MAKE | BUILD | HACK | CREATE

HackSpace

TECHNOLOGY IN YOUR HANDS

hsmag.cc | April 2024 | Issue #77

CUSTOM CONTROLLERS

Make the perfect keyboard, mouse, and gamepad

DIY Robot Dog
Build your best friend

Chip design
Create personalised silicon

3D printer calibration
Perfect prints every time

Apr. 2024
Issue #77 £6

DOLPHINS **CARBONIZING** MUSIC **AIR**

New look
Same focus

We've refreshed our brand, but our commitment to customer-centric experiences remains constant.

And as always, our goal is to accelerate progress for every designer, buyer, and builder.

Learn more at digikey.co.uk

DigiKey

we get technical

ECIA MEMBER
Supporting The Authorized Channel

Welcome to HackSpace magazine

Computers have cold, hard, silicon brains, while we're squishy sacks of meat. In order for us to use these machines, we need a way of connecting our neurons with the computer's transistors, and until we poke wires directly into our craniums, we need buttons, lasers, sliders, joysticks, and other sensors. Press a button to close a contact, twiddle a knob to change the resistance, gaze into the unblinking stare of an OLED. These are the boundaries between man and machine.

If we want to get the most out of our symbiotic relationship with silicon, we have to optimise this boundary. This month, we're looking at how we can build our own interfaces to bridge this divide and make our computers as responsive as possible to our thoughts. Let's hack this interface to gain maximum control.

BEN EVERARD
Editor ben.everard@raspberrypi.com

Got a comment, question, or thought about HackSpace magazine?

get in touch at **hsmag.cc/hello**

GET IN TOUCH

hackspace@raspberrypi.com

hackspacemag

ONLINE

hsmag.cc

EDITORIAL
Editor
Ben Everard
ben.everard@raspberrypi.com
Features Editor
Andrew Gregory
andrew.gregory@raspberrypi.com
Sub Editors
David Higgs, Nicola King

ADVERTISING
Charlotte Milligan
charlotte.milligan@raspberrypi.com
+44 (0)7725 368887

DESIGN
Head of Design
Jack Willis
Designers
Sara Parodi, Natalie Turner
Illustrator
Sam Alder
Photographer
Brian O Halloran

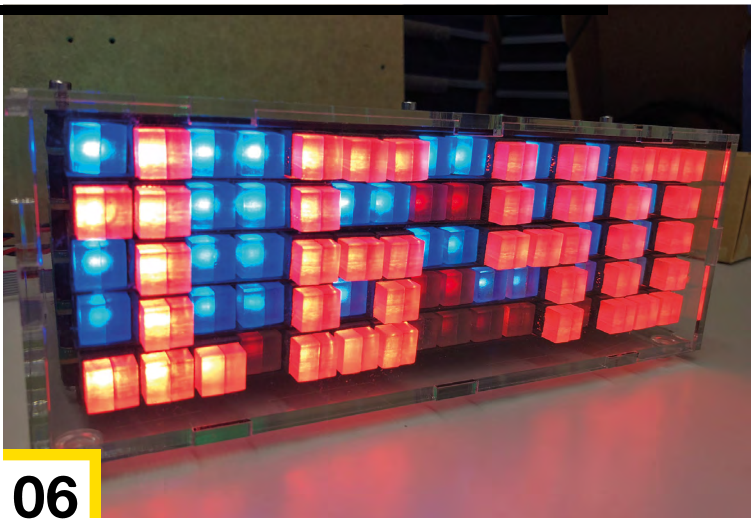
CONTRIBUTORS
Marc de Vinck, Stewart Watkiss, Jo Hinchliffe, Rob Miles, Nicola King, Aaed Musa, Phil King

PUBLISHING
Publishing Director
Brian Jepson
brian.jepson@raspberrypi.com
Director of Communications
Liz Upton
CEO
Eben Upton

DISTRIBUTION
Seymour Distribution Ltd
2 East Poultry Ave,
London EC1A 9PT
+44 (0)207 429 4000

SUBSCRIPTIONS
Unit 6 The Enterprise Centre
Kelvin Lane, Manor Royal,
Crawley, West Sussex, RH10 9PE
+44 (0)1293 312193
hsmag.cc/subscribe
hackspace@subscriptionhelpline.co.uk

Contents


06

Cover Feature

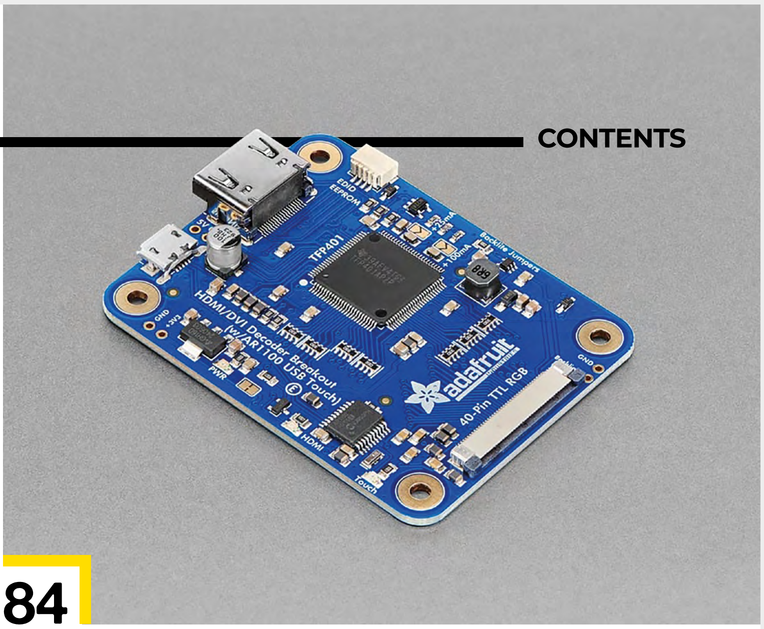
18

28

Tutorial

Gigabutton

62 Touchscreens are rubbish – you want a giant 3D-printed button!

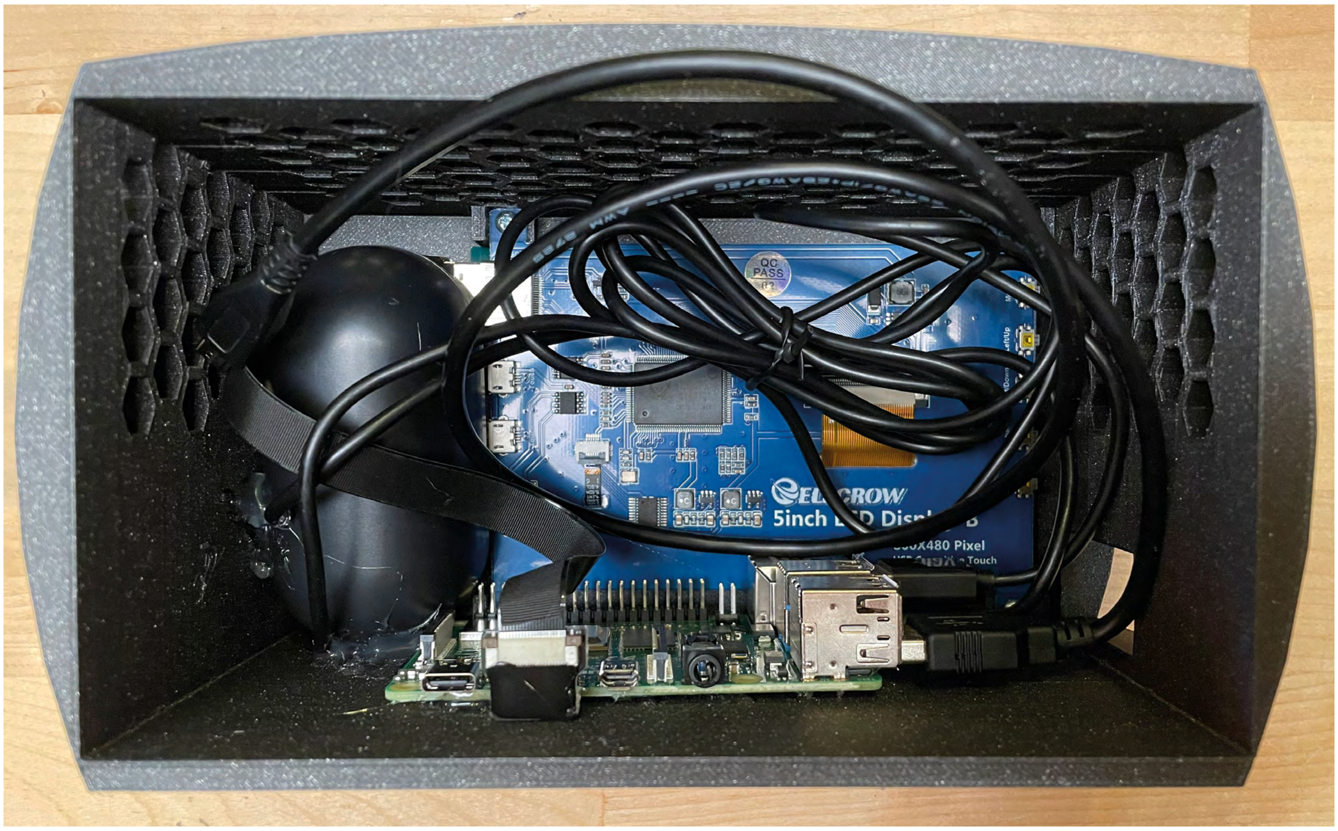


Cat TV

By **Becky Stern** hsmag.cc/CatTV

My old pet dog (RIP Henry) used to love nature programs; he'd show a special interest in anything featuring lions. He'd raise his head, stretch a little, and imagine himself out there on the savannah with them. Apparently, some people prefer cats to dogs, and if you're one of them, you might want to give your furry housemate something to watch. YouTube is reassuringly full of pet-oriented content, and Becky Stern has crafted a delightfully simple device to stream reassuring scenes of birds and squirrels to pique her cats' interest.

She's kept it simple: it uses a 5-inch capacitive touch LCD screen, a Raspberry Pi, a USB speaker, mouse and keyboard, and a 3D-printed enclosure. Even this was one feature too many – the cats kept pawing at the screen, so she replaced the data USB connection with a power-only cable, to disable the touchscreen capability. □



Above Becky used the honeycomb shape generator in Tinkercad to make ventilation holes large enough to let warmth out, but too small for her cats to stick their paws through

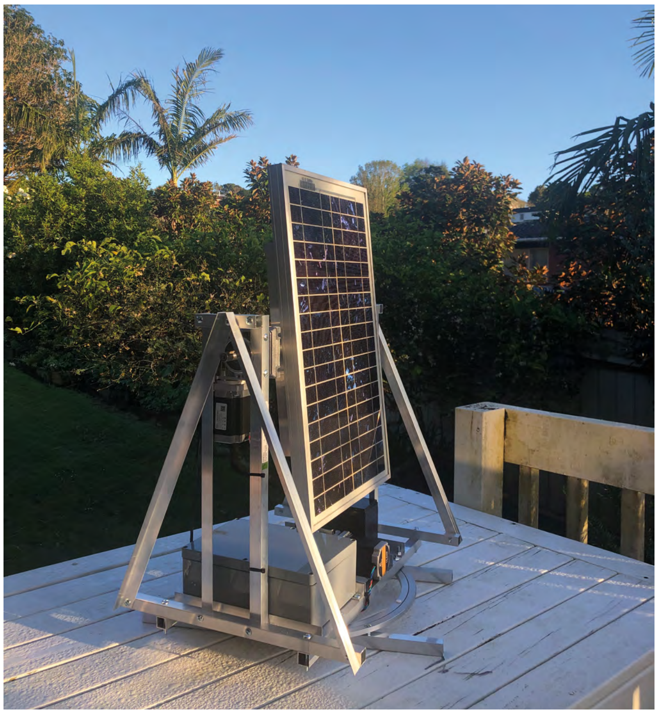
Dual-axis solar tracker

By Tim Ritson

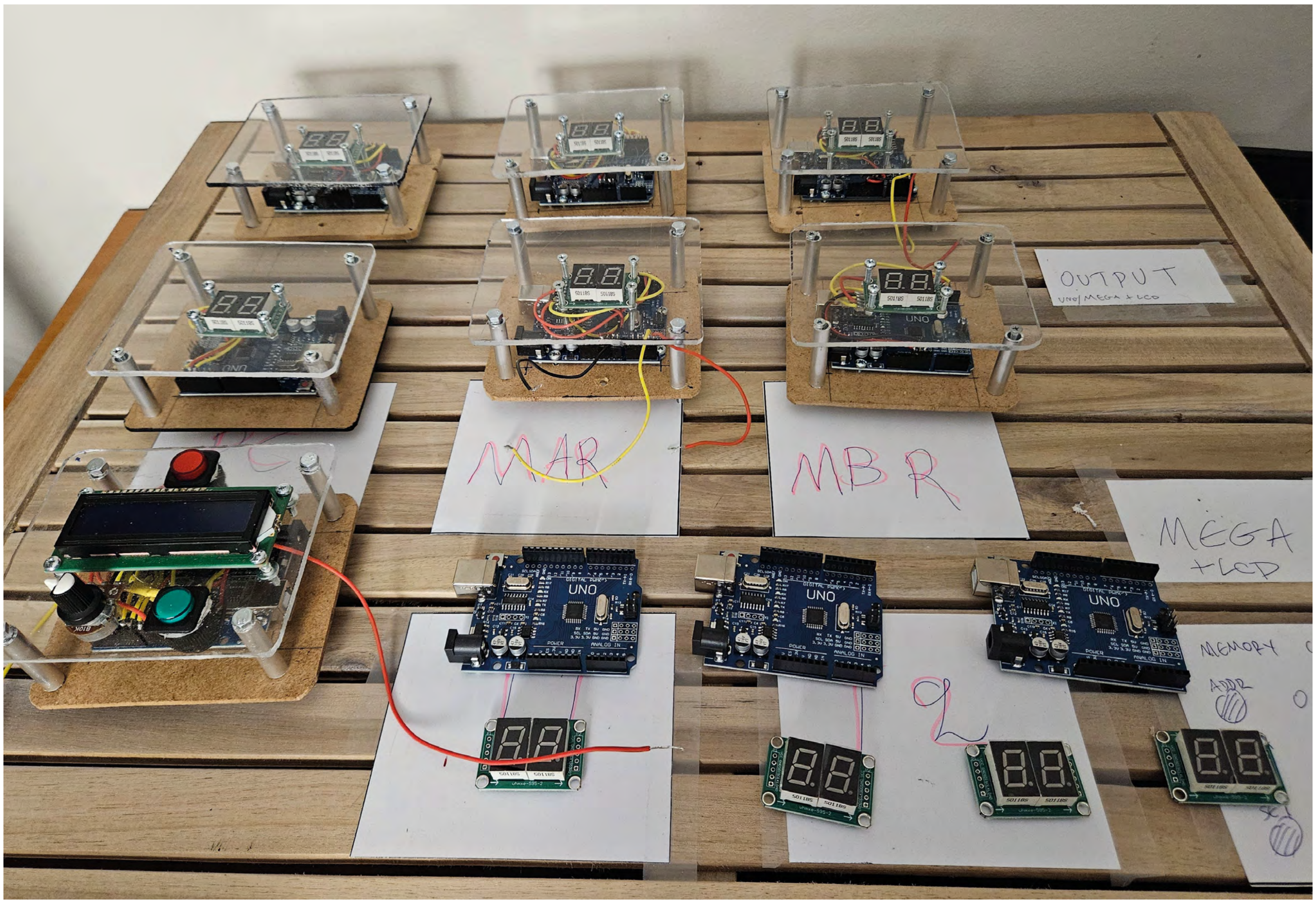
hsmag.cc/DAST

In the northern hemisphere, spring is on its way. The days are getting longer, and warmth bathes the land. It's the perfect time to dabble in solar power! Tim Ritson, like everybody in New Zealand, is exactly six months ahead of us, and he's already had the chance to develop this dual-axis solar tracker, which rotates and pivots a solar panel to maximise the amount of light it collects during the day.

Tim used extruded aluminium for the frame of the tracker, as it's soft enough to cut and drill (slightly too soft, as steel bolts will strip aluminium threads very easily if you're not careful) while offering you plenty of strength without too much weight. Tim used parts from his local hardware shop, electronics store, and AliExpress. The whole thing is mounted on a lazy Susan and controlled by an Arduino Uno and two stepper motors with reducing gearboxes.



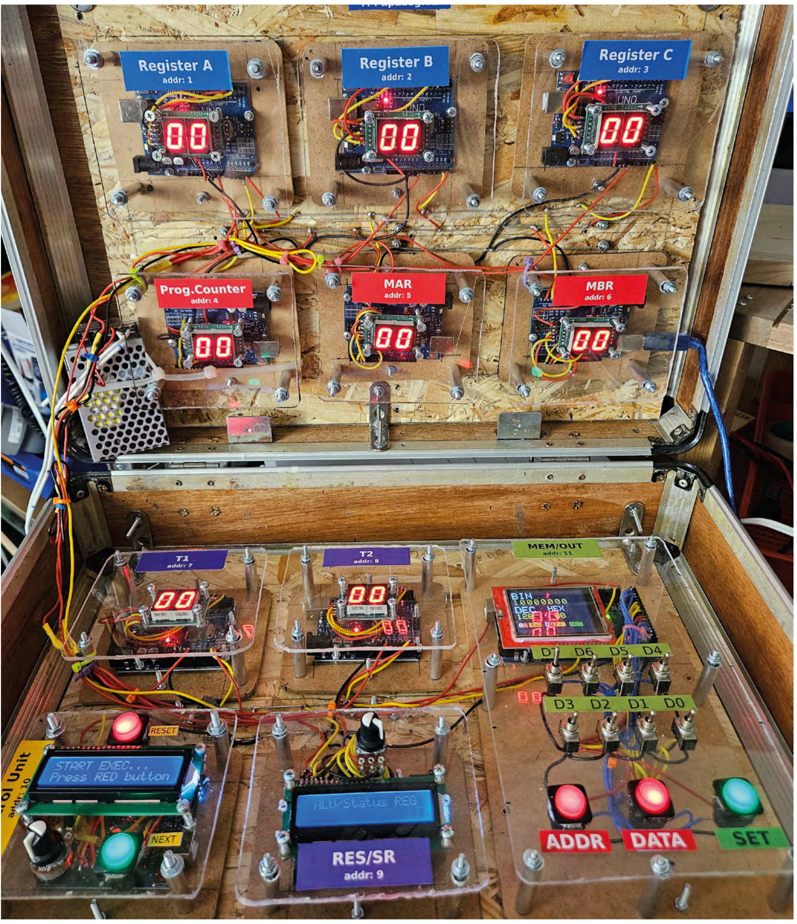
Right Want to make your own? Tim's agreed to write up his experience building a solar tracker for a forthcoming issue of Hackspace magazine

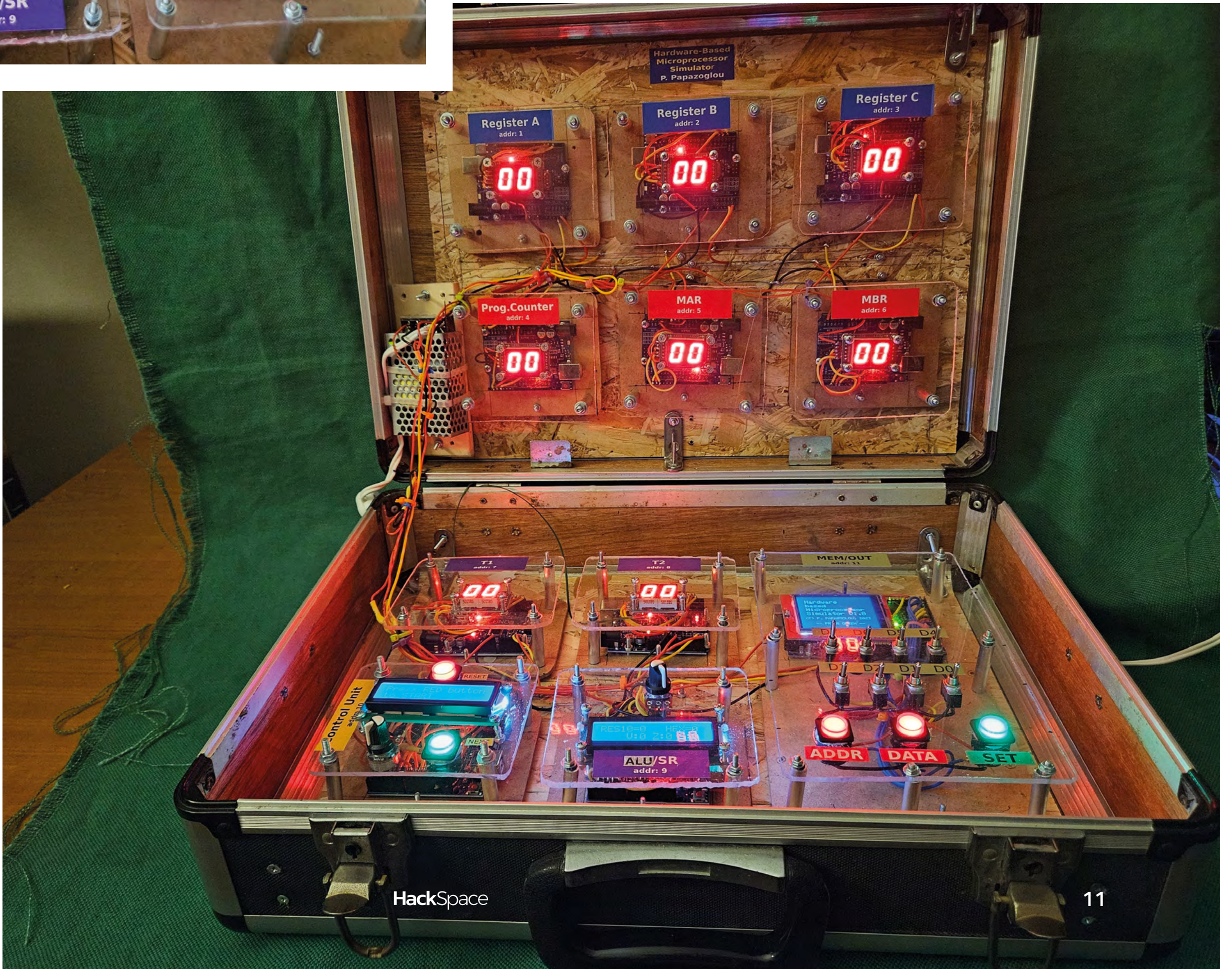


HOMS

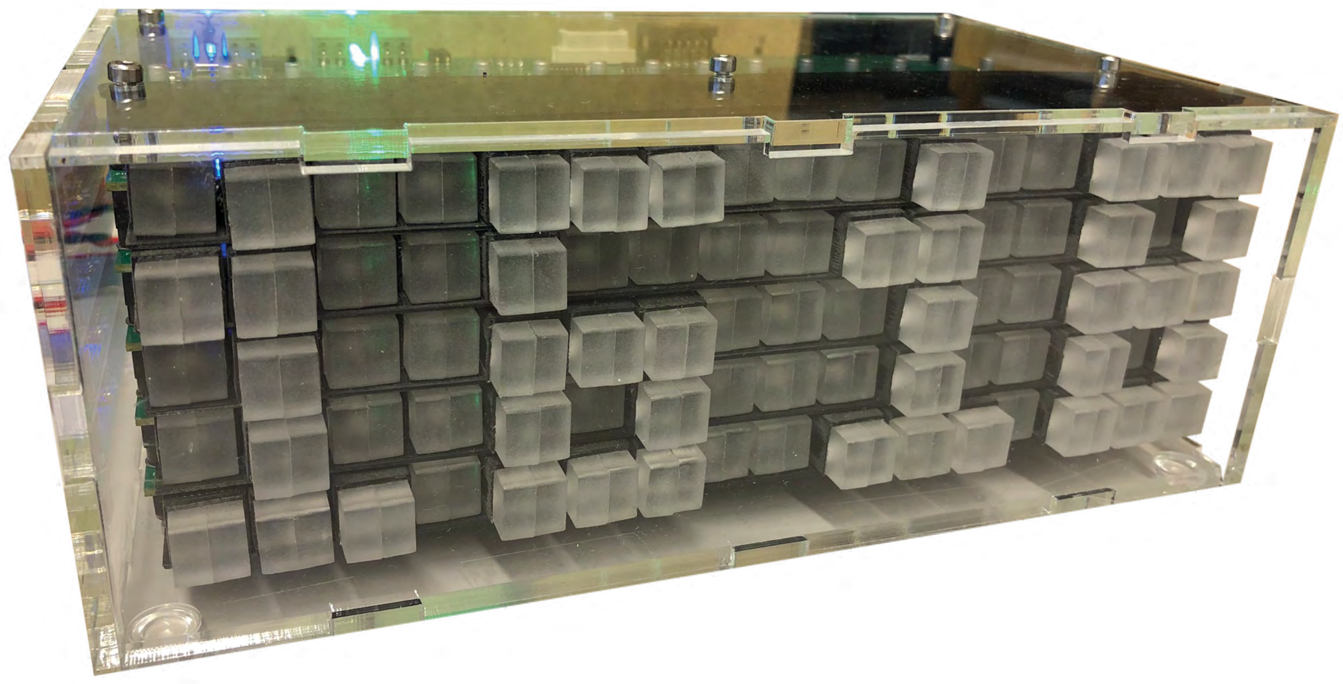
By **Panayotis Papazoglou** hsmag.cc/HOMS

You can buy a box of ten RP2040 microprocessors for £9. You just need a couple of minutes to visit the Pimoroni or The Pi Hut website and enter your credit card details. Depending on your location, it can take a day or two for the delivery to arrive. It's easy. Too easy for Panos Papazoglou! He's made a fully functional microprocessor out of discrete bits of hardware. The registers that would normally be etched at microscopic level into silicon are here replaced with Arduinos, with added displays so the user can see what's going on at any given time in any part of the device. It's brilliantly geeky, and a superb way of visualising what's going on inside a processor for anyone who's looked at a chip and wondered what's going on in there. ◻





Right If the chip shortage strikes again, maybe we'll end up using something like this



Moving Pixel Clock

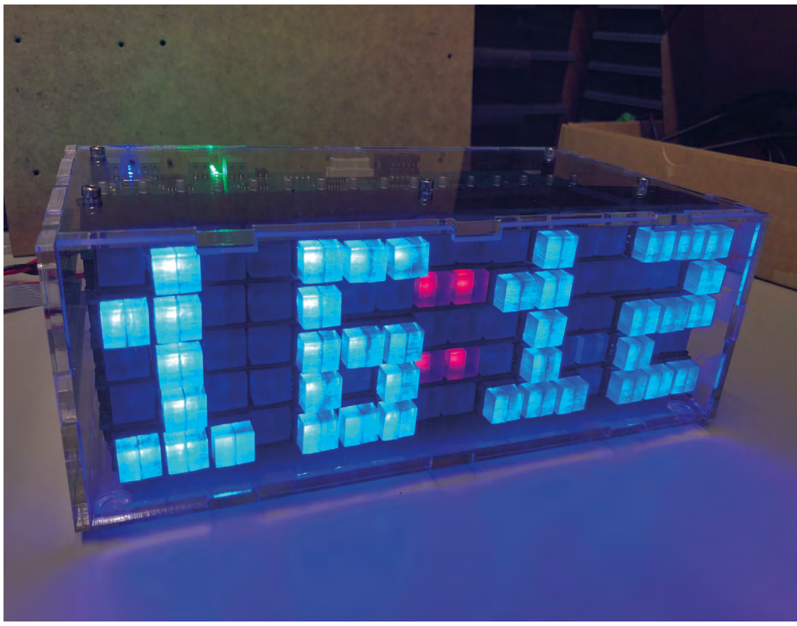
By **Erich Styger** hsmag.cc/MovingPixelClock

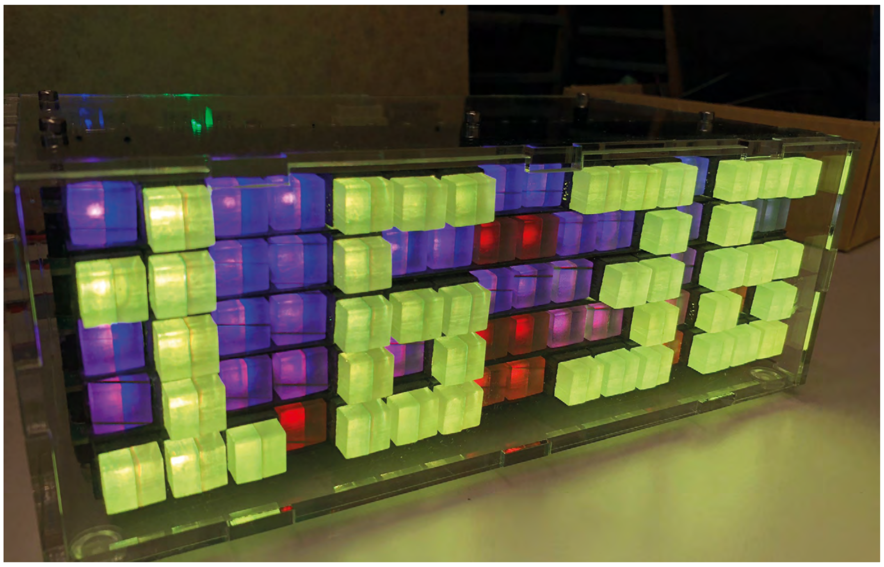
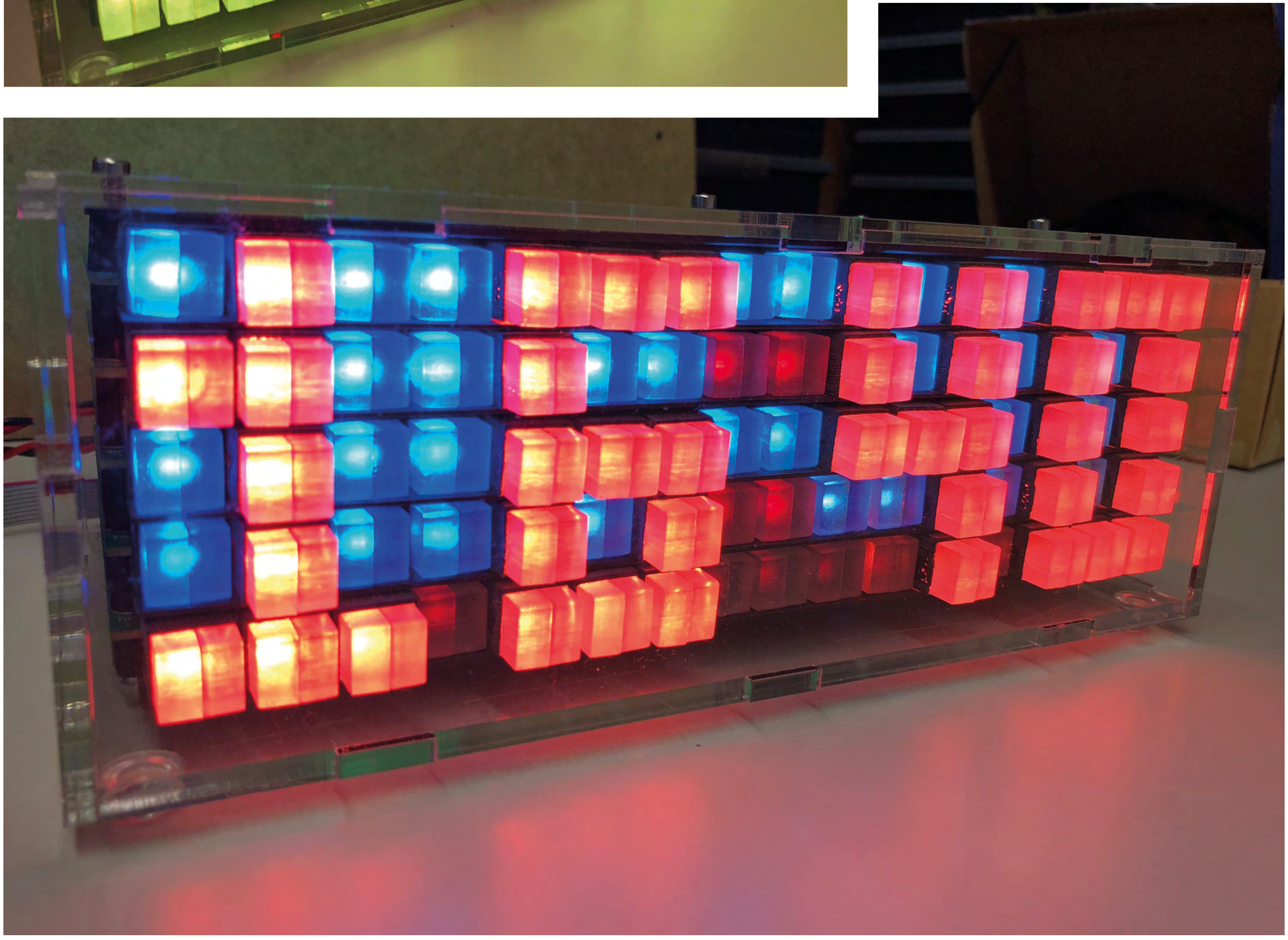
"C**lock projects are cool.** LED projects are cool, too. And if it moves, all the better. So why not build a moving pixel LED mechatronics clock?" With this thought began Erich Styger's 18-month development of the Moving Pixel Clock.

More accurately, the project began when Erich spotted a bunch of tiny stepper motors for sale online. They came with no instructions, no datasheet, and no information at all other than the name – 'Mini stepper motor @5V'. They have a travel of around 10 mm, which takes around two seconds, and Erich thought it would be a good idea to use them to add some interest to a digital clock display.

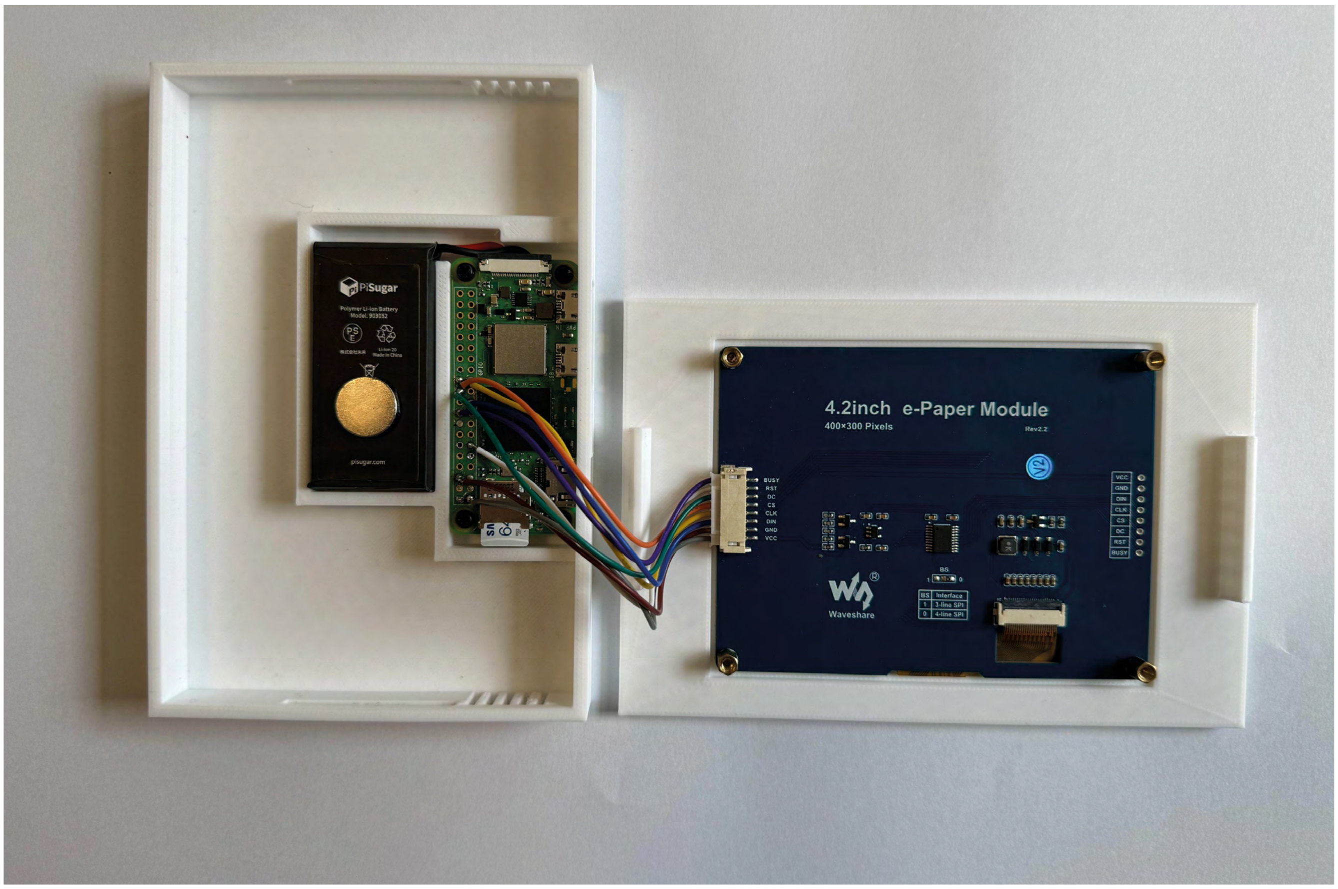
The final build comprises five stacked PCBs housing the stepper motors and stepper motor drivers; each PCB has 16 stepper motors, with each one hosting an addressable RGB LED. That gives you a grid of 16 × 5 moving pixels to display the time. It's simple, but in an admirably complex way.

