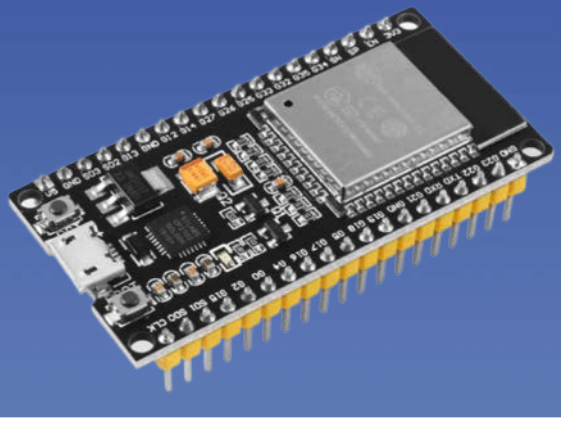
DEUTSCHLANDS GEFÄHRLICHSTES DIY-MAGAZIN

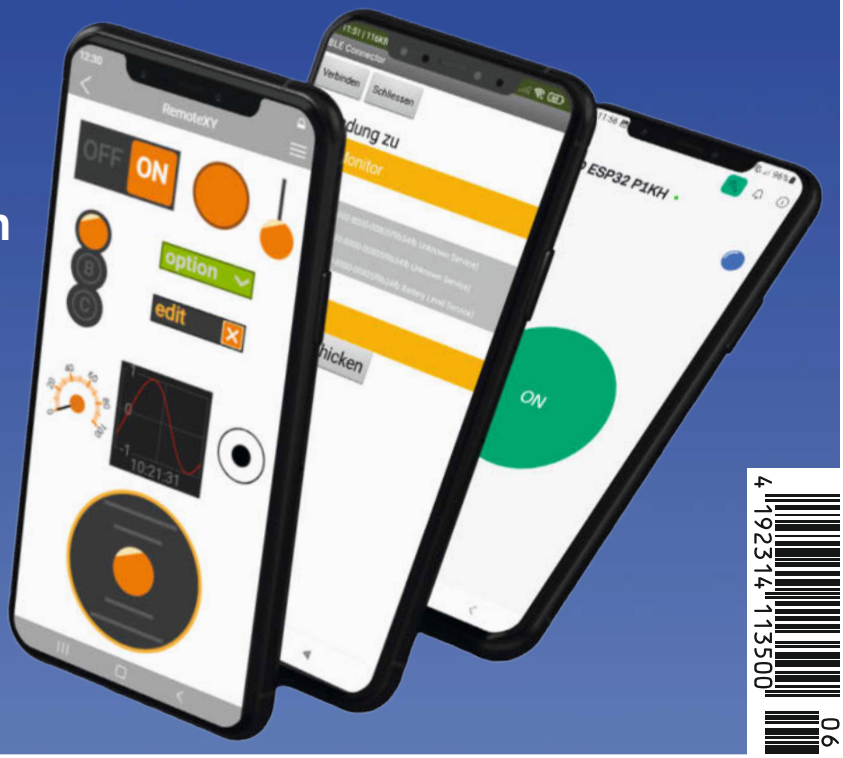
c't Make: SONDERHEFT

ALTE HANDYS NEUE PROJEKTE

Maker-Apps programmieren

- Leichter Einstieg, schnelle Ergebnisse
- Arduino, ESP & Co steuern
- Bedienoberflächen gestalten
- Abläufe automatisieren





6/24
18.10.2024
CH CHF 26.50
AT 14,90
Benelux 15,90
€ 13,50

4 192314 113500 06

Praktische Tools

- Termux: Linux für die Hosentasche
- Onshape: CAD unterwegs
- Phyphox: Physiklabor als App

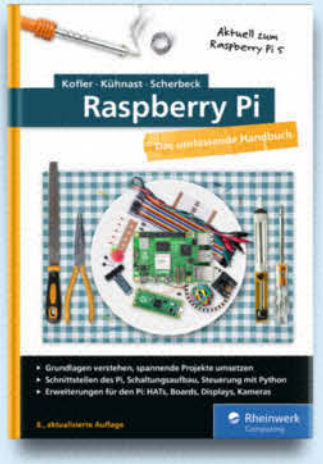
DIY-Projekte

- Wasserwaage mit Neopixeln
- Fernsteuerbarer Mediaplayer
- Hau drauf: Quiz-Buzzer

www.make-magazin.de

Tüfteln, Knobeln, Basteln!

Mit unseren Büchern werden Sie zum Alles-Erfinder. Lassen Sie sich von spannenden Projekten für Maker und Tekkies inspirieren: Tomatenzucht mit dem Mikrocontroller, geniale Roboter-Autos und moderne Smart-Home-Technik. Garantierter Tüftelspaß – bis der Lötkolben raucht!



Exklusiv als E-Book

www.rheinwerk-verlag.de/maker

Rheinwerk

Trennungsschmerz

Als Kinder haben wir Sperrmüllsammlungen durchforstet, um Lautsprecher für Gitarrenverstärker, Anzeigelämpchen für Raumkapsel-Cockpits und mechanische Bauteile für Roboter auszuschlachten. Manchmal haben wir daraus etwas gebaut, aber noch öfter einfach nur gesammelt.

Heutzutage gibt es noch viel mehr Elektroschrott. Einen großen Anteil davon machen unsere allgegenwärtigen Smartphones und Tablets aus. Mit der Zeit wurden diese Geräte immer kleiner und hochintegriert und man kann sie nicht mehr wirklich ausschlachten. Oft wird bei der Herstellung geklebt statt geschraubt. Für die nächste Generation Apps sind sie nach ein paar Jahren einfach nicht mehr schnell genug, Akkus haben nur eine begrenzte Lebenszeit und sind fest verbaut. Die tollen Displays sind undokumentiert und benutzen schwer zu bekommende Stecksysteme.

Aber man kann sie sammeln! In der Schublade. Zu schade für den Müll.

Wir Maker sammeln nicht, weil wir Messies sind, sondern weil wir immer an das nächste Kopf-Projekt denken. Also ein Projekt, das erst einmal eine tolle Idee ist und sobald Zeit vorhanden, dann ganz bestimmt gemacht wird. Manchmal. Denn seien wir ehrlich, so richtig fertig werden wir nie und der „Pile of Shame" wird immer höher.

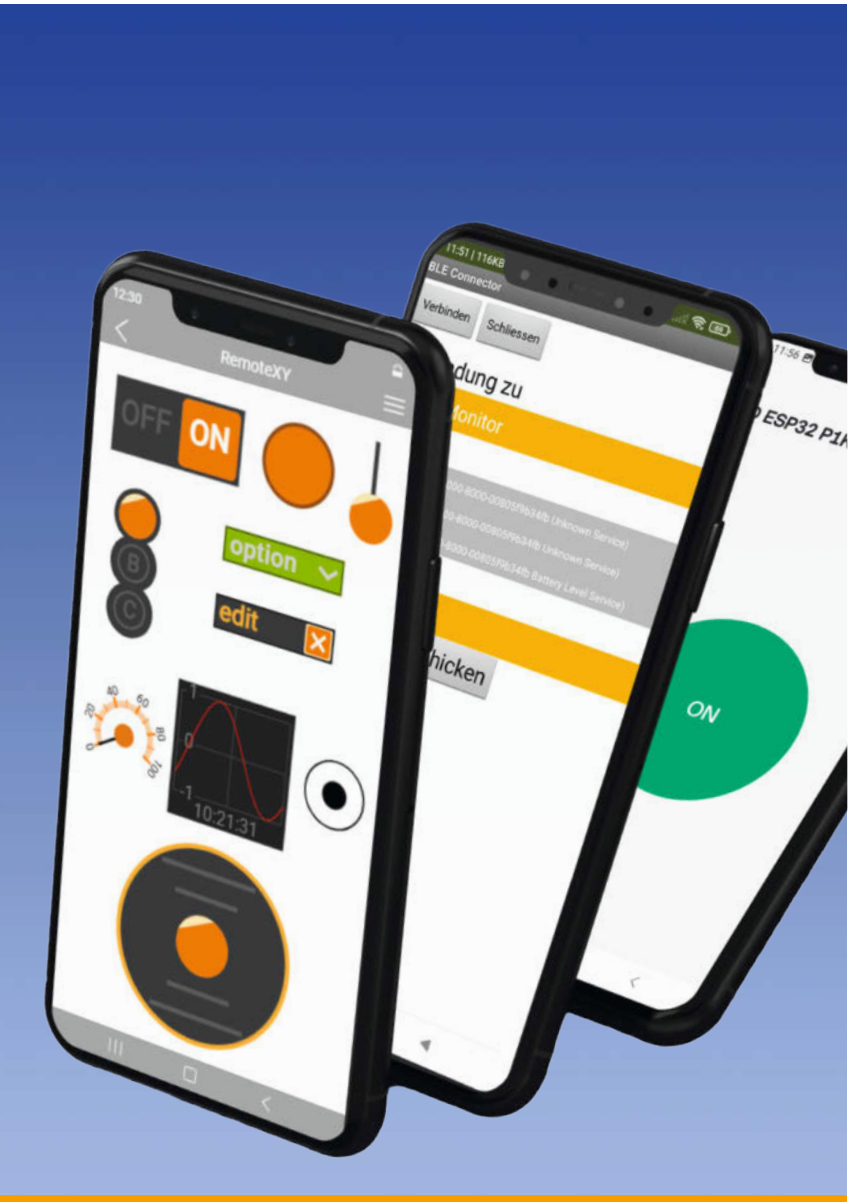
Um zumindest Ihren Stapel zu verkleinern, dachten wir uns, wir zeigen Möglichkeiten, was man mit (alten) Smartphones so alles steuern und sonst noch so anfangen kann. Ziel ist es hier, möglichst das gesamte Gerät zu verwenden, denn Smartphones sind wirklich flinke Computer mit interessanter Hardware wie einem Display mit Touchfunktion, Mobilfunk, WLAN und tollen Sensoren.

Wir möchten euch motivieren: elektronische Geräte anders verwenden, länger nutzen und sie wieder einem sinnvollen Zweck zuführen. Und wenn es doch nicht klappt, bringt sie zum Recycling!

Carsten Wartmann

▶ **make-magazin.de/x1a3**

Inhalt



Apps schnell gemacht

Eine maßgeschneiderte Android-Anwendung kann viele Maker-Projekte bereichern, sei es durch die Abfrage von Sensoren, die Steuerung von Mikrocontrollern oder die Darstellung von Inhalten auf einer Benutzeroberfläche. Die Programmierung solcher Apps ist jedoch oft mit viel Aufwand verbunden. Mit den richtigen Werkzeugen wird das Erstellen einer App jedoch zum Kinderspiel. Wir stellen die besten Tools dafür vor und zeigen, wie man sie benutzt.

Tools für unterwegs

Gerade unterwegs kommen einem gute Ideen, die man am liebsten sofort umsetzen möchte. Zum Glück lässt sich mit dem Smartphone in der Tasche vieles realisieren. Mit der App Termux haben wir Linux dabei, mit Onshape können CAD-Zeichnungen schnell bearbeitet werden. Und mit Phyphox werden die Sensoren des Smartphones genutzt, um Bewegungen, Schall und Magnetfelder zu erforschen.

Windows & Linux im Handy

Keinen Bock mehr auf Android? Es gibt gute Alternativen, um die Lebensdauer alter Handys zu verlängern und dabei Desktop-Feeling zu genießen. Mit PostmarketOS erhalten wir auf unterstützten Geräten ein nativ laufendes Linux-Betriebssystem. Für Windows-Enthusiasten existiert das „Lumia WOA Project", das die Installation der Smartphone-freundlichen ARM-Version von Desktop-Windows auf Lumia-Smartphones ermöglicht.

Tonbandgerät aufpimpen

Nichts erfreut Auge und Ohr mehr als Musik von einer alten Bandmaschine. Um das Abspielen noch eleganter zu machen, verwandeln wir in diesem Projekt ein altes Smartphone in eine kabellose Fernbedienung. Und das schaffen wir, ohne das alte Revox A77 Tonbandgerät verändern zu müssen. Für das Projekt brauchen wir nur ein altes WLAN-fähiges Smartphone oder Tablet, einen WEMOS und ein paar Relais.

Themen von der Titelseite sind rot gesetzt.
Grafik Titelthema: 801/freepik.com

Messen und spielen

In drei DIY-Projekten demonstrieren wir die Steuerung eigener Hardware mit alten Smartphones. Zunächst entsteht ein Quiz-Buzzer für den Spieleabend, gesteuert per RemoteXY-App. Das zweite Projekt erklärt, wie man mit RemoteXY Smartphone-Sensoren ausliest, um eine Wasserwaage mit Neopixeln zu realisieren. Im letzten Projekt machen wir aus einem alten Handy einen fernsteuerbaren Mediaplayer für die Stereoanlage.

Shutterstock

Schubladen-Smartphones weiternutzen

Ehemalige High-Performance-Smartphones und Tablets verrotten in der Schublade, weil Anwendungen immer schlapper laufen. Im Maker-Labor lassen sich viele Geräte trotzdem noch sinnvoll einsetzen. Unser Heft liefert konkrete Vorschläge und ganz viel Inspiration. Eine Einführung ins Thema.

von Daniel Bachfeld

Über unseren WhatsApp-Kanal machen wir regelmäßig kleine Umfragen unter unseren Lesern. Zu einer Frage gibt es mehrere Antworten, die man anklicken kann, und wir sehen anhand der Häufung, welche Themen beispielsweise gerade besonders relevant sind.

Bei der Frage „Was macht ihr eigentlich mit alten Smartphones?“ lag die Antwort „Schublade“ mit weitem Abstand ganz vorne. Ähnlich dürfte es bei Tablets aussehen. Im dreiköpfigen Haushalt des Autors dieses Artikels liegen 5 Android-Smartphones, 4 Android-Tablets, 1 iPhone und 2 iPads mehr oder minder ungenutzt rum. Von den alten Nokias, Ericssons und Siemens' mal ganz zu schweigen.

Laut Bundesamt für Statistik belief sich die Zahl der weltweit verkauften Smartphones im Jahr 2023 auf rund 1,17 Milliarden Stück. Im Jahre 2022 bunkerten die Deutschen über 200 Millionen Alt-Handys und -Smartphones. Man möchte lieber nicht wissen, wie viele Milliarden Smartphones gerade weltweit in Schubladen vergammeln.

Aber warum können sich die wenigsten von ihren alten Geräten trennen? Weil man – zumindest als Maker – irgendwie das Gefühl hat, das könnte man noch für irgendwas verwenden. Denn in den wenigsten Fällen gehen Smartphones und Tablets wirklich kaputt. Vielmehr reagieren sie meist nur noch träge, weil offenbar die Hardware nicht mehr zu den Anforderungen aktueller Software passt.

Das Phänomen kennen viele schon von Laptops und PCs. Office-Suiten, Browser, Mediaplayer und immer mehr parallel im Hintergrund laufende Dienste fressen mit jedem Update weitere Ressourcen, sodass irgendwann der Speicher knapp wird und die Auslastung der CPU-Kerne weiter steigt.

Diese Beobachtung hat sogar einen Namen: „Wirthsches Gesetz“, benannt nach der Informatik-Legende Niklaus Wirth. Er postulierte bereits 1995, dass Software schneller langsam wird, als Hardware schneller. Ursache sei Software, die durch erhöhte Komplexität immer ineffizienter und zudem nachlässig programmiert werde.

Vergessen darf man aber auch nicht die Strategien der Hersteller, die mit besseren Kameras, schärferen Display und neuen Funktionen in aktuellen Modellen werben und so den Kauf neuer Modelle schmackhaft machen. KI-Unterstützung ist aktuell der neueste Trend.

Supercomputer in der Tasche

Vergleicht man Smartphones mit mehr oder minder aktuellen Single-Board-Computern wie Raspberry Pi, Odroid, Banana Pi und an-

deren, wird schnell klar, dass selbst ältere Smartphone-Modelle ähnlich leistungsfähig und zudem noch besser ausgestattet sind.

Neben einem Touch-Display, Mehrkern-Prozessor, Speicher, Akku, Kamera, Mikrofon, Lautsprecher, USB, WLAN und Bluetooth enthalten sie zusätzlich noch Module für Mobilfunk, GPS und Glonass, NFC sowie Sensoren für Drehwinkel (3-Achs-Gyroskop), Beschleunigung (3-Achs-Accelerometer), Magnetfeld (3-Achs-Magnetometer), Helligkeit, Temperatur und oft Luftdruck (Barometer). Vieles davon muss man beim Raspberry Pi und anderen Boards erst mal nachrüsten.

Damit sind Smartphones und Tablets die ideale Grundlage für eigene Ideen und Projekte und machen an der einen oder anderen Stelle den Kauf eines neuen Mikrocontroller-Boards überflüssig.

Eigene Projekte

Allerdings sind die Systeme nicht ganz so offen und flexibel wie ein Raspberry Pi oder ein Arduino, auf denen man eigene und fremde Software nach Belieben installieren und konfigurieren kann, um angeschlossene Sensoren auszulesen und Aktoren zu steuern.

Apples Geräte sind leider so abgeschottet, dass man nur als Profi-Entwickler über den App-Store eigene Software installieren kann. Zwar gibt es nützliche Maker-Apps unter iOS, denen mangelt es jedoch meist an echten Steuerfunktionen über USB respektive die emulierte serielle Schnittstelle. Deshalb dreht es sich in diesem Heft – mit einer Ausnahme auf Seite 66 – nur um Android-Geräte.

Android ist offener, erlaubt die Installation eigener Software und bietet vergleichsweise einfachen Zugriff auf die interne Hardware. Über die USB-Buchse kann man die serielle Schnittstelle nutzen, um per Kabel angeschlossene Elektronik zu steuern. Alternativ liest und sendet man Daten per WLAN oder Bluetooth und sogar per NFC.

In diesem Heft haben wir viele Ideen zusammengetragen und auf insgesamt 132 Seiten in Artikel gegossen. Darin zeigen wir Wege auf, wie man Android-Geräte in eigenen Projekten einsetzt und nachhaltig weiter nutzt. Das reicht von der vorgefertigten Anwendung aus dem Google Play Store über die Konfiguration von Bedienoberflächen für Mikrocontroller bis zur Programmierung eigener Apps in Basic oder Java.

Vielleicht gelingt es unseren Lesern ja, Apps zu entwickeln, die effizienter als die bisher bekannten Browser-, Musik- und Messaging-Anwendungen arbeiten. Vielleicht entsteht über unsere Inspiration auch eine gänzlich neue Anwendung. Sagen Sie uns Bescheid!

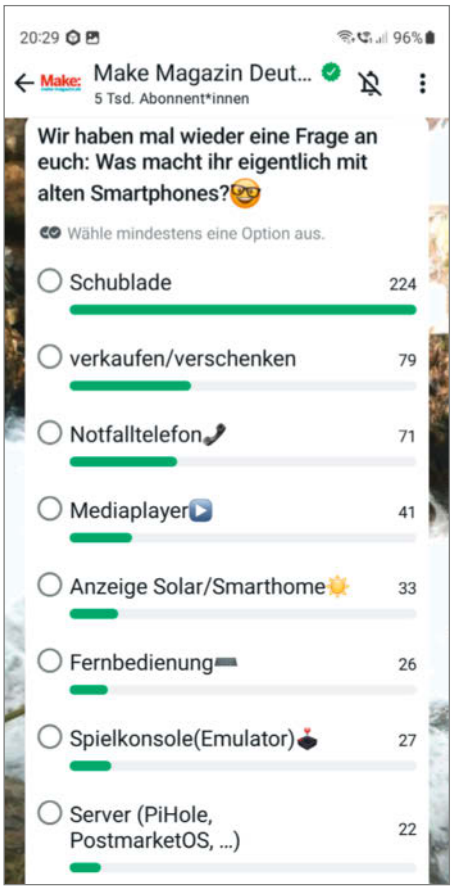
Im Folgenden geben wir in diesem Artikel noch ein paar Anregungen, die wir in keinem anderen Artikel im Heft unterbringen konnten.

Mehr Ideen

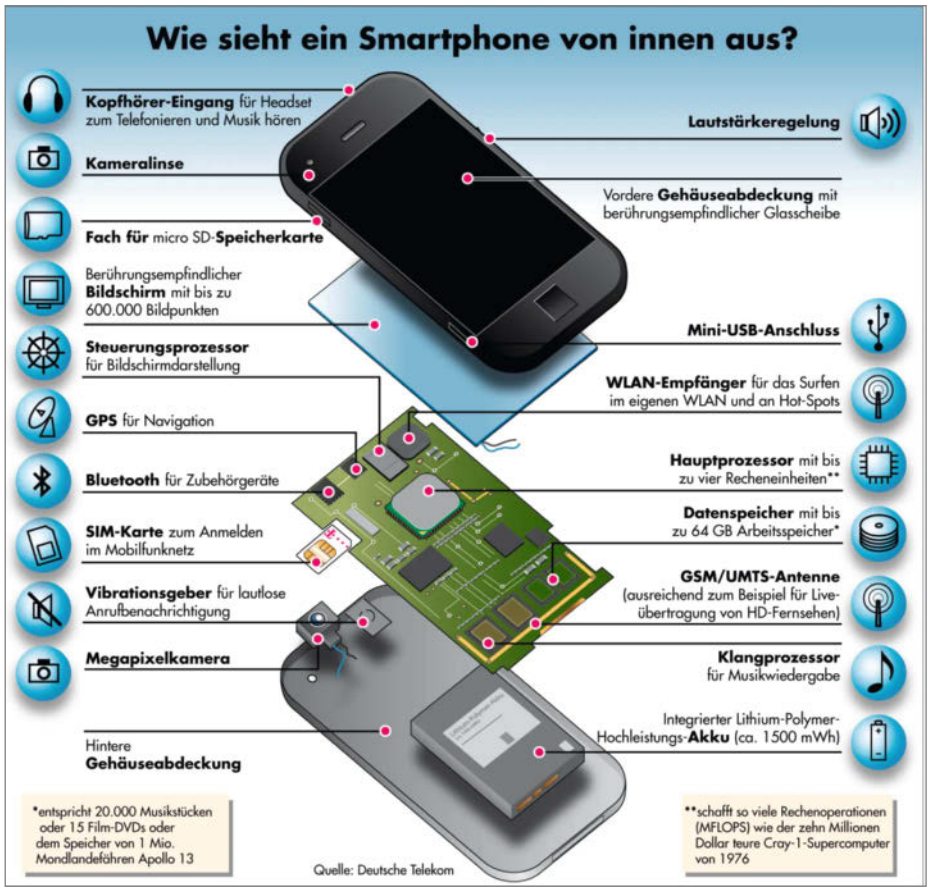
Smartphones sind mit ihren Sensoren und Netzwerkschnittstellen prädestiniert für mobile Datenerfassung. Im weitesten Sinne zählt dazu auch die interne Kamera, die sich per WLAN über Anwendungen wie DroidCam oder iVCam als Webcam einbinden lässt, um Dinge zu überwachen. Sei es der Fortschritt des 3D-Druckers, der Fräse oder des Lasercutters in der Werkstatt – die man ja eigentlich eher nicht unbeaufsichtigt arbeiten lassen sollte.

In Kombination mit der Streaming- und Recording-Software OBS kann man DroidCam sogar über ein extra Plugin einbinden, um beispielsweise Nahaufnahmen zu machen. Zusammen mit Handyhalterungen mit Saug- oder Schraubfuß, Schwanenhals und anderen Lösungen kann man die Smartphones stabil montieren und ausrichten.

In der Make 2/22 haben wir ein Projekt vorgestellt, wie man ein Smartphone als Steuereinheit für einen Fahrroboter aus Pappe, Motoren, leitfähigem Klebeband und einfacher Elektronik einsetzt – ganz ohne Mikrocontroller. Die Anleitung ist ein tolles Projekt für Schulen und Familien (siehe Link).



Unsere Umfrage in unserem WhatsApp-Kanal



Das steckt in einem älteren Smartphone alles drin.

Ein altes Smartphone steuert einen Fahrroboter aus Pappe.

OpenBot ist ein offenes Projekt von Intel und nutzt die Kamera von Smartphones für autonomes Fahren.

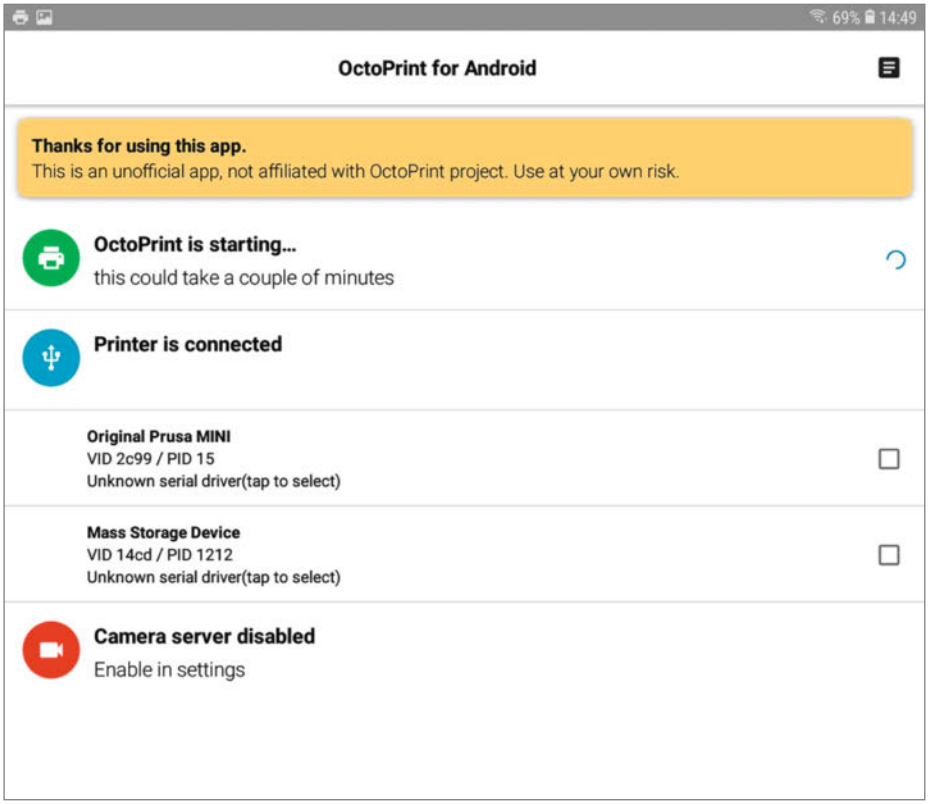
Ebenfalls für den Unterricht geeignet ist der OpenBot, aber eher für ältere Schülerinnen und Schüler. Zu dem Chassis aus Pappe oder 3D-gedrucktem Material fügt man vier Motoren, eine Motortreiber-Platine und einen Arduino Nano hinzu. Der wird per USB vom Smartphone gesteuert. Mittels der Kamera und einer „Person Follower App" kann man den Roboter dazu bringen, Personen hinterherzufahren.

Spannender sind die dazugehörigen Anleitungen, wie man den OpenBot mit einem Gamepad manuell über einen Parkour steuert, um Bilder aufzunehmen, die später als Trainingsdaten für ein Tensorflow-Modell einer KI dienen. Ziel ist, dass der OpenBot damit eine Straße autonom entlang fährt (siehe Link).

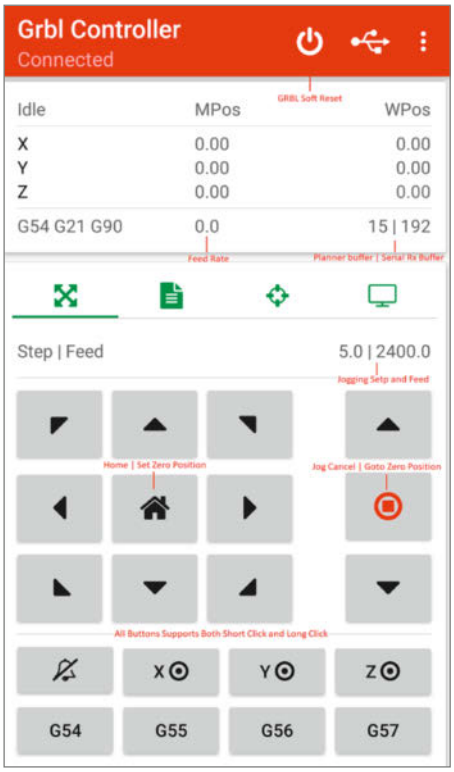
Werkstatt-Apps

3D-Drucker und Fräsen sind im weitesten Sinne auch (Fertigungs-)Roboter, ihre Steuerungen fahren nämlich mithilfe von Motoren und Encodern präzise Positionen an. Eine unserer beliebtesten Anwendungen bei 3D-Druckern ist OctoPrint, das man üblicherweise auf einem Raspberry Pi installiert. OctoPrint macht 3D-Drucker netzwerkfähig und bietet per Weboberfläche viele Einstellmöglichkeiten zum komfortablen Drucken. Das erspart uns, mit dem USB-Stick zum Drucker gehen zu müssen.

Mit dem unabhängigen Fork Octo4A kann man einen Pi durch ein Smartphone ersetzen. Alles was man dafür benötigt, ist (bei älteren Modellen) ein USB-OTG-Adapter.



Octo4A installiert Octoprint auf Android-Smartphones. Damit spart man sich den Raspberry Pi.



Die „Grbl Controller"-App steuert Fräsen und andere Gerät mit grbl-Firmware v1.1.

USB on-the-Go

Dank USB OTG können Smartphones und Tablets als Host fungieren, das heißt, man kann an sie andere Geräte wie an einen PC anschließen. Speichersticks, Webcams, Mäuse, Keyboards und so weiter. Da Android im Kern ein Linux-System ist, ist die Unterstützung dafür softwaremäßig durch passende Treiber bereits vorhanden.

OTG ermöglicht es einem Smartphone, sowohl als Host als auch als Peripherie zu arbeiten. Als Host muss es angeschlossenen Geräten jedoch Strom liefern. Bei Modellen mit Mini- und Micro-USB ist ein OTG-Adapter erforderlich, um die passenden USB-Buchsen anzubieten. Zudem zieht der Adapter beim Einstecken den Sense-Pin auf 0V. Android stellt dann den Port auf Host-Betrieb um und meldet das im Display.

Während native USB-Geräte das ziemlich komplexe USB-Protokoll unterstützen müssen, kann das Smartphone mit einem externen USB-zu-Seriell-Wandler mit Geräten kommunizieren, die nur eine serielle Schnittstelle mitbringen. Die erforderlichen Treiber bringt Android meist mit.

Auf diese Weise funktionieren viele Projekte mit Mikrocontrollern, die per USB angeschlossen sind. Damit eröffnet sich ein ganzes Universum an Dingen, die man steuern oder auslesen kann. Auch frühere

Über einen OTG-Adapter kann man PC-Peripherie an Tablets und Smartphones anschließen.

Modelle der Saugroboter von Roomba haben eine (versteckte) serielle Schnittstelle. Sogar Gasthermen sollen welche haben. Schauen Sie mal nach.

Octo4A installiert man allerdings nicht über den Google Play Store. Stattdessen lädt man von GitHub eine apk-Datei und installiert diese. Android ist etwas argwöhnisch mit Apps aus fremden Quellen, sodass man explizit seine Erlaubnis für die Installation erteilen muss. Die Installation benötigt 800MByte freien Speicherplatz und läuft weitgehend automatisch. Einzig bei der Wahl des USB-zu-seriell-Treibers muss man eingreifen. Beim Prusa Mini+ ist beispielsweise CDC die richtige Option in der Treiberauswahl, um später in der Oberfläche die Verbindung zwischen Drucker und OctoPrint aufbauen zu können.

Ein Probedruck mit einem Samsung Tablet A6 als Steuerung und einem OTG-Adapter zur Verbindung war erfolgreich. Zum parallelen Laden des Tablets und dem gleichzeitigen Datenverkehr über den Adapter benötigt man noch ein Y-Kabel.

Für CNC-Fräsen mit GRBL-Steuerung (Version 1.1) gibt es im Play Store die App „Grbl Controller". Per USB oder Bluetooth verbindet man das Smartphone mit der Steuerelektronik seiner Fräse, um manuell Positionen anzufahren und ganze GCode-Dateien zu streamen.

Als Display

Insbesondere die großen und scharfen Displays von Tablets lassen einen zögern, die alten Geräte wegzuwerfen. Das Ausbauen führt selten zum Ziel, weil es keine passenden Adapter gibt, um sie etwa an einen Raspberry Pi anzuschließen. Per USB OTG, einem USB Video Grabber für unter 10 Euro und kostenlosen Apps kann man Tablets als Display weiternutzen.

Wir haben das mit unterschiedlich alten Modellen probiert: einem Asus Nexus 7 (2013), einem Samsung Galaxy Tab A6 (2016), einem Haier Pad 971 (2016) und einem Samsung Galaxy Tab S5e (2019).

Auf den Tablets eignet sich die Grabber-Software „USB Camera" (ShenYao) zur Darstellung. Auch „USB Camera Standard" funktioniert zur Anzeige des gegrabbten Bildes. Abgesehen vom Haier funktionierte die Darstellung des Pi-Desktop auf allen oben aufgeführten Tablets.

Beim Galaxy Tab S5e haben wir noch einen Video-Capture-Stick mit USB-C getestet. Auch der funktionierte ohne Probleme. Sogar die Audio-Ausgabe war auf dem Tablet zu hören. Jetzt müsste man nur noch einen Weg zurück finden, um die Touch-Eingaben auf dem Display an den Pi zu senden und so Keyboard und Maus einzusparen. Zusammen mit einem Pi 400 und einer Maus ist es trotzdem eine platzsparende Kombi für unterwegs. *—dab*

Alles zum Artikel im Web unter *make-magazin.de/x5cm*

Ein HDMI-Grabber für unter 10 Euro zeigt auf Android-Geräten das Bild anderer Geräte an.

Der Desktop eines Raspberry Pi auf dem Display eines Nexus 7

Quiz-Buzzer mit RemoteXY

Bei einem Quiz kann es gerne mal drunter und drüber gehen. Selbstgebaute Buzzer mit ESP32 sorgen da schon für etwas Ordnung. Richtig fair wird es aber erst mit einem unparteiischen Spielleiter. Hier kann eine RemoteXY-App helfen, die den aktuellen Spielstand auf einem Smartphone-Bildschirm abbildet und für den typischen Buzzer-Sound sorgt.

von Bernd Heisterkamp

Auf der Suche nach etwas Unterhaltung für das 50-jährige Firmenjubiläum kam mir die Idee, ein Quiz mit Fragen rund um das Ereignis zu organisieren. Wie bei den bekannten Fernsehshows sollten die Teilnehmer auf einen Buzzer hauen, sobald sie die richtige Antwort wissen, und bei korrekter Antwort Pluspunkte erhalten, ansonsten aber einen Punktabzug.

Zunächst war ich verleitet, das Spiel mit großen Not-Aus-Schaltern zu realisieren, aber das hätte den Kostenrahmen für diesen Spaß-Event etwas gesprengt. Neben der Elektronik habe ich mir daher auch Gedanken zum Design eines soliden Schalters gemacht, der mit einem billigen elektronischen Taster realisiert werden kann.

Die Anzeige des aktuellen Punktestands, das Aufleuchten der „Lampe" für den Spieler, der als Erster den Buzzer gedrückt hat, und der Alarm-Sound erfolgen auf einem Mobiltelefon oder Tablet. Das ist per Bluetooth an den ESP32 des ersten Buzzers (Haupt-Buzzer) angebunden. In zwei weiteren Buzzern sind nur Schalter verbaut, die mit dem Haupt-Buzzer verkabelt werden. Die Lösung funktioniert mit Android und iOS.

Schlussendlich hat mich die Lösung weniger als 10 Euro gekostet. Dazu kommen noch die auf Seite 57 erwähnten Gebühren für die Pro-Version von RemoteXY, damit man nicht auf fünf Elemente in seinen Benutzeroberflächen beschränkt ist. Den Quellcode und die Pläne für die Buzzer gibt es im GitHub-Repository des Projekts zum Download.

Daten mit RemoteXY anzeigen

Wie man mit RemoteXY ein Smartphone als kabelloses Display für Mikrocontroller-Projekte verwendet, erklärt der Artikel auf Seite 30. Daher gehe ich hier nur auf die Elemente ein, die ich für dieses Projekt verwendet habe.

Das schmale, rechteckige „LED"-Element über dem Play- und Config-Button (Bild 1), das aussieht wie ein Streifen, wird bei den Element-Details mit einer Blinkperiode von 1250 ms konfiguriert. Sie zeigt später an, dass das Spiel läuft und noch niemand den Buzzer gedrückt hat. Indem man das „Sound"-Element auf den Screen zieht, kann man für den Buzzer bei falschen und richtigen Antworten sowie für ein gewonnenes Spiel entsprechende Klänge hinterlegen. Sichtbar ist dieses Element später auf dem Screen natürlich nicht. Die drei „Linear-Level" dienen der Fortschrittsanzeige und die drei „RGB-LEDs" signalisieren, welcher Spieler den Buzzer zuerst gedrückt hat. Über die beiden Button-Elemente „NOK" und „OK" wählt der Spielleiter später, ob die richtige Antwort gegeben wurde.

Über das „Page switcher"-Element kann man einen zweiten Screen definieren (Bild 2).

Kurzinfo

» **Quiz-Buzzer mit ESP32-Mikrocontroller bauen**
» **Mehrere Buzzer mit RemoteXY und Smartphone steuern**
» **Elektronikprojekt mit Holz**

Checkliste

Zeitaufwand:
6 bis 8 Stunden

Kosten:
20 bis 30 Euro

Material

» **ESP32C3 Xiao** von Seeed Studio
» **3,7-V-Akku**
» **3 Taster,** 12×12 mm
» **3 Federn**
» **3 JST XH 3-Pol-Stecker und -Buchsen**
» **2-adriges Kabel,** 2 m
» **Pinheader mit Jumper** oder Ein-/Ausschalter
» **Multiplex- oder MDF-Platte,** 6 und 12 mm dick
» **Lack** in Farbe nach Wahl
» **Holzleim**

Werkzeug

» **Lötkolben** mit Zubehör
» **Holzsäge, Fräse und / oder 3D-Drucker**

Mehr zum Thema

» Tobias Vogel, Romménator, Make 2/22, S. 38
» Bernd Heisterkamp, Task-Reminder, Make 1/24, S. 8
» Stefan Draeger, Whack-A-Mole mit dem Raspberry Pico, Make 6/22, S. 56

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

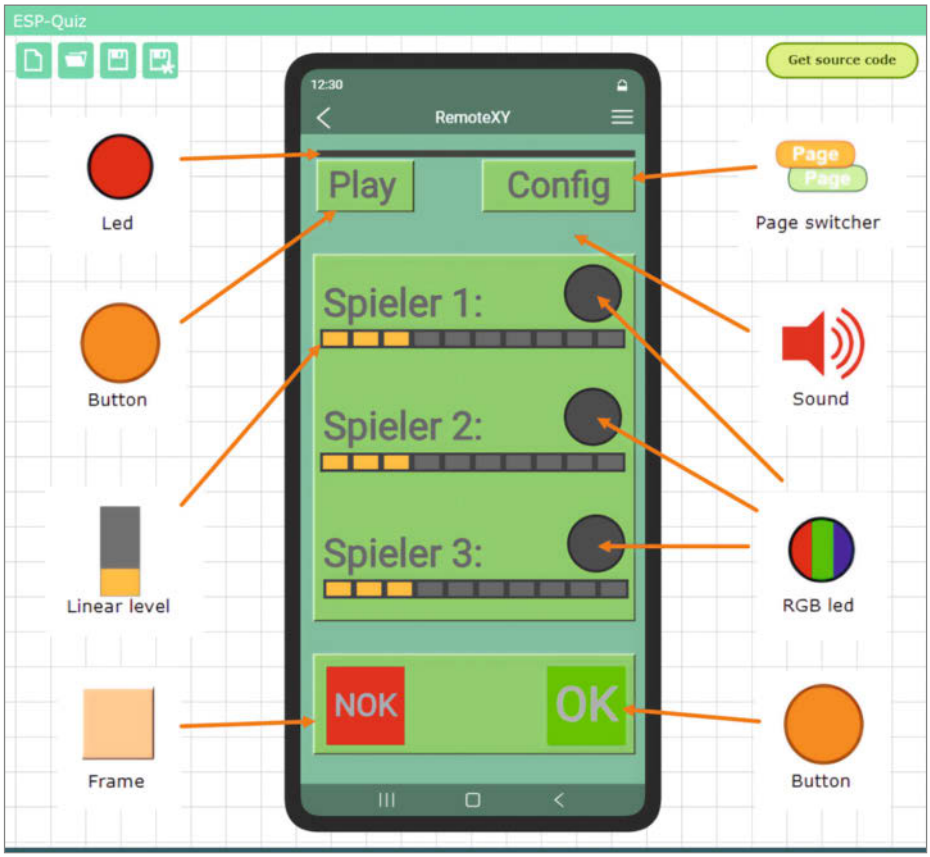


Bild 1: Die in RemoteXY erstellte Spielleiter-App behält den Überblick über die Buzzer und den Punktestand.

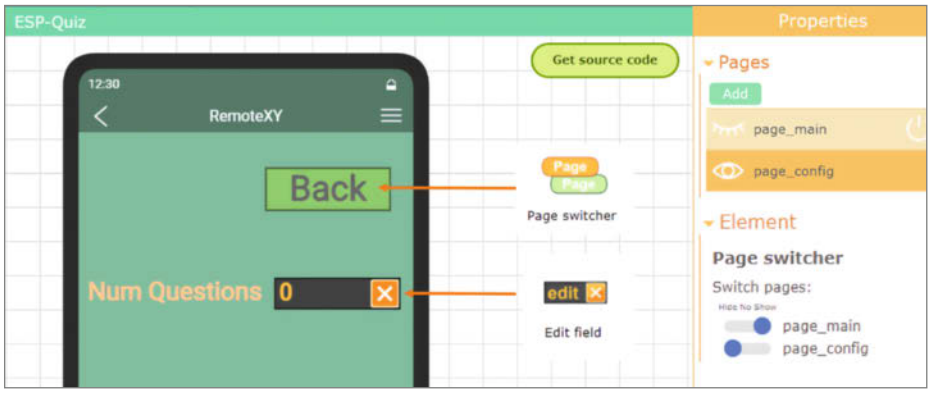


Bild 2: Mit dem „Page switcher“ kann man zwischen verschiedenen Screens wechseln.

Diesen nutze ich zur Konfiguration des Spiels. In dieser ersten Version lässt sich nur die Anzahl der richtigen Fragen einstellen, aber später sollen noch weitere Konfigurationsmöglichkeiten dazukommen. Bei den Element-Details des „Page switcher“ wird eingestellt, welcher Bildschirm beim Klicken des Elements aufgerufen wird.

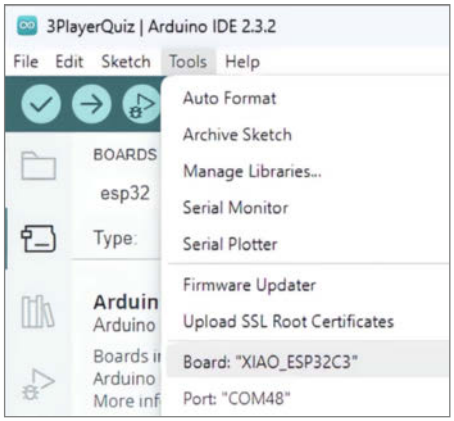


Bild 3: In der Arduino IDE muss man das richtige XIAO-Board auswählen.

ESP32 programmieren

Das verwendete ESP32-Xiao-Board habe ich mit der Arduino IDE programmiert. Dafür muss man im Boards Manager zunächst den Arduino Core für ESP32 (Version 2) installieren (eine Anleitung gibt es über den Link in der Kurzinfo). Danach lässt sich im Menü „Tools“ das Board „XIAO_ESP32C3“ auswählen (Bild 3) und man muss noch den entsprechenden COM-Port einstellen, um mit dem Mikrocontroller kommunizieren zu können.

Das Programm, das die Buzzer steuert, ist wie folgt aufgebaut: Die drei Schalter für die Buzzer sind mit den Eingängen D1 bis D3 verbunden (Bild 4) und in den Zeilen 34 bis 36 definiert:

```
34  #define BUTTON_PIN_PLAYER1 D1
35  #define BUTTON_PIN_PLAYER2 D2
36  #define BUTTON_PIN_PLAYER3 D3
```

Der Listing-Kasten „void loop()“ zeigt die Hauptschleife des Programms. In Zeile 115 wird die Anzahl der Fragen zum Gewinnen des Spiels aus dem Textfeld des Konfigurations-Screens eingelesen. Zeile 121 setzt das Spiel zurück, wenn der Play-Button gedrückt wird, zugleich wird die Variable `gameRunning` auf `True` gesetzt, was bewirkt, dass die Spiel-LED zu blinken beginnt (s.a. Zeile 118). In den Zeilen 125 bis 134 wird ausgewertet, ob einer der drei Buzzer gedrückt wurde, um dann mit der Funktion `handleBuzzerPress()` die entsprechende LED anzuzeigen und weitere Eingaben zu blockieren.

Innerhalb der Funktion `handleBuzzerPress()` wird dann evaluiert, ob der Spielleiter den „OK“-Button (`RemoteXY.button_CorrectAnswer`) oder den „NOK“-Button (`RemoteXY.button_WrongAnswer`) gedrückt hat (siehe Listing „handleBuzzerPress() – gekürzt“). Bei richtiger Antwort erhält der Spieler einen Punkt (Zeile 200), anschließend wird für `1100` ms (Zeile 202) der Sound `1031` (Merope) abgespielt (Zeile 201) und dann wieder ausgeschaltet (Zeile 203). Eine Liste der Sounds und die Möglichkeit, diese auszuprobieren, findet ihr in der RemoteXY-App unter „Settings/Sound list“.

Über den Aufruf der Funktion `UpdateProgress(playerIndex)` wird der Fortschritts-Graph des Spielers aktualisiert, der die Antwort gegeben hat. Innerhalb der Funktion wird dafür sein Punktestand auf einen Wert zwischen `1` und `100` skaliert, dies ist die notwendige Eingabe für die „Linear-Level“-Anzeige von RemoteXY.

void loop()

```
111  void loop()
112  {
113    RemoteXY_Handler ();
114    if (RemoteXY.edit_NoQuestions > 0) {
115      targetScore = RemoteXY.edit_NoQuestions;
116    }
117
118    LEDGameRunning(gameRunning);
119    // Check if new game button is pressed
120    if (RemoteXY.button_newGame == 1) {
121      resetGame();
122      gameRunning = true;
123    }
124
125    if (gameRunning) {
126    // Check if any player has pressed the buzzer
127      if (digitalRead(BUTTON_PIN_PLAYER1) == LOW) {
128        handleBuzzerPress(0);
129      } else if (digitalRead(BUTTON_PIN_PLAYER2) == LOW) {
130        handleBuzzerPress(1);
131      } else if (digitalRead(BUTTON_PIN_PLAYER3) == LOW) {
132        handleBuzzerPress(2);
133      }
134    }
...
139  }
```

Verdrahtung der Schalter

Die Verdrahtung der drei Schalter ist sehr einfach. Sie können, wie in Bild 4 gezeigt, einfach an die Eingänge D1 bis D3 und GND angeschlossen werden. Die Buzzer muss man nicht entprellen und auch Widerstände werden nicht benötigt.

Um die Buzzer autark von Steckdosen nutzen zu können, kann an zwei Pins auf der Unterseite des ESP32-Boards ein Lithium-Akku mit 3,7 V angeschlossen werden. Dieser lässt sich über den USB-Anschluss des ESP laden. Den Ein-/Ausschalter baut man direkt in die Zuleitung des Akkus (Bild 5).

Schalter aus Multiplex-Platten

Der rote Auslöser (Bild 6) wird durch drei Seitenfedern nach oben gegen die Buzzer-Abdeckung gepresst. In dieser Stellung ist die Mittelfeder, die auf den elektronischen Taster drückt, entspannt und schließt den Kontakt nicht. Wird der Buzzer nach unten gedrückt,

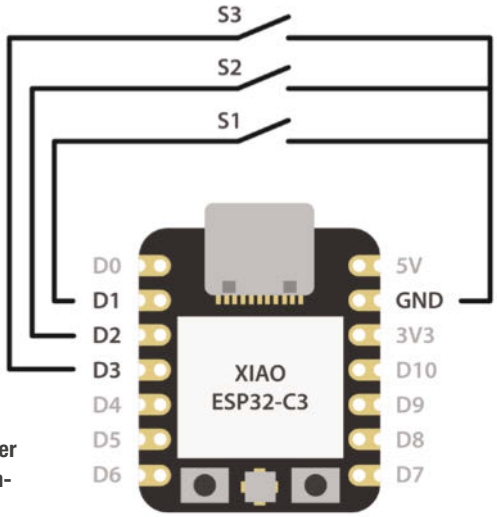


Bild 4: Die Buzzer sind mit den Eingängen D1, D2 und D3 verkabelt.

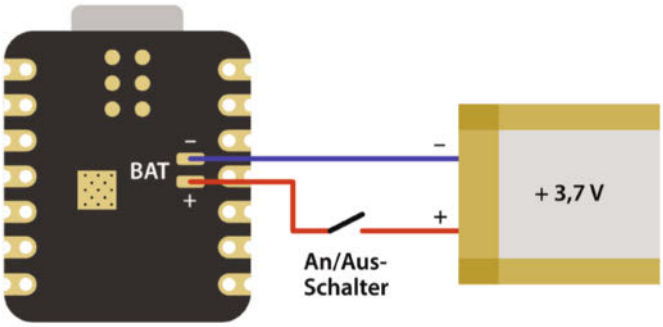


Bild 5: Auf der Rückseite des XIAO-Boards befinden sich zwei Anschlüsse für einen 3,7-V-Akku. Achtet beim Verbinden unbedingt auf die Polarität.

bewegt er sich nur bis zur Basis, die Mittelfeder wird gespannt und löst den Schalter aus, ohne dass zu viel Druck ausgeübt wird. Dadurch kann selbst ein herzhafter Schlag den elektronischen Taster des Buzzers nicht beschädigen. Für den ESP ist ein Hohlraum vorgesehen, um ihn im Inneren des Buzzers zu verstecken.

Zunächst hatte ich die Verwendung eines ESP32 vom Typ Wemos D1 vorgesehen, mich dann aber später für das XIAO-Board entschieden, da es eine Akkuladefunktion besitzt und mehr Platz für die Verdrahtung lässt. Für die Verbindung der drei Buzzer untereinander habe ich in der ersten Version nur Pin-Header verwendet, die sich aber zu schnell lösen. Im aktuellen Modell ist ein dreipoliger Stecker vom Typ JST-XH vorgesehen (Bild 7).

Als An/Aus-Schalter ist ein dreipoliger Pin-Header neben dem ESP befestigt. Werden die oberen beiden Kontakte mit einem Jumper geschlossen, versorgt der Akku den ESP mit Strom. Zum Ausschalten wird der Jumper auf die unteren beiden Pins gesteckt. Alternativ kann man auch einen Mini-Schalter verwenden.

Die elektronischen Komponenten haben durch die Aussparungen bereits einen gewissen Halt, zusätzlich sind sie mit Heißleim fixiert.

Den Buzzer, beziehungsweise seine Basis, den Ring und das Auslöseroberteil habe ich aus 12-mm-MDF gefräst, für das Buzzer-Auslöserunterteil und die Abdeckung habe ich 6-mm-Multiplex verwendet. Mit Ausnahme der Basis sind die Komponenten aber auch leicht mit Säge und Bohrer herzustellen. Wer keine CNC-Fräse, aber einen 3D-Drucker hat, kann die Buzzer-Basis auch drucken. Alternativ ist sicher auch eine Bearbeitung mit einer Oberfräse oder einem Stechbeitel möglich. Im GitHub-Repository findet ihr sowohl die STL-

handleBuzzerPress() – gekürzt

```
158  void handleBuzzerPress(int playerIndex){
...
199    if (RemoteXY.button_CorrectAnswer == 1) {
200      playerScores[playerIndex]++;
201      RemoteXY.sound_01 = 1031;
202      RemoteXY_delay(1100);
203      RemoteXY.sound_01 = 0;
204    } else if (RemoteXY.button_WrongAnswer == 1) {
205      RemoteXY.sound_01 = 1035;
206      RemoteXY_delay(1100);
207      RemoteXY.sound_01 = 0;
208      playerScores[playerIndex]--;
209    }
...
220  }
```

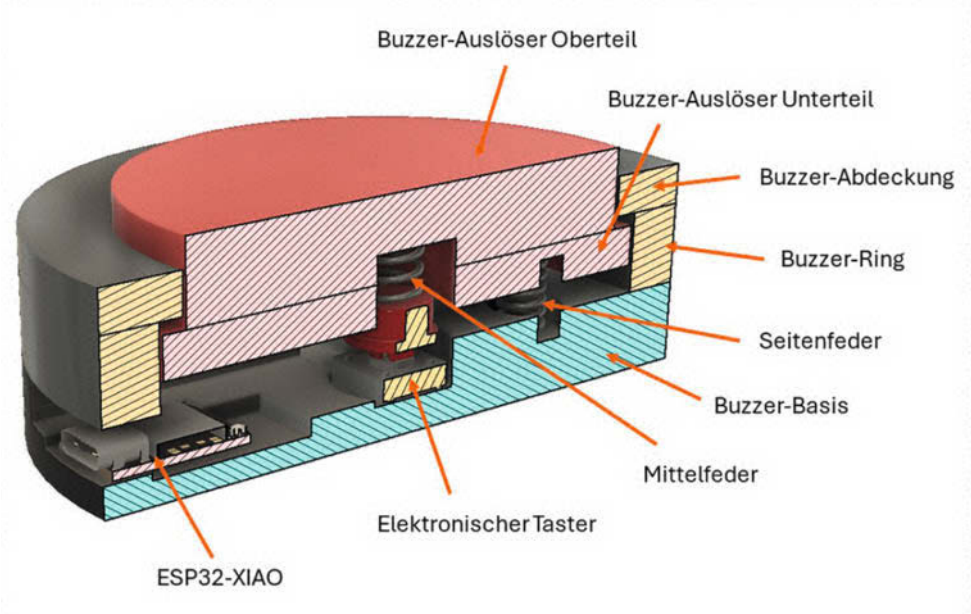


Bild 6: Querschnitt durch den Buzzer

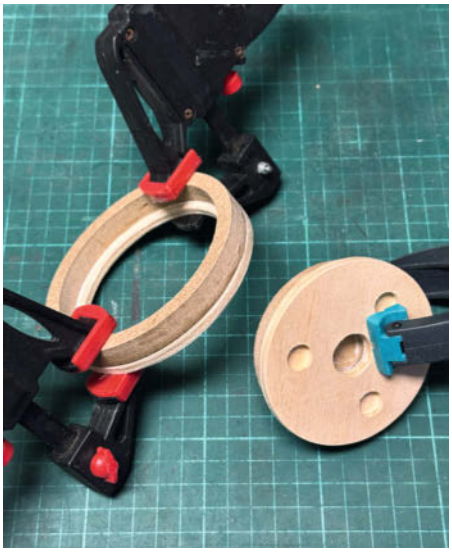


Bild 8: Die Holzteile für den Buzzer müssen gut verleimt werden.

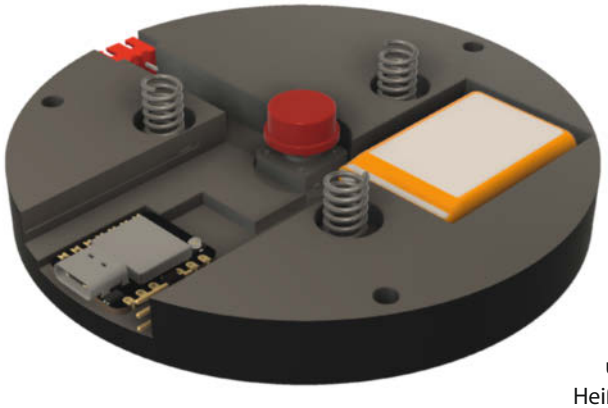


Bild 7: 3D-Skizze der Buzzer-Basis

Dateien als auch die DXF-Zeichnungen der Komponenten.

Für den Zusammenbau leimt ihr das Unterteil und das Oberteil des Auslösers zusammen und den Buzzer-Ring auf die Buzzer Abdeckung. Beide Komponenten müssen dann gut trocknen, bevor der Zusammenbau fortgesetzt wird (Bild 8). Nach dem Verleimen ist es Zeit für etwas Farbe. Ich habe den Auslöser rot und die anderen Teile des Buzzers schwarz lackiert. Das Ergebnis seht ihr in Bild 9.

Bevor der ESP, der Taster und der Steckverbinder mit Heißleim eingeklebt werden, muss man zuerst die Verkabelung verlöten. Der Taster lässt sich direkt mit D2 und GND verbinden, der 3-Pol-Steckverbinder wird dann an D1, D3 und GND gelötet. Die Buzzer 1 und 3 haben keinen ESP, sondern nur den Taster. Sie werden gemäß dem Schema in Bild 4 verkabelt. Auch wenn es dafür nur einen zweipoligen Steckverbinder braucht, empfehle ich, auch diese mit einem 3-poligen Stecker zu versehen, sodass der mechanische Aufbau für alle Buzzer gleich bleibt.

Ist die Verkabelung fertig, werden der ESP, der 3-Pin Header, der Taster und der Steckverbinder in ihre Positionen geklebt. Zudem werden auf den Knopf des Tasters und in die drei Löcher am Rand die Federn geklebt. Diese habe ich aus einem Standardsortiment aus dem Baumarkt. Sie sollten leicht einzudrücken sein. Die richtige Länge findet man am besten durch Ausprobieren heraus, indem man die Federn erst etwas zu lang lässt und sie dann so lange kürzt, bis man den richtigen Druckpunkt gefunden hat. Insbesondere bei der Mittelfeder sollte man messen, ob der Schalter nicht schon in Ruhestellung einen Kontakt herstellt.

Als Letztes muss noch das Verbindungskabel der Buzzer gelötet werden. Vom GND-Pin des 3-Pol-Steckverbinders gehen zwei Kabel auf die Satelliten-Buzzer 1 und 3, die beiden anderen Pins werden mit je einem der Satelliten verbunden. Ist alles vorbereitet, steckt man den Auslöser auf die Basis, dann die Abdeckung darüber und schraubt anschließend von der Unterseite 3-mm-Holzschrauben durch die Basis in den Ring. Nach dem Funktionstest ist das Quiz einsatzbereit (Bild 10).

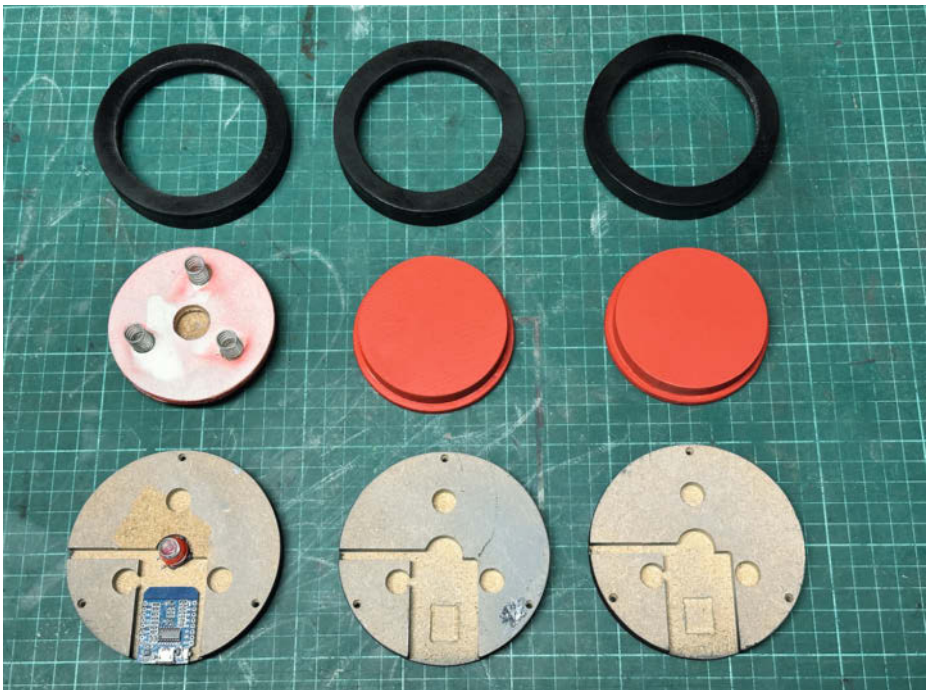


Bild 9: Erste Prototypen der Buzzer sind lackiert und teilweise schon bestückt. Hier noch mit einem Wemos D1 mini und ohne Batteriefach.

Bild 10: Die drei Buzzer: fertig lackiert und zusammengebaut. Jetzt kann das Quiz beginnen.

Praxistest

Der 50. Jahrestag des Hotels, für das ich die Buzzer gebaut habe, hat bereits stattgefunden und das Quiz war sehr beliebt. Ich habe einen Fragenkatalog mit etwa 20 Fragen zu Geschichten, die das Hotel oder den Ort betreffen, zusammengestellt und humorvolle Multiple-Choice Antwortmöglichkeiten dazu gegeben. Dank der Buzzer hatten wir eine Menge Spaß und die Fragen waren Anstoß für viele interessante Gespräche zur Historie des Hotels. *—akf*

Pinguin auf Reisen

Schnell im Zug ein paar Daten aus dem heimischen Gewächshaus abholen und mit einem Python-Skript auf dem eigenen Android-Smartphone auswerten? Das gelingt mit Termux, das ein fast vollständiges Linux-System auf Ihr mobiles Gerät holt – bei Bedarf mit grafischer Benutzeroberfläche.

von Tim Schürmann

3Dalia/Shutterstock.com

Um Linux kommen Maker kaum herum. Es steckt in vielen IoT-Geräten, treibt das Smart Home an und bildet als Standard-Betriebssystem auf dem Raspberry Pi den Ausgangspunkt für unzählige pfiffige Projekte. Dank des fast unerschöpflichen Software-Arsenals dient Linux zudem vielen Makern als Entwicklungsumgebung. So steuern selbstgestrickte Python-Skripte die Bewässerungsanlage im Garten oder klöppeln aus den Daten einer Lärm-Messstation ein hübsches Diagramm. Die besten Ideen für solche Projekte kommen natürlich in der Hoteldusche, auf einer langen Wanderung oder am Strand. Wer den Geistesblitz dann schnell umsetzen möchte, müsste eigentlich immer einen schweren Laptop mit sich führen. Holen Sie sich alternativ einfach ein komplettes Linux-System auf Ihr Android-Smartphone oder Tablet.

Kurzinfo

» **Ein Linux-System unter Android**
» **Zugriff auf Android-Gerätehardware**
» **Bash, Python, C, Go, X Window System unterwegs nutzen**

Checkliste

Zeitaufwand:
2 Stunden

Kosten:
0 Euro

Mehr zum Thema

» Robert Kränzlein, Frühjahrsputz im Smart Home, Make 2/24, S. 56
» Carsten Wartmann, WSL2 für Maker, Make 1/24, S. 94

Alles zum Artikel im Web unter
make-magazin.de/x1zp

Mobiles Linux

Möglich macht das die App Termux. Sie stellt ein einfaches Basissystem bereit, in dem Sie bei Bedarf zahlreiche weitere Linux-Pakete nachinstallieren können. Neben einer grafischen Benutzeroberfläche fallen darunter auch Programmiersprachen wie Python, C und Go. Auf diese Weise verwandelt sich Ihr Tablet-PC in eine vollwertige Entwicklungsumgebung oder eine mobile Datenauswertungsstation. Ebenfalls hinzuholen lassen sich bekannte Netzwerkprogramme, allen voran ein SSH-Client. Über den greifen Sie verschlüsselt auf Ihre Rechner zu Hause zu und starten dort ein eventuell stillstehendes Bewässerungssystem neu.

Des Weiteren stellt Termux spezielle Kommandozeilenprogramme bereit, über die Sie unter anderem Text in die Zwischenablage schieben oder die Kamera ein Foto schießen lassen können. Eingebunden in eigene (Shell-) Skripte lassen sich damit komplett neue Projekte auf Basis des Android-Geräts realisieren. So mutiert etwa ein altes, ausgedientes Smartphone zu einer Wildtier-Beobachtungskamera. Termux selbst ist dabei extrem genügsam: Sie benötigen lediglich ein handelsübliches Smartphone oder ein Tablet mit Android ab Version 5.0. Sie müssen das Gerät weder „rooten", noch anderweitig modifizieren. Um das volle Potenzial von Termux ausschöpfen zu können, müssen Sie sich aber bereits gut mit der Linux-Kommandozeile auskennen.

Arbeitsweise

Termux öffnet ein Fenster zum Linux-System, das in Android unter der grafischen Bedienoberfläche schlummert. Die App arbeitet folglich nur als sogenannter Terminal-Emulator und stellt nicht selbst eine komplette Linux-Umgebung bereit. Einige mitgelieferte Softwarekomponenten sorgen allerdings dafür, dass sich weitere Linux-Anwendungen bequem nachinstallieren und nutzen lassen. Dem setzen die in Android standardmäßig aktiven Sicherheitsmechanismen und Restriktionen aber enge Grenzen. Um sie zu umgehen, müssen die Termux-Entwickler zu einigen Tricks greifen.

So gab sich Termux lange Zeit als eine alte Android-9-App aus, die auch auf neuen Geräten standardmäßig mehr Rechte erhielt. Aus Sicherheitsgründen hat Google jedoch mit jeder Android-Version die Daumenschrauben für Apps immer weiter angezogen. Ab dem Jahr 2020 bot der Play Store die Termux-App nicht mehr auf Geräten an, die mit Android 10 oder höher liefen. Das Termux-Team musste deshalb notgedrungen einen Kompromiss eingehen: Die aktuell im Google Play Store offerierte Termux-Fassung meldet sich jetzt als Android-11-App. Damit lässt sie sich zwar auch auf neuen Android-Versionen installieren, bietet im Gegenzug aber nicht mehr alle Funktionen. So fehlen ihr mehrere Pakete für die grafische Oberfläche (X Window System, X11), einige Linux-Programme arbeiten nicht wie gewohnt und der Zugriff auf die meiste Hardware bleibt verwehrt. Abschließend läuft die Termux-Fassung aus dem Google Play Store nur auf 64-Bit-fähigen Geräten, die mindestens Android 11 antreibt.

Voller Funktionsumfang

Wenn Sie mit den ganzen Einschränkungen nicht leben möchten, können Sie am Play Store vorbei eine unbeschnittene Variante von Termux installieren. Die läuft sogar bereits auf Geräten ab Android 5.0. Ganz wichtig: Sie können die App aus dem Play Store nicht parallel zur anderen Fassung installieren. Sie müssen sich folglich für eine der beiden Versionen entscheiden.

Die Installation der unbeschnittenen Version gelingt auf einem von zwei Wegen: Zum einen stellen die Entwickler die App auf GitHub bereit. Dort müssen Sie allerdings erst die für Ihr Gerät passende Datei finden und diese dann auch noch manuell in Ihr Android-System schieben. Obendrein sind Sie selbst für Updates verantwortlich. Deutlich komfortabler gelingt die Installation über

Abbildung 1: Voller Funktionsumfang mit Termux aus dem alternativen App-Store F-Droid.

Installation via F-Droid

Um Termux über F-Droid einzuspielen, gehen Sie wie folgt vor: Steuern Sie auf Ihrem Mobilgerät mit dem Chrome-Browser die Seite von F-Droid an (siehe Link in der Kurzinfo) und tippen Sie auf „Download F-Droid“. Erlauben Sie das „Herunterladen“ der Datei, lassen Sie sie „Trotzdem herunterladen“ und „Öffnen“ Sie sie. Sollte das nicht gelingen, tippen Sie in der Dateimanager-App Ihres Geräts (häufig als „Files“ oder „Eigene Dateien“ bekannt) die heruntergeladene Datei „F-Droid.apk“ an.

Da die App nicht aus dem Play Store stammt, müssen Sie die Ausführung explizit gestatten. Das gelingt, indem Sie in der erscheinenden Meldung in die „Einstellungen“ wechseln und den dortigen Schalter umlegen. Letztgenannter heißt häufig „Dieser Quelle vertrauen“ oder „Berechtigung erteilen“. „Installieren“ Sie F-Droid und „Öffnen“ Sie die App. F-Droid holt jetzt Informationen zu den in seinem Store verfügbaren Open-Source-Apps, was einige Sekunden dauern kann.

Starten Sie die F-Droid-App, tippen Sie in ihr auf die Lupe und suchen Sie nach „Termux“. In den Ergebnissen wählen Sie den „Termux Terminal-Emulator mit Programmpaketen“ und lassen diese App dann „Installieren“. Auch Termux müssen Sie die Ausführung gestatten. Dazu gehen Sie wieder über die angezeigte Meldung in die „Einstellungen“, aktivieren den Schalter („Dieser Quelle vertrauen“ beziehungsweise „Berechtigung erteilen“) und lassen Termux endgültig „Installieren“.

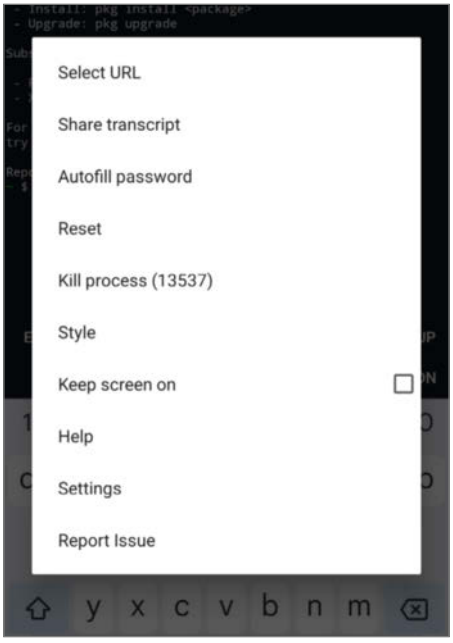


Abbildung 3: „Select URL“ kopiert eine Liste mit allen Internetadressen, die irgendwo in der Ausgabe stehen.

Abbildung 2: Termux zeigt ein paar allgemeine Hinweise und setzt Sie direkt an der Kommandozeile ab.

den alternativen App-Store F-Droid (Abbildung 1). Ihn empfehlen auch die Termux-Entwickler als primäre Bezugsquelle für ihre App. Sofern Sie F-Droid noch nicht nutzen, erklärt der Kasten „Installation via F-Droid“ alle notwendigen Handgriffe.

Bedienung

Bei seinem ersten Start (Abbildung 2) bittet Sie Termux darum, Benachrichtigungen senden zu dürfen. Wenn Sie der App dies gestatten, erscheint umgehend eine solche. In ihr können Sie via „Acquire wakelock“ verhindern, dass sich die App schlafen legt. Doch Vorsicht: Termux nuckelt dann kontinuierlich an der Batterie.

Danach setzt Sie Termux direkt vor eine Kommandozeile. Die funktioniert wie ihr Pendant unter Raspberry Pi OS. Standardmäßig läuft dabei die Bash. Für sie geschriebene Shell-Skripte lassen sich folglich auch innerhalb von Termux starten. Die Schrift vergrößern und verkleinern Sie, indem Sie mit den Fingern eine Zoom-Geste ausführen. Apropos Fingertipp: Wenn Sie Ihren Finger auf einem Text parken, öffnet sich ein Menü, über das Sie ihn in die Zwischenablage kopieren können. Hinter „More …“ verstecken sich zudem weitere hilfreiche Funktionen (Abbildung 3). Unter anderem lässt sich der markierte Text mit anderen Personen über WhatsApp und Co. teilen („Share selected text“).

Befehle tippen Sie über Ihre bekannte Bildschirmtastatur ein. Gegenüber einer richtigen PC-Tastatur fehlen ihr allerdings einige Sondertasten, allen voran Alt und Strg (Ctrl im US-Layout). Insbesondere die letztgenannte Taste benötigt man häufiger unter Linux, etwa um mit der Kombination Strg + C hängende Programme oder Skripte abzuwürgen. Termux offeriert daher diese Sondertasten in einer eigenen zusätzlichen Leiste direkt über der Bildschirmtastatur. Esc, Alt und Co. bleiben auch dann am unteren Bildschirmrand eingeblendet, wenn Sie die Tastatur verstecken.

Auf der Kommandozeile funktioniert die vor allem auf Smartphones nützliche Autovervollständigung: Geben Sie beispielsweise `cl` ein und tippen Sie je nach Termux-Version entweder auf die TAB- oder die Bildschirmtaste mit dem Doppelpfeil. Die Bash ergänzt dann automatisch den Befehl zu `clear`, der den Bildschirm leeren würde. Über die Pfeilsymbole blättern Sie in den zuvor abgesetzten Kommandos, mit dem Finger wischen Sie in den Ausgaben zurück nach oben. Termux merkt sich allerdings standardmäßig immer nur die letzten 2.000 Zeilen. Trotz dieser Hilfen geht das Tippen auf dem Bildschirm nicht so flüssig von der Hand wie auf einer realen Tastatur. Insbesondere wenn Sie längere Skripte schreiben möchten, sollten Sie über die Anschaffung einer (kleinen) Bluetooth-Tastatur nachdenken.

Mirrors wählen

Termux stellt zunächst nur ein minimales Linux-System bereit. In ihm sind zwar Standardwerkzeuge wie Curl und Grep enthalten, es fehlen jedoch viele nützliche Netzwerkprogramme wie ein SSH-Client. Ihn und viele weitere Softwarepakete bietet das Termux-Team in einem eigenen Repository an. Aus dieser Lagerstätte

holt der Paketmanager „pkg“ das benötigte Programm und spielt es ein. Für Linux-Experten: Im Hintergrund spannt der Befehl `pkg` den aus Debian beziehungsweise Raspberry Pi OS bekannten Kollegen Apt ein; die Softwarepakete nutzen ebenfalls das DEB-Format.

Um die Anfragen von vielen Termux-Nutzern besser verteilen zu können, stellen gleich mehrere Server das Repository (Mirrors, also Spiegelserver) bereit. Bevor Sie mit „pkg“ Ihr Lieblings-Tool hinzuholen können, müssen Sie einen dieser Mirrors als Quelle auswählen. Dazu rufen Sie `termux-change-repo` (Abbildung 4) auf. Es erscheint ein textbasiertes Menü, das Sie mit den Pfeiltasten steuern. Die Leertaste wählt aus, die Eingabe bestätigt. Entscheiden Sie sich für die vorausgewählte „Mirror Group“ und wählen Sie im nächsten Schritt die Region, in der Sie sich gerade befinden. Nach der Bestätigung testet „pkg“ die Erreichbarkeit aller Mirrors in dieser Region und wählt einen davon selbstständig aus. Zukünftig zapft „pkg“ übrigens immer wieder einen anderen Mirror aus der Region an. Das verteilt die Last nicht nur besser; fällt einer der Server aus, kann „pkg“ auch zu einem anderen wechseln. Den Befehl „termux-change-repo“ dürfen Sie jederzeit erneut aufrufen und so abhängig von Ihrem aktuellen Aufenthaltsort die Server mit der besten Anbindung wählen.

Softwareinstallation

Sobald „pkg“ die Mirrors kennt, lässt sich endlich der SSH-Client hinzuholen. Ob er überhaupt verfügbar ist, klärt schnell ein:

```
pkg search ssh
```

Dieser Befehl kramt im Repository alle Softwarepakete heraus (Abbildung 5), in denen irgendwo der Begriff „ssh“ auftaucht. Zu jedem gefundenen Paket erhalten Sie eine Kurzbeschreibung, anhand derer Sie schnell das passende Exemplar finden. Der SSH-Client steckt beispielsweise im Paket „openssh“. Um das zu installieren, übergeben Sie seinen Namen an „pkg install“:

```
pkg install openssh
```

Wie auch viele andere Werkzeuge benötigt OpenSSH weitere Softwarepakete. Die installiert „pkg“ automatisch, sobald Sie mit der Eingabe den Startschuss dafür geben. Anschließend können Sie sich via „ssh“ auf dem heimischen Raspberry Pi einloggen und die Gartenbewässerung manuell neu starten.

Nach dem gleichen Prinzip holen Sie beliebige andere Pakete hinzu. Die bei Makern beliebte Skriptsprache Python richtet etwa `pkg install python` ein, der zugehörige Paketmanager „Pip“ steckt im Paket „python-pip“. Um ein Softwarepaket wieder loszuwerden, deinstallieren Sie es mit „pkg uninstall“.

Der Paketmanager holt nicht nur weitere Software hinzu, sondern bringt mittels `pkg upgrade` auch alle installierten Softwarepakete auf den aktuellen Stand. Diesen Befehl sollten Sie möglichst regelmäßig aufrufen, damit keine Sicherheitslücken oder Bugs in der Linux-Software verbleiben.

Hauseigene Tools

Termux bringt mehrere eigene Werkzeuge mit, über die Sie mit dem Android-System interagieren können. Zunächst liefert „termux-info“ verschiedene Informationen über Ihr Gerät, einschließlich der Android-Version; „termux-setup-storage“ (Abbildung 6) wiederum gewährt den Zugriff auf verschiedene Android-Verzeichnisse, wie den Foto-Ordner. Das Tool erstellt im Heimatverzeichnis den Ordner „storage“, in dem symbolische Links zu den entsprechenden Medienverzeichnissen führen. Alle geknipsten Bilder finden Sie beispielsweise im Ordner „~/storage/dcim“.

Termux setzt Sie immer in Ihrem Heimatverzeichnis ab. Dies hat die App extra für Sie angelegt, es ist zu Beginn noch komplett leer. Möchten Sie dort neue Shell- oder Python-Skripte entwickeln, benötigen Sie einen Texteditor. Bereits in Termux enthalten sind nur die wenig komfortablen Klassiker Vi und Nano. Wenn Sie häufiger programmieren, empfiehlt sich deshalb eine auf Android zugeschnittene App. Python-Programmierer können beispielsweise zum kostenlosen Acode greifen. Diese Apps dürfen allerdings nicht auf Ihr Heimatverzeichnis zugreifen. Dies lässt sich mit einem schnellen Handgriff beheben.

Termux bringt mehrere eigene Werkzeuge mit, über die Sie mit dem Android-System interagieren können. Zunächst liefert „termux-

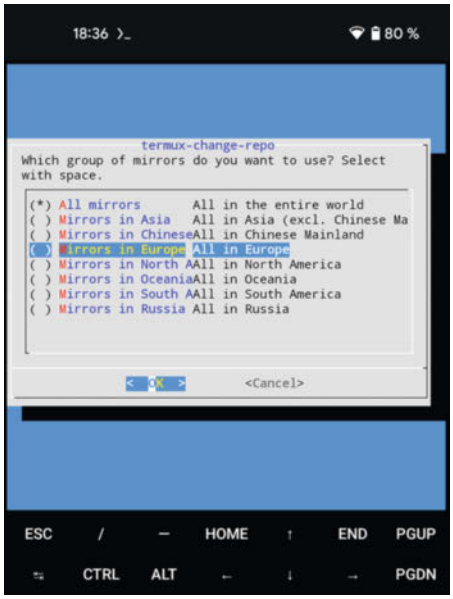


Abbildung 4: Je näher die Mirrors sind, desto besser und schneller ist die Verbindung.

Eingabeprobleme

Einige Bildschirmtastaturen legen unter Termux ein merkwürdiges Verhalten an den Tag. So kommt es mitunter vor, dass Sie den eingetippten Text immer erst mit der Eingabe bestätigen müssen, bevor er in der Kommandozeile landet. Möchten Sie dann nicht gleich die Bildschirmtastatur wechseln, können Sie probeweise die Zeichenverarbeitung in Termux ändern.

Dazu öffnen Sie die Datei „termux.properties“, die in Ihrem Heimatverzeichnis im Ordner „.termux“ liegt (beachten Sie den vorangestellten Punkt im Namen). Sofern das Tippen noch einigermaßen gelingt, können Sie dazu unter Termux den Editor Nano heranziehen: `nano ~/.termux/termux.properties`. Suchen Sie am Ende der Datei die folgende Zeile:

```
enforce-char-based-input = true
```

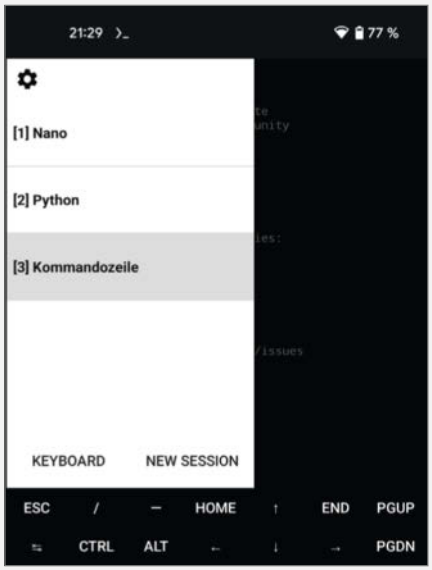
Entfernen Sie das vorangestellte # und speichern Sie die Änderung ab. In Nano gelingt das per Strg + O, Eingabetaste, um den Namen zu bestätigen, gefolgt von Strg + X, um Nano zu beenden. Starten Sie anschließend Termux einmal komplett neu.

Die Einstellung funktioniert vielfach, aber nicht bei allen Bildschirmtastaturen. Abhilfe schafft dann nur noch der Wechsel zu einer anderen Bildschirmtastatur. Das ist auch dann notwendig, wenn Ihre Tastatur unter Termux ein falsches Layout zeigt.

Abbildung 5: Stöbern im Softwareangebot

Sitzungspräsident

Wenn Sie irgendwann den Bedarf nach weiteren Terminal-Fenstern verspüren, halten Sie Ihren Finger an den linken Bildschirmrand. Es erscheint eine Schublade, die Sie nach rechts aufziehen können. Per „New Session" erstellt Termux eine weitere separate Sitzung. In dieser können Sie beispielsweise einen Editor öffnen, während Sie in der anderen Ihr selbstgestricktes Skript aufrufen. Um zu einer anderen Sitzung zu wechseln, ziehen Sie wieder die Schublade auf und tippen auf die entsprechende Sitzungsnummer. Halten Sie den Finger auf eine Sitzung, können Sie dieser einen sprechenden Namen verpassen.



Termux merkt sich alle Sitzungen und stellt sie nach einem Neustart wieder her. Per „exit"-Befehl beenden Sie die gerade angezeigte Sitzung. Wenn Sie Termux das Senden von Benachrichtigungen erlaubt haben, meldet die App stets die Anzahl der derzeit geöffneten Sessions. Tippen Sie dort auf „Exit", schließt Termux sämtliche Sitzungen.

Sollten Sie Ihr Linux-System irgendwann einmal kaputt-konfiguriert haben, etwa mit einer fehlerhaften „~/.bashrc", ziehen Sie die Schublade auf, halten Sie den Finger auf „New Session" gedrückt und wählen Sie „Failsafe" aus. Termux öffnet jetzt eine Sitzung mit Standardwerten, aus der heraus Sie Ihr Linux-System reparieren können.

info" verschiedene Informationen über Ihr Gerät, einschließlich der Android-Version. „termux-setup-storage" wiederum gewährt den Zugriff auf verschiedene Android-Verzeichnisse, wie etwa den Foto-Ordner. Das Tool erstellt im Heimatverzeichnis den Ordner „storage", in dem wiederum symbolische Links zu den entsprechenden Medienverzeichnissen führen. Alle geknipsten Bilder finden Sie beispielsweise im Ordner „~/storage/dcim".

Datensicherung

Auf diesem Weg können Sie nicht nur recht elegant Dateien zwischen Ihrem Linux-System und anderen Android-Apps wie Acode austauschen, sondern auch ein Backup Ihres Linux-Systems anlegen. Dazu kopieren Sie Ihre persönlichen Dateien etwa nach „~/storage/downloads". Den kompletten Rest Ihrer Linux-Umgebung, einschließlich aller installierten Softwarepakete, sichert das Tool „termux-backup":

```
termux-backup \
    ~/storage/downloads/backup.tar.xz
```

Dieses Backup stellt „termux-restore" bei Bedarf dann wieder her:

```
termux-restore /sdcard/backup.tar.xz \
    ~/storage/downloads/backup.tar.xz
```

Auf diesem Weg lässt sich auch elegant die komplette Linux-Umgebung auf ein anderes Android-Gerät übertragen. Doch Obacht: Abhängig von den installierten Softwarepaketen kann der Sicherungsvorgang mehrere Minuten dauern und ein recht großes Archiv hervorbringen.

