROKER)
35   client.set_callback(on_message)
36   # LED-Objekt für Pin #13 (rote LED auf dem HUZZAH32-Board) anlegen
37   led = Pin(13, Pin.OUT, value=0)
38   try:
39     # Mit dem MQTT-Client verbinden und TOPIC abonnieren
40     client.connect()
41     client.subscribe(CLIENT_ID +"/" +TOPIC)
42     print("Connected to `{0}`, subscribed to `{1}`".format(MQTT_BROKER,
           TOPIC))
43     try:
44       while True:
45         # Prüfen, ob neue MQTT-Nachrichten eingetroffen sind; statt zu
46         # warten, kann hier auch etwas anderes erledigt werden
47         client.check_msg()
48         time.sleep(0.02)
49     finally:
50       # Verbindung zum MQTT-Broker beenden
51       client.disconnect()
52       print("Done.")
53   except OSError as err:
54     print("Error: Cannot connect to MQTT broker ({0})".format(err))
```

Der ESP32 wartet auf MQTT-Nachrichten unter dem Topic *led* und schaltet je nach Inhalt die rote Onboard-LED an oder aus. Wie bei Listing 1 müssen im Skript die Angaben zu WLAN und MQTT-Broker angepasst werden.

wobei auch die IP-Adresse des lokalen Brokers genommen werden kann (Option `-h`). Wichtig ist nur, dass in *NETWORK.py* derselbe Broker definiert ist. Das Topic (Option `-t`) muss durch GUID des eigenen Robotlings ersetzt werden (mit „robotling_“ vor der GUID). Der komplette String wird auch ausgegeben, wenn der Robotling bootet. Nach dem Starten von *mosquitto_sub* sollten Nachrichten wie die folgende ausgegeben werden:

```
[1] robotling_b4e62da1dccd/raw
{"state": 3, "timestamp_s": 53.663,
"power": {"motor_load": [0, 63],
"battery_V": 3.936561}, "debug": [1,
0.0], "sensor": {"compass":
{"pitch_deg": 96, "heading_deg":
342.8, "roll_deg": -7},
"distance_cm": [5, 4, 11]}}
```

Welche Daten der Robotling verschickt, hängt natürlich von dessen Ausstattung ab. Zum Beispiel taucht der Schlüssel `power : { motor_load : [0, 63], ...}` nur auf, wenn die Stromüberwachung (`USE_LOAD_SENSING` in *hexbug_config.py*) aktiviert ist und das Robotling-Board mit den Bauteilen für die Messung des Motorstroms ausgestattet ist.

Es fällt auf, dass die Nachrichten nicht in der oben erläuterten Topic-Struktur verschickt werden, sondern als komplexer String unter dem Topic *robotling_b4e62da1dccd/raw*. Dies geschieht aus Performancegründen; die Daten einzeln als MQTT-Nachrichten zu publizieren (z. B. den Wert 3.936561 unter dem Topic *robotling_b4e62da1dccd/battery_V*) würde zu viel Rechenzeit des ESP32 verbrauchen. Wie man MQTT-Inhalte strukturiert, hängt von der Anwendung ab. Hier geht es darum, den Zustand des Roboters möglichst zeitnah und häufig zu erfahren, um ihn quasi in Echtzeit visualisieren zu können. Meldet hingegen ein Thermostat alle 10 Minuten die aktuelle Temperatur und den Reglerzustand, so spielt der zeitliche Aufwand für das Verschicken einzelner Nachrichten keine Rolle und man kann eine übersichtlichere Topic-Struktur wie zum Beispiel *haus123/thermostat7/temperatur_C* etc.) verwenden, die unterschiedliche Daten getrennt aufbewahrt.

Um mit einer Nachricht pro `housekeeper`-Runde auszukommen, sammelt der Robotling alle Daten erst in einem Python-Dictionary, welches mittels der Funktion `ujson.dumps` in einen JSON-String umgewandelt und verschickt wird. JSON („JavaScript Object Notation“) ist ein einfaches Format für den Datenaustausch, das viele Programmiersprachen inklusive Python unterstützen. Auf dem PC wandelt die Funktion `json.loads` den JSON-String wieder in ein Python-Directory zurück.