Erich's blog detailing the project build is a masterpiece of clarity; in it he outlines the build process, from single stepper motor to a wall of moving pixels. Each 3D-printed part and custom PCB is laid bare, giving us a glimpse into the work that's gone into making this weird and wonderful timepiece. ◻



Left Erich credits Leoni Etter, Jan Rohrer, and Livio Stadelmann for their help in building this bonkers device



TypeWryter

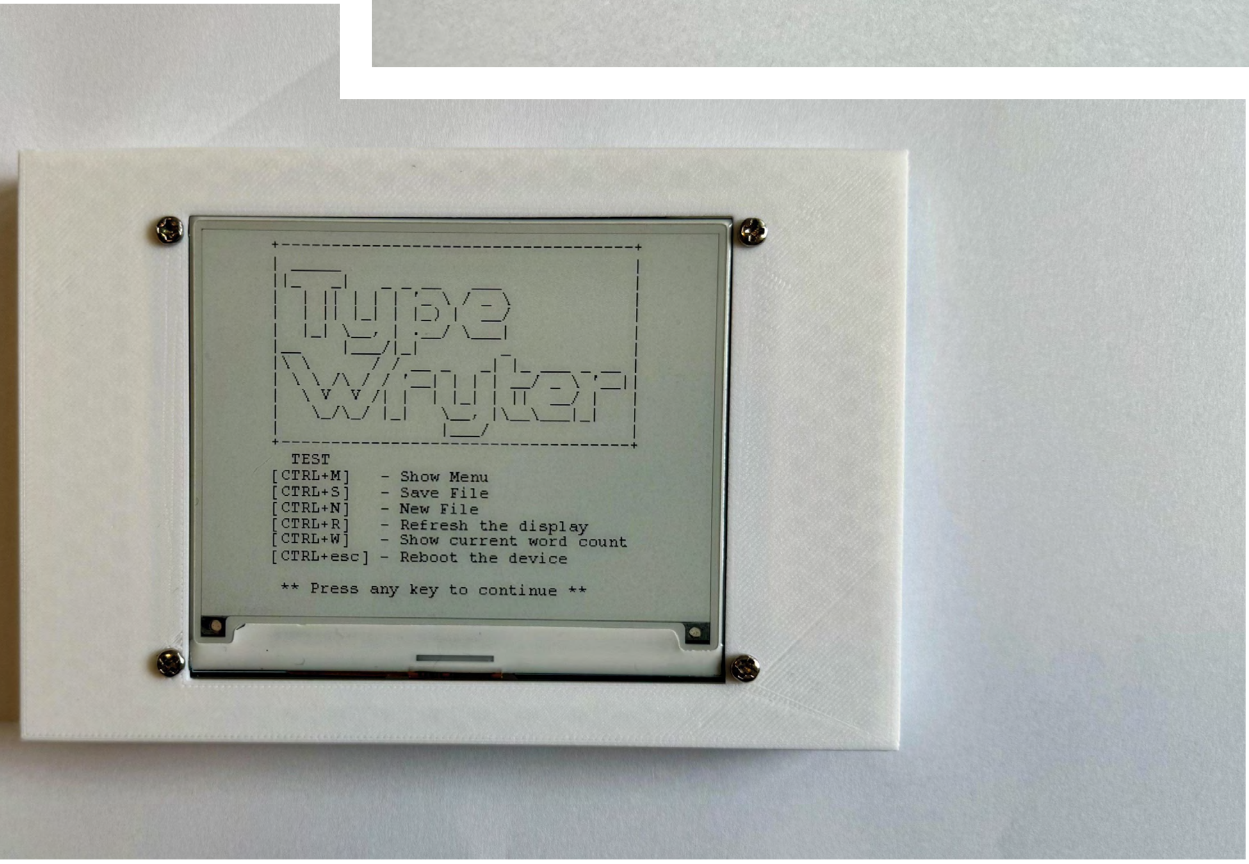
By CertifiedWerewolf

hsmag.cc/TypeWriter

Right Apparently, it's easier to write when you're not constantly distracted by funny videos of baby animals. We might try it someday

Reddit user CertifiedWerewolf (aka Ryan) wanted a distraction-free, light, portable typewriter – something very like the Freewrite Traveler, in fact, but without the £483 price tag. So, he put together a Raspberry Pi Zero 2 W, a Waveshare 4.2-inch e-paper display, a PiSugar 3 battery and a custom-made 3D-printed case, and a keyboard he already had, and made his own for a princely $90.

We love how simple this build is: the clean lines, the minimalist design, and the ethos of doing one thing and one thing well. The fact that it's homemade, and Ryan has shared the code for it on GitHub, makes it even better.



```
 TEST
[CTRL+M]   - Show Menu
[CTRL+S]   - Save File
[CTRL+N]   - New File
[CTRL+R]   - Refresh the display
[CTRL+W]   - Show current word count
[CTRL+esc] - Reboot the device

 ** Press any key to continue **
```

Free eBook!

FreeCAD
FOR MAKERS
Create 3D prints, laser cuts, folded sheets, and more with free design software
By Jo Hinchliffe
Foreword by Yorik van Havre
FROM THE MAKERS OF HackSpace MAGAZINE

Download your copy from

hsmag.cc/freecadbook

HackSpace
TECHNOLOGY IN YOUR HANDS

LENS

HACK | MAKE | BUILD | CREATE

Uncover the technology that's powering the future

PG 28

HOW I MADE: TOPS

Behold the Traverser Of Planar Surfaces robot dog – a non-yapping, non-shedding canine companion

PG 34

INTERVIEW: MICHAEL OMOTOSHO

Like Michael Knight, one man can make a difference – in this instance, to our air quality

PG 18

CUSTOM CONTROLLERS

Gaming devices, keyboards, and everything else – control whatever you want, however you want

FEATURE

CUSTOM CONTROLLERS

Phil King

A long-time Raspberry Pi user and tinkerer, Phil is a freelance writer and editor with a focus on technology.

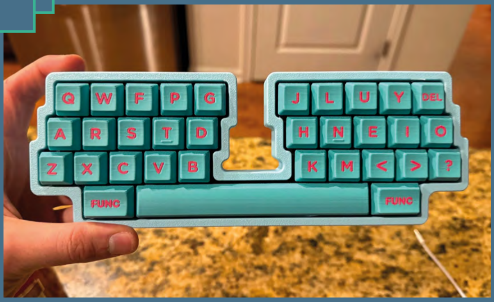
By building your own custom controller, you can determine the aesthetic, layout, features, and exact specifications to suit your own preferences and requirements. And the build will often cost you less than a commercial equivalent – if there is even one available.

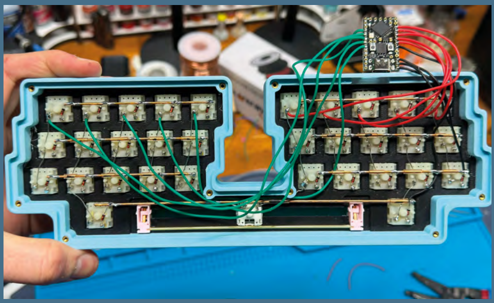
We've rounded up some of the best DIY controller makes around, from mechanical keyboards and macro keypads to gamepads and mice. While just a small sample of what's out there, this selection showcases the variety of designs created by makers, incorporating features such as solar power and keycaps with tiny screens in them.

Naturally, you can modify any of these open-source projects to your own preference, or be inspired to come up with a new design. The beauty of creating custom controllers is that you can let your imagination run wild and make something truly unique, as well as useful.

KEYBOARDS

These mechanical and ergonomic keyboards make typing a joy





ScottoKatana

HSMAG.CC/SCOTTOKATANA

Left
The finished katana-style keyboard has an ergonomic, partially split layout

Below Left
The 33 keys are hand-wired in a matrix of four rows and ten columns

Keyboards needn't be of the usual rectangular format. This one designed by Joe Scotto has a pleasingly symmetrical, partially split layout (aka 'katana') with the two sides connected by a space bar. All 33 keys use Gateron Milky Yellow switches, hand-wired in a row-and-column matrix and connected to the GPIO pins on a microcontroller board – you can use either an RP2040 or ATmega32U4 Pro Micro. Correlating the row and column GPIO input signals enables it to detect which key has been pressed.

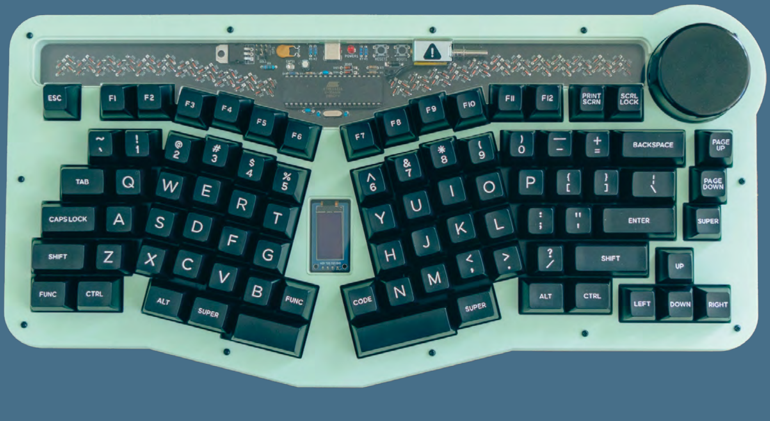
Both the case and keycaps are 3D-printed, with the files made available in Joe's GitHub repo, which also houses preconfigured firmware files based on the open-source QMK (Quantum Mechanical Keyboard) standard. So why not have a go at making one yourself?

Solenoid Keyboard

HSMAG.CC/SOLENOIDKB

Depending on the switches used, mechanical keyboards can range from near silent to full clicky-clacky. Designed by Ming-Gih Lam, this special experimental version of the Red Herring keyboard can do either. With Silent Alpaca switches used for the keys, in normal operation the keyboard is very quiet. A flick of the switch turns it into typewriter-style mode, with each key press triggering a 4.5V solenoid on the board to make that classic click-clack noise. Check out the difference in this video sound test: **hsmag.cc/SolenoidKeysYT**.

The keyboard itself has a 75% ortholinear design with staggered columns and a split ergonomic layout. Diodes are arranged in a unique herringbone pattern, giving it its name, while it also features a large rotary encoder knob and 64×128 OLED screen. All the details and files are in the GitHub repo.



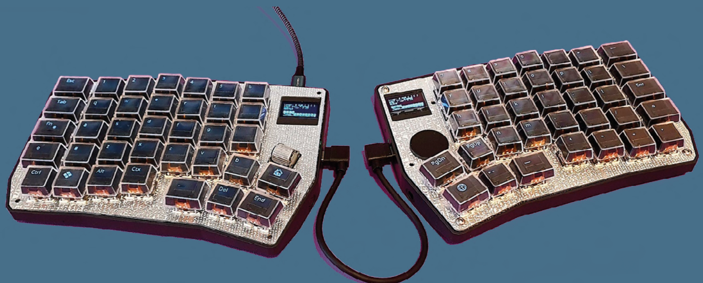
Below
This special version of the Red Herring keyboard can be near silent or clicky-clacky

PolyKybd Split72

HSMAG.CC/POLYKYBD

Incredibly, this 72-key split keyboard features a tiny (0.42-inch) OLED screen in every keycap. Overkill? Perhaps, but it does have the advantage of enabling you to relabel the keys for different languages, so could come in handy for polyglots. The character set can be switched using a custom fork of QMK running on an RP2040 microcontroller.

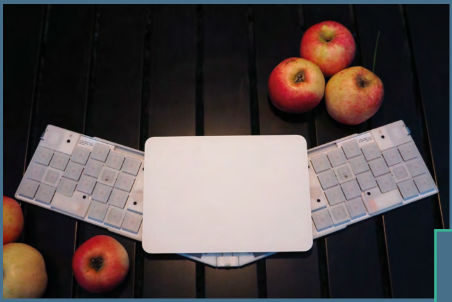
Designed by Thomas Pollak, the keyboard features 3D-printed key stems and caps with a holder for the OLED and a transparent cover to protect it. A flex cable connects each OLED to a slot in a custom PCB underneath. All the project files – including STLs for the 3D-printed case and Gerbers for the PCBs – can be found in the GitHub repo. Thomas also plans to launch a kit version.



Above
Each keycap has a tiny OLED, so can easily be relabelled if needed

Crabapplepad

HSMAG.CC/CRABAPPLEPAD

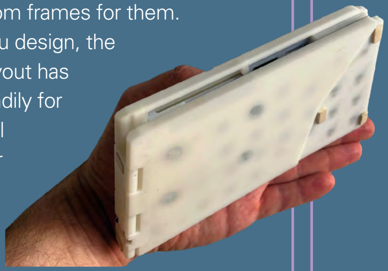


Right
The area between the two keyboard halves can house an Apple Magic Trackpad or phone

Below Right
The hinged keyboard folds up neatly into a pocket-size package

Seeking a better portable keyboard for his coffee shop visits, Sergei Silnov set about crafting the 'perfect keyboard' with a folding mechanism and comfortable typing action. A central 3D-printed hinge enables the two keyboard halves to fold up into a rectangular portable package with 2 cm thickness. When unfolded, a central kickstand can be lifted up to hold a phone; or the two halves can be pulled further apart to accommodate an Apple Magic Trackpad, attached magnetically.

With limited keycap options available for the low-profile Kailh PG1425 X switches, Sergei opted to create custom frames for them. Based on the Miryoku design, the ergonomic 44-key layout has Shift keys placed handily for your index fingers. All the files, including for the KiCad-designed PCB, are available on GitHub.



Above Left
It may look like your average keyboard, but it has extremely low latency

Nyan Keys

HSMAG.CC/NYANKEYS

Keyboard latency is the slight delay between pressing a key and it being input. While not a major issue when typing, it could be the difference between life and death when playing games. Professional gamers prefer a latency of 15 milliseconds or less, but the Nyan Keys mechanical keyboard can do a whole lot better than that: 30 µs (microseconds) in a worst-case scenario.

Maker Portland.HODL achieved this amazing feat by basing the keyboard around an FPGA with parallel per key inputs and debouncing. He then added an STM32F723 MCU to enable USB 2.0 High Speed 480Mb/s communication to the host computer at 8000Hz to achieve the lowest latency possible on a mechanical keyboard that would accept standard Cherry-compatible switches. The maker has also created a 'NyanOS' firmware for the MCU, available on GitHub: **hsmag.cc/NyanOS**.

Birdy44

HSMAG.CC/BIRDY44

Above
Resembling a pair of wings, the Birdy44 is a highly portable split keyboard

Looking to create a lightweight and highly portable keyboard, RalphCoder13 came up with Birdy44. A hand-wired split design that resembles two wings, it is slimmer than his earlier DactylWave model (**hsmag.cc/DactylWave**). The pieces are 3D-printed and he even designed a handy protective case to carry them around.

Each 'wing' features a tiny Waveshare RP2040-Zero microcontroller socketed with Mill-Max round pin headers. Along with the keys, which use Kailh Choc V1 switches, there's a 40 mm Cirque trackpad on each half. Find all the details and 3D print models in the GitHub repo.

Aloidia

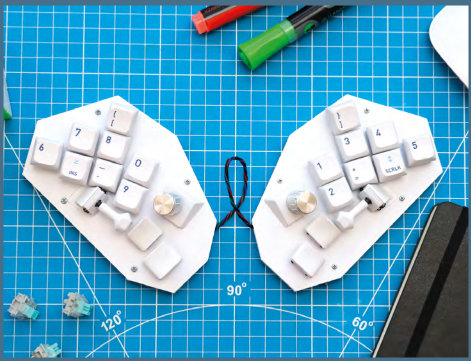
HSMAG.CC/ALOIDIA

Nguyen Vincent's split wireless keyboard features a magnetic interlocking system to attach the two halves and optional palm rests. It also packs in a couple of hot-swappable rotary encoders and a five-way navigation switch in reach of both thumbs. The standout feature, however, is the row of solar panels along the top of the case; these allow the Aloidia's battery to be trickle-charged during daylight hours or under artificial lighting.

A couple of mini memory displays are located either side of the solar panels, while the keyboard case parts are 3D-printed and have embedded tenting to raise them from the surface. Accepting Cherry MX switches, the custom PCBs have a staggered layout. Schematics for the two halves can be downloaded from the Hackaday project page.

Left
Solar panels along the top of the case trickle-charge the Aloidia keyboard

Below
The two halves of the keyboard snap together using magnets and have built-in tenting



Right
Each half features a thumb-stick, rotary coder, and optional key risers

Fulcrum

HSMAG.CC/FULCRUM

Aiming to avoid repetitive strain injury, coder David Schiller decided to replace his full-size keyboard with something a little more ergonomic and finger-friendly. Fulcrum is an example of a 'chorded' keyboard, featuring a limited number of keys that are used in combination to type characters or even whole words. Its keys feature optional risers for comfort.

As David notes, "Our thumbs are criminally underutilised on regular keyboards." Hence the addition of two five-way thumb joysticks. Each half also features a rotary encoder.

The Fulcrum runs KMK CircuitPython firmware on an nRF52840 microcontroller. In addition to the 20-key model, David has designed a 40-key version that looks more like a traditional split keyboard.

MACRO KEYPADS

Ideal for repetitive actions, media controls, and gaming

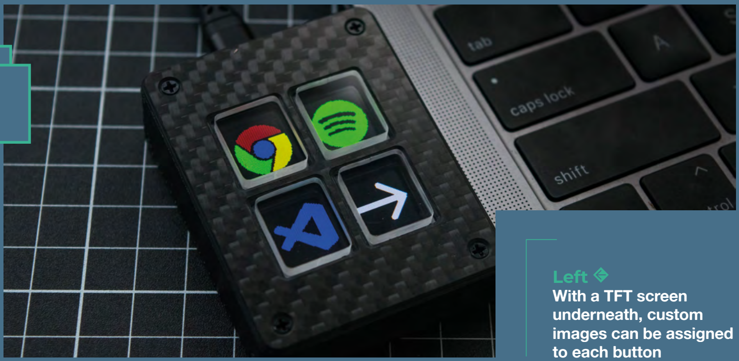
Open Deck

HSMAG.CC/OPENDECK

Designed by Josh R as a lower-cost alternative to commercial stream decks, this open-source macro pad can show a customisable image on each of its four buttons. A 128×160 TFT display fits into a 3D-printed frame with dividers to split it into quarters.

Running on an ESP8266-based Wemos D1 Mini V4 microcontroller, the software includes a GUI settings menu to change the images and three macros each for up to six applications using a folder structure. Even better, Open Deck has MQTT support so you can interface it wirelessly with a Home Assistant server to control your smart home.

Use the project details and files in the GitHub repo to build it, or buy the parts or a fully assembled unit on Tindie.



Left
With a TFT screen underneath, custom images can be assigned to each button

Banana Split

HSMAG.CC/BANANASPLIT

How do you know if a banana-shaped split macro pad is for you? "You'll peel it in your bones," quips maker Dan Bostian. This fun controller features four keys on each half, continuing the fruity theme with Cherry MX switches. The two split sections are connected by a TRRS audio cable.

All the enclosure parts are 3D-printed, including a clip for the small lithium-ion battery pack. Each banana half features a custom PCB, while Dan's ZMK-based firmware runs on a nice!nano microcontroller board with built-in Bluetooth for wireless connectivity.

Assembly instructions, firmware, and all the files can be found in the GitHub repo, so there's no excuse for not building your own banana-shaped macro pad!

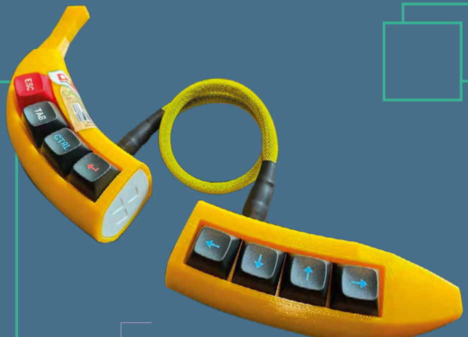
MacroPad 2

HSMAG.CC/MACROPAD2

Well-known for building re-creations of classic computers (as featured back in HackSpace #54), Michael Gardi created a tile-based macro pad, but found he kept trying to press the labelled tile instead of the key next to it. So he reconfigured the device to use the tiles themselves as buttons, equipped with Futaba MD-4PCS switches.

For version two of the device, he designed a custom PCB to replace the original 'dead bug' wiring, and 3D-printed the case and tiles, among other parts. All the files, along with firmware for the macro pad's Arduino Pro Micro board, can be found in the project's GitHub repo: **hsmag.cc/MacroPad2GH**.

Above
Label your 3D-printed tiles and press them to trigger the assigned macros



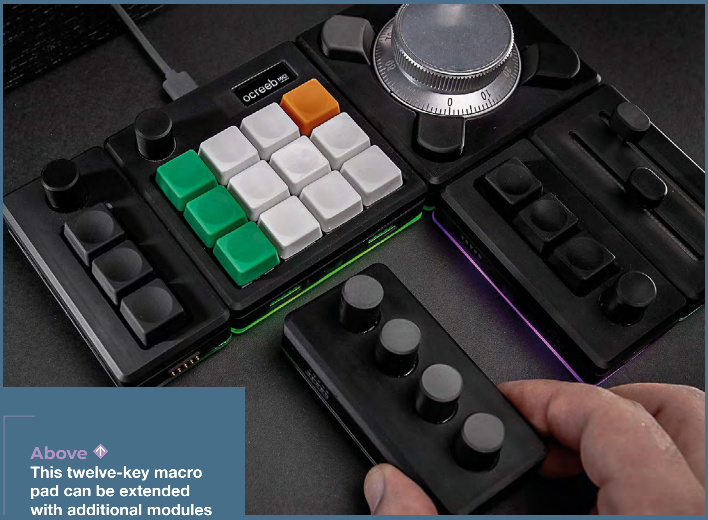
Above
Well, this one is a bit different: a banana-shaped split macro pad

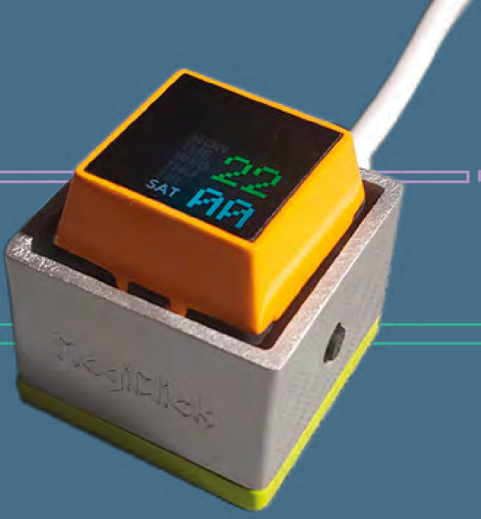
Ocreeb MK2

HSMAG.CC/OCREEBMK2

After listening to feedback for his original Ocreeb twelve-key mechanical macro pad, maker Salim Benbouziyane was inspired to create a new, improved version. The big change is that he's turned it into part of a modular ecosystem. Using magnets and pogo pins, other modules can be snap-fitted to it to create the exact custom layout you want. Modules include a three-key 'combo' unit, dials, and faders.

The main macro pad incorporates a KB2040 microcontroller that runs the KMK keyboard firmware, so you can map whatever actions you want to all those buttons, knobs, dials, and sliders. Salim's Instructables guide (linked above) has the full build details and a link to the GitHub repo for all the files, including firmware, 3D print STLs, and custom PCB Gerbers.


Above: This twelve-key macro pad can be extended with additional modules


Right: This single-key macro pad features a built-in 0.85-inch display

MagiClick

HSMAG.CC/MAGICLICK

While most macro pads have several keys, this one comprises a single button in a 3D-printed frame. Atop the button is a tiny 0.85-inch 128×128 colour screen, protected by a transparent plastic keycap, that can even run a mini GUI. The device is based around a suitably small ESP32-S3 microcontroller and powered by a 300 mAH LiPo battery.

You may wonder what possible use such a single-button device could have. MagiClick's creator 'MakerM0' lists a few, including a stopwatch, clock, pomodoro timer, Bluetooth camera shutter, and music playback controller. With a built-in six-action motion sensor and speaker, we're sure some other interesting uses can be found.

Below: This Bad Thing is a good thing when you're playing Diablo III

Bad Thing of the Edge

HSMAG.CC/BADTHING

A classic use for a macro pad is for gaming, since you can map all the functions you need into an ergonomic layout for maximum efficiency and speed. Fed up with the tiny keys on a small 7-inch laptop when playing Diablo III, Rodrigo Feliciano built the 'Bad Thing of the Edge'. A mechanical keypad designed for the left hand, it features keys for commonly used game functions such as skills, potion, map, and inventory. It can be used for others games, too, if you alter the buttons table in the firmware. For extra excitement, the macro pad features LED strips for a variety of switchable lighting effects. Under the faceplate, the unit is powered by a Raspberry Pi Pico mounted on a custom PCB housing the Cherry MX switches. All the build details and files are in the GitHub repo.

GAMES CONTROLLERS

Improve your gaming with these custom controllers

Voice-Enabled Video Game Controller

HSMAG.CC/VOICEPAD

Gamers with limited mobility may well find the voice control capability on Bob Hammell's custom controller very useful. With a built-in microphone and voice recognition module, this Arduino Micro-powered gamepad enables you to interact with games using voice commands, in addition to using the standard buttons and thumb-stick, which can be mapped to emulate various keystrokes for the game.

When using voice commands, an animated microphone icon is shown on the mini OLED screen, to show that the controller is listening. Whenever a command is recognised, a relevant icon is displayed to confirm it. Voice commands can be used for seven standard game functions: jump, run, shoot, up, down, left, and right.

Below
This retro-style gamepad features a microphone to issue voice commands

Alpakka

HSMAG.CC/ALPAKKA

Above
The Alpakka game controller includes two gyros for tilt control

Input Labs has designed a controller under a Creative Commons licence so you can build it and customise it to your own liking. Based on a familiar twin-handle gamepad design with a D-pad, buttons, and analogue thumb-stick, it also features a touchpad, scroll wheel, and two gyro sensors for tilt mouse control. The makers say it's suitable for playing first-person shooter or mouse-cursor games. It's also ideal for anyone requiring controller modifications to suit particular needs.

By following the instructions and 3D printing the parts, you can build the standard reference design – or you can use community modifications, or make your own customisation choices. As well as a custom PCB, you'll need a Raspberry Pi Pico to run the open-source firmware, which includes per-genre profiles. Everything you need is detailed on the site.

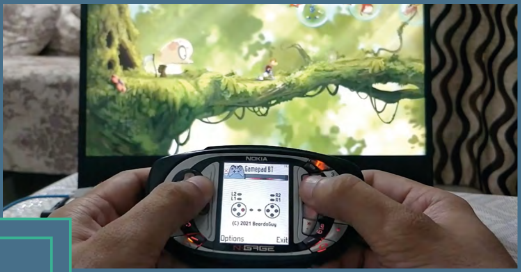
N-Gage QD Bluetooth

HSMAG.CC/NGAGEPAD

Rather than building your own controller, you could always upcycle an old piece of hardware. That's exactly what The Beardo Guy did with a 20-year-old N-Gage QD handheld, by creating an app to convert it into a Bluetooth controller for gaming on a PC or other device – you can watch him playing Rayman Origins with it on YouTube: **hsmag.cc/NGagePadYT**.

The software can be found in the GitHub repo and will run on all Symbian Series 60 1st and 2nd Edition devices. The repo also contains details for building a DIY USB dongle to use as a Bluetooth receiver on the target device.

Below
With the app running, the N-Gage can be used as a wireless controller for another device



MICE

Make a mouse to suit your exact needs

OS3M Mouse

HSMAG.CC/OS3M

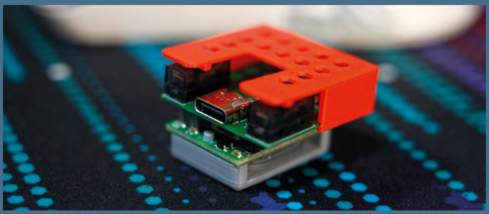
Labelled the 'Awesome Mouse' by its maker Colton Baldridge, this is a 6DOF, 3D-printable 3D mouse designed for use with CAD programs and other modelling applications. While commercial equivalents are expensive, Colton's aim was to create a low-cost ($20 or less), open-source 3D mouse. To this end, all the parts can be 3D-printed, apart from the PCB, fasteners, and three dime coins (used as sensing targets). All the files, including PCB Gerbers, are on GitHub, along with the firmware for an STM32 microcontroller and the PC software. The mouse comprises a hollow 3D-printed knob mounted on a flexure atop the custom PCB which has six coils to detect changes in an AC magnetic field as the mouse knob is moved around – up, down, left, right, or rotated in either direction.

Below This low-cost 3D mouse comprises mostly 3D-printed parts

Tiny Gaming Mouse

HSMAG.CC/TINYMOUSE

Seeking to explore the limits of size and weight in gaming peripherals, maker J opted to build an ultra-small, ultra-light gaming mouse from scratch. Featuring a Pixart PMW3389 motion sensor and ATmega32U4 microcontroller on a custom PCB, it weighs a mere 10g, with a volume of 18 cm^3. A 3D-printed skirt is used on the base to protect the sensor lens and provide a smooth surface for movement, while a shell fits on top for easier handling. You can find all the files on the project page.



Above This truly tiny mouse is fully functional and designed for gaming

CRAZY CONTROLLERS

Here are a few of the more 'out there' controller designs we've seen…

- Doom Keycap (**hsmag.cc/DoomKeycap**): Yes, you really can play DOOM on the tiny screen embedded in this RP2040-based keycap!
- FlopKey (**hsmag.cc/FlopKey**): This utterly impractical 'keyboard' involves inserting a different floppy disk to type each letter.
- The Thing That Goes Clack and Ding (**hsmag.cc/ClackDing**): Keyboard not noisy enough for you? Add this clacker.
- Rubik's Cube Keyboard (**hsmag.cc/CubeKB**): The keys are located on three sides of a cube, although it doesn't twist… yet.
- Haunted Keyboard (**hsmag.cc/HauntedKB**): One for Halloween, this will scare folks with its eerie AI responses.
- Mintboard (**hsmag.cc/Mintboard**): A tiny Bluetooth keyboard that fits inside an Altoids mints tin.

SUBSCRIBE TODAY

FREE Pico W for subscribers!

GET SIX ISSUES FOR JUST:

£30 UK / **€43** EU / **$43** USA & Canada

SUBSCRIBER BENEFITS:

- > **Get every issue of HackSpace magazine delivered to your door**
- > Learn from hackers and makers in our in-depth tutorials
- > Early access to the PDF edition
- > **Get a free Raspberry Pi Pico W**

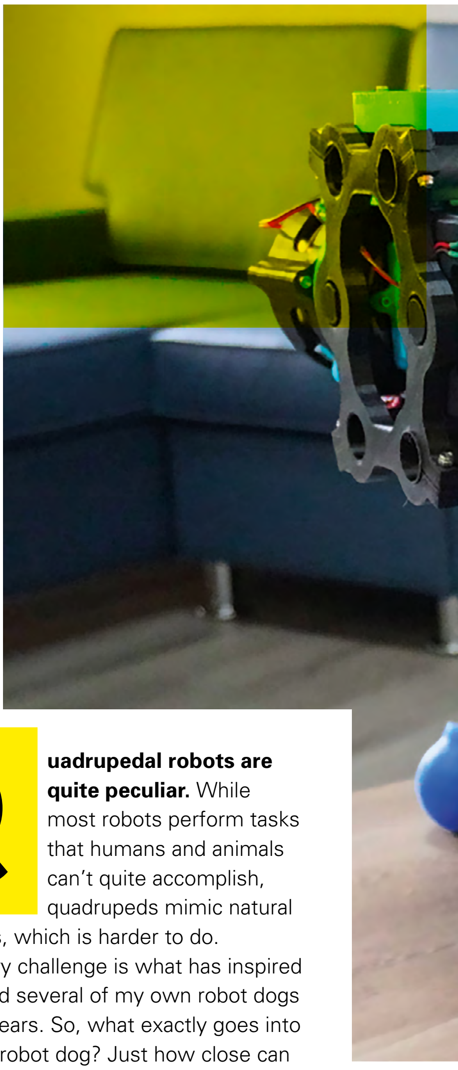
hsmag.cc/subscribe

Subscribers will get a voucher giving them the chance to purchase one Raspberry Pi 5 from reserved stock at The Pi Hut (thepihut.com) for full retail price. Reserved stock means that these will be available even if they are out of stock for general purchase.

HOW I MADE

By **AAED MUSA**

TOPS - TRAVERSER OF PLANAR SURFACES



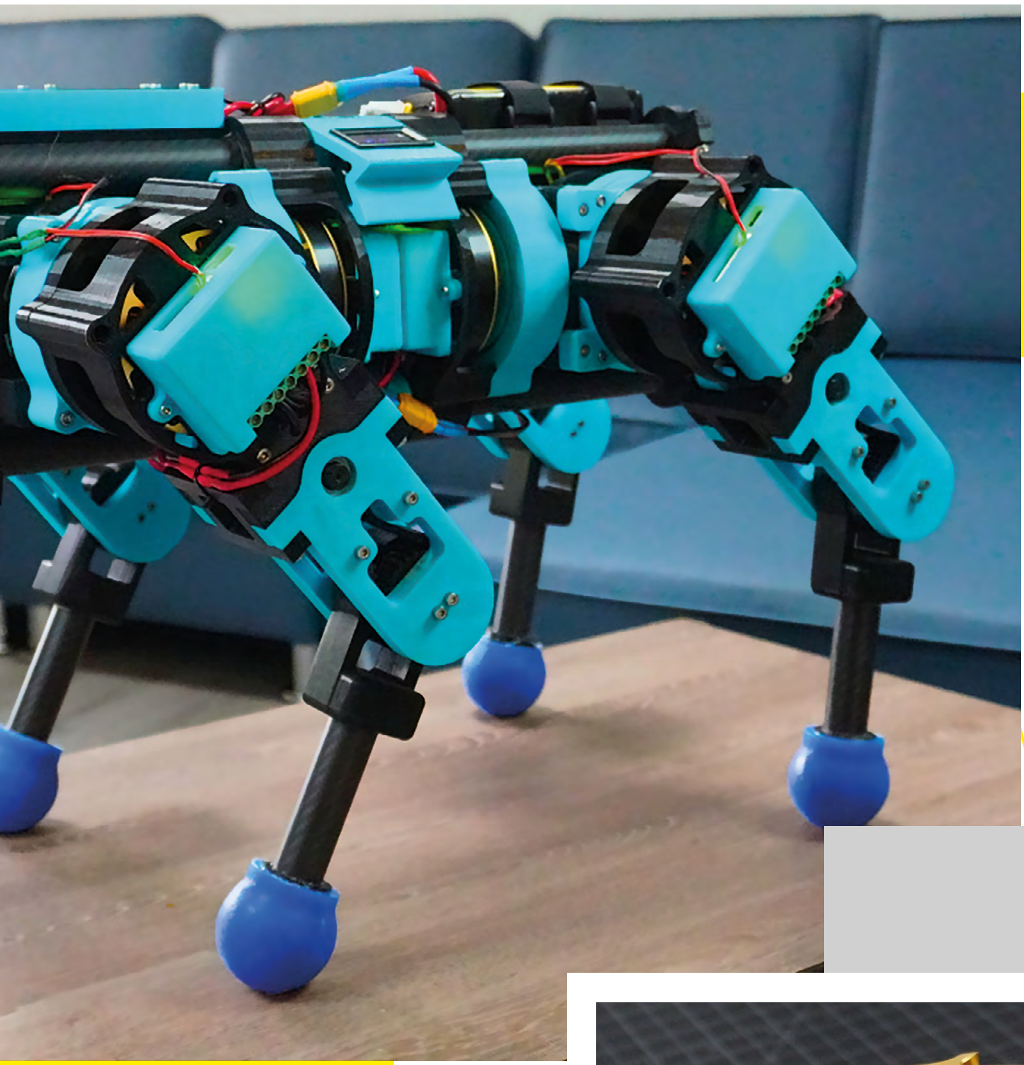
Quadrupedal robots are quite peculiar. While most robots perform tasks that humans and animals can't quite accomplish, quadrupeds mimic natural movements, which is harder to do.

This very challenge is what has inspired me to build several of my own robot dogs over the years. So, what exactly goes into building a robot dog? Just how close can you come to replicating an actual dog? And how cheaply can this be done?

Setting out to answer these questions led me to build my latest quadruped named TOPS (Traverser of Planar Surfaces). TOPS is an open-source twelve degrees of freedom (DOF) quadrupedal robot that can walk, trot, and dance. Other than the motors, bearings, and screws, TOPS is fully 3D-printed.

The name TOPS also spells SPOT backward: SPOT being the famous Boston Dynamics robot.

This project was heavily inspired by the YouTuber James Bruton, who built openDog V3, another open-source quadrupedal robot. The goal of this project was not only to create a functional robot, but to make a robot that provided a sense of realism. Getting a quadruped to walk is one thing, but getting a quadruped to walk with dynamic motion is another.



Left
My fully assembled robotic pet

MAKING MOTION

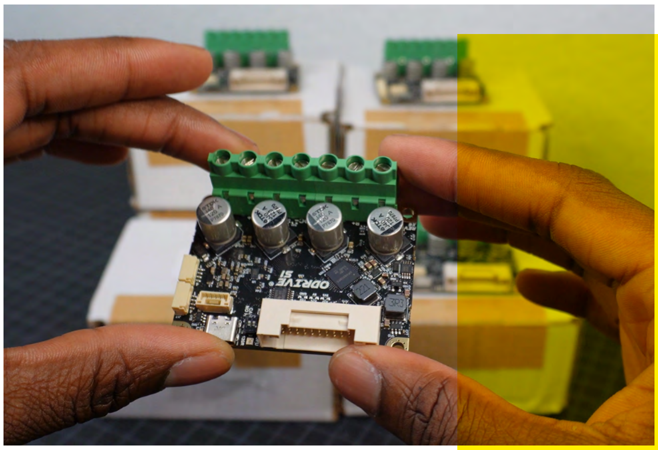
The actuator design of TOPS was the first step in the build process. The most ideal actuator design for a quadrupedal robot is one that follows the quasi-direct drive (QDD) scheme. A QDD actuator can be defined as an actuator that has a low enough gear reduction (under 10:1) to retain the benefits of a direct drive actuator (like efficiency, speed, and backdrivability) while also having a high torque output. To ensure that high torque is achieved with a low gear ratio, it is best to use a high-torque motor. I decided to use Eaglepower 8308 90KV brushless motors that I found on AliExpress for only $60 each. Apart from being cheap, I chose these agricultural drone motors because of their flat and wide pancake-like shape.

Pancake-style brushless direct current (BLDC) motors have high torque densities due to having a large gap radius or distance from the centre of the motor to the rotor. I found the theory and design of an optimal actuator best summarised in an article by the MIT Biomimetic Robotics Lab entitled 'Optimal Actuator Design' (**hsmag.cc/OAD**).

Above
These motors gave me plenty of power for just $60

Right
The motor controllers help hold the motor in the right position

Below
Planetary gears give good reductions without taking up too much space

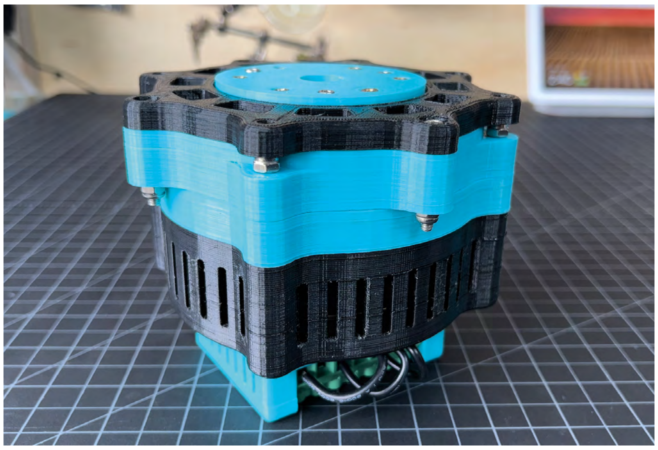


After motor selection, I had to figure out the gear reduction. Planetary gear drives are the common choice for QDD actuators due to their compactness and simplicity. I decided to go with a 9:1 planetary gear reduction since it puts the peak theoretical speed of the actuator at around 222 rpm at 22.2 V. This seemed to be more than fast enough. I used helical gears since they provide gradual contact between engaged teeth, making them both quieter and able to withstand higher loads when compared to standard spur gears.

Motor control was the next item on the actuator design list. Motor controllers are what transform a purely mechanical actuator into a dynamic robotic limb. While drones and RC cars use ESCs (electronic speed controllers) to control a brushless motor, field-oriented control (FOC) controllers are the preferred boards for robotic applications.