Add-ons

Über weitere Termux-Werkzeuge steuern Sie die im Android-Gerät verbaute Hardware. Unter anderem lesen Sie mit ihnen den Füllstand des Akkus aus (Abbildung 7), lauschen am Mikrofon oder schießen ein Foto mit der eingebauten Kamera. Dazu benötigen die Tools allerdings die Hilfe einer App namens „Termux:API". Sie vermittelt den Termux-Werkzeugen den Zugriff auf die Hardwarekomponenten Ihres Geräts. Die App steht derzeit allerdings nicht für Nutzer der Termux-Fassung aus dem Google Play Store bereit. Wenn Sie hingegen Termux über F-Droid bezogen haben, öffnen Sie die F-Droid-App, suchen Sie nach „Termux" und installieren Sie die App „Termux:API, Über Termux auf Android-Funktionen zugreifen". Da diese App den Funktionsumfang von Termux erweitert, bezeichnet sie das Termux-Team auch als Add-on.

Anschließend müssen Sie innerhalb von Termux noch die eigentlichen Werkzeuge hinzuholen. Die stecken im Paket „termux-api", das der Befehl `pkg install termux-api` einspielt. Ein Foto im JPEG-Format mit dem Dateinamen „foto1.jpg" knipst dann der Befehl:

```
termux-camera-photo foto1.jpg
```

Beim ersten Aufruf vieler Termux-Werkzeuge müssen Sie zunächst explizit den Zugriff auf die entsprechenden Hardwarekomponenten gestatten. Die Funktion `termux-camera-photo` fordert etwa den Zugriff auf die Kamera. Die Werkzeuge warten dabei nicht Ihre Antwort ab, sondern beenden sich einfach. Rufen Sie dann nach erteilter Erlaubnis das Werkzeug einfach noch einmal auf.

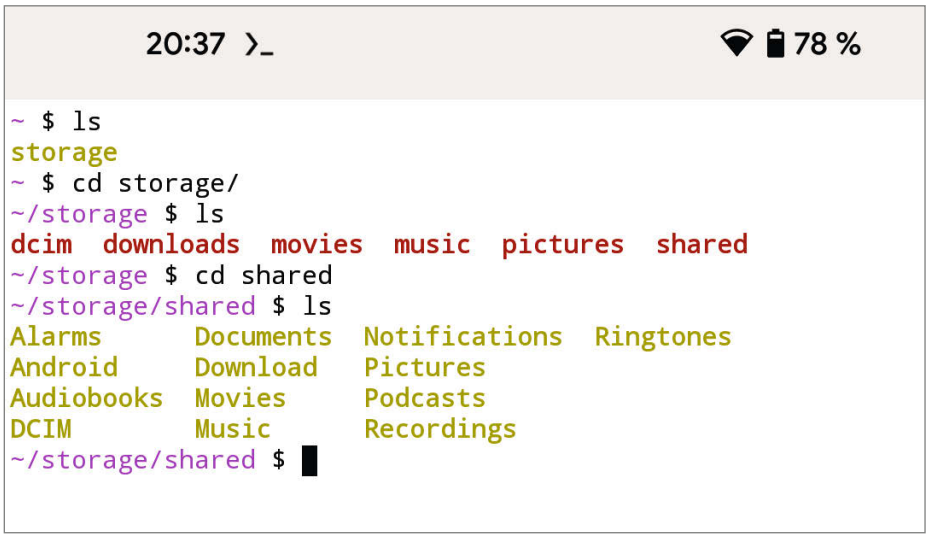


Abbildung 7: Android-Dateisystem.

Skript-freundlich

Sämtliche Termux-Werkzeuge lassen sich leicht in eigene Shell-Skripte einbinden. So ermittelt etwa das folgende Kommando, wie viele Personen in den Kontakten „Tim" heißen:

```
termux-contact-list | grep "Tim" | wc -l
```

Der Befehl `termux-contact-list` holt dabei sämtliche Kontaktdaten aus dem Adressbuch, aus denen `grep` alle Einträge mit dem Wort `Tim` herausfiltert, während `wc -l` die von `grep` ausgegebenen Zeilen zählt.

Einige weitere Termux-Tools lesen Sensoren aus oder liefern Hardwareinformationen. So verrät etwa „termux-camera-info" einige Daten über die verbaute Kamera, während „termux-battery-status" den Batteriezustand ausspuckt. Wie auch „termux-contact-list" verpacken diese Tools ihre Ausgaben in das JSON-Format, das sich in (Python-)Skripten leicht weiterverarbeiten lässt.

Die Vorstellung sämtlicher Termux-Befehle und ihrer Möglichkeiten würde einen eigenen Artikel füllen. Eine vollständige Liste samt einer Beschreibung der einzelnen Tools finden Sie im Termux-Wiki (siehe Link in der Kurzinfo).

Grafische Anwendungen

Unter Termux können Sie auch grafische Linux-Anwendungen starten. Bis die auf dem Bildschirm erscheinen, sind allerdings zahlreiche Handgriffe notwendig. Überdies unterstützt Termux derzeit nur das X Window System (X11). Mit dessen Nachfolger Wayland kommt die App noch nicht zurecht. Alle für das X Window System notwendigen Komponenten sammeln die Termux-Macher in einem eigenen Repository. Dies aktivieren Sie über den Befehl `pkg install x11-repo`.

Termux selbst gibt als Terminal-Emulator nur reine Texte aus. Sie müssen daher die gleich startenden X11-Anwendungen auf einem anderen Weg auf den Bildschirm holen. Am einfachsten gelingt das mit zwei weiteren Komponenten: Innerhalb von Termux leitet die Software TigerVNC die Bilder an die App RealVNC Viewer auf Android weiter. Die wiederum zeichnet den Desktop auf den Bildschirm und meldet alle Eingaben und Wischgesten an TigerVNC zurück. Die Kommunikation zwischen dem Duo erfolgt mit dem für diese Zwecke entwickelten VNC-Protokoll. Anstelle des RealVNC Viewer können Sie folglich auch einen anderen VNC-Client zur Anzeige verwenden.

VNC-Server und Window Manager

Um alle genannten Komponenten in Betrieb zu nehmen, installieren und starten Sie zunächst TigerVNC in Termux:

```
pkg install tigervnc
vncserver -localhost
```

Denken Sie sich jetzt ein Passwort mit mindestens fünf Zeichen aus und tippen Sie es ein. Ihre Eingabe zeigt TigerVNC nicht an, Sie müssen das Passwort folglich „blind" eingeben. Damit sich keine Tippfehler einschleichen, geben Sie das Passwort anschließend noch einmal ein. Das so festgelegte Passwort verlangt TigerVNC von jedem VNC-Client, der sich mit ihm verbinden möchte. Die abschließende Frage beantworten Sie mit einem „n" für Nein. In den Ausgaben von TigerVNC finden Sie jetzt eine Zeile der Art:

```
New 'localhost:1 ()'
            desktop is localhost:1
```

Merken Sie sich die dort hinter „localhost" angegebene Zahl (die Displaynummer). Wie hier im Beispiel sollte es in den meisten Fällen die „1" sein. Diese Ziffer setzen Sie jetzt in den folgenden Befehl ein und rufen ihn auf:

```
export DISPLAY=":1"
```

Des Weiteren benötigen Sie noch einen Window-Manager, der die X11-Fenster verwaltet. Termux unterstützt unter anderem Fluxbox und Openbox. Die beiden arbeiten ressourcenschonend und eignen sich vor allem für kleine Smartphone-Schirme, wobei Flux-

Programmabbrüche

Android ab Version 12 beobachtet laufende Apps und unterbricht unter Umständen von Termux gleichzeitig gestartete Prozesse (sogenannte Phantom-Prozesse) sowie Anwendungen, die eine zu hohe Rechenleistung verlangen. Die betroffenen Programme brechen dann einfach ab, häufig gefolgt von der Meldung „[Process completed (signal 9) - press Enter]". Wenn Sie von diesem Problem betroffen sind, ist eine Lösung nur mit Android-Kenntnissen und der Lektüre eines ellenlangen Artikels im Termux-Wiki (Link über den Kurzlink zum Artikel) zu umschiffen. Der einfachere Weg besteht darin, Ihre Skripte ressourcenschonender zu gestalten, beziehungsweise weniger Programme oder Prozesse gleichzeitig zu starten.

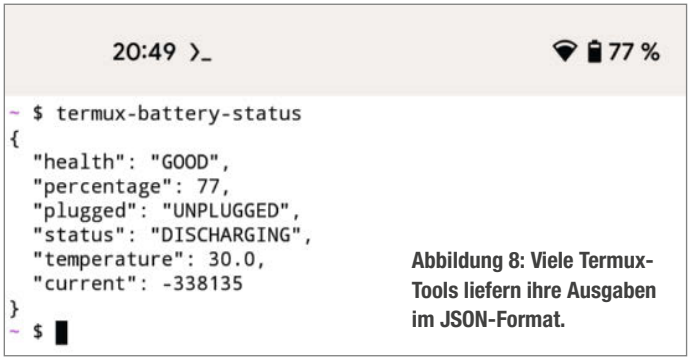


Abbildung 8: Viele Termux-Tools liefern ihre Ausgaben im JSON-Format.

Abbildung 9: Nach diesen drei Befehlen läuft ein X Window System mit Fluxbox.

box ein wenig einfacher einzurichten ist: Holen Sie den Window-Manager per

```
pkg install fluxbox
```

auf Ihr System und starten Sie ihn über die folgenden zwei Befehle:

```
fluxbox-generate_menu
fluxbox
```

Die Fehlermeldungen (Abbildung 8) können Sie ignorieren. Soll Fluxbox zukünftig automatisch mit TigerVNC starten, stellen Sie sicher, dass in der Textdatei „~/.vnc/xstartup" nur die folgenden Zeilen stehen:

```
#!/data/data/com.termux/\
                files/usr/bin/sh
fluxbox-generate_menu
fluxbox &
```

Auf Tablets mit etwas größeren Bildschirmen können Sie auch eine vollwertige Desktop-Umgebung wie Mate, LXQt, und Xfce einspielen – Letztgenannte holen Sie beispielsweise per „pkg install xfce4" hinzu. Ihre Einrichtung erfordert allerdings den Eingriff in verschiedene Konfigurationsdateien; Sie müssen sich folglich mit der gewählten Desktop-Umgebung hervorragend auskennen.

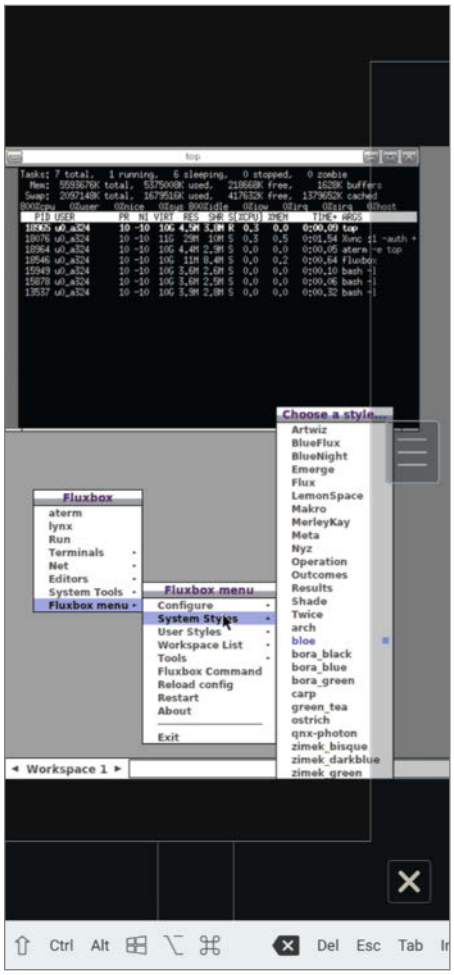


Abbildung10: Die Zonen am unteren Rand ersetzen einen Links-, Mittel- und Rechtsklick.

VNC-Client einrichten

Als letzte Komponente fehlt noch der RealVNC Viewer, den Sie aus dem Google Play Store installieren. Starten Sie die App, tippen Sie sich durch die Begrüßungsbildschirme und ignorieren Sie die immer wieder geforderte Anmeldung („Sign in"). Tippen Sie stattdessen auf das grüne Plus-Symbol rechts unten und erstellen Sie so eine neue Verbindung. Geben Sie als „Address" den Wert „127.0.0.1:5901" ein. Damit kontaktiert der RealVNC Viewer gleich das aktuelle Gerät (das immer unter der IP-Adresse „127.0.0.1" zu erreichen ist) und klopft dort am Port 5901 an. Dahinter wartet wiederum TigerVNC. Eventuell müssen Sie die Port-Nummer anpassen: Addieren Sie zu 5900 die vorhin gemerkte Ziffer (die Displaynummer). Im Beispiel war dies die „1", womit sich 5901 als korrekter Port ergibt. Vergeben Sie abschließend noch einen Namen wie „Termux", tippen Sie auf „Create" und bauen Sie die Verbindung per „Connect" auf. Die Warnung vor einer fehlenden Verschlüsselung ignorieren Sie per „Ok". Der RealVNC Viewer fragt Sie jetzt nach dem Passwort, das Sie bei der Einrichtung von TigerVNC vergeben haben.

Nach dem optionalen RealVNC-Tutorial sollte auf dem Bildschirm der Desktop mit Fluxbox erscheinen. Passt der Desktop nicht vollständig auf den kleinen Smartphone-Schirm, zeigt der RealVNC Viewer (Abbildung 9) nur einen Teil an. Sobald Sie mit dem Zeiger an den Rand stoßen, scrollt der Schirm in die entsprechende Richtung. Über die simulierte rechte Maustaste holen Sie ein Menü mit den bereits vorinstallierten X11-Anwendungen hervor; über die mittlere Maustaste verwalten Sie die virtuellen Arbeitsflächen (Workspaces).

Rechnen Sie damit, dass sich vor allem auf Smarpthones viele grafische Anwendungen nur fummelig bedienen lassen. Sinnvoll ist die Nutzung von X11-Anwendungen nur auf größeren Tablets oder zur reinen Anzeige von Diagrammen und Grafiken. Möchten Sie das X Window System nach einem Neustart von Termux wieder hochfahren, rufen Sie direkt `vncserver -localhost`, dann `export DISPLAY=":1"` und gegebenenfalls Fluxbox auf. Anschließend können Sie sich wieder mit dem VNC-Client verbinden.

Stolperfallen

Aus Sicherheitsgründen dürfen Android-Apps nur in ausgewählte Verzeichnisse schreiben. Termux parkt daher sämtliche Programme, Werkzeuge und Bibliotheken unterhalb von „/data/data/com.termux/files/usr". Alle in Ihrem eigenen Heimatverzeichnis abgelegten Dateien landen hingegen im Ordner „/data/data/com.termux/files/home". Da Termux alles in den beiden Verzeichnissen verstaut, fehlen folglich die auf einem normalen Linux-System vorhandenen Ordner „/usr", „/home" und übrigens auch „/tmp", in dem viele Linux-Programme temporäre Daten ablegen. Des Weiteren verhindert Android den Zugriff auf einige systemnahe Schnittstellen unter „/proc". Hierdurch funktionieren Netzwerkprogramme wie „htop" nicht korrekt. Wenn Sie Shell-Skripte oder eigene Programme für Termux schreiben, müssen Sie übrigens die langen Pfade nicht immer abtippen. Sie finden sie auch stets in den Umgebungsvariablen „$PREFIX" und „$HOME".

Und es gibt noch einen wichtigen Unterschied zu den meisten Linux-Systemen: Termux stellt kein Mehrbenutzersystem bereit. Aus diesem Grund müssen Sie sich beim Start der App auch nicht mit Benutzernamen und Passwort anmelden, die Konfigurationsdatei „/etc/passwd" ist leer. Es existiert auch kein Benutzer „root", Sie selbst sind in Ihrem System allmächtig und dürfen an allen Schrauben drehen – und es somit sogar jederzeit unbrauchbar machen. Auch sämtlichen in Termux bereitstehenden Programmen fehlen die Mehrbenutzerfunktionen. Unter anderem lässt sich nicht das sogenannte Setuid-Bit setzen. Sie selbst besitzen dennoch einen Benutzernamen, den „whoami" verrät. Er folgt dem Schema „u0_a123" und ist der gleiche Name, unter dem auch die Termux-App läuft.

Alle genannten Einschränkungen müssen Sie vor allem dann im Hinterkopf behalten, wenn Sie fertige Skripte oder Codeschnipsel übernehmen wollen. Führen Sie insbesondere keine Skripte und Programme aus unbekannten Quellen aus. Durch die weitreichenden Rechte können diese im und aus dem Linux-System heraus Schaden anrichten, etwa indem sie es zu einer Spam-Schleuder degradieren oder das Verzeichnis „/data/data/com.termux/files/usr" zerstören.

Bastelfreude und Linux-Luft

Termux verlangt von seinen Anwendern neben Bastelfreude mitunter auch etwas Frustresistenz. So ist die Eingabe längerer Befehle auf kleinen Bildschirmen mühsam, das X Window System läuft nur stotternd und die Linux-Umgebung weicht notgedrungen von einer vollwertigen Linux-Distribution ab. Dennoch erhält man mit Termux ein kleines, individuelles, flexibles und erstaunlich leistungsfähiges Linux-System. So steht etwa eine vollständige Python-Umgebung bereit. Als Linux-affiner Maker möchte man die App daher vor allem unterwegs nicht mehr so schnell missen. —*caw*

» Continuous Lifecycle »

[Container Conf]

13./14. November 2024
Mannheim

Die Konferenz für Developer Experience, Platform Engineering und mehr

Die CLC setzt 2024 rund um das Container-Ökosystem Themenschwerpunkte zu KI-gestütztem **DevOps, Security** und **FinOps** sowie **Nachhaltigkeit.**

Highlights aus dem Programm:

- Pipeline als Produkt denken: Modularisierung, Versionierung, Testen, …
- Der KI-gestützte Entwickler – Tools, Datenschutz, Mindset
- Sicherheitsrisiken von CI/CD-Systemen erkennen und vermeiden
- Nachhaltigkeit in der Cloud – Herausforderungen meistern
- Praxisbericht: Ressourcen reduzieren und Cloud-Kosten senken

Jetzt Tickets sichern!

Workshops am 12. November **continuouslifecycle.de**

Veranstalter

Gold-Sponsoren

Silber-Sponsoren

Handy als Steuerung für Bandmaschine

Auch als Retrofan kann man die Vorzüge einer Fernbedienung für die ordentlich instand gehaltene A77-Bandmaschine nicht abstreiten. Mit einem alten WLAN-fähigen Smartphone oder Tablet, einem WEMOS und ein paar Relais kann man ohne Modifikation des Tonbandgerätes eine moderne Bedienung bauen.

von Hans Borngräber

Für die Revox A77 existieren bereits Fernbedienungen. Allerdings handelt es sich bei diesen um Bedienelemente, die mit langen Kabeln am Gerät angeschlossen sind. Mit etwas Maker-Geschick lässt sich das aber auch in moderner Form kabellos umsetzen.

Man nehme ein altes Handy oder Tablet und schaue, ob man auch ohne SIM-Karte ins heimische WLAN kommt und ob der Webbrowser noch funktioniert. Wenn das klappt, hat man schon mal die Hardware und die letztendliche Fernbedienung. Der zweite Teil ist etwas anspruchsvoller und erfordert Kenntnisse über Elektronik und über Arduino- und ESP8266-Programmierung. Wie man die A77 ins WLAN bringt und wie die Fernbedienungsschnittstelle der A77 funktioniert, wird im folgenden Artikel erklärt.

Kurzinfo

- **Retrotechnik mit Smartphone fernsteuern**
- **Relais mit WEMOS steuern**
- **WEMOS über WLAN bespielen**

Checkliste

Zeitaufwand:
1 Tag

Kosten:
50 Euro

Material

- **SMD-Bauteile** Stückliste online
- **Kabel**
- **WEMOS D1 mini**
- **OLED-Display** Typ SSD1306
- **5 Relais** Typ HFD3/5

Werkzeug

- **Seitenschneider**
- **Feinlötkolben** mit Lötzinn

Mehr zum Thema

- Hans Borngräber, Revox-Bandmaschinen reparieren, Make 6/23, S. 82
- Hans Borngräber, Häufig defekt, Make 6/23, S. 132

Alles zum Artikel im Web unter *make-magazin.de/x9zr*

Die Fernbedienungsschnittstelle der A77

A77-Besitzern ist mit Sicherheit schon mal der schwarze Stecker an der Rückseite des Tonbandgerätes aufgefallen. Dieser ist ohne Kabelanschluss und beinhaltet eine Drahtbrücke (Pin 1-2). Fehlt dieser Stecker, läuft die A77 nicht mehr, denn ohne die Drahtbrücke ist die Maschine dauerhaft gestoppt.

Es handelt sich dabei um einen WIST10-Stecker der Firma Hirschmann. Er wird schon seit Jahren nicht mehr produziert, ist aber für mittlere zweistellige Beträge noch beschaffbar. An die Buchse wird nach dem Entfernen des Steckers die Fernbedienung angeschlossen. Das gilt sowohl für die originale als auch die selbst gebaute Version (die originale kommt natürlich mit einem eigenen Kabel). Im Projekt ist eine Platine dokumentiert, die den Stecker nachbildet und mit einem 10-poligen Flachkabel verdrahtet ist. Denn der WIST10-Stecker ist durch sehr wenig Platz zwischen den Pins extrem schlecht zu verlöten.

Systemübersicht

Wie arbeitet die moderne Technik jetzt mit dem Tonbandgerät zusammen? Auf dem Handy läuft nur der Webbrowser, keine App. Dadurch ist die Fernsteuerung plattformagnostisch und kann sowohl auf Android und iOS als auch auf dem PC bedient werden. Durch den Verzicht auf Javascript und HTML5 funktioniert diese Webseite auch auf älteren Endgeräten noch tadellos. Die Bedienung erfolgt dann entweder per Touch oder per Maus. Am Tonbandgerät selbst werden über Relais die Kommandos an die A77 weitergegeben.

Webserver und Programmierwerkzeuge

Als Webserver kommt ein WEMOS D1 mini mit dem Standard Arduino-ESP8266WebServer zum Einsatz. Eine Anleitung, wie man den WEMOS D1 mini in die Arduino IDE einfügt, verlinken wir in der Kurzinfo. Bilder für das UI der Webseite werden im LittleFS-Dateisystem der Arduino-Umgebung abgelegt, um Speicherplatz zu sparen. Was man machen muss, um das LittleFS-Dateisystem in die Arduino IDE zu integrieren, wird im Kasten „LittleFS-Plug-in“ genau erklärt.

Die Bilder für die grafische Bedienoberfläche der Fernsteuerungs-Website werden erst während der Ausgabe der Seiten in den HTML-Code eingefügt. Sie müssen im Base64-Format formatiert sein. Einen Link zu einem passenden Online-Konverter befindet sich in der Kurzinfo.

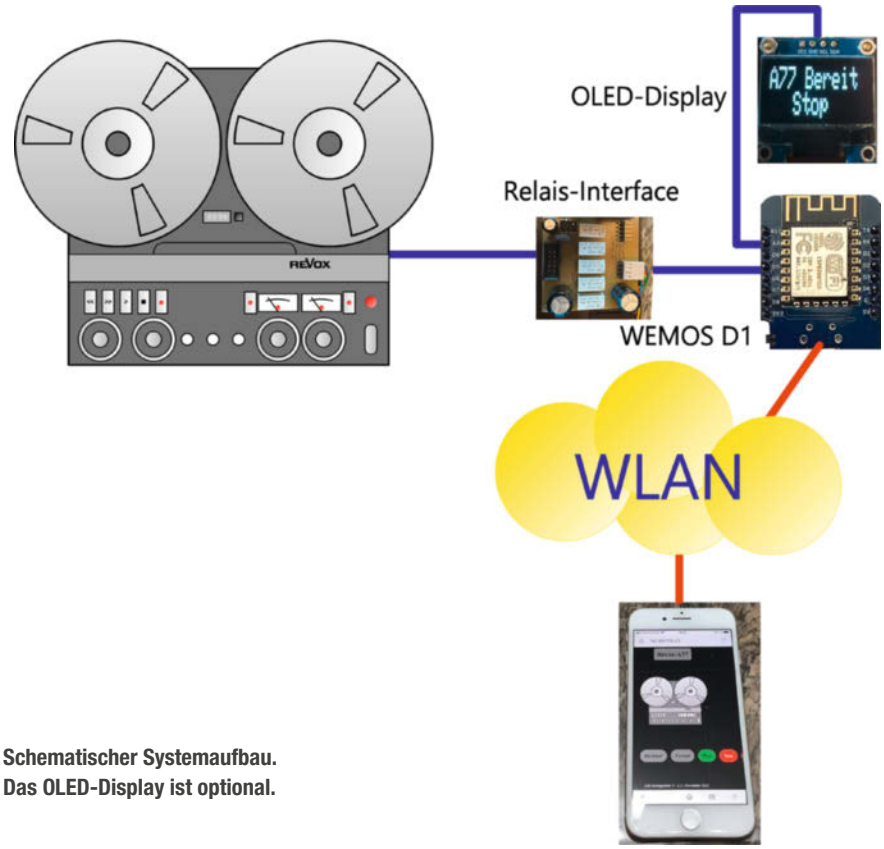
Der HTML-Code für die Webseite wurde mit dem NVU-HTML-Editor erstellt. Dieser Editor ist so alt, dass der erzeugte Code zu allen Browsern passt (Abwärtskompatibilität). Egal ob Smartphone, Tablet, PC etc. – Hauptsache, es läuft ein Webbrowser darauf. Auch der ESP8266WebServer hat mit diesem Code keine Probleme. Einen Link zum Download findet man in der Kurzinfo.

Der vom NVU-Editor erzeugte HTML-Code wird mit der Arduino IDE in den Programmcode der Steuerung eingefügt. Der komplette Code ist im Download-Paket (das in der Kurzinfo verlinkt ist) enthalten, Abtippen ist also nicht notwendig.

Für Software-Updates ist ARDUINO OTA (**O**ver **t**he **A**ir) im Code enthalten. Das heißt, es können über die WLAN-Verbindung neue Programmversionen eingespielt werden. Eine Kabelverbindung ist nur einmalig am Anfang notwendig. Später wird sie nicht mehr benötigt.

Das funktioniert, indem man in der Arduino IDE (nach dem ersten Übertragen des Codes) unter „Tools/Port“ einen Netzwerk-Port auswählt, unter dem das Board angezeigt wird. Eine genaue Beschreibung ist in der Kurzinfo verlinkt.

Oben: Originalstecker, unten: Platinennachbau



Schematischer Systemaufbau. Das OLED-Display ist optional.

LittleFS-Plug-in

Das LittleFS-Plug-in ermöglicht es, Daten in den Speicher eines Mikrocontrollers zu schreiben. Um diese Funktion nutzen zu können, muss man das Plug-in manuell installieren. Dafür geht man als Erstes auf das GitHub-Repository des Arduino-littlefs-upload-Projekts (Link in der Kurzinfo) und lädt dort die Datei arduino-littlefs-upload-1.1.8.vsix herunter. Die Zahlenfolge am Ende (1.1.8) kann bei einem neuen Release abweichen.

Diese Datei muss man jetzt in den plugins-Ordner der Arduino IDE kopieren. Das ist normalerweise C:\Users\<BENUTZERNAME>\.arduinoIDE\plugins. Sollte der plugins-Ordner nicht existieren, muss man ihn anlegen.

Nachdem das Plug-in in diesen Ordner kopiert und die IDE neu gestartet wurde, kann man mit der Tastenkombination Strg+Umschalt+P eine Suchleiste öffnen und dort mit „little“ das Plug-in finden.

Beim Aufrufen des LittleFS-Eintrages sucht das Programm nach einem Ordner mit dem Namen Data im aktuellen Sketch-Verzeichnis. Die Dateien, die in diesem Ordner gespeichert sind, werden dann von der IDE auf den Mikrocontroller überspielt. Das muss vor dem Übertragen des normalen Programmcodes passieren.

Für ältere IDE-Versionen funktioniert die Installation des Plug-ins etwas anders. Wer eine veraltete Arduino-IDE nutzt, findet in der Kurzinfo einen Link zu einer Installationsanleitung.

In der Suche findet man das Plug-in.

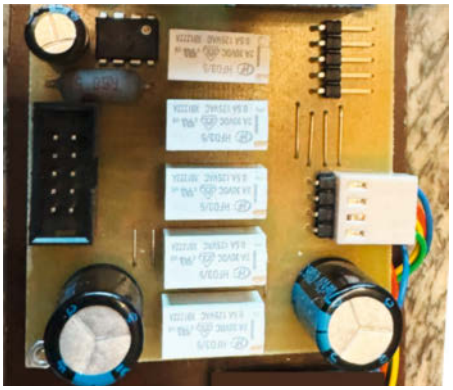
Umsetzung der Steuerung mit Relais

Die elektromechanische Steuerung der A77 besteht aus drei Motoren und zwei großen Hubmagneten. Bremse rechts/links und Capstan = Play/Record. Diese werden entweder über das eigene Tastenfeld der A77 oder – wie in unserem Fall – über ein Relais-Interface mit dem richtigen Timing und Signalpegel weitergegeben. Revox hat die Ansteuerung etwas kompliziert gestaltet: Die Schaltung des Relais-Interface entspricht der Tastensteuerung der Kabelfernbedienung. Bei der Fernbedienung für den Nachfolger B77 wurde das vereinfacht, was für die Inkompatibilität der neuen Fernbedienung mit dem Vorgängergerät sorgt. Eine direkte Ansteuerung über MOSFET (Metal Oxide Semiconductor Field-Effect Transistor) führte immer wieder zu Störungen beim ESP8266, auch Optokoppler helfen nicht. Deshalb werden Relais zur galvanischen Trennung von Steuerung und Webserver verwendet. Die großen Hubmagneten verursachen ordentlich Störungen trotz der Freilaufdioden. Ist ja auch eine per Tasten und Relais gesteuerte Umgebung. Da gab es noch keine EMV-Probleme.

Die A77 hat von Hause aus einen Konstruktionsfehler: Man kann Play starten, wenn die Wickelmotoren sich noch im schnellen Vor- oder Rücklauf befinden. Das gibt jedes Mal Bandsalat. Um dieses Problem zu umgehen, wurden in der Steuerung Abläufe realisiert, die diesen Konstruktionsfehler kompensieren, aber nicht final beheben – quasi ein Hotfix.

- schneller Vorlauf → Play ist verriegelt → Stop → 6 Sekunden Bremszeit → danach ist die Play-Taste wieder entriegelt
- schneller Rücklauf → Play ist verriegelt → Stop → 6 Sekunden Bremszeit → danach ist die Play-Taste wieder entriegelt
- Record wird zusammen mit Play gestartet. Es müssen nicht Play und Record gleichzeitig gedrückt werden, das geht automatisch.

Die A77 verfügt über eine 27-Volt-Stromversorgung für Diaprojektoren. Über diesen An-



Die Relais-Platine

schluss entnimmt man den Strom für den Webserver. Es sind maximal 50 mA, die benötigt werden. Der maximale Entnahmestrom darf 150 mA betragen. Über einen Schaltregler werden aus den 27 V kleinere 5,6 V erzeugt. Die 0,6 V mehr über TTL-Pegel (Transistor-Transistor-Logik) werden für die Überbrückung des Spannungsabfalls an der Einschaltverzögerung des Relais-Moduls benötigt. Damit sind EMV-Störungen kein Thema mehr.

Das Relais-Interface bildet 1:1 die kabelgebundene Originalfernbedienung nach. Alle Tasten der Originalfernbedienung werden durch Relais dargestellt. Die Ansteuerung erfolgt über einen Darlington-Treiber mit integrierten Freilaufdioden für die Entstörung. Die Schaltleistung der eingesetzten Relais beträgt 220 V / 60 W. Betriebsspannung der Relais ist 5 V. Die beiden Transistoren Q1 & Q2 bilden eine Einschaltverzögerung für die Relais. Denn es hat sich gezeigt, dass beim Booten des WEMOS D1 die Relais wild anfangen zu klappern, bis alle Ports im richtigen Betriebsmodus initialisiert sind. Das mag die A77 gar nicht. Deshalb bleiben die Relais alle am Anfang für ein paar Sekunden spannungsfrei.

Als zusätzliche Betriebskontrolle wurde der Laufwerkssteuerung ein OLED-Display spendiert. Hier werden die gerade aktivierten Betriebsmodi des Laufwerks angezeigt. Der vierpolige Anschluss passt zu diversen 0,96-Zoll-OLED-Displays. Aufpassen muss man bei der Belegung der Stromversorgungspins, da manche Hersteller GND und VCC vertauschen.

Das Display kann über die vierpolige Kabelverbindung betrieben werden. Der I²C-Bus unterstützt bis zu 15 m, aber ab 1 m sollte man auf gut geschirmte Kabel setzen. Das Display ist optional und muss nicht zwingend installiert werden, es gibt aber einiges an Informationen, wenn mal Schwierigkeiten mit der WLAN-Verbindung auftauchen.

Aus Gründen der Unversehrtheit des Tonbandgerätes wurde das Display nicht in die A77 eingebaut. Auch die Steuerungsplatine hat innerhalb der A77 nichts verloren. Die Aufnahme- und Wiedergabeverstärker mögen keine fremde Hochfrequenztechnik. Ein guter Platz ist die äußere Rückwand der A77 mit lösbaren Klebepads sowie mit oder ohne Abdeckung der Platine.

Hardware

Die Daten für die Herstellung der Steuerungsplatine befinden sich im Download-Bereich der Kurzinfo. Das Datenformat ist Gerber. Jeder professionelle Platinenhersteller kann diese Daten verarbeiten. Die Platine ist doppelseitig ausgeführt. Die verwendeten SMD-Bauteile Größe 1206 können ohne Probleme auf der Platinenunterseite von Hand verlötet werden. Einen Bestückungsautomat braucht man nicht. Online ist auch eine Stückliste mit allen Bauteilen verfügbar.

Die drei Steckverbinder auf der Platine haben folgende Funktion:

– oben links: Erweiterungsstecker mit noch freien Ports und 3,3-V-Stromversorgung

Auf dem Bildschirm stehen weitere Informationen zum Wiedergabestatus.

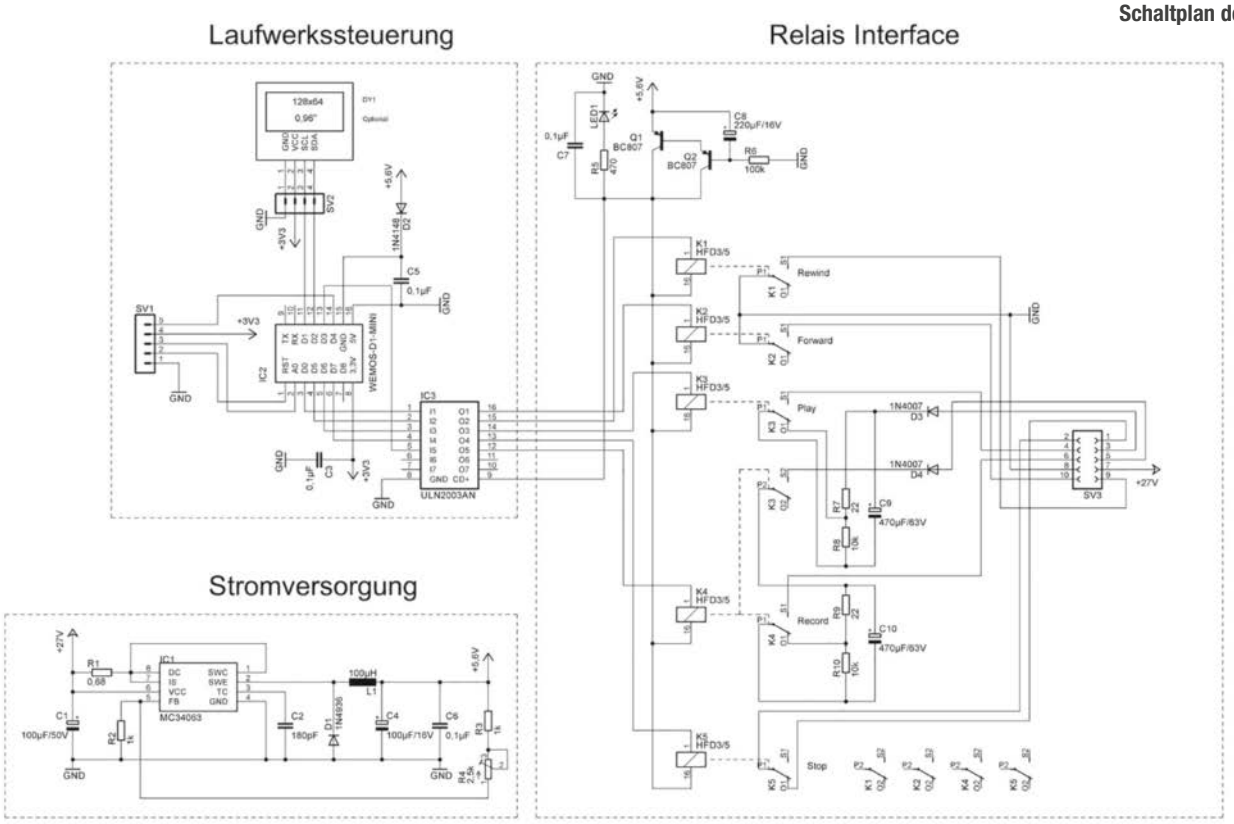
– oben rechts: OLED-Anschluss mit I²C-Bus und 3,3-V-Stromversorgung

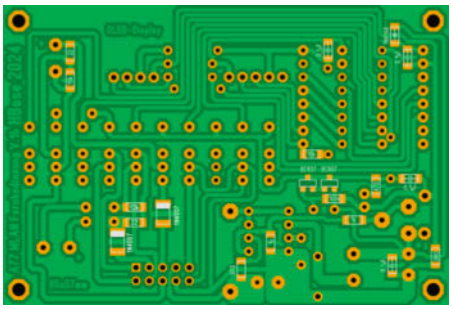
– unten: Fernsteuerungsanschluss für die A77

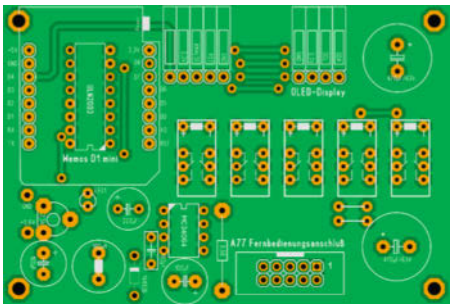
Die WIST10-Platine bildet den Original-Hirschmann-Stecker nach. Er kommt zum Einsatz, wenn das Original fehlt oder man sich die Kabelverbindung einfach machen möchte. Die Daten für die Platine befinden sich ebenfalls im Download-Bereich der Kurzinfo.

Die Platine muss mit zehn Stück 1,3-mm-Pins bestückt werden. Sollten diese nicht beschaffbar sein, kann man sich mit 1,5-mm²-Kupferdraht (aus einem NYM-Kabel) behelfen. Eine Ader abisolieren und den Draht recken, damit er gerade wird. Hierzu den Draht einseitig in den Schraubstock spannen und das andere Ende mit einer Zange greifen und kräftig ziehen. Der Draht wird kerzengerade und fest.

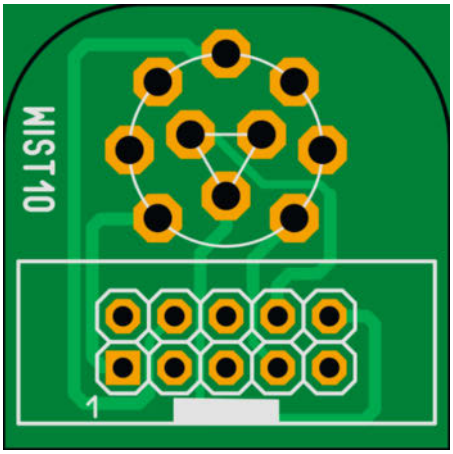
Schaltplan der Steuereinheit







Links: Platinenoberseite, rechts: Platinenunterseite



Steckerplatine

Dann zehn 13 mm lange Pins aus dem gerecken Draht schneiden und auf die Platine löten. Darauf achten, dass die Pins gerade sitzen.

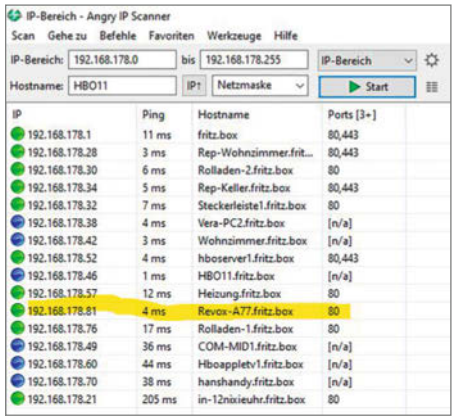
Eine Anmerkung noch zum Verlöten des Originalsteckers: Die Kontakte des Steckers stammen aus einer Zeit, in der es noch kein bleifreies Lötzinn gab. Deshalb nimmt die Oberfläche der Pins bleifreies Lötzinn nur widerwillig an. Daher entweder mit verbleitem Lötzinn löten oder Lötwasser von Fellner verwenden. Lötfett oder Ähnliches darf es auf keinen Fall sein. Das zerstört die vernickelte Oberfläche, was Wackelkontakte und kalte Lötstellen zur Folge hätte.

Software

Für das Projekt befindet sich der komplette Arduino-Sketch im Download-Bereich der Kurzinfo. Die benötigten Standard-Libraries können über den Library-Manager der IDE installiert werden. Auf den Einsatz eines WIFI-Managers wurde aus Speicherplatzgründen verzichtet. Deshalb müssen die eigene SSID und das WLAN-Passwort in den Sketch eingetragen werden. Es handelt sich um die Zeilen 19 und 20. Bevor der so modifizierte Sketch per USB übertragen wird, muss die Grafikdatei aus dem Data-Bereich des Sketch-Ordners mit LittleFS auf den WEMOS D1 übertragen werden (siehe Kasten „LittleFS-Plug-in“). Sie beinhaltet die in der HTML-Seite befindliche Grafik. Ohne diese Grafik bricht das Programm ab. Danach kann der modifizierte Sketch per USB in den WEMOS D1 überspielt werden.

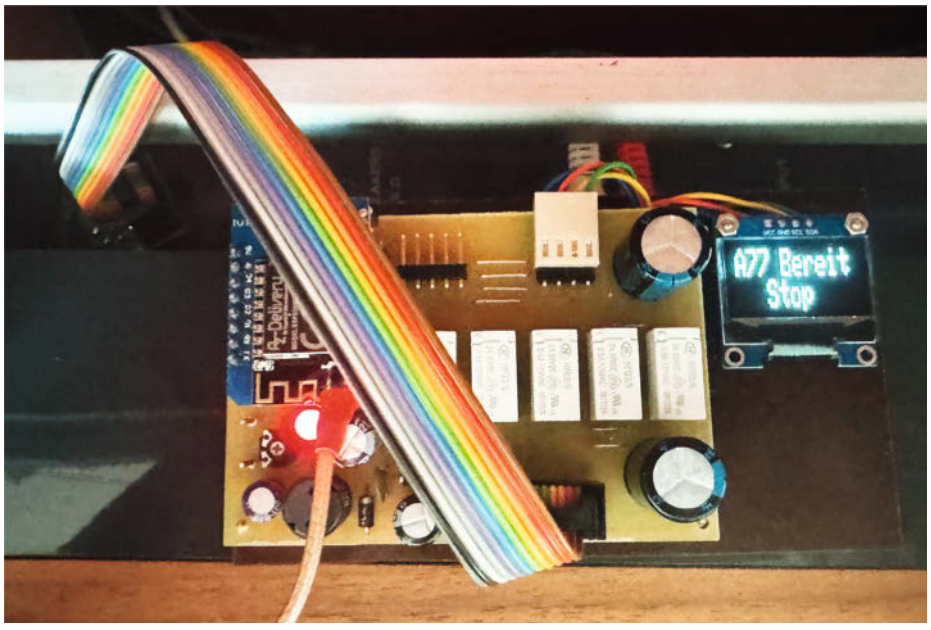
Jetzt ist der WEMOS D1 per OTA erreichbar und die Fernbedienungssoftware läuft. Die USB-Verbindung wird nur noch gebraucht, wenn man den seriellen Port benötigt.

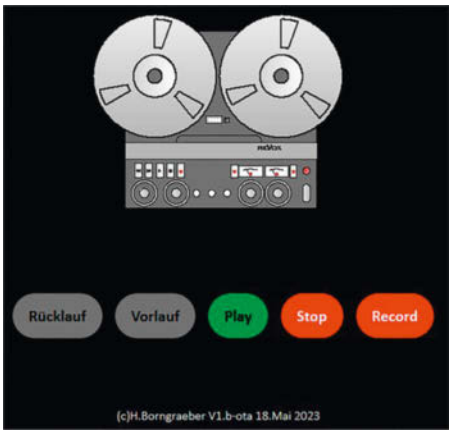
Die Fernsteuerung kann an die ausgeschaltete A77 angeschlossen werden. A77 einschalten, jetzt noch die per DHCP vom WLAN-Router vergebene IP-Adresse finden, diese in den Browser eingeben und es kann losgehen. Die IP-Adresse kann man z. B. mit dem Angry IP Scanner finden oder auf der Startseite einer Fritzbox. *—das*



Angry IP Scanner hilft beim Finden der IP-Adresse.

OTA erfolgreich gestartet.





So muss es aussehen, wenn alles geklappt hat.

Alles im Blick

mit dem Sonderheft zur ESP32-Kamera

Inklusive ESP32-CAM Development Board + OV2640 Kameramodul

Das 80-seitige Make-Special zeigt Ihnen, wie Sie aus dem mitgelieferten ESP32-CAM-Board samt 2-MP-Kameramodul und Programmier-Adapter das Meiste rausholen. Damit können Sie sofort Ihr erstes Funk-Kamera-Projekt starten!

- ► Kamera-API im Griff
- ► Projekte: Nistkasten mit WLAN, Zeitrafferkamera, Objekterkennung mit KI und mehr…
- ► Bilder per Mail verschicken
- ► Tipps & Hacks: Externe Antenne anschließen, Kamera auf Infrarot umbauen, Reset-Pin nachrüsten

Heft inklusive ESP32-CAM Development Board + OV2640 Kameramodul 29,90 €

shop.heise.de/make-esp32cam

vittaya pinpan/Shutterstock.com

Smartphone-Apps mit RemoteXY

In der Heim-Automation ist es mittlerweile normal, Klimaanlagen, Beleuchtungen oder Rollläden per Smartphone-App zu steuern. Bei Hobby-Projekten gibt es diesen Komfort eher selten. Wie man die eigene Hardware ganz leicht mit RemoteXY kontrollieren kann, zeigen wir in diesem Artikel.

von Maik Schmidt

Mikrocontroller zu programmieren, ist immer noch eine Herausforderung, aber über die letzten Jahre wurde es auch dank Arduino immer einfacher. Das liegt daran, dass sowohl die Programmierwerkzeuge (Arduino IDE) als auch die Boards immer leistungsfähiger wurden. Trotzdem kommuniziert die überwiegende Mehrheit der Projekte mit der Außenwelt meist noch über die serielle Schnittstelle oder im besten Fall über ein winziges Display. Im Vergleich zu einem modernen Smartphone mutet das geradezu spartanisch an.

Für manche Anwendungen ist es darüber hinaus oft notwendig, Sensoren an schwer zugänglichen Stellen zu platzieren. In solchen Fällen bieten eingebaute Displays wenig Mehrwert, denn es ist zumeist schwer bis unmöglich, sie abzulesen.

Da wäre es doch wünschenswert, den Mikrocontroller mit einem mobilen Display in Form des Smartphones zu koppeln und diesen so zu kontrollieren und zu überwachen. Eine Android-App, die per Bluetooth mit dem Mikrocontroller kommuniziert und dessen Messdaten schick aufbereitet, wäre eine feine Sache. Diese zu schreiben, ist aber leider alles andere als trivial.

Solche Mikrocontroller-Anwendungen arbeiten im Prinzip aber alle auf die gleiche Weise, das heißt, das Smartphone sendet Kommandos an den Mikrocontroller und dieser schickt seinen aktuellen Zustand zurück. Die Smartphone-App bereitet dann diesen Zustand grafisch auf und bietet darüber hinaus Kontrollelemente zur Steuerung der Anwendung an.

Ein findiger Entwickler kam daher auf die Idee, einen Generator für solche Helferlein zu schreiben, und so entstand RemoteXY.

Kurzinfo

- Interaktive Apps erstellen per Web-App
- Low-Code und Arduino
- Mikrocontroller ansprechen

Checkliste

Zeitaufwand: ab 1 Stunde

Kosten: ca. 10 Euro

Material

- **Mikroprozessor-Board,** hier ESP32
- **Sensor** BME280
- **Breadboard**

Mehr zum Thema

- Bernd Heisterkamp, Task-Reminder, Make 1/24, S. 8
- Carsten Wartmann, Internet-of-Things-Dienste für Maker, Make 3/21, S. 32

Alles zum Artikel im Web unter *make-magazin.de/x5h6*

Wie funktioniert das?

RemoteXY ist eine Low-Code-Plattform, die im Wesentlichen aus drei Teilen (siehe Bild 1) besteht. Da wäre zunächst ein grafischer UI-Editor (UI = User Interface), der im Browser läuft und zur Erstellung der Oberfläche einer Smartphone-Anwendung dient. Ebenfalls mit dabei ist eine Software-Bibliothek, die mit einer Reihe von Mikrocontrollern kompatibel ist und dafür sorgt, dass der Controller mit einer Smartphone-App kommunizieren kann. Genau diese Smartphone-App ist das letzte Puzzle-Teil, denn zu RemoteXY gehört eine Smartphone-App, in der man mithilfe der zuvor erstellten Benutzeroberfläche einen verbundenen Mikrocontroller steuern kann.

Damit dieser mit RemoteXY kommunizieren kann, erstellt die Plattform einen Quellcode für den gewünschten Mikrocontroller, der zusammen mit der RemoteXY-Bibliothek übersetzt werden kann. Dieses Projekt bildet allerdings nur das Gerüst für die eigentliche Anwendung, denn es benötigt noch weitere Funktionen.

In erster Linie sind das natürlich die eigentlichen Features auf dem Mikrocontroller-Board, wie zum Beispiel die Abfrage von Sensoren oder die Steuerung von Motoren. Dies unterscheidet sich kaum von einem regulären Maker-Projekt. Für die Integration mit RemoteXY muss die Firmware die UI-Elemente befüllen und auf Nutzereingaben reagieren.

Zuerst ist es notwendig, die Ziel-Umgebung genauer zu definieren, denn RemoteXY unterstützt eine Reihe von Mikrocontrollern, Entwicklungsumgebungen (IDE) und Übertragungswegen. Auch gibt es die Smartphone-App sowohl für Android als auch für iOS (siehe Kasten RemoteXY-Hardwareunterstützung).

Die Projekte in diesem Artikel basieren alle auf der Android-App und einem ESP32-Board

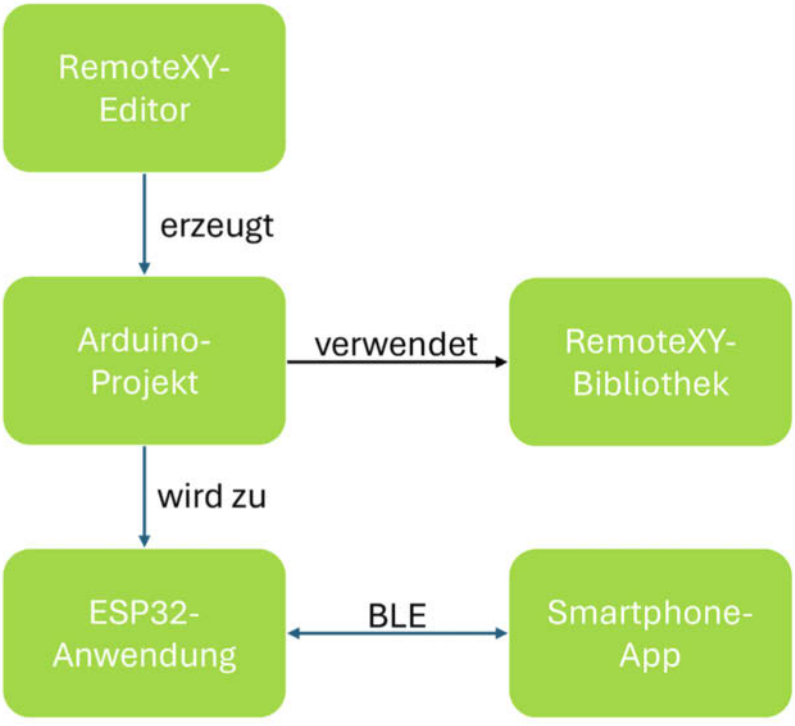


Bild 1: Zusammenspiel der Komponenten bei RemoteXY

RemoteXY-Hardwareunterstützung

Die Website von RemoteXY wirkt zunächst ein wenig unscheinbar, aber das Projekt unterstützt eine überraschend große Anzahl an Systemen. So läuft die RemoteXY-Bibliothek auf den folgenden Mikrocontrollern:

- Arduino Uno, Mega, Leonardo, Pro Mini, Nano, Micro und kompatiblen AVR-Boards
- ESP8266 und ESP32
- STM32F1
- chipKIT Uno32, chipKIT uC32, chipKIT Max32

Die RemoteXY-Smartphone-App kann wie folgt mit dem Mikrocontroller kommunizieren:

- übers Internet mithilfe eines Cloud-Servers
- per WLAN als Client oder Access-Point
- per Bluetooth mit dem Mikrocontroller als Access-Point
- per Ethernet im lokalen Netz
- per USB On-The-Go (OTG)

Nicht jeder Mikrocontroller verfügt wie der ESP32 über eine eingebaute Bluetooth-Einheit und Ethernet-Anschlüsse gibt es auch selten. Daher arbeitet RemoteXY mit den folgenden Erweiterungsmodulen zusammen:

- Bluetooth HC-05, HC-06
- Bluetooth BLE HM-10
- ESP8266 als Bluetooth-Modem
- Ethernet-Shield W5100

Als Entwicklungsumgebungen werden neben der Arduino IDE auch noch FLProg und MPIDE unterstützt.

Die Smartphone-App gibt es für Android und für iOS und sie läuft noch mit sehr alten Versionen dieser Betriebssysteme. Bei iOS gelten jedoch die üblichen systemspezifischen Einschränkungen, das heißt, dort funktionieren beispielsweise Bluetooth-Serial und OTG nicht.

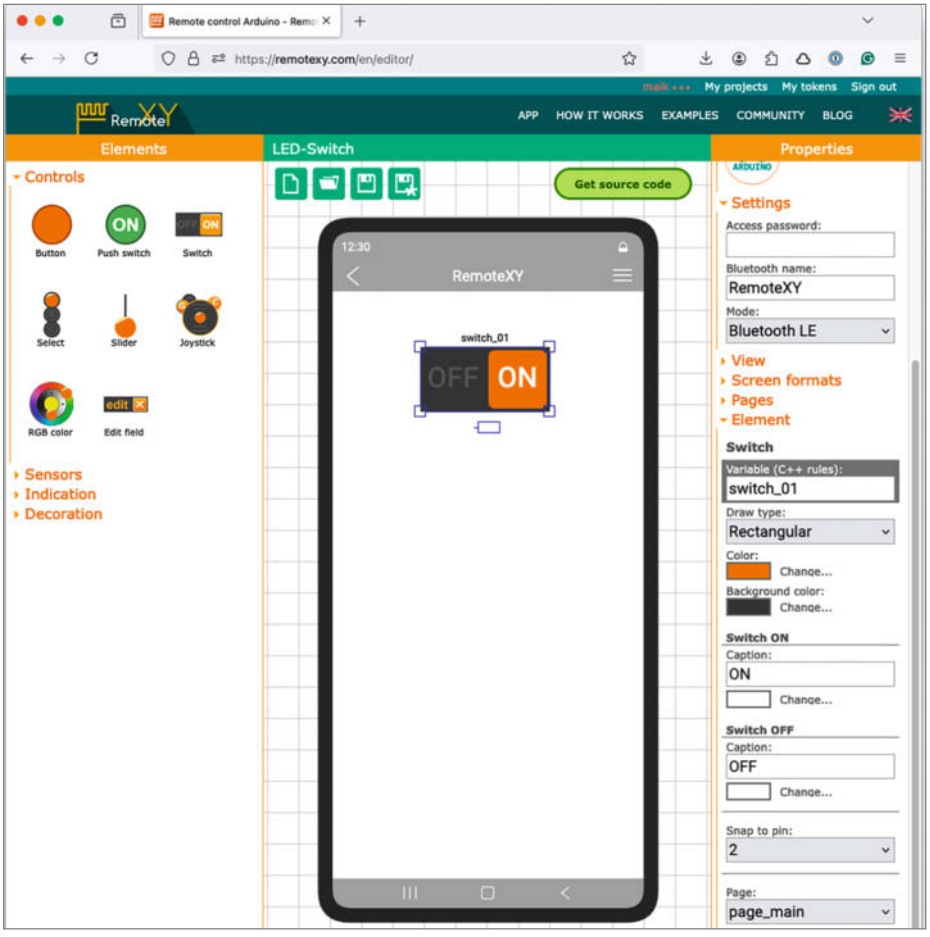


Bild 3: Ein Lichtschalter ist vielleicht die einfachste RemoteXY-Anwendung.

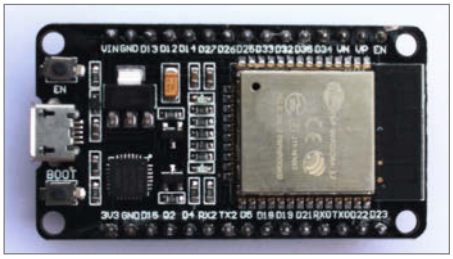


Bild 2: Ein ESP32-Board ist optimal für erste Experimente mit RemoteXY.

(Bild 2). Zur Erstellung der Mikrocontroller-Software dient die Arduino IDE und die Kommunikation zwischen dem ESP32 und dem Smartphone erfolgt per Bluetooth Low Energy (BLE).

Lichtschalter

Das erste Projekt realisiert einen kleinen Lichtschalter, der die Status-LED des ESP32-Boards mittels einer Smartphone-App ein- und ausschaltet. Die Oberfläche der App ist denkbar einfach und besteht nur aus einem einzelnen Schalter.

Mit dem UI-Editor (Bild 03) von RemoteXY ist es leicht, die Oberfläche zusammenzuklicken. Nach dem Start zeigt der Editor ein leeres Smartphone-Display an. Auf der linken Seite des Editors befinden sich mehrere Abschnitte, die verschiedene UI-Elemente gruppieren. Gleich in der ersten Gruppe (Controls) gibt es ein Schalterelement namens Switch, das man per Maus auf das Handy ziehen kann.

Sobald das Switch-Element aktiv ist, also den Fokus hat, verändert sich die rechte Seite des Editors. Im Abschnitt Elemente ist es dann möglich, die wichtigsten Eigenschaften des Schalters anzupassen. Dazu gehört unter anderem die Beschriftung für den Schalter, die per Voreinstellung die Bezeichnung ON beziehungsweise OFF trägt. Auch die Form sowie die Hintergrund- und Schriftfarbe sind frei wählbar.

Spannender sind die Einträge unter den Überschriften „Variable“ und „Snap to pin“. Letztere gibt an, welchen GPIO-Pin des Mikrocontrollers der Schalter steuern soll. In diesem Fall ist das Pin 2, weil die Status-LED des verwendeten Boards mit diesem Pin verbunden ist. Der Eintrag „Variable“ definiert den Namen der C++-Variablen, die am Ende den Zustand des Pins enthält. Diese Variable ist Bestandteil des Quelltextes, den RemoteXY am Ende aus dem Editor-Inhalt erzeugt. Per Voreinstellung hat sie den Namen „switch_01“.

Es lohnt sich, ein wenig mit den Einstellungen zu spielen, um ein Gefühl für die Möglichkeiten des Editors zu bekommen. Dabei ist es wichtig, neue Werte in Form-Feldern durch einen Druck auf die Enter-Taste zu bestätigen. Sonst kann es passieren, dass der

Editor Änderungen nicht verlässlich mitbekommt.

Selbst für kleinere Projekte empfiehlt es sich übrigens, einen kostenlosen Account auf der Seite anzulegen. Damit ist es möglich, Projekte zu speichern und später wieder zu laden.

Noch ein paar Handgriffe

Sobald die Oberfläche steht, muss RemoteXY noch wissen, auf welchem Mikrocontroller das Programm laufen soll und über welchen Weg es mit der Smartphone-App kommuniziert. Diese Einstellungen (Bild 4) erfolgen auf der rechten Seite des Editors im Abschnitt „Configuration“.

Von oben nach unten sind hier vier Entscheidungen zu treffen: Ganz oben steht der Übertragungsweg, in unserem Fall ist das Bluetooth. Direkt darunter erfolgt die Auswahl des Boards, für die Beispiele in diesem Artikel ein ESP32-Board.

Weil die meisten Mikrocontroller nicht über eine interne Bluetooth-Einheit verfügen, sondern auf ein externes Modul angewiesen sind, gibt es im dritten Abschnitt die Möglichkeit, ein passendes Modul auszuwählen. Beim ESP32 fällt die Wahl leicht, denn der hat Bluetooth mit an Bord. An letzter Stelle steht ein Menü zur Auswahl der IDE bereit, ich benutze hier die Arduino IDE.

Damit ist die Konfiguration fast abgeschlossen, aber die Bluetooth-Verbindung benötigt noch eine letzte Einstellung. Im „Settings“-Menü auf der rechten Seite des Editors können Nutzer den Namen des Bluetooth-Access-Points setzen und es ist zudem möglich, die Verbindung mit einem Passwort zu sichern.

Noch wichtiger ist aber die Konfiguration des Bluetooth-Modus, denn hier stehen Classic Bluetooth und Bluetooth Low Energy (BLE) zur Auswahl. Für das Beispiel-Projekt ist BLE die richtige Option.

Bild 4: RemoteXY unterstützt einige Mikrocontroller-Boards und Übertragungswege.

Her mit dem Code!

Damit ist die Konfiguration erledigt und wer einen RemoteXY-Account besitzt, sollte das Projekt nun erst einmal speichern. Anschließend führt ein Klick auf den „Get source code“-Button zu einer Seite (Bild 5), die den generierten Quelltext nicht nur anzeigt, sondern auch als Zip-Datei zum Download anbietet.

Diese enthält ein Arduino-Projekt mit dem wenig fantasievollen Namen „project“. Nach dem Auspacken der Datei ist es daher sinnvoll, den Ordner und die Datei project.ino umzubenennen.

Endlich aufs Board

Um das Projekt übersetzen zu können, muss die Arduino IDE lernen, mit ESP32-Boards umzugehen. Wie das funktioniert, erklären wir in

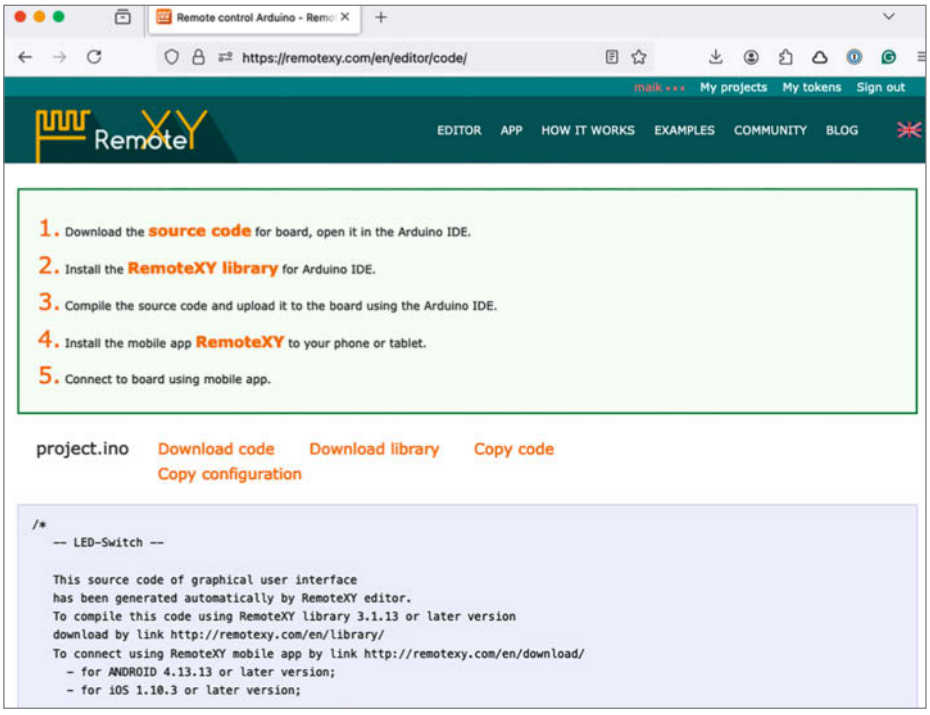


Bild 5: Das Herunterladen des generierten Codes ist einfach und komfortabel.

LedSwitch.ino

```
#define REMOTEXY_MODE__ESP32CORE_BLE
#define REMOTEXY_BLUETOOTH_NAME "RemoteXY"

#include <BLEDevice.h>
#include <RemoteXY.h>

#pragma pack(push, 1)
uint8_t RemoteXY_CONF[] =
  { 255,1,0,0,0,29, ... 70,0 };

struct {
  uint8_t switch_01;
  uint8_t connect_flag;
} RemoteXY;
#pragma pack(pop)

#define PIN_SWITCH_01 2

void setup() {
  RemoteXY_Init();
  pinMode(PIN_SWITCH_01, OUTPUT);
}

void loop() {
  RemoteXY_Handler();
  digitalWrite(PIN_SWITCH_01, (RemoteXY.switch_01 == 0) ? LOW : HIGH);
}
```

einem Online-Artikel, den Sie über den Link in der Kurzinfo erreichen.

Zur Übersetzung der RemoteXY-Anwendung fehlt noch die RemoteXY-Bibliothek. Deren Installation erfolgt über den Bibliotheksverwalter der IDE und bedarf nur weniger Mausklicks (Bild 6).

Ein genauerer Blick

Das Listing LedSwitch.ino (gekürzt, Code über den Kurzlink) enthält den erzeugten Quelltext in fast unveränderter Form. Das Programm beginnt mit einer auskommentierten `#define`-Direktive, die zur Steuerung des Logging-Verhaltens dient. Wenn `REMOTEXY__DEBUGLOG` definiert ist, gibt die RemoteXY-Bibliothek Nachrichten auf der seriellen Schnittstelle aus, die bei der Fehlersuche helfen können. Per Voreinstellung ist die Bibliothek aber stumm.

Die nächste `#define`-Anweisung bestimmt das Kommunikationsverhalten des Programms, in diesem Fall per BLE, was exakt der Konfiguration im RemoteXY-Editor entspricht.

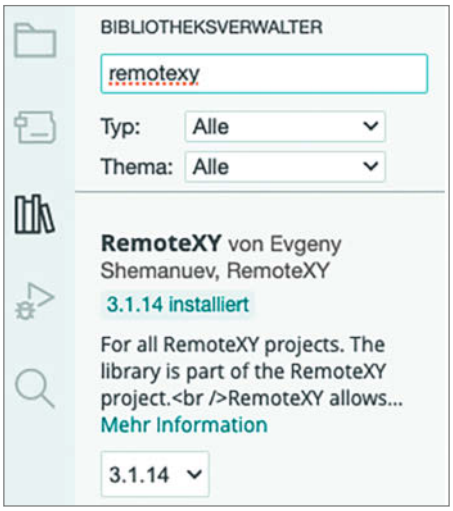


Bild 6: Die Installation der RemoteXY-Bibliothek ist dank des Bibliothek-Verwalters trivial.

Schließlich legt die dritte `#define`-Zeile den Namen des Access-Points fest, den das ESP32-Board aufspannen soll.

Zwei `#include`-Anweisungen binden die BLE-Bibliothek für den ESP32 und die RemoteXY-Bibliothek ein und darauf folgen die ersten RemoteXY-Spezifika des Programms. Eingebettet zwischen zwei `#pragma`-Anweisungen definiert der Code ein etwas kryptisches Feld namens `RemoteXY_CONF`. Die Zahlenkolonne ist aber lediglich eine kompakte Codierung der Bedienoberfläche der Smartphone-App, die der Mikrocontroller per BLE an das Smartphone sendet.

Spannend ist die Struktur mit dem Namen RemoteXY, denn diese enthält die Variablen, die zu den einzelnen UI-Elementen der App

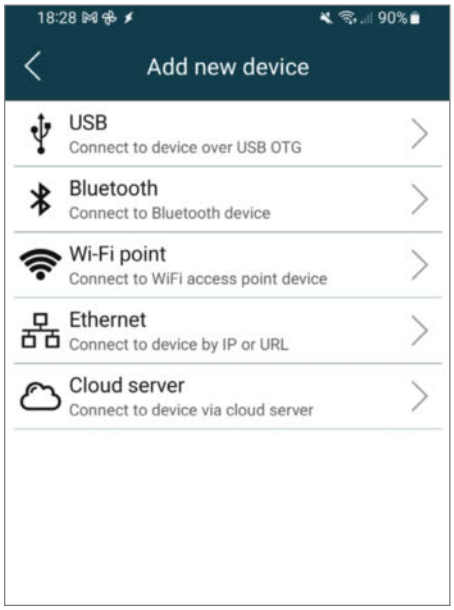


Bild 7: Die RemoteXY-App verbindet sich auf vielfältige Art mit Mikrocontrollern.

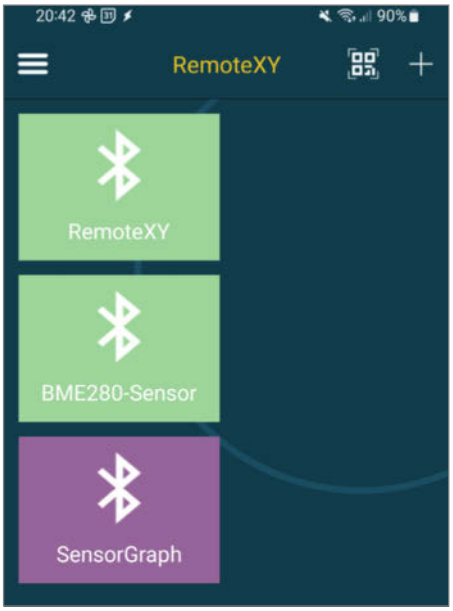


Bild 8: Die RemoteXY-App findet BLE-Geräte in der Umgebung.

Bild 9: Die erste App steuert die Status-LED des ESP32-Boards.

gehören. Wie zu erwarten war, findet sich hier auch die Variable `switch_01`, die den Zustand des Schalters enthält, der zur Steuerung der Status-LED dient.

Die Pin-Nummer der Status-LED definiert der Befehl `#define PIN_SWITCH_01 2` und die `setup()`-Funktion nutzt diese, um den Pin mittels `pinMode()` in den Ausgabe-Modus zu versetzen. Zuvor ruft sie allerdings die Funktion `RemoteXY_Init()` auf, die die RemoteXY-Bibliothek initialisiert. Dieser Aufruf ist zwingend erforderlich und muss am Anfang der `setup()`-Funktion erfolgen.

Ähnliches gilt für die Funktion `RemoteXY_Handler()`, die am Anfang der `loop()`-Funktion steht. Diese Funktion sorgt dafür, dass der Mikrocontroller und die Smartphone-App kontinuierlich Daten austauschen. Dieser Austausch darf nicht blockiert werden und so beinhaltet die RemoteXY-Bibliothek unter anderem die Funktion `RemoteXY_delay()`, die ein nicht blockierender Ersatz für die Arduino-Funktion `delay()` ist.

Den Schluss bildet ein Aufruf von `digitalWrite()`, der die Status-LED in Abhängigkeit vom Attribut `switch_01` in der RemoteXY-Struktur ein- oder ausschaltet.

Das ist auch schon die ganze Mikrocontroller-Anwendung und es spricht nichts dagegen, sie zu übersetzen, aufs ESP32-Board zu transferieren und in Betrieb zu nehmen.

App dafür!

Nach außen hin passiert aber erst einmal nicht viel, sobald die Anwendung ihren Dienst auf dem ESP32 antritt. Sie benötigt nämlich noch ihr Gegenstück in Form der RemoteXY-App. Diese App gibt es kostenlos im Google Play Store und auch im Apple Store für iOS-Geräte.

Nach dem Start der App führt ein Klick auf das Plus-Symbol in ein Menü zur Auswahl des Kommunikationsmediums (Bild 7). Hier ist Bluetooth die richtige Wahl und auf der darauffolgenden Seite ist der Button mit der Aufschrift „Select BLE“ anzuklicken.

Anschließend zeigt die App alle BLE-Geräte (Bild 8) in der Umgebung an und dabei sollte ein Gerät mit der Kennung „RemoteXY“ auftauchen, das zur Anwendung auf dem ESP32 gehört.

Ein Klick auf den Namen stellt die Verbindung zum ESP her und die App gibt dabei ein paar Log-Nachrichten aus. Wurde die Anwendung mit einem Passwort versehen, fragt die App jetzt nach dem Passwort. Sobald die Verbindung steht und die Mikrocontroller-Anwendung gültige Daten sendet, zeigt die App die Bedienoberfläche an, die man zuvor im RemoteXY-Editor erstellt hat.

In diesem Fall (Bild 9) ist das ein einzelner Schalter mit den Stellungen „ON“ und „OFF“. Mit diesem ist es nun möglich, die Status-LED des ESP32-Boards ein- und auszuschalten.

Wetterstation

Nach einem ersten erfolgreichen Test wird es Zeit für ein etwas praktischeres Projekt. Es soll eine App werden, die die aktuelle Temperatur und Luftfeuchtigkeit anzeigt. Gemessen werden diese Werte mit einem BME280-Sensor (Bild 10), der mit dem ESP32-Board verbunden ist.

Der BME280 ist ein I²C-Sensor, der neben der Temperatur und der Luftfeuchtigkeit auch noch den aktuellen Luftdruck ermittelt. Er lässt sich mit nur vier Drähten mit dem ESP32 verbinden und es gibt eine ganze Reihe von handlichen Bibliotheken, um ihn auszulesen. Eine dieser Bibliotheken ist die BME280-Bibliothek von Adafruit (Bild 11), die im Bibliotheksverwalter der Arduino IDE zur Installation bereitsteht.

Der Anschluss des Sensors ans ESP32-Board ist einfach. Die beiden GND-Pins bilden ein Paar und der VIN-Pin des BME280 gehört an den 3V3-Pin des ESP32. Der SCL-Pin des BME280 muss an Pin 22 und der SDA-Pin muss an Pin 21 des ESP-Boards.

Bild 10: Der BME280 misst Temperatur, Luftfeuchtigkeit und Luftdruck.

Langsam rantasten

Bei komplexeren Projekten ist es ratsam, klein anzufangen und so implementiert das Listing BME280-Test.ino ein minimales Test-Programm, das sämtliche Messwerte des BME280 auf der seriellen Schnittstelle ausgibt.

Bild 11: Die Adafruit-Bibliothek macht das Auslesen des BME280 zum Kinderspiel.

BME280-Test.ino

```
#include <Adafruit_BME280.h>
const float SEA_LEVEL_PRESSURE_HPA = 1013.25;
Adafruit_BME280 sensor;

void setup() {
    Serial.begin(9600);
    if (!sensor.begin(0x76)) {
        Serial.println("BME280-Sensor wurde nicht gefunden!");
        while (true)
            delay(10);
    }
}

void loop() {
    Serial.printf("Temperatur = %.2f °C\n", sensor.readTemperature());
    Serial.printf("Luftdruck = %.2f hPa\n", sensor.readPressure() / 100.0f);
    Serial.printf("Geschätzte Höhe = %.2f m\n", sensor.readAltitude(SEA_LEVEL_PRESSURE_HPA));
    Serial.printf("Luftfeuchtigkeit = %.2f %%\n\n", sensor.readHumidity());
    delay(5000);
}
```

Das Programm ist nur 20 Zeilen lang und denkbar einfach aufgebaut. Es bindet in der ersten Zeile die BME280-Bibliothek von Adafruit ein und definiert anschließend eine Konstante mit dem Luftdruck auf Meereshöhe, gemessen in Hektopascal (hPa). Diese Konstante verwendet das Programm im weiteren Verlauf, um aus dem gemessenen Luftdruck die aktuelle Höhe zu berechnen.

Die globale Variable `sensor` ist vom Typ `Adafruit_BME280` und kümmert sich um die Kommunikation mit dem BME280. Allerdings muss das Programm diese Kommunikation erst einmal initiieren und das passiert in der `setup()`-Funktion. Die initialisiert zu Anfang die serielle Schnittstelle und ruft dann die `begin`-Methode des `sensor`-Objekts mit der hexadezimalen Zahl `0x76` auf. Diese Zahl entspricht der Adresse des BME280 auf dem I²C-Bus und die `begin`-Methode versucht, das Gerät auf dem Bus zu finden. Falls das nicht klappt, gibt das Programm eine Fehlermeldung aus und startet eine Endlosschleife.

Andernfalls wurde ein Sensor gefunden und dann geht es mit der `loop()`-Funktion weiter. Die besteht hauptsächlich aus vier `printf()`-Anweisungen, die die Messdaten (Bild 12) ausgeben. Wegen der `delay`-Funktion passiert das alle fünf Sekunden.

Bild 12: Mit wenigen Handgriffen sendet der BME280 sämtliche Messdaten an die serielle Schnittstelle.

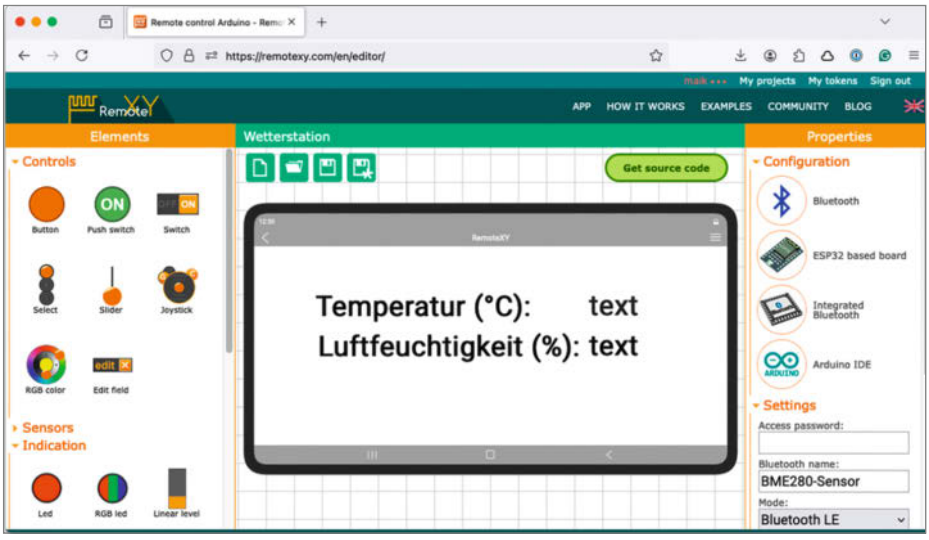


Bild 13: Zur Ausgabe der BME280-Daten reichen zwei Labels und zwei Text-Felder.

Stück für Stück

Wenn der Sensor funktioniert, ist es recht einfach, die Messwerte in einer RemoteXY-App anzuzeigen. Die Erstellung der Oberfläche erledigt man im UI-Editor: Für die einfachste Version (Bild 13) reichen zwei Labels mit den Texten „Temperatur (°C):“ und „Luftfeuchtigkeit (%):“. Rechts von den Labels steht jeweils ein Textfeld, das der ESP32 mit den aktuellen Messwerten füllt. Die Textfelder bekommen die Namen „text_temp“ bzw. „text_humidity“ und haben eine maximale Länge von zwölf Zeichen.

Der Code, den RemoteXY anschließend generiert, sieht in etwa so aus wie das (gekürzte) Listing Wetterstation.ino (Download siehe Link in der Kurzinfo).

Selbstverständlich ist es notwendig, den generierten Code um den Zugriff auf den BME280-Sensor zu ergänzen, denn davon weiß der RemoteXY-Editor ja nichts. Der kümmert sich ausschließlich um die Steuerung der Oberfläche.

Folgerichtig bindet das Programm die BME280-Bibliothek ein und definiert die globale Variable `sensor`. Zwischen den `#pragma`-

Wetterstation.ino

```
#define REMOTEXY_MODE__ESP32CORE_BLE
#define REMOTEXY_BLUETOOTH_NAME "BME280-Sensor"

#include <RemoteXY.h>
#include <BLEDevice.h>
#include <Adafruit_BME280.h>

Adafruit_BME280 sensor;

#pragma pack(push, 1)
uint8_t RemoteXY_CONF[] =
  { 255,0,0,24,0,82,0,17,0,0,0,31,1
    ...
    101,105,116,32,40,37,41,58,0 };

struct {
  char text_temp[12];
  char text_humidity[12];
  uint8_t connect_flag;
} RemoteXY;
#pragma pack(pop)

void setup() {
  RemoteXY_Init();
  Serial.begin(115200);
  if (!sensor.begin(0x76)) {
    Serial.println("BME280-Sensor wurde nicht gefunden!");
    while (true)
      delay(10);
  }
}

void loop() {
  RemoteXY_Handler();
  dtostrf(sensor.readTemperature(), 0, 1, RemoteXY.text_temp);
  dtostrf(sensor.readHumidity(), 0, 1, RemoteXY.text_humidity);
}
```

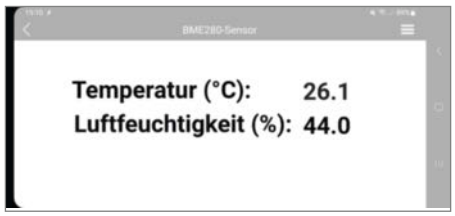


Bild 14: Eine kleine Wetter-App ist in wenigen Minuten fertig.

Anweisungen steht wie gewohnt die RemoteXY-Konfiguration. Die Pragma-Direktive sorgt dafür, dass die Datenstruktur so kompakt wie möglich gespeichert wird.

Die Struktur RemoteXY enthält nur zwei Zeichenketten mit den Namen `text_temp` und `text_humidity`. Diese muss das Programm im weiteren Verlauf kontinuierlich mit den entsprechenden Messwerten füllen.

Keine Überraschungen gibt es in der `setup()`-Funktion, sie initialisiert die RemoteXY-Bibliothek und den Sensor an Adresse `0x76`.

Sehr überschaubar ist auch die `loop()`-Funktion, denn sie füllt in erster Linie die Zeichenketten für die Temperatur und die Luftfeuchtigkeit. Dazu nutzt sie die Funktion `dtostrf()` der Arduino-Umgebung. Diese Funktion wandelt Fließkommazahlen in Zeichenketten um und bietet diverse Parameter zu deren Formatierung. In diesem Fall sorgt das dritte Argument beispielsweise dafür, dass die Zahlen mit nur einer Nachkommastelle angezeigt werden.

Nachdem die ESP32-App übersetzt und aufs Board gespielt wurde, kann die Smartphone-App sie unter dem Namen BME280-Sensor finden. Sofort zeigt sie kontinuierlich die Temperatur und die Luftfeuchtigkeit auf dem Handy-Display (Bild 14) an. Viel einfacher geht es nicht.

Jetzt in bunt

Eine textuelle Darstellung geht für manche Messwerte in Ordnung, aber oft machen schicke Diagramme deutlich mehr her. Das ist mit RemoteXY dank des Graph-Elements (Bild 15) kein Problem. Dieses Element unterstützt die Ausgabe von bis zu zehn unabhängigen Werten als Graph. Fügt man es im Editor der Smartphone-App hinzu, kann man zunächst die Anzahl der Variablen auswählen. Anschlie-

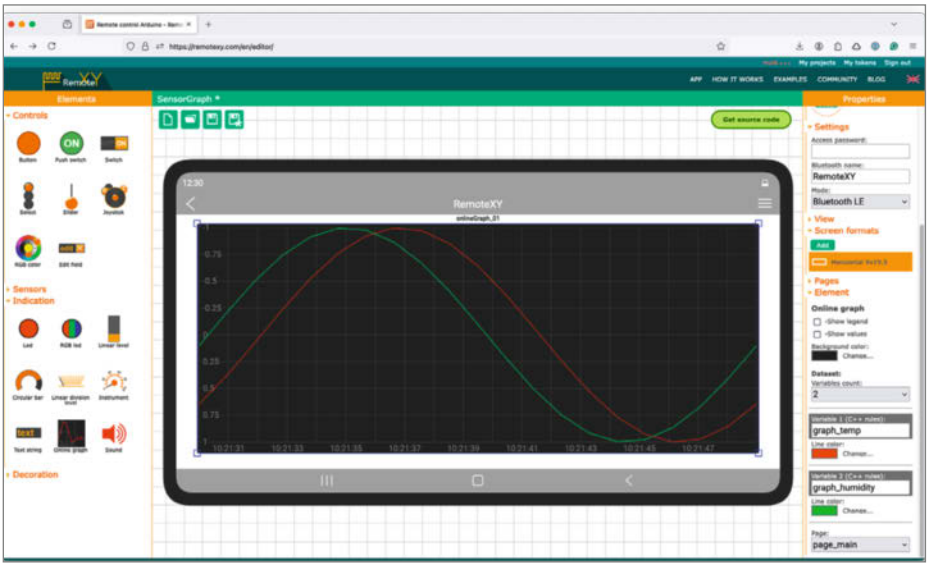


Bild 15: UI-Element zur grafischen Visualisierung von Daten.