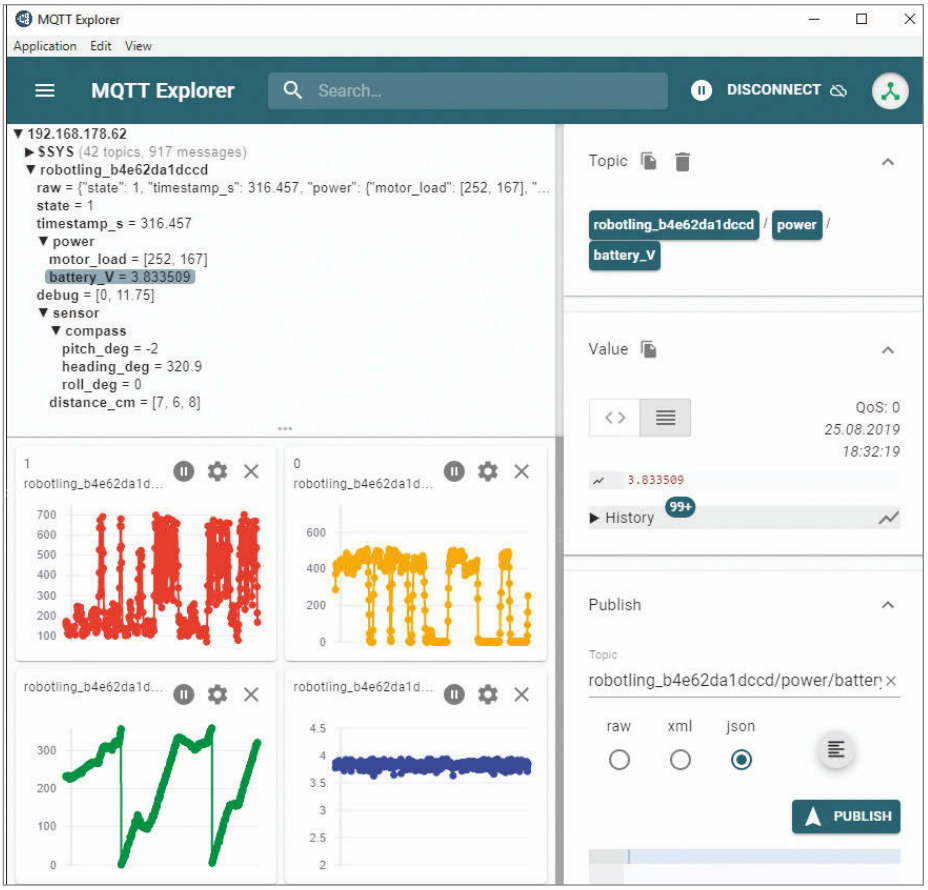
Das Programm *hexbug_relay.py* im Code-Verzeichnis benutzt paho-mqtt (siehe Kasten) und illustriert, wie man MQTT-Nachrichten unter Python auf dem PC empfängt. Für Details sei hier aus Platzgründen auf die Kommentare im Programmcode verwiesen. Es publiziert seinerseits die empfangenen Robotling-Nachrichten in einer ordentlicheren Topic-Struktur, was die Visualisierung der Daten mit MQTT-Tools wie dem kostenlosen MQTT-Explorer (Download siehe Kurzlink) vereinfacht. Im Screenshot ist auf der linken Seite des Fensters nicht nur das Topic *raw* zu sehen, sondern auch Topics wie *timestamp_s* und *power/battery_V*. Die MQTT-Telemetrie erlaubt es so, den aktuellen Zustand des Roboters aus der Ferne zu verfolgen und einfach nachzuvollziehen, welche Sensordaten zu welchem Verhalten führen.

Grundlage des Robotlings ist der Hexbug Spider.

Ausblick

Die Informationen, die Robotling verschickt, lassen sich nach Belieben erweitern, wenn der Roboter mit zusätzlichen Sensoren aufgerüstet wird. Anstatt ein allgemeines MQTT-Visualisierungstool zu verwenden, könnte man eine eigene GUI schreiben, die die Daten sensorspezifisch grafisch aufbereitet, zum Beispiel mit einer Kompass-artigen Darstellung für die Orientierungsdaten und einem Ladebalken als Batterieanzeige.

Denkbar ist auch, die Kommunikation beidseitig anzulegen, um dem Roboter Anweisungen zu schicken. Dies würde komplexere Verhalten ermöglichen, weil dafür die weit größere Rechenkapazität eines PC zur Verfügung steht. Dabei sollte man aber grundlegende Funktionen, wie das Stoppen vor Hindernissen oder Kanten, auf dem Mikrocontroller lassen. So kann der Roboter eigenständig weiterhin in Gefahrensituationen reagieren, falls die Qualität der MQTT-Verbindung mal schwankt. *—hch*



Mit dem Programm MQTT-Explorer kann man die Nachrichten des Robotlings anzeigen und sogar visualisieren. Im Screenshot werden die Belastung der Motoren (`motor_load`) in Rot und Gelb sowie die Laufrichtung (Grün, als Winkel in Grad) und die Batteriespannung (Blau, in Volt) in Diagrammen dargestellt.

Roboter selbst bauen

13 Bot-Anleitungen für Maker

Ein wenig führt das Cover dieses Buches in die Irre, denn die gezeigten Bürsten-"Roboter" mit Vibrationsmotor-Antrieb stellen nur die simpelsten Einsteiger-Projekte dar, die in diesem Buch beschrieben werden. Die Modelle für Fortgeschrittene verwenden hingegen den BBC micro:bit als Steuerplatine für einen per Smartphone ferngesteuerten Kriech-Bot, ein Katapult mit Bewegungsmelder, einen Blumengieß-Wächter oder eine Zweibein-Laufmaschine. Den krönenden Abschluss bilden Maschinen mit Arduino-Herz, unter anderem ein CNC-Stiftplotter aus zwei alten CD-Laufwerken.