FOC is a control method for brushless motors that allows for closed-loop position, velocity, and torque control. FOC controllers can smoothly drive a BLDC motor with an attached load, which is why they are best suited for robotics. FOC controllers essentially turn a BLDC motor into a virtual spring that can be dampened or stiffened by changing different gain parameters. This added compliance to a robot's actuators enables abilities like shock absorption, recovery from being pushed, and the ability to walk across uneven terrains. The FOC controller that I chose to use for this project was the ODrive S1. The controller setup is quite simple: the motor's three phases connect to the controller and an encoder magnet is added to the motor's shaft. An onboard or external encoder is then able to read the motor's position. From there, the motor can be calibrated and configured with different settings.

Below
My actuator ready to be assembled into the robot

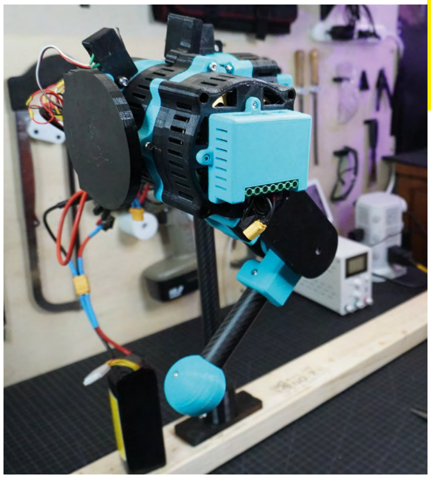


ALL TOGETHER

The full actuator design has three sections: bottom, middle, and top. The bottom section houses the ODrive controller which reads the position of the brushless motor using the ODrive's onboard encoder.

The middle section houses the brushless motor. This section features slots around the radius of the housing which act as air vents for the passive cooling of the motor.

The top section houses the planetary gear-box and the output shaft. The sun gear is directly mounted onto the BLDC motor and the planet gears are suspended on a planet carrier. The actuator utilises



Left
The first leg design was a bit big and heavy

13 3D-printed parts that I printed on my Creality CP-01.

Below are the specs for a single actuator:

- Total costs: $247
- Total weight: 935g
- Dimensions: 133 mm × 105 mm
- Peak torque:16.36 Nm.

FIRST LEG DESIGN

With the actuator design complete, the next step in making TOPS was to design a single 3DOF leg. I went through two different leg designs before moving to the full robot design.

A 12DOF quadrupedal robot is made up of four legs, each leg having three actuators. The three actuators are the abduction/adduction (abad), knee, and hip actuators; each is named after the joint that they control.

One design choice that was made early on was to integrate parts of the leg design into each actuator's design rather than designing the leg around a standard actuator. In other words, each of the actuators has a slightly different exterior design. This helped to limit the number of extraneous parts needed.

The knee actuator, which rotates the forearm, uses a 1:1 belt pulley reduction to reach the knee joint. The forearm is made from a carbon fibre tube. The foot is in the shape of a sphere and was simply 3D-printed for this prototype. In total, the single-leg prototype weighed 3.42kg.

To test the leg, the actuators were connected to a Teensy 4.1 microcontroller via UART.

While the ODrive's onboard encoder can measure the absolute position of the BLDC, it cannot measure the absolute position of the output shaft due to the 9:1 reduction. I therefore decided to use limit switches to home the actuator.

The first step in programming the leg was to derive inverse kinematic equations to accurately place the foot in a known position in 3D space. The derivation process largely consisted of simplifying the leg design into a series of lines and solving for different angles by forming triangles with those lines. These equations allow the computer to know exactly how to position each actuator to place the foot at an input X, Y, Z position.

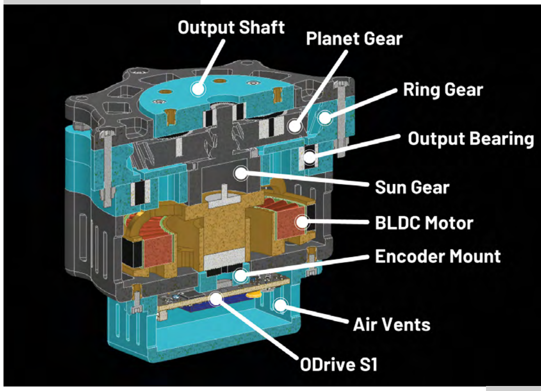
SECOND LEG DESIGN

Following the first leg design, I saw room for improvement in the areas of weight, communication, homing, and overall aesthetic.

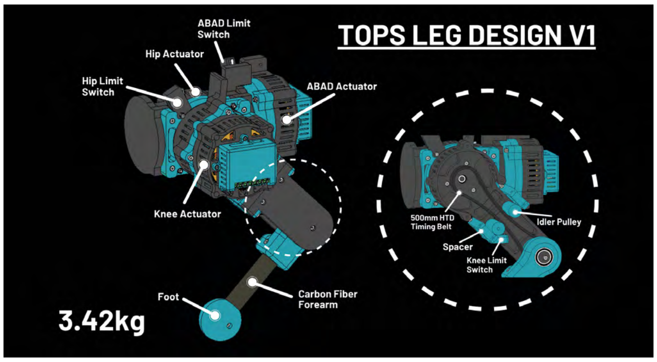
To reduce the weight of the leg, I decided to add weight-reducing slots and holes to all of the parts; however, the biggest weight reduction came from completely redesigning the knee actuator. Previously, the knee actuator design, like the other actuators, had a 9:1 planetary gear-box. It was then connected to a 1:1 belt pulley system to reach the knee-joint. This time, I decided to remove the planetary gear set from the knee actuator and incorporate the 9:1 reduction into the belt pulley system instead. This also reduced the overall width of the leg, therefore needing less torque to be applied to the abad actuator.

The second leg design prototype weighed 2.98kg, which is 0.44kg lighter than the initial prototype. →

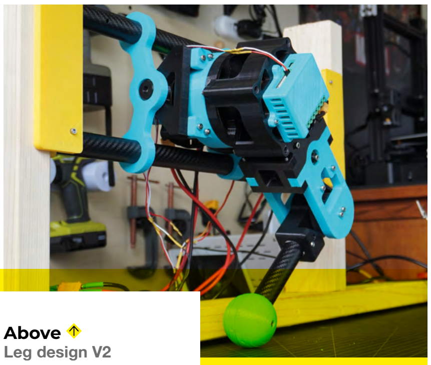
Below
The actuator brings together the motor and the gear-box



Left
All the actuators for one leg



Right →
Leg design V1 drawing



Above ↑
Leg design V2

Previously the communication protocol used to communicate the ODrives with the Teensy was UART. Unfortunately, the Teensy only has eight UART ports when twelve are needed for the full robot. I decided to switch to CAN bus communication.

CAN bus is commonly used in modern vehicles, and it simplified the wiring as only two pins on the Teensy are needed: CAN High and CAN Low.

Another change made in the second prototype was to get rid of the limit switches for homing and, instead, utilise the physical limits of each joint. When performing the homing sequence, each joint is moved to its physical limit. Since this position is unchanging, it can be considered the home position. This home position can then be worked into the inverse kinematic equations as constant offset values.

FULL ROBOT DESIGN

With the second prototype made, I felt comfortable moving onto the full quadruped design. I used four carbon fibre tubes to make up the frame of the robot. Each leg then slides onto two of the tubes and is secured with clamps built into the abad actuator housing. The front of the robot houses a small 16×2 LCD screen to show the robot's current mode of operation.

TOPS is controlled with an eight-channel RC remote and is powered by a 6S (22.2 V) 5200mAh LiPo battery. The actuators are directly powered by the battery while the Teensy is powered by a 5 V regulator. The battery is monitored by a voltage display on the left side of the robot.

It took three weeks to print everything on my single printer, after which I had to put everything together.

First, I built the twelve actuators, then the four legs. Finally, I brought these together to make the full robot.

I decided to cast the feet in 30A silicone, which is somewhere between the squishiness of a rubber band and an eraser. First, I had to 3D-print the sole and the mould.

To cast the feet, each sole was suspended in the mould and then I poured in the A-B silicone mix to cover the surface of the sole. As a result, the foot has a squishy and high-traction silicone exterior and a rigid 3D-printed sole that allows it to be connected to the forearm.

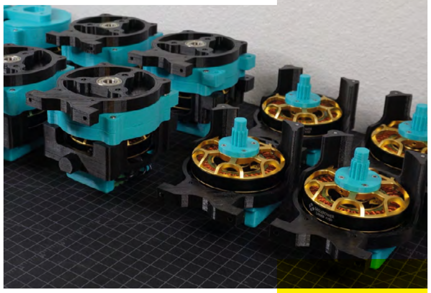
GAIT SEQUENCING

The basic principle of programming a quadruped to walk is to have a pair of diagonal feet in contact with the ground at any given moment: this maintains balance and is otherwise known as trotting. Getting a single foot to take a step forward involves four commands:

1. Move foot up
2. Move foot forward
3. Move foot down
4. Move foot backward to original position

Getting the robot to walk forward is essentially a two-step sequence. First, two diagonal feet must simultaneously take a step. As these feet touch the ground, the two other diagonal feet begin to take their

Below ↓
Assembled QDD actuators



step. As those feet begin to touch the ground, the sequence is repeated.

To walk in another direction, step 2 of the above step sequence is replaced with the desired direction of travel (backward, right, or left), and step 4 is replaced with the direction opposite to the direction of travel (forward, left, or right) to move the feet back to their original positions.

To rotate the robot, a slightly different approach is taken. In this case, diagonal feet are moved sideways but in opposite directions. While the sequences themselves are simple, programming them is a bit more challenging. There are many variables to consider, like the amount of time each leg is off the ground, how far the legs lift off the ground, how big each step is, and how fast the legs accelerate. These factors not only determine if the robot can walk in the first place, but how dynamic that walking motion is.

LOOKING BACK

- Total cost: $3300
- Total project timeline: three months
- Total weight: 13.4kg
- Weight of 3D-printed parts: 4.5kg
- Full battery run time: 10–15 mins

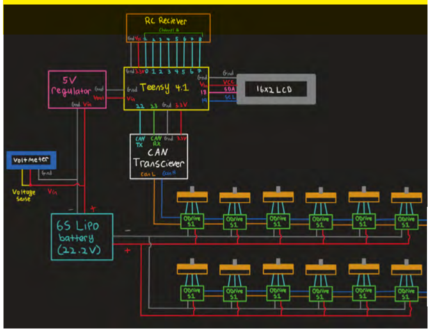
Constructing a robotic system from the ground up was a great engineering exercise. As it turns out, a 3D-printed planetary gear-box can work quite well for large-scale projects with heavy loads. I plan to continue to try out different gearing, motors, and manufacturing methods for more optimal actuator designs. In terms of dynamic gait, I think that TOPS performed fairly well; however, there is lots of room for improvements.

The main mechanical limitation of this project is that the knee-joints skip a lot due to the small contact area between the belt and the smaller gear. Tensioning the belts did not fully solve this issue. In the future, I think it would be best to keep a similar configuration to the initial leg design and have the gear reductions built into the knee and then use a 1:1 belt pulley.

One of the shortcomings of this project on the software side is having to manually calibrate each leg on startup. This became quite a labour-intensive process throughout the testing phase.

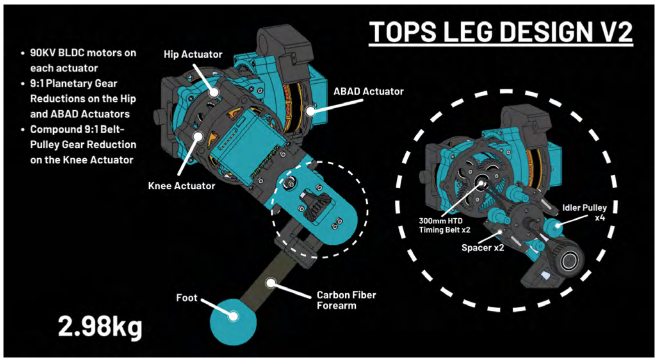
Another software shortcoming was a lack of environmental feedback. In the future, it would be best to use an inertial measurement unit (IMU) so the robot can correct itself if it encounters something unexpected.

I also feel that the stepping sequences are a bit rudimentary and could be smoothened to reflect a more natural gait. I hope to address these in my future robot dog projects. Until then, it's safe to say that TOPS is my most advanced robot dog to date.

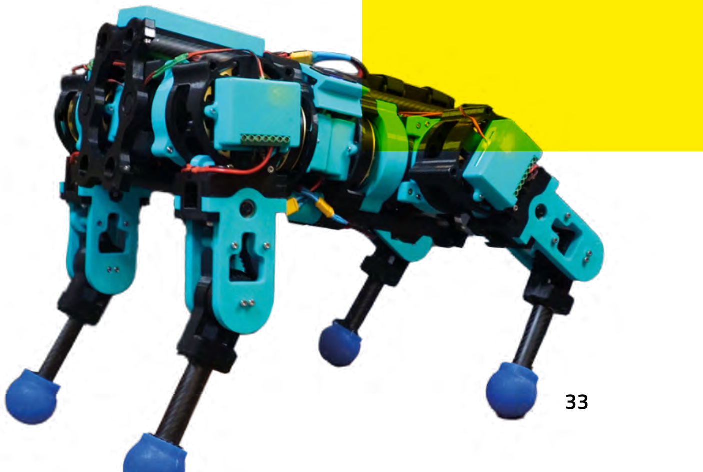


Left
TOPS circuit schematic

Below
Leg Design V2 Drawing


TOPS LEG DESIGN V2
90KV BLDC motors on each actuator
9:1 Planetary Gear Reductions on the Hip and ABAD Actuators
Compound 9:1 Belt-Pulley Gear Reduction on the Knee Actuator
Hip Actuator
ABAD Actuator
Knee Actuator
300mm HTD Timing Belt x2
Idler Pulley x4
Spacer x2
Foot
Carbon Fiber Forearm
2.98kg

Below
TOPS dancing



HackSpace magazine meets...

Michael Omotosho

Sometimes data isn't enough. Sometimes we need a little robot that tells us what to do!

Michael Omotosho is a design engineer. He's currently working at Jaguar Land Rover, but he's worn several hats in his career, working on projects to help elderly people retain their independence for longer, providing solar power to communities in Tanzania, and judging the V&A Innovate National Schools Challenge 2022–23. Like any good designer, when he sees a problem, he wants to fix it. That goes for invisible problems too, like air pollution.

The statistics around air pollution are shocking. 44,000 deaths a year in the UK have air pollution as a contributing factor. Worldwide, that figure rises to seven million. Air pollution leads to children growing up with asthma, reduced brain development, and a whole host of other complaints. We need to make the invisible visible, and to do that, we need an engineer. →

Above
Information is everywhere – Michael is helping to get it into the right hands

HackSpace Morning Michael. First thing's first: who are you?

Michael Omotosho I'm a design engineer by profession. I studied automotive design at university. I've always had a mindset for solving problems regardless of how big or small they are. It started with tinkering at home when I was a kid, and breaking things most of the time, and from there it's just evolved into my passion. To see ideas come to life better than what's already existing… I don't want to use the analogy of making the world a better place, but that's the kind of motto I go with.

From there, I've just spiralled into coming up with various random things that probably don't even make sense. Think of something better, that was my mindset. From there, I went on to work in engineering and had the chance to start my own company as a design engineer and industrial designer. I used to consult for NGO-related companies looking to enhance local communities that were lacking in some way, like creating solar power hubs for use in communities and local businesses in Tanzania, for example: community-focused, and helping to improve people's way of life. That's kind of my passion project. That and questions around disabilities, like how can we make $50 prosthetics with additive manufacturing? Right out of university, those were the projects that I wanted to involve myself in. That's where my passion grew to helping people and doing something purposeful.

I currently work for Jaguar Land Rover as an innovation and technology lead. And that's an interesting space to be in as well. In relation to moving from internal combustion engines to electrical vehicles. How do we enhance that? How do we meet our sustainability goals, and so on?

HS To borrow a phrase, that sounds very much like you're an activist engineer.

MO Yeah, pretty much. I am an activist engineer – I'm constantly trying to try to be doing something. And that leads into the Mindful Droid and all the things that I'm looking into with that.

I came across an opportunity thanks to a colleague of mine, Jude Pullen. He pointed me in the direction of an environmental challenge organised by RS Components, after the COP26 conference (the 2021 United Nations Climate Change Conference, which was held that year in Glasgow, Scotland).

I have to be honest, I was quite ignorant and naive about air pollution. I knew it existed, but I didn't know how bad it was in day-to-day life. And I was introduced to this project with a brief to take the air quality data that we already have and make it more meaningful.

"I have to be honest, I was quite naive and ignorant about air pollution"

And I came across an article about Ella Kissi-Debrah. At that moment, I thought to myself, wow – this goes further than I realise. It takes a huge percentage of lives, and there's no real awareness or action about that. [Ella was a nine-year-old girl whose death was the first in the UK to be officially attributed to air pollution, after a long fight by her mother, Rosamund Kissi-Debrah, to increase awareness of the health impacts of air pollution and what we can do about it].

I stopped there and thought about the companies that I've worked with, and I realised that I have a chronic cough, which is occupational asthma. I realised that I've actually been inhaling the fumes from cooking oil, because one of the industries I was able to work in was food manufacturing, and we used to make food machinery. They used to test the machinery, so the cooking oil that came from it used to go up into the air, and I just used to cough a lot.

Once I got into that rabbit hole, I was like – crikey! This thing had a big impact on my health, and I didn't think anything of it at the time. I left the company, and then my cough reduced. At that moment, I started to wonder what else is out there that's killing us slowly that we don't know about.

Taking my kids to school, I realised that there's something out here during the high traffic times, and I should validate the notion in my head that this is actually an issue. So I went to the school and start asking questions and asking parents, and I started seeing the vehicles idling and traffic jams and so on.

I tried to have conversations with parents, but most of them just didn't know anything about it.

So I put a survey out to the schools to understand what they knew about air pollution, and the result was quite shocking. No one really cared about it.

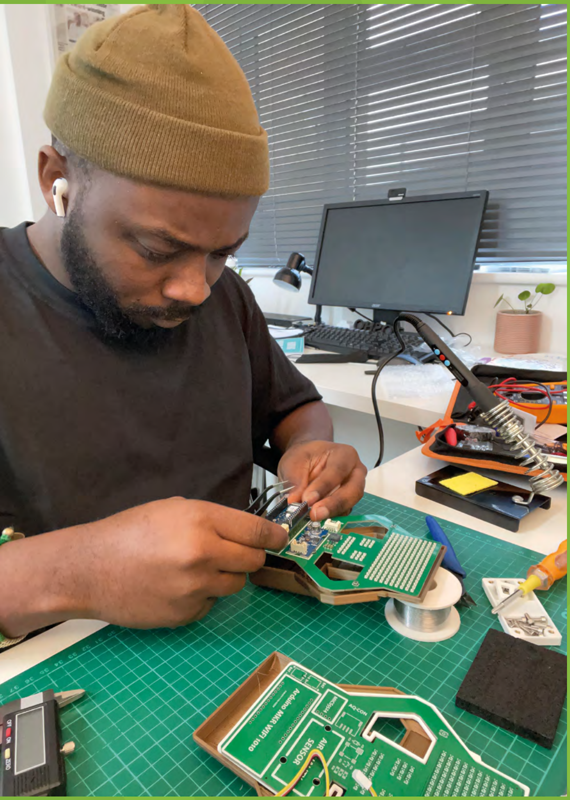
I had to find a way to bring about change. How do we get more Greta Thunbergs out there but in a passive way?

How can we make them be passive activists? As a parent, I'm doing my best within citizen science to bring that awareness. Once I finish talking, nothing happens in the morning, but the kids can carry on and push this further than I can.

That's where the idea for the Mindful Droid came about. I wanted to get kids involved in making and building something and actually understanding what the benefits are.

HS So, what does it do? Are there sensors onboard the Mindful Droid? Or does it pull data from other places?

MO The idea for the Mindful Droid is to be a mobile air quality monitor. But it's not just that. It also displays readings from existing outdoor air quality monitors. →



Above
Michael wants to give children and schools an easy way to demonstrate that their air quality needs to be better

The company that sponsored me, RS Components, created an outdoor air quality monitor. So the goal was to be able to pick up the API data from its readings and also be able to – let's say you are within school premises, for example, and you've got your Mindful Droid with you, it should be able to get the exact reading of what's going on around the school.

Then if you were to go away from the school, you can tune into the Mindful Droid's standalone sensors, which are VOC and carbon monoxide sensors.

The aim of this is to collect data. It's logging data in order to send that data to the local authorities where the action is being taken.

You push it to the people higher up, the policymakers. We can send them a constant stream of emails around the level of pollution around local schools, which probably isn't very nice for them, but that's the power of being a passive activist: you're not screaming in people's faces, but you're passively sending emails and getting the information across.

For now, we know that kids are actually learning, they're bringing awareness and they're letting people know that air pollution is killing us.

HS So you've managed to get the Mindful Droid into schools?

MO We did a demo with a school in Bradford, which is still in progress. The plan is to have the Mindful Droid be a free source of data collection for kids to have so that they can go beyond just what I have scoped for in my head. There's an opportunity there to go into the Internet of Things and hack it – that's the goal. Obviously it comes with a cost as well. So far, one school has the Mindful Droid and the outdoor monitor, but the hope is for every child to be able to build their own Mindful Droid from their home or from their classroom, and push it beyond its limits.

HS You designed it with the intention that it would be built by children; did that present you with any challenges?

MO Big time! At the moment, the challenge is with the PCB and the sensors, being able to put everything together. I'm trying to make it as simple to put together as possible, rather than presenting children with the technical challenges of let's say, soldering. How can we create something that's as easy as Lego for kids to put together?

So that's the vision; that's also been the challenge, to be honest. Because the next stage for the Mindful Droid is the independence of actually building one without going through the extensive work of learning how to solder, and splicing of wires and whatnot.

"It's usually the more economically deprived areas that are the most polluted"

HS That's quite a barrier, isn't it?

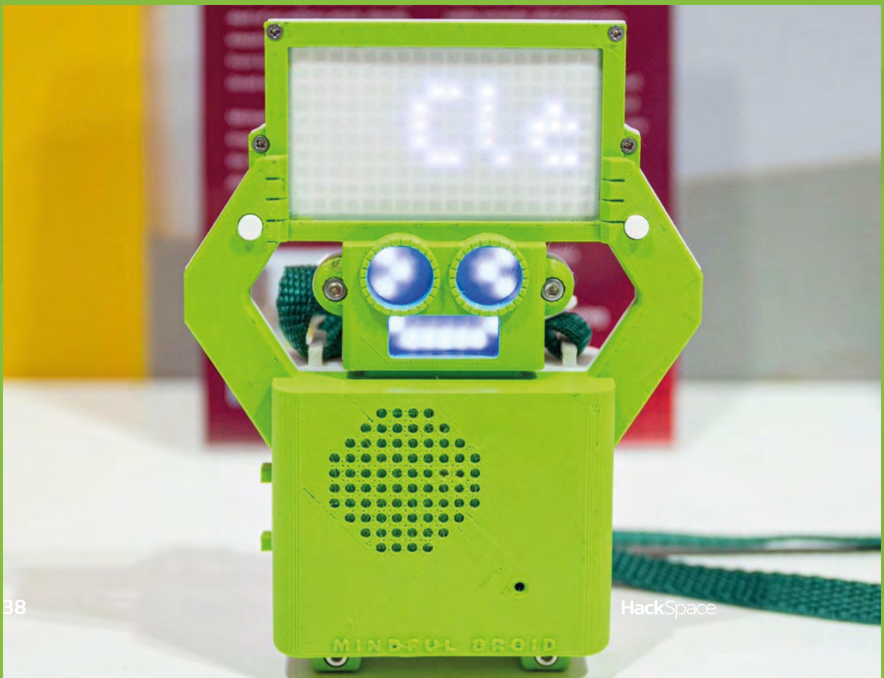
MO It is. I haven't soldered since I was in sixth form. Then I picked up a soldering iron when I started working on this and was like: wait a second, it's not the same as it was before. There are so many things that should have been obvious to me, but I didn't realise until I went on that journey myself. If it was a challenge for me, then I could imagine what a challenge it would be for children as well. So that's where it's like we need to make this simpler.

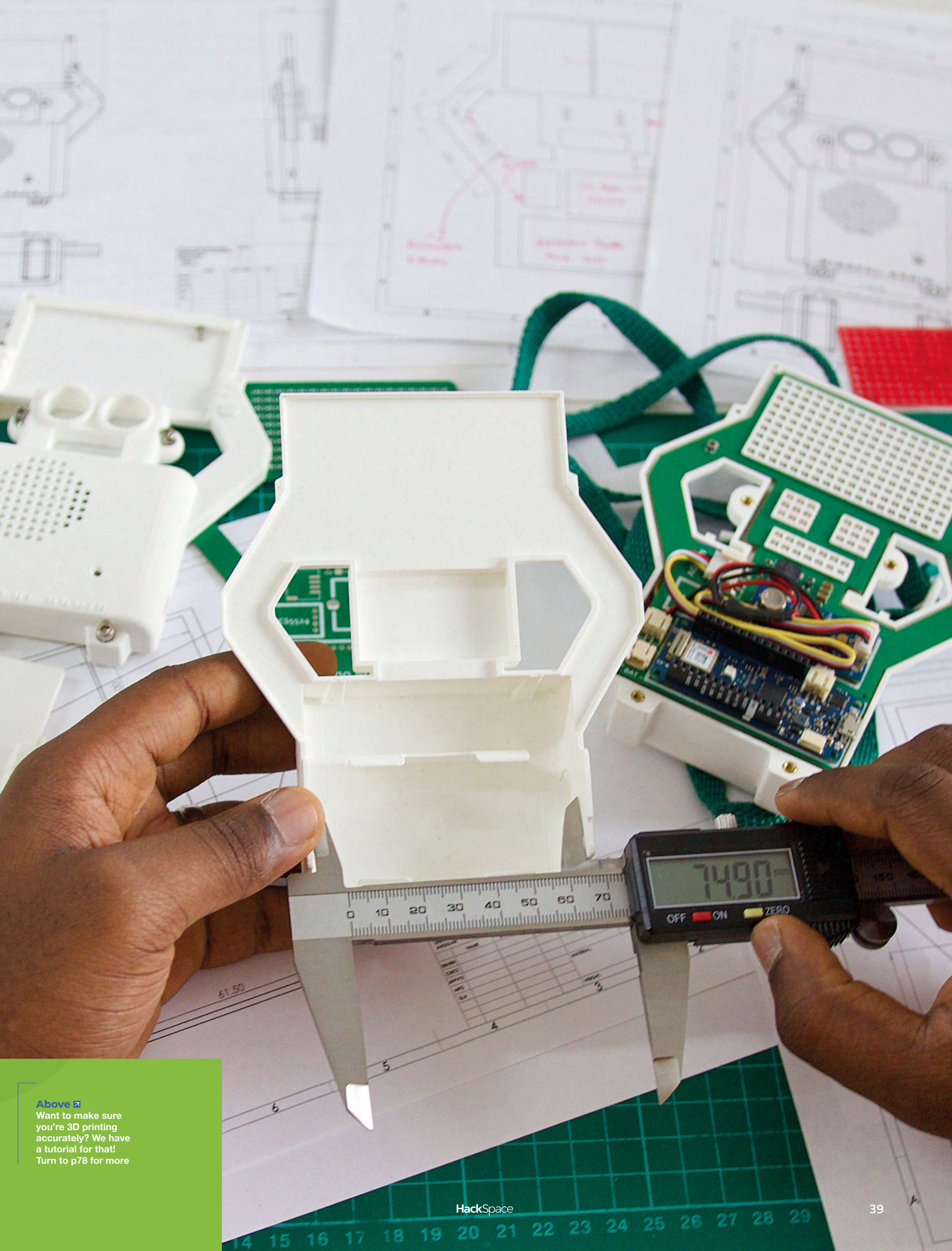
HS For anyone who doesn't know the topography of West Yorkshire, Bradford's quite hilly. Does that have an effect on pollution levels?

MO Yes, 100%. Bradford does have high pollutant levels. There's an organisation called Born in Bradford which monitors children, newly born kids and their growth levels and how pollution is contributing to their health. One of the areas of research is how air pollution is affecting children's brain development.

Another aspect of air pollution is that it's usually the more economically deprived areas that are the most polluted. →

Below Data is more useful on an LED matrix than it is locked away in a text file





Above ↗
Want to make sure you're 3D printing accurately? We have a tutorial for that! Turn to p78 for more

Bradford's a great place to be – there are so many things happening there right now, but we have to realise that this is a problem, and this is what they're doing to solve the problem.

HS OK, so what would success look like for Mindful Droid?

MO That's a very, very good question. And it's still something I'm figuring out. I think success for me is having children being able to build a Mindful Droid by themselves, and using it so stories like Ella Kissi-Debrah's doesn't come back to bite us again. That's what success looks like to me.

It's more about – I wouldn't say acceptance, but the will for people to have the awareness that this is what the dangers are, and this is how to do it. Rather than going out there to spend money to save yourself and keep yourself in better health, why not use these tools in the education system, in institutions, and say this is an opportunity for you to live better.

I want every child to understand the effects of air pollution and knowing that we're taking preventive measures. Informing people, and having the social responsibility to make things better. And like I said, the Mindful Droid is a template to just start something; if you want to put more activist information on it to make people stop doing what they're doing, then by all means, hack the life out of it and push it out there. That's the goal.

HS There's a phrase that keeps popping up in your video series about the development of the Mindful Droid: "validating the decision-makers", or rather, giving them the information to validate their choices. I guess the people in charge of city councils know that pollution is bad. But unless they can see how bad, it's not a priority.

MO 100%. It's making the invisible visible – we can all have debates in high places where they'll just move on to the next one after the allotted 30 minutes of council time or whatever. But if you constantly have your own children screaming at you saying this is bad, and we have data to show that it's bad, then in the back of your mind you're going to feel bad for not taking action.

So validating it from their own source is what matters. Rather than making it look secondary, it's primary to them. And that's where the action happens.

I feel like in the space of air pollution, there is no validation. Ella Kissi-Debrah's mom – she went through that experience. And she's done a lot in terms of Ella's Law and so many [other] things that she's doing to bring awareness. But she had to go through all that validation for herself to be able to bring that law to life.

It's a similar situation for all the policymakers out there – they need to go for that validation process as well. And if Mindful Droid gets into a household, then it can speak louder than just the words we say, because having a physical device giving you sensor readings is relatable.

HS Have you spoken to Ella's mum as part of the Mindful Droid development process?

MO Yes. And this feels great. She is looking into it further than I am because she's asking how we can actually eradicate air pollution, not just monitor it. She played a big role persuading the Mayor of London to introduce the Ultra Low Emissions Zone (ULEZ).

She's been very supportive of Mindful Droid – seeing that people outside of my bubble relate to it means it's going beyond just my passion project.

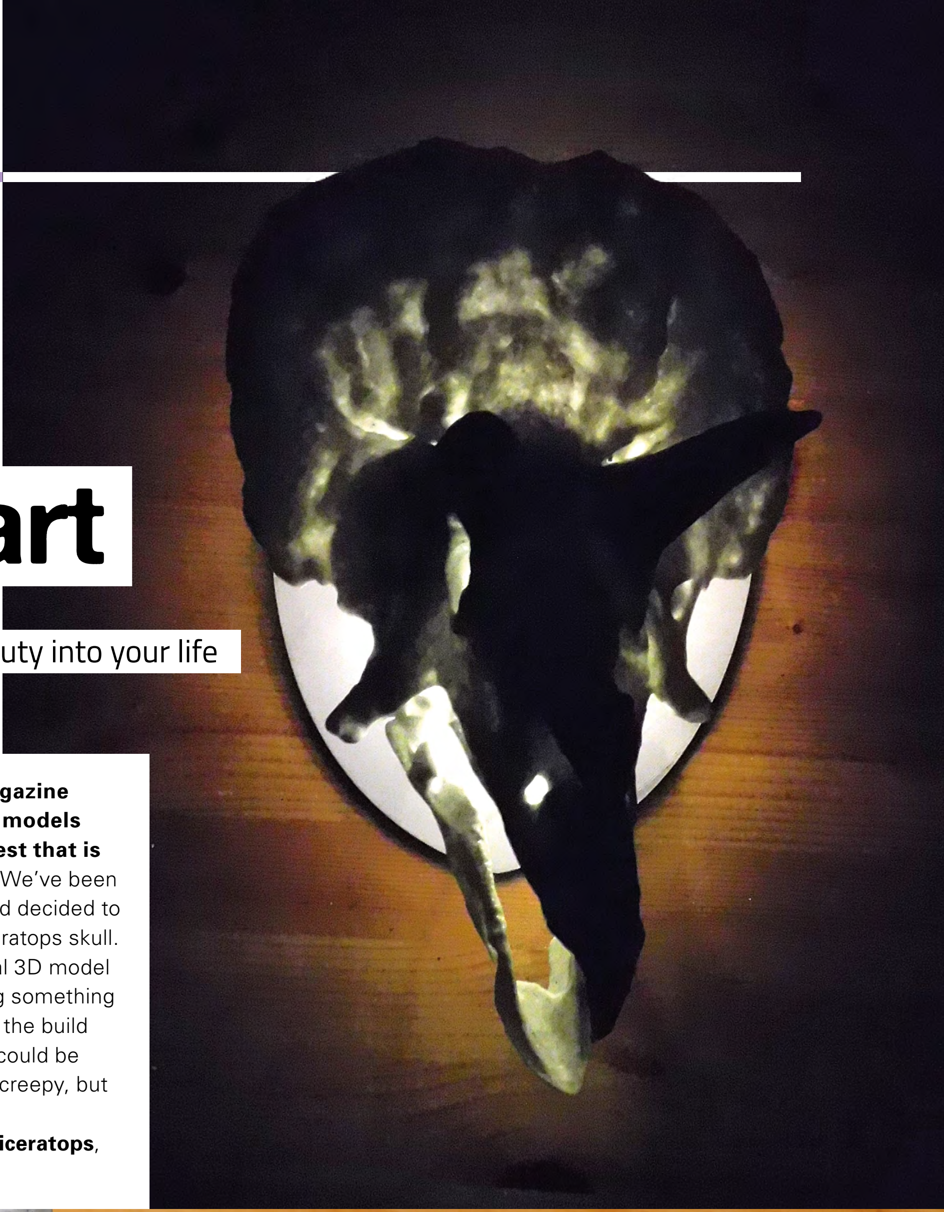
HS I've looked at scientific data that's come out of initiatives such as Bradford Clean Air Day, and I think I saw one statistic saying that 1 in 20 deaths in West Yorkshire is linked to air pollution, which is unbelievable.

MO Yes. It's very strange. And sometimes I don't understand how it's still so big. When I was in the thick of it before, I was so enraged by what was going on. And I think that's what got me into the Clean Air Day opportunities, because I was super-frustrated with the fact that, besides the ULEZ in London and a few clean air zones in other cities, in reality, nothing has actually been done to take action.

The frustrating part is that while we're moving to electric vehicles, it's really hard to justify why parents should stop driving their internal combustion engine-powered vehicles and switch to EVs, because again, the cost of it is not cheap.

From my point of view, the action is driving further away from school and walking in a few minutes to school, even when I'm late, because I know what I know about the effects of cars around schools. I'm trying to do better, and I think that's what really matters. Smaller levels, smaller changes, that's where the differences come in. And I think that's where it gets better for all of us. □

Left Making things out of cardboard isn't just for kids: real engineers use it all the time as a cheap, easy-to-work-with prototyping material

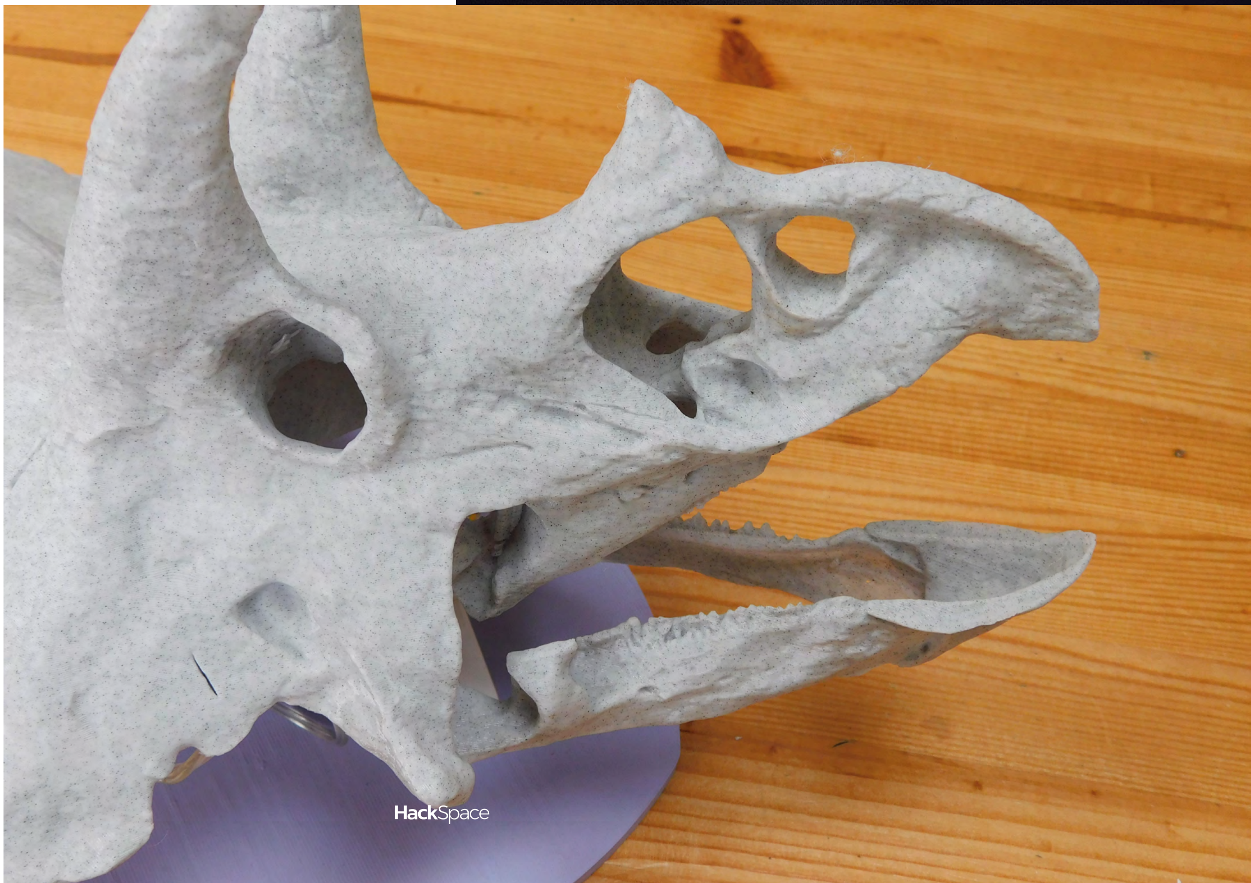


Objet 3d'art

3D-printed artwork to bring more beauty into your life

We've talked a few times in this magazine about how 3D scans of biological models contain a level of detail and interest that is rarely found in designed models. We've been testing out the Prusa XL recently, and decided to put the large print volume to the test with this Triceratops skull. We then created a mount for it (by using the original 3D model as a negative volume in PrusaSlicer, and so creating something that it mounted on perfectly). We finally completed the build by wrapping a string of LEDs behind the skull so it could be backlit. Apparently, some people find backlit skulls creepy, but we find them fascinating.