SensorGraph.ino (gekürzt)

```
...

#pragma pack(push, 1)
uint8_t RemoteXY_CONF[] =
  { 255,0,0,8,0,21,0,17,0,0,0,31,1,200,84,1,1,1,0,68,
  7,3,186,76,2,8,36,135 };

struct {
  float graph_temp;
  float graph_humidity;
  uint8_t connect_flag;
} RemoteXY;
#pragma pack(pop)

...

void loop() {
  RemoteXY_Handler();
  RemoteXY.graph_temp = sensor.readTemperature();
  RemoteXY.graph_humidity = sensor.readHumidity();
}
```

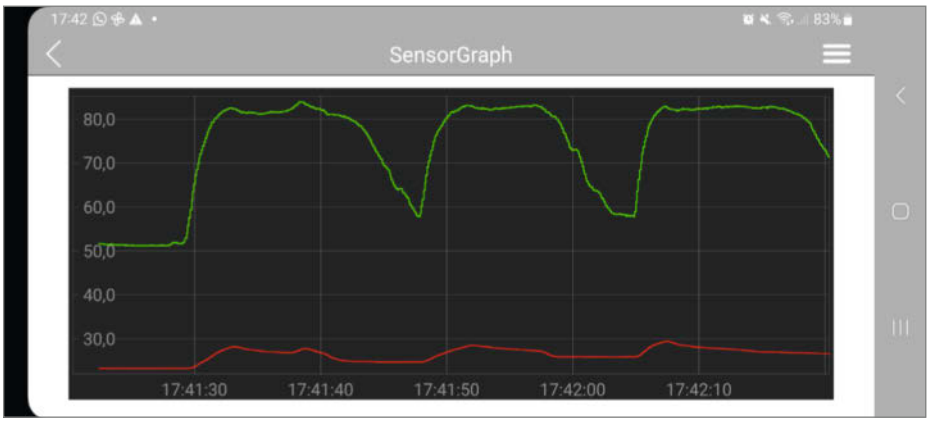


Bild 16: RemoteXY stellt Messwerte grafisch dar.

Nichts ist umsonst

Prinzipiell sind der UI-Editor, die Smartphone-App und die benötigten Bibliotheken von RemoteXY kostenlos. Allerdings nur für bis zu fünf UI-Elemente. Sobald eine App mehr als fünf nutzt, läuft die App auf dem Smartphone nur 20 Sekunden lang. Sämtliche Beispiele in diesem Artikel funktionieren aber mit der kostenlosen Variante von RemoteXY.

Welche Optionen es zum Kauf gibt, lesen sie in diesem Heft ab Seite 54.

ßend ist es möglich, jeder Variablen einen individuellen Namen und eine eigene Farbe zuzuweisen.

Diese Variablen werden zu Attributen der RemoteXY-Struktur. Listing SensorGraph.ino zeigt, wie das für die beiden Variablen `graph_temp` und `graph_humidity` aussieht. Ansonsten unterscheidet sich dieses Programm kaum vom Code in Wetterstation.ino. Hier entfällt lediglich die Umwandlung in Zeichenketten.

Die ESP32-Anwendung macht sich unter dem Namen „SensorGraph“ bekannt und die Smartphone-App (Bild 16) stellt nach dem Verbinden über BLE die Messwerte des BME280 in Form von zwei Kurven dar. Die grüne Kurve repräsentiert die Luftfeuchtigkeit und die rote zeigt die Temperatur. Temperatur und Feuchtigkeit schwanken üblicherweise nicht besonders schnell, daher habe ich den Sensor ein paar Mal angehaucht. Bei Sensoren, die Phänomene mit größerer Varianz erfassen, sieht das ganz anders aus und es ist darüber hinaus kein Problem, die Anzeigebereiche des Graphen zu verändern.

Fazit

RemoteXY ist ein spannendes Projekt und dieser Artikel kratzt nur an der Oberfläche der Möglichkeiten. Beispielsweise unterstützt der UI-Editor Apps mit mehr als einer Seite und es ist möglich, sowohl vertikale als auch horizontale Layouts zu erstellen. Das gesamte System – inklusive der Smartphone-App – ist äußerst komfortabel und stabil und somit steht Anwendungen, die komplexe Hardware wie einen Roboter steuern, prinzipiell nichts im Wege.

Etwas Sorge bereitet der Speicherbedarf, denn das LedSwitch-Beispiel belegt bereits 86 Prozent des Programmspeicherplatzes im ESP32 und das SensorGraph-Projekt bringt es auf beachtliche 89 Prozent. Immerhin stehen im RAM noch knapp 90 Prozent der Ressourcen zur Verfügung. —*caw*

HOHOHO! FROHE... ÄH.*

*Es ist doch noch nicht Weihnachten?! Stimmt! Aber **c't verschenken** können Sie das ganze Jahr. Schenken Sie **14 Ausgaben** für nur 88,90 € und überraschen Sie Ihre Liebsten mit den neuesten Technik-Innovationen, passenden Hard- und Softwareempfehlungen und erweitertem IT-Fachwissen. Zusätzlich gibt es ein **Geschenk Ihrer Wahl** on top. Jetzt bestellen und Technikfreude verschenken!

Jetzt bestellen:

ct.de/hohoho

ct.de/hohoho

+49 541/80 009 120

leserservice@heise.de

Photour.1904/ Shutterstock.com

IoT-Wasserwaage mit RemoteXY

Mit RemoteXY kann man nicht nur Apps bauen, um Daten von einem Mikrocontroller abzufragen. Andersherum ist es auch möglich, diesen mithilfe der Smartphone-Sensoren zu steuern. So lässt sich im Handumdrehen eine Wasserwaage mit ESP32 und NeoPixeln bauen, die darauf reagiert, wie man gerade sein Smartphone hält.

von Bernd Heisterkamp

RemoteXY kann über verschiedene Schnittstellen mit einem Mikrocontroller kommunizieren (siehe Artikel S. 30). Eine davon ist der Datenaustausch über die Cloud. Im folgenden Beispiel zeige ich, wie man mit RemoteXY die Sensoren des Smartphones auslesen und über eine Cloud-Verbindung zur weiteren Verarbeitung an einen Mikrocontroller übertragen kann. Um komplett auf eine physische Verdrahtung zu verzichten, verwende ich einen virtuellen ESP32 im Online-Schaltungssimulator Wokwi (Make-Artikel zu Wokwi findet ihr über den Link in der Kurzinfo). Den Quellcode und das Wokwi-Projekt könnt ihr über den Link in der Kurzinfo aus dem GitHub-Repository des Projekts herunterladen.

Wasserwaage mit NeoPixeln

Als Erstes erstellen wir in RemoteXY eine Benutzeroberfläche, die die Messungen des Lagesensors beziehungsweise den Roll- und Kippwinkel des Smartphones anzeigt und es erlaubt, die Nullstellung zu kalibrieren. Über die beiden Messwerte wird dann die Neigungsrichtung errechnet und auf einem an den ESP angeschlossenen Neopixel-Ring angezeigt. Zudem wird der Wert zusätzlich über ein Zeigerinstrument auf dem Smartphone-Display dargestellt (Bild 1).

Zur Auswertung des Lagesensors wählt man den Sensor „Orientation“ aus. Soll das Symbol selbst später nicht auf dem Display erscheinen, aktiviert man die Option „Hide on screen“ (unten rechts in der Abbildung). Um die gemessenen Werte auszugeben, verwende ich zwei „Value“-Textfelder, die ich „text_pitch“ und „text_roll“ genannt habe. Links daneben befinden sich zwei passende „Label“-Elemente, damit man die angezeigten Werte zuordnen kann. Darunter sitzt ein Button, der „button_zero“ heißt. Für die „Instrument“-Anzeige habe ich die Einstellungen gemäß Bild 2 gewählt, um die Richtung über die vollen 360 Grad sauber anzeigen zu können.

Damit man den RemoteXY-Cloud-Server nutzen kann, muss man ein Benutzerkonto anlegen und dann unter dem Menüpunkt „MyTokens“ einen neuen Token erstellen. Zudem muss die Konfiguration in RemoteXY für die Cloud-Verbindung entsprechend an-

gepasst werden (Bild 3). In diesem Fall wählt man für den virtuellen ESP32 eine WLAN-Verbindung ohne Passwort, mit der Wokwi-SSID „Wokwi-GUEST". Im Feld Token im Abschnitt „Cloud Server" gibt man dann den Token an, den man zuvor in RemoteXY angelegt hat.

Mit einem Klick auf „Get source code" erhält man das Programm als Arduino-Sketch, der sich 1 : 1 in Wokwi einsetzen lässt.

Wokwi mit RemoteXY verbinden

Für die virtuelle Schaltung in Wokwi habe ich auf der Startseite des Simulators (siehe Link in der Kurzinfo) das ESP32-Board und dann das „Starter Template ESP32" ausgewählt. Danach fügt man über den blauen „+"-Button einen NeoPixel-Ring mit 16 LEDs hinzu und verkabelt ihn, wie in Bild 4 zu sehen. Der Dateneingang des Rings ist mit Pin 25 des ESP verbunden.

Den RemoteXY-Programmcode kopiert man in Wokwi links in den Editor. Im Deklarationsbereich des Programms muss man noch ein paar Einstellungen vornehmen, damit die Verbindung zwischen RemoteXY und Wokwi funktioniert (Bild 5). In Zeile 39 setzt man unter anderem den selbstgenerierten RemoteXY-Token ein. Danach wechselt man zum „Library Manager" und fügt über das „+"-Symbol die Bibliotheken „Adafruit NeoPixel" und „RemoteXY" hinzu (Bild 6).

Hat man diesen Teil erledigt, kann man das Programm kompilieren und starten. Am WLAN-Symbol sieht man dann, ob der Wokwi-ESP eine Internetverbindung hat. Dies kann bis zu einer Minute dauern. Sobald die Verbindung steht, muss man die RemoteXY-App auf dem Smartphone nur noch mit dem erstellten Token verbinden. Dazu tippt man entweder auf das QR-Code-Symbol (Bild 7), um den QR-Code zu

Kurzinfo

» **RemoteXY-Cloud-Server verwenden**
» **Sensordaten vom Smartphones an einen ESP32 schicken**
» **Virtueller Aufbau mit Wokwi**

Mehr zum Thema

» Ákos Fodor, Mikrocontroller-Projekte simulieren mit Wokwi, Make 3/23, S. 88
» Bernd Heisterkamp, Smarte Werkstattboxen, Make 5/23, S. 38
» Michael Scheuerl, Smartphone-Roboter für den Unterricht, Make 2/22, S. 62

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

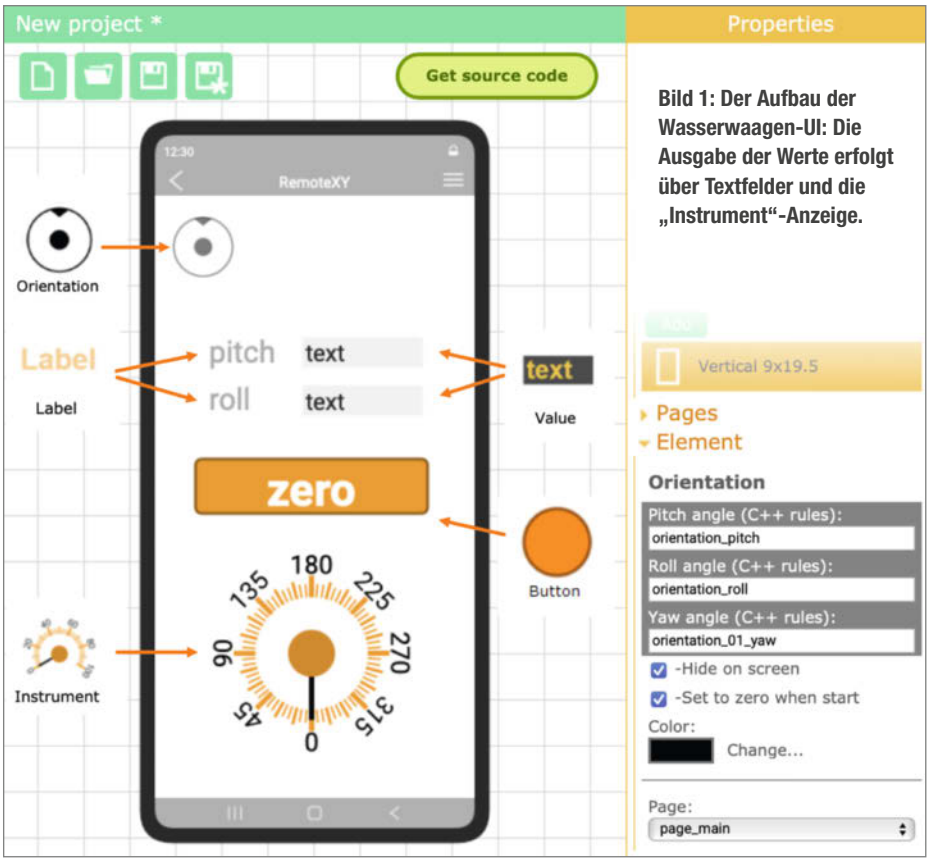


Bild 1: Der Aufbau der Wasserwaagen-UI: Die Ausgabe der Werte erfolgt über Textfelder und die „Instrument"-Anzeige.

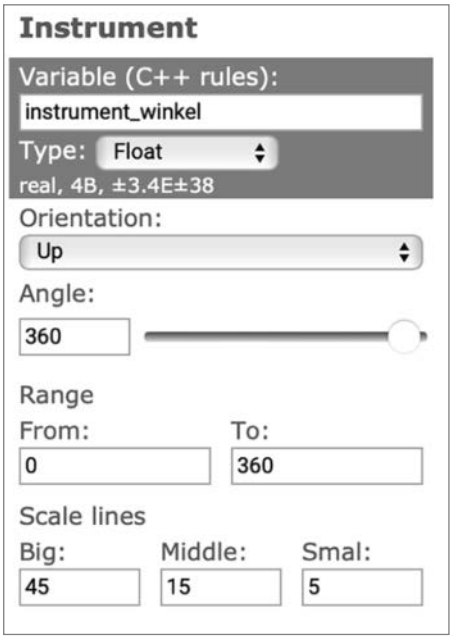


Bild 2: Die Skala der „Instrument"-Anzeige lässt sich an das Projekt anpassen.

Bild 3: Die Projekt-Einstellungen müssen wie folgt aussehen, wenn man den Cloud-Server nutzen möchte.

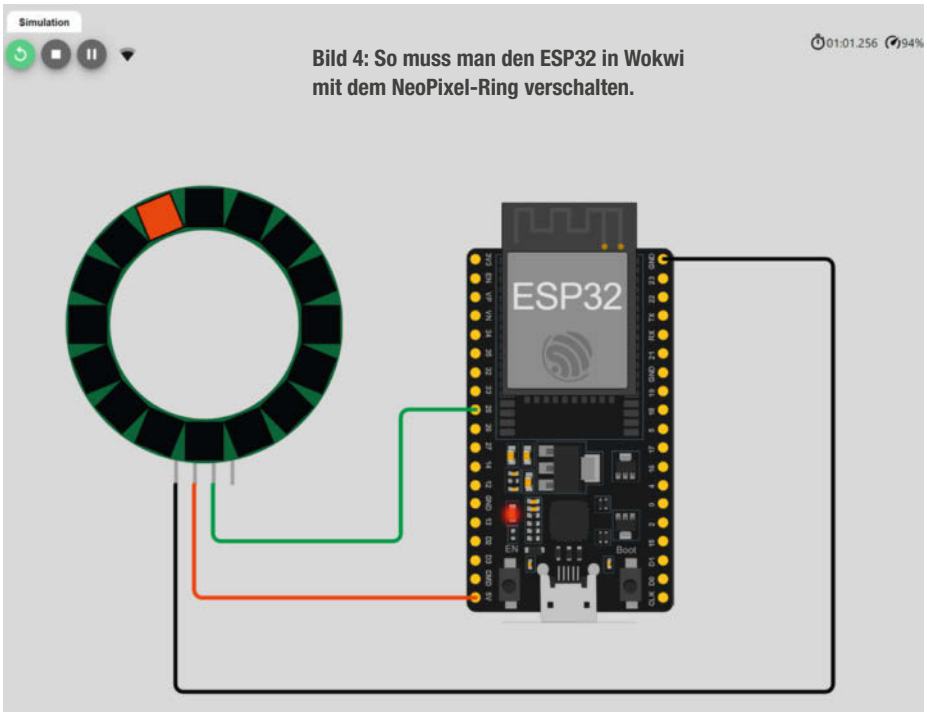


Bild 4: So muss man den ESP32 in Wokwi mit dem NeoPixel-Ring verschalten.

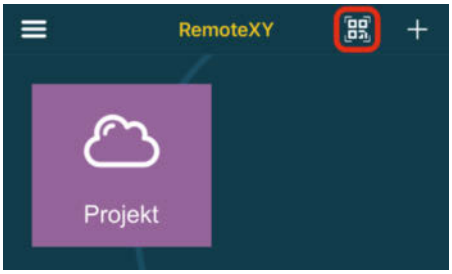


Bild 7: Den QR-Code-Scanner erreicht man in der RemoteXY-App, indem man auf das rot markierte Symbol tippt.

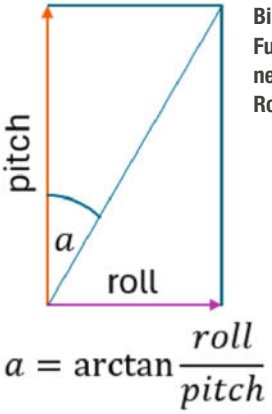


Bild 8: Die ArcTan-Funktion berechnet den Winkel aus Roll und Pitch.

scannen, den RemoteXY beim Erstellen des Programmcodes im Webbrowser erzeugt hat (unterhalb des Arduino-Codes) – oder man tippt auf das Plus-Symbol, wählt dann Cloud aus und gibt den Token manuell ein. Daraufhin erscheint die erstellte App auf dem Smartphone-Bildschirm – allerdings natürlich noch ohne Funktion, denn die Anzeigen und der Button sind bisher nicht programmiert.

Wenn man die Gratis-Version von RemoteXY nutzt, ist die Laufzeit der App auf 20 Sekunden beschränkt, weil wir mehr als fünf UI-Elemente verwenden. Man könnte als Lösung z. B. die beiden Label („pitch“ und „roll“) weglassen oder holt sich die Pro-Version (siehe Artikel Seite 54).

Daten lesen und anzeigen

Jetzt fehlt nur noch die Logik, die alle Elemente miteinander verbindet. Das Listing „wasserwaage.ino“ zeigt die Hauptschleife unseres Programms. Mit `RemoteXY.button_zero == true` wird der Messwert des Lagesensors erfasst, sobald man den Zero-Button drückt. Diese Werte werden dann von den weiteren Messungen abgezogen und in den Variablen `flt_pitch` und `flt_roll` gespeichert, damit in der Ruhelage die seitliche Neigung (engl: roll) und der Sturz (engl.: pitch) als `0` angezeigt werden.

Über die ArcTan-Funktion `atan2()` wird der Kippwinkel (0 bis 360 Grad) mithilfe der beiden Neigungswinkel berechnet (siehe auch Bild 8) und dem Zeigerinstrument auf dem Display

```
18  //////////////////////////////////////////////
19  //        RemoteXY include library          //
20  //////////////////////////////////////////////
21
22  // you can enable debug logging to Serial at 115200
23  //#define REMOTEXY__DEBUGLOG
24
25  // RemoteXY select connection mode and include library
26  #include <Adafruit_NeoPixel.h>
27  #include <WiFi.h>
28  #include <WiFiClient.h>
29  #include <WebServer.h>
30  #include <uri/UriBraces.h>
31
32  #define REMOTEXY_MODE__WIFI_CLOUD
33  #define WIFI_CHANNEL 6
34  // RemoteXY connection settings
35  #define REMOTEXY_WIFI_SSID "Wokwi-GUEST"
36  #define REMOTEXY_WIFI_PASSWORD ""
37  #define REMOTEXY_CLOUD_SERVER "cloud.remotexy.com"
38  #define REMOTEXY_CLOUD_PORT 6376
39  #define REMOTEXY_CLOUD_TOKEN "eigenerToken"
40
41  #include <RemoteXY.h>
```

Bild 5: Damit der ESP32 mit der RemoteXY-Cloud kommunizieren kann, muss man den generierten Code noch etwas anpassen.

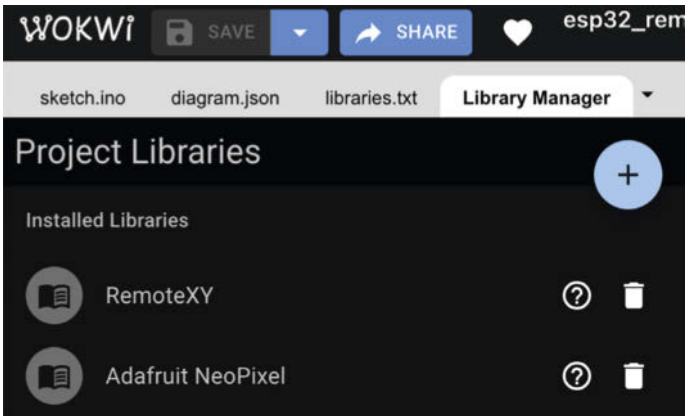


Bild 6: Wie in der Arduino IDE muss man auch in Wokwi Bibliotheken einbinden, damit die angeschlossene Hardware funktioniert.

zugewiesen (`RemoteXY.instrument_winkel`). Der errechnete Wert wird danach auf die 16 LEDs skaliert und die entsprechende LED wird schlussendlich mithilfe einer `for`-Schleife angesteuert und auf Rot gesetzt.

Wenn man jetzt die Simulation in Wokwi erneut startet und sein Smartphone über den QR-Code mit RemoteXY verbindet, kann man das Gerät neigen und sieht den Neigungswinkel auf dem Smartphone-Bildschirm und auch die korrespondierende NeoPixel-LED in Wokwi aufleuchten.

Falls ihr den Sketch wasserwaage.ino heruntergeladen habt und das Programm mit eurer selbst erstellten Benutzeroberfläche testen wollt, reicht es, wenn ihr den Code-Abschnitt unterhalb des auskommentierten `END RemoteXY include` in eurem (von RemoteXY generierten) Programm-Gerüst austauscht. Damit keine weiteren Änderungen notwendig sind, müssen die Bezeichnungen der einzelnen RemoteXY-Elemente (z. B. `button_zero`) so lauten, wie in diesem Artikel beschrieben. Wir wünschen viel Spaß beim Experimentieren. —*akf*

wasserwaage.ino

```
void loop() {

  RemoteXY_Handler();
  if (RemoteXY.button_zero) {
    offset_roll = RemoteXY.orientation_roll;
    offset_pitch = RemoteXY.orientation_pitch;
  }

  flt_pitch = RemoteXY.orientation_pitch - offset_pitch;
  flt_roll = RemoteXY.orientation_roll - offset_roll;

  RemoteXY.instrument_winkel = atan2(flt_roll,-flt_pitch) * 180.0 / PI + 180;

  dtostrf(flt_pitch * 180.0 / PI, 10, 4, RemoteXY.text_pitch);
  dtostrf(flt_roll * 180.0 / PI, 10, 4, RemoteXY.text_roll);

  int ledIndex = map(RemoteXY.instrument_winkel, 0, 360, 0, NUM_LEDS - 1);
  for (int i = 0; i < NUM_LEDS; i++) {
    if (i == ledIndex) {
      strip.setPixelColor(i, strip.Color(255, 0, 0));
    } else {
      strip.setPixelColor(i, strip.Color(0, 0, 0));
    }
  }

  strip.show();
  RemoteXY_delay(100);
}
```

Bluetooth-LE-Sensoren in die Cloud bringen

In Fitness-Armbändern, Reifendrucksensoren und in Apples AirTags sind stromsparende Sensoren eingebaut, die ihre Werte über Bluetooth Low Energy weitergeben. In diesem Bericht zeige ich, wie man mit einem alten Smartphone die Messwerte solcher Sensoren ohne viel Aufwand weiter in die Cloud leiten kann.

von Ramon Hofer Kraner

Mit Bluetooth Low Energy (BLE) lassen sich viele spannende und interaktive Projekte umsetzen. So kann man etwa ein smartes Türschloss entwickeln, das sich per BLE vom Smartphone entriegeln lässt. Oder ein System zur Raumklimaüberwachung, das die Daten direkt in die Cloud sendet. Wenn Sie schon immer einen stromsparenden Sensor mit dem Smartphone koppeln und dessen Werte in die Cloud übertragen wollten, sind Sie hier richtig.

Zuerst erkläre ich, wie BLE funktioniert und wie man damit eine Verbindung zum Handy herstellt. Dafür brauchen wir kein hochmodernes Smartphone. Das klappt auch mit den alten, ausrangierten Android-Geräten, die man vielleicht noch in der Schublade hat. Sie müssen dafür lediglich Android 5.0 oder höher unterstützen.

Dann geht es weiter mit der Cloud-Anbindung. Um die Daten dort hinzubringen, bauen wir mit dem MIT App Inventor per Drag & Drop eine maßgeschneiderte App. Programmierkenntnisse sind kaum erforderlich. Um sicherzustellen, dass alles in der Praxis funktioniert, benötigen wir einen passenden Sensor. In diesem Artikel benutzen wir dafür ein Potentiometer, das wir über ein MKR Grove Shield mit einem Arduino MKR WiFi 1010 verbinden, um eine Art Fake-Batteriemonitor zu bauen. Der Ausgangswert des Potentiometers simuliert hier den Ladezustand der Batterie. Da es mit einer Handbewegung verändert werden kann, können wir leicht überprüfen, ob die Werte übertragen werden.

Kurzinfo

- Mit App Inventor eine BLE-fähige App erstellen
- Wie Kommunikation mit BLE-Geräten funktioniert
- Datenübertragung vom Smartphone zur Cloud

Checkliste

Zeitaufwand:
2 Stunden

Kosten:
ca. 60 Euro

Material

- **Arduino MKR WIFI 1010** oder ähnliches Board mit BLE
- **MKR Grove Shield**
- **Potentiometer mit Grove-Anschluss**

Software

- **Arduino IDE**
- **Tago.IO**
- **MIT App Inventor**
- **nRF Connect for Mobile**

Mehr zum Thema

- Heinz Behling, Funktechniken fürs Smart Home, Make 5/24, S. 116
- Heinz Behling, Heizung unter Kontrolle, Make 2/21, S. 22

Alles zum Artikel im Web unter *make-magazin.de/x6tz*

Wie BLE funktioniert

Viele Sensoren, die auf Stromsparen getrimmt wurden, nutzen für den Datenaustausch eine Bluetooth-Verbindung zum Smartphone. Noch sparsamer im Energieverbrauch ist Bluetooth Low Energy (BLE). Dies wird hauptsächlich erreicht durch die Nutzung von speziellen Tiefschlaftechniken und kurzen Datenaustausch mit geringer Datenrate. Im Gegensatz zu Bluetooth bleibt Bluetooth Low Energy im Ruhezustand, bis eine Verbindung hergestellt wird. Die Verbindungszeit beträgt nur wenige Millisekunden, die für die Übertragung kleiner Datenmengen nötig sind. Diese periodische Datenübertragung eignet sich zwar nicht für Musik oder andere Aufgaben mit kontinuierlichem Datenfluss, aber für Anwendungen mit gelegentlichem Datenaustausch, wo Energiesparen angesagt ist, ist sie eine bessere Option.

Die Struktur einer BLE-Verbindung

Eine Verbindung zu einem BLE-Gerät wird über die sogenannten Services auf dem Gerät definiert. Ein Service kann z. B. ein Herzfrequenz-Service oder ein Batteriezustands-Service sein. Jeder Service wiederum besteht aus einer oder mehreren Charakteristiken.

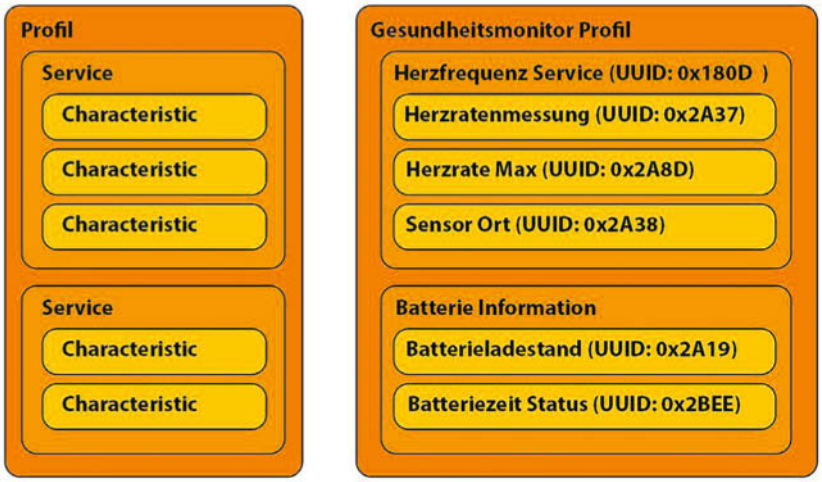


Bild 1: BLE nutzt einen logischen Aufbau von Profilen mit Services und deren Charakteristiken.

Characteristic Name	Requirement	Mandatory Properties	Optional Properties	Security Permissions
Heart Rate Measurement	M	Notify		None.
Heart Rate Measurement Client Characteristic Configuration descriptor	M	Read, Write		None.
Body Sensor Location	O	Read		None.
Heart Rate Control Point	C.1	Write		None.

Bild 2: Welche Charakteristiken z. B. ein Herzfrequenz-Sensor-Profil haben muss und darf, wird von der Bluetooth Special Interest Group festgelegt.

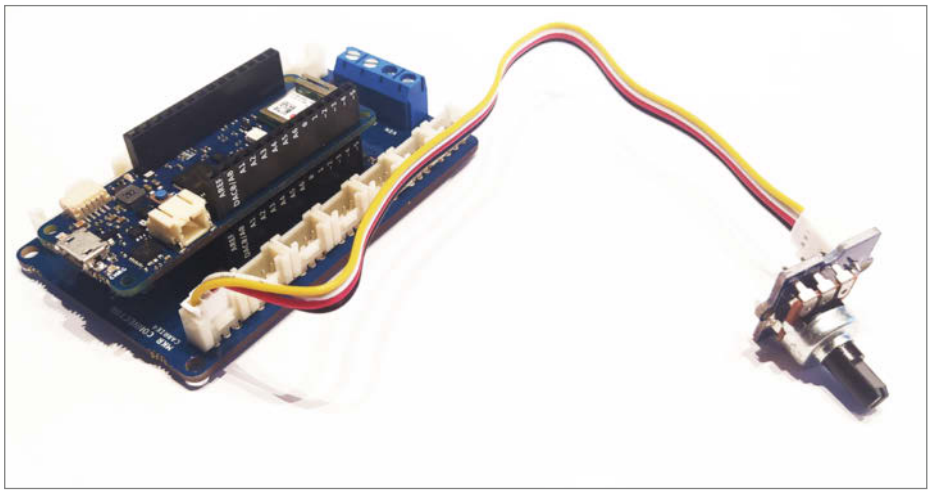


Bild 3: Der Arduino MKR WiFi 1010 wurde mit einem MKR Grove Shield und einem Potentiometer kombiniert. Durch Drehen des Potentiometers können wir schnell neue Werte erzeugen, um die Übertragung zu testen.

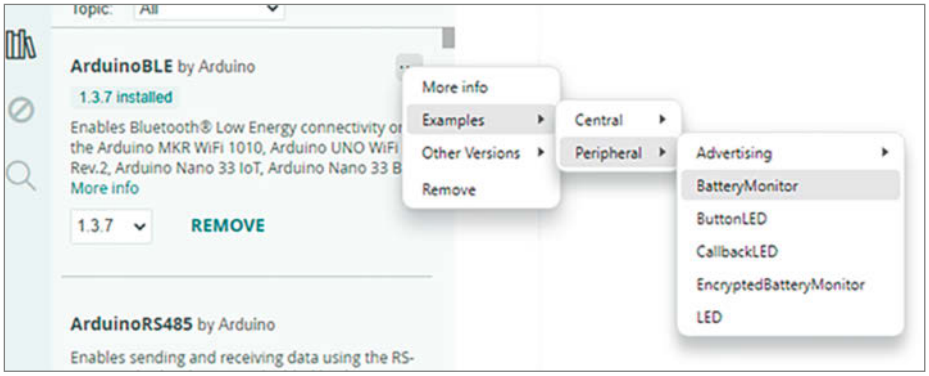


Bild 4: Arduino-BLE-Library installieren und das Beispiel BatteryMonitor laden

Darin werden die eigentlichen Werte der Sensoren aufbewahrt. Ein Herzfrequenz-Service kann z. B. Charakteristiken für die Herzrate, die Maximalrate oder den Sensorort beinhalten.

Ein BLE-Profil fasst diese verschiedenen Services zusammen und bietet so einen Überblick darüber, was das Gerät alles anzubieten hat (Bild 1).

Die für die Verwaltung und Pflege des Bluetooth-Standards zuständige Organisation ist die Bluetooth SIG (Special Interest Group). Für alle verbreiteten Services und Charakteristiken hat sie spezielle, eindeutige Nummern vergeben. Diese bestehen aus einem 16-Bit-Code (z. B. `0x180D` für den Herzfrequenz-Service). Insgesamt ist die Nummer ganze 128 Bit lang und wird UUID (Universally Unique Identifier) genannt. Die SIG-Nummern basieren auf einer Basisnummer, welche für alle gleich ist. Nur der kurze 16-Bit-Code unterscheidet sich. Die Basisnummer lautet:

```
00000000-0000-1000-8000-00805F9B34FB
```

Am Beispiel des Herzfrequenz-Services wird der 16-Bit-Code wie folgt eingefügt:

```
0000180D-0000-1000-8000-00805F9B34FB
```

Die definierten SIG-Standard-Nummern für Charakteristiken und Services können in den entsprechenden Dokumenten, herausgegeben von der Bluetooth Special Interest Group, nachgelesen werden (siehe Link in der Kurzinfo). Ebenfalls sind für Standardfälle Profile definiert, welche zeigen, welche Services und Charakteristiken in einem Profil vorhanden sein sollten oder müssen (Bild 2).

Wer sich also an den Standard halten bzw. sein Gerät möglichst kompatibel halten will, der sollte diese Nummern verwenden.

Bild 5: Der BLE-Analyzer nRF Connect for Mobile

Das Ganze funktioniert aber auch mit frei gewählten UUIDs für die Services und Charakteristiken – es sollte nur darauf geachtet werden, dass die Nummern sich nicht mit den offiziellen SIG-Nummern in die Quere kommen. Dazu kann man sich online UUIDs mit einem UUID-Generator erzeugen lassen. In der Kurzinfo gibts dazu einen Link.

Mit diesen UUID-Nummern kommuniziert man nun mit dem Peripheriegerät. In unserem Fall wird also das Smartphone beim Arduino nachfragen, wie beim Herzfrequenz-Service die Herzratenmessung aussieht. Neben der reinen Abfrage kann bei Bedarf in die Charakteristiken geschrieben werden. Damit können auch Daten an die Peripheriegeräte zurückgesendet werden.

Endlich geht es los

Um über BLE senden zu können, benötigen wir einen BLE-fähigen Mikrocontroller. Ich habe hier den MKR WiFi 1010 von Arduino verwendet. Er verfügt sowohl über Bluetooth Classic als auch über das anvisierte Bluetooth LE und WLAN. Wir könnten also sogar über WLAN direkt in die Cloud gehen. Ich möchte hier aber die Kopplung über BLE zu einer Smartphone-App aufzeigen und daher wird die Verbindung zuerst über das Smartphone laufen.

Nun brauchen wir einen Sensor, der uns Werte zum Ablesen schicken kann. Wie ich zu Beginn des Artikels erklärt habe, verwenden wir dazu ein Potentiometer, um eine Batterieladezustandsanzeige zu imitieren. So ist es möglich, eine Live-Demo von sich ändernden Werten zu erzeugen, wenn man den Drehknopf bewegt. Ich habe dazu ein MKR Grove Shield genommen und ein Potentiometer am Anschluss A0 angeschlossen (Bild 3). Wer kein Grove Shield hat oder kaufen möchte, kann auch ein Potentiometer direkt an den Arduino anschließen.

Damit wir BLE nutzen können, muss in der Arduino IDE die Library „Arduino BLE by Arduino“ über den Library Manager installiert werden. Danach befinden sich im Menü unter „Examples/Peripheral“ einige Beispiele (Bild 4). Für einen ersten Test lädt man sich das Programm BatteryMonitor auf den Arduino.

Sehen und gesehen werden

Da wir nun noch nichts sehen, müssen wir uns ein erstes wichtiges Tool auf unserem BLE-fähigen Smartphone installieren: einen BLE-Sniffer oder -Analyzer. Hier kann z. B. das Tool „nRF Connect for Mobile“ von Nordic aus dem Google Play Store oder dem Apple App Store installiert werden (Bild 5). Die App benötigt Standortfreigabe und natürlich Bluetooth.

Damit im BLE-Analyzer das Gerät gefunden wird, muss in der Arduino IDE der serielle

Schnittstellenmonitor eingeschaltet werden, denn das BatteryMonitor-Beispielprogramm wartet auf diese Verbindung beim Aufruf: `while(!Serial)`. Sobald eine serielle Verbindung hergestellt ist, sollte beim Drücken auf „Scan“ schnell ein BatteryMonitor gefunden werden (Bild 6).

Wir klicken danach auf „Connect“ und sehen nun drei Services, darunter unseren gesuchten „Battery Service“ mit der UUID `0x180F`. Klappt man diesen wiederum auf, finden wir die Charakteristik „Battery Level“ mit der UUID `0x2A19` (Bild 7). Leider sehen wir noch keinen Wert des Batterielevels. Dazu muss man auf den Download-Knopf mit drei Pfeilen drücken. Es findet nun ein kontinuierliches Abfragen des Wertes statt und der Wert wird auch live in Prozent angezeigt (Bild 8).

Falls alles erfolgreich in Betrieb genommen wurde, sieht man nun, dass die UUIDs im Code (Bild 9) genau den IDs entsprechen, die auch in der „nRF Connect for Mobile“-App angezeigt werden. Bei Änderung des Potis wird der angezeigte Batteriewert tatsächlich verändert. Die Übertragung per Bluetooth LE funktioniert!

Die Beispiel-App laden

Nun soll eine App auf dem Smartphone die BLE-Daten empfangen und verwerten. Dafür nutze ich das Online-Tool „App Inventor“ des MIT. Diese intuitive, visuelle Programmierumgebung ermöglicht es jedem, voll funktionsfähige Apps zu erstellen (mehr dazu über den Link in der Kurzinfo). Um sich mit App Inventor eine neue App zu bauen, muss man sich dort ein Benutzerkonto erstellen oder sich z. B. mit einem Google-Konto verlinken.

Meine Beispiel-App (BLEtoCloud.aia) kann über den Link in der Kurzinfo heruntergeladen

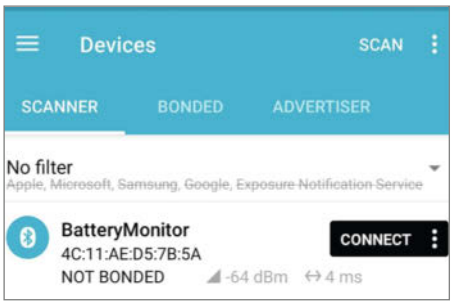


Bild 6: BatteryMonitor gefunden

werden. Sie lässt sich dann über den Eintrag „Import project“ im Menü „Projects“ importieren und öffnen (Bild 10). Die Beispiel-App sollte bereits die Bluetooth-LE-Erweiterung integriert haben. Ist dies nicht der Fall, kann der Vorgang durch einen Klick in der Palette links unter „Extension/Import Extension“ nachgeholt werden

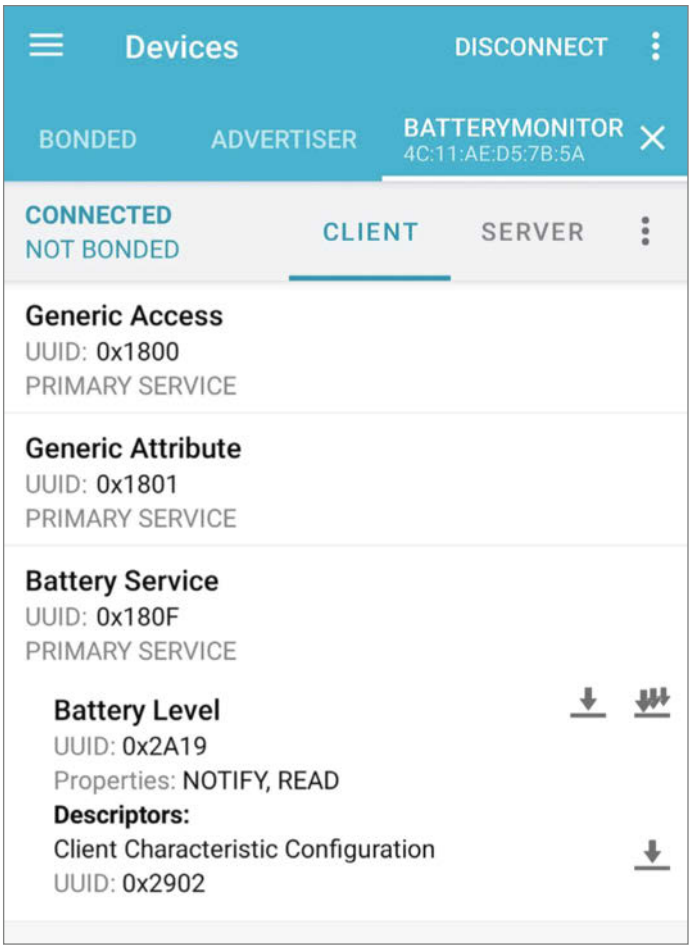


Bild 7: Unter dem Service finden wir die Charakteristiken.

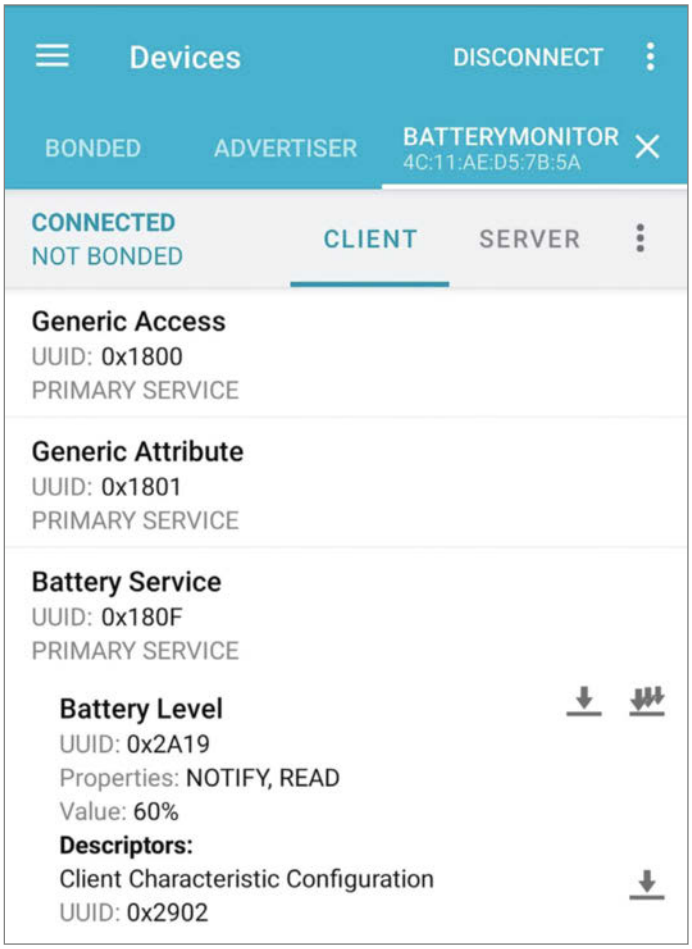


Bild 8: Den Download-Knopf mit drei Pfeilen drücken, damit ein kontinuierliches Abfragen des Wertes stattfindet

```
22  // Bluetooth® Low Energy Battery Service
23  BLEService batteryService("180F");
24
25  // battery level
26  BLEUnsignedCharCharacteristic batteryLevelChar("2A19",  // standard 16-bit characteristic UUID
27      BLERead | BLENotify); // remote clients will be able to get notifications if this characteristic changes
```

Bild 9: Im Arduino-Beispielprogramm BatteryMonitor lassen sich die UUIDs der Services und Charakteristiken eingeben.

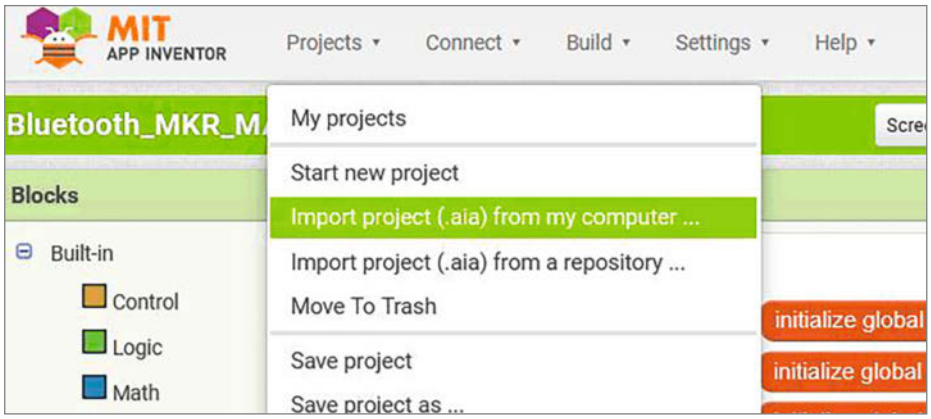


Bild 10: Hier wird mein Beispielprojekt importiert.

(Bild 11). Die entsprechende Library ist auch über den Link in der Kurzinfo abrufbar.

Ausführen der App

Um die App auf dem Smartphone zu testen, installiert man sich zuerst die „MIT App Inventor Companion"-App (siehe Link in der Kurzinfo). Mit ihr kann man erstellte Apps mit einem Klick auf das Smartphone laden. Die Apps werden bei Änderungen sofort aktualisiert und ermöglichen so ein sehr angenehmes Debugging. Alternativ kann natürlich auch eine APK-Datei erstellt und auf ein Android-Smartphone geladen und installiert werden, was für finale Versionen Sinn ergibt.

Um eine App mit der Companion-App auszuprobieren (auch als MIT AI2 Companion bezeichnet), muss ein Code eingegeben werden (Bild 12). Dieser erscheint auf der App-Inventor-Oberfläche, sobald wir auf das Menü „Connect" und dann auf „AI Companion" klicken (Bild 13).

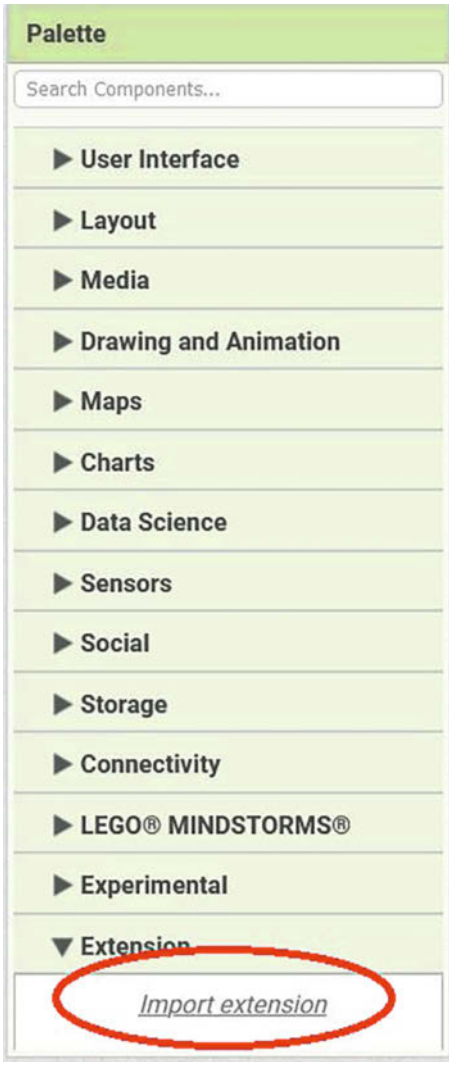


Bild 11: Falls die BLE-Erweiterung doch fehlen sollte, kann sie hier nachinstalliert werden.

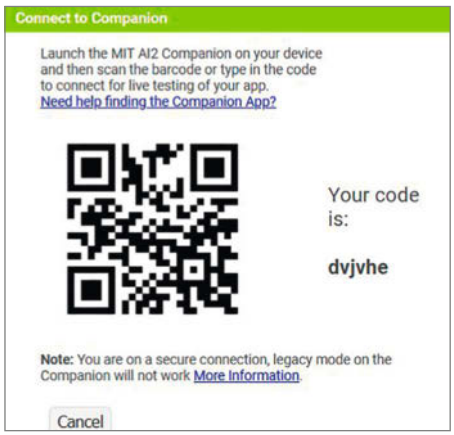


Bild 12: Der Code wird benötigt, um Smartphone und Online-Editor miteinander zu verbinden.

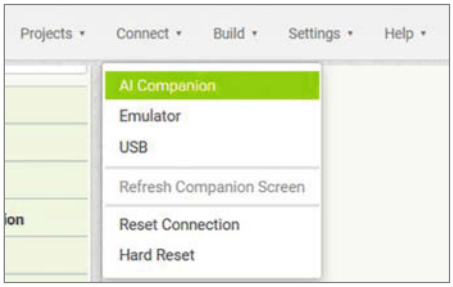


Bild 13: Mit der Companion-App wird das Debuggen einfacher.

Die App

Ganz oben rechts im Browserfenster von App Inventor befinden sich zwei unscheinbare, aber wichtige Buttons. Mit „Designer" und „Blocks" wechseln wir zwischen zwei Ansichten. Im Designer kann das Layout der App erstellt werden, also wie sie später auf dem Smartphone aussehen soll (Bild 14).

In der „Blocks"-Ansicht können wir unsere App mithilfe von „Code-Blöcken" programmieren, indem wir diese hin- und herschieben und miteinander verbinden. Hier sehen wir auch sofort die drei verschiedenen Blöcke, aus denen meine Beispiel-App gebaut ist: „Bluetooth Verbindung", „Button Aktionen" und „Cloud Anbindung" (Bild 15).

Was die drei Blöcke genau beinhalten und worauf zu achten ist, werde ich im Folgenden für jeden Block einzeln erläutern. Beginnen wir mit dem Bluetooth-Teil.

Der Bluetooth-Bereich

Der detailliertere Aufbau des Programms orientiert sich an diversen Beispielprogrammen und ist relativ übersichtlich (Bild 16). So werden zuerst einige globale Variablen initialisiert (wie genau, wird im nächsten Abschnitt erklärt). Danach folgen zwei dunkelgelbe Abschnitte für `BluetoothLE1.Connected` und für `BluetoothLE1.BytesReceived`.

Im ersten dunkelgelben Abschnitt sind die lilafarbenen Elemente entscheidend, da der Block `RegisterForBytes` definiert, wie die

Bild 14: So sieht meine Beispiel-App auf dem Smartphone aus.

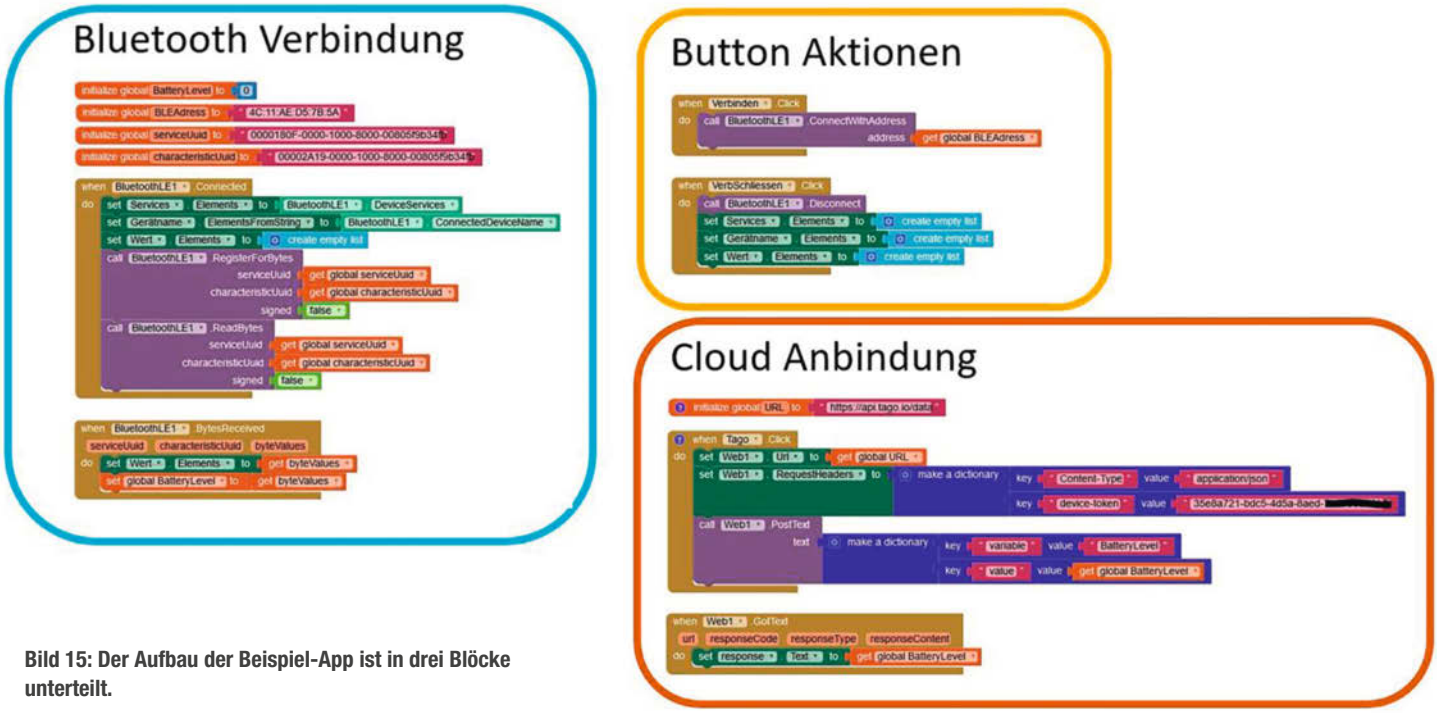


Bild 15: Der Aufbau der Beispiel-App ist in drei Blöcke unterteilt.

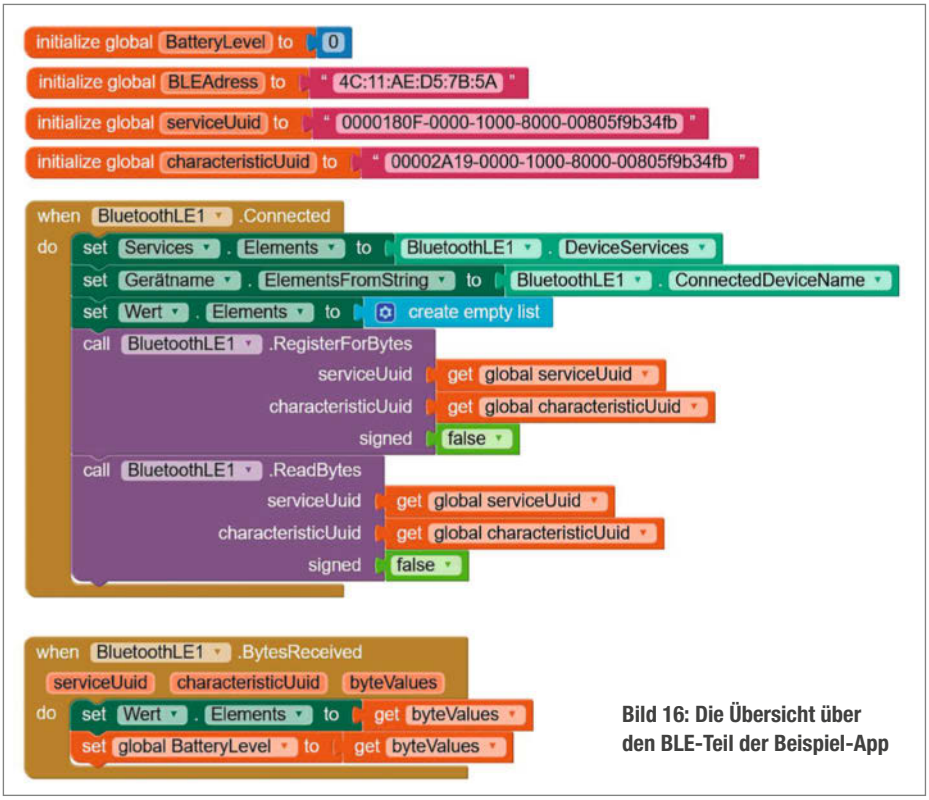


Bild 16: Die Übersicht über den BLE-Teil der Beispiel-App

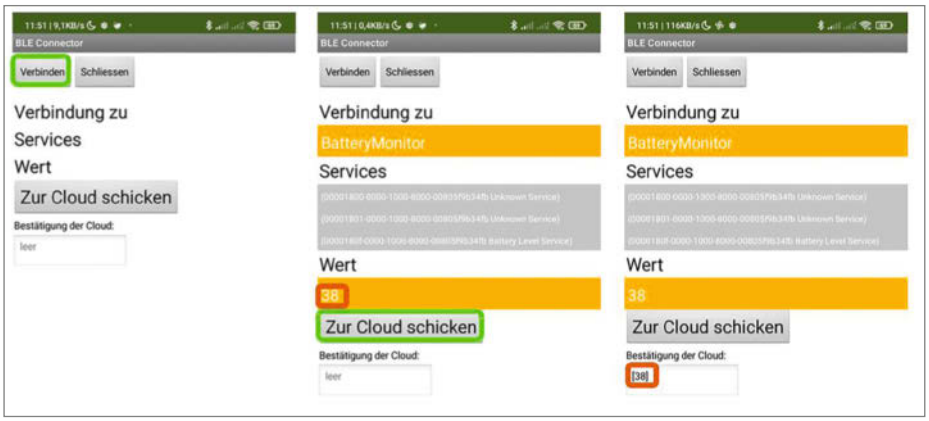


Bild 17: Der Aufbau der App: Eine BLE-Verbindung kann erstellt und abgebrochen werden. Es werden Services und der Poti-Wert angezeigt. Nach der weiteren Einrichtung kann der Wert auch in die Cloud gesendet werden.

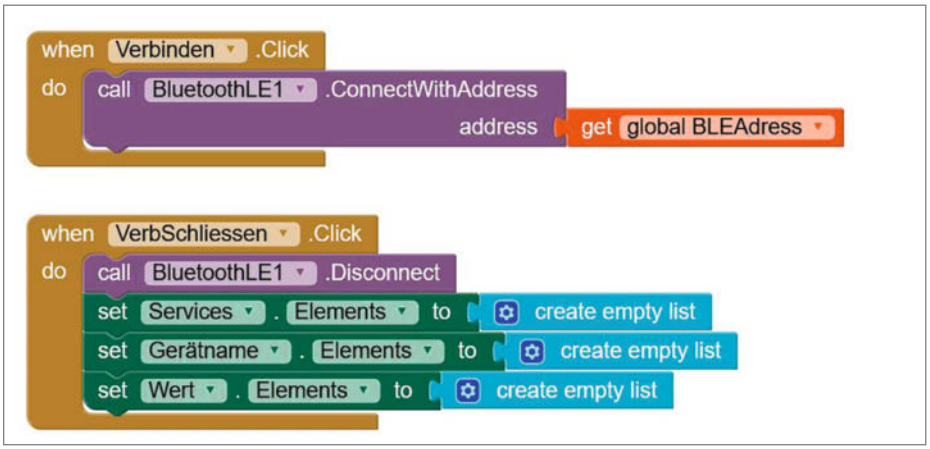


Bild 18: Die beiden Routinen der Buttons

empfangenen Daten interpretiert werden. In diesem Fall also in Bytes. Zusätzlich stellt er sicher, dass bei jeder Änderung des Wertes Bytes gelesen werden. Das heißt auch, dass der dunkelgelb gefärbte Block `BluetoothLE1.BytesReceived` dazu passen muss. Der zweite lilafarbene Block heißt `ReadBytes`. Hier werden die Daten ausgelesen und der zweite dunkelgelbe Block `BluetoothLE1.BytesReceived` getriggert. Grüne Elemente ändern die Daten in den Anzeigeelementen (z. B. Listen oder Textfeldern).

Erste Verbindung

Jetzt können wir eine erste Verbindung zwischen der App und dem Arduino herstellen. Dazu müssen einige kleine Anpassungen im Blockcode gemacht werden: Als Erstes muss die Bluetooth-MAC-Adresse des Arduino MKRs unter „initialize global BLEAdress" eingegeben werden. Man findet sie mit der „nRF Connect for Mobile"-App, wenn man nach dem Arduino sucht, und sie präsentiert sich in der Form `xx:xx:xx:xx:xx:xx`. Als Zweites müssen in den folgenden zwei Zeilen die UUID-Nummern für Service und Charakteristik definiert werden. Ich habe hier die SIG-Nummern für den BatteryMonitor verwendet und mit der Base-UUID verheiratet.

Sobald die Beispiel-App auf dem Smartphone angezeigt wird, lässt sich mit einem Klick auf „Verbinden" eine Verbindung zum Arduino über BLE aufnehmen (Bild 17). Es werden einige Daten angezeigt: der Name des Gerätes, die darauf angebotenen Services und der Wert des Potis. Dieser wird live angezeigt und ändert sich auch beim Betätigen des Potis. Sobald dann auch die Cloud-Konfiguration abgeschlossen ist (mehr dazu später), lässt sich beim Klick auf „Zur Cloud schicken" der aktuelle Poti-Wert zur Cloud senden. Ob die Übertragung erfolgreich war, können wir im Textfeld unterhalb des Buttons ablesen.

Der Button-Bereich

Jetzt wollen wir uns mit dem zweiten Block beschäftigen (Bild 18). Dieser Teil definiert, was passieren soll, wenn die beiden Buttons „Verbinden" und „Schließen" im oberen Teil der App angeklickt werden. Mit `Verbinden` wird eine Verbindung zum Arduino mit der entsprechenden Adresse aufgebaut. Mit `VerbSchliessen` wird sie getrennt und alle Text/Listen-Felder auf null zurückgesetzt.

Verbindung zur Cloud

Die meiste Arbeit haben wir mit dem dritten Block, bei dem es um die Verbindung zur Cloud geht. Zuerst müssen wir einen Cloud-Dienst auswählen, damit wir Daten in die Cloud senden können. Es gibt Tausende von Anbietern. Ich habe mich hier für Tago.IO entschieden

(siehe Link in der Kurzinfo), weil man dort kostenlos und ohne Kreditkarte bis zu zehn Geräte und Dashboards einrichten kann (Bild 19).

Die weitere Vorgehensweise ist bei den anderen Anbietern meist ähnlich und besteht aus folgenden Punkten:

– ein neues Gerät erzeugen
– dem Gerät einen Zugangscode (Access Token) geben
– das Gerät so konfigurieren, dass es die Daten richtig überträgt

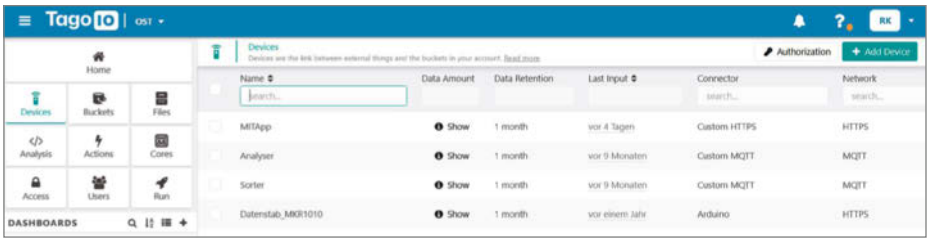


Bild 19: Der Homescreen des Online-Cloud-Dienstes Tago.IO

Mit einem Klick auf „Devices“ kann ein neues Gerät hinzugefügt werden. Es muss ein „Connector“ ausgewählt werden (Bild 20), der definiert, über welches Protokoll das Gerät kommuniziert. In diesem Falle sollte „Custom HTTPS“ genommen werden. So können über simple POST-Anfragen die Daten zu Tago.IO gesendet werden. Nach Angabe des Namens des Device ist es auch schon angelegt. Das Erstellen eines Access Tokens wurde beim Anlegen ebenfalls bereits vom System übernommen. Es muss nun nur noch kopiert wer-

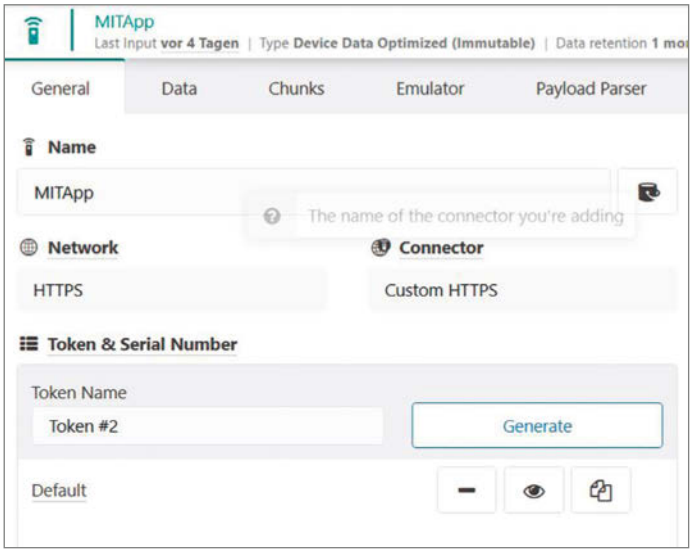


Bild 20: In Tago.IO wird ein neues Gerät angelegt.

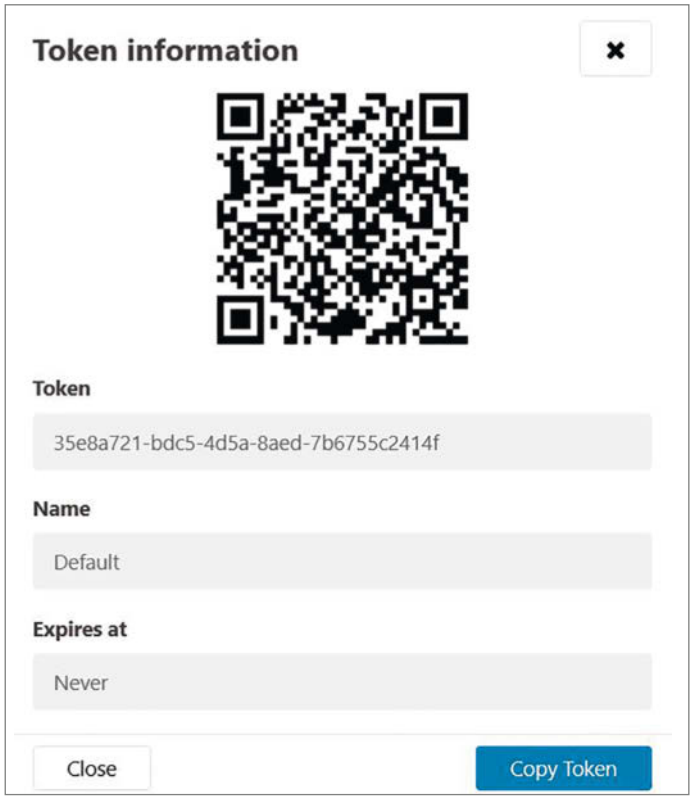


Bild 21: Nach dem Erstellen des Device ist ein Access Token verfügbar. Dieser ist unverzichtbar, um das Gerät mit der Cloud zu verbinden.

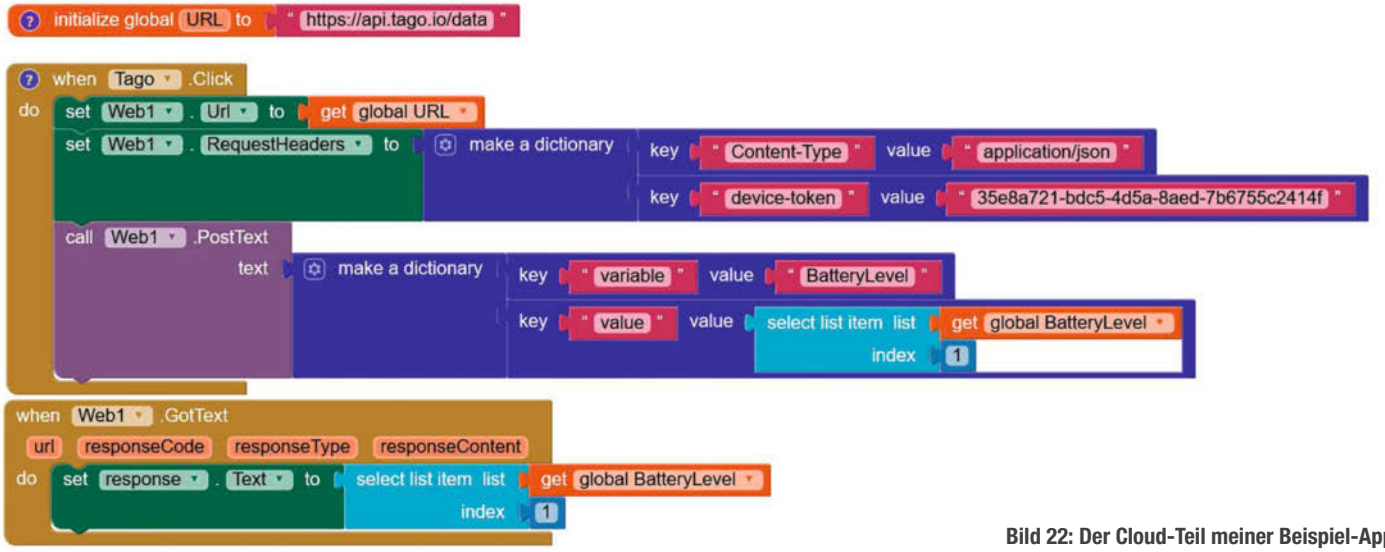


Bild 22: Der Cloud-Teil meiner Beispiel-App

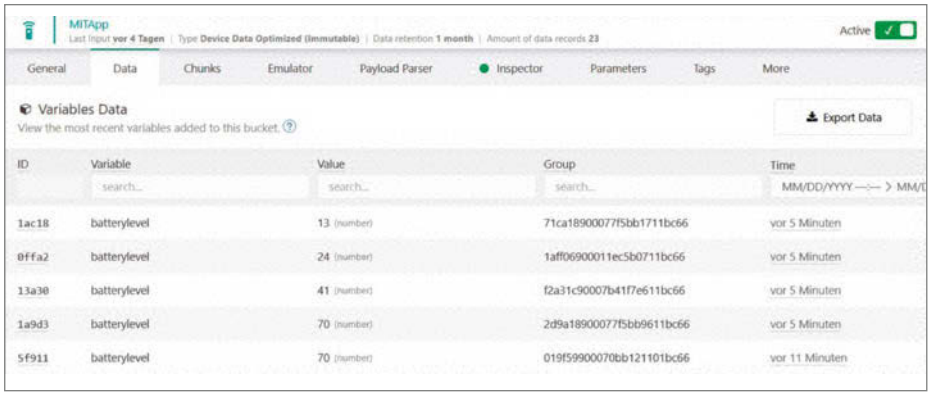


Bild 23: Der Eintrag in der Cloud-Datenbank hat funktioniert!

den. Dazu muss das Device geöffnet werden und das Token erscheint als QR-Code oder Nummer (Bild 21). Diese Nummer wird benötigt, um der App den Zugang zu gewähren.

Der Cloud-Bereich

Jetzt müssen wir uns mit einigen Einträgen im Block „Cloud Anbindung" beschäftigen (Bild 22). Mit einem einfachen HTTP-Request wird die Verbindung zur Cloud hergestellt.

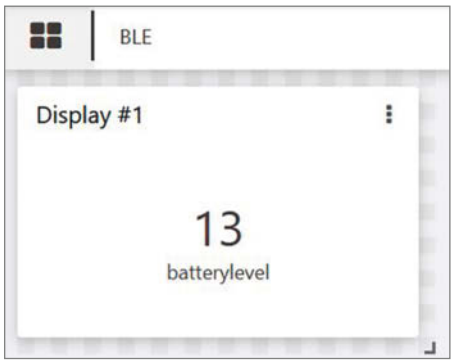


Bild 24: Anzeige des Wertes im Dashboard von Tago.IO. Dieser kann auch öffentlich gemacht werden, falls nötig.

Hier steckt der Teufel im Detail: So ist es sehr wichtig, dass die richtige URL verwendet wird. Im Fall von Tago.IO lautet diese: https://api.tago.io/data

Der zweite Stolperstein ist, dass die Authentifizierung per Token einen speziellen Eintrag im Header des Requests benötigt. Dieser heißt `device-token` und enthält den Token des zuvor auf Tago.IO erstellten Device. Falls die Daten strukturiert als JSON verschickt werden sollen, bietet sich der Eintrag `Content-Type value application/json` an.

Die blauen `make a dictionary`-Blöcke können genutzt werden, um die Daten z. B. im JSON-Format zu strukturieren. So kann im Request ein Variablenname und deren Wert mitgegeben werden. Zusätzlich ist es erforderlich, den `BatteryLevel`-Wert noch aus der Liste auszuschneiden, da die Rückgabe der Byte-Werte automatisch eine Liste erstellt.

Jetzt sollte der Übertragung nichts mehr im Wege stehen. Klickt man in der App auf dem Smartphone auf „Zur Cloud schicken", erscheint der Wert in Tago.IO. Hierzu lohnt es sich, in Tago.IO den Inspector auf „Play" zu schalten. Damit werden nun sämtliche Interaktionen mit dem Dienst angezeigt. Dies dient auch wunderbar zur Fehlersuche. In Bild 23 sieht man, dass die Variable `batterylevel` mit dem Wert 13 zur Datenbank hinzugefügt wurde.

Ein Meilenstein ist erreicht, der Workflow funktioniert:

1. Bewegung des Potis
2. Poti-Wert via BLE zum Smartphone übertragen
3. Smartphone-App empfängt die Daten
4. Daten können mit einem Button-Klick in die Tago.IO-Datenbank geschrieben werden

Nun stehen alle Wege offen, um eigene Projekte mit diesem Workflow zu erstellen und anzupassen. Ein erster nächster Schritt dürfte die Erstellung eines Online-Dashboards sein. Dazu klickt man auf dem Homescreen des Tago.IO auf das „+" bei „Dashboards". In diesem neuen Dashboard lässt sich mit einem Klick auf „Add Widget" eine Anzeige hinzufügen. Hier lassen sich dann das Gerät und die anzuzeigenden Daten auswählen. Nach dem Abschluss sollte bei jedem Klick auf den „Zur Cloud senden"-Button in der Smartphone-App der Wert des Potis in dem Widget angezeigt werden (Bild 24).

Verschiebt man nun im App Inventor den gesamten dunkelgelben Block `tago.click` (Bild 22) in den Block `whenbytes received` (Bild 16), so wird die Anzeige in Tago.IO auch ohne Klick auf den Button bei neuen Werten aktualisiert. Jede Bewegung des Potis wird automatisch in die Datenbank geschrieben.

Ausblick

Mit diesem Workflow können wir nun beliebige weitere Projekte realisieren. Mit ihm ließe sich ein neuartiger Schrittzähler bauen oder ein Onlinespiel mit einem Körpersensor steuern. Oder wie wäre es mit einem speziellen Präsenz-Detektor? Vielleicht lässt sich auch ein Herzratensensor zur Steuerung eines Geräts nutzen? Ich bin gespannt auf Anwendungen. *—mch*

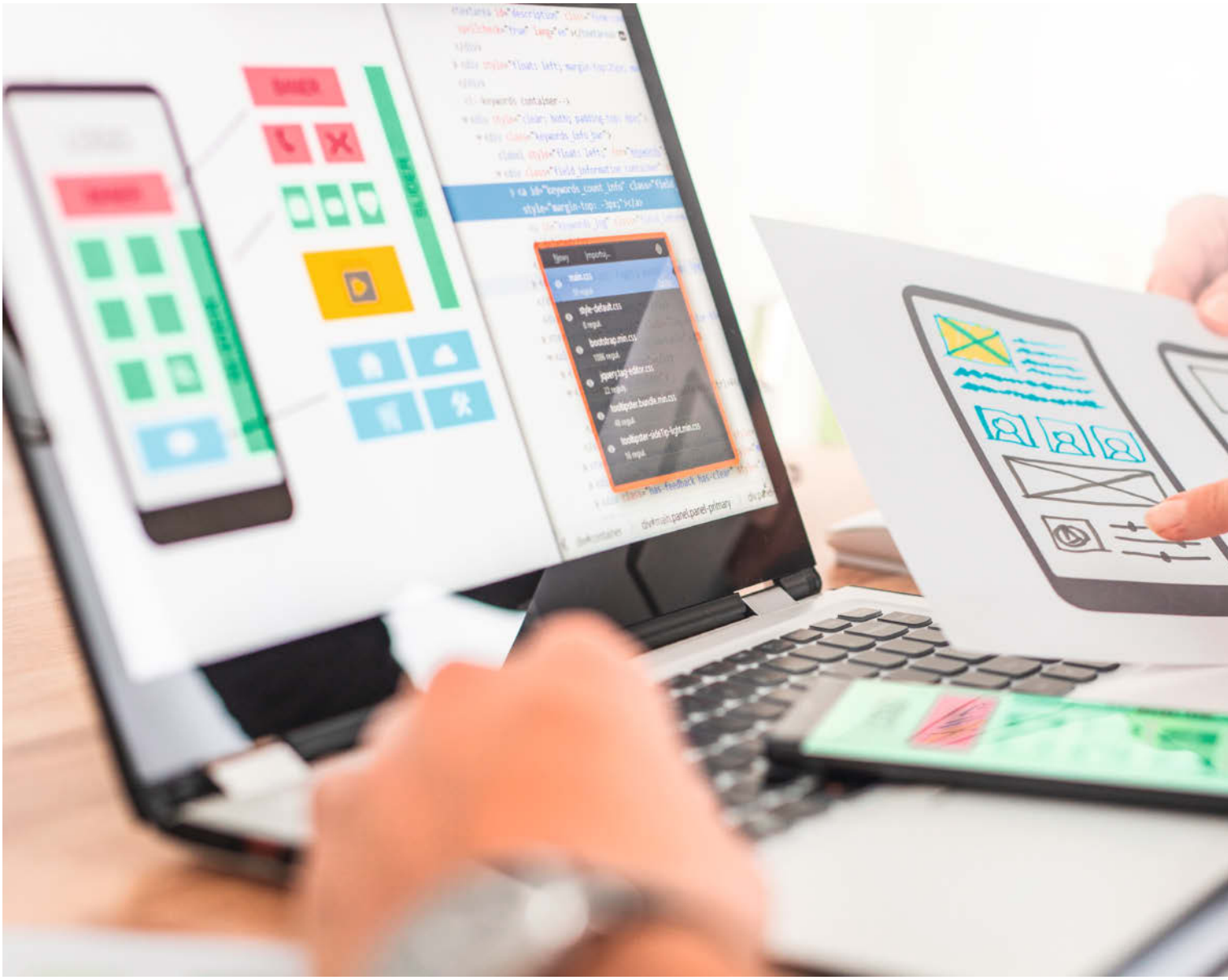
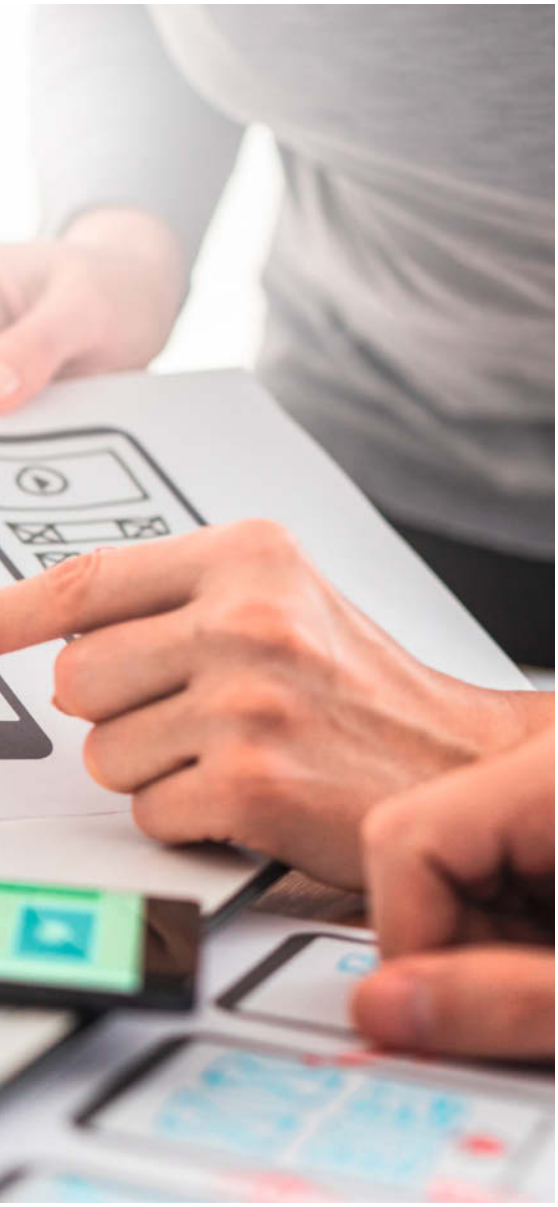


Bild: REDPIXEL.PL/Shutterstock.com

Reißbrett für Android-Apps

Für so manch ein Projekt ist eine passende, maßgeschneiderte Android-App eine große Bereicherung. Allerdings ist die Programmierung solcher Anwendungen üblicherweise recht aufwendig. Mit den richtigen Helfern wird die App-Erstellung aber fast zum Kinderspiel.

von Maik Schmidt



Die native Entwicklung von Android-Apps ist wirklich kein Zuckerschlecken. Sie setzt ordentliche Kenntnisse der Programmiersprachen Kotlin oder Java voraus und sie erfolgt am besten in der Entwicklungsumgebung Android-Studio, die nicht ganz einfach zu erlernen und zu meistern ist. Obendrauf müssen App-Entwickler noch das Android-Framework selbst kennen und das ist ebenfalls ein ziemlicher Brocken. Wer all das beherrscht, kann dann aber so gut wie jede App fürs Smartphone oder Tablet programmieren.

In vielen Fällen sind so viele Freiheitsgrade aber gar nicht notwendig und daher existieren eine Reihe von Generatoren, die einfach strukturierte Anwendungen mit nur wenig oder sogar ganz ohne handgeschriebenen Code erzeugen können. Ferner gibt es Projekte, die einen gehörigen Teil der Komplexität nativer Android-Entwicklung hinter einfacheren Programmiersprachen und -umgebungen verstecken.

Thunkable

Bereits seit 2016 bietet die Firma Thunkable aus San Francisco eine No-Code-Plattform zur Generierung von Android- und iOS-Apps an. Um die kostenlose Variante des Produkts nutzen zu können, ist eine Anmeldung auf der Website erforderlich. Thunkable unterstützt Google- bzw. Apple-Accounts und darüber hinaus die Registrierung per E-Mail-Adresse. Bei letzterer ist kein Passwort notwendig, denn Nutzer bleiben automatisch für 30 Tage angemeldet. Danach ist eine erneute Anmeldung mit derselben E-Mail-Adresse fällig.

Die Website bietet zwei Werkzeuge zur Erstellung einer Smartphone-App, nämlich einen Design-Editor (Abbildung 1) und einen sogenannten Block-Editor. Der Design-Editor dient zur Gestaltung der App-Oberfläche. Hier stehen viele gängige UI-Elemente (UI: User Interface) zur Auswahl und Entwickler können sie beinahe beliebig auf dem virtuellen Smartphone anordnen und konfigurieren.