Vom Schwierigkeitsgrad der Projekte her, der Aufmachung des Buches und der oft Wackelaugen tragenden Roboter eignen sich zumindest die ersten zehn Anleitungen gut für Kinder und (junge) Jugendliche (sofern sie löten können). Die letzten drei Projekte hingegen stellen schon höhere Anforderungen bei der mechanischen Konstruktion, aber vor allem auch im Hinblick auf die Software: Beim Code verweist der Autor auf seine Github-Seiten und die Dokumentation dort ist durchgängig nur auf Englisch zu lesen. *—pek*

Autor	Daniel Knox
Verlag	dpunkt.verlag
Umfang	160 Seiten
ISBN	978-3-86490-537-7
Preis	19,95 €

Awesome Robot Videos

Video-Freitag bei IEEE Spectrum

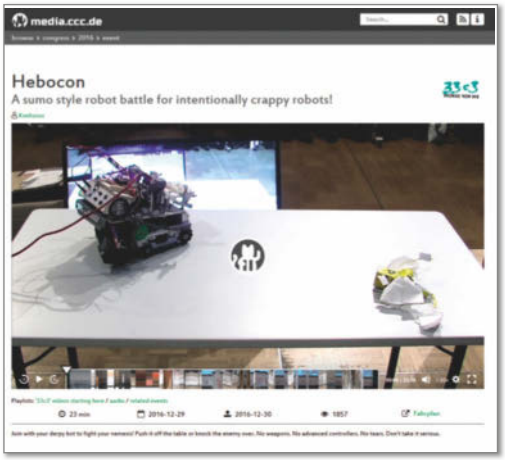
Ob lustiger Roboter-Unfall oder die fast schon furchteinflößenden Tricks der Roboter vom Forschungsunternehmen Boston Dynamics: Das Internet ist voll von unterhaltsamen Robotervideos. Da den Überblick zu behalten ist eine Kunst. Zum Glück sammelt das Fachmagazin IEEE Spectrum jede Woche die besten neuen Clips und stellt sie kompakt freitags zum Nachsehen zusammen. Die Zeitschrift gehört zum Institute of Electrical and Electronics Engineers (IEEE), der weltweit größten Organisation in den Ingenieurwissenschaften.

Entsprechend forschungsorientiert ist die Auswahl. Viele Videos sind Roboter-Demonstrationen von Forschungsgruppen, die als Ergänzung zu ihren wissenschaftlichen Veröffentlichungen entstanden sind. Neue Tricks vierbeiniger Lastenroboter werden ebenso gezeigt wie weiche Greifarme und die Brezelschlingen-Fertigungsanlage eines deutschen Maschinenbauers. Dazu werden auch neue Produkte vorgestellt, vom Lernroboter bis zur autonom fliegenden Drohne. Zum Abschluss des Video-Freitags gibt es außerdem noch Aufzeichnungen von Podiumsdiskussionen oder Vorträgen, wie etwa eine Masterclass von Robotik-Forscherin Madeline Gannon. Sie gab im Frühjahr dieses Jahres auf der Musik- und Technikkonferenz *Sónar D* einen Ausblick auf Zukunftsszenarien der Robotikentwicklung. *—hch*

URL spectrum.ieee.org/tag/video+friday

media.ccc.de

Medienplattform des Chaos Computer Club



Abgeschottete Hackerevents waren gestern. Mit seiner Medienplattform media.ccc.de bietet der Chaos Computer Club inzwischen ein riesiges Vortragsarchiv an, das komplett kostenlos zu nutzen ist. Über 7700 Aufnahmen von über 200 Technik-Events stehen dort aktuell zum Anhören oder Nachschauen zur Verfügung. Darunter sind nicht nur die Aufnahmen der jährlichen CCC-Congresse seit dem Jahr 2000, sondern auch von vielen weiteren chaosnahen Veranstaltungen wie dem Hannoveraner Hackover, dem Vintage Computing Festival oder der Nachhaltigkeitskonferenz Bits & Bäume.