The skull's available for download at **hsmag.cc/triceratops**, and you can scale it to any print volume.



Above LEDs make everything better

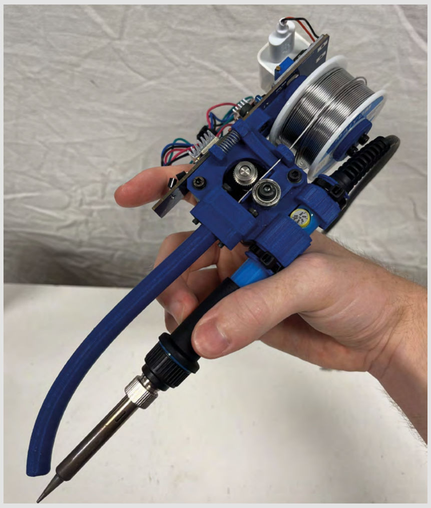
Right While their beak-like mouth is very bird-like, Triceratops did have teeth

Above
The largest Triceratops skull ever found was over 2.5 m high, so ours is a bit tiddly in comparison

Letters

ATTENTION ALL MAKERS!

If you have something you'd like to get off your chest (or even throw a word of praise in our direction) let us know at **hsmag.cc/hello**



ONE-HANDED

The solder sustainer looks great, but tell us honestly – is it any good? It looks more like a film prop from a dystopian cyberpunk epic than a maker tool, but I'm no good at soldering, and I'll take any help I can get.

Harry
Luton

Ben says: I have no idea; I've never tried it. However, the version we featured is already outdated. Take a look at the Solder Sustainer v2 here: hsmag.cc/sustainerv2.

EXTRA LARGE; EXTRA SPECIAL

After reading your review of the Prusa XL, it feels like it could become an era-defining 3D printer in the same way that the Ender 3 did when it brought reliable(ish) 3D printers to a whole new group of makers.

Mark
Coventry

Ben says: We're super-excited about the possibilities of the Prusa XL – and there's no doubt that it's capable of doing things that no other printer on the market can do. Era-defining? Maybe. The Ender 3 made 3D printing accessible to so many people by being a functioning 3D printer at a price people could afford. The Prusa XL is a great 3D printer, but is it at a price people can afford? These are difficult economic times, and £1700 is a lot of money, and that only gets you the single tool-head version.

BROKEN GLASS

It never clicked for me before that stained glass is soldered together. I'm not sure how I thought it was connected, but for some reason, a soldering iron just didn't seem like part of the process. After seeing the stained glass succulent in HackSpace issue 76, I've decided to give it a go.

Freddy
Frome

Ben says: **Electronics and stained glass seem a long way removed, don't they? There are specialist soldering irons that you can get for stained glass but, as far as I can tell, the only difference is that they can transfer heat a bit quicker, which means you can create the large blobs of solder quicker. Let us know how you get on.**

DEAD COWS

I really enjoyed Nicola King's article on leather craft. It's not a material I'd thought too much about before now, but looks like it could be a fun skill to learn, and might even prove to be useful.

Henry
Truro

Ben says: **Readers with phenomenal memories might remember that back in issue 1 of HackSpace mag, we went to Cheltenham Hackspace to learn how to make a leather belt. I still wear this belt almost every day. It might be psychological, but it just feels more comfortable than any of my shop-bought belts.**

HAPPY BIRTHDAY RASPBERRY PI

I can't believe it's been twelve years since Raspberry Pi first launched. Hats off to Eben and the whole team. I must admit I was a bit sceptical when they first launched (I thought it was just going to be a flash in the pan). Oh boy, have they proved me wrong.

Elizabeth
Glasgow

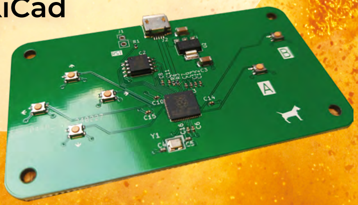
Ben says: **Absolutely. Due to launching on the 29th of February, Raspberry Pi just celebrated its third birthday, despite being twelve years old. This is traditionally named the Poseidon birthday – there's an old tradition of leap day birthdays being named after Greek gods, with the first birthday (after four years) being the Apollo birthday, the second (eight years) being Athena. Poseidon – where we are now. We'll see you all in another four years for the Dionysus birthday.**

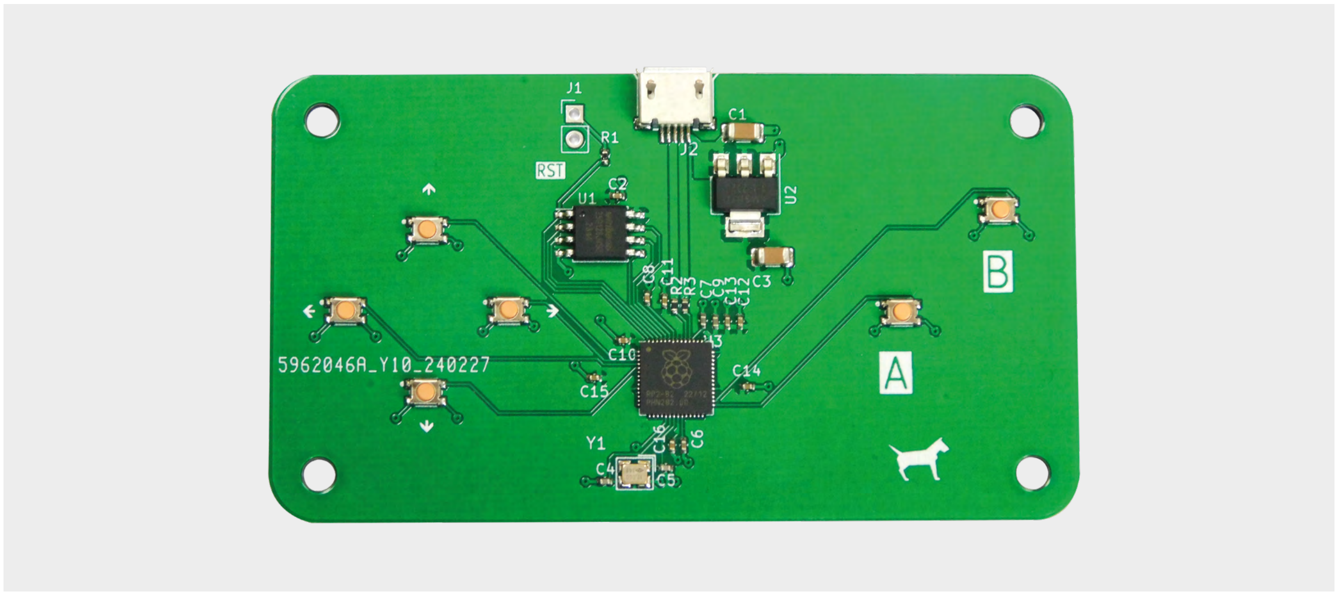
HackSpace
TECHNOLOGY IN YOUR HANDS

FORGE

HACK | MAKE | BUILD | CREATE

Improve your skills, learn something new, or just have fun tinkering – we hope you enjoy these hand-picked projects





KiCad: making an RP2040 game controller

Let's explore adapting our RP2040 layout to make a USB game controller

Jo Hinchcliffe

Jo Hinchliffe (AKA Concretedog) is a constant tinkerer and is passionate about all things DIY space. He loves designing and scratch-building both model and high power rockets, and releases the designs and components as open-source. He also has a shed full of lathes and milling machines and CNC kit!

Figure 1 The completed game controller PCB

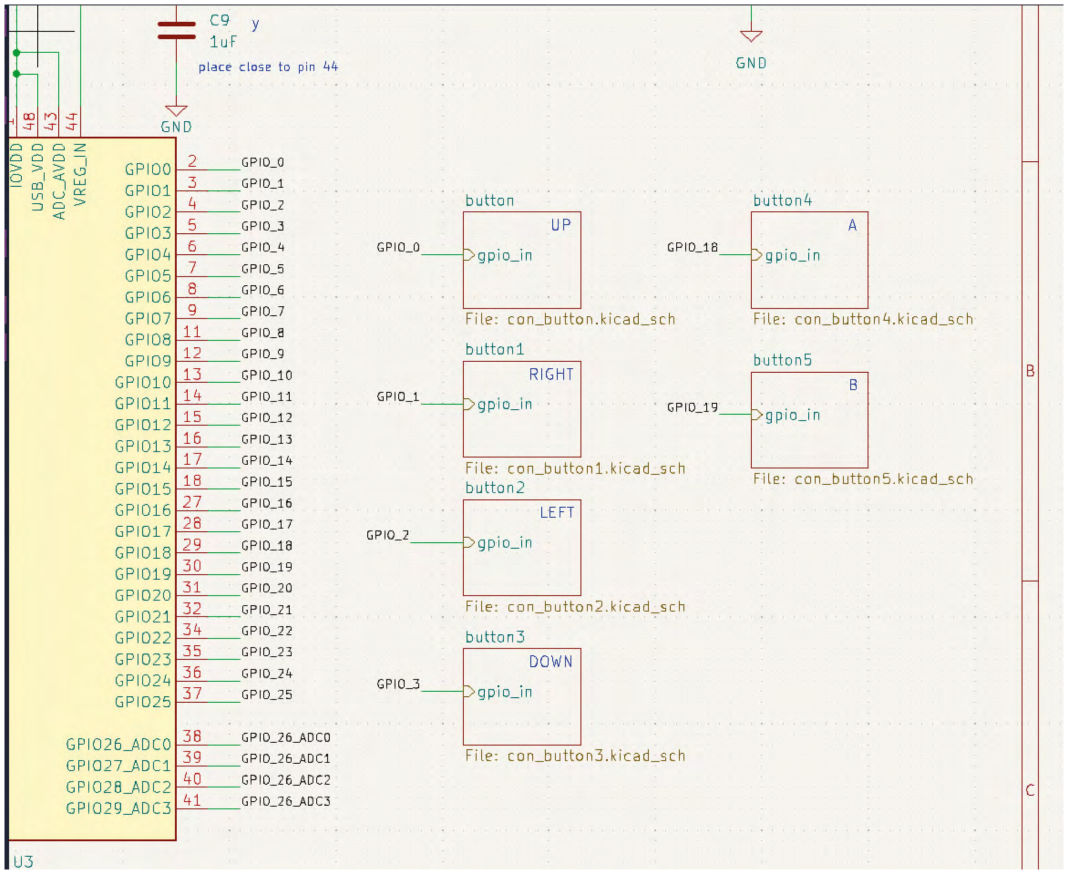
In earlier articles in this series, we established that we have a reasonable working RP2040 layout, so now it's pretty trivial to create new RP2040 devices. In the last section we made an 'Urumbu'-style motor driver board, and in this section we are going to create a simple USB game controller (**Figure 1**).

It's largely the same process we undertook for the Urumbu project, but simpler. We started by making a copy of our Urumbu project and cleaned out the files in the new project copy that we wouldn't need. So, any particular files like the board edge geometry, the Gerbers, and CSV files can be deleted as we will replace them with ones generated for the new project. We also quickly deleted all the Urumbu parts we didn't need from the schematic. Note that we don't really need to delete items in the PCB Editor, as when we eventually pull in the updated netlist and bill of materials, we can automatically delete unreferenced footprints, and the new footprints will be brought in.

We want to add six tactile buttons to our RP2040: four in a D-pad arrangement and two as A- and B-style buttons. We want these buttons to be momentary press buttons and 'push to make'. We then plan to use these buttons to connect one side to a GPIO and the other side of the button to ground.

Scouring the JLCPCB parts library, we came across the C221902 button. This part looked a nice size, so we took a look at the EasyEDA schematic and footprint. It has four pins and, reading the schematic, we could see that if we connected pin 2 to a GPIO and then connected all the other pins to ground, it would work as we wanted. Additionally, with the four SMD pads, it should mechanically be pretty strong.

With our choice made, we used the excellent Wokwi EasyEDA 2 KiCad website to convert the supplied footprint to a KiCad format: (**hsmag.cc/easyEDA2KiCad**). We then added it to our custom library. We covered this in earlier sections of this series, but it's pretty straightforward. You upload the EasyEDA JSON file, and it then downloads a KiCad PCB file with the footprint loaded into it.



To add the buttons to our schematic, we created a custom 4-pin schematic symbol and inserted it into a hierarchical sheet. We wired the GPIO pin and the other pins to ground and then brought out the GPIO hierarchical pin. We then copied the hierarchical sheet to create six versions, one for each button, adjusting the label and the sheet name as we added each (**Figure 2**). Again, we've covered this in earlier sections.

Next, we assigned the new footprints to the schematic symbols and began to edit the PCB layout. We created a new board edge geometry SVG in Inkscape with some mount holes before carrying out the usual exporting of Gerbers, BOM, and positional files for JLCPCB services (**Figure 3**).

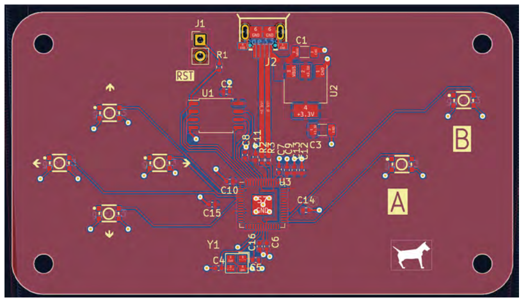


Figure 2 Using hierarchical sheets makes it easy to add multiple similar connected schematic blocks, such as the buttons

Figure 3 Our completed PCB layout

"**Additionally, with the four SMD pads,** our PCB should mechanically be pretty strong"

Having ordered the boards, one final fun activity on the hardware side of this build was to export a STEP file from KiCad to model around in FreeCAD. To export a basic STEP file from the KiCad PCB Editor, select File > Export and then choose STEP as the output format. Note that we haven't added custom 3D models for all of our custom components, so obviously the STEP file isn't completely correct, but it serves as a good enough guide to model around in FreeCAD. →

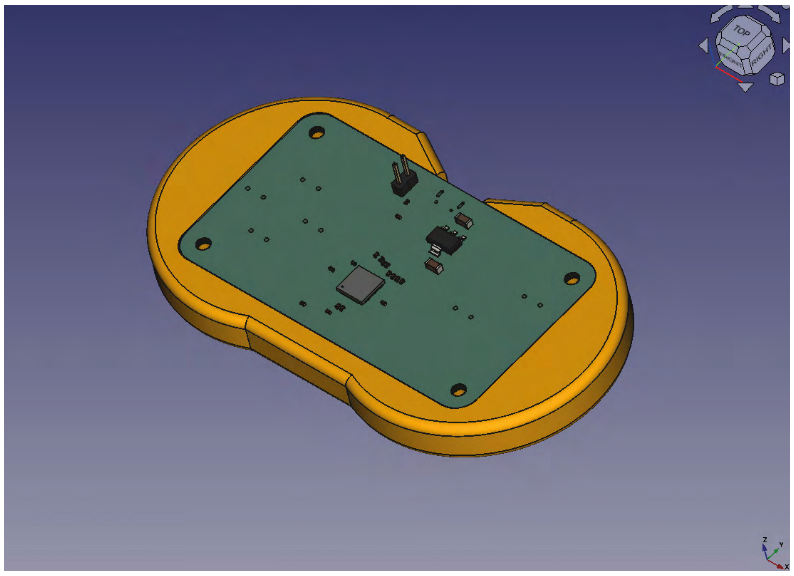


Figure 4
Modelling a simple enclosure in FreeCAD to make our controller a little more comfortable to hold

HIGH PERFORMANCE

In this tutorial, we've looked at creating a gamepad that's easy to understand and extend. However, if you're looking to build a high-performance gamepad, then there are lots of things that you need to take into account. Part placement is obviously a large part of it, as you need to be able to press buttons consistently and accurately.

However, another part is the software. Our CircuitPython code could be improved, but ultimately, if you're looking for high performance, CircuitPython isn't the right choice. Fortunately, there is another option.

GP2040-CE is a firmware for RP2040-based devices. You can configure it with details of what hardware is connected where. It understands more than just buttons, so you can add analogue inputs as well.

There's documentation on the project website: **hsmag.cc/GP2040-CE**.

In the free-to-download book *FreeCAD For Makers*, we explored using FreeCAD and the KiCad StepUp workbench that allows you to create and position custom 3D component models for use in KiCad. We also explored all the skills needed to create all kinds of models. With the knowledge you gain from this book, you could certainly make a controller enclosure like the one we quickly modelled (**Figure 4**).

THE SOFTER SIDE

Now we have created our board, it's time to write some code for it. We could write our code in C using the Pico SDK. We could also use the Pico build of MicroPython or CircuitPython. However, since we've created a new board, let's create a firmware tailored specifically for it – we'll create a custom build of CircuitPython. This allows us to do a couple of things. Firstly, it lets us name the specific pins, so rather than using, say, GPIO0, we can use BTN_A. Secondly, it lets us select which modules we want to include. In our case, we'll add Adafruit HID, which enables us to use our game controller as an input device.

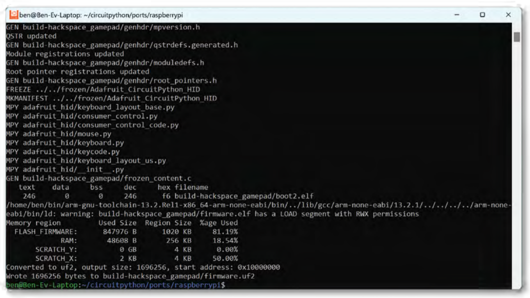
The general process for creating a build of CircuitPython is given in the documentation at **hsmag.cc/BuildCP**. We won't go through it in detail, so follow that guide to set up your environment.

Once you have everything set up, you need to create this board. In the directory **circuitpython/ports/raspberrypi/boards**, copy the Raspberry Pi Pico directory into a new one named appropriately for the gamepad. We've called ours **hackspace_gamepad**.

There are two files that we need to adjust to take into account our board. Firstly, there's **pins.c**, which should have the following:

```
#include "shared-bindings/board/__init__.h"

STATIC const mp_rom_map_elem_t board_module_
globals_table[] = {
    CIRCUITPYTHON_BOARD_DICT_STANDARD_ITEMS

    { MP_ROM_QSTR(MP_QSTR_UP), MP_ROM_PTR(&pin_
GPIO0) },
    { MP_ROM_QSTR(MP_QSTR_RIGHT), MP_ROM_PTR(&pin_
GPIO1) },
    { MP_ROM_QSTR(MP_QSTR_LEFT), MP_ROM_PTR(&pin_
GPIO2) },
    { MP_ROM_QSTR(MP_QSTR_DOWN), MP_ROM_PTR(&pin_
GPIO3) },
    { MP_ROM_QSTR(MP_QSTR_BTN_A), MP_ROM_PTR(&pin_
GPIO18) },
    { MP_ROM_QSTR(MP_QSTR_BTN_B), MP_ROM_PTR(&pin_
GPIO19) }
};
MP_DEFINE_CONST_DICT(board_module_globals, board_
module_globals_table);
```

In this, we're adding items to the board module. Specifically, one for each button.



```
GEN build-hackspace_gamepad/genhdr/mpversion.h
QSTR updated
GEN build-hackspace_gamepad/genhdr/qstrdefs.generated.h
Module registrations updated
GEN build-hackspace_gamepad/genhdr/moduledefs.h
Root pointer registrations updated
GEN build-hackspace_gamepad/genhdr/root_pointers.h
FREEZE ../../frozen/Adafruit_CircuitPython_HID
MKMANIFEST ../../frozen/Adafruit_CircuitPython_HID
MPY adafruit_hid/keyboard_layout_base.py
MPY adafruit_hid/consumer_control.py
MPY adafruit_hid/consumer_control_code.py
MPY adafruit_hid/mouse.py
MPY adafruit_hid/keyboard.py
MPY adafruit_hid/keycode.py
MPY adafruit_hid/keyboard_layout_us.py
MPY adafruit_hid/__init__.py
GEN build-hackspace_gamepad/frozen_content.c
   text    data     bss     dec     hex filename
    246       0       0     246      f6 build-hackspace_gamepad/boot2.elf
/home/ben/bin/arm-gnu-toolchain-13.2.Rel1-x86_64-arm-none-eabi/bin/../lib/gcc/arm-none-eabi/13.2.1/../../../../arm-none-
eabi/bin/ld: warning: build-hackspace_gamepad/firmware.elf has a LOAD segment with RWX permissions
Memory region         Used Size  Region Size  %age Used
  FLASH_FIRMWARE:      847976 B      1020 KB     81.19%
             RAM:       48608 B       256 KB     18.54%
       SCRATCH_Y:          0 GB         4 KB      0.00%
       SCRATCH_X:          2 KB         4 KB     50.00%
Converted to uf2, output size: 1696256, start address: 0x10000000
Wrote 1696256 bytes to build-hackspace_gamepad/firmware.uf2
ben@Ben-Ev-Laptop:~/circuitpython/ports/raspberrypi$
```

Left We found it easiest to build CircuitPython using Windows Subsystem for Linux

"We'll add Adafruit HID, which enables us to use our game controller as an input device"

OTHER GAMEPADS

This example should get you started in the world of game controllers, and there are loads that you can look at for inspiration:

- The Arduino Esplora is now retired, but was one of the first hackable game controllers on the market: **hsmag.cc/ArduinoEsplora**
- There's an online community at PCBWay's shared projects site that includes many game controllers, including: **hsmag.cc/PicoGamepad**
- Gamepads come in many shapes. They're usually designed around ergonomics, but you can get a little creative. For example, this maker has built a bat-shaped controller: **hsmag.cc/BatController**

Next, we need to edit **mpconfigboard.mk** to be the following:

```
USB_VID = 0x1209
USB_PID = 0xB182
USB_PRODUCT = "HackSpace gamepad"
USB_MANUFACTURER = "HackSpace magazine"

CHIP_VARIANT = RP2040
CHIP_FAMILY = rp2

EXTERNAL_FLASH_DEVICES = "W25Q128JVxQ"

CIRCUITPY__EVE = 1


FROZEN_MPY_DIRS += $(TOP)/frozen/Adafruit_
CircuitPython_HID
```

In this file, we define the type of flash chip we have and also add any 'frozen' modules we want. Frozen modules can be anything that we want to be included on the build by default (other than the core modules that are automatically included). Frozen modules have to be in the **circuitpython/frozen** directory, but you should find that the Adafruit_CircuitPython_HID module is already there.

You can now create your build by going to **circuitpython/ports/raspberrypi** and running:

```
make BOARD=hackspace_gamepad
```

This will compile your code, and you should end up with a **build-hackspace_gamepad** directory. In there, you'll find a **firmware.uf2** file that you can load onto your gamepad just as you would any other UF2 file.

Obviously this isn't complete firmware as it's only the programming language. We now need to write a program to get everything working. Fortunately, we have all the modules we need baked in, so there's no need for anything there. We've drawn inspiration from the CircuitPython example code here: →

```
import time
import board
import digitalio
import usb_hid
from adafruit_hid.keyboard import Keyboard
from adafruit_hid.keyboard_layout_us import
KeyboardLayoutUS
from adafruit_hid.keycode import Keycode

# A simple neat keyboard demo in CircuitPython

# The pins we'll use, each will have an internal
pullup
keypress_pins = [board.UP, board.DOWN, board.LEFT,
board.RIGHT, board.BTN_A, board.BTN_B]
```

LEAD-FREE

It's often cheaper to get boards made using leaded solder. However, this might be a false economy. Leaded solder is harmful to both your health and the health of our planet. In the case of a games controller – something that you're going to hold in your hand time and again – it's more important than usual to opt for lead-free solder. Even if only a tiny amount gets on your hands each time you use it, that will still add up over the course of the controller's life, and could have negative effects on your health.

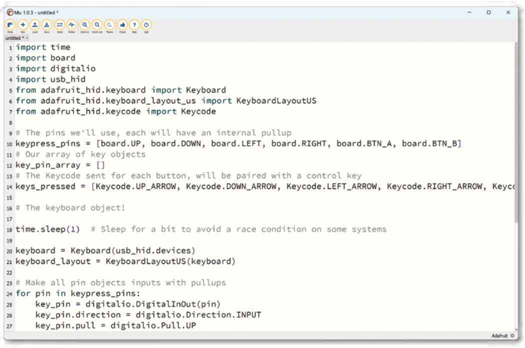
```
# Our array of key objects
key_pin_array = []
# The Keycode sent for each button, will be paired
with a control key
keys_pressed = [Keycode.UP_ARROW, Keycode.DOWN_
ARROW, Keycode.LEFT_ARROW, Keycode.RIGHT_ARROW,
Keycode.A, Keycode.B]

# The keyboard object!

time.sleep(1)  # Sleep for a bit to avoid a race
condition on some systems

keyboard = Keyboard(usb_hid.devices)
keyboard_layout = KeyboardLayoutUS(keyboard)

# Make all pin objects inputs with pullups
for pin in keypress_pins:
    key_pin = digitalio.DigitalInOut(pin)
    key_pin.direction = digitalio.Direction.INPUT
    key_pin.pull = digitalio.Pull.UP
    key_pin_array.append(key_pin)

print("Waiting for key pin...")

while True:
    # Check each pin
    for key_pin in key_pin_array:
        i = key_pin_array.index(key_pin)
        key = keys_pressed[i]
        if not key_pin.value:  # Is it grounded?
            print("Pin #%d is grounded." % i)
            # "Type" the Keycode or string
            keyboard.press(key)  # "Press"...
        else:
            keyboard.release(key)
    time.sleep(0.01)
```

As you can see, we can use **board.UP**, **board.DOWN**, **board.LEFT**, **board.RIGHT**, **board.BTN_A**, and **board.BTN_B** in our code. This has a couple of advantages. Firstly, it is more intuitive for programmers. Secondly, if we created another version of the board with the buttons on different pins, the same code could still run on both.

This code is a bit lazy. For example, there's no debouncing on the buttons. In practice, we've found that this doesn't cause many problems, especially with the **time.sleep(0.01)** in there. This means it's not the most responsive controller, so if you're playing games where hundredths of a second matter, you probably want to use something different, including tuned debouncing, written in C. However, this controller isn't suitable for that type of game anyway. This is also fairly cavalier with the number of reports it sends (a report being a status update sent from keyboard to computer). This will send six of them every loop, which means several hundred a second. Again, this isn't great for performance. However, it works reliably and is easy to understand.

With this code loaded, you should be able to plug the controller into any computer and it will recognise it as a USB keyboard. Press one of the buttons and the computer should recognise that button press just as it would from any keyboard. With this, you can control any game that takes input from a computer.

Creating a custom version of CircuitPython isn't essential when you build a new board; however, once you've been through the process once, it's easy, and makes life a little bit nicer, especially if you're distributing the board to other people. □



```
import time
import board
import digitalio
import usb_hid
from adafruit_hid.keyboard import Keyboard
from adafruit_hid.keyboard_layout_us import KeyboardLayoutUS
from adafruit_hid.keycode import Keycode

# The pins we'll use, each will have an internal pullup
keypress_pins = [board.UP, board.DOWN, board.LEFT, board.RIGHT, board.BTN_A, board.BTN_B]
# Our array of key objects
key_pin_array = []
# The Keycode sent for each button, will be paired with a control key
keys_pressed = [Keycode.UP_ARROW, Keycode.DOWN_ARROW, Keycode.LEFT_ARROW, Keycode.RIGHT_ARROW, Keyco

# The keyboard object!

time.sleep(1)  # Sleep for a bit to avoid a race condition on some systems

keyboard = Keyboard(usb_hid.devices)
keyboard_layout = KeyboardLayoutUS(keyboard)

# Make all pin objects inputs with pullups
for pin in keypress_pins:
    key_pin = digitalio.DigitalInOut(pin)
    key_pin.direction = digitalio.Direction.INPUT
    key_pin.pull = digitalio.Pull.UP
```

Right Our custom build of CircuitPython brings everything we need, including pin names and modules the

COMPUTERS THAT MADE BRITAIN

"The Computers That Made Britain is one of the best things I've read this year. It's an incredible story of eccentrics and oddballs, geniuses and madmen, and one that will have you pining for a future that could have been. It's utterly astonishing!"

- **Stuart Turton**, bestselling author and journalist

Buy online: **hsmag.cc/ctmb**

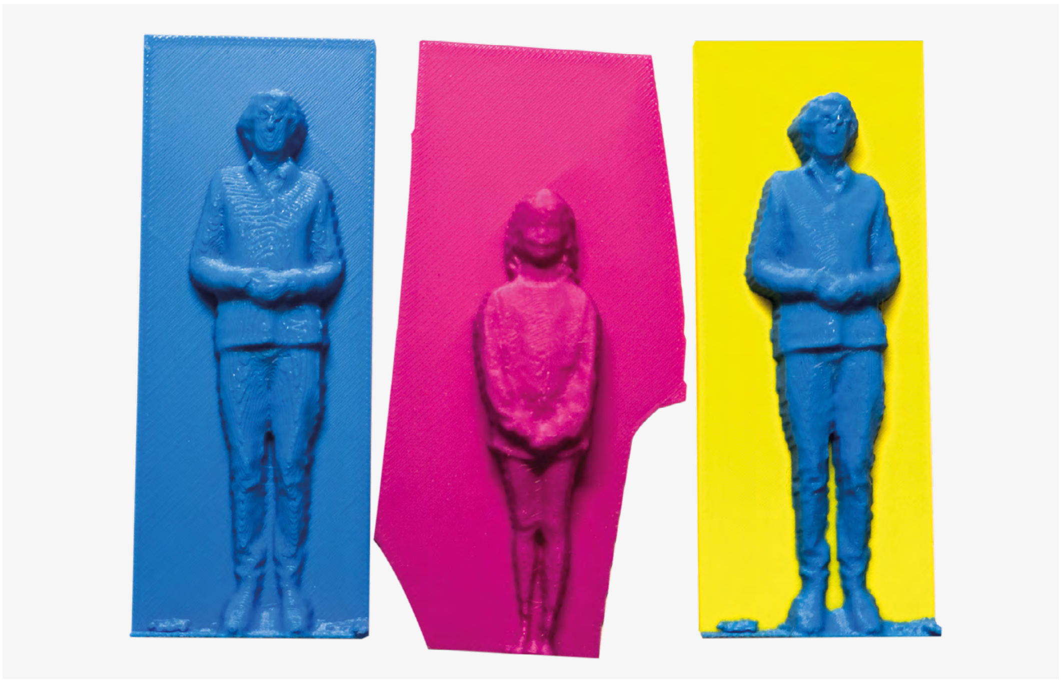
Available on amazon

STL files and the Kinect Carbonizer

Discover how 3D print files work and embed your friends in carbonite

Rob Miles

Rob has been playing with hardware and software since almost before there was hardware and software. You can find out more about his so-called life at **robmiles.com**.



QUICK TIP

Note that you can express a particular direction using lots of different vector values. The vectors (0, 0, 1), (0, 0, 2) and (0, 0, 3) all point in the same direction – which is straight up.

In this article, we are going to discover how we can create and manipulate solid objects in computer code. We're going to start with a look at stereolithography (STL) files and discover how these describe objects for a 3D printer to print. Then, we are going to move on to making interesting and colourful designs from mathematical functions and, finally, use a Kinect camera to create our own version of the best scene from the best *Star Wars* movie – the bit where Han Solo is encased in a block of carbonite so that only the top part of his body (including his scary screaming face) is visible sticking out of the block. (Spoiler alert: Don't worry – he is freed in the next film).

Figure 1, above, shows the result of the author 'carbonizing' himself and another family member. The two figures on the left were printed using a 0.2 mm layer height, and took around half an hour to print. The right-hand image was printed using a 0.1 mm layer height, and took an hour to print. You can find the program and links to all the software you will need at the GitHub repository for this project: **hsmag.cc/Kinect3DPrinter**.

However, before we can discover how to carbonize people, we need to learn a bit about how the process works, starting with the STL file which contains the carbonized object. And, before we can talk about the positions of things in an STL file, we also need to meet a couple of characters whose names sound as though they came from a superhero movie.

Figure 1 We are planning a carbonite group picture

VERTEX VS VECTOR

Vectors and vertices are expressed as coordinates. They are not superheroes. A coordinate gives a

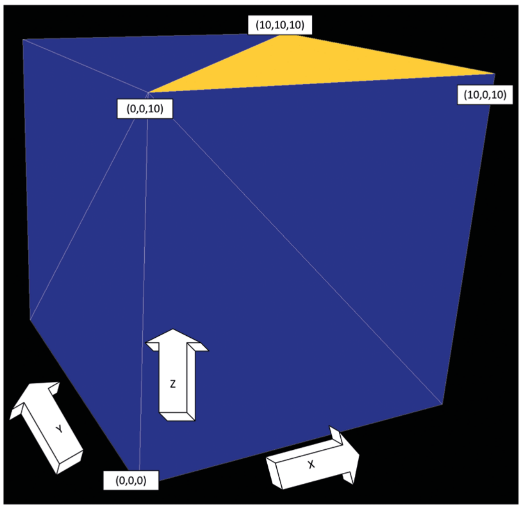


Figure 2
The author took a long time to draw this diagram. He hopes you will spend a long time studying it

position in space. It is like an instruction on a pirate treasure map. For example, a grizzled old pirate might write: 'Start ye at the Olde Oake Tree and walk ye twelve paces east and six paces north and there ye must dig down two paces deep.' We could express this as a coordinate of (12, 6, -2) with the origin (0, 0, 0) at the Olde Oake Tree.

In this coordinate system, we are equating east and west with X. The X coordinate increases if we go east and decreases if we go west. The Y coordinate increases if we go north and decreases if we go south with Y. The Z coordinate increases when we go up and decreases when we go down. We could use this system to express the position of any point in space relative to the Olde Oake Tree. A point in 3D space is called a vertex.

DIGGING DOWN

The value of Z in our pirate coordinates for treasure location is negative because we have decided that positive values of Z go up, and the treasure is buried below ground level. It is important that we get this right, otherwise the pirates might expect the treasure chest to be hovering in the air two paces above the ground. In other words, to properly describe things in a 3D space, we also need to be able to express direction. We use a vector to do this. You can think of a vector as an arrow.

The 'tail' of the arrow is at the point (0, 0, 0) and the pointy bit of the arrow (to use a technical term) is at the vector coordinates. We look along the arrow to determine the direction the vector is expressing. We could say that the Z axis for our pirate treasure map points along the vector (0, 0, 1), which would be an arrow pointing up into the sky. We can describe the X and Y directions using vectors as well. In the case of our pirate coordinate system, the X axis is on the vector (1, 0, 0) and the Y axis is on the vector (0, 1, 0).

Computers love vectors and vertices because of the way that they can use simple mathematical transformations (often using things called matrices) to work on their values. Now that we know how we can express position (vertex) and direction (vector), we can describe a 3D shape using them.

MAKING A CUBE FROM TRIANGLES

Any object that has edges which are straight lines can be expressed in terms of triangles. **Figure 2** shows a cube that is made up of twelve triangles. The figure shows the six triangles on the front and top of the cube. The other six are not visible because they are on the back and the bottom of the cube.

One of the triangles (the yellow one on the top) has been annotated with the positions of each of the three vertices (aka vertexes) which describe it. The full description of the cube uses twelve triangles so we can store the entire cube description as 36 vertices, three for each of the twelve triangles. An STL file does just that. →

YOU'LL NEED

- **A Windows PC** to build and run the software
- **A 3D printer** (preferably one that can switch between different filaments)
- **An Xbox One Kinect sensor** (you might have one in the loft)
- **A Kinect PC adapter** (Search your favourite e-commerce sites for 'Kinect PC adapter')

Right The key question you need to consider is 'Which way does depth (Z) go?'

QUICK TIP

When you are playing a video game, this drawing process is taking place with thousands of triangles many times a second. A graphics card contains hardware specially designed to perform vector calculations and to map images (textures) onto triangles.

MAKING A CUBE IN STL

An STL file isn't much more than a list of triangles. The STL description of the yellow triangle in **Figure 2** looks like this:

```
facet normal 0 0 1
   outer loop
     vertex 0 0 10
     vertex 10 0 10
     vertex 10 10 10
   endloop
 endfacet
```

Each triangle description is enclosed between `facet` and `endfacet` elements. A facet is a flat element that is part of a surface. Apparently, the surface of a diamond is made up of facets, although the author has never got close enough to a diamond to check this out.

The facet description contains an `outer loop` enclosing the three vertices, one for each point of the triangle. There is also a `normal` vector. What does that mean? The `normal` vector expresses the direction which is perpendicular to that facet. In the case of the top facet, the normal value is (0, 0, 1), which is straight up.

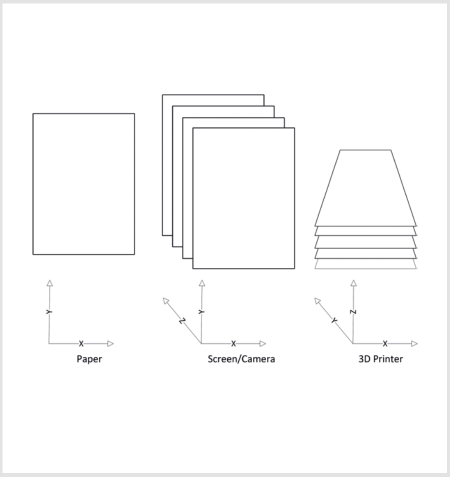
The normal vector of a facet is used to work out which way a facet is facing. If you look at **Figure 2**, you will see that the triangles that describe the back of the cube have not been drawn. The program drawing the cube has placed the 'camera' at a particular position and then only drawn the triangles which have normal vectors that point towards the position of the camera. The good news for us is that 3D editors will calculate the normal values for a model automatically, so we don't need to compute these values when we create STL files of our own.

Some of this will probably have hurt your head. But you now know how STL files and 3D graphics work. When we design something, we can break the design into triangles and then save these in an STL file. Then we feed this file into a 'slicer' program which converts it into instructions which tell our 3D printer how to make the described object.

CONFUSING COORDINATES

A coordinate lets you specify positions using their distance from an origin (the place in the coordinate space with all coordinate values of zero). A coordinate is two values (for flat 2D surfaces) or three values (for 3D). The three coordinate values can be called X, Y, and Z. If you are drawing a graph on a piece of paper, you put the origin in the bottom left-hand corner and the value X describes how far across the page a point is, and the value Y describes how far up the page the point is.

When you want to express three dimensions, you need to add a Z value to the coordinate, and this is where it gets annoying. If you are viewing the graph on a screen in front of you, the natural thing to do is use Z to express the third dimension as the distance into the screen. However, if you are printing on a 3D printer (which builds things up in layers), you would like to use Z as the distance up from the baseplate.



The figure above shows how this all fits together. An STL file uses the '3D printer' view of coordinates. When the author was learning how to perform 3D design, he was coming from a background of video game writing, in which the Z value goes 'into' the screen. It took him a while to come to terms with the way that 3D designs work.