Zu jeder Komponente der App-Oberfläche gehört ein Stückchen Code, das die App für zuvor definierte Ereignisse ausführt. Beispielsweise kann dieser Code ausgeführt werden, sobald ein Nutzer einen Button in der App berührt. Zur Erstellung des Codes dient eine grafische Programmiersprache, die stark an MIT-Scratch erinnert.

Nach einer kurzen Einarbeitungszeit geht die Arbeit mit den Editoren gut von der Hand und es ist problemlos möglich, komplexe Anwendungen innerhalb von Minuten zusammenzuklicken. Diese Apps verfügen nicht nur über ein ansprechendes UI, sondern sie können auch anspruchsvolle Logik implementieren. Das liegt nicht zuletzt an der einfachen Integration mit Web-APIs und verschiedenen Datenquellen wie Google Sheets. Selbst eine Integration mit OpenAI ist möglich.

Zwar liegt der Fokus von Thunkable nicht auf Apps, die mit Mikrocontrollern zusammenarbeiten, aber es ist durchaus machbar. Beispielsweise bietet Thunkable eine Komponente zur Kommunikation per Bluetooth Low Energy (BLE). Diese Komponente liefert eine Vielzahl von Funktionen, um BLE-Geräte zu finden und um mit ihnen zu kommunizieren. Es dauert nur wenige Minuten, um eine Smartphone-App, welche die Namen aller BLE-Geräte in der Umgebung anzeigt, zu erstellen und aufs Telefon zu bringen (Abbildung 2).

Das liegt zum einen an der ausgezeichneten Dokumentation, aber auch an den vielfältigen Möglichkeiten zum Testen. Unter anderem gibt es in Thunkable eine Web-Preview, mit der Entwickler den aktuellen Stand ihrer Kreationen im Browser ausprobieren können. Das ist natürlich nur ein Emulator, der sich von der App auf einem echten Smartphone unterscheidet – schon allein deswegen, weil die meisten PCs keinen Touch-Screen haben. Schwerwiegender ist aber, dass der Emulator manche Hardware-Eigenschaften wie BLE nicht bereitstellt.

Daher ist es sinnvoll, so früh wie möglich mit Tests auf echten Endgeräten zu beginnen. Dabei hilft die „Thunkable Live"-App, mit der Entwickler die Projekte in ihrem Account auf dem Smartphone ausprobieren können. Wer sich mit seinem Google- oder Apple-Konto registriert hat, kann direkt loslegen. Alle anderen geben ihre E-Mail-Adresse an und bekommen anschließend einen Code, den sie auf der Website eingeben müssen.

Das ganze Verfahren klappt tadellos und die App ist sowohl flott als auch komfortabel. Einer schnellen Entwicklung und ausgiebigen Tests steht somit nichts im Weg.

Als weitere Möglichkeit bietet Thunkable den Download einer APK-Datei der eigenen App an (Abbildung 3). Diese können Entwickler auf dem Gerät speichern und manuell installieren. Dazu ist es notwendig, die Installation von Apps aus nicht vertrauenswürdigen Quellen zuzulassen. Auch ist in den meisten Fällen die Installation eines alternativen Browsers wie Firefox auf dem Android-Gerät notwendig, weil Chrome sich weigert, die APK-Datei herunterzuladen. Für kostenlose Accounts ist das Herunterladen von APK-Dateien nur zweimal pro Monat erlaubt und

Kurzinfo

- Übersicht: Einfach Android-Apps erstellen
- Bedienoberfläche der App-Generatoren
- Funktionen mit Code oder No-Code

Mehr zum Thema

- Rütten, Stepien, Weller, IoT-Wecker im Retro-Stil, Make 06/23, S. 98
- Bernd Heisterkamp, Smarte Werkstattboxen, Make 5/23, S. 38
- Carsten Wartmann, Internet-of-Things-Dienste für Maker, Make 3/21, S. 32

Alles zum Artikel im Web unter
make-magazin.de/x5t7

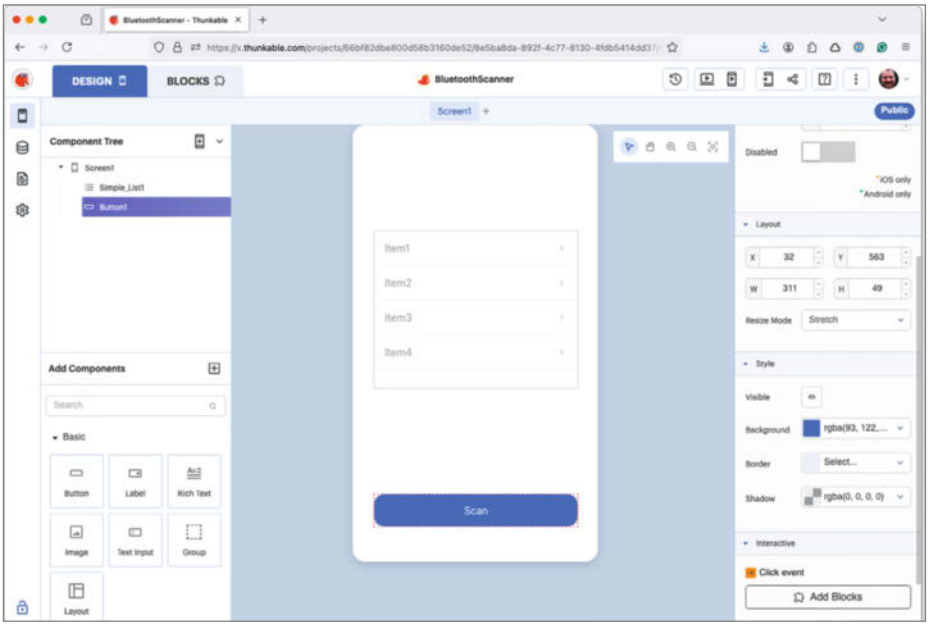


Abbildung 1: Thunkables Design-Editor ist nicht nur schick, sondern auch einfach zu bedienen.

es kann schon mal eine halbe Stunde dauern, bis die E-Mail mit dem Download-Link ankommt.

Schließlich gibt es sogar die Möglichkeit, die eigene App bei Google Play, im Apple Store oder als Webapplikation zu veröffentlichen. Das funktioniert allerdings nur mit einem Pro-Account.

Ein solcher Account schlägt mit 38 US-Dollar pro Monat zu Buche und dürfte sich für Hobbyisten kaum lohnen. Mit dem kostenlosen Paket lassen sich immerhin zehn öffentliche Projekte anlegen, die jeder Thunkable-Nutzer sich ansehen kann; das sollte für erste Experimente reichen. Wer seine Projekte lieber für sich behalten möchte, kann das mit dem Starter-Tarif für 13 US-Dollar im Monat erreichen.

Das ist alles nicht ganz günstig, aber es passt zur Zielgruppe von Thunkable, die scheinbar im Wesentlichen aus Unternehmen besteht, die ein paar einfache mobile Apps brauchen, aber dafür keine Entwickler einstellen wollen. Trotzdem eignet sich Thunkable hervorragend für Mikrocontroller-Projekte, solange die verwendeten Geräte mit WLAN oder BLE ausgestattet sind.

RemoteXY

Ganz ähnlich wie Thunkable funktioniert RemoteXY, aber bei diesem Projekt stehen Mikrocontroller deutlich im Vordergrund. Entwickelt und betrieben wird es seit 2014 von einem einzelnen Entwickler.

Um eine Anwendung mit RemoteXY zu erstellen, ist eine Anmeldung auf der Website nicht zwingend erforderlich. Sie ist aber kostenlos und ermöglicht, Projekte zu speichern und zu laden.

Die Erstellung eines Projekts beginnt im UI-Editor (Abbildung 4) , in dem die Bedienoberfläche entsteht, die am Ende eine Mikrocontroller-Anwendung steuert. Zwar gibt es nicht allzu viele UI-Elemente, aber für die Mehrheit typischer Anwendungen reichen sie allemal.

Außerdem ist es leicht möglich, eine App mit mehreren Seiten zu erstellen. RemoteXY unterstützt sowohl vertikale als auch horizontale Layouts.

Jedes UI-Element kontrolliert später eine oder mehrere Variablen in der Mikrocontroller-Anwendung. Ein Schalter kann zum Beispiel mit einer Variablen verknüpft werden, die den aktuellen Zustand eines GPIO-Pins enthält. Je nach Schalterstellung kann der Pin dann im Zustand HIGH oder LOW sein.

Ist die Oberfläche fertig, muss der Entwickler noch festlegen, mit welchem Mikrocontroller die dazugehörige Anwendung laufen soll, und den Kommunikationsweg konfigurieren, über den das Smartphone und der Controller Daten austauschen sollen. Die Smartphone-App kann etwa per Bluetooth, Bluetooth Low Energy, WLAN und USB On-The-Go mit dem Mikrocontroller kommunizieren.

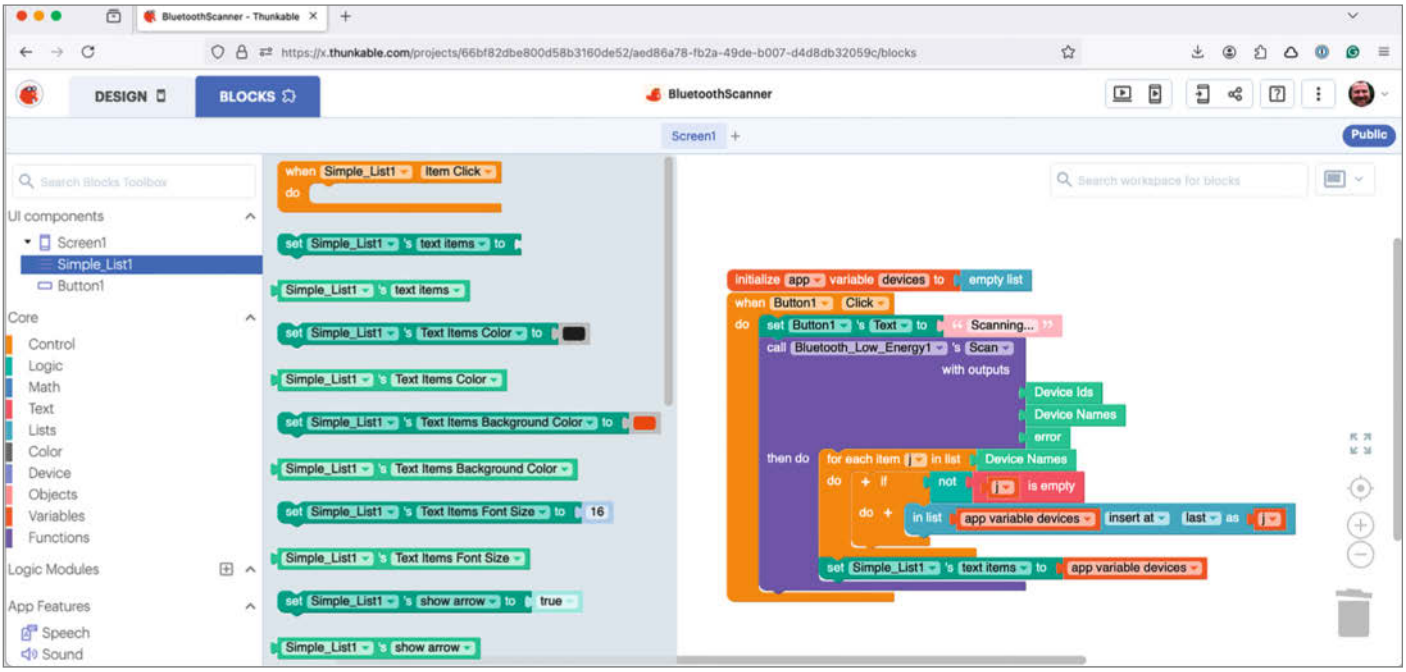


Abbildung 2: Wenige Mausklicks reichen für eine Thunkable-App, die nach BLE-Geräten in der Umgebung sucht.

RemoteXY unterstützt insbesondere den ESP32, der idealerweise WLAN und Bluetooth schon an Bord hat. Für andere Boards wie den Arduino Uno und ähnliche sind entsprechende Shields oder Breakout-Boards vonnöten.

Sobald die Konfiguration abgeschlossen ist, erzeugt RemoteXY den Mikrocontroller-Code im Zip-Format. Den kann man wie gewohnt mit der Arduino IDE übersetzen und aufs Board spielen. Allerdings ist zunächst nicht viel zu sehen, denn man muss noch die RemoteXY-App (Abbildung 5) aus dem Google Play Store auf sein Smartphone laden. Diese sucht nach kompatiblen Geräten in der Umgebung und liest deren UI-Konfiguration aus. Anschließend erscheint die zuvor im Design-Editor definierte Oberfläche auf dem Smartphone.

Im Heft sind weitere Artikel (siehe S. 10, 30 und 40), die RemoteXY im Detail anhand von Beispielen erklären. Das Tool lässt sich mit bis zu fünf UI-Elementen kostenfrei nutzen. Gelangt man darüber, benötigt man die Pro-Version, ansonsten beendet sich die selbst erstellte App nach 20 Sekunden. RemoteXY-Pro kostet auf Android entweder 1,09 Euro im Monat oder einmalig 21,99 Euro. Unter iOS gibt es nur eine Abo-Option für 3, 6 oder 12 Monate (für 2,99/5,99/8,99 Euro).

Blynk

In einer etwas höheren Liga will Blynk, eine Firma aus Miami, spielen, die bereits seit 2014 am Markt ist. Deren Produkt ist eine komplette IoT-Plattform (Internet of Things), die weit über einen App-Generator hinausgeht. Blynk wendet sich daher eher an mittlere und große Unternehmen, die viele IoT-Geräte provisionieren, verwalten und aktualisieren müssen.

Solche Anwendungsfälle gibt es in der Heim-Automatisierung zuhauf. Hersteller, die etwa ein vernetztes Heizungsthermostat entwickeln, finden mit der Blynk-Plattform nicht nur eine gute Umgebung für Prototypen, sondern durchaus auch für ein finales Produkt.

Die Plattform (Abbildung 6) kümmert sich im Hintergrund um all die „langweiligen" und zeitintensiven Aufgaben, wie die Visualisierung von Messdaten oder Firmware-Aktualisierungen. Auch die Verwaltung von Rollen und Berechtigungen kann sie übernehmen und erlaubt nebenher die Automatisierung wiederkehrender Aufgaben.

Dazu betreibt Blynk eine eigene Cloud-Umgebung, mit der Geräte über verschiedene Protokolle wie MQTT, HTTP, HTTPS oder das hauseigene Blynk-Protokoll kommunizieren können. Als Übertragungsmedium kommen WLAN, Ethernet, Mobilfunk, LoRaWAN oder Satellitenverbindungen infrage.

IoT-Geräte können auf diese Weise Daten an die Blynk-Cloud senden und Benutzer umgekehrt Geräte über die Cloud kontrollieren und konfigurieren. Das klappt sowohl über einen Webbrowser als auch über eine Smartphone-App.

Dazu muss auf dem IoT-Gerät eine Software laufen, die mit der Blynk-Cloud kommunizieren kann. Wegen der Unterstützung von MQTT und HTTP kann prinzipiell jedes Gerät mit einer Internetverbindung den Dienst nutzen und Blynk stellt für populäre Mikrocontroller fertige Bibliotheken bereit. Die Liste der unterstützten Boards ist erfreulich lang: Unter anderem gehören diverse Arduinos sowie Boards mit ESP8266 und ESP32 dazu.

Die Voraussetzungen zur Entwicklung spannender Projekte mit Blynk sind also nicht schlecht, aber bei null anzufangen, ist immer schwer. Deshalb bietet der Hersteller ein paar Blueprints (Blaupausen) für Projekte an, die den Einstieg erleichtern (Abbildung 7).

Diese Blaupausen enthalten Templates, also Schablonen für bestimmte Geräteklassen. Für Projekte mit einer großen Anzahl von Geräten, die im Kern alle gleich sind, aber ein wenig anders konfiguriert werden müssen, ist das eine feine Sache.

Eine besonders einfache Blaupause steuert die Status-LED auf einem ESP32-Board und dieses Beispiel eignet sich hervorragend, um das Blynk-Ökosystem kennenzulernen. Vieles läuft dabei weitestgehend automatisch.

Nachdem der Nutzer das Beispiel ausgewählt hat, muss er nur noch einem Assistenten und der Dokumentation folgen. Eine wichtige Voraussetzung ist die Arduino IDE und die Installation der Blynk-Bibliothek über den Bibliotheksverwalter. Blynk erzeugt automatisch einen Arduino-Sketch (Abbildung 8), den man als Zip-Datei herunterladen kann. Ferner gibt Blynk zwei `#define`-Anweisungen vor, die im

Abbildung 3: Die erste Thunkable-App findet BLE-Geräte.

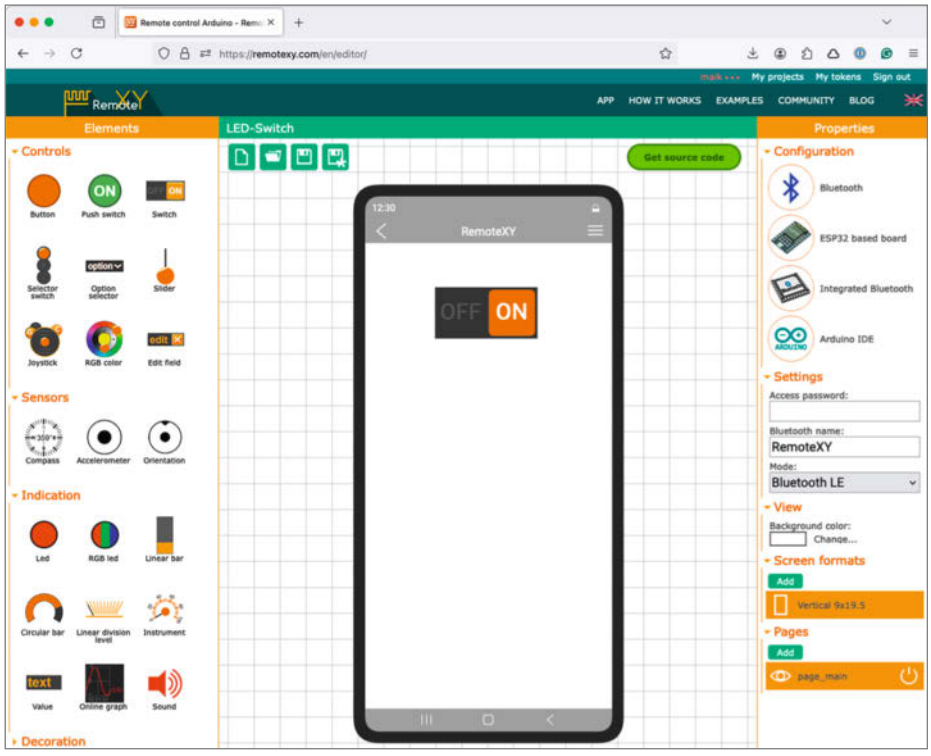


Abbildung 4: Der RemoteXY-Editor ist übersichtlich und funktional.

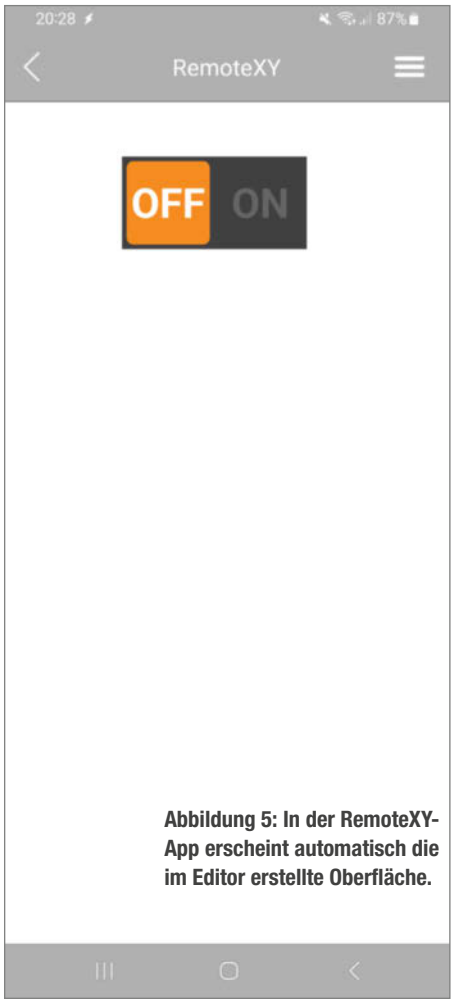


Abbildung 5: In der RemoteXY-App erscheint automatisch die im Editor erstellte Oberfläche.

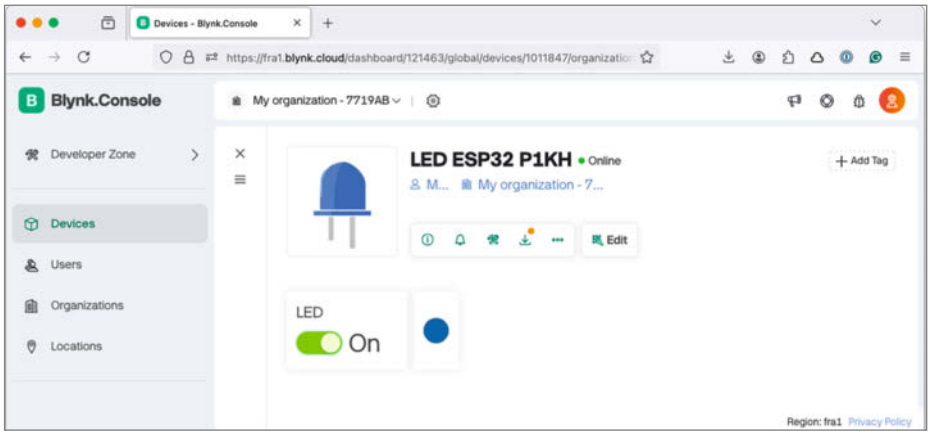


Abbildung 6: Die Verwaltung der Blynk-Geräte im Browser ist einfach und komfortabel.

Sketch zu ersetzen sind. Diese definieren den Namen und die ID des Templates.

Beim Übersetzen in der Arduino IDE muss man darauf achten, nur ESP32-Cores bis Version 2.0.14 zu verwenden. Neuere funktionieren nämlich derzeit nicht. Dieser lässt sich im Boardverwalter entsprechend herunterladen.

Sobald die Software auf dem Board läuft, fehlt nur noch die Installation der Blynk-App (Abbildung 9) auf dem Smartphone. Nach der Anmeldung läuft automatisch die Suche nach Blynk-Geräten in der Umgebung (per Bluetooth) und das klappt alles reibungslos. Die App fragt noch nach dem WLAN-Passwort und anschließend steht der Kontrolle der Status-LED per Web und per App nichts mehr im Wege.

Es ist erstaunlich, wie stabil das ganze System ist. In meinen Versuchen funktionierte die Steuerung der Status-LED (Abbildung 10) in der App und im Web ohne irgendwelche Schwierigkeiten oder für mich merkbare Latenzen.

Sowohl im Web als auch in der App ist aber noch viel mehr möglich. Im Grunde sind die Blynk-Apps nur Dashboards (Abbildung 11), die den aktuellen Zustand einer Anwendung visualisieren. Daher ist es ein Leichtes, diese Dashboards um weitere Widgets (Abbildung 12) anzureichern. Die meisten dieser Widgets gibt es allerdings nur in den kostenpflichtigen Versionen.

Die Software auf dem Mikrocontroller kommuniziert mit der App auf dem Smartphone über sogenannte Datastreams (Datenströme). In der kostenlosen Variante können das bis zu fünf Ströme sein und ein solcher Strom kann zum Beispiel als virtueller Pin fungieren. Das ist beim Status-LED-Beispiel der Fall, denn hier gibt es einen virtuellen Pin, der den GPIO-Pin der Status-LED auf dem ESP32-Board repräsentiert.

Dieses Verfahren ist recht clever, denn die Blynk-Apps arbeiten nur mit virtuellen Pins, welche die Mikrocontroller-Anwendungen auf die echten Pins abbilden müssen. Das macht es leicht, Blynk-Anwendungen auf andere Mikrocontroller zu portieren.

Datenströme sind aber nicht auf digitale Pins beschränkt und sie funktionieren unter anderem auch mit analogen Pins oder ganz anderen Datenquellen. Sowohl die UI-Elemente als auch die Mikrocontroller-Anwendung können sich auf diese Datenströme beziehen und Daten miteinander austauschen.

Das kostenlose Angebot reicht für erste Tests, ist aber ansonsten stark eingeschränkt. Allerdings gibt es einen günstigen Maker-Tarif für 6,99 US-Dollar pro Monat, der schon viel mehr bietet als die Free-Version. Die nicht eingeschränkte Variante für Profis kostet dann aber schon ab 99 US-Dollar im Monat. Die genauen Fähigkeiten der Varianten sind auf den Seiten von Blynk aufgelistet.

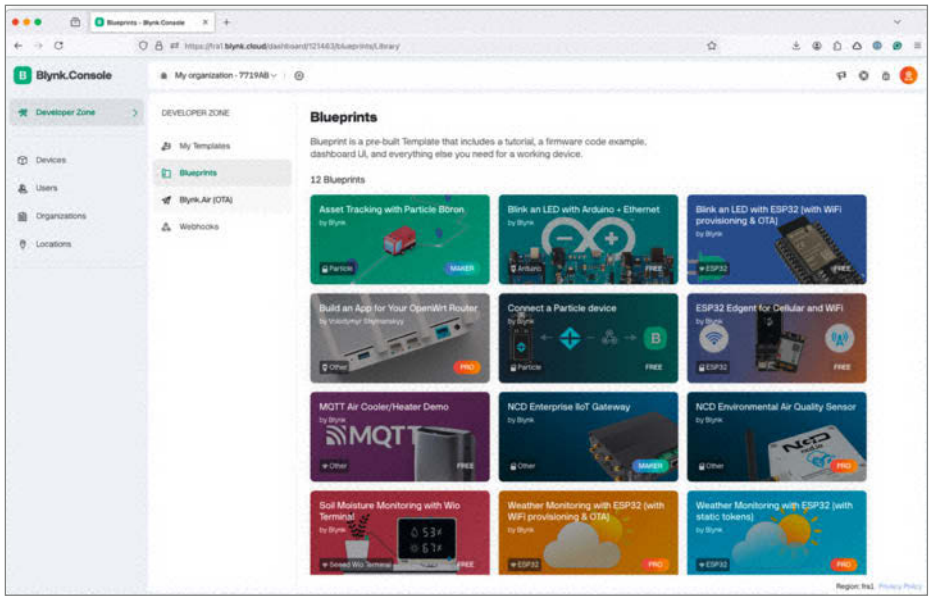


Abbildung 7: Blaupausen für Apps erleichtern den Einstieg in die Blynk-Umgebung.

B4A

Einen anderen Weg zur Vereinfachung der App-Entwicklung beschreitet die Firma Anywhere Software mit ihren B4X-Produkten. B4X ist so etwas wie eine moderne Version der

beliebten Programmiersprache Visual Basic (Classic). Sie steht im Zentrum von vier Werkzeugen, deren Namen in gewisser Weise konsistent, aber leider nicht sonderlich eingängig sind.

Da ist zum Beispiel B4R, das die Programmierung von B4X-Applikationen auf Mikrocontrollern ermöglicht. Ferner gibt es B4J zur Erstellung von Desktop- und Server-Anwendungen, und B4I dient zur Entwicklung von iOS-Apps. Für diesen Artikel ist B4A (Abbildung 13) besonders interessant, denn damit können Entwickler Android-Apps mittels B4X programmieren.

B4A besteht aus mehreren Komponenten: Zentral ist die B4A-IDE zur Entwicklung von Android-Apps mittels B4X. Diese IDE ist nicht ganz so umfangreich wie Android Studio, aber das ist in diesem Fall kein Makel. Schließlich erleichtert sie den Einstieg in B4A und B4X.

Für die Installation gibt es einen Installer, aber die erforderlichen Java- und Android-Kommandozeilen-Tools müssen die Nutzer separat installieren.

Abbildung 8: Der Code einer LED-Blink-Blynk-App ist recht überschaubar.

Abbildung 9: Die Android-Blynk-App findet Geräte in der Umgebung schnell und zuverlässig.

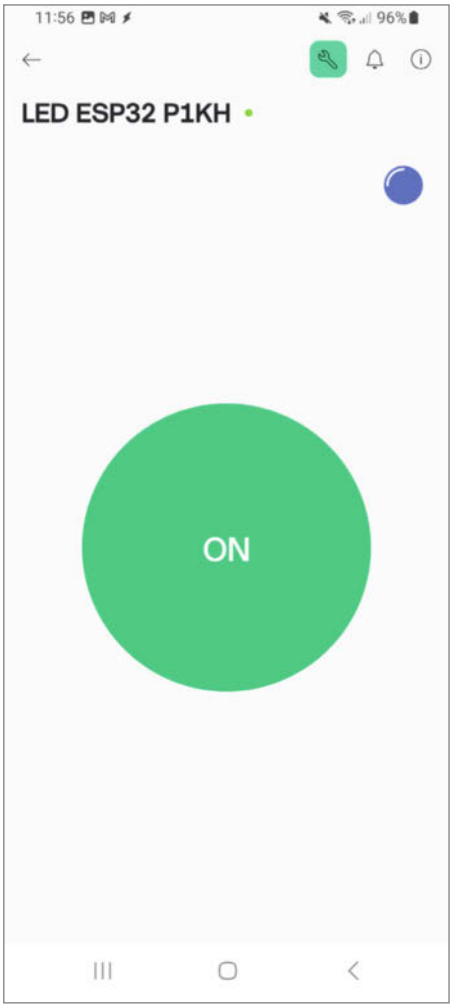


Abbildung 10: Das „Hello World!“ der Mikrocontroller unter Blynk

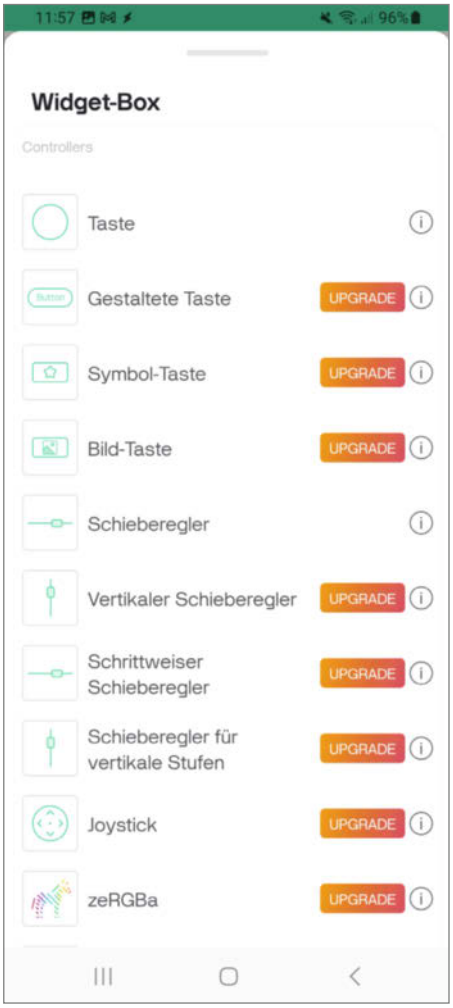


Abbildung 12: Das Erstellen einer Blynk-App ist sogar auf dem Smartphone möglich.

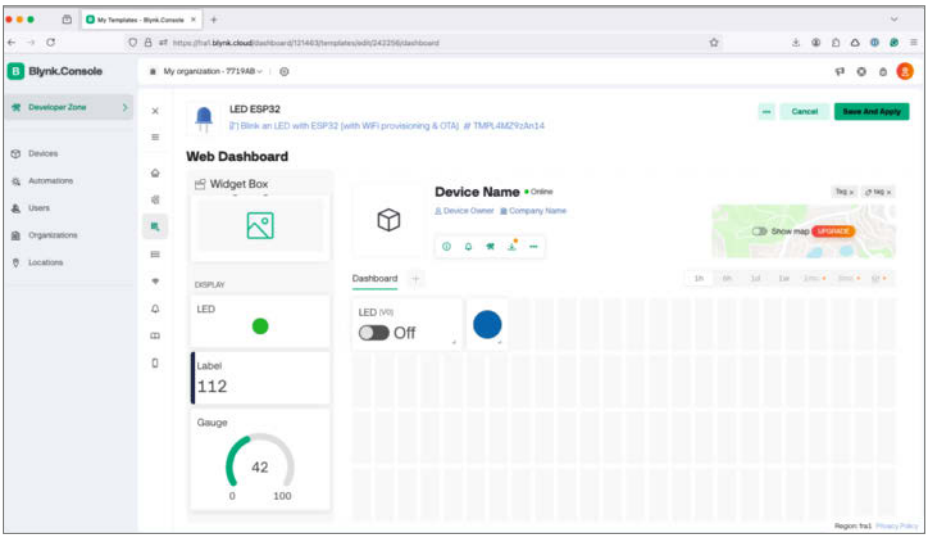


Abbildung 11: Der Dashboard-Editor von Blynk ist übersichtlich.

Neben einem komfortablen Code-Editor enthält die B4A-IDE (Abbildung 14) ein Design-Werkzeug zur Erstellung von Android-Oberflächen. Das Tool ist ebenfalls ein wenig eingeschränkt, bietet aber ausreichend Funktionen und Widgets für anspruchsvolle Anwendungen.

Besonders gelungen ist die Umsetzung eines prototypischen Entwicklungsprozesses, denn die IDE kann sowohl per USB als auch drahtlos mit dem Smartphone (Abbildung 15) kommunizieren. Letzteres funktioniert mit der B4A-App, die mit der IDE über ihre IP-Adresse verbunden werden kann. Anschließend sind Änderungen im Design-Editor sofort auf dem Android-Gerät sichtbar und das Ganze funktioniert sogar umgekehrt. Das heißt, Entwickler können UI-Elemente auf dem Smartphone hinzufügen oder verschieben und diese Änderungen spiegelt die IDE in Echtzeit wider.

Beim Öffnen eines neuen Projekts stellt die B4A-IDE automatisch Code für eine Beispielanwendung bereit, die nur einen Button enthält (Abbildung 16). Wenn ein Nutzer auf diesen tippt, gibt die App den Text „Hello world!" aus (Abbildung 17). Es reicht ein Mausklick, um diese App zu übersetzen und per USB oder WLAN aufs Smartphone zu spielen. Zusammen mit der sehr ausführlichen Dokumentation ist es darüber hinaus einfach, die Beispielanwendung zu verstehen und größere Projekte zu wagen.

Allerdings gilt das nur für erfahrene Entwickler, denn B4A-Apps müssen textuell programmiert werden. Zwar ist die Programmiersprache B4X einfacher zu lernen als Java oder Kotlin, aber die Komplexität des Android-Frameworks kann sie nicht gänzlich verbergen. Bis auf B4I sind übrigens alle B4X-Produkte kostenlos.

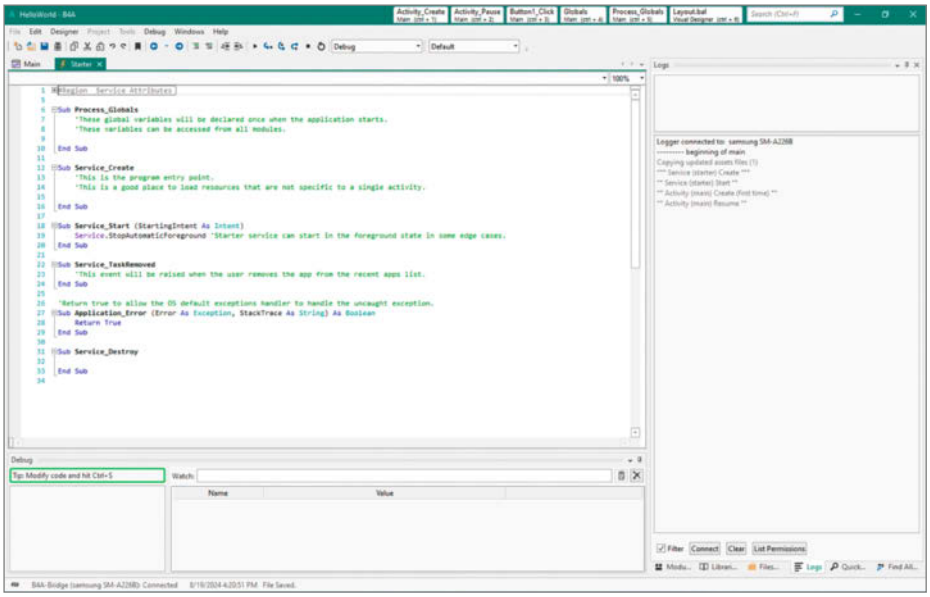


Abbildung 13: Die B4A-IDE bietet eine Menge an Funktionen.

Es gibt noch mehr

Für beinahe jede Programmiersprache existiert mindestens ein Framework zur Erstellung von Android-Apps. Manchmal wurden diese Frameworks extra für Android entwickelt, aber meistens funktionieren sie auch für andere Zielumgebungen, wie iOS, Windows oder macOS.

Ein gutes Beispiel ist Kivy. Das ist ein Python-Projekt zur Erstellung grafischer Anwendungen auf einer Vielzahl von Betriebssystemen – und Android ist eines davon. Wer mit Kivy arbeiten will, muss die Programmiersprache Python beherrschen. Ferner erfordert die Umwandlung einer Kivy-App in eine Android-App einige Tools, die nicht auf allen Betriebssystemen gleich gut funktionieren. Prinzipiell ist es also möglich, Kivy-Apps auf Android-Geräten auszuführen, aber es sollte nicht die erste Wahl für jemanden sein, der nur Android-Apps entwickeln möchte.

Ganz ähnlich sieht die Situation bei Processing for Android aus. Processing ist eine Programmiersprache, die irgendwo zwischen Java und JavaScript angesiedelt ist und sie ist in erster Linie für grafische Anwendungen gedacht. Oft basieren diese Anwendungen auf Sensor-Daten, die ein Mikrocontroller liefert. Für Desktop-Anwendungen ist Processing ziemlich ausgereift und Processing for Android macht es möglich, Processing-Anwendungen auf Android-Geräten auszuführen. Auch hier ist die Integration nicht immer ganz einfach, denn zum einen sind solide Kenntnisse der Processing-Sprache wichtig und zum anderen sind die Tools zur Entwicklung und Paketierung von Android-Apps nicht allzu komfortabel.

Und sonst?

Neben all den Frameworks und Diensten existieren weitere Möglichkeiten, um Anwendungen auf Android-Geräte zu bringen. Eine davon sind Progressive Web Applications (PWA), die mittlerweile eine ganze Menge an Funktionalitäten bieten. Im Kern sind sie Webanwendungen, die auf HTML, CSS und JavaScript basieren. Für Nutzer sehen sie aber wie native Android-Apps aus und haben auch Zugriff auf viele Geräte-Hardwares und Eigenschaften.

Des Weiteren existieren unzählige Programmiersprachen, die als Apps auf Android-Geräten laufen. Von Forth über Lisp bis hin zu Basic ist alles vertreten. Aber nicht immer integrieren sich die Systeme gut in das Android-Ökosystem. Insbesondere bei Anwendungen, die mit

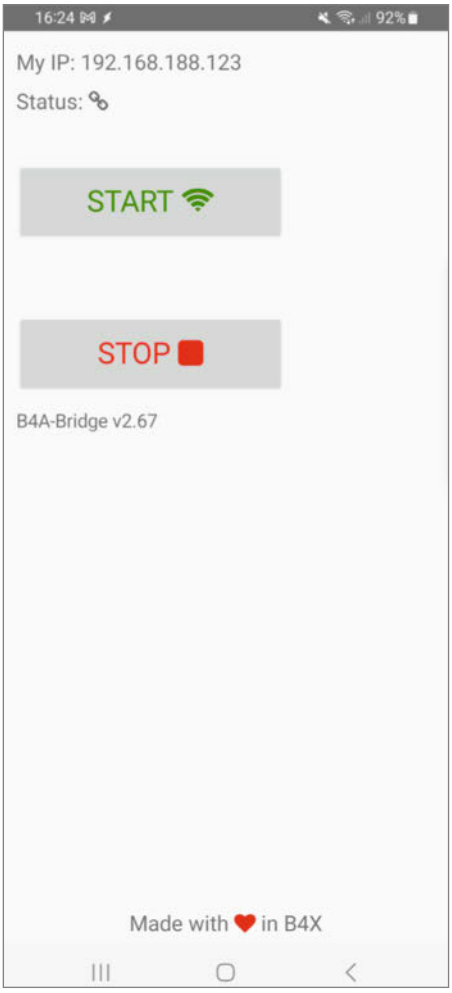


Abbildung 15: Die B4A-IDE und das Smartphone finden sich im selben Netzwerk über die IP-Adresse.

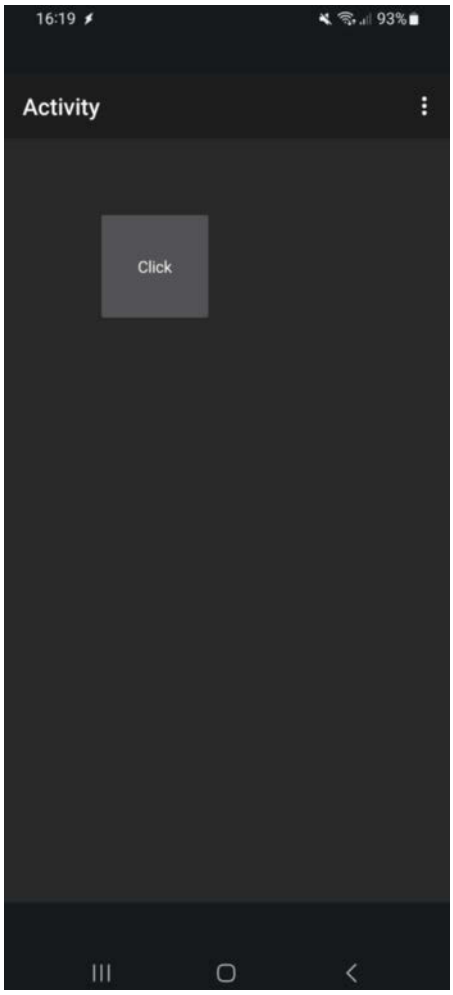


Abbildung 16: Auch auf dem Mobiltelefon kann die GUI bearbeitet werden.

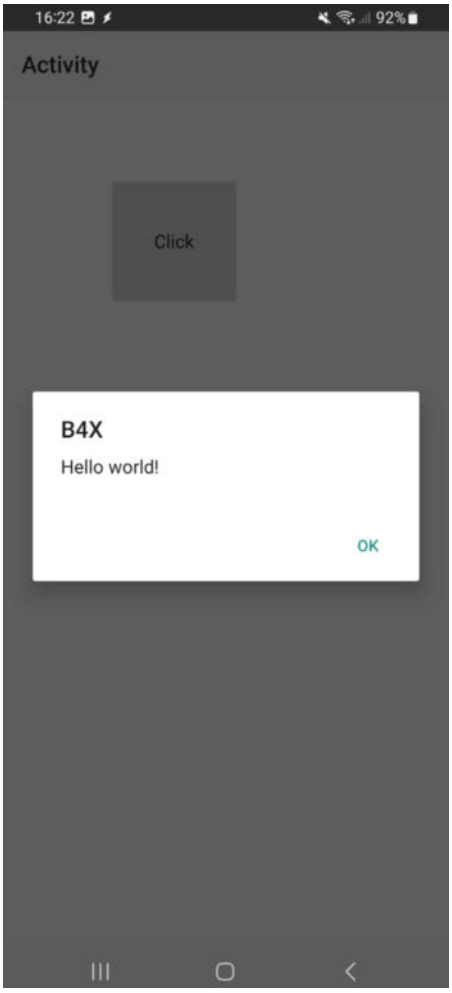


Abbildung 17: Eine erste B4A-App ist fix auf dem Handy.

einem Mikrocontroller kommunizieren sollen, ist es wichtig, gleich zu Anfang zu prüfen, ob die gewählte Sprache die notwendigen Funktionen überhaupt anbietet.

Für die Entwicklung von Android-Apps sind Programmiersprachen wie Kotlin oder Java also längst kein Muss mehr. Interessierte Laien können kleinere Anwendungen, die sogar mit Mikrocontrollern kommunizieren, in vielen Fällen ganz ohne Code bauen.

Für mehr Flexibilität sind Projekte wie B4A oder Progressive-Web-Apps interessant. Sie setzen zwar Programmierkenntnisse voraus, sind aber deutlich zugänglicher als die Android-Entwicklungsumgebung.

Letzten Endes sind Menschen mit Programmiererfahrung aber ohnehin im Vorteil, denn die meisten Plattformen automatisieren nur die Erzeugung der Android-Apps. Den Mikrocontroller-Code erzeugen sie oft nur partiell und in manchen Fällen gar nicht. Da ist noch gute, alte Handarbeit gefragt. —*caw*

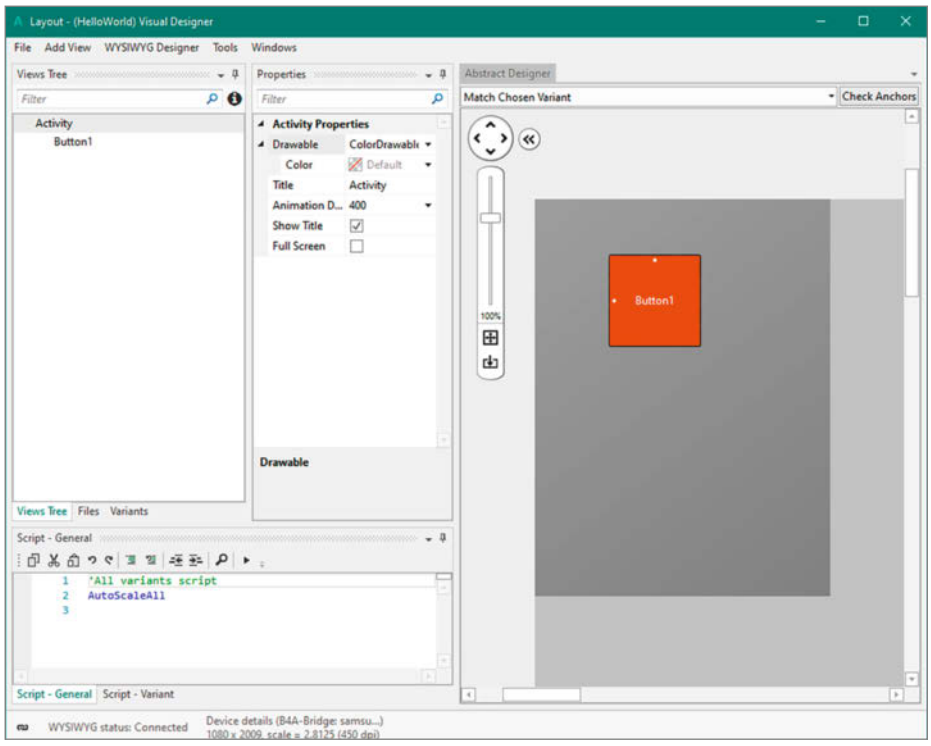


Abbildung 14: Mit dem B4A-Designer sind Bedienoberflächen für Android-Apps schnell erstellt

PostmarketOS auf dem Handy

Nach Ablauf des Softwaresupports eines Smartphones wird aus dem 500-Euro-Luxus-Device ein sehr teurer Briefbeschwerer. Um die Lebenszeit alter Handys zu verlängern, kann man mit PostmarketOS ein nativ laufendes Linux-Betriebssystem auf unterstützte Geräte aufspielen.

von Daniel Schwabe

PostmarketOS hat eine lange Liste von unterstützten Geräten. Neben dem PinePhone und dem Librem 5 gibt es viele Builds für andere Geräte, die von der Community gepflegt werden. In diesem Artikel wird der Installationsprozess beispielhaft an einem OnePlus 6 gezeigt. Für andere Geräte kann der Prozess an einigen Stellen abweichen.

Das Projekt basiert auf Alpine Linux und verwendet den Mainline Linux-Kernel. Aktuell werden 51 Geräte unterstützt. Neun weitere befinden sich im Teststadium.

Software

Als Erstes muss man einige Dateien aus dem Internet herunterladen – angefangen mit dem Betriebssystem. Auf der PostmarketOS-Website gibt es eine Liste mit allen unterstützten Geräten. Den Link dahin findet man in der Kurzinfo. In dieser Liste sucht man sich das eigene Gerät raus und klickt dort auf die Zahl, die unter Stable aufgeführt wird (zum Zeitpunkt dieses Artikels 24.06). Das führt zur Downloadseite der Betriebssystem-Images für das Gerät.

Zuerst muss man sich entscheiden, welche grafische Oberfläche man installieren möchte. Zur Entscheidungshilfe gibt es auf der PostmarketOS-Website eine Übersicht mit Screenshots. Der Link dahin befindet sich ebenfalls in der Kurzinfo. Für diesen Artikel wird „gnome-mobile“ verwendet und davon die Version, die als „latest“ ausgewiesen wird. Der Download besteht aus zwei XZ-Dateien (das ist ein komprimiertes Format wie ZIP).

Wenn diese beiden Dateien heruntergeladen sind, braucht man noch die Android SDK Plattform Tools (Link in der Kurzinfo). Diese entpackt man an einen leicht wiederfindbaren Ort auf dem Computer. In dem Ordner, in den die Plattform-Tools gespeichert wurden, erstellt man einen Ordner namens OS. In diesen

müssen die beiden Dateien für das Betriebssystem entpackt werden.

Die beiden XZ-Dateien kann man mit dem Programm 7zip öffnen. Darin enthalten sind zwei IMG-Dateien. Diese Dateien haben sehr lange Namen, in diesem Fall 20240925-0547-postmarketOS-v24.06-gnome-mobile-2-oneplus-enchilada und 20240925-0547-postmarketOS-v24.06-gnome-mobile-2-oneplus-enchilada-boot. Die Namen werden gebildet aus der Betriebssystemversion, der gewählten Oberfläche und dem Gerät, für das sie konzipiert sind. Das bedeutet, dass sich diese Namen von Gerät zu Gerät und mit einem neuen Release ändern. Wichtig ist, dass es eine Datei mit dem Suffix boot gibt und eine ohne.

Das Handy vorbereiten

Bevor man die neue Software aufspielt, muss das Handy als Erstes auf den neusten Stand upgedatet werden. Dafür sucht man in den Einstellungen unter „System“ nach „Updates“. Danach muss man die Developer-Optionen

Kurzinfo

- Natives mobile Linux
- Desktop-Linuxprogramme
- Viele unterstützte Geräte

Checkliste

Zeitaufwand:
1 Stunde

Kosten:
150 Euro

Material

- **OnePlus 6** oder ein anderes unterstütztes Handy

Werkzeug

- WIndows PC

Mehr zum Thema

- Daniel Bachfeld, Neue Aufgaben für alte Smartphones, Make 3/18, S. 28
- Carsten Wartmann, WSL2 für Maker, Make 1/24, S. 94

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

Plattform-Tools im Path installieren

Die hier gezeigte Nutzung der Plattform-Tools für das Flashen des Betriebssystems verwendet einen direkten Aufruf der EXE-Dateien der Programme. Dafür muss man entweder den genauen Pfad zum Programm im Terminal eingeben oder wie in diesem Fall das Terminal aus dem Speicherort der Programme nutzen und mit `./<Programmane>.exe` aufrufen. Allerdings kann man die Plattform-Tools auch in die sogenannten Path-Variablen von Windows eintragen, wodurch die Programme dann von überall mit dem einfachen Aufruf <Programmname> aufrufbar sind.

Dafür öffnet man in der Einstellungs-App „System/Info“ und klickt dort unter „Verwandte Links“ auf „Erweiterte Systemeinstellungen“. Es öffnet sich ein neues Fenster, in dem man im Tab „Erweitert“ ganz unten auf den Button „Umgebungsvariablen“ klickt.

Es öffnet sich wieder ein neues Fenster. Im unteren Kasten dieses Fensters befindet sich eine Tabelle. Ein Eintrag in dieser Tabelle heißt Path. Auf diesen Eintrag klickt man doppelt.

Im sich öffnenden Fenster klickt man wiederum auf den Button „Neu“. Dadurch wird ein neuer Eintrag angelegt, der direkt ausgewählt ist. Jetzt kann man entweder manuell den Pfad zu einem Programm eintippen oder auf den Button „Durchsuchen“ klicken, um das Programm im Dateiexplorer zu suchen.

Wenn man den richtigen Ordner, in dem sich die Plattform-Tools befinden, eingegeben hat, kann man alles mit „OK“ bestätigen und jetzt statt `./fastboot.exe` einfach `fastboot` im Terminal schreiben.

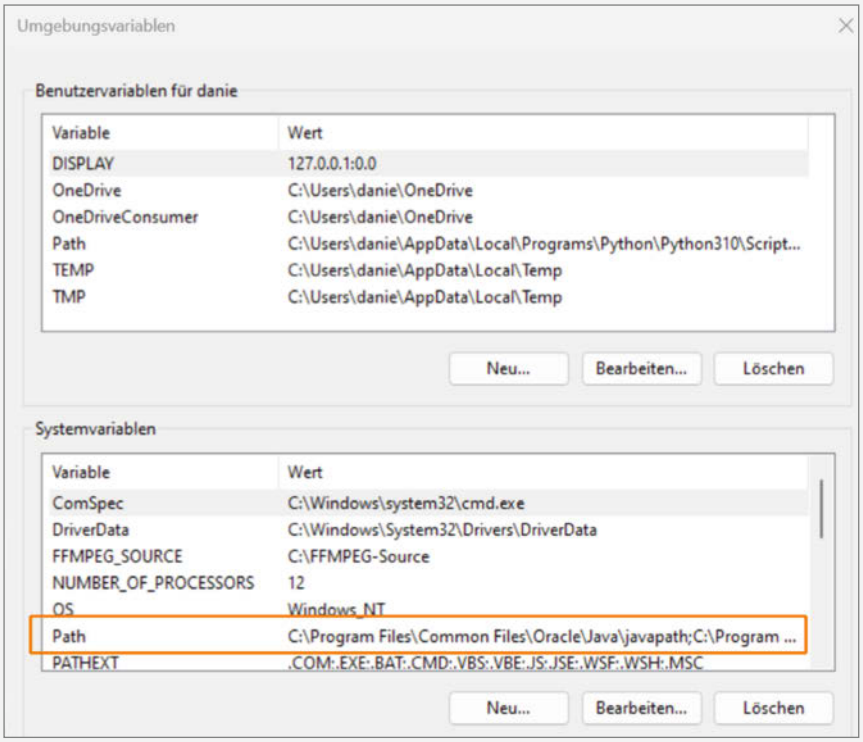


Abbildung 1: Hier kann man Programme eintragen, um sie leichter aufzurufen.

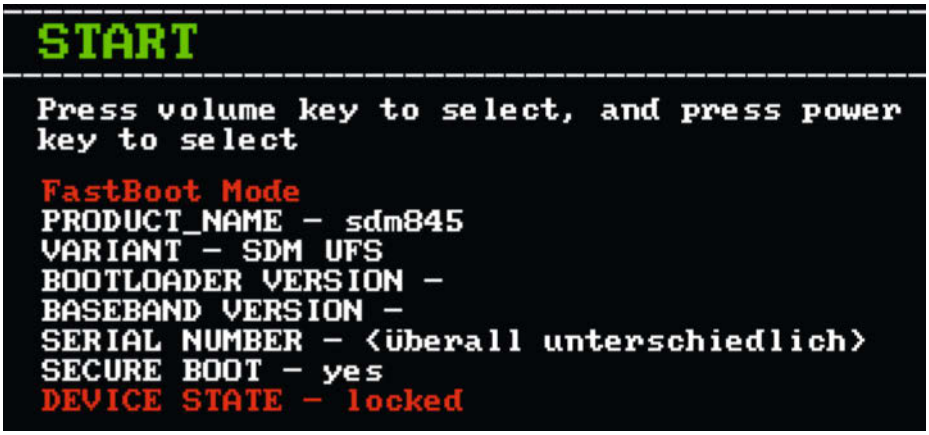


Abbildung 2: Wenn die Buttons richtig gedrückt wurden, erscheint dieser Bildschirm.

freischalten. Dafür scrollt man in den Einstellungen nach unten und tippt unter „Über das Telefon" 10-mal auf den Eintrag „Build-Nummer". Wenn es geklappt hat, erscheint auf dem Bildschirm die Meldung „Du bist jetzt Entwickler".

Jetzt muss man in den Einstellungen unter „System/Entwickleroptionen" die Option „OEM-Entsperrung" einschalten und bei dem folgenden Pop-up auf „Aktivieren" tippen. Jetzt das Handy komplett ausschalten.

Betriebssystem aufspielen

Achtung: Der folgende Vorgang löscht alle Daten vom Handy. Nach der Installation von PostmarketOS gemäß dieser Anleitung kann man Android nicht mehr nutzen. Es ist deshalb wichtig, dass man alle Daten sichert, bevor man mit der Anleitung weitermacht.

Das ausgeschaltete Handy muss man jetzt in den „Fastboot-Modus" booten. Dafür muss man gleichzeitig die Power- und die Lautertaste gedrückt halten. Hat man das richtig gemacht, erscheint der in Abbildung 2 gezeigte Bildschirm.

Jetzt schließt man das Handy mit einem USB-Label an den Computer an und öffnet den Ordner, in dem die Plattform-Tools liegen. Dort hält man die Umschalttaste gedrückt und macht einen Linksklick in einen leeren Bereich des Ordners. Im aufgehenden Menü klickt man auf „In Terminal öffnen". Das öffnet eine Eingabeaufforderung.

In dieser führt man den Befehl `.\fastboot.exe oem unlock` aus. Das führt zu einer neuen Meldung auf dem Handy. Man muss dort manuell bestätigen, dass man den OEM-Unlock wirklich durchführen will. Dafür navigiert man mit den Lautstärketasten auf den Menüpunkt „UNLOCK THE BOOTLOADER" und bestätigt mit einem Antippen der Power-Taste.

Wenn der Vorgang mit der Meldung „Finished" fertig ist, kann man die im Abschnitt „Software" heruntergeladenen Image-Dateien auf das Handy spielen. Dafür muss man aber erst den Speicher löschen, indem man in derselben Eingabeaufforderung den Befehl `.\fastboot.exe erase dtbo` eingibt. Auch dieser Vorgang endet mit der Meldung „Finished".

Danach spielt man die Datei mit dem Suffix boot auf. Das geschieht mit dem Befehl `.\fastboot.exe flash boot .\OS\20240925-0547-postmarketOS-v24.06-gnome-mobile-2-oneplus-enchilada-boot.img`. Dieser Befehl muss bei anderen Builds und Handys abgeändert werden. Nach dem „Finished" für diesen Befehl kommen jetzt noch die Userdaten mit

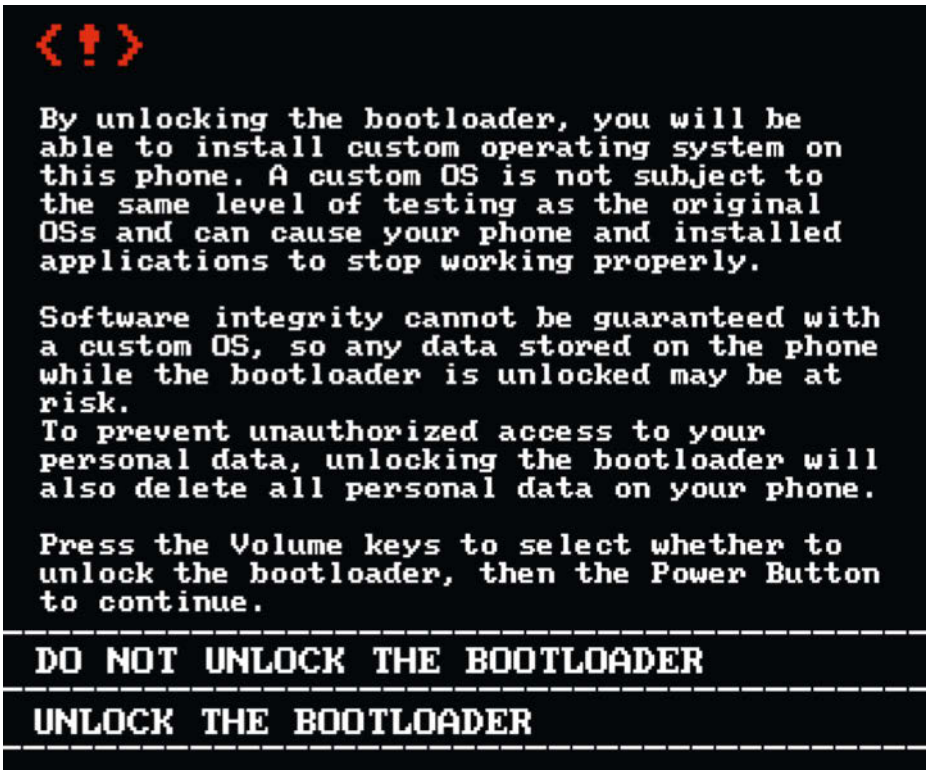


Abbildung 3: Zur Sicherheit muss man das Freischalten des Bootloaders noch am Handy bestätigen.

Abbildung 4: Der Hauptbildschirm ist aufgebaut wie bei einem Android-System.

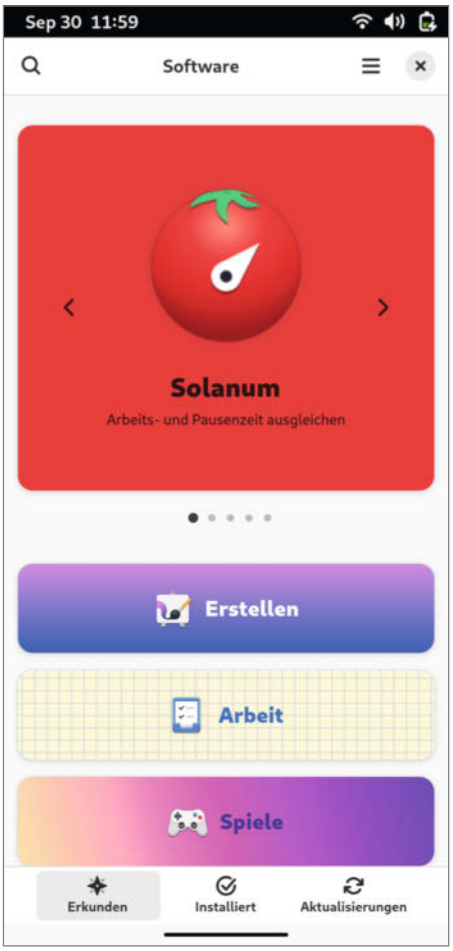


Abbildung 5: Der Store gibt einem eine gute Smartphone-Erfahrung.

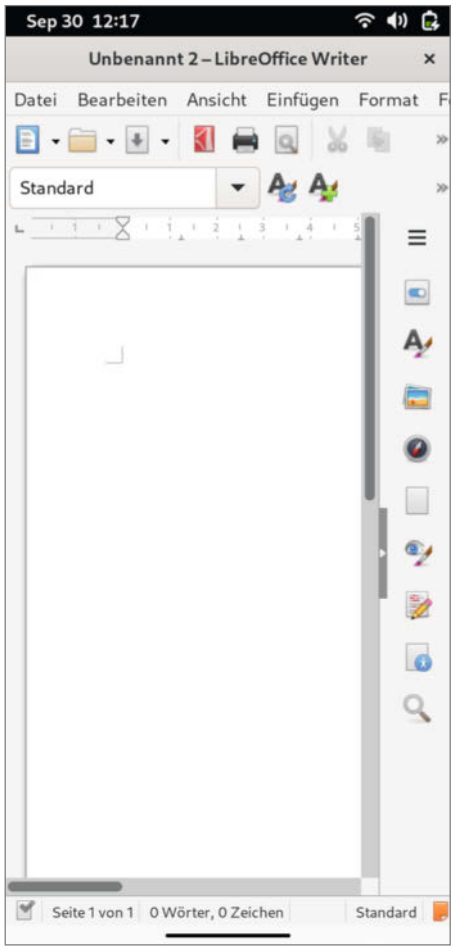
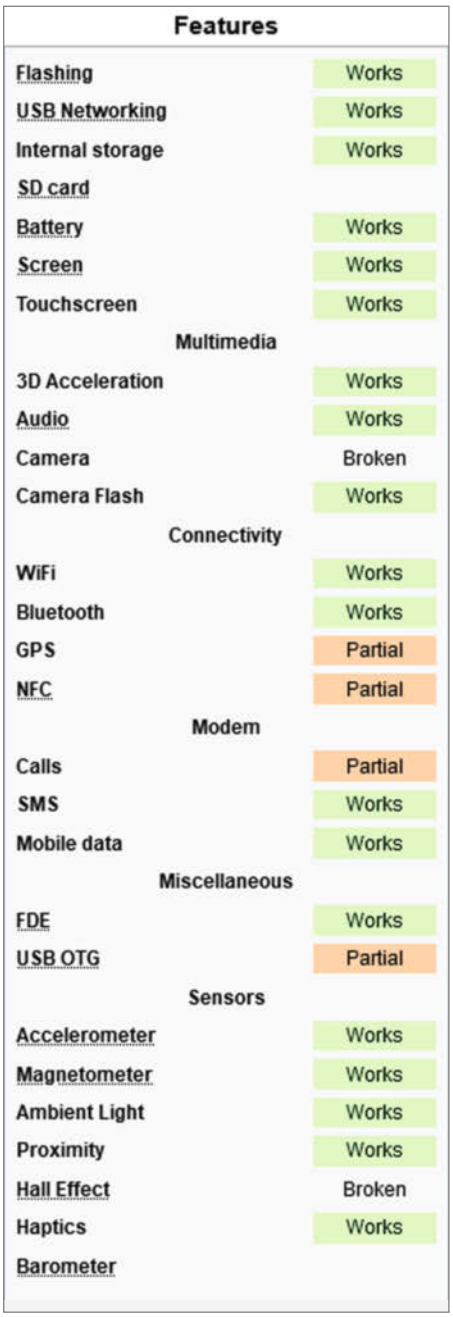


Abbildung 6: Etwas klein und vollgepackt, aber das ist das normale Desktop-LibreOffice.



Features	
Flashing	Works
USB Networking	Works
Internal storage	Works
SD card	
Battery	Works
Screen	Works
Touchscreen	Works
Multimedia	
3D Acceleration	Works
Audio	Works
Camera	Broken
Camera Flash	Works
Connectivity	
WiFi	Works
Bluetooth	Works
GPS	Partial
NFC	Partial
Modem	
Calls	Partial
SMS	Works
Mobile data	Works
Miscellaneous	
FDE	Works
USB OTG	Partial
Sensors	
Accelerometer	Works
Magnetometer	Works
Ambient Light	Works
Proximity	Works
Hall Effect	Broken
Haptics	Works
Barometer	

Abbildung 7: Jedes Gerät hat auf dem PostmarketOS-Wiki eine solche Tabelle.

https://wiki.postmarketos.org

`.\fastboot.exe flash userdata .\OS\20240925-0547-postmarketOS-v24.06-gnome-mobile-2-oneplus-enchilada.img` auf das Handy.

Jetzt muss man nur noch mit `.\fastboot.exe reboot` das Handy neu starten. Nachdem alles eingerichtet ist, kann man sich in den Account „Linux user" mit dem Passwort 147147 einloggen.

Das System

Die Bedienung des Systems ist sehr ähnlich zu einem Android-Handy. Von oben kann man ein Dock herunterziehen, in dem man Quick Settings für Bluetooth, WLAN etc. hat. Tippt man in ein Eingabefeld, erscheint auch eine Bildschirmtastatur.

Nach dem ersten Start schwebt auf dem Home-Bildschirm in der Mitte ein kleines weißes Fenster. Wenn man das antippt, öffnet sich ein Willkommensbildschirm, der einen durch manche Funktionen des Systems leitet.

Über die App „Software" kann man sich neue Programme installieren. Falls dort nichts angezeigt werden sollte, muss man in dieser App unten rechts unter „Aktualisierungen" ein Update installieren. Danach sollten verfügbare Programme wieder richtig angezeigt werden.

Auf dem Handy funktionieren jetzt dank Linux auch bekannte Desktop-Anwendungen wie LibreOffice. Man kann sogar eine richtige Linux-Eingabeaufforderung nutzen, um Programme mit dem „apk add"-Befehl zu installieren. In den Einstellungen kann man unter „System" auch SSH (Secure Shell) aktivieren, um das Handy von einem entfernten Terminal aus zu steuern.

Selbstverständlich sind auch Apps zum Telefonieren und SMS-Versenden vorinstalliert.

Je nachdem, auf welchem Gerät man PostmarketOS installiert, ändert sich, was am Handy funktioniert. Beispielsweise funktioniert die Kamera mit dem hier genutzten OnePlus 6 nicht. Für jedes unterstützte Gerät gibt es im PostmarketOS-Wiki eine Tabelle für den Status der einzelnen Hardwarekomponenten.

Auf Seite 66 stellen wir in einem Artikel vor, wie man Desktop-Windows auf ein altes Lumia-Handy installiert. Im direkten Vergleich funktioniert PostmarketOS bei dem hier gewählten OnePlus 6 sehr viel besser. Das liegt vor allem an den 8 GB RAM, durch den das System viel Raum nach oben hat. Leider ist durch einen fehlenden HDMI-über-USB-C-Support die Funktionalität für Produktivaufgaben mit dem OnePlus 6 eingeschränkt. Das ist aber, wie gesagt, von Gerät zu Gerät unterschiedlich. Es lohnt sich auf jeden Fall, die eigenen, alten Geräte bzgl. Kompatibilität mit PostmarketOS zu überprüfen. Damit können durch veraltete Software obsolete Smartphones einem neuen Zweck zugeführt werden. *—das*

Desktop-Windows auf Lumia-Handys

Seit 2020 bekommen Lumia-Handys keine Updates mehr. Und auch davor sah die App-Unterstützung nicht besonders gut aus. Mit dem „Lumia WOA Project“ kann man die ARM-Version von Desktop-Windows auf so einem Gerät installieren.

von Daniel Schwabe

Kurzinfo

» **Windows 11 auf dem Lumia 950 XL**
» **Dualboot mit Windows Mobile**
» **Desktop Programme auf dem Handy**

Checkliste

Zeitaufwand:
4 Stunden

Kosten:
150 Euro

Material

» **Lumia 950**
» **USB-C-Kabel**

Werkzeug

» **Windows Computer**

Mehr zum Thema

» Prof. Kai Schauer, Universal-IR-Fernbedienung, Make 1/24, S. 72
» Carsten Wartmann, WSL2 für Maker, Make 1/24, S. 94

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

Als 2017 mein Nokia N-Gage Alterserscheinungen zeigte, habe ich mich entschieden, auf ein modernes Smartphone upzugraden, und zwar auf ein Lumia 950 XL. Zu diesem Zeitpunkt hatte das „Windows Mobile Betriebssystem“ noch einen Marktanteil von 0,82 Prozent unter den Mobile-Betriebssystemen.

Wenn man sich dieses Handy heute anschaut, kann es mit seinen Features teilweise noch mit den aktuellen Konkurrenten mithalten: kabelloses Laden, herausnehmbarer Akku, Speichererweiterung über microSD-Karte, Display-Output über USB-C und ein physischer Kameraknopf.

Aber hinsichtlich seiner Leistung ist es natürlich nicht mehr vergleichbar. Eine Übersicht der Spezifikationen ist in der Tabelle Technik aufgelistet. Das größte Problem für das alte Handy ist aber das Betriebssystem. Selbst als die Plattform Windows 10 Mobile noch aktiv von Microsoft unterstützt wurde, gab es wenige Apps im Store. Anwendungen wie der Whatsapp Messenger haben wichtige Updates nicht bekommen und wurden schlecht optimiert, was zu vielen Abstürzen geführt hat. Und das, obwohl Windows 10 Mobile eine wirklich tolle Nutzererfahrung bot.

Mittlerweile sind vier Jahre vergangen, seitdem das letzte Update ausgespielt wurde, und Lumia-Handys sind nur noch schicke Briefbeschwerer. Mit dem „Lumia WOA Project“ kann man allerdings eine aktuelle ARM-Version von Windows 11 auf solch ein Handy aufspielen und dem Gerät wieder neues Leben einhauchen.

In diesem Artikel wird gezeigt, wie man Windows 11 im Dualboot mit Windows 10 Mobile auf einem Lumia 950 XL installiert.

Vorbereitung

Damit alles richtig funktioniert, muss man als Erstes sicherstellen, dass alle Updates auf dem Handy installiert sind. Dafür im Home-Bildschirm das Dock von oben nach unten ziehen und dort auf „Alle Einstellungen“ tippen; dann dort nach unten scrollen und den Punkt „Update und Sicherheit/Windows Update“ auswählen. Jetzt kann man mit dem Button „Nach Updates suchen“ das System auf den neusten Stand bringen.

Außerdem sollte man jetzt alle Daten, die so wichtig sind, dass man sie in den letzten vier Jahren noch nicht gesichert hat, backupen. Und zu guter Letzt empfiehlt es sich, den Akku einmal voll aufzuladen.

Der Bootloader

Um ein neues Betriebssystem auf das Handy zu bekommen, muss man nun den Bootloader entsperren. Dieser ist eine Software, die nach dem Einschalten des Gerätes das richtige Betriebssystem startet. Hierfür braucht man die Software WPinternals. Den Link dazu findet man in der Kurzinfo. Nach dem Download muss die heruntergeladene ZIP-Datei entpackt und WPinternals gestartet werden.

Die geöffnete „Getting-Started“-Seite (Abbildung 1) bietet alle Informationen zu Hardware- und Softwarekompatibilität. Weiter unten auf der Seite, unter „Required Third Party Downloads“, befinden sich eine Reihe von Treibern, die installiert werden müssen: das Windows Device Recovery Tool und die Microsoft-Treiber für die Handys. Die Links zu diesen Programmen sind noch einmal in der Kurzinfo verlinkt. Nach dem Download der Dateien werden die Software und die Treiber mit einem Doppelklick installiert. Danach muss WPinternals neu gestartet werden.

Jetzt ist es Zeit, das Handy per USB mit dem Computer zu verbinden. Nachdem das

Technik

Display	1440 x 2560 Pixel
Speicher	32 GB
RAM	3 GB
Kamera	20 MP/5MP
CPU	2,00 GHz

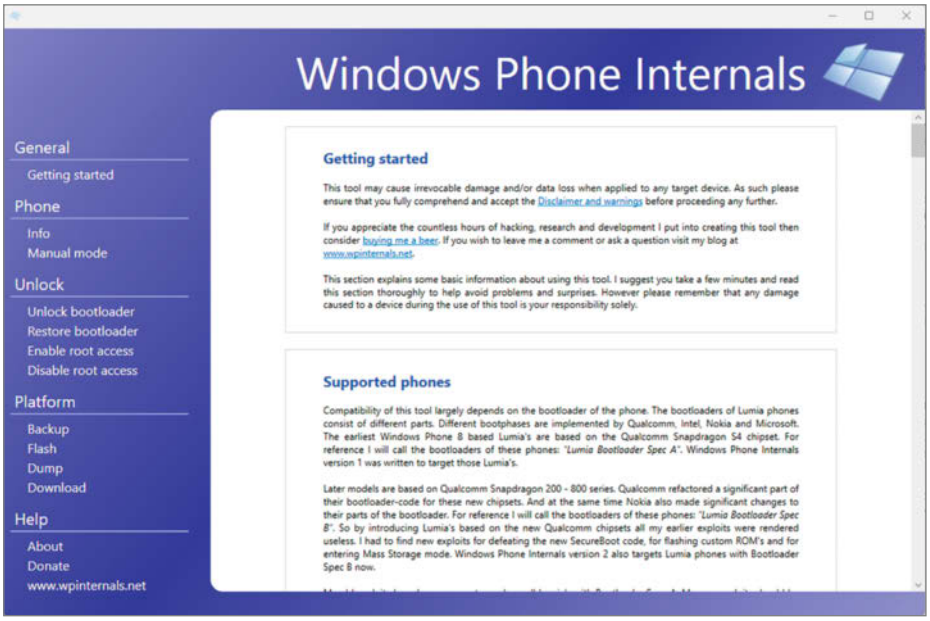


Abbildung 1: Windows Phone Internals erlaubt es, den Bootloader freizuschalten.

Abbildung 2: Die Software ist mit ausführlichen Warnhinweisen ausgestattet.

Handy hochgefahren ist (das geschieht von selbst, sobald es Strom bekommt), geht es weiter zum Datei-Download. In WPinternals klickt man dazu auf der linken Seite auf Download, scrollt in diesem Abschnitt etwas runter und klickt erst auf den Button „Search" und dann auf „Download all". Dieser Vorgang lädt Image-Dateien herunter, die speziell für das angeschlossene Handy vorgesehen sind. Die Dateien werden im Ordner „C:\ProgramData\WPInternals\Repository" gespeichert.

Telefonieren nach 18363

Um das Telefon-Feature in neueren Windows-Versionen zu nutzen, muss man zunächst eine alte Version bis 18363 aufspielen.

Ist dies geschehen und das Gerät fertig eingerichtet, legt man bei ausgeschaltetem Handy eine SIM-Karte ein und bootet in Windows. Dort muss man dann im Programm regedit den Pfad HKEY_LOCAL_MACHINE\SOFTWARE\OEM\RILNITSVC\ aufrufen. Dort befinden sich zwei Variablen: (Default) und iCan0. Der Wert für iCan0 ist eine lange Zeichenkolonne. Diese ist für jede SIM-Karte unterschiedlich. Den Wert muss man sich notieren.

Danach installiert man die neue Windows-Version und ruft in regedit wieder den gleichen Pfad auf. Sollte es den Pfad nicht geben, muss man ihn erstellen. Das funktioniert ähnlich wie bei einem Ordner. Wenn es einen Teil des Pfades nicht gibt, geht man zum letzten verfügbaren Ordner, klickt rechts und wählt Neu/Schlüssel.

Ist man dann im richtigen Ordner, legt man dort die `iCan0` Variable an. Dafür macht man ebenfalls einen Rechtsklick und geht dann auf Neu/Zeichenfolge. Als Name gibt man `iCan0` ein und als Wert den notierten Wert der alten Windows-Version. Nun speichert man und startet das Handy neu.

Sobald der Download abgeschlossen ist, hat man alles beisammen, um den Freischaltungsprozess des Bootloaders zu beginnen.

Dafür klickt man in der linken Bedienspalte auf den Punkt „Unlock bootloader". Es erscheint eine Meldung, dass das Handy zum Unlocken in den Flash-Modus versetzt werden muss. Diese Nachricht muss man einfach bestätigen, WPinternals bootet das Handy danach automatisch in den richtigen Modus. Während dieses Vorganges verbindet sich das Handy neu mit dem Computer, das ist normal. Sobald das Handy im Flash-Modus ist, erscheint wieder einiges an Text auf der Seite (Abbildung 2), was noch einmal über die Risiken des Vorgangs aufklärt. Das Handy zeigt einen gefährlich aussehenden roten Bildschirm an.

Am unteren Ende der Seite muss man jetzt über die anklickbaren Textstellen „Select your FFU-image for profiling …" und „Select the EDE-File for your phone" jeweils die passenden Dateien auswählen, die man im vorherigen Schritt heruntergeladen hat.

Danach wird der Unlock-Button klickbar. Der Vorgang dauert etwas, währenddessen wird das Handy sehr oft neu starten. An einer Stelle wird man dazu aufgefordert, das Handy manuell neu zu starten. Dazu muss man den Power- und den Leiser-Button an der Seite des Geräts so lange halten, bis es kurz vibriert (Abbildung 3).

Hat alles geklappt, steht am Ende des Prozesses „Bootloader unlocked successfully!". Unter Umständen kann das Unlocken aber auch fehlschlagen. Dann muss man es noch einmal probieren.

Windows Desktop

Der Bootloader ist jetzt unlocked und das Handy bereit, mit einem neuen Betriebssystem bespielt zu werden. Um dieses herunterzuladen, geht man auf die Webseite uupdump.net (siehe Link in der Kurzinfo).

Nun muss man sich entscheiden, welches Betriebssystem man nutzen möchte. Da Windows 10 bald End of Life ist, wird in diesem Artikel Windows 11 in Version 22631.2506 verwendet. Spätere Versionen von Windows 11 funktionieren nicht mehr auf dem Lumia 950 .

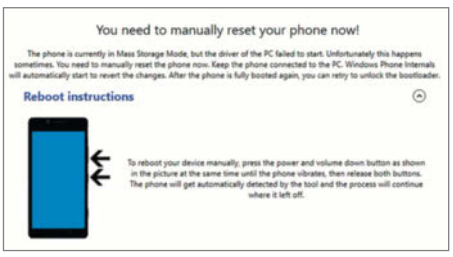


Abbildung 3: Während des Flash-Vorgangs muss man das Handy selbst neu starten.

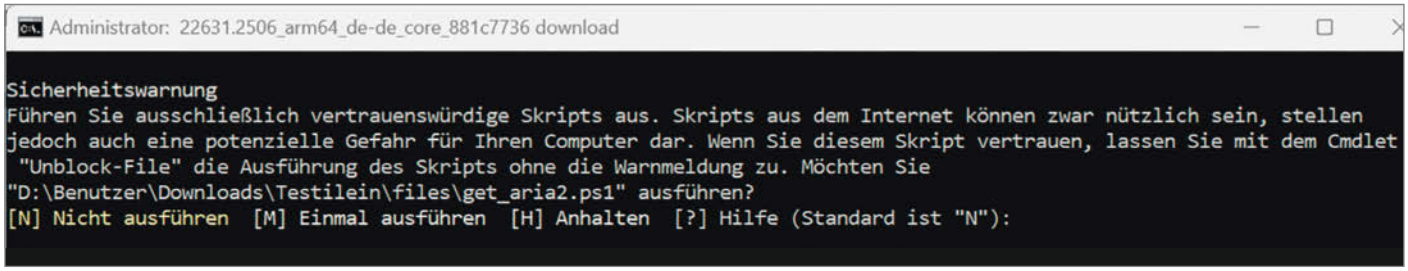


Abbildung 4: Die Sicherheitswarnung muss man mit „N“ bestätigen.

Allerdings gibt es mit der Wahl dieser Betriebssystemversion ein Problem: Telefonieren funktioniert nicht mehr. Bis Windows 10 Version 1909 Build 18363 konnte man einfach eine SIM-Karte ins Handy einlegen und unter Desktop Windows 10 mit dem Lumia telefonieren. Möchte man das mit einer neueren Version machen, muss man einige extra Schritte gehen. Diese sind im Kasten „Telefonieren nach 18363“ erklärt.

Hat man sich für ein Windows-Release entschieden (den Link zu der in diesem Artikel verwendeten Version findet man in der Kurzinfo), wählt man eine Sprache aus, klickt auf Weiter und entscheidet sich für Pro oder Home (für diesen Artikel wurde Windows 11 Home genutzt). Danach klickt man noch einmal auf weiter und dann auf „Downloadpaket erstellen“.

Nach dem Download der ZIP-Datei mit dem schönen Namen „22631.2506_arm64_de-de_core_881c7736_convert“ muss man diese entpacken und die Datei „uup_download_windows“ mit einem Doppelklick ausführen. Im sich öffnenden Fenster wird als Erstes eine Sicherheitswarnung erscheinen (Abbildung 4). Dort bestätigt man mit „N“, dass das Programm ausgeführt werden soll und wartet dann, bis alles fertig heruntergeladen ist. Es kann an einigen Stellen so aussehen, als ob das Programm nichts mehr macht. Das ist normal und man muss einfach abwarten.

Nachdem das Programm alles fertig heruntergeladen hat, befindet sich im Programmordner eine ISO-Datei für eine Windows-Installation. Die Datei hat ebenfalls einen sehr markanten Namen: in diesem Fall „22631.2506.231018-1809.23H2_NI_RELEASE_SVC_PROD3_CLIENTCORE_OEMRET_A64FRE_DE-DE“. Aus dieser ISO-Datei muss man jetzt die install.wim-Datei extrahieren. Das kann man entweder mit einem Programm wie 7zip oder WinRAR machen, indem man auf die ISO rechtsklickt und dann dort mit 7zip/WinRAR öffnet, oder die Datei in ein virtuelles Laufwerk einhängt, indem man auf die Datei rechtsklickt und dann auf „Bereitstellen“ klickt. Die install.wim befindet sich in der geöffneten ISO-Datei im Unterordner Source. Die install.wim muss man dann in einen Ordner extrahieren, den man am besten gut wiederfindet.

Windows installieren

Um das heruntergeladene OS jetzt auf das Handy zu spielen, muss das Lumia mit WPinternals in den „Mass Storage Mode“ versetzt werden. Um den Modus zu wechseln, klickt man in WPinternals auf der linken Navigationsleiste auf „Manual mode“ und dort dann auf „Switch to Mass-Storage-mode“. Wie beim Unlocken des Bootloaders muss man auch jetzt eine Weile warten und das Handy einmal manuell neu starten.

Falls das Wechseln in den „Mass Storage“-Mode nicht klappt, das Handy danach im Flash-Modus ist (roter Bildschirm mit einem drehenden Pfeil) und sich auch bei mehreren neuen Versuchen weder in den „Mass Storage“-Mode oder den normalen Windows-Modus versetzen lässt, gibt es einen Trick. Indem man den Prozess für das Unlocken des Bootloaders wieder aufruft (das wird fehlschlagen, aber das ist okay, da der Bootloader ja schon freigeschaltet ist), wird das Handy wieder im normalen Windows-Modus gestartet und der Wechsel zum „Mass Storage“-Mode kann noch einmal begonnen werden.

Ist das Handy dann endlich im richtigen Modus angekommen, muss man WPinternals schließen und das Programm „WOA Deployer for Lumia“ herunterladen (Link in der Kurzinfo).

Backup vor der Installation

Mit dem Programm „Win32 Disk Imager“ (Link in der Kurzinfo) kann man ein Image des Dateisystems des Handys anlegen. Dafür muss man das Programm starten, unter Datenträger das Handy auswählen und mit einem Klick auf den kleinen Ordner links daneben einen Speicherort für das Image auswählen. Zum Schluss muss man unten auf „Lesen“ klicken und warten.

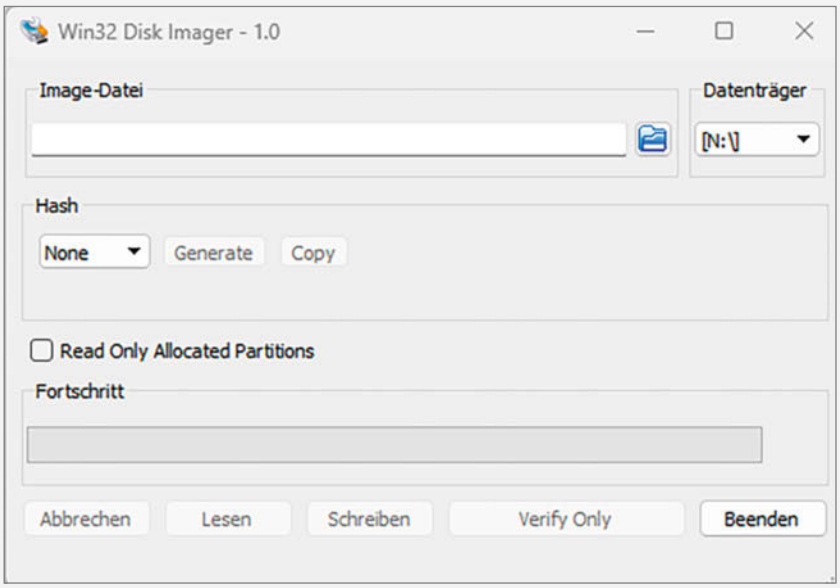


Abbildung 5: Win32 Disk Imager sichert die alten Smartphone-Daten.

Man entpackt diese ZIP-Datei nach dem Download und startet mit einem Doppelklick auf WoaDeployer das Programm.

Beim Starten öffnet sich ein Warnhinweis, dass man alle Daten sichern soll, bevor man irgendetwas mit WOA Deployer macht. Wie man dabei am besten vorgeht, steht im Kasten „Backup vor der Installation". Nachdem man diese Nachricht mit einem Klick auf „OK" geschlossen hat, befindet man sich in der richtigen Programmoberfläche (Abbildung 6).

Jetzt muss man die install.wim-Datei heraussuchen, die man im vorherigen Schritt extrahiert hat. Dafür klickt man auf den „Browse"-Button auf der rechten Seite und navigiert im sich öffnenden Explorer-Fenster zu der gespeicherten install.wim und öffnet sie.

Sollte eine Meldung „Invalid WIM file" erscheinen, muss die Datei repariert werden. Wie das geht, steht im Kasten „WIM reparieren". Wenn WOA-Deployer die WIM akzeptiert hat, überprüft man die Einstellungen im „Advance"-Tab (Abbildung 7).

Hier muss man sich entscheiden, ob man Windows Mobile für einen Dualboot installiert lässt oder es löscht und nur Windows 11 behält. Die Einstellungen in der Abbildung sind für Dualboot.

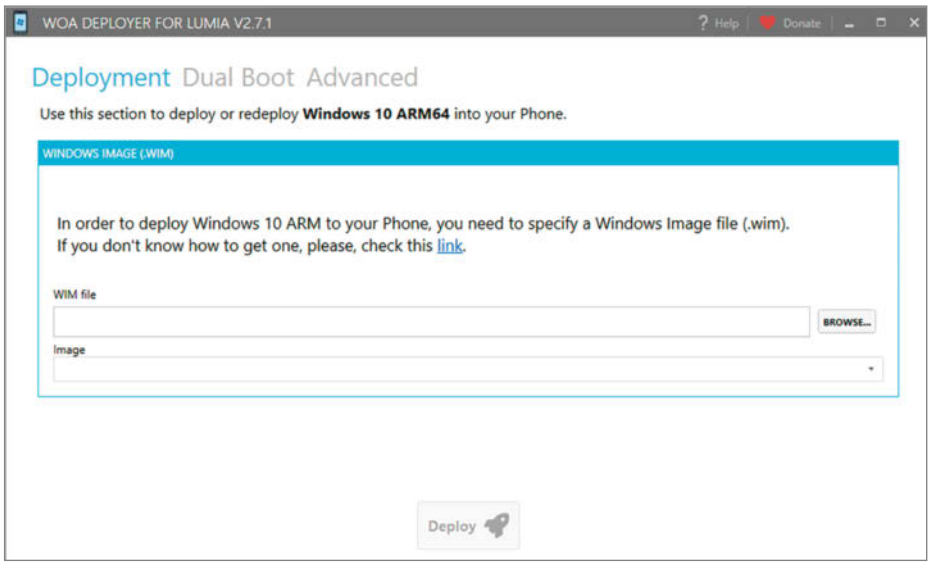


Abbildung 6: WOA-Deployer ist sehr übersichtlich.

WIM reparieren

Wenn die install.wim-Datei nicht gelesen werden kann, muss man sie reparieren. Dafür sucht man nach cmd in der Windows-Suchleiste. Das Ergebnis „Eingabeaufforderung" muss man als Administrator starten.

Mit dem Befehl `cd <Pfad-zur-wim>` navigiert man in den Ordner, in dem die install.wim gespeichert ist. Jetzt führt man in der Eingabeaufforderung den Befehl `DISM /Export-Image /SourceImageFile:".\install.wim" /SourceIndex:1 /DestinationImageFile:".\fixed.wim" /CheckIntegrity` aus. Sobald der Prozess durchgelaufen ist, muss man in WOA-Deployer die Datei fixed.wim öffnen.

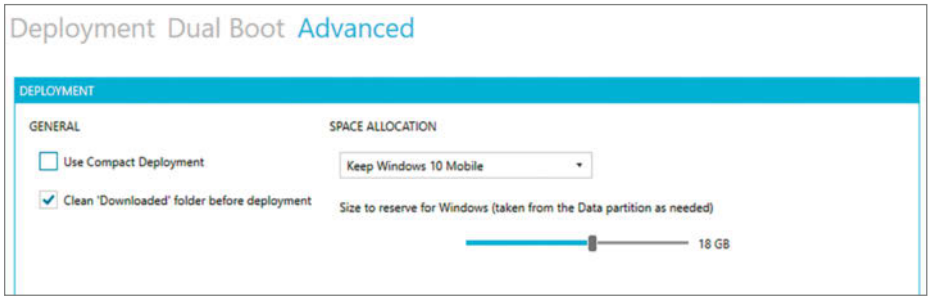


Abbildung 7: Diese Einstellungen muss man für Dualboot tätigen.

Ist dort alles richtig konfiguriert, wechselt man zurück zum „Deployment"-Tab und klickt auf „Deploy". Kurz nach dem Prozessbeginn öffnet sich ein Fenster mit Informationen über die aktuelle Version der Treiber. Das kann man mit einem Klick auf „close" einfach schließen.

Wieder einmal muss man eine Weile warten. Im ersten Schritt wird ein Developer-Menü auf dem Handy installiert. Ist das geschehen, öffnet WOA-Deployer eine Nachricht diesbezüglich und man muss erneut eine Aktion am Handy ausführen (Abbildung 8).

Um den Vorgang fortzusetzen, muss man nun das Handy neu starten. Dafür hält man den Power-Button so lange gedrückt, bis das Handy vibriert. Anstatt jetzt in Windows Mobile zu booten, startet auf dem Handy allerdings das Boot-Menü (Abbildung 9). In diesem Menü kann man mit der Lauter- und der Leisertaste navigieren und mit dem kleinen Kamera-Button (ganz unten an der rechten Seite des Handys) bestätigen.

Nun muss man zuerst zum Punkt „Developer Menu" navigieren, mit der Kameraataste bestätigen und im erscheinenden Untermenü den Punkt „Mass Storage Mode" auswählen.

Sobald auf dem Handy die Nachricht „UEFI Mass Storage CONNECTED" erscheint, kann man am Computer die WOA-Meldung mit „Continue" wegklicken und der Installationsvorgang geht weiter.

Ab jetzt erscheinen auf dem Computer mehrere Windows-Meldungen (zum Beispiel, dass angeschlossene Datenträger formatiert werden müssen) und Explorer-Fenster. Diese können ignoriert werden, da der Vorgang korrekt läuft.

Sobald alles fertig kopiert und installiert wurde, meldet sich WOA-Deployer mit einer Nachricht. Das Handy muss jetzt neu gestartet werden. Zum Neustarten hält man einfach den Power-Button so lange gedrückt, bis das Handy vibriert.

Das Gerät bootet jetzt in das Boot-Menü. Bevor man aber eine Option auswählt, muss man das Handy an ein Netzteil anschließen. Mit der Kameraataste kann man im Boot-Menü „Windows 10" (egal ob man Windows 10 oder 11 installiert hat, im Menü steht immer 10) auswählen und das Gerät beginnt mit dem letzten Schritt der Installation.

Das Handy startet währenddessen wieder mehrere Male neu. Sobald alles fertiggestellt ist, startet Windows 11 und die reguläre Einrichtung von Windows beginnt. Dort muss man dann wie bei der normalen Einrichtung von Desktop-Windows die eigenen Microsoft-Daten eingeben und Einstellungen zur Personalisierung vornehmen.

Im Setup muss man sich mit einem Netzwerk verbinden. Sollte das nicht funktionieren und immer wieder die Meldung erscheinen, dass die Verbindung abgebrochen

ist, muss man mit einem USB-C-auf-USB-A-Adapter eine Tastatur anschließen. Auf dieser drückt man dann die Tastenkombination Umschalt + F10. Dadurch öffnet sich eine Konsoleneingabe, in der man den Befehl `OOBE\BYPASSNRO` eingibt. Das Handy startet neu und wird diesen Schritt dann überspringen.

Ist alles fertig eingerichtet, startet man in den bekannten Windows-Desktop (Abbildung 11).

Dualboot

Der letzte Schritt ist jetzt, Dualboot mit Windows Mobile zu aktivieren. Man startet das Handy neu und wählt beim Hochfahren im Boot-Menü wieder über das Developer-Menü den „Mass Storage Modus" aus. Dann startet man den WOA-Deployer und verbindet das Handy per USB. Im WOA-Deployer im Tab „Dual Boot" klickt man den Button „Check Status" an und wartet. Wenn die Überprüfung abgeschlossen ist, erscheint der Button „Enable Dual Boot". Diesen betätigt man und wartet auf die Bestätigung.

Danach kann man beim Hochfahren des Handys im Boot-Menü zwischen Windows Phone und Windows 11 wählen.

Was mache ich jetzt damit?

Natürlich ist dieses Handy kein technisches Powerhouse mehr. Aber für einfache Arbeiten wie Textverarbeitung ist es immer noch gut geeignet. Mit einem USB-C-Dongle kann man über HDMI einen Monitor und über USB eine Maus und Tastatur anschließen. Mittlerweile gibt es auch viele native „Windows on ARM"-Apps. Eine Liste dazu ist in der Kurzinfo verlinkt.

Leider funktioniert auch nicht alles. Die Kameras werden vom System nicht erkannt und sind unter Windows somit nutzlos. Deshalb ist es nützlich, Dualboot einzurichten, denn die Kamera war eines der großen Argumente pro Lumia 950 XL, und unter Windows Mobile funktioniert sie weiterhin problemlos. Mit dem neu aufgespielten System bekommt mein Lumia 950 XL eine Chance für ein interessantes zweites Leben. *—das*

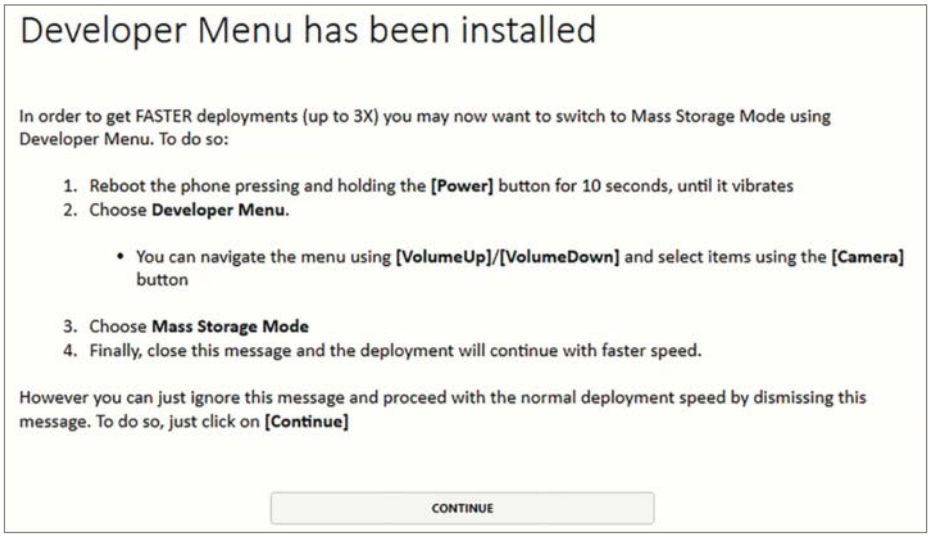


Abbildung 8: WOA-Deployer installiert ein neues Menü auf dem Handy.

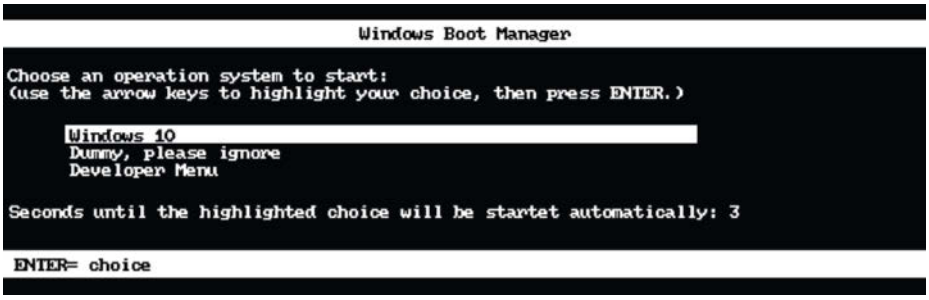
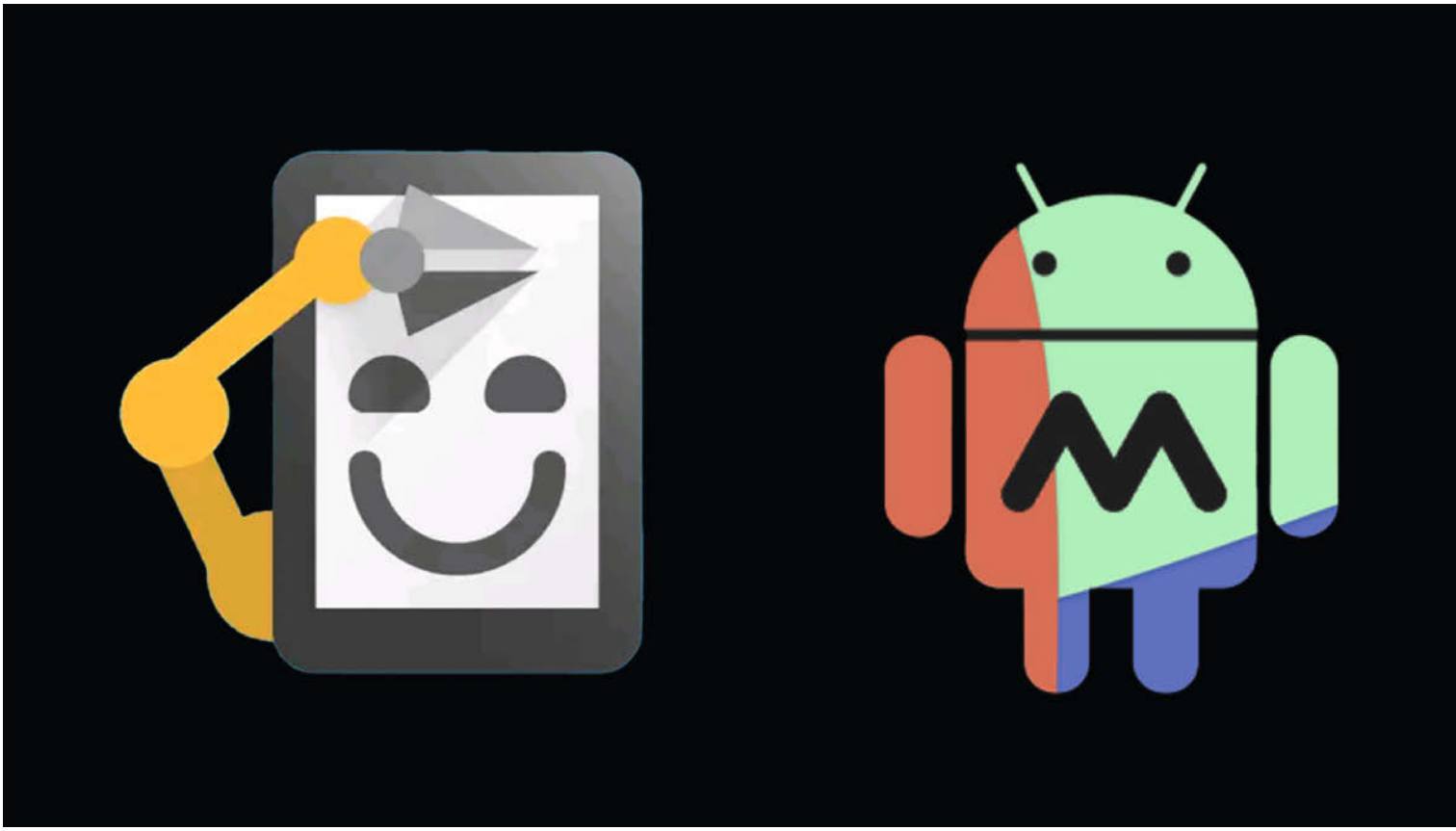


Abbildung 9: Das Boot-Menü lässt sich mit den physischen Tasten am Handy steuern.

Abbildung 10: Ein neuer Bootscreen ist auch dabei.

Abbildung 11: Zwar in einem seltsamen Seitenverhältnis, aber das ist Desktop-Windows.



Automatisieren mit Android

Um eigene Maker-Projekte mit dem Smartphone zu steuern, braucht man keine komplexen, maßgeschneiderten Apps zu entwickeln. Schneller und einfacher gelingt es oft mit Automatisierungs-Apps wie Tasker, Automate und MacroDroid. In diesem Artikel erkunden wir die Potenziale und Einsatzmöglichkeiten dieser Apps in Verbindung mit einem ESP32-gesteuerten Sensor.

von Maik Schmidt

Mit den meisten Android-Automatisierern ist es beispielsweise recht einfach, eine SMS mit dem Text „Bin gleich zu Hause." zu senden, sobald man eine bestimmte Autobahnausfahrt passiert. Diese Tools eignen sich aber auch hervorragend für typische Maker-Projekte und als Beispiel dient in diesem Artikel ein selbst gebauter Funk-Sensor für die Temperatur. Dieser besteht aus einem ESP32 mit einem BME280-Sensor, dessen Messdaten das Gerät als JSON-Dokument im WLAN bereitstellt.

Der Funk-Sensor

Den Anschluss des BME280 an den ESP32 beschreibt der Artikel „Smartphone-Apps mit RemoteXY" auf Seite 30 und das hier aufgeführte Listing SensorWebServer.ino enthält den Code, der notwendig ist, um der Schaltung Leben einzuhauchen (Download siehe Link in der Kurzinfo). Das Programm basiert auf der Arduino IDE und es nutzt die BME280-Bibliothek von Adafruit, die über den Bibliotheksverwalter zu installieren ist.

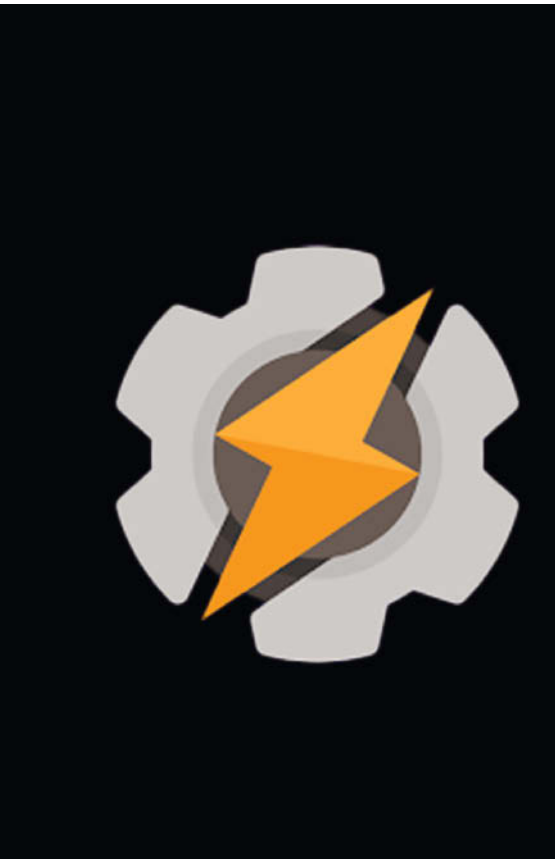
Zu Anfang binden diverse `#include`-Anweisungen alle benötigten Bibliotheken ein und anschließend definiert das Programm zwei Konstanten mit den Namen `SSID` und `PASSWORD`, bei denen man die SSID und das Passwort des WLAN-Netzwerks eintragen muss, mit dem der ESP32 sich verbinden soll.

Danach werden zwei Objekte initialisiert: ein WebServer-Objekt namens `server`, das den Standard-Port 80 verwendet und HTTP-Anfragen empfangen sowie auf diese antworten kann; und ein BME280-Objekt für den Sensor.

WLAN-Verbindung aufbauen

Im weiteren Verlauf nutzt die Funktion `initWifi()` die WiFi-Bibliothek, um eine Verbindung mit einem WLAN-Netzwerk herzustellen. Wenn eine Verbindung hergestellt wurde, gibt das Programm die IP-Adresse aus, die dem ESP32 im Netzwerk zugeteilt wurde. Weil es umständlich ist, mit IP-Adressen zu hantieren und weil sich die Adresse auch bei jeder Verbindung ändern kann, nutzt das Programm mDNS (Multicast DNS) und macht den ESP32 im lokalen Netzwerk unter dem Namen „esp32" bekannt. Das funktioniert nicht unbedingt in allen Netzwerken, aber für den Notfall gibt der Code die IP-Adresse des Boards aus.

Das JSON-Format ist bei Web-Diensten beliebt, weil es sowohl kompakt als auch einfach



Kurzinfo

- Automatisierungs-Apps nutzen, um eigene Maker-Projekte zu steuern
- Kurzvorstellung von Tasker, Automate und MacroDroid
- Mit selbstgebautem Temperatur-Sensor und ESP32 die Apps ausprobieren

Checkliste

Zeitaufwand:
ab 1 Stunde

Kosten:
ca. 10 Euro

Material

- **ESP32-Board**
- **Sensor** BME280
- **Breadboard mit Kabeln**

Software

- **Tasker**
- **Automate**
- **MacroDroid**
- **Arduino IDE**

Mehr zum Thema

- Bernd Heisterkamp, Task-Reminder, Make 1/24, S. 8
- Bernd Heisterkamp, Smarte Werkstattboxen, Make 5/23, S. 38
- Carsten Wartmann, Internet-of-Things-Dienste für Maker, Make 3/21, S. 32

Alles zum Artikel im Web unter
make-magazin.de/xe7r

ist. Die Funktion `sendData()` verpackt daher die Temperatur und die Luftfeuchtigkeit, die der BME280 ermittelt hat, in ein JSON-Dokument und sendet es an den Client, in diesem Fall die Automatisierungs-Apps. Der Status-Code ist immer 200 (OK) und der MIME-Typ ist `application/json`, sodass anfragende Clients gleich wissen, womit sie es zu tun haben.

In der `setup()`-Funktion initialisiert das Programm zunächst die serielle Schnittstelle und den BME280-Sensor. Anschließend erzeugt es eine WLAN-Verbindung und bindet dann den Pfad `/data` an die Funktion `sendData`. Damit wird jeder Aufruf der URL http://esp32/data von der Funktion `sendData` beantwortet. Der Aufruf von `server.begin()` startet den Web-Server. Die `loop()`-Funktion prüft danach permanent, ob eine neue Anfrage vorliegt und bearbeitet sie gegebenenfalls.

Sobald das Programm übersetzt und auf das ESP32-Board geladen wurde, sollte der ESP32 unter dem Namen „esp32“ im Netz bekannt sein. Jeder Browser im selben Netzwerk kann dann über die URL http://esp32/data auf die Messdaten zugreifen (Bild 1).

Der Zugriff auf den Sensor über einen Browser ist für Testzwecke bequem, hat aber auch ein paar handfeste Nachteile. Zum einen ist das JSON-Format zwar nicht allzu kompliziert, aber es ist sicherlich nicht das, was die meisten Anwender sehen wollen. Zum anderen erfolgt die Abfrage der Daten explizit und muss durch den Nutzer jedes Mal angestoßen werden. Darüber hinaus ist das Ganze keine wirkliche Anwendung, die zum Beispiel auf besonders hohe oder niedrige Temperaturen automatisch reagieren könnte.

Um die Informationen ansprechender zu präsentieren und auf spezielle Ereignisse gesondert reagieren zu können, benötigen Android-Geräte eigentlich eine dedizierte App. Ich stelle im Folgenden drei sogenannte Automatisierungs-Apps vor, die dies (und noch viel mehr) leisten können.

Automate

So ganz ohne irgendeine Form der Programmierung geht es dann aber doch nicht, aber Automate von LlamaLab basiert auf einer grafischen Programmierumgebung, die ein wenig an die Programmierumgebung MIT Scratch erinnert. Statt Quelltexte in einem Text-Editor einzugeben, modellieren Anwender ihre Problemlösung grafisch in Form von Flussdiagrammen, kurz Flows (Bild 2).

Dafür stellt Automate knapp 400 Blöcke zur Verfügung, mit denen Anwender auf so gut wie jede Funktion ihres Smartphones zugreifen können. Ein Ablauf, der auf den ESP32 per HTTP zugreift und die Sensor-Daten anschließend in Form einer Notifikation auf dem Smartphone anzeigt, besteht aus gerade einmal acht Blöcken.

Jeder Flow beginnt mit einem Start-Block und im Beispiel delegiert dieser die eigentliche Arbeit direkt an einen Block für HTTP-Anfragen (Bild 3). Ein solcher Block bietet eine große Menge an Konfigurationsmöglichkeiten: Neben der anzufragenden URL, kann man hier die HTTP-Methode (GET, PUT, POST etc.) und noch viele weitere Eigenschaften festlegen.

Unter anderem ist es möglich, die Rückgabewerte des Servers, also den Status-Code und den eigentlichen Inhalt, automatisch verschiedenen Variablen zuzuweisen. Auf diese Variablen können Blöcke im weiteren Verlauf des Flows zugreifen. Im aktuellen Beispiel landet der Status-Code in der Variable `status` und der Antwort-Text, also das JSON-Dokument, steht in der Variable `response`.

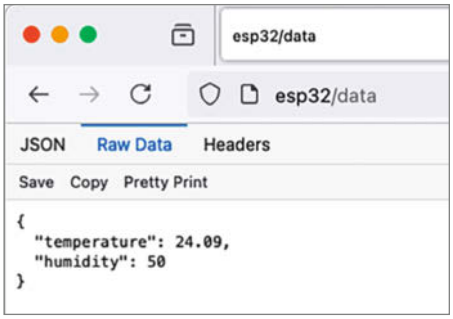


Bild 1: Für erste Tests mit dem Web-Sensor eignen sich alle aktuellen Browser.

Mit einem Entscheidungs-Block testet der Flow dann, ob der HTTP-Status den Wert 200 hat. Wenn dem nicht so ist, erzeugt der Flow eine Notifikation mit einer Fehlermeldung. Andernfalls geht der Prozess mit einem Funktions-Block weiter, der den Inhalt der Variable `response` mit der Funktion `jsonDecode` in eine Dictionary namens `sensorData` umwandelt (Bild 4). Diese Dictionary enthält die Schlüssel `temperature` und `humidity` mit den dazugehörigen Messwerten. Mit einem Notifikations-Block gibt der Flow diese Werte anschließend hübsch aufbereitet aus (Bild 5) und der Flow ist damit beendet.

Sobald der Flow fertig ist, wird es Zeit, ihn zu testen. Die Automate-App listet alle erstellten oder heruntergeladenen Flows auf und bietet die Möglichkeit, jeden einzelnen per Knopfdruck zu starten (Bild 6). Je nachdem, welche Blöcke ein Flow verwendet, kann es notwendig sein, der App noch entsprechende Berechtigungen zu erteilen.

Wenn der ESP32 empfangsbereit und das Smartphone mit demselben WLAN verbunden ist, sollte der Flow in Sekundenschnelle eine Benachrichtigung mit den aktuellen Messdaten anzeigen (Bild 7). So einfach kann die Integration zwischen einem Smartphone und einem Mikrocontroller aussehen.

Da geht noch was

Natürlich ist das nur die Spitze des Eisbergs und Automate kann noch viel mehr: Beispielsweise können die Flows richtige Benutzeroberflächen erzeugen und sie können sowohl zeitgesteuert als auch bei bestimmten Ereignissen gestartet werden. Der Beispiel-Flow könnte also auch periodisch für eine Notifikation mit den Messdaten sorgen oder zum Beispiel nur eine Notifikation sen-

SensorWebServer.ino

```
#include <WiFi.h>
#include <NetworkClient.h>
#include <WebServer.h>
#include <ESPmDNS.h>
#include <Adafruit_BME280.h>

const char* SSID = ".........";
const char* PASSWORD = ".........";

WebServer server(80);
Adafruit_BME280 sensor;

void initWiFi() {
  WiFi.mode(WIFI_STA);
  WiFi.begin(SSID, PASSWORD);
  Serial.println("");

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());

  if (MDNS.begin("esp32")) {
    Serial.println("MDNS responder started");
  }
}

void sendData() {
  String message = "{\n";
  message += "  \"temperature\": ";
  message += String(sensor.readTemperature(), 2);
  message += ",\n  \"humidity\": ";
  message += String(sensor.readHumidity(), 0);
  message += "\n}\n";
  server.send(200, "application/json", message);
}

void setup() {
  Serial.begin(115200);
  if (!sensor.begin(0x76)) {
    Serial.println("BME280-Sensor wurde nicht gefunden!");
    while (true)
      delay(10);
  }
  initWiFi();
  server.on("/data", sendData);
  server.begin();
}

void loop() {
  server.handleClient();
  delay(2);
}
```

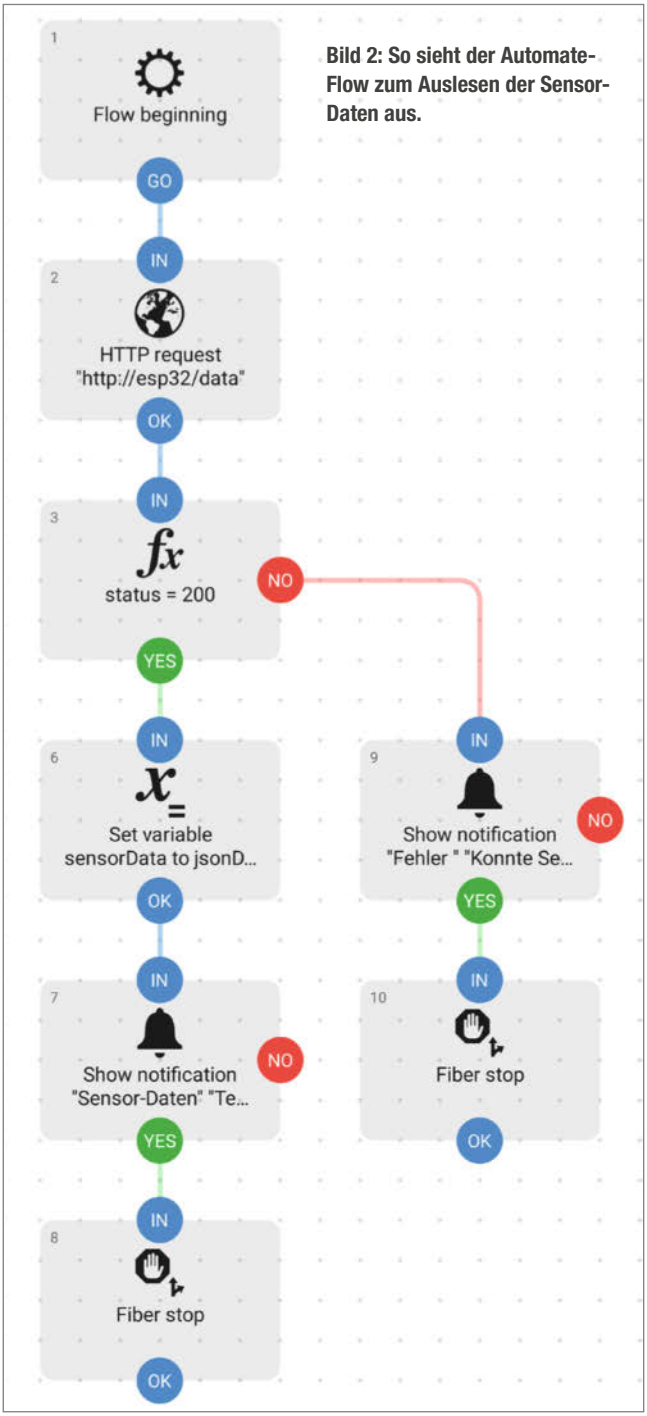


Bild 2: So sieht der Automate-Flow zum Auslesen der Sensor-Daten aus.

den, wenn die Temperatur zu hoch oder zu niedrig ist.

Weil Flows prinzipiell auf alle Funktionen des Telefons zugreifen und beliebig viele Flows parallel laufen können, ist es möglich, Beschränkungen festzulegen. Diese können Prozesse zum Beispiel daran hindern, zu lang zu telefonieren oder zu viele SMS zu senden. Auch Beschränkungen, die zur Schonung des Akkus dienen, sind dabei.

Diese Beschränkungen sind nicht zuletzt deshalb wichtig, weil man Flows mit Automate exportieren und teilen kann. Es gibt im Internet eine Community, die nützliche Abläufe teilt, aber wie immer ist es in solchen Fällen ratsam, lieber zweimal hinzusehen, was der Flow im Detail tut.

Allerdings ist das gar nicht so einfach, denn Flows können schnell sehr kompliziert werden und die Debugging-Möglichkeiten sind recht eingeschränkt, weil es den Editor nur auf dem Telefon gibt. Automate arbeitet intern mit kooperativem Multitasking und erlaubt auch die asynchrone Ausführung von Blöcken. Wenn dann auch noch die Integration mit anderen Apps auf dem Telefon ins Spiel kommt, kann man schon mal den Überblick verlieren.

Prinzipiell ist Automate kostenlos, aber dann dürfen Flows nur bis zu 30 Blöcke haben. Für viele Projekte ist das ausreichend, aber wer mehr benötigt, bekommt für einmalig 3,90 Euro ein Upgrade auf die Pro-Version.

MacroDroid

Eine Anwendung, die ganz ähnlich arbeitet wie Automate, heißt MacroDroid (siehe Link in der Kurzinfo). Allerdings hat diese App ein etwas anderes Bedienkonzept und ein eher ungewöhnliches Design.

Das beginnt schon beim Startbildschirm, der im bunten Kachel-Look daherkommt (Bild 8). Dieser wirkt auf den ersten Blick etwas unruhig, aber nach einer kurzen Gewöhnungsphase erweist sich das Ganze als sehr effizient und flexibel.

Im Fokus von MacroDroid stehen sogenannte Makros, die im Wesentlichen den Flows von Automate entsprechen. Sie werden aber auf etwas andere Weise erstellt und bestehen immer aus drei Komponenten: Auslösern, Aktionen und Bedingungen (Bild 9).

Auslöser starten ein Makro und kommen in vielfältigen Ausprägungen daher. Beispielsweise kann ein Button auf dem Display ein Makro starten, aber es kann auch ein NFC-Tag oder eine bestimmte Uhrzeit sein. MacroDroid kann auch auf Geo-Koordinaten, IDs von Mobilfunkmasten und vieles mehr reagieren. Die Verknüpfung von Auslösern ist ebenfalls möglich. Allerdings werden die verschiedenen Auslöser immer mit einem logischen ODER verknüpft. Das heißt, sobald eines der Ereignisse eintritt, startet MacroDroid das Makro.

Die Aktionen legen fest, was passieren soll, sobald das Makro läuft (Bild 12). Wie bei Automate ist hier eine ganze Menge möglich und die Bandbreite reicht von Ausgaben auf dem Display bis hin zum Versenden von E-Mails oder WhatsApp-Nachrichten. Der Zugriff auf

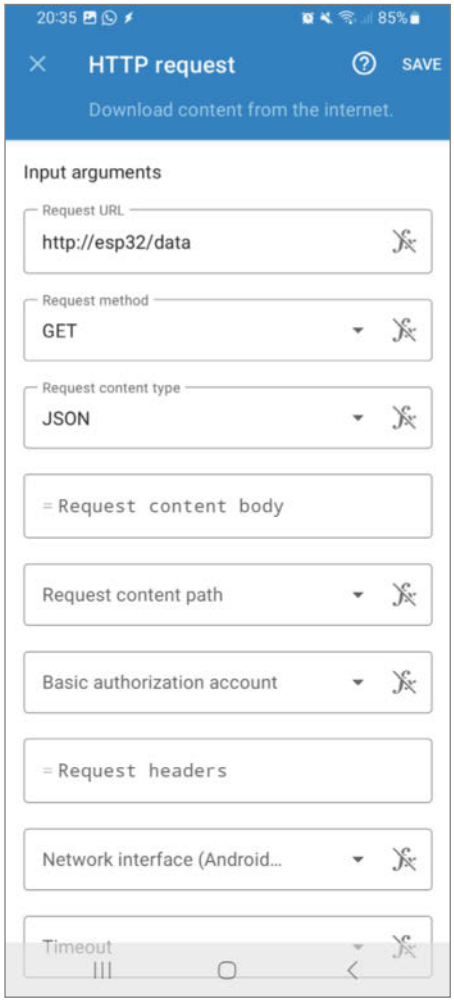


Bild 3: Auch für HTTP-Anfragen gibt es einen Block.

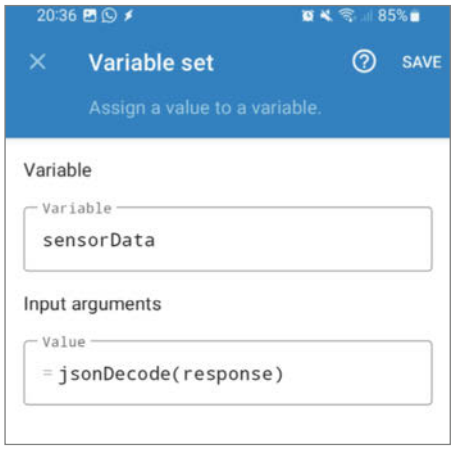


Bild 4: Mit Automate ist die Verarbeitung von JSON-Dokumenten ein Klacks.

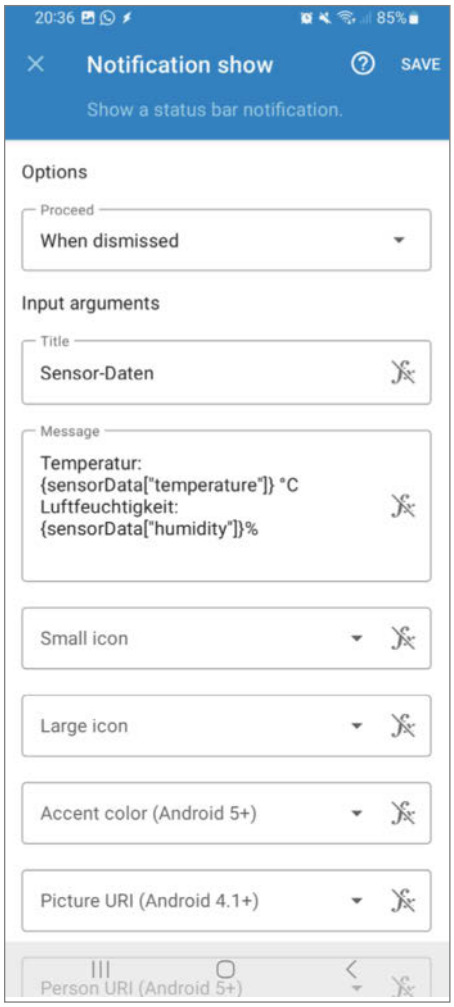


Bild 5: Zur Ausgabe von Informationen kann Automate unter anderem Android-Benachrichtigungen verwenden.

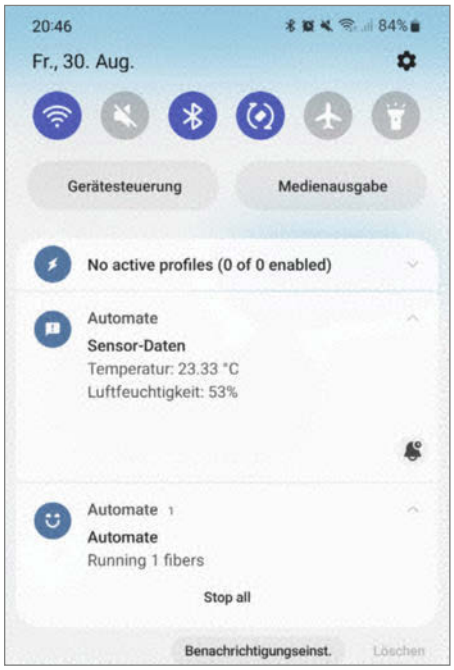


Bild 7: Ein kurzer Flow reicht, um die Sensor-Daten als Benachrichtigung anzuzeigen

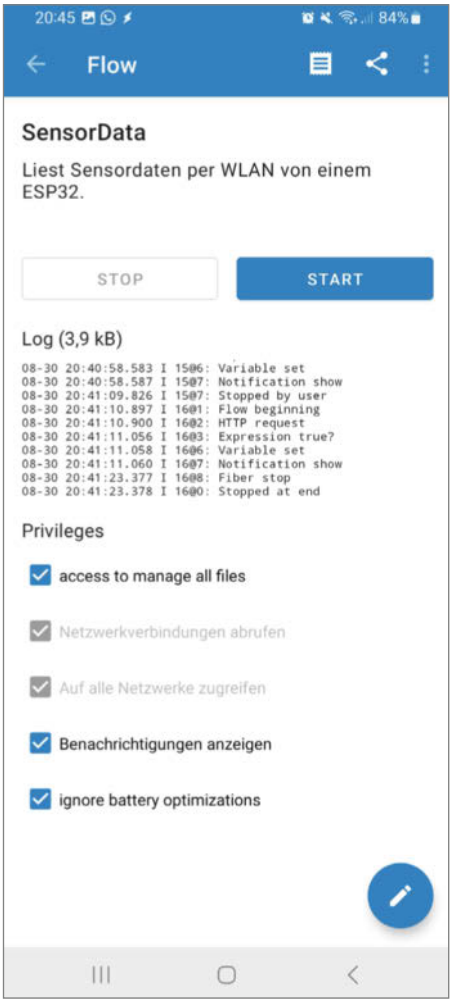


Bild 6: Nutzer können Flows auf vielfältige Arten starten.

Telefonie und Internet-Dienste ist ebenfalls kein Problem.

Schließlich gibt es noch Bedingungen, die festlegen, unter welchen Umständen ein Makro seine Aufgabe erledigen kann. Manche Makros sind etwa auf eine Bluetooth-Verbindung angewiesen oder bestehen darauf, dass ein Kopfhörer eingestöpselt ist. Solche Voraussetzungen können die Bedingungen sicherstellen.

Das eigene Makro

Beim Anlegen eines neuen Makros sind Auslöser, Aktionen und Bedingungen noch leer. Für das ESP32-Beispiel soll ein frei beweglicher Button als Auslöser dienen. Natürlich kann es auch ein beliebiges anderes Ereignis sein, aber zum Testen sind Buttons oft dankbarer als zum Beispiel zeitgesteuerte Abläufe.

Die Aktionen stehen bei Makros in der Regel im Fokus und das gilt auch für den Zugriff auf den Web-Sensor. Wenig überraschend ist daher die erste Aktion eine HTTP-Anfrage an die URL http://esp32/data (Bild 10). Wie Automate kann auch MacroDroid die Antwort des Web-Dienstes automatisch einer Variablen zuweisen. Wie zuvor bekommt diese Variable den Namen `response`.

Nach einer erfolgreichen HTTP-Anfrage enthält `response` ein JSON-Dokument, das MacroDroid mit einer Aktion des Typs „JSON-Parsing" in eine Dictionary namens `sensorData` umwandelt. Die dekodierten Messwerte gibt das Makro dann mit einer entsprechenden Aktion als Benachrichtigung aus.

Wie auch bei Automate kommt bei der Formulierung der Benachrichtigung eine Template-Sprache mit vielen eckigen und geschweiften Klammern zum Einsatz (Bild 11). Programmierer sind so etwas gewohnt, aber allzu komfortabel ist das nicht. Dankenswerterweise hat der Entwickler von MacroDroid seiner App eine gewisse Intelligenz verliehen. Sie kann die Struktur der JSON-Dokumente nämlich lernen, wenn das Makro während der Entwicklung schon mal gestartet wurde.

Beim Bearbeiten des Benachrichtigungstextes kann der Nutzer dann die lokale Variable `sensorData` auswählen und MacroDroid präsentiert eine Liste aller möglichen Schlüssel der entsprechenden Dictionary (Bild 12).

Damit ist das Makro komplett und einem ersten Test steht nichts im Wege (Bild 13). Dazu bietet die App eine Vielzahl von Möglichkeiten und die einfachste führt über den Menüpunkt „Makro testen". Hier finden sich noch jede Menge weiterer nützlicher Funktionen, zum Beispiel die Option, einzelne Aktionen zu testen. Auch zur Verwaltung und zum Teilen von Makros gibt es an dieser Stelle allerlei Möglichkeiten.

Wenn alles glatt läuft, verbindet sich das Makro mit dem ESP32 und gibt die aktuellen Messwerte in Form einer Benachrichtigung aus (Bild 14). Andernfalls zeigt das Makro mangels einer expliziten Fehlerbehandlung einen Java-Stacktrace an. Erfahrenen Entwicklern dürfte der oft aber schon weiterhelfen.

Selbstverständlich ist es mit MacroDroid auch möglich, solche Fehler ordentlich abzufangen. Die Dokumentation des Projekts in Form eines Wikis ist ausreichend, aber lange nicht so ausführlich und strukturiert wie die von Automate. Dafür gibt es einen eigenen YouTube-Kanal mit Tutorial-Videos.

Prinzipiell ist MacroDroid kostenlos, aber die Gratis-Variante hat ein paar Einschränkungen bezüglich der Anzahl an Makros, und auch innerhalb der Makros scheint es ein paar Restriktionen zu geben. Beispielsweise war es bei Tests mit der Gratis-Version nicht möglich, das Ergebnis einer JSON-Parsing-Aktion einer Variablen zuzuweisen.

Zu all dem kommt, dass die kostenlose Version nur eine Woche lang funktioniert und Werbung anzeigt. Pro angezeigter Werbe-Anzeige verlängert sich die Laufzeit der App immerhin um drei weitere Tage. Um diese Beschränkungen aufzuheben, wird eine einmalige Gebühr von 5,99 Euro fällig.

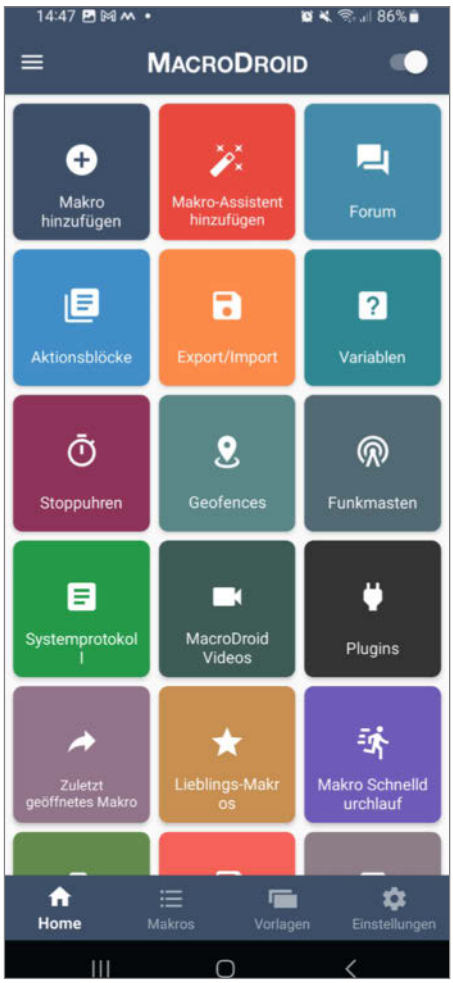


Bild 8: Der Startbildschirm von MacroDroid ist bunt.

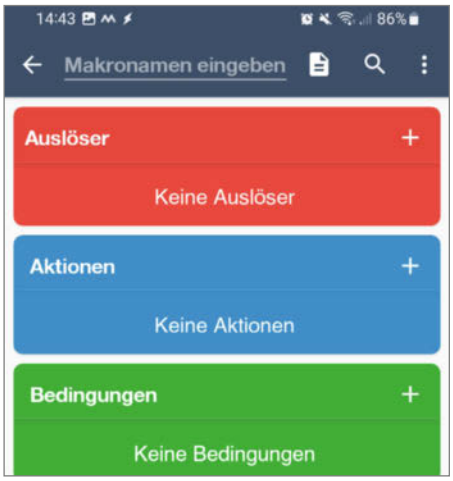


Bild 9: Makros bestehen in MacroDroid aus drei Kernkomponenten.

Tasker

Eine der ältesten und ausgereiftesten Applikationen für die Automatisierung von Aufgaben auf Android-Geräten ist Tasker. Diese App ist so etwas wie das Schweizer Taschenmesser der Automatisierung und hat über die Jahre eine treue Fan-Gemeinde um sich geschart.

In Bezug auf die Bedienung erinnert Tasker ein wenig an MacroDroid, ist aber längst nicht so bunt und einladend. Ganz im Gegenteil: Die Oberfläche wirkt eher trist, spartanisch und beinahe abweisend (Bild 15).

Das gilt auch für die Dokumentation, hat aber eine Unmenge von Entwicklern nicht davon abgehalten, eine Vielzahl von Aufgaben

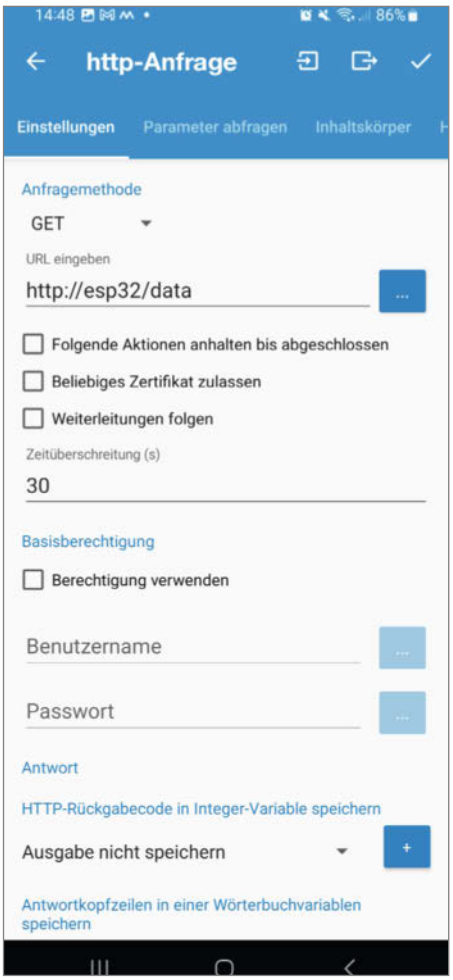


Bild 10: Die HTTP-Aktion von MacroDroid bietet alles, was HTTP-Clients brauchen.

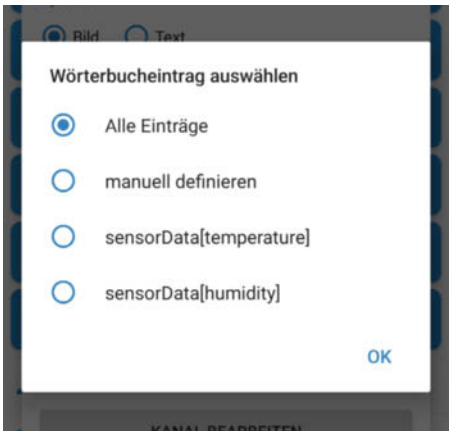


Bild 11: MacroDroid wandelt die Inhalte von JSON-Dokumenten automatisch in Referenzen für die eigene Template-Sprache um.

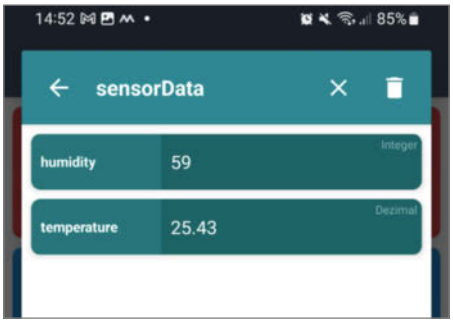


Bild 12: MacroDroid hat aus dem JSON-Dokument eine Dictionary namens `sensorData` gemacht.

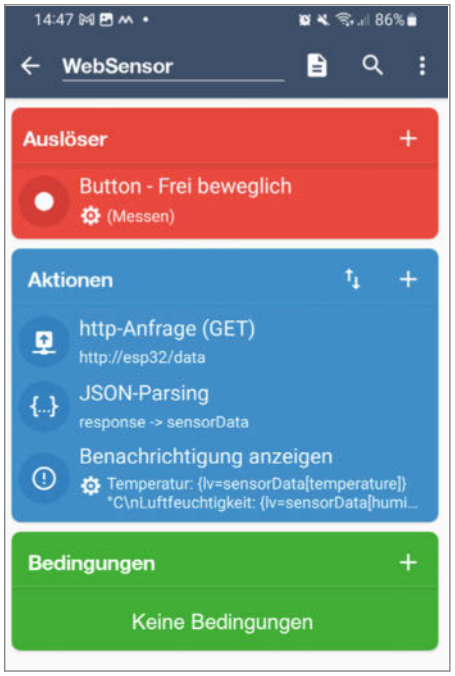


Bild 13: Ein Makro für den Zugriff auf den ESP32-Web-Sensor passt bequem auf eine Bildschirmseite.

zu automatisieren und die Lösungen im Nachhinein zu teilen.

Ein erster Blick

Beim ersten Öffnen von Tasker müssen Nutzer sich entscheiden, ob sie die Anwendung im Original-Modus oder im Einsteiger-Modus (Beginner Mode) starten wollen. Der Einsteiger-Modus ist in vielerlei Hinsicht gar nicht so viel einfacher und so empfiehlt sich gleich der Griff zum Original-Modus.

Trotz der Ähnlichkeiten zu MacroDroid und zu Automate ist Tasker doch ein wenig anders und bedarf einer gewissen Einarbeitung. Nach dem ersten Start ist der Bildschirm beinahe leer und zeigt nur vier Tabs mit den Namen „Profile", „Tasks", „Szenen" und „Var".

Profile sind vergleichbar mit den Auslösern in MacroDroid. Sie bestimmen, unter welchen Umständen Tasker bestimmte Aktionen (Tasks) auslöst. Im Gegensatz zu MacroDroid sind die Profile aber deutlich komplexer. Sie haben Namen, Kontexte und auszuführende Aufgaben. Nur wenn alle Kontexte eines Profils erfüllt sind, wird das Profil aktiv und startet die zugewiesenen Tasks. So kann man in Tasker etwa ganz einfach ein Profil definieren, das nur aktiv wird, wenn der Akkustand niedriger als 10 Prozent und gerade keine Bluetooth-Verbindung aktiv ist.

Tasks entsprechen den Aktionen in MacroDroid und Automate und sie können so gut wie alles auf dem Smartphone oder Tablet tun. Sie können beispielsweise die Lautstärke auf Null reduzieren oder eine SMS senden. Tasker kann Tasks beliebigen Profilen zuweisen, das heißt, es ist möglich, Tasks in verschiedenen Situationen wiederzuverwenden.

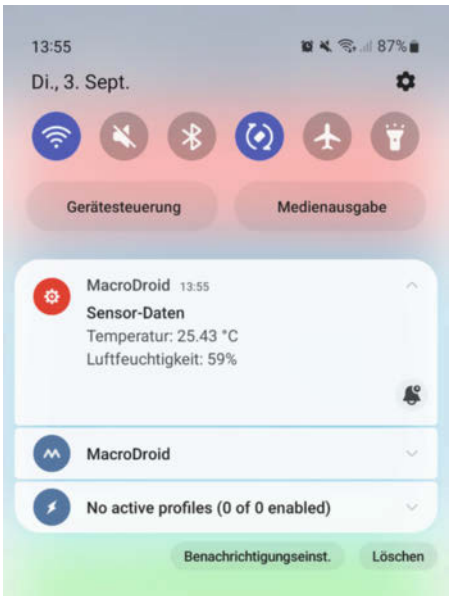


Bild 14: Auch mit MacroDroid ist es kein Problem, die Messwerte als Benachrichtigung anzuzeigen.

Mit Szenen lassen sich Benutzeroberflächen definieren, die Profile und Tasks nutzen können. Die Bandbreite ist recht groß . Sobald ein bestimmtes Ereignis eintritt, kann Tasker beispielsweise einen Dialog öffnen, der den Nutzer über Details des Ereignisses informiert. Es lassen sich aber auch Widgets erstellen oder Overlays und Erweiterungen für bestehende Applikationen implementieren.

Zu guter Letzt gibt es noch einen eigenen Tab für Variablen, die Tasker etwas anders behandelt als Automate und MacroDroid. Tasks können lokale Variablen definieren, deren Namen immer aus Kleinbuchstaben bestehen. Im Variablen-Tab tauchen alle Variablen auf, die nicht lokal sind, und auf die können unterschiedliche Tasks zugreifen und so Informationen austauschen.

Wer tiefer in Tasker einsteigen möchte, muss das Zusammenspiel von Profilen, Tasks, Szenen und Variablen verinnerlichen. Weil das Zusammenspiel zwischen diesen Komponenten so wichtig ist, bietet Tasker auch die Möglichkeit, sie zu Projekten zusammenzufassen. Diese können Nutzer mit anderen teilen und sogar im Google Play Store zum Verkauf anbieten.

Ein Tasker-Beispiel

Mit Tasker ist es keine allzu große Sache, den Web-Sensor abzufragen und die Messdaten als Notifikation anzuzeigen. Zum Einstieg bietet sich die Definition eines neuen Tasks an, der eine HTTP-Aktion startet. Die befindet sich in der Netzwerk-Kategorie.

Wie zuvor ist die URL http://esp32/data einzutragen, um auf den ESP32 zuzugreifen, aber im Gegensatz zu MacroDroid und Automate bietet die HTTP-Aktion von Tasker nicht an, die Ergebnisse der Anfrage in Variablen abzulegen. Stattdessen geht Tasker einen anderen Weg und legt die Antwort automatisch in einer Variable mit dem Namen `http_data` ab. Das Tool geht sogar noch einen Schritt weiter und wandelt strukturierte Daten wie JSON- oder XML-Dokumente automatisch um.

Darüber hinaus legt Tasker auch Variablen für Cookies, HTTP-Header, die Länge der Antwort und weitere Attribute an (Bild 16).

Dieses Verhalten ist ziemlich praktisch und macht die Definition der Notifikation sehr einfach. Dazu muss man lediglich eine Benachrichtigungsaktion erstellen und deren Text mit den Inhalten der Variable `http_data` füllen. Auch Tasker arbeitet mit einer Template-Sprache und in dieser werden Variablen mit einem Prozentzeichen identifiziert. Um also auf die Variable `http_data` zu verweisen, muss man die Zeichenkette `%http_data` verwenden. Der Zugriff auf das Feld `temperature` im JSON-Dokument erfolgt dann über den Ausdruck `%http_data[temperature]` (Bild 17).

All das ist möglich, weil Tasker eine Menge Aufgaben implizit und hinter den Kulissen erledigt. Unter anderem übernimmt Tasker die Umwandlung des JSON-Dokuments und legt gleichzeitig entsprechende Variablen an. Manchen Entwicklern gefällt das, während andere gerne die Kontrolle behalten.

Für einen ersten Test reichen die wenigen Handgriffe aber schon aus (Bild 18) und ein Klick auf den Play-Button unten links sorgt dafür, dass eine neue Benachrichtigung auf dem Gerät erscheint (Bild 19).

Mit der Zeit gehen

Mit den Tasker-Profilen ist es auf viele verschiedene Arten möglich, den Web-Sensor abzufragen. Naheliegend ist eine zeitliche Steuerung, in der Tasker zum Beispiel alle fünf Minuten automatisch eine Anfrage startet.

Mit dem Zeit-Profil ist das trivial, denn das wurde genau für solche zeitgesteuerten Aufgaben geschaffen. Neben einer Start- und End-Uhrzeit kann man dort ein Intervall angeben, in dem Tasker eine bestimmte Aufgabe anstößt. So lässt sich beispielsweise zwischen 8 und 22 Uhr alle fünf Minuten die aktuelle Temperatur ermitteln (Bild 20).

Das Editieren des Profils ist recht leicht, aber beim Aktivieren ist es wichtig, nicht nur den Slider des Profils nach rechts zu schieben, sondern auch das Häkchen im oberen Bereich des Bildschirms anzuklicken. Anschließend verrichtet Tasker brav alle fünf Minuten seine Arbeit.

Von den vorgestellten Werkzeugen ist Tasker das mächtigste, aber es ist leider auch ein wenig störrisch. Die Benutzeroberfläche ist oft wenig intuitiv und die Dokumentation ist alles andere als einsteigerfreundlich. Selbst der Beginner-Modus sorgt kaum für Verbesserung, denn er versteckt hauptsächlich Funktionen wie die Tabs für Variablen und Projekte.

Bild 15: Tasker wirkt auf den ersten Blick sehr spartanisch.

Bild 16: Tasker legt alle wesentlichen Teile einer HTTP-Antwort automatisch in Variablen ab.

Die Test-Version von Tasker ist sieben Tage lang aktiv. Danach werden 3,59 Euro fällig.

Fazit

Automatisierungs-Apps sind auf Android-Geräten erstaunlich nützlich und leistungsstark. Die drei vorgestellten sind nicht allein im Play Store, aber sie gehören zu den beliebtesten.

Mit wenigen Klicks können Nutzer mit solchen Apps selbst komplexe Aufgaben automatisieren und das hilft nicht nur bei Maker-Projekten, sondern auch bei vielen Problemen des Alltags. Wer zum Beispiel einen ersten

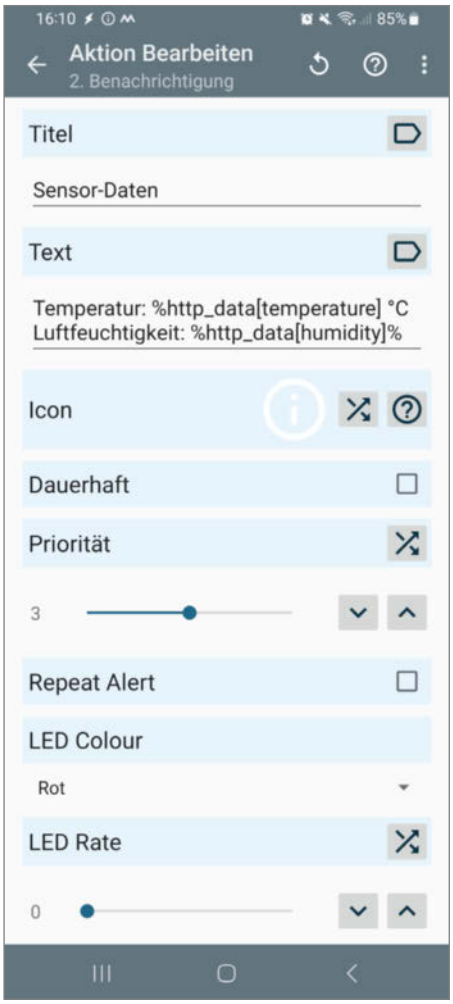


Bild 17: Die Template-Sprache macht den Zugriff auf Attribute eines JSON-Dokuments recht einfach.

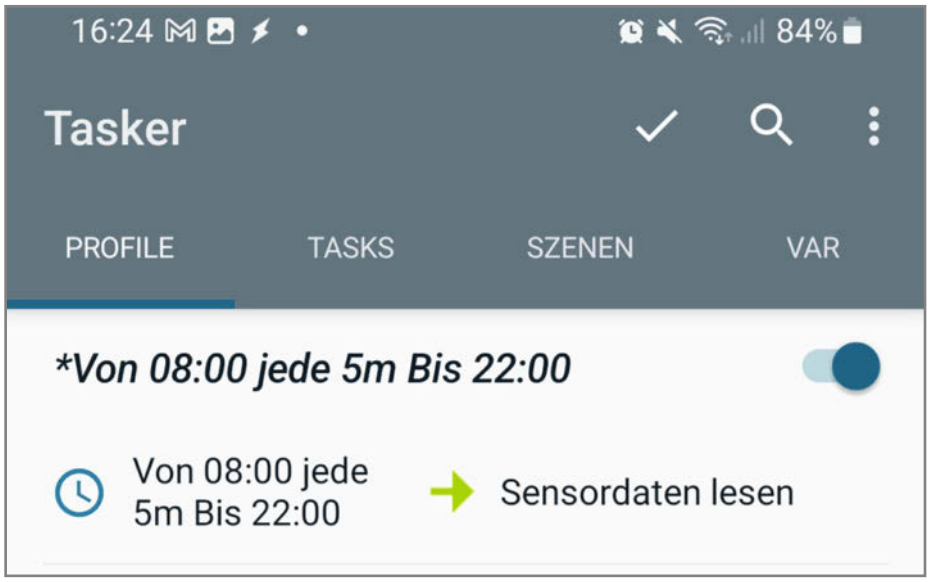


Bild 20: Dieses Profil sorgt dafür, dass Tasker zwischen 8 und 22 Uhr alle fünf Minuten aktiv wird.

Flow erstellt hat, der automatisch eine Nachricht sendet, wenn man nur noch zwei Kilometer von seinem Zuhause entfernt ist, bekommt ein gutes Gefühl für das, was alles möglich ist.

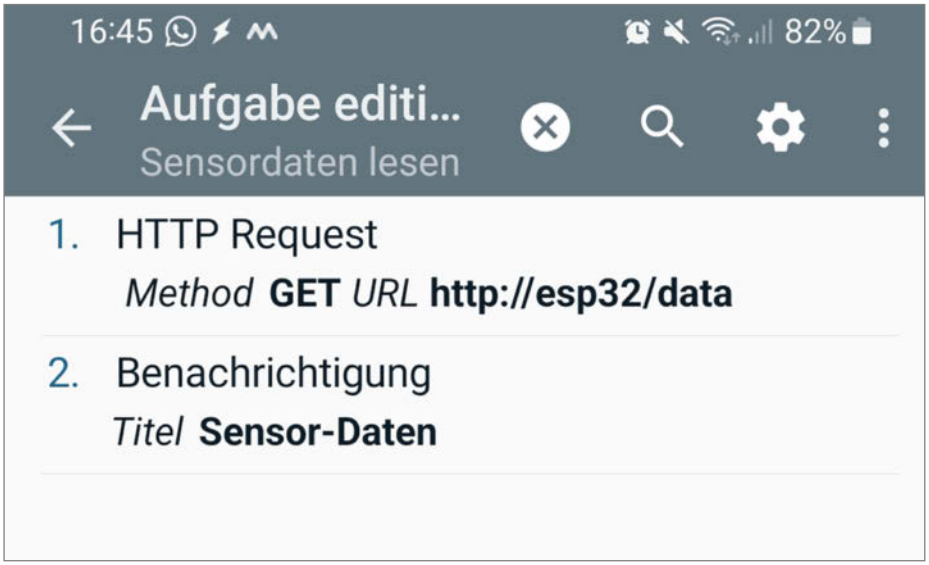
Das Editieren der Prozesse auf dem Telefon ist manchmal ein wenig friemelig, aber trotzdem sind die Oberflächen der vorgestellten Helfer so gut, wie es eben geht. Allerdings sind Nutzer mit guten Englischkenntnissen bei allen Apps klar im Vorteil, denn selbst wenn es eine deutsche Übersetzung der Oberfläche gibt, ist sie zumeist nicht sonderlich gut.

Das schlägt sich nicht zuletzt in den Suchfunktionen der Apps nieder und so macht es bei Tasker beispielsweise einen großen Unterschied, ob man nach „Notification“ oder „Benachrichtigung“ sucht. Beide Anfragen liefern Ergebnisse, aber sie sind völlig unterschiedlich.

Natürlich haben solche Apps auch ein paar Nachteile. Sie können beispielsweise die Akku-Laufzeit ordentlich verkürzen. Ferner benötigen sie für manche Aktionen wie den Start eines Telefonanrufs weitgehende Berechtigungen. Hier ist insbesondere bei der Verwendung von Flows aus dem Internet Vorsicht geboten. Seit Android 14 sind daher bestimmte Aktionen prinzipiell nicht mehr möglich.

Alles in allem sind diese Automatisierer für viele Anwendungen eine echte Alternative zu nativen Apps. *—mch*

Bild 18: Zwei Schritte reichen in Tasker, um Daten vom ESP32 zu lesen und in eine Benachrichtigung zu verwandeln.



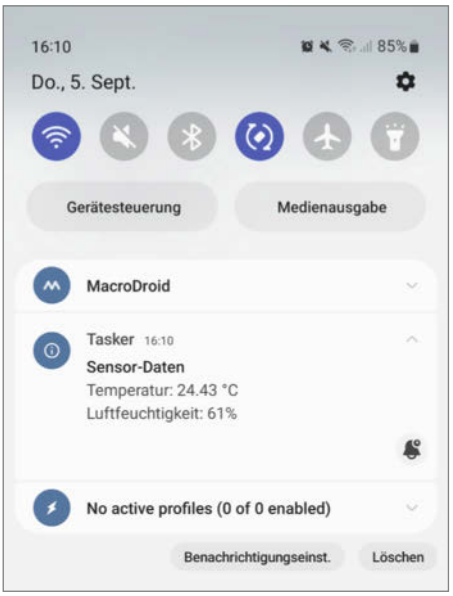


Bild 19: Tasker macht es leicht, die Daten des Web-Sensors als hübsch aufbereitete Benachrichtigung anzuzeigen.

Neuer Input für Maker

Make Elektronik Special

Make Elektronik Special bietet einen einfachen und praxisorientierten Einstieg in Transistorschaltungen, die Maker in eigenen Projekten einsetzen können. Das mitgelieferte Experimentierset inkl. Breadboard, Kabeln und 45 Elektronikbauteilen enthält alles, um die gezeigten Schaltungen sofort nachbauen und testen zu können.

Heft + Experimentierset für 44,95 €

shop.heise.de/make-elektronik21

Inklusive Experimentierset und Breadboard

Make Operationsverstärker Special

Das Make-Sonderheft bietet einen praxisorientierten Einstieg in Schaltungen mit Operationsverstärkern inkl. Experimentierset. Will man Sensorsignale verarbeiten oder verstärken, Spannungen überwachen oder Audiosignale filtern: Mit geringem Aufwand und ohne komplizierte Berechnungen setzt man Operationsverstärker ein. Das Heft erklärt, wie alle Schaltungen funktionieren.

Heft + Experimentierset für nur 49,95 €

shop.heise.de/make-opv

Inklusive Experimentierset

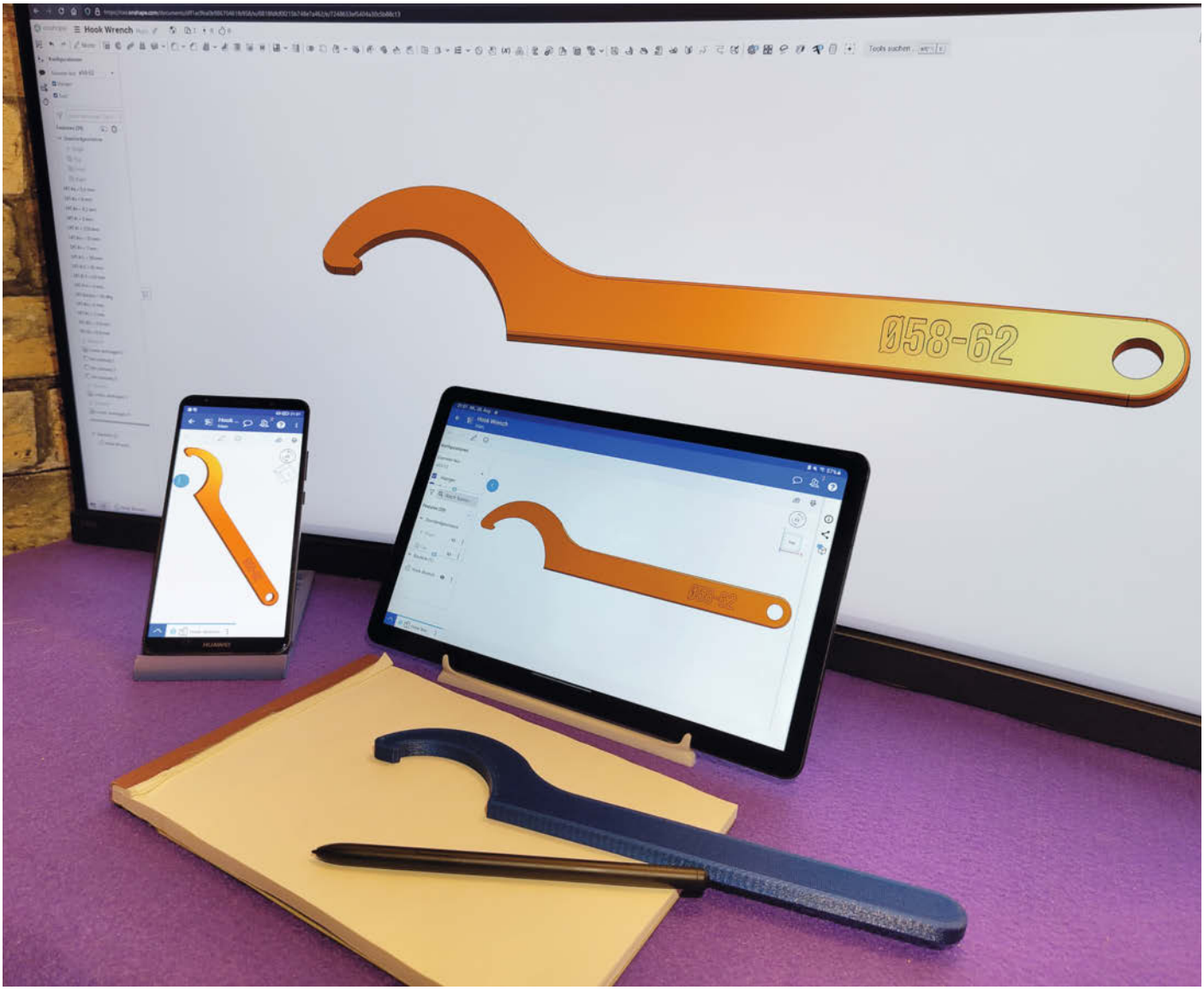
Make PI Pico Special

Mit dem Make Special Pi Pico steigen Sie ein in die Welt der Programmierung von ARM-Mikrocontrollern. Make zeigt in dem 64-seitigen Special, welche Entwicklungsumgebungen es für den Rasperry Pi Pico gibt, wie man sie installiert und wie man sie nutzt.

Heft + Raspberry Pi Pico für 24,95 €

shop.heise.de/make-pico

Inkl. Raspberry Pi Pico RP2040



Mobil 3D-konstruieren

Für ein umfassendes CAD-Programm braucht man nicht unbedingt eine Workstation. Nutzt man nämlich Onshape, ist die gesamte Rechenleistung auf die Server des Entwicklers ausgelagert. So kann man über eine Internetverbindung auch mit schwächeren Clients komplexe Modelle bauen. Wie gut das auch mit mobilen Geräten funktioniert, haben wir ausprobiert.

von Martin Siegmann

Die Mär der Lehrer, man werde nicht immer einen Taschenrechner bei sich haben, ist mit dem Smartphone und anderen Mobilgeräten nicht besonders gut gealtert. Gibt es doch für alles eine App mit der Möglichkeit, Dinge unterwegs zu tun. Aber wie sieht es mit Programmen aus, die man seit jeher nur auf Desktop-Systemen nutzt, z. B. CAD-Software? Mit Onshape bietet die Firma PTC eine praktikable Lösung an, die sogar auf mobilen Geräten läuft. Schauen wir uns diese doch mal näher an!

Kurzinfo

- Mit Onshape 3D-CAD-Modelle erstellen
- Unterschiede zwischen Browser-Variante und mobiler App
- Vor- und Nachteile der Software

Mehr zum Thema

- Martin Siegmann, Eigene Ideen 3D-drucken, Make 6/23, S. 106
- Matthias Mett, 3D-Entwurf mit FreeCAD, Make 1/21, S. 128
- Peter König und weitere Autoren, Gratis-3D-CAD für Maker, Make 4/22, S. 76

Alles zum Artikel im Web unter *make-magazin.de/xd9u*

Was ist Onshape?

Die CAD-Software Onshape ist ein cloudbasiertes Konstruktionsprogramm, das im Browser oder per App ausgeführt wird. Anders als etwa Tinkercad richtet es sich an professionelle Anwender. Allerdings gibt es für die nichtkommerzielle Verwendung auch ein „Free-Abo". Wie üblich bietet die Software in dieser Variante nicht den vollen Funktionsumfang, aber den braucht man als Hobby-Anwender in der Regel auch nicht (oder vielleicht nur in einzelnen Ausnahmefällen). Die kostenfreie Version bietet immerhin alle CAD-Funktionen zum Konstruieren, Revisionieren und Zeichnen. Beschränkt werden eher Zusatzmodule wie Simulationen, Rendering sowie Module für Freigaben oder Produktlebenszyklen, wie sie wirklich nur Unternehmen verwenden werden.

Alles in der Cloud

Als klassischer Desktop-Software-Nutzer, zu denen ich mich auch zählen würde, ist die Speicherverwaltung in der Cloud im ersten Moment etwas ungewohnt. Offene Dokumente werden stets „live" gespeichert und die Ordnerstruktur legt man vollständig auf dem Onshape-Server an. Dokumente sind in dem Free-Abo außerdem immer „öffentlich" und lassen sich nach bestimmten Kriterien suchen. Das fördert den Austausch und das gegenseitige Lernen, aber natürlich schwingt dabei auch mit, dass die eigenen Dateien nicht vor anderen Nutzern geschützt sind. Je nach Geheimprojekt muss jeder selbst entscheiden, ob er dafür auf ein kostenfreies Cloud-Tool zurückgreifen möchte.

Seine Konstruktionen über Links mit anderen zu teilen, hat konkret zwei Vorteile: Zum einen sind die Konstruktionen konfigurierbar (einfacher als mit OpenSCAD). Das heißt, die Person, der ich einen Entwurf schicke, kann mit dem Link auf eine angenehme Benutzeroberfläche zugreifen, die ich zuvor parametrisiert aufgebaut und freigegeben habe.

Zum Zweiten muss man keine aufwendige Ordnerstruktur per Mail oder USB-Stick ver-

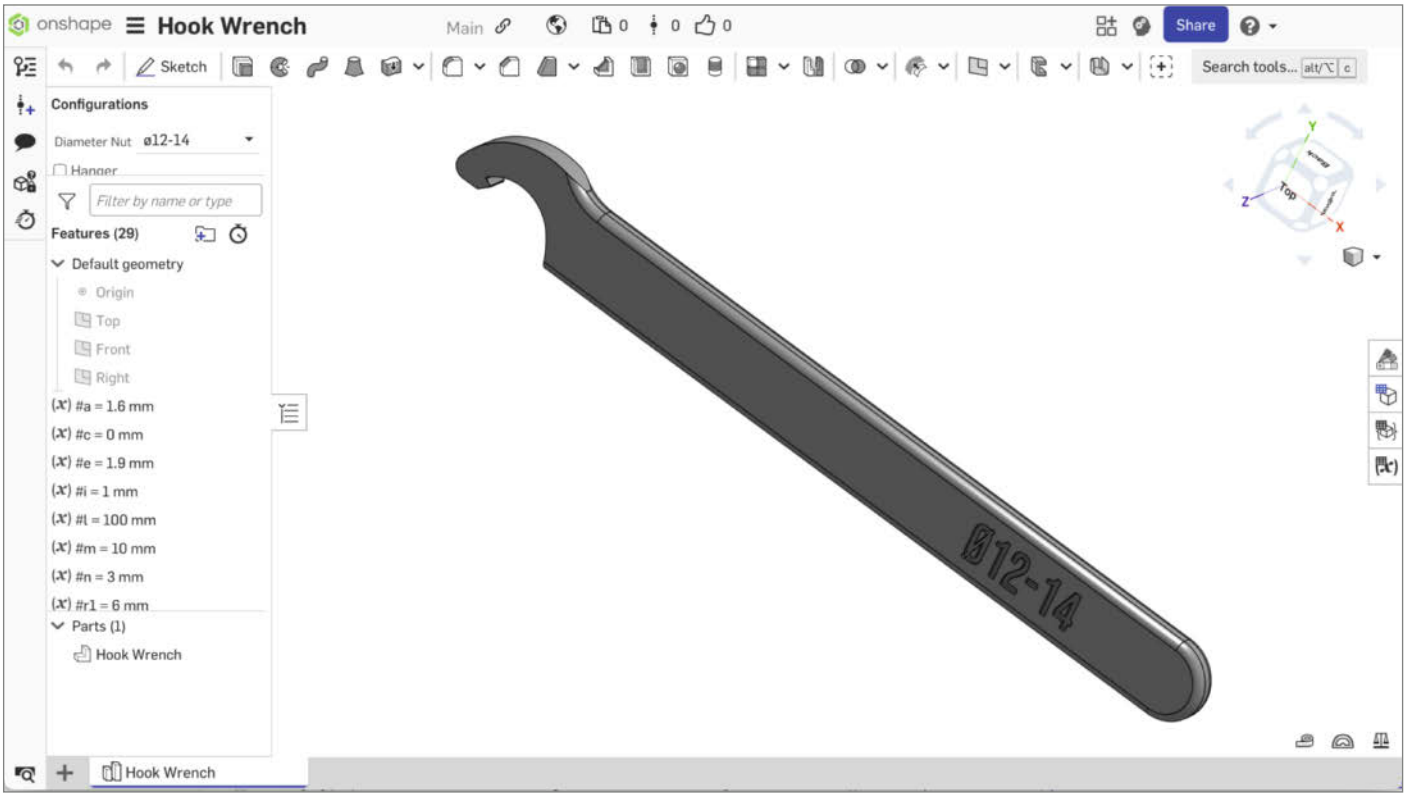


Bild 1: Onshape läuft zwar im Browser, sieht aber aus wie CAD-Programme am PC.

breiten, um jemanden nach seiner Meinung zu einer Konstruktion zu fragen oder mit ihm zusammenzuarbeiten. Man teilt einfach den Link und die Struktur dahinter ist schlicht dabei. Selbst wenn jemand keinen Onshape-Account hat, kann er sich Modelle mit nur einem Klick ansehen – auch unterwegs in der Onshape-App.

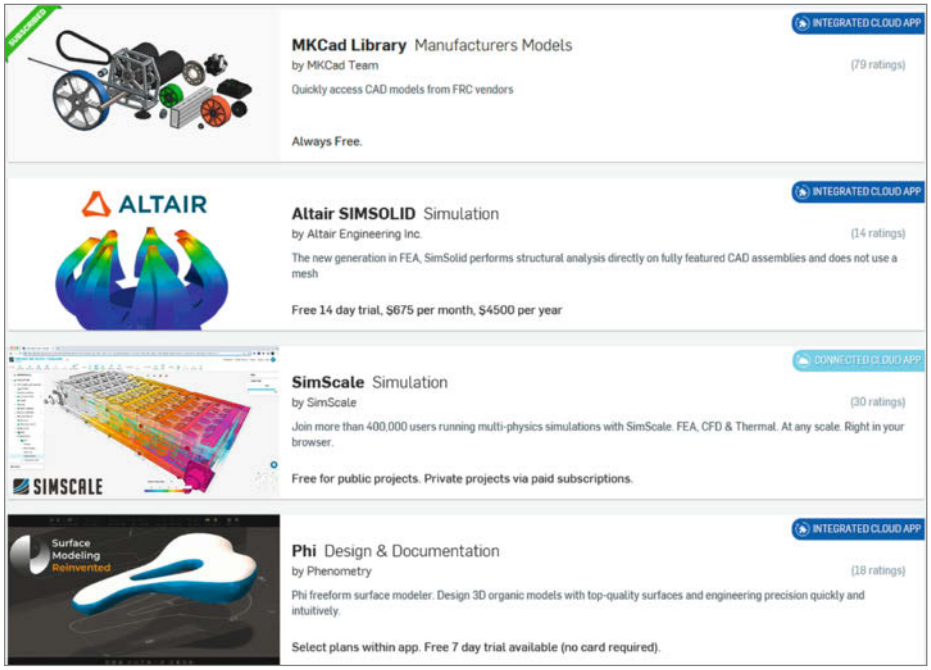


Bild 2: Die Funktion von Onshape lässt sich je nach Bedarf mit Zusatzmodulen erweitern.

Onshape im Browser

Mit dieser Variante liefert PTC eine klassische CAD-Software-Benutzeroberfläche und bietet einen entsprechend flachen Einstieg für erfahrene CAD-Nutzer. Über die Startseite erstellt man neue CAD-Dokumente oder öffnet bereits vorhandene Dateien. Daraufhin erscheint eine Modellierungsumgebung, wie man sie von gängigen CAD-Programmen kennt. Links ist der Strukturbrowser, oben das Befehlsmenü und das Modell in der Mitte des Bildschirms (Bild 1).

Diese gewohnte Umgebung ist zudem mit dem klassischen Hardware-Setup am PC kompatibel. Computermaus, Tastatur – und für versierte Nutzer noch eine 3D-Maus – ergeben die wohl effizienteste Befehlseingabe für ein Konstruktionsprogramm. Über den Monitor ist somit alles schön übersichtlich und die Eingabemöglichkeiten lassen sich an die eigenen Bedürfnisse anpassen.

In der Browser-Version sind zudem alle Zusatzmodule (Onshape-Apps) verfügbar und im vollen Leistungsspektrum nutzbar. Sie ermöglichen beispielsweise den Zugriff auf CAD-Bibliotheken oder Simulationen (Bild 2). Welche nutzbar sind, hängt vom verwendeten Onshape-Abo ab und wie viele Funktionen man gebucht hat. Auch ein paar üblichere

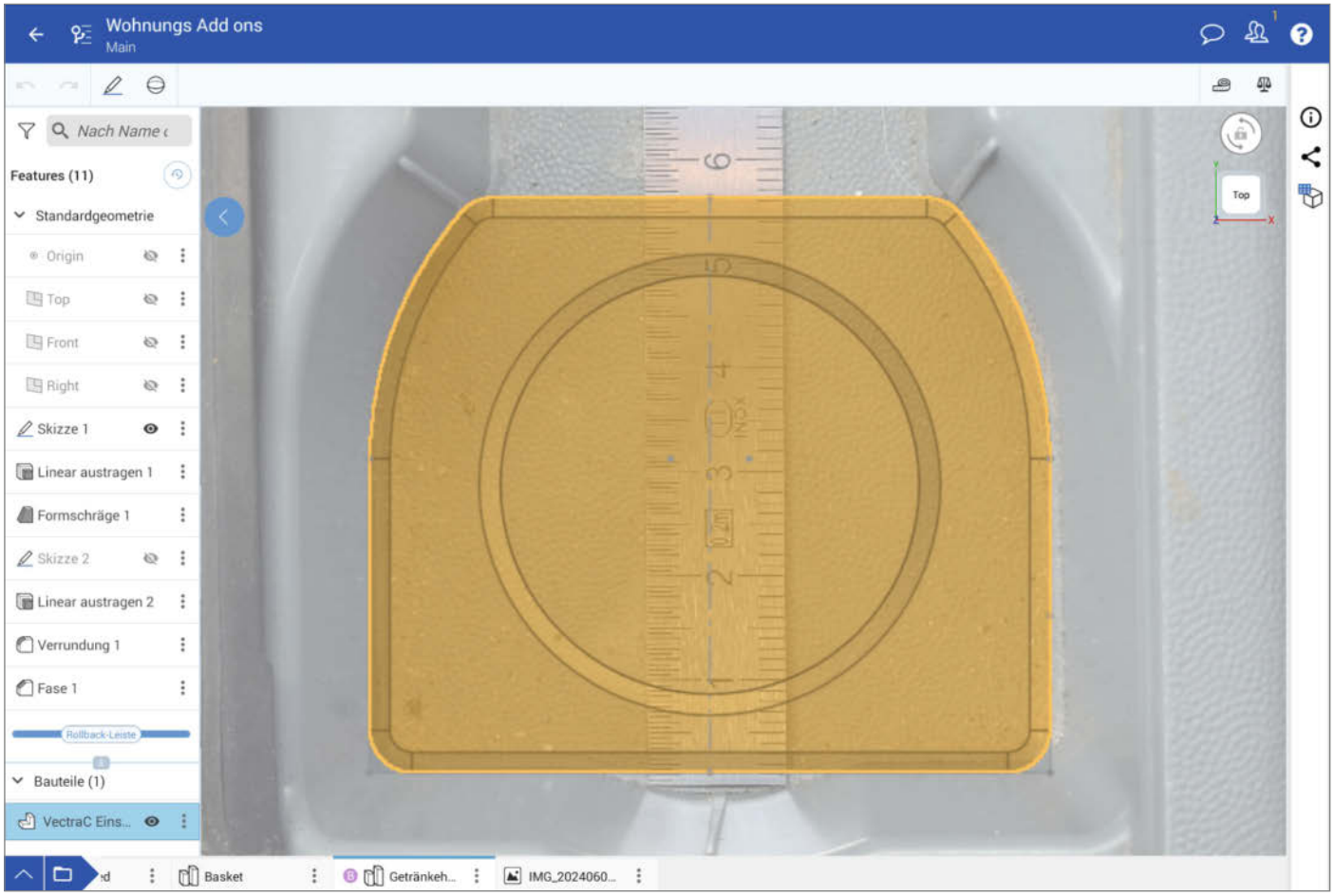


Bild 3: Gerade hat man mit dem Tablet in der Werkstatt noch was abfotografiert, schon lässt sich direkt ein Bauteil in der mobilen App darüberbauen.

Funktionen bleiben der Browser-Variante vorbehalten. Dies sind z. B. Zeichnungen, Ordner innerhalb eines Dokumentes und Variable-Studios, mit denen sich Parameter zentral steuern und auf unterschiedliche Projekte übertragen lassen.

Durch den Cloud-Zugriff nutzt man die Dienste immer in der aktuellsten Software-Version. Natürlich könnten so auch neue Bugs mit einer Aktualisierung ausgerollt werden, aber das kommt extrem selten vor und wurde bisher immer schnell gefixt. Der Cloud-Zugriff hat jedoch auch einen klaren Nachteil: den Internetzwang. Onshape ist nur mit bestehender Internetverbindung nutzbar. Eine hohe Download- und Upload-Geschwindigkeit sowie niedrige Latenzzeiten verbessern die Reaktionsfähigkeit und das Laden großer 3D-Modelle. Leider ist es bisher nicht möglich, die Software auch offline zu nutzen. Man kann bei einer schlechten Internetverbindung unterwegs (z. B. im Zug) nicht weiter modellieren und anschließend die Dateien synchronisieren.

Voraussetzungen

Obwohl man Onshape nicht installieren muss und das Programm keine Highend-Hardware erfordert, hängt die Performance trotzdem ein wenig vom Endgerät ab und damit auch, wie viel Spaß man mit komplexeren Modellen oder größeren Baugruppen hat sowie mit allem, was eben ein wenig mehr Leistung benötigt.

Neben der Internetverbindung beeinflussen vor allem die Grafikkarte, die Browser-Wahl und der Arbeitsspeicher die Performance. Onshape stellt die 3D -Modelle mittels WebGL dar. Die erforderliche Leistung hierfür liefert die Grafikkarte des Nutzers. Hat man nur einen einfachen On-Board-Grafikchip, übernimmt die CPU die Darstellung des 3D-Modells. Onshape empfiehlt für eine gute Performance, eine Low-End-Gaming-Grafikkarte zu verwenden. Aus Erfahrung kann ich sagen, dass schon sehr komplexe Modelle oder Baugruppen mit mehreren Tausend Teilen nötig sind, bis man Leistungseinbußen merkt. Das wird in gängigen Maker-Projekten nur in Ausnahmen vorkommen.

Grundsätzlich werden alle aktuellen Internetbrowser von Onshape unterstützt. Vorteile zeigen sich aber vor allem bei der Nutzung von Chrome und Firefox. Laut Onshape sei Chrome ideal für häufige Tabwechsel in Verbindung mit kleinen bis mittleren Modellen, da der Browser hier am schnellsten arbeitet. Allerdings begrenze Chrome den Zugriff auf RAM und VRAM ab einem gewissen Punkt (siehe Link in der Kurzinfo). Hier sei Firefox die bessere Wahl. Dieser limitiere die Hardware nicht und sei somit bei großen Modellen in wenigen Tabs vorzuziehen. Da wir hier über die Leistungsgrenzen sprechen, muss deshalb aber niemand mit Browser-Hopping anfangen. Ich konnte bisher mit Firefox als Standard-Browser jegliche Kleinteile mit vielen Tabs darin bearbeiten.

Mehr Informationen und Tipps zu den Systemvoraussetzungen von Onshape gibt es in der Online-Dokumentation, die man über den Link in der Kurzinfo erreicht.

Auch mobil einsetzbar

Anders als bei Fusion 360 von Autodesk ist die mobile App von Onshape kein reiner Viewer. Sie ist ebenso eine vollwertige Konstruktionssoftware und funktioniert sogar überraschend gut! Skizzen erstellen, Volumenkörper bearbeiten, alle klassischen Funktionen sind vorhanden.

Besonders die Ortsunabhängigkeit ist angenehm, wenn der Schreibtisch, an dem man sonst konstruiert, nicht in der Werkstatt steht,

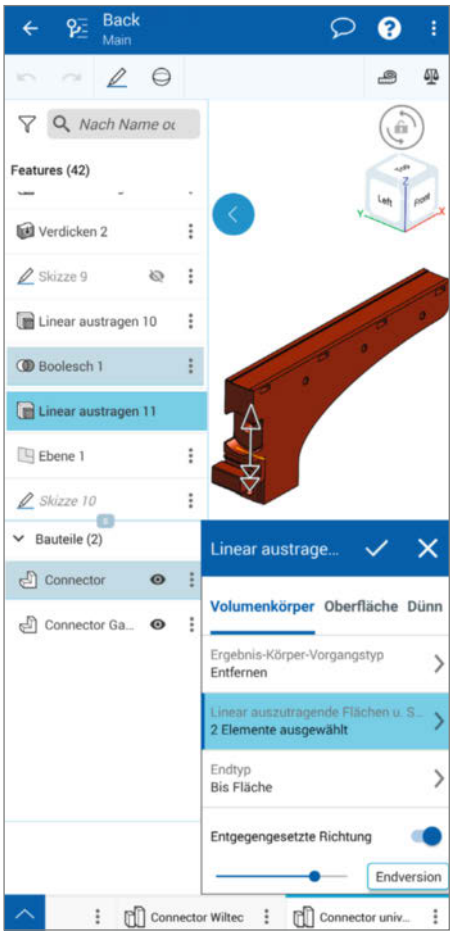


Bild 4: Der Smartphone-Bildschirm eignet sich nicht, um komplexe Modelle damit zu bauen.

in der man letztlich die konstruierten Bauteile verwenden möchte. Mit dem Smartphone oder Tablet in der Hand kann man sich nun frei in der Werkstatt bewegen und die Teile vermessen und eingeben, die man für seine nächste Konstruktion benötigt. In der Kommentarspalte notiert man sich noch eine Seriennummer oder das Werkzeug, zu dem man etwas konstruieren möchte.

Ebenso einfach lassen sich Korrekturmaßnahmen an 3D-gedruckten Teilen vornehmen. Ist die erste Version eines Prototyps gedruckt und ausprobiert, sind oft hier und da noch kleine Änderungen nötig. Wie üblich müssen Flächen um ein paar Zehntel verschoben oder Fasen ergänzt werden. Das muss man nicht notieren und später am PC machen, sondern kann die Änderung sofort in der App umsetzen. Je nach verwendetem 3D-Drucker (z. B. Bambu Lab oder Anker) kann das Modell direkt per Smartphone übertragen und die nächste Version gedruckt werden.

Grundsätzlich sind kleine, einfache Konstruktionen auf dem Mobilgerät fast so schnell erstellt wie am PC. Außerdem hat man sein Smartphone oder Tablet meist in direkter Griffweite. Man kann also überall mit einer schnellen 3D-Skizze seine Ideen festhalten und später in aller Ruhe am Rechner ausarbeiten. Gleichermaßen schnell und einfach zeigt man Bekannten und Kollegen auf dem Smartphone, woran man gerade so bastelt und kann sich Tipps einholen.

Auch der Umgang mit Fotos ist leichter. Diese lassen sich bereits seit Jahren in CAD-Programme importieren, um so Referenzen, Bezugskanten und Ähnliches abzubilden. Allerdings muss man sie normalerweise erst auf den PC übertragen. Mit der mobilen Variante von Onshape lässt sich ein Foto (idealerweise mit einem Maßstab drauf) direkt nach dem Aufnehmen in das Projekt laden und eine Skizze darüber legen (Bild 3).

Anders auf kleinem Display

Das alles mit dem Touchscreen zu steuern, gestaltet sich mit der mobilen Benutzeroberfläche größtenteils intuitiv, hat aber oft genug seine Tücken. Einfaches Antippen entspricht einem normalen Mausklick. Ebenso zoomt und dreht man Modelle mit den üblichen Wischgesten. Ein Rechtsklick entspricht dem Tippen mit zwei Fingern. Außerdem erzeugt langes Tippen und Halten mit einem Finger ein Fadenkreuz, um eine präzise Auswahl zu treffen. Das ist praktisch, wenn kleine Flächen oder Kanten dicht beieinander liegen.

Dieses Feature ist aber ungünstig, falls man das Tablet beispielsweise gerne mit einem Stift bedient. Dann muss man immer wieder mal zwischen dem Stift und einem schnellen Tippen mit zwei Fingern hin und her wechseln. Hinzu kommt, dass es schier unmöglich ist, übereinander liegende Skizzenelemente auszuwählen. Das ist bereits im Browser knifflig, aber in der mobilen Version muss man dafür meist Elemente löschen.

Abhilfe schafft manchmal das aufziehbare Auswahlfenster. Das erzeugt man mit zwei Fingern durch Drücken und Halten. Hier darf aber keine Bewegung beim Tippen erfolgen, sonst bewegt sich nur das Modell.

Mangels Tastatur muss man bei jeder Eingabe eines Zahlenwertes die Bildschirm-Tastatur einblenden, danach wieder ausblenden, beim nächsten Wert dann wieder einblenden … Das macht bei zu vielen Wiederholungen keinen Spaß und wird nur schlimmer, wenn das eingeblendete Numpad nicht ausreicht, weil man eine Variable eingeben möchte.

Zudem ist die Befehlsauswahl in ein zusätzliches Untermenü gesteckt, um Platz zu sparen. Einzelne Befehle kann man leider nicht in die obere Menüzeile packen, auch wenn diese bis auf vier Befehle komplett leer ist.

Geduld ist gefragt

Besonders mühsam ist das Erstellen von Features und einer dazugehörigen Mehrfachauswahl, etwa wenn man sein Modell zum Ende der Konstruktion mit Fasen versehen möchte. Die Mehrfachauswahl eines solchen Befehls funktioniert mobil anders als am PC. Dort klickt man einfach Kanten und Flächen an, lässt sie berechnen und falls der PC nicht gleich mitkam, generiert er nach einem kurzen Moment das gesamte Feature auf einmal.

Mobil funktioniert dies nicht. Hier ist stets abzuwarten, bis das Feature auf dem gewählten Element erstellt wurde, bevor man das nächste Element auswählt. Habe ich in einem Fasen-Befehl bereits sieben Kanten in der Auswahl und wähle die achte Kante aus, bevor die Fasen für die vorherigen Kanten generiert sind, springt die Auswahl auf die zuletzt gewählte Kante um. Das ist immer wieder ärgerlich, bremst einen aus und zeigt, dass mobil lieber doch an einfachen Modellen gearbeitet werden sollte und einige Dinge mehr Geduld erfordern

Da komplexe Konstruktionen irgendwann eine entsprechend hohe Fingerakrobatik erfordern, was mehr nervt als es hilft, halte ich lange Arbeitssessions in Onshape daher grundsätzlich lieber am PC als auf Mobilgeräten ab (Bild 4).

Einen Blick wert

Onshape bietet sowohl am PC als auch mobil über die App verschiedene Vorteile, die je nach Anwendungsfall genutzt werden können. Am PC kann man in einer vorbildlichen CAD-Umgebung komplexe Modelle erstellen und Befehle effizient nutzen.

Mit der mobilen App hingegen lassen sich jederzeit und überall schnelle Entwürfe und Konzeptzeichnungen erstellen. Ein Basisteil kann man mobil modellieren und später am PC weiter verfeinern. Kleine Änderungen lassen sich direkt nach der Fertigung eines Teils vornehmen und es ist mobil einfacher, bestehende Befehle zu ändern, als neue aufzubauen.

Notizen kann man schnell und unkompliziert hinzufügen, und das Präsentieren und Teilen von Modellen mit Freunden und Kollegen ist sehr leicht möglich. Für all das benötigt man kein modernes Highend-Smartphone. Das Endgerät darf auch etwas älter und leistungsschwächer sein. Vor allem bei moderneren Mobilgeräten, die mit Android laufen, ist die Einbindung einer Computermaus und Tastatur über Bluetooth ohne weitere Konfiguration möglich, sodass man prinzipiell ganz auf den PC verzichten könnte.

Die mobile App von Onshape unterstützt Android rückwirkend bis Android 5 Lollipop von 2014. Auf Apple-Geräten läuft die App ab iOS 16 von 2022. Das beinhaltet auch das iPhone 8 von 2017 und iPads aus diesem Veröffentlichungsjahr. *—akf*

Für Nerds und Maker

shop.heise.de/highlights2024

Zubehör und Gadgets

Oxocard Artwork Creative Coding

Lernen Sie die Grundlagen der Computeranimation mit dem ESP32-Chip. Erzeuge beeindruckende visuelle Effekte wie in Spielen und Filmen dank leistungsfähiger Hardware.

Ideal für Einsteiger!

~~69,90 €~~

39,90 €

Oxocard Science Plus GOLD Edition

Hochwertige Computerplatine mit 8 Sensoren, 16 Werten, Experimentierplatine und offener Programmierschnittstelle zur Beobachtung und Änderung der Programme.

Im praktischen Kreditkartenformat!

119,90 €

c't 3003-Hipbag/Bauchtasche

Total praktisches c't 3003-Merch.
Dieses ultimative Fashion-Statement fällt garantiert überall auf und es passt jede Menge rein.
Mit Innentasche und verstellbarem Hüftgurt.

Sieht garantiert ghyle aus!

14,90 €

Cyber Clean Professional Reinigungsmasse

High-Tech-Masse entfernt 99,99% der Keime, reinigt strukturierte Oberflächen und Zwischenräume, ohne Feuchtigkeit abzugeben. Ideal für empfindliche Oberflächen und elektronische Geräte.

Für Hygiene und Wohlbefinden!

16,90 €

Nitrokey Passkey

Schützen Sie Ihre Accounts zuverlässig gegen Phishing und Passwort-Diebstahl mit sicherem, passwortlosem Login und Zweifaktor-Authentifizierung (2FA) durch WebAuthn/FIDO2. Praktisches USB-A Mini Format für den Schlüsselbund.

Qualität made in Germany!

34,90 €

AUCH ALS USB-A/C-VERSION

Nitrokey-Secure-Bundle C/C

Der Nitrokey 3A NFC ist ein starker Security Token für mobile Geräte. Der USB-C Daten Blocker schützt vor unerwünschter Datenübertragung. Inklusive c't-Security-Checklisten als PDF.

Schutz gegen Massenüberwachung und Hacker!

64,90 €

c't Jumbotasse „Kein Backup? Kein Mitleid!“

Unsere Tasse erinnert Ihre Kollegen an regelmäßige Updates. Jetzt mit 450 ml für mehr Kaffeegenuss.
Nie wieder Stress ungesicherter Daten:
Kein Backup? Kein Mitleid!

Natürlich spülmaschinengeeignet!

17,90 €

Messbecher „Wissenschaft“

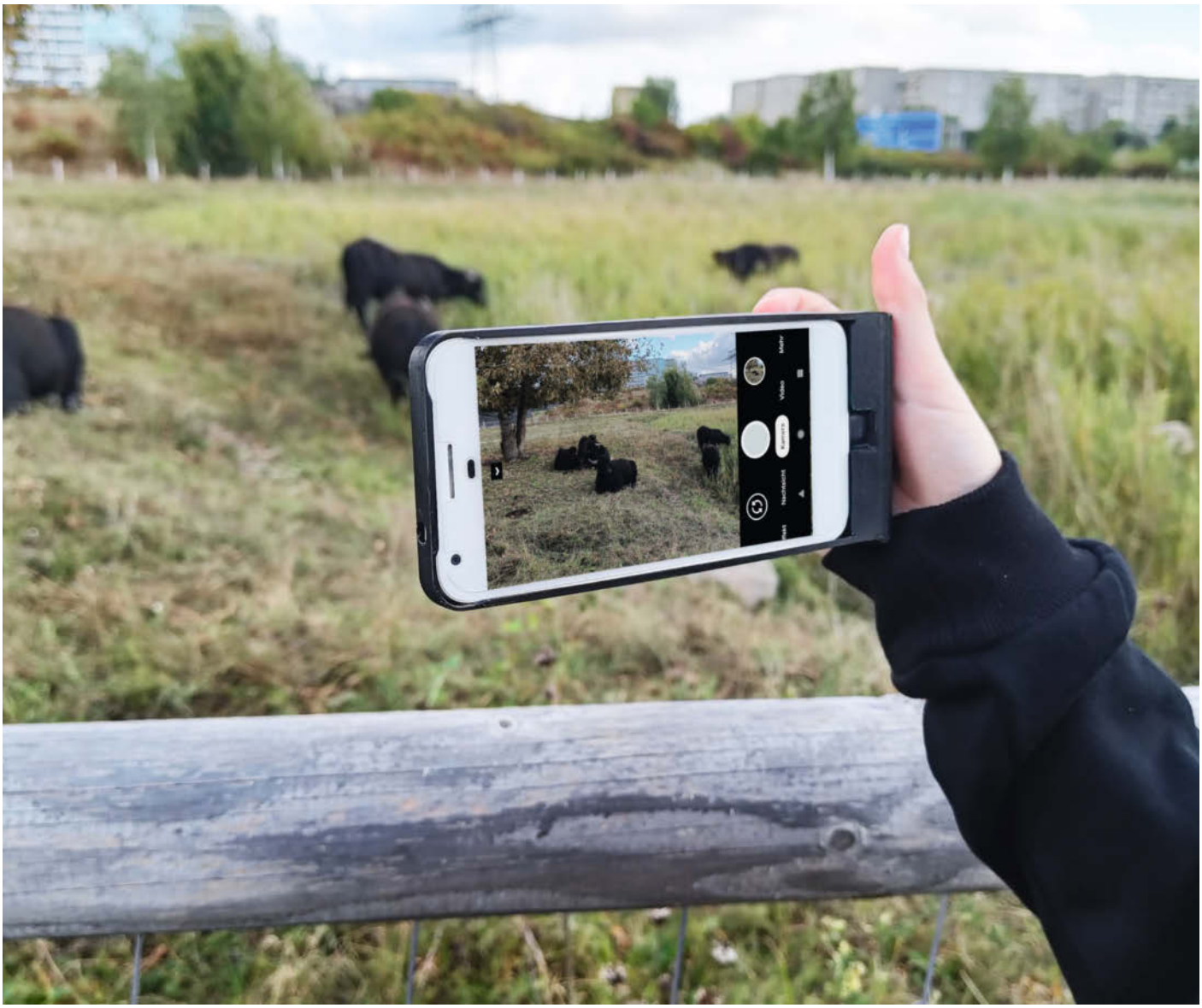
Schluss mit Langeweile in der Küche! Auf diesem Messbecher stehen 14 nerdige Fun Facts. Fragen wie „Wie viel Platz nehmen 30.000 Reiskörner ein?“ werden beantwortet.

Aus hitzebeständigem Borosilikatglas!

19,90 €

Generell portofreie Lieferung für Heise Medien- oder Maker Media Zeitschriften-Abonnenten oder ab einem Einkaufswert von 20 € (innerhalb Deutschlands). Nur solange der Vorrat reicht. Preisänderungen vorbehalten.

heise shop



Kamera-Handy

Das alte Handy hat eine tolle Kamera, aber der Akku ist mau und ein Tausch ist nicht mehr rentabel? Powerbanks sind eine Möglichkeit, das Telefon zu retten, aber unhandlich. Mit einem 3D-gedruckten Akkugriff wird das Telefon jedoch zur supersmarten Kamera.

von Carsten Wartmann

Mein altes Pixel XL hat immer noch einen gewissen Charme und die Kamera ist auch nach heutigen Standards gut für alltägliche Aufgaben geeignet. Nur der Akku ist schon sehr schwach. Der Tausch ist allerdings kaum noch ökonomisch und vom Aufwand her wenig sinnvoll: Das verklebte Display muss entfernt werden, um dann das verklebte Innenleben aus dem Gehäuse zu hebeln, um an den Akku zu kommen. Bei iFixit ist der Akkutausch mit „schwierig" gekennzeichnet. Wenn man denn überhaupt noch den passenden Akku bekommt, wird das Ganze auch recht teuer, vor allem wenn man noch Werkzeug und Verbrauchsmaterial dazu rechnet.

Eierlegende Wollmilchsau

Eine Powerbank wäre möglich, aber wer möchte die schon mitschleppen und durch das Kabel eingeschränkt sein. Besser wäre ein Ansteck-Akku, aber der belastet die USB-C-Buchse sehr.

Schaut man sich auf den Verkaufsplattformen um, so findet man natürlich Lösungen: Kameragriffe mit eingebautem Akku und Auslöser, aber auch einige Hersteller möchten das allgegenwärtige Fotografieren und Filmen mit dem Smartphone angenehmer machen und bieten solches Zubehör ab Werk an. Will man es noch professioneller, dann geht es weiter mit ganzen Rigs, die neben dem Smartphone auch Licht enthalten, oder Gimbals, die automatisch eine mechanische Bildstabilisierung bieten. Damit schleppt man aber wie zu (D)SLR-Zeiten einen großen Gerätepark mit sich herum. Da kann man dann auch eine „richtige" Kamera nehmen. Aber ein Upload der Fotos auf soziale Medien oder in die Cloud zu Hause wäre schon schön, doch natürlich siegte hier wieder meine Trägheit: Das neue Smartphone ist ja da und das alte kann stationär mit herkömmlichem Stativ und Powerbank benutzt werden, also habe ich das (Kopf)-Projekt erst mal auf Eis gelegt.

Nun interessiert sich meine Tochter (9 Jahre) immer mehr für die Foto- und Videografie und noch mehr für Smartphones und soziale Medien, in denen kleine YouTuberinnen Videos ihrer Spielzeugpferde machen – und ein Reiturlaub stand auch an. Eine Kamera musste her!

Die kommerziellen Klemmbefestigungen für Smartphones sind klobig und die Passform wird nur für moderne Geräte garantiert. Magnetische Befestigungen sind schon schlanker, aber erfordern das Ankleben einer Gegenplatte auf dem Telefon, sofern dies nicht schon ab Werk eingebaut ist.

Je besser die Passform und geringer die „Klobigkeit", desto weniger Modelle werden unterstützt. Daher waren auch die Handyhüllen mit eingebautem Akku für mich sinnlos. Eine Handyhülle modifizieren? Das könnte eine Idee sein. Aber auch hier trifft der Fluch des alten Pixel XL: Hüllen gibt es kaum noch zu kaufen. Also warum nicht gleich selbst drucken?

Also musste ich nur noch das Akkuproblem lösen. Ansteck-Akkus gibt es zuhauf, aber auch die sind an der schmalen Seite eines Telefons recht klobig. Dann fand ich einen 5000-mAh-Akku, der von der Breite passte und schon beinahe wie ein Kameragriff (Bild 1) aussieht. Als Clou ist der USB-C-Stecker klappbar.

Der Akku wurde bestellt und tatsächlich: Auch eingesteckt konnte man den Akku so weit nach vorn (zur Kameraseite) klappen, dass es nicht nur schon wie ein richtiger Kameragriff aussah, sondern auch die Displayseite komplett glatt war. Der billige BT-Fernauslöser funktionierte auch.

Damit kam wieder Fahrt in das Projekt: Ein 3D-gedrucktes Gehäuse, das das Pixel XL zu einem Fotoapparat macht.

Bild 1: Ein erster Layout-Test

Kurzinfo

» **Altes Handy mit müdem Akku retten**
» **Akku-Griff durch 3D-Druck**
» **Ohne Eingriff in die Hardware**

Checkliste

Zeitaufwand:
ein Wochenende

Kosten:
25 Euro

3D-Druck:
Druckbettgröße = Handylänge plus etwa 5 cm

Material

» **Smartphone** am besten mit USB-C-Anschluss
» **Ansteck-Akku** mit klappbarem Anschluss
» **Bluetooth-Auslöser**
» **Druckmaterial** PETG oder PLA

Werkzeug

» **Lötkolben** und Zubehör
» **Maker-Werkzeug** Schraubendreher, Cutter, Zangen
» **Kleber** geeignet für das Druckmaterial

Mehr zum Thema

» Martin Spendiff, Photon: Open-Source-Belichtungsmesser, Make 1/24, S. 78
» Lisa Ihde, Ersatzteile mit Blender konstruieren, Make 06/23, S. 116

Alles zum Artikel im Web unter
make-magazin.de/x2mb

Bild 2: Der Bumper passt endlich.

3D-Konstruktion

Alle Maße am Akku und Handy selbst zu messen und in Blender zu konstruieren, war mir in der Zeit nicht mehr möglich (die Ferienreise!). Also könnte man doch etwas Fertiges zusammenklauen, äh, ich meine herunterladen und kombinieren. Design musste vor Funktion, 3D-Druckbarkeit und Stabilität zurückstecken.

Die Basis war zuerst ein 3D-Modell des Pixel XL, was aber auch einiges an Sucharbeit erfordert hat. Oft waren die Modelle dann doch ein normales Pixel oder ein Pixel 2 XL oder scheinbar Fantasiemodelle mit Maßen und Features, die einfach nicht stimmten. Auch das beste Modell passte nicht ganz genau: keine gute Ausgangslage. Damit wurde das 3D-Modell dann nur ein Statist, um die Konstruktion im Ganzen anzuschauen.

Ähnlich war die Lage bei den 3D-druckbaren Hüllen oder „Bumpern" (Schutz für die Seiten) von Thingiverse oder Printables. Die Vielfalt reduzierte sich auf nur wenige 3D-Modelle, die ich zuerst prüfte, bevor ich Testdrucke machte. Damit wollte ich verhindern, dass sie nur mit genau dem Drucker und Material funktionieren, das der jeweilige Konstrukteur verwendet hat. Es kommt hier tatsächlich auf Zehntel Millimeter an. Die beste Hülle diente letztlich nur noch als Lieferant für den Querschnitt des Bumpers, den ich dann entlang einer Kurve extrudiert habe. Eine schützende Rückseite brauchte ich ja nicht, also habe ich diese weggelassen.

So kam ein parametrisches Modell heraus, das ich mit nur wenig Probedrucken schnell passend machen konnte. So dachte ich. Was mir etwas Augenmaß und ein Messschieber in ein paar Sekunden am echten Pixel XL hätten klarmachen können, verschwieg mir das heruntergeladene 3D-Modell des Telefons. Das Pixel XL ist auf der Seite mit der Kamera über einen Millimeter dicker. Da staunt man, wenn es in der Hülle schlackert und nach der Änderung in Blender gar nicht mehr passt. Nun musste noch ein Modifier her, der die Hülle der Länge nach auf einer Seite flacher macht.

Als Nächstes wurde die Halterung für den Akku konstruiert, zuerst als Quader und dann an den Kanten abgerundet. Dabei habe ich die Radien des Akkus mit einer Radienlehre bestimmt, was dann zu einem ausgezeichnet passenden Modell führte. Zum Glück hat der Konstrukteur ganzzahlige Radien in Millimetern benutzt.

Die Aussparungen für USB und den Anschaltknopf wurden natürlich auch gleich per Boolean-Modifier mit bedacht. Dies macht es möglich, die schwierig zu messenden Positionen und die Maße der Aussparungen schnell zu ändern. Ich habe bei alledem versucht, einen Druck ohne Stützen zu ermöglichen. Wo es doch zu haarig wurde, habe ich selbst Stützen hineinkonstruiert.

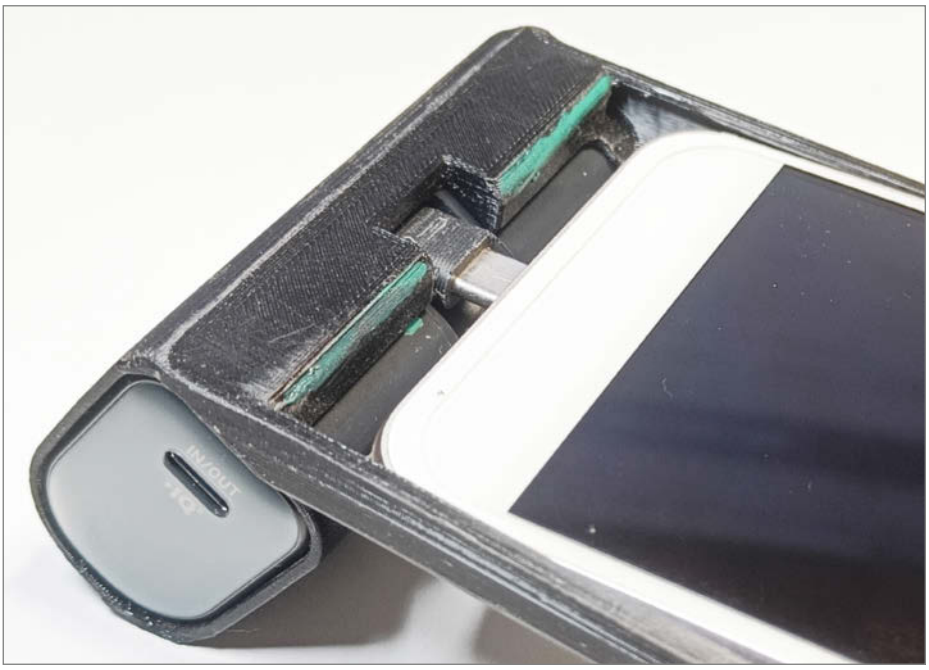


Bild 3: Anprobe: Problematische Bereiche wurden grün markiert.

Druck und Fehldruck

Damit waren dann zwei Teile konstruiert und ich konnte den Rahmen (Bumper) mit dem Akkufach vereinen. Ich habe alles in PETG gedruckt, dies ist etwas flexibler als PLA und bei guten Druckparametern hat es eine so gute Layer-Haftung, dass es eher quer zu den Layern bricht. Ich verwende auf meinem alten Prusa Mk2 240 °C bei 60 °C Betttemperatur. Der Bauteillüfter läuft mit nur mit 30 Prozent Leistung, außer bei Brücken. Selbst die schmalen Kanten des Bumpers halten gut auf dem glatten PEI-Druckbett. Um sicherzugehen, dass sich nichts löst, kann man im Slicer zusätzlich einen kleinen Brim (engl. für: Rand, Krempe) verwenden. Beim Entfernen des Drucks muss man schon etwas vorsichtig sein und das Bett am besten nicht zu lange abkühlen lassen.

Die erste „Anprobe" passte fast sofort, nur die Aussparung für den USB-C-Port war zu klein und wurde erst mal mit einer Zange und einem Cutter „angepasst". Den so durch Zerstörung ermittelten Wert habe ich dann in das Modell übertragen. Ein erster Test (Bild 5) zeigte eine gute Handhabung; das Telefon liegt durch den Griff gut in der (Kinder-)Hand, wenn auch noch der Auslöser fehlt.

Dann ging es daran, den Auslöser in ein kleines Gehäuse zu packen. An sich kein Pro-

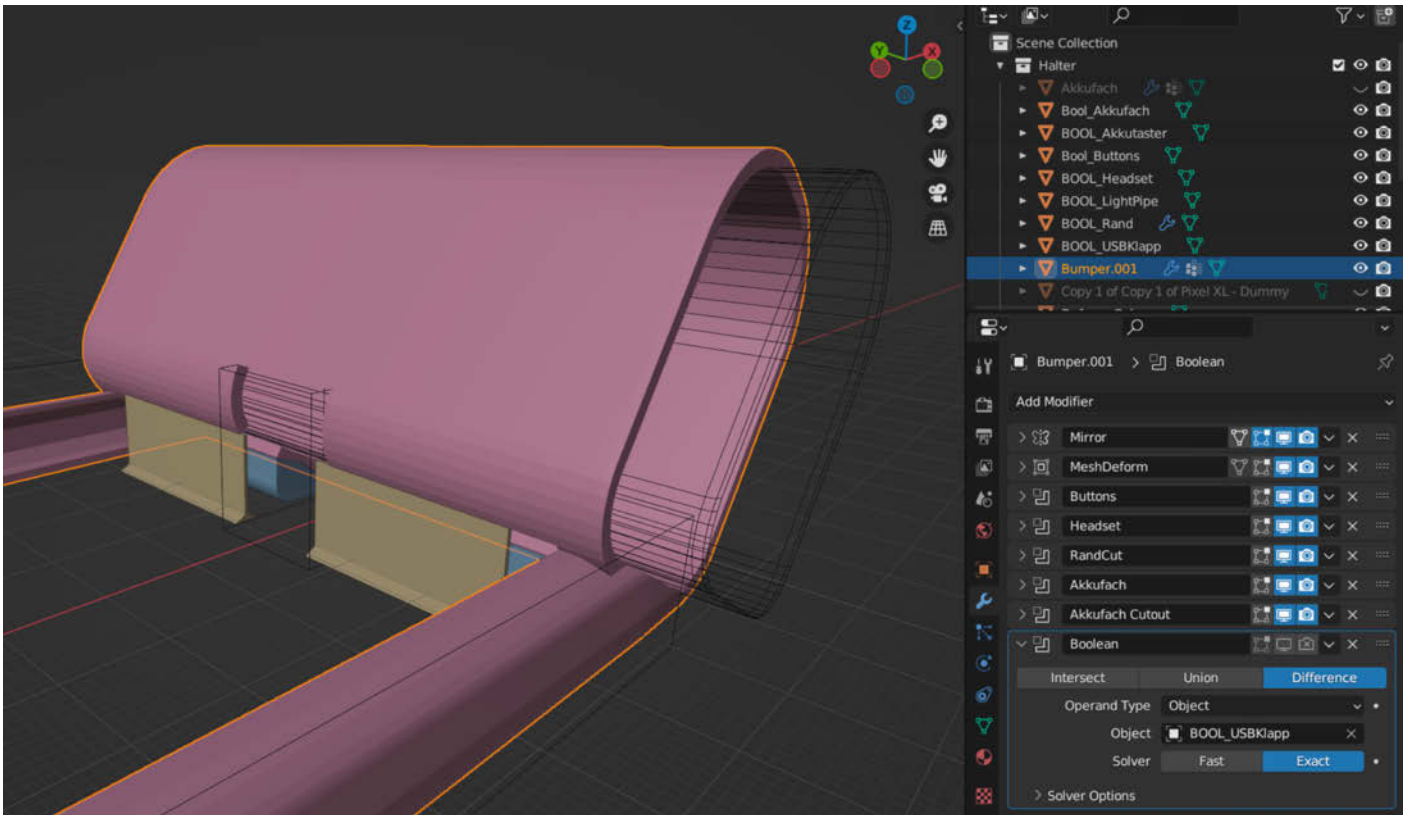


Bild 4: Die Stützstruktur in Beige und Boolean-Modifier für die Hülle

blem, mir saß jetzt aber schon die Zeit im Nacken und so habe ich nach einigen Versuchen und Fehldrucken (durch starke Überhänge) den Auslöser separat mit getrenntem Boden und anschraubbarem Deckel konstruiert. Innen sind nur zwei lange Nasen, in welche die Platine eingesteckt wird. Den Boden habe ich dann mit PETG-Kleber angeklebt und den kompletten Aufbau schließlich an den Akkukasten. Nicht perfekt, aber gut genug.

Optimierungen

Wenn man die Radien unten an der Akkuseite und die Aussparungen für den Stecker richtig hinbekommt, schwingt das Telefon praktisch ohne Widerstand in den Bumper hinein und rastet ein. Dies hat allerdings ein paar Drucke gebraucht, weil die Lage des Drehpunkts vom USB-Stecker nur schwer messbar ist und auch mit der Position des Akkus relativ zum Bumper variiert.

Der Auslöser könnte unten noch eine Schräge oder Rundung vertragen, da die ge-

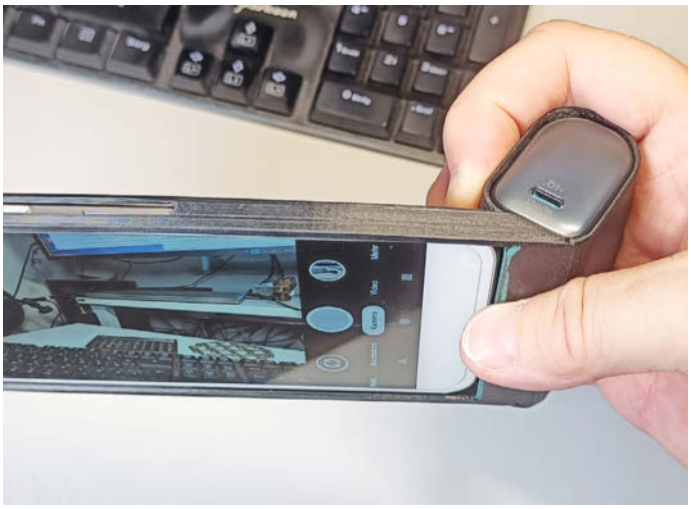


Bild 5: Ohne den Auslöser liegt es noch nicht perfekt in der Hand.

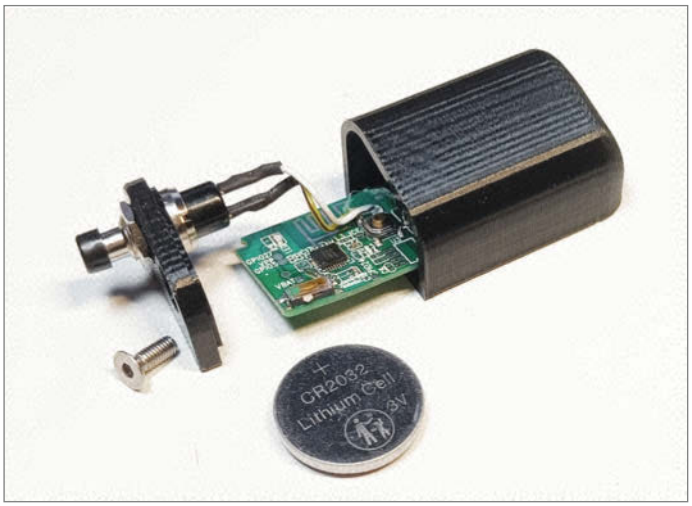


Bild 6: Montage der Auslöserbox

rade Kante nicht optimal für den Ringfinger ist, der dort unterstützend und stabilisierend zum Liegen kommt, wenn man das Gerät in der Hand hat.

Natürlich habe ich eine Aussparung am Akkugehäuse vorgesehen, durch die man die Ladestandanzeige des Akkus sieht. Sie besteht aus vier weißen und winzigen punktförmigen LEDs, die man tagsüber nur schwer sieht. Ich bin ja ein Fan von Lichtleitern und habe klares Filament (siehe „Mehr zum Thema“) schon für Ähnliches verwendet. So habe ich dann auf den Durchbruch zur Ladestandanzeige ein Stück klares Filament geklebt. Durch den Linseneffekt werden die punktförmigen LEDs zu Strichen abgebildet, die cool aussehen und gut abzulesen sind. Stark gebogenes Filament bekommt man übrigens mit etwas Kreppklebeband auf Papier fixiert und mit einer Heißluftpistole schön gerade.

Elektronik des Auslösers

Den Auslöser öffnet man vorsichtig mit einem Spatel (oder Plektrum), er ist nur geklipst. Wenn man vorsichtig arbeitet, geht nichts kaputt.

Den Auslöse-Taster habe ich mit einem robusten Drucktaster überbrückt, zwei kurze Kabel schaffen etwas Raum für den Einbau. Am besten misst man mit einem Multimeter und Durchgangstest, welche der vier Kontakte am SMD-Taster die richtigen sind.

Der Batteriehalter hält nun die CR2032-Lithium-Batterie nicht mehr selbst. Man muss sie also in die Kontakte drücken und dann zusammen mit der Platine in das Gehäuse, hinter die Nasen schieben. Dann schließt man den Deckel und schraubt ihn fest. Ich habe mir nicht die Mühe gemacht, ein Gewinde zu scheiden; die M3-Senkkopfschraube schneidet sich selbst einen Weg. Nicht vergessen, den BT-Auslöser vorher einzuschalten!

Die Batterie hält bisher gut, ohne dass ich einen Weg vorgesehen hätte, den Auslöser auszuschalten. Er geht wohl nach einer Weile in den Tiefschlaf und wacht bei einem Tastendruck wieder auf.

Hat das verwendete Handy einen Headset-Anschluss (TRRS, vierpolig), so gibt es übrigens auch die Möglichkeit, mit einem Taster, in Reihe mit einem 220-Ohm-Widerstand, die Pins GND und AUX zu überbrücken, um auszulösen. Es gibt zwei Varianten dieser Stecker, bei denen GND und AUX vertauscht sind, es sind aber immer die beiden Kontaktringe, die am weitesten von der Spitze entfernt sind.

Erweiterungen

Eine Sache, die ich gern noch einmal angehen würde, ist ein Stativgewinde zu integrieren. Diese sind Grobgewinde mit 1/4"- 20 UNC oder nach DIN 4503. Man kann sie drucken, aber ich würde auch ein älteres Smartphone ungern den wenigen Layern Plastik anvertrauen. Man kann aber einfach passende Muttern einkleben oder in einer Druckpause einlegen und dann „umdrucken“. Da der Bumper-Rahmen nicht sehr viel Material bietet, wäre der wohl beste Platz für so einen Stativanschluss der Akkugriff selbst.

Was wir im „Feld“ noch bemerkt haben: Der Griff trägt etwas auf, allerdings kann man das Foto-Handy in die Hosentasche stecken (Bild 9) und schnell und sicher am Griff ziehen, um ein Foto zu machen. Und in der linken Hand gehalten, kann man für soziale Medien wunderbar aufrechte Fotos und Videos machen.

Weitere Ideen wären eine Aufnahme für Zubehör („Blitzschuh“) für ein LED-Licht oder ein Mikrofon. Aber damit ist man dann schon wieder dabei, ein richtiges Kamera-Rig zu bauen, für das man wahrscheinlich besser andere Materialien verwendet.

Die Druckbarkeit wird ebenfalls durch jede Erweiterung schlechter. Man muss also entweder mit Support arbeiten oder die Teile einzeln drucken und dann mit Schrauben oder doch mit Kleber montieren. Zeit für ein „richtiges“ CAD? Vielleicht, wenn nur das Zeitproblem nicht wäre. —*caw*

Bild 7: Einlegen des Telefons

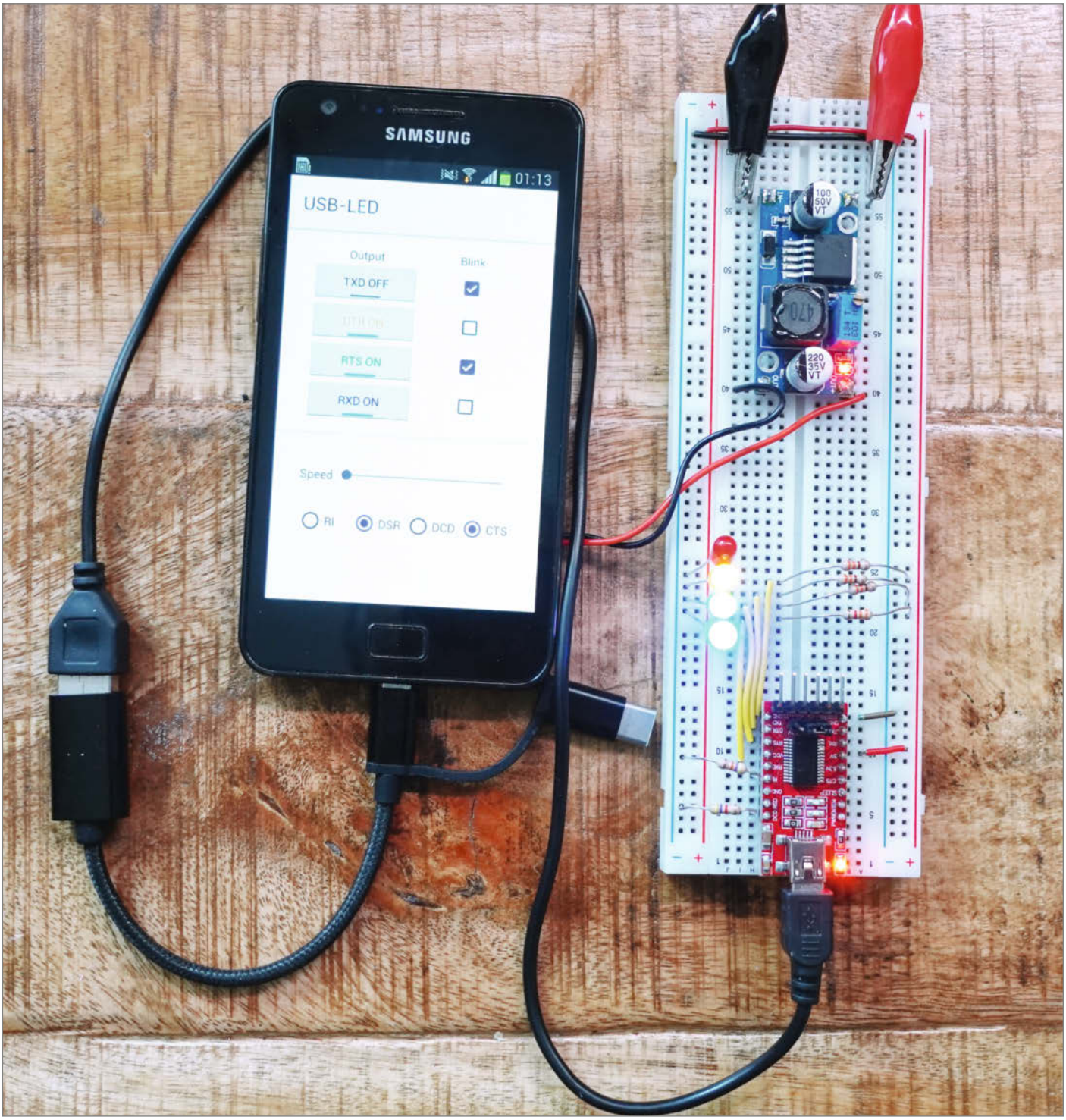
Bild 8: Filament-Linse in Aktion

Bild 9: Schnell mal ein Foto schießen

Ausgediente Smartphones als Arduino-Ersatz

Mithilfe eines FTDI-Moduls kann man ein Smartphone wie einen Mikrocontroller benutzen und damit Elektronikprojekte umsetzen.

von Michael Linsenmeier



Smartphones und Tablets bestehen aus wertvollen Bauteilen, für die man viel Geld bezahlt. Dazu gehören beispielsweise ein hochauflösendes Touchdisplay, Kameras, ein Lithium-Ionen-Akku, zahlreiche Sensoren, ein Mikrofon und ein Lautsprecher sowie Module für Bluetooth, WLAN und mobiles Internet. Dazu kommt ein leistungsfähiger und energiesparender ARM-Mikroprozessor, der es spielend mit jedem Raspberry Pi aufnehmen kann. Kurz gesagt: Wieso ersetzt man den teuren, neuen Raspberry Pi nicht durch ein bereits gekauftes Handy?

Nun haben Smartphones keine digitalen I/Os oder SPI- oder I²C-Schnittstellen, wie sie für Elektronikbasteleien benötigt werden. Mit etwas Geschick könnte man zwar das Gehäuse öffnen und die Anschlüsse für die Steuerungstasten oder Benachrichtigungs-LEDs herausführen und als I/Os nutzen. Von diesem Vorgehen rate ich jedoch ab, denn durch den äußerst kompakten Aufbau sind deren Kontakte entweder gar nicht oder nur sehr schwer erreichbar. Beim Versuch können schon kleinste Fehler und Missgeschicke das Gerät endgültig zu Elektroschrott machen (was ich selbst schmerzlich erfahren musste). Ich werde im folgenden Artikel Wege aufzeigen, wie man sein Smartphone zerstörungsfrei für Elektronikbasteleien verwenden kann. Grundsätzlich gibt es dazu zwei Ansätze: drahtlos und kabelgebunden.

Drahtlos (WLAN oder Bluetooth)

Moderne Mikrocontroller wie der ESP32 haben integriertes WLAN und Bluetooth. Das Smartphone greift über WLAN auf eine einfache Webseite zu, die im Flash-Speicher des ESP32 oder einer SD-Karte gehostet ist. Mit HTML und CSS können Entwickler eine breite Palette an Bedienelementen wie Schaltflächen, Schieberegler, Anzeigen und mehr erstellen, wobei der Kreativität kaum Grenzen gesetzt sind. Allerdings hängt die korrekte Darstellung dieser Elemente entscheidend von der Version des verwendeten Webbrowsers ab – nur wenn der Browser die entsprechende HTML- und CSS-Spezifikation vollständig unterstützt, werden die Bedienelemente wie vorgesehen angezeigt.

Sollte ein älterer Browser eingesetzt werden, der die neuesten Webstandards nicht

Kurzinfo

» **Elektronikprojekte mit dem Smartphone**
» **Selbst programmierte App per WLAN übertragen**
» **Akkuschacht über Netzteil mit Strom versorgen**

Checkliste

Zeitaufwand:
3 Stunden

Kosten:
10 Euro

Material

» **Altes Android-Smartphone oder -Tablet**
» **USB-Seriell-Adapter FT232R**
» **Bauteile zum Experimentieren**
z. B. LEDs, Widerstände und Kabel
» **USB-OTG-Adapter**
» **Breadboard**

Mehr zum Thema

» Bernd Heisterkamp, Smarte Werkstattboxen, Make 5/23, S. 38
» Michael Scheuerl, Smartphone-Roboter für den Unterricht, Make 2/22, S. 62

Alles zum Artikel im Web unter
make-magazin.de/x693

Gefahr
bei fest verbauten Akkus

Fest eingebaute Akkus müssen von der externen Versorgungsspannung getrennt werden, um Zerstörung und Brände durch Überladen zu vermeiden.

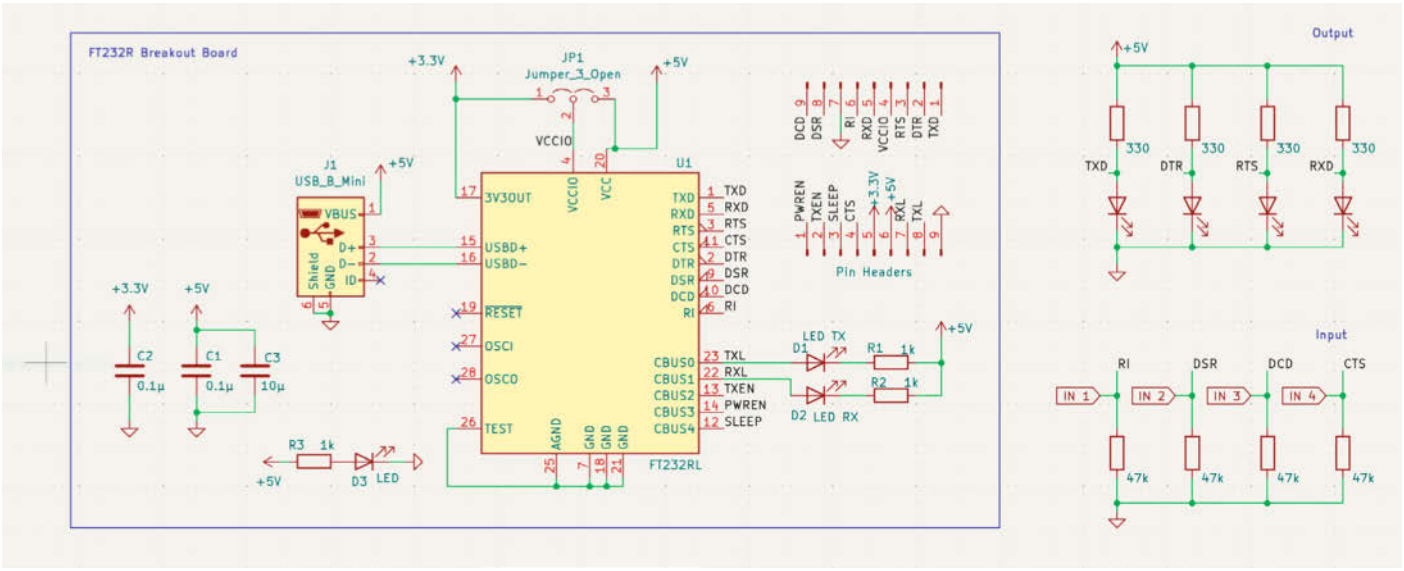


Abbildung 1: Beschaltung des Beispiels mit LEDs an den Ausgängen

Open-Collector-Schaltung

Bei einer Open-Collector-Schaltung überbrückt der Transistor in einem Pin die LED, wenn er leitet, und die LED geht folglich aus. Wenn der Transistor sperrt, geht sie wieder an. Alternativ kann man die LED auch zwischen den Pin und 5 V schalten. Sie würde sich dann genau umgekehrt verhalten: Sie leuchtet, wenn der Transistor leitet, und erlischt, sobald der Transistor sperrt.

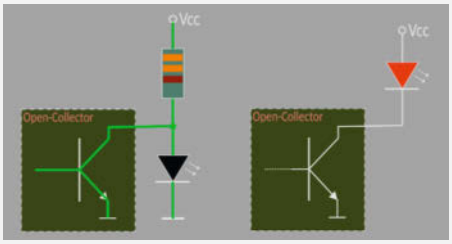


Abbildung 2: LED-Schaltung

vollständig umsetzt, lohnt es sich, zu prüfen, ob für das vorhandene Betriebssystem (sei es Android oder iOS) eine aktuellere Browserversion verfügbar ist. Ansonsten kann die Webseite natürlich auch in einer älteren Spezifikation dieser Sprachen geschrieben werden.

Im Falle eines ESP32 sendet man mithilfe von JavaScript die Benutzereingaben in JSON-Strings verpackt und über das Websocket-Protokoll an den Mikrocontroller. Dort werden die Daten im Sketch entpackt und dienen anschließend dazu, die Anwendung zu steuern. Dementsprechend schickt der Mikrocontroller seine Daten zur Darstellung an das Handy zurück. Dabei kann der ESP32 selbst als Accesspoint konfiguriert oder als WLAN-Client in einem bereits bestehenden WLAN-Netz angemeldet sein.

Diese Option der Handy-Wiederverwertung wird in diesem Heft auf Seite 24 für den Artikel „Handy als Steuerung für Bandmaschine“ verwendet.

Dieses Vorgehen hat Vor- und Nachteile.

Vorteile:

– Funktioniert unabhängig vom mobilen Betriebssystem. Lediglich die verwendete HTML-, CSS- und JavaScript-Spezifikation muss mit dem Browser kompatibel sein.
– Leicht zu implementieren. Es gibt im Internet zahlreiche Programmieranleitungen und Beispiele. Mit ESP Home und Tasmota stehen sogar fertige Frameworks zur Verfügung, mit denen webbasierte Anwendungen für die Heimautomatisierung ohne Programmierkenntnisse erstellt werden können.
– Gleichzeitige Bedienung durch verschiedene Geräte ist möglich.

Nachteile:

– Durch aktiviertes WLAN oder Bluetooth verbraucht das Gerät mehr Strom.
– Je nachdem, wie verseucht die Umgebung durch andere WLANs ist, können störende Verzögerungen auftreten.

Kabelgebunden über USB

Probleme, die bei der Funkübertragung auftreten können, lassen sich durch eine kabelgebundene Übertragung vermeiden. Seit der im Jahr 2011 erschienenen Version 3.0 (Honeycomb) unterstützt Android den USB-Host-Modus. Das bedeutet, dass man – wie bei einem PC – USB-Geräte wie Tastaturen, Mäuse usw. anschließen kann.

Wenn man also elektronische Schaltungen mit einer USB-Schnittstelle ausstattet, kann man sie über eine Smartphone-App bedienen. Dazu kann man die Schaltungen ganz einfach mit einem FTDI-Board ergänzen, das wiederum über ein OTG-Adapterkabel (On-The-Go) mit dem USB-Port des Handys verbunden wird. FTDI steht für „Future Technology Devices Limited“ und bezeichnet ein schottisches Halbleiterunternehmen, das sich auf die Herstellung von USB/Seriell-Wandler-Chips spezialisiert hat. Ihre Produkte finden sich in zahlreichen Consumer-Produkten mit USB-Schnittstelle, aber auch in manchen Entwicklungsboards für Mikrocontroller.

Zahlreiche Hersteller aus Fernost bieten für wenig Geld Breakout-Boards für den FT232R an, mit denen man selbst auf einer einfachen Lochrasterplatine oder einem Breadboard eine vollwertige USB-Schnittstelle realisieren kann. Im Folgenden beschreibe ich eine Beispiel-Applikation, mit der die Pins eines solchen Boards gelesen und geschrieben werden können.

Der Bitbang-Modus des FT232

Am FT232 sind neben den für die eigentliche Datenübertragung vorgesehenen Leitungen (Rx und Tx) auch sechs Modem-Kontrollsignale (DTR, RTS, RI, DSR, DCD und CTS) herausgeführt. Im sogenannten Bitbang-Modus kann jede dieser acht Leitungen unabhängig von den anderen als digitaler Eingang oder Ausgang genutzt werden.

In der Beispiel-App im Aufmacher kann man die LEDs mit den vier Schaltflächen ein- und ausschalten. Mit den Kontrollkästchen rechts davon kann man sie blinken lassen. Die restlichen vier Steuersignale sind als Eingänge konfiguriert. Ihre Pegel werden von den Radio-Buttons in der unteren Reihe angezeigt.

Ein Ausgangs-Port des FT232 im Bitbang-Modus ist als Open-Collector-Port ausgeführt. Um ihn als Ausgang nutzen zu können, muss er mit einem Widerstand gegen 5 V (oder 3,3 V je nach Stellung des Jumpers auf dem FT232-Board) geschaltet werden. Die 330-Ω-Widerstände dienen gleichzeitig als Vorwiderstände für die LEDs.

Ports, die man als Eingänge verwenden will, legt man mit Widerständen mit Werten zwischen 10 und 47 kΩ an Masse.

Programmierung der Android-App

Apps für das Android-Betriebssystem werden in Java oder Kotlin geschrieben. Die Programmierumgebung Android Studio kann man kostenlos von Googles Android-Entwicklersei-

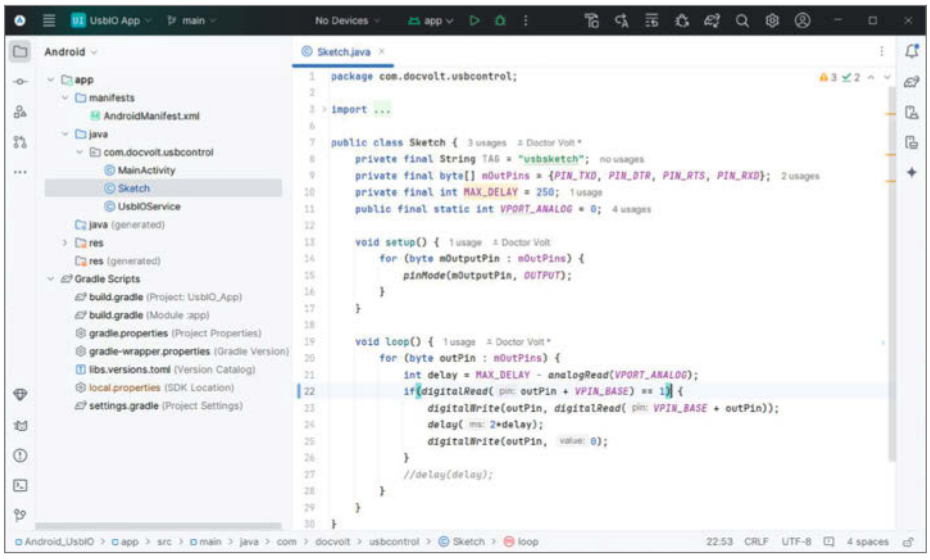


Abbildung 3: Arduino-Sketch für Android in der Android Studio IDE

ten herunterladen. Die Beispiel-App besteht aus drei Java-Quelldateien:

MainActivity.java

In der `MainActivity` ist die Interaktion mit dem Benutzer der App definiert (User-Interface oder UI). Drückt man auf eine der vier Schaltflächen, veranlasst das UI, dass der zugeordnete Ausgang gesetzt wird und die LED leuchtet. Die Kontrollkästchen lassen die LEDs blinken. Der Schieberegler darunter legt die Blinkfrequenz fest. So ändern sich die vier Radio-Buttons in der unteren Reihe, wenn einer der 47-kΩ-Widerstände gesteckt oder herausgezogen wird.

UsbIOService.java

Das User-Interface hat keinen direkten Zugriff auf die USB-Schnittstelle des Android-Gerätes. Dieser geschieht in einem Hintergrundservice, der zyklisch mit der höchst möglichen Geschwindigkeit die digitalen Eingänge des FT232 ausliest. Falls diese sich zwischen zwei Durchläufen ändern, wird die MainActivity darüber benachrichtigt, damit sie dies durch ihre UI-Elemente entsprechend anzeigen kann. Natürlich setzt der `UsbIOService` bei Anforderung auch die digitalen Ausgänge.

Sketch.java

Hier findet man die eigentliche Applikationslogik. Wer schon Erfahrung im Erstellen von Arduino-Sketches hat, wird sich schnell zurechtfinden: Die `setup()`-Funktion wird vom `UsbIOService` einmalig beim Starten der App aufgerufen. Mittels der `pinMode()`-Funktion werden die vier Ausgangs-Ports konfiguriert – per Voreinstellung sind die Ports Eingänge; braucht man nur Eingänge, muss man nichts neu konfigurieren. Der `UsbIOService` ruft zyklisch die `loop()`-Funktion auf. Die klassischen Befehle aus dem Arduino-Framework lassen hier die LEDs blinken.

Softwarearchitektur

Die Abbildung verdeutlicht das Zusammenwirken der drei Komponenten. Die virtuellen Pins dienen dem Austausch von Daten zwischen dem UI und dem Arduino-Sketch. Es gibt insgesamt 24 davon. Auf die acht physikalischen Ports des FTDI haben sowohl das UI als auch der Sketch Zugriff. Beim USB-IO-Service handelt es sich um eine einzelne Java-Quelltextdatei, die im Quellverzeichnis der App liegt.

Stromversorgung

Das FT232-Board und weitere Elektronik werden vom Android-Gerät über dessen USB-Port mit Strom versorgt. Das geht allerdings auf Kosten der Akkulaufzeit, weil der eingebaute Akku zusätzlich belastet wird. Das ist ein großer Nachteil für alle Anwendungen, die längere Zeit laufen sollen. Man braucht also eine Lösung dafür, wie man das Handy oder Tablet selbst mit Strom versorgt. Hier denkt man natürlich zuerst an dessen USB-Anschluss. Aber der USB-Port im OTG-Modus ist selbst die Stromquelle für das angeschlossene Gerät und kann daher nicht zum gleichzeitigen Laden genutzt werden. Manche Genies haben es nach dem Rooten und Hacken der Firmware wohl irgendwie hinbekommen, der Aufwand würde aber den Rahmen komplett sprengen.

Bei einigen Modellen mit einem externen Ladeanschluss oder der Möglichkeit zum berührungslosen Laden ist die Lösung durch die Nutzung dieser Optionen sehr einfach.

Bei Geräten ohne diese Möglichkeiten muss man ein bisschen tricksen und die Stromanschlüsse des Akkus herausführen. Bei alten Geräten ist das noch sehr leicht möglich, weil dort die Akkus oft nicht fest verbaut sind. Die Abbildung zeigt ein Samsung Galaxy S2. Den Akku habe ich mit einer Streifenrasterplatine ersetzt (Abbildung 5), die ich zuvor so zurechtgeschnitten habe, dass sie genau in die Aussparung passt. An der Stelle der Federkontakte sorgen Lötzinntropfen für guten Kontakt. Die Pins für die Batteriespannung und Polung sind auf dem Akku aufgedruckt und konnten daher leicht identifiziert werden.

Damit kann das Smartphone mit einer externen Stromversorgung zwischen 3 und 4,2 Volt betrieben werden. Das entspricht dem Spannungsbereich eines Lithium-Ionen-Akkus. Es ist zwar denkbar, dass das Gerät auch 5 Volt an den Batterieklemmen verträgt, aber ich wollte es nicht darauf ankommen lassen.

Das hier verwendete Samsung Galaxy S2 kann sich kurzzeitig Ströme bis zu 1 Ampere genehmigen, hinzu kommt der Bedarf der über OTG versorgten Elektronik. Das muss man bei der Auswahl der Stromversorgung berücksichtigen. Ein DC/DC-Wandlermodul wie das im Bild gezeigte HW-411 erfüllt diesen Zweck sehr gut.

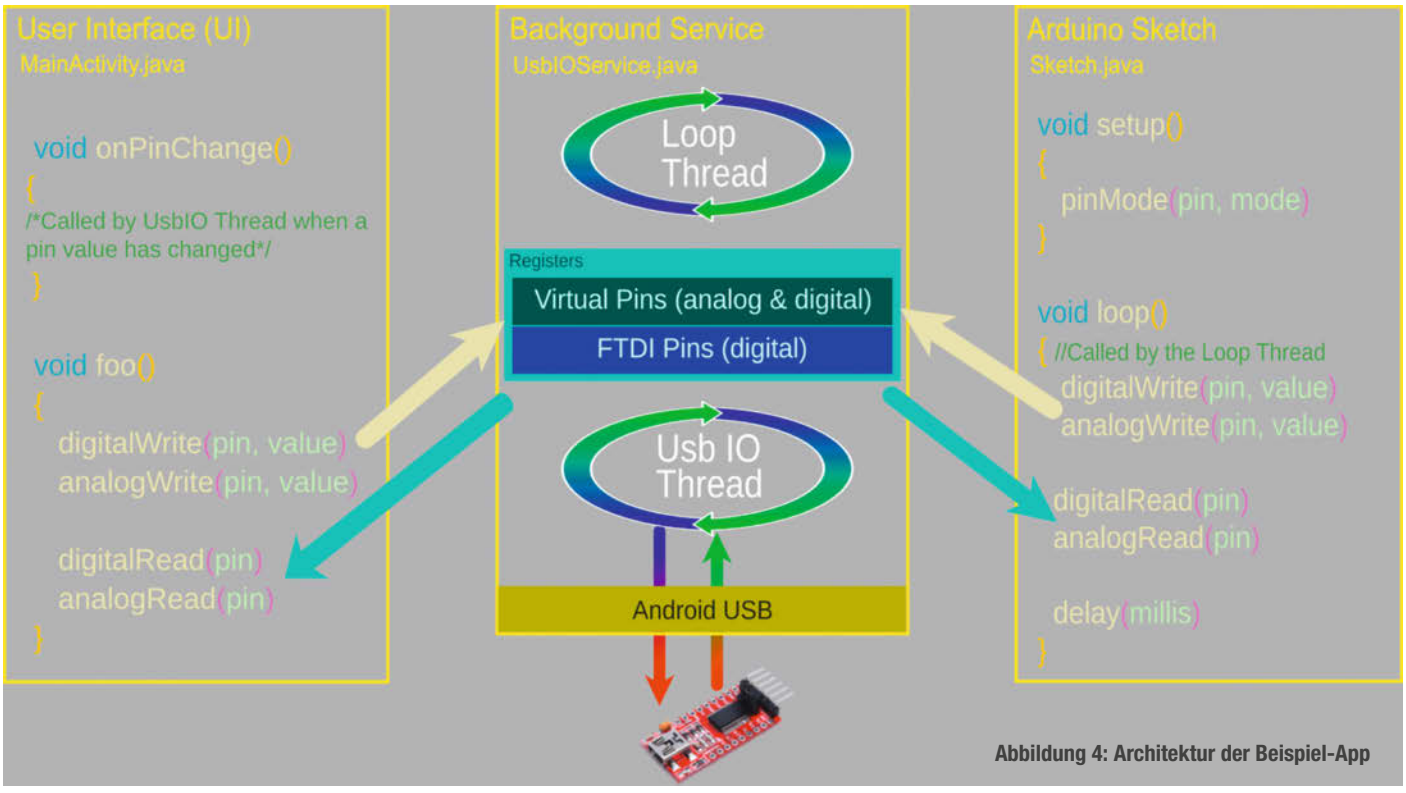


Abbildung 4: Architektur der Beispiel-App

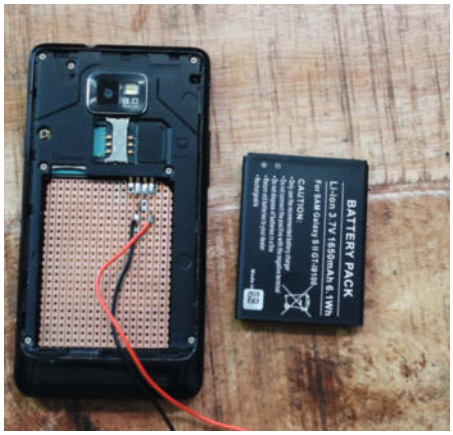


Abbildung 5: Modifiziertes Samsung Galaxy S2 für externe Stromversorgung.

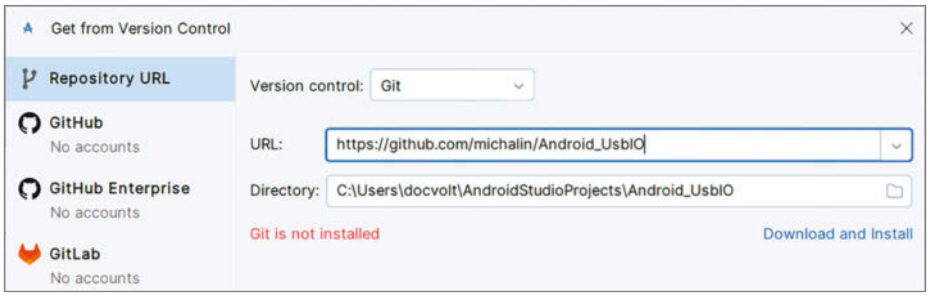


Abbildung 6: Falls der Hinweis „Git is not installed“ zu sehen ist, wird der Link „Download and Install“ genutzt, um es zu installieren. Dann klickt man auf „Clone“.

Erste Schritte in Android Studio

Zunächst ist Android Studio von der verlinkten Website in der Kurzinfo herunterzuladen. Die folgenden Schritte beziehen sich auf die Version 2024.1.2 (Koala). Beim ersten Start des Programms startet der Android Studio Setup Wizard.

Klickt man auf „Next“, wird man zur Installation des Android-SDK aufgefordert. Es enthält alle Tools und Bibliotheken, die zum Kompilieren von Android-Apps nötig sind. Nach dem Akzeptieren der Nutzungsbedingungen beginnt der Download (ca. 600 MB). Das kann je nach Internetverbindung eine Weile dauern.

Am Ende klickt man auf „Get from VCS“ und gibt die folgende URL ein: `https://github.com/michalin/Android_UsbIO` (siehe Abbildung 6; diese URL ist auch in der Kurzinfo zu finden).

Nun werden die Quelldateien für die Beispiel-App heruntergeladen.

Es dauert einige Minuten, bis das Buildsystem Gradle das Projekt erzeugt hat, aber wenn alles erfolgreich war, kann man bereits einen kurzen Blick auf das Beispielprojekt aus dem Repository werfen.

Im linken Verzeichnisbaum findet man die drei oben beschriebenen Java-Quelldateien. Die Anordnung der Elemente der Benutzeroberfläche ist in einer XML-Datei beschrieben. Android Studio stellt diese als Vorschau dar, in der die Bedienelemente mit der Maus angeordnet werden können (siehe Abbildung 7). Rechts davon werden deren Eigenschaften (Beschriftungen, Farben, Größe, Abstände zu benachbarten Elementen etc.) festgelegt.

App auf das Android-Gerät laden

Damit die App auf das Smartphone oder Tablet geladen werden kann, muss zuerst USB-Debugging beim Android-Gerät aktiviert werden. Dazu öffnet man die Einstellungen und scrollt ganz nach unten. Dort öffnet man die Entwickleroptionen, schaltet diese ein und aktiviert „USB-Debugging“ (Abbildung 8).

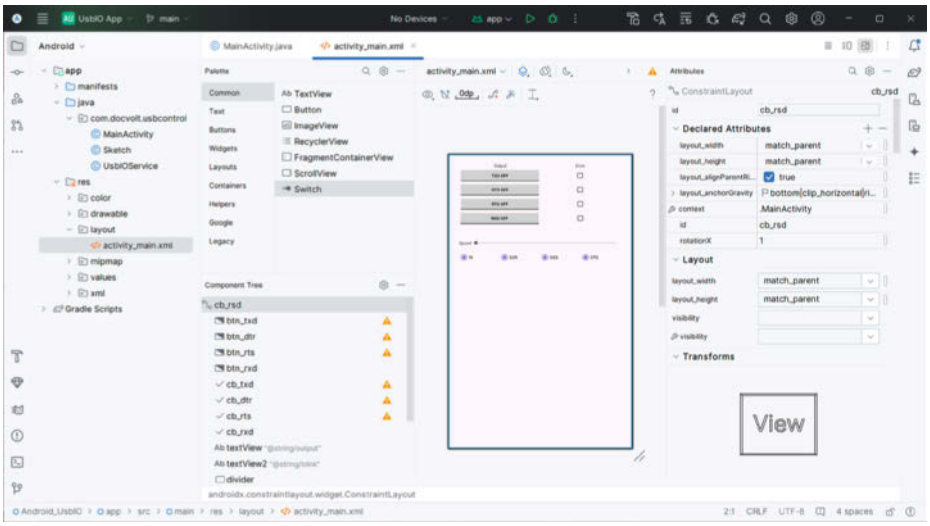


Abbildung 7: Die grafische Oberfläche der App kann in einem WYSIWYG-Editor bearbeitet werden.

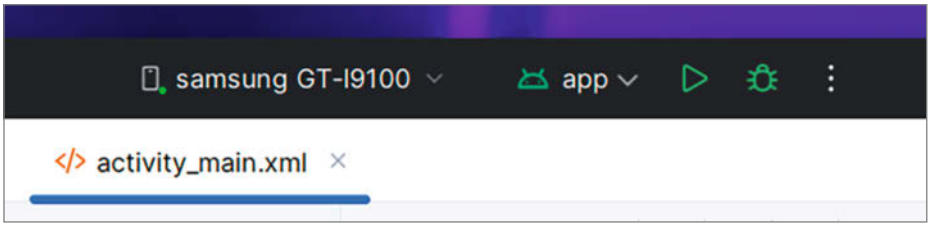


Abbildung 9: Ein Klick auf das grüne Dreieck lädt die App auf das Zielgerät, ein Klick auf den Käfer rechts davon startet den Debugger.

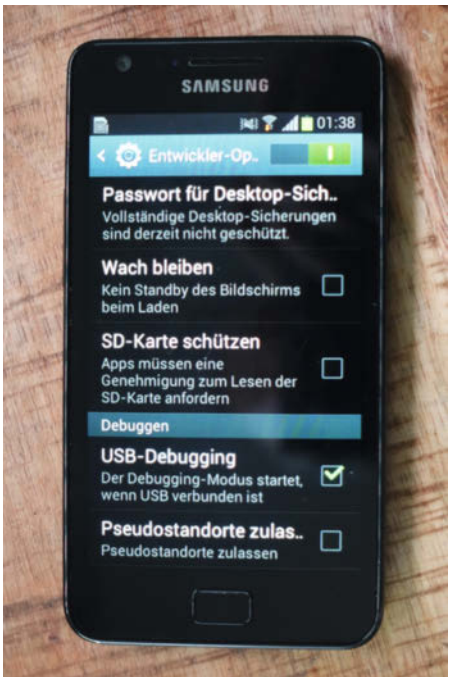


Abbildung 8: In den Entwickleroptionen kann man USB-Debugging aktivieren.

Ab Android Version 4.2 sind die Entwickleroptionen versteckt und müssen zuvor aktiviert werden. Dazu tippt man auf „Telefoninfo“, dort auf „Software-Informationen“ und dann muss man siebenmal auf die Build-Nummer tippen. Bei einzelnen Herstellern kann ein abweichendes Vorgehen erforderlich sein. Die Suchmaschine oder KI Ihres Vertrauens hilft hier gerne weiter.

Dann schließt man das Gerät über USB an den Computer an. Es erscheint ein Dialog mit der Frage, ob man dem Computer erlauben möchten, das Mobilgerät zu debuggen. Nach dem Bestätigen erscheint der Name des Geräts im oberen Rand von Android Studio.

Jetzt steckt man den Stecker des OTG-Kabels mit dem FT232 in die USB-Buchse. Das Modul wird automatisch erkannt und die USB-Zugriffsberechtigung angefordert. Danach kann man die Ausgänge mittels der Schaltflächen ein- und ausschalten oder durch Auswahl der Checkboxen blinken lassen. Mit dem Schieberegler lässt sich die Blinkgeschwindigkeit einstellen. Die vier Optionsfelder zeigen den Pegel der als Eingänge konfigurierten Pins an. Jetzt kann man Elektrobauteile mit dem Handy steuern, wie mit einem Mikrocontroller. *—das*

App über WLAN laden

Ein ständiges Umstecken zwischen Computer und FTDI-Modul ist beim Programmieren der App ziemlich lästig und unpraktisch. Zum Glück gibt es jedoch die Möglichkeit des Ladens über WLAN. So kann die USB-Schnittstelle mit dem FTDI-Modul verbunden bleiben, während man die App bequem über Android Studio auf das Gerät laden und auch debuggen kann. Und so wird's gemacht:

Abbildung 10: Über das Terminal wird der ADB-Pfad hinzugefügt.

Abbildung 11: Die letzte Einstellung für die WLAN-Übertragung

Zuerst öffnet man das Terminalfenster in Android Studio. Dann fügt man das Verzeichnis der Android Debug Bridge (ADB) zum Pfad hinzu. Sie liegt im Verzeichnis `C:\Users\{username}\AppData\Local\Android\Sdk\platform-tools`.

Die Android Debug Bridge (ADB) ist ein vielseitiges Kommandozeilenwerkzeug, das im Android SDK enthalten ist und es ermöglicht, Apps zu installieren, zu debuggen, Dateien zu übertragen und Android-Geräte umfassend zu verwalten. Jetzt muss man folgenden Befehl eingeben:

```
[Environment]::SetEnvironmentVariable("PATH",$Env:PATH+";$Env:LOCALAPPDATA\Android\Sdk\platform-tools",[EnvironmentVariableTarget]::"User")
```

Alternativ kann der Pfad auch über die Windows-Einstellungen konfiguriert werden: Im Suchfeld der Taskleiste sucht man nach „Systemumgebung“ und startet die App „Systemumgebungsvariablen bearbeiten“. Dort klickt man dann im Tab „Erweitert“ auf den Punkt „Umgebungsvariablen“. Dort kann man den ADB-Pfad unter „Benutzervariablen“ festlegen.

Danach startet man Android Studio neu, damit die Einstellungen wirksam werden. Jetzt verbindet man das Mobilgerät über USB mit dem Computer und gibt `adb tcpip 5555` ein. Es ist wichtig, dass der Computer und das Gerät im gleichen WLAN-Netz sind. Nun gibt man `adb connect <IP Adresse>` ein. Jetzt ist man über WLAN mit dem Android-Gerät verbunden und kann Apps über WLAN auf das Smartphone laden.

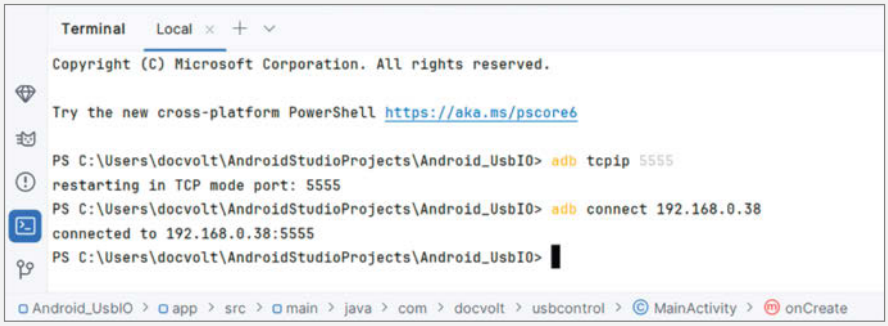


Abbildung 12: Die Verbindung über das Netzwerk hat geklappt.

Hinweise:

- Die IP-Adresse kann man entweder in den WLAN-Einstellungen des mobilen Gerätes oder der Konfigurationsseite des WLAN-Routers finden.
- Vor einer neuen Debug-Session muss man lediglich `adb connect <IP Adresse>` aufrufen. Das funktioniert auch über WLAN, das Gerät braucht dafür keine USB-Verbindung.
- `adb tcpip 5555` aktiviert WLAN-Debugging. Das bleibt so lange aktiv, bis USB-Debugging deaktiviert oder Android neu gestartet wird. Aus Sicherheitsgründen sollte man USB-Debugging stets ausschalten, wenn es nicht benötigt wird.

Physik-Experimente mit dem Smartphone

Mit den Sensoren deines Smartphones kannst du die Drehzahl deiner Fräse messen, die Vibrationen deines 3D-Druckers optimieren oder gleich dein komplettes Projekt steuern. Wie das mit der kostenfreien App phyphox funktioniert, zeigen wir in diesem Artikel.

von Sebastian Staacks

Als Leser der Make habt ihr bestimmt genau wie ich eine Schublade voll mit kleinen Breakout-Boards mit Sensoren. Wenn ihr nur lange genug mit Elektronik bastelt, entsteht unweigerlich eine bunte Mischung aus Temperatur-, Abstands- oder Hall-Sensoren und vielen anderen, die auf das nächste Projekt warten. Wenn ihr dann aber mal eben was in der Werkstatt messen wollt, wird der Einsatz dieser Sensoren schnell zu einem eigenen Projekt, denn mal ehrlich: Selbst wenn der Beispielcode, der den Sensor ausliest, innerhalb von Minuten laufen sollte, kann ein schusselig vergessener Pull-down-Widerstand Stunden kosten.

Daher ist es ganz praktisch, dass wir mit unserem Smartphone noch einen weiteren Satz Sensoren jederzeit einsatzbereit in der Hosentasche haben. Dieses ist nämlich voll davon und es kann ganz nützlich sein, diese zu kennen und mal eben nutzen zu können, etwa auch, um das nächste Projekt zu steuern.

Kurzinfo

- Mit der App phyphox auf die Sensoren im Smartphone zugreifen
- Experimente mit Vibrationen, Frequenzen, Magnetfeldern und weiteren Umgebungsvariablen durchführen
- Leichte Anbindung an Arduino- und MicroPython-Projekte über BLE

Checkliste

Zeitaufwand:
15 Minuten für den Einstieg

Kosten:
0 Euro

Alles zum Artikel im Web unter
make-magazin.de/xyw2

Mehr zum Thema

- Ramon Hofer Kraner, Im Maker-Paradies: Aktoren und Sensoren vom Schrott, Make Sonderheft 2022, S. 64
- Carsten Meyer, Breakout-Boards, Make 1/21, S. 10
- Guido Burger und Klaus-Uwe Gollmer, Flusspegel messen mit der Citizen Science Box, Make 1/23, S. 54

Die Sensoren

In der Regel findet man nicht jede Art von Sensor in seinem Smartphone und nicht jeder Sensor ist gleichermaßen nützlich. Von den am Anfang erwähnten Temperatursensoren gibt es beispielsweise oft gleich mehrere, aber diese dienen der Überwachung von Akku- und CPU-Temperatur oder dem Kalibrieren anderer Sensoren und sie sind meist so verbaut, dass man mit ihnen nicht wirklich etwas anderes messen kann.

Mit einem Hall-Sensor lässt sich schon mehr anfangen, denn fast alle Smartphones nutzen ein 3D-Magnetometer als Kompass. Da es empfindlich genug ist, um das eigentlich eher schwache Erdmagnetfeld zu messen, kann man damit auch die Felder moderater Gleichströme detektieren. Für Wechselstrom ist der Sensor nur leider meist zu langsam, aber dazu später mehr.

Hinzu kommen noch diverse kleinere Sensoren, beispielsweise um die Helligkeit des Displays zu dimmen, für Annäherung, um versehentliche Wangen-Eingaben beim Telefonieren zu verhindern und natürlich das Mikrofon, das man nicht als Sensor unterschätzen sollte.

Die App phyphox

Und warum kommt man an die Daten dieser Sensoren nun schneller ran, als an die in unseren Elektronik-Kram-Schubladen? Weil es dafür Apps gibt. Viele sind spezialisiert auf bestimmte Sensoren und bieten hier besonders ausgefeilte Funktionen, aber als Entwickler einer solchen App habe ich da natürlich eine Präferenz für den Allrounder: phyphox.

Die App wird von mir und ein paar Kollegen seit acht Jahren an der RWTH Aachen University mit dem Ziel entwickelt, die Smartphone-Sensoren zum Experimentieren in Schule und Uni verfügbar zu machen – daher auch der Name phyphox, der für Physical Phone Experiments steht. Mit dem Einsatz in der Lehre vor Augen ist die App nicht nur kostenfrei, werbefrei und im Hinblick auf den Datenschutz unbedenklich, sondern sie ist auch open source. Wer aktuell in irgendeiner Form Physik lernt, hat dank dieser App gute Chancen, früher oder später mal im Unterricht sein Handy an ein Pendel zu hängen oder auf einem Fahrradreifen kreisen zu lassen (Bild 1).

Für fortgeschrittene Nutzer bietet phyphox aber auch ein Dateiformat, um eigene Messkonfigurationen mit Auswertung zu erstellen (Bild 2), eine REST API, eine Netzwerkschnittstelle, um Daten per HTTP oder MQTT zu senden, und eine Bluetooth-Schnittstelle.

Vibrationen messen

Beim ersten Kontakt mit der App ist vieles davon aber eher verborgen, da sie die Lehre fokussiert und dort ein schnelles Auspacken und Messen ohne ablenkende Extras wichtiger ist. Und das ist auch für uns Bastler schon sehr praktisch.

Sensoren für Bewegung

Beschleunigungssensor

- Typische Abtastrate: 100 Hz bis 500 Hz
- Messgröße: Beschleunigung in m/s^2
- Achsen: 3
- Funktion: Dieser Sensor erkennt vor allem die Lage des Smartphones, da er auch die Erdbeschleunigung erfasst. Er ist beispielsweise dafür verantwortlich, dass sich das Bild dreht, wenn ihr das Gerät quer haltet.

Gyroskop

- Typische Abtastrate: 100 Hz bis 500 Hz
- Messgröße: Drehrate in rad/s
- Achsen: 3
- Funktion: Dieser Sensor unterstützt das Smartphone dabei, seine Bewegung zu rekonstruieren. Er sorgt für eine weichere Bewegungssteuerung in Spielen, und ohne Kenntnisse der Rotation wäre es nicht möglich, in den Daten des Beschleunigungssensors zwischen der Beschleunigung des Geräts und der Erdbeschleunigung zu unterscheiden.

Magnetometer

- Typische Abtastrate: 50 Hz bis 100 Hz
- Messgröße: Magnetische Flussdichte in µT
- Achsen: 3
- Funktion: Dieser Sensor kann sehr kleine Magnetfelder messen und dient auf dem Smartphone meist als Kompass. Bei großen Magnetfeldern setzt er leider aus und eine Magnetisierung des Smartphones verhindert eine korrekte Funktion als Kompass. Dies kann das Gerät jedoch in der Regel selbstständig erkennen und herausrechnen.

Sensoren rund um Licht

Helligkeitssensor

» Typische Abtastrate: 1 Hz bis 4 Hz (oder nur bei signifikanter Änderung)
» Messgröße: Beleuchtungsstärke in lx
» Funktion: Dieser Sensor passt die Helligkeit des Bildschirms eurer Umgebung an. Leider ist er auf vielen Geräten eher träge und auf iPhones für Apps nicht auslesbar.

LiDAR / ToF

» Typische Abtastrate: 10 Hz bis 30 Hz (in der Regel an Kamera-Bildrate angepasst)
» Messgröße: Distanz in mm
» Funktion: Gibt die Tiefeninformation zum Kamerabild wieder und kann in phyphox zur Distanzmessung genutzt werden. Leider nur auf teuren Geräten verfügbar und unter Android sehr selten anzutreffen. Bei iPhones kann (mit kleinen Einschränkungen) auch die FaceID-Kamera der Nicht-Pro-Modelle genutzt werden.

Kamera (in Beta-Phase)

» Typische Abtastrate: 30 Hz
» Messgröße: Leuchtdichte, Farbwert …
» Funktion: Die aktuelle offene Beta von phyphox enthält erste Funktionen, um Messgrößen aus dem Kamerabild zu gewinnen. Hier werden noch viele Funktionen folgen. (Details zur Beta-Version gibt es über einen Link in der Kurzinfo.)

Bild 1: Anhand der Drehung einer Fahrradfelge wird die Zentripetalbeschleunigung mit Smartphone-Sensoren gemessen.

Ihr müsst eine Neigung messen und habt keine Wasserwaage zur Hand? Nehmt den Beschleunigungssensor eures Smartphones bzw. die Neigungsfunktion in phyphox (Achtung: Eure hervorstehende Smartphone-Kamera trägt zur Neigung des Gerätes bei!). Ihr braucht einen Funktionsgenerator? Der Kopfhörerausgang kann wunderbare und sehr präzise Sinus-Funktionen mit einstellbarer Frequenz bis hin zum kHz-Bereich liefern. Ihr wollt die Vibrationen eures 3D-Druckers optimieren? Legt das Smartphone auf den Rahmen des Druckers und zeichnet die Beschleunigung auf, während ihr ihn einen Vibrationstest durchlaufen lasst.

Die häufigsten Anwendungen, die ich aber bei mir selbst im Einsatz sehe, sind Frequenzmessungen. Abstrakt gesagt: Sobald irgendetwas Periodisches passiert, dessen Frequenz ich messen möchte, ist phyphox meine erste Wahl. Das liegt wohl daran, dass für solche Messungen verschiedene Sensoren genutzt werden können.

An meiner CNC-Fräse kann ich zwar die Drehzahl regeln, aber nicht auslesen. Kein Problem, ich halte das Mikrofon meines Smartphones dran und kann mit der Funktion „Audio Spektrum“ anhand des Geräuschs die Drehzahl ablesen (Bild 3) und gegebenenfalls mit 60 multiplizieren, falls man den Wert in U/min statt Hz benötigt.

Die Drehzahl einer Festplatte messe ich, indem ich das Smartphone darauflege und mir das Signal des Beschleunigungssensors als Spektrum anzeigen lasse (Bild 4). An fast alles, was sich bewegt, kann ich einen kleinen Magneten anbringen und wie bei typischen Fahrrad-Tachos mit Magnet in den Speichen

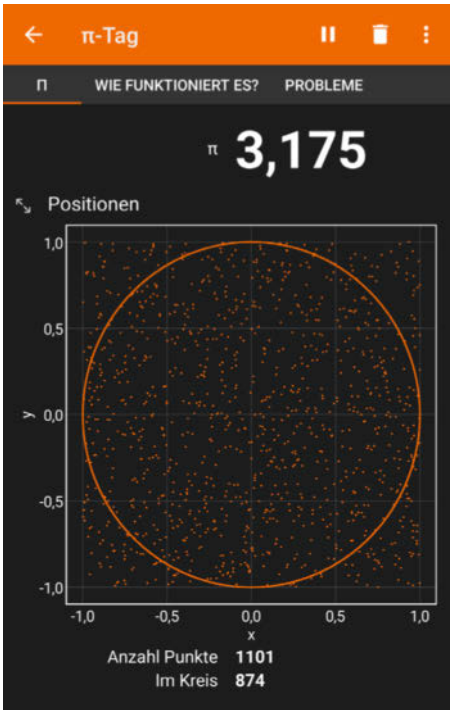


Bild 2: Mit dem Rauschen des Beschleunigungssensors lassen sich zufällige Koordinaten generieren, deren Häufigkeit im Einheitskreis die Kreiszahl π annähert.

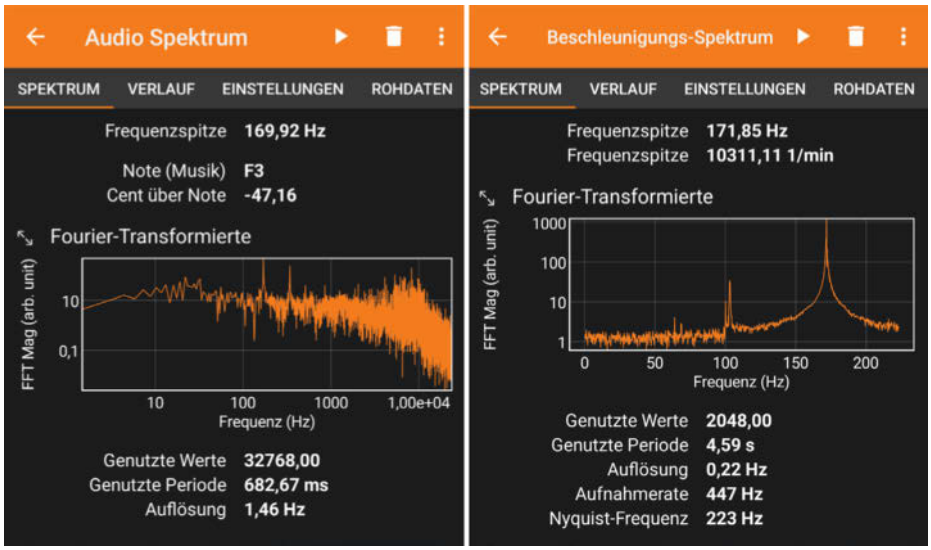


Bild 3: Messung der Drehzahl der Fräse im Titelbild. Links wurde das Mikrofon genutzt, rechts das Smartphone abgelegt und über den Beschleunigungssensor die Vibration gemessen. Dafür muss der Sensor aber schnell genug sein.

Über das +-Menü in phyphox kannst du mit diesem QR-Code die Konfiguration aus Bild 2 laden.

Sonstige Sensoren

Druck

- Typische Abtastrate: 1 Hz bis 20 Hz
- Messgröße: Luftdruck in hPa
- Funktion: Dieser Sensor dient primär dazu, Höhenunterschiede über den barometrischen Druck besser auflösen zu können, als es das GPS kann. Je nach Gerät können Höhenunterschiede von deutlich weniger als einem Meter gemessen werden. Wasserdichte Gehäuse können allerdings dafür sorgen, dass die Druckänderung im Gerät beim Anfassen größer ist als dieser kleine Höhenunterschied. Leider oft nicht auf günstigen Geräten verfügbar.

GPS

- Typische Abtastrate: 1 Hz
- Messgröße: Position (Längen-/Breitengrad und Höhe)
- Funktion: Vor allem in der Navigation eingesetzt, ist die Satellitennavigation auch gut dazu geeignet, um Geschwindigkeiten zu bestimmen. Eine freie Sicht auf den Himmel ist allerdings Voraussetzung. Die meisten Smartphones sind mit vielen verschiedenen Satellitensystemen kompatibel (etwa GLONASS).

Näherungssensor

- Typische Abtastrate: 1 Hz bis 4 Hz (oder erst bei signifikanter Änderung)
- Messgröße: Abstand nah/fern
- Funktion: Nominell als Abstand in cm wird hier eigentlich nur erkannt, ob sich etwas direkt vor diesem Sensor befindet. Er sitzt in der Regel beim Lautsprecher am oberen Bildschirmrand und erkennt, wenn man sich das Smartphone ans Ohr hält, um dann den Bildschirm abzuschalten.

Virtuelle Sensoren

- Typische Abtastrate: 50 Hz bis 100 Hz
- Funktion: Wenn ein Smartphone über ein Gyroskop verfügt, bietet es in der Regel auch virtuelle Sensoren mit fusionierten Daten von mehreren Sensoren. Dies ist beispielsweise die lineare Beschleunigung (in phyphox „Beschleunigung ohne g"), bei der Beschleunigungssensor und Gyroskop genutzt werden, um die Erdbeschleunigung herauszurechnen. Es gibt auch einen virtuellen Lagesensor, der die absolute Rotation des Smartphones wiedergibt. Je nach zugrunde liegendem Algorithmus und Situation funktioniert dies mehr oder weniger gut. Wenn ein Gerät solche Sensoren ohne Gyroskop anbietet, solltet ihr sehr skeptisch sein.

die Drehzahlen über das Magnetometer des Smartphones bekommen. Elektromotoren liefern eh ihr eigenes Magnetfeld, wobei man hier aufpassen muss, da diese mehrere Magneten bzw. Windungen pro Umdrehung haben, sodass man ein Vielfaches der Drehzahl sieht. Ähnliches gilt auch für Verbrennungsmotoren, die mit mehreren Takten pro Umdrehung arbeiten, aber ansonsten für den Beschleunigungssensor gut messbare Vibrationen erzeugen. Um solche Frequenzmessungen richtig und gut durchführen zu können, sollte man allerdings das Prinzip dahinter verstehen.

FFT messen

Es gibt viele Möglichkeiten, Frequenzen eines periodischen Signals zu bestimmen, aber die bekannteste, die ich hier auch nutze, ist die FFT. Den Begriff habt ihr vielleicht schon einmal gehört. FFT steht für „Fast Fourier Transform" und bezeichnet den Algorithmus zur Berechnung des mathematischen Werkzeugs, um das es eigentlich geht: die (diskrete) Fourier-Transformation.

Diese wiederum liefert nicht einfach eine Frequenz, sondern hat die Aufgabe, ein beliebiges Messsignal in einzelne Sinusfunktionen zu zerlegen. Die Summe dieser Sinusfunktionen ergibt wieder das ursprüngliche Signal. Das findet z. B. eine praktische Anwendung beim Komprimieren und Dekomprimieren von MP3-Dateien.

Am besten stellt ihr euch das wie eine Bauanleitung für Signale vor. Im Bild 5 seht ihr als untersten Plot ein mögliches Messsignal. Die Frage, die uns die FFT beantworten kann, lautet: Welche Sinusfunktionen muss ich mit welchen Anteilen zusammenbauen, um exakt dieses Signal zu erhalten? Die Antwort seht ihr in den drei Plots darüber. Das Messsignal kann man als die Summe von drei Sinusfunktionen darstellen, nämlich ein 5-Hz-Signal mit einer Amplitude von 1, ein schwächeres 10-Hz-Signal und ein noch mal schwächeres 20-Hz-Signal.

Eine solche Zerlegung ist immer eindeutig möglich, wenn man so viele verschiedene Frequenzen erlaubt, wie es Datenpunkte im Eingangssignal gibt. Was uns zum Begriff des Spektrums führt: Das ist eine Grafik, die für all diese Frequenzen angibt, wie stark sie im Messsignal auftreten. Für unser Beispiel seht ihr das in Bild 6. Das künstliche Signal aus den drei Sinusfunktionen hat drei Spitzen bei 5 Hz, 10 Hz und 20 Hz, deren Höhe den Amplituden der Signale entspricht (das ist mathematisch allerdings eine Frage der Normierung). Bei realer Messung rauschen natürlich alle Frequenzen ein wenig, aber ein wirklich periodisches Signal ist in der Regel gut zu erkennen (Bild 7).

Frequenzen und Grenzen

Nun liefert uns die Fourier-Transformation aber leider nicht einfach die eine Frequenz, die wir suchen, sondern eben für einen bestimmten Satz Frequenzen die jeweiligen

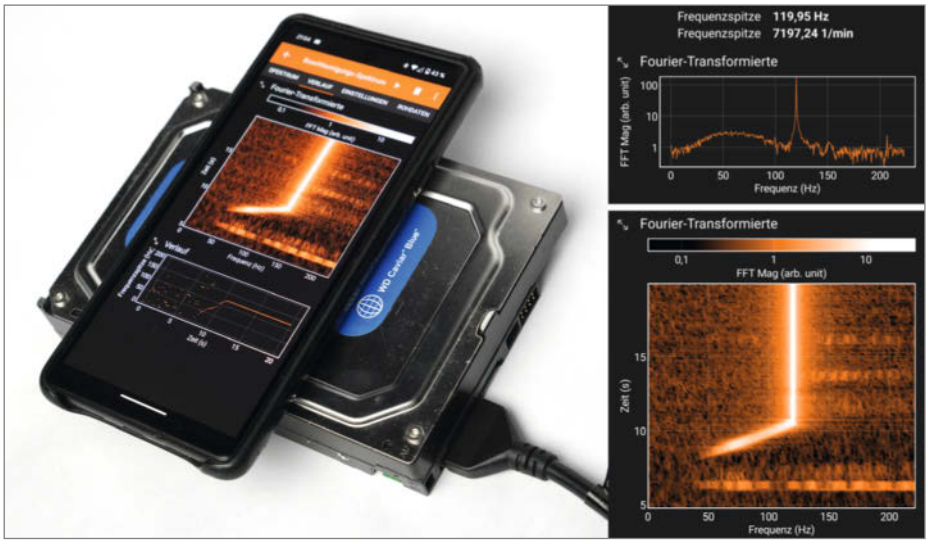


Bild 4: Die Drehzahl einer Festplatte kann über deren Vibration mit dem Beschleunigungssensor gemessen werden. Oben rechts kann man im Spektrum gut erkennen, dass es sich um ein Modell mit 7200 RPM handelt. Unten rechts kann man im zeitlichen Verlauf auch das Hochfahren des Motors sehen.

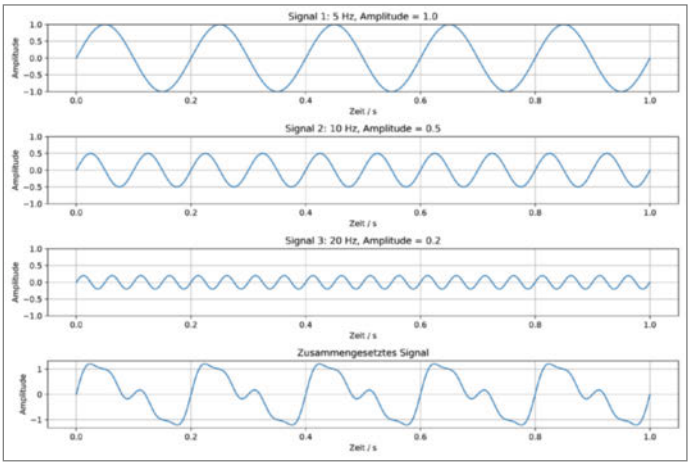


Bild 5: Das unten gezeigte zusammengesetzte Signal kann als Summe der drei oberen Sinusfunktionen dargestellt werden.

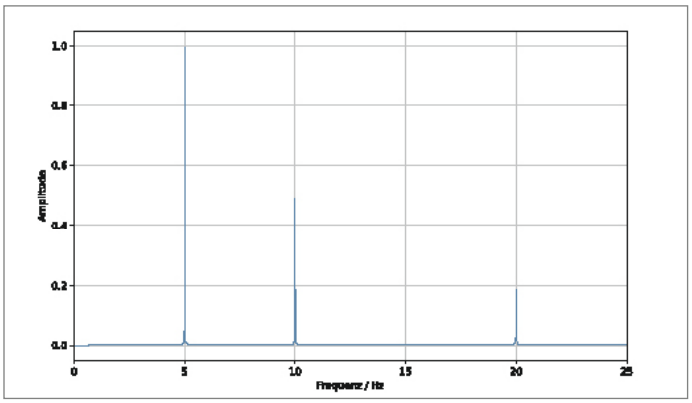


Bild 6: Das Fourier-Spektrum des zusammengesetzten Signals aus Bild 5. Gezeigt wird nur ein Ausschnitt von 0 Hz bis 25 Hz. Das Ergebnis wurde normiert, um den Zusammenhang zu den drei Einzelsignalen deutlicher zu machen.

Anteile. Soll heißen, dass wir beispielsweise ein Signal mit einer Frequenz von 440 Hz reinstecken, aber im Spektrum nur die Anteile für 422 Hz und 445 Hz angezeigt werden und wir nicht ohne Weiteres erkennen können, dass die gesuchte Frequenz dazwischen liegt.

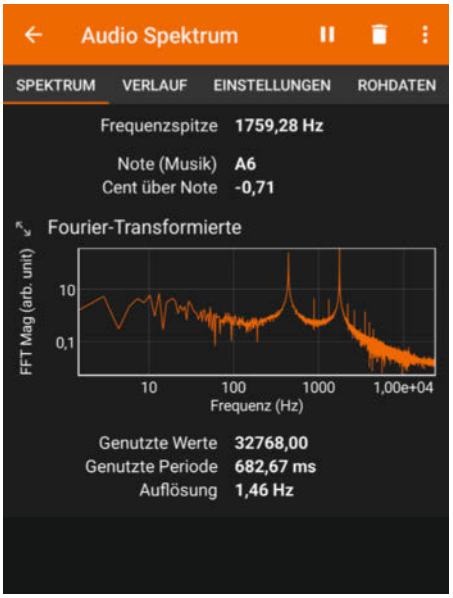


Bild 7: Reale Messung von zwei Sinustönen über das Mikrofon des Smartphones. Die Töne haben die Frequenzen 440 Hz und 1760 Hz.

Die Auswahl dieser Frequenzen hängt vom Eingangssignal ab. Ohne auf die mathematische Begründung einzugehen, erhalten wir halb so viele Frequenzen, wie wir Datenpunkte reinstecken, und diese verteilen sich gleichmäßig von 0 Hz bis zur Hälfte der Abtastrate unseres Sensors.

Dazu gebe ich am besten ein Beispiel: Ein Sensor misst 60 Werte pro Sekunde und wir stecken eine Messung über 4 Sekunden in die Fourier-Transformation. Dann erhalten wir 240 Werte, die uns 240/2 = 120 Frequenzen geben. Diese 120 Frequenzen verteilen sich bis zur Hälfte der Abtastrate, also von 0 Hz bis 60 Hz/2 = 30 Hz. Somit können wir Frequenzen auf 30 Hz/120 = 0,25 Hz genau bestimmen.

Im Grunde müsst ihr euch nur zwei Dinge merken:

- Je länger ihr messt, desto genauer wird das Spektrum. (In den Spektrum-Experimenten von phyphox kann man die Anzahl der Punkte im Einstellungen-Tab wählen.)
- Die höchste messbare Frequenz ist die Hälfte der Abtastrate des Sensors und hängt somit von eurem Gerät ab.

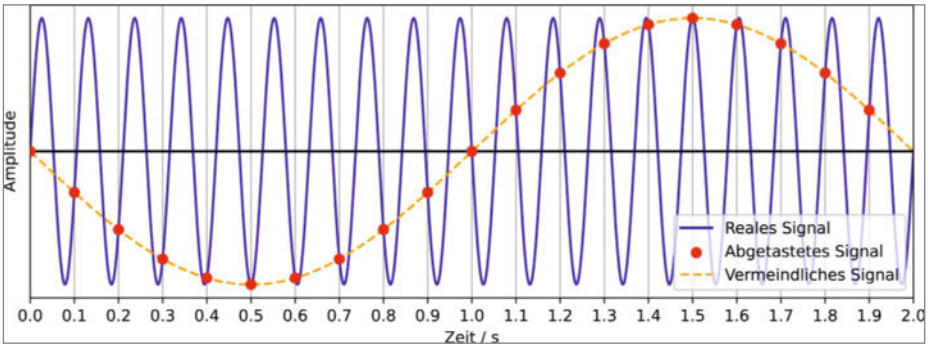


Bild 8: Illustration des Alias-Effekts: Das reale Signal (blau) wird von einem Sensor mit zu kleiner Abtastrate gemessen (rote Punkte). Aus den Messwerten kann man nicht mehr auf das reale Signal schließen, sondern die Werte scheinen dem gelben Signal zu entsprechen.

Der Alias-Effekt

Was passiert denn, wenn euer Messsignal eine höhere Frequenz als die Abtastrate hat? Dann lernt ihr den Alias-Effekt kennen. Den Begriff habt ihr vielleicht schon im Kontext von Spielgrafik gehört. Er beschreibt Probleme, die auftreten, wenn die Abtastung eines Signals zu grob für die Details ist, die eigentlich im Signal stecken. In der Computergrafik ist das die Auflösung des erzeugten Bildes gegenüber den Details in der Spielszene, also beispielsweise zu grobe Pixel, um eine scharfe Kante gut darstellen zu können. Beim Messen geht es aber vor allem darum, dass ihr zu wenige Messpunkte habt um die hohe Frequenz zu erkennen.

Schaut euch dazu mal das Bild 8 an. Das blaue Signal ist ein hochfrequentes Signal, das ihr messen wollt. Euer Messgerät schafft aber nur 10 Messwerte pro Sekunde. Ihr bekommt also nur bei jeder Markierung der x-Achse, also alle 0,1 s, einen Messwert, der hier mit einem roten Punkt hervorgehoben ist. Nun stellt euch vor, ihr habt nur die roten Punkte gemessen und kennt das blaue Signal nicht. Welche Frequenz würdet ihr aus den roten Punkten interpretieren? Sicherlich die gelbe Linie mit einer wesentlich kleineren Frequenz.

Genau diese zu kleine Frequenz liefert auch die FFT in einem solchen Fall. Während man einem Spektrum in der Regel eine zu kleine Auflösung anmerkt, kann man durch den Alias-Effekt schnell eine vollkommen falsche Frequenz messen, ohne es zu merken. Manche Smartphones messen intern mit einer höheren Rate, als sie an Apps wie phyphox weitergeben, was diesen Effekt ver-

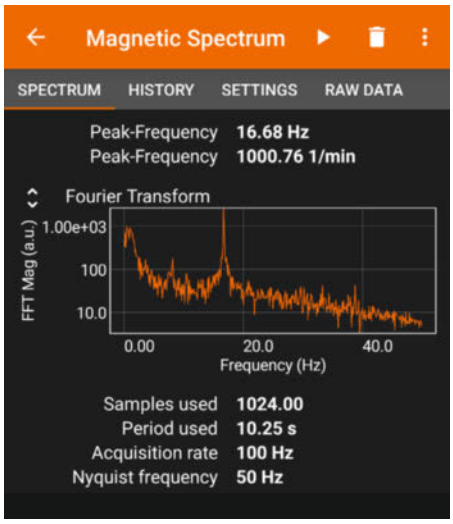


Bild 9: Messung des Magnetfeld-Spektrums bei einer Bahnfahrt. Man erkennt das Stromnetz der Bahn, welches mit einer Frequenz von 16,7 Hz arbeitet.

hindern kann. Aber man sollte sich des Problems bewusst sein und zumindest eine Vorstellung haben, ob das zu messende Signal im Messbereich des Sensors liegt.

Welchen Sensor wofür?

Und das führt unweigerlich zu der Frage, welche Sensoren denn nun für welche Bereiche geeignet sind. Der schnellste ist hier ohne Frage das Mikrofon, das mit 48 kHz (oder manchmal 44,1 kHz) und damit einem Spektrum bis 24 kHz den Bereich des menschlichen Hörens abdeckt. Insbesondere hohe und sehr niedrige Frequenzen sind allerdings etwas in ihrer Amplitude gedämpft und bei Frequenzen unter 50 Hz wird es je nach Gerät sehr schwer, noch etwas zu messen.

Der nächste spannende Sensor für Vibrationen ist der Beschleunigungssensor, der je nach Smartphone in der Regel eine Abtastrate zwischen 100 Hz und 500 Hz liefert. Damit könnt ihr die meisten mechanischen Vibrationen gut messen, wobei es bei der oben gezeigten Festplatte mit langsamen Sensoren aber schon knapp werden kann.

Von den für uns interessanten Sensoren ist das Magnetometer leider das langsamste. Hier sind 50 Hz oder 100 Hz und damit ein Spektrum bis 25 Hz bzw. 50 Hz üblich. Das Ärgerliche daran ist, dass der Wechselstrom unseres 50-Hz-Stromnetzes damit gerade herausfällt. Ich vermute sogar, dass die meisten Geräte intern höhere Raten einsetzen, um eben den Einfluss von Stromleitungen für die doch sehr empfindliche Anwendung als Kompass zu filtern. Um langsamere Drehungen mit einem Magneten zu markieren, reicht es aber doch, und wenn ihr das nächste Mal mit der Bahn fahrt, könnt ihr euch mal das Magnetfeld-Spektrum dort anschauen. Die Bahn nutzt nämlich ein Drittel der normalen Netzfrequenz und liefert daher ein (je nach Sitzplatz) deutlich erkennbares Signal, sobald sie losfährt (Bild 9).

Bluetooth für Mikrocontroller

Wir müssen uns aber auch nicht darauf beschränken, die Sensoren des Smartphones nur als schnelles Messwerkzeug zu verwenden. Phyphox bietet eine Bluetooth-Low-Energy-Schnittstelle (BLE), um sich mit anderen Geräten verbinden zu können.

Das war ursprünglich vor allem für das Einbinden externer Sensoren gedacht – also nicht zum Senden der Smartphone-Sensordaten an das eigene Projekt, sondern zum Darstellen der Projektdaten auf dem Smartphone-Bildschirm. Hierzu können Messkonfigurationen über ein XML-Format angelegt werden, die einerseits bestimmen, welche Sensoren ausgelesen, ausgewertet und dargestellt werden. Außerdem kann man auch BLE-Geräte definieren, an die Daten gesendet oder von denen Daten empfangen werden sollen. Hierbei bietet phyphox umfangreiche Möglichkeiten, um

BLE mit Arduino

```
#include <phyphoxBle.h>

void setup() {
 PhyphoxBLE::start("phyphox device");
}

void loop() {
 float randomNumber = random(0,100);
 PhyphoxBLE::write(randomNumber);
 delay(50);
}
```

BLE mit MicroPython

```
from phyphoxBLE import PhyphoxBLE, Experiment
import time
import random

def main():
    p = PhyphoxBLE()
    p.start()

    while True:
        randomNumber = random.randint(0,100)
        p.write(randomNumber)
        time.sleep_ms(500)
```

das binäre Format festzulegen, sodass die App auch mit Geräten sprechen kann, die nicht für phyphox ausgelegt sind.

Da unser eigenes XML-Format aber erst mal gelernt werden möchte und die Details der BLE-Kommunikation nicht jedem vertraut sind, haben wir noch eine einfachere Alternative in Form einer Arduino-Bibliothek entwickelt. Diese könnt ihr einfach aus der Bibliotheksverwaltung der Arduino IDE installieren, indem ihr dort nach „phyphoxBLE“ sucht.

Einmal installiert, benötigt ihr nur drei Zeilen Code, um die Daten eines existierenden Projekts an phyphox zu senden und dort zu plotten. Im Listing „BLE mit Arduino“ seht ihr ein Beispiel, in dem Zufallszahlen stellvertretend für einen Sensor auf dem Arduino generiert und in phyphox dargestellt werden. Ihr müsst nur die Bibliothek mit `#include` einbinden, im Setup-Block initialisieren (ab dann ist euer Arduino per BLE-Scan auffindbar) und dann eure Daten der Bibliothek übergeben. Natürlich sind auch Befehle verfügbar, um Achsenbeschriftungen hinzuzufügen, weitere Messwerte zu übertragen und diese in verschiedenen Varianten und Kombinationen darzustellen.

Die Arduino-Bibliothek ist derzeit mit den gängigsten BLE-fähigen Arduinos (z. B. Arduino Nano 33 BLE oder Arduino UNO R4 Wifi) kompatibel und unterstützt auch nahezu alle ESP32-basierten Boards. Für letztere bieten wir auch ein MicroPython-Paket an, das über PyPI installiert werden kann und ähnliche Funktionen bietet (siehe Listing „BLE mit MicroPython“).

Wenn ihr also doch einen Sensor aus der Schublade nehmt und beispielsweise mal eben die Ausgabe eines Abstandssensors plotten möchtet, müsst ihr euch nicht mit einer Zahlenkolonne in der Konsole zufriedengeben, sondern könnt euch die Daten komfortabel auf dem Smartphone ansehen (Bild 10).

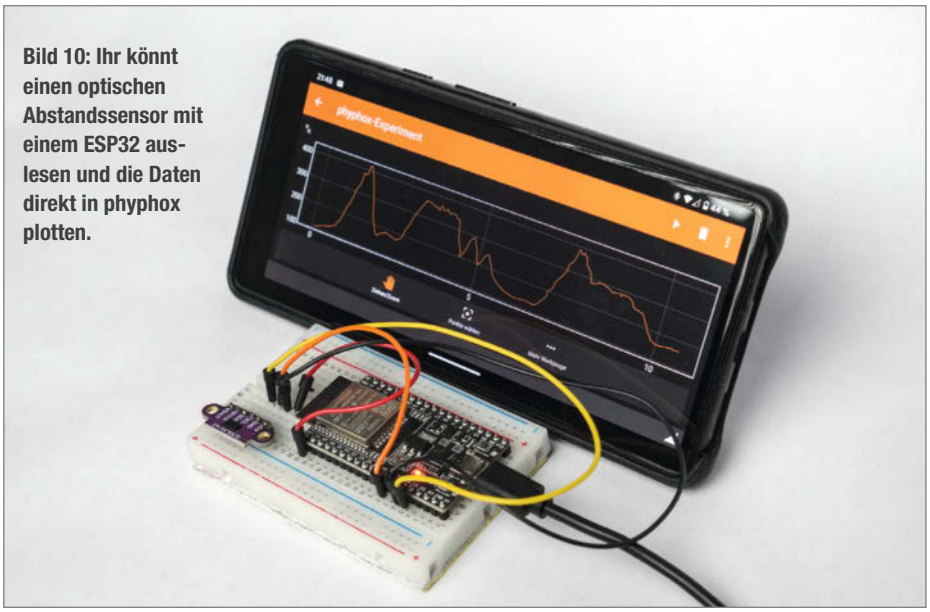


Bild 10: Ihr könnt einen optischen Abstandssensor mit einem ESP32 auslesen und die Daten direkt in phyphox plotten.

Sensordaten empfangen

```
#include <phyphoxBle.h>

void setup() {
  PhyphoxBLE::start();
  PhyphoxBLE::configHandler=&receivedData;

  PhyphoxBleExperiment::View view;
  view.setLabel("Note");

  PhyphoxBleExperiment::InfoField infoText;
  infoText.setInfo("Sensordaten werden über BLE gesendet.");
  firstView.addElement(infoText);

  PhyphoxBleExperiment::Sensor smartphoneAcc;
  smartphoneAcc.setType(SENSOR_ACCELEROMETER);
  smartphoneAcc.setAverage(true);
  smartphoneAcc.setRate(80);

  smartphoneAcc.mapChannel("x",1);
  smartphoneAcc.mapChannel("y",2);
  smartphoneAcc.mapChannel("z",3);

  PhyphoxBleExperiment exp;
  exp.addView(firstView);
  exp.addSensor(smartphoneAcc);

  PhyphoxBLE::addExperiment(exp);
}

void loop() {
  // Code deines Projekts
}

void receivedData() {
  float x,y,z;
  PhyphoxBLE::read(x,y,z);
}
```

Vom Smartphone zum Mikrocontroller

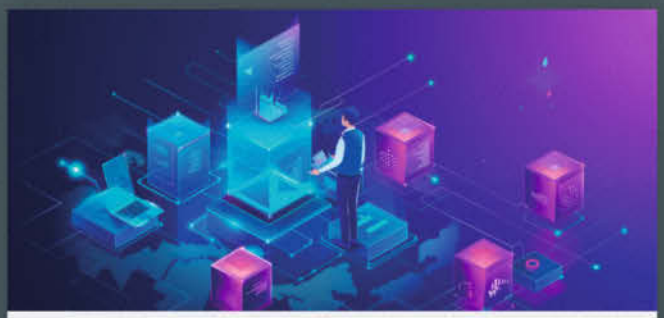
Ihr könnt das Ganze auch umdrehen, denn die Arduino-Bibliothek bietet auch die Möglichkeit, Daten vom Smartphone am Mikrocontroller zu empfangen. Wenn man eine phyphox-Konfiguration als XML erstellt, kann diese beliebige Daten von jedem Sensor enthalten. Aber um natürlich auch hier ohne Kenntnis unseres Dateiformats arbeiten zu können, gibt es für die gängigsten Sensoren auch eine Lösung mit wenigen Zeilen Arduino-Code (siehe Listing „Sensordaten empfangen“). Wählt in eurem Code einfach den Sensor aus und welche Achsen ihr empfangen möchtet, und schon stehen die Daten eurem Projekt zur Verfügung.

Vielleicht wollt ihr einfach mal Luftdruckdaten mit dem Smartphone messen, bevor ihr einen entsprechenden Sensor für euer Projekt kauft. Möglicherweise wollt ihr aber auch euren neuen Roboter durch das Neigen des Smartphones über den Beschleunigungssensor steuern. Wir freuen uns darauf, zu erfahren, was ihr mit diesem Werkzeug anstellt.

Weitere Informationen zu phyphox findet ihr über den Link in der Kurzinfo. Von dort gelangt ihr in unser Wiki mit der Dokumentation der Schnittstellen, aber auch zu unserer Sammlung von Experimenten, die mit phyphox umsetzbar sind. Wenn ihr euch den Quellcode sowie Details zur Arduino-Bibliothek und dem MicroPython-Modul anschauen wollt, besucht außerdem gern das verlinkte GitHub-Repository. *—akf*

iX @ heise academy

WORKSHOPS 2024



13. – 14. November

Startklar: Ihre Rolle als Informationssicherheitsbeauftragte

Sie erhalten einen umfassenden Überblick über die erforderlichen organisatorischen Kenntnisse.

19. – 21. November

Docker und Kubernetes für Cloud-native Softwareentwicklung

Sie erwerben fundiertes Wissen neuster Container-Technologien von Docker und der Orchestrierung mit Kubernetes.

22. November

Recht für Admins

Rechtssicherer IT-Betrieb: Grundlagen, Stolperfallen und praktische Lösungen

Dieser eintägige Online-Workshop beleuchtet die wichtigsten rechtlichen Themen von System-Administrierenden.

27. – 28. November

Windows 11 im Unternehmen absichern

Die Schulung behandelt das Zusammenspiel der zahlreichen Sicherheitsmechanismen und -Tools in Windows.

27. – 28. November

Digital Forensics & Incident Response

Schützen Sie Ihr Unternehmen vor Cybervorfällen. Erfahren Sie in dieser Schulung, , wie Sie bei Phishing, Viren und Ransomware richtig reagieren.

28. November

Exchange Migration: Von Microsoft Exchange Server zu Exchange Online

In diesem Workshop lernen Sie die verfügbaren, hybriden Migrationsmöglichkeiten kennen.

Weitere Infos unter **heise-academy.de/marken/ix**

Alte Handys als Musikabspieler

Ein ehemaliges Schubladenhandy als Streaminggerät für Musik: platzsparend, kostengünstig und stromsparend. Die Komplettlösung zum Basteln inklusive Akkuersatz stellt dieser Artikel vor.

von Uwe Schilling

Smartphones sind vollwertige und leistungsfähige Rechner mit einer Vielzahl von Ein- und Ausgabegeräten und einem riesigen Fundus an Software, die man nicht einfach wegwerfen will. Die Rechenleistung ist vergleichbar mit einem Raspberry Pi und Schnittstellen wie Bildschirm mit Touch, Tastatur, Audio-DA-Wandler sind inklusive. Kann man also für rund um die Uhr verfügbare Dienste nicht einfach ein altes Handy verwenden, statt einen Raspberry Pi neu zu kaufen?

Der folgende Artikel zeigt am Beispiel eines Musik-Servers und -Renderers, dass das möglich ist. Besonderer Wert wurde auf Betriebs- und Brandsicherheit gelegt, weil Smartphones eigentlich nicht für den jahrelangen unbeaufsichtigten Dauerbetrieb am Stromnetz gedacht sind. Die im Akku gespeicherte Energie erzeugt ein höheres Brandpotenzial, als man es von üblicher Heimelektronik gewohnt ist.

Kurzinfo

» MP3-Streaming für Audioverstärker
» Netzbetrieb von Smartphones mit herausgenommenem Akku
» Große Speicherkarten in alten Smartphones

Checkliste

Zeitaufwand:
Ein Wochenende

Kosten:
Ab 1 Euro, je nach vorhandener Hardware

Löten:
Grundkenntnisse

Mehr zum Thema

» Carsten Meyer, Bluetooth-Audio-Module, Make 1/22, S. 32
» Dr. Andreas Gräßer, Power aus dem Nabendynamo, Make 4/20, S. 58
» Elke Schick, Mobiltelefone und Mobilfunk, Make 5/20, S. 96

Alles zum Artikel im Web unter *make-magazin.de/x9n7*

Werkzeug

» **Lötwerkzeug** Lötkolben, Lötzinn, Schwamm
» **Multimeter**
» **Heißklebepistole**

Material

» **Smartphone** etwa Baujahr 2012 – 2017 mit herausnehmbarem Akku
» **MicroSD-Karte** bis 512 GByte
» **USB-Steckernetzteil** 5 V / 1 A
» **Leistungs-p-n-Diode** 2 – 5 W
» **Elektrolytkondensator** 200 – 2000 µF, > 10 V
» **Widerstand** ca. 5 kΩ
» **Widerstand** 2,2 Ω, 2 W
» **Schalter** Ein / Aus
» **Micro-USB-Stecker** eventuell vom Steckernetzteil
» **Audiokabel** Stereo, 3,5 mm Klinke auf Cinch
» **Klingeldraht** 30 cm
» **Platine** selbst geätzt oder Lochraster

Dauerläufer

Unter alten Android-Versionen wie 4.3 (Jelly Bean) ließ sich mit einem Smartphone nicht viel anfangen, weil das Betriebssystem immer mal Apps beendete, selbst wenn nur eine einzige App lief und RAM-Speicher noch in Massen verfügbar war. Ab Android 10 sind vor allem in dem alternativen LineageOS alle Stellschrauben verfügbar, um einem dauerhaft am Strom hängenden Handy diese Flausen auszutreiben.

Dank einer großen Community werden für ehemalige Spitzengeräte noch neuere Versionen des Android-Betriebssystems portiert. Daher kann es sich lohnen, solch ein altes Gerät auf dem Flohmarkt oder über Kleinanzeigen zu kaufen. Auf Sicherheitsupdates muss man aber für ältere Geräte verzichten.

Die wichtigste Quelle für aktuelles Android auf alten Handys ist vermutlich LineageOS. Man darf nicht erwarten, dass jedes Feature der Originalgeräte unterstützt wird. Zum Beispiel fehlt mir der auf Samsung-Geräten obligatorische Hardwaretest durch die Wahl von `*#0*#`. Die Funktionsfähigkeit des Handys sollte man deshalb überprüfen, bevor man ein alternatives OS aufspielt. Im Großen und Ganzen kommt mit LineageOS ein schlankes System ohne unnütze Werbung für kostenpflichtige Apps auf das Handy und auf Wunsch auch ganz ohne Googles proprietäre Apps.

Brandgefahr Akku

Ein Problem gibt es aber noch, den Akku. Wenn das Gerät dauerhaft am Netz hängt, ist der Akku immer vollgeladen. Durch Selbstentladung verliert er ein wenig seiner Ladung und wird deshalb regelmäßig wieder auf die maximal erlaubte Zellenspannung gebracht. Für einen Lithium-Akku ist das Aufladen an der oberen Kapazitätsgrenze eine besondere Tortur, die sich negativ auf die Haltbarkeit auswirkt. Noch schlimmer ist nur das Aufladen eines tief entladenen Akkus. Hier kann man gleich drauf warten, dass er sich aufbläht.

Für viele teure Geräte bedeuten aufgeblähte Akkus (Bild 1) das Lebensende, weil der Akku das Front- oder Rückglas sprengt, um sich Platz zu verschaffen. Meistens das Frontglas, weil es dünner ist als die Glasrückseite.

Für mich ein noch größeres Problem als ein zerstörtes Gerät ist die latente Brandgefahr.

Bild 1: Aufgeblähte Akkus

UPnP – DLNA – Open Home

UPnP ist der ursprünglich von Microsoft erfundene Universal-Plug-and-Play-Standard, der neben Audiogeräten hauptsächlich Drucker an der Firewall des Routers vorbei automatisch ins interne Netz bringt. Audio- und Videogeräte können in den Rollen Mediaserver (Server), Abspieler (Renderer) und Kontrollpunkt (Controller) vorkommen. In Verruf geraten war das Protokoll, weil einige Router die Geräte auch im Internet sichtbar machten.

DLNA (Digital Living Network Alliance) war die Zertifizierungsorganisation, die ein funktionierendes Miteinander verschiedener UPnP-Geräte sicherstellen sollte. Sie hat sich 2017 aufgelöst, weil sie ihre Ziele erreicht wähnte.

Open Home ist eine von der Firma Linn entwickelte Erweiterung des UPnP-Protokolls, bei dem die aktuell abgespielte Titelliste nicht auf dem Kontrollpunkt liegt, sondern nach der Musikauswahl auf dem abspielenden Gerät. Deshalb müssen zu Beginn jedes Songs nicht mehr drei Geräte miteinander reden (Controller mit Server und Renderer), sondern nur noch der Server mit dem Renderer. Das reduziert die Fehleranfälligkeit und auch unterbrechungsfreies Abspielen (Gapless Playback) ist einfacher zu realisieren.

Ein voller Akku speichert 8 bis 10 Wh. Abgegeben wird die Energie normalerweise über einen längeren Zeitraum als hosentaschentaugliches halbes Watt. Im Falle eines Kurzschlusses geht aber viel mehr, zum Beispiel einige Minuten lang 50 W. Das reicht, um brennbare Stoffe zu entzünden. Für wen sich die Watt-Angaben zu abstrakt anhören, der erinnere sich bitte daran, wie lange man warten musste, um eine 60-W-Glühbirne (die mit dem Wolfram-Glühdraht drin) nach dem Ausschalten anfassen zu können.

Meine eigene Erfahrung mit Akkubränden beschränkt sich zum Glück auf einen glühenden Klingeldraht, der an einem selbst gebauten Solar-Akku-Lader mal versehentlich einen Kurzschluss des 1000-mAh-Akkus verursacht hat. Das habe ich schnell gemerkt, weil es aus der Deckeltasche des Rucksacks qualmte und verschmort roch.

Für den durchgehenden unbeaufsichtigten Betrieb mehrerer Smartphones über Jahre hinweg wird mir das Brandrisiko aber zu groß. Das heißt, der Akku muss weg.

Ohne Akku gibt es nur das übliche, sehr viel geringere Brandrisiko von Heimelektronik. Nur die Kondensatoren speichern noch ein wenig Energie und bei einem Kurzschluss knallt es mal kurz, bis die Geräte- oder Wohnungssicherung die Energiezufuhr unterbricht.

Akkuloser Betrieb

Glücklich darf sich schätzen, wer ein so altes Smartphone besitzt, dass er den Akku einfach durch Öffnen der Rückseite herausnehmen kann. Bei Samsung boten die Spitzengeräte bis Galaxy S5 einen Wechselakku. Auch Derivate wie das Galaxy Alpha und die Juniorversionen J3 und J5 kamen bis 2016 mit herausnehmbarem Akku. Ohne Akku startet allerdings fast kein Smartphone. Mir ist bisher ein einziges begegnet, das ohne Akku startet: ein Huawei Y300. Alle anderen sagen ohne Akku keinen Mucks.

Der Grund liegt darin, dass beim Start kurzfristig viel Strom verbraucht wird, um alle Baugruppen zu initialisieren, den Speicher zu befüllen, WLAN- und Telefonnetz zu suchen, sich überall wieder anzumelden und frische Werbung aus dem Netz zu ziehen. Ein normal zum Telefonieren genutztes Samsung Galaxy S3 mit originalem Android 4.3 zog bei mir wenige Sekunden lang mehr als 1,4 A aus einem mittels programmierbarem Netzteil simulierten halb vollen Akku mit 4,0 V Zellenspannung. Mitgeliefert wurde damals ein Micro-USB-Steckernetzteil für maximal 1 A an 5 V. Das könnte also

LineageOS aufspielen

Auch LineageOS unterstützt Smartphones nicht ewig. Aktuell werden nur noch Geräte ab dem S9 unterstützt. Für ältere Modelle gibt es häufig noch inoffizielle LineageOS-Builds; für das sehr verbreitete S3-Grundmodell zum Beispiel von diversen Websites (siehe Link in der Kurzinfo). Alles, was ich damit tun will, funktioniert, aber es gibt keinen offiziellen Support von Lineage oder Sicherheitsupdates von Google mehr. Bevor ein Custom-ROM aufgespielt wird, hat man bei allen Samsung-Geräten letztmalig die Möglichkeit, mit `*#0*#` die Funktionsfähigkeit der Hardware zu testen. In Custom-ROMs ist dieser Test nicht implementiert.

Um ein alternatives ROM auf ein Samsung-Gerät zu bringen, gibt es viele Wege. Man kann dem Ablauf auf der Website von LineageOS folgen und das mit Heimdall und der LineageOS-Recovery erledigen. Ich mache das mit dem TWRP (Team Win Recovery Project) und Odin, einem weiteren Tool von Samsung. Der Ablauf funktioniert wie folgt:

- Download und Installation des Odin-Flash-Tools von Samsung auf einem Windows-PC
- Das Recovery-System TWRP herunterladen und nach Anleitung mit Odin installieren (unbedingt auf genaue Modellnummer achten!). Sobald das Handy den Reset durchführt, den Akku entnehmen, wieder einsetzen und mit den Tasten Home, Lauter und Power in die Recovery booten, sonst überschreibt das alte ROM die gerade installierte Recovery wieder mit der Originalversion.
- Das Custom-ROM (ein ZIP-Archiv) auf eine MicroSD-Karte kopieren und mit TWRP installieren. Vor dem Neustart noch einmal im Menü zurückgehen und mit „Advanced Wipe“ die Partitionen Cache, Dalvik-Cache, Data, Internal Storage, Preload (der Ort, von dem aus die Hersteller ihre Bloatware installieren) und SD-Card (nicht die MicroSD-Card!) formatieren. Sonst landet man meistens in einer Boot-Schleife. Punkte, die nicht im Menü auftauchen, gibt es auf diesem Handy auch nicht.
- Die Google-Apps braucht man für dieses Projekt, wenn man die BubbleUPnP-App registrieren möchte, um z. B. den Autostart freizuschalten und die Werbung loszuwerden. Die separate License-App benötigt den Google Play Store. Im Auswahlmenü des Links muss man das zur CPU (ARM) und Android-Version passende Paket in der entsprechenden Geschmacksrichtung herunterladen (normalerweise reicht „pico“) und gleich bei der Installation des Custom-ROMs als weitere ZIP-Datei angeben. Dann werden einige Lineage-Komponenten durch Google-Äquivalente ersetzt. Man kann die Google-Apps auch nachträglich über die Recovery installieren, muss dann aber noch mal das Einrichtungsmenü durchlaufen. Für andere Anwendungen braucht man Google meist nicht zu installieren, sondern kann die APK-Dateien über die SD-Karte auf das Handy laden.

Boot-Loader & Co.

Der **Boot-Loader** wird nach dem Einschalten als Erstes durchlaufen. Er wertet den Neustart-Wunsch vom letzten Ausschalten aus und springt je nach den zusammen mit dem Einschaltknopf gedrückten Tasten in die Recovery, ins Betriebssystem oder in den Download-Mode.

Die **Recovery** startet, wenn das Smartphone kein Betriebssystem findet. Mit der originalen Samsung-Recovery lässt sich nicht viel anfangen, nur die Caches löschen und das Telefon zurücksetzen. Das hilft nur bei manchen Softwareproblemen oder falls das Telefon sehr langsam wird. Für das Aufspielen eines alternativen ROMs benötigt man auf jeden Fall was anderes. Mir gefällt TWRP ausgezeichnet, das sich im Laufe der Zeit zu einem ordentlichen Dateimanager gemausert hat. Damit kann man auch durch einfaches Umbenennen von Dateien Sperrbildschirme deaktivieren oder bei technischen Problemen etwa die Bilder auf SD-Karte retten. Auch LineageOS bringt seine eigene Recovery mit, die schlicht Recovery heißt.

Das **Custom-ROM** ist das Betriebssystem, das man anstelle des originalen Samsung-ROMs auf dem Handy laufen lassen möchte. Neben LineageOS gibt es noch einige Alternativen wie „Resurrection Remix", „e" oder das schon einige Jahre nicht mehr aktive CyanogenMod. Wenn etwas nicht klappt, erhält man Original-Samsung-ROMs von Sammobile.com. Das Kürzel für deutsche ROMs ohne Modifikation durch Telefongesellschaften ist DBT. Nach Installation so eines ROMs startet man wieder bei null. Auch die Recovery ist wieder mit der Originalen überschrieben.

Download-Mode ist der Modus, in dem sich bei Samsung-Telefonen neue Software aufspielen lässt. In diesen Modus gelangt man, wenn man das Handy mit gedrückter Leiser- und Home-Taste einschaltet und nach der grünen Meldung die Lauter-Taste drückt. Bevor irgendwas mit Odin oder Heimdall auf das Telefon gebracht werden kann, muss Windows erst mal den Treiber für diesen Modus installieren. Erst danach kann das Telefon in diesen Programmen auftauchen.

die kurzzeitigen Verbrauchsspitzen gar nicht liefern. Wenn ich am programmierbaren Netzteil nur geringere Maximalströme zulasse, stürzt das Handy ab und startet neu.

Musik-Zuspieler

Nachdem ich vor Jahren meinen Plattenspieler und die beiden 200-CD-Wechsler in den Ruhestand geschickt habe, konsumiere ich Musik nur noch aus MP3-Dateien. Über die Jahrzehnte ist meine Sammlung gewachsen, sodass sie nicht mal eben aufs aktuelle Handy passt.

Schon seit einigen Jahren betreibe ich Handys als Musikabspieler, die über WLAN gesendete MP3-Dateien in Stereosignale umwandeln, die ich auf die Analog-Eingänge alter HiFi-Verstärker geben kann. Die technische Basis ist das 25 Jahre alte „Universal Plug-and-Play" (UPnP) mit der Open-Home-Erweiterung. In diesem System können Geräte als Medienzulieferer, -abspieler oder Steuergeräte fungieren. Open Home macht das System praktikabel, indem im Gegensatz zu UPnP oder DLNA (Digital Living Network Alliance) die Playlist auf dem abspielenden Gerät ver-

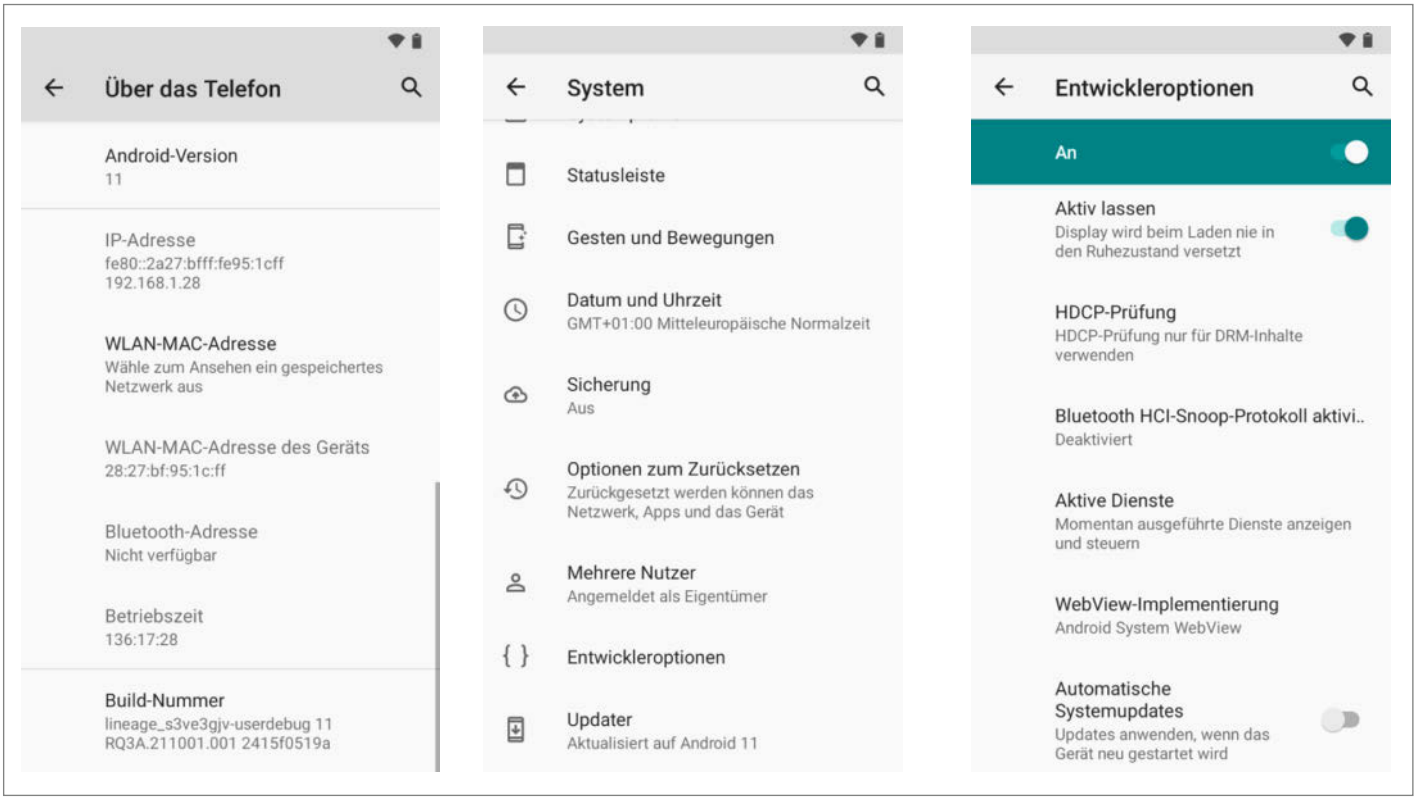


Bild 2: Die wichtigsten Systemeinstellungen

Bild 3: Speicher

waltet wird und nicht auf dem Controller (Steuergerät). Ich kann also mit meinem normalen Alltagshandy die Musik auswählen und das Handy danach wieder ausschalten und in die Tasche stecken oder auch mal den Raum verlassen, und die Musik läuft weiter.

Auf dem Papier las sich UPnP auch vor 20 Jahren schon toll. Woran das System krankte, war neben der schlechten Praxistauglichkeit, die es erst durch Open Home bekommen hatte, vor allem die geringe Robustheit des kompletten Systems von Server, Kontrollpunkt und Renderer. Eine erste Verbindung zwischen zwei Geräten konnte man schnell hinbekommen. Bei dreien wurde es schon kniffliger und wenn man nach zwei Tagen mal nachschaute, ob es noch lief, kam schnell Ernüchterung auf. Auf Android-Handys hatten sich die Programme beendet oder die WLAN-Verbindung war weg oder die Mediathek oder die Geräte kannten sich nicht mehr oder, oder, oder. Eine endlose Quelle des Frustes. Oft lief es auf den Neustart von zwei Handys hinaus, bis sich alle Geräte erneut gefunden hatten. Da hatte ich schon wieder vergessen, was ich eigentlich hören wollte.

Meine Anforderung an ein Musiksystem ist, dass es mindestens 10, besser 20 Jahre störungsfrei durchläuft, ohne dass ich mich wieder damit beschäftigen muss. Ist das utopisch? Die anderen Komponenten des Systems erfüllen diese Anforderungen problemlos. Audioverstärker sind seit 50 Jahren ausentwickelt, LAME als Encoder für MP3s ist seit 25 Jahren eine hervorragende Wahl und Cinch-Kabel checke ich auch nur einmal alle 50 Jahre, ob sie schon wegkorrodiert sind. Warum sollte nicht auch das Zuspielsystem so lange nutzbar sein? Den Router werde ich sicher häufiger mal wechseln müssen, seitdem ich nicht mehr mit einem extra Router ein separates Netz nur für die Musik aufspanne. Aber mit gleichem WLAN-Passwort sollte sich alles schnell wieder finden.

Ausgangssituation

Seit 15 Jahren gebe ich UPnP gelegentlich eine Chance und bin zwischenzeitlich aus Frust immer wieder auf die Notlösung, direkt vom Laptop abzuspielen, zurückgekommen. Mittlerweile habe ich eine Lösung gefunden, mit der einige Jahre lang alles rund lief: Die Musiksammlung lag auf einem alten Laptop ohne Akku. Als Server-Programm nutzte ich den Windows-Media-Player. Ein HTC One S (noch mit Akku) unter Android 4.1.1 hing als Zuspieler am Verstärker und ein weiteres Handy lag als Steuergerät auf dem Tisch, weil ich es zu unbequem finde, meine Musik am Verstärker stehend auszusuchen. Auf beiden lief BubbleUPnP, das einzige Programm, das ich finden konnte, das alle drei Rollen im Open-Home-System einnehmen kann. Damit ein Musikwunsch zügig umgesetzt werden konnte, ohne auf den Start von vielen Geräten warten zu müssen, waren alle drei Geräte ständig an. Nur der Verstärker wurde extra zum Musikhören ein- und ausgeschaltet, weil dort der Löwenanteil der Leistung verbraten wird.

Das Ableben des Laptops, die geringere Brandgefahr sowie der Wunsch, Strom zu sparen, waren die Motivation für den Umstieg auf Smartphones ohne Akku. Zu Beginn sollte die Musiksammlung auf einen USB-Stick am Router umziehen, der separate Musik-Router in Rente geschickt und das Smartphone akkulos gemacht werden. Ich musste dann feststellen, dass der TP-Link 750 die Musiksammlung nur bis zum Buchstaben B bereitstellt. Typisch DLNA, alles ist zertifiziert, funktioniert aber trotzdem nicht in der Praxis. Deshalb habe ich die Musiksammlung am Schluss ebenfalls auf das Handy umgezogen.

Als Bastelobjekte boten sich alte Samsung Galaxy S3 an, die seit dem Wegfall des UMTS-Netzes ihre eigentliche Bestimmung nicht mehr erfüllen können. Von diesen Handys wurden laut Wikipedia mehr verkauft als alle Raspis bisher zusammengenommen. Der Flohmarktpreis liegt bei 10 Euro. Auf dem Display muss noch was zu erkennen sein, Lade- und Klinkenbuchse, Home- und Einschalt-Tasten, SD-Card-Slot und WLAN müssen funktionieren. Zum Flashen braucht man außerdem noch einen nicht komplett toten Akku und die Lauter-/Leiser-Tasten.

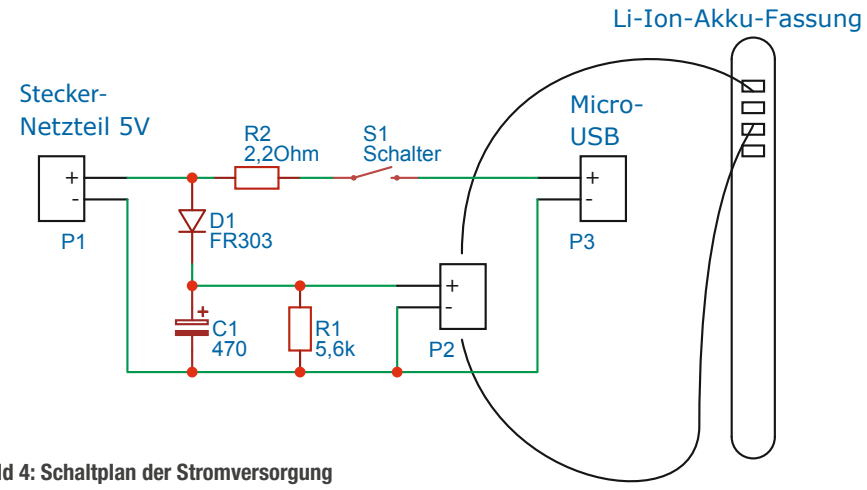


Bild 4: Schaltplan der Stromversorgung

Android-Konfiguration

Wenn wir dauerhaft etwas auf dem Display angezeigt bekommen wollen, ohne dass das Handy den Bildschirm ausschaltet, können wir in den Entwicklereinstellungen das Ausschalten des Displays verhindern. Entwickler werden wir durch siebenmaliges Tippen auf die Buildnummer (Bild 2, links) unter „Über das Telefon“ ganz unten. Danach taucht unter „System/Erweitert“ der neue Menüpunkt „Entwickleroptionen“ (Bild 2, Mitte) auf. Dort wählen wir ganz oben „Aktiv lassen“ (Bild 2, rechts).

OLED-Displays sind nicht besonders gut für den Dauerbetrieb mit statischen Inhalten geeignet, weil benutzte Pixel altern und sich schnell Einbrenn-Effekte zeigen. Deshalb und auch, um Strom zu sparen, setze ich das Theme auf „Dunkel“, entferne unnütze Icons vom Startbildschirm sowie aus der Statusleiste und lade einen rein schwarzen Bildschirmhintergrund entweder als Datei oder durch Aufnahme eines Fotos im Dunklen ohne Blitz. Die Bildschirmhelligkeit stellt man entweder ma-

nuell sehr dunkel ein oder belässt sie auf Automatik. Die Displaysperre schalte ich komplett aus. Ich habe zwar schon einige Displays mit starken Einbrennmustern gesehen, weiß aber nicht, wie lange ich dies mit diesen Maßnahmen aufhalten kann. Weil sich schwarze Bildschirme in diesem Heft schlecht drucken lassen, habe ich zur Dokumentation das helle Design gewählt.

Die weiteren Einstellungen des Systems sind eher Geschmackssache. Ich schalte alles ab, was Strom verbraucht: alle Funktechniken außer WLAN, alles, was nach Hause telefoniert, bimmelt, vibriert, nervt. Je weniger Programme und Funktionen aktiv sind, desto geringer ist die Chance, dass während der langen Betriebsdauer real nutzbare Sicherheitslücken gefunden werden.

Speichermedium

Meine MP3-Sammlung passt auf eine 256 GB große MicroSD-Karte (Bild 3), die es aktuell für rund 20 Euro zu kaufen gibt. Schaut man in die von Samsung gepflegte Liste, welche maximalen Kapazitäten die alten Handys unterstützen, wäre beim S3 bei 64 GB Schluss. Glücklicherweise ist die Liste nicht korrekt! Die einzige nützliche Information zum S3 ist, dass es damals mehr als 32 GB unterstützt hat. Daraus folgt, dass das SDXC-Format unterstützt wird. Der Vorgänger SDHC ging nur bis 32 GB. Während die Anschlüsse gleich geblieben sind, ist bei SDXC der „Ultra High Speed"-Modus (UHS) hinzugekommen. Es spricht also nichts dagegen, dass auch MicroSD-Karten mit laut Spezifikation möglichen 2 TByte unterstützt werden.

Meine Karte kam exFAT-formatiert. Nach dem Einstecken in das Gerät will Android die Karte formatieren, stürzt dabei allerdings reproduzierbar ab. Ich vermute, dass Lineage die Lizenzkosten für exFAT nicht an Microsoft zahlen möchte und stattdessen lieber ext4 (Linux-Dateisystem) verwendet. Prüfen konnte ich das nicht. Man könnte die Musiksammlung also über einen Linux-Rechner mit ext4 formatiert aufspielen. Oder man formatiert die MicroSD-Karte als Windows-Nutzer im NTFS-Format. Ich spiele also meine Musik auf die Karte und versenke sie in dem Gerät.

Die Stromversorgung

Erst nachdem alle Einstellungen im Betriebssystem getätigt und alle Programme, die auf dem Handy noch laufen sollen, installiert und konfiguriert sind, ersetze ich den Akku durch eine Netzstromversorgung. Als Stromquelle verwende ich ein USB-Netzteil mit 5 V und 1 A. Beim Handystart treten jetzt bei mir keine höheren Ströme mehr auf, sodass das Handy ordentlich hochfährt.

Den Strom (Bild 4) führe ich direkt über die Kontakte des Akkus zu. Die positive Betriebsspannung kommt an den außen liegenden Pin, Masse an den übernächsten Pin. Dazwischen liegt der Kapazitätsindikator des Akkus, ein Widerstand von 1 kΩ pro Ah nach Masse. Für den akkulosen Betrieb muss da nichts anliegen. Der innerste Pin kontaktiert den Temperatursensor des Akkus. Es gibt auch originale Akkus ohne Temperatursensor, deshalb wird der nicht zwingend benötigt.

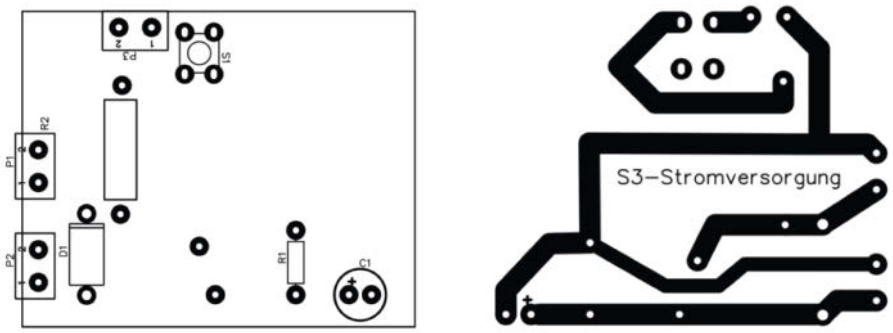


Bild 5: Bestückungsplan und Platinenlayout (Kupferseite) für das S3

Damit am Plus-Pin eine Spannung von etwa 4 bis 4,4 V anliegt, die das Handy erwartet, reduziere ich die 5 V des Netzteils über den Spannungsabfall an der Diode D1. Wenn beim Booten kurzzeitig 1 A fließt, sollten etwa 1 V über die Diode abfallen, die dann in 1 W Verlustwärme umgesetzt werden. Diese Verlustleistung muss die Diode aushalten. Der genaue Typ ist unkritisch. Ich nehme, was die Bastelkiste mir bietet, d. h., was ich in alten Schaltnetzteilen finde. Es sind p-n-Dioden, für 2 bis 5 A Dauerstrom geeignet, keine Schottky-Dioden, da diese einen zu geringen Spannungsabfall haben. Ein Blick ins Datenblatt verrät, welche Flussspannung bei 1 A, 25 °C zu erwarten ist. Bei einer 5-A-Diode ist das weniger als bei einer 2-A-Diode. Dementsprechend zeigt mir der Aufbau mit ersterer später immer einen zu 100 Prozent vollen Akku an, bei letzterer ermittelt das Handy einen Füllstand von 70 bis 100 Prozent und versucht gelegentlich mal nachzuladen.

Der Widerstand R1 lässt auch bei ausgeschaltetem Handy einen geringen Strom durch das Gerät fließen, damit sich die Spannung auch ohne Verbraucher einstellt. Der Elektrolytkondensator C1 glättet kurze Verbrauchsspitzen und Störungen auf der Zuleitung. Ich weiß nichts darüber, welche Spitzen in der Realität auftreten und wie viel davon das Netzteil abfedern kann. Deshalb entstammt der Wert allein dem Bauchgefühl. Weil normalerweise an Batterieklemmen die Spannung nicht extra durch einen Kondensator gestützt werden muss, fehlt er möglicherweise im Handy. Störimpulse, die die Zuleitung einfängt, würden dann nicht eliminiert werden.

Um zu verhindern, dass das Handy seinen Bildschirm ausschaltet, lege ich an den Micro-USB-Stecker eine Spannung von 5 V an und sage dem Handy in den Entwickler-Optionen, dass der Bildschirm beim Laden nicht abge-

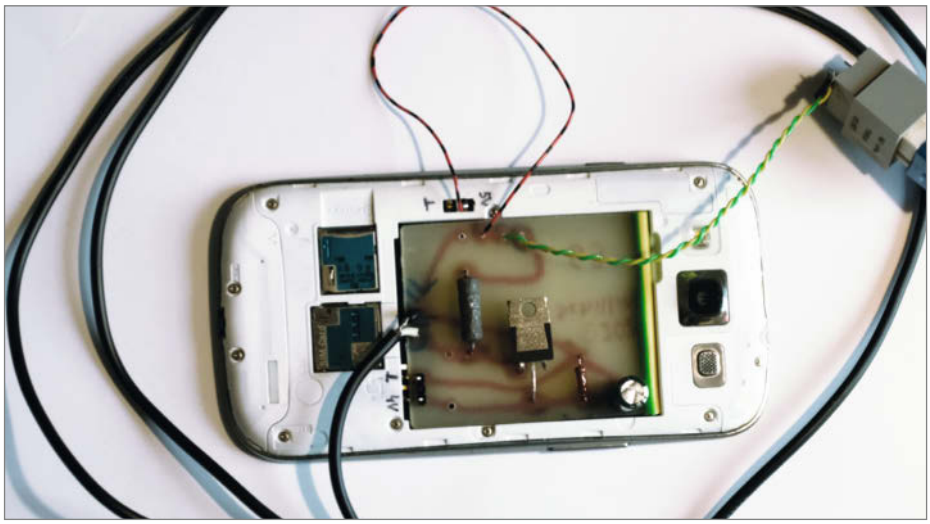


Bild 6: Der fertige Aufbau

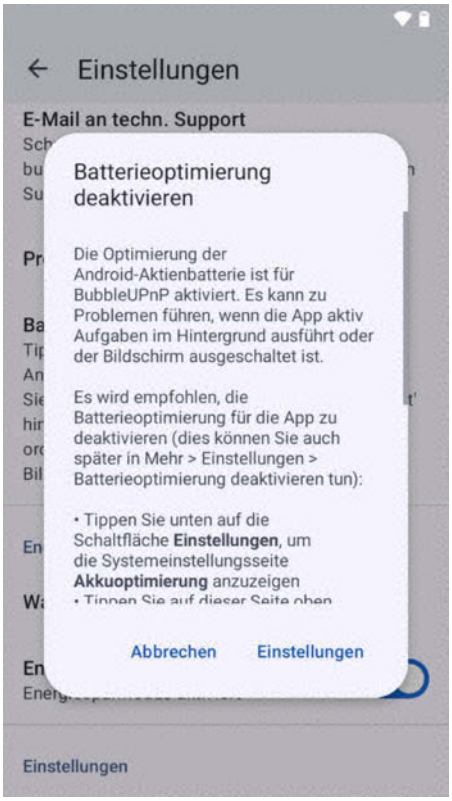


Bild 7: Batterieoptimierung einstellen

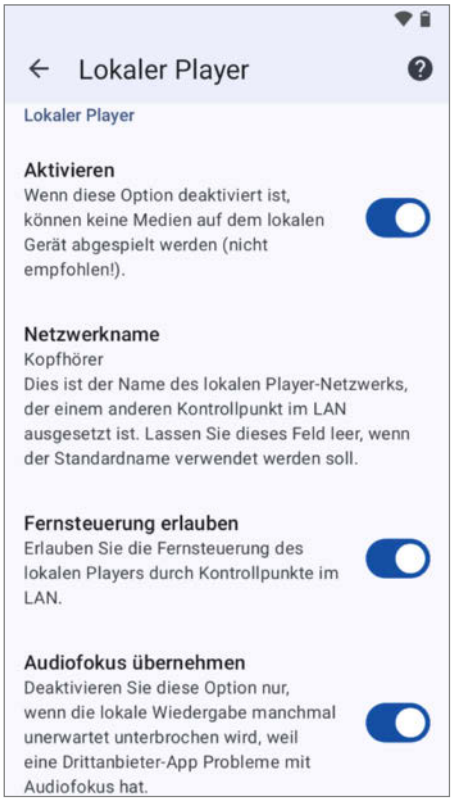


Bild 8: Lokalen Player aktivieren

schaltet werden darf. R2 hatte ich eingeführt, damit ich bestimmen kann, woher das Handy seinen Strom bezieht: also über die Akkukontakte und nicht über den Micro-USB-Anschluss. Leider hat sich herausgestellt, dass es für ein dauerhaft wach bleibendes Handy nicht ausreicht, hochohmig am USB-Stecker 5V anzubieten.

Sobald ich dort eine Stromquelle anschließe, testet das Handy die Stromergiebigkeit und nur, wenn die Quelle niederohmig genug ist, schaltet es in den dauerhaft aktiven Modus. Bei meinem S3 war das mit einem Widerstand von 2,2 Ω der Fall, ein doppelt so hoher Widerstand funktionierte nicht. Im schlimmsten Fehlerfall (Kurzschluss am USB-Stecker) verbraucht der Widerstand bei 1A maximalem Ausgangsstrom des Netzteils 2,2 Watt. Die Belastbarkeit sollte also größer als 2W sein.

Die Spitzen-Modelle S3 und S3 LTE waren für eine separat zu erwerbende Rückseite vorbereitet, mit der sich die Geräte drahtlos auf Qi-Ladestationen laden ließen. Statt am Micro-USB-Stecker lassen sich die 5V für Dauerbetrieb auch an diesem Kontakt zuführen. Bei nur zwei Kontakten ist der richtige der, der nicht mit dem Akku-Minus verbunden ist.

Auch andere der damaligen Spitzen-Smartphones von Samsung wie das S4, S5 und Alpha waren für eine Rückseite mit Lade-Antenne vorbereitet. Die haben teilweise bis zu fünf Kontakte. Der richtige Kontakt ist beim S4 der äußerst linke. Für andere Geräte kann man sich die Bilder von Qi-Rückseiten im Internet ansehen, diese besitzen nur die nötigen Kontakte.

Weil das Handy mit angeschlossener Stromversorgung manchmal nicht startet und man das in diesem Fall nicht einfach durch Abziehen des Steckers lösen kann, habe ich einen Schalter zur Unterbrechung dieser Verbindung vorgesehen. Den klebe ich einfach mit Heißkleber direkt neben die Klinkenbuchse an das Gehäuse.

Bei den späten Versionen wie S3 Neo ist die kabellose Ladeoption dem Rotstift zum Opfer gefallen. Man muss die also mit eingestecktem Micro-USB-Stecker betreiben oder man verzichtet auf ein dauerhaft eingeschaltetes Display.

Platinenlayout für Samsung S3

Für eine „Schaltung" mit nur vier Bauelementen entwerfe ich normalerweise keine Platine, sondern löte alles frei schwebend zusammen. Erst für den dritten Aufbau habe ich eine einlagige Platine (Bild 5), konkret für das S3 und seine Varianten, entworfen und mit der alten Kulturtechnik Federzeichnung mit Abdecklack realisiert. Mir steht der gesamte Raum im Batteriefach zur Verfügung, deshalb muss ich mit dem Platz nicht knausern, und die Bauelemente sind luftig angeordnet.

Die Platine bietet zwei Bestückungsvarianten für die Diode: entweder als axiale Version oder im TO-220-Gehäuse, wenn man von einer Doppel-Diode einen Strang verwenden möchte. Statt die Batteriekontakte direkt anzulöten, habe ich zwei Pfosten-Pins aufgelötet und so umgebogen, dass sie einen lösbaren Kontakt ergeben. Damit alles passt, habe ich die Lötstellen flach gehalten und die Anschlüsse der Bauelemente vor dem Verlöten flach umgebogen.

Die Platine (Bild 6) kommt in das Akkufach und die Kabel werden zur Zugentlastung mit Heißkleber an das Gehäuse geklebt. Ich habe mir erspart, die Rückseite mit Löchern zu versehen, sondern statt der originalen Rückseite alles mit Klebeband zugeklebt. Den SD-Karten-Slot habe ich ausgespart, weil die Karte einfach wechselbar bleiben soll.

BubbleUPnP

Das aktuelle Programm kann man im Google Play Store, aber auch im XDA-Forum herunterladen, falls man den Play Store nicht installieren will. Dann lässt sich allerdings die Lizenz nicht aktivieren und man muss mit Werbung und einigen Einschränkungen leben. Eine Lizenz für 5 Euro berechtigt zur Nutzung auf beliebig vielen Geräten mit demselben Konto oder den Geräten von fünf Mitgliedern eines Familienkontos.

Das Programm ist sehr mächtig und man sollte sich die Zeit nehmen, jeden der vielen Menüpunkte anzusehen und auf seine Bedürfnisse einzustellen. Hat man einmal das Optimum gefunden, kann man die Konfiguration mittels „Daten importieren/exportieren" und dann „Import von laufender BubbleUPnP-Instanz" direkt auf ein zweites Handy im gleichen WLAN übertragen.

Die wichtigste Einstellung (Bild 7) für den dauerhaften Betrieb ist „Batterieoptimierung deaktivieren". Der Eintrag führt zu der entsprechenden Funktion des Betriebssystems. Ziel ist, dass BubbleUPnP in der Liste „Nicht optimiert" auftaucht. Man muss also das Programm erst in der Liste „Alle Apps" suchen und dort den Menüpunkt „Nicht optimieren" wählen. Der nächste Menüpunkt „Energiesparmodus" wird für die Server-Handys ausgeschaltet. Unter „Läuft jetzt" gibt es noch den Eintrag „Bildschirm eingeschaltet lassen", auch der bleibt an.

In den Menüpunkten „Design" bis „Mediathek" nehme ich viele Änderungen an den Einstellungen vor, um die Anzeige in den einzelnen Tabs an meine Vorstellungen anzupassen. Das kann jeder für sich ausprobieren.

Ab dem Menüpunkt „Lokaler Player" (Bild 8) geht es ans Eingemachte: Der lokale Player muss aktiviert sein. Ich empfehle, einen aussagekräftigen Namen zu wählen. Nur mit den

voreingestellten Handynamen käme man schnell durcheinander, wenn man mehrere S3 in unterschiedlichen Rollen betreibt. Die Fernsteuerung muss man erlauben und die „Unterbrechungsfreie Wiedergabe" ebenfalls.

Viel weiter unten kommt der Menüpunkt „Open Home Player aktivieren ", der einen zweiten Netzwerknamen mit dem Anhängsel „(Open Home)" für die „Open Home"-Version des Players erzeugt. Alle Abspielziele sollten grundsätzlich immer diese Endung haben, damit man auch wirklich die Vorteile des „Open Home" nutzt.

Im Menü „Local und Cloud" wird der Inhalt der SD-Karte bereitgestellt: „Enable Local and Cloud Library" muss eingeschaltet sein. Unter „UPnP/DLNA media server settings" wird der Netzwerkname für die Mediathek auf diesem Handy festgelegt. „Im LAN melden" und „Remote-Browsing aktivieren" aktiviert man. In der Zeile „Erlaubte Medien für Remote Browsing" schalte ich die Android-Mediathek aus und den „Lok. Speicher/Bereitstellungspunkt" ein. Letzteren muss man erst erstellt haben, wie im nächsten Absatz beschrieben. Ich will direkt auf das freigegebene Verzeichnis zugreifen, ohne dass erst der Android Media Store die Daten indizieren muss. Ob der Weg über die Android-Mediathek vielleicht Geschwindigkeitsvorteile beim Einlesen der Musiksammlung bringt, habe ich bisher nicht ausprobiert. Alle Cloud-Inhalte schalte ich aus, weil ich sie nicht nutze.

Zurück im Menü „Local und Cloud" wird unter „Lokale Inhalte" der „freizugebende Lok. Speicher/Bereitstellungspunkt" gewählt. Dort wähle ich „Aktivieren" und „Versteckte Ordner/Dateien ausblenden". Unter Bereitstellungspunkt 1 wählt man den freizugebenden Ordner auf der SD-Karte aus und gewährt dem Programm die Berechtigung zum Zugriff darauf. Auch hier vergibt man einen aussagekräftigen Titel. Alle Webdienste schalte ich einzeln aus, damit sie mir später meine Musikauswahlliste nicht überfrachten.

In „Gerätesichtbarkeit konfigurieren" (Bild 9) blende ich über das Dreipunkte-Menü alle Player aus, die kein „(Open Home)" am Ende haben, damit ich später nur die richtigen Player zur Auswahl angezeigt bekomme. Die Liste zeigt nur vier Einträge an, ist aber scrollbar.

Der nächste Punkt, an dem ich drehe, ist auf der Seite „Steuerung". „Beim Start auf Player warten" erspart die Player-Auswahl, wenn man nur auf einem Verstärker abspielt. „Beim Hochfahren starten" ist nur in der bezahlten Version verfügbar und startet BubbleUPnP beim Handystart gleich mit. Der Schließen-Knopf ist vorteilhaft, weil man nach jeder Änderung von Netzwerknamen BubbleUPnP neu starten soll, damit die Änderungen wirksam werden. „Bei Inaktivität beenden" ist eine weitere Option, die sich auf den Dauerbetrieb auswirkt. Sie muss deaktiviert werden. Alle nicht benötigten Geräteunterstützungen sollte man ausschalten, um die Angriffsfläche gering zu halten.

Uff, fast geschafft! Nun noch im Player-Tab (nicht in den Einstellungen) über das Dreipunkte-Menü den Equalizer aufrufen und AudioFX ausschalten, weil wir ja keinen schmalbrüstigen Ohrhörer an der Klinkenbuchse betreiben, sondern einen ausgewachsenen Verstärker. Online haben wir eine komplette Bilderstrecke angelegt, die den Vorgang nochmals illustriert; den Link finden Sie über den Kurzinfo-Link.

Steuerungshandy

Auf das Handy in der Tasche (den Kontrollpunkt) kopiere ich die Einstellungen vom Server-Handy, schalte den lokalen Player aus und lasse den Punkt „Enable Local and Cloud Library" eingeschaltet, deaktiviere aber alle Bereitstellungspunkte und Webservices außer UPnP/DLNA-Mediatheken.

In umfangreicheren Umgebungen ist es wieder wichtig, in „Gerätesichtbarkeit konfigurieren" alle nicht „Open Home"-Player auszublenden.

Alle Batteriesparmechanismen kann man hier zulassen. Nach der Auswahl des Albums hat das Handy seine Schuldigkeit getan und kann wieder in die Hosentasche wandern. Wenn man später sehen will, was aktuell läuft, ist die App schnell wieder gestartet und nach kurzer Synchronisierung sieht man im Player-Tab den gerade gespielten Titel.

Audioqualität

Bisher ist mir noch kein Smartphone-Testbericht untergekommen, der auf mehr als das Vorhandensein einer Klinkenbuchse eingeht. Angaben zur Audioqualität, also Frequenzgang, Jitter, Klirrfaktor, Rauschabstand, Netzbrumm und Pegel habe ich noch nirgends gelesen.

Zum Test des Rauschabstands habe ich mir mit Audacity in eine leere Datei (Download über den Kurz-Link) zuerst 30 Sekunden Stille und danach in die Stille kurze Abschnitte von Rauschen in absteigender Intensität eingefügt. Die ersten –60 dB (Faktor 0,001) sind zum Einstellen des Lautstärkereglers; mit –80 dB (Faktor 0,0001) und –86 dB (Faktor 0,00005) überprüfe ich, ob Störgeräusche dieses Rauschen verdecken, und die –100 dB (Faktor 0,00001) testen, ob man an Wahnvorstellungen leidet. Der MP3-Standard kann nur –86 dB liefern. Bei meinen umgebauten S3-Smartphones war alles in Ordnung. Das –86-dB-Rauschen ist klar zu hören und keine anderen Geräusche trüben den Klang. Das ist nicht selbstverständlich, weil ja der Akku zum Glätten der Versorgungsspannung nicht mehr zur Verfügung steht und es durchaus möglich wäre, dass Störungen vom 5-V-Netzteil auf den Ton durchschlagen.

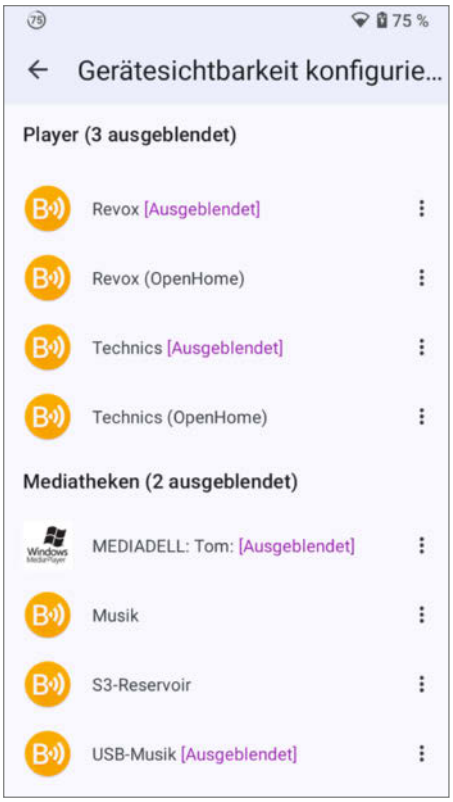


Bild 9: Gerätesichtbarkeit

Für andere Tests muss ich mich auf meine Ohren verlasse, weil mir kein spezielles Equipment zur Verfügung steht. Der Ausgangspegel ist relativ gering, was ich durch voll aufgedrehte Pre-Amp-Gain-Regler am Verstärker ausgleiche. Ansonsten klingt die Musik so, wie ich es von direkt angeschlossenen kommerziellen MP3-Abspielern gewohnt bin.

Meine Konfiguration

Aktuell bespiele ich mit zwei S3-Handys je einen Verstärker im Wohnzimmer und in der Küche und habe noch ein weiteres für einen Kopfhörer eingerichtet. In allen Handys stecken 256 GByte oder 512 GByte große SD-Karten mit Musiksammlungen. Die gleichzeitige Beschallung beider Räume mit Musik vom jeweils anderen Handy funktioniert problemlos. Bis sich die erste Verzeichnisebene mit den Bandnamen einer 256 GByte großen Musiksammlung auf einem entfernten Handy öffnet, braucht es 35 Sekunden. Das ist für mich noch zu verschmerzen.

Im Dauerbetrieb benötigt ein so eingerichtetes Smartphone etwa 1W. Bei 40 Cent/kWh kostet der Spaß etwa 3,50 Euro pro Jahr. Natürlich kann man das Display auch noch für weitere Anzeigen nutzen, wenn es schon mal an ist. Etwa den Wetterbericht oder die Uhrzeit.

So, nun viel Spaß beim Basteln und entspannten Musikhören! —*caw*

Festnetzanschluss im Smartphone nutzen

Mit wenigen Schritten können wir unser Smartphone über einen Festnetzanschluss nutzen. Das bringt nicht nur Kostenvorteile für die anrufende Verwandtschaft, sondern spart besonders im Ausland Gebühren.

von Rainer Maria Kreten

Die gute alte Festnetznummer hat noch lange nicht ausgedient. Sie ist von anderen Festnetzanschlüssen aus normalerweise zum Nulltarif erreichbar, lässt sich geografisch zuordnen und drückt damit auch die regionale Bindung aus. Um einen Festnetzanschluss auf dem Smartphone nutzen zu können, brauchen wir eine App, die VoIP-Funktionalität (Voice over IP) bereitstellt. Bei einigen Smartphones sind VoIP-Clients vorinstalliert, mit unterschiedlichen Funktionen und zuweilen nur trickreich zu nutzen. Nervenschonender ist es, auf eine allgemein verständliche und auf Open Source basierende Lösung zu setzen. Die Software Linphone (Linux phone) aus Frankreich lässt kaum Wünsche offen und läuft neben Android auch auf iOS und Desktop-Rechnern.

Kurzinfo

- **Heimische Festnetzanschlüsse mobil nutzen**
- **VoIP-App Linphone konfigurieren**
- **VPN einrichten**

Checkliste

Zeitaufwand:
2 bis 4 Stunden

Software

- **Linphone**
- **OpenVPN**

Mehr zum Thema

- Elke Schick, Mobiltelefone und Mobilfunk, Make 5/20, S. 96
- Matthias Wendt, 100 Jahre Rundfunk – von Hertz bis Gigahertz, Make 4/19, S. 8

Alles zum Artikel im Web unter *make-magazin.de/xc33*

Mit Linphone loslegen

Die Einrichtung von Linphone ist sehr einfach, da wenige Parameter genügen, um die App zu beleben. Wer einen Standardanschluss der Telekom hat, muss nur die Telefonnummer, das Kundenpasswort und die Serveradresse eingeben (Bild 1). Danach lassen sich mehrere sogenannte SIP-Konten (Session Initiation Protocol) einrichten, die gleichzeitig auf Anrufe warten. Ein SIP-Account ist eine Art Benutzerkonto, das für die Einrichtung von VoIP verwendet wird.

Diese lassen sich komfortabel einzeln aktiv und (z. B. am Feierabend) inaktiv schalten. Der etwas missverständliche Schiebeschalter „Als Vorgabe verwendet" legt eines der Konten für abgehende Anrufe fest (Bild 2). Musterkonfigurationen für SIP-Accounts sind in der App vorhanden. Alle Eingaben sind in der Tabelle „Konfiguration von Linphone" zusammengefasst.

Damit Linphone funktioniert, muss es natürlich im Hintergrund laufen und auf Anrufe warten. Einen ankommenden Anruf muss das Programm kundtun und dazu bei der Installation oder beim Einrichten die Berechtigung bekommen, uns benachrichtigen zu dürfen. Der Hinweis dazu findet sich in dem Benachrichtigungsbereich, den man erreicht, wenn man von ganz oben nach unten wischt. Stoppt man irrtümlich den Dienst, kommen keine Anrufe mehr an.

Linphone bringt einen eigenen Klingelton mit, der als Standard eingestellt bleiben sollte. Ist auf dem Gerät noch ein normaler GSM-Telefonclient aktiv, so kann man die Anrufe auf die Handynummer sonst nicht von VoIP-Calls unterscheiden und tippt beim Klingeln unter Umständen auf das falsche Icon.

Nomadische Nutzung

Die Anschlüsse der Telekom sind nur über das eigene Heimnetz verwendbar. Richtet man seine Telefonnummer auf dem Smartphone ein, spart man sich zu Hause ein separates Schnurlostelefon. Das schöne alte Ding auf dem Aufmacherfoto könnt ihr dann entsorgen. Andere Betreiber bieten Anschlüsse an, die von jedem Internetanschluss in der Welt aus genutzt werden können. Der Fachausdruck dafür ist „Nomadische Nutzung". Bei Firmen, die SIP-Anschlüsse losgelöst vom Internetanschluss anbieten, muss man natürlich ein Abo abschließen.

Die rechtlichen Rahmenbedingungen in Deutschland verpflichten Telefonanbieter, die Identität ihrer Kunden zweifelsfrei zu prüfen und Rufnummern aus dem Vorwahlbereich des Hauptwohnsitzes zu vergeben. Der damit verbundene Verwaltungsaufwand macht Privatkunden zunehmend unattraktiv für die Unternehmen, weshalb viele bekannte Anbieter und einstige Pioniere wie Sipgate sich mittlerweile auf den Geschäftskundenbereich

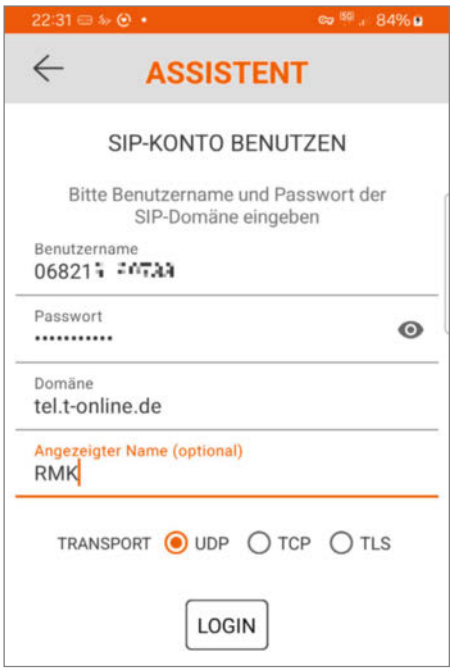


Bild 1: Mit Passwort ist das Kundenpasswort der Deutschen Telekom gemeint.

Konfiguration von Linphone

Parameter	Deutsche Telekom	Sipgate Basic
Benutzername	Telefonnummer mit Vorwahl z.B. 06819876543	Vergebener Benutzername (nicht Telefonnummer)
AuthID	bleibt leer	bleibt leer
Passwort	das allgemeine Passwort für Kundencenter etc.	vergebenes Passwort
Domäne	tel.t-online.de	sipgate.de
Anzeigename	z. B. 06819876543	z. B. 06819876543
Verwaltung Deaktivieren	nein	nein
als Vorgabe verwendet	bei mehreren Konten Vorgabe für abgehende Rufe	es kann nur eine geben
Transport	UDP	UDP
SIP-Proxy	Sip:tel.t-online.de;transport=udp	Sip:sipgate.de;transport=UDP
läuft ab nach	3600 Sekunden	3600 Sekunden
Bündelmodus deaktivieren	ja	ja

Bild 2: Musterkonfigurationen für SIP-Konten sind vorhanden.

VoIP-Anbieter für Deutschland

Fonial	drei Nummern, ohne Grundgebühr, Gesprächsminute mit 2,26 Cent relativ hoch
EasyBell	zwei Rufnummern, Grundgebühr 99 Cent/Monat, Minutentarife für Privatkunden auf der Website nicht ersichtlich
VOIP2GSM	flexibel auch für Privatanwender mit einer Rufnummer
MEGAvoip	keine deutschsprachige Website, Parameter vor der Anmeldung nicht einsehbar
NeXXt Mobile	privat nutzbare Einzelrufnummer
simply Connect	ankommende Gespräche verursachen Gebühren
sipload	15 Euro Jahresgebühr
dus.net	ohne Grundgebühr, zwei Rufnummern, geringe Gesprächskosten
ClouFON	ohne Grundgebühr, geringe Gesprächskosten

konzentrieren, Altverträge aber weiter bedienen. Eine Übersicht über verschiedene Anbieter in Deutschland – ohne Anspruch auf Vollständigkeit – findet ihr in der Tabelle „VoIP-Anbieter für Deutschland".

Da wir aber fast alle über SIP-Accounts unserer Internetprovider verfügen, konzentrieren wir uns darauf, diese mobil zu nutzen. Das Zauberwort dafür lautet VPN. Internetrouter wie Fritzbox & Co. verfügen über zwei Funktionen, die wir dafür benötigen: zum einen die Einrichtung eines dynamischen DNS (DDNS), über den der Router von außen über eine statische URL angesprochen werden kann; zum anderen einen VPN-Client.

DDNS einrichten – eine Voraussetzung

Die meisten unserer DSL- und Glasfaseranschlüsse werden durch den Internetanbieter in den frühen Morgenstunden kurz vom Netz getrennt und starten dann mit einer neu vergebenen IP in den Tag. Wollen wir von außen auf unser Heimnetz zugreifen, so müssten wir daher eigentlich allmorgendlich im Router nach der aktuellen IP schauen und diese dann verwenden.

Viel eleganter ist es, das einem DDNS-Dienstleister zu überlassen, von dem man eine dauerhafte Internetadresse bekommt. Der Router meldet regelmäßig seine IP an den Dienstleister und dieser sorgt dafür, dass diese an die Nameserver dieser Welt weitergereicht wird. Egal wie sich die IP ändert, wir bleiben immer unter der gleichen URL erreichbar. Die großen Marken für Internetrouter haben bereits Profile für DDNS-Dienstleister vorinstalliert oder sie bieten diesen Service gleich selbst an.

Einen DDNS zu nutzen, eröffnet auch bei anderen Projekten viele Möglichkeiten bis hin zum Betreiben eines eigenen Webservers. Natürlich ist das auch eine Einladung für Leute, die nicht unbedingt höflich nachfragen, ob sie sich in unserer digitalen Privatsphäre tummeln dürfen. Das heimische Netz für Datenverkehr von außen zu öffnen, stellt immer ein Sicherheitsrisiko dar. Daher ergibt es Sinn, den eingehenden Datenverkehr grundsätzlich über einen VPN-Tunnel laufen zu lassen.

VPN – daheim ist überall

Mit einem VPN können wir mobil so arbeiten, als wären wir zu Hause im WLAN oder auch in der Firma mit dem Netz verbunden. Auf dem Weg durch das Internet sind die Daten verschlüsselt. Ist das VPN gestartet, gehen alle Daten diesen Weg über den heimischen Router, das Homebanking aus dem Hotel genauso wie der Streamingdienst, der via Tunnel klaglos seine Inhalte auch in fremde Länder liefert.

Und natürlich „merkt" der VoIP-Dienst nicht, dass er von ganz woanders aus genutzt wird. Gegenüber anderen Lösungen wie Proxy-Servern und Port Forwarding hat VPN zudem noch einen weiteren Vorteil. Die einzelnen Programme brauchen nicht speziell darüber verbunden oder konfiguriert zu werden. Der komplette Netzverkehr läuft automatisch durch den Tunnel.

Welche VPN-Software wir nutzen, hängt davon ab, was auf dem jeweiligen Router implementiert ist und wofür auch ein Client für das Smartphone existiert. In unterschiedlichen Umgebungen nachbausicher soll die Lösung auch sein und aktuellen Sicherheitsstandards genügen. Schließlich muss die Technik auch bei mobiler Nutzung funktionieren, bei der Verbindungsabbrüche häufig vorkommen.

Die Wahl fiel auf OpenVPN. Es wird ständig weiterentwickelt, die letzte Version für Android ist 3.4.2 vom Juli 2024 und es läuft auf einer Vielzahl an Plattformen, darunter die Fritzbox und das Router-Betriebssystem OpenWRT. Bei der Einrichtung auf dem Router wird dort eine etwa 6 KB große Schlüsseldatei mit der Endung OVPN generiert. Diese speichern wir auf dem mobilen Gerät.

Nach der Installation der Client-Software sind es nur noch wenige Schritte zum Ziel:

- Über das „Hamburger"-Menü und den Menüpunkt „Import Profile" wird die eben erzeugte Datei importiert. Danach am besten vom Mobilgerät löschen, denn es handelt sich um unseren digitalen Haustürschlüssel.

Im Menüpunkt „Settings" (Bild 3) müssen folgende Einstellungen vorgenommen werden:

- Als „VPN-Protokoll" ist „UDP" sinnvoll. Dort ist der Anteil an Kontroll- und Steuerdaten am geringsten. Deren Aufgabe wird ohnehin durch das Programm auf einer höheren Ebene erledigt. Der Eintrag muss deckungsgleich mit der Konfiguration des Routers sein.
- „Connection Timeout" stellen wir auf „continously retry" ein. Die Verbindung wird so nach einer Unterbrechung der Internetverbindung wieder aufgebaut.
- „Battery Saver" muss deaktiviert bleiben, denn natürlich darf OpenVPN nicht in den Ruhemodus gehen, wenn der Bildschirm des Endgerätes abdunkelt.

Alle anderen Einstellungen bleiben auf den Vorgabewerten.

Sollte eine Verbindung nicht zustande kommen, weil die Softwareversion des Routers die neuesten Protokollversionen noch nicht unterstützt, so lässt sich das in den „Advanced settings" durch Wechsel des Sicherheitslevels auf „Legacy" oder gar „Insecure" umgehen.

Das war bei mir notwendig, weil der hier eingesetzte Router aus dem Hause TPLink nicht die TLS-Version 1.3. beherrscht. Der Hersteller bietet zwar eine Beta-Firmware an, mit den üblichen Hinweisen auf potenzielle Fehler, das offizielle Update lässt indes auf sich warten.

Erfahrungen im Betrieb

Seit mehreren Monaten läuft die mobile Nutzung der Festnetzanschlüsse sehr zuverlässig. Nur sehr selten kam trotz passabler mobiler Datenverbindung beim Rufaufbau die Meldung „Dienst derzeit nicht verfügbar“. Auf meiner Urlaubsreise durch Deutschland, Österreich und Frankreich konnte ich das System ausgiebig testen und das Projekt ausreizen.

Es ist auch möglich, mehrere Smartphones auf den gleichen Account gleichzeitig zu registrieren. Bei ankommenden Anrufen klingeln dann alle Endgeräte und wer als Erster den Ruf annimmt, kann das Gespräch führen. Konferenzgespräche sind damit aber nicht möglich.

Auch ein Handover funktioniert. Gespräche werden bei einem Wechsel der Funkzelle nur ein bis zwei Sekunden stumm und können dann ohne neue Anwahl weitergeführt werden.

Bei einem nomadisch nutzbaren SIP-Account ergibt es ebenfalls Sinn, das VPN zu aktivieren. Unterbrechungen durch eine geänderte IP-Nummer beim Wechsel zwischen mobilen Daten und örtlichem WLAN werden so vermieden.

Bei längeren Wanderungen durch funkschattige Täler gibt OpenVPN trotz anders lautender Konfiguration irgendwann auf, die Verbindung nach Rückkehr in den funkversorgten Bereich wieder herzustellen. Man muss also in solchen Situationen die Verbindung kontrollieren und ggf. manuell wieder einschalten.

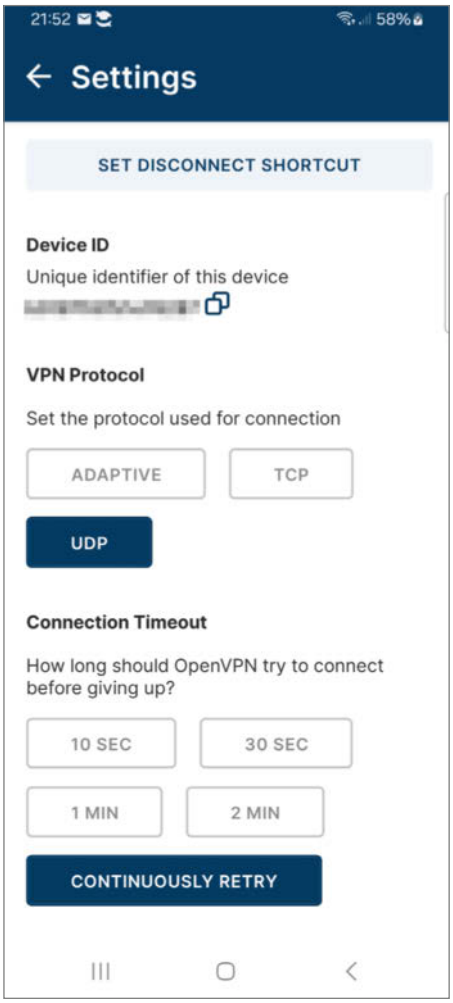


Bild 3: Die Einrichtung von OpenVPN bietet wenige Überraschungen

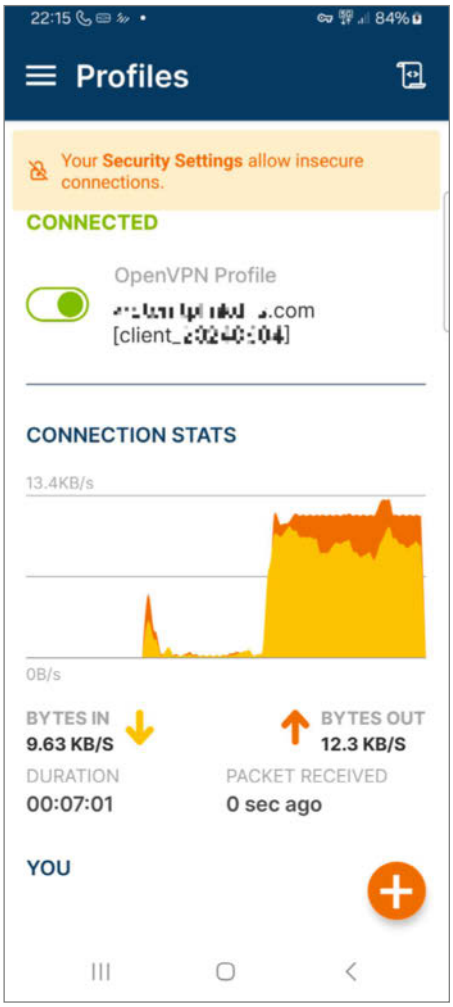


Bild 4: Der Datenverbrauch hält sich mit ca. 12 Kbit/s in beide Richtungen in Grenzen.

Geringer Datenverbrauch

OpenVPN funktioniert nicht, wenn man mit dem heimischen WLAN verbunden ist. Das wäre ja auch sinnfrei. Aber statt im Hintergrund zu warten, gibt es endlose Verbindungsversuche. Die Daten laufen aber dennoch sauber zwischen Smartphone und Router hin und her. Wünschenswert wäre hier, eine Blacklist von WLAN-Netzen zu haben, sodass das Smartphone die (nicht benötigte) VPN-Verbindung deaktiviert, sobald man zu Hause oder bei der Arbeit ist.

Der Datenverbrauch beim Telefonieren ist relativ gering. Wie die grafische Anzeige in OpenVPN zeigt, werden beim Telefonieren etwa 12 Kbit/s (jeweils zum Senden und Empfangen) benötigt (Bild 4). Der Standby-Verbrauch fällt kaum ins Gewicht. Hier waren es knapp 70 MB in vier Monaten.

Da Linphone das reguläre Telefonbuch des Smartphones nutzt, müssen wir die Nummern so abspeichern, wie man sie zu Hause auch nutzt, in Deutschland also ohne +49 am Anfang und z. B. in Frankreich mit 0033.

Da der klassische GSM-Telefon-Client das gleiche Telefonbuch nutzt, müssen dessen Einstellungen ggf. angepasst werden. Der Hersteller Samsung bietet diese Möglichkeit. Hier ist also noch zu prüfen, wie sich das eigene Gerät verhält.

Nachdem man Änderungen an den Einstellungen in Linphone gemacht hat, sollte das Programm zur Sicherheit über den Menüpunkt „Beenden“ verlassen und neu gestartet werden. Damit erzwingt man eine Neu-Registrierung beim SIP-Anbieter.

Notrufe sollte man über diesen Weg nur absetzen, wenn man zu Hause ist oder die lokale Festnetznummer des benötigten Hilfsdienstes wählt. Dem Disponenten in der Leitstelle zu erklären, dass seine Technik zur Geolokalisierung des Notfalls gerade in die Irre führt, könnte zu Missverständnissen führen.

Fazit

Den größten Nutzen unserer mobil genutzten Festnetzanschlüsse sehen wir im sozialen Umfeld. Für Freunde und Familie gibt es eine zentrale Rufnummer statt mehrerer Mobilfunkanschlüsse. Die Anrufe sind in der Regel kostenlos. Auch wenn wir von unserem Zweitwohnsitz im Ausland anrufen, erscheint die bekannte Nummer.

Zusammenfassend ergibt sich ein interessantes Bild der modernen Telekommunikation und der Wiederverwertung von Technologie: Ein Smartphone, das vielleicht schon in den Ruhestand geschickt wurde, kann zu Hause über das WLAN und eine entsprechende VoIP-App effektiv als schnurloses Telefon wiederverwendet werden. Durch die Registrierung bei einem freien SIP-Anbieter kann man zudem die Vorteile des weltweiten Telefonierens zu moderaten Kosten genießen.

Eine weitere Möglichkeit bietet die Nutzung eines VPN-Tunnels, durch den man sogar über den heimischen Festnetzanschluss telefonieren kann. Dies ermöglicht es Anrufern, uns über unsere heimische Festnetznummer zu erreichen, unabhängig davon, wo wir uns gerade aufhalten. Ebenso erlaubt die Technologie, multiple Smartphones simultan unter einer SIP-Nummer zu registrieren, was die Wiederbelebung der traditionellen gemeinsamen Familien-Telefonnummer in einem modernen Kontext erlaubt. *—mch*

Tipps & Tricks

Zur Inspiration und zum Nachbasteln hier im Stile unserer Tipps-&-Tricks-Rubrik ein paar kurze Beiträge unserer Autoren und Leser. Diese geben Inspiration, was sonst noch so mit Smartphones und Tablets machbar ist, die für die Schublade einfach zu schade sind.

von Michael Gaus, Miguel Köhnlein, Carsten Romahn, Paul Srna, Alexander Wankerl und Jörg Wölber

Smartphone als Anzeige für OpenHAB

Für die Hausautomation gibt es zahlreiche gute und umfangreiche Open-Source-Softwarelösungen, die lokal gehostet werden können und auf vielen verschiedenen Plattformen laufen. Eine dieser Lösungen, die sich für mich bewährt hat, ist OpenHAB. Neben den üblichen Automatisierungsaufgaben nutze ich einen Raspberry Pi, um Daten meiner PV-Anlage (Photovoltaik), Wärmepumpe und meines Stromzählers via RS485-Bus abzufragen und über MQTT an den OpenHAB-Server zu senden.

Um festzustellen, ob die PV-Anlage gerade genug Strom liefert oder ob die Batterie ausreichend aufgeladen ist, um Geräte wie die Spülmaschine oder den Wäschetrockner mit Sonnenstrom zu betreiben, ist eine stationäre Anzeige praktisch. Diese sollte leicht ablesbar sein, beispielsweise im Vorbeigehen. Es gibt unzählige Möglichkeiten für solch eine Anzeige, von einem kompletten Eigenbau mit einem ESP32 und Display bis hin zu kreativen Lösungen wie dem Umbau einer IKEA-Leuchte (Obegränsad, Frekvens). Ich habe mich jedoch dafür entschieden, mein altes Smartphone wiederzuverwenden.

Das Google Pixel 3a (Bild 1) lag mit einer defekten Kamera schon länger ungenutzt in der Schublade. Da OpenHAB eine eigene App anbietet, die es ermöglicht, die Daten direkt und ohne Browser anzuzeigen, war das Smartphone die ideale Wahl. Die App bietet zudem Optionen wie das Deaktivieren der automatischen Bildschirmsperre und die Darstellung im Vollbildmodus. Mit der passenden Dockingstation wurde aus dem alten Smartphone eine ansprechende, stationäre Anzeige.

Ein weiterer Vorteil: Das Pixel 3a unterstützt „Akku-schonendes Laden“ (Bild 2), wodurch der Akku nur zu 70 bis 80 Prozent aufgeladen wird. Dies verlängert die Lebensdauer des Akkus und ermöglicht bei Bedarf weiterhin einen gelegentlichen mobilen Einsatz.

Carsten Romahn

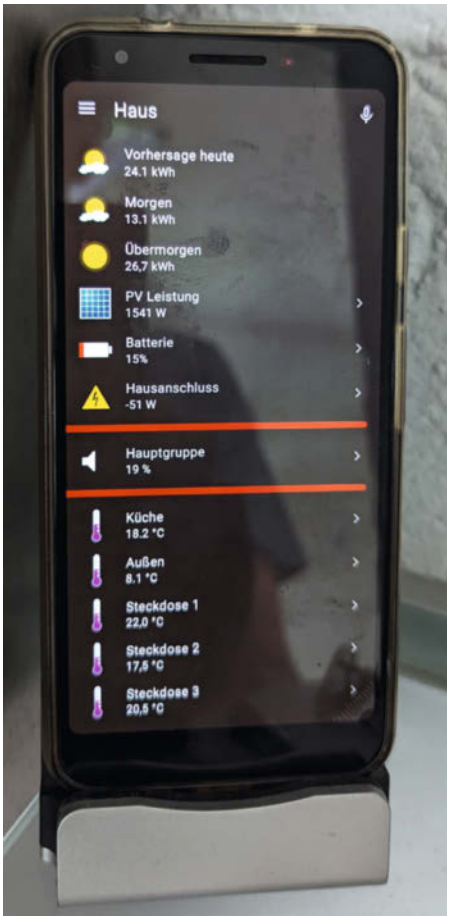


Bild 1: Smartes Display mit den wichtigsten Daten

Bild 2: Einstellung für den Akku

Android-Tablet als Navi

Wenn die Augen nicht mehr ganz so wollen wie früher und daher „Maps“ am Smartphone zu klein wird, ist es Zeit für ein größeres Display. Wenn dann noch dazu kommt, dass es für das große, alte Navi keine Updates mehr gibt und die Halterung aus Altersschwäche ihren Dienst aufgibt, dann ist es Zeit, an einen Ersatz zu denken.

In meinem Fall hat es sich so ergeben, dass bei unseren – inzwischen erwachsenen Kindern – ein Tablet zur weiteren Verwendung durch „die Oldies“ frei wurde, weil der Akku den Ansprüchen an die Dauer der mobilen Nutzung einfach nicht mehr entsprochen hat und auch die Größe des Bildschirms uncool geworden war.

So konnte ich – zugegeben mit Nachhilfe in puncto Apps – das altgediente Navigationsgerät durch ein etwa gleich großes Android-Tablet ersetzen. Wobei mir der Umstieg von den vertrauten Mac-Anwendungen auf Google / Android von unseren „Digital Natives“ leichter gemacht wurde.

Aber zurück zum realisierten Projekt: Die Installation und Verwendung von Google-Maps in der „Android Go Edition" auf dem Lenovo TB-7104I war selbst für mich alten Mac-User kein Problem, auch wenn ich mich an die etwas längeren Lade- und Reaktionszeiten erst gewöhnen musste. Das Display mit knapp 18 cm Diagonale (7 Zoll) entsprach dem in die Jahre gekommenen Navi „Moov Spirit V735", für das keine aktuellen Karten und Updates mehr zu bekommen waren. Auf den DVB-T Tuner im Moov konnte ich leichten Herzens verzichten. Außerdem war das Lenovo etwas schlanker, was sich dann allerdings später als Problem herausstellen sollte. Das größere Display (Bild 3) erleichtert natürlich die Lesbarkeit – auch wenn ich vorwiegend nach Sprachansage fahre.

Ursprünglich wollte ich die stabile und praktische Halterung des Moov gern weiterverwenden, da sie in einem weiten Bereich verstellbar und mit einem Saugfuß leicht zu befestigen war. Allerdings waren alle Reparaturversuche der Saugplatte erfolglos oder nur von kurzer Dauer. Daher habe ich mich dann doch dazu entschlossen, eine neue Halterung (Bild 4) in Fusion 360 zu konstruieren und 3D zu drucken.

Aus dem Projekt haben sich dann ein paar weitere Probleme ergeben, aber auch zusätzliche Tipps für eventuelle Nachbauten:

- PLA ist für die Halterung nicht gut geeignet, vor allem, wenn die Halterung aus schwarzem PLA und in einem schwarzen Auto ist, das im Sommer in der Sonne steht. Da sind Temperaturen um die 70 °C leicht möglich!
- Weißes PET-G schaut zwar weniger cool aus und kann sich in der Frontscheibe spiegeln, ist aber besser geeignet.
- Sonneneinstrahlung kann auch schwarze Tablets auf (für den Akku) gefährliche Temperaturen bringen – Abkleben der Rückseite mit Alu-Klebeband hilft sehr gut.
- Ein USB-Adapter für den Zigarettenanzünder versorgt auch Geräte mit schon schwachem Akku verlässlich mit Strom während der Fahrt.
- Ein Smartphone kann als WLAN-Hotspot aktuelle (Verkehrs-)Informationen an das Tablet-Navi liefern.
- Zu Hause kann man alternativ die geplanten Routen in der Maps-Anwendung speichern; das ist zwar etwas weniger genau, aber für die meisten Fälle ausreichend.

Paul Srna

Tablet als WLAN-Display

Selten findet man auf dem Markt die eierlegende Wollmilchsau, eigentlich eher nie. Für meine Hausautomation suchte ich eine Wetterstation mit Webserver, von dem die Daten geliefert und leicht durch einfache Text-Parser ausgelesen werden können. Leider sind solche Systeme auf dem Markt Mangelware; die meisten übermitteln ihre Daten an Cloud-Dienste, welche die Daten verarbeiten und bereitstellen. Letztlich fiel meine Wahl auf die Renkforce WH2600 Wetterstation, die diese Merkmale erfüllt. Allerdings verfügt sie nicht über ein Display, auf dem die Werte klar und einfach dargestellt werden.

Mehrere Anläufe, auf den Webserver der WH2600 zuzugreifen, mit ESP via Text-Parser (zu Ressourcen beanspruchend) und Mega 2560 mit ESP-01-WLAN-Modul (MQTT Protokoll, Umweg über Homeserver) funktionierten nur teilweise oder waren unzuverlässig. Daher war eine andere Lösung notwendig. Zunächst wollte ich einen Raspberry Pi mit einem entsprechenden Display kombinieren. Da fiel mir das ASUS 7-Zoll-Tablet ein, das ich als „Refurbished" für schmales Geld gekauft hatte. Damit musste so etwas doch auch möglich sein.

Anstatt wieder den Umweg mittels Text-Parser oder MQTT-Protokoll zu gehen, greift

Bild 3: On the road again

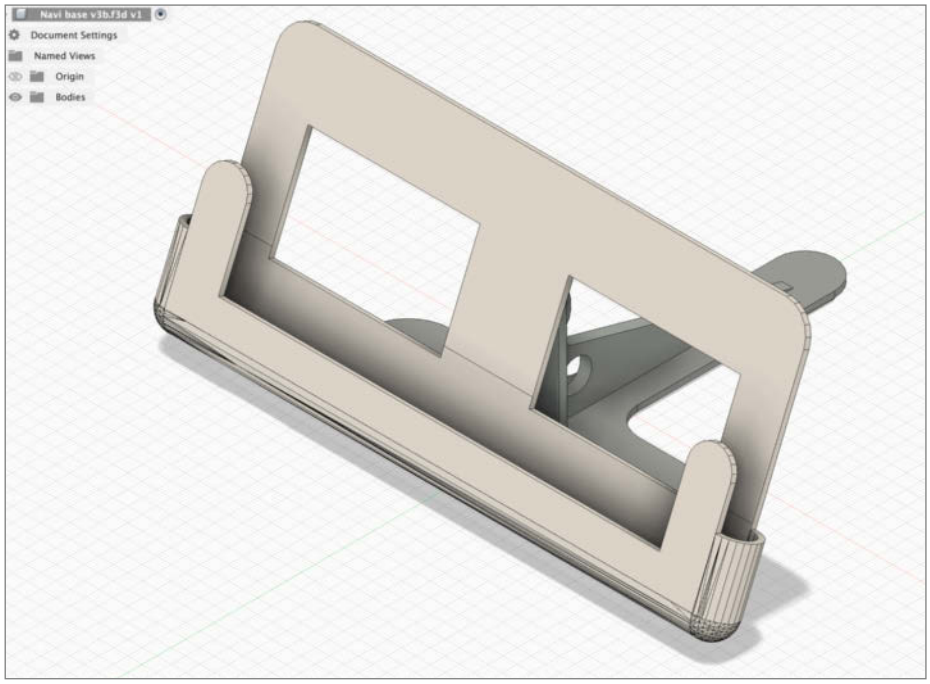


Bild 4: Die aktuelle Version des Halters in Fusion360

Bild 5: Anzeige der Wetterdaten via Webserver vom Homeserver, unten sieht man den PIR-Sensor

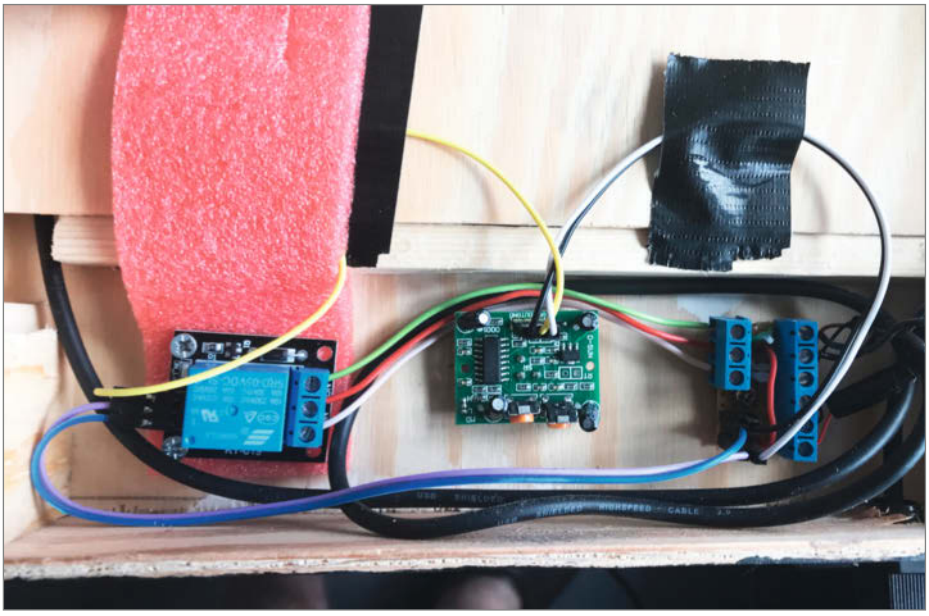


Bild 6 (v.l.n.r.): 1-Kanal-Relais-Modul (mit Schaumstoff als akustische Dämpfung), PIR-Modul, selbstgebautes Anschlussterminal mit Schraubklemmen und Pins

der Opera-Browser (der als einziger Browser auf dem alten Tablet den Vollbildmodus unterstützt) direkt auf den Webserver meines Homeservers zu, auf dem die IP-Symcon-Automatisierungsumgebung läuft. Damit das Display nicht ständig aktiv ist und so der Akku des alten Tablets zu stark belastet wird, mache ich mir einen Trick zunutze: Bekommt das Tablet über die Micro-USB-Buchse einen kleinen Stromstoß, so aktiviert sich das Display (Bild 5).

Daher wurde ein kleines PIR-Modul (Bild 5, Pyroelektrischer Sensor, englisch *Pyroelectric Infrared Sensor*) verwendet, das bei Bewegungserkennung für wenige Sekunden einen Spannungsimpuls liefert und so die Aktivierung vornimmt. Aufgrund des großen Erfassungsbereichs und der hohen Empfindlichkeit musste der eigentliche Sensor nochmals abgeschirmt werden, sodass nur bei unmittelbar frontaler Annäherung eine Aktivierung erfolgt. Hauptgrund hierfür war, dass das Display im Wohnzimmer neben der Couch platziert ist und sonst ständig aktiv wäre, falls sich jemand auf dem Sofa bewegt. Nicht gerade hilfreich beim Fernsehschauen!

Zum regulären Laden des Tablets ist momentan einfach ein Kippschalter angebracht, der die Stromzufuhr auf Dauerstrom schaltet. Eine akustische Warnung durch das Tablet informiert uns, falls die Akkukapazität unter 15 Prozent fällt, sodass der Ladevorgang rechtzeitig gestartet werden kann. Eine Erweiterung wäre ein Bluetooth-Relais, das durch das Tablet selbst geschaltet wird, sobald der Akkustand zu niedrig ist. Leider habe ich momentan für diese Änderung zu wenig Zeit, und da das System bislang soweit stabil und sicher läuft, besteht auch kaum eine Motivation, dies zu ändern.

Alexander Wankerl

MacroDroid als Außenlichtsteuerung

An meinem Haus ist eine Eingangsbeleuchtung angebracht, die nachts über einen handelsüblichen Näherungsschalter aktiviert wird (Bild 7). Mein Ziel war nun, diese Außenbeleuchtung bei einer Helligkeit von unter 10 Lux um 7:00 Uhr anzuschalten und bei über 10 Lux wieder auszuschalten. Falls um 7:00 Uhr bereits mehr als 10 Lux vorhanden sind, wartet das Programm bis 15:00 Uhr und prüft dann alle 15 Minuten, ob 10 Lux unterschritten sind; und schaltet ggf. die Außenbeleuchtung bis 21:00 Uhr ein. Ab 21:00 bis 7:00 Uhr ist dann nur noch der Näherungsschalter aktiv.

Diese Steuerung war für mich am einfachsten mit MacroDroid (siehe auch Seite 72) auf einem alten Smartphone zu realisieren, da dort alle notwendigen Steuerungselemente

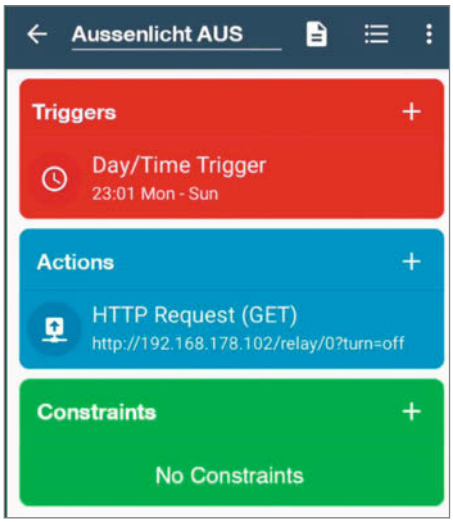


Bild 8: Ein Teil des MacroDroid-Programms

vorhanden sind. Eine Einführung in MacroDroid finden Sie über den Kurzlink in diesem Artikel. Eine Klickanleitung, wie man die Steuerung über MacroDroid anlegt, haben wir ebenfalls online gestellt.

Vor der Programmrealisierung sollte man einen detaillierten Ablaufplan des Programms erstellen, da MacroDroid-Programme schwer nachträglich zu ändern sind. Dieses Programm (Auszug in Bild 8) habe ich auf einem alten Samsung-Handy implementiert und dabei den dort vorhandenen Fotosensor zur Lichtmessung verwendet. Dazu schaut das Handy aus dem Fenster meines Arbeitszimmers auf den Hauseingang.

Die notwendige Übertragung der Schaltbefehle in das 230-V-Netz erledigt ein „Shelly 1"-Schalter; die Funkverbindung vom Handy zum Shelly erfolgt über mein WLAN-Heimnetz.

Im Heimnetz habe ich für dieses Handy und den Shelly eine Internetsperre eingerichtet, damit sie nicht „nach Hause telefonieren" und sich eventuell unerwünschte Updates herunterladen können. Zusätzlich habe ich meinen Router angewiesen, nur auf Geräte mit bekannter MAC-Adresse zu reagieren; dies dient zum Schutz vor Hacks.

Jörg Wölber

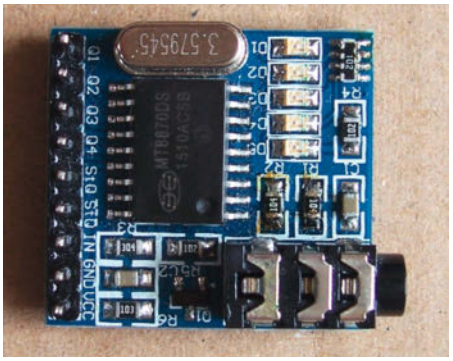


Bild 9: DTMF-Decodermodul

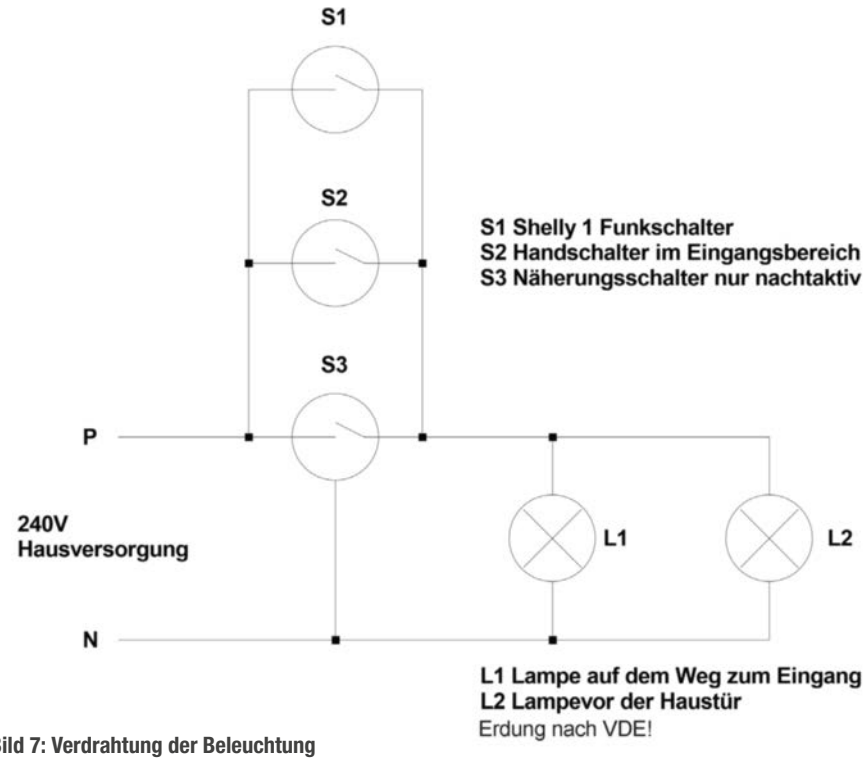


Bild 7: Verdrahtung der Beleuchtung

DTMF-Töne steuern Geräte

Die Überlegung, Handys möglichst universell als Steuerung benutzen zu können, führte uns zu einem ganz alten System: DTMF (Dual Tone Multi-Frequency). Gedacht als Steuerung von Vermittlungsstellen und Anrufbeantwortern hat, dieses Verfahren auch beim Hacking von Telefonnetzen eine Rolle gespielt.

Unabhängig davon, ob älteres Handy oder Smartphone, ob Android oder iPhone, ob aktuelle OS-Version oder ältere, ob Bluetooth-fähig oder nicht und ohne spezielle App: DTMF-Töne können praktisch alle Telefone erzeugen. Entweder kabelgebunden per Kopfhörer- oder Headsetausgang oder drahtlos per Bluetooth-Audio-Interface. Zudem generieren fast alle Modelle beim Betätigen einer Taste auf der Telefontastatur einen DTMF-Ton, der dann auch über die Audio-Schnittstelle ausgegeben werden kann. Viele Handys haben auch einen Musik-Player, sodass DTMF-Töne durch Abspielen einer entsprechenden Musikdatei erzeugt werden können. Für moderne Geräte gibt es immer noch Apps, welche diese Töne erzeugen.

„Dual-Tone Multi-Frequency" bedeutet übersetzt so viel wie „Doppelton-Mehrfrequenz". Es können bis zu 16 Töne durch Überlagerung von jeweils zwei sinusförmigen Signalen unterschiedlicher Frequenzen (siehe Tabelle 1) erzeugt werden. Dieses Verfahren ist in deutschsprachigen Ländern auch unter dem Namen Mehrfrequenz-Wahlverfahren bekannt. Es wurde bei der analogen Telefontechnik zur Übermittlung der gewünschten Rufnummer an die Vermittlungsstelle verwendet.

Wenn man auf dem Handy eine Telefonnummer wählt, werden dort fast immer auch DTMF-Töne auf dem Lautsprecher ausgegeben (teilweise kann dies in den Einstellungen konfiguriert werden). Schließt man ein Headset an, so werden die Töne dort ausgegeben und können mit einem geeigneten Decoder ausgewertet werden.

Passende DTMF-Decodermodule, z. B. mit dem MT8870 Chip (Bild 9) gibt es fertig aufgebaut günstig zu kaufen. Sie decodieren das Tonsignal und stellen passende Signale für Mikrocontroller an Pin-Leisten zur Verfügung. Das DTMF-Audiosignal wird über eine Klinkenbuchse eingespeist, wobei nur der rechte

Tabelle 1: Zuordnung der DTMF-Frequenzpaare zu den Tasten

	1209 Hz	1336 Hz	1477 Hz	1633 Hz
697 Hz	1	2	3	A
770 Hz	4	5	6	B
852 Hz	7	8	9	C
941 Hz	*	0	#	D

Bild 10: Sogar mit einem solch altertümlichen Handy ist eine Steuerung möglich!

Bild 11: Bluetooth-Headset-Adapter mit Kopfhörerausgang

Kanal der Stereobuchse angeschlossen ist. Das Signal wird zunächst intern über einen Operationsverstärker im MT8870 verstärkt. Über das Widerstandsverhältnis R2/R1 ist die Verstärkung des invertierenden Verstärkers einstellbar. Falls das Eingangssignal zu gering sein sollte, sodass keine Decodierung stattfindet und die Lautstärke am Handy bereits auf Anschlag steht, kann der Widerstand R1 verkleinert werden. Der Eingangspegel muss laut Datenblatt im Bereich zwischen 21 und 869 mV liegen. Damit ein DTMF-Ton erkannt wird, muss dieser außerdem eine Mindestdauer von ca. 50 ms aufweisen. Zwischen den Tönen muss eine Mindestpause im ungefähr gleichen Zeitbereich vorhanden sein.

Das dekodierte Signal wird über die Pins Q1 bis Q4 bereitgestellt. Zusätzlich wird der 4-Bit-Code über die LEDs D1 bis D4 angezeigt. Das Bitmuster bleibt so lange stehen, bis ein neues DTMF-Signal erkannt wurde. Während der Dauer des Tonsignals wird der Ausgang STD auf HIGH-Pegel gezogen, was auch durch Aufleuchten der LED D5 sichtbar ist. Dadurch kann man STD als Taktsignal zur flankengesteuerten Datenübernahme verwenden. Zusätzlich wird dieses Signal auch invertiert über einen Transistor am Pin / STD bereitgestellt.

An die Ausgänge Q1 bis Q4 sowie STD des DTMF-Moduls kann ein Arduino angeschlossen werden. Hierzu werden 5 Eingangspins am Arduino benötigt (Bild 10). Den Sketch und die Schaltpläne stellen wir online (siehe Kurzlink) zur Verfügung.

Nun müssen die DTMF-Signale zu dem Arduino mit DTMF-Decoder gelangen. Je älter das Telefon ist, desto schwieriger wird dies. Von Headset- oder Klinkenbuchse, dem Mediaplayer, der DTMF Töne abspielt, DTMF-Apps, bis zu Bluetooth-Lautsprechern (damit hat man dann wieder eine Funkübertragung, Bild 11) reicht die Palette, um die DTMF-Töne zu übertragen.

Eine kreative und niederschwellige Anwendung ergab sich aus der Notwendigkeit, zu einem Soundtrack passende Lichtstimmungen abzuspielen. Stereowiedergabe war nicht nötig und so wurden die steuernden DTMF-Töne auf einem Kanal passend zu der Musik auf dem zweiten Kanal aufgenommen. Die Bearbeitung erfolgte in Audacity, indem die Stereomusik auf einen Kanal heruntergemischt wurde und dann in Audacity an den passenden Stellen die DTMF-Töne mit dem Menü „Erzeugen/DTMF-Töne“ eingefügt wurden.

Je nachdem, wie viele Töne man abspielt, kann man viele „Tasten“ übertragen und so auch komplexere Botschaften an den Mikrocontroller übermitteln. In einem Video haben wir den Prototyp einmal in Aktion aufgenommen. Stellen Sie sich jetzt einen Escape-Room oder Ähnliches vor … Mit Sample-Player-Apps auf dem Smartphone wären auch komplexere Abläufe möglich und wenn dies mit Sensoren (Mikrofon, Beschleunigungsmesser etc.) kombiniert wird, ergeben sich noch ganz andere Möglichkeiten für dieses fast schon antike Verfahren.

Michael Gaus, Miguel Köhnlein

Machen Sie mit!

Kennen Sie auch einen raffinierten Trick? Wissen Sie, wie man etwas besonders einfach macht? Wie man ein bekanntes Werkzeug oder Material auf verblüffende Weise noch nutzen kann? Dann schicken Sie uns Ihren Tipp – ganz gleich, aus welchem Bereich (zum Beispiel Raspberry Pi, Arduino, 3D-Druck, Elektronik, Platinenherstellung, Lasercutting, Upcyling …).

Wenn wir Ihren Tipp veröffentlichen, bekommen Sie das bei Make übliche Autorenhonorar. Schreiben Sie uns dazu einen Text, der ungefähr eine Heftseite füllt, und legen Sie selbst angefertigte Bilder bei. Senden Sie Ihren Tipp mit der Betreffzeile Lesertipp an:

▶ mail@make-magazin.de

Alles zum Artikel im Web unter
make-magazin.de/xjg2

WILLKOMMEN IM NEUEN IOT-ÖKOSYSTEM

Mit LoRaWAN und C-Programmierung über lange Distanzen messen und steuern

Heft + LoRaWAN-Set

Im Make Special LoRaWAN führen Sie 15 Artikel Schritt für Schritt in die Hardware, LoRaWAN, The Things Network und ihre Programmierung ein. Es wird Schritt-für-Schritt erklärt, wie Sie aus den mitgelieferten LoRaWAN- und Sensormodulen einen Umweltsensorknoten entwickeln und noch vieles mehr.

DARUM GEHT'S:

- Einstieg in STM-Mikrocontroller
- Programmieren mit der STM32CubeIDE
- Spannungen, Temperatur und Luftfeuchte messen
- Mit LoRaWAN senden und empfangen
- Daten im The Things Networks verarbeiten
- Werte mit TagoIO visualisieren
- Refresher: Programmieren in C

JETZT BESTELLEN!

Make Special LoRaWAN inkl. Experimentierset für 64,90 €

shop.heise.de/make-lorawan24

tomeqs/ Shutterstock

Elektronische Würfelhelden

Ausgediente Smartphones und Tablet-PCs bereichern so manchen Brettspielabend. Passende Apps werfen dann nicht nur elektronisch die Würfel und führen über den Spielstand Buch, sie lesen auch Texte stimmungsvoll vor, stellen zusätzliche Aufgaben oder ersetzen einen Spielleiter. Ein paar dieser Apps stelle ich in diesem Artikel vor.

von Tim Schürmann

Sind die Würfel beim letzten Umzug spurlos verschwunden, möchte man seinem Lieblings-Würfelspiel auch am Strand frönen oder hat man sich eines der zahlreichen kostenlosen Print-and-Play-Spiele aus dem Internet ausgedruckt, dann schlägt die Stunde der App „Dice“ (ab Android 5.1, iOS 12.0). Sie wirft per Fingertipp einen oder mehrere Würfel in beliebigen Farben, die sogar mehr als sechs Seiten haben dürfen. Dice ist dabei nicht die einzige App: Wer einen elektronischen Würfel sucht, hat die Qual der Wahl zwischen zahlreichen Varianten mit unterschiedlichen Darstellungen und Funktionen. So präsentiert der nur für Android verfügbare „Würfelroller!“ seine Ergebnisse schlichter als sein Konkurrent, dafür aber auch deutlich größer und übersichtlicher (ab Android 5.0).

Zwischen- und Endwertungen notiert man unter Android im elektronischen „Spielblock“ (ab Android 4.0), unter iOS hingegen in der App „Spielergebnisse“ (ab iOS 17.2). Beide Apps generieren sogar ein Diagramm, das unter anderem verrät, welcher Spieler im Verlauf der letzten Partien deutlich besser geworden ist – oder schlechter.

Redselige Begleiter

Während die virtuellen Würfel und Blöcke recht universell einsetzbar sind, existieren für erstaunlich viele Gesellschaftsspiele maßgeschneiderte Helfer. So liest etwa in der App zum Brettspiel „Der verfluchte Geburtstag“ ein professioneller Sprecher sämtliche Texte stimmungsvoll vor (ab Android 8.0, iOS 12.0). Wer hingegen dem Kartensammelspiel „Lorcana“ verfallen ist, verwaltet mit der passenden „Disney Lorcana TCG Begleit-App“ seinen Bestand. Während einer Partie hilft zudem der eingebaute Legendenzähler (ab Android 8.0, iOS 15.0).

Aufspüren lassen sich solche Apps, indem man im Google Play Store oder Apple App Store nach dem Spieltitel sucht. Häufig, aber leider nicht immer, firmieren die Helfer als Companion- oder Begleit-Apps. Die meisten von ihnen lassen sich kostenlos herunterladen und setzen wie schon die Würfel-Apps teilweise erstaunlich niedrige Android- und iOS-Versionen voraus.

Pfiffiger Erklärbär

Welche Spielhilfen eine App im Einzelnen offeriert, erfährt man in den App-Stores häufig erst durch das explizite Abrufen der ausführlichen Beschreibung. So erklärt etwa der „Catan Assistent“ scheinbar nur die Spielregeln der mittlerweile legendären Catan-Spiele (ab Android 5.1, iOS 12.0). Wie jedoch ein Blick in das Kleingedruckte verrät, hält die App unter anderem auch noch für „Catan Duell“ einen elektronischen und äußerst nützlichen Siegpunktzähler bereit. Spieler des „Catan Big Game“ freuen sich zudem über einen Timer und die Moderation.

Erklär-Apps wie der „Catan Assistent“ sind übrigens auch dann extrem nützlich, wenn man die Regeln bereits kennt: Unter anderem hilft sie neuen Mitspielern bei einem schnellen Einstieg und frischt ruckzuck das eigene Regelwissen auf, wenn man das Spiel längere Zeit nicht mehr auf den Tisch gebracht hat. Wer in den App-Stores keine Erklär-App für sein Lieblingsbrettspiel findet, muss lediglich zu YouTube wechseln. Dort existiert inzwischen zu fast allen jemals erschienenen Brettspielen mindestens ein Regelvideo, teilweise allerdings als Spieletest getarnt. Die Videos sind zudem deutlich kürzer und kompakter als so manch eine gedruckte Anleitung. Während einer Partie auf einem Tablet geöffnet, kann jeder Mitspieler bei Regelfragen schnell noch einmal im Video zur passenden Stelle zurückspulen.

Einige Spiele verlangen nach einem Spielleiter, der dann zusätzlich zu seinen eigenen Aktionen stets noch die Züge der gegnerischen Monster nachhalten oder andere lästige Verwaltungsaufgaben erledigen muss. Diese Rolle übernimmt für „Werwölfe Vollmondnacht“ die gleichnamige App (ab Android 6.0, iOS 12.3). Einsetzen lässt sie sich zudem bei den Werwölfe-Varianten „Morgengrauen“, „Vampirdämmerung“, „Epic Battle“ und „Harry Potter – Kampf gegen die dunklen Mächte“. Auch die App für den teuren Gruselklassiker „Villen des Wahnsinns“ steuert auf Wunsch die Monster, verteilt Gegenstände im Gebäude und erzeugt während des Spiels passende Ereignisse (ab Android 5.0, iOS 11.0). Diese umfassende Hilfe setzt allerdings voraus, dass man die Aktionen

Kurzinfo

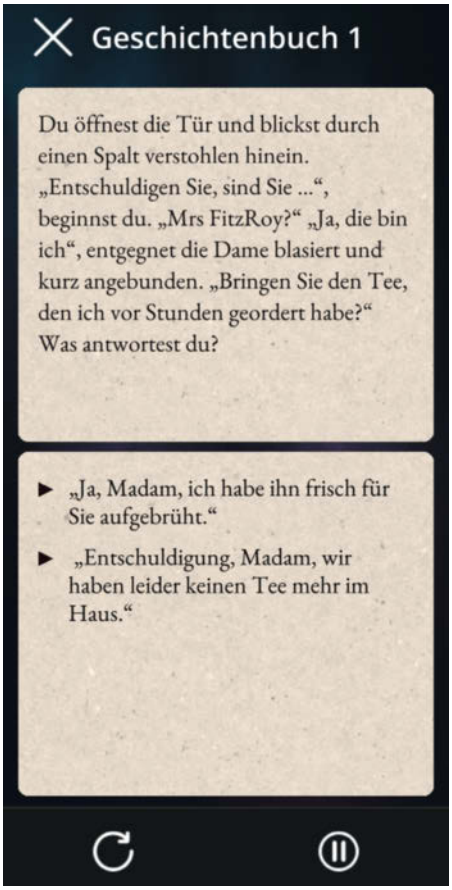
- Tablets und Smartphones für Brettspiele nutzen
- Notizen machen, würfeln und Regeln schneller verstehen
- Alte Spiele mit Companion-Apps neu erleben

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

App-Zwang

Auf der Empfehlungsliste zum Spiel des Jahres 2023 stand auch Hitster. Dabei lauschen die Spieler den ersten Takten von bekannten Melodien und müssen die Musikstücke dann in die zeitlich richtige Reihenfolge bringen. Die Songs spielt dabei die Hitster-App an, die wiederum auf den Streamingdienst Spotify zurückgreift.

Genau wie Hitster sind einige Brettspiele mittlerweile ohne App gar nicht oder nur erschwert spielbar. Ein weiteres populäres Beispiel bilden die Echoes-Spiele, die akustische Rätsel verwenden (ab Android 7.0, iOS 12.0). Aber selbst diese Apps stellen nur geringe Anforderungen an die Hardware und laufen daher auf vielen ausgemusterten Geräten. Dann ist es auch leichter zu verschmerzen, wenn in einer hitzigen Hitster-Runde mal ein Glas Rotwein über das Smartphone kippt.



Die App zu „Der verfluchte Geburtstag“ liest die Texte aus dem beiliegenden Geschichtenbuch vor, was die geisterhafte Atmosphäre des Spiels noch einmal steigert.

Im Catan Assistent erklärt der Professor Schritt für Schritt die Brettspiele aus dem Catan-Universum ...

... zudem gibt es diesen grafisch hübschen Punktezähler für die Duell-Variante.

Bei Villen des Wahnsinns bestimmt die App den Aufbau des Spielplans und treibt die Story voran.

der realen Spielfiguren auf dem Brett auch immer konsequent in der App nachstellt. Bei „Villen des Wahnsinns" ist dies jedoch recht gut gelöst, denn für jede Aktion reicht meist ein schneller Fingertipp.

Mehrwert

Einige Apps fügen sogar neue Inhalte hinzu. Dank des „Robinson Crusoe Companion" muss man auf der einsamen Insel keine Ereignis- und Aktionskarten mehr jonglieren, von denen die App gleich 300 neue Exemplare mitbringt (ab Android 5.1, iOS 12.0). Obendrauf kommen weitere zu meisternde Aufgaben, die man aber bis auf eine Ausnahme kostenpflichtig freischalten muss. Die App für Robinson Crusoe liegt zudem derzeit ausschließlich in Englisch vor. Man muss daher die Texte im Kopf in die deutschen Begriffe auf dem Spielplan übersetzen.

Manche Apps fügen nicht nur Inhalte hinzu, sondern ändern oder erweitern sogar das komplette Spielgeschehen. Letzteres trifft beispielsweise auf „Scotland Yard Master" zu. Bei dieser Variante des Klassikers jagen die Spieler wie gehabt den mysteriösen Mister X quer durch London. Alle Akteure erhalten durch die App jedoch weitere Aktionsmöglichkeiten. Unter anderem erlaubt sie das Befragen von Zeugen oder eine Handyortung. Bei letztgenannter Methode hält man die Kamera auf das Spielbrett, woraufhin die App direkt in das Bild Hinweise auf den Aufenthaltsort von Mister X einblendet (ab Android 8.0, iOS 12.0).

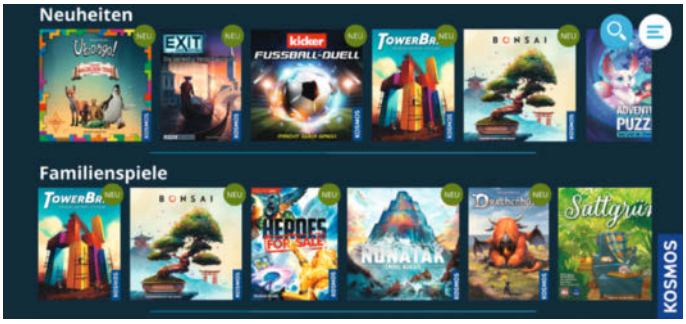
Versteckspiel

Wird man für das eigene Brettspiel in den App-Stores nicht auf Anhieb fündig, sollte man den Namen des Herstellers eintippen. So fasst etwa der Kosmos-Verlag die elektronischen Hilfen für seine Gesellschaftsspiele in der „KOSMOS Erklär-App" zusammen (ab Android 8.0, iOS 12.0). Neben „Cascadia" finden sich hier auch „Ubongo", „Machi Koro" und die „Exit"-Spiele wieder. Anders als der App-Name vermuten lässt, gibt es zu einigen Brettspielen nicht nur eine Regelerklärung. Für die Exit-Spiele hält die App etwa einen „atmosphärischen Timer" bereit, „Sokieba"- und „Kodama"-Spieler bekommen einen Punktezähler spendiert. Übrigens hat Kosmos auch die eingangs erwähnte Catan-App in seine Erklär-App integriert. Nur für den Klassiker muss man folglich nicht beide Apps installieren.

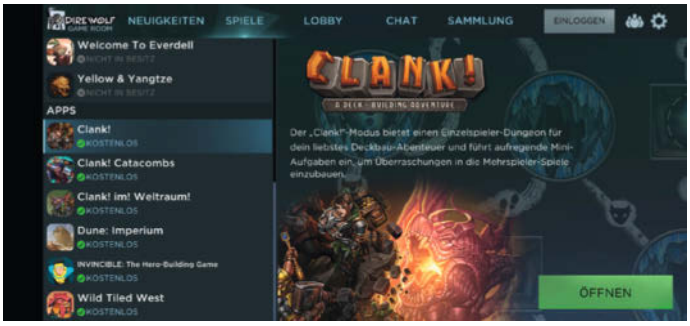
Bleibt selbst die Suche nach dem Verlag erfolglos, sollte man es mit dem englischsprachigen Titel probieren. Bei „Klong!" wird man unter dem Originaltitel „Clang!" allerdings genauso wenig fündig wie bei einer Suche nach dem Schwerkraft-Verlag. Erst eine Fahndung nach dem ursprünglichen amerikani-

Das Smartphone mit dem „Robinson Crusoe Companion" legt man auf dem Spielbrett an die Stelle der beiden Kartenstapel und „zieht" dann ab sofort in der App die Karten.

Die App zu Scotland Yard Master arbeitet teilweise mit Augmented Reality.



Die Verlage Kosmos ...



... und Dire Wolf Digital bündeln in ihren Apps mehrere Brettspiele – letztere allerdings nur auf Englisch.

schen Spieleentwickler Dire Wolf Digital liefert eine passende App: Wie Kosmos bündelt auch dieser Verlag mehrere Spiele in seinem „Dire Wolf Game Room" (ab Android 9, iOS 13.0). Es lohnt sich folglich, alle Spieltitel und alle (ausländischen) Verlage durch die Suchfunktion der App-Stores zu jagen. Die „Dire Wolf Game Room"-App bietet für Klong! neben zusätzlichen fiesen Ereignissen auch einen sehr empfehlenswerten Solomodus. Für den Ableger „Klong! im! All!" generiert sie zudem neue Spielplan-Kombinationen.

Eigenbau

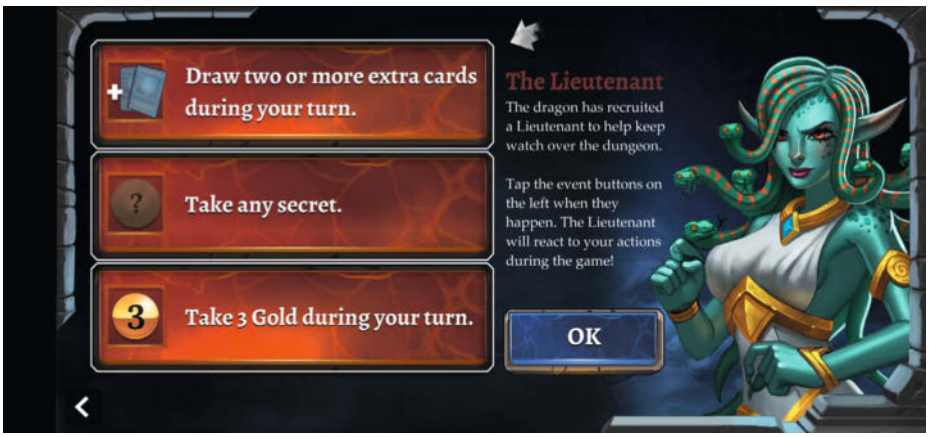
Veröffentlicht ein Verlag keine App, greifen manche Fans zur Selbsthilfe. Das gilt insbesondere für äußerst beliebte Brettspiele. So zählt etwa „Time To Count" von Daniele Folatelli die Punkte bei „Zug um Zug" (ab Android 5.0, iOS 13.0). Analog behält man mit dem „Munchkin Companion" den Spielablauf des kultigen Kartenspiels im Blick. Obendrein wirft die App auch noch die Würfel (ab Android 5.0). Der „Catan Map Generator" erzeugt unter Android per Zufall Catan-Karten (ab Android 5.0), der „Catanous - Catan Map Generator" erledigt diesen Job auf Apple-Geräten (ab iOS 15.0). Optik und Qualität solcher inoffiziellen Apps sind allerdings teilweise etwas gewöhnungsbedürftig. Darüber hinaus stehen unter Android deutlich mehr Apps zur Wahl als unter iOS – hier wirkt sich vermutlich aus, dass Apple höhere Anforderungen an die Veröffentlichung einer App stellt.

Auf von Fans entwickelte Apps muss man erstaunlicherweise auch beim Krimiklassiker Cluedo zurückgreifen. Die (eventuell ausgegangenen) Notizzettel ersetzt unter anderem die Android-App „Detective Notes" (ab Android 7.0). Dessen Konkurrenten findet man in den App-Stores am schnellsten über den englischen Titel „Clue". Nicht alle Cluedo-Apps unterstützen jedoch auch alle Sondereditionen, Überarbeitungen und Sprachfassungen. Die „Detective Notes" orientieren sich beispielsweise an den englischen Ausgaben, in der etwa Oberst Günther von Gatow als Colonel Mustard auftaucht. Eine Stolperfalle lauert zudem bei den offiziellen Cluedo-Apps des Marmalade Game Studio: Ihr „Cluesheet Companion" und der „Cluedo Companion" sind Ergänzungen für die Videospiele und somit beim Brettspieleabend nutzlos. Die offiziellen Videospiele verbergen sich wiederum hinter den Apps „Cluedo: Klassische Ausgabe" und „Cluedo (2024)".

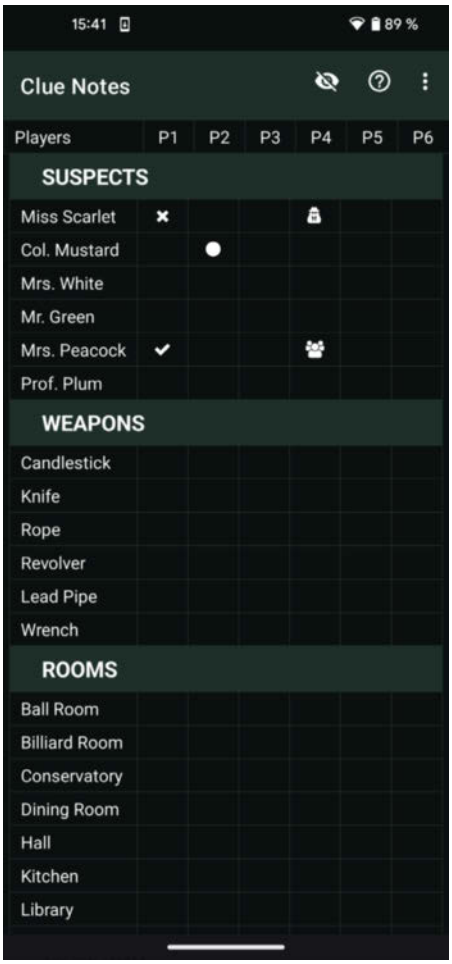
Fazit

Ob die Apps beim Brettspielabend helfen, bereichern oder doch eher stören, muss man selbst in der jeweiligen Runde ausprobieren. Da die Apps kostenlos sind, braucht es dazu nur wenige Fingertipps – und etwas Platz: Einige der Apps gönnen sich über hundert MByte und laden wie die Kosmos- und Catan-Apps unter Umständen weitere Inhalte nach. In vielen Fällen lohnt sich jedoch ihr Einsatz. Und sei es nur, dass ein professioneller Sprecher die Regeln im Schnellgang wieder auffrischt.

Über den Link in der Kurzinfo gelangt ihr zu einer Tabelle, die alle vorgestellten Apps inklusive Links noch einmal auflistet. —*akf*



Mit dem gemeinen Lieutenant bringt der Dire Wolf Game Room noch mehr Abwechslung in eine gepflegte Partie Klong!.



Wer die Mini-Notizzettel der Reiseausgabe von Cluedo leid ist, kann seine Notizen deutlich größer und bequemer mit den Detective Notes führen.

IMPRESSUM

Make: Nächste Ausgabe erscheint am 29. November 2024

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)

Redaktion: Johannes Börnsen (jom), Ákos Fodor (akf), Marcus Hansson (mch), Daniel Schwabe (das), Dunia Selman (dus, Social Media), Carsten Wartmann (caw)

Mitarbeiter dieser Ausgabe: Hans Borngräber, Michael Gaus, Bernd Heisterkamp, Michael Linsenmeier, Ramon Hofer Kraner, Rainer Maria Kreten, Miguel Köhnlein, Carsten Romahn, Uwe Schilling, Maik Schmidt, Tim Schürmann, Martin Siegmann, Sebastian Staacks, Paul Srna, Alexander Wankerl, Jörg Wölber

Assistenz: Susanne Cölle (suc), Martin Triadan (mat)

Layout und Satz: Steffi Martens, Lisa Reich, Nicole Wesche, Heise Medienwerk GmbH & Co.KG

Korrektorat: Dörte Bluhm, Lara Bögner, Marei Stade, Christiane Tümmler, Heise Medienwerk GmbH & Co. KG

Titel: Nicole Wesche

Fotografie und Titelbild: Andreas Wodrich

Digitale Produktion: Melanie Becker, Thomas Kaltschmidt, Pascal Wissner

Hergestellt und produziert mit Xpublisher: www.xpublisher.com

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

Geschäftsführung: Ansgar Heise, Beate Gerold

Anzeigenleitung: Daniel Rohlfing (-844) (verantwortlich für den Anzeigenteil), mediadaten.heise.de/produkte/print/das-magazin-fuer-innovation

Leiter Vertrieb und Marketing: André Lux (-299)

Service Sonderdrucke: Julia Conrades (-156)

Druck: Dierichs Druck + Media GmbH & Co.KG, Frankfurter Str. 168, 34121 Kassel

Vertrieb Einzelverkauf:
DMV DER MEDIENVERTRIEB GmbH & Co. KG
Meßberg 1
20086 Hamburg
Telefon: +49 (0)40 3019 1800
Telefax: +49 (0)40 3019 1815
E-Mail: info@dermedienvertrieb.de
Internet: dermedienvertrieb.de

Einzelpreis: 13,50 €; Österreich 14,90 €; Schweiz 26.50 CHF; Benelux 15,90 €

Abonnement-Preise: Das Jahresabo (7 Ausgaben) kostet inkl. Versandkosten: Inland 80,50 €; Österreich 88,90 €; Schweiz 123.90 CHF; Europa 95,20 €; restl. Ausland 100,80 €

Das Make-Plus-Abonnement (inkl. Zugriff auf die App, Heise Magazine 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




Was sammelt sich außer ausgedienten Smartphones noch an ausrangierten Geräten in euren Schubladen?

Ramon Hofer Kraner

Herisau, verbindet auf S. 44 Sensoren per Bluetooth mit Mikrocontrollern.

Ich finde regelmäßig diverse Adapter, welche von alten Schnittstellen zu noch älteren Schnittstellen verbinden. Aber ich bin sicher, irgendwann brauche ich sie dann doch noch.

Michael Linsenmeier

Karlsruhe, zeigt auf S. 94, wie man ein FTDI-Board steuert.

Ich habe noch Uralt-Handys teilweise defekt, ein Tablet, ausgediente Fritzboxen und eine ausgediente Bluetooth-Box. Vieles davon auch teilweise zerlegt.

Uwe Schilling

Dresden, nutzt auf S. 108 ein Smartphone als Mediaplayer.

Als bekennender Messie habe ich von allem viel. Einiges kommt zu mir über die Frage von Freunden: „Soll ich das wegwerfen oder willst Du das haben?" Fiese Erpresser!

Maik Schmidt

Willich, beschreibt in mehreren Artikeln, wie man Apps programmiert.

In meiner Schublade finde ich drei alte iPods und auch ein paar echte Schätze, wie eine Playstation Portable PSP-1000 und eine Ouya-Spielekonsole, die ich irgendwann einmal wieder zum Leben erwecken möchte.

Inserentenverzeichnis

Ein Teil dieser Ausgabe enthält Beilagen der DIMABAY GmbH, München.