Leider sind die ursprünglichen Veranstaltungen derzeit die einzige Einordnung der Aufnahmen. Wer Talks zum Thema Robotik sucht, sollte „robot" und „roboter" in die Suchleiste eintippen und stöbern. Musikspielende Roboter? Die Hebocon-Roboterkämpfe des 33C3?-Videos dazu stehen scheinbar unsortiert neben Vorträgen über Staubsaugerroboter als Abhörgeräte oder das Upgrade mechanischer Dinosaurier aus einem Freizeitpark mittels Raspis. Wer ein bestimmtes Thema sucht, muss etwas Zeit mitbringen. Auch der Link „next" für weitere Suchergebnisse taucht nur relativ klein am Anfang der Seite auf. Dafür gibt es insbesondere bei neueren Vorträgen die Möglichkeit, die Ansicht auf die Vortragsfolien zu ändern oder statt der Originalsprache eine Übersetzung zu hören – die Standards sind Deutsch und Englisch. Außerdem sind viele Beiträge in verschiedenen Auflösungen und Dateigrößen herunterladbar und können auf anderen Seiten eingebunden werden. *—hch*

URL media.ccc.de/search/?q=robot

Mensch, Roboter!

Leben mit Künstlicher Intelligenz und Robotik

Von antiken Schreibmaschinen bis zum iPhone gibt es im Paderborner Heinz Nixdorf MuseumsForum schon seit über 20 Jahren die Geschichte des Computers anzuschauen. Seit Oktober 2018 gehört jetzt auch eine Übersicht über Robotik zur Dauerausstellung. Star der Ausstellung ist der überdimensionale Roboterarm Beppo, der wie sein Namensgeber in Michael Endes Roman Momo den ganzen Tag am Fegen ist. Der festinstallierte Industrieroboter wiegt 900 Kilogramm und verfügt über eine Reichweite von 2,2 Metern. Statt Schrauben festzuziehen schiebt er allerdings rote Kunststoffpartikel durch die Gegend und jagt einem Lichtpunkt hinterher, den das Publikum steuern kann.

Neben Beppo gibt es noch eine Reihe weiterer Roboter zu entdecken, von humanoiden bis zum Spinnenroboter. Sie vermitteln, wie unterschiedlich Roboter sind, wie sie die Welt wahrnehmen oder sich sogar in ihr bewegen können. Mit Petra und Peter arbeiten im Museum sogar schon seit einigen Jahren zwei mobile Roboter, die in den Obergeschossen bei Fragen weiterhelfen. Wer nicht nur Ausstellungsstücke anschauen möchte, hat an verschiedenen Mitmachstationen etwa die Gelegenheit, gegen eine Künstliche Intelligenz „Vier gewinnt“ zu spielen oder Bilderkennung auszuprobieren. Weitere Vorträge und Workshops rund um Technik und Robotik schließlich gehören zum Begleitprogramm der Ausstellung.

Bild: Heinz Nixdorf MuseumsForum

Das Heinz Nixdorf MuseumsForum ist von Dienstag bis Freitag von 9 bis 18 Uhr geöffnet sowie am Wochenende von 10 bis 18 Uhr. Der ermäßigte Eintrittspreis beträgt 5 Euro, Erwachsene zahlen 8 Euro. Noch günstiger wird es für Gruppen ab 10 Personen. Angemeldete Gruppen von Schulen und Hochschulen können das Museum sogar kostenlos besuchen. *—hch*

URL hnf.de/veranstaltungen/mensch-roboter.html

I was a Robot

Science-Fiction und Popkultur

Auf der Suche nach Inspiration für den eigenen Roboter? Das Essener Museum Folkwang widmet sich in einer Sonderausstellung dem Roboterbild in Science-Fiction und Popkultur. Der Schwerpunkt liegt dabei auf menschenähnlichen Apparaten, die seit ihrem Aufkommen unsere Fantasie beflügelt haben. Auf der einen Seite sind sie Freund und Helfer, auf der anderen Feind und Zerstörer. Anhand von Maschinenwesen aus Büchern, Filmen, Comics und Computerspielen wird das Spannungsfeld zwischen ungehemmtem Fortschrittsglauben und dystopischer Weltuntergangsstimmung beleuchtet.

Insgesamt sind in der Ausstellung über 250 Exponate versammelt, darunter Filmplakate zum Klassiker Metropolis von 1927 und der Terminator-Reihe sowie Titelgestaltungen japanischer Mangas, von Isaac-Asimov-Büchern und der Perry-Rhodan-Serie. Sie stammen aus der Sammlung des schweizerischen Museums Maison d’Ailleurs, das sich der Science-Fiction und außergewöhnlichen Reisen verschrieben hat. Wer die Bilder mit nach Hause nehmen möchte oder es es nicht nach Essen schafft, findet viele der Bilder auch im bestellbaren Ausstellungskatalog.