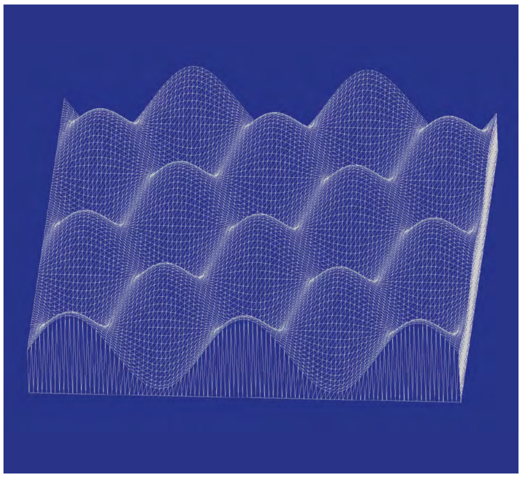


Figure 3
We add detail to the mesh by making the triangles smaller

MAKING A MESH OF THINGS

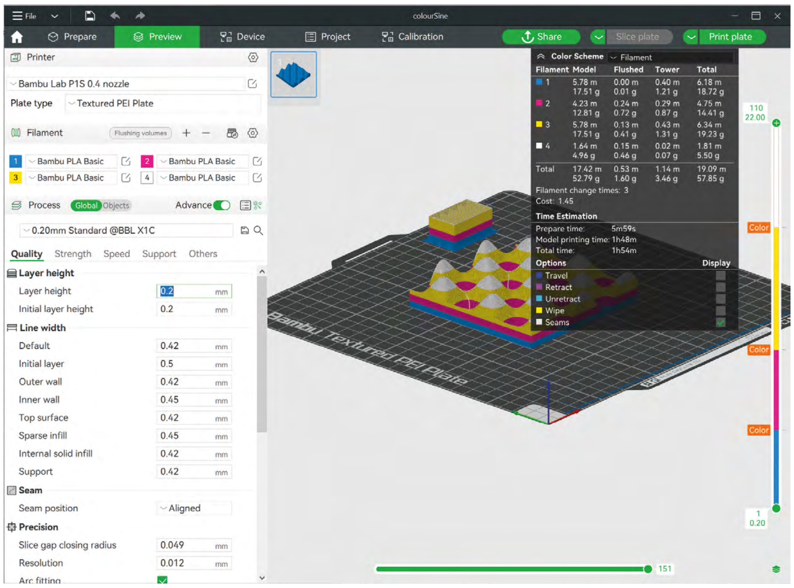
We could express our object shapes in terms of lots of tiny cubes (called voxels – short for volumetric pixel), but it is more useful to think of a 3D object as a mesh of triangles. **Figure 3** shows how we can use a mesh to express a surface which expresses the equation:

```
Z = Sin(x) + Cos(y)
```

If you look at **Figure 3**, you will see that the surface is made up of tiny triangles. The program that generated the mesh calculated the height value for a range of values for X and Y, and then generated a mesh with triangles which had vertices at those positions. It then added solid sides and a base and saved the resulting object as an STL file.

Once you have your STL file, the next thing to do is to slice it into layers to be sent for printing. **Figure 4**, above, shows this process in action using the Bambu Studio program (download it from **hsmag.cc/BambuLab**). You will need a compatible printer to be able to print in multiple colours.

Figure 5 shows the print from this design. You could use this technique to make 3D-printed versions of any mathematical equation or set of data values. This can be a very useful way to visualise data. If you want to do this kind of thing in the Python language, the Trimesh library (**trimesh.org**) is a great place to get started. You can use it to create meshes from software and export these to STL files for slicing and printing. Then, you too can print out some surfaces that you might not have any use for. →



Printer
Bambu Lab P1S 0.4 nozzle
Plate type: Textured PEI Plate

Filament (Flushing volumes)

1 Bambu PLA Basic	2 Bambu PLA Basic
3 Bambu PLA Basic	4 Bambu PLA Basic

Process: Global Objects — Advance
0.20mm Standard @BBL X1C
Quality | Strength | Speed | Support | Others

Layer height	
Layer height	0.2 mm
Initial layer height	0.2 mm
Line width	
Default	0.42 mm
Initial layer	0.5 mm
Outer wall	0.42 mm
Inner wall	0.45 mm
Top surface	0.42 mm
Sparse infill	0.45 mm
Internal solid infill	0.42 mm
Support	0.42 mm
Seam	
Seam position	Aligned
Precision	
Slice gap closing radius	0.049 mm
Resolution	0.012 mm
Arc fitting	✓

Color Scheme: Filament

Filament	Model	Flushed	Tower	Total
🟦 1	5.78 m 17.51 g	0.00 m 0.01 g	0.40 m 1.21 g	6.18 m 18.72 g
🟥 2	4.23 m 12.81 g	0.24 m 0.72 g	0.29 m 0.87 g	4.75 m 14.41 g
🟨 3	5.78 m 17.51 g	0.13 m 0.41 g	0.43 m 1.31 g	6.34 m 19.23 g
⬜ 4	1.64 m 4.96 g	0.15 m 0.46 g	0.02 m 0.07 g	1.81 m 5.50 g
Total	17.42 m 52.79 g	0.53 m 1.60 g	1.14 m 3.46 g	19.09 m 57.85 g

Filament change times: 3
Cost: 1.45

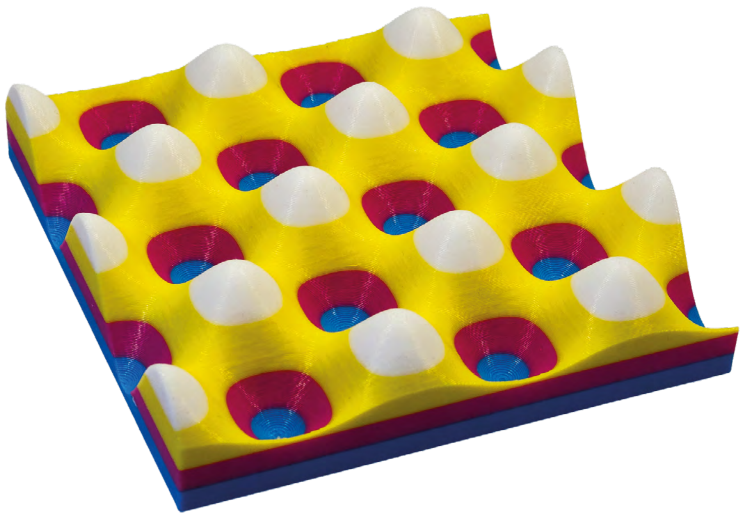
Time Estimation
Prepare time: 5m59s
Model printing time: 1h48m
Total time: 1h54m

Options	Display
Travel	
Retract	
Unretract	
Wipe	
Seams	✓

"You could use this technique to make 3D-printed versions of any mathematical equation or set of data values"

Figure 4
The slicer lets you set filament colours for different layers in the print

Figure 5
The author is very pleased with this print, although he is not sure what to do with it



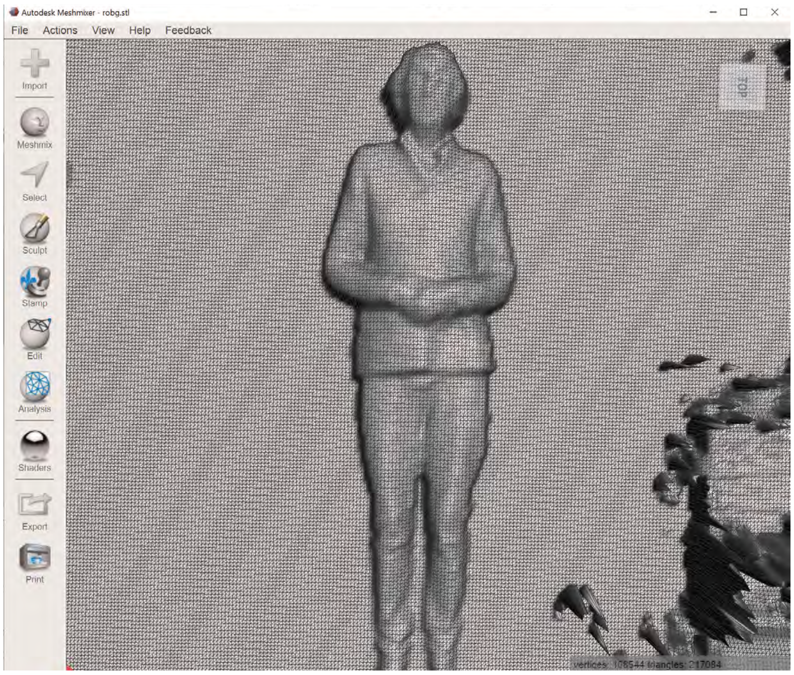


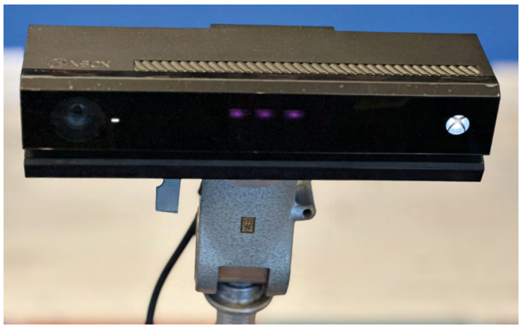
Figure 6 Meshmixer is a free download from meshmixer.com

Figure 7 The depth camera is on the left of the device. The sensor also contains four microphones and a high-resolution webcam

QUICK TIP

It turns out that there is a lot of common ground between 3D printing and video games. The 3D design tool Blender (**blender.org**) used by game creators can also be used to create STL files for 3D printing.

When we make the 'carbonized' STL file, we create a mesh which contains triangles that describe the carbonized victim. **Figure 6**, above, shows the Meshmixer program displaying the mesh that was used to create the carbonized Rob in **Figure 1**. You can see that there is more detail in the STL file than the printer has rendered. Now that we know what will be in our carbonized mesh file, it is time to discover how it is created.



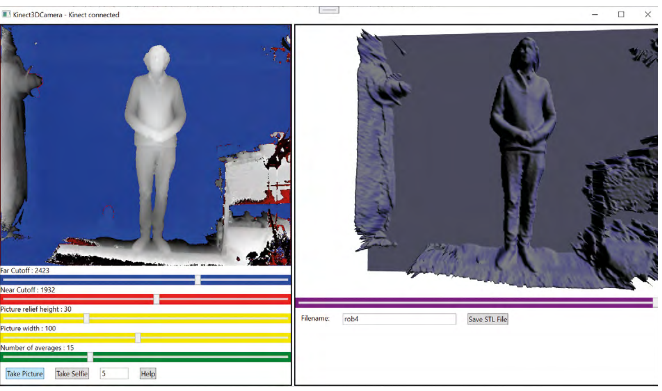
KINECT SENSOR ORIGINS

The original Kinect sensor, made for the Xbox 360, holds the Guinness World Record for the 'Fastest-Selling Consumer Electronics Device' after selling 8 million units in its first 60 days of release (from 4 November 2010 to 3 January 2011). The Kinect One sensor was released at the same time as the Xbox One console and was part of the launch bundle for that console.

Figure 7 shows an Xbox One Kinect sensor mounted on a tripod and ready to carbonize. The author first created this project ten years ago when it seemed like the Xbox One and the Kinect sensor were going to be the next big thing (spoiler alert: they weren't). However, the Kinect sensor is a fine piece of engineering. It contains a depth camera which measures the time taken for infrared pulses to bounce off objects in front of the sensor. The sensor then generates an image which is a 'depth map' of the scene. Each pixel in the depth image is not a measure of the light level at that point (as it would be for a camera) but instead, a distance value (how far into the scene the sensor has detected something). Games that use the Kinect can track the body of the player. This is great for fitness and dance-type games. However, the market for such games turned out to be limited and the sensor software made heavy demands on the console hardware so, after a while, the Kinect sensor was discontinued. You can now pick up a Kinect sensor for a very low price. The sensor was designed to work with the Xbox One, although a PC interface is also available which powers the Kinect sensor and provides a USB 3.0 connection to a PC.

To use the Kinect sensor with your PC, you will need the Kinect SDK. This is a downloadable package of code for Windows which provides the USB drivers for the Kinect, software libraries, and sample programs that show what the sensor can do. Full instructions for installation of the SDK and fault finding are on the GitHub repository for this project: **hsmag.cc/Kinect3D**.

Once we have got our Kinect sensor installed and working, we can make some software that uses data from the camera to make the carbonized objects.


Kinect3DCamera - Kinect connected

Far Cutoff : 2423
Near Cutoff : 1932
Picture relief height : 30
Picture width : 100
Number of averages : 15

Filename: rob4

Figure 8 There is a self-timer you can use for 3D selfies

SLIDE TO TRAP

The Kinect Carbonizer program runs on Windows 10 and 11. It takes a 3D picture and creates an STL mesh from it. **Figure 8** shows it in use. The picture on the left is the image from the depth camera. The closer an object is to the camera, the brighter it appears. The Carbonizer program contains a 'Far Cutoff' slider (coloured blue) which is used to set the maximum distance the image will contain. This allows us to create a flat region behind the person being carbonized which serves as the carbonized slab. Objects beyond the flat region are coloured blue. The trick to a successful 'carbonization' is to bring the blue region forward so that rear parts of the person being carbonized are in the blue region so that they are embedded in the carbonite.

When the user clicks the 'Take Picture' button, the program takes the depth image and uses it to create a 3D mesh which is previewed in the right-hand window. If the depth image looks OK, the 'Save STL File' button can be used to save the mesh for printing.

KINECT CODING

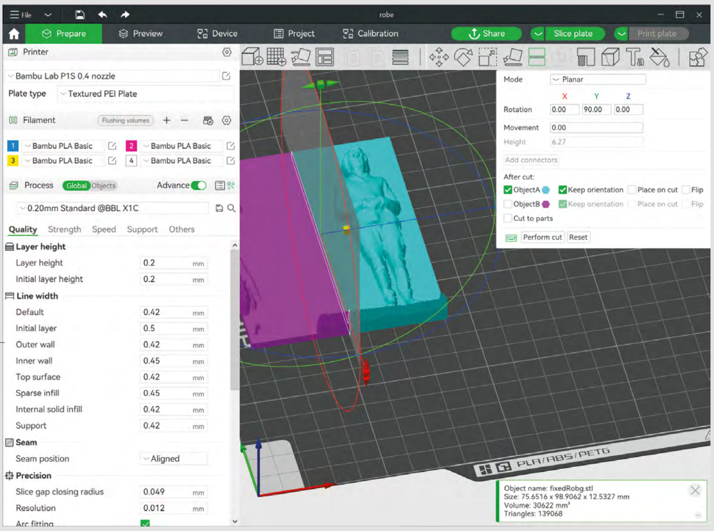
We wrote the Kinect Carbonizer in C#, which just happens to be the author's favourite programming language. If you are familiar with JavaScript or C, the C# program constructions will be quite familiar. The Kinect SDK provides a library of C# objects which represent a connection to the sensor and the data values it returns. Each time the sensor gets a new frame from the depth camera, an event is triggered in the program. The depth sensor data is a list of distance values which are converted into colours for the pixels in the display on the left of **Figure 8**. Any pixels which are beyond the far cut-off are coloured blue. The right-hand display in **Figure 8** is produced from the same mesh of triangles which are used to create the STL file. These are fed into a graphic renderer which is part of the Windows application display system.

The depth values from the sensor contain a lot of 'noise' which causes successive values for the same distance to vary over time. When the distance sensor values are processed, they are averaged and any values which are deemed to be noisy are rejected. This makes the mesh look nice and smooth. There is a slider in the application which can be used to set the number of averages that are to be used. The author was very pleased to discover that the statistics he learnt in school many years ago are actually useful sometimes.

Below You can also use the Meshmixer program to crop and modify meshes

CARBONIZED CROPPING

The Carbonizer process works best if the subject is a reasonable distance away from the sensor. This stops objects on the front of the person from casting shadows on the things behind them. This means that the carbonized images often contain elements you don't need. The best way to get rid of these is to use the 'cut' feature in the slicer program to remove those parts of the mesh.


Printer: Bambu Lab P1S 0.4 nozzle
Plate type: Textured PEI Plate
Filament: 1 Bambu PLA Basic, 2 Bambu PLA Basic, 3 Bambu PLA Basic, 4 Bambu PLA Basic
Process: 0.20mm Standard @BBL X1C

Layer height	0.2	mm
Initial layer height	0.2	mm
Default	0.42	mm
Initial layer	0.5	mm
Outer wall	0.42	mm
Inner wall	0.45	mm
Top surface	0.42	mm
Sparse infill	0.45	mm
Internal solid infill	0.42	mm
Support	0.42	mm
Seam position	Aligned	
Slice gap closing radius	0.049	mm
Resolution	0.012	mm

Mode: Planar
Rotation: X 0.00, Y 90.00, Z 0.00
Movement: 0.00
Height: 6.27
After cut: ObjectA ✓ Keep orientation; ObjectB ✓ Keep orientation

Object name: fixedRobg.stl
Size: 75.6516 x 98.9062 x 12.5327 mm
Volume: 30622 mm³
Triangles: 139068

CARBONIZE YOUR FRIENDS

The carbonizer works well, within the limitations of the sensor and the 3D printing process. It is remarkable how recognisable the images can be, even when printed quite small. ◻

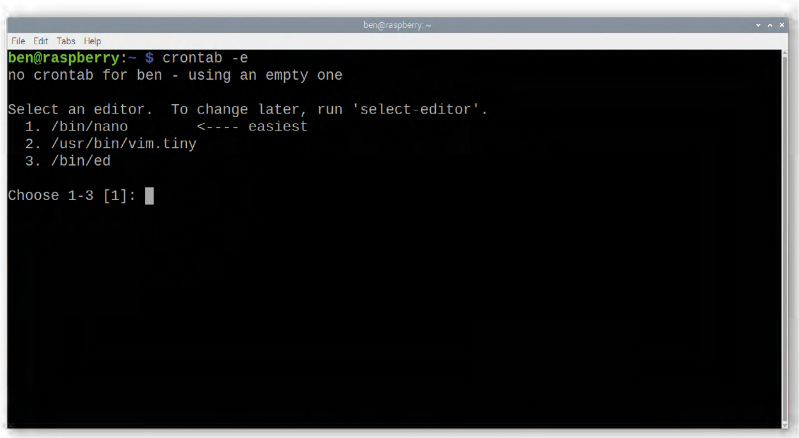
Linux 101: Master time

Schedule tasks to run when you want them to

Ben Everard

Ben's house is slowly being taken over by 3D printers. He plans to solve this by printing an extension, once he gets enough printers.

Raspberry Pi and other small computers typically run the Linux operating system. This is a hugely powerful system, but can seem a little like a hodge-podge of different parts sticky-taped together because, in many ways, it is a hodge-podge of different parts sticky-taped together. The system, which is broadly based on UNIX, harks back to the 1960s. It wasn't so much designed as added to, as different people needed different features. Some of these additions have proved so popular, they have become a standard part of the system. Others fell by the wayside or were replaced. In this article, we're going to look at one such add-on, cron.



```
ben@raspberry:~ $ crontab -e
no crontab for ben - using an empty one

Select an editor.  To change later, run 'select-editor'.
  1. /bin/nano        <---- easiest
  2. /usr/bin/vim.tiny
  3. /bin/ed

Choose 1-3 [1]:
```

The name 'cron' comes from the Greek word 'chronus', meaning time, and it's used to run tasks at a specific time. There are two parts to cron: crond, or the cron daemon, is the software that actually runs the tasks when you want them to run; and crontab is the file that stores the tasks you want to run. The crontab file is specific to each user, including system users such as root, so when you edit this file, make sure that you are currently the user you expect to be.

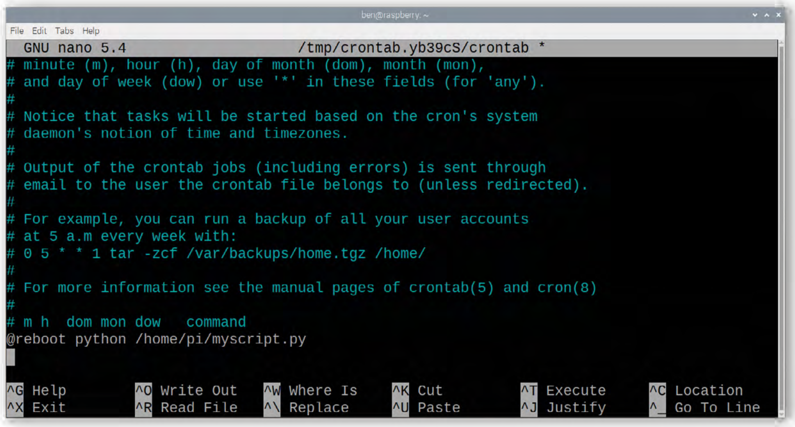
Generally, you don't open the crontab directly in a text editor, but do it using the crontab command.

```
crontab -e
```

The **`-e`** flag tells crontab that you want to edit the file. The alternative is **`-l`** which lists the contents of the file. The first time you run this, you will usually be asked which text editor you want to use. Nano is the easiest of the command-line editors, and the one we'll assume that you're running in this tutorial.

Each line of the crontab gives a task that should be run – unless it's a comment, in which case it will start with a '#'. Each task includes a time and the command to be run. We're going to start with the most unusual format for the time, because it's the most useful for makers: **`@reboot`**.

If your Raspberry Pi is embedded in a project, and you want it to run the appropriate code as soon as it's turned on, this is the way to do it. For example, adding the following line will run the Python script in your home directory each time it is turned on:



```
GNU nano 5.4                    /tmp/crontab.yb39cS/crontab *
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').
#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command
@reboot python /home/pi/myscript.py

^G Help       ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location
^X Exit       ^R Read File  ^\ Replace    ^U Paste      ^J Justify    ^_ Go To Line
```

Below Left
The crontab file is a regular text file, so you can edit it with any text editor

Right
The @reboot directive is especially useful for embedded projects

```
@reboot python /home/pi/myscript.py
```

ON TIME

Reboot timers are great for a lot of uses, but they're not perfect for everything. For example, you might want a backup task to run every night. The easiest way to do this is:

```
@daily /home/pi/my_backup.sh
```

This will run the shell script **`my_backup.sh`** in your home directory every midnight. Note that the **`my_backup.sh`** file must be properly set up as a Shell script with the appropriate shebang at the top; e.g.:

```
#!/bin/bash
```

If midnight isn't the right time for you, then you need to use the full crontab syntax. This is five different bits of data separated by white space. Each bit can either be a number or a wild card (*). The five entries are: minute hour day-of-month month-of-year day-of-week. Crond will check the tasks every minute, and if they match that minute, the task will run. So, for example, the following will run at one minute past every hour of every day:

```
1 * * * * /home/pi/mytask.sh
```

The following will run at 8am every morning:

```
0 8 * * * /home/pi/mytask.sh
```

You can have multiple numbers separated by commas. For example,.,; the following will run on the hour and at half-past every hour:

```
0,30 * * * * /home/pi.mytask.sh
```

You can also give ranges of hours. For example, the following task runs every hour, but only between 9am and 5pm Monday to Friday (Sunday is day 0 to cron):

```
0 9-17 * *1-5 /home/pi.mytask.sh
```

CATCHING PROBLEMS

One potential pitfall of cron is that if you're not there to see your code run, it can be easy to miss any errors that happen. An easy fix to this is to redirect any output to a file.

```
@reboot python /home/pi/myscript.py >>/home/pi/
cronoutput.txt 2>&1
```

There are a couple of bits to this. Firstly, the **>>** redirects any normal output to the file, then **2>&1** redirects any errors to the same place as normal output. The end result of this is that any text output from the command ends up in the **`cronoutput.txt`** file.

You can use cron to manage software you want starting for every powerful tool that's available on almost every Linux system, from tiny Raspberry Pi computers to huge servers. The syntax can seem a little daunting at first, but there isn't too much to it, so once you've got your head around it, it's pretty straightforward. □

SYSTEMD TIMERS

Cron is great, but the format is a little anachronistic, and can make some things harder than they need to be. Not least, the way the tasks are scattered about different crontab files. This author has spent more of his life than he'd care to admit searching through various users' home directories trying to find out what, exactly, is running when on a server.

On most modern Linux systems, you can use systemd's timers to gain more control over what's running. There's a good overview of this setup at **hsmag.cc/timers**.

Big internet-connected button

One whack to send the magazine to press

Ben Everard

Ben likes buttons of all sorts, but his favourite are ones big enough to hit with his whole hand. He's very much looking forward to finishing this issue so he can whack this one.

Each month at HackSpace mag towers, we create a magazine. We source content, edit it, massage it into the right format, add images, and check through everything with a fine-toothed comb. Once this is all done, we send it to the printers to translate our digital files onto paper.

It's this final step that feels a little, well, underwhelming. You might imagine that sending a magazine to press involves hitting a big important-looking button. At least, that's what this writer thought when he first joined the world of publishing many years ago. However, reality is less impressive, and we actually just send a message to someone to say that we're happy with it.

When reality is underwhelming, we can choose to accept that, or we can choose to enliven it. This issue, we choose the latter and we've created our own big, important-looking button. OK, technically we haven't created our own button, we bought it (the aptly named Massive Arcade Button 100 mm from Adafruit). We did, however, make a mount for it, attach it to a Raspberry Pi Pico W, and program it to send the appropriate message when we hit it.

Now, rather than type out our approval message, we just hammer the big red button once the issue's ready to send.

Obviously, it's unlikely that you happen to need a button to send a magazine, but we'll look at how to make a big red button that does – well, anything you like really.

First, you'll need the 3D-printable file for the box. You can grab this from **hsmag.cc/button_box**. It's designed to work with a multi-colour printer, but you can print it in a single colour. It's a pretty simple design – just a cube with the unnecessary bits removed, leaving a hole in the top for a button, and a hole in the back for power. There's no mounting for the Raspberry Pi Pico W inside – we prefer to use a bit of hot glue but, if you're averse to that sort of thing, you could add a few holes.

The text was added in PrusaSlicer, and can be edited in that software as well.

The electronics for the switch are in a removable assembly in the bottom of the button. Twist it a few degrees and it should pop off. There's an LED in the button, so we wire that up to a GPIO pin (at present, we just leave it on, so we could wire it up to 3.3 V

ALTERNATIVE BUTTONS

We used a massive arcade button for this, but there are plenty of other options. At its heart, the button is just a microswitch, and you could set many different things to also trigger the microswitch. Alternatively, in CircuitPython, you can create touch sensors on Pico. Find details at: **hsmag.cc/rp2040_cap_touch**. Using this, you could create a pad out of something conductive that sends a message when you touch it. Perhaps a tinfoil hand that triggers the message when you high-five it?

" **When reality is underwhelming,** we can choose to accept that, or we can choose to enliven it "

or VBUS, but we thought we'd keep the option for adding light effects in the future). The LED has a built-in resistor, so we can just connect it up directly to pin 0 and ground. The LED pins are on either side of the switch (see **Figure 1**). You can use either one as ground and power, and then just turn the LED around to match.

There are three pins for the switch: Common, Normally Open, and Normally Closed. We wired up Normally Open and Common directly between GPIO 1 and ground. The Common pin is on the bottom of the unit, while the Normally Open is the bottom of the two pins on the end. We can then add a pull-up resistor in software, and the button will read 1 when it's not pressed and 0 when it is.

That's the extent of the hardware for this build, let's now take a look at the software.

We've used MicroPython, so you'll need to install that to the Pico W.

We need to send our message via Slack, and this has an application programming interface (API) that lets us interact with it from our script. If you want to work with a different service, you need to find out the suitable details of its API.

DIALLING IN

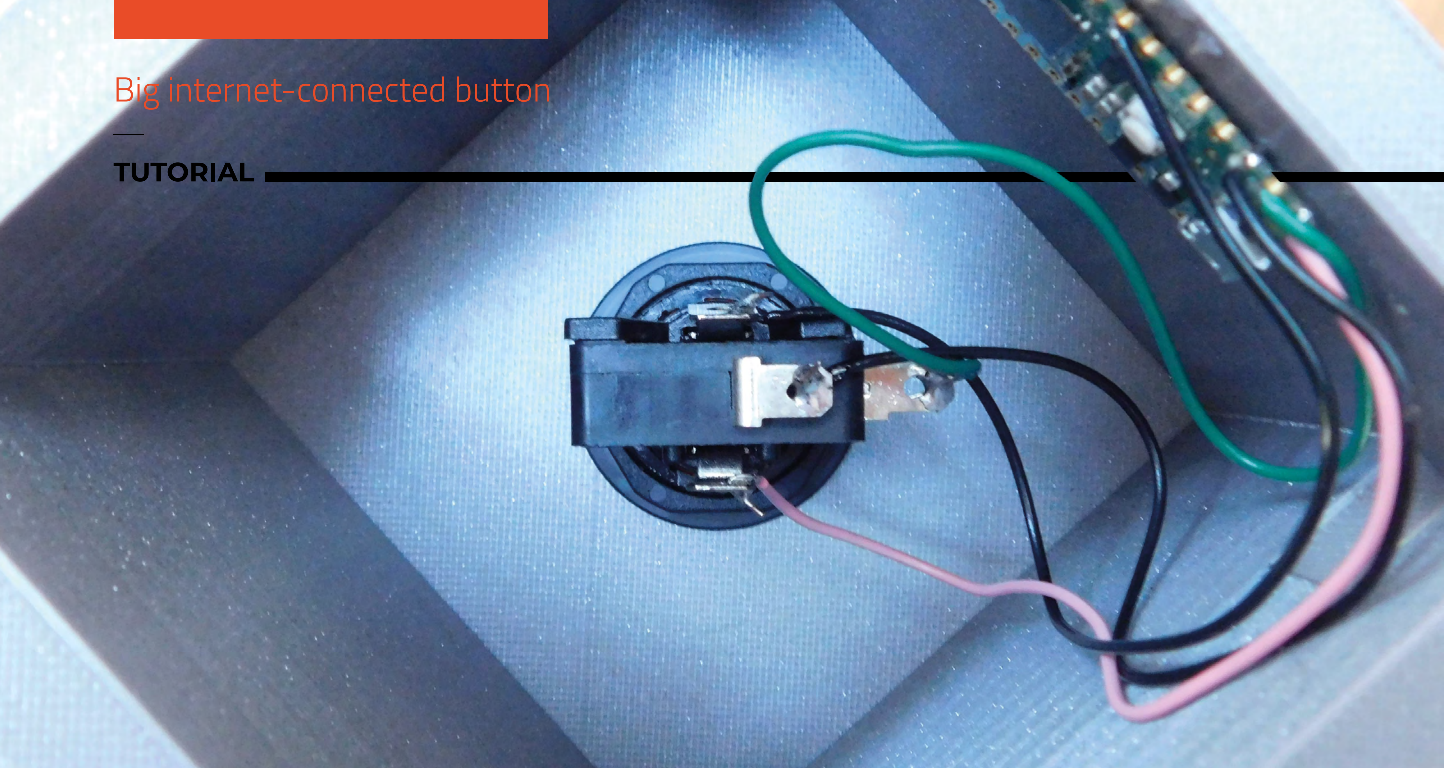
Interacting with the Slack API is a little fiddly because you have to get the security details right. Fortunately, someone far smarter than us has already done the hard work and written it up on the Raspberry Pi blog. There are full details at **hsmag.cc/slackbot**. We won't rehash all of this here. The most important part, for our purposes, is setting up the security details on the Slack website (depending on your configuration, you may need to get your workspace admin to approve the bot).

Once you've set all this up, and noted down the security details, you need to get the code from: **hsmag.cc/git_slack**.

The key files we need are in the MicroPython directory. We need all of these except **main.py**, for which we'll use our own code (below). These need to be copied onto the Pico W, and you can do this using the Thonny MicroPython editor. Go to View > Files, then you can navigate to the place you downloaded them. Select them, and upload them to the **/** directory on Pico. →

Left
The combination of different coloured letters, plus a slight embossing, really helps the letters stand out

Below
There are few things quite as exciting as a massive red button that's just begging to be pressed



Above Figure 1. The button has five connections – we use four of them

TYPE IT OUT

If you want to interact with a service that doesn't have an API you can use, you can get Pico to connect to your computer as a keyboard and type out a message when it's pressed.

At the time of writing, support for this in MicroPython is incomplete, but you can get the code from here: **hsmag.cc/micropyton_hid**.

Alternatively, you can use CircuitPython, which supports Pico and has full support for emulating a keyboard. You can find out more at this link: **hsmag.cc/circuitpython_hid**.

SIMPLE SCRIPT

Let's now look at our own code (below).

The one bit that you have to change here is the `channel_id`. In Slack, you can send a message to a person, a group of people, or a channel, but all three options use a channel ID. To find out the channel ID of a channel, click on the channel, then the little drop-down arrow next to the channel name, and scroll to the bottom of the box that pops up, and you should see the ID.

For a message to one person, go to that person's profile, click on the three dots, and you should get the option to copy the member ID. This can be used in place of a channel ID.

For a group of people, you have to already have a chat with the group going. Click on the conversation, and then there's a drop-down arrow next to the title (which is the names of the people in the chat). Click on this, then go to the About tab and you should see the channel ID at the bottom.

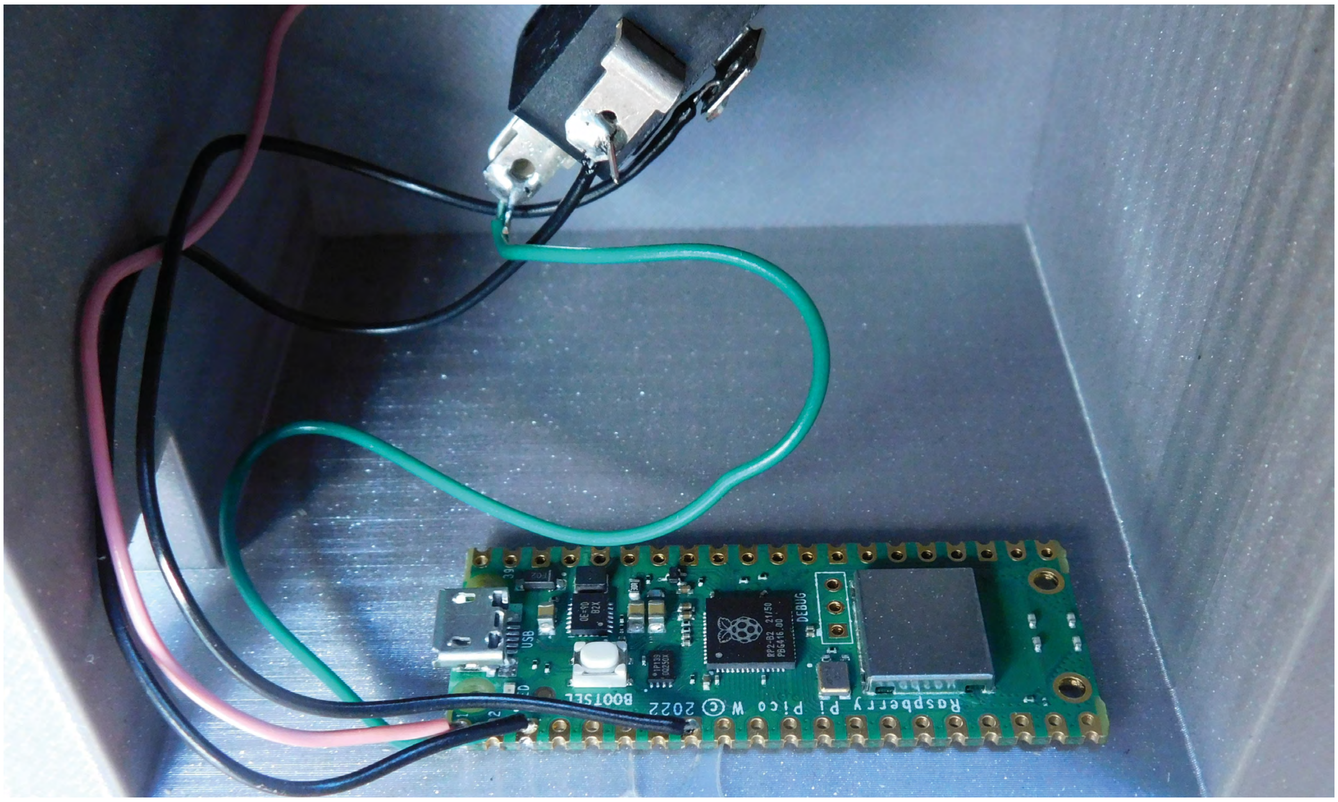
The code is pretty basic. First, it turns on the LED, and connects to the network. Once it's connected, it starts to loop, waiting for the button to be pressed. When it is, the code sends the Slack message.

We need to be a little careful that, with each press of the button, we only send the message once. When a button is pressed, two contacts come together and, as they do, they can connect and disconnect multiple times before settling together. This is known as bouncing.

We have a little loop that runs to make sure that the button has been fully unpressed before it can fire again.

The full code for this is:

```
import network
import time
import config
from slack_bot import SlackBot

channel_id = "put your channel here"
message = "Let's send it"

led = machine.Pin(0, machine.Pin.OUT)
led.on() # let's just have it light up
button = machine.Pin(1, machine.Pin.IN, machine.
Pin.PULL_UP)

# initialize the Wi-Fi interface
wlan = network.WLAN(network.STA_IF)

# activate and connect to the Wi-Fi network:
wlan.active(True)
wlan.connect(config.WIFI_SSID, config.WIFI_PASSWORD)

while not wlan.isconnected():
    time.sleep(0.5)

print(f"Connected to Wi-Fi SSID: {config.WIFI_
SSID}")
print("Now waiting for the button")
while True:
    if not button.value():
```



```
        #The button has been pressed
        slack_bot = SlackBot(config.SLACK_APP_
TOKEN, config.SLACK_BOT_TOKEN)
        print("posting message")
        slack_bot.post_message(message, channel_
id)
        print("posted message")

        #wait for button to be unpressed
        counter = 0
        while True:
            if button.value():
                counter += 1
            if not button.value():
                counter -= 1
            if counter > 10:
                break
            time.sleep(0.01)
```

Load the code onto Pico W (along with all the libraries from the original project).

Make sure that you have the configuration file set up, and you should be able to send messages with the press of a button. Whether it is to send something to manufacture, share a joke, or anything else you might need, whack the button to send the message. ◻

MESSAGE BACK

Our button simply sends messages to Slack, but the API allows messages to be sent the other way as well. Take a look at the original link for more details.

Depending on what you want to do, you could send a message to 'arm' your button before it's used, or light it up. You could add servos or colourful LEDs to display some information when it's ready to go.

Above A spot of glue mounts Pico W

Below Everything's good – let's go to press!

Part 11

Design a circuit with KiCad

Design an electronic circuit for controlling high-power LED lights or model railway lighting

MAKER

Stewart Watkiss

Also known as Penguin Tutor. Maker and YouTuber who loves all things Raspberry Pi and Pico. Author of *Learn Electronics with Raspberry Pi*.

penguintutor.com

twitter.com/stewartwatkiss

This tutorial will provide guidance on how to design your own circuit using KiCad. It will show how you can design a circuit that can be used with Raspberry Pi Pico. This will include choosing suitable components and designing a schematic diagram. This will then lead to creating your own custom printed circuit board (PCB) in the next tutorial.

This circuit is to control three sets of lights using buttons on the PCB or through a web interface. The design can be used for 5 V or 12 V lights, making it suitable for either home automation or model railway lighting.