Die Sonderausstellung „I was a Robot“ ist noch bis zum 15. März 2020 in Essen zu sehen. Das Museum Folkwang hat dienstags und mittwochs sowie an Wochenenden und Feiertagen von 10 bis 18 Uhr geöffnet; donnerstags und freitags sogar von 10 bis 20 Uhr. Der Eintrittspreis beträgt ermäßigt 3,50 Euro und 5 Euro (regulär). Darin inbegriffen ist auch der Zutritt zur Sonderausstellung „Made in Japan“ mit Plakaten des japanischen Grafikdesigners Shin Matsunaga. Der Eintritt zur ständigen Sammlung des Museums ist frei. *—hch*

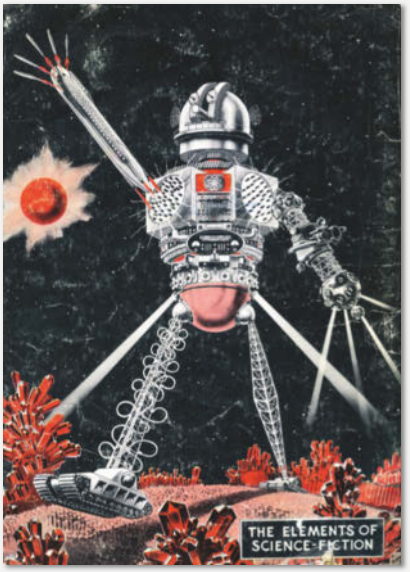


Bild: Maison d’Ailleurs, Schweiz / Agence Martienne

Frank Rudolph Paul, *The Elements of Science-Fiction*, 1953

URL museum-folkwang.de

Robotiklabor

Der Podcast rund um Robotikthemen

Die Podcast-Auswahl rund um Roboter ist leider recht übersichtlich. Bereits seit 2011 sendet immerhin das *Robotiklabor* aus Hannover. Der monatlich erscheinende Podcast bringt es inzwischen auf über 100 Folgen und wer sie nachhören möchte, hat viel zu tun. Die Folgen sind sehr lang und schlagen schnell mit drei bis fünf Stunden Hörzeit zu Buche. Das liegt auch an den mitunter eskalierenden Ausschweifungen. Deutlich kürzer und einsteigergeeigneter sind die Specials, die das Robotiklabor jedes Jahr auf der Maker Faire in Hannover produziert.

Neben Robotik geht es auch um alle möglichen Themen rund herum, vom Arduino über Hausautomation bis hin zu Filmtipps. Die Moderatoren Markus, Martin und Alexander kennen sich bereits seit ihrer Ausbildungszeit und stehen zu ihrem „gesunden Halbwissen". Alexander Moser dürften Make-Leser übrigens kennen, er fühlte in der Make 5/15 seinem Elektroauto auf den Zahn. Markus Knapp erklärt in diesem Heft auf Seite 98 Laserscanner. Seit einem Jahr ergänzt schließlich Moderatorin Jojo das Team. Ab und an sind auch Gäste mit dabei, etwa Make-Chefredakteur Daniel Bachfeld. *—hch*

URL robotiklabor.de

Robohub

Connecting the robotics community

Der Robohub soll Interessierte aus Forschung, Wirtschaft und Bildung zusammenbringen. Da in öffentlichen Debatten oft überzogene Erwartungen an Roboter-Fähigkeiten im Vordergrund stünden, sei es an der Zeit, Technik zu entmystifizieren und Wissen möglichst offen zugänglich zu machen, sagen die Betreiber. So finden sich auf der Seite etwa Mitteilungen aus Forschungsinstituten wie dem Labor für Künstliche Intelligenz der Uni Berkeley, die neue Algorithmen für Roboterbewegungen oder autonom fahrende Vehikel vorstellen. Andere Autoren berichten vom Bau ihrer Lego-Roboter nach Vorbild des zwölfbeinigen Strandbeest (siehe Make 1/16) oder von Robotik-Wettbewerben.

Neben Nachrichten, Meinungen und Lernangeboten gibt es noch einen Podcast, der sogar schon seit 2006 läuft. Inzwischen umfasst das Archiv knapp 300 Folgen, wobei der Themenschwerpunkt noch mehr in Richtung Wissenschaft und Robotik-Unternehmen geht. Pro Monat erscheinen ein bis zwei neue Interviews, jeweils mit einer Laufzeit zwischen 20 und 45 Minuten. Themen sind etwa das Robot Operating System (ROS), aber auch Fragen rund um Ethik in der Mensch-Maschine-Interaktion oder wie Start-ups an Gelder kommen. Ursprünglich „Talking Robots" genannt, war der Podcast das Projekt von Dario Floreano, Professor der schweizerischen Uni EPFL, in dem seine Promovierenden ihre Projekte vorstellen konnten. *—hch*

URL robohub.org

Motorsteuerung

mit Arduino & Raspberry Pi

Was sich bewegt, zieht Blicke auf sich. Im Zeitalter von Mikrocontrollern und unkompliziert zu handhabenden Kleinstcomputern bietet es sich an, Ansteuerungsmanagement und Bewegungsabläufe von Motoren softwaregestützt zu gestalten. Ibrahims Buch hilft zunächst dabei, eine Schneise durch den unübersichtlichen Dschungel infrage kommender Motoren zu schlagen. Verschiedene Motorentypen werden mit ihren wichtigsten Eigenschaften vorgestellt, Aufbau und Funktionsprinzipien werden erläutert. Im weiteren Verlauf kommen dann aber nur noch bürstenbewehrte Gleichstrommotoren sowie Schritt- und Servomotoren zum Einsatz.

Ibrahim berücksichtigt bei allen Projekten Arduino Uno und Raspberry Pi Zero parallel. Schaltpläne, Blockdiagramme, Quelltexte und Fotos gibt es jeweils zu beiden Plattformen. Den Arduino programmiert der Autor in C++, auf dem Raspberry Pi nutzt er Python.

Die ersten Beispielprojekte verdeutlichen programmgesteuertes Ein- und Ausschalten, Geschwindigkeitskontrolle und Drehrichtungswahl für Gleichstrommotoren. Sobald die Grundlagen gelegt sind, wird es komplexer: Es entsteht ein Roboter, der zunächst nur dunklen Linien folgen kann. Anschließend wird das Projekt um Bluetooth- und WLAN-Anbindung erweitert. Die Steuerung erfolgt zeitgemäß über eine App.

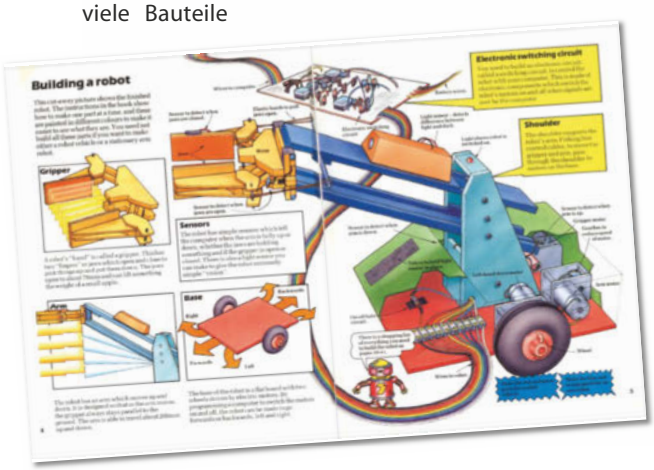
Das Buch richtet sich in erster Linie an Leser, die bereits einige Erfahrung mit Elektronik und der Programmierung von Mikrocontrollern haben, besonders was den Umgang mit Arduino und Raspberry Pi betrifft. Für die handwerklichen Basics liefert es wenig Hilfestellung, wenn man von seinem knappen Anhang zum Raspi absieht. Die Programmlistings sind etwas zu üppig kommentiert, die zahlreichen Fotos haben leider nur mäßige Qualität und der Index verdient seine Bezeichnung nicht. Wer nach einer guten Referenz zur Motorsteuerung mit einem der beiden populären Mikrocontrollersysteme sucht, kommt bei Ibrahims sachkundig geschriebenem Buch dennoch voll auf seine Kosten. *—Maik Schmidt/hch*

Autor	Dogan Ibrahim
Verlag	Elektor
Umfang	264 Seiten
ISBN	978-3-8957-6336-6
Preis	33 €

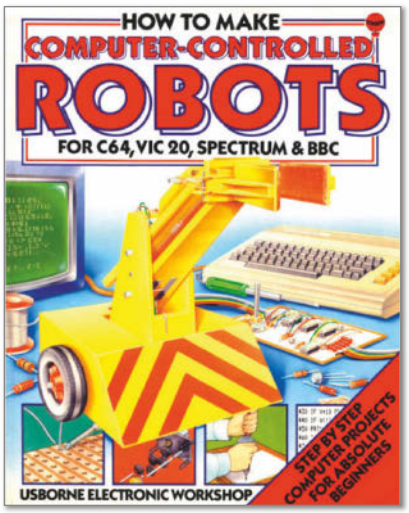
How to make computer-controlled robots

Retro-Einstieg

Neu ist nicht immer besser. Auch wenn es oft scheint, als überschlügen sich bei Computern die Neuerungen, sind viele grundlegenden Konzepte seit Jahrzehnten gleich geblieben. Seit den 1980er Jahren erklären die Bücher des Usborne Verlags Kindern Technik- und Programmierkonzepte praxisorientiert und mit vielen, aber übersichtlichen Bildern. Das Buch zum Bau eines fahrenden Greifroboters ist da keine Ausnahme. Von Rädern über LEDs bis hin zu Schrittmotoren sind viele Bauteile in aktuellen Kreationen die gleichen und die Hinweise zum Löten sind ebenfalls bis heute anwendbar. Schließlich ähnelt selbst der grundsätzliche Aufbau des Roboters einem Roomba und viele Eigenbauten bei den Roboterwettkämpfen Hebocon sehen ebenso aus.