01 Design idea

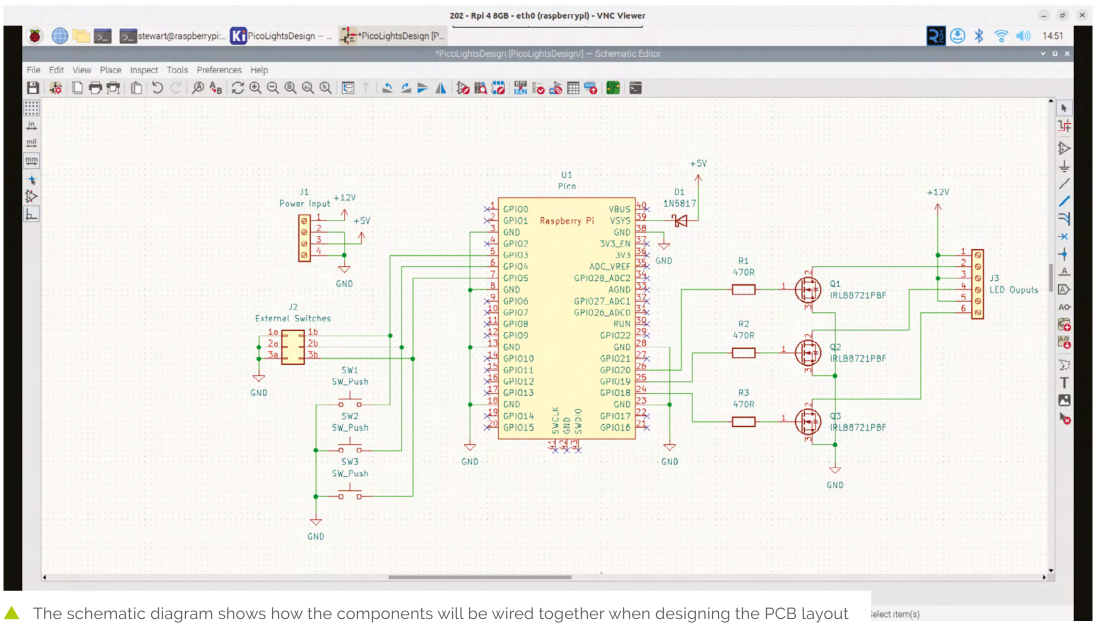
All projects start with an idea. When creating a breadboard circuit, you have an opportunity to experiment and change the design as required. Creating a custom PCB involves additional time and cost, so it is important to spend additional time in the design phase to get the circuit just how you want it.

It is often useful to create a design specification that lists the features that you want, anything you want to avoid, and any restrictions it needs to be designed for. There may be restrictions on size, or you may want to provide additional flexibility to

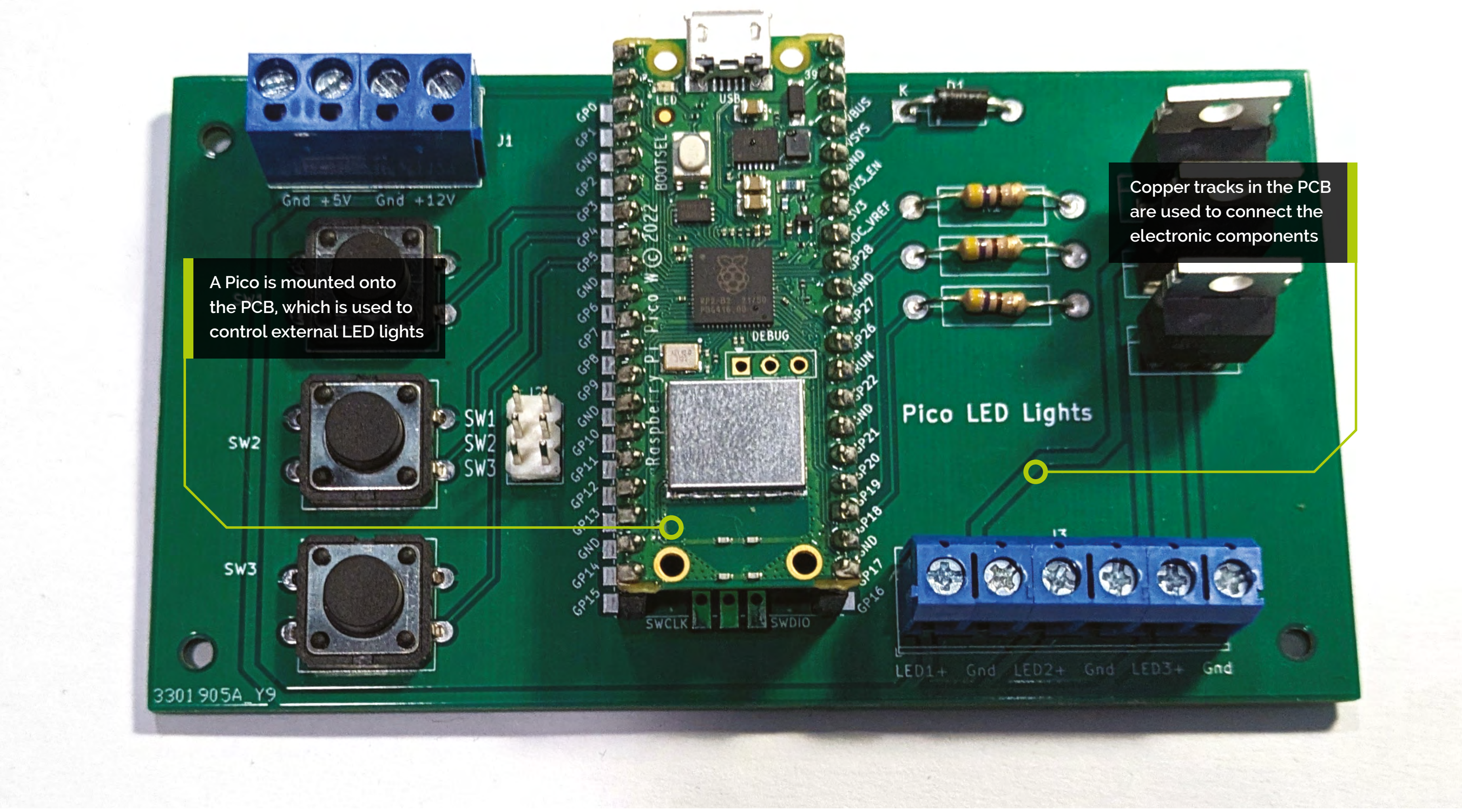
Warning!
Electrical Safety

Whilst 12 V will not cause electrocution, it can cause a fire. Ensure power supplies have over-current protection and consider adding a fuse.

magpi.cc/electricalfires



The schematic diagram shows how the components will be wired together when designing the PCB layout


A Pico is mounted onto the PCB, which is used to control external LED lights
Copper tracks in the PCB are used to connect the electronic components
Pico LED Lights

add extra features. Sometimes these may conflict with each other, in which case you may need to make some compromises. Listing these up-front helps to keep your design on track.

02 Creating the initial design

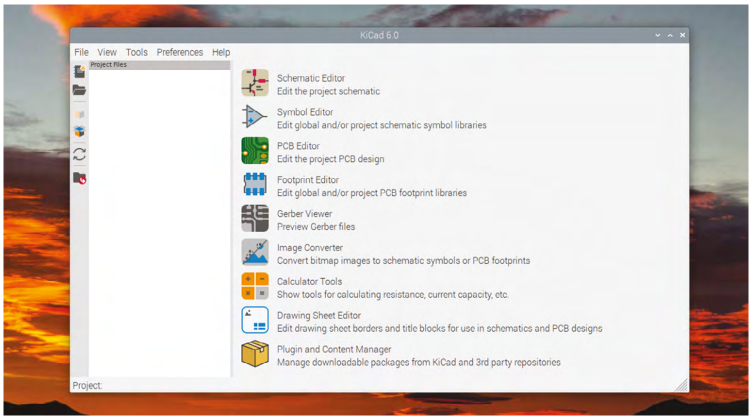
With the idea and specification ready, you can start to make basic decisions about the circuit. Our first decision was to use a Raspberry Pi Pico. This project could be made using a Raspberry Pi computer, but it doesn't need that amount of power for simple switching. Without the overhead of an operating system, Pico is more responsive, more reliable, and cheaper. A Pico W can be used to provide Wi-Fi access.

The plan is to switch high-power LEDs which are more than can be powered just using the GPIO pins on a Pico, so this is going to need MOSFET switch circuits.

It also needs to use switches for input, and these can be wired between GPIO pins and ground, using the internal pull-ups in Raspberry Pi Pico.

03 Flexible design

One thing to consider when designing a PCB is whether it will be used for a single circuit or whether it can be useful for multiple purposes. It can be useful to include additional flexibility as that helps justify the cost of having a PCB made, but adding extra features will increase the size and cost of the PCB.



Schematic Editor
Edit the project schematic

Symbol Editor
Edit global and/or project schematic symbol libraries

PCB Editor
Edit the project PCB design

Footprint Editor
Edit global and/or project PCB footprint libraries

Gerber Viewer
Preview Gerber files

Image Converter
Convert bitmap images to schematic symbols or PCB footprints

Calculator Tools
Show tools for calculating resistance, current capacity, etc.

Drawing Sheet Editor
Edit drawing sheet borders and title blocks for use in schematics and PCB designs

Plugin and Content Manager
Manage downloadable packages from KiCad and 3rd party repositories

Figure 1: KiCad includes different tools which can be used to help in designing printed circuit boards

You'll Need

- 470 Ω resistors **magpi.cc/470ohm**
- 1N5817 diode **magpi.cc/1n5817**
- Switches **magpi.cc/12x6switches**
- PCB screw terminals **magpi.cc/pcbterminal**
- IRLB8721 MOSFET **magpi.cc/mosfet**
- 5V COB LED light **magpi.cc/cobled**
- Power adapter **magpi.cc/jacktoscrew**

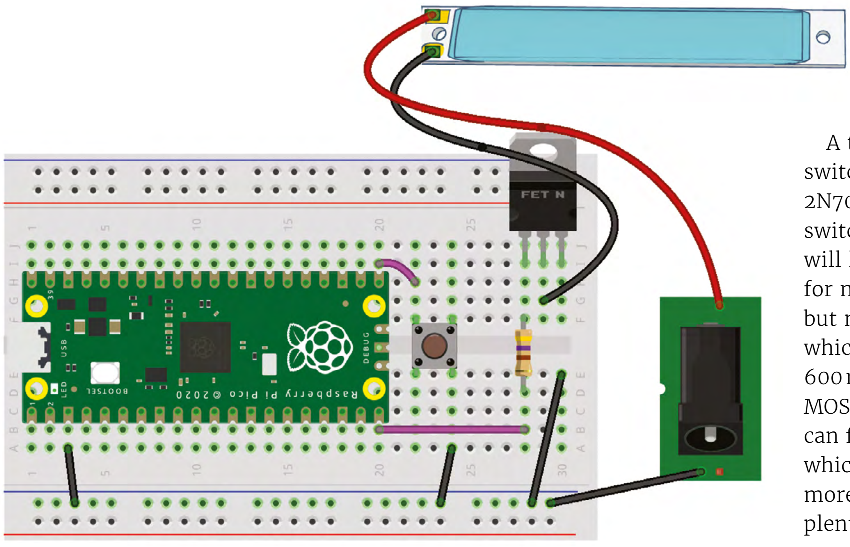


Figure 2: A breadboard prototype with a COB LED light strip and a connector for a 5 V power supply

A typical MOSFET for switching LEDs is the 2N7000. This MOSFET can switch up to 200 mA, which will likely be sufficient for model railway lights, but not for the light strips which can draw up to 600 mA. Looking at what MOSFETs are available, you can find the IRLB8721PBF which supports up to 62 A, more than we need with plenty to spare. It is more expensive than the 2N7000, but the savings in having a single transistor type for different requirements means that we can make savings by using a single PCB design for multiple projects.

The main thing here is what LED lights are to be controlled. In the specification for this build, it was decided that the board should be capable of controlling both 12 V or 5 V LED lights. The 12 V lights would be useful for camping lights or model railways, and 5 V would be useful for COB (chip-on-board) lighting strips. An appropriate voltage power supply is needed to match the LEDs used.

04 Component selection

Having decided on the LEDs, you can now choose a MOSFET that is sufficient for controlling them.

05 Prototyping

The next stage is to test if your design will actually work. This is where the breadboard comes in useful because it allows you to try out different circuits and values. You may want to use a multimeter, or even an oscilloscope to check that the outputs are what you expect.

> "The next stage is to test if your design will actually work"

In this case, the electronics are made up of common circuit configurations, but as it's a different MOSFET than before, you may want to test it to see if it behaves in the way that you expect. The diagram in **Figure 2** shows an example using single switches and a single COB LED light strip to test the main components.

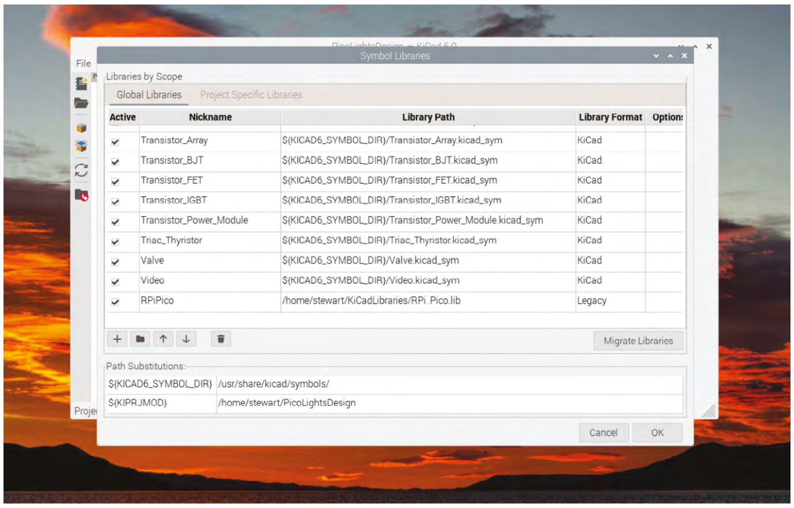
06 Moving to KiCad

In the design so far, you've hopefully been making notes about the decisions you've made. Now it's time to convert those to create a schematic diagram. The schematic diagram is

THE MAGPI

This tutorial is from The MagPi, the official Raspberry Pi magazine. Each issue includes a huge variety of projects, tutorials, tips and tricks to help you get the most out of your Raspberry Pi. Find out more at **magpi.cc**

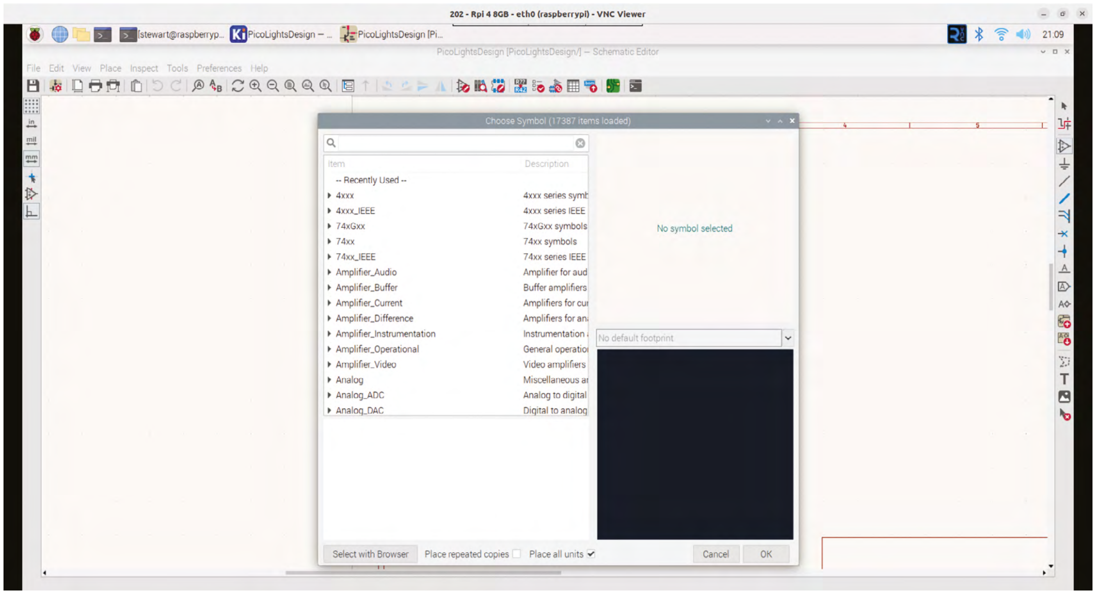
Figure 3: To include a Pico in the design, add the RPi_Pico.lib library through the Manage Symbol Library menu option



Active	Nickname	Library Path	Library Format	Options
✓	Transistor_Array	${KICAD6_SYMBOL_DIR}/Transistor_Array.kicad_sym	KiCad	
✓	Transistor_BJT	${KICAD6_SYMBOL_DIR}/Transistor_BJT.kicad_sym	KiCad	
✓	Transistor_FET	${KICAD6_SYMBOL_DIR}/Transistor_FET.kicad_sym	KiCad	
✓	Transistor_IGBT	${KICAD6_SYMBOL_DIR}/Transistor_IGBT.kicad_sym	KiCad	
✓	Transistor_Power_Module	${KICAD6_SYMBOL_DIR}/Transistor_Power_Module.kicad_sym	KiCad	
✓	Triac_Thyristor	${KICAD6_SYMBOL_DIR}/Triac_Thyristor.kicad_sym	KiCad	
✓	Valve	${KICAD6_SYMBOL_DIR}/Valve.kicad_sym	KiCad	
✓	Video	${KICAD6_SYMBOL_DIR}/Video.kicad_sym	KiCad	
✓	RPiPico	/home/stewart/KiCadLibraries/RPi_Pico.lib	Legacy	

Path Substitutions:

${KICAD6_SYMBOL_DIR}	/usr/share/kicad/symbols/
${KIPRJMOD}	/home/stewart/PicoLightsDesign


202 - RpI 4 8GB - eth0 (raspberrypi) - VNC Viewer

PicoLightsDesign [PicoLightsDesign/] — Schematic Editor

Choose Symbol (17387 items loaded)

Item	Description
-- Recently Used --	
4xxx	4xxx series symb
4xxx_IEEE	4xxx series IEEE
74xGxx	74xGxx symbols
74xx	74xx symbols
74xx_IEEE	74xx series IEEE
Amplifier_Audio	Amplifier for aud
Amplifier_Buffer	Buffer amplifiers
Amplifier_Current	Amplifiers for cu
Amplifier_Difference	Amplifiers for an
Amplifier_Instrumentation	Instrumentation
Amplifier_Operational	General operatio
Amplifier_Video	Video amplifiers
Analog	Miscellaneous a
Analog_ADC	Analog to digital
Analog_DAC	Digital to analog

No symbol selected

No default footprint

Figure 4: There are over 15,000 different symbols available in the KiCad symbol library and many more are available to download from vendors

useful because it shows each of the components wired together without the difficulty of trying to follow individual wires.

The tool used here is KiCad, which is open-source software capable of creating professional circuits. You can run KiCad on most computers, including Raspberry Pi. To install on a Raspberry Pi, use the terminal command:

```
sudo apt install kicad kicad-packages3d
```

07 Creating the schematic diagram

After launching KiCad, you should see the project window. There should be no project selected and the right-hand side of the window shows the various tools which make up KiCad. This is shown in **Figure 1** (previous page). Create a new project by clicking on the new project icon and giving it a name, such as PicoLights. It will create a new directory and create a KiCad project file. You can then click on the Schematic Editor on the right to start a new schematic diagram.

You are presented with a blank drawing area. You can move around using a mouse with the right button pressed, and zoom using your mouse scroll wheel.

08 Adding a Pico footprint to KiCad

Whilst various models of Raspberry Pi are included in KiCad's component library, it does not currently include a Pico. To add a Pico symbol, first download the file from **magpi.cc/kicadzip** (note: direct download).

This file is a complete PCB design, but within the zip file are the files **RPi_Pico.lib** and **RPi_Pico_SMD_TH.kicad_mod**. Copy both files to a suitable directory (eg. KiCadLibraries) and choose Tools > Manage Symbol Libraries.

Click on the '+' icon to create a new library, name it 'RPiPico', then click on the folder icon in the Library Path column and select the lib file. It will add it as a Legacy library. This is shown in **Figure 3**. That .kicad_mod file will be used later when creating the PCB.

09 Adding components to the schematic

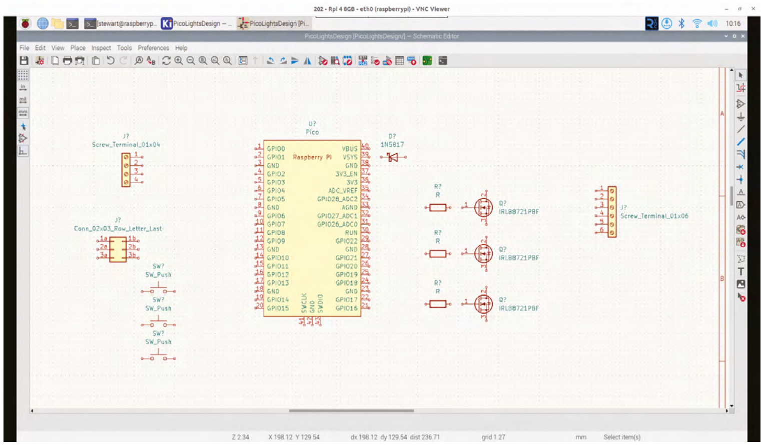
You can now add components by choosing Add Symbol from the Place menu, or by using the icon on the right-hand side of the schematic editor. Click the screen to bring up the symbol selector dialog, search for Pico, and select the one inside

Top Tip

Power options

Your Pico can be powered through a 5V connection to the screw terminal, or by using the USB port on your Pico

Figure 5: All components are added and can be arranged by moving into position using the **M** key



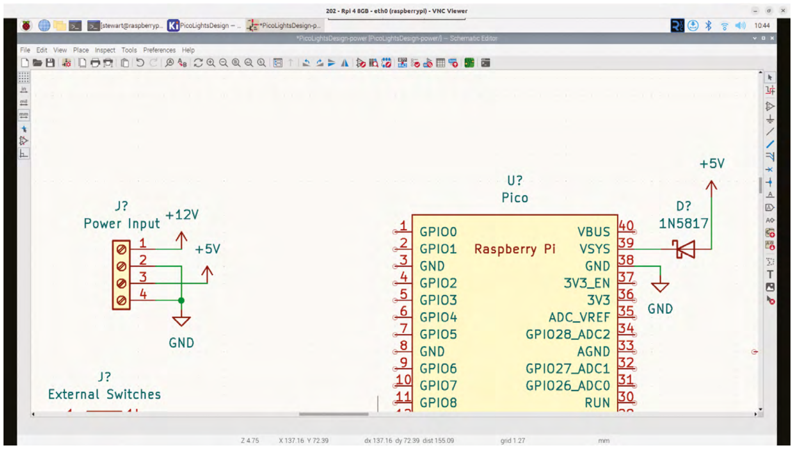


Figure 6: Power label symbols are used to indicate the power supplies. Labels with the same name are treated as though they are connected

the RPiPico library, then click OK to add it to the schematic diagram. The symbol selector is shown in **Figure 4**.

For the button switches, use SW_Push. This will also provide headers which can be used to connect to an external switch that can be added as 'Conn_02x03_Odd_Even'.

The MOSFET is the IRLB8721PBF and add gate resistors (search for 'R' for the resistor symbol).

simple-lights.py

Language: **MicroPython**

DOWNLOAD THE FULL CODE:
magpi.cc/picolights

```python
from machine import Pin
import utime

sw1 = Pin(3, Pin.IN, Pin.PULL_UP)
sw2 = Pin(4, Pin.IN, Pin.PULL_UP)
sw3 = Pin(5, Pin.IN, Pin.PULL_UP)

out1 = Pin(20, Pin.OUT)
out2 = Pin(19, Pin.OUT)
out3 = Pin(18, Pin.OUT)

while(1):
    if (sw1.value() == 0):
        # Toggle out1
        out1.value(1-out1.value())
        utime.sleep (0.5)
    if (sw2.value() == 0):
        out2.value(1-out2.value())
        utime.sleep (0.5)
    if (sw3.value() == 0):
        # Toggle out3
        out3.value(1-out3.value())
        utime.sleep (0.5)
```

10 Powering the circuit

To provide power, a screw terminal is used. This needs two power supplies: one for your LEDs (12 V or 5 V depending upon LEDs) and one for the Pico (5 V). These will be supplied using a 4-way terminal connector, Screw_Terminal_01x04. A 1N5187 diode is also needed to allow either the USB or screw terminal to be used.

The output will also be provided through screw terminals to connect to your choice of LED. Choose 'Screw_Terminal_01x06'.

The components can be moved into an appropriate position and rotated or mirrored using the right-click menu. Arrange the components so that they are similar to **Figure 5**.

11 Adding power connections

The value of the components can be changed using the 'Edit' value from the right-click menu. Change the resistors to 470R and rename the connectors to represent their purpose.

Rather than having to wire up all the different places where power is needed, power symbols can be used. Use +12 V, +5 V, and a common ground for both supplies' GND. If you place multiple power symbols with the same reference (you can use duplicate from the right-click menu) then they will all be connected together, reducing the number of wires needed. Connect the power symbols to the various components using 'Add wire' from the Place menu. The power connections for the input and Pico are shown in **Figure 6**.

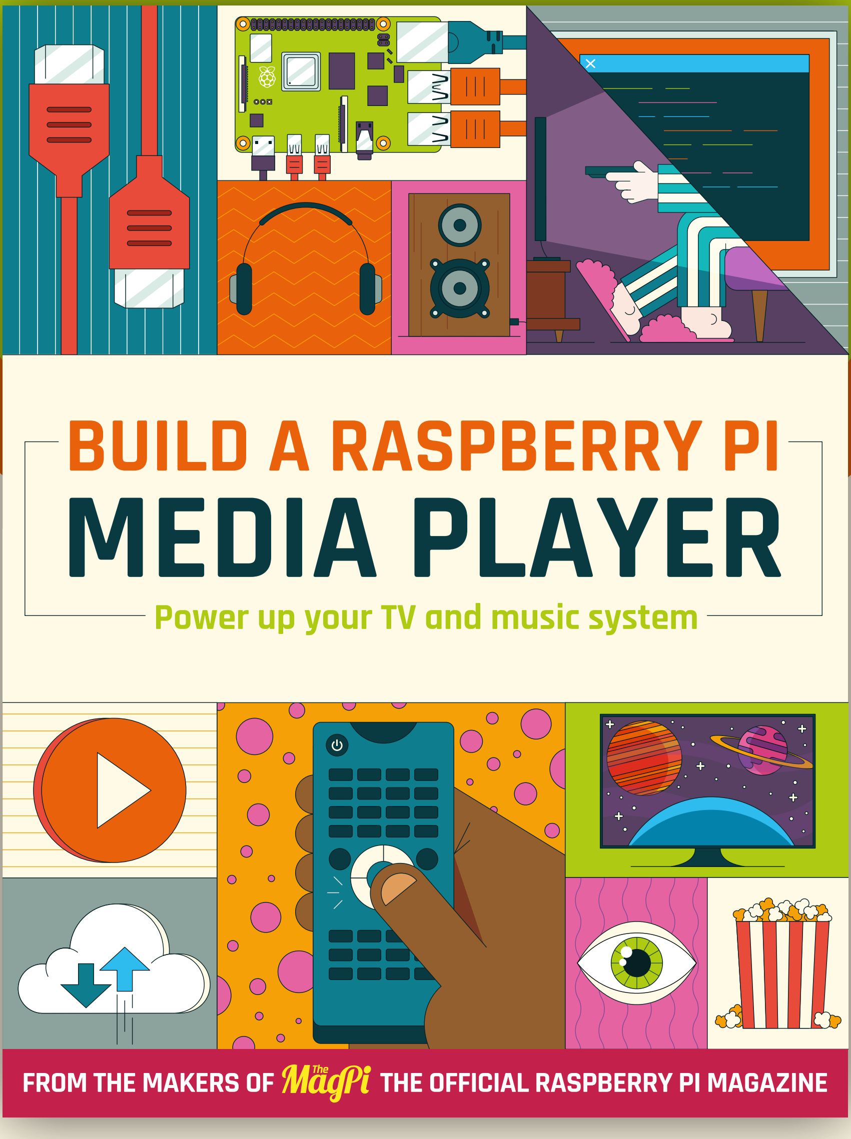
12 Wiring

The rest of the components can be wired together to complete the circuit. When joining wires, there will be a circle over the connection to show that they are connected. If there is no circle, then crossing wires are not electrically connected.

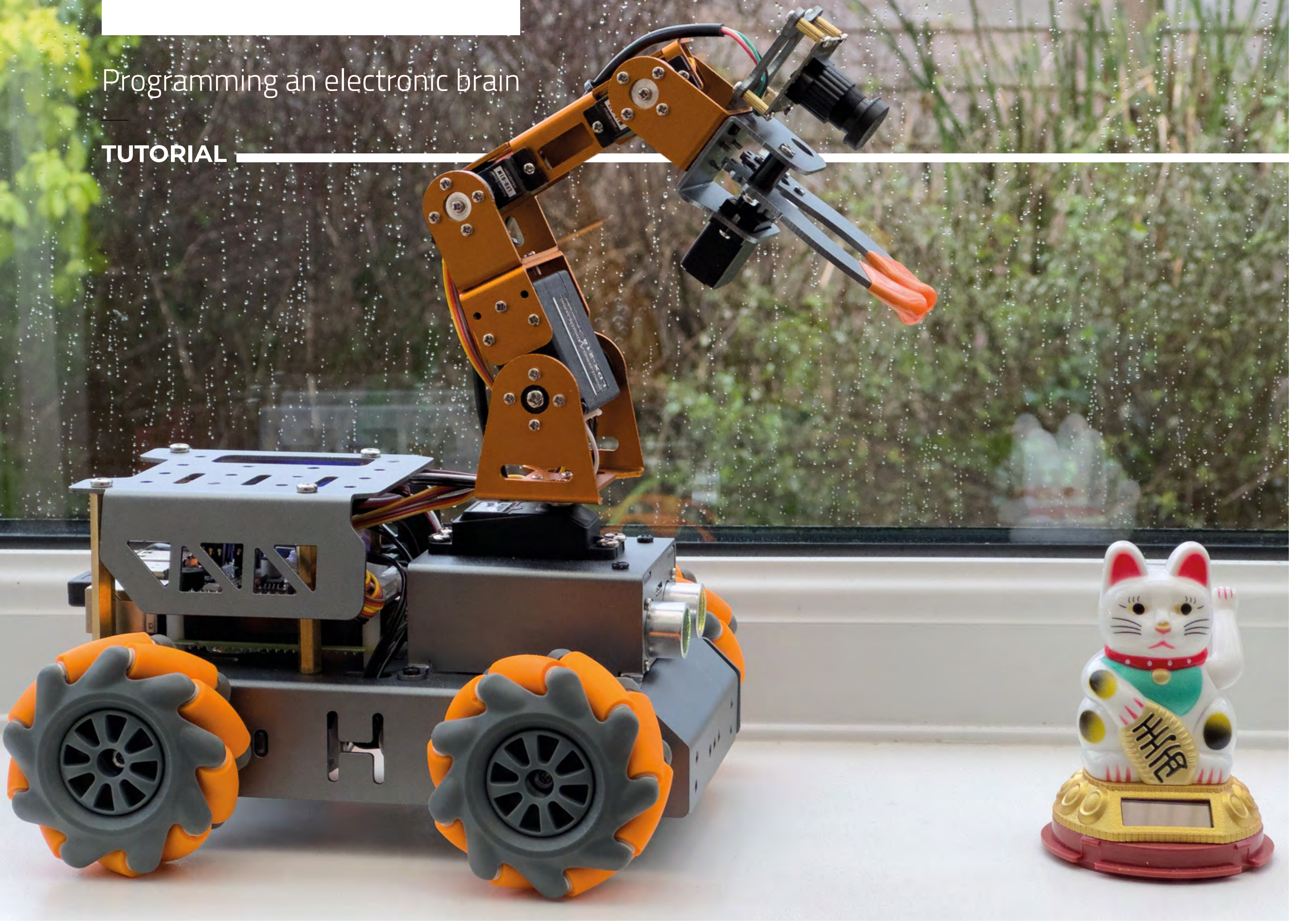
You should also annotate the components by giving each a unique reference. You could do this manually using the properties of each component, changing 'J?' to 'J1' etc., or use 'Annotate Schematic' from the tools menu.

Code is provided (**simple-lights.py**) to allow you to test the prototype using the button LED. In the next tutorial, you will use the schematic to create a professional printed circuit board.

Your FREE guide to making a smart TV


BUILD A RASPBERRY PI
MEDIA PLAYER
Power up your TV and music system
FROM THE MAKERS OF The MagPi THE OFFICIAL RASPBERRY PI MAGAZINE

magpi.cc/mediaplayer



Programming an electronic brain

How the Robot Operating System helps you build complex robots

Figure 1 This robot is called the MasterPi and is made by Hiwonder: hiwonder.com

Figure 2 The USB connection on the right is for the camera on the robot arm

Rob Miles

Rob has been playing with hardware and software since almost before there was hardware and software. You can find out more about his so-called life at **robmiles.com**.

Getting into robot development has never been easier or cheaper. You can pick up a robot kit for not much more than the price of a video game. If you spend a little more money, you can get one with a robot arm on top, like the one shown in **Figure 1** above. Many of the robots use mecanum wheels, which allow the robot to move in any direction.

Figure 2 shows what is controlling the robot. A Raspberry Pi sits underneath a 'HAT' which manages the power supply (two 18650 lithium batteries) and the signals to control the motors and the robot arm. The robot runs a set of Python programs which control the hardware and allow it to perform preprogrammed tasks.

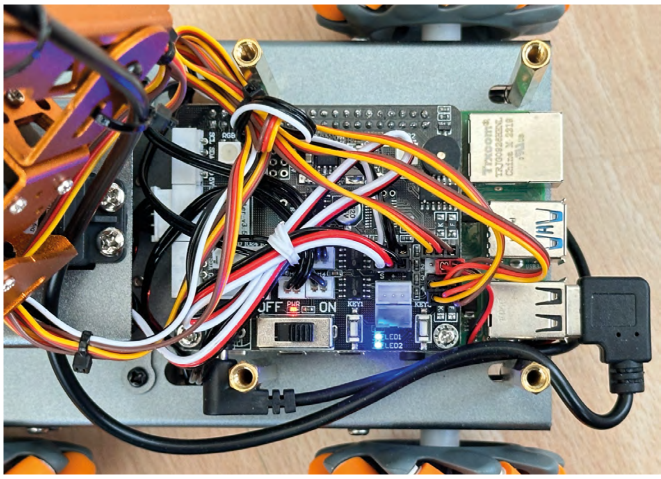


Figure 3 shows the remote-control application you can use to tell the robot what to do. The robot hosts a Wi-Fi access point to which you connect the mobile application. You can then select from pre-built behaviours. This works well, but what if you want to do more? The author was very keen to use his robot as a platform for learning the Robot Operating System (ROS). So that is what he has decided to do. The robot is presently driven by Python programs; the idea is to turn these into ROS nodes. But first, we must learn a bit about ROS itself.

COMING UP ROS-ES

An operating system is something you add to hardware to make it useful. Examples are Windows, MacOS, or Linux. The operating system takes the raw ability of the hardware (running programs, reading keyboards, saving data, displaying images on screens, etc.) and provides a user interface. The operating system lets you select the program you want to run. When you start the program, the operating system fetches the selected program from mass storage and then performs the instructions in the program. Some of the instructions will ask the operating system to do things; for example, a word processor will ask for a document file to be loaded into memory.

ROS takes the abilities of a computer system and makes it useful to a robot creator. ROS lets you break a system down into cooperating components. Components are created inside packages, which makes it easy to manage complex solutions. There are many pre-built components that you can incorporate into your solutions. ROS also provides tools you can use to express the physical design of your robot (or other mechanical system controlled by ROS elements) and simulate behaviour in a virtual (i.e. computer-generated) environment. ROS is a rich and complex system which can take a while to master. It provides effective solutions to robotic problems that we aren't even aware we have. Learning it will hurt your head a bit, but it is worth the effort.

INSTALLATION

ROS sits on top of the computer operating system and is closely coupled to it. This means that the versions of ROS you can use are determined by the operating system on your computer. You can run ROS directly on a Windows PC, but the installation is not for the faint-hearted as it involves compiling the ROS program source. It is much easier to use a Linux-based machine for which you can obtain compiled binary versions of ROS. On a Windows PC, you can use the Windows Subsystem for Linux (WSL) to make a Linux environment into which you can install ROS. There is a step-by-step guide to the installation process in the GitHub repository for this article (**hsmag.cc/RosIntro**). The process uses the amazing Docker tool, which makes it possible to host any version of Linux on your machine, whatever its architecture or operating system.

ROS is a large application that needs plenty of memory to run. It will not fit on smaller devices as it needs at least 4GB of RAM. If you are using a Raspberry Pi, you can add a swap file to your system which uses file space to extend main memory. However, if you do this, you may find that your SD card (the place where your files are stored) wears out as the operating system will continually write to the swap file as programs are started and stopped. If you are feeling brave (and have a few spare SD cards), there are instructions on how to do this in the installation guide.

INSIDE THE WORLD OF ROS

ROS breaks a robot application down into nodes. A node is implemented as a running program. The nodes talk to each other in a well-defined way, and they cooperate to keep the robot running. The nodes can all run as individual tasks on a single computer, or they can run on lots of different processors connected to the local network.

Figure 4, overleaf, shows a ROS installation which contains a controller, a robot, and a camera. Perhaps we want to make a robot litter picker which will wander around searching for litter. The robot is constructed as three devices: a controller, a robot rover, and a camera. Each device is running one or more nodes that form part of the ROS solution.

Nodes provide services to other nodes and accept commands from them. In addition, any node can publish data items that any other node can subscribe to. There are lots of advantages to organising a robot application this way. It is easy to move nodes between devices. The connections between the →

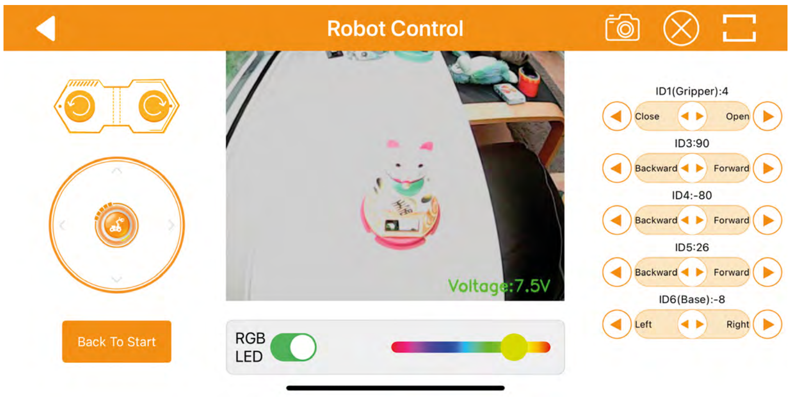


Figure 3 This application is running on an iPhone. A version of the code is also available for Android

YOU'LL NEED

- **A reasonably powerful desktop computer, laptop, or Raspberry Pi (preferably a 4 or 5 with 4GB of RAM)** to run the development environment and ROS
- **Your own little robot or the parts to make one**

QUICK TIP

You can use your knowledge of ROS to power a pre-built robot or to take control of a robot that you have created yourself out of components you have chosen. There are tips on hardware choices in the GitHub repository for this article: **hsmag.cc/RosIntro** nodes are well-defined. We could move all the nodes onto a single powerful computer, or replace the vision system with one that uses a different camera.