Dennoch zeigt sich das Alter des Buches spätestens bei den Programmierhinweisen für Heimcomputer wie den C64 und den BBC Model B. Außerdem gibt es inzwischen auch zweibeinige Roboter und Sprachausgaben finden sich in jedem Smartphone. Insbesondere die vielen Grafiken und kindgerechten Erklärungen sorgen am Ende aber dafür, dass wir das Buch noch heute empfehlen. Erwachsene Roboter-Fans dürften sich außerdem an dem Retro-Charme erfreuen. Das Buch stellt der Verlag – wie viele weitere alte Ausgaben rund um Mikrocontroller und Programmierung – kostenlos zum Download zur Verfügung. *—hch*



Verlag	Usborne Publishing
Umfang	50 Seiten
Preis	kostenlos
URL	usborne.com/browse-books/features/computer-and-coding-books

Roboter

Wie funktionieren die Maschinen der Zukunft?

Dieses Sachbuch behandelt Roboter in allen Varianten. Es ist eine Übersetzung aus dem Englischen, richtet sich an Kinder ab etwa 10 Jahren und besteht aus vielen Bildern und kurzen, verständlich geschriebenen Texten.

Zunächst fällt die bunte Gestaltung und das moderne Design auf – speziell das hochwertig verarbeitete Hardcover macht einen positiven Eindruck. Das Buch ist auf dem aktuellen Wissensstand.

Am Anfang erklären die Autoren und Autorinnen die Herkunft der Roboter und Automaten und deren Entwicklung im Verlauf der Geschichte. Auch die gesellschaftliche und kulturelle Einordnung der Roboter ist Thema. Die einzelnen Roboter werden mithilfe eines stichpunktartigen Textes charakterisiert. Außerdem werden die wichtigsten Merkmale zu jedem Roboter mit den Symbolen in einer Übersichtsleiste am oberen Seitenrand dargestellt. So fällt es leicht, sich schnell zu orientieren. Die Roboter wurden nach verschiedenen Einsatzzwecken sortiert, etwa Industrieroboter, Roboter für zuhause und Roboter für extreme Umgebungen. Mit dieser großen Bandbreite dürfte für jeden etwas Neues dabei sein. Zu den verschiedenen Typen erklärt das Buch auch die unterschiedlichen Techniken, mit denen die jeweiligen Roboter arbeiten – sowohl in Bezug auf die Hardware als auch auf die Software. Die unterschiedlichen Bestandteile von Robotern werden vorgestellt und auch der Begriff Künstliche Intelligenz erklärt.

Am Ende des Buches gibt es ein Glossar, welches die wichtigsten verwendeten Begriffe verständlich erläutert. Gerade diese große und leicht verständliche Übersicht gefällt gut. Negativ fällt auf, dass es teilweise kleinere fachliche Fehler gibt – so wird in der Beschreibung eines Roboters der Motor als Energiequelle bezeichnet. Trotzdem ist das Buch empfehlenswert für alle Nachwuchs-Ingenieure, nicht zuletzt aufgrund des guten Preis-Leistung-Verhältnisses *—Jonas Kriegel/esk*

Autorinnen	Laura Buller, Clive Gifford, Andrea Mills
Verlag	Dorling Kindersley
Umfang	160 Seiten
ISBN	978-3-8310-3673-8
Preis	16,95 €

Impressum

Redaktion

Make: Magazin
Postfach 61 04 07, 30604 Hannover
Karl-Wiechert-Allee 10, 30625 Hannover
Telefon: 05 11/53 52-300
Telefax: 05 11/53 52-417
Internet: www.make-magazin.de

Leserbriefe und Fragen zum Heft: info@make-magazin.de

Die E-Mail-Adressen der Redakteure haben die Form xx@make-magazin.de oder xxx@make-magazin.de. Setzen Sie statt „xx" oder „xxx" bitte das Redakteurs-Kürzel ein. Die Kürzel finden Sie am Ende der Artikel und hier im Impressum.

Chefredakteur: Daniel Bachfeld (dab)
(verantwortlich für den Textteil)

Stellv. Chefredakteur: Peter König (pek)

Redaktion: Heinz Behling (hgb), Helga Hansen (hch), Carsten Meyer (cm), Rebecca Husemann (rehu), Elke Schick (esk)

Mitarbeiter dieser Ausgabe: Thomas Euler, Sam Groveman, Detlef Heinze, Markus Knapp, Jonas Kriegel, Imen Mguedmini, Niq Oltman, Brandon Satrom, Kai Schade, Maik Schmidt, Daniel Springwald

Assistenz: Susanne Cölle (suc), Christopher Tränkmann (cht), Martin Triadan (mat)

DTP-Produktion: Nicole Judith Hoehne (Ltg.), Martina Bruns, Martina Fredrich, Jürgen Gonnermann, Birgit Graff, Angela Hilberg, Wolfgang Otto (Korrektorat), Astrid Seifert, Dieter Wahner

Art Direction: Martina Bruns (Junior Art Director)

Layout-Konzept: Martina Bruns

Layout: Nicole Wesche

Fotografie und Titelbild: Andreas Wodrich, Melissa Ramson

Verlag

Maker Media GmbH
Postfach 61 04 07, 30604 Hannover
Karl-Wiechert-Allee 10, 30625 Hannover
Telefon: 05 11/53 52-0
Telefax: 05 11/53 52-129
Internet: www.make-magazin.de

Herausgeber: Christian Heise, Ansgar Heise

Geschäftsführer: Ansgar Heise, Dr. Alfons Schräder

Verlagsleiter: Dr. Alfons Schräder

Stellv. Verlagsleiter: Daniel Bachfeld

Anzeigenleitung: Michael Hanke (-167)
(verantwortlich für den Anzeigenteil),
www.heise.de/mediadaten/make

Leiter Vertrieb und Marketing: André Lux (-299)

Service Sonderdrucke: Julia Conrades (-156)

Druck: Firmengruppe APPL echter druck GmbH, Delpstraße 15, 97084 Würzburg

Vertrieb Einzelverkauf:
VU Verlagsunion KG
Meßberg 1
20086 Hamburg
Tel.: 040/3019 1800, Fax.: 040/3019 145 1800
E-Mail: info@verlagsunion.de
Internet: www.verlagsunion.de

Einzelpreis: 10,90 €; Österreich 11,90 €; Schweiz 18,00 CHF; Benelux, Italien, Spanien 11,90 €

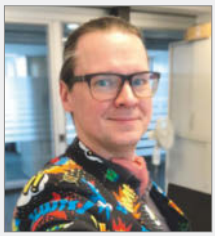
Abonnement-Preise: Das Jahresabo (7 Ausgaben) kostet inkl. Versandkosten: Inland 65,10 €; Österreich 66,50 €; Schweiz/Europa: 72,10 €; restl. Ausland 88,20 €

Das Make-Plus-Abonnement (inkl. Zugriff auf die App, Heise Select sowie das Make-Artikel-Archiv) kostet pro Jahr 6,30 € Aufpreis.

Abo-Service:

Bestellungen, Adressänderungen, Lieferprobleme usw.:
Maker Media GmbH
Leserservice
Postfach 24 69
49014 Osnabrück
E-Mail: leserservice@make-magazin.de
Telefon: 0541/80009-125
Telefax: 0541/80009-122

ISSN 2364-2548

Nachgefragt

Welche Begegnung mit einem Roboter hat Dich geprägt?

Kai Schade

aus NRW, stellt ab Seite 64 seinen Roboter Miracolo vor

Als Kind hat mich der Film *Nummer 5 lebt* sehr inspiriert. So wie Nummer 5 stets begierig auf neuen Input war, stelle ich mir einen Roboter vor, der neues Wissen sucht, welches er selber lernen möchte.

Daniel Springwald

Bochum, hat seinem Roboter das Go-Spiel beigebracht und beschreibt ihn ab Seite 70

Als Kind hat der Filmroboter *Johnny 5* den größten Eindruck bei mir hinterlassen. Später als Jugendlicher war es dann aber eindeutig *Lieutenant Commander Data*.

Markus Knapp

Hannover, führt ab Seite 98 in den Umgang mit Laserscannern ein

Ich habe als Kind mit einem schwarzen Spielzeugroboter gespielt und fand dessen Lichter und Geräusche faszinierend – und ich war immer neugierig, wie so etwas von innen aussieht und wohl funktioniert.

Thomas Euler

Tübingen, kommuniziert ab Seite 112 mit seinem Krabbelroboter über MQTT

Mich haben mehr die Menschen geprägt, die sich mit Robotern spielerisch auseinandersetzen – so wie Stanisław Lem in seinen „Robotermärchen".

Inserentenverzeichnis

NEU

Das Beste aus einem Jahrgang c't:

c't Schlagseitenkalender 2020

Die besten Schlagseiten von Ritsch & Renn des abgelaufenen c't-Jahrgangs in einem hochwertigen Kalender für 2020.
Der perfekte Wandschmuck für Heim und Büro echter c't-Fans!

7,90 € für c't-Netzwerker!

9,90 € ›

PORTOFREI
AB 15 €
BESTELLWERT

› Generell portofreie Lieferung für Heise Medien- oder Maker Media Zeitschriften-Abonnenten oder ab einem Einkaufswert von 15 €. Nur solange der Vorrat reicht. Preisänderungen vorbehalten.

heise shop
shop.heise.de/kalender2020 ›

Make:

Das perfekte Maker-Geschenk!

Ihre Vorteile:

- ✓ **7× im Jahr** Maker-Ideen verschenken
- ✓ **Conrad-Gutschein** für Sie oder den Beschenkten
- ✓ Inklusive **Geschenk-Gutschein**
- ✓ **Versandkostenfrei**

Jetzt bestellen: make-magazin.de/schenken