We don't have to write the code for all the nodes. ROS also includes a library mechanism that makes it easy to import node code. To properly understand all of ROS, you must understand all the problems that it solves, and there are many of those. Let's start with something simple. How do ROS nodes help us get our 'litter picker' robot going?

QUICK TIP

ROS is not just good for robots. If you've got a complex application that you want to break down into cooperating components, you should look at what ROS offers.

CLEANING UP

To activate the litter-picking robot, the user could press the 'Search' button on the controller device. The buttons on the front panel of the controller are managed by the **buttons** node, which publishes the button states on a topic called 'buttons'.

The **manager** node has subscribed to this topic, so it receives a message informing it that the Search button has been pressed. The **manager** then sends a service request to the **vision** node asking, "Can you see anything?"

The vision system has subscribed to frames of image data published by the **camera** node and is looking for litter in each frame it receives. If the **manager** gets a response indicating that some litter has been spotted and giving a direction to that litter, the **manager** sends commands to the **motor** node in the robot to head that way. The **manager** also publishes the message 'moving' to a robot status topic.

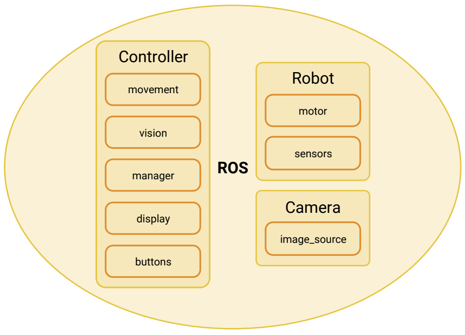


Figure 4
There are eight nodes in this system

The **display** node has subscribed to this topic, so the status 'moving' is displayed. As the robot moves, the **sensors** node on the robot would be publishing information about things the robot is detecting around it. ROS provides the environment in which all this would be made to work. We design the structure of the published data, write the code for each node, and then decide how the nodes will interact, but ROS sits underneath and makes everything work.

PUSH THE BUTTON

We can discover how ROS does this by considering the very start of this process: the front panel of the robot. At the very least, the robot will have a button to make it start and a button to make it stop. It should also have a way of indicating what it is doing, perhaps a coloured light or a text display. The diagram in **Figure 4** shows that the Controller is running a **buttons** node which receives user commands and a **display** node that displays the robot status.

Let's look at the code in this node, which is implemented by a Python class called **ButtonPublisher**. This class extends the **Node** class which is part of ROS. An instance of **ButtonPublisher** class is created when the robot system starts up. Below, you can see the constructor method which runs when a **ButtonPublisher** is created.

```
class ButtonPublisher(Node):

    def __init__(self):
        super().__init__('button_publisher')
        self.buttonReader = ButtonReader()
        self.start_publisher()
        self.start_timer()
```

The constructor first calls the **__init__** method for the **Node** parent class, supplying the call with the name of the publisher. This tells ROS that there is a new node called **button_publisher** in town. The next statement in **__init__** creates a **ButtonReader** object which will be used to read the buttons on the physical console.

The **ButtonReader** reads the buttons from the controller hardware, perhaps by using GPIO pins (although it could also connect to a computer keyboard or touchscreen). Next, the constructor calls **start_publisher** to create a publisher to publish button events to anyone who is subscribed to them. Then it calls the **start_timer** method to start the button scanning timer. There is not much code in the **start_publisher** method:

```
def start_publisher(self):
    self.publisher_ = self.create_
publisher(String, 'buttons', 10)
```

The publisher to be used by **ButtonPublisher** is stored in a class member called **publisher**. The **create_publisher** method is inherited by

ButtonPublisher from its **Node** parent class. The **publisher** method accepts three parameters. The first indicates that the publisher will publish a String, the second gives the topic for the published message (in this case, 'buttons'), and the third parameter (the value 10) means 'keep the last ten published items and use the "RELIABLE" level of quality of service' – which means that published items are guaranteed to arrive at the receiver. The second method called by the **ButtonPublisher** constructor is **start_timer**:

```
def start_timer(self):
    timer_period = 0.1  # seconds
    self.timer = self.create_timer(timer_period,
self.timer_callback)
```

This method creates a timer which fires ten times a second. Each time the timer fires, the **timer_callback** method is called.

```
def timer_callback(self):

    button = self.buttonReader.scan_buttons()
    if button!="":
        msg = String()
        msg.data = button
        self.publisher_.publish(msg)
        self.get_logger().info('Publishing: "%s"'
% msg.data)
```

The **timer_callback** method calls the method **scan_buttons** on the button reader. This scans the buttons and returns the name of the button which is pressed, or an empty string if no buttons are pressed. If a non-empty string is returned, the callback publishes the name of the button that is pressed on the 'buttons' topic.

The final part of the button node program is the mechanism which starts the node itself. This is performed by the **main** method in the node code which is called when the node is loaded.

```
def main(args=None):
    rclpy.init(args=args)

    button_publisher = ButtonPublisher()

    rclpy.spin(button_publisher)

    # Destroy the node explicitly
    button_publisher.destroy_node()
    rclpy.shutdown()
```

The **main** method creates a **ButtonPublisher** instance and then calls the **button_publisher** method on this. The **rclpy.spin** function keeps the node alive; responding to events and managing callbacks. It ends if the node is terminated by ROS, at which point the publisher node is destroyed and the node shuts down.

All the nodes in the robot are started in this way. The robot controller code can contain a launch method which starts all the nodes when the robot begins running.

SUBSCRIPTION MODEL

We know how a node running in a robot can post a message on a topic. Next, we need to consider how another node can receive it.

```
class DisplaySubscriber(Node):

    def __init__(self):
        super().__init__('display_subscriber')
        self.subscription = self.create_
subscription(
            String,
            'buttons',
            self.listener_callback,
            10)
        self.subscription  # prevent unused
variable warning

    def listener_callback(self, msg):
        self.get_logger().info('Button: "%s" was
pressed' % msg.data)
```

The code above defines a **DisplaySubscriber** class which subscribes to the **buttons** topic and calls the **listener_callback** function each time a message is received. This listener callback simply logs the button name (although it could put it on a text display). Code in the **manager** node could also subscribe to the **buttons** topic so that the manager is informed when a button is pressed.

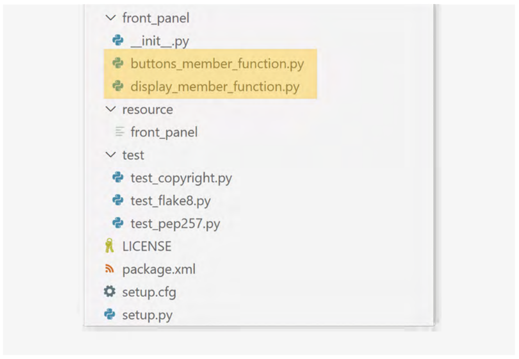
PACKAGE HOLIDAY

Code for ROS applications is organised into packages. A package brings together a set of related behaviours. We could create a package called **front_panel** which contains the code for the **buttons** and **display** nodes.

Figure 5, overleaf, shows the package files for the **front_panel** package. The two highlighted files contain the code for the **buttons** and **display** Python nodes that we have seen above. The three files at the bottom of the package are what ties the package together. The **package.xml** file contains a description of the package and identifies any dependencies that the package has. →

QUICK TIP

The most recent version of ROS is called ROS2. This is the one you should be using.



```
front_panel
  __init__.py
  buttons_member_function.py
  display_member_function.py
resource
  front_panel
test
  test_copyright.py
  test_flake8.py
  test_pep257.py
LICENSE
package.xml
setup.cfg
setup.py
```

Figure 5 ROS can generate a template package for you to fill in

One package can use elements from another, and this is where you would identify the source packages. The **setup.cfg** file specifies where the files are to be placed, and the **setup.py** is a chunk of Python which is run to set up the project. This file is worth looking at:

```
from setuptools import find_packages, setup

package_name = 'front_panel'

setup(
    name=package_name,
    version='0.0.0',
    packages=find_packages(exclude=['test']),
    data_files=[
        ('share/ament_index/resource_index/
packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.
xml']),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='rob',
    maintainer_email='rob@hullpixelbot.com',
    description='Provides a front panel containing
buttons and a text display',
    license='Apache-2.0',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            'buttons = front_panel.buttons_member_
function:main',
            'display = front_panel.display_member_
function:main'
        ],
    },
)
```

Above is the **setup.py** file for the `front_panel` package. The most important part of this file is the entry point list at the bottom. It specifies the Python files that are to be run when the nodes are activated. If we added a third node to the panel (perhaps a buzzer controller), we would create the Python code for the node and then add the entry point here. The above package serves as a template for any future packages you want to create. Just add the Python code into the same folder as the existing nodes and configure **setup.py** with the new entry points. Your code can then be run as a node in the robot system.

BUILDING THE PACKAGE

Now that we have our package, the next thing we need to do is build it. This is the process of collecting all the package components and getting them into a state where ROS can run them. Python programs are not compiled, but ROS nodes can also be created from C++ code, which does need to be compiled before it can run. A package that contains C++ code has a slightly different format and contains a **CMakelists.txt** file which describes how to build the code. It makes creating a package a bit trickier, but it does mean that you can use both languages in your solution. Furthermore, you can import packages containing nodes programmed in C++ and they will work alongside your Python nodes. The ROS system provides a command called `colcon` (collect components) that performs the build process:

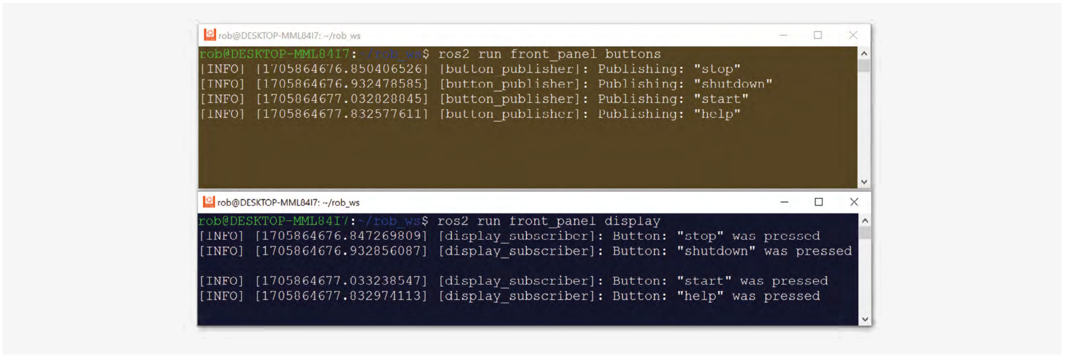
```
colcon build --packages-select front_panel
```

The command above would be performed to build the `front_panel` and make the files ready to run. So, at last, we can run our robot nodes. There's just one more thing we need to know about (sorry), and that is all to do with sourcing our commands. We talk to the operating system from within a 'shell' environment which accepts our commands and then performs them. On Linux, this is usually the 'bash shell'. Some of the commands are 'built-in' to the shell; for other commands, the shell will go off and look for a program to run. We can tell the shell where to look for our robot node code:

```
source install/setup.bash
```

The `source` command means 'find this file and execute it as if it has been typed in'. The command file is called `setup.bash`, and it is created for us by `colcon` when the package is built. Once we have performed this command, we can use the `ros2 run` command to start our two nodes running:

Figure 6 shows two bash shells running the `buttons` and `display` nodes. The two nodes were



```
rob@DESKTOP-MML84I7: ~/rob_ws
rob@DESKTOP-MML84I7:~/rob_ws$ ros2 run front_panel buttons
[INFO] [1705864676.850406526] [button_publisher]: Publishing: "stop"
[INFO] [1705864676.932478585] [button_publisher]: Publishing: "shutdown"
[INFO] [1705864677.032828845] [button_publisher]: Publishing: "start"
[INFO] [1705864677.832577611] [button_publisher]: Publishing: "help"
```

```
rob@DESKTOP-MML84I7: ~/rob_ws
rob@DESKTOP-MML84I7:~/rob_ws$ ros2 run front_panel display
[INFO] [1705864676.847269809] [display_subscriber]: Button: "stop" was pressed
[INFO] [1705864676.932856087] [display_subscriber]: Button: "shutdown" was pressed

[INFO] [1705864677.033238547] [display_subscriber]: Button: "start" was pressed
[INFO] [1705864677.832974113] [display_subscriber]: Button: "help" was pressed
```

Figure 6
These packages are running on a PC under the Windows Subsystem for Linux, but they could just as easily be running on a Raspberry Pi

started using the commands at the top of each shell. The **`buttons`** node has published four button messages and the **`display`** node has displayed them.

ROS COMPLICATIONS

At this point, you might be thinking one of two things: 'Blimey, this is complicated' or alternatively, 'Blimey, this is powerful'. The complexity is there for a reason. We want to be able to break our robot controller down into smaller reusable components which can be easily swapped without affecting the rest of the application.

One robot might use a physical button to start it, but the next might have a touchscreen. Because this input is managed by a node, we just need to swap out the package node for another. Later, we might decide to combine the camera and the controller into a single device. In that situation, we just have to change the **`image_source`** node to run on the controller, and everything which uses images will just work.

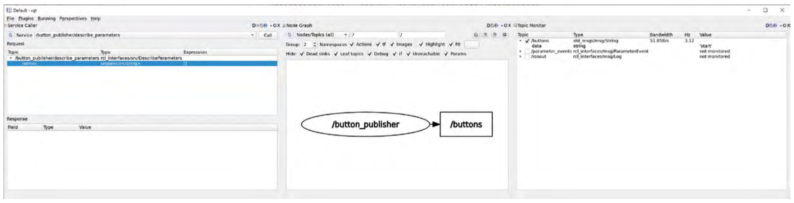
The package folder contains everything needed to build and run a particular set of robot nodes. There are many pre-built packages for robot interfaces and behaviours that you can slot into your robot and interact with via their services and topics.

NODE MANAGEMENT

One of the many wonderful things about ROS is the way that things 'just work'. If you want two nodes to communicate, you just run them and they can magically see each other.

ROS works over your local area network (LAN). Network communications are based on the DDS (Data Distribution Service) standard, which provides the discovery protocol by which nodes can identify themselves and find other nodes. ROS provides commands we can use to discover nodes, topics, and services.

Figure 7 shows the output from three ROS commands which can be used to view an active ROS installation. The **`node list`** command lists all the active nodes – in this case, the **`button`** and the **`display`**. The **`topic list`** command shows all the active topics. The **`buttons`** topic is active, along with two which are provided by ROS. The final command shows all the services that are provided. Note that the button and the display components expose a set of services which can be used to work with them. This makes it possible for a ROS application to automatically discover what nodes can do and how to use them, which makes possible self-configuring systems. We can see this in action if we start up the rqt tool, which is supplied as part of ROS. This contains a range of plug-ins you can use to create interfaces with your ROS application.

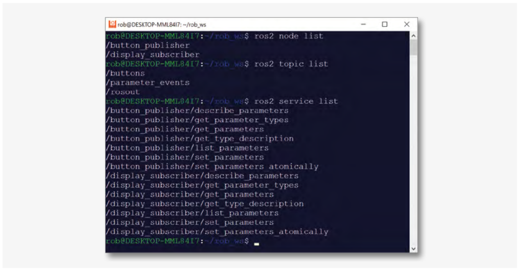


Default - rqt
File Plugins Running Perspectives Help
Service Caller
Service /button_publisher/describe_parameters
Call
Request
Topic Type Expression
/button_publisher/describe_parameters rcl_interfaces/srv/DescribeParameters
names sequence<string> []
Response
Field Type Value
Node Graph
Nodes/Topics (all)
Group: 2 Namespaces Actions tf Images Highlight Fit
Hide: Dead sinks Leaf topics Debug tf Unreachable Params
/button_publisher
/buttons
Topic Monitor
Topic Type Bandwidth Hz Value
/buttons std_msgs/msg/String 51.85B/s 3.12
data string 'start'
/parameter_events rcl_interfaces/msg/ParameterEvent not monitored
/rosout rcl_interfaces/msg/Log not monitored

Figure 8
You can add plug-ins and lay out the display as you like

The rqt window in **Figure 8** shows a view of the topics available on the left, a diagram showing the active robot topics in the middle, and a log of button topic messages on the right. We can see that the last message that was sent was the 'start' message. These views are all updated in real time.

Now we know how ROS applications are structured and how the components interact. In the next article, we'll build on this. □



```
rob@DESKTOP-MML84I7: ~/rob_ws
rob@DESKTOP-MML84I7:~/rob_ws$ ros2 node list
/button_publisher
/display_subscriber
rob@DESKTOP-MML84I7:~/rob_ws$ ros2 topic list
/buttons
/parameter_events
/rosout
rob@DESKTOP-MML84I7:~/rob_ws$ ros2 service list
/button_publisher/describe_parameters
/button_publisher/get_parameter_types
/button_publisher/get_parameters
/button_publisher/get_type_description
/button_publisher/list_parameters
/button_publisher/set_parameters
/button_publisher/set_parameters_atomically
/display_subscriber/describe_parameters
/display_subscriber/get_parameter_types
/display_subscriber/get_parameters
/display_subscriber/get_type_description
/display_subscriber/list_parameters
/display_subscriber/set_parameters
/display_subscriber/set_parameters_atomically
rob@DESKTOP-MML84I7:~/rob_ws$
```

Figure 7
If other ROS nodes were active, they would appear in this output, too

3D printer testing and calibration

Make sure your replicator is running smoothly and reliably

Ben Everard

Ben's house is slowly being taken over by 3D printers. He plans to solve this by printing an extension, once he gets enough printers.

3D printers are fantastic. Get a model, slice it, and out pops a perfect duplicate. However, are you sure it's a perfect duplicate? It looks OK, but what if the dimensions matter? Is 1 cm in your CAD software creating 1 cm in the real world? Or is it really 0.9 cm? Sometimes it doesn't matter, but sometimes it really does.

What about supports? Depending on your models, you could be throwing away a large proportion of your filament, but do you really need to? Could you change some settings to mean you need to waste less?

In this tutorial, we're going to run through a few useful calibration tests to check your printer is performing at its best. It's worth running some, or all, of these when you set up a new printer, perform some maintenance, or just periodically to make sure that everything is still running smoothly.

AHOY THERE!

The famous Benchy test print is popular because it's quick and cute. However, it's actually a good test for a 3D printer – if you can print a Benchy, you can print most things. While it might seem almost mundane now, readers who have been 3D printing for a long time will remember when getting a good-looking Benchy from a printer was an achievement. Nowadays, being able to print a Benchy is pretty much a given, and the question is how fast you can print it. However, that doesn't mean that there's nothing to be learned from printing this humble test piece.

Benchy is designed to be just 3D-printable, and it's a great first print if you're looking to test out a new printer or filament. Take a look at opposite for details of what to look out for.

CUBISM

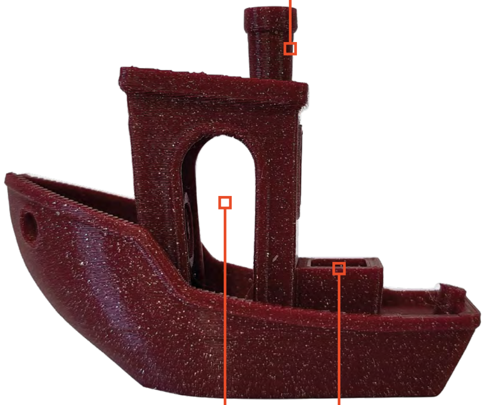
A calibration cube is simply a cube that has the same length on each side – usually 20 mm. You could just create a cube in your slicer, but it's a bit easier if the sides are labelled. Otherwise, once you remove it from the print bed, it's hard to tell what orientation it was printed in. We used **hsmag.cc/cal_cube**, but there are plenty available online.

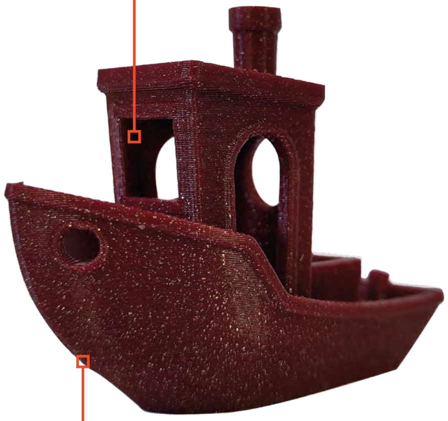
Once you've printed it, pop it off the bed and measure it. Hopefully it's the size you expect. If it's not, then there's a problem with your printer. It could be the belt tension, or a problem with the frame – check with the printer manufacturer's documentation for help.

Do take the readings from digital callipers with a pinch of salt, though. A £10 set of callipers may give you two decimal points worth of output, but they are not accurate down to 0.01 mm. It's with good reason that some machinists call them 'guessing sticks'. We'd consider any measurement within 0.1 mm to be good – beyond this, it's hard to know whether any discrepancy is in the measuring device or the 3D printer. →

Cooling chimney
The chimney has a small cross-section, which means that it's piling the next layer on top of the previous layer very quickly. If your cooling isn't performing well enough, this can turn into one massive blob

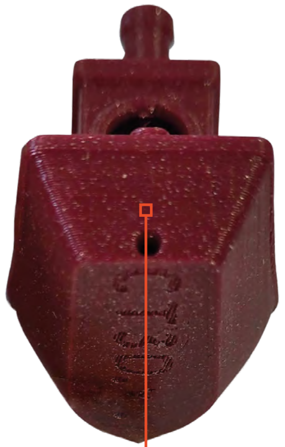
Bridging
You can print unsupported spans, provided they have contact at each end. There will always be some sagging, and it is up to you if you are willing to accept that





Stringing
In the cabin, there are four separate uprights. You might get stringing between them – this can either be caused by your printer's retraction settings or damp filament

Bow overhang
This 40-degree overhang matches the default maximum overhang in PrusaSlicer. If this prints, then so will any unsupported overhang with the default settings



Stern box
This acts as a dimension test as it should be 8 mm side to side and 7 mm front to back. The total Benchy height should be 48 mm. Though, we would recommend a calibration cube to test dimensional accuracy

Nameplate
This text is embossed at 0.1 mm and is probably the one feature that many printers fail to replicate. This test can often highlight the difference between quick and accurate slicer profiles. It's particularly prone to disappearing (as it has in this print) when input shaping is enabled

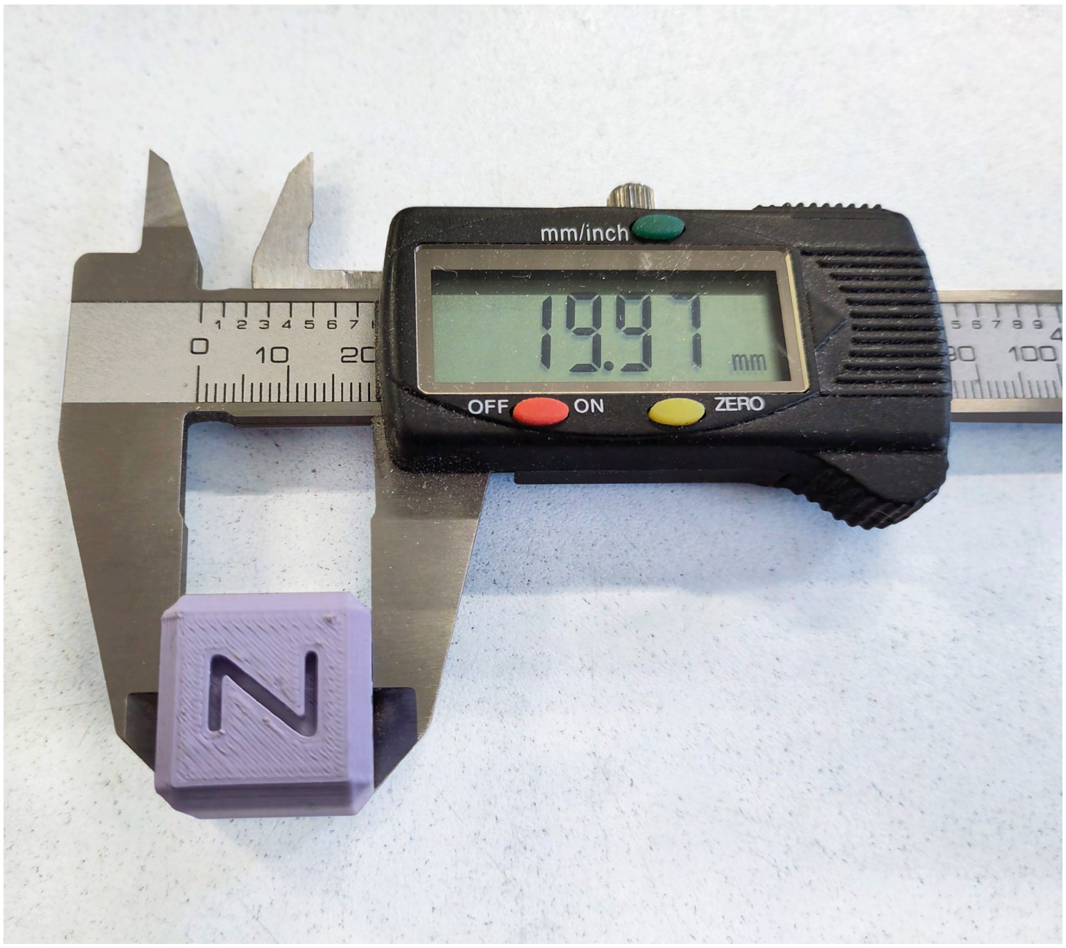


Figure 1
The callipers show 19.97 mm, but that's claiming a degree of accuracy that they don't have. Between 19.9 mm and 20.1 mm is good enough for us

CLINGING ON

One of the key tests of a printer is the overhang angle it can print. The steeper the angle, the fewer supports you need. However, it's not quite as simple as that because there is often not a single point where the overhang fails. It gets progressively worse. The maximum unsupported overhang angle you have really depends on the quality of print you're willing to accept. If you want perfect prints every time, then you need quite a shallow angle. If saving filament and print time is more important to you, then you might be able to get away with a steeper angle. But what is a shallow angle and what is a steep angle? It's time to do some tests!

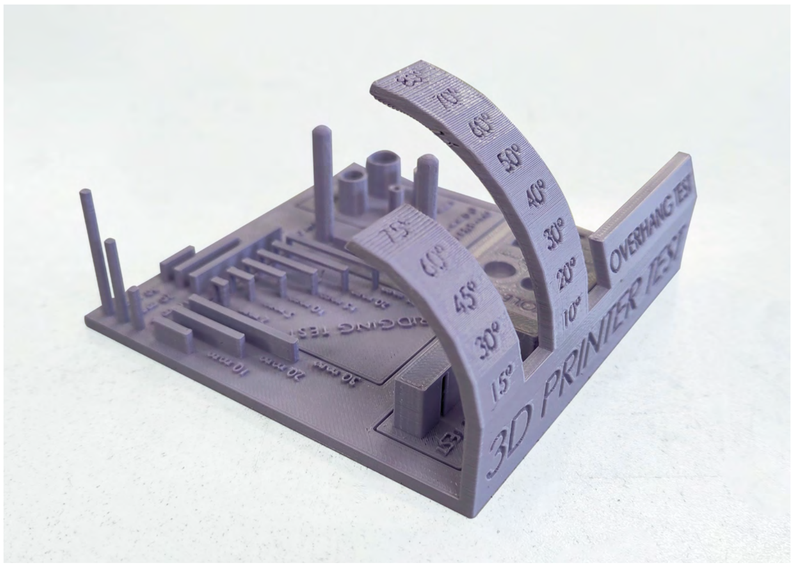
We printed the test model from here: **hsmag.cc/printtest**. This model claims to be a complete test of a 3D printer, and while it does a good test of a lot of features, it's the overhang that we're really interested in.

You can see our results in **Figures 2** and **3**. From this, we know that we can get more or less perfect results up to 50 degrees, and printable overhangs at 85 degrees. The layer height can have an effect on overhangs, so it's worth testing this out at the layer height you want to print at.

Based on your testing, you may want to change the maximum overhang angle before supports are added. In PrusaSlicer, this is called the Overhang Threshold. Depending on your model, and the quality you're after, you may be able to save a lot of print time and filament by increasing it. There is more than one aspect to overhangs – the profile of them can also have an impact on their performance, so you might not want to go all the way to 85 degrees unless you're willing to risk some failures.

GETTING HOT

What temperature should you print your filament at? In many cases, the answer is to just select the filament type in your slicer and let it figure it out. That's a good choice in many cases, but it hides the fact that there isn't a perfect temperature, instead,



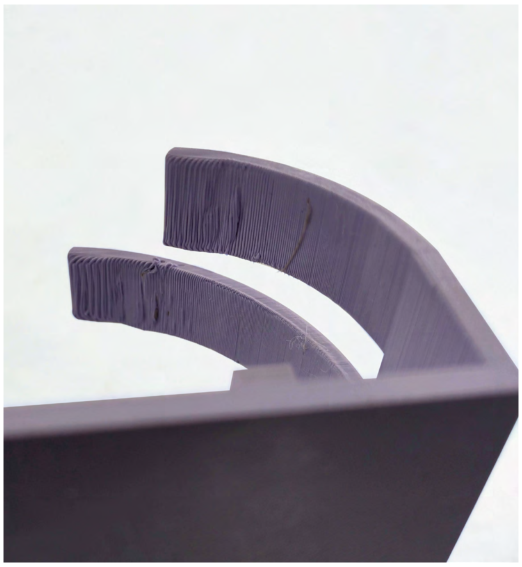


Figure 2 (left) From the top, these overhangs look good, even up to 80 degrees

Figure 3 (right) You can see some artifacts on the bottom of the overhangs. Whether or not these are a problem depends on what you want from the model

there's a set of choices that deliver different pay-offs. Back when we started 3D printing, many people recommended around 190 °C for PLA, but many modern printers default to 230 °C. What's changed?

Slicer programmers try to pick a good compromise for most models, but this won't necessarily be the optimum for all cases. As you heat up a thermoplastic like PLA, PETG, and the other plastics that we use for 3D printing, they become less viscous – or to put it another way, more runny. This causes a few effects. It means you can get plastic out the nozzle and onto the print quickly. This is obviously important for high-speed printing. However, it also means that the plastic has to cool down more before it's solid, which is bad news for bridges and small prints (as one layer has to harden before the next can go on top of it).

When you buy PLA, you're rarely buying 100% pure PLA – almost all manufacturers add things to it. This includes colourants, but often also things to help the PLA flow better. The end result is that two reels of PLA from two different manufacturers may behave differently at the same temperature, and so may have different optimum temperatures for printing.

The hard way of calibrating temperature is to slice the same model at different temperatures and print it out to see what it looks like. Fortunately, there's an easy way. There's a type of test called a 'temperature tower'. This adjusts the temperature as it moves in the Z axis so you can see how well it prints. Typically they include long bridges, overhangs, and stringing tests, as these are three common problems associated with print temperatures. You can't slice a temperature tower in the normal way. Instead, you have to get G-code created specifically for your printer. For many Ender-3 clones, the G-code is often transferable. For more modern printers, you might have to get G-code created specifically for your printer. There are lots of options online if you search Thingiverse or Printables. We used the Prusa MK4 version available at **hsmag.cc/mk4temptower**.

We were pretty surprised at how little difference we got across a range of temperatures. We last ran a tower like this on a printer about four years ago and then there was a huge range of problems at different temperatures (and before you ask, yes, we did check that it really was printing at the temperature given on the tower).

It's fine margins, but if you look at the smaller cone, there's better detail on the higher temperature, though this is also joined by a slight increase in stringing.

YOUR MILEAGE MAY VARY

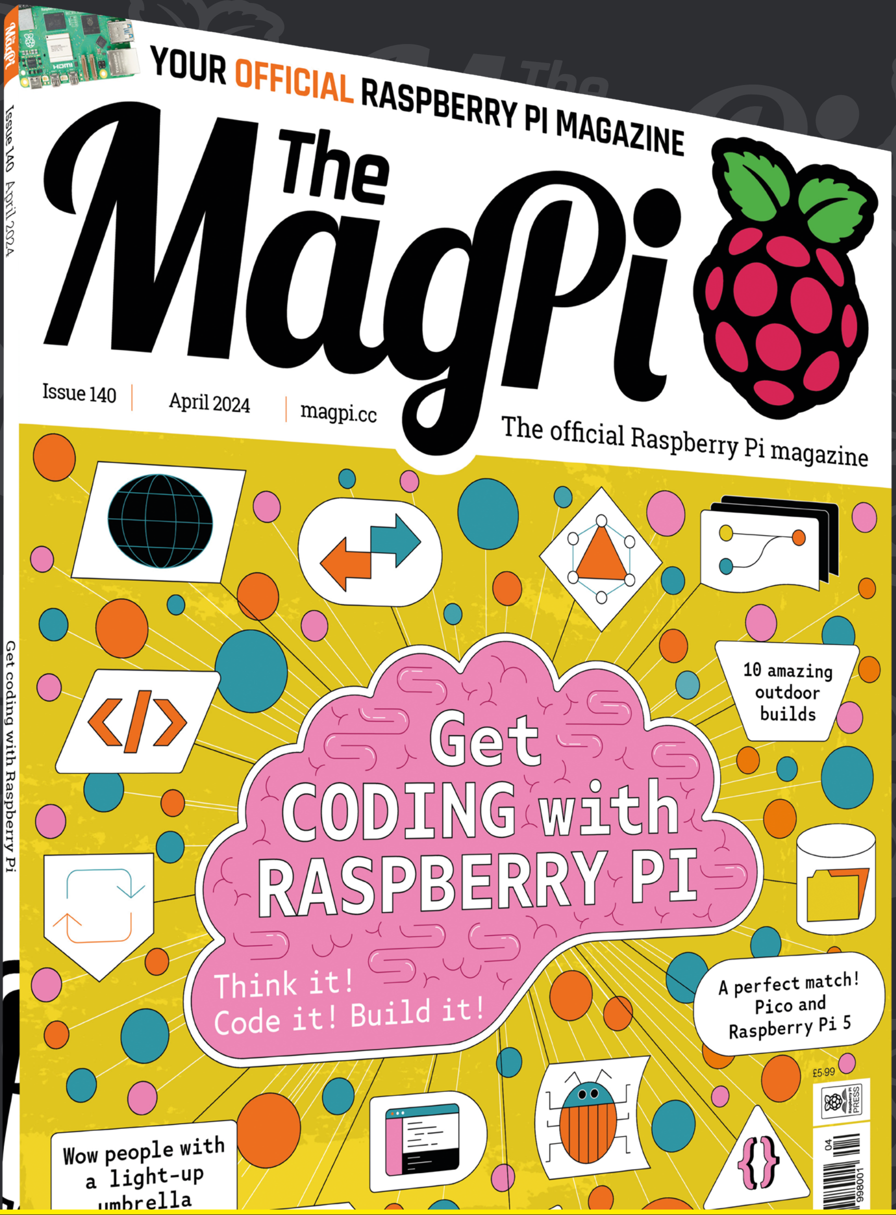
Depending on your printer, you may get a result like ours, or a more pronounced result. Remember that this may give different results with different brands of filament so, while there are some printer-specific factors, this isn't just a one-time calibration.

When you see the results for your printer, you can make a decision as to what temperature to print at.

If you decide you want to change the temperature that you're printing at, it's worth running the overhang test again to check that you're still having the results you hope for there.

There are a myriad of different tests you can run on your printer, but these basic ones will highlight the majority of the problems that you might face. Get them printing, and you'll both be able to calibrate your printer and understand its limitations.

DON'T MISS THE **BRAND NEW** ISSUE!


YOUR OFFICIAL RASPBERRY PI MAGAZINE
The MagPi
Issue 140 | April 2024 | magpi.cc
The official Raspberry Pi magazine
Get CODING with RASPBERRY PI
Think it! Code it! Build it!
10 amazing outdoor builds
A perfect match! Pico and Raspberry Pi 5
Wow people with a light-up umbrella

SUBSCRIBE
FOR JUST
£10!

- **THREE!** issues of The MagPi
- **FREE!** Raspberry Pi Pico W
- **FREE!** delivery to your door

+ FREE
RASPBERRY PI
PICO W*

* While stocks last

Three issues and free Pico W for £10 is a UK-only offer. Free Pico W is included with a 12-month subscription in USA, Europe and Rest of World. Not included with renewals. Offer subject to change or withdrawal at any time.

magpi.cc/subscribe

HackSpace
TECHNOLOGY IN YOUR HANDS

FIELD TEST

HACK | MAKE | BUILD | CREATE

Hacker gear poked, prodded, taken apart, and investigated

ONLY THE BEST

New products for a new year

A collection of new boards and accessories

By **Marc de Vinck**

Welcome to 2024! Things certainly have changed since I started my electronics journey so many years ago. I still remember programming a POV from Adafruit via a serial port, and the amazing feeling of accomplishment that followed, as I waved it in the air the first time, and I could read the word 'HELLO'. Since those early days of learning to read schematics, breadboard, solder, and correctly identify components, I have played around with electronics both as a hobby and as a profession. And one of my favourite things is still to get that feeling of amazement as new products are launched, and learning about what they can do.

My journey started out with the BASIC Stamp, then Arduino, followed by Raspberry Pi. It's amazing how much has changed, and how much easier and more powerful everything's become. Blinking LEDs was fun back in the day – OK, it's still a lot of fun. But now, hooking up an LCD display and adding machine learning with object detection has become almost as easy, and incredibly affordable. In this Best of Breed, I'll be looking at some of the new products I want to experiment with, and hopefully you do too.

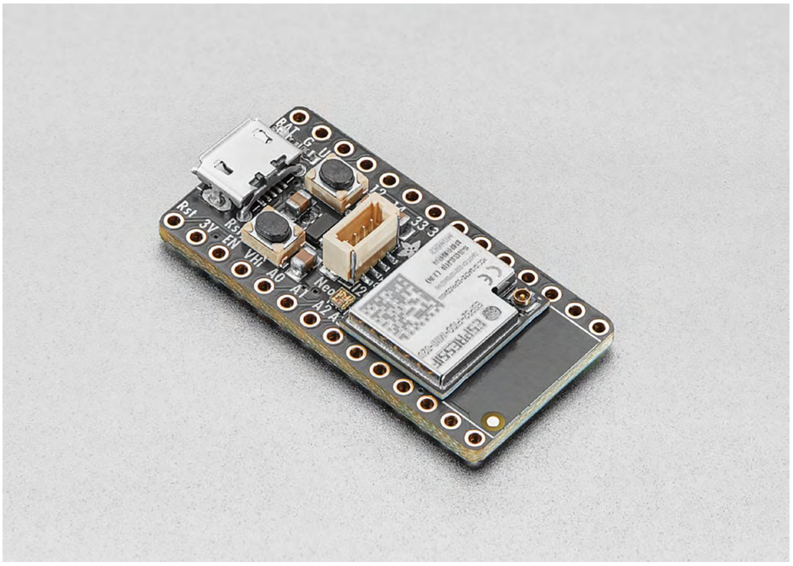
Adafruit ItsyBitsy ESP32 vs Espressif ESP32-S3-BOX-3B

ADAFRUIT $14.95 | adafruit.com

ADAFRUIT $47.50 – $49.95 | adafruit.com

The Adafruit ItsyBitsy ESP32 has been in the works for some time. The recent pandemic, and all the associated supply chain issues, certainly added to the delay. But now, you can finally get your hands on this diminutive microcontroller. So, what does this little board do? A lot!

The board features an ESP32 Pico module, which is an FCC-certified module containing an ESP32 chip with dual-core 240MHz Tensilica processor, built-in Wi-Fi and Bluetooth classic + BLE, along with 8MB of flash memory and 2MB of PSRAM. Coupled with all that power are 20 GPIO pins, including PWM output on any pin. It also has hardware-based UART and SPI, and the ability to use eight pins as capacitive touch inputs. There is more that this board can do, and there are different antenna connectors available, so you really need to check out the product page for all the details.



The ESP32-S3-BOX-3B, available from Adafruit, is a perfect platform for creating new AIoT systems. It's a fully open-source kit developed around the ESP32-S3 AI SoC, making it a perfect solution for more advanced projects. It also has built-in Wi-Fi + Bluetooth, AI acceleration capabilities, 512KB SRAM, 16MB of Quad flash, and 16MB of Octal PSRAM.

The ESP32-S3-BOX-3 can run Espressif's speech recognition framework, allowing you to build a fully offline AI voice assistant featuring far-field voice interaction, wake-up interruption, continuous voice recognition, and the ability to recognise 200+ user-customisable words. And, if you want to get into some even more advanced AI, you can also integrate ChatGPT into your project, which can lead to some amazing possibilities.

Adafruit offers this kit in a bare-bones version, but it also has one that includes additional sensors and components for just a few dollars more, making it great value.

VERDICT

Adafruit ItsyBitsy ESP32

A tiny ESP32 board with lots of functionality.

10 /10

VERDICT

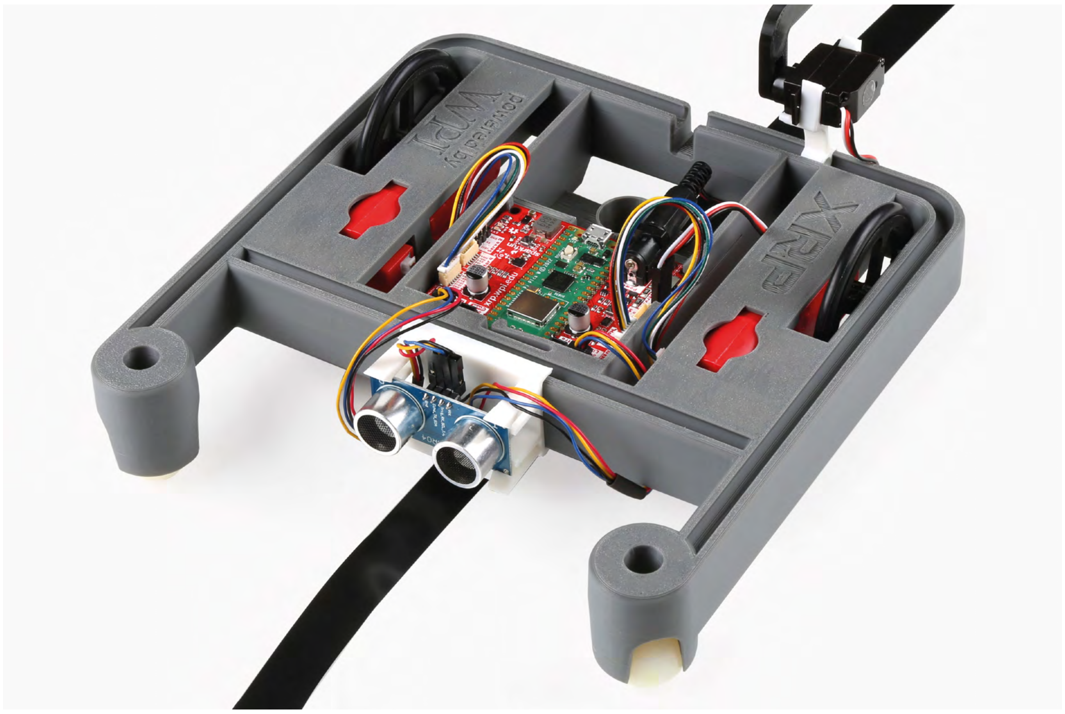
Espressif ESP32-S3-BOX-3B

A great platform for an AIoT project.

9 /10

SparkFun Experiential Robotics Platform

SPARKFUN ◈ **$115** | **sparkfun.com**



SparkFun has a new board with loads of potential called the Experiential Robotics Platform (XRP). It's designed to allow for easy hardware and software iterations of robots thanks to its compatibility with Blockly, Python, and the FIRST Robotics coding development tool. At the core of the XRP board is Raspberry Pi Pico W, enabling seamless wireless LAN and Bluetooth connectivity, as well as a familiar GPIO interface.

In addition to the Pico W, the board also features a six-axis motion sensor, dual-channel motor drivers, and a host of Qwiic connectors for adding sensors without the need to solder.

If you are getting started with robotics or are part of a FIRST team, you should look at this board. SparkFun also includes a great 'Getting Started with The XRP Controller Guide', so be sure to head on over to its site to learn more about this handy Pico-based dev board.

VERDICT

SparkFun Experiential Robotics Platform

A handy no-solder solution for robotics.

8 /10

NVMe Base for Raspberry Pi 5

PIMORONI $14.29 | pimoroni.com

Most people who build projects with the Raspberry Pi will eventually run out of storage space. There have been a few different options for adding storage in the past, but most are not as fast and easy as the NVMe board from Pimoroni. It's the perfect solution for turning your Raspberry Pi 5 into a file server or streaming media server.

This bare-bones board allows you to bring your own M.2 NVMe drive with simple plug-and-play. Don't have a M-Key NVMe drive yet? Don't worry – they sell two other variants of this board with a built-in 250GB SSD or 500GB SSD drive.

VERDICT

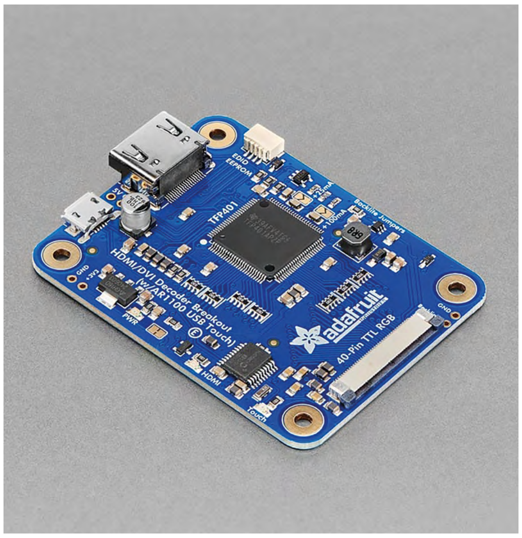
NVMe Base for Raspberry Pi 5

A simple way to add lots of storage.

10 /10

TFP401 HDMI/DVI Decoder to 40-Pin TTL Breakout - With Touch

ADAFRUIT $29.95 | adafruit.com



The TFP401 HDMI/DVI Decoder is a small board that serves a useful purpose. Now you can easily add a 40-pin TTL/TTF display, via HDMI/DVI video, to a Raspberry Pi project. What makes this board a little more unusual, other than its diminutive size, is the addition of an AR1100 USB resistive touchscreen driver. You can power the board over USB and send the signal out to a small 40-pin TFT display.

If you need to connect an 800 × 400 LCD display to your Raspberry Pi, and you want a simple way to drive it, and add touchscreen capabilities, this is the perfect solution.

Adafruit also designed a version of this board without the included touchscreen electronics. It will save you a couple of bucks, but you'll lose the beauty and functionality of integrating a touchscreen.

VERDICT

TFP401 HDMI/DVI Decoder to 40-Pin TTL Breakout - With Touch

A simple way to add a high-definition screen to Raspberry Pi.

9 /10

ARGON NEO 5 M.2 NVME PCIE CASE FOR RASPBERRY PI 5

PIMORONI $36.83 | pimoroni.com

Anyone who's been playing around with the Raspberry Pi pre Raspberry Pi 5 knows that, although the form factor is very similar, there are some distinct differences from its predecessors. With the Raspberry Pi 5, you must consider things like the heatsinks and active cooling from the fan. These add a bit of height to the overall Raspberry Pi, making many cases incompatible. The Argon NEO 5 M.2 NVMe PCIE Case is a perfect solution for your Raspberry Pi 5, allowing ample room for the board. It also features an expansion board that lets you connect an NVMe M.2 drive for simple and fast SSD storage. Pimoroni is offering a version of this case with and without the NVMe expansion board – head to its website to see the different variations.

xTool Screen Printer

Make T-shirts with a laser cutter

XTOOL ◈ From £249 | hsmag.cc/screen_printer

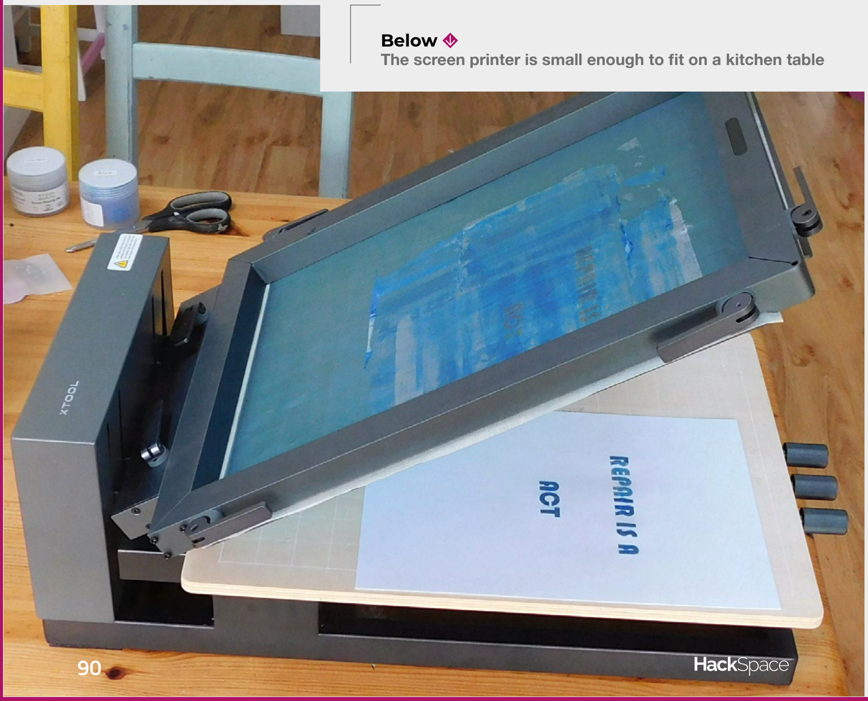
By **Ben Everard**

Right ◈
We had a few failures while we learned how to use the machine

When you think of printing, you probably think of hitting a button, then waiting while a perfect replica of the thing you have on screen pops out of a machine. That's how it works with paper printers, and these days it's pretty common for 3D printers as well. Our experiences with screen printing, however, have been a bit different. It's a more creative process that gives you plenty of scope for producing interesting, or terrible, results.

GET IF
You want to get creative with ink

Below ◈
The screen printer is small enough to fit on a kitchen table



Screen printing has been around for a long time. The basic idea is that you have a fine mesh of fabric. You cover most of this mesh with some sort of impermeable barrier, but the bit where you want ink (or paint) to get through, you leave uncovered. You then hold this screen against the thing you want to print onto and squeegee the ink (or paint) through it. If all goes well, you should have a perfect replica of your design on the surface below.

That's screen printing in general. The xTool Screen Printer is a pretty standard screen printing setup. It has a frame for holding the screens in, which mounts onto the system for moving it up and down, to allow you to place your subject below before pressing the screen down on top. All this works well and is easy to use. The thing that makes the xTool Screen Printer unique is the stainless steel mesh that's used. This is compatible with laser cutting, so you can etch out the design you want using a laser cutter (as opposed to a more common light-exposure technique). You can etch your screens using a regular laser cutter and don't need any additional equipment.

We made a mistake with our first few and etched the screens before putting them in the frame. This made it much harder than it needed to be to position the different colours in the correct place on the final object. Once we realised our mistake, and etched the screens in their frames, this got much easier. It's fairly easy to get your designs in the same place consistently, provided you have a little tolerance. If your design requires millimetre-level accuracy, then it might be hard to get it perfect every time.

This is an easy machine to use – at least by screen printing standards. We could create a design in Inkscape and etch it onto a screen using xTool Creative Space (the default laser cutting settings for the S1 laser worked perfectly, though there is a calibration design you can use for different laser cutters). This slots into the screen printer and it all works well together. You can apply and squeegee it across. The whole process can be done in under an hour. At this point, you can be as creative as you like. You can use a single block of ink, or mix colours to get a marbling effect. There are glitter inks, glow-in-the-dark inks, and other options you can try out.

It is quite a messy affair, so if you're planning on trying this in a home (rather than a workshop), you probably want to make sure you have some old newspaper or dust sheets to keep the ink off things that it shouldn't be on.

A MEDIUM-SCALE MEDIUM

Screen printing is a medium-run manufacturing technique. If you just want one T-shirt, it's far easier to use another option, such as heat-transfer vinyl, sublimation printing, or getting a T-shirt printed commercially. However, if you want a few of them done, screen printing gives you a time- and cost-effective way of making them.

The creative control you have with mixing inks gives you capabilities unlike almost any other printing technique, and can create really beautiful results. It also lets you print onto a wide range of materials. If you already have access to a laser cutter, then this is by far the easiest and cheapest way into screen printing. □

SCREEN REUSE

The screens cost £52.99 for a pack of four (you can save an additional 20% for a yearly fee of £49.99, if you're a regular user). A single screen – once etched – can replicate the design many times. However, if you want to make lots of different designs, or designs with lots of colours, the cost here could quickly rack up.

You can, in theory at least, reuse the screens with new designs (in screen printing terms, reconditioning a screen so you can put a new design on it is called reclaiming). xTool suggests doing this in the same way as with regular screen printing screens. That is, using emulsion remover to first clean the screens, then use photo emulsion to recover the screen.

We can't say how often you can do this before the screens become too worn, though ours did start to get kinked and creased after a few uses. We've had it suggested to us that you could use regular emulsion paint rather than photo emulsion (since we don't need to use light to set the pattern). However, we haven't tested this out. It may even be possible to remove the previous design using the laser rather than emulsion remover, though this might shorten the life of the screen.

VERDICT

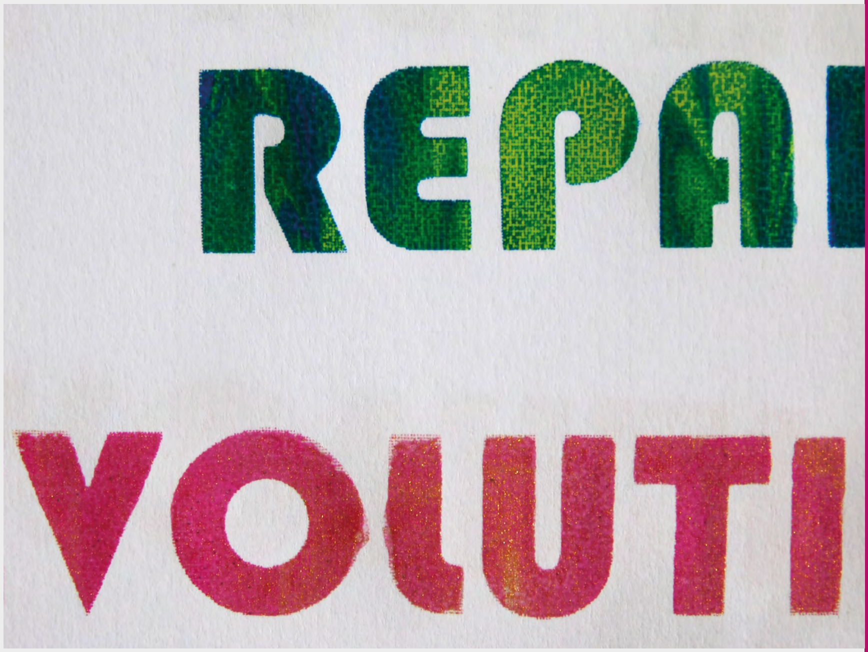
The best screen printer for hobbyists – if you own a laser cutter.

10 /10

AVOID IF

You only want one T-shirt

Below ◈
The prints didn't come out perfectly every time (this might be our inexperience), but we love the effect you can create by mixing ink



REVIEW

Video Game Module for Flipper Zero

A module that's not really for video games

FLIPPER DEVICES INC. ◈ $49 (not including Flipper Zero) | hsmag.cc/flippervgm

By **Ben Everard**

The recently released Video Game Module sits on top of the Flipper Zero; so, before we can look at what the former does, we need to understand the latter.

At a basic level, it's a microcontroller with a range of short-range wireless connectivity strapped on. That includes sub-GHz radio (which can read and send at the popular 866 and 433MHz frequencies), RFID, NFC, infrared, and iButton (which is a sort of circular electronic key). You can use this to read, understand, replay, or transmit data.

Much attention has been paid to the Flipper Zero's use as a device for attacking hardware, and you'll see videos online of it being used to bypass security, particularly on cars. There are, undoubtedly, some older models of car that are vulnerable to things that the Flipper Zero is able to do. However, they only work under very specific circumstances and, frankly, if you're looking to steal cars, there are far better devices out there than the Flipper Zero (and no, we won't recommend any). Despite this, the government of Canada has decided to ban the Flipper Zero from sale there.

While the Flipper Zero might not be the key to instant criminal success, it is a great tool for learning about how the wireless communications that whiz through the air all around us actually work. For example, while the Flipper Zero almost certainly won't unlock your car, it can be used to read the signals that go between your car and the key fob, and so you can learn about why such a simple device can't unlock a car (hint: it's rolling codes).

You may have some remotely controllable devices sending data back and forth – such as a weather station or other sensor. These commonly operate in the sub-GHz frequency range, and using Flipper you can see what data is being sent. You could do this because you're interested in the security implications of sending the data, because you're interested in intercepting the data for other purposes (such as getting the sensor data into a computer), or simply because you're interested in understanding a bit more about wireless data transfer.

There is also, for reasons we don't fully understand, a dolphin as a digital pet.

Right ◈
You can play games with the module, but it's not the best arcade interface

Below Right
The Video Game Module plugs into the GPIO on the Flipper Zero

GET IF
You want to investigate wireless on a big screen

VIDEO OUT
VIDEO GAME MODULE
Powered by Raspberry Pi
BOOT
RESET
TO ENTER UF2 FLASH MODE HOLD BEFORE POWER-UP
RASPBERRY PI RP2040 USB PORT
Install Official Firmware
Install Firmware from File

The Video Game Module can be used to play video games. There's an app called Air Arkanoid that lets you play the classic block-breaking game on a TV or monitor, and a game engine (**hsmag.cc/FZGameEngine**) that you can use in your own games.

While there is a long history of computer hardware that's ostensibly for serious work really being used to play games, this, we believe, might be the first bit of hardware that is ostensibly for playing games that will be used mostly for work. The two key things that enable it to play video games – graphical output that will work with most modern TVs and monitors, and motion sensitivity that allows input from tilting the device – are also useful for a range of other things.

NOT JUST VIDEO GAMES

The above is achieved using the official firmware for the module. However, you can program it just as you would any other RP2040-based device. In fact, you can store a range of pre-compiled firmware on your Flipper Zero and load them on the go without needing a computer.

Writing additional firmware enables you to access the full range of features of the module and the RP2040 chip on it. The eleven additional GPIOs can be used with the RP2040's Programmable IO system which makes it possible to implement a wide range of low-level protocols. This combines well with the Flipper Zero's existing capabilities to create a device that can analyse both wired and wireless traffic. However, at present, achieving this would require writing the interfacing code for the RP2040 yourself as it doesn't currently exist. Obviously, this could be done without the Flipper Zero, and a standalone RP2040 board, but if you're using a Flipper Zero for other analysis, this brings more into the same tool. This setup would also let you interface almost any hardware with Flipper Zero's wireless capabilities.

"You don't actually have to have a Flipper Zero to use the Video Game Module – it's a standalone device**"**

You don't actually have to have a Flipper Zero to use the Video Game Module – it's a standalone device. For example, there's Scopy firmware available that turns the module into an oscilloscope (when paired with a phone that acts as the front end). This is probably best viewed as a 'nice to have' rather than a fundamental feature. We can't imagine many people wanting the module for this alone, but if you need the module for something else, then it's a handy addition to have an RP2040 board available.

The primary use of the Video Game Module, we suspect, will be to provide video output – whether that's for demonstrating the hardware, or just because a particular person prefers to use a big screen. However, the combination of flexible I/O on the RP2040, and flexible radio on the Flipper Zero, could lead to some great applications. □

AVOID IF

You just want to play games

Above You can flash firmware onto RP2040 directly from Flipper

VERDICT

Add video output and very capable GPIO to your dolphin-based RF analyser.

10 /10

Tiny Tapeout

Can you learn chip design in an evening?

TINY TAPEOUT ◈ $150 | tinytapeout.com

By **Ben Everard**

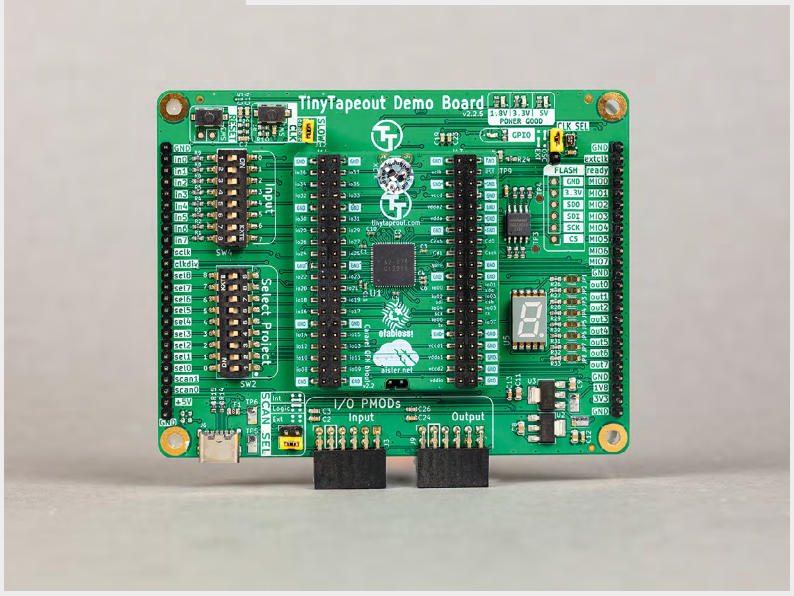
GET IF

You want to learn more about how modern electronics work

About 18 months ago, this reviewer took a class in Bristol Hackspace. There, Matt Venn took a group of novices through the very basics of chip design.

Microchips have been, perhaps, the defining innovation of the past 100 years. They are what enable the modern world to exist. But what are they? If you peel back the black plastic coating, what would you find inside?

Right ◈
The microchip comes on a removable daughterboard



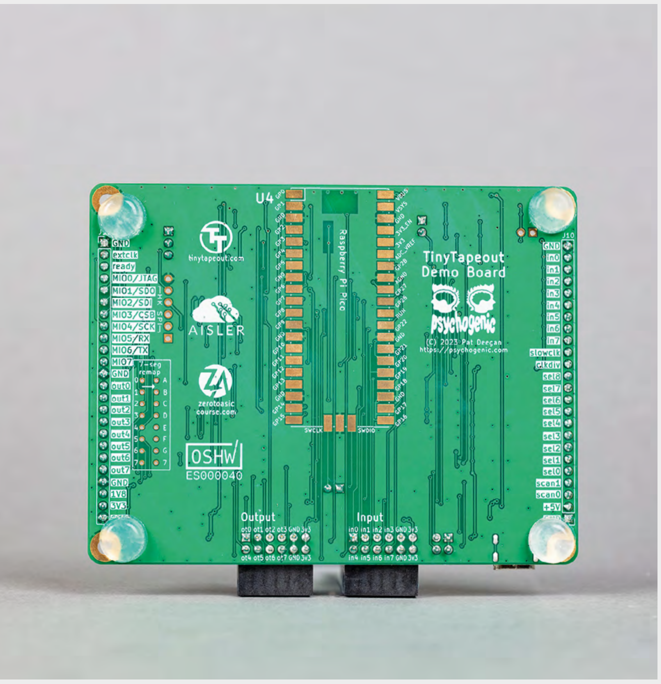
Here at HackSpace magazine, we're very keen on learning by making. If you want to know how something works, building it is an excellent place to start. However, historically this has been very difficult if not impossible to do with microchips. The costs involved put it out of reach for most mere mortals.

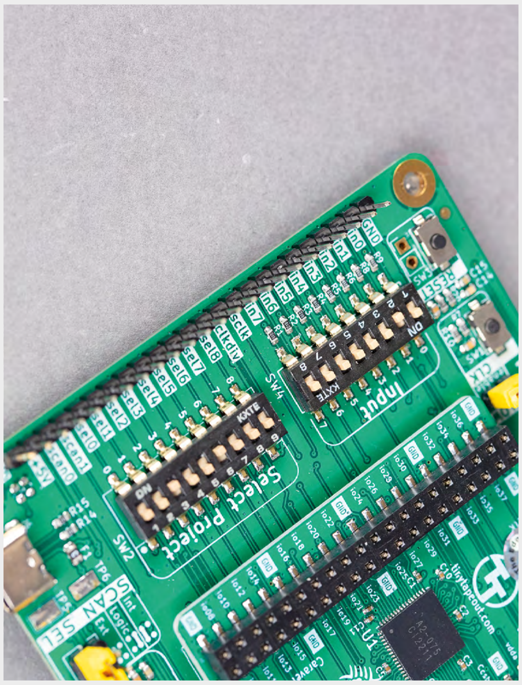
Costs have come down in recent years, but they're still pretty big. One man, however, has a solution. Matt Venn's approach with Tiny Tapeout is to split up the die of a microchip into multiple parts in such a way that a single chip can hold many different designs. There are eight inputs and outputs, but these can be sent to any of up to a few hundred designs. This, coupled with some subsidised chip manufacturing, brings the cost of getting a microchip made down to $150 for the latest run. While this isn't exactly pocket change, it's within the reach of many makers.

For this price, you get the microchip made and mounted on a PCB that includes a clock source, LEDs for output, and switches for project select and input. There is also mounting for a Raspberry Pi Pico on the back, to allow you to control your section of the chip from software.

Following Matt's class, we spent a bit of time working on our design, and came up with an LED flasher that could flash a few different patterns, depending on the state of some inputs.

You can create your designs in a few different ways. We used the graphical Wokwi tool, which enables you to drag and drop components together. You can also use a hardware description language such as Verilog.





AVOID IF

You're too impatient to wait over a year for hardware

Above Left
Solder on a Pico and you'll be able to interact with the projects from software

Above Right
Flip the selector switches to pick the project you want to enable

Once we'd created our design, we added it to the GitHub project which compiled our design to the format needed by the microchip fabricator.

EXPLORING THE SILICON

Chip design isn't a speedy process at the best of times. We can get a bit frustrated when PCBs take a few weeks to get made and shipped, but it was over a year before we finally received our boards. The chip arrived already mounted on a demo board that made it easy to test out the projects, though the chip itself does come on a daughterboard, so can be dropped into another PCB if it's a part of a larger project.

On the board is not just our design, but 165 designs that you can select and try out. Some of these need specialist hardware to run, but many of them we can test out. Every author submitted a datasheet with details of what their design does and how to test it out – find these at **tinytapeout.com/runs/tt02**. Our board arrived set up to run Skyler Saleh's 'Hello Generator' design which flashes out the letters 'h', 'e', 'l', 'l', and 'o' on the seven-segment display.

We just have to flip the selector switches to our design. With an 18-month turnaround time, we were really hoping that it worked. Fortunately, it did. The lights flashed on and off as they were supposed to. It's probably not a design that we'll take forward or use in a project (OK, maybe we might have the world's most over-engineered Christmas lights next year), but it's a chip with our design etched into the silicon. If you want to explore further, there are ten different CPU designs on this particular run of Tiny Tapeout (2) that you can try out and code for (though don't expect it to be easy), as well as many others to explore. The documentation may, or may not, be good for any of them, but the source code of the hardware itself is online for you to look through. It's an open-source chip full of designs for you to explore and play with.

LEARNING TO DESIGN

We attended an in-person event, but that's not necessary. Anyone can submit a design, and if you've not done any chip design before, there are a series of lessons on the Tiny Tapeout website to help you get started: **tinytapeout.com/digital_design**.

Here at HackSpace magazine, we're in a tremendously privileged place where we can try out a huge number of different techniques and styles of making. However, among these, Tiny Tapeout stands out. It gave us a chance to peer behind the curtain and see how these things are made. Our design was barely more than a 'hello world' of chip design, but it was ours and it got made, and that's special. If you're interested in computing, then we'd highly recommend the experience of designing your own microchip. □

VERDICT

Learn how microchips work by designing one.

10 /10

CROWDFUNDING NOW

BLSTRsander

Sand-blasting set free

From $299 | hsmag.cc/blstrsander | **Delivery:** Jul 2024

This claims to be a new sort of power tool, which is stretching the truth a little. It's a sand-blaster, and we've had those for years. However, it does work in a slightly unusual way – instead of compressed air firing the sand, it uses spinning discs. Does this matter? The Kickstarter doesn't really tell us. It's slightly more portable, but a connected air hose isn't really the biggest impediment to movement. Yes, you do need to be connected to a compressor, but unless you're working out of the back of a van, that's usually not a problem. Given that the BLSTRsander is corded, the difference between being tethered by a power cable and an air hose doesn't seem huge.

It's not even a saving in price over a regular sand-blaster plus compressor (even if you have to buy the compressor).

One graphic on the Kickstarter implies that you get a bit more control over the power than on a traditional blaster, but there's not much detail on this.

Overall, it's an intriguing idea, but until we get hold of one we're struggling to see the benefit. □

BUYER BEWARE

When backing a crowdfunding campaign, you are not purchasing a finished product, but supporting a project working on something new. There is a very real chance that the product will never ship and you'll lose your money. It's a great way to support projects you like and get some cheap hardware in the process, but if you use it purely as a chance to snag cheap stuff, you may find that you get burned.

Spark Analyzer

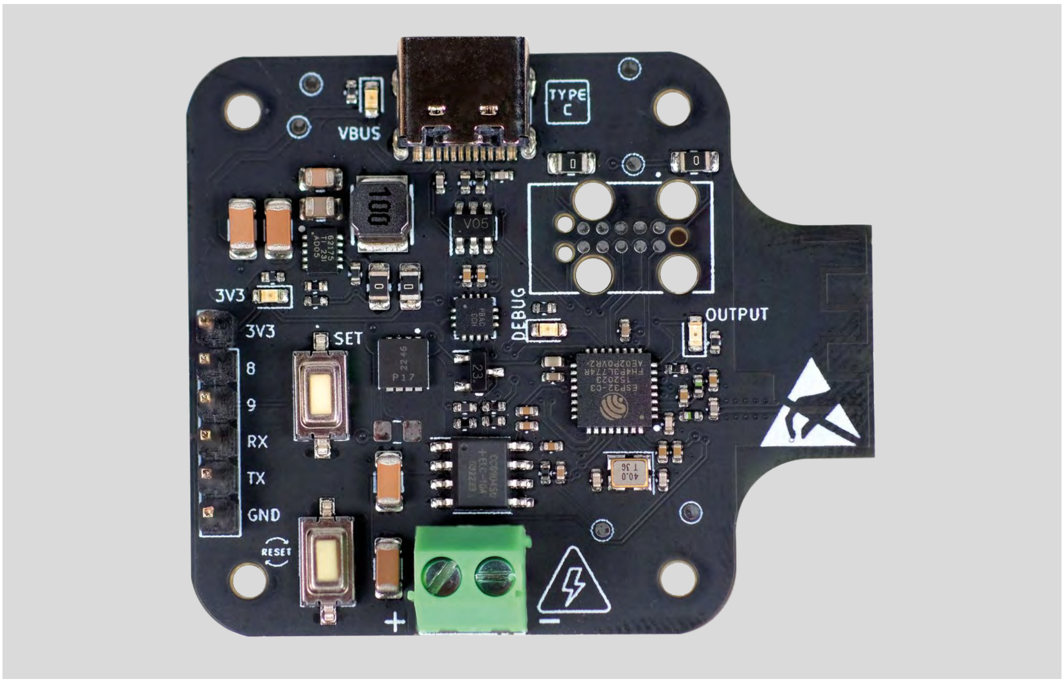
Monitor your power usage

From $49 | **hsmag.cc/spark_analyzer** | **Delivery:** Aug 2024

This little board takes USB-C Power Delivery (PD) input, and outputs a variable voltage while monitoring the current usage. All this data is then available in a phone app that you can use to remotely adjust the voltage and control the device.

There are a couple of downsides to this device. Firstly, the output voltage isn't adjustable so much as selectable. It can be 5 V, 9 V, 15 V, or 20 V. These match the voltages in the USB PD spec. For example, there's no 3 V option. The Crowd Supply page doesn't list the accuracy, but the screenshot shows it measuring current in milliamps, so this won't be suitable for monitoring low power and sleep options on microcontrollers, which typically draw less than this.

While this does have limitations, there are definitely benefits to this. It looks like it'll be an easy and flexible way of keeping track of the current draw on your medium-power projects. □



next month

issue #78

ON SALE
25 APRIL

The Future of making

ALSO

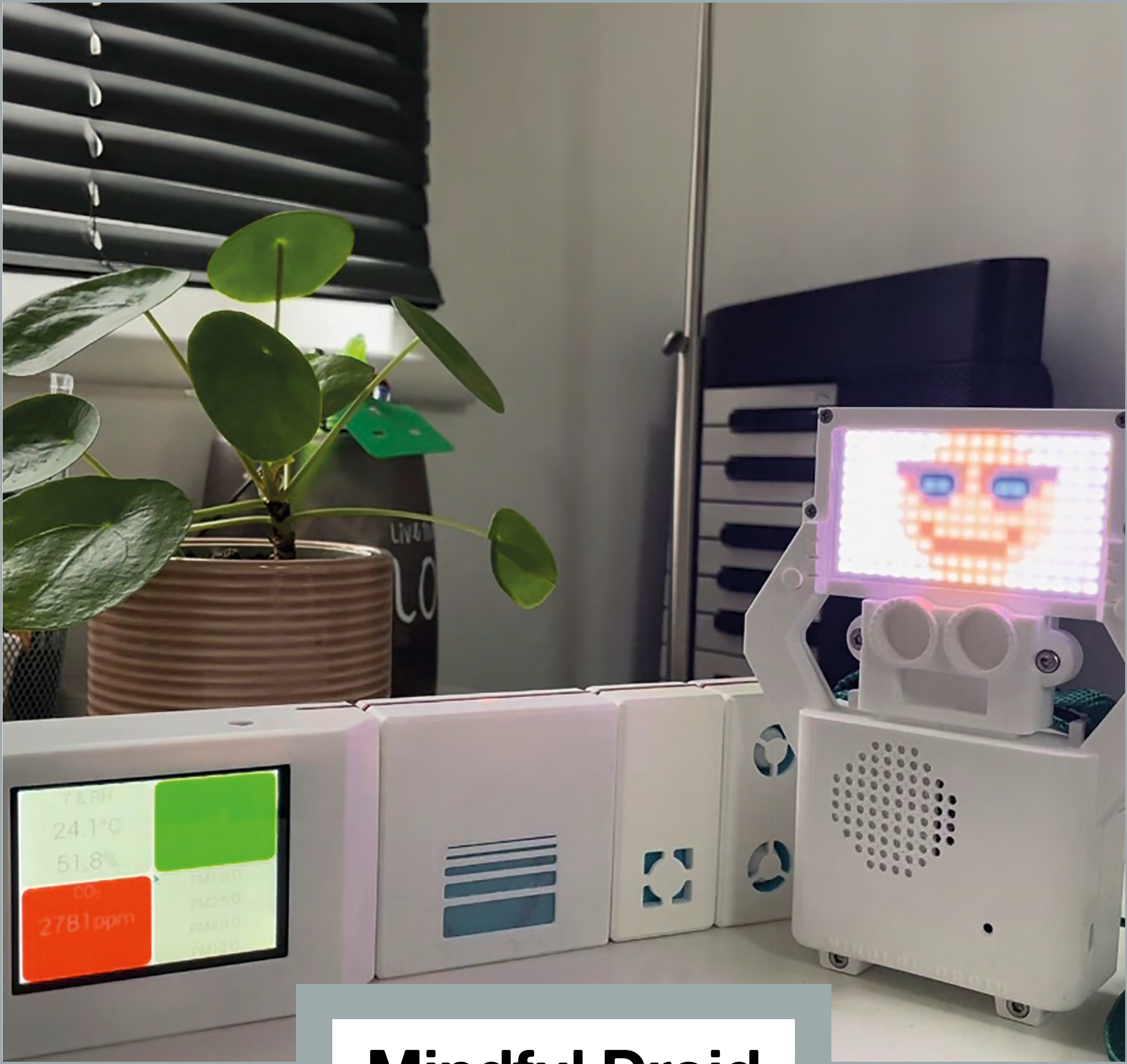
- RASPBERRY PI
- 3D PRINTING
- PYTHON
- MUSIC

AND MUCH MORE

—

DON'T MISS OUT

hsmag.cc/subscribe



Mindful Droid

We're living in a golden age for citizen science. Sensors are more plentiful and cheaper than ever; microprocessors are easier to work with than they have ever been; networking options proliferate. But without an imaginative way of representing it, data can still seem somewhat abstract. That's especially so when the thing you're measuring is air quality. Step forward, Michael Omotosho's Mindful Droid.

Taking data from its own on-board VOC and CO_2 sensors, and receiving data from the sensors on the Environmental Sensor Development Kit from DesignSpark, the Mindful Droid makes it impossible to ignore the pollution in the air around us. It's a great example of engineering being more than just making physical things – it's imagination, and activism too.

PiKVM

Remote control redefined

Manage your servers or PCs remotely!

POWERED BY Raspberry Pi

PiKVM V4 Mini

Small, cost-effective, and powerful!

- Power consumption in idle mode: just 2.67 Watts!
- Transfer your mouse and keyboard actions
- Access to all configuration settings like UEFI/BIOS
- Capture video signal up to 1920x1200@60 Hz
- Take full control of a remote PC's power

PiKVM V4 Plus

The most feature-rich edition

- More connectivity
- Extra storage via internal USB 3.0
- Upgraded powering options
- More physical security features
- Extra HDMI output
- Advanced cooling solution

A cost-effective solution for data-centers, IT departments or remote machines!

Available at the main Raspberry Pi resellers

core ELECTRONICS · PiShop.ca · RPishop.cz · Welectron. · elektorstore · PIMORONI

Pi-Shop.ch · The PiHut · PiShop.us · CloudFree · KUBII · Cytron Marketplace

ROBU.IN Your ideas, Our Parts · NETTOP COMPUTERS & ELECTRONICS · netsket mall · DIREKTRONIK · RaspberryPi.dk

HiPi.io

No reseller in your country?
Check shop.hipi.io (import fees might apply).

List of official resellers by country: