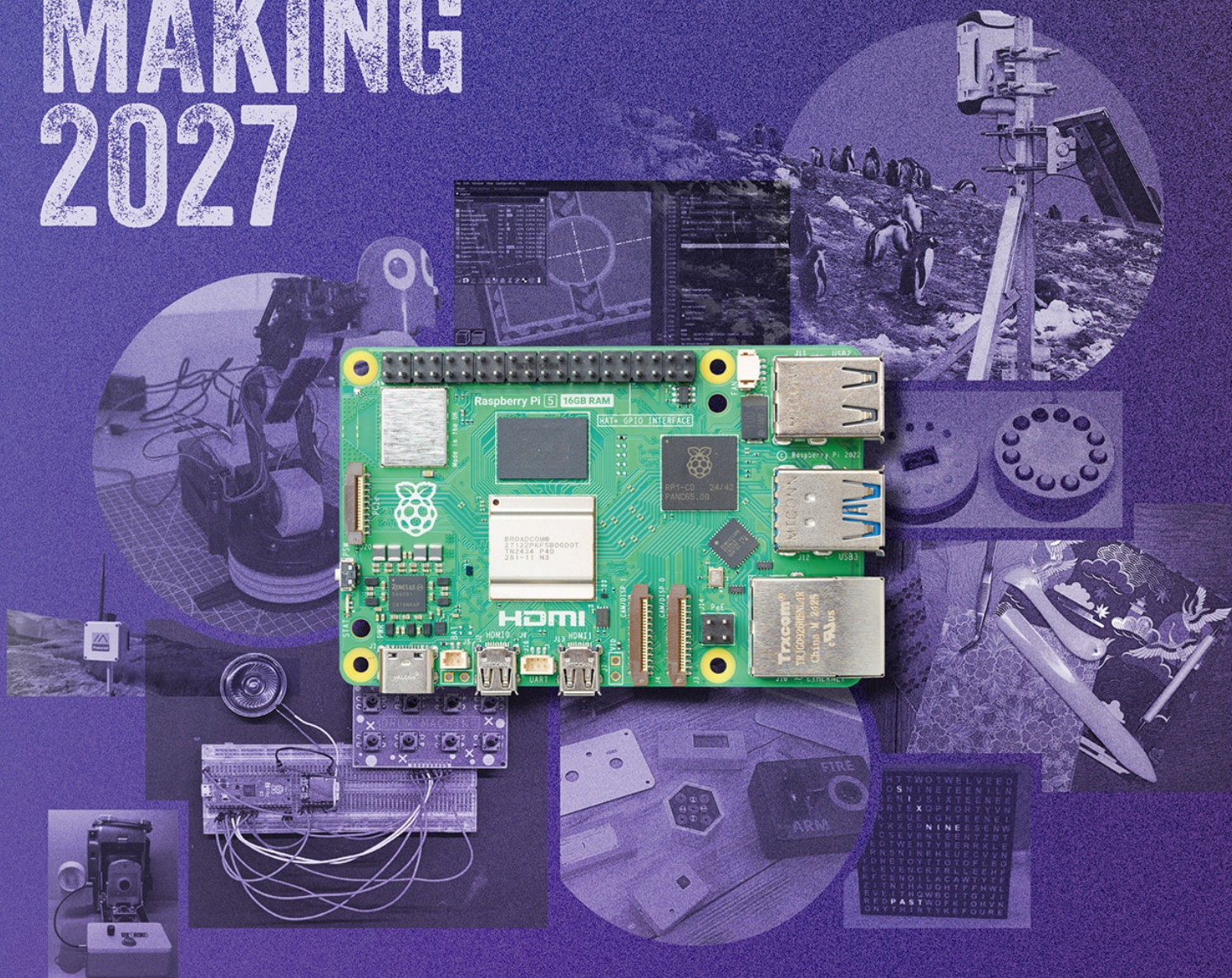


PROJECTS FOR MAKERS AND HACKERS

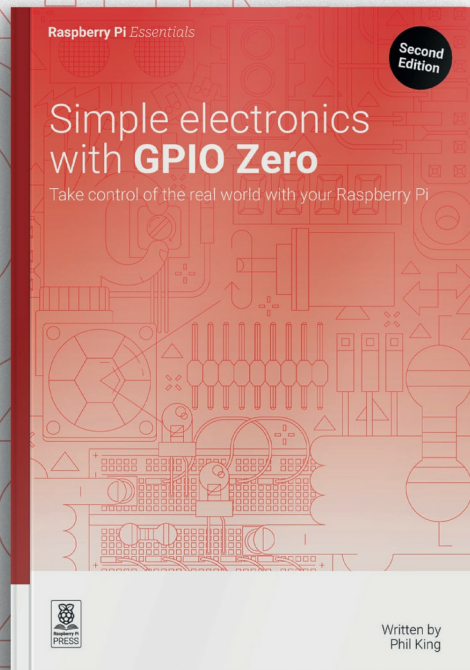


Raspberry Pi
Book of Making
2027

Raspberry Pi BOOK OF MAKING 2027



LED origami | Smart CO₂ sensor | Make your own
PCBs | Underwater rover | Wordsearch clock
Build a jukebox | Control a robotic arm | And more



Raspberry Pi's GPIO header allows you to connect electronic components and control them with code you've written yourself. Python is the most popular programming language for controlling electronics on a Raspberry Pi, particularly the functions in the GPIO Zero library. With this book, you'll learn how to use GPIO Zero as you build a series of simple electronics projects.

■ **Simple electronics projects including:**

- Program some LED lights
- Add a push button to your project
- Build a motion-sensing alarm
- Create your own distance rangefinder
- Make a laser-powered tripwire
- Build a Raspberry Pi robot

BUY ONLINE: rpimag.co/gpiozerobook

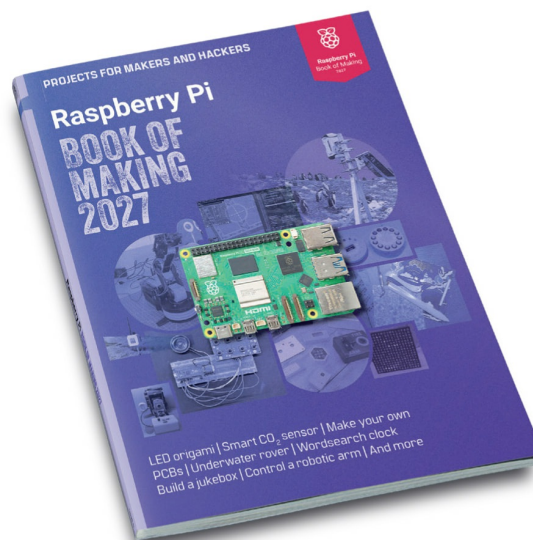
Welcome to the Book of Making



We're in the middle of an industrial revolution. Not a loud, dirty industrial revolution, powered by steam, diesel, and men with hammers and pickaxes; this one is quieter, cleaner, and entirely digital. We're living in an age where you can design a prototype, run it through a 3D printer, program the electronics yourself, then send the files to a factory to get made at scale. All from home, and all digitally.

Book of Making 2027 is our attempt at capturing some of the possibilities of this new digital frontier. We've rounded up a few of our favourite makes previously published in *Raspberry Pi Official Magazine* to fire your imagination. Now get making!

Andrew Gregory – Editor



Editorial

Editor

Andrew Gregory
andrew.gregory@raspberrypi.com

Sub Editor

Phil 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

Brand Manager

Brian O Halloran

Contributors

PJ Evans, Ben Everard, Lucy Hattersley, Jo Hinchliffe, Rob Miles, Nicola King, Phil King, KG Orphanides

Publishing

Publishing Director

Brian Jepson
brian.jepson@raspberrypi.com

Director of Communications

Helen Lynn

CEO

Eben Upton

Distribution

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

Subscriptions

The Maltings, West Street
Bourne, Lincolnshire, PE10 9PH
+44 (0) 1778 392472
rpimag.co/subscribe
raspberrypi@warnersgroup.co.uk



This magazine is printed on paper sourced from sustainable forests and the printer operates an environmental management system which has been assessed as conforming to ISO 14001. Raspberry Pi Official Magazine is published by Raspberry Pi Ltd, 194 Cambridge Science Park, Milton Road, Cambridge, England, CB4 0AB. The publisher, editor, and contributors accept no responsibility in respect of any omissions or errors relating to goods, products, or services referred to or advertised in the magazine. Except where otherwise noted, content in this magazine is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported (CC BY-NC-SA 3.0).



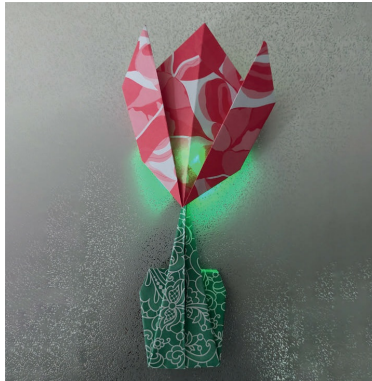
ISBN: 978-1-916868-09-0

Contents

Afternoon

Soap and mould making

3D print fragrant Christmas gift



Origami paper circuits

Make PCBs out of mashed-up wood fibres

Sewing with LEDs

Add lighting effects to your coolest clobber

Publish your own zines

Your kitchen table publishing empire starts here!

Flower pressing

Preserve the beauty of nature with Raspberry Pi

Build miniature rooms

Control lighting effects in a diorama

Sew a touch sensitive quilt block

Add capacitive touch sensing to your bedclothes

Get started with electronics

Our introduction to the wonderful world of the electron

Weekend

Smart CO2 monitor

Keep an eye on the air that you breathe

Electronics enclosures

Custom cases for your electronic creations

Pico drum machine

Keep the rhythm going with a Raspberry Pi Pico

Hack a robot arm part 1

Control a shop-bought robot with Python

Hack a robot arm part 2

More off-the-shelf robotics

Camera shutter release

Take photographs remotely with Raspberry Pi Pico

Make a digital jukebox

Listen to music like you're in a 1950s diner

Underwater Raspberry Pi

Amazing aquatic projects powered by Raspberry Pi

Animate a toy skeleton

Bring a spooky skeleton to life with sensor input, stepper motors, and Raspberry Pi

Inspirations

Design your first PCB

Build a rain monitor using a Raspberry Pi Pico W

Industrial 3D printing

Professional printing at pocket-money prices

Manufacture a complete build

See your idea turned into a factory-made prototype

Bake your own circuits

Pop your ingredients in the oven, and out comes a PCB

Meshtastic part 1

Low power, long-range communications with Raspberry Pi Pico



[Meshtastic part 2](#)

[Add solar power to your remote communication node](#)

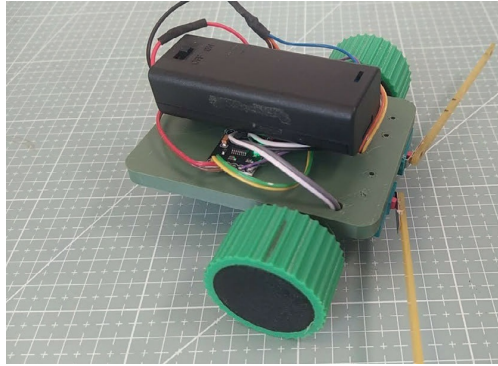
[Build an underwater rover part 1](#)

[Turn 3D printing and PVC pipes into a submarine](#)



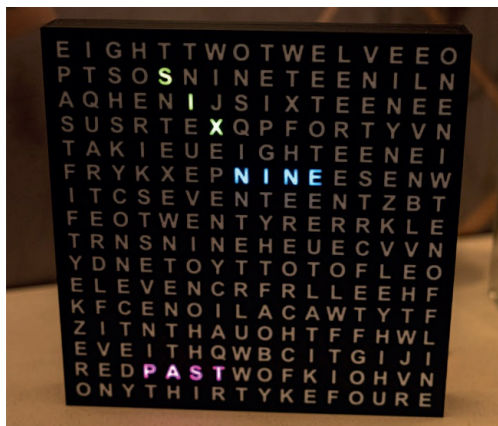
[Build an underwater rover part 2](#)

[Add a camera and a computerised drive control system](#)



Simple sensor robot

[Build a small, but perfectly formed, robot using a Raspberry Pi Pico](#)



Wordsearch clock

[Design and build a unique maker timepiece](#)

Some of the tools and techniques shown in this book are dangerous unless used with skill, experience and appropriate personal protection equipment. While we attempt to guide the reader, ultimately you are responsible for your own safety and understanding the limits of yourself and your equipment. Book of Making is intended for an adult audience and some projects may be dangerous for children. Raspberry Pi Ltd does not accept responsibility for any injuries, damage to equipment, or costs incurred from projects, tutorials or suggestions in this book. Laws and regulations covering many of the topics in this book are different between countries, and are always subject to change. You are responsible for understanding the requirements in your jurisdiction and ensuring that you comply with them. Some manufacturers place limits on the use of their hardware which some projects or suggestions in this book may go beyond. It is your responsibility to understand the manufacturer's limits.

Afternoon projects

When you have a couple of hours to yourself, these projects are quick to make – have fun!



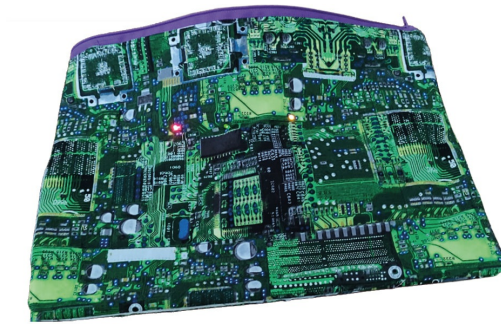
Soap and mould making

3D print a unique shape, then fill it with stuff to make you smell nice



Origami paper circuits

[Build complex three-dimensional shapes... that light up!](#)



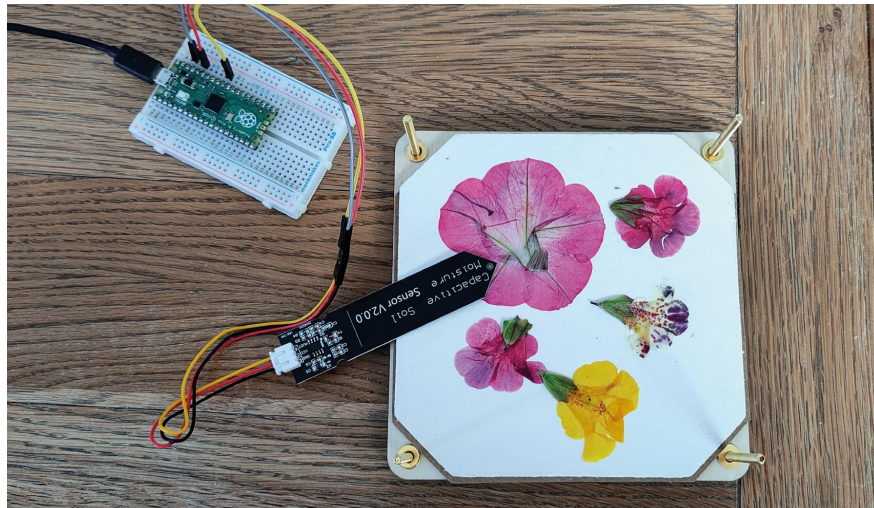
[Sewing with LEDs](#)

[Add custom lighting effects to your clothes](#)



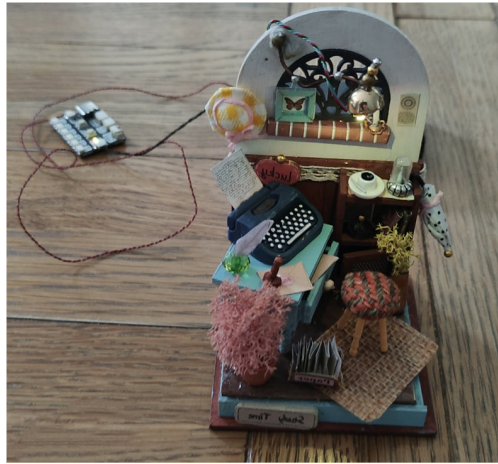
[Publish your own zines](#)

[Drag your eyes away from the glowing screen and read something on real paper!](#)



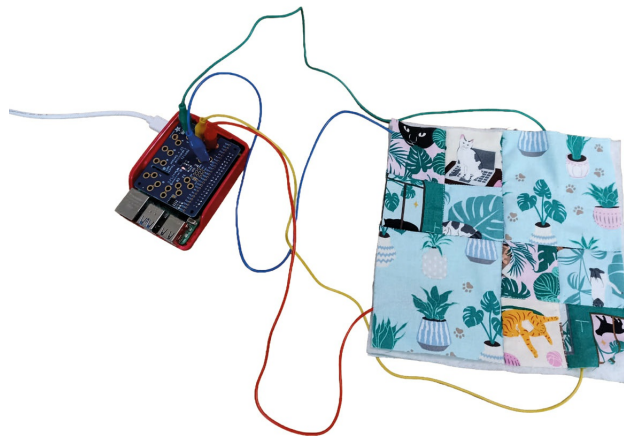
[Flower pressing](#)

[Use a moisture sensor and Raspberry Pi to preserve the beauty of nature](#)



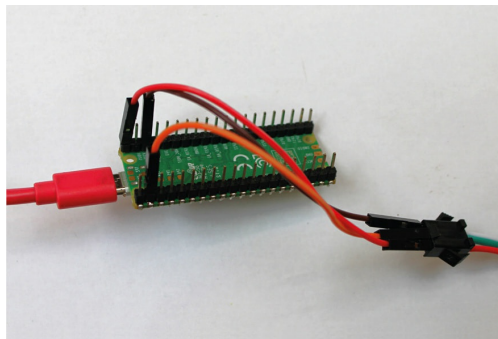
[Build miniature rooms](#)

[A simple way to control lighting effects in a diorama](#)



[Sew a touch-sensitive quilt block](#)

[Add capacitive touch sensing to a quilt block being warm was never so high-tech!](#)



[Get started with electronics](#)

[Capacitors, resistors, circuits – make sense of it all with our introduction to electronics](#)

Soap and mould making

3D-print your own mould and hand-make a bar of an all-natural daily staple



Maker

Nicola King

Nicola is a freelance writer and sub-editor. Her load of flannel this month is very much intended to be used with some soap.



@holtonhandmade

We'll take a look at soap bar making, with an eye on 3D-printing your own soap mould too

It's a daily necessity and common household substance that we all use in some form or another as, not only does soap cleanse, but it also sanitises and destroys bacteria. Following the Industrial Revolution, alongside a growing awareness of sound sanitary practices and advertising to encourage bathing with soap, soap bars became popular and mass production swiftly followed. Today, we don't think twice about buying bars of soap with our groceries, and may sometimes opt for slightly fancier soap bar versions (perhaps a certain French brand that this author is partial to) as gifts for friends and family.

Soap, though, is also something you can make yourself, to your precise specifications in whatever colours, shapes, or sizes that you like, and as gifts for others. You can make your soap nourishing, exfoliating, scented, embossed, and customise as you wish. In this tutorial, we'll take a look at soap bar making, with an eye on 3D-printing your own soap mould too, and using software (Blender) running on a Raspberry Pi to design a mould.

Soap science

A bar of soap is usually made of a fat/oil/wax combined with an alkaline substance often referred

to as 'lye' (aka sodium hydroxide). Saponification is the result when the fat and the alkali are heated together and a natural chemical reaction occurs. In addition, soap molecules are both hydrophilic and hydrophobic, so they bind to both grease and water – so when you wash with soap and water, the molecules help lift the dirt, germs, and oils from your skin and can be rinsed away.

QUICK TIP

If adding mica powder/colour to your soap, be subtle. Too much will bleed out into any lather from your soap.

Other soap-making methods

Apart from the 'melt and pour' technique already discussed, which is best for beginners, there are other ways of making soap. However, great care is required when dealing with some of the ingredients:

- Cold process (at relatively low temperatures, 30–50°C) soap making is where you make soap from scratch. First of all, anyone wanting to try this technique must be aware that this method involves the use of sodium hydroxide, to form something called 'lye'. Gloves, goggles, a well-ventilated workspace, protective clothing, and a face mask are essential when dealing with this substance. Basically, this process involves adding the sodium hydroxide to water and whisking, which causes a heat reaction and vapours – that you do not want to breathe in – until the sodium hydroxide has dissolved. Then, a melted oil (perhaps olive or coconut) is mixed with the lye solution when both of these solutions have cooled and are at around the 40°C mark. As the mixture turns opaque and thickens, the maker can add fragrance etc., before pouring it into a mould. It then needs to cure/set, and it can sometimes take 4–6 weeks before the soap is fully cured and ready to use.
- Hot process soap making – as with cold process, you get to choose all of the ingredients that you want contained **within** your soap. The only real difference here is that a source of heat is used to accelerate that saponification chemical reaction – instead of pouring the mixture into a mould immediately, heat is used to force saponification using a slow cooker or suchlike (don't use it for food preparation afterwards!). This means your soap is ready more quickly once cooled, but it still does benefit from a week or two to cure.
- Milling/rebatching – not as popular a technique, but it involves grating down previously made cold process soap and then mixing it with a liquid in order to create a new remoulded batch of soap.

As a footnote, soap making has existed for many centuries. The earliest recorded evidence of soap making dates to around 2800 BCE, when fats are thought to have been heated with wood ashes.



Our magnificent 3D-printed mould, in a rather fetching colour. This took around an hour to 3D-print, and we used a freely available file for the print. There are many free files out there, but you can also buy your favourite design file or, of course, design your own bespoke and unique soap mould

YOU'LL NEED

- Melt-and-pour soap base
- A heat-resistant jug/silicone spatula to melt the soap base in
- Scales
- Knife
- Essential oil/s (optional)
- Colour/dried flowers/mica powders (optional)
- Isopropyl alcohol (optional)
- Mould (3D-printed or purchased)
- 3D-printer and filament to print a mould (optional)
- Protective clothing

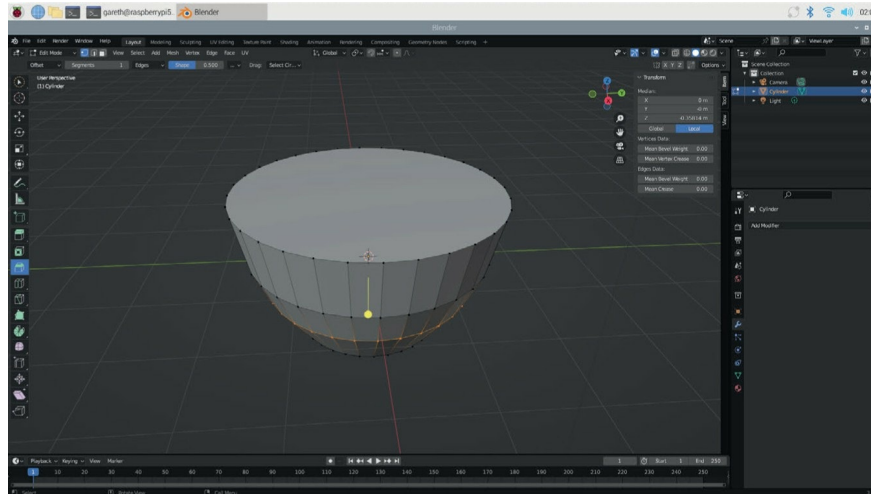


Warning!

Chemicals

We're using the beginner-friendly 'melt and pour' technique here, but if you do want to go all industrial and start using sodium hydroxide to make soap from scratch, please do it in a ventilated area and always use the correct PPE.

rpimag.co/chemicals



By adding and scaling Loop Cut rings to a cylinder in Blender, you can shape it for your 3D-printable mould

Now, whilst it would be great to have the time to make soap from scratch and watch the saponification process unfold, we've chosen to take the easiest and quickest soap-making route here: the 'melt and pour' technique, working with a premade soap base. There are many positives to this: it is by far the safest method, especially if you want to involve children in the making process, it hardens in just a few hours, and it gives you more time to concentrate on either the aesthetic look of your mould and/or your soap. The choice of soap-making technique is obviously down to individual choice. But, before we make a bar or two, let's think about what to make it in, and how a 3D printer, some software, and a Raspberry Pi can help us.

Mould making

The simple mould that we have 3D-printed for our soap was made using the following file, available under a Creative Commons licence: rpimag.co/flowersoapmould. It's designed by Lavenderguy and is a perfect, uncomplicated design that creates a really nicely shaped soap. It's also a mould that we'll be able to use again and again!

You can use any CAD software to design a 3D-printable mould, but we found the easiest to use is Blender. You can install it on a Raspberry Pi with the following terminal command:

```
sudo apt install blender
```

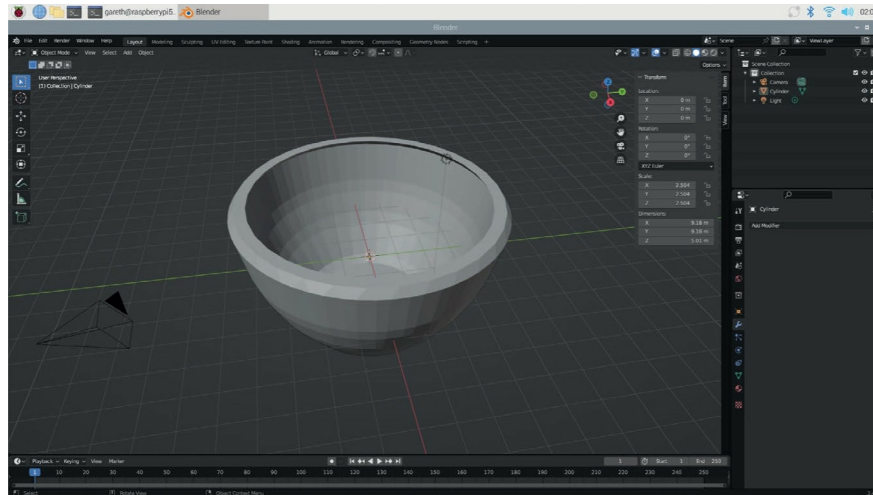
Note that as the Raspberry Pi hardware doesn't support the required OpenGL 3.0 API by default, you'll need to use a prefix to run Blender with software rendering:

```
LIBGL_ALWAYS_SOFTWARE=true blender
```

The application will then open on the desktop. Note that Blender works well on a Raspberry Pi 5 (ours has 8GB RAM), but it will run a little slower on a Raspberry Pi 4 (or earlier model).

The easiest way to create a simple mould design is by using a preset 3D shape in Blender. For instance, select Add > Mesh > Cylinder. (Delete the existing cube.) To widen the top of the cylinder, switch to Edit Mode select the top points, press **S**, and move the mouse to adjust the scale. Add a Loop Cut in the middle of the shape with **CTRL + R**, then move it up a little and

scale it outwards after pressing **S**. You can then create a lower Loop Cut in similar fashion. You should now have a rough bowl shape. You can smooth the edges a little by selecting all (press **A**) and using the Bevel tool.



To make the Blender shape hollow, create a duplicate, scale that down slightly, and perform a Boolean Difference operation to cut it from the original shape

You now have a solid 3D shape, but need to make it hollow to make a usable mould. The easiest method is to Duplicate (**SHIFT + D**) the shape in Object Mode, move the duplicate straight up (holding **Z**) a little so it protrudes above the original's top, then scale the duplicate (Cylinder 001) down a little. Now select the original shape (Cylinder), click on Modifiers (wrench icon in the right-hand toolbar), and select Add Modifier > Generate > Boolean. With the default Difference option selected, click in the Object field and choose Cylinder 001, then click the down arrow icon and Apply. Delete Cylinder 001 and you'll be left with a hollow bowl shape. If you want to stretch it into an ellipsoid, select the shape in Object Mode, then click on Item in the sidebar and increase the X or Y value.

Before exporting the shape as an STL file for 3D printing, you should note that metres (m) in Blender equate to millimetres (mm) in the STL. So, adjust the X, Y, and Z dimensions accordingly in Blender. Finally, select File > Export > STL and click on Export STL.

To create more complex shapes, you'll need to hone your Blender skills, but you can access some extra preset shapes by installing the Extra Mesh Objects add-on (rpimag.co/blenderextraobjects). For instance, for a rounded bar mould, select Add > Mesh > Roundcube, then change Operator Preset to Clay Bar. Adjust the X, Y, Z dimensions to your preference, then slice off the top part of the bar: in Edit mode, switch the camera to the X preset viewpoint, toggle X-ray view on (so you can select hidden points), then select the top half of the shape and right-click > Delete Vertices. To add thickness to the shape's walls, use the Solidify modifier: select Add Modifier > Generate > Solidify, adjusting the Thickness value before selecting Apply.

Shape some soap

With mould in hand, whether bought or 3D-printed, let's now make some bars. First of all, take a

look at the size of your mould(s) and judge how much soap you are going to need to fill it/them. We cut some slices from the base with a sharp knife and chopped them up into small pieces (around 300g in total, for filling two moulds) to aid the melting process, putting them in our heat-resistant jug.

Next, we boiled some water and placed it in a large bowl, and sat the jug in the water, obviously being careful not to touch the water ourselves. It can take a few minutes for the soap to melt, but give it time. Another option is to use a microwave to heat it, best done by using in 30-second bursts, but do make sure that your jug is microwaveable and won't be used for any culinary purposes afterwards. Be aware that the melted soap will be hot, so don't splash yourself. Also be aware that you don't want the soap to overheat, so ensure that it does not get above around 70°C. Stir it with a spatula to make sure the clumps of soap base are breaking down. Some suggest spraying the mixture, once melted, with some isopropyl alcohol (to help disperse any bubbles), but that is optional.

QUICK TIP

In the tutorial we used Blender to make a 3D model for a mould, but you could use other software such as FreeCAD.

More on moulds

Many of the soap moulds that are available to buy are made of silicone, as this makes it way easier to get the soap out of the mould once it is set – it is simply a question of bending the mould back and easing the soap out, as silicone is pliable, durable, and flexible.

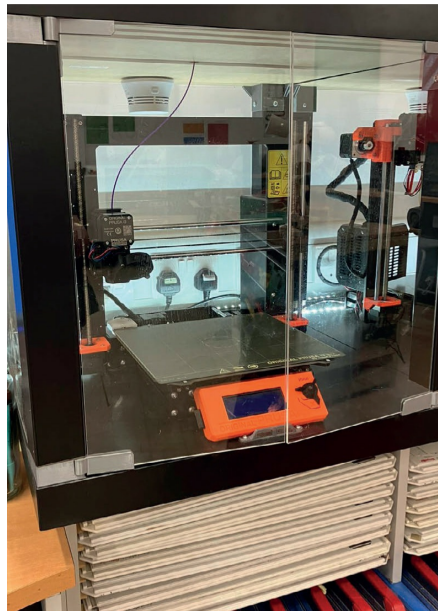
Our 3D-printed mould was made from a fairly standard, rigid filament. To get the soap out of the mould, we had to slightly warm it by floating it in some hot water for a couple of minutes – until the soap slid out. Another option is to place it in the freezer for 24 hours, and this should also help loosen the soap bar.

That said, you might want to consider what you make your soap mould out of before you even switch your 3D printer on. Here are some specific points to think about:

- Consider using a flexible filament (such as TPU – thermoplastic polyurethane) for your 3D print. These filament types can be more tricky to print with and stringing can occur, but if you can get over that hurdle, it's worth it to have a flexible mould at the end of the process.
- Think also about making your very own silicone mould by casting it inside a rigid 3D-printed mould. You would have to think very carefully about the design of the latter so that once you've poured liquid silicone into it, you achieve the shape you want.



Dried lavender tops off our finished soaps, also scented with lavender



The large 3D printer that was used to produce our mould. A huge thank-you to the Design and Technology department at Bournemouth School for Girls (this author's sister teaches there) who undertook the 3D print for us. If you don't own a 3D printer, see what facilities might be available to you locally, perhaps in a makerspace, or seek out local individuals who might offer a 3D printing service

QUICK TIP

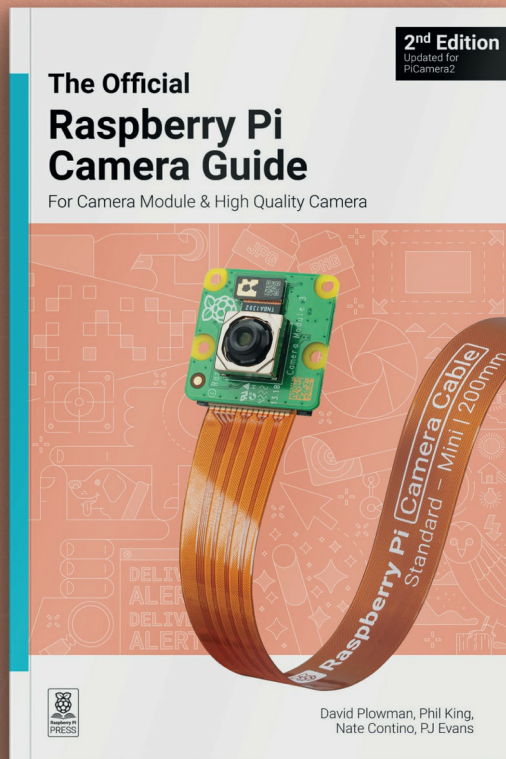
It's always a good idea to wear old clothing when making anything like this, especially if you use any dye... general mess is part of the process!

With your soap base melted, you are now ready to add any colouring or scent. We added some purple mica powder, but found we needed very little – the same applies to any essential oils (again lavender). Be sparing, and stir in until dispersed and the colour is even. We found that we had to work fast once the base had melted, as the mixture will start to thicken reasonably quickly. That said, you can always blast it in the microwave for 15 seconds to loosen it again. Then, you are ready to pour your mixture into the mould. We added some dried lavender in the base of each mould, slowly poured the mixture over the top to avoid bubbles, and left it to set for 24 hours, although it was properly set after about six hours. You can then remove it from the mould, lather up, and enjoy! 🟢

Other ideas

There are plenty of ways of making interesting and unique soap bars. Here are just a few suggestions:

- Lovely layers – different layers of soap together in one bar. Multicoloured, or maybe ombré, effects look great. Once the first layer has started to harden up with a 'skin' on top, you can add the next layer.
- Embeds – for example, make some soap letters (tip: a more shallow sugar fondant mould is great for this), then embed them in your layered soap bars.
- We added some dried lavender to our soap bars, but you can add plenty of other things too in the base of your moulds – dried flowers, oats, dried citrus slices, poppy seeds, coffee beans, seaweed, grains, raw honey, Epsom salts, etc. Some of these additions have exfoliating properties, and some just look and smell beautiful.
- Add relief designs to a finished bar of soap, embedding a clear stamp in the soap before it hardens, then carefully removing it once set. You could also make holes in bars of soap and feed them through some natural hemp so they can be hung up.



Add the power of HDR photography, Full HD video, and AI image recognition to your Raspberry Pi projects with Camera Modules.

- *Getting started*
- *Capturing photos and videos*
- *Control the camera with precision*
- *Add artificial intelligence with the AI Kit*
- *Time-lapse photography*
- *Selfies and stop-motion video*
- *Build a bird box camera*
- *Live-stream video and stills*
- *...and much more!*

BUY ONLINE: rpimag.co/cameraguide

Origami paper circuits

From a flat square to a finished, illuminated sculpture... let's return to the fold



Maker

Nicola King

Nicola is a freelance writer and sub-editor. There's a good reason why so many ancient and heritage crafts are still popular today, and she's all in!



@holtonhandmade

QUICK TIP

You can use all sorts of paper for origami – the key characteristics needed though are suppleness, an ability to hold a fold well, and some degree of strength. Wrapping paper works a treat.

Origami is an ancient Japanese art form, and the word itself stems from two words – ‘ori’ (from the root verb ‘oru’) means ‘to fold’ and ‘kami’ is one of the Japanese words for paper. The aim of the craft is to create something very beautiful from a flat two-dimensional sheet of paper. Equipped with knowledge of just a few basic folds, you can create some fairly intricate designs.

If you are looking for an inexpensive hobby, this one surely hits that mark, as all you really need is some paper and a scoring and folding tool. In this tutorial, we are going to make some simple origami shapes. We are then going to use a Raspberry Pi Pico and LED to shine a light on our creations – blending an ancient craft with the very modern, if you will. Great for hand-eye coordination and mental concentration, this mindful art is an excellent way to boost your creativity. At the end of the origami process, we'll have exactly what we started with, i.e. a piece of paper, but we'll have transformed it into something a little more interesting. We'll then attach it to a mount and add the circuitry.

Tiptoe through the tulips

We are keeping this simple – we’re new to origami, we want to start with the basics and work our way up, and we only have a few pages in which to cover the entire topic. Our aim is to open the door of the origami rabbit hole that you can delve more deeply into. Plus, we’ll be spending some time lighting up our creation too. So, a simple tulip shape is the goal, and it should take you a few seconds to make each one.

Before we get underway, it’s worth emphasising a couple of points. Firstly, the key to producing a high-quality piece of origami is in the quality of folding itself, so don’t rush it and pay attention to accuracy. Use a scoring tool with a ruler if you want really crisp, precise folds, especially if your paper is a bit thicker. Secondly, be aware which side of the paper should be facing you as you fold – if you are using origami paper, as we are, one side is coloured and one side is white, so you need to ensure that you have the correct side facing you, per the instructions you’re following.

YOU’LL NEED

- Colourful paper to fold (perfect squares in shape)
- Scoring tool and ruler (optional)
- Tweezers
- Folding bone
- Scissors
- A4 piece of cardboard/ mountboard
- Blu tack/other adhesive putty
- RGB LED
- Raspberry Pi Pico
- Resistors
- Jumper wires
- Power source



Our very simple origami make with some Raspberry Pi thrown in for good measure. Once you've learnt the basic folds, you'll find that much more complex designs are within your reach, and challenging yourself is the best way to improve

We're using an RGB LED controlled by a Raspberry Pi Pico to create a glowing effect that cycles through all the colours of the rainbow

Origami origins

Origami's main requirement in terms of tools is, obviously, paper. It's thought that the first paper was made under the Chinese Han dynasty way back in 105 CE. Japan was not far behind in its move into paper-making in around the 6th century, and that country's love of paper since then has been enduring. It's hardly surprising then that the art of folding paper appears to have originated in one of these two countries; no one is completely sure who can claim that title.

Initially, folding paper was used for ceremonial purposes as paper was something of a luxury, but it's believed that from around 1600 onwards, the art of paper folding had become much more recreational as mass-produced paper became more affordable to the everyday person. People have been folding paper for centuries, giving each other paper-folded gifts to cement friendships.

Originally known as orikata (paper folding), the descriptive term that we now recognise, origami, started to be used much more around the late 1800s. It's now popular the world over. As well as being a meditative hobby, origami is used by teachers to teach geometry (e.g. folding used to illustrate symmetry and fractional concepts), spatial relationships, design principles, and problem-solving skills, as origami is rooted in mathematical principles.

We're using origami papers that are 15cm by 15cm, so completely square in shape. With the white/wrong side facing you, fold the paper diagonally in half to make a triangle – this is called a valley fold (an inward pointing line), and is the most basic origami technique. Then fold that triangle in half, and open the fold just made, ensuring that the folded line is facing upwards. Next, take the points on the base of the triangle upwards, as shown in [Figure 1](#). Turn the flower over, and then fold the sides as in [Figure 2](#). Turn the flower over again, and you have the head of your tulip.

QUICK TIP

Take care when using any tools; paper is delicate, and a gentle touch will always be the best policy.

Stalk talk

The stem of the flower would be perfect in a green colour, if you have it. Wrong/white side facing you, fold the paper diagonally again as before. Then open it and fold as per [Figure 3](#). Turn it up the other way and then fold both sides per [Figure 4](#), so you have a kind of kite shape. Then fold both sides in again, per [Figure 5](#). Turn the stem over and fold the pointed end up to the top. Then fold in half, per [Figure 6](#). Pinch the pointed end and pull it out slightly, and turn the outer layer outwards so it resembles a stem.

Brighten your bloom

The final step is the circuitry part, which is actually quite straightforward. We estimated where our tulip would sit on an A4 piece of cardboard, and made a small hole with some scissors that an LED (which will illuminate the head of the flower) could be poked through.

QUICK TIP

When you are making very small origami shapes, you can also use a small pointed stick, like a cocktail stick, to help gently push the paper into the shape you want.

In this example, we're using an RGB LED controlled by a Raspberry Pi Pico to create a glowing effect that cycles through all the colours of the rainbow using PWM (pulse-width modulation). First, wire up the circuit on a breadboard as shown in [Figure 7](#). Our RGB LED is a common cathode type with one cathode leg to connect to a GND pin on Pico. The other three legs are for red, green, and blue elements of the LED. The order is typically red, cathode (longer leg), green, blue – if yours differs, adjust the connections accordingly. We connected our red, green, and blue legs via resistors (we used 150Ω ones) to the GPIO 13, 14, and 15 pins respectively.



You don't need much to begin origami

Crazy for cranes

If you are at all familiar with origami shapes, the classic crane shape is probably a design that you will have heard of, as it is possibly the most famous and loved of all origami designs, and is famous throughout the world.

The crane, or 'orizuru' to give it its correct term, is a symbol of peace, hope, and good fortune, so if someone gives you one as a gift, that's a thoughtful gesture that should be long appreciated. The crane itself is a bird that is seen as mystical and is held in high regard in Japan.

There was even a Japanese book published in 1797 called *Hiden Senbazuru Orikata* which described instructions on how to fold 1000 cranes, and is thought to be one of the first books on origami. There's a popular legend that if someone folds that number of cranes, then they will be granted one wish, but over time the making of 1000 cranes is now entwined with the aims of general happiness, good luck in personal pursuits, and so on.

If you'd like to make one yourself, this link provides some excellent instructions, and in around 15 minutes you should have your own crane to admire: origami.me/crane.

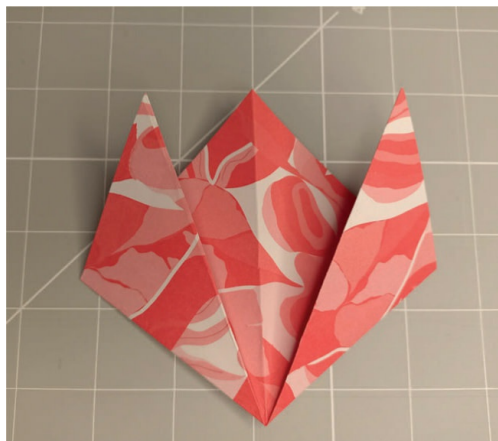


Figure 1: Once you've made a few of these flowers, muscle memory will kick in and they will take you seconds to make

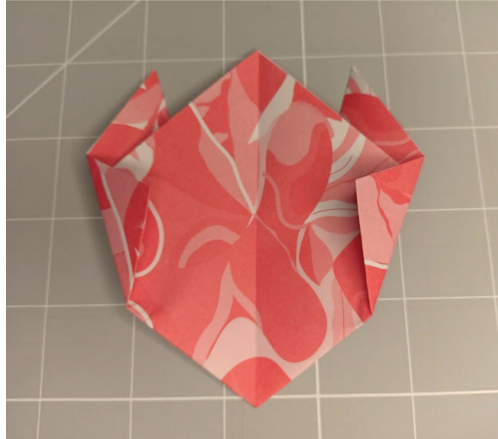


Figure 2: Making origami shapes is a great way of using up scrap paper that's lying around, as well as being a relaxing pastime

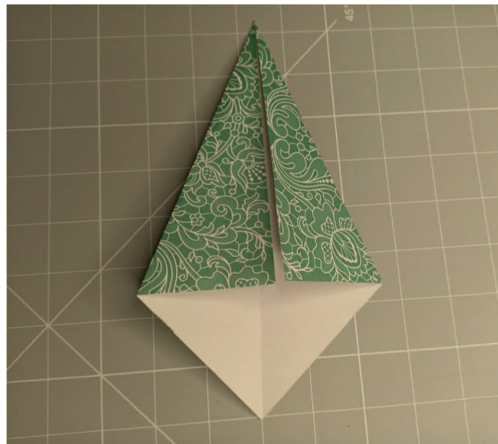


Figure 3: Starting the stem. If you have a hole in base of your flower, you can just feed the top of the stem into it

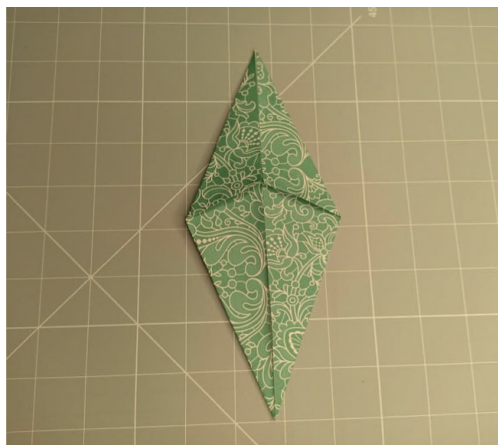


Figure 4: It's very important to make folds absolutely precise, otherwise the rest of the process ends up askew and errors multiply

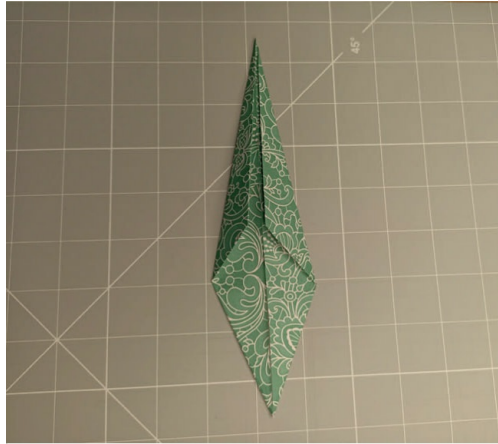


Figure 5: Sounds obvious, but make sure your hands are clean before you start so that you don't taint your piece in any way

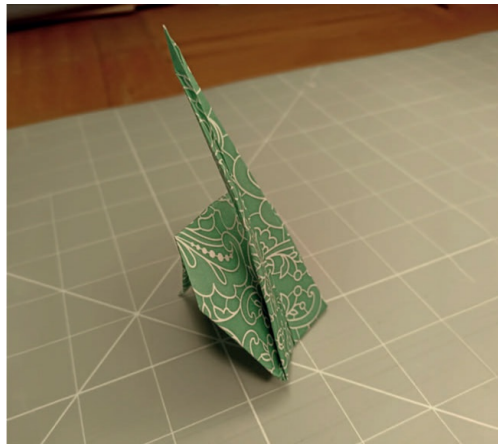
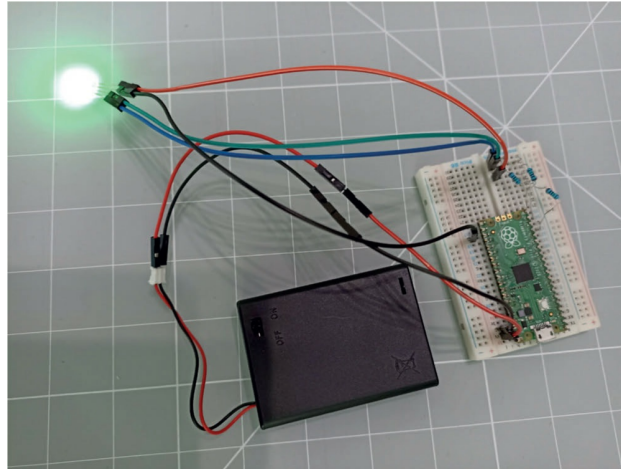


Figure 6: You can make a soft crease before you commit to any origami fold, so if you are slightly off you can correct it more easily

We made a small hole with some scissors that an LED (which will illuminate the head of the flower) could be poked through

QUICK TIP

Do make sure that you are folding your paper on a completely flat surface, as any ridges or dents will impact the aesthetic of your final piece.



The electronic circuit, using Pico. As shown here, you can take the RGB LED off the breadboard to use it in the origami flower

You can check everything is connected correctly by first cycling through red, green, and blue separately. We're using the picozero MicroPython library to make it simpler. Connect Pico to a computer and write the following code using Thonny (or another IDE):

```
from picozero import RGBLED
from time import sleep

rgb = RGBLED(red = 13, green = 14, blue = 15)

while True:
    rgb.color = (255, 0, 0)
    sleep(0.5)
    rgb.color = (0, 255, 0)
    sleep(0.5)
    rgb.color = (0, 0, 255)
    sleep(0.5)
```

Run the code and the RGB LED should light red, green, and blue in sequence, with an interval of half a second. The code for gradually cycling through the whole range of colours – between red and green, green and blue, then blue and red – is even simpler, thanks to picozero's built-in cycle method:

Technology and origami: a complementary relationship

A number of very clever people are folding paper and inserting some tech into their work at the same time, and are even using origami to help engineer other structures, using plenty of mathematics and experimentation. If you are looking for some inspiration, these projects are sure to make you fans of folding:

Gain some inspiration from Dr Matthew Gardiner (matthewgardiner.net), who is an artist well known for his work with origami and robotics. In fact, back in 2003, he coined the term 'Oribot', and his website is a fascinating treasure trove of images of his work, videos, kits, and so on.

Ross Symons is the origami artist behind White on Rice (white-onrice.com/tutorials), a website where you can find some great tutorials on how to fold origami-style. In particular, we'd draw your attention to his Origami

Meditation Mural, a clever piece of interactive wall art made of 64 origami flowers. The really smart bit though is that when you press some of the flowers, a sound is played. He used Raspberry Pi (of course!), a Pi Cap add-on board, and some electric paint and copper tape in its construction. Capacitive sensing is at the heart of this super project: youtu.be/zUX7ya_oAgw.

Origami engineering – yes, it's a thing! Engineers are using the principles of origami to help solve engineering challenges. Origami is a useful tool because the goal is it to take a flat sheet of paper and transform it through folding and sculpting. You can apply these conventions to any flat material. This form of engineering allows objects to fold compactly, while the structural integrity is maintained. Examples would include collapsible packaging for the benefits of recycling, implements involved in medical diagnostics, or even origami-inspired robot plants that can monitor environments, to name a few. This interesting video provides some enlightening background to the topic: youtu.be/ThwuT3_AG6w.

NASA even has a page on its website which delves into 'Space Origami', and shows you how to make your own Starshade model, a NASA device that will be used to shield telescopes' cameras from the bright light of stars, and will fold out to its full size in space: rpimag.co/starshade.

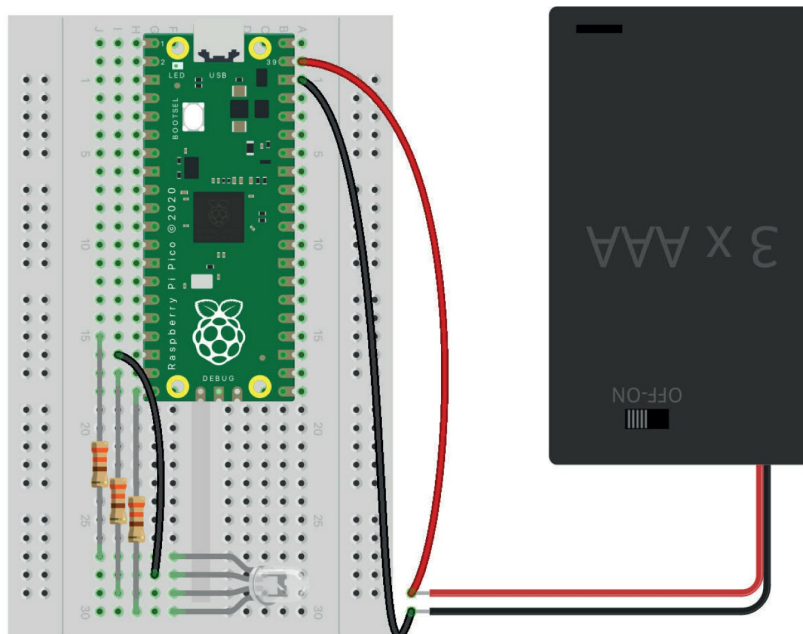


Figure 7: The wiring diagram for connecting Pico to the RGB LED on a breadboard

```
from piczero import RGBLED

rgb = RGBLED(red = 13, green = 14, blue = 15)
rgb.cycle()
```

If you want the colours to cycle more slowly or in a different order, you can add parameters inside the brackets of cycle(). For instance:

```
rgb.cycle(fade_times=3, colors=((0, 0, 255), (0, 255, 0), (255, 0, 0)))
```

This example will increase the fade time between main colours from 1 to 3 seconds and go in the order blue, green, red.

Save your colour-cycling code as main.py

Save your colour-cycling code as **main.py** on Raspberry Pi Pico and it will then run automatically when connected to a power source. We used a 3 × AA battery pack connected to Pico's VSYS and GND pins; alternatively, just use a USB power bank.

Now take your RGB LED off the breadboard and push its legs, toward the rear, through the small hole you made in the cardboard. Connect long jumper wires from its legs back to the holes in the breadboard where they were previously inserted.

Turn on the power to Pico and the RGB LED should create a colour-cycling glow to illuminate your origami flower and the card behind it. For best effect, try it in a dark place for an impressive glowing flower. 🌸

Sewing with LEDs and Raspberry Pi Pico

Why should a handy storage pouch be boring?
Bling it up with some light-emitting silicon



Maker

Nicola King

Nicola is a freelance writer and sub-editor. Sewing with a little bit of tech thrown in is hitting the spot this month.



@holtonhandmade

QUICK TIP

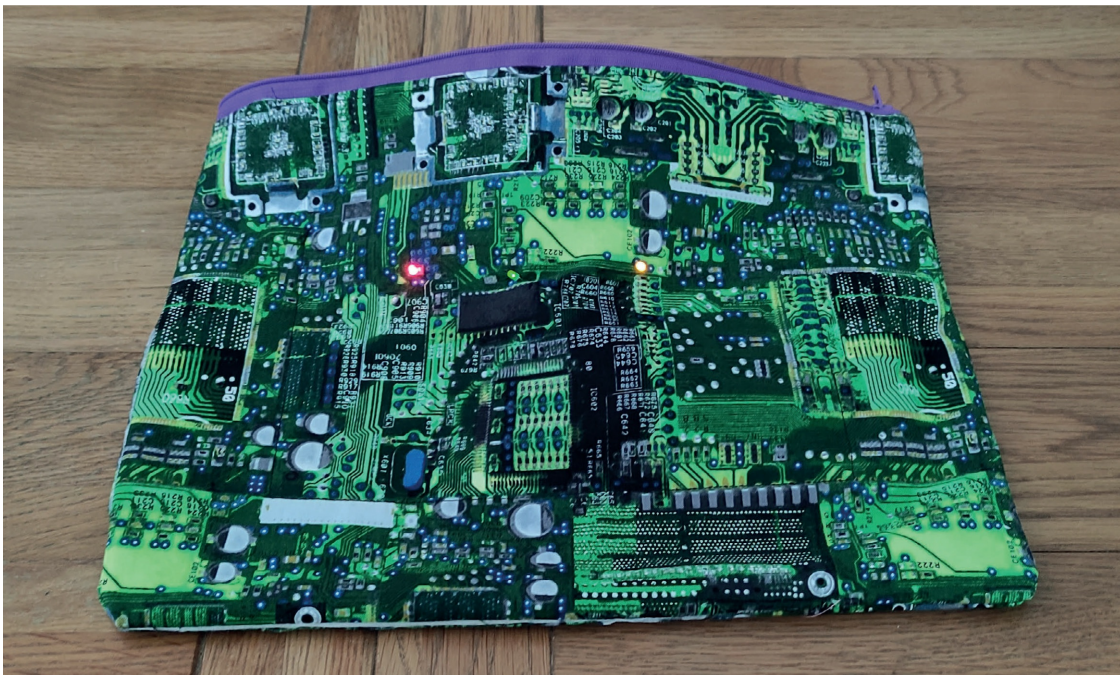
Felt would be another great base fabric to use if you are a beginner – it is stable, very easy to sew through, won't fray, and will easily hold the weight of components.

Combining LEDs into a sewn project is easy to do, and we're not just referring to wearable tech here. You can also craft items that you can actually utilise for a functional purpose and, at the same time, add a little Raspberry Pi and LED luminescence to your design. So, this tutorial takes a look at how to add some LEDs to a simple storage pouch that we are also going to make. In fact, these zipper pouches – which can be made in absolutely any size you prefer – are so uncomplicated to make that we've whipped up quite a number in order to house various components, plus our general odds and ends that need a home.

Sew straightforward

We decided that we wanted a decent-sized pouch, so, we first cut out two rectangles of fabric 25cm by 35cm in our main fabric. Now, a quick word on your choice of base fabric before we move on: your base is important, and we've used cotton backed with a medium-weight fusible interfacing to give it some structure. Cotton is generally a great choice because it is lightweight, flexible, and highly suitable for sewing with electrical components. If your fabric is difficult to sew through, you will have issues, so give this some thought. If you have some lightweight denim on hand, some calico, or even faux leather, they might also be good alternatives.

If you want to add a backing to strengthen your fabric, cut out two pieces in a fusible interfacing the same size and adhere with an iron. Then cut out a 'pocket' in a different-coloured fabric – we cut a rectangle 10cm by 15cm – and back that with interfacing too. The reason we are adding a pocket is so that all of our Pico/LED workings can sit inside it comfortably and separately from whatever else we choose to keep in the pouch. To attach the pocket, fold in the bottom and the two sides of the small rectangle by about 0.5cm, and press, then pin the pocket to the wrong side of one of your pouch pieces. Use your sewing machine to sew the pocket to the main fabric, keeping a consistent seam – 0.5cm works well. As our fabric was dark in colour, the dark thread that we used to sew the pocket blended well into the fabric and is barely visible on the right side of the pouch.



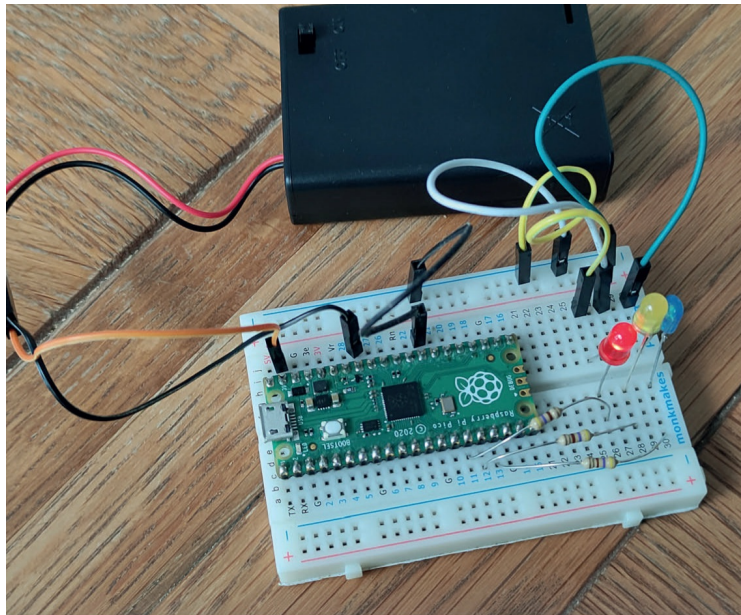
We had some 'circuit' fabric in our stash that seemed highly appropriate here (rpimag.co/frumblecircuit). Just some fabric and a zip, along with our components, meant this was a very low-cost project. Stashbust and see what you have!

Cotton is lightweight, flexible, and highly suitable for sewing with electrical components

Next, take your zip and place it along the top side of one side of pouch, right sides facing, and pin it. Then sew it with a zipper foot on your sewing machine if you have one. Attach the other side of the pouch to the other side of the zip and sew. Then press the seams out that you have just sewn and top-sew along the length of the zip on the right-side of the fabric to give a neat finish. Next, open the zip halfway, then turn right sides of pouch to right sides (so the inside of the pouch is on the outside) and pin in place. Sew around all three pinned sides in one go, carefully removing pins as you sew.

YOU'LL NEED

- Cotton fabric (¼ metre)
- Fusible interfacing (optional)
- Standard cotton thread
- A zip (cut to length required)
- Sewing machine
- Pinking shears (optional)
- Scissors/sewing pins/iron/tape measure
- LEDs (single colour) × 3
- Jumper wires × 6
- Resistors × 3
- Power source
- Raspberry Pi Pico
- Needle-nose pliers (optional)



Testing the LED circuit and program on a breadboard before wiring everything directly to Raspberry Pi Pico

Power sources

A number of options exist for powering your sewn circuits:

- Coin cell batteries – sewable coin cell holders can be integrated into projects, creating a circuit with some easy-sew LEDs and conductive thread. These batteries are usually around 3V. All of the components are small and inconspicuous while still giving you a great LED effect.

- Battery packs – as we've used in this tutorial, powering Raspberry Pi Pico and the LEDs simply with three AA batteries.
- For a more detailed look at power sources, see the tutorial published in issue 76 of HackSpace magazine: rpimag.co/hs76.

Learn more....

The e-textiles (electronic textiles) field is pretty vast, and there are plenty of sources of inspiration. Here are a few ideas to take you further:

- *Wearable Tech Projects*, a HackSpace publication written by Sophy Wong, was published in 2019, but is still hugely relevant and endlessly inspiring. If you own a copy already, dig it out. If not, you can download the free PDF here: rpimag.co/wearableprojects.
- You can find out more about the LilyPad system, a set of sewable electronic pieces designed by Leah Buechley, and sewable electronics, at sparkfun.com/about-lilypad. Leah also has a book on the subject, called *Sew Electric*, which is worth a flick through – sewelectric.org.
- Interesting work is going on in the subject area of e-textiles. Take a look at this research which is concentrating on making e-textiles sustainable and biodegradable – rpimag.co/textilestech.

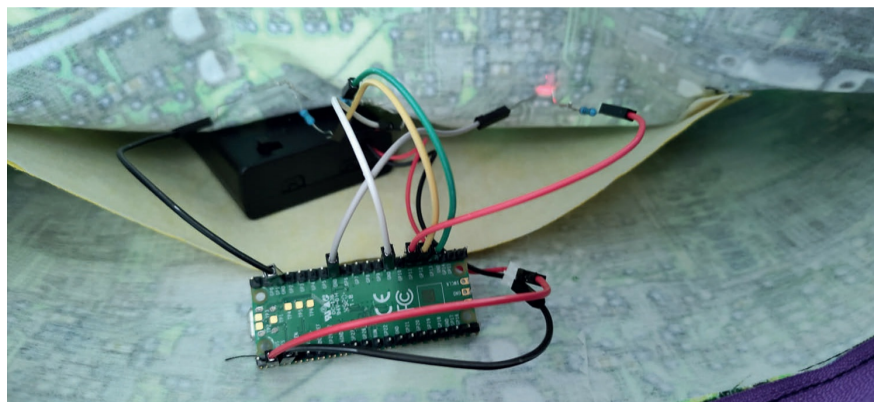
Finally, trim loose threads and any excess zip fabric, and use pinking shears to cut away excess from the seams and to ensure that fraying is kept to a minimum. As you opened the zip halfway, you should now be able to turn the pouch so that the wrong side is now inside and then press it out with an iron – one storage pouch ready for use. Note that we have purposely created a very simple design here for the purposes of speed and ease, so we can now concentrate on the electronics. If you have the time, you can line the pouch with another fabric to make the inside tidier.

QUICK TIP

One way of securing standard LEDs to fabric is through a button – you'll need a two-holed button where the width between the holes is that same as the width between the LED legs. Poke the legs through and perhaps secure the legs with a few stitches using standard thread.



Make sure that you press your sewn seams as you go, and your finished product will look much neater and crisper. Make the pouch as large as your want... in fact, the larger the better if you are looking to store cables etc. We've found that A3 or A2 sizes are perfect, but you will need slightly more fabric



Raspberry Pi Pico, the battery pack, and wiring all fit in an interior pocket to keep them separate from the main bag contents

QUICK TIP

If you use standard single-colour LEDs in your circuit, as we have, ensure you mark the legs so you know which is positive and negative before you flatten/twist them.

Pico programming

For this simple example project, we're just going to light three LEDs in sequence. So, in essence, we just need a program to turn the GPIO pin for each LED on (and off) in turn.

We're using MicroPython for our program – flash it to Raspberry Pi Pico by downloading the latest UF2 file from rpimag.co/picomicropython, making sure it's the correct one for your particular Pico model. Then, while holding its BOOTSEL button, connect Pico to a computer via

USB to mount Pico as a drive. Now drag the UF2 file onto the Pico drive to install it.

We're using the Thonny IDE (thonny.org) to program Pico. In Thonny, click 'Local Python 3' at the bottom right of the window, then select 'MicroPython (Raspberry Pi Pico)'.

We start the program by importing two modules we need.

```
import machine, time
```

We define variables for our three LEDs, assigned to the GPIO pins to which they are connected, as an output.

```
led1 = machine.Pin(11, machine.Pin.OUT)
led2 = machine.Pin(12, machine.Pin.OUT)
led3 = machine.Pin(13, machine.Pin.OUT)
```

We then create a list of these:

```
leds = [led1, led2, led3]
```

Finally, we create an infinitely repeating `while` loop with a `for` loop nested within it. The latter takes a value from the `leds` list in turn, starting with `led1`, and turns the respective LED on with `led.value(1)`. We set a `time.sleep` delay for how many seconds it will stay on, then turn it off with `led.value(0)`.

```
while True:
    for led in leds:
        led.value(1) # Turn LED on
        time.sleep(0.5) # Delay for 0.5 seconds
        led.value(0) # Turn LED off
```

With the LEDs connected on a breadboard, as in the [Figure 1](#) wiring diagram, try running the program to see them light up in sequence. To make the program run automatically whenever Pico is connected to a power supply, save it as **main.py** (selecting Raspberry Pi Pico as the location).

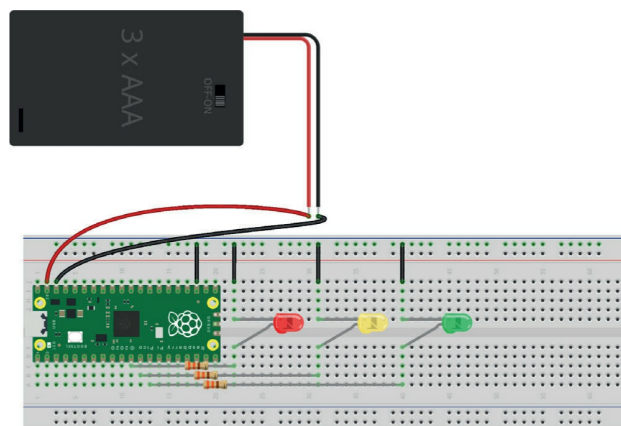
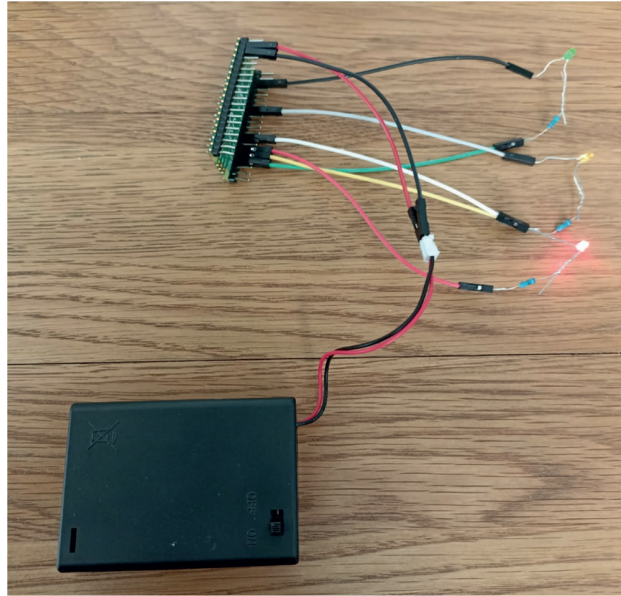


Figure 1: The wiring diagram for testing – once you've got it all working, you can dispense with the breadboard and connect everything directly to Pico's pins



The raw electronics, out of the bag. We used 10cm jumper wires to connect everything directly to Pico's pins

Once everything is working as expected, you can reconstruct the circuit without the breadboard – after positioning the LEDs where you want in the fabric, poking their legs through it towards the inside of the pocket. (Cotton fabric is very easy to poke the legs through.) Coiling the end of a resistor around the positive leg of an LED should hold it in place, or you may opt to solder it for a more secure connection. We plugged the other end of the resistor into a socket-to-socket jumper going to the respective GPIO pin on Pico; ditto for the negative leg of the LED, connected via a jumper to a GND pin. If your Pico doesn't have pin headers attached, you could just solder the wires directly to the pin holes.

You can easily add extra LEDs to your circuit (and program), but may need to join some ground wires together to connect them all to Pico's GND pins if you run out of the latter. You could also create alternative lighting programs and perhaps fade LEDs in and out using PWM to adjust the brightness.

We're powering our circuit from a 3 × AA battery pack, which supplies around 4.5V, connected directly to Pico's V_{SY}S and GND pins. Pico is fairly voltage tolerant, so this should work fine. If using a Raspberry Pi Zero computer instead, you may well want to use a mini voltage regulator board.

A twinkling tote

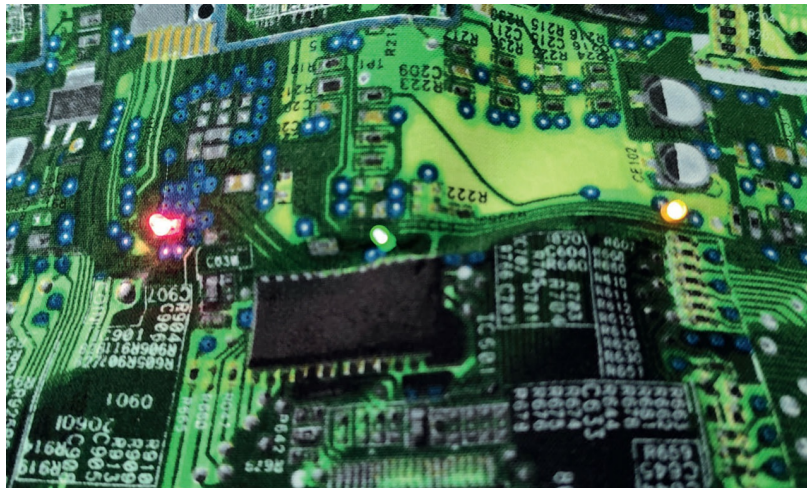
The aim of this tutorial has been to offer up a simple idea on how to combine some crafty sewing and LEDs in one project. There are many ways of doing this, so just see this as a starting point and do please take it further. Make a larger tote bag, introduce sewable LEDs such as a LilyPad and maybe some conductive thread, take inspiration from our 'Learn more' box, and create your own sewn and Raspberry Pi or Pico combo project. 🟢

Light-emitting diodes

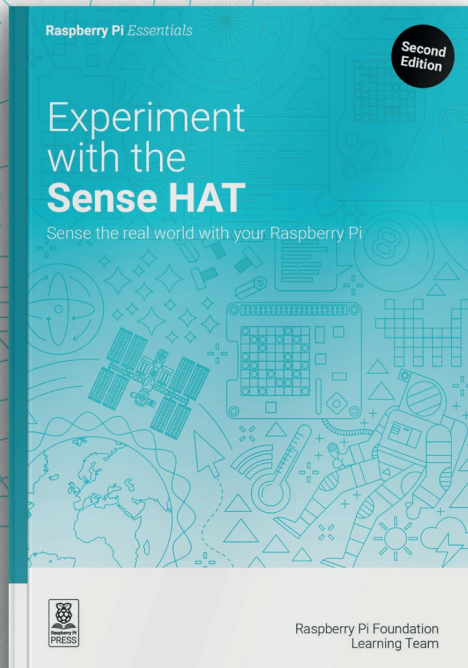
There are a number of options when it comes to choosing which LEDs to use for your project. Let's take a brief look at a few different types which might be suitable here:

- **Sewable LED modules** – these are typically very slim, pre-mounted LEDs designed specifically for sewing purposes, as they usually have a hole in either side that you can get a needle and conductive thread through in order to attach them to a fabric.
- **NeoPixel LEDs** – these are individually addressable, so can be chained together and controlled from a single pin on Raspberry Pi Pico or a Raspberry Pi computer.
- **Single-colour, standard LEDs** – these might be clear, and the light that they generate is often directed and focused. Or, they might be diffused LEDs – the light here is often slightly softer or 'milkier' in nature, and these give a more even and consistent light. Single-colour LEDs usually will have two legs (a long one for the anode or positive, and a short one for the cathode or negative). You can often buy 'super-bright' or 'ultra-bright' versions too, depending on what you are aiming for in terms of brightness levels.
- **RGB LEDs** – these combine three tiny LEDs: red, green, and blue. They usually have four legs (one for each colour, and a shared cathode/ anode leg). Essentially, by varying the brightness of each LED element, a wide spectrum of colours can be generated.

You can also purchase addressable flexible strips of LEDs, which are very useful if you want fine control of a lot of LEDs – you can cut the strip to a particular length to suit your needs. Super-flexible 'noodle' filaments are also available, as are strips with power switches, strips with light sensors, and more. Your choice of which to purchase ultimately depends on what you are making, your power source, and the light effect you want to achieve.



A close-up of the LEDs; we just poked their legs through the fabric, then connected them to resistors and Pico on the inside



The Sense HAT is an incredibly versatile and flexible bit of kit with plenty of obvious uses, along with a huge number of less obvious ones, that you'll love to make and share. Updated for the latest Raspberry Pi devices and hardware, this book has everything you need to get started.

■ **Getting started with Sense HAT**

■ **Learn by building:**

- *A digital twist on the Magic 8 Ball*
- *Your own interactive pixel pet*
- *A sparkly light show*
- *An environmental data logger*
- *Flappy Astronaut, a low-res, high-fun video game*

BUY ONLINE: rpimag.co/sensehatbook

Make and publish your own zine

Get into on- and offline DIY publishing with Inkscape



Maker

K.G. Orphanides

K.G. has been making digital and print zines for fun and (remarkably little) profit since the 1990s.



twoot.space/ @owlbear

Zines are DIY magazines that you can make yourself, or with friends. They can be anything from a single folded page of A4, to a photocopied magazine, or an ornate hand-constructed work of papercraft art. We used Inkscape to create our zine, and it's our preferred choice. There are a few different ways to get hold of this popular open-source vector illustration program, which tidily serves double duty as a flexible DTP program, but only the Flatpak version is currently updated to the latest version 1.4.2, which we've been using and has all the features and templates we need to make this task easy.

Installing Inkscape

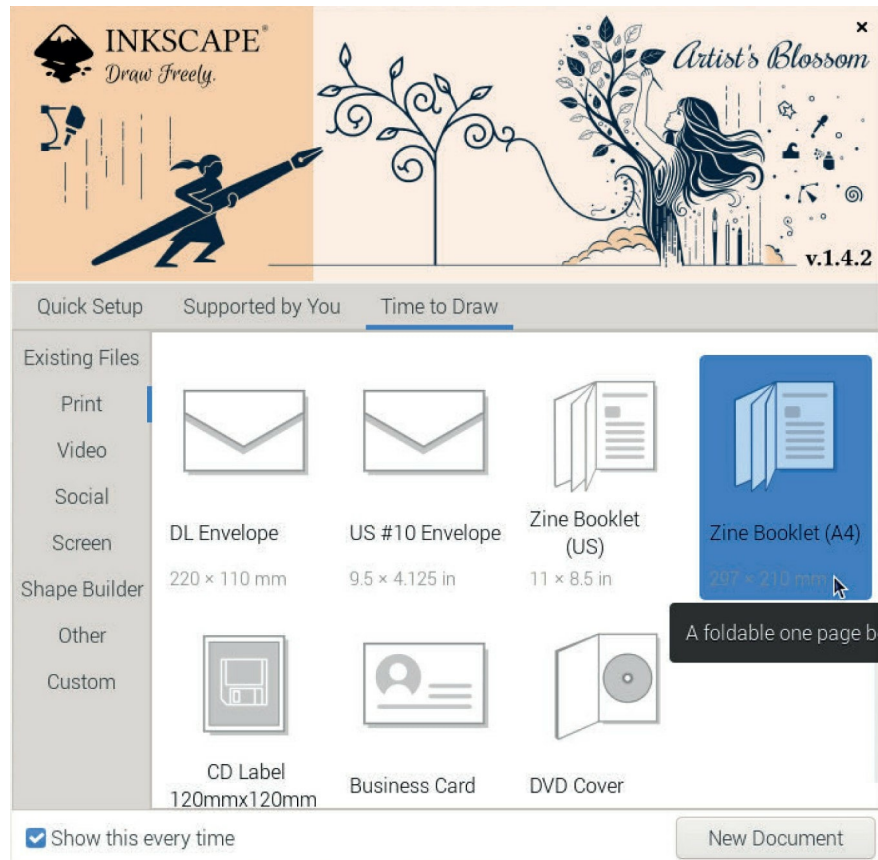
We prefer to install Inkscape from the Flathub repository ([rpimag.co/flathubinkscape](https://flathub.org/repo/flathub)), which provides a convenient way to install the latest stable release and keep it up to date.

To install the Flatpak version, open a terminal and type:

```
$ flatpak remote-add --user --if-not-exists  
flathub https://flathub.org/repo/flathub.  
flatpakrepo  
$ flatpak install --user flathub org.inkscape.  
Inkscape
```

You can then run Inkscape directly from the Graphics section of Raspberry Pi OS's application menu, or from the terminal by typing:

```
$ flatpak run org.inkscape.Inkscape
```

Inkscape 1.4.2 has a built-in template for foldable mini zines



Warning!

Sharp blades

This project can involve the use of cutting blades or scissors. Always use sharps responsibly. Children should be supervised if blades are used.

Formatting your zine

A zine can look however you want it to, from a broadsheet, to chapbooks or classic folded mini zines.

If you're publishing for online reading, then the easiest way is by setting up A4 pages and exporting them as a PDF. Readers can then easily download it and flip through its pages using their favourite PDF or e-book reader, and you can upload it for easy online reading using an embedded reader with realistic page flip mechanics.

If you'd rather make a tiny zine, recent versions of Inkscape come with a template that makes it easier to produce a folded zine from a single sheet of A4, which you'll find under the Print section of the Time to Draw tab in Inkscape's startup interface. These look great, save

paper, and are great for giving to friends and putting out at public events.

We've distributed [zines] at art galleries, protests, pubs, on the Internet Archive...

Of course, the sky – and your imagination – is the limit, not only when it comes to form factor, but also design.

Zines are often as much about your DIY artistic instincts as they are about the content. Whether you're using clip art, making collages, or illustrating your own comic strips, you can give your zines tons of personality. Think about what you find engaging on the page, rather than making something look professional and polished. If unevenly hand-defined text flow boxes and wonky images and photos stacked on top of each other bring you joy, then there's no reason to seek 'professional' polish.

Layout and navigation

If you're making an A4 zine, particularly if you plan to distribute it on the web, then you'll appreciate one of Inkscape's most useful innovations: the multi-page SVG file. While you can only guarantee that current versions of Inkscape will handle these files without first exporting them to PDF, it's a great format to work with, allowing you to easily see, add, and flow content between multiple pages.

The page tool is the bottom icon on the left-hand toolbar. Select it, and another icon, a page with a plus sign on it, will appear above the left-hand toolbar. Click this to add pages to your document. To the right of that, you'll find options that allow you to define page sizes, add, delete, reorder, and extend a selection to include the entire content of pages. Clicking on a page will select it, and you can delete a page from the toolbar or by pressing **DELETE**.

You can move around the workspace, no matter how many pages you're working with, by holding **CTRL** and using the arrow keys, or by scrolling with the mouse wheel (with **SHIFT** to pan horizontally). Hold **CTRL** and scroll the mouse wheel to zoom in and out. There are multiple keyboard and mouse shortcuts for every tool in Inkscape, and you'll find them all listed at rpimag.co/inkskeys.

A critical feature of any DTP tool – and that's what we're using Inkscape as right now – is the ability to flow text between pages so it automatically adjusts when you edit your words. Make text flow into boxes by drawing one or more rectangles and pasting your text in a text box. Now use **CTRL** + left click to select all your rectangles and your text box, and either press **ALT+W** or go to the Text menu and select the 'Flow into frame' option. If you're working with text that you've previously flowed and want to put it into different frames, you may need to select it and use the 'Unflow' or 'Convert to text' options from the Text menu before reflowing it.

In smaller font sizes, called for in minizines in particular, you might find that there's too much space between each line of text. To change the line spacing, select the text tool (the icon in the left-hand toolbar that looks like a letter A followed by a caret), select the text block you need to adjust, and then look towards the centre of the tool ribbon above your editing area, where

you'll be able to select your font, its style, size, and then spacing between lines.

Free images

Creative Commons Media Search: rpimag.co/ccmedia

National Portrait Gallery: rpimag.co/npg

New Old Stock: nos.twinsnd.co

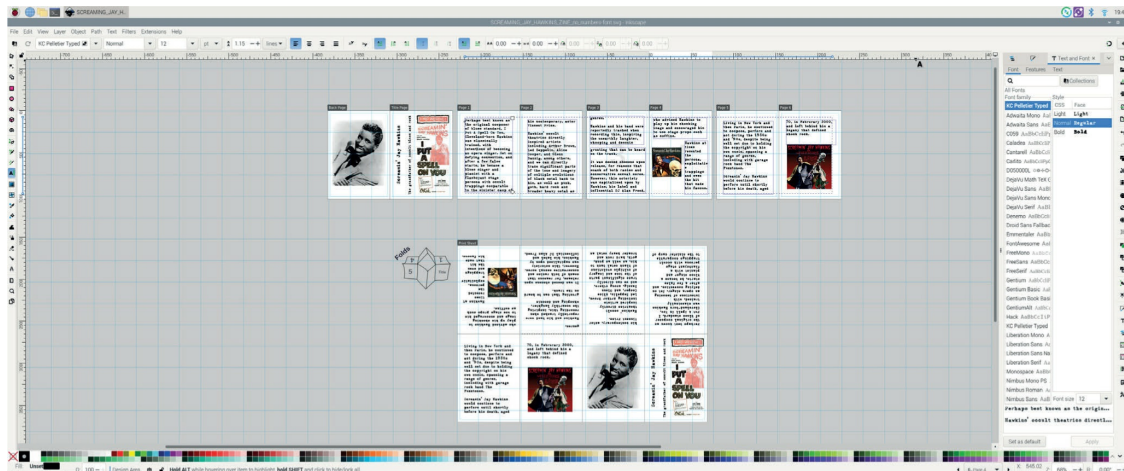
Vintage photos in the public domain.

Smithsonian Open Access: rpimag.co/smithsonian

Many other museums, state-funded scientific organisations and art galleries have released images under Creative Commons licenses.

Wikimedia Commons: rpimag.co/wikicommons

Understanding Creative Commons licences rpimag.co/ccllicences

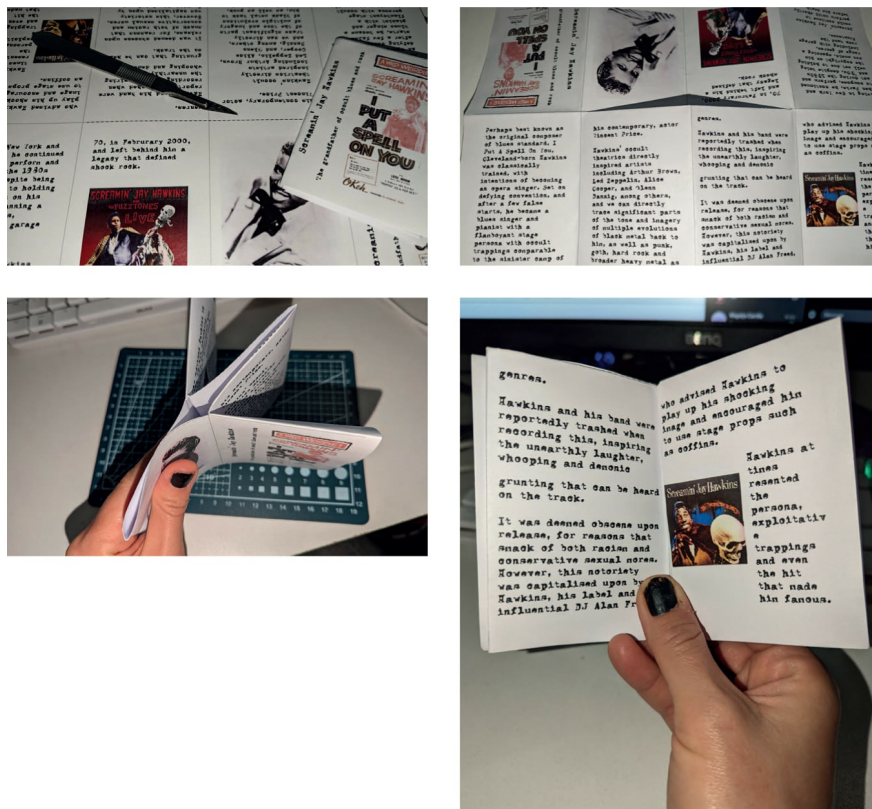


Inkscape was designed for vector graphics, but like Corel Draw before it, it makes a great desktop publishing tool

Once you've aligned your text where you want it, you'll probably wish to dispense with the coloured rectangles you used to lay it out. To render these invisible, select them. The easiest way to do this in a busy document is to go to the right-hand tool bar where you'll also find your text editing tools, select the Layers and Objects tab and **CTRL**-select all your rect objects. Now select the Fill and Stroke tab, and the Fill tab below it, and then click the 'No paint' option.

To add graphics, you can just drag PNG or JPEG images into your SVG document, and embed them, which means that they're imported from the external files. Keep the files together with your SVG to avoid losing them. You can also import your images into the SVG if you don't mind increasing its file size. You'll probably need to resize them: the easiest way to do this is to select the image, look at the tool ribbon above the working area, click the lock icon between the W and H measurements, switch their unit to percent, and adjust your image size down that way. You can also resize to millimetres or pixels if you need something more precise.

To export a multi-page A4 zine for easy sharing, save it as a PDF. If you're using Inkscape's one-page zine template, you should move its watermarks – the numbers indicating how you should fold it – to the 'Guides (do not print)' level of the Layers and Objects tab and use the Print (CTRL+P) dialogue to either send its Print Sheet to your printer or print to file to produce a PDF you can use elsewhere. Once printed, cut along the dotted line and fold it as shown.



Fold and cut and refold your mini zine per the template, and share it with your friends

Publish and be damned

Now that you've created your zine, where to publish it? We've distributed ours at art galleries, protests, pubs, on the Internet Archive (archive.org), and on strange little 2000s throwback web pages.

If you want to sell your zine, then Itch (itch.io) is your best bet. Creating an account to distribute zines (or games, music, or almost anything else) for free is quick and easy. If you want to accept donations or sell your zines, setting up the kind of account that can sell digital downloads – without having to handle your own VAT payments – isn't too much harder. Note that, due to a tightening of restrictions by payment processors Visa and Mastercard, Itch can no longer make payouts on some adult content.

If you want to sell anything on Itch, you'll have to provide a Taxpayer Identification Number (TIN) and fill out a US withholding tax form for sales to countries under their local VAT systems, such as VAT MOSS in the EU. The fees for this will be taken from your first payment, and the form will be processed when you request your first Itch payout, which means that this can take longer to be processed than subsequent payouts – we had to wait nearly a

month for our initial payout. 

Free fonts

Fontsource: fontsource.org

Kernclub: rpimag.co/kernclub

Rotating selection of fonts available for \$0.00

Uncut: uncut.wtf

Velvetyne: velvetyne.fr

Flower pressing

Press and preserve your blooms, measuring moisture with Raspberry Pi Pico



Maker

Nicola King

Nicola is a freelance writer and sub-editor. She's also the gardener in the family, trying to create a very tiny slice of Kew Gardens in deepest Dorset.



@holtonhandmade

QUICK TIP

When choosing flowers, make sure they are at their peak and close to full bloom, and ideally pick them in the morning on a dry day. Avoid any which have any blemishes or have started to wilt.

At the height of summer, our garden is bursting with flowers, and we sometimes wish we could preserve just a little bit of that colour and beauty to look back on later on. Aside from taking some snaps, there is a way... flower pressing or, more grandly, botanical preservation. This is another very traditional craft, but one that will always be relevant when we have beautiful flowers that we want to preserve, whether for sentimental reasons or because you want to create a record of what you've grown. It's also a very inexpensive pastime as you need very little aside from some flowers, from a garden or a bouquet, and something to press them with. In this tutorial, we are going to dabble in flower pressing, but we'll also engage a Raspberry Pi Pico and a sensor in our project while we are drying our blooms, to monitor the levels of moisture within the petals and leafage.

A moisture sensor can be placed onto petals to measure how damp they are

As to the type of flowers that you can press, some are more suitable or successful than others, and our research suggests that the optimal choices are often those that only have one layer of petals, as they are able to lay flatter in the press. Also, some flowers have more moisture within them, such as bluebells for example, so they will take much longer to dry out. A few flowers that pretty much guarantee pressing success for a beginner include daisies, pansies, cosmos, violas, asters, poppies, ferns, leaves and grasses, buttercups, single-petal roses, or snapdragons.



For pressing to be effective, you must have the right pressure placed against the entirety of the flower, but you also need to use effective paper within it to absorb the moisture. You can use absorbent kitchen towel, copier paper, or more expensive blotting paper. Avoid parchment paper or anything shiny, as it won't absorb the moisture and may cause mould to develop

YOU'LL NEED

- Fresh flowers of your choice
- Flower press (homemade or purchased)
- Absorbent paper
- Cardboard dividers
- Tweezers
- Moisture sensor (capacitive or resistive)

- Raspberry Pi Pico
- Jumper wires
- Breadboard (optional)

QUICK TIP

Pressed flowers are extremely delicate, so take care when you peel them from the paper, and plan your flowery designs before you remove them.



This is a tool that hasn't really changed much over the centuries. Vintage presses were actually quite small, were known as 'field presses', and were used by botanists and amateurs alike while out in the great outdoors. Bear in mind that if the wood panels are thicker, it will result in more weight and pressure and a more effective press

Freedom of the press

There are various ways that you can press flowers. We have chosen to use a wooden press, but the choice is yours:

1. A bookish approach

A tried and tested method of pressing flowers, this involves opening up a heavy book somewhere towards the back, then placing two sheets of blotting paper on top of the page, carefully setting down your flowers individually onto the paper face down, then placing two

more sheets of paper over the top. If you want to press other flowers, open another page and repeat these actions. You then close the book delicately, place it somewhere warm and well-ventilated, and maybe put a couple of other books on top to weight it down even further, but do try to make sure the weight is even. Generally, it's suggested that you leave these for two to four weeks to fully dry out.

2. Let's iron it out

Astonishingly, one way of pressing flowers that gives instant results is to use a clothes iron – placing fresh flowers between two sheets of baking or greaseproof paper, and pressing the iron on a low heat (with no steam) over the paper for around 10–15 seconds. Leave to cool, then repeat. The flowers may need an additional two or three days to fully dry out. Not a traditional approach, and results may be patchy as this method may remove some colour because the iron does not dry evenly, and you could turn the blooms brown if you aren't careful, but it's still worth trying if your options are limited.

QUICK TIP

We also read that some people use felt or cotton fabric in their presses – highly absorbent and you don't need to swap them out as often as paper.

3. Traditional press

From what we have read on the subject, traditional flower presses do work better when it comes to colour preservation and they result in less wrinkling, so that's what we opted for. We actually purchased our inexpensive (around £10) generic press some time ago and, unassembled, it consists of some squares of plywood with holes already drilled, bolts and wing nuts, blotting paper, and some cardboard – essentially basic ingredients that you are likely to have laying around already, so just upcycle whatever you have to hand. We do talk about creative approaches to flower presses in the second boxout, so take a look for inspiration.

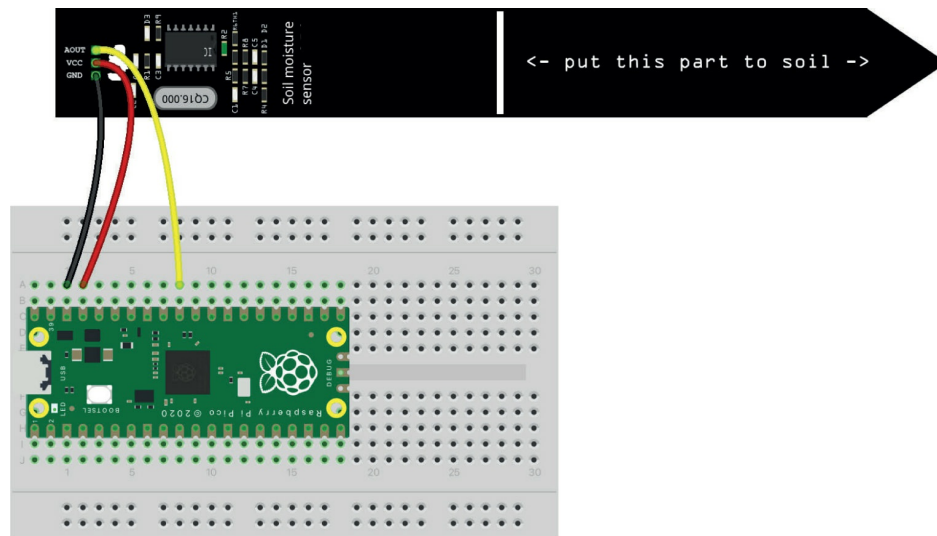


Figure 1: The wiring diagram for Pico and the capacitive moisture sensor

QUICK TIP

Bricks are a handy substitute for heavy books when you are weighting your pressed flowers down.

Nip it in the bud

So, with your chosen press to hand, the actual act of pressing the flowers could not be easier. We put our press together in a matter of minutes, and placed a layer of cardboard on the bottom, followed by a piece of blotting paper. Carefully placing the flowers that we'd picked early in the morning on that, we then laid a second piece of paper over them and another cardboard divider. We completed further layers, placed the top plywood layer on, and tightened the wing nuts. It's important to note that the paper sheets will need to be changed every couple of weeks to stop the flowers turning brown.

Moisture management

As the craft of pressing flowers is essentially very simple, we thought it would be interesting to see how we could use Raspberry Pi Pico to improve our flower-pressing experience. As the key to successful flower pressing is the elimination of any moisture within the flower leaves and stem, that seemed to be where Pico could assist us.

Pressed for time

As with many of the crafts that this author has written about, flower pressing is an old one... a very old one. In fact, it's something that people have been doing for thousands of years, and was probably a pastime that the ancient Greeks and Egyptians dabbled in, as preserving flowers was linked historically to rebirth and purity.

Back in 16th century Japan, flower pressing was an art form called 'oshibana' and they used dried flowers to make whole pictures and landscapes. In more recent times, Grace Kelly, aka Princess Grace of Monaco, practised oshibana as it presented her with a valuable creative outlet, and she actually wrote a book on the subject entitled *My Book of Flowers* in 1980.

Carrying on the royal theme, but going back a couple of hundred years, flower pressing really became a quintessential pursuit during the Victorian era when it became hugely popular, particularly amongst women in Victorian society. Indeed, Queen Victoria herself was a flower-pressing fan, and some of the blooms that she squished, including ferns and irises, are still preserved (rpimag.co/victoriaflowers). So, if you store these things correctly, they can literally last hundreds of years. Back then, people often thought of specific flowers as having certain meanings, and preserving them meant that they could also preserve their precious memories and sentiments, and that of course still holds true today.

Aside from looking pretty, pressed flowers also became important from the botanist's scientific angle, for documenting native flora and observing and sharing plant knowledge. Herbariums, or collections of plant specimens, have been a crucial tool for the study of plants, and one of the earliest has been traced back to 16th century Italy. Closer to home, Kew Gardens' herbarium receives many scientific visitors each year, as it contains over 7 million specimens (data.kew.org).



You'll need to remove the upper layers of the press to test the moisture levels of the flowers with your Picoconnected sensor after a few days; be careful not to press it on the petals too hard or for too long

QUICK TIP

Don't use sticky tape to attach pressed flowers to any surface as it will discolour your flowers – PVA glue is a great option.

A moisture sensor is typically placed in soil, but can be placed onto petals (or other items) to

measure how damp/dry they are. Moisture sensors come in two main types: resistive (with two prongs) and capacitive (one prong). We used a capacitive one from The Pi Hut (rpimag.co/pihutmoisture).

In our code we read the sensor's analogue output signal on GPIO 26 as an unsigned 16-bit integer every two seconds, which we convert into a voltage figure (in millivolts) that's printed to the shell so you can view it.

Connect the sensor to Pico as in the [Figure 1](#) wiring diagram – the breadboard is optional, as you could wire the sensor straight to Pico's pins if you want. The sensor's VCC pin is connected to 3.3V power on Pico, and GND to GND, while the AOUT analogue output goes to GPIO 26, one of Pico's analogue inputs. With Pico connected to a computer, create the following MicroPython program in an IDE (we used Thonny):

```
import machine, time
moisture = machine.ADC(26)
conversion_factor = 3300 / (65535)

while True:
    voltage = moisture.read_u16() *
conversion_factor
    print("Voltage is", voltage, "mV")
    time.sleep(2)
```

Pressing the sensor onto a damp petal results in a lower reading when using a capacitive sensor (or higher with a resistive one). By inserting the sensor into our flower press, we can check the current moisture level.

Whether you want to use your pressed flowers in some art work, embed them in soap or candles, incorporate them into some homemade paper, make some jewellery, or add them to a journal of herbarium sheets recording what you've grown, there are plenty of uses they can be put to. 🌱



If you have a bumper crop of flowers, think about preserving some with a flower press

Creative flower press options

Aside from the book method or using a flower press that you've purchased for your pressing projects, another option might be to make your own press:

- If you are pushed for time or just can't be bothered to wait for your foliage to dry out, take a leaf (preferably pressed) out of this maker's book: they have created a 3D-printed press that can go in the microwave to speed up the drying process – rpimag.co/3dprintedflowerpress. It's been designed using Tinkercad and everything that you need to get under way is detailed in the Instructable. So, if you have access to a 3D printer, get printing and then pressing!
- Alternatively, this maker has created their own laser-cut press from some plywood, nuts, bolts, paper, and cardboard – rpimag.co/lasercutflowerpress. They have chosen a lovely botanical image to laser-cut on the top. You could press some flowers and seal them onto the top of the press with some varnish.
- If money is no object, this made-to-order solid oak flower press is a beautiful object for any workshop – rpimag.co/oakflowerpress. The black finish is created by oxidising the wood.
- We could not end without also mentioning this fantastic idea: upcycling a vintage hardcover book, that might be missing a few pages, into a flower press – rpimag.co/bookflowerpress. They've cleverly drilled holes in each corner, and have used wing nuts, washers, and screws to create a press for smaller blooms. Aesthetically, this really hits the spot and the heritage/vintage vibe is right up our street.

Miniature rooms

Downsize your world because good things do come in small packages



Maker

Nicola King

Nicola is a freelance writer and sub-editor. This month she wishes she was six inches tall, as Lilliput is quite literally down the road from her (it really is... it's a place in Dorset!)



@nicoladj100

QUICK TIP

We found that superglue may be more effective to use than any glue provided in a kit, so have some to hand.

These kits also leave some room for creativity on the part of the maker

In case you weren't aware, miniature rooms and doll's houses in general are having something of a 'moment', with makers young and old alike. Your author was first alerted to the trend a couple of years ago while wandering around our local garden centre. For those not based in the UK, garden centres here often seem to make more revenue from 'bought in' goods – such as jigsaws, and overpriced candles, toiletries, kitchenware, and leather goods – than they do from plants and gardening paraphernalia. Their tearooms are often quite enticing too!

The items that caught our eye on that particular trip were a range of tiny make-it-yourself room 'scenes', which incorporate very detailed features including things like little books, a diminutive writing desk, and even some wee wall art in one library kit that we saw. Basically, the attention to detail is meticulous and impressive and that might be one reason why these scaled-down scenes have become so popular. Making something like this is a way of escaping the real world and transporting yourself to a tiny alternative world, perhaps with a vintage theme that's

comforting, cosy, and tranquil. Many people just find joy in such charming and adorable little things, but these kits also leave some room for creativity on the part of the maker, and that's what we want to explore here.

Scaled-down study

We decided to make up a small shop-bought kit of a cheerful little retro study, complete with desk and typewriter. One thing we would advise from the start is to ensure that you leave enough time to assemble and finesse your miniature. Despite the fact that the box for this kit was tiny, there were lots of elements to making it up that took more time than we had bargained for. We made an assumption that it would be a quick make – you'd think we'd know by now never to make assumptions. For example, the kit came with paint and a paintbrush, and a number of items required delicate painting. Clearly, you then need to allow time for the paint to dry, so just factor that in. It's also worth mentioning that, fortunately, this author had her eyes tested a month or so ago, so glasses are up to spec – we're just underlining that working with such small pieces can be problematic if your eyes are struggling. A magnifying glass may help.



Tiny but timeconsuming... if you've ever made anything miniature before – a house, plane, or car, for example – you will know that minutes turn into hours, turn into days, and so on

YOU'LL NEED

- A miniature room kit or doll's house

- Tiny FX, rpimag.co/tinyfx
- RGB LED dot, rpimag.co/tinyfxrgb
- Battery pack or USB power supply
- Tweezers
- Craft knife
- Small and large scissors
- Ruler/tape measure/pencil
- Magnifying glass (for detailed work)
- Glue selection, e.g. wood, fabric, tacky, super
- Sandpaper

QUICK TIP

Keep any leftover kit pieces for future projects – we had lots of fabric, wire, and other bits left for our next project.

Miniature marvels

Of course, miniature rooms and houses are nothing new and their history is thought to go back at least as far as 400 years. In fact, the considered view is that the original doll's houses were primarily designed for adults and were markers of wealth and status.

Cabinet doll's houses were popular in Germany and the Netherlands, for example, as a way of showing how a household functioned and as status symbols. The film *The Miniaturist*, based on the novel of the same name, is set in 17th century Amsterdam, and is inspired by the doll's house of rich merchant's wife Petronella Oortman, who had a miniature house crafted over a number of years. The house was extremely lavish, and she had it decorated with some very expensive materials.

Continuing the extravagant theme, if you have ever visited Windsor Castle, then you may remember seeing the historically significant Queen Mary's Doll's House (rpimag.co/qmdollshouse), generally regarded as one of the most famous miniature houses in the world, and when you see it in person, the craftsmanship is certainly very impressive. In fact, 2024 was the 100th anniversary of Queen Mary's Doll's House. Designed by Sir Edwin Lutyens, and made to one twelfth scale, it is housed in a huge glass cabinet that you can walk around to view every minute Edwardian detail. Conservators regularly check the contents in case repairs are needed, and it's mesmerising. The doll's house even includes electricity and running water!

If you are looking for some inspiration on television, we can highly recommend *The Great Big Tiny Design Challenge*, a Channel 4 series hosted by Sandi Toksvig, where miniature-making skills are tested as crafters work to try and transform a doll's house that just happens to be a mini mansion.



We used this tiny typewriter to write this tutorial... only kidding, but we do wish we could shrink down and live in this lovely space

The basic kit took a couple of days to complete in total, due to waiting for paint and glue to dry, but it was worth taking the process slowly – you may as well invest some time in order to do it properly. Plus, if you have a modelling toolkit from previous forays into making models, then do dig it out. One point we would emphasise is to read the instructions through before you get started, as this will aid the process immeasurably. Also, we found a video on YouTube of someone making the exact same kit as ours, so that was handy when we had queries. At the end of the day, remember that when you are doing something like this, the whole point is to enjoy it and slow down, not to stress yourself out.

Create then illuminate

To make your scene really come to life, the optional final step is to light it up. Some kits come with LED lighting included, but ours didn't. You could use a Raspberry Pi Pico to control one or more LEDs, as in our origami tutorial in issue 155, but we wanted a tiny LED dot to fix to the lamp in our scene, so we opted to use Pimoroni's RP2040-based Tiny FX microcontroller board ([rpimag.co/tinyfx](https://pimoroni.com/tinyfx)). Its starter kit includes several single-colour white LEDs and RGB ones with thin wires and JST-SUR connectors to plug into the board, along with a 3 × AA battery pack. It's great for LEGO lighting too.

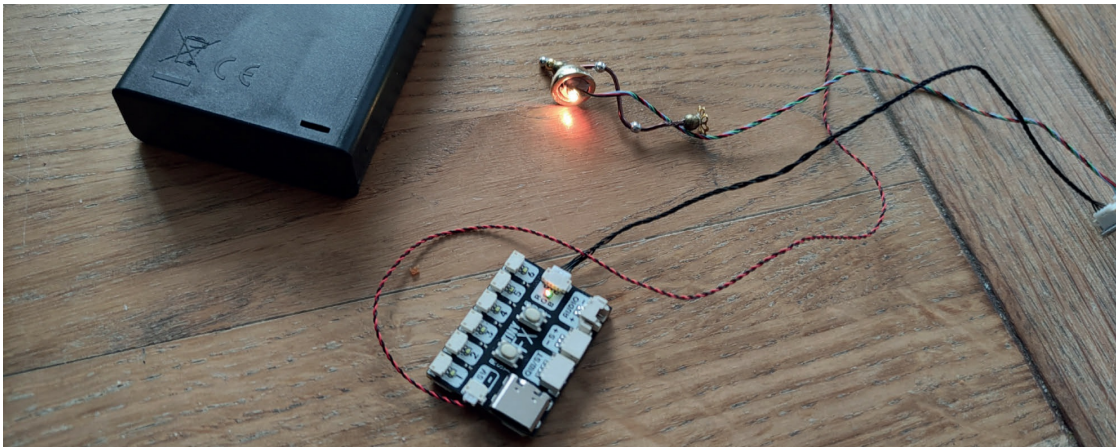
We attached an RGB LED dot to our scene's wall lamp. Pimoroni's PicoFX MicroPython library ([rpimag.co/picofx](https://pimoroni.com/picofx)) for LED dots and strips makes coding easier. Using this, we were able to get the RGB LED to switch between several shades by pressing Tiny FX's BOOT button.

QUICK TIP

Projects like this are not suitable for very small children due to the tiny pieces involved, so do bear that in mind before involving youngsters in your make.



We used an RP2040-based Tiny FX board and battery pack to light up our scene



We used Blu Tack to stick our LED dot into the tiny lamp; you could use glue instead

Here's our code:

```
from tiny_fx import TinyFX
from picofx.colour import RGBFX
from picofx import ColourPlayer
import time

# Setup
```

```

tiny = TinyFX()
player = ColourPlayer(tiny.rgb)

# Define a list of RGB tuples
colors = [
    (128, 64, 0),    # Orange yellow
    (112, 80, 0),    # Yellow
    (104, 64, 24),   # Warm white
    (72, 72, 48),    # Cold white
    (0, 0, 0),       # Off
]

idx = 0
player.effects = RGBFX(*colors[idx])
player.start()

try:
    last_state = False
    while True:
        pressed = tiny.boot_pressed()
        if pressed and not last_state:
            # Button just pressed: cycle color
            idx = (idx + 1) % len(colors)
            player.effects = RGBFX(*colors[idx])
            last_state = pressed
            time.sleep(0.05)

finally:
    player . stop()
    tiny . shutdown()

```

Here, we use a list called `colors` to store a range of RGB tuples for different shades (after experimenting to find ones we like). An `idx` variable, used to select a colour from the list, is incremented every time the BOOT button is pressed during our infinite `while True` loop. We also use a `last_state` variable and short time delay as a debounce mechanism to prevent multiple button presses being registered when pressing once. As usual, save the code as **main.py** so it'll run automatically on bootup. The end result: our miniature room can now be lit up subtly in different shades to suit our mood.



This is a different kit, but it gives you a flavour of what you'll find in the box: everything from paper cutouts to laser-cut and plastic pieces

QUICK TIP

You could use wallpaper samples or scrapbooking paper for wall coverings, and carpet samples for the floors of your room/house.

To sum up, we'd highly recommend going small and just spending some time thinking about nothing else other than painting tiny things and squinting hard to get that dinky coffee cup stuck down in the right place... it's mindful and it's enjoyable. Bringing Raspberry Pi RP2040 or Pico into the mix is easy to do, whether that's with some lighting or even perhaps some sound effects... and on that note, we'll be delving into the realm of adding sound to a crafting project in a future tutorial, so stay tuned! 🟢

Lilliputian options

1. Get kitted out

We took the easy option and bought a DIY kit. There are numerous places to find kits online; our only advice would be to opt for one that has some decent instructions so you are not left frustrated and regretful within five minutes of opening the box. Popular brands include Rolife (robotimeonline.com) and CuteBee (cutebee.net).

2. Be your own architect

If you don't want to buy a miniature kit and you have the time required, how about designing and constructing your very own tiny house, shop, or room? Researching the topic, this author realised that she had written a project

showcase on the very subject some seven years ago. Two creative students, Maks Surguy and Yi Fan Yin, created an interactive doll's house for their master's degree course using machine learning. Take a look here at their clever work for some itty-bitty inspiration: rpimag.co/myhouse.

Making your own miniatures means you can use whatever materials you like in your makes, perhaps recycling fabrics, cardboard, plywood, or paint that you have in your stash. The kits mentioned above do tend to include a fair bit of plastic, so the recycling route might be more planetfriendly. Plus, the creative freedom that you have will know no bounds! Use some air-dry clay to make fun-size food items, use fabric scraps to make a mini quilt for a bed or scaled-down sofa... you get the idea. You could even make a version of your childhood bedroom to keep for posterity. So, base it on something you know as a guide.



As you can see, it can be messy, so protect your surfaces with some old newspaper

QUICK TIP

For precision work, toothpicks are a handy way of applying glue and getting into fiddly corners. Bulldog clips are also useful for holding things together as they dry!



Kits these days rely heavily on laser-cut pieces, making the process a little easier as things fit together with precision



- Available now: rpimag.co/store

Touch sensor quilt block

Quilt yourself a little piece of history and let Raspberry Pi play its part too



Maker

Nicola King

Nicola is a freelance writer and sub-editor. One thing she was not short of for this project was fabric – her stash of all things crafty knows no bounds!



@holtonhandmade

QUICK TIP

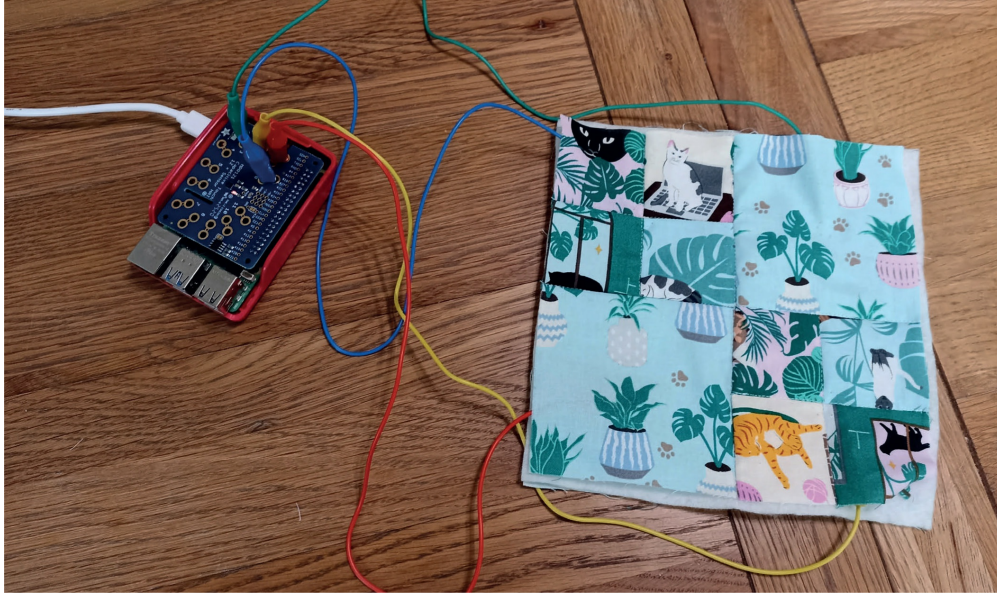
Press and starch quilting fabric before you cut into it – it gives stability, control and this also aids precision.

Incorporating sensors that will trigger a sound when touched

Quilt making has a history that can be traced back many hundreds of years and, as well as being items that provide much-appreciated warmth and comfort to the user, quilts often become cherished heirloom pieces with the unique handiwork, history, and love of the maker literally deeply embedded within them. Early American settlers in particular were highly skilled in quilt making, and sitting together to hand-sew their pieces became something of a social event, with stories shared and friendships strengthened. Today's quilters often employ the same techniques used many years ago, but new and contemporary designs have evolved too, and a quilt can easily become an e-textile project if you want it to be. The Tilt Sensing Quilt (rpimag.co/tiltquilt), for instance, uses 41 textile tilt sensors and the maker wrote a basic application that displays the tilt direction of each square as a unique colour.

Inspired by quilt-makers both old and new and by the use of technology by some very savvy and current quilt makers, in this tutorial we're going to make a simple quilt block, and we're

going to bring this historic hobby bang up-to-date by incorporating some sensors that will trigger a sound when touched. The hope is that you will be inspired to make your own bespoke quilt, maybe even a full-sized version, to deliver warmth plus, and maybe a wow factor to boot if you would like to try incorporating sensors too.



Adding the electronics to the project was straightforward using a Raspberry Pi and a Capacitive Touch HAT



The individual fabric elements of the top layer of the quilt. You can really use any fabric, but cotton is a favoured choice for many quilters

Quilt sandwich

So, a quilt is usually made up of three layers: your top layer which is the colourful, carefully planned layer, the middle layer of padding or ‘wadding’, and the backing to the quilt which is usually in a plain fabric. Use whatever you have to hand for the top and bottom layers – perhaps some old clothes that you want to upcycle, or fabric from a stash. We are using some fat quarters that complement each other, and a plain backing fabric to contrast. We are also using some very simple shapes here for the sake of ease and speed, but the more adventurous among you may

want to employ more complicated shapes.

We cut eight 5cm squares in differing fabrics, to use in each of two four-square blocks in our design. (Note that usually quilters measure in inches, but we'll stick with metric here, and also note we'll be using a 5mm seam allowance when we sew.) We also cut two larger 9cm squares in another fabric. We then took two smaller squares and machine-sewed them together along one seam with a 5mm seam allowance, right sides to right sides. We repeated that with two other small squares, then sewed those two strips together (to form a square) along one seam, right sides facing right sides, pressing all seams with an iron as we went, and matching the seams to each other and pinning before joining. We repeated that with the other four small squares, and then took each four-square block and sewed it along one seam to one of the larger 9cm blocks. Then it's just a question of sewing the two pieces, each containing one large square and one four-square block, together to form a final larger square.

QUICK TIP

Accurate cutting and accurate sewing are the keys to points lining up and a professional-looking result, so measure twice, cut once!

YOU'LL NEED

- Top layer of cotton fabric
- Plain backing cotton fabric
- Wadding layer
- Iron
- Sewing machine
- Hand sewing needle
- Regular thread
- Pins/seam ripper
- Scissors and rotary cutter
- Cutting mat
- Ruler and tailor's chalk
- Raspberry Pi computer
- Adafruit Capacitive Touch HAT, rpimag.co/touchhat
- Crocodile clips
- Conductive thread (optional)



The three layers of the quilt, ready to be sewn together. The middle layer is a wadding material and there are lots of options that you can purchase

Next, we cut a piece of wadding and a piece of backing fabric to the same size as our top layer, and we then stitched all three layers together (with the right side of the backing layer showing on the bottom), and quilted by employing the ‘stitch in the ditch’ technique, whereby we sewed along the seams or channels that we’d already made on the top layer of fabric to produce one quilted block. Now, let’s look at the electronics.

Touch and go

To make our quilt interactive, we’ll use touch sensors that will trigger sounds. For this, we opted to use a Raspberry Pi equipped with an Adafruit Capacitive Touch HAT. You will need to use the Blinka compatibility library to make use of the required CircuitPython libraries in Python on Raspberry Pi. In a command-line terminal, we create a new virtual environment, then download and run a setup script:

```
$ mkdir touch
$ python -m venv touch --system-site-packages
$ source touch/bin/activate
$ pip3 install --upgrade adafruit-python-shell
$ wget https://raw.githubusercontent.com/adafruit/Raspberry-Pi-Installer-Scripts/master/raspi-blinka.py
$ sudo -E env PATH=$PATH python3 raspi-blinka.py
```

QUICK TIP

As with anything, make sure you have good lighting and are sitting comfortably when you're working.

It'll take a little while and then ask you to reboot, so enter **Y**. Reactivate the virtual environment and install the mpr121 library for the Touch HAT:

```
$ source touch/bin/activate
$ pip3 install adafruit-circuitpython-mpr121
```

Move to the **touch** directory and create a new file:

```
$ cd touch
$ nano touch_test.py
```

We'll now write some code to test the touch inputs on the HAT:

```
import time
import board
import busio
import adafruit_mpr121

# Create I2C bus
i2c = busio.I2C(board.SCL, board.SDA)

# Create MPR121 object
mpr121 = adafruit_mpr121.MPR121(i2c)

# Loop forever, play sounds upon touch
while True:
    # Loop through inputs (0-11) or how many you need
    for i in range(12):
        if mpr121[i].value:
            print(f"Input {i} touched!")
            time.sleep(0.25) # short delay
```

Before running it, connect some crocodile clips to several of the numbered holes on the Touch HAT. When the code is run – with `python3 touch_test.py` – touching the other end of each cable will trigger a message to say it's been touched.

We'll now sense when an input is touched and play a sound assigned to it. For the latter, we downloaded some Creative Commons-licensed sound effects from freesound.org, which we renamed **boing.wav**, **crash.wav**, **horn.wav**, and **meow.wav** and placed in the **touch** directory. The code for our **touch_sounds.py** program is:

```
import time
import board
import busio
import adafruit_mpr121
import pygame
#Initialise Pygame
pygame.init()

# Create I2C bus
i2c = busio.I2C(board.SCL, board.SDA)
# Create MPR121 object
mpr121 = adafruit_mpr121.MPR121(i2c)

# Assign sounds and create list
boing = pygame.mixer.Sound('boing.wav')
crash = pygame.mixer.Sound('crash.wav')
horn = pygame.mixer.Sound('horn.wav')
```

```

meow = pygame.mixer.Sound('meow.wav')
sounds = [boing, crash, horn, meow]

# Loop forever, play sounds upon touch
while True:
    # Loop through inputs (0-3) or how many you need
    for i in range(4) :
        if mpr121[i].value:
            print(f"Input {i} touched!")
            pygame.mixer.Sound.play(sounds[i])
            time.sleep(0.25) # short delay

```

Touching the crocodile clip then triggers the connected input on the Touch HAT and plays the assigned sound. We used a Bluetooth speaker for the audio.

We found that our touches were sensed when the crocodile clips were placed under our thin top layer of fabric. If yours is thicker, you may need to use conductive thread to connect the clips to the surface, but take care not to create a short-circuit otherwise it'll register phantom presses.

Of course, you could also use touch inputs to trigger any other action Raspberry Pi can perform, such as lighting LEDs. So, there are plenty of possibilities to get creative! 🟢



The back layer after being quilted using the stitch-in-the-ditch method. The back can be a plain fabric if you wish

QUICK TIP

If you have a 'walking foot' for your sewing machine, use it when quilting – this helps pull through all the layers of fabric at the same time.

The mathematics of quilting

When you make and sew anything with fabric, a certain amount of maths is always required, so that you measure and cut the correct amount of fabric for example. However, when it comes to quilting, this author would argue that the mathematics involved is more complex, and geometry sits at the heart of it all. If you take a look at many quilts, more complex than our own simple block, you can see the precision of the angles within the blocks, the

symmetry – these are things that quilters strive to achieve and this takes planning and accuracy. Triangles, hexagons, chevrons, whatever repeating shapes a quilter is employing within their design, numbers are a part of the art.

The Fibonacci sequence and the golden ratio are just two mathematical concepts that some quilters use when making very intricate quilts (rpimag.co/quiltgeometry), and once you fall down this rabbit hole, you'll want to keep exploring.

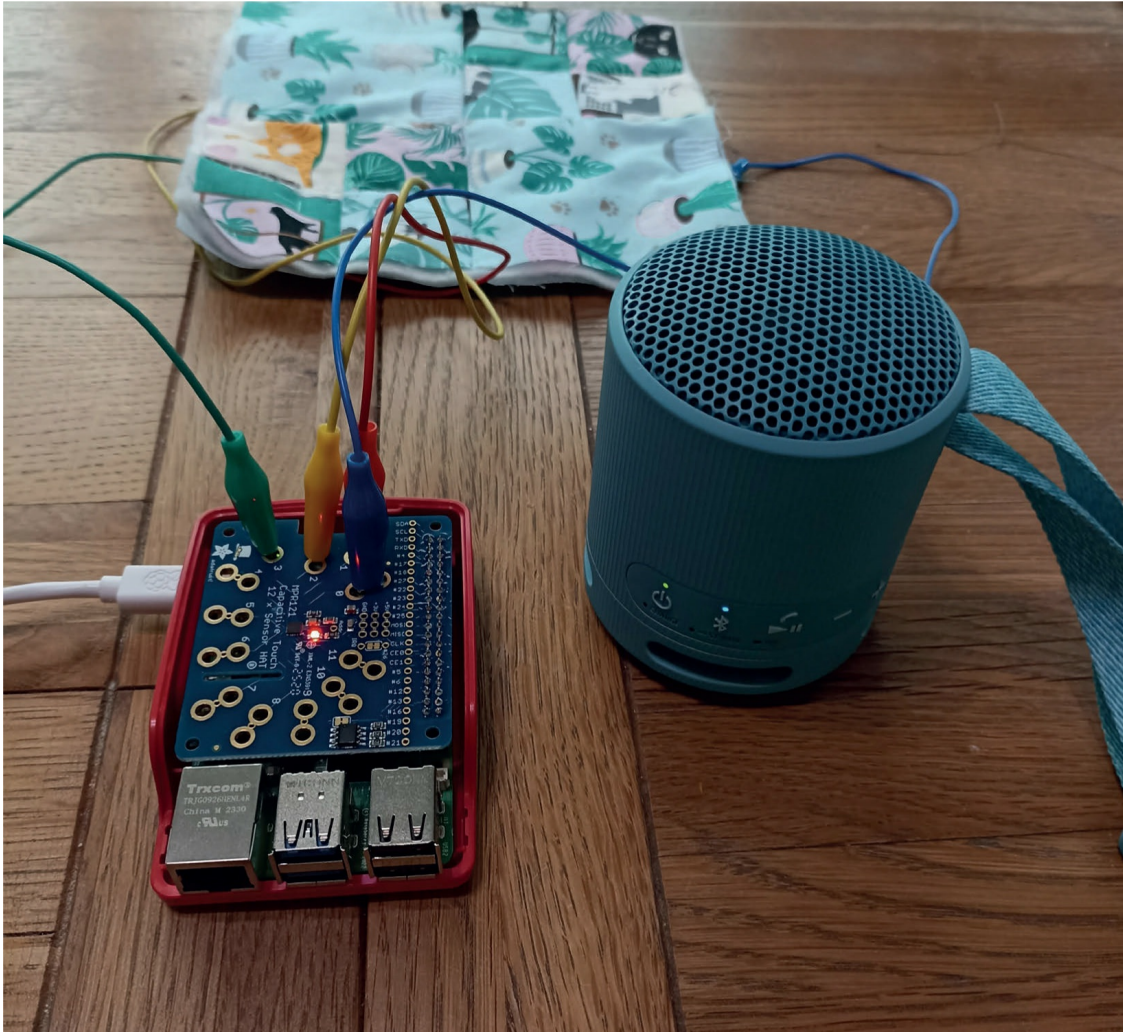
QUICK TIP

Graph paper is a handy tool when planning – draw the pattern on it, and use it to help calculate the amount of fabric required.

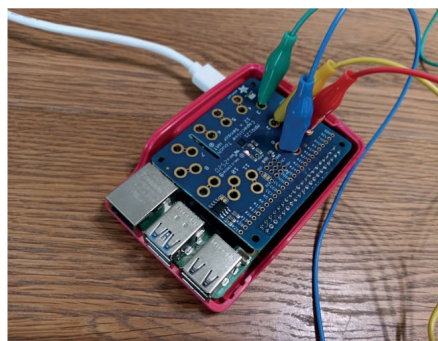
Innovative quilting projects

Here are two more examples of very original quilt projects that are taking the concept of quilting in a fresh and exciting direction:

1. Made back in 2013, this beautiful interactive quilt (rpimag.co/interactivequilt) in Australia has been purposed as a huge wall hanging, and has incorporated electronics, including sensors and LEDs, to fully illuminate the amazing sewing work of the makers. Made by a team of individuals who each worked on their own sections, when touched, the interactive panels react in a very illuminated fashion!
2. This clever and creative project (rpimag.co/newmumsquilt) led by Swansea University brought together mothers of young babies so that they could explore their experiences of feeding. Pieces of their babies' clothing were stitched together to create an interactive quilt, and digital sensors were also incorporated. When the sensors are pressed, audio clips play and reflect the mothers' feelings around and experiences of feeding their babies, which can often be a challenging and testing part of motherhood. Quilted spiral shapes, often seen on Welsh quilts, have been included to reflect a new mother's spiralling emotions.



We connected a Bluetooth speaker to output the audio samples triggered by the touch sensors



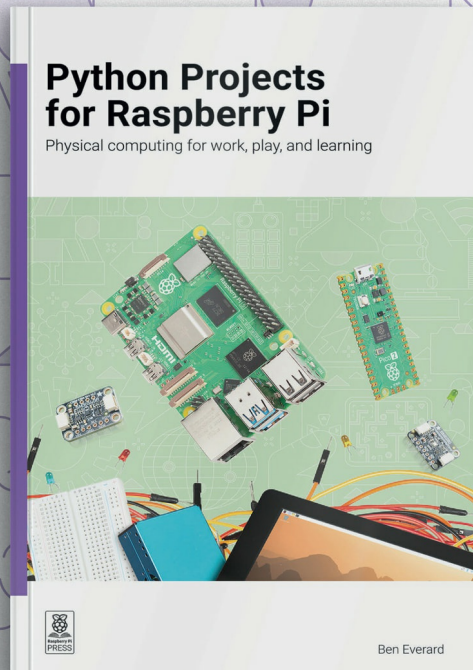
We used a Capacitive Touch HAT, but the same MPR121 touch sensor chip is also available on a breakout board that could be used with Raspberry Pi or Pico

A quilting quest

Whether you want to hand-sew or machine-sew a whole quilt or just a smaller block, this is a pastime that offers

so many possibilities, and you can literally create any design you want, constructing a unique and beautiful piece of art in the process. If you would like to learn more, here's some inspiration:

1. For a contemporary take on quilting, this is a great read and shows how quilting can even be a social commentary: rpimag.co/vamquilters.
2. If you want to feel part of a community, how about joining a group where you can find support on the journey (quiltersguild.org.uk), or find a group local to you? You can impress with your e-textile knowledge and change some perspectives!
3. For a one-stop shop guide to quilting, find yourself a 'bible' that quite literally lays it all out for you, like this one: rpimag.co/quiltbible.



Computers are embedded into almost everything we own. Our doorbells, kitchen gadgets, vacuum cleaners, and media players are all powerful computing devices running software that someone has written.

Using the flexible Python programming language, *Python Projects for Raspberry Pi* shows you how to get the most out of the Pico range of microcontrollers and Raspberry Pi computers like Raspberry Pi 5.

■ **Learn how to:**

- *Interact with sensors and motors*
- *Build a weather station*
- *Make your own games console*
- *Add web interfaces to your projects*
- *Recognise gestures with machine learning*
- *Get the most out of your Raspberry Pi Pico with MicroPython*
- *Extend Python with C*

BUY ONLINE: [rpimag.co/pythonprojectsbook](https://www.rpimag.co/pythonprojectsbook)

SUBSCRIBE TODAY FOR JUST £10

Get **3 issues** + **FREE** Pico 2 W

SUBSCRIBER BENEFITS

Free delivery

Get it fast and for free

Exclusive offers

Great gifts, offers, and discounts

Great savings

Save up to 37% compared to stores

SUBSCRIBE FOR £10

Free Pico 2 / Pico 2 W

3 issues of Raspberry Pi Official Magazine

£10 (UK only)

SUBSCRIBE FOR 6 MONTHS

Free Pico 2 / Pico 2 W

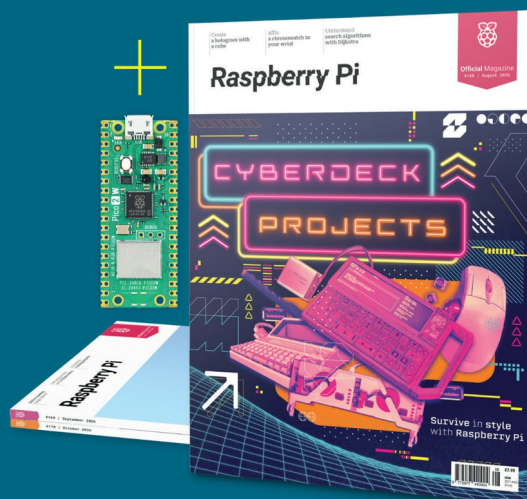
6 issues of Raspberry Pi Official Magazine

£30 (UK)

\$43 (USA)

€43 (EU)

£45 (Rest of World)



☎ Subscribe by phone: +44 (0)1778 392472

🌐 Subscribe online: rpimag.co/subscribe

✉ Email: raspberrypi@warnersgroup.co.uk

Subscribe for £10 is a UK-only offer. The subscription will renew at £15 every three months unless cancelled. A choice of free Raspberry Pi Pico 2 W or Pico 2 is included with all subscriptions.

SUBSCRIBE TODAY AND GET A

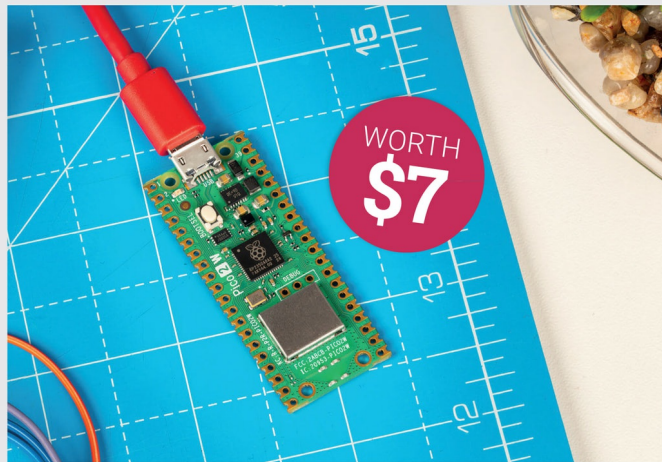
FREE Raspberry Pi Pico 2 W
(or Pico 2)

Subscribe in print
today and get a **FREE**
development board

A brand new RP2350-based Raspberry Pi
Pico 2 W / Pico 2 development board

Learn to code with electronics and
build your own projects

Make your own home automation
projects, handheld consoles, tiny robots,
and much, much more



Free Pico 2 or Pico 2 W. Accessories not included. This is a limited offer. Not included with renewals.
Offer subject to change or withdrawal at any time.



 Buy now: rpimag.co/subscribe

SUBSCRIBE
on app stores
From **£2.29**

 Available on the
App Store

GET IT ON
 Google Play



STARTER ELECTRONICS

Get started with electronics with this
simple one-dimensional game for
Raspberry Pi or Raspberry Pi Pico

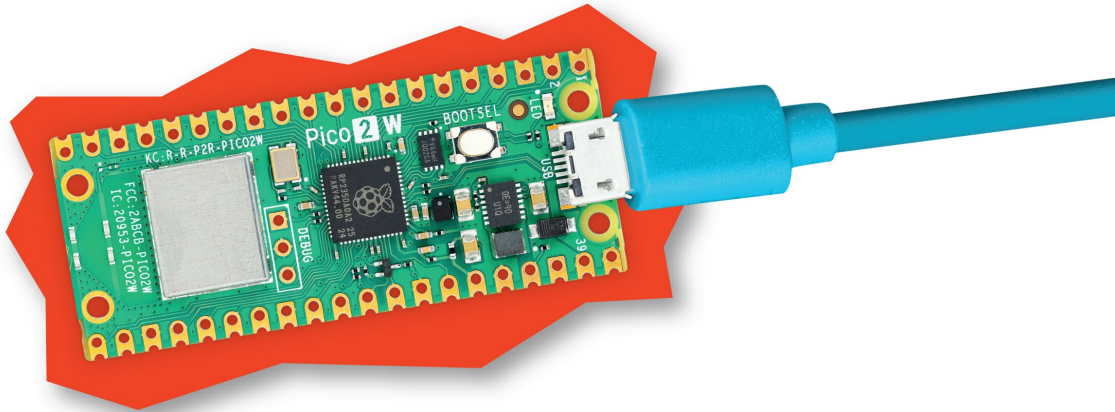


Maker

Ben Everard

Ben's main mission
in life is to add more
LEDs to the world. He
is yet to find a project
that isn't improved by
adding more LEDs.

glowingart.co.uk



DOWNLOAD THE FULL CODE:

rpimag.co/github

Language: CircuitPython

Raspberry Pi computers and microcontrollers are great tools for interacting with things in the real world. We can add our own hardware or interface them with other electronics using their GPIO (General-Purpose Input/Output) pins. These pins can detect voltage (input), send out voltage (output), and even transfer more complex data using low-level protocols. In this article, we'll use these three different abilities to build a game.

We're going to use CircuitPython, because this Lets us use the same code on both Pico and Raspberry Pi

We're going to use CircuitPython, because this lets us use the same code on both Pico and Raspberry Pi. This allows you to use whatever hardware you already have, and it also helps you understand a little more about the capabilities of each device. If you've not used CircuitPython before, take a look at the boxes throughout this guide to get started.

Button basics

Let's start with one of the simplest uses for a GPIO pin: reading voltages. Most GPIO pins are digital, which means they don't measure the exact voltage level. Instead, they only detect whether the voltage is high (3.3V on Pico and Raspberry Pi) or low (0V). (Some pins on Pico can measure varying voltage levels – these are called analogue pins – but we won't be using those in this project.)

To read data using a GPIO pin, we need to build a simple circuit that gives us 3.3V when something is 'on' and 0V when it's 'off'.

It's important to understand that 0V is not the same as an unconnected pin. 0V is a defined voltage level, while an unconnected pin (called 'floating') has no definite voltage and may behave unpredictably. In this state, the pin can randomly read as HIGH or LOW due to electrical

noise or interference. This is why we use a pull-up or pull-down resistor – to give the pin a default, reliable state when nothing else is connected.

Figure 1 shows a simple circuit for connecting a button to a GPIO pin on Pico. When the button is not pressed, the GPIO pin is connected to 3.3V via the pull-up resistor so reads HIGH. When the button is pressed, it connects the GPIO pin directly to ground (the GND pin). This creates a clear path for current to flow to ground, and the GPIO reads a LOW signal (0V).

Since pull-up and pull-down resistors are very often needed with GPIO pins, both Pico and Raspberry Pi have internal resistors that you can use. You just have to initialise them in your program.

Now, with that theory behind us, let's connect a button to Pico. There are lots of different buttons you could use for this. If you want, you could even just use a wire to connect GND and a GPIO pin and attach and detach the wire to simulate pressing the button.

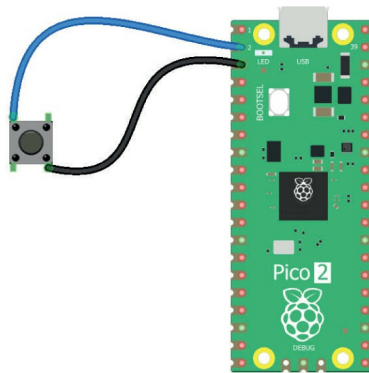
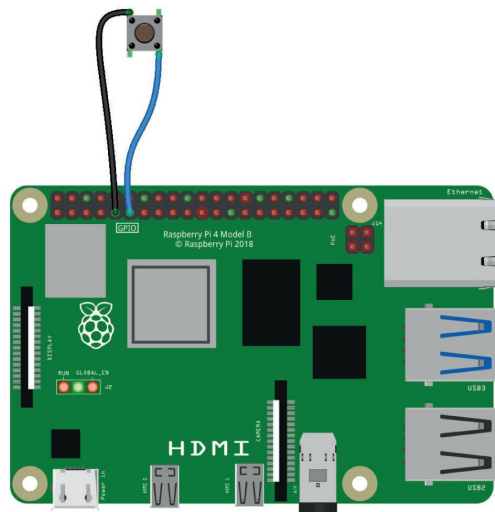


Figure 1: If your button has four connections, use diagonally opposite ones



Buttons are more correctly called momentary push switches. They are usually 'open' and when you press them, they 'close' the connection and complete the circuit

Using Raspberry Pi

You can use CircuitPython code on a Raspberry Pi by installing Blinka. Blinka acts as a compatibility layer, allowing CircuitPython features to work with regular Python. To install this, we need to first create a virtual environment.

```
$ python3 -m venv venv --system-site-packages
$ source venv/bin/activate
```

Next, set up your Raspberry Pi and install Blinka with these commands:

```
$ pip3 install --upgrade adafruit-python-shell
$ wget https://raw.githubusercontent.com/adafruit/Raspberry-Pi-Installer-Scripts/master/raspi-blinka.py
$ sudo -E env PATH=$PATH python3 raspi-blinka.py
```

The first command installs a required package, the second downloads the Blinka installer script, and the third runs the installer.

You may be asked to reboot the machine when this runs.

Once the installation is complete, you can use Blinka to run CircuitPython code with the regular Python interpreter. Just make sure your virtual environment is activated. You can activate it at any time by running the following command in the terminal:

```
$ source venv/bin/activate
```

To run any of the code in this article, save it to a file and run:

```
$ python <filename.py>
```

The code to read this in Circuit Python is shown below.

The biggest difference between Pico and a Raspberry Pi computer is the pin numbering is a bit different. Pico pins are called GP0 to GP27 while Raspberry Pi pins are D0 to D26. Not all the pins can be used for everything as some are reserved for particular functions. In general, the pins on Pico are more flexible than the pins on a Raspberry Pi computer.

We've provided both pin versions in the code below and you'll need to comment out (add a # at the start of the line) the one you're not using, and uncomment (remove the # at the start of the line) the one that you are using.

Buttons usually have either two or four connections. If your button has two connections, then you can connect it either way around. If it has four connections, then there are really two sets of two. The easiest way to do it is to use diagonally opposite connections – either diagonal works.

We'll show the direct wire connections for the circuit, but you can do this using wires or a breadboard.

```
import board
import digitalio
import time

#Pico
BUTTON_PIN = board.GP1
```

```
#Raspberry Pi
#BUTTON_PIN = board.D17

button = digitalio.DigitalInOut(board.GP14) # Use appropriate GPIO pin
button.direction = digitalio.Direction.INPUT
button.pull = digitalio.Pull.UP # Enables internal pull-up resistor

while True:
    if not button.value : # Button pressed (pin reads LOW)
        print("Button is pressed")
    else:
        print("Button is not pressed")
    time.sleep(0.1)
```

One slightly odd thing about this code is the line:

```
if not button.value
```

You might expect that to read `if button.value`, but we're using a pull-up which reads HIGH (which evaluates as `True`) when the button is pressed and LOW (i.e. `False`) when it is pressed. This might seem a bit backwards at first, but it helps if you think about what's going on at an electrical level.

Blink and you'll miss it

We've created a simple input device, and now let's look at the simplest output device: a light-emitting diode, or LED. LEDs light up when you put a voltage across them. However, LEDs are sensitive to too much current, which can damage them. You can calculate the exact resistor value needed using formulas based on voltages and current limits, but to keep things simple, just use a 330Ω resistor in series with the LED. [Figure 2](#) (overleaf) shows you how to connect up a resistor and an LED to a GPIO pin. Unlike pull-up or pull-down resistors, Pico doesn't provide internal current-limiting resistors for LEDs, so we need to add an external one.

CircuitPython on Pico

This tutorial will work on any version of Pico. To install CircuitPython on Pico, you first need to download the appropriate UF2 file from pimagic.co/circuitpython. Then, press and hold the BOOTSEL button on the Pico while connecting it to your computer via USB. Keep holding the button until the Pico is fully connected.

Once it's connected, you should see a new drive called 'RP2' appear. Drag and drop the UF2 file onto the RP2 drive. This installs CircuitPython onto Pico. After a few seconds, the RP2 drive will disappear, and a new drive called 'CIRCUITRY' will appear.

Now download the Mu code editor from codewith.mu and install it on your computer. You can use this to enter, edit, and save the code we're creating in this article. Click the Serial button in Mu to view the output from CircuitPython running on Pico. This can be particularly useful in finding any errors.

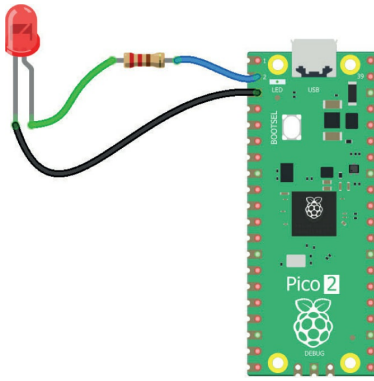


Figure 2: We need to add a resistor to prevent too much current from damaging the LED

Unlike buttons, LEDs have an orientation. That means that they have to be connected the right way around. Usually, there is a flat side and this marks the side that should be connected to ground (GND). You won't damage the LED by connecting it the wrong way around, so you can just pop it in and if it doesn't work, turn it around.

Here's some CircuitPython code to blink an LED:

```
import time
import board
import digitalio
#Pico
#LED_PIN = board.GP1

#Raspberry Pi
LED_PIN = board.D17

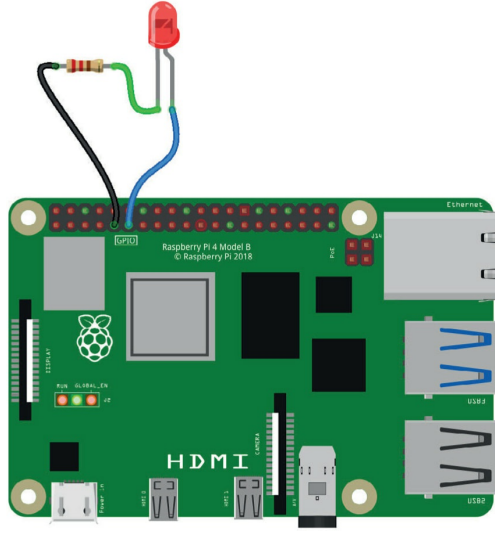
led = digitalio.DigitalInOut(LED_PIN)
led.direction = digitalio.Direction.OUTPUT

while True:
    led.value = True # Turn LED on
    time.sleep(0.5) # Wait for 0.5 seconds
    led.value = False # Turn LED off
    time.sleep(0.5) # Wait for 0.5 seconds
```

LED there be (more) light

Buttons and LEDs both work with a single bit of data. That means they can only be in one of two states. A button is either pressed or not pressed, and an LED is either on or off. Many other devices are more complex and need to send and receive more detailed information – for example, to control colours, positions, or movement.

We're going to use WS2812B LEDs, which are also known as NeoPixels. These allow you to control many RGB LEDs from a single GPIO pin using a special communication protocol. Each LED's colour can be changed by adjusting how much red, green, and blue light it emits.



You can put the resistor on either side of the LED and it will work

Wiring them up is simple. You just need to connect the 5V pin on the LED strip to the 5V output on Pico, GND to GND, and the Data In pin on the strip to a GPIO pin. Be aware that these LED strips are directional. This means you must connect the GPIO pin to the correct end, which is usually marked with 'DIN' or an arrow. If you connect to the wrong end, the LEDs won't respond.

WS2812B LEDs come in a lot of different forms, but when you get them, there is typically a wire with a three-pin connector on the end. Usually, the socket end of the connector is on the Data In side, but this isn't completely standard. For testing things out, we like to use pin-to-socket headers and we can poke the pin into the socket on the LED strip, then connect the socket to a GPIO on either Pico or Raspberry Pi.

Since these have a slightly more complex communications protocol, we need to add a module that implements this protocol. The process for adding modules is a little different on each platform.

On Pico, head to rpimag.co/circpylibraries and download the bundle for the version of CircuitPython you have. This will be a zip file. Extract it and copy the **neopixel.mpy** file from within the **lib** folder to the **lib** folder on the CIRCUITPY drive.

On Raspberry Pi, make sure you're in the virtual environment you created when you set up Blinka, then run:

```
pip install adafruit-circuitpython-neopixel-spi
```

Here is some code to test everything out by flashing up some colours.

Initialising WS2812B LEDs is a bit different on each platform, so there are two blocks of code: one for Raspberry Pi and one for Pico.

```
import board
import busio
import time

NUM_PIXELS = 10
BRIGHTNESS = 0.5
```

```

#For Pico
#import neopixel
#PIXEL_PIN = board.GP0
#BUTTON_PIN = board.GP1
#pixels = neopixel.NeoPixel(PIXEL_PIN, NUM_PIXELS, brightness=BRIGHTNESS, auto_write=True)

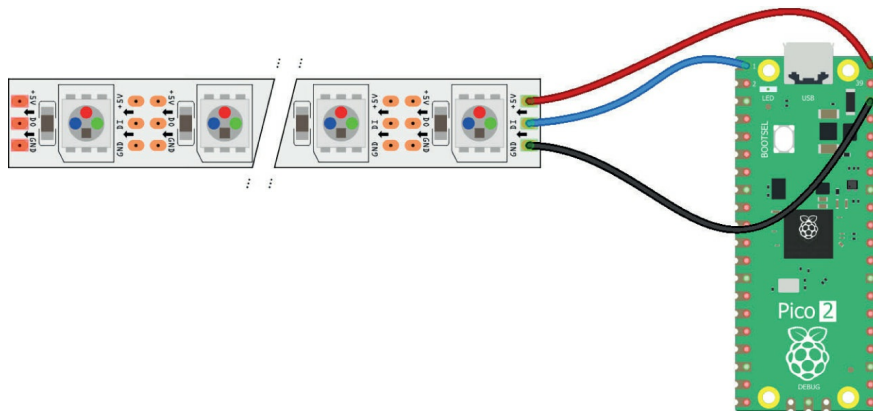
#For Raspberry Pi
#import neopixel_spi
#PIXEL_PIN = board.D10
#CLOCK_PIN = board.D11
#pixels = neopixel_spi.NeoPixel_SPI(busio.SPI(clock=CLOCK_PIN, MOSI=PIXEL_PIN), 10)

colors = [
    (255, 0, 0),      # Red
    (0, 255, 0),      # Green
    (0, 0, 255),      # Blue
    (100, 100, 100)   # White
]
while True:
    for color in colors:
        pixels.fill(color)
        time.sleep(0.5)

```

On Pico, we can use any pins for the LEDs; however, on Raspberry Pi we are more limited. We have to use GPIO 10 for the pixel pin because this one is attached to the SPI peripheral, and it's this that we're using to send the data out to the LEDs.

On Raspberry Pi we've used two pins for the LEDs, but we'll leave one of them unconnected. It's because we're using a bit of hardware designed to do something different to control the NeoPixels. We do have more flexibility with the button pin, but can't just use any as some are reserved for specific functions.



There are different types of addressable LEDs, so make sure you have 5V WS2812B LEDs. If you have something else, they won't work

WS2812B LEDs come in a Lot of different forms


```

#For Pico
#import neopixel
#PIXEL_PIN = board.GP0
#BUTTON_PIN = board.GP1
#pixels = neopixel.NeoPixel(PIXEL_PIN, NUM_PIXELS, brightness=BRIGHTNESS, auto_write=False)

#For Raspberry Pi
#import neopixel_spi
#PIXEL_PIN = board.D10
#CLOCK_PIN = board.D11
#BUTTON_PIN = board.D17
#pixels = neopixel_spi.NeoPixel_SPI(busio.SPI(clock=CLOCK_PIN, MOSI=PIXEL_PIN), 10)

DELAY = 0.3

GREEN = (0, 255, 0)
RED = (255, 0, 0)
WHITE = (100, 100, 100)
FLASH_DURATION = 0.6

# Button setup
button = digitalio.DigitalInOut(BUTTON_PIN)
button.direction = digitalio.Direction.INPUT
button.pull = digitalio.Pull.UP

def flash_color(color, duration= FLASH_DURATION) :
    pixels.fill(color)
    pixels.show()
    time.sleep(duration)

while True:
    red_pos = 0

    # Move the red pixel forward
    while red_pos < NUM_PIXELS:
        # Draw strip
        for i in range(NUM_PIXELS):
            if i == red_pos:
                pixels[i] = RED
            else:
                pixels[i] = GREEN
        pixels.show()

        # Wait for DELAY, check for button press

        t0 = time.monotonic()
        pressed = False
        while time.monotonic() - t0 < DELAY:
            if not button.value:
                pressed = True
                break

        if pressed:
            if red_pos == NUM_PIXELS - 1:
                # SUCCESS!
                flash_color(WHITE)
            else :
                # FAIL!
                flash_color(RED)
            break

        red_pos += 1
    else:
        # If it reaches the end and you didn't press, flash red for "miss"
        flash_color(RED)

```



```
# Small pause before restarting  
time.sleep(0.7)
```

Here ends our whirlwind tour of electronics on Pico and Raspberry Pi. We've skipped over some of the detail to help you get up and running quickly, but hopefully you now know the basics of connecting other hardware to these two devices.

You've also probably got a bit more of an idea about the differences between Pico and Raspberry Pi. The former has more flexible I/O options and a far simpler software setup, and the latter has several orders of magnitude more processing power. While there is some overlap between which projects work on which device, they are really very different platforms.

If you take the skills you've learned in this article further, we'd love to hear from you. Every Monday we take to social media to find interesting projects, so if you're posting pictures at the start of the week, tag #MakerMonday. 🍷

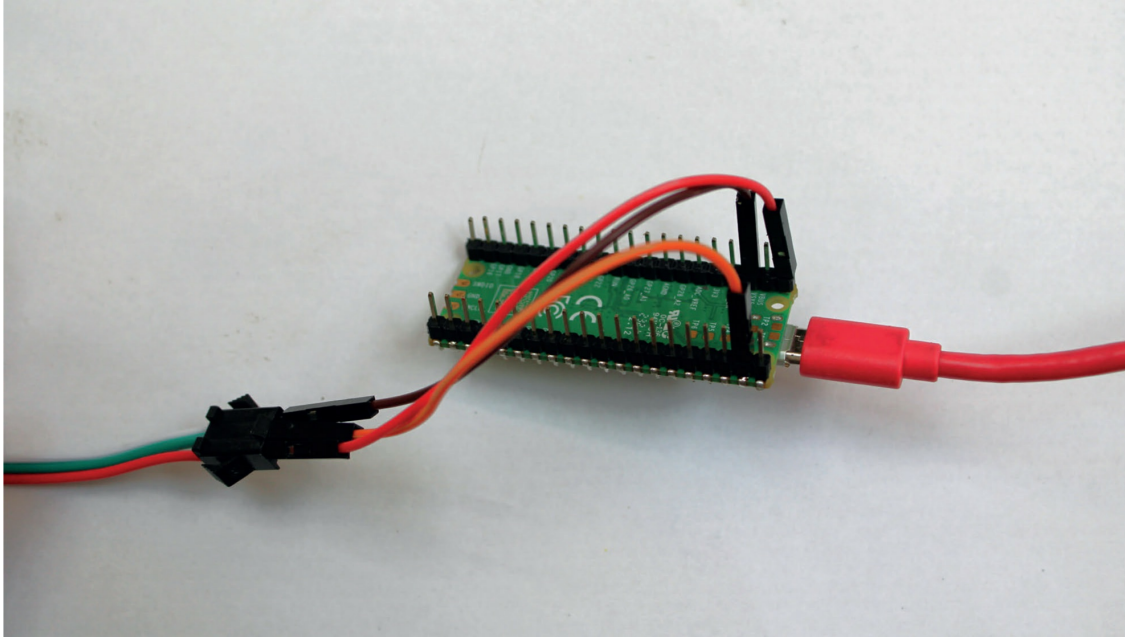
Voltage problems

WS2812B LEDs require the digital signal voltage to be at least 70% of the supply voltage. This means that it should be at least 3.5V. Pico and Raspberry Pi operate at 3.3V. This is so close that almost all the time you'll be fine connecting the data-in line to the GPIO pin. Just occasionally, you'll get a strip that behaves strangely.

You can do one of two things. You can cross your fingers and hope for the best (and this is what most people do) or you can correct the problem, which typically involves using level shifters. We almost always just use a 3.3V data signal and hope for the best, unless we're creating a project that has to work reliably, especially in adverse conditions (such as outside in the rain).

Colour order

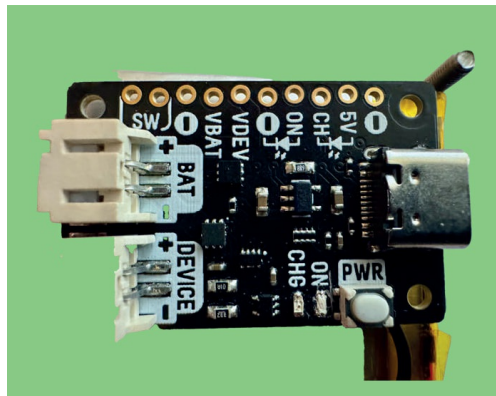
The WS2812B protocol describes how to send data to the LEDs, but not which bit of data corresponds to which colour, so you may find that the colours aren't right on your LEDs. You can configure which bit of data corresponds to which colour by changing the code.



If your LEDs come with a three-pin connection, you can poke pin headers into them to connect them to Pico or Raspberry Pi

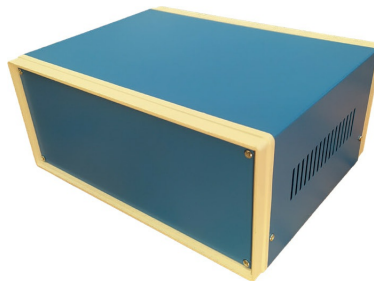
Weekend projects

When you've got a bit more time to yourself and can spend longer in the shed/garage/workshop, these brilliant builds will keep you occupied and expand your making skills



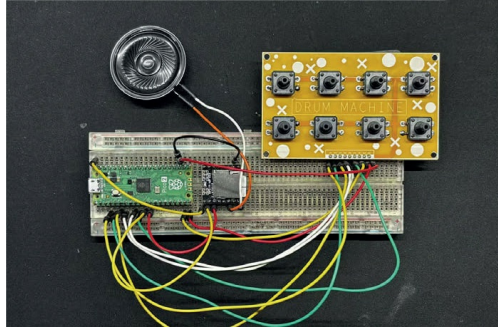
Smart CO2 monitor

Keep an eye on your environment with this intelligent sensing solution



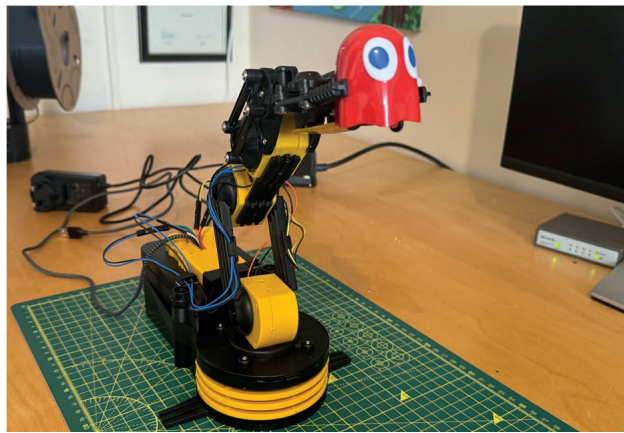
Electronics enclosures

Keep your electronics projects safe from the outside world



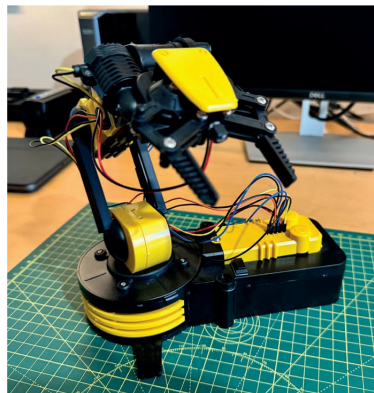
Pico drum machine

[Generate electronic repetitive beats on affordable hardware](#)



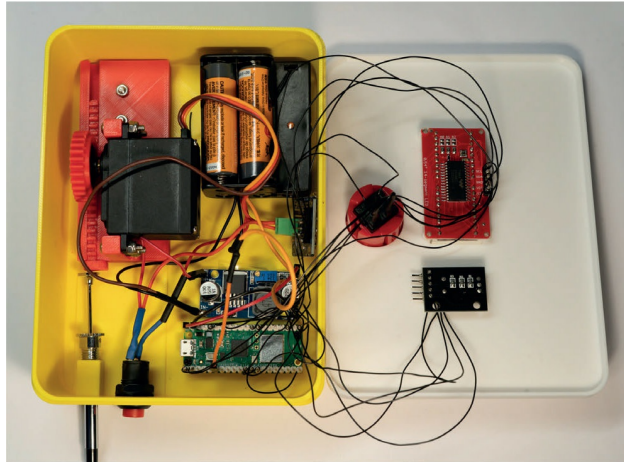
Hack a robot arm part 1

[Control an off-the-shelf robot with Raspberry Pi and Python](#)



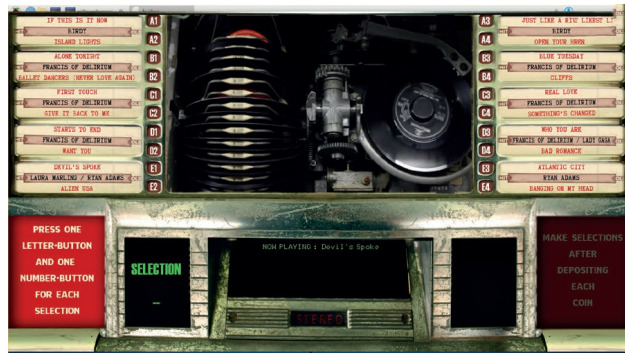
Hack a robot arm part 2

[More adventures in robotic programming](#)



Camera shutter release

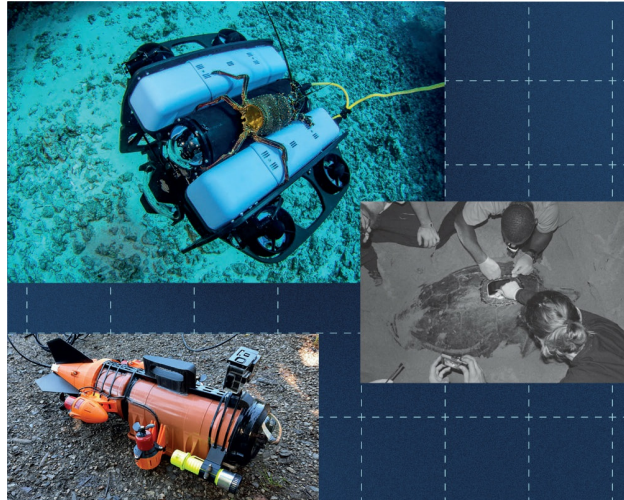
Use a Pico microcontroller to take photographs remotely



Make a digital jukebox

Play music like in the olden days

Feature



[Underwater Raspberry Pi](#)

[Explore the incredible projects that makers have sent underwater](#)

Feature



[Animate a toy skeleton](#)

[Add sensor input, stepper motors and Raspberry Pi to bring a cheap spooky skeleton to life](#)

Build a smart CO₂ sensor

Is your environment affecting your alertness? Become air aware with a Raspberry Pi-powered CO₂ sensor



Maker

PJ Evans

PJ is a writer, software engineer and tinkerer. Working at home for so long, he's learnt to open a window now and again.



mastodon.social/@mrpjevas

YOU'LL NEED

- Badger 2040 W
rpimag.co/badger2040w
- SCD41 CO2 sensor
rpimag.co/scd41
- LiPo Amigo
rpimag.co/lipoamigo
- 1200 mAh 3.7 V LiPo battery
rpimag.co/1200lipo
- 4-pin JST-SH cable (50 mm)
rpimag.co/jstsh

There's something in the air. Well actually: there's lots of things in the air, not just good old oxygen. We exhale carbon dioxide (CO₂) and that means that in an enclosed space containing one or more people, the amount of CO₂ can increase unless there's good ventilation. When CO₂ goes over 1000 ppm (parts per million) in a room, it can start to have consequences for your clarity of mind. At 2000 your cognitive ability can become seriously impaired. Do you know if your room or workplace is giving you brain fog? Let's build a monitor and find out!

01. Burn your badger!

Please leave any black and white furry mammals alone. What we mean is make sure your Badger 2040 W is ready for use. This groovy little Raspberry Pi Pico W and e-ink screen combo was intended as an interactive name badge, but it's also great for making a tiny display. We need to be sure that it's running the Pimoroni flavour of MicroPython so it has all the libraries it needs to communicate with the screen and buttons. It should come with everything you need, but if you're looking for a fresh start, follow the instructions to install Badger OS here: rpimag.co/badger2040git. On restart, you should see a menu displayed. You're good to go.

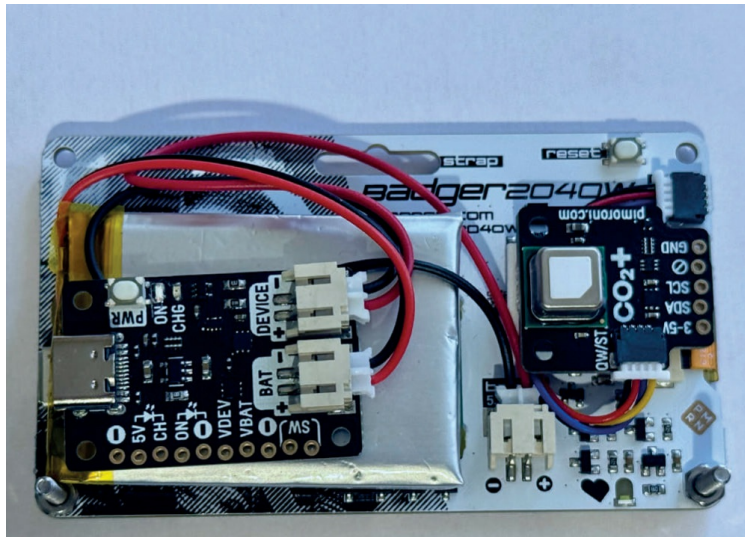
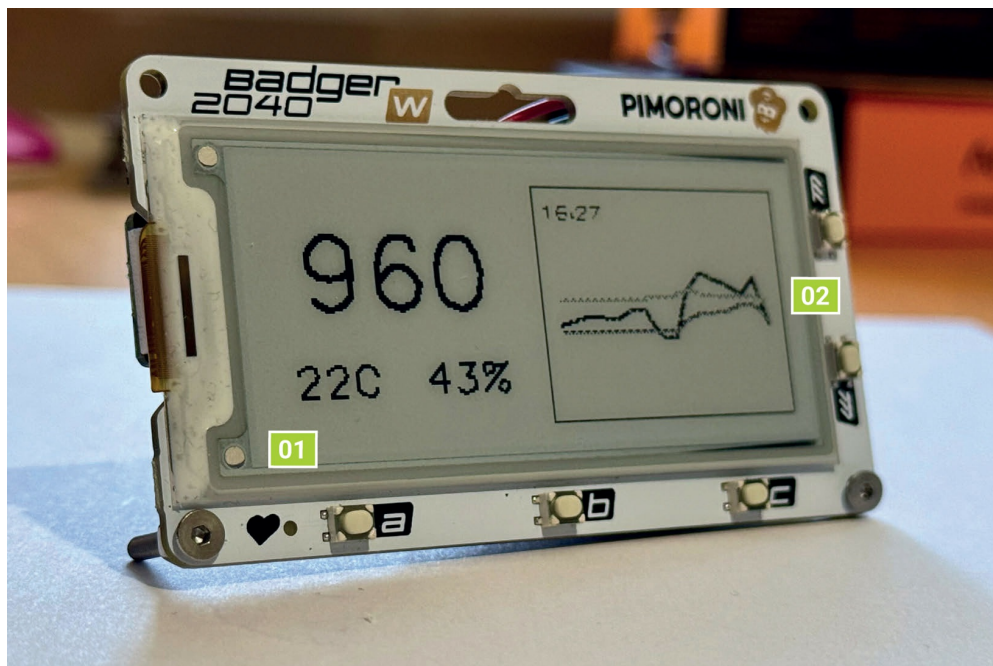


Figure 1: Through the use of QT connectors and some sticky pads, the project is easy to assemble



01. The easy-to-read e-ink screen gives clear readouts

02. Hiding behind the screen are the sensor, battery, and Raspberry Pi Pico W

02. Set up your environment

We're going to use Thonny to upload code to the Badger. Thonny is installed by default in Raspberry Pi OS (thonny.org), so launch it. Connect the mounted Pico W to your computer using a USB cable. At the bottom-right of the screen you will see 'Local Python'. Click on the text and you should be able to select your Pico W. When you select it, the Shell window should display a welcome message and the familiar '>>>' of Python. If not, you may need serial drivers for your computer to allow the Pico W to communicate. There's more help here: rpimag.co/badger2040starter.

03. Text on your badger!

Again, please do not take any kind of marker to our furry friends. Let's test the screen drivers are working correctly. From the prompt in the Shell, type in the following code:

```
import badger2040
badger = badger2040.Badger2040()
badger.set_pen(15)
badger.text("Hello Badger", 20, 20)
badger.update()
```

You should get a black screen and 'Hello Badger' in white. We've now proven that everything is working and can move on to the CO2 sensor.

04. Plug in your CO2 sensor

Disconnect your Raspberry Pi Pico W from the computer and make sure it has no other form of power to it. Carefully (static strap recommended!) remove the CO2 sensor from its packaging. You'll see a connector on the side marked 'QW\ST'. Connect your 50 mm QT cable. Make sure it's the right way up – you shouldn't need to force it in. Now find the QW\ST connector on the Badger, next to the reset button. Again, carefully insert the connector. Double-check everything before reconnecting the Badger to any kind of power.

Is your room giving you brain fog?

05 Testing the sensor

Before we start sticking anything to anything else, let's check the sensor is working as expected. Connect your Badger back to the computer and make sure you can get to the prompt in the Shell windows as before. Now create a new file on the Badger by right-clicking in the 'Raspberry Pi Pico' file list and select 'New file'. Save it as **sensor_check.py** and add the code in the listing (overleaf). Save again then press the 'Play' button in the top menu bar. Watch the Shell window. After a few seconds, readings of CO2, temperature, and humidity should appear.



Warning!

LiPo batteries

LiPo batteries must be treated with respect and charged/ discharged within the parameters listed in their datasheet. You should never dent, bend, puncture, crush, or otherwise damage them. You should only consider these kinds of batteries for your projects if you've done your research and understand the risks.

rpimag.co/batterysafety

06. Get sleepy

One of the many great features of the Pico W is its deep sleep mode. When activated, the system will shut down except for a very low-powered timer that will tick over, barely consuming power, until it's time to start up again. The result, for our project, is that we can extend battery time from hours to weeks by only consuming power when reading the sensor and updating the display. Try this out by adding:

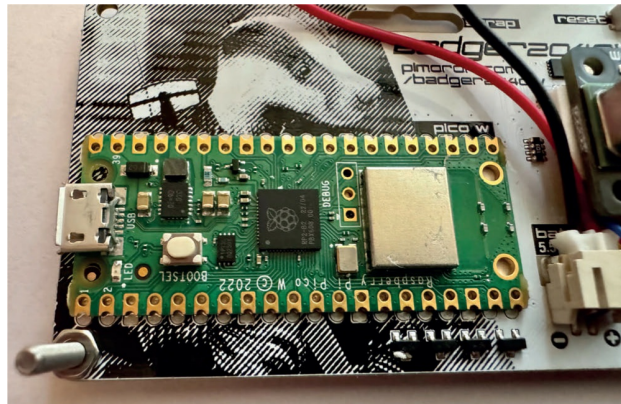
```
import badger2040
```

...at the top of the file and...

```
badger2040.sleep_for(1)
```

...after the print statement.

Now, your Pico W will measure the CO2 once a minute and snooze in-between.



The flush-mounted Raspberry Pi Pico W sits behind the battery

07. Let's get pretty

Now we've tested the screen, the sensor and deep sleep, let's put them all together. The code is a little long, so we've uploaded it to rpimag.co/badgerwatchgit where you can download it:

```
git clone https://github.com/mrpjevans/badgerwatch
```


The code will take a measurement from the sensors and update the display with the readings. As a bonus, it'll also draw a simple graph so you can track changes over time. It will then sleep for five minutes before reading again. Create a file called **co2.py** in the root of your Pico W file system and paste the code in. Running it in Thonny should update the display.

08. Bringing it online

If you've got a Badger 2040 with the W at the end, it means you have a Wi-Fi-capable Pico (if not, skip the next two steps). So, let's take our sensor one step further and connect it to your local network. For this first example, we'll just add a timestamp by using your local time server (NTP) to get the current time. This script connects to your local Wi-Fi, then queries the NTP server. Once it has the time, it adds it to the graph. Copy the file **co2_wifi.py** to your Badger. Find the file **WIFI_CONFIG.py** which will already be on the Badger. Open that file up and change the SSID, password, and country code to match your settings. Run **co2_wifi.py** in Thonny and check everything is working.

09. Adding MQTT

Let's make the internet connection a little more useful. In our final version of the script, we'll transmit the readings to an MQTT broker which can then be read by services such as Home Assistant. If you're new to MQTT, you'll need to set up a broker, so here's a tutorial that will guide you through it: rpimag.co/donotdisturb. From rpimag.co/badgerwatchgit, upload **co2_mqtt.py**, **umqtt_simple.py** and **MQTT_CONFIG.py** to your Badger as before. Edit **MQTT_CONFIG.py** to set your client name, broker address, and topic. Save it and then start the **co2_mqtt.py** script in Thonny. Use your favourite MQTT monitoring tool to view the result.

10. You have the power

Two jobs left: first, we need to start the script on Pico boot. In Thonny, edit the file **main.py** on the Badger. This file is always run when power is applied. Change it to say:

```
import co2_mqtt
```

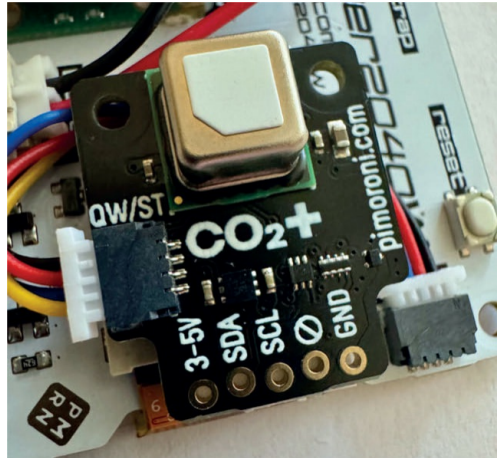
...and nothing else. (If you're not using MQTT, just change it to **co2** or **co2_time** as you want)

Next, disconnect the Badger from your computer. Connect your LiPo charging circuit to the power connector on the Badger and then your LiPo battery to the circuit where marked. Press the power button and your CO2 monitor should come to life.

11. Tidy-up time

Once you're confident everything is working, you can mount the CO₂ monitor, battery, and charger onto the back of the Badger. Sticky pads work well for this, but make sure the USB-C port of the LiPo charger is accessible. See [Figure 1](#) for an example of how to do this. If you have some long 2.5mm nuts and bolts, you can create feet by adding them to the bottom screw holes.

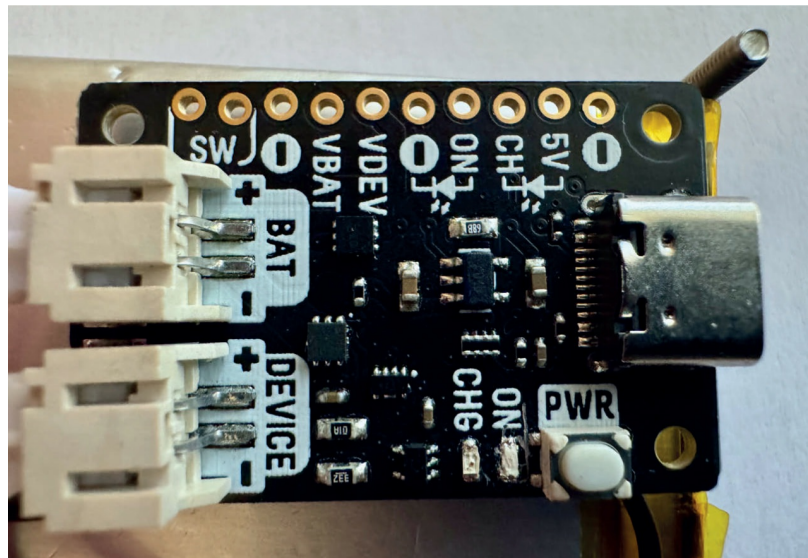
With MQTT publishing, you can expect your sensor to last three to four weeks on battery power, or even longer if not using Wi-Fi.



The all-important CO2 sensor also provides temperature and humidity measurements

12. Over to you

This project demonstrates how you can make use of a small e-ink display and take advantage of sensors to gather information and send it anywhere you want. Pimoroni and other vendors have many different types of input and output devices that use either QT or STEMMA, so it's not just CO2 you can measure. Discover what else you can build or see what you can do with a stream of MQTT data. Most of all, have fun with it and hopefully you'll know when that window needs opening for a bit. ▣



This nifty circuit safely charges your battery

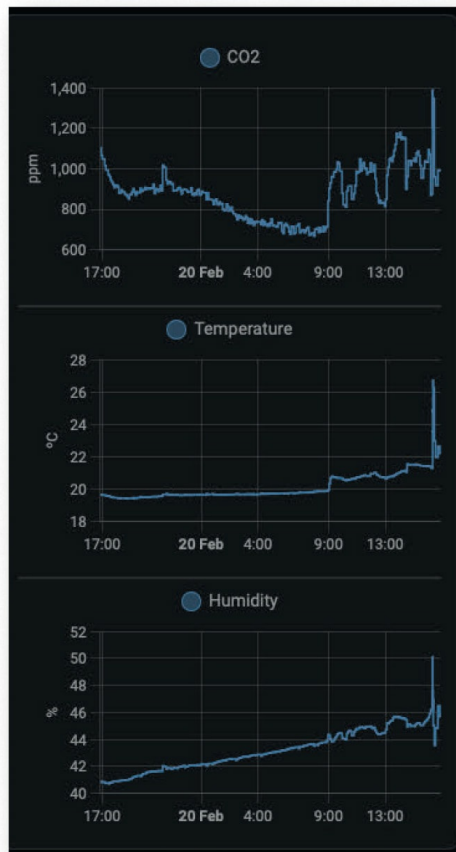
Use a good sensor

Beware of cheaper CO2 sensors that just measure particles. They are nowhere near as accurate as the one we've used here.

Safe CO2 Levels

400–800	Good
800–1200	Fair
1200–1800	Poor
1800–2000+	Bad

More info: rpimag.co/usingco2monitors



Use the Pico W's Wi-Fi capability to chart levels in Home Assistant

**DOWNLOAD
THE FULL CODE:**



rpimag.co/sensorcheckpy

sensor_check.py

> Language: **Python**

```
001. import breakout_scd41
002. from pimoroni_i2c import PimoroniI2C
003. import time
004.
005. BREAKOUT_GARDEN_I2C_PINS = {"sda": 4, "scl":5}
006. i2c = PimoroniI2C(**BREAKOUT_GARDEN_I2C_PINS)
007.
008. breakout_scd41.init(i2c)
009. breakout_scd41.start()
010.
011. while True:
012.     if breakout_scd41.ready():
013.         co2, temperature, humidity = breakout_scd41.measure()
014.         print(co2, temperature, humidity)
015.         time.sleep(1)
```

Electronics enclosures

Projects can live a long life on a breadboard, or as a collection of point-to-point soldering on a workbench – but there comes a time where we might need to move our creation into something more robust



Maker

Jo Hinchliffe (AKA Concretedog)

With a house and shed full of lathes, milling machines, 3D printers and more, Jo is a constant tinkerer and is passionate about making. Obsessed with rockets and robots and much more besides, he often releases designs and projects as open source.



concretedog.blogspot.com

When it comes to enclosures, it's fair to say that there is a never-ending variety of approaches out there in the world of electronics projects (Figure 1). From improvised recycled enclosures, to ordering thousands of injection moulded cases, there's lots of interesting options in between. Let's take a look at some approaches. A great starting point for when you simply want to take your project or experiment out into the world is an improvised enclosure. As an example in [Figure 2](#), we wanted to place a device outside for a while in a static location as an experiment. It didn't need to be a permanent solution, so we stuffed the device and battery inside a plastic peanut butter jar. Using a drill, we made a hole for the antenna connector. We weren't aiming to deploy this project outdoors for extended periods and could choose to avoid wet weather, so we didn't really need to waterproof the antenna connector. However, we were confident that if we misjudged the weather, it would survive a short period in some light rain. Another popular item for a cheap and somewhat weatherproof enclosure is a small Tupperware (or similar style) food storage tub; these are readily available in most supermarkets.



Figure 2: A plastic jar destined for the recycling box can make an excellent temporary enclosure and is a good reuse for plastic waste



Figure 1: A collection of different enclosure options for a variety of electronics projects



Alert!

Power Tools

Be mindful of problems caused by power tools. Be careful of strain from vibration, noise hazards and (if necessary) wear protective clothing and goggles. Do your research first!

rpimag.co/powertools

Once you start to consider items as enclosures, you will start to see ideas everywhere. Indeed, there are some classic enclosures made from packaging, none more so than the Altoids tin. This enclosure took a simple mint confectionery tin and spawned a huge movement of building projects in that form factor; in fact, there are numerous kits and products that are designed to be built into this classic tin.

Once you start to consider items as enclosures, you will start to see ideas everywhere

Drilling holes

When you get into using either commercially available cases or found items to home your projects, you will often end up drilling to create holes for panel components. A good starter tip to preserve the good appearance of your enclosure is to use masking tape on it to mark out your drill positions. Apart from not marking your enclosure directly, the masking tape provides a soft surface for your first drill or centre punch to press into, thereby reducing the risk of it bouncing across the panel surface and causing marks and scuffs. It's a good idea to abide by the old adage here when drilling: it's definitely a process in which to take your time and measure twice and drill once!

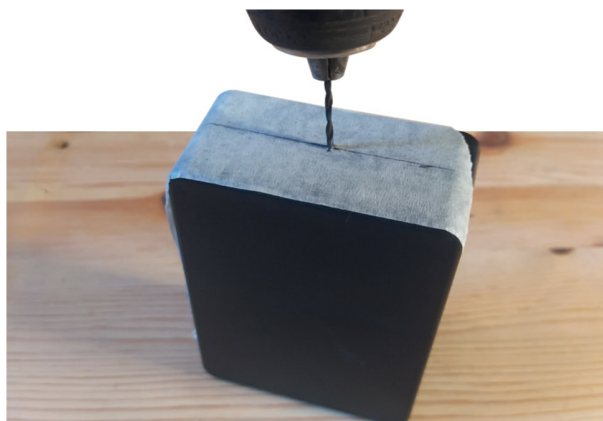




Figure 3: A collection of small commercially available plastic project boxes

Off the shelf

Moving to commercially available enclosures, we can find an amazing amount of choice available online. Probably the most common option is the simple ABS plastic box ([Figure 3](#)). Whilst perhaps not the most visually exciting, there is a wide array of sizes, shapes, colours, and features for ABS enclosures, across a large number of companies making them. Many come with screw-on lids and you can of course drill, cut, and grind shapes for components through the faces of enclosure. Some will have mounting points inside, allowing for PCBs to be slid into position, and some will have screw mounts for specific-size PCBs to be robustly mounted. Whilst not the full level of engineering rigour, it's also common to use double-sided tape or even hot glue to mount electronics into ABS boxes and this can be a very successful approach. Some ABS boxes will feature drill-to-size cable grommets. These are rubberised stepped small panels cast into the faces of the enclosure into which you drill a slightly undersize hole for the cable you wish to send through the surface. You can then push the cable through the hole to create a reasonably waterproof ingress/egress point. It's fair to say that these types of grommets are OK, but not as robust a solution as buying grommets that are specified for the size of cable and you fit entirely into a pre-drilled hole.

Another consideration with ABS plastic enclosures is that you'll find there is a range of fastening types, from simple self-tapping screws, and bolts with captive nuts, to custom twist-to-close plastic fittings. Also, you can find enclosures with rubber seals/o-rings to create a more weatherproof seal between the enclosure and its lid. Often, at the cheaper end of cases you won't have any particular weather rating, but if you are buying from an electronics retailer you'll often see enclosures with Ingress Protection or 'IP' ratings. These give an accurate measure of two different types of potential ingress: solid objects (dust and other particulates) and water. IP ratings are supplied as the letters 'IP' followed by two digits. The first digit refers to the solid object ingress and the second the water ingress rating. So, for example, an enclosure with an IP rating of IP22 would offer less protection than one rated IP99. You can find lists of charts

explaining the IP rating numbers online; a good example can be found on the IEC website (iec.ch/ip-ratings). A common higher-end level of protection for an electronics enclosure is IP65 and, looking at the IEC site, we can see that the '6' means that the enclosure is 'dust-tight' and that the '5' means that it should endure jets of water being sprayed on it – often considered more than enough protection for an electronics project destined for the garden!

A really nice off-the-shelf solution for electronics enclosures that is pretty popular is the Hammond-style enclosure. These come in various shapes and sizes, including aluminium extruded versions as well as all manner of plastic injection-moulded.

Benchtop enclosure

Whilst we've mainly considered smaller portable electronics projects in this article, it's worth noting that there are interesting commercial options for benchtop device enclosures. This aluminium panelled case costs less than £15 online and definitely has that lab desk feel about it. It would be perfect for test equipment or perhaps for a DIY variable power supply.

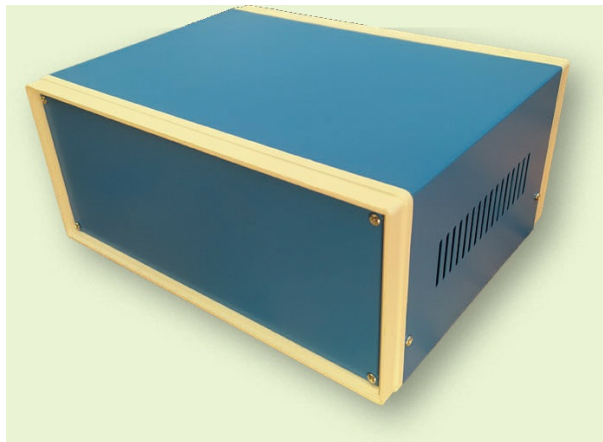


Figure 4: A small Hammond 1 593QBK enclosure which features a small panel insert, a battery compartment, and internal PCB slots

Hammond cases use removable sections which can be modified to create custom panels. The panels used can be aluminium, fibreglass or plastic, but often people opt to replace them with a custom panel, either hand-cut from material, or indeed by creating a custom PCB panel. In [Figure 4](#), the small Hammond enclosure has a heap of features to mount projects internally and a snap-fit battery compartment; it's perfect for many pocket-sized electronics devices.

Bespoke

If you have access to some makerspace type equipment, you may have lots of options for creating enclosures and front panel designs for projects. One of the most highly regarded tools for this work is the laser cutter. Of course, you can use a laser cutter to create all manner of boxes and enclosures, but perhaps even more attractive is that you can use a laser to create accurate panels for through-mount components without drilling and you can laser-engrave into the panel to label your project. Another option with this is to mix and match approaches. Using a ruler and a pair of digital callipers, it was pretty straightforward to design a front panel for a model rocket launch controller project in the wonderful free and open-source Inkscape. The panel is designed to fit on top of our simple 100×75 mm ABS enclosure ([Figure 5](#)). Whilst we then have a spare ABS lid, there's no denying that this is a neat approach for custom panels.



Figure 5: Using a laser cutter to create a front panel for an existing plastic enclosure

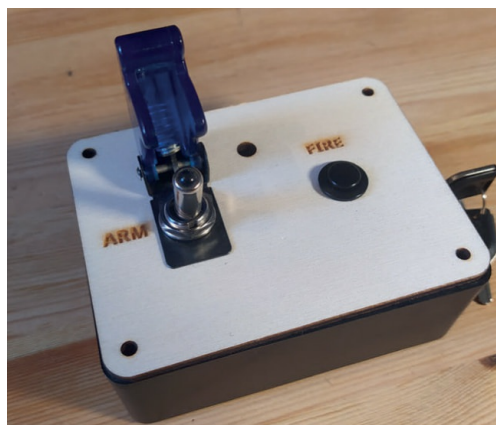


Figure 6: Using a vinyl cutter can be an attractive way to make durable decals for an enclosure panel

Again, using the 100×75mm ABS enclosure as an example of using makerspace type tools, this time in [Figure 6](#), we've used the original enclosure panel, carefully using our tape and drilling techniques to create the holes in the right places in the box and panel.

Setting out some labels for vinyl cutting is again pretty straightforward in Inkscape, although we added a little complexity by creating a circle decal to make the fire button extra-visible. Once cut on the vinyl cutter, we simply weeded away the waste vinyl and then used some transfer tape – this allows you to lift off the entire design off the backing paper, to accurately place the sticky vinyl.

Of course, it would be remiss to write a piece on electronics enclosures and DIY and not mention 3D printing. This has been a game-changer in terms of being able, with some CAD knowledge, to create accurate custom enclosures for projects. However, although we definitely think people should develop their CAD skills, it's also worth checking out what designs exist for projects and enclosures on sites such as Printables, GrabCAD, Thingiverse, and more. Many established electronics projects will have predesigned enclosures, saving you some time and effort.

A nice option if you have, or are looking to develop, some CAD skills for both mechanical design and for electronic PCB design is to perhaps emulate a Hammond-style case. Using KiCad, it's pretty easy to create panels for project enclosures and you can use the silkscreen tools to add text or graphical items to label parts of your design. We opted, without a specific project in mind, to try and create a PCB panel that could be used for creating small panels in future projects. We set up a design and, for visual interest, made the design hexagonal; in fact, it's the same dimensions as hexbin-compatible stickers at 50.8mm along its longest axis. Into the hexagonal PCB we added some through-hole pads for 5mm LEDs, as an LED is a common addition to a panel ([Figure 7](#)). We then added five small uncoated holes through the board, with the idea that these could be drilled out to larger sizes to accommodate other devices like switches, buttons, potentiometers, and more. Rounding out the design, we stuck a logo and email address on the reverse of the panel, meaning that these also have a dual function as a cheap PCB give-away business card.

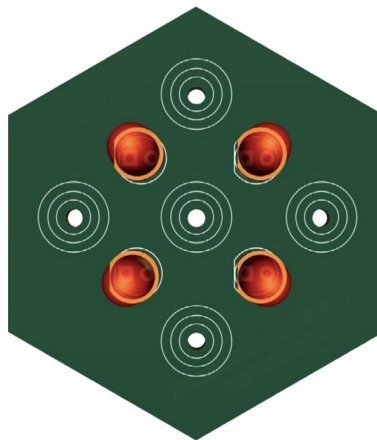


Figure 7: Creating a small PCB in KiCad that can act as a front and rear panel in DIY enclosures



Figure 9: A completed DIY enclosure with PCB front and rear panels; it would be simple to extend the case length in FreeCAD for larger projects

Made to measure

Moving to the enclosure, we fired up the excellent FreeCAD. Our approach here was to model an entire hexagonal tube part, modelling in a groove at each end that would accommodate the PCB edges ([Figure 8](#)). Once this basic geometry was created, we then used a large rectangle drawn over exactly half of the design and used a pocket operation to cut away half the hexagon. As a simple but effective closure, we've added a hole on one side and on the other a small internal rectangle piece with a hole in it that will align with the outer wall hole. The nice thing about this approach is that both sides of the enclosure are identical. 3D printing is also pretty straightforward as if you place the half hexagon on a flat surface on the build plate of the 3D printer slicer, the rest of the object's walls will sit at printable angles with no supports needed.

Having printed a pair of enclosure halves, once assembled the whole unit is pretty rigid. With a front and a rear PCB panel combined with the two M2 self-tapping bolts, there is no skew or twisting in the design. Whilst not water- or weatherproof, it would certainly make a decent indoor enclosure that's simple to extend and reprint for different lengths. Dare we say it, we also think it looks pretty cool ([Figure 9](#)).

Hopefully, you've found some hints and tips in this article interesting and if nothing else perhaps it's inspired you to box up some of those long-term breadboard projects we all have! 🟢

QUICK TIP

If you are looking for a primer on KiCad, that takes you all the way to having full PCB assembly skills, you might want to check out our *Design An RP2040 Board With KiCad* book: rpimag.co/kicad2040.

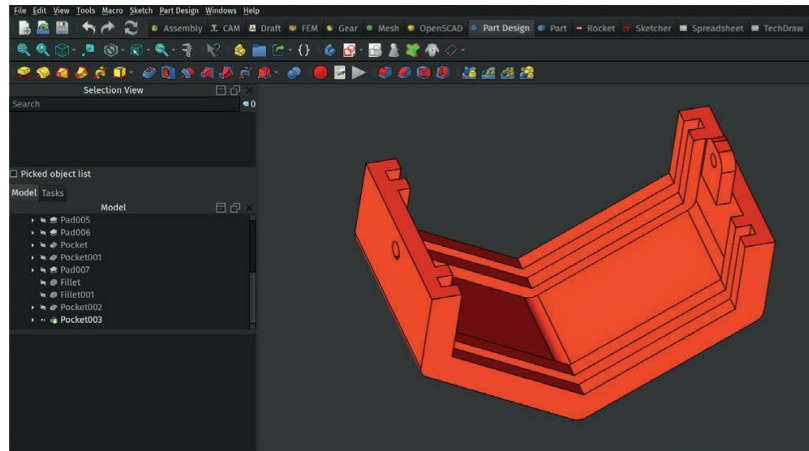
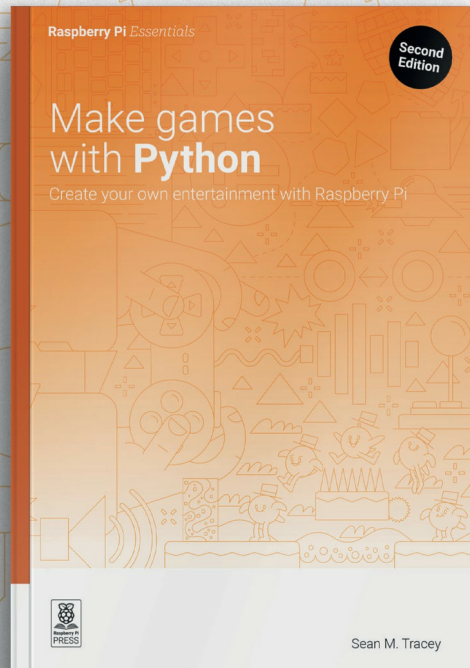


Figure 8: Designing a hexagonal enclosure in FreeCAD



While millions of us enjoy nothing more than spending hours racking up high scores on our favourite video games, too few are exposed to an even more gratifying way to spend time — making them.

This book teaches Python and Pygame development, helping you to understand the games you play and create almost anything your imagination can come up with.

■ ***As you work your way up to creating your own shoot-'em-up game, you'll learn how to:***

- *Create shapes and paths*
- *Move sprites and detect collisions*
- *Handle keyboard, mouse, and gamepad input*
- *Add sound and music*
- *Simulate physics and forces*

BUY ONLINE: rpimag.co/makegamesbook

Pico drum machine

Play drum samples at the press of a button



Maker

Phil King

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



philkingeditor.com

YOU'LL NEED

- Raspberry Pi Pico (any model) with GPIO headers soldered on
- DF Mini Player
- microSD card
- Mini speaker (up to 3W)
- 2 × half-size breadboards (or one full-size)
- 6 × momentary push buttons
- 19 × pin-to-pin jumper wires
- Drum samples

While the first ever drum machine is considered to be the Rhythmicon, developed by Leon Theremin (yes, he invented that instrument too) in the early 1930s, digital drum machines came to the fore in the 1980s with the likes of the Linn LM-1. The latter cost the equivalent of over \$19,000 today, and yet we can now create a DIY drum machine using a \$5 Raspberry Pi Pico.

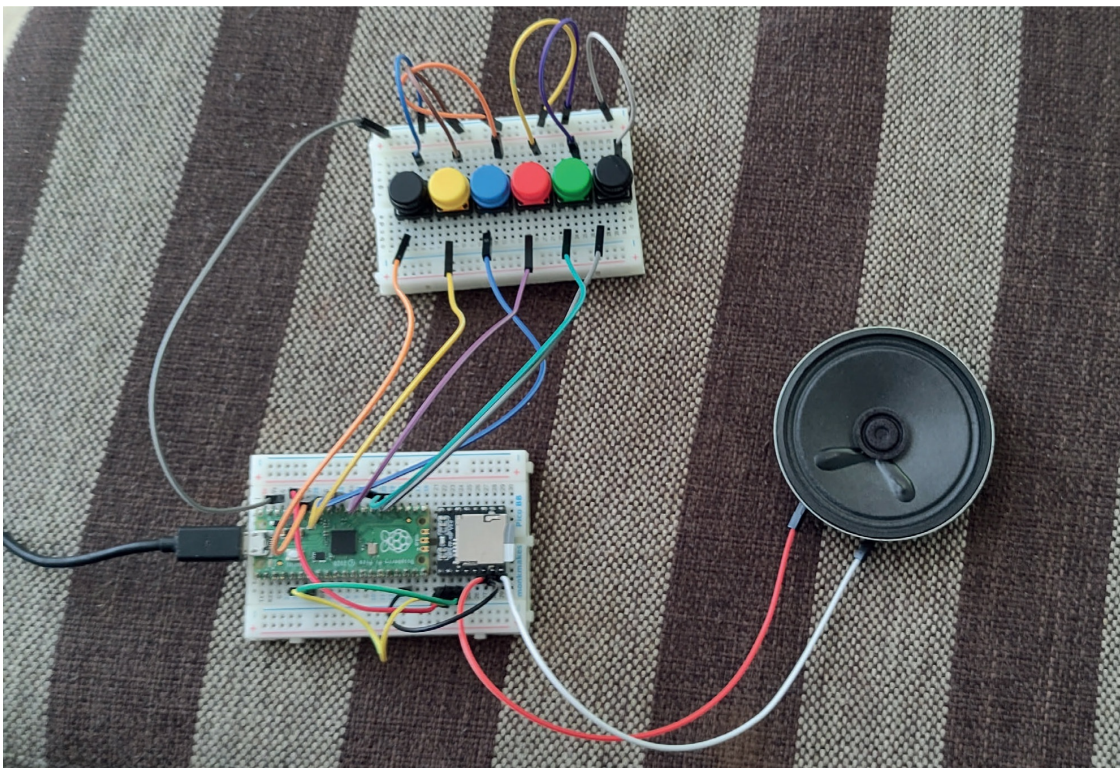
Inspired by Arnov Sharma's project on Hackster (rpimag.co/picodrummachine), your author decided to try to create a Pico-powered drum machine using a DF Mini Player to play the drum samples. While Arnov's version uses a custom-created PCB for the buttons, we decided to keep it simple with a breadboard-based design, the downside being a spaghetti of jumper wires to connect everything up. Still, the principle is the same: you press different push buttons to trigger different drum samples played by the DF Mini Player, outputting audio to a mini speaker.

Building the circuit

To make it easier to wire the buttons to Raspberry Pi Pico, we opted to put the buttons on a second half-size breadboard, as you can see in the [Figure 1](#) wiring diagram. Each four-legged button spans the division in the middle of the breadboard, with the pins on one side connected to a ground rail. The other side of each button is connected to a GPIO pin on Pico – we used GPIO 28, 27, 26, 21, 20, and 19 – pulled up (in the code). So when you press the button, the GPIO pin is pulled low, and the code senses this and triggers the DF Mini Player to play the relevant sample – more on that later.

The DF Mini Player is placed on the same breadboard as Pico, connected via UART RX and TX pins, along with 3.3V power and ground. We use its speaker output pins to connect to a mini speaker (ours was 2W, 8Ω).

To test our drum samples were playing correctly, we created a program to play them in sequence



We used two breadboards for our Pico drum machine, although the wiring is still a bit spaghetti-like

Warning!

Hot solder

Soldering irons get very hot, and stay hot for a long time after they're unplugged. Make sure that you put the iron in the stand when you're not using it and don't touch the metal parts – even after it's unplugged.

rpimag.co/soldering

To play some sounds, we need some drum samples. There are lots of free, open-source ones available online. We got ours from here: rpimag.co/drumsamplesgh. We found that some of the samples were a little long – and the DF Mini Player can only play one file at a time – so we opted to edit them in Audacity (available from audacityteam.org). If, after trimming the end of a sample, you find it ends too abruptly, you can always apply a fade-out effect to the end.

The microSD card must be formatted as FAT32, so we erased it in Raspberry Pi Imager: select Choose OS > Erase, then Choose Storage and select the card. In addition, the files must be named 0001, 0002, 0003, etc., with the relevant suffix – you can use MP3 or WAV files. As we used the latter, ours were named **0001.wav, 0002.wav... 0006.wav**. We found that it doesn't matter whether you put them in a folder or not.

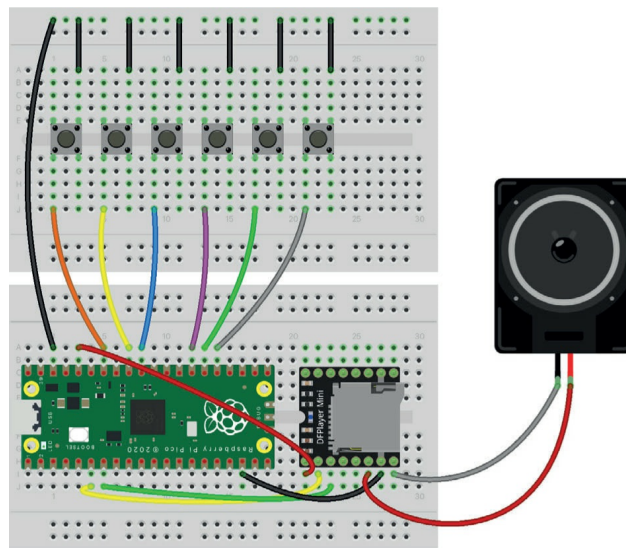
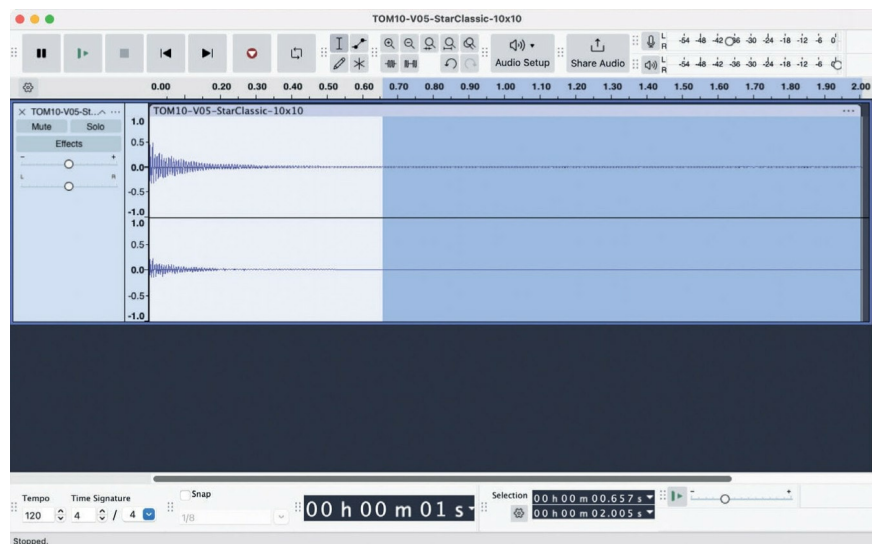


Figure 1: The wiring diagram for the Pico drum machine



We edited our drum samples in Audacity to reduce the length of some of the longer ones

Coding it

Unlike Arnov, who programmed his drum machine in C, we opted to use MicroPython. For this, we made use of Stewart Watkiss's DF Mini Player library at rpimag.co/dfminilibrarymp. Just download the **dfplayermini.py** script from there and then, using the Files tab in Thonny IDE, right-click and upload the file to your connected Pico (which already has MicroPython installed). You can then call the library in your programs.

To test our drum samples were playing correctly, we created a program (**play_drums_seq.py**) to play them in sequence:

```
from dfplayermini import DFPlayerMini
import time

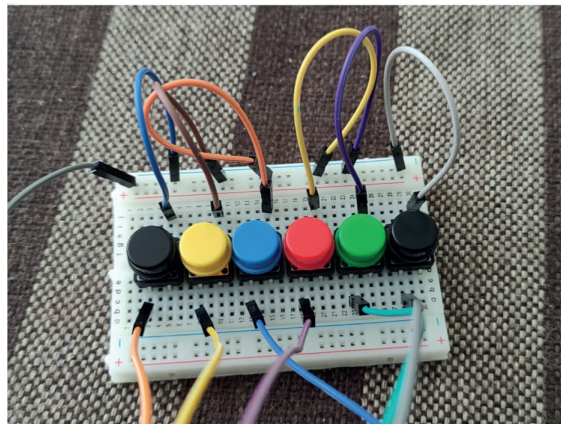
player1 = DFPlayerMini(1, 4, 5)
player1.reset()

print ("Set SD Card")
read.value = player1.select_source('sdcard')

print ("Set Volume 30")
read_value = player1.set_volume(30)

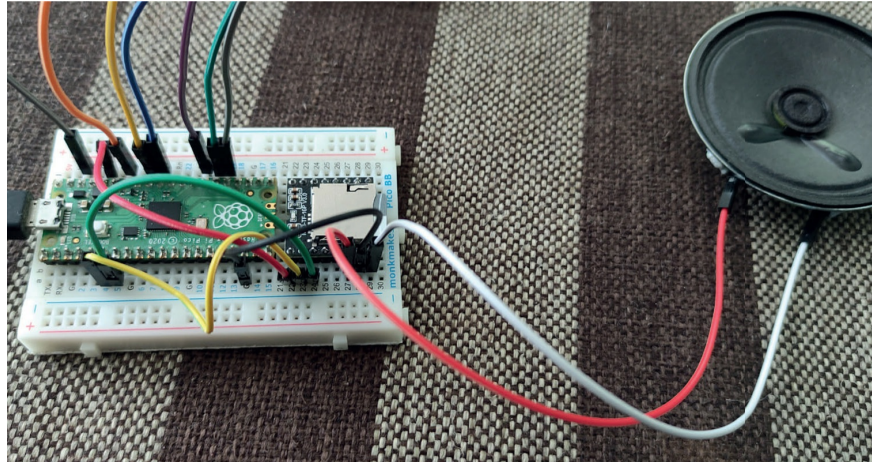
read.value = player1.query_num_files()
print (f'Num files {read.value}')

for i in range(6) :
    print("Play", i+1)
    read_value = player1.play(i+1)
    time.sleep(1)
```

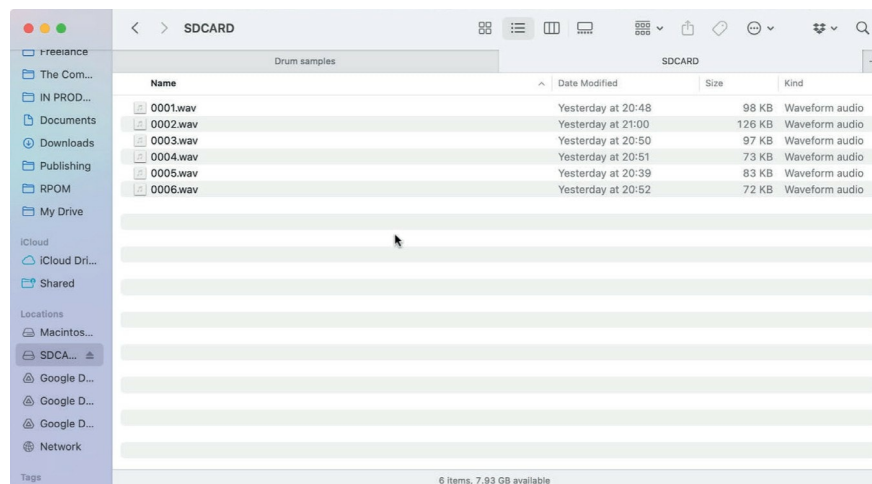


A close-up of the buttons. Each has one pin wired to the ground rail, and a pin on the other side wired to a GPIO pin on Pico

After importing the libraries, we set up a `player 1` object to work with UART 1 on GPIO pins 4 and 5. We then reset the DF Mini Player so it's ready to start communicating. We select the SD card as the audio source, then set the volume – we maxed ours up to 30. We then query the number of files on the card (it should be six), before playing each one in turn.



Along with Pico, we placed the DF Mini Player on another breadboard and connected it to a mini speaker



You need to name the files on the microSD card in (four-digit) numerical order for the DF Mini Player to recognise them

Next, we adapted this to create our main program to read the button presses and trigger the sounds accordingly:

```
from dfplayermini import DFPlayerMini
from machine import Pin
import time

player1 = DFPlayerMini(1, 4, 5)
player1.reset()

print ("Set SD Card")
read_value = player1.select_source('sdcard')

print ("Set Volume 30")
read_value = player1.set_volume(30)

read_value = player1.query_num_files()
print (f"Num files {read_value}")

# Define GPIO pins for buttons
button_pins = [28, 27, 26, 21, 20, 19]
```

```
# Initialize input pins
buttons = [Pin(pin, Pin.IN, Pin.PULL_UP) for
pin in button_pins]

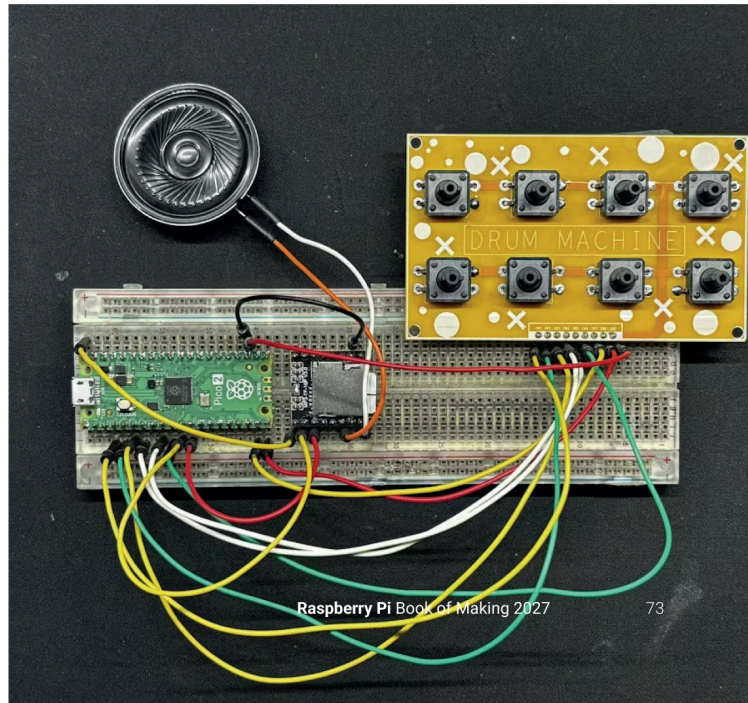
# Main loop
while True:
    for i in range(len(buttons)) :
        # Read button state
        if buttons[i].value() == 0:
            player1 .stop() # stop current sound
            read_value = player1.play(i+1)
            time.sleep(0.01) # Debounce delay
```

Speaker soldering

The SPK_1 and SPK_2 pins on the DF Mini Player should be connected to the positive and negative terminals (or wires), respectively, of your mini speaker. While you can get away with wrapping the stripped ends of the jumper wires around the terminals, it's best to solder them in place for a more secure and reliable connection. The sound from our mini speaker was a little tinny, so you might get more bass out of it by mounting it in some kind of enclosure.

Here, we add a line at the top to import the `Pin` method from the machine library. We create a list to set the GPIO pins for the buttons, then initialise them as inputs with the pin pulled up. In the main loop, we read the button state and then, after stopping any currently playing sound, play the relevant drum sample. We added a very short debounce delay to prevent a button press causing multiple triggers.

Hands up: the performance wasn't as good as we'd hoped, with a noticeable lag between pressing a button and the sound being played. As already mentioned, the DF Mini Player can only play one file at a time, which is a major limitation for a drum machine. An alternative would be to use an I2S-based audio board such as the Waveshare Pico-Audio to play the drum sounds. Still, our little Pico drum machine does work and you could use it to trigger other samples, such as spoken phrases or funny noises. Alternatively, using four of the buttons, you could try out Stewart Watkiss's MP3 player project (rpimag.co/dfminimp3player) to play songs stored on the microSD card. 🟢



This project was inspired by Arnov Sharma's Pico drum machine, for which he created a custom PCB to house the buttons

Hack a robot arm

Upcycle a toy robot arm to add Raspberry Pi smarts and internet remote control



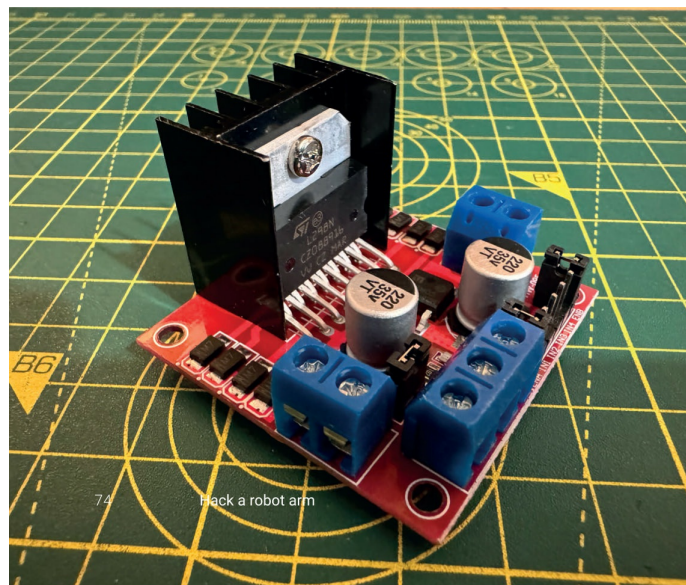
Maker

PJ Evans

PJ is a writer, software engineer, and very average roboticist. His robot arm enjoys teasing the cat.



mastodon.social/@mrpjevas



The L298N contains two H-bridge circuits on a single board

Ever buy a gadget? Does it get played with for a while then consigned to the shelf or cupboard? Upcycling old gizmos and sprinkling a little Raspberry Pi magic is one of our favourite things. Take the OWI-535 (rpimag.co/owi535) robot arm. This fun toy, with several points of articulation and a gripping claw, could be found in electronics shops under various brands. It's still widely available. Although a USB version is now on sale, we wanted to see if

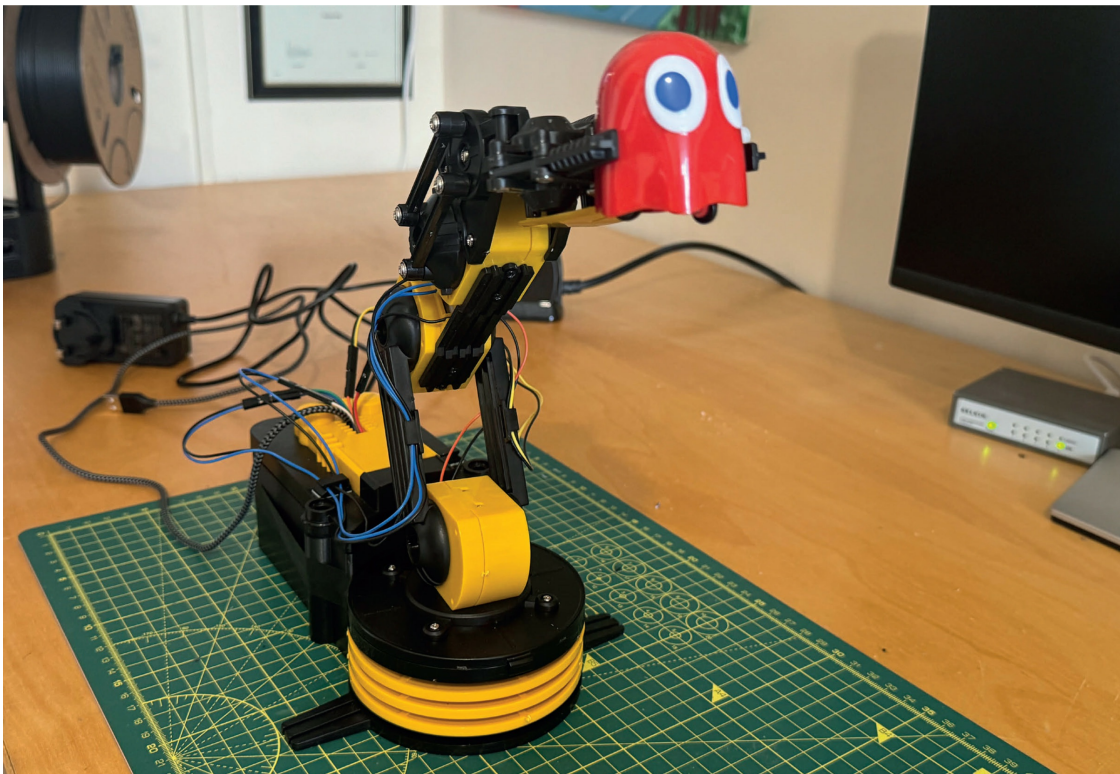
we could convert the original remote control version and make it Raspberry Pi powered. Even if you don't have this particular device, this should be a useful guide to motor control.

How the arm works

The OWI-535 robot arm is made up of five DC motors, each controlling a different point of articulation. The base rotates, there is a shoulder, elbow, and wrist, then finally the claw. Each point is bidirectional: the base can rotate either way, the joints can move back and forth, and the claw can adjust its grip. Each motor is connected to a control board by standard jumper cables (useful!) and has two wires. To control the arm, we need to be able to power the motors and also be able to reverse the DC current in order to change direction.

Raspberry Pi challenges

The use of two-wire DC motors presents us with a problem. If it were three wires, a ground and a positive per-direction, it would be straightforward, but a motor that will spin based on polarity of the current applied is not something Raspberry Pi's GPIO can handle out of the box. Also, the current required by the motors to operate is well out of the reach of a Raspberry Pi and we would start damaging things. We need an interface between the two that can help us switch the motors. Enter the humble H-bridge.



Robot! Put Blinky down right now!

H-bridge over troubled motors

The H-bridge is a very common circuit in robotics that solves our problem. It's a wiring method that uses four switches to reverse the current to a motor. See how the diagram ([Figure 1](#), overleaf) shows both states, causing the motor to spin one way or another. Using five H-bridge circuits and a separate power supply, we can safely control all the motors. We'll do this using three L298N (rpimag.co/1298n) circuit boards. These are cheap and widely available. Each one contains two H-bridge circuits onboard and they are easy to wire up.

Take your time and check everything again and again

Understanding the L298N

The L298N board can control two motors. Each connects its two wires to the board. Power is provided by an external source connected to the power terminals. We need 12V at about 1A. Such supplies are widely available. Finally, control is provided by providing a signal to each 'Enable' line (this allows us to control speed) and by sending one of two lines to 'high', controlling direction. We now have the three-wire setup we originally wanted. The control lines can be safely connected to your Raspberry Pi without risk of damage.

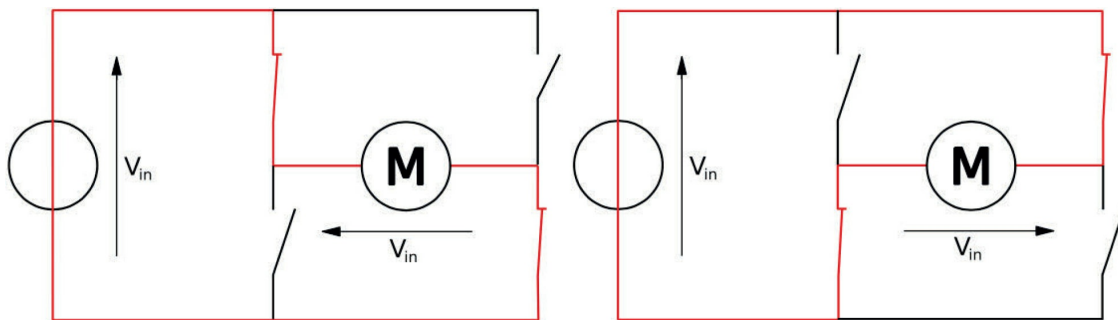


Figure 1: This shows how the H-bridge circuit switches polarity (Credit: Cyril Buttay)

Prototyping

Before doing any irreversible work on your arm, we recommend you do some experimenting first as this will help you understand how everything works. We're using a Raspberry Pi Pico W for our project. Choose one of the motors – the base is a good start. Using some jumper leads, connect its two wires to one of the motor drivers on an L298N. Hook up a 12V supply to the power terminals (but don't switch on yet) and then connect EN1 to GPIO 2, IN1 to GPIO 3, and IN2 to GPIO4. Finally, connect a wire between any GND (ground) pin on Raspberry Pi Pico to the GND terminal where you connected the power supply.

First moves

Using USB cable, hook your Raspberry Pi Pico W up to another computer that can run Thonny IDE. Ensure the Pico has the MicroPython firmware loaded (rpimag.co/micropython). Thonny should be able to see the Pico and its file system. To help us drive the motors, we need the

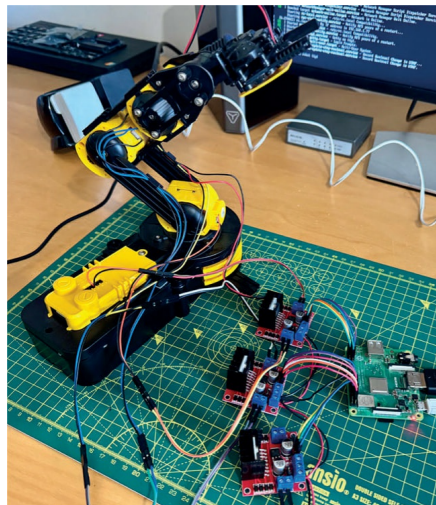
DCMotor library that you can download here: rpimag.co/dcmotorpy. Copy it to the Pico and then enter the code in the listing (or download it from rpimag.co/owi535testpy). Turn on your 12V supply and run the code. If all is well, the base should rotate one way for half a second and then the other. If it doesn't, check all your wiring carefully.

Preparing the arm

We'll come back to the code in the next part of this tutorial. For now, we're going to concentrate on making our project look great, so make sure everything is powered down. The OWI-535 comes with an elongated base designed to take four D-cell batteries. That gives us some potential room to house our electronics, but not quite enough. To make everything fit, we decided to 3D-print an extension to the housing, making it tall enough to safely contain the three L298NS and Raspberry Pi Pico. If you have access to a 3D printer, you can download the STL file here: rpimag.co/owi535stl. We then removed the circuit board and carefully removed the stands that held the batteries in place to create plenty of room. With the extra space provided by the 3D print, we are now able to wire up and install the electronics.

Full wire-up

The final part for this month requires a little patience. Following the example for one motor in [Figure 2](#), wire everything up as shown in the mapping diagram ([Figure 3](#)). We recommend a Pico with pre-soldered headers to make connections easier. Every L298N needs to be wired up to the 12V supply and ground by chaining them together, then we need to connect all the control lines to the GPIO pins of the Pico as shown. Finally, the motors need to be connected to the terminals. Use additional jumper cables between the terminals and the motors so they fit and can easily be changed around. Take your time and check everything again and again. Don't apply any power until you're sure it's OK.



Prototyping and testing is an essential part of any project

That's all for now

Next month we'll test everything out, install the circuitry, and then add software to take control of the robot arm. Even if you don't have this particular model, the same principle of driving motors using H-bridge circuits can be applied to many different projects. 🟢

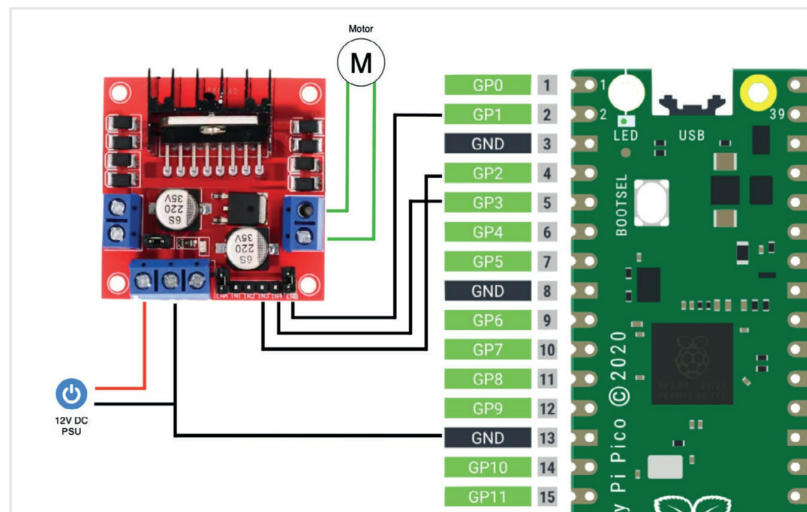


Figure 2: It's easier to show a single motor wire-up – repeat this for all five motors

Take your time and check everything again and again

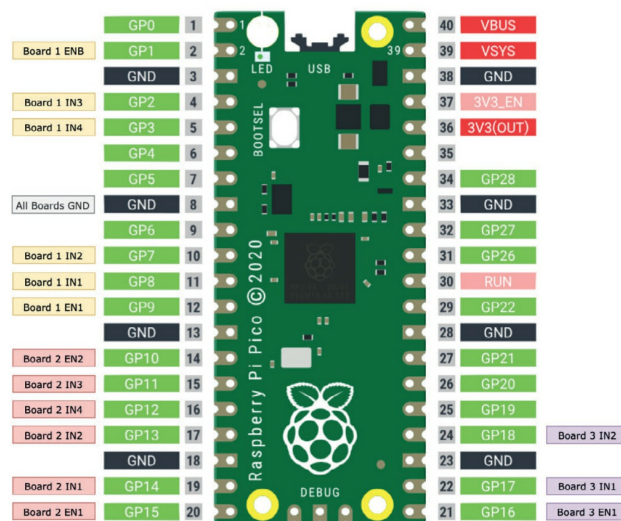


Figure 3: This maps the L298N control lines to the Raspberry Pi Pico W – start with one motor and test it

**DOWNLOAD
THE FULL CODE:**



rpimag.co/owi535testpy

owi535_test.py

> Language: MicroPython

```
001. from machine import Pin, PWM
002. from time import sleep
003. from dcmotor import DCMotor
004.
005. # Pins (Motor 1 , Motor 2, Enable)
006. motor_pins = [2,3,1]
007. frequency = 1000
008. speed = 10
009.
010. motor = DCMotor(Pin(motor_pins[0], Pin.OUT),Pin(motor_pins[1], Pin.OUT),PWM(motor_pins[2],
frequency))
011.
012. print("Forward")
013. motor.forward(speed)
014. sleep(0.5)
015.
016. print("Backward")
017. motor.backward(speed)
018. sleep(0.5)
019.
020. print("Stop")
021. motor.stop()
```

Hack a robot arm

Now the wiring is sorted, it's time to assemble our robot arm and give it some smarts



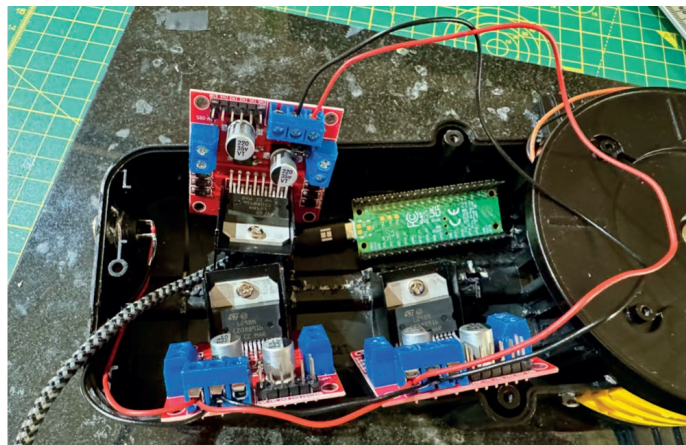
Maker

PJ Evans

PJ is a writer, software engineer, and maker. The robot arm keeps knocking his coffee over and he's worried it's doing it on purpose.



mastodon.social/@mrpjevas



Placing the circuit boards takes time and patience (and a multimeter!)

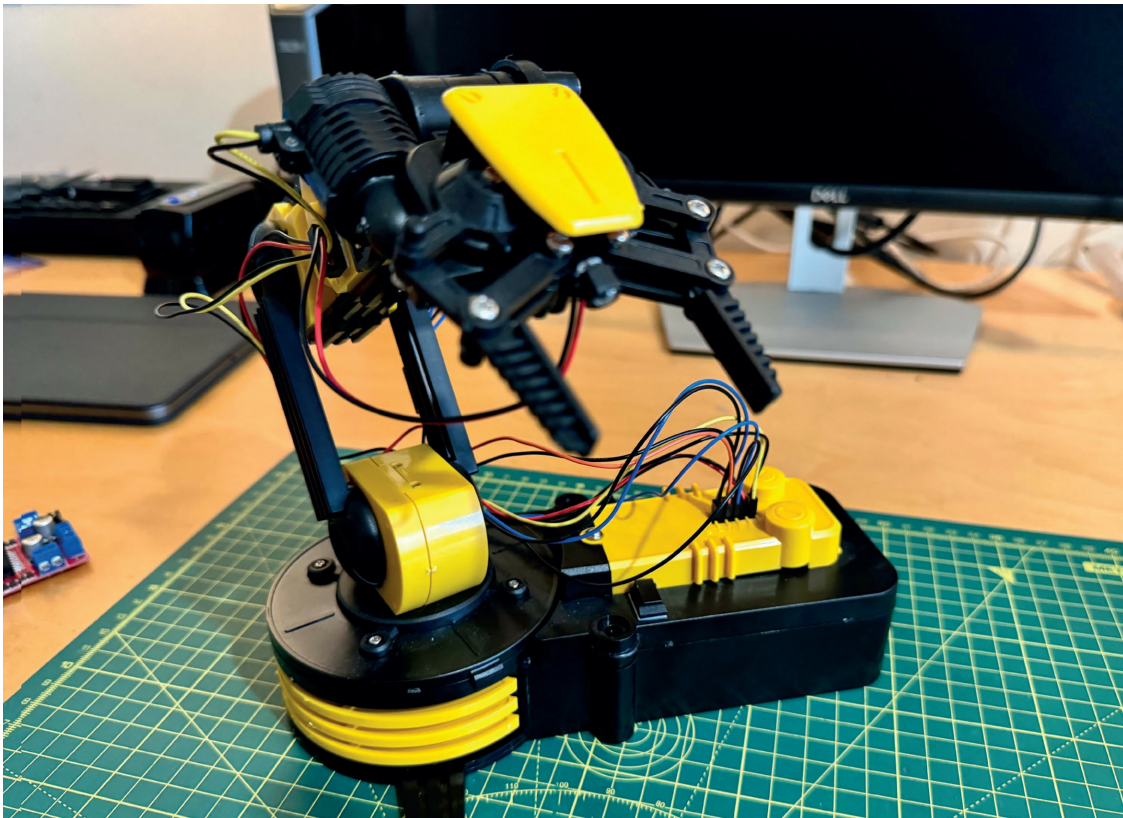
Last month we started the process of upcycling the common OWI-535 robot arm into something a bit more internet connected. There's still a little assembly to do and we're yet to tackle the software. Although this is a specific project, the examples given here can be transferred to a multitude of different robotics projects. The actual assembly of the robot arm is irrelevant, as the Raspberry Pi Pico has no idea about that; all it's doing is controlling a series of DC motors. So whatever you can think of that uses motors like these, you can automate with Raspberry Pi.

Last time on robot arm hacking

Follow the tutorial in the previous four pages (also published in Raspberry Pi Official Magazine issue 155 – rpimag.co/155) to set up your robotic arm. Hopefully, by now you've been able to wire up the five motors to your Raspberry Pi Pico W. Remember all the motors need to be connected to motor drivers on the H-bridge circuits, the Raspberry Pi Pico must be wired to the IN and EN pins (three per motor) and the 12V supply to the 12V IN on each H-bridge board. Finally, they need to share a common ground, connecting the Raspberry Pi Pico's GND pin to the GND for the PSU. Once you've checked all of this (again, and again) you're ready to continue. A multimeter is very useful here!

Check each motor

Using the MicroPython test script we used in the previous issue, change the wiring values to test each motor individually. If you followed the same wiring pattern as us, this should be as listed in the code snippet overleaf. One-by-one, update the values and run the script. Each motor should make a small back-and-forth movement. Repeat until each motor is working, and make a note of which motor you have connected to which circuit (the order is not important). Starting from the bottom, we've named them the base, shoulder, elbow, wrist, and grip.



Our goal – an upcycled web-connected robot arm

Software time

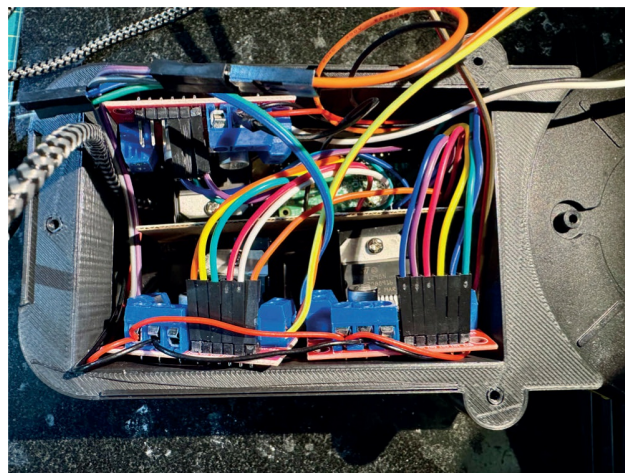
Before we assemble the arm, we should make sure the final software implementation is working. Take your time on this step, there's no rush to get all the circuits in place and it gives you space to debug both the electronics and the software (as you'll still have easy access to your Pico's USB port). In fact, there's no real need to assemble it all if you are not planning on a permanent installation: the arm will work just fine without. For now, make sure your Pico is hooked up to your computer and that the Thonny IDE can see it (see previous issue).

Just add web

The easiest way to control the arm is using a simple web server. Raspberry Pi Pico W surprisingly has enough power to not only control the arm, but to run a little web server too, so you can use any web browser or even your phone to make the arm move around. We've prepared the server for you, so download the code from GitHub (rpimag.co/owi535gpioweb), or clone the whole **owi535gpio.git** repository:

```
git clone git@github.com:mrpjevans/owi535gpio.git
```

Control your robot arm from any web browser in your home, even your phone!



Once fitted, insulate the boards with cardboard or plastic



Add a barrel connector for a neat finish

There are a few files to deal with. Firstly, our control software that runs the web server and handles the commands as they come in, the HTML template that controls the display, and the configuration file **WIFI_CONFIG.py** (you can reuse the same one). For now, you can run the web server in Thonny for testing. See the README if you need more help.

Remote control

When the server starts, you should see Pico's IP address being used in Thonny's console window. Open up a web browser on your computer and enter `http://` followed by the IP address. You'll then see a simple web interface. Make sure your 12V PSU is powered up, then try pressing the buttons. It's possible they do not match the motors you have connected. You can either swap the motor wiring about or change the wiring assignments in the code to match; it's completely up to you. If everything is working as expected, you can now control your robot arm from any web browser in your home, even your phone.

Final step

Feel free to tinker with the code. There's lots of scope for making it look better, or changing the default behaviour. You could even have a look at the included MQTT example to see how you could control the arm using an MQTT server over the internet (or even hook it up to Home Assistant!). When you've settled on an approach, the final step is to edit **main.py** and change it to read:

```
import owi535_web
```

Save **main.py** and transfer it to your Pico. Now the Pico will run the web software whenever power is applied, so it can run off a normal 5V power adapter (you will still need a 12V supply for the motors).

Tidy-up time

When you are confident that everything is working, it's time to assemble the final project. If you've 3D-printed the extension module from last issue (rpimag.co/owi535stl), you can place that on top of the main base. Remove all the circuitry in the top section to create as much space as you can. The H-bridge circuits can be carefully placed vertically in the 'well' of the base with the Pico alongside, towards the arm itself. Please take your time doing this. Throwing it all in there and jumping up and down on it will almost certainly cause a short circuit. We also recommend placing a piece of cardboard or thin plastic in the centre so the opposing circuits do not touch. For a finishing touch, drill a hole (beforehand!) in the base to mount a barrel connector for your power supply. You can now add the original top of the base, feeding the wires through the hole in the centre.

Over to you

Congratulations, you have a web-connected robot arm, but that's only the beginning! This project is ripe for improvements. Here's some things to try.

How about adding timing controls to stop the arm when it travels too far in one direction?

Better still, add microswitches that stop the arm in certain positions

You could add cameras and use machine learning to detect objects

With a more powerful Raspberry Pi, you could add full autonomous movement

Use MQTT to extend control to other devices using Home Assistant

As ever, the possibilities are endless. Have fun! 🟢



The completed base can be hooked up to the arm. Pico needs its own power

Robot Arm Controls

Base:	Forward	Reverse
Shoulder:	Forward	Reverse
Elbow:	Forward	Reverse
Wrist:	Forward	Reverse
Grip:	Forward	Reverse

STOP

A simple web interface, just waiting for customisation

**DOWNLOAD
THE FULL CODE:**



rpimag.co/owi535testpy

owi535_test_all.py

> Language: **MicroPython**

```
001. from machine import Pin, PWM
002. from time import sleep
003.
004. from dcmotor import DCMotor
005.
006. # Pins (Motor 1 , Motor 2, Enable)
007. base = [2, 3, 1]
008. shoulder = [7, 8, 9]
009. elbow = [11, 12, 10]
010. wrist = [13, 14, 15]
011. grip = [17, 18, 16]
012.
013. # Choose the motor to test
014. motor_pins = base
015.
016. frequency = 1000
017. speed = 10
018.
019. motor = DCMotor(Pin(motor_pins[0], Pin.OUT), Pin(motor_pins[1], Pin.OUT), PWM(motor_pins[2],
frequency))
020.
021. print("Forward")
022. motor.forward(speed)
023. sleep(0.5)
024.
025. print("Backward")
026. motor.backward(speed)
027. sleep(0.5)
028.
029. print("Stop")
030. motor.stop()
```

Pico-controlled cable release

Make an old camera shutter computer controlled using a Raspberry Pi Pico and a slightly beefy servo



Maker

Rob Miles

Rob has been playing with software and hardware since almost before there was software and hardware.



robmiles.com



Figure 2: This cable release has been half-pressed to show how it works

YOU'LL NEED

- Raspberry Pi Pico or Pico W (version 1 or 2)
- MG995 micro digital servo
- 0.54-inch 14-segment red and orange LED display (HT16K33)
- CJMCU-219 bidirectional DC current sensor (INA219)
- Large push-button (preferably a nice colour)
- Connecting wire (search for '30 AWG wire wrap') which needs a wire wrap tool (search for 'wire wrap tool'). You can use Dupont cables if you would prefer
- A box. There is a 3D printable design, or you can put the release in any box you fancy
- Screws. You'll need some screws sized M2 24 mm in length to fix things to the case (search for 'laptop screws')
- Cyanoacrylate (super) glue to stick the 3D-printed gear to a servo horn

In this article, we'll build a remote release for use with older cameras which have a cable release fitting. This will allow accurate long exposures and provide a self-timer you can use to get yourself into the picture. This is also a good way to explore microcontroller-triggered mechanical actuation. We'll also discover how you can use current monitoring to detect when your device is in the process of destroying itself, which is always something that is good to know. You can find all the software and build instructions here: rpimag.co/picocamerareleasegh

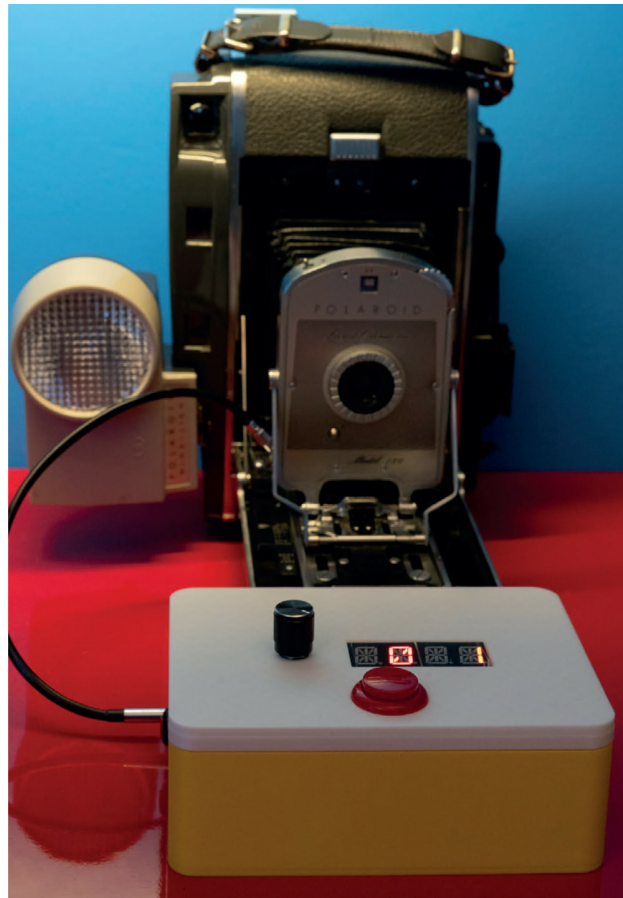


Figure 1: The camera is a Polaroid Land camera which has been converted to use modern instant film

[Figure 1](#) shows the remote release connected to a camera. Pressing the red button at the front will trigger the camera shutter for a set time. You turn the knob to select the exposure time. If you hold the knob down and turn it, you select the selftimer delay in seconds. The display shows the delay in red and the exposure in orange. In [Figure 1](#) the exposure time is set to 1 second and the self-timer delay is 0. The box contains a Picocontrolled servo which presses the cable release you can see coming out of the left-hand side of the box. Let's start with a look at what a cable release does.

[Figure 2](#) shows a camera cable release. If you press the plunger at the top, a metal pin is pushed out of the fitting at the bottom. The fitting is screwed into the camera and the pin triggers the shutter. Using a cable release is a good way to reduce blurring in a photograph, especially if you are using a long exposure. Triggering an exposure with a button on the camera can cause camera shake, but a cable release isolates the movement, keeping the camera steady. Cable

releases of this design this have been in use for over 100 years.

Figure 3 shows the first version of the shutter actuator. It is based on a design you can find here: rpimag.co/servoactuator. There are two versions of the design for different-sized servos. The author started with a micro servo that he had to hand. When the servo turns, the rack is moved in and out. The end of the rack presses the button on the cable release. This worked very well right up to the point where the author tried to use it to drive his camera. At that point, the servo made some very unhappy noises and nothing moved. The camera button was too hard for the servo to press.

It turns out that the servo is strong enough to damage itself

QUICK TIP

Some modern cable releases have a flexible outer sheath which stretches when the plunger is pressed, and the pin tries to push the mechanism in the camera. These might not work with older cameras which may have shutters that are hard to press. If you have an old camera, it is worth seeking out an older camera release (preferably with a metal sheath) to use with it.

Shutter cocking blues

When a film camera takes a picture, tiny blades in the shutter move to open a hole to let light onto the film. At the end of the exposure, the blades are closed. In older cameras, the blades are moved by a spring-loaded mechanism. Before the shutter can be fired, a spring must first be wound up, or 'cocked'. Usually, this is performed before taking the photo; sometimes, cocking the shutter takes place as part of the action of winding the film onto the next frame. Unfortunately, the Polaroid camera used in this project has a mechanism that tensions the shutter as the shutter button is being pressed. This makes the shutter simple to make and less prone to wear, but it also means that you must press the button quite hard take a picture. Unfortunately, a micro-servo doesn't provide enough torque. It was time to level up the servo.

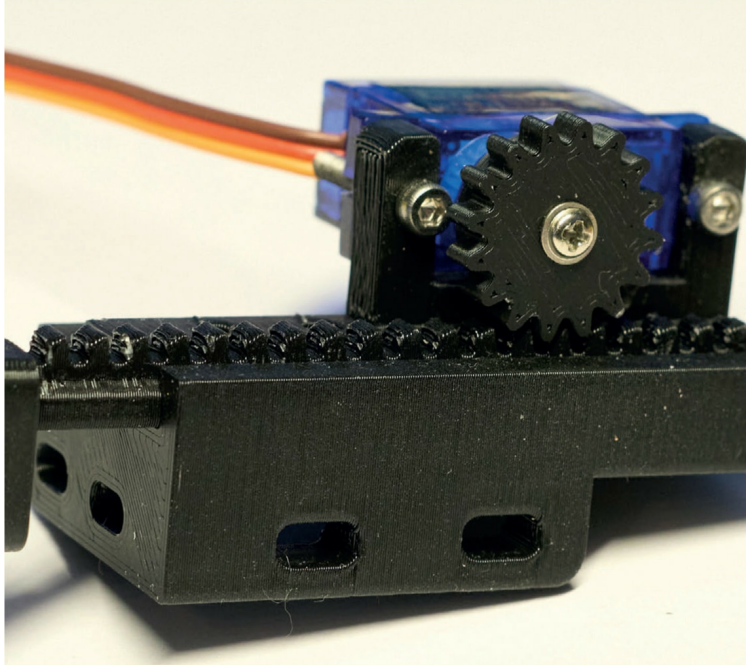
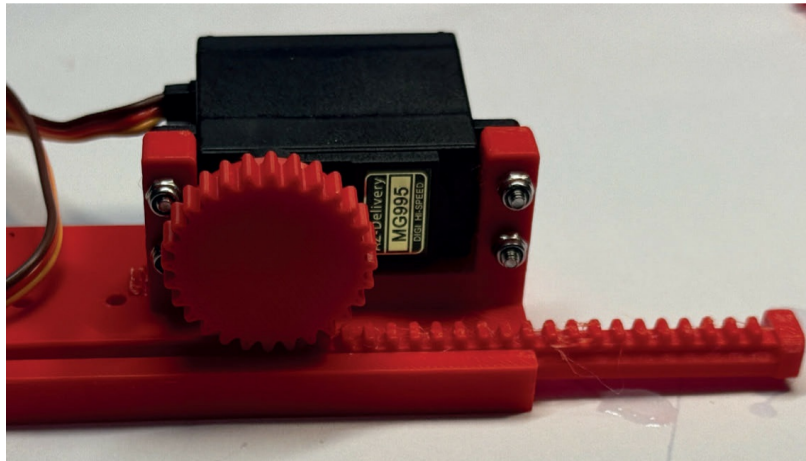


Figure 3: The gear is glued onto the servo output



The servos can be controlled by the same software, but the large servo needs a higher voltage power supply

[Figure 4](#) shows the complete circuit diagram for the remote release. It is powered by five AA batteries that provide 7.5 volts for the servo. The 7.5V is reduced to 5V by a DC-DC voltage converter to power the Raspberry Pi Pico. A four-character text display and a rotary encoder provide the user interface. The encoder contains a push switch that is triggered when the user pushes in the control. The C JMCU-219 sensor reads the current flowing through the servo. The hardware is all controlled by a Circuitpython program.

Classy photography

A class called `CableReleaseController` manages the operation of the device. The code below constructs an instance of this class (called `cable`) with the default hardware configuration. If you use different ports for your build, you can change this to match your hardware.

```

cable = CableReleaseController(
    i2c_sda=board.GP0, i2c_scl = board.GP1,
    encoder_a = board.GP8, encoder_b=board.GP9,
    encoder_button=board.GP22,
    time_button=board.GP16,
    servo_pin=board.GP4 )

```

The cable object provides a run method which is called to run the cable release.

```

cable.run()

```

The run method never returns. It calls the update method in the cable object 100 times a second:

```

def run(self):
    while True:
        self.update()
        time.sleep(0.01)

```

The update_shutter method checks to see if the button has been pressed to trigger the shutter. If the button has been pressed, the method pauses for the length of the self-timer delay and then opens the shutter for the length of the exposure. The update_times method checks to see if the rotary encoder has been turned; if so, the method updates the self-timer or the exposure value as appropriate.

Servo specifics

The Pico produces a sequence of control pulses to set the angle of rotation of the servo output. The servo class is used to generate these pulses.

```

from adafruit_motor import servo

```

The statement above imports the servo class from the library. The servo object is created when the CableReleaseController instance is created:

```

pwm_servo = pwmio.PWMOut(servo_pin,
    duty_cycle=2 ** 15,
    frequency=50)
self.servo = servo.Servo(
    pwm_servo,
    min_pulse=500,
    max_pulse=2200
)

```

The statements above are in the CableReleaseController constructor. They create a servo object and add it to the class as the member self.servo. The servo is then used to control the camera triggering. A program sets the position of the servo by setting the angle property to a value in the range 0 to 180 degrees.

QUICK TIP

You can set the **min_pulse** and **max_pulse** values to tune the code for your servo.

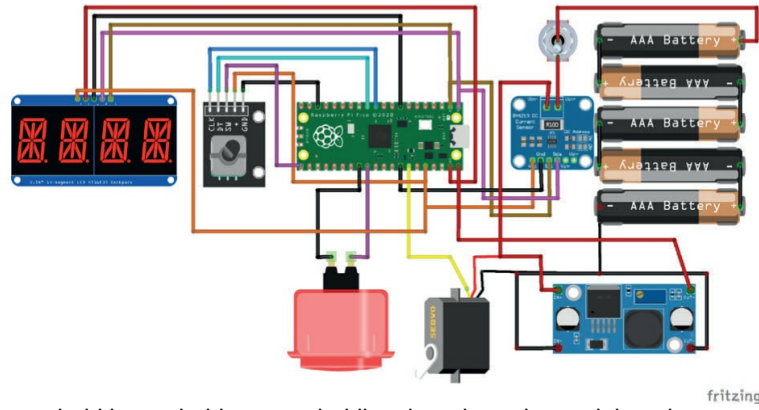


Figure 4: The batteries are held in two holders, one holding three batteries and the other two

```
 cable.servo.angle=70
```

The above statement would set the servo to 70 degrees.

The software moves the servo between a minimum servo angle where the cable release plunger is released to a maximum servo angle where the plunger is pressed in far enough to trigger the shutter.

```
MAX_SERVO_ANGLE=107
MIN_SERVO_ANGLE=20
```

The two values above give the maximum and minimum values for the servo in the cable release controller. These may be different for your release and the camera that you are using.

```
def servo_home(self):
    self.servo.angle=self.MIN_SERVO_ANGLE

def servo_trigger(self):
    self.servo.angle=self.MAX_SERVO_ANGLE
```

The `servo_home` and `servo_trigger` methods are called to move the cable release in and out. The values of `MIN_SERVO_ANGLE` and `MAX_SERVO_ANGLE` must be configured for a particular cable release and camera, otherwise the cable release might push the trigger too far and damage something. The author discovered by accident that the servo mechanism is quite capable of breaking itself when the servo tried to pull the actuator too far back and snapped the end off.

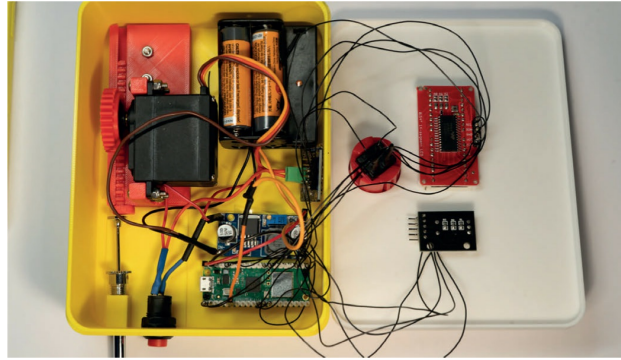
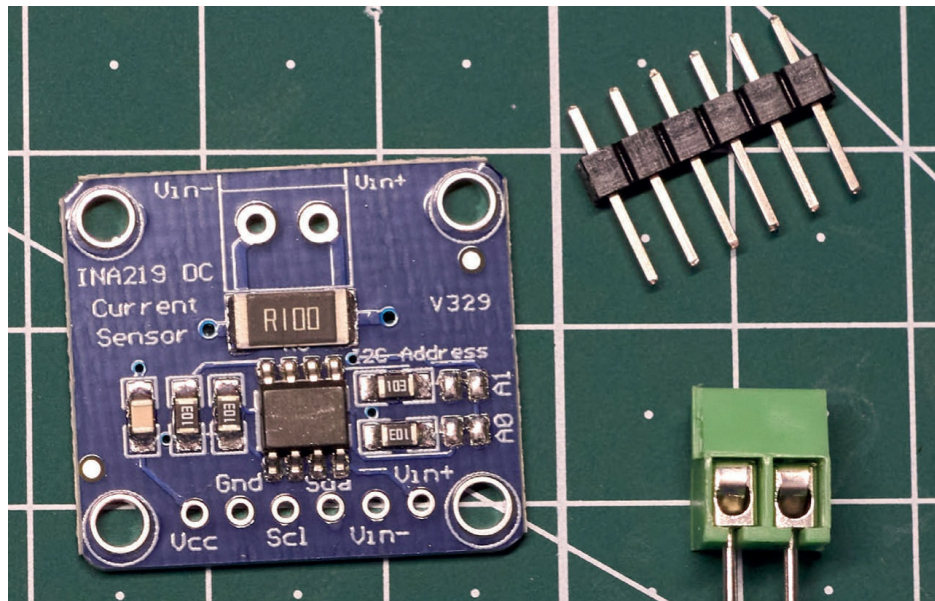


Figure 5: The R100 component of the current sensor is the shunt resistor, which is placed in series with the circuit to be monitored



The current sensor is stuck on the side of the box on the right

Disable self-destruct

It turns out that the servo is strong enough to damage itself if sent to a position outside the range of movement available. A current monitoring device is used to determine the range of movement automatically and avoid creating a device which will destroy itself the first time it is switched on.

[Figure 5](#) shows a CJMCU-219 current sensor which is fitted with an INA219 device. It measures the current flowing in a circuit by placing a small resistor called a ‘shunt resistor’ in series with the circuit being monitored. When current moves through the shunt resistor, it causes a voltage to appear across it. As the current increases, so does this voltage.

The INA219 device reads this voltage, works out the current that the voltage represents, and delivers information to Pico over an I2C connection. The harder the servo is working (and the more potential for damage), the higher the current. By monitoring the current value, a device can make sure that it doesn’t try to do something that would break the hardware.

You must be a little bit careful when using the current value in this way. When the servo is moving from one position to another, it can consume a lot of current for a very short time. This might make software think that something was wrong when it wasn’t. The current should be

checked once the servo can be considered to have arrived at the requested position.

QUICK TIP

During assembly, the servo angle should be set to 180 before the actuator rack is inserted into the gear. Reducing the angle of the servo should then pull the rack into the actuator.

QUICK TIP

When 3D-printing the gear and the actuator, you should use a smaller layer height (perhaps 0.08 mm) and print with a larger infill value (perhaps 60 or so). This will make them stronger.

```
from adafruit_ina219 import INA219
```

The `adafruit_ina219` library contains the `INA219` class which lets CircuitPython programs talk to the INA219 sensor.

```
self.i2c = busio.I2C(i2c_scl, i2c_sda)
self.ina = INA219(self.i2c)
```

The statement above creates an I2C connection and then uses this to create an instance of the `INA219` class. The instance is referred to by the variable `self.ina`. This instance exposes a set of properties given the current and voltage present in the system:

```
def show_current(self):
    bus_voltage = self.ina.bus_voltage
    current = self.ina.current
    power = self.ina.power
    shunt_voltage = self.ina.shunt_voltage

    # Print readings
    print(f"Bus Voltage: {bus_voltage : .2f} V")
    print(f"Current: {current : .2f} mA")
    print(f"Power : {power : .2f} mW")
    print(f"Shunt Voltage: {shunt_voltage : .2f} mV")
```

The method `show_current` above reads data from the sensor and then prints it to the console. This would allow manual monitoring, but it would be nice to have the cable release automatically determine the start and end points.

```
def limit_test(self, start, delta, max_current) :
    result = start
    while True:
        self.servo.angle = result
        time.sleep(1.5)
        current=self.ina.current
        print(f" Current: {current}")
    Angle: {result}")
    if self.ina . current >= max_current:
```

```

    result = result - delta
    self.servo.angle = start
    print(f"Result: {result}")
    return result
result = result + delta

```

The `limit_test` function receives a start angle, a delta amount the angle should change for each step and `max_current`, the maximum current that should be used. The function puts the servo in the start position and then moves it in delta-sized steps until the current flowing to the servo exceeds the max value. The function then returns the angle before that point was reached.

```

def set_limits(self):
    self.servo.angle = self.CENTRE_SERVO_ANGLE
    # do the inner limit first
    self.MIN_SERVO_ANGLE = self.limit_test(
        self . CENTRE_SERVO_ANGLE, -1, self.MAX_
CURRENT)
    self.MAX_SERVO_ANGLE = self.limit_test(
        self.CENTRE_SERVO_ANGLE, 1, self.MAX_
CURRENT)

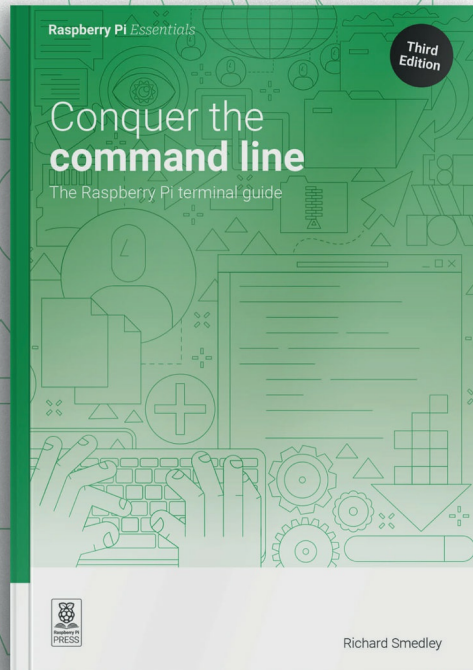
```

The `set_limits` function calls `limit_test` to set the servo limits. Note that for the inner limit, the delta value is negative so that the actuator is moved back from the cable release. The `set_limits` function can be called each time that the cable release is started, or you can use it once to obtain the maximum and minimum values and set them in the code.

Happy snaps

The release works well and there is considerable scope for development. The Pico could host a web page to allow the camera to be triggered remotely. The release could be programmed to take multiple pictures when the button is pressed, rather than just one. The current monitoring could be performed during movement, rather than just when the release is set up.

The cable release mechanism can also be used in any situation where you want a Pico-powered device to move something mechanical. You could even use it to create a mechanical ‘thumb wrestler’ which measures the strength of the opponent by tracking the current value, although the author hasn’t done that yet. 🟢



If you're not comfortable when faced with the \$ prompt, don't panic! In this fully updated book, we'll quickly make you feel at home and get you familiar with the terminal on Raspberry Pi (or any Mac and Linux computer). Updated for the latest Raspberry Pi software, this book has everything you need to get started.

■ **Build essential skills, including:**

- Read and write text files
- Find and install software
- Manage removable storage
- Use Secure Shell (SSH) for remote access
- Write disk images to SD cards
- Browse the web from the command line

BUY ONLINE: rpimag.co/commandlinebook

Make a digital jukebox

Create a music-playing box of tricks with your Raspberry Pi



Maker

Phil King

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



philkingeditor.com

YOU'LL NEED

- Raspberry Pi (4 or 5 recommended)
- microSD card
- Fruitbox software
- Music tracks
- USB thumb drive
- Touchscreen (optional)
- Push-buttons and connector wires (optional)

Joan Jett may have advised us to put another dime in the jukebox (baby), but these traditional mechanical music-playing marvels are not quite as commonplace as they once were, largely thanks to many bars and venues now streaming music instead. While some classic jukeboxes are now collector's items that can cost tens of thousands, there is a much less expensive option: build your own digital jukebox using a Raspberry Pi.

The free and open-source Fruitbox software (GitHub repo at [rpimag.co/fruitboxv2](https://github.com/rpimag/fruitboxv2)) we're using here is fairly easy to set up, works on a range of Raspberry Pi models, and offers a range of different skins to emulate classic and more modern jukeboxes. It can even play OGV format video files, although we won't cover that in this guide.

You can display your digital jukebox on a standard HDMI monitor, or use a touchscreen such as the Raspberry Pi Touch Display. There's also the option of mounting your display in a DIY enclosure or cabinet and adding physical buttons connected to the GPIO pins.

QUICK TIP

Useful keyboard controls include **BACKSPACE** to skip a track, **SPACE** to play a random song, left/right arrows to switch between groups of tracks, and +/- to adjust volume.

Back in time

The first thing to note is that you can't use the latest version of Raspberry Pi OS Bookworm, which is incompatible with Fruitbox. Therefore, you can either use the legacy Bullseye version of Raspberry Pi OS, available in the Raspberry Pi Imager tool, or an earlier version of Bookworm. As we used a Raspberry Pi 5, we opted to install the 15 March 2024 64-bit version of Bookworm, which does work with Fruitbox. For this, go to the official Raspberry Pi OS downloads page (rpimag.co/rpiosimages), click on **raspios_arm64-2024-03-15/**, and then select the top file (**2024-03-15-raspios-bookworm-arm64.img.xz**) to download the OS image. You can then write it to a microSD card in Raspberry Pi Imager, via the Choose OS > Use Custom option. In the OS customisation settings, we found it's best to stick with 'pi' as the username so you don't need to change any Fruitbox default file paths later on.



Fruitbox with the default SB-M100 jukebox skin; just enter the alphanumeric code for a song to play it or add it to the queue

```
pi@raspberrypi: ~/fruitbox
File Edit Tabs Help
GNU nano 7.2 fruitbox.ini *
[attract mode]
Enable = Yes
Choice = Random
Interval = 1.0 Minute
MaxVolume = 50
FadeIn = 5.0 Seconds
FadeOut = 10.0 Seconds
FadeInPosition = 60.0 Seconds
FadeOutPosition = 70.0 Seconds
SkipLoad = Yes

[skins]
File = skins/SB-M100/fruitbox.cfg
File = skins/Cassette/button_fruitbox.cfg

[buttons]
Quit = KEY_ESC
Coin1 = KEY_F1
Coin2 = KEY_F2
Coin3 = KEY_F3
```

Want a different look? You can add the file paths to other skins to the **fruitbox.ini** config file

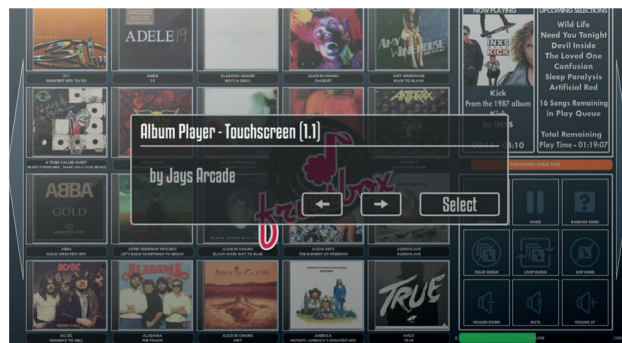
QUICK TIP

For more details on all the options in Fruitbox, including a list of default controls, visit the Fruitbox4 Dummies site: rpinmag.co/f4d. There's even a guide to creating your own skins.

You can press a letter followed by a number to select the relevant song

Picking fruit

Once you've booted up your Raspberry Pi with your fresh build of Raspberry Pi OS, you're ready to install Fruitbox. Note that there are both 64-bit and 32-bit versions available for Raspberry Pi 4 or 5, so you'll need to install the one matching your OS. For the 64-bit version, enter this terminal command:



With multiple skins added, launching Fruitbox will show a skin selector to choose one; some are designed for touchscreen (or mouse) control

```
wget https://github.com/chundermike/rpi-fruitbox-v2/raw/main/install\_fruitbox\_pi4
```

Even though it's labelled 'pi4', it will also work on Raspberry Pi 5. If you need the 32-bit version instead, enter:

```
wget https://github.com/chundermike/rpi-fruitbox-v2/raw/main/install\_fruitbox\_pi4\_32
```

You then need to make the install script executable:

```
chmod +x ./install_fruitbox_pi4
```

...or (for 32-bit version):

```
chmod +x ./install_fruitbox_pi4_32
```

Now run the script:

```
./install_fruitbox_pi4
```

...or (for 32-bit version):

```
./install_fruitbox_pi4_32
```

Fruitbox will take a little while to install, then you'll be able to run it with:

```
cd ~/fruitbox && ~/fruitbox/fruitbox
```

A startup screen will then appear with an error message saying 'No songs found'. Time to add some...

Add some music

Copy a few of your own (legally obtained) MP3 files to a USB thumb drive and insert it in your Raspberry Pi. You can always add more later. By default, Fruitbox looks in the **/home/pi/Music** folder for songs. So you can either copy all of your files over to that, or change the path in the **fruitbox.ini** configuration file. For the latter, enter `nano fruitbox.ini` to edit it; under '[general]', change the MusicPath. To point it at a folder called 'Music' in our USB thumb drive, for instance, we used:

```
MusicPath = /media/pi/USB_Drive/Music
```

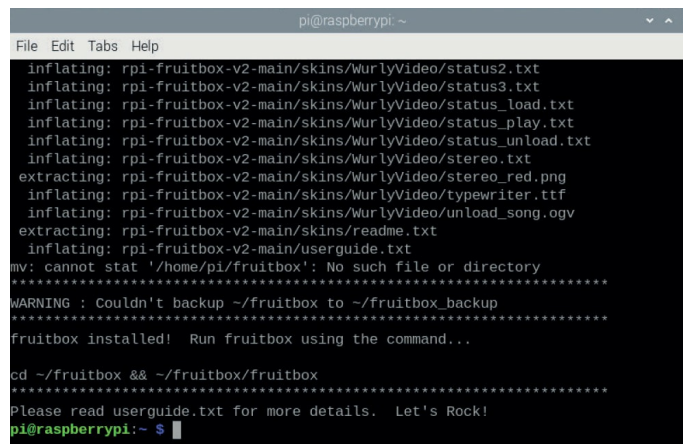
Change this to your own file path for your music files (which you can discover by opening their folder in the desktop File Manager). Note that the files do need to be in a folder, not loose on the drive. In addition, you can opt to organise them into artist and album folders – which you'll need to do if using an album-based skin.

With your song files in place, you can now restart Fruitbox and it will find them, add them to its database, then present you with the tracks loaded into the default jukebox skin. Just like on a standard jukebox, you can press a letter followed by a number to select the relevant song. It

will then be played on a pretend vinyl record. You can press more combinations to add songs to the queue. By default, free play mode is enabled, so you don't even need to add any credits – although you can change this in the settings (press **TAB** to bring up the main menu).

QUICK TIP

To bring up a Search menu, press the / (slash) key. You can then search the database for the song you want – useful if you have a very large collection of music tracks.



```
pi@raspberrypi: ~
File Edit Tabs Help
inflating: rpi-fruitbox-v2-main/skins/WurlyVideo/status2.txt
inflating: rpi-fruitbox-v2-main/skins/WurlyVideo/status3.txt
inflating: rpi-fruitbox-v2-main/skins/WurlyVideo/status_load.txt
inflating: rpi-fruitbox-v2-main/skins/WurlyVideo/status_play.txt
inflating: rpi-fruitbox-v2-main/skins/WurlyVideo/status_unload.txt
inflating: rpi-fruitbox-v2-main/skins/WurlyVideo/stereo.txt
extracting: rpi-fruitbox-v2-main/skins/WurlyVideo/stereo_red.png
inflating: rpi-fruitbox-v2-main/skins/WurlyVideo/typewriter.ttf
inflating: rpi-fruitbox-v2-main/skins/WurlyVideo/unload_song.ogv
extracting: rpi-fruitbox-v2-main/skins/readme.txt
inflating: rpi-fruitbox-v2-main/userguide.txt
mv: cannot stat '/home/pi/fruitbox': No such file or directory
*****
WARNING : Couldn't backup ~/fruitbox to ~/fruitbox_backup
*****
Fruitbox installed! Run fruitbox using the command...

cd ~/fruitbox && ~/fruitbox/fruitbox
*****
Please read userguide.txt for more details. Let's Rock!
pi@raspberrypi:~$
```

Fruitbox is relatively easy to install, although you'll need an earlier version of Raspberry Pi OS

Change your skin

While the default SB-M100 jukebox skin is pretty cool, there are plenty of others to choose from. To add more, edit the **fruitbox.ini** file again and add their file paths under '[skins]'. Just take a look in the **skins** folder beforehand to see what's available. Once you have more than one skin listed, you'll get a 'skin selector' when you start up Fruitbox – use the left/right arrow keys to browse them, and **ENTER** to select.

Some skins are designed to be used with a touchscreen, after enabling touchscreen control in Fruitbox – either in the Controls menu or in **fruitbox.ini**. We tried hooking up our Raspberry Pi 5 to a Touch Display 2 screen, mounted in a sleek Pibow Frame (rpimag.co/pibowframetd2), but had issues getting the Fruitbox skins to scale to the correct size and work with touch input. So a bit of tweaking may well be needed. For audio output, we used a Bluetooth speaker, since Raspberry Pi 5 lacks a 3.5mm jack.

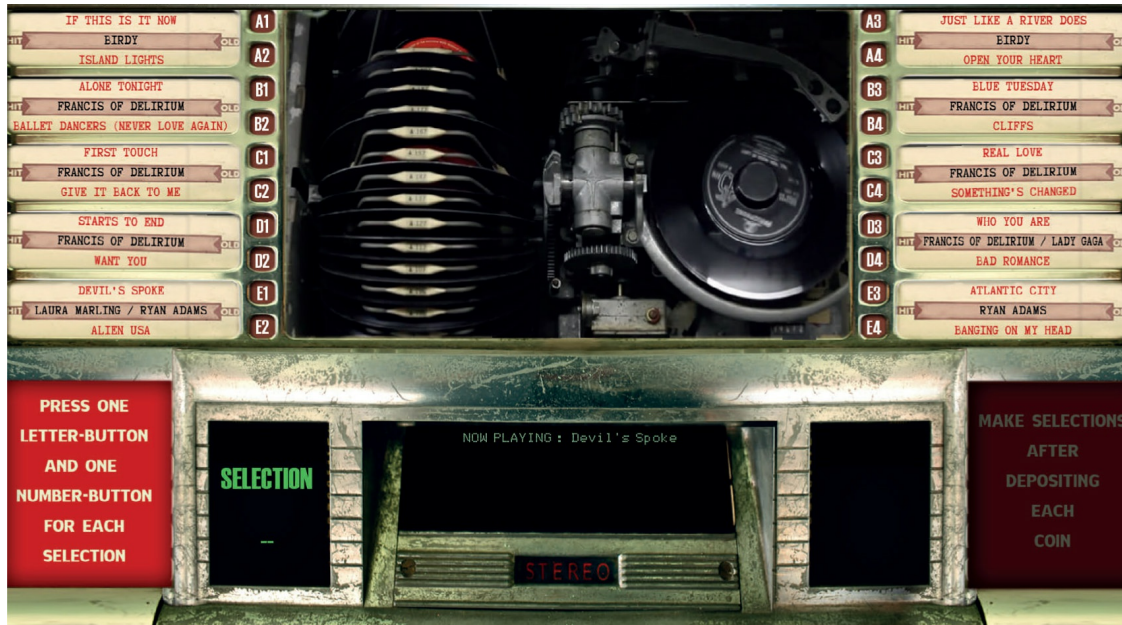
Finally, there's the option of using chunky push-buttons connected to Raspberry Pi's GPIO pins, for that true jukebox feel. You could then fit them into a 3D-printed or wooden enclosure, as some makers have done. However you implement it, Fruitbox is a lot more fun than your average music player. 🟢



This cassette player skin is a fun choice, and there's no danger of the tape spooling out!

Other Raspberry Pi models

While a Raspberry Pi 4 or 5 is recommended, Fruitbox will also work on any model from a Raspberry Pi Zero through to Raspberry Pi 3. For those, you'll need to use a legacy Buster version of Raspberry Pi OS Lite, however, due to sound issues on later versions. You should also set the GPU memory to 256MB: enter `sudo raspi-config`, then select Performance > GPU Memory.



The classic Wurlly skin features a neat recordchanging visual in the middle

QUICK TIP

On Raspberry Pi 5, opening the button mapping tool from the main menu will crash Fruitbox, so you'll need to change button settings (such as for GPIO pins) in the **fruitbox.ini** config file.

DIVE INTO MARINE TECHNOLOGY

Discover Raspberry Pi sea-based projects
and build an underwater robot.

By Lucy Hattersley

The ocean is a vast subject: mysterious to humans, both physically and technically. It's striking how much more we know about space than our own ocean floor. We have mapped the Moon, Mars, and Venus more than our own oceans.

The ocean is physically challenging, with high pressure, total darkness, and extreme cold

It's no surprise to find
Raspberry Pi equipment working
under the ocean

adding mechanical considerations on top of the usual project build requirements. Salt water is tough on equipment, and opaque to light. Radio waves do not travel well underwater, especially those at higher frequencies like Bluetooth, Wi-Fi, or standard radio communication. The oceans are huge!

Well-engineered Raspberry Pi hardware is at the forefront of this challenge. It's no surprise to find Raspberry Pi equipment working under the ocean, measuring everything from marine populations to mapping the ocean floor.

In this feature, we'll look at the professionals. Then Jo Hinchliffe is going to take us over TOUV (a Tiny Open-source Underwater Vehicle). From offering inspiration to your practical starter guide to underwater robotics, this feature will submerge you in detail.

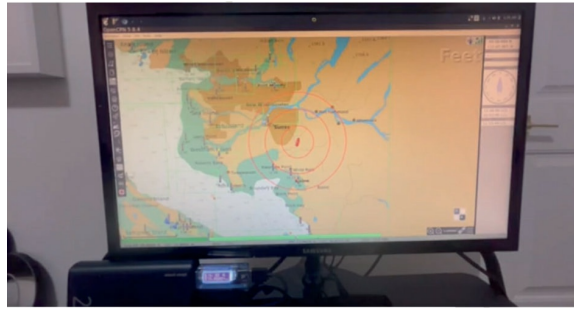
Let's dive on in...



Nemo-Pi

Save Nemo creates underwater weather stations for coral reef monitoring. These solar-powered buoys measure water conditions including temperature, pH, visibility, current strength, and chemical concentrations. The systems upload realtime data to public servers, helping divers and researchers investigate reef conditions.

 nemopi.com



Dave the MMP

Dave is retrofitting a Cal 2-30 sailboat using a marine navigation system built on Raspberry Pi 5. This provides a powerful alternative to expensive commercial chart plotters, running OpenCPN (opencpn.org) software with full NMEA (National Marine Electronics Association) integration. His YouTube channel shows a full integration of Raspberry Pi 5 with his boat.

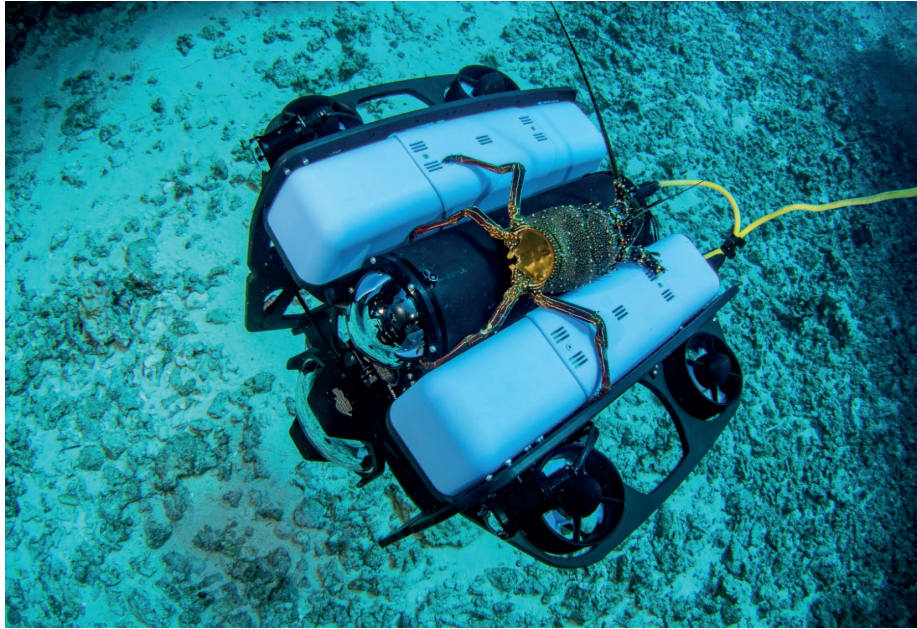
🌐 rpimag.co/davethemmp



Arribada Initiative

This project uses Raspberry Pi Zero to follow the journey of green sea turtles. The high cost of camera equipment presented a huge obstacle to conservationists monitoring species populations and behaviours. Arribada's affordable and robust Raspberry Pi-based kits proved to be a game-changer. The turtle tag comprises a Raspberry Pi Zero W sporting a Raspberry Pi Camera Module, a PiRA power management board, two lithium-ion cells, and an enclosure. The footage captured from the backs of these magnificent creatures is incredible.

🌐 arribada.org



Blue Robotics

The six-thruster craft BlueROV2 captures live 1080p video footage with a 200-millisecond latency. BlueROV2 can operate at depths of up to an impressive 300 metres, and beyond this is primarily limited by the physical tether cable that it uses for communication. The BlueBoat surface vessel, meanwhile, is connected via radio and can venture up to a kilometre from its base station – or much further if higher-performance radios or 4G are used.

Blue Robotics offers more than 350 products in the field of marine robotics and plans to continue using Raspberry Pi in its professional productions.

◉ rpimag.co/bluerobotics

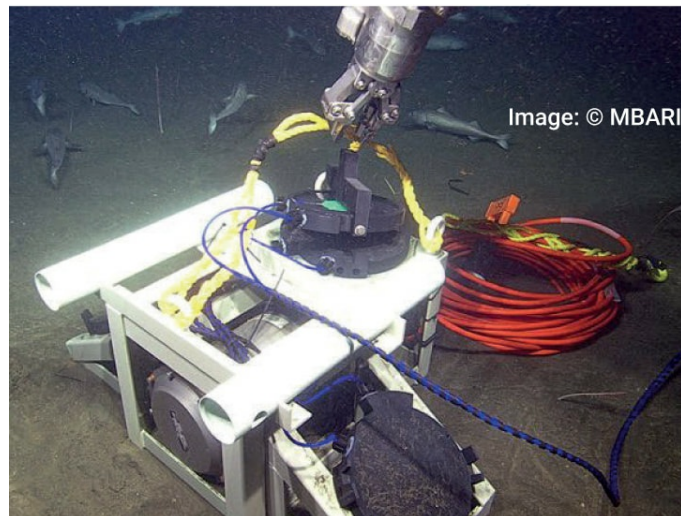


Image: © MBARI

Raspberry Shake

We're big fans of Raspberry Shake. Normally this equipment is used to measure seismic activity on the surface of the globe, but in 2020 the Monterey Bay Aquarium Research Institute placed one 900 metres

below sea level. The final unit consists of a small fibreglass frame with a titanium pressure housing that holds Raspberry Shake. On the sides and top of the frame are black plastic 'paddles' that use radio waves to transmit power and data wirelessly underwater.

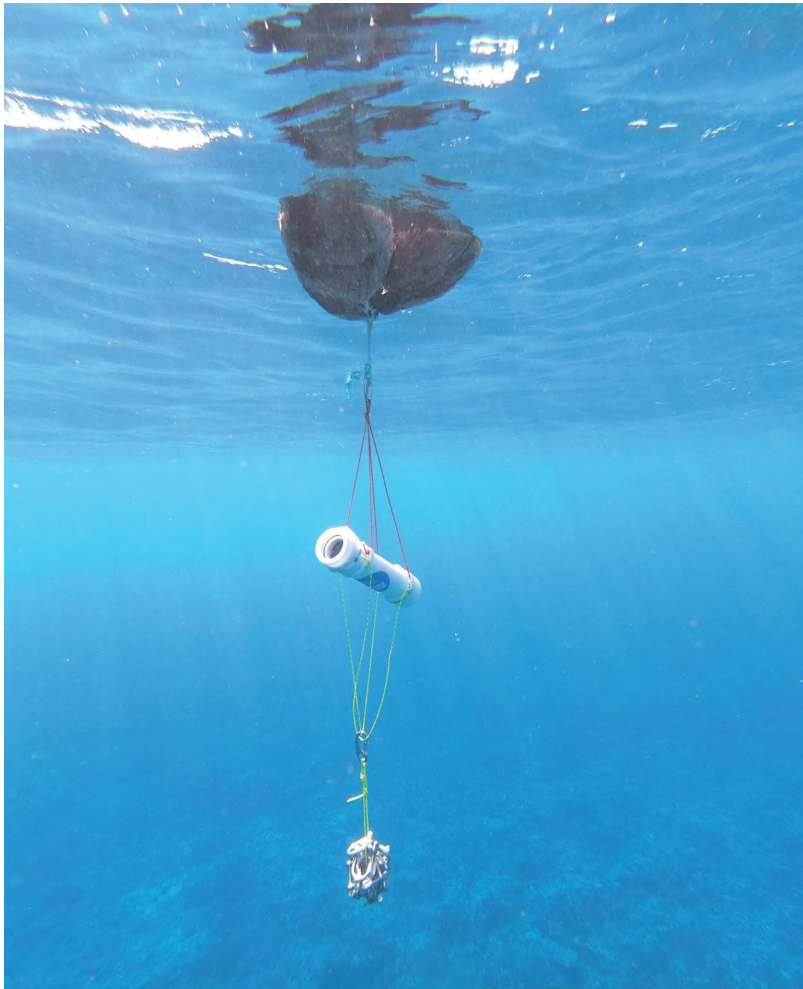
○ rpimag.co/shakeocean



OpenPlotter

On the ocean surface, Raspberry Pi has become the foundation for marine computing systems. Once dominated by expensive, closed-source systems, the world of marine electronics has been upended by OpenPlotter. Tailored for Raspberry Pi, this Linux-based operating system transforms the single-board computer into a complete marine navigation hub. It brings together weather information, chart plotting, GPS coordinates, and nearby AIS (automatic identification system) targets. It even provides an onboard wireless hotspot.

○ openmarine.net/openplotter



Maka Niu

Back in *The MagPi* magazine issue #125 (rpimag.co/125) we looked at Maka Niu, a modular deep-sea explorer powered by Raspberry Pi and Camera Module 2. This robotic deep-sea imaging system collects several types of data including video and still imagery, depth, temperature, and GPS location when out of the water. It can go up to 1500 metres below sea level – the team has designs to go 6000 metres, which would enable it to reach 99 percent of the sea floor.

Katy Groff Bell, president of the Ocean Discovery League, says: “By making this technology open and available to anyone, we hope to bring more equity to this important field.”

◉ rpimag.co/makaniu

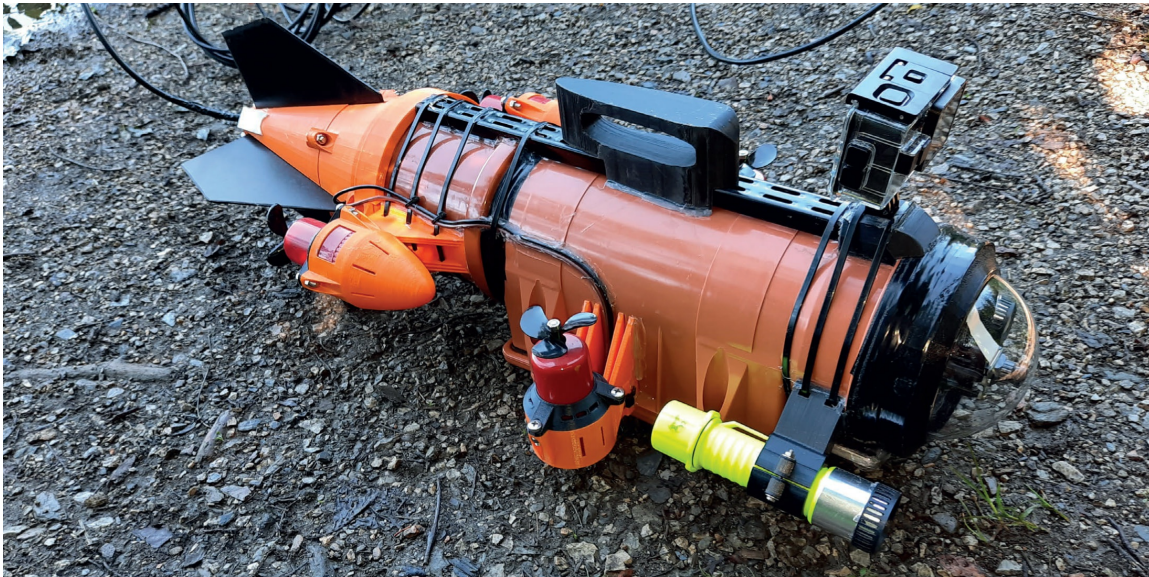
It can go up to 1500 metres below sea level

Penguin population monitoring

The Arribada Initiative also uses Raspberry Pi as part of Penguin Watch – an ongoing project to monitor penguins using Raspberry Pi hardware surviving multiple hard Antarctic winters. This project was covered alongside turtle monitoring in *The MagPi* magazine issue 123

(rpimag.co/123). Arribada initiative director Alasdair Davies explains that Raspberry Pi has always been one of the tools he uses “because it’s so accessible and affordable for anyone to get involved. You’re always partnering with a researcher or a local community member who works with an NGO and has a specific challenge. We get called in to solve it with technology. They’ll always say it has to be affordable, it has to be repairable and accessible.”

○ rpimag.co/arribada



Underwater ROV

This homemade remotely controlled submarine combines with a Unity app for an Android device that is used to record video and drive the submarine.

It uses a Raspberry Pi as its central control unit, a Camera Module 2 for video recording, sensors, and drivers, as described in the schematic (linked below). A test vehicle was constructed out of PVC-U sewer pipe with custom 3D-printed PLA parts and an acrylic dome, all sealed together using epoxy glue.

It's a great example of homemade hacking and what can be done to get started with putting a Raspberry

Pi underwater.

◉ rpimag.co/underwaterrov

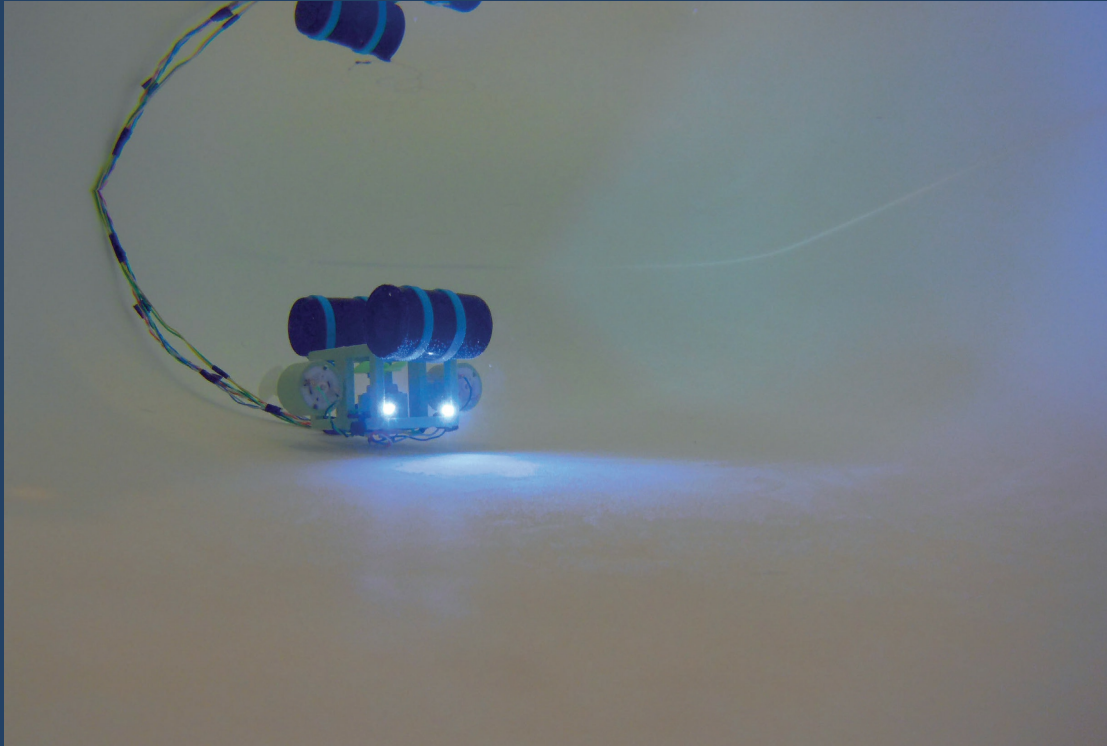


Autonomous recording for marine ecology

Cetacean species, including whales, dolphins and porpoises, are considered indicators of the health of marine ecosystems around the world. While a number are known to be endangered, a lack of data means that the population size and conservation status of many species are impossible to estimate. These animals are vulnerable to the effects of human activities and the noise they cause.

A team from the University of São Paulo in Brazil published a paper about the flexible, low-cost autonomous recorder they have built, based on a Raspberry Pi. The hydrophone – an underwater microphone – is protected by a cage made of stainless steel. This enclosure successfully withstands pressures of up to 10 bar, equivalent to those experienced almost 100m underwater, in pressure-chamber testing. 📌

◉ rpimag.co/underwaterrecording



The Tiny Open-source Underwater Vehicle (TOUV) being put through its paces in fresh water (salt water will corrode components)

BUILD A TINY OPEN-SOURCE UNDERWATER VEHICLE

With some 3D printing, motors, and a Raspberry Pi Pico, you can be up and running with an underwater explorer!
By **Jo Hinchliffe**

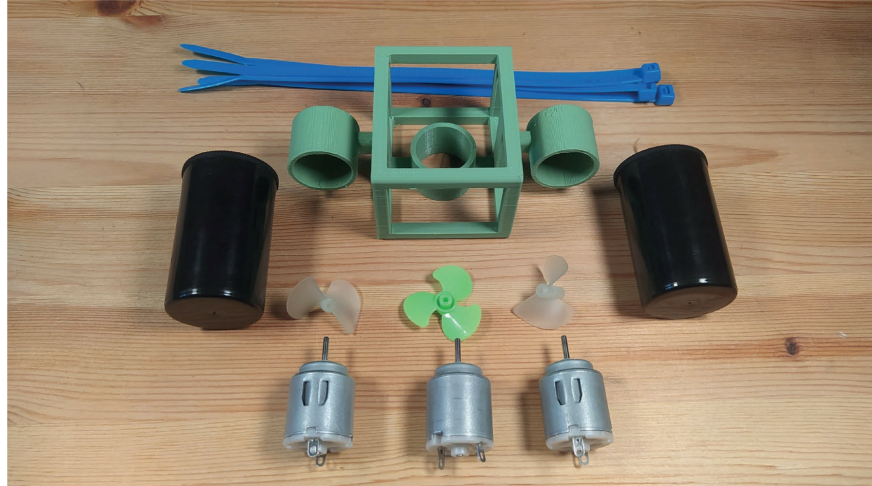


Figure 1: the main structural components for a TOUV – a chassis frame, 3 x 30mm motors, 2 x 35mm film canisters, some propellers, and some cable ties

About a decade ago, we were inspired by a workshop at Liverpool Makefest where an organisation called the Dark Water Foundation made all manner of water monitoring devices.

Dark Water Foundation got attendees to make miniature palm-sized underwater rovers using Lego, 35mm film canisters, and some small brushed DC motors. “DC motors?” I hear you cry. Well yes, the revelation on seeing this workshop was that small 3–6V DC motors run really well in fresh water without any attempts to waterproof them. The caveat being you are probably shortening their life somewhat, and you need to do a little work to dry them out after each mission.

Many of the original links to this workshop are now missing; so, with absolute respect for the original project, we decided to make a more modern attempt influenced by that original design. There still is an Instructables page with some images and component lists, but sadly the links to components from the page aren’t currently working, so we’ve made some educated guesses and experimented to come up with a working solution.

For our more serious testing, we moved to a second-hand inflatable hot tub

The original design was around a 60-75mm cube for the frame made from Lego. We set out to 3D-print a similar-sized frame using the free and open-source FreeCAD package to realise our ideas. The original design used 30mm brushed DC motors and we discovered that these at the time were rated to 5400rpm. We couldn’t find that exact specification, but found some close enough listed as 3–6V and 6500rpm. These are perfect because we intend to make a moderately more advanced controller for our design – we can use motor drivers to control our speed.

Once the motors arrived ([Figure 1](#)), we could then grab a set of callipers and take some motor measurements and begin our CAD work. In FreeCAD, we used the Part Design workbench to create the chassis, building up the frame by adding sketches to faces and extruding

them ([Figure 2](#)). Our main chassis area is a cuboid frame and the three thrusters: two set up for forward and turning (and potentially reverse if needed) on the sides of the frame and one inside the frame, set in the vertical axis to control vertical position. The two horizontal thruster mounts are offset slightly from the centre in an attempt to distribute the weight a little.

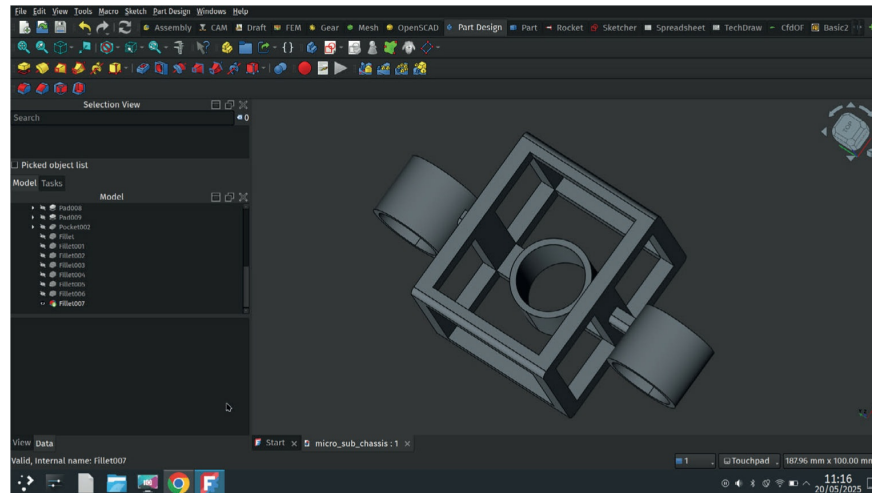


Figure 2: Designing the chassis in FreeCAD was a pretty simple task

Part of the design led us to consider what our buoyancy approach might be. There are two or possibly three options. We can either aim for absolute neutral buoyancy, where the vehicle neither descends or ascends in the water until a motor turns, positive buoyancy where the vehicle always wants to float and the Z axis thruster pushes it down, or negative buoyancy where the vehicle always sinks until the Z axis thruster supplies lift. Of course, the optimal is neutral buoyancy, but this is hard to achieve, particularly when you have tether wires which change the mass of the setup as it moves; also, for neutral buoyancy, we'd need to be able to switch the rotation of the Z axis thruster.

Negative buoyancy is reasonably easy to achieve. We are going to add two 35mm film canisters (still widely available online) and we can simply fill these with water to make the vehicle sink. Neutral buoyancy has some advantages and a disadvantage. An advantage is that if you completely fill the film canisters with water then they are equal and balanced – if they are part full, it can be difficult to not have the water move around and change the orientation of the vehicle. The disadvantage of negative buoyancy is that it uses a little more power as you'll use the thruster a lot more. And if for some reason the power or motors fail, your vehicle will sink – perhaps not too much of an issue at this scale as you can pull it out by the wire tether.

After printing a test frame in PETG filament, the assembly of the vehicle is pretty straightforward. With a reasonably well-calibrated 3D printer, the thruster mounts are tight enough for the motors to be a push fit ([Figure 3](#)). We used some 26mm two-blade nylon propellers for the horizontal thrusters and a larger three-blade 30mm propeller for the Z axis, making sure to order ones with the correct 2mm holes to mount to our 30mm 3–6V brushed DC motors.

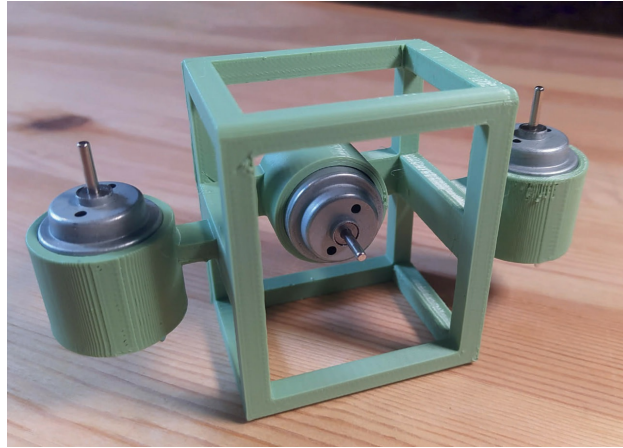


Figure 3: the motors are a friction fit in the chassis, meaning you can adjust their position a little, and are easy to replace if ever needed

It's important to try and find a balance of long wires that are thin and flexible to create the power lines for the motors. We found an old three-metre network patch cable that we pulled apart to get long lengths of suitable wire. If you wanted to spend rather than find something from your junk cable pile, you could probably find thinner and more flexible solutions. As the chassis sits quite nicely on a worktop, we inserted the motors into the chassis and then soldered the wires on, using the chassis as a helping hand.

Adding the buoyancy tanks is as simple as two cable ties wrapped around the chassis and the 35mm film canisters ([Figure 4](#)). You can experiment with the position of the tanks slightly after they are fitted. If you place them more out in the front and the rear, if the vehicle is floating you might find your Z axis thruster sits slightly out of the water. Raising the tanks above the vehicle means the Z axis propeller is always covered in water.

Once you have your motors in place and wired and you have fitted the buoyancy canisters, it's time to test. We ran our first tests by simply holding the motor wires across an 18650 battery momentarily to work out the polarity. With all that identified, we then couldn't resist playing with it in a washing-up bowl full of water. This proves quite worthwhile as although you can't really do too much, you can get a sense for if you want to have positive or negative buoyancy and play with adding different amounts of water to the floatation canisters.

At this point, you could certainly make up a straightforward controller box with just some buttons or switches for each motor and some batteries. However, we went with the addition of a Raspberry Pi Pico as a controller, as this allowed us to drive the motors using pulsewidth modulation (PWM), which allows us to control their speed and direction. To do this, we needed to use some motor drivers and we actually used a StoRPer board, which is a PCB we made as part of our *Design an RP2040 board with KiCad* book (rpimag.co/kicadbook) and articles that featured in *HackSpace Magazine*. The StoRPer board ([Figure 5](#)) is essentially a breakout board that connects some of Pico's GPIO pins to four DRV8833 motor drivers. You could totally emulate this board on a breadboard using the StoRPer schematic and components; however, DRV8833 motor drivers are commonly sold as small modules on breakout boards and that would make for a simple option.

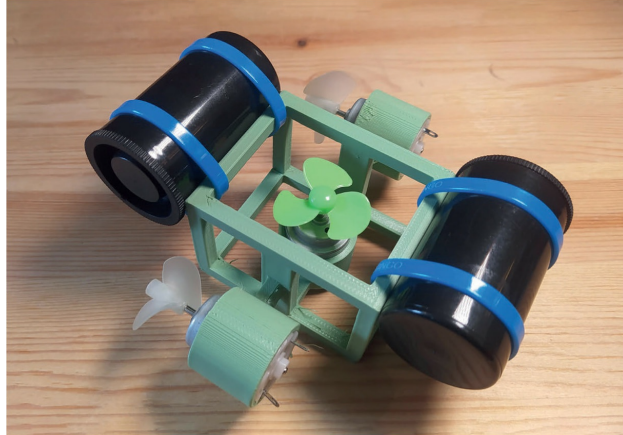


Figure 4: Adding the 35mm film canisters to act as buoyancy tanks

ALERT!

Electrical issues

Directly shorting wires across a battery carries a slight risk of sparks. Consider building a test circuit. Li-ion cells can also cause burns or fire if shorted. The TOUV project uses 3-6V DC motors. Please do not use higher voltages or larger voltage motors.

Motors and water

One of the main reasons this project has been in our minds ever since seeing the Dark Water Foundation Lego robots at Liverpool Makefest a decade ago is that we were shocked to see that DC motors ran underwater with no attempt at waterproofing. This is not a common practice, but it does work.

We can't say that it is the healthiest environment for small brushed motors, but it certainly isn't as bad an idea as it sounds. The motors run fine and we've tended to take some care when we finish an underwater session. We've spun the motors out of the water for a good two minutes of constant running to push out remaining moisture. We've also given them a good blast of either canned air, or a decent blow from the mouth when our canned air ran out.

We are sure that running DC motors in fresh water is not good for them and we are definitely shortening their lifetime; however, ours have survived a lot of underwater time now and we are still on the original pair. We also found our motors online at less than 60 pence each when buying a ten pack, so it isn't the most costly thing to replace them if and when they fail.

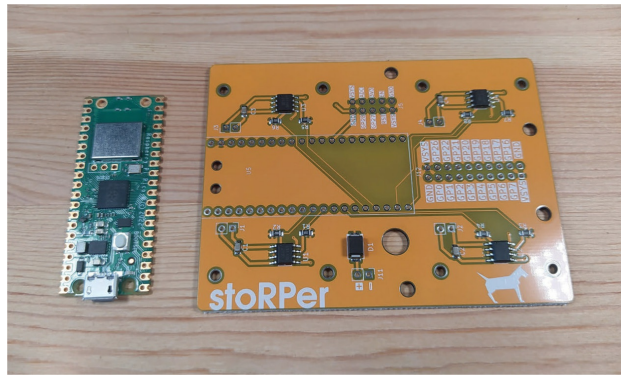


Figure 5: Using a StoRPer board is functionally the same as adding some DRV8833 modules to a Pico

For our first controller system, we mounted three momentary press-to-make buttons in a panel ([Figure 6](#)) and connected them to Pico's GPIO. In the MicroPython code ([Figure 7](#)), we have identified the GPIO pins for the buttons and defined them and set up a PWM system for each of the motors. Swapping the pair of pin number values for each motor can then reverse that motor and we can also change the speed values for each motor.

For our more serious testing, we moved to a second-hand inflatable hot tub. These can often be found second-hand, neglected and filthy, for very little money and make for an excellent large test environment for water-based projects. We gave ours a pretty good scrub and when we filled it, we gave it a dose of chlorine to kill off any bugs a couple of days before using it (to allow the chlorine to dissipate). As an aside, it's great fun to place an action camera in a waterproof case in the tub/tank and capture some footage/images of your underwater test missions! Here is a link to an edited minute of underwater testing: rpimag.co/touvvid.

One thing we will change is that our wires are hard-wired to the controller and it would be beneficial, especially at the testing stage, to be able to quickly remove the wires so the controller alone can be taken back to the laptop inside (not near the hot tub) for code tweaks. We did a lot of drying of the vehicle and wires every time we needed to make a change. We powered our system a couple of different ways: either with an 18650 cell or with a USB power bank. With the 5V of the power bank, the motors spin quite quickly and the vehicle has too much thrust and is difficult to control, but this is easily rectified with changes to the speed values in the MicroPython script. We're currently set up with a slight positive buoyancy with the Z axis thruster set to turn a lot slower than the horizontal thrusters as it's nice to be able to descend smoothly and gradually. We've also ended up drilling the small holes in the chassis to flood the chassis when submerged.



Warning!

Chemicals

Please handle chemicals such as chlorine responsibly.

rpimag.co/chemicals



Figure 6: Adding a small panel into which we mounted the buttons as a first attempt at a controller

We mounted three momentary press-to-make buttons in a panel

LEDs for looks

Sometimes you add features just because. This was certainly the case with the pair of LEDs we mounted to our build. Obviously if we had some kind of camera system on board then lighting might have a purpose, but we just added these for the cute factor. It's as straightforward as soldering a current-limiting resistor in series to the anode of an LED – we went with 100 ohms, which offered protection across the range of voltages we were intending to use on our build (3.7–5V).

Once soldered to some wires, we added a little heatshrink to stop the LED anode and cathode potentially touching, and for a little structural reinforcement. We then used some small cable ties to attach the LEDs to the chassis.

It's great fun to tinker with and we have a few ideas for improvements. It would be great to make the wiring a little more flexible with some thinner and more flexible wires. We'd also like to add more functions to the controller. A switch to reverse the direction of the horizontal thrusters would be nice, as well as some latching switches that can set motors to be permanently running at a slow rate, perhaps even with potentiometers to vary the speeds. For example, we could set a slow speed for the Z axis motor that, when activated, counters the positive buoyancy, allowing the TOUV to have a kind of 'altitude hold'. It could be fun to add attachments like a hook or a magnet for fun 'rescue the object' games and challenges. We would of course love to get a camera on board, and sensors and more functionality, but we think that might be best on a larger DIY submersible project. Keep an eye out for that in the future. In the meantime, if you want to build a TOUV, the files are in this GitHub repository: rpimag.co/touvgit. 📁

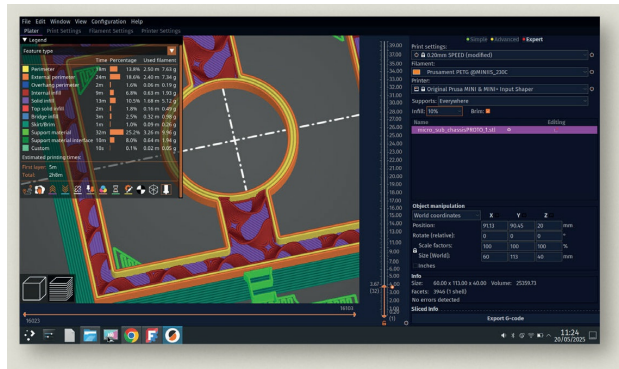
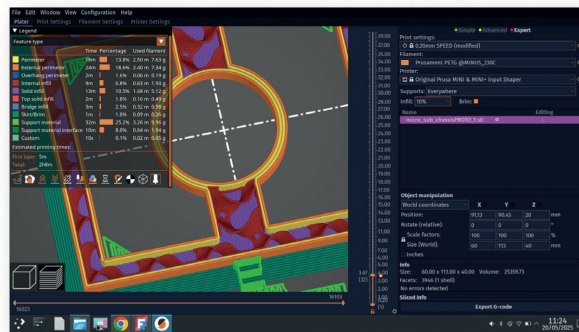


Figure 7: Using MicroPython on the Pico meant we could quickly make changes to motor speeds and directions

Larger vehicles

Researching DIY underwater vehicles, there is a whole scene of larger vehicles with their chassis built from PVC or ABS plumbing pipes. Most of these builds have small holes drilled into the pipes to allow the chassis to fill with water when submerged. This has some great benefits; the added mass stabilises the vehicle in the water and it means that you are only dealing with your actively added floatation tanks or tubes when tuning the setup.



After some pondering about designing holes into the small chassis, we realised this could be achieved a different way. Inside filament-type 3D-printed structures, you tend to get your slicer software (that prepares 3D objects for printing) to automatically add a percentage of infill. This infill is generated in patterns to create some kind of lattice inside your object, adding to the strength of the structure. Most slicers offer a collection of different patterns that can be used to create infill. A lot of the time, a square or hexagonal mesh is fine, but there are some interesting options. One option is a curvy gyroid infill. One interesting aspect of gyroid infill is that although it creates strong bonds between the parts it fills, it doesn't ever close off sections within the object. This means that if you print with a gyroid infill and then drill some small holes in the corners of the chassis, the chassis can completely fill with water and also completely drain water after an underwater session.

Animate a Halloween skeleton

Build your own motion-activated spooky decoration



MAKER

Ben Everard

Ben saves money on Halloween treats by scaring the children away.



glowingart.co.uk

YOU'LL NEED

- 4 × 9g servos
- Raspberry Pi Pico (any version)
- Halloween skeleton
- 2 × WS2811 fairy lights
- Distance sensor
- Backing wood (ply or OSB)
- Screws
- Black paint

As October draws to a close, the veil between the worlds of the living and the dead grows thin, allowing spirits to cross. At least, that's what watching too many horror/comedy films has taught us. Since the real world is rarely as interesting as Hollywood depicts, we're not going to rely on the dead rising up, and give Halloween a helping hand by creating our own animated skeleton. You can build your own by following along, or adapt the techniques to animate whatever spooky (or not spooky) decorations you like.

Our build starts with a 90cm glow-in-the-dark skeleton that we purchased from AliExpress. The key part of this build relies on it being a '2D' skeleton where all the joints move in the same plane. There are some 3D skeletons available that are more anatomically accurate, but also harder to animate. Unfortunately, it can be a little hard to work out from the listing whether it's a

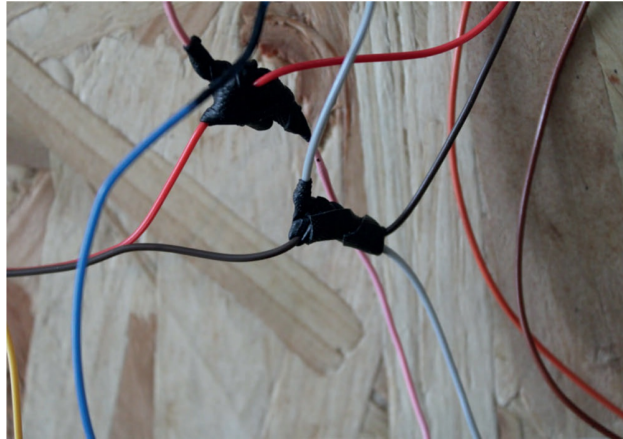
2D or 3D skeleton, but if it's 90cm, glow-in-the-dark, and about £3, then it should be the right one.

We'll be creating our own animated skeleton

The skeleton came as ten separate parts: two for each limb, one for the body, and one for the head. We're going to add control to the hip and shoulder joints, meaning that our skeleton can perform a few moves. The knees and elbows we'll leave loose to flail around.



Our assembled spooky skeleton



There are – arguably – better ways for joining four wires together and insulating the joint, but they all require more parts and complexity than a solder joint and self-amalgamating tape.



Always make sure you position your servos before connecting the horns

Before we get to all that though, we need a few additional parts. The skeleton is very flimsy, so we need a firm backing that we can place it on to keep everything in place. We used Oriented Strand Board (more commonly known as OSB) because we happened to have some available, but plywood would also work well. It needs to be at least $45 \times 21\text{cm}$, though if it's bigger, that's fine (and might protect your skeleton a bit more). We painted ours black to blend into the dark night.

For the motion, we used servos. Servos are a little like motors in that you send a signal to make them turn, but unlike motors, they are used for turning a particular amount and usually operate a range of about 180 degrees. You send the servo a signal to let it know the angle you want, and it turns to that angle.

Servos come in different sizes and strengths, and we opted for 9g servos which are just about strong enough for this. If you use anything bigger, you might find that they draw too much current, so we'd recommend sticking with little ones unless you want to add an external power supply.

We wanted our skeleton to react to people around it, so we added a distance sensor to detect when someone is close to it.

Finally, we need a controller to make everything do what we want to. A Raspberry Pi

computer will do the job, but for simplicity, we've used a Pico. Ours is a Pico W, but we're not using the wireless interface and any model should do just fine.

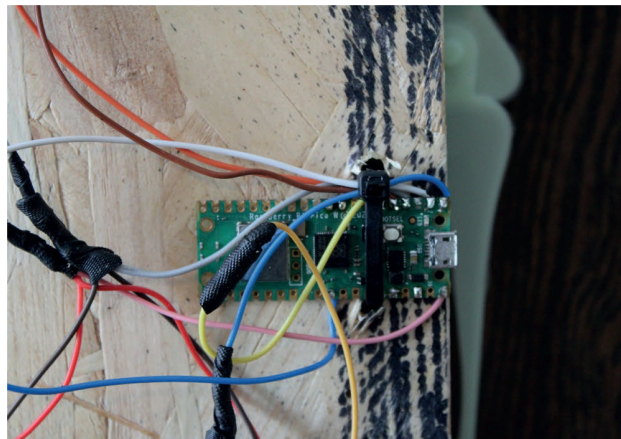
You'll also need a few 3.5mm wide screws that are smaller than your board is thick, some mounting hardware, and a cable tie.

Bare bones

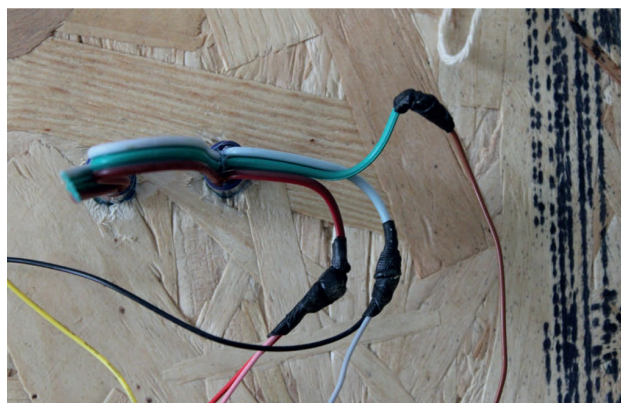
Our skeleton came with moveable joints with a pin-and-socket connector. Our first thought was to keep these as hinges and then hook up an actuator to the limbs to pull them into the position we want (like making a puppet). However, we found that the flimsy plastic of the skeleton had a habit of twisting rather than rotating properly at the joint, so we needed a different plan.

The first step is to disconnect the limbs from the skeleton and they should simply pop out. We're not going to join our skeleton directly, but have the servos mounted directly to the limbs, and held in place near the joint on the main body.

We now need a way of attaching a servo to each limb. Servos have a press-fit drive shaft that connects tightly to 'horns', and you usually get a selection of different-shaped horns with your servo. We used the single horn and screwed it close to the original joint.



Make sure Pico is mounted in such a way that you can access the USB port



Power, ground, and data – three connections that let you add as many LEDs as you need.

Your servos should come with screws that go through the holes on the horn. The only problem with these is that they poke through the thin plastic of the skeleton and could scratch

someone who gets too close. You can use a blob of hot glue or attach a small bit of plastic or wood to the end of the screws if you're worried about this.

Once you have the horns attached to each limb, you now need to worry about the other side of the servos. Servos do have mounting screws, but these are usually higher up the servo, designed for a recess, so we need some form of mount to hold them in place.

If you have access to a 3D printer, then the easiest option is to print one. We used this one: rpimag.co/9gservomount, but depending on your servo, you might need to use a different option. The important thing is that it holds the servo vertically.

If you don't have access to a 3D printer, it should be possible to create a hole in your backing wood – either by chain-drilling or by drilling a starter hole and using a jigsaw – then mount the servo in there (perhaps with a liberal amount of hot glue).

Once you have your servo mounts, you should have all the key pieces of your motion control hardware.



Your servos should come with mounting hardware

Eye lights

It's well established in cinema that scary things have red eyes. We're not really sure why this is, since the only thing this author has ever seen with red eyes was an albino rabbit, and that little ball of fluff was the least scary thing in existence. However, since that is the established trope, our skeleton should have red eyes.

There are a couple of options for this: we could use red LEDs, or we could use RGB LEDs and just neglect two of the three colours. Although the latter option might seem a bit wasteful, it simplifies the build a bit and gives us options for the future, so we'll go with it. The specific LEDs we'll use are 12mm WS2811 LEDs. These come in chains (usually of 50) running at either 5V or 12V. We want the 5V version, and we just need two of the LEDs, but we can cut these from a chain.

Dark as the night

The physical setup of the skeleton is based around the back board. This should be a rectangular sheet of engineered wood strong enough to hold everything in place. The skeleton comes with a mounting hole at the top, and you can also use the holes from the shoulder and hip joints to screw it down securely.

Additionally, it'll need 12mm holes in the eyes for the LEDs, and 10mm holes near each joint to allow the wires from the servos to be tidied out the way.

At this point, you might also want to paint the board black so the skeleton stands out.



Screw the horns onto the limbs as close as possible to the original joint

Zero in

We're now ready to attach the servos, but before we do this we need to set up the electronics a bit. Servos don't rotate a full turn. Typically, they can turn about 180 degrees. Because of this, we have to know their position before we attach the arms and legs, otherwise we might not be able to rotate them into the correct position once they are attached. So, before we can continue with the physical build, we need to wire them up and rotate them into a known position.

We've got four servos, but they all need to be connected to the same power pins, so some creative wiring is needed. We wired it up as shown in [Figure 1](#). This requires soldering a four-way joint. You might need to solder some extra bits of wire onto the servo wires for extensions. Make sure that the servo wires go through the holes in the board to allow the servos to be mounted properly before they're soldered together.

Once you've got the servos soldered up, you can now find out what state they're in and for this we'll need to program Pico.

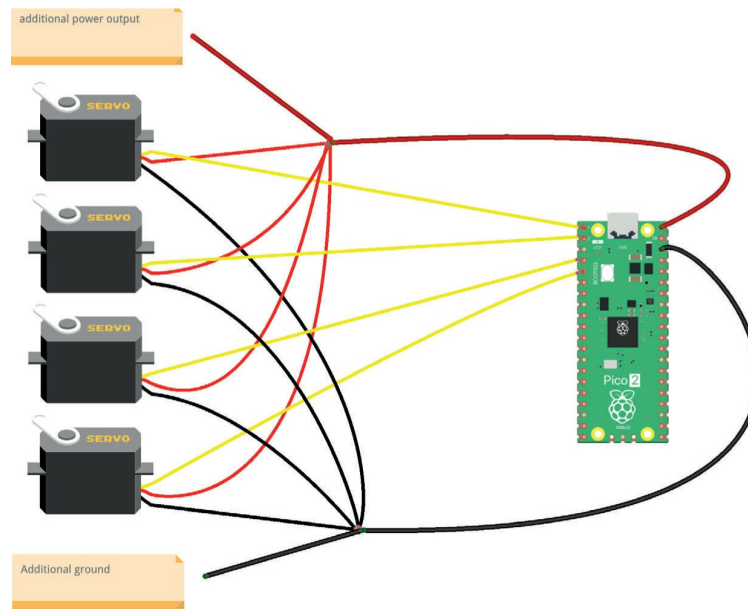


Figure 1: The wiring isn't particularly complex, but there is enough to get in a bit of a tangle if you're not careful. Using different-coloured wires makes the build a lot easier

There are a few options you could use to detect people

INSULATING JOINTS

Conventional wisdom has it that you should use heat-shrink tube over soldered wire connectors to insulate them. This can work for simple wire-to-wire joints but if you're doing anything slightly unusual, then it often doesn't work. This author has now switched to using self-amalgamating silicone tape.

This comes as a tape (usually black) on a backing (usually white). You peel the tape off the backing and wrap it around the joint. The tape will stick to, and amalgamate with, the tape below making a block of silicone. You can wrap it around whatever junction you have and it will insulate it. It's very stretchy, so make sure you pull it tight as you wrap it round to create a good joint. As well as insulation, it can provide strain relief. We find it creates a snugger fit and better protection than heatshrink (and it helps that you don't have to remember to slide the heat shrink on before soldering which we had a habit of forgetting to do). While it's probably not completely water-tight, we've used it successfully on projects that have survived outside in bad weather for months.



The simple pin-and-hole hinge joints let the elbows and knees move freely as the skeleton dances

Code configuration

If you've not used MicroPython before, take a look at the getting started guide at rpimag.co/micropythonsdk. The short version of this is that you need to install Thonny (from thonny.org), and then install the MicroPython firmware on Pico (see rpimag.co/micropython). At this point, you can use the Thonny IDE to write code and send it to Pico.

You'll also need the servo library. You can download this from rpimag.co/micropyservo. Using Thonny's file manager, create a folder called **servo** on Pico and copy the `__init__.py` and **main.py** files into it.

Attach something easily removable to the drive shafts (we used flags made of electrical tape), and run the test code below.

```
import time
from servo import Servo

left_hip = Servo(pin_id=0)
left_shoulder = Servo(pin_id=1)
right_hip = Servo(pin_id=2)
right_shoulder = Servo(pin_id=3)
servos = [left_hip, left_shoulder, right_hip, right_shoulder]
down_angle = [180, 180, 0, 0]
up_angle = [90, 90, 90, 90]

while True:
    print("down")
    for i, servo in enumerate(servos) :
        servo.write(down_angle[i])

    time.sleep(2)

    print("up")

    for i, servo in enumerate(servos) :
        servo.write(up_angle[i])

    time.sleep(2)
```

This alters the state between two different states: down and up. You should be able to adjust the flags so that down points down and up points at 90 degrees. This then tells you the angle you

want to add the limbs at. In other words: once the flags are adjusted so they point in this direction, you can then add the limbs so they point in the same direction as the flags.



The original joint holes make convenient mounting holes

However, before we add the limbs, let's add the final two parts of the electronics: a distance sensor and the LEDs.

There are a few options you could use to detect people. Passive infra-red (PIR) sensors are popular choices for triggering things like this, and one could certainly work in this project. However, they can be triggered very easily and we prefer our skeleton to activate when someone gets closer.

It is possible to do something advanced with a camera and some form of machine-learning person detector, but these don't always work well in the dark (particularly with changing light), so aren't a great choice for nighttime installations.

We've used a plain distance sensor. Specifically, a Sharp GP2Y0A02YK.



Don't tell your woodworking teacher, but wiggling the drill a little as you make the eye holes makes it easier to mount the LEDs

This outputs an analogue voltage that corresponds with the closest thing to it (up to about 1.5m). It's easy to work with as we just have to hook it up to an analogue input. The one downside is that there's quite a small field of view, so it doesn't always trigger when you hope it would. Our search for a perfect person detector continues, but for now, this works well enough.

Wiring the sensor requires 5V, ground (both of which are built off the power chain for the servos), and an analogue input (we used A0 on GPIO 26).

The final part of the electronics are the LEDs for the eyes. These should come as a string and you'll need to cut two off. Each LED has six wires attached to it: two for power, two for ground, one for data-in, and one for data-out. This means that the string has three wires for input and three for output, but it's not always obvious which end of the string is input and which is output. If you look very closely at the string, you should see an arrow on the PCB and this points to the data-out line. The data-in line is opposite this. Power should be red and ground should be white.

Power requires 5V, so along with the distance sensor, that's another two devices that need to be connected to the single VBUS connector on Pico, so you'll need to extend the chain of wires built off this pin. We connected the data-in to GPIO 5, but any should work.

It might be tempting to add more than two of the LEDs to the display. You could certainly add more to act as lightning or any other effect. However, we'd recommend that if you do this, you add an additional power supply as we're stretching the USB power supply about as far as we should really push it.

Eye eye

Now let's finish up the assembly. The LEDs push into their holes – just. Although they require 12mm holes, they have barbs to hold them in place. With a bit of force, you should be able to push them in far enough for them to hold in place. If you can't, then the best option is to use a 12mm drill bit and wiggle it in the hole to enlarge it slightly.

Mounting Pico is a bit tricky as it's hard to find wood screws that fit through the M2.5 holes. The most secure option is to use a case with mounting points. However, a quicker and easier method is to drill two 5mm holes, one either side of Pico, and use a cable-tie to hold it in place. It's a bit of a hack-y solution, but good enough for a prop that's going to be out for a few days a year. Just make sure that the USB port is at the edge of the backing otherwise you won't be able to plug in a cable, and also make sure that the cable tie doesn't go over the BOOT button.

Finally, you need a way of mounting your skeleton, and this will depend a little on where you want to display it. We fastened two eye screws into the top of the backing, which will allow us to hang it up. You could also drill through the backing and screw it into place, or just G-clamp where you want it to be.



Eye screws in the end of the board aren't the most secure attachment, but it should hold the weight of the skeleton



OK, the wiring could be tidier, but if you want to secure it further, you'll need to box in the back of the board



Warning!

Power Tools

Be mindful of dangers posed by power tools. Be careful of strain from vibration, noise hazards, and (if necessary) wear protective clothing and goggles. Do your research first!

rpimag.co/powertools

Toughening

Our skeleton is up and running and should be fine for light use, but it could certainly be toughened if you plan to use it a lot. The most obvious thing is boxing in the back. This should be fairly straightforward using a bit of dimensioned wood around the side and another sheet of engineered wood at the back. You'll have to make a hole for the USB cable. This boxing will protect the wires and electronics from any knocks.

Ready for action

That's our skeleton fully set up and working; let's program it. The following MicroPython code will test out all the components. It'll use the distance sensor to trigger the eyes lighting up and the limbs moving.

```
from machine import Pin, ADC
from time import sleep
import neopixel
from servo import Servo

left_hip = Servo(pin_id=0)
left_shoulder = Servo(pin_id=1)
right_hip = Servo(pin_id=2)
right_shoulder = Servo(pin_id=3)
servos = [left_hip, left_shoulder, right_hip, right_shoulder]
down_angle = [0, 0, 180, 180]
up_angle = [90, 90, 90, 90]

np = neopixel.NeoPixel(machine.Pin(4), 2)

distance = ADC(Pin(26))

legs_down = True

while True:
    print(distance.read_u16())
    if distance.read_u16() > 10000:
        np.fill((0, 255, 0))
        np.write()
        dancing = True
    else:
        np.fill((0, 0, 0))
        np.write()
        print("off")
        dancing = False

    if dancing:
        if legs_down:
            legs_down = False
            for i, servo in enumerate(servos):
                servo.write(down_angle[i])
        else:
            legs_down = True
            for i, servo in enumerate(servos):
                servo.write(up_angle[i])

    sleep(1)
```

If you save this to Pico as **main.py**, it will launch automatically when you power up Pico, so you just need to connect a USB power supply (capable of at least 2A), and your skeleton should activate.

At this point, we have a fully working person-activated skeleton, but there's one bit that you might need to change.

Let's work out the right distance value to trigger the action. This will depend on the place you want it physically set up. If it's in a corridor, alleyway, or other area where there's a wall

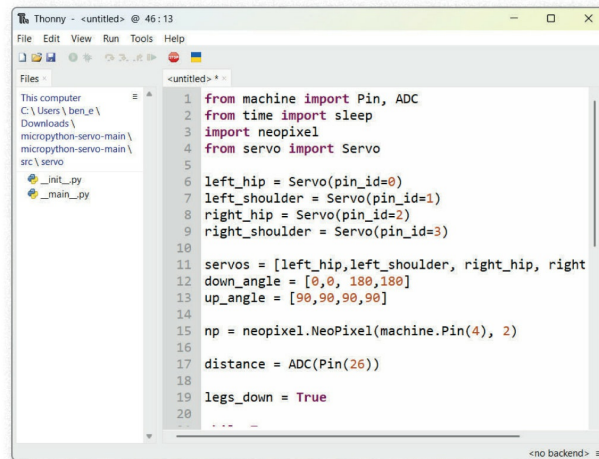
opposite, then you might need to make the number higher so that it's not triggered by the wall. Other than this, it's really up to you how close you want people to be before it activates. The closer people are to it, potentially, the scarier it might be to them, so you might want to take this into account, especially if there are likely to be young children.

In either case, the number you need to change is the one in this line:

```
if distance.read_u16() > 10000:
```

As the method suggests, it's an unsigned 16-bit number, so the maximum possible value is 65,535.

That's our animated Halloween skeleton set up and ready to scare the local trick-or-treaters.



The code running in Thonny IDE

The closer people are to it, the scarier it might be

**DOWNLOAD
THE FULL CODE:**



rpimag.co/github

skeleton_test.py

> Language: **MicroPython**

```
001. import time
002. from servo import Servo
003.
004. left_hip = Servo(pin_id=0)
005. left_shoulder = Servo(pin_id=1)
006. right_hip = Servo(pin_id=2)
007. right_shoulder = Servo(pin_id=3)
008.
```



```

009. servos = [left_hip, left_shoulder, right_hip, right_shoulder]
010. down_angle = [180, 180, 0, 0]
011. up_angle = [90, 90, 90, 90]
012.
013. while True:
014.     print("down")
015.     for i, servo in enumerate(servos):
016.         servo.write(down_angle[i])
017.
018.         time.sleep(2)
019.
020.     print("up")
021.
022.     for i, servo in enumerate(servos):
023.         servo.write(up_angle[i])
024.
025.         time.sleep(2)

```

Going further

Perhaps the most obvious way of taking this project further is to add some audio. This could be a creepy laugh or some ominous music. There are some options for playing audio directly from Pico in MicroPython, but the easiest choice would be to add an audio board like DFPlayer.

As well as audio, you could up the lighting effects. More LEDs would be fun, and that's fairly straightforward as you'd only need an additional power supply. Anything that can output 5V at around 4A or 5A should be able to handle quite a few LEDs as well as the current setup.

The same basic technique for using servos to make things move and LEDs to light things up can be added to all manner of Halloween props. It could be a ghost that is hoisted by a servo arm, or a pumpkin whose eyes light up when someone approaches. Your imagination is the only limit here.

We'd love to see what you come up with. Be sure to tag us in your creations on social media.

**DOWNLOAD
THE FULL CODE:**



rpimag.co/github

skeleton.py

> Language: **MicroPython**

```

001. from machine import Pin, ADC
002. from time import sleep
003. import neopixel
004. from servo import Servo
005.
006. left_hip = Servo(pin_id=0)
007. left_shoulder = Servo(pin_id=1)
008. right_hip = Servo(pin_id=2)

```

```
009. right_shoulder = Servo(pin_id=3)
010.
011. servos = [left_hip, left_shoulder, right_hip, right_shoulder]
012. down_angle = [0, 0, 180, 180]
013. up_angle = [90, 90, 90, 90]
014.
015. np = neopixel.NeoPixel(machine.Pin(4), 2)
016.
017. distance = ADC(Pin(26))
018.
019. legs_down = True
020.
021. while True:
022.     print(distance.read_u16())
023.     if distance.read_u16() > 10000:
024.         np.fill((0, 255, 0))
025.         np.write()
026.         dancing = True
027.     else:
028.         np.fill((0, 0, 0))
029.         np.write()
030.         print("off")
031.         dancing = False
032.     if dancing:
033.         if legs_down:
034.             legs_down = False
035.             for i, servo in enumerate(servos):
036.                 servo.write(down_angle[i])
037.         else:
038.             legs_down = True
039.             for i, servo in enumerate(servos):
040.                 servo.write(up_angle[i])
041.
042.
043.
044.
045.     sleep(1)
```



Get Started with MicroPython on Raspberry Pi Pico

The Official Raspberry Pi Pico Guide

Learn how to use your new Raspberry Pi Pico microcontroller board and program it using MicroPython. Connect hardware to make your Pico interact with the world around it. Create your own electro-mechanical projects, whether for fun or to make your life easier

- Set up your Raspberry Pi Pico and start using it
- Start writing programs using MicroPython
- Control and sense electronic components
- Discover how to use Pico's unique Programmable IO

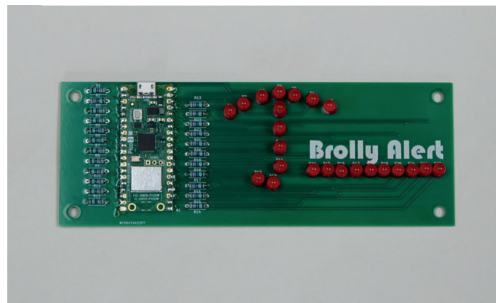
BUY ONLINE: rpimag.co/picobook

Inspirations

A collection of quite brilliant projects to dazzle and inspire

[Design your first PCB](#)

[Build a rain monitor using a Raspberry Pi Pico W](#)



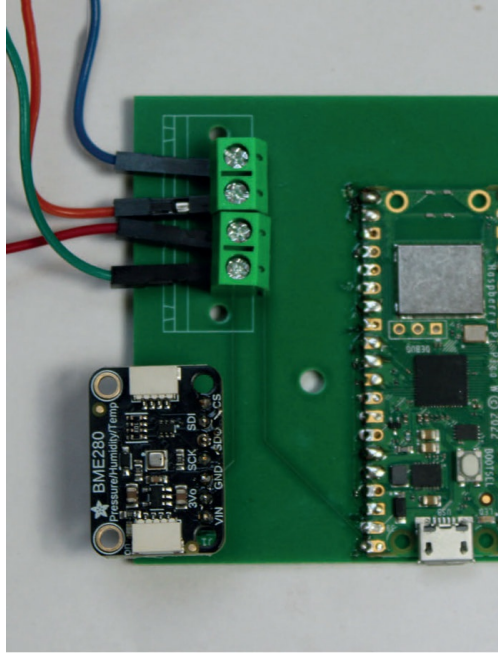
[Industrial 3D printing](#)

[Print your designs in professional materials, at pocket money prices](#)



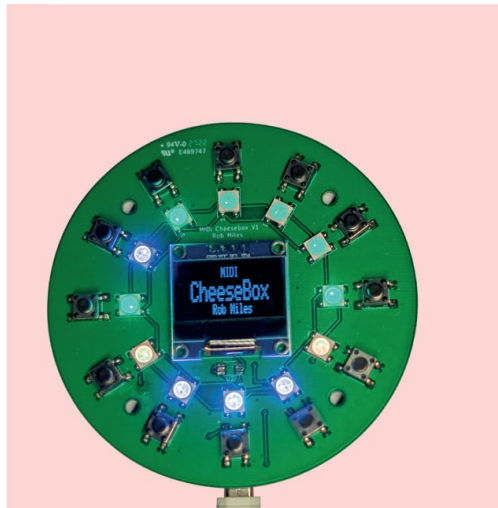
[Manufacture a complete build](#)

[Send your design away to a factory, get a working prototype back – it's like magic!](#)



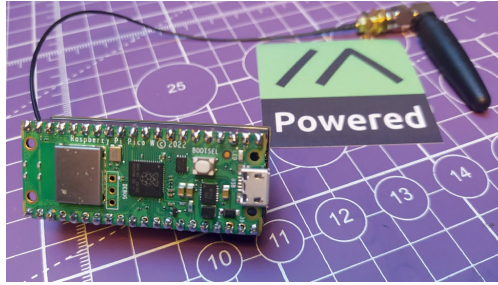
[Bake your own circuits](#)

[Build PCBs like the pros with surface mount components, solder paste... and an oven](#)



[Meshtastic part 1](#)

[Create a long-range communication device using a Raspberry Pi Pico W](#)



Meshtastic part 2

Add solar power to your Meshtastic node so you can leave it out in the wild



Build an underwater rover part 1

Make a submarine out of PCB piping and 3D printed parts



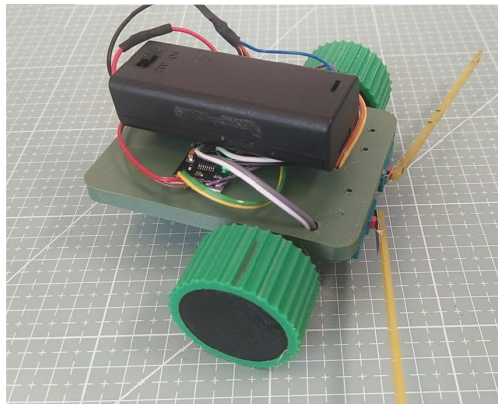
Build an underwater rover part 2

Add drive control systems and a camera to your underwater rover



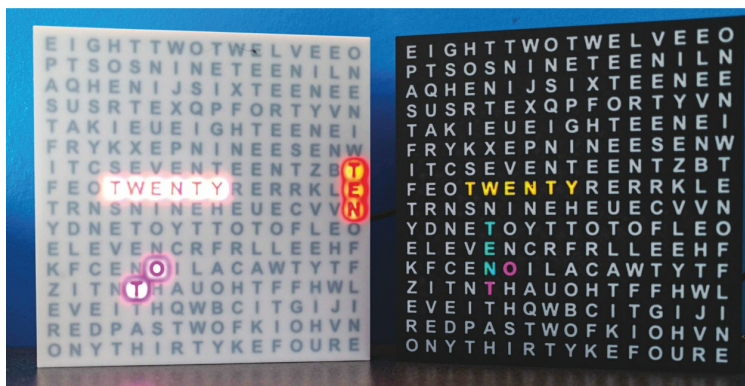
Simple sensor robot

Robots don't have to be complicated – here's a basic one that you can build from scratch



Wordsearch clock

Tell the time with a wordsearch, RGB LEDs and a Raspberry Pi Pico



Design your first PCB

Use Pico W to alert you to rain and never forget your brolly again



Maker

Ben Everard

Ben is a freelance writer and artist. Both occupations involve an abundance of LEDs.



glowingart.co.uk

Many electronics projects begin life on a breadboard. It's an easy way to test things out as you build your circuits. You can adjust, change, develop, and build things. However, what do you do when you want to make your projects permanent? You could leave it on a breadboard, but sooner or later, things will fall out. There is protoboard – PCB that's just made up of through-holes that you can solder components into. This can work, but it's a bit messy and it is awkward to use with surface-mount parts. The solution is to design your own PCB.

This sounds like a complex process, but it can be surprisingly straightforward. PCB design tools are becoming easier to use, and there are loads of PCB manufacturers that are well set up for hobbyists.

In this article, we're going to design and fabricate a simple PCB. It's a 'Brolly Monitor' with a Pico W that grabs a weather forecast. If there's rain forecast, it lights up some LEDs in the shape of an umbrella. You can mount this PCB near your door and hopefully, it'll remind you to grab your brolly if there's rain forecast.

Let's take a look at how to do this with KiCad, an open-source PCB design tool that runs on Windows, Linux, and macOS that you can download from kicad.org. The latest version is 10 and if you have an older version already installed, we'd recommend you upgrade before continuing.

Designing a PCB with KiCad is a four-stage process:

- Design your schematic

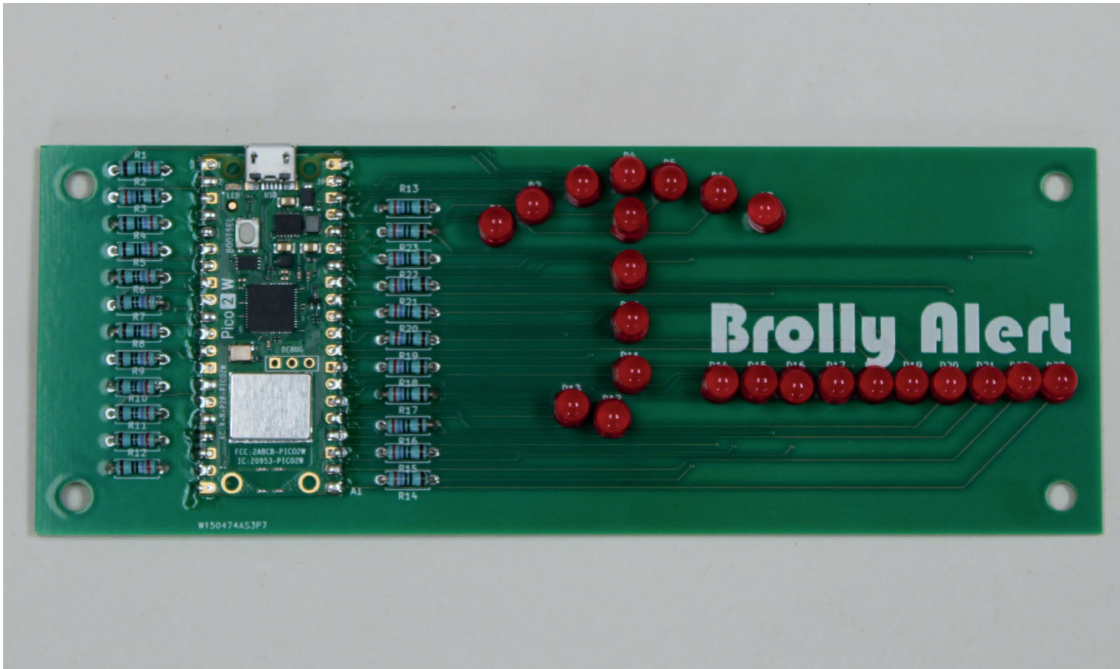
- Assign footprints to the parts you've used

- Lay out the PCB

- Export the files for manufacture

Let's look at each one in turn.

This is our board electrically complete now, but we need to add a couple of bits – some mounting holes and some text



Warning!

Hot solder

Soldering irons get very hot, and stay hot for a long time after they're unplugged. Make sure that you put the iron in the stand when you're not using it and don't touch the metal parts – even after it's unplugged.

rpimag.co/soldering

Schematic

A schematic diagram shows the logical arrangement of electronic parts, and abstracts away a lot of the detail. For example, when you draw an LED in a schematic, you don't usually mark what colour it is, what size it is, whether it's through-hole or surface-mount, or any of that detail.

Similarly, the position of the objects on a schematic isn't the same as their position in the real world. All we're interested in is the type of object and the connections between them.

In KiCad, schematics are created using the aptly named Schematic Editor, so open KiCad, create a new project, and click on Schematic Editor. This will open a new window with a blank schematic in it.

The first thing we need is some components: a Pico W, an LED, and a resistor. To place a component (known as a symbol in the Schematic Editor), click on the op-amp icon in the righthand toolbar (a triangle with a '+' and in it). This will open up a dialog box where you can choose what symbol you want. Search for Pico and select RaspberryPi_Pico_W. Since they are pin-compatible, this works for either a Pico W or a Pico 2 W. Even though the Pico W is pin-compatible with a non-wireless Pico, we'd recommend using the specific Pico W symbol as the associated footprint has keep-out areas that tell the PCB design tool not to block the Wi-Fi signal.

Once you've highlighted the Pico W, you can click OK to return to the schematic. The symbol will be attached to the mouse pointer, so you can place it where you want to and then left-click to deposit it on the page. Now we do the same for a resistor (search resistor and select device R), and LED (search LED and select device LED).

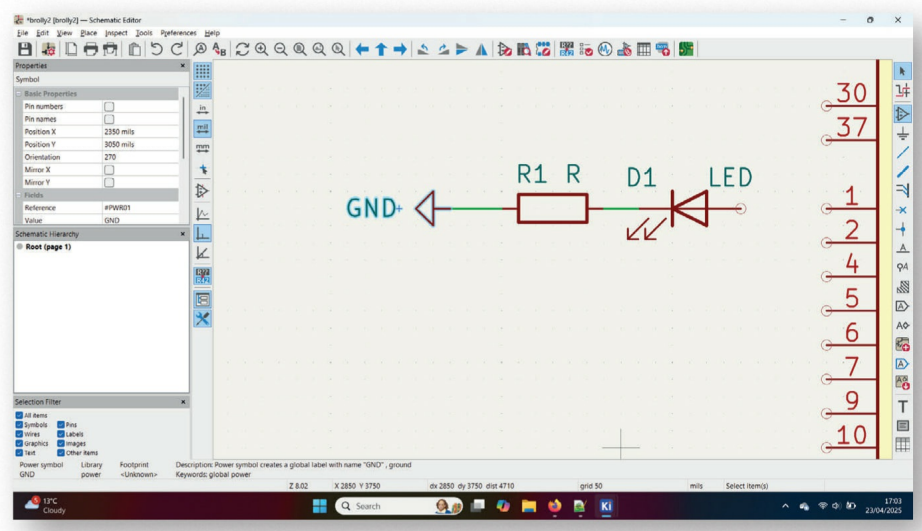


Figure 1: All budding PCB designers start with an LED and a resistor – it's tradition

We'll need lots of LEDs and resistors, but only add one for now. Each LED will be attached to a GPIO pin on one side and a resistor on the other which will in turn connect to ground.

With these components added, you can switch back to the mouse pointer tool, and we'll organise the things we need. Highlight the LED by clicking in the middle of it (make sure the entire LED is highlighted as it's possible to only select part of a symbol), then click and drag it so it's near GPIO 0. The resistor is by default placed vertically, and we want it horizontal, so highlight it, and click the rotation icon on the top toolbar until it's horizontal. Now place it just to the left of the LED. We want to connect the resistor to ground, so select the ground icon (a line arrow pointing down) on the right-hand toolbar. This will open a dialog box that can be used to create a variety of power connections. Click to expand the power section, then select GND and click OK, then place the ground just to the left of the resistor. Again, rotating this to horizontal will help keep things compact.

If you zoom in so you can see the LED closely, you'll see that it's labelled LED and D1. These labels are a little way away from the symbol, and since we're about to create quite a few copies, it's a bit tidier if we click and drag them close to the symbol. This time, click into the characters rather than the symbol itself and you can drag them in. Do the same with the resistor.

Now we're ready to connect the LED, resistor, and wire together. Select the Draw Wires tool from the right-hand toolbox (a thin blue line), and you can create the connections. Each component has a small circle where a connection can join. Click into one of these and then again on the circle on the next component.

Once all the connections are made, you should have something that looks like [Figure 1](#).

Growing the circuit

You can now select this mini circuit of LED, resistor, and ground by clicking and dragging over the lot and copy-paste it until you have 21 identical circuits. Position them so that they're close to the GPIO pins and connect each LED to a GPIO pin 0–22.

The one final thing we need to do is connect the ground on Pico to ground. We need to do this because the GND we've connected the resistors to isn't really a ground, it's what's called a network (or net for short). This means that everything in the same net gets connected together. It's very common to have one called ground like we have done. However, unless we connect the ground net to an actual ground connection, it's not really ground. We'll be supplying power through our Pico's USB port, so this is where our ground will come from.

The final schematic should look like [Figure 2](#).

QUICK TIP

The 'Ki' part of KiCad can be pronounced either like 'key' or to rhyme with 'buy'. Both are widely used.

Code

We've written a simple MicroPython script (**broolly.py**, listed on the last page of this tutorial) to run this PCB, which you can download from rpimag.co/broollypy. You'll just need to put in your own latitude and longitude, as well as your Wi-Fi credentials. It grabs a forecast from Open Meteo, which is a weather forecasting service that's free for noncommercial use.

Our MicroPython script is quite simple. It just lights up for rain forecast and off for no rain. Each LED is individually controlled, so you can program your own patterns.

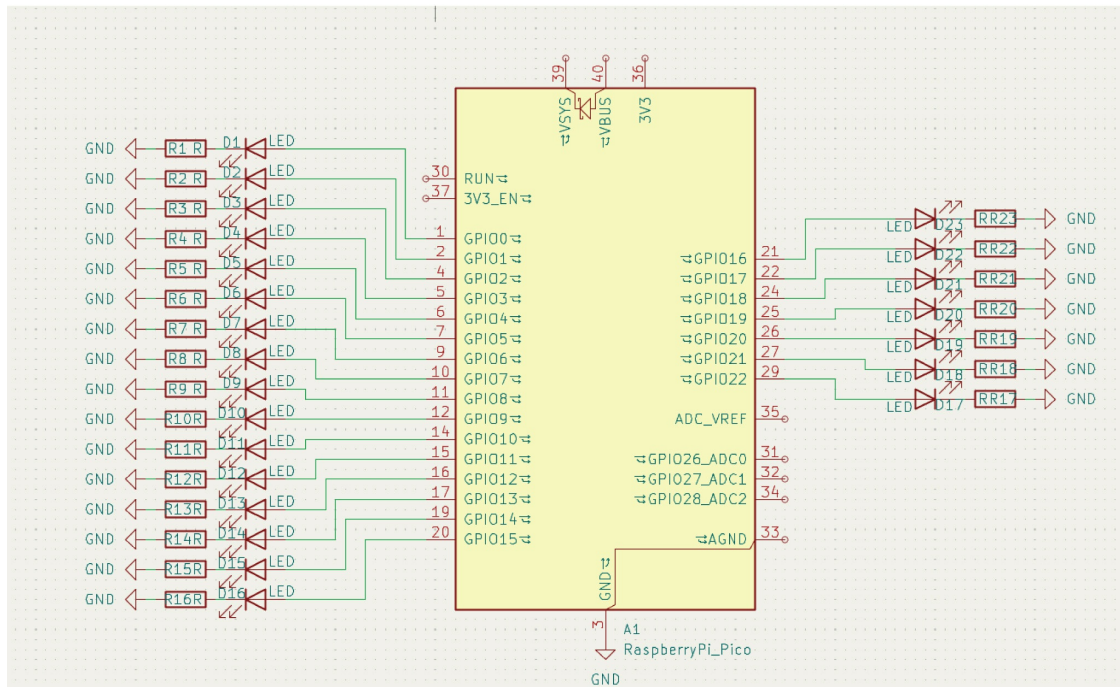


Figure 2: Part of drawing a good schematic is keeping things tidy

Leaving footprints

After designing our schematic, we need to assign components to the symbols. For example, we've placed LEDs, but we haven't said what physical package these LEDs will be in. Similarly with the resistors. Click on the Assign Footprints button in the top toolbar and in the new window, you can give a physical 'footprint' (this means the actual PCB shape) to the specific component. You can use whatever type of resistor and LED you want, but we went with through-hole LEDs (footprint 'LED_ D3.0mm') and resistors (footprint 'R_Axial_DIN207_L6.3mm_D2.5mm_P7.62mm_Horizontal').

Getting on track

Now we've defined our circuit, let's design the physical PCB layout. Save your project, then click on the PCB icon in the top toolbar (Switch to PCB Editor). This will give you a blank PCB. You can bring in the components from your schematic with Tools > Update PCB from Schematic. Click Update PCB in the dialog to bring things in. You should now have a bunch of components dumped fairly haphazardly on your PCB. You'll also see a set of pale blue lines showing the connections that you need to make.

It's time to arrange things how you'd like them. Usually, at this point in the PCB design process, you want to think about how things will connect and how to organise them to make that easy. However, we're going to arrange things how we'd like them to look and then worry about connections later.

First, drag and drop the components where you'd like them. We're going to place all the components on the top layer of the PCB, so make sure F.Cu (for Front Copper) is selected in the layers box on the left-hand side.

Once you've got them where you want them, you need to define the board outline. Select the `edge.cuts` layer and then the Draw Rectangles tool and draw a box around where you want the edge of your PCB to be. You want the USB port on your Pico to be on the edge of the board so that there's space for the plug, and you want to make sure that there's a bit of space on the left and right for mounting holes that we'll add later (you can adjust the size of the PCB if you need to later on).

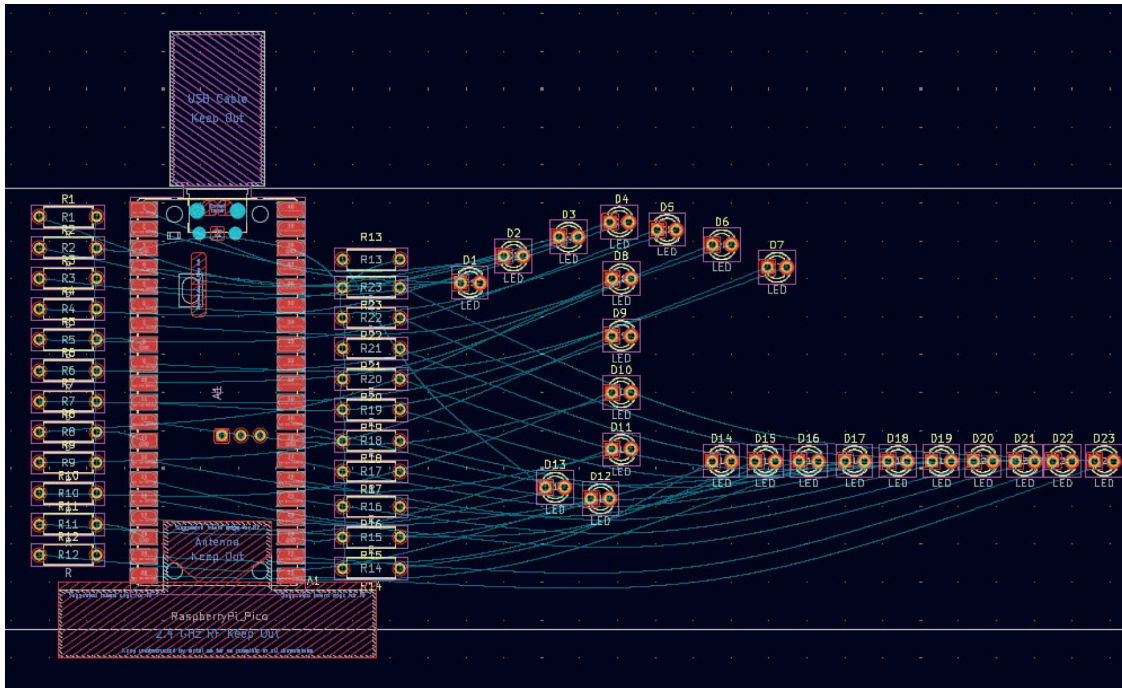


Figure 3: You don't have to lay out your components like this – it's up to you

Your PCB should now look something like [Figure 3](#). Everything's where we want it, but there are no traces joining the components together. Before we get to the traces, we'll connect up the ground. This can be done with traces and for a PCB like this, that would work fine. However, another option is to make a 'ground plane'. This is where you dedicate one side of the board to being just a ground connection. Across the whole thing, there's nothing but ground (actually, we'll carve a few bits out for traces in the next step, but it's mostly all ground). In this introductory article, we're not going to delve into the physics of it too much, but it's generally good practice to have a ground plane.

We're going to use the back layer as the ground plane, so select B.Cu in the layers list, then use the Draw Filled Zone tool (the blue polygon in the right-hand toolbar) and click around the outside of the PCB. The exact shape doesn't matter too much as long as it encompasses the whole PCB (going outside it is fine). After the first click, you'll be asked to select the net and you should choose GND. When you're finished, double-click to close the area.

Going further

This article is a really quick introduction to getting a PCB designed. We've cut a few corners to focus on getting a working product quickly and simply. As you go further in your PCB design journey, you'll need to learn to use more of the features of KiCad. The book *Design an RP2040 Board with KiCad* is designed to help you do just that.

Once you've created the outline, you can switch back to the mouse pointer tool and right-click on the new outline and select Zones > Fill All Zones. Most of the PCB should now turn blue to show that the back layer is mostly copper (if it turns a different colour, you probably have the wrong layer selected). If you zoom in, you'll see that the ground connections are all connected to this plane, but other pins aren't.

Now let's draw the rest of the traces. You can do this manually, but we've got quite a lot of traces and they're not particularly challenging ones (by which we mean they're not high-speed or likely to have problems caused by electrical interference), so it's a bit easier to let the autorouter deal with this for us. KiCad doesn't have a built-in autorouter, so we'll need to use a plug-in. Switch back to the main KiCad window (not the PCB or Schematic Editor) and click on Plugin and Content Manager. Search for Freerouting, click on Install and then on Apply Pending Changes. Freerouting does require Java to be installed, but it can do this automatically on the first run, so don't worry about it now.

Once it's installed, switch back to the PCB Editor and go to Tools, External Plugins > Freerouting. The router needs some guidance for things like how wide to make each trace and how close together traces can be put. It will get this directly from the Design Rules Checker (DRC). We're not going to look at this here because the defaults are fine for us, but as you progress on your PCB making journey, you might find that you have to adjust these values for things like high current traces, or the capabilities of a particular PCB manufacturer.

A new window will flash up and you'll see it gradually filling in the traces. Once it's filled, your PCB should be mostly complete, but the ground plane will now need to be adjusted to take account of some of the traces going across the back. Right-click on the outline of the fill zone and again use the Fill All Zones option to regenerate it. You should now see that any traces on the rear of the board (ones in blue rather than red) have a blank outline where the ground plane is broken.

This is our board electrically complete now, but we need to add a couple of bits – some mounting holes and some text. Mounting holes are footprints, so click on the Place Footprints tool (shaped like a chip in the right-hand toolbar). Search for 'mounting' and you should have a range of options. Most are sized with metric machine screw sizes. We're going to use wood screws to mount our PCB, so we needed a fairly large hole. Mounting holes often have an electrical connection to ground, but we don't want that, so we've selected MountingHole_4.3mm_M4_ISO7380. You can then place the component. We want four of these – one in each corner. You should be able to do this without breaking any traces, but you can expand the PCB outline if you need to (though make sure to refill the ground plane if you do).

The mounting holes have two bits of text with them describing the components. We deleted these to make it a bit tidier. We've not worried about the exact position of the holes and aimed to keep them symmetrical and looking correct. However, if you're planning on matching the PCB with a 3D print or other automatically fabricated part, you'll need to place them precisely.

For the final bit of text, we just wanted to highlight what the PCB actually was. First, select

the front silkscreen layer (f.silkscreen) and then the text tool. You'll be able to edit the text, select the size, and then place it on the PCB. You should, finally, have a PCB that's something like [Figure 4](#). You can check everything looks good by going to View > 3D viewer. This gives you a 3D rendering of your PCB and is a great way of checking that you've added everything to the right layer.

Factory files

Now we've got everything set up how we want it, we need to create some files to upload to our PCB manufacturer, and the standard for this is Gerber. A Gerber file is actually a zip file containing separate files for each layer of our PCB and a drill file for the holes.

Most PCB manufacturers use Gerber files, but unfortunately, they're not completely standard and there are a few options you might need to change. We used PCBWay, and the defaults worked fine for us. However, if you're using a different manufacturer, take a look at their documentation for what settings they need.

Project files

We'd strongly recommend that you go through the process of designing the board in KiCad yourself. However, if you want to compare your work to ours, the KiCad files are available at rpimag.co/brollykicad

You should find both the schematic and the PCB design there.

Go to File > Fabrication Outputs > Gerbers. First, select an output directory. This should be an empty directory. Give it a sensible name and maybe add a version number into the name just in case you end up needing to redo it. Click Generate Drill Files > Generate, then Plot. Together, this should create all the files you need. Finally, you just need to compress the directory to a zip file using your operating system's tools.

The process for getting it manufactured is a bit different for each PCB manufacturer. We're using PCBWay, so head to pcbway.com, then click on Instant Quote and Quick Order PCBs. Here you'll be able to upload the Gerber file zip and see a render of the PCB. If this doesn't match what you saw in the KiCad render, then stop and take a look first. If you can't work out why, use PCBWay's chat feature to ask for help – we've found them very helpful in the past.

There are a lot of options, but for the most part, the defaults are fine. The only one we changed is the Surface Finish to HASL Lead Free. Lead in hobbyist electronics isn't always a huge concern, but the price difference is pretty minimal, so we prefer to remove the health risk, especially on an exposed PCB.

That's all there is to it. Select your postage, and complete the purchase. It can take a week or two to get your PCBs (depending on the postage selected). Once they arrive, solder them up and program your Pico (see 'Code' box), then enjoy the smug feeling of always having your brolly when you need it. 🍂

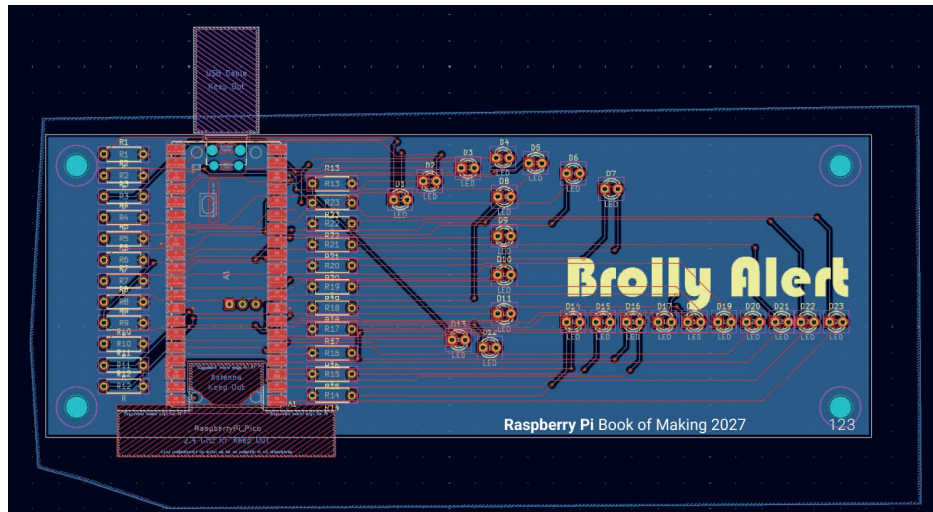


Figure 4: At this point, you can inspect the PCB to make sure all the connections are there

**DOWNLOAD
THE FULL CODE:**



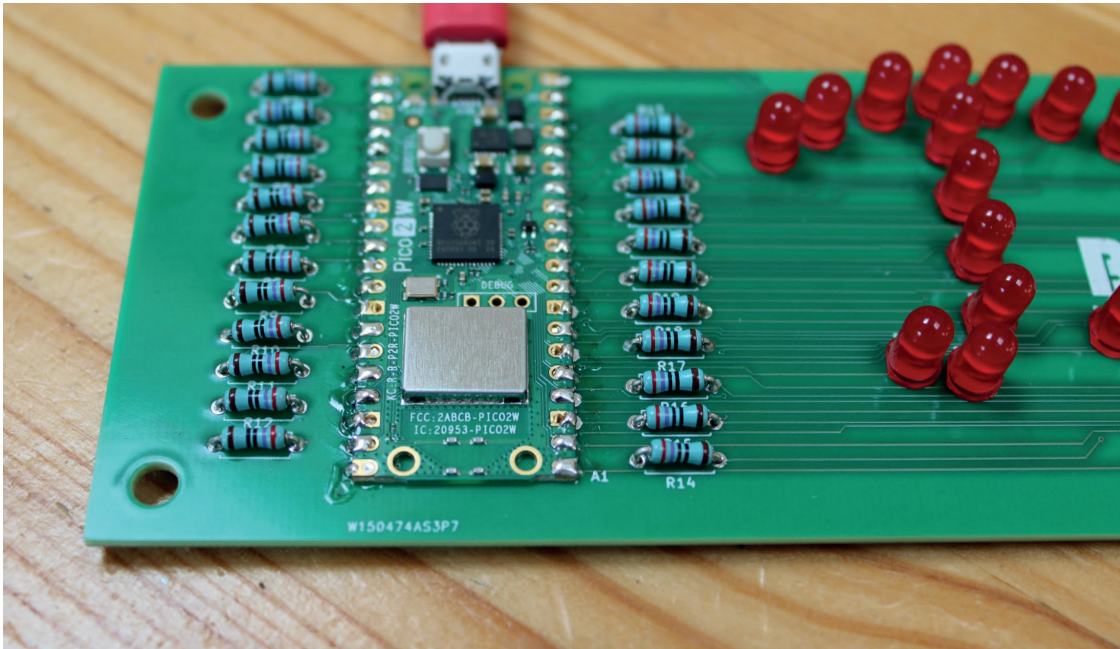
rpimag.co/brollypy

brolly.py

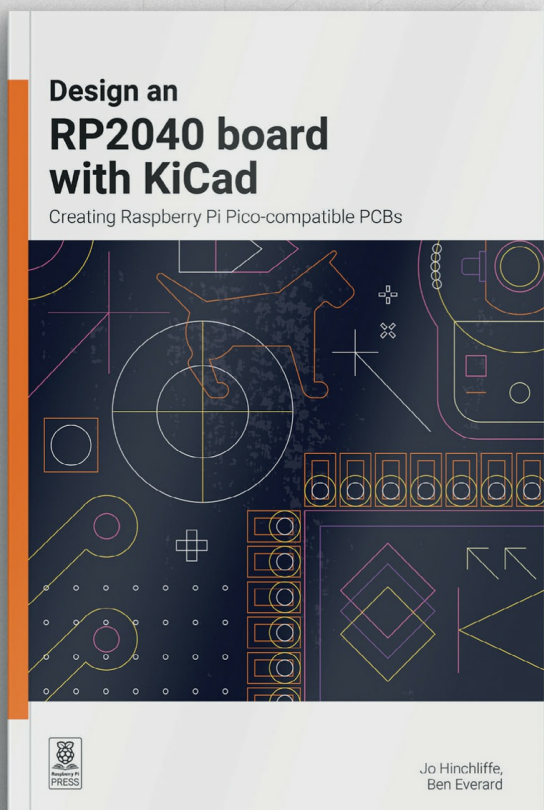
> Language: **MicroPython**

```
001. import machine
002. import time
003. import network
004. import socket
005. import urequests as requests
006. import json
007.
008. ssid = 'YOUR_WIFI_SSID'
009. password = 'YOUR_WIFI_PASSWORD'
010.
011. lat = "52.52"
012. long = "13.41"
013.
014. url = "https://api.open-meteo.com/v1/ forecast?latitude="+lat+"&longitude="+long+
"&daily=precipitation_probability_max&time zone=Europe%2FLondon&forecast_days=1"
015.
016. wlan = network.WLAN(network.STA_IF)
017. wlan.active(True)
018. wlan.connect(ssid, password)
019. while wlan.isconnected() == False:
020.     print('Waiting for connection...')
021.     time.sleep(1)
022.
023. print(wlan.ifconfig())
024.
025. pins = []
026. for i in range(23):
027.     pins.append(machine.Pin(i, machine.Pin. OUT))
028.
```

```
029.  
030. while True:  
031.     res = requests.get(url=url)  
032.     if (float(res.json()['daily'] ['precipitation_probability_max'])[0])) > 0.3:  
033.         #run for approx one hour  
034.         for pin in pins:  
035.             pin.on()  
036.     else:  
037.         for pin in pins:  
038.             pin.off()  
039.     time.sleep(60*60)
```



The standard footprint for Pico W is surface-mount, but it's not particularly tricky to solder



KiCad is an amazing piece of free and open source software that allows anyone, with some time and effort, to make high-quality PCB designs.

- *Create a schematic for a microcontroller board using Raspberry Pi's RP2040*
- *Select the right components*
- *Customise the hardware for your needs*
- *Lay out and route the PCB design*
- *Prepare your board for manufacture and assembly*
- *Write software to get your design working*

Buy online: rpimag.co/kicad2040

Industrial 3D printing for hobbyists

Get access to high-end 3D-printed parts without having to buy a new 3D printer



Maker

Ben Everard

Ben 3D-prints things that should not be 3D printed. He once had to spend a day with his hands in his pockets because his entirely 3D-printed belt snapped while crawling through a cave.



glowingart.co.uk

By far the most popular 3D printing technology is Fused Deposit Modelling (FDM). In this process, a print head heats up plastic and squirts the molten material through a nozzle. It moves around the print volume, building up your print layer-by-layer. This is a great, well tried-and-tested method that millions of makers use every day. It's (fairly) reliable, (fairly) easy to use, and (fairly) cheap to run. However, while the parts can be great for some uses, if you have demanding requirements, they can fall a little short. In this article, we're going to look at some industrial 3D printing techniques that are now available to makers.

Performance plastic

Nylon can be printed on an FDM printer, but it's a particularly horrible material to work with. It warps a preposterous amount, will suck moisture out of the air at any given opportunity, and will just generally make your life miserable. It's far more suited to a style of printing called Selective Laser Sintering or SLS (this is different to Stereolithography SLA). In the SLS process, a layer of powder is spread over a surface, then a laser melts the material in the first layer. The bed then drops and another layer of powder is spread, which is then melted by the laser. This process repeats until your object is fully built up.

Nylon SLS parts are robust, durable, and can have complex shapes (as overhangs are automatically supported by unsintered powder). They're also well suited to medium-run manufacturing as they can batch-print parts vertically as well as across the print bed.

Like nylon, TPU can be printed on FDM printers, but with some caveats. TPU has a habit of bending as it prints, meaning thin features can end up as a succession of blobs. With careful

slicing, chunky designs are possible, but more delicate designs can be challenging. Again, SLS comes to the rescue, allowing much more complicated and intricate parts to be printed of this flexible material.

When 3D-printing, it's often possible to print a delicate part that might survive, but also might not



If you just want one part with a particular technology – such as a transparent case for your Raspberry Pi – getting the part made for you can be cheaper and easier than buying a new printer (or even a bottle of resin)

The problem with SLS printers is that they're expensive and messy. Fortunately, you don't need to have an SLS printer to create SLS prints – you just need a company that does.

There are a few companies that offer industrial-level printing services to makers. We're going to take a look at PCBWay.

Head to [pcbway.com](https://www.pcbway.com), then use the CNC | 3D Printing > 3D Printing menu to get to the quote page. Most of the time, this will give a full quote for your part, but if you have unusual requirements, you might get 'RFQ' as the price total. This stands for Request For Quotation and means that someone will get back to you with the final price.

PEEK

Polyether ether ketone (PEEK) is perhaps the ultimate FDM material. It's strong, stable at high temperatures, and

resistant to a lot of chemicals. It is, however, an absolute pain to print. We have seen some people manage something approximating successful prints in PEEK on home 3D printers, but really it needs a very high-end setup to work well, and even then it can take some dialling in. If you just need the full capabilities of PEEK for a few parts, it's probably not worth the time and effort to do it yourself, so why not let someone else do the hard work? After all, it's eye-wateringly expensive for a spool (expect to pay over £500 per kg), so it only takes a few failed prints for it to be cheaper for someone else to do it who can get it right first time.



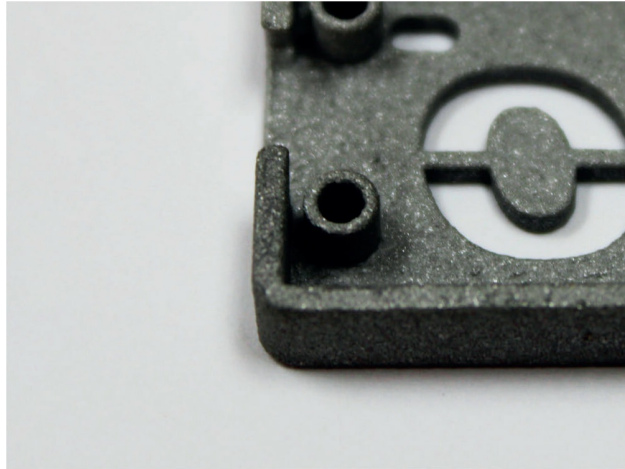
If you want ultimate protection for your parts, 3D-printed metal is incredibly strong – just be careful that you don't accidentally short-circuit any of your components

First you need to upload your design file. This will usually be your STL or STEP rather than a pre-sliced file. You don't have as much control over the slicing as you might do if you have your own printer, but this is usually a good thing. It means that parts are printed with tried-and-tested slicing settings. If you do need anything specific, you can add it to the Other Special Requests box at the bottom of the page, or open up a chat session with the support.

Next, you need to enter the quantity of parts you need. This is the number of copies of the same part you want. If you want different parts, you need to add them separately to your order. The cost per part does decrease as you order more copies, but not significantly.

Selecting the material will give you options for the process, colour and surface treatment. For example, if you choose Nylon, you'll get the option for PA12, HP-PA-12 and Glass Fibre Nylon. If you're not sure what you want, click on one, then in the Selected Material box you can click on Show Material Description to find out more including viewing the material data sheet.

There is the option to upload a technical drawing, but for most cases, the 3D design uploaded first should contain all the information needed. However, a technical drawing is needed if you want PCBWay to tap threads, add heat-set inserts, silkscreen or laser engrave your parts, or if you want PCBWay to assemble your parts. With these options, it's easy to see how the 3D print service from PCBWay could be part of a small-scale manufacturing line.



SLS metal prints have a rough surface texture, and feel a little strange to the touch

CNC machining

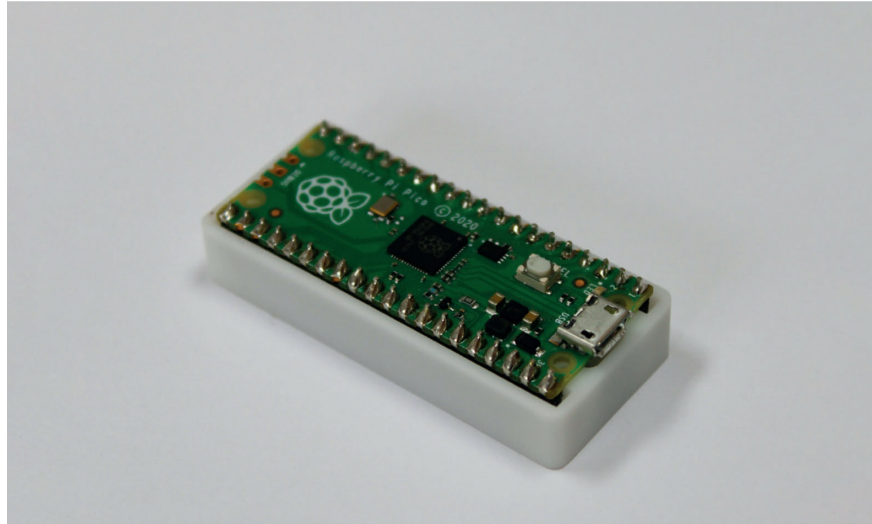
3D printing is a type of 'additive manufacturing'. This means that you start with nothing and add material to build up to your desired shape. An alternative is subtractive manufacturing, where you start with a block of material and remove material until you have the part you want.

Just as 3D printing has limits on what it can produce, so does CNC machining. The limits depend on the machine, but on common mills, the limits depend on the number of axes. For example, in a three-axis CNC mill, the tool head (the bit that holds the spinning bit that removes material) can move like the print head on a 3D printer. This can remove material from the top of the part, but not underneath. As you add more axes to the machine, you can make more complex shapes.

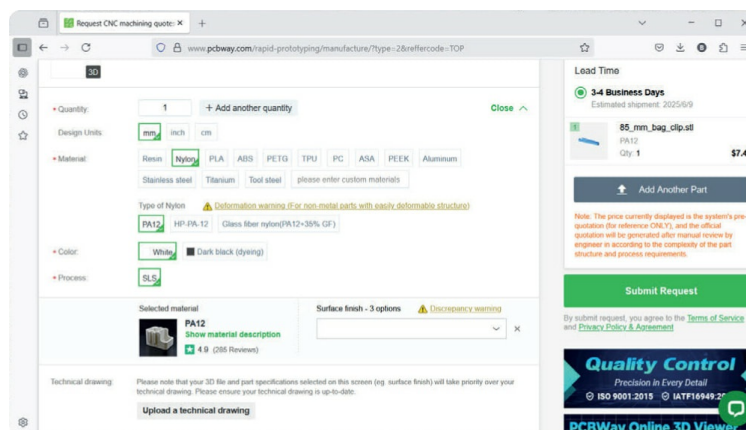
You can get quotes for CNC machining parts online at [PCBWay](#) and other sites.

When 3D-printing, it's often possible to print a delicate part that might survive, but also might not. [PCBWay](#) has minimum wall thicknesses and thread pitch (if you're printing a threaded hole). If you want to push these limits, you can, but you just need to be prepared to accept the risk that they may not work. This author has gone beyond the limits a few times and nothing untoward has happened, but it's up to you whether or not to take the risk. If you want to, make sure the Wall Thickness Risk Taken or Non-Standard Printed Threads Risk Taken boxes are checked. The actual minimums vary by printing technology. Hover over the 'E.G.' text next to each option for more information.

The final option is perhaps the strangest. You need to select a Product Description category. This is for customs purposes when your prints are sent to you. Depending on where in the world you live, selecting different options may impact what happens when the package arrives in your country.



SLS nylon is significantly tougher than most FDM 3D prints and can be a cost-effective way of doing a small run of plastic parts



The same part printed with different techniques can have startlingly different properties

With all that entered, you can submit your request. Most of the time, the final price will be the displayed price, but if your part is unusual, it sometimes comes out a bit differently. The final price is usually sent over within a day or so.

The price doesn't include delivery and you may also have to pay import duty. In some cases, you can pay the import duty directly through PCBWay and they'll get it delivered straight to you. In other cases, you'll have to organise this with your local customs.

Metal

Sooner or later, you'll probably want to make something out of metal. Whether this is for strength or purely looks (this author has 3D-printed jewellery previously), this technology is pretty hard to use at home or even in a small workshop. Again, industrial 3D printers come to our rescue. Quite a few metals can be 3D-printed, and PCBWay has options for Stainless Steel, Tool Steel, Titanium, and Aluminium.

The basic technique for this is known as Selective Laser Melting, and is quite similar to the SLS process used for nylon and TPU. There's a powder that's melted by lasers and built up layer

by layer. This does give quite a rough surface finish (though you can polish this up by hand if you wish to).

Commercial 3D printing for hobbyists is now at a point where it's a cost-effective way of getting access to advanced techniques. If you just need a few parts, it can be cheaper than buying a new spool of filament or bottle of resin, and save you hours of messing around with uncooperative materials. There will always be a place for home 3D printing, but this can be a great way to augment your builds with higher-end parts. 🟩

PolyJet

PolyJet is one of the more unusual 3D printing technologies. It's a bit like a mashup of an inkjet printer and a resin printer. A print head moves across the surface, squirting resin and then curing it with UV light.

The main reason a hobbyist might want PolyJet printing is that it has the best colour capabilities of any 3D printing technology. You can get a huge range of colours, including gradients, in a single print – far more than is possible with any other method you're likely to get access to.

Multi Jet Fusion

Multi Jet Fusion is similar to SLS in that the part is built up from powdered nylon layer-by-layer. However, it's done in a slightly different way. All the powder is heated to just under melting point, then a fusing agent is deposited in the areas that should be joined and a detailing agent on the edges.

The final parts are similar to SLS parts and typically done with nylon.

Manufacture a complete build

Use on-demand fabrication to create projects for you



Maker

Ben Everard

Ben is melting in the summer heat of the workshop, so is outsourcing his builds until the weather becomes more suitable.



glowingart.co.uk

In the last couple of articles in this series, we've looked at getting a PCB made, and getting 3D printing done commercially. In this issue, we're going to bring them together and get our project made commercially. We're going to build a weather station where the PCB and enclosure are made for us. All we have to do is solder the electronic components onto the PCB and program it.

Before we dive into the project, let's think a bit about why we might want to do this. We, as makers, have many tools and techniques available to us – including factories that can make things. What you need to do is decide what you want to do. There's a huge range of considerations that might be important to you in thinking about whether you want to use a factory or a more hands-on route.

For example, here are a few reasons why you might want to outsource your project to a factory:

One of the most challenging parts of a project like this is ensuring that everything fits together

If a professional final appearance is important to you

If you lack space, time, or tools to build things

You might get better longevity

You enjoy the process of design more than the process of building things

However, you might want to use a more DIY route if:

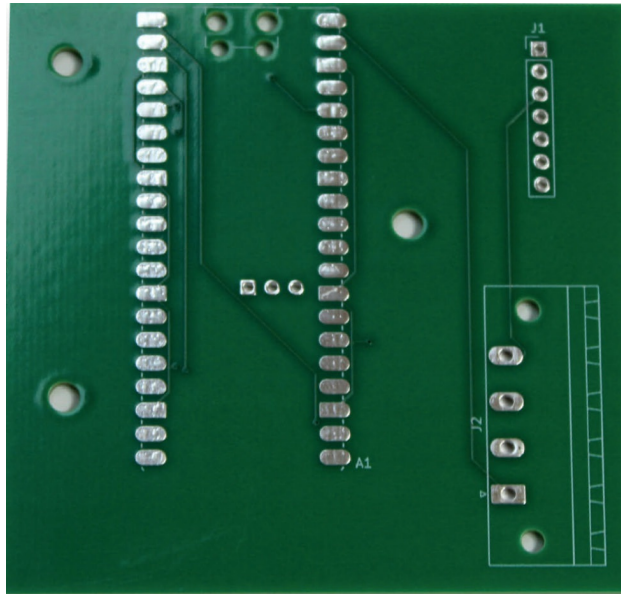
You enjoy the hands-on process of making things

You don't enjoy doing CAD

You like to hack things together as you go rather than finalising the design up front

Keeping costs low is a priority

There isn't a universal right-or-wrong here; it's about finding the process that works for you.



Our PCB as it came back from the factory



Warning!

Hot solder

Soldering irons get very hot, and stay hot for a long time after they're unplugged. Make sure that you put the iron in the stand when you're not using it and don't touch the metal parts – even after it's unplugged.

rpimag.co/soldering

Feeling board

Our weather station needs a PCB to hold the electronics and an enclosure to keep it safe outside. Let's start by looking at the PCB.

The electronics we're using are a Bosch BME280 temperature, humidity and pressure sensor, and a PMS5003 particulate matter (air pollution) sensor. We'll use a Raspberry Pi Pico W to control these and upload the results to an online repository.

We could design a PCB to hold the bare BME280 sensor, but it's a tiny device and quite hard to solder. If you like a challenge, then you might want to do this, but we took the easy

option and used a module that holds the BME280 and a few other components to help it work: an Adafruit BME280 module. You should be able to use one from another vendor, but the pinout might be different, so you'll need to check this and make any changes if necessary.

The PMS5003 connects via a cable that requires a particular socket that's a bit hard to source, so we've opted to cut the ribbon cable and instead put a set of screw terminals on the PCB. We don't need all the connections, just 5V, Ground, Rx, and Tx.

We won't go through the full design process, but it's a fairly simple PCB using the same method that we looked at in part 1 of this series. You can download the full design at rpimag.co/github.

Once the PCB has arrived, you'll need to solder everything together and wire it up. We soldered socket headers for the BME280 and then pushed it into place. The PMS5003 is wired up as follows:

There are four pins we need:

Power

Ground

Rx

Tx

These are the first, second, fifth, and fourth pins respectively on the PMS5003 (where pin 1 is closest to the centre of the device).

These connect to the PCB in the following order:

Ground

Tx

Rx

Power

...starting with the end closest to the USB port.

Boxed in

Now that we have our PCB, we need an enclosure. We've got the full range of industrial 3D printer materials to choose from, and we selected sintered nylon. This produces parts that are significantly tougher than those made on most desktop printers, so our weather station should survive even if it takes a few bumps along the way.

Our design is a box that's open at the bottom, has a sloped lid to shed rain, and includes two screw holes for mounting.

You could design something like this in any CAD package. We used FreeCAD because it's the tool we're most familiar with. You can get the design files from rpimag.co/github.



The PMS5003 is a low-cost particulate matter sensor that's great for DIY projects

Making it fit

One of the most challenging parts of a project like this is ensuring that everything fits together. In this case, that meant making sure the mounting holes for the bolts holding the PCB in place aligned properly, and that the enclosure was the right size to hold all the components.

FreeCAD does have the ability to import KiCad models; however, that would only work if all the parts we used in our PCB had 3D models associated with them, which ours don't.

In lieu of tools that make co-designing these parts easy, we've kept the interface as simple as possible: just three M2.5 holes. With fewer elements overall, there's less to keep track of, and less that can go wrong. That said, there are still a few traps that could derail the project if we're not careful.

We nearly sent the project off with screws that were impossible to tighten. It's all well and good making sure all the holes are aligned, but if there isn't enough clearance to fit a screwdriver in, it's all a bit pointless. Fitting components in three dimensions is significantly more complex than fitting components in two dimensions, so it's best to err on the side of caution and give yourself as much room as you can.

We had both parts made by PCBWay, which offers PCB fabrication and 3D printing services, and good maker support.

Cloud bound

Now that we've got our hardware sorted, let's move on to the software. We're using Pico W because we want to upload the sensor readings to the internet.

Of course, 'the internet' is a big place. We want somewhere to store the raw values and visualise them as charts – maybe even set alerts if a reading is particularly high (for example, if outdoor pollution is unusually bad).

There are a few options we could use for this. For example, Google Sheets has an API that

allows data to be sent from a device like the Pico W — but we're going to use Adafruit IO. It's a system built for makers to store sensor readings (among other things). The free tier is perfectly capable for our purposes.

You'll need an Adafruit account. If you've used any of the firm's services before, you'll already have one; if not, you can create one for free now. Head to adafruit.io and click on either Get Started For Free or Sign in. Once you're set up and logged in, you'll see a key icon in the top right. Click on this to see the API keys which you'll need shortly.

We're going to use CircuitPython for our code because it supports everything we're trying to use. You can download the firmware from circuitpython.org.

Install the following libraries from the CircuitPython Library bundle.

```
Adafruit_bme280
Adafruit_io
Adafruit_minimqtt
Adafruit_pm25
Adafruit_connection_manager.py
Adafruit_requests.py
Adafruit_ticks.py
```

Once these are installed, you can use the following weather station code to read the data and upload everything to Adafruit IO. You'll need to update the username, SSID, and password, then save the file as **code.py** on the CircuitPython device. It should then just work automatically, uploading data every minute – if there are any errors, they'll be reported on the serial console.

```
import adafruit_connection_manager
import socketpool
import adafruit_requests
import board
import wifi
import ssl
import busio
from adafruit_io.adafruit_io import IO_HTTP, AdafruitIO_RequestError
from adafruit_pm25.uart import PM25_UART
from adafruit_bme280 import basic
import time

#included here for brevity, but more secure to
#include them in environmental variables
aio_username = "XX"
aio_key = "XX"
ssid = "XX"
password = "XX"

#set up hardware
uart = busio.UART(board.GP0, board.GP1, baudrate=9600)
pm25 = PM25_UART(uart, None)
i2c = busio.I2C(board.GP3, board.GP2, frequency=100000)
bme280 = basic.Adafruit_BME280_I2C(i2c)

#connect to wifi
wifi.radio.connect(ssid, password)
pool = socketpool.SocketPool(wifi.radio)
requests = adafruit_requests.Session(pool, ssl.create_default_context())
```

```

#connect to Adafruit IO
io = IO_HTTP(aio_username, aio_key, requests)
print("connected to io")

try:
# get feed
    temp_feed = io.get_feed("temp")
    humid_feed = io.get_feed("humid")
    pm25_feed = io.get_feed("pm25")
except AdafruitIO_RequestError:
# if no feed exists, create one
    temp_feed = io.create_new_feed("temp")
    humid_feed = io.create_new_feed("humid")
    pm25_feed = io.create_new_feed("pm25")

while True:
    print("\nTemperature: %0.1f C" % bme280.temperature)
    print("Humidity: %0.1f %" % bme280.relative_humidity)
    io.send_data(temp_feed["key"], str(bme280.temperature))
    io.send_data(humid_feed["key"], str(bme280.relative_humidity))

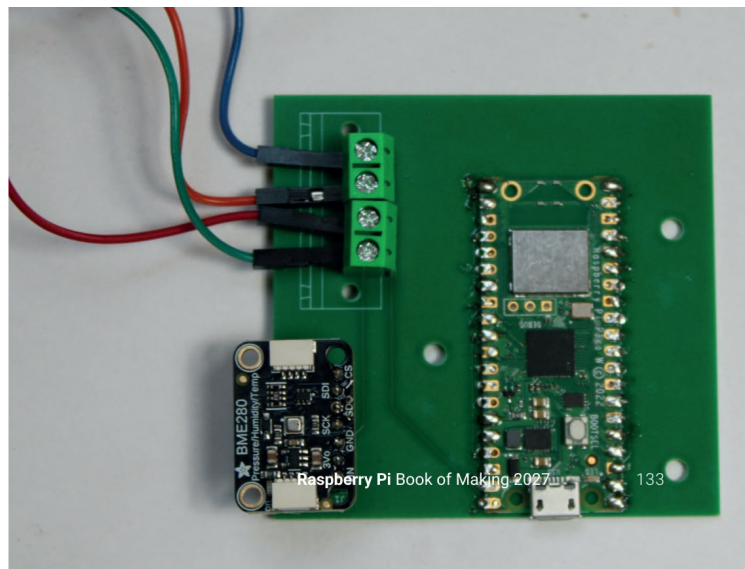
    aqdata = pm25.read()
    print("PM2.5: %0.1f" % aqdata["pm25 standard"])
    io.send_data(pm25_feed["key"], str(aqdata["pm25 standard"]))

    time.sleep(60)

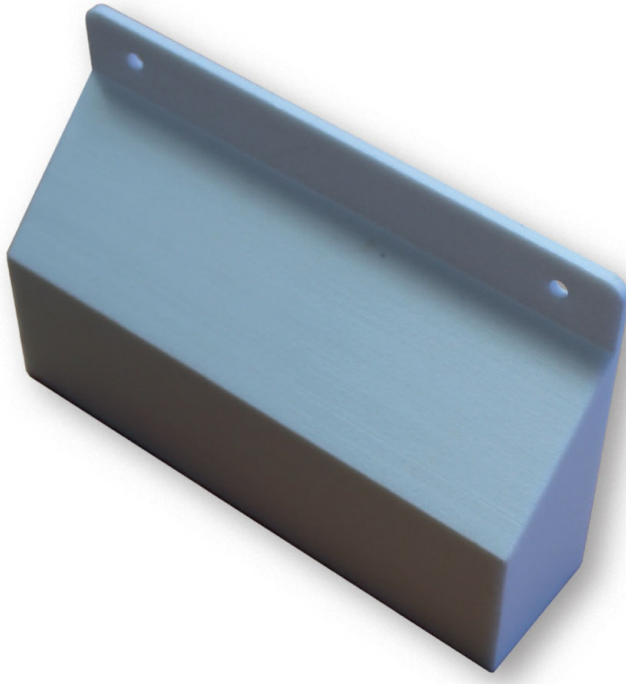
```

This code sends data to Adafruit IO, where it can be viewed by logging into the **adafruit.io** website and navigating to the appropriate feeds. From there, Adafruit IO provides tools to visualise the data using various formats such as charts and gauges.

In Adafruit IO, the script creates three separate feeds. To visualise this data with graphs, you can create a dashboard. We'll leave this as an exercise for the reader. It should be straightforward because Adafruit IO offers a user-friendly graphical interface for creating dashboards.



The soldering ended up a little messy as we had to desolder Pico and realign it after we'd soldered some of the pins



The 3D-printed nylon enclosure, ready to be used

In this project, we've designed our parts and sent them to be made at a factory. This approach has become affordable for one-off projects in recent years, and if you prefer the digital side (design and programming) to the physical side of making, then it's a great option. Once your designs are set up, you just send them off and then wait for everything to be delivered to your door.

New to CircuitPython?

If you haven't used CircuitPython before, here's a quick primer. It's a version of Python 3 built for microcontrollers, though it also works on computers.

Perhaps the most useful feature is the huge array of libraries for maker hardware. You can download both CircuitPython and the Library Bundle from circuitpython.org.

CircuitPython programs are just text files interpreted on the device itself. However, there are some minor issues with how it reads files, so we recommend using the Mu editor (codewith.mu).

CircuitPython has excellent documentation, and you can find more information at rpimag.co/WhatisCircuitPython. 

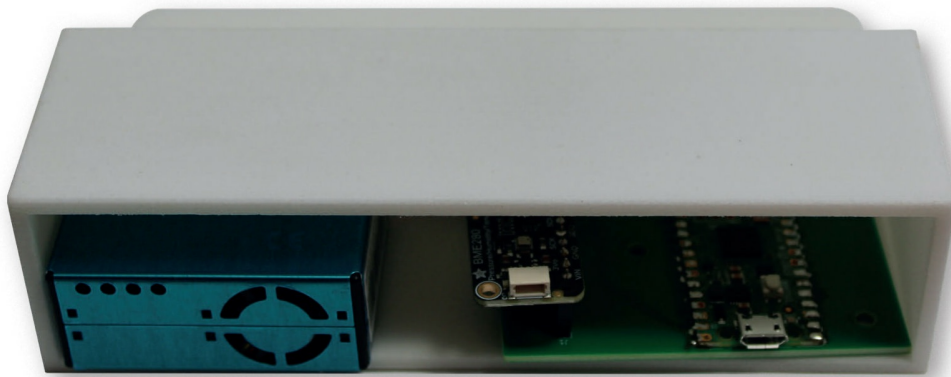
We've designed our parts and sent them to be made at a factory

Productionising

The code we've got here is enough to get started, but it lacks a couple of features that you may want for a more robust setup.

Firstly, storing passwords and keys directly in the main code increases the risk of accidentally committing sensitive information to version control systems like Git.

Secondly, the code lacks error handling for runtime issues such as Wi-Fi disconnections. While rebooting the microcontroller can often resolve issues, implementing automatic recovery logic – such as reconnecting to Wi-Fi – would improve robustness.



It's a bit of a squeeze, but everything fits into the enclosure

The Computers that Made the World



How twelve pioneering computers built
between 1939 and 1950 helped shape
modern technology

BUY ONLINE: rpimag.co/computersworldbook

Bake your own circuits

Assemble your own printed circuit boards without getting your fingers burnt



Maker

Rob Miles

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



@robmiles

YOU'LL NEED

- A printed circuit board and components to solder in place
- A tiny circuit board hotplate
- Some right-angled tweezers
- A syringe containing 138°C low-temperature flux solder paste
- A syringe holder (search for 'welding oil booster')

Powerful free tools such as KiCad have greatly simplified circuit board design and there are lots of companies who can turn your designs into physical boards. But how do you assemble your circuit? In this article we're going to explore the tools and techniques you can use to create your finished surface-mount devices.

Tiny keyboard

Our example project for surface-mount soldering, the MIDI Cheesebox, has twelve buttons and twelve coloured pixels. You can use it as a tiny musical keyboard and three-track sequencer. It contains a Raspberry Pi Pico running a CircuitPython program to emulate a MIDI instrument. You can find out more about it on GitHub here: rpimag.co/cheesebox. **Figure 1** shows the original Cheesebox (as featured in Raspberry Pi Official Magazine issue 113, rpimag.co/113) on the right-hand side. This contains twelve switches and a ring of NeoPixels. On the left you can

see the much slimmer version that uses a custom printed circuit board.

[Figure 2](#) shows the wiring inside the original Cheesebox. Each of the twelve switches is connected to an input pin on the processor. The author was keen to make it easier to create the hardware and fortunately he knew someone who could create a KiCad design for the circuit. Thanks for that, Brian.

It turned out to be very easy to send the circuit for manufacture and around a week later, six shiny boards arrived. It took a little while longer for the surface-mount LEDs and switches to turn up, but soon the author had all the parts on hand and no excuse for not building something.

Soldering on

We've been soldering things together for over 5000 years, which is around the same age as some of the plumbing in the author's house. Soldering was originally used in jewellery and tools, but it is now widely used in electronic devices.

Solder is a metal alloy that can be melted to bond metal elements together

Solder is a metal alloy that can be melted to bond metal elements together. The first electronic circuits were created by hand-soldering the components and wires together. In the first version of the Cheesebox, the author used a soldering iron to melt solder onto each of the connections. Solder used for electrical work contains a 'flux' element which cleans things up and removes oxides.

QUICK TIP

A pick-and-place machine is not beyond the means of the determined maker. They can be bought for around the price of a high-quality 3D printer and look like they would be great fun.



Figure 1: Switching to a printed circuit board design made it possible to add an OLED display to the device

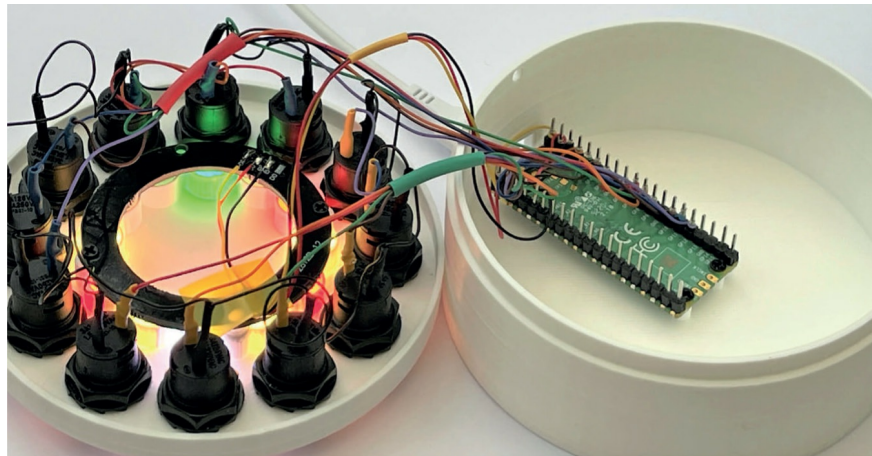


Figure 2: It took the author a couple of hours to make a Cheesebox from scratch

Soldering individual components together is painstaking work and limits the complexity of the circuits you can make. Halfway through the last century, someone had the bright idea of ‘printing’ copper circuits onto fibreglass. The first printed circuit boards (PCBs) had holes into which component leads were inserted, with solder used to hold them in place. Then another bright person suggested ‘surface-mount’ boards covered in solder pads onto which components were placed. These days, ‘pick-and-place’ machines are used to put the components onto the solder pads covered in solder paste and everything is soldered together by simply heating the whole component until the solder melts and forms the connections.

For the Cheesebox project, the author plays the role of the pick-and-place machine using a pair of right-angled tweezers. But before we can place any components, we need to get some

solder onto the board.

[Figure 3](#) shows the solder paste syringe used by the author. This worked very well. The first time the author did some surface-mount work, he didn't have the benefit of the 'welding oil booster', which made dispensing precise amounts of solder paste much more difficult.



Figure 3: The 'welding oil booster' is the yellow component. You tighten the bolt down the syringe as the solder is used up

[Figure 4](#) shows solder paste applied to the solder pads. It also shows some components in place. You need to be very methodical and make sure that you put some solder on every pad. Once you've got the solder in place, the next thing to do is place the components onto the board.



Figure 4: You don't need to be super-precise, but you mustn't bridge any connections

Masking up

If you want to make the process slightly easier, you can order a solder mask along with your boards. This is a metal stencil with holes for each connection. You lay it on top of your board and align it with the connection points. Then you spread solder paste over the stencil, depositing a blob of solder on each connection through the holes. Then you just have to place the components on the board. A mask makes things easier and quicker, but it is an extra expense. The author managed fine without one.

Surface-mount components are usually enclosed in tapes which are rolled into reels. A pick-and-place machine pulls the top off the tape and removes each component from it before placing

it on a board. For the Cheesebox, the author had to remove all the components from tapes before putting them in place. He used the right-angled tweezers shown in [Figure 5](#). These made it much easier to pick up and position the very small components on the board. The final part of the process involves heating the whole board so that the solder paste melts and forms the connections.

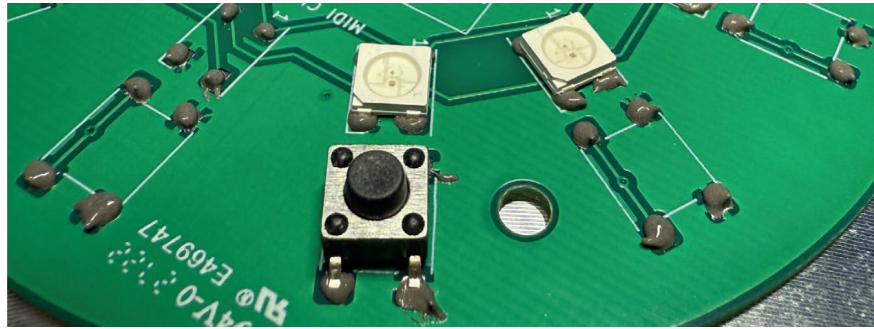


Figure 5: A pair of right-angled tweezers is essential for accurate placement of tiny components



Warning!

Hot solder

Soldering irons and hotplates get very hot, and stay hot for a long time after they're unplugged. Don't touch the hotplate or the circuit board on top of the hotplate while it's switched on, and remember to leave it to cool down after you've finished using it.

rpimag.co/soldering

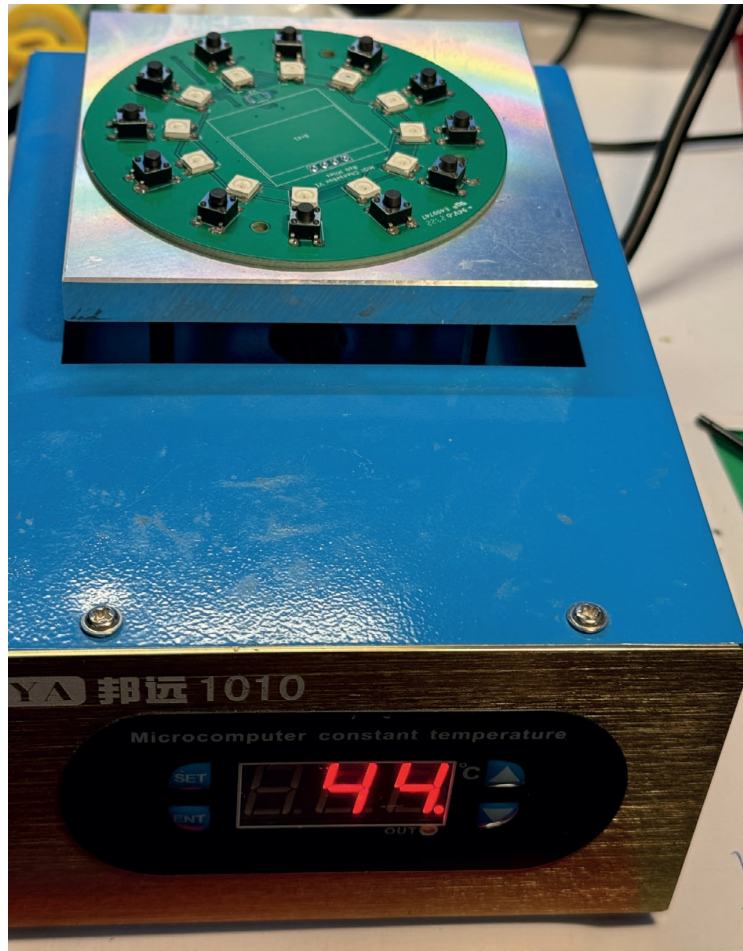


Figure 6: You set the desired temperature on the front

[Figure 6](#) shows a small hotplate specifically designed for soldering. It was bought from the author's favourite Chinese component supplier. The temperature of the top plate can be set on the front panel, which also displays the current plate temperature when the oven is in use. Setting the temperature to 160°C worked well. The oven was heated up from cold with the assembled board on top. When the temperature display showed 160, the oven was switched off and the board left to cool. It is important to avoid any sudden changes in temperature.

The hotplate gets as hot as the top of a stove. Be very careful not to touch the circuit board or the top of the plate when it is in use. And don't touch the top of the plate to see if it is still hot like the author did.

QUICK TIP

Buy the smallest syringe that you can and do all your soldering as quickly as possible. The paste tends to dry up once you open it, so it is unlikely you will be able to use it again after a few days.

When the board heats up, the grey paste turns to solder and the flux burns away, cleaning the

connections. It is fun to watch as the surface tension in the liquid solder pulls components into place. When the solder has cooled, you should find that all the components are held in place and connected together.

The Pico which controls the Cheesebox is soldered onto the back of the board

Figure 7 shows a completed board working well. The Pico which controls the Cheesebox is soldered onto the back of the board. This had to be done by hand once the board had cooled down. However, you can use surface-mount techniques with a Pico because it has ‘castellated’ edges.

Figure 8 shows the Pico soldered onto the back of the circuit board. When you solder a Pico to a board like this, use a piece of sticky tape to hold the Pico in place while you solder the first few connections on one end. Then remove the tape and solder the rest of the connections. The author was surprised by how easy it was to create working devices this way. He plans to make more circuit boards in the future. 🟢

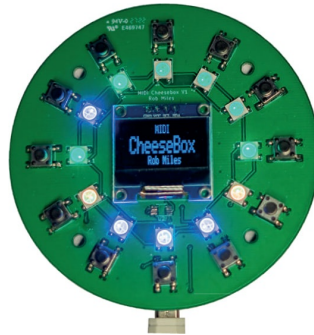


Figure 7: The boards all worked first time

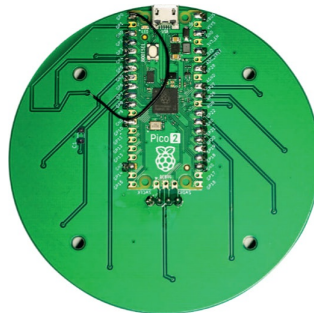


Figure 8: The extra jumper wire is there because the original design included a signal buffer which turned out not to be needed. The wire bypasses this

Meshtastic

Use Raspberry Pi Pico to enhance your network



Maker

Jo Hinchliffe (aka Concretedog)

With a house and shed full of lathes, milling machines, 3D printers, and more, Jo is a constant tinkerer and is passionate about making. Obsessed with rockets and robots and much more besides, he often releases designs and projects as open source.



concretedog.blogspot.com

We looked at the Meshtastic project a while ago, back in *HackSpace* issue 80 (rpimag.co/hs80), which is a great read that you should definitely check out if you are unfamiliar with Meshtastic. As a brief introduction, Meshtastic is an open-source project that uses affordable LoRa-equipped hardware to create off-grid communications networks ([Figure 1](#)). Beyond that, it's a capable and fun platform for experimenting with different hardware and antennas and other technical areas.

In the first part of this two-part look at Meshtastic topics, we will explore options to set up a budget Meshtastic device using an RP2040-based Raspberry Pi Pico W, and we'll also look at adding a cheap GPS module to a Meshtastic device. In the second part of the series, we are going to build a standalone solar-charged RP2040/Pico device which can act as a Meshtastic repeater, and we'll look at some interesting use cases for the device. Finally, we'll look at connecting Meshtastic to another open-source application to allow teams to share location and messaging whilst completely off grid.

As a primer for Meshtastic, there are numerous ways in which the devices can be operated. The essential idea is you have a Meshtastic device which is running the Meshtastic firmware and there are various ways that a Meshtastic device might be interacted with. One approach is to have a complete Meshtastic device, which may have a screen and a keyboard, and therefore one can send and receive messages, and view other statistics about the local Meshtastic network, all on one device. There are a few mass-manufactured devices for this, such as the Lilygo T-Deck Plus.

At the more budget-conscious end of the range of Meshtastic options, there's a heap of devices which can potentially be used in collaboration with your mobile phone, laptop, or other portable Wi-Fi/Bluetooth or USB device. A really common device for this is the Heltec WiFi

LoRa 32 V3, which has a small screen on board, but you interact with the device, sending and receiving messages, via the Meshtastic app on your phone and a Bluetooth connection ([Figure 2](#)). Sometimes, though, you might want to have a device that's just a node that you don't particularly want to connect to directly. The use case for this is as a Meshtastic repeater, or you could consider it a range extender. As a working example, I get much better results from a Meshtastic device placed in my garden than I do inside my (thick-walled) Welsh miner's cottage. However, I don't particularly want to stand in my garden all the time to interact with the node. A decent approach here is to place a standalone device in the garden in a good position and then use another device in the house, perhaps on a windowsill, with which I have a Bluetooth connection to my phone.

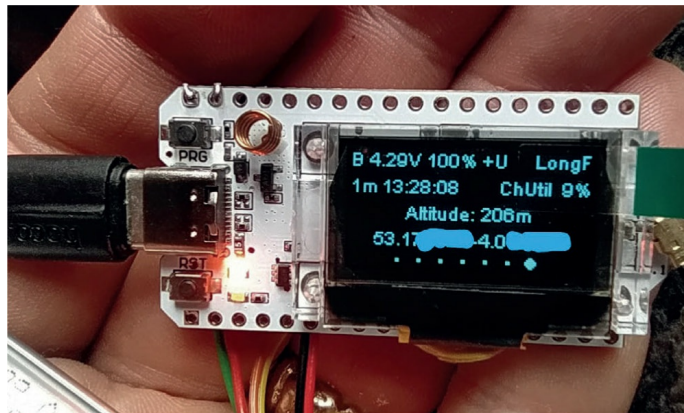


Figure 2: A Heltec V3 board is a common choice for a budget Meshtastic device

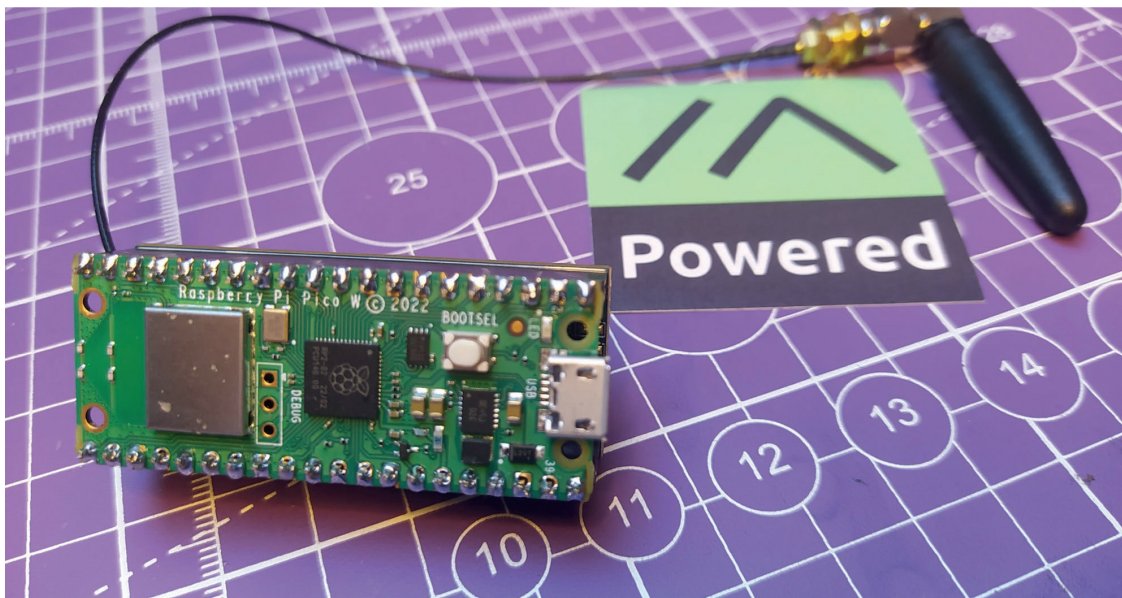


Figure 1: A Meshtastic device made from a Pico W and an add-on module

Outdoor connection

The device in the garden in this example doesn't need to have a screen, or even really a way to directly connect to it via Bluetooth. For these devices, a Pico or other RP2040-powered device

can be a great option. With no screen attached and no Bluetooth or wireless active, they can be reasonably efficient in terms of power consumption and they are certainly at the affordable end of the available Meshtastic device options. Let's take a look at an option.

Flashing a RP2040/Pico device with the Meshtastic firmware is really straightforward

As we already have some Pico W boards, a good option is to add a WaveShare SX1262 LoRa Node Module (**Figure 3**). This LoRa module is designed specifically for Pico – and if your Pico has header pins soldered, you can plug it straight into the WaveShare SX1262 Module. The module has a handy silkscreen graphic indicating where to line up the USB port on the Pico. You'll notice that the WaveShare device has a connector for a single-cell battery on board and, excellently, it will charge the battery via the Pico USB port; indeed, the WaveShare SX1262 we bought came with an included LiPo cell. The SX1262 also has on board a small IPEX 1 connector to connect the coaxial cable for the included antenna. Press this connector carefully into place and then screw on the supplied small antenna. These antennas are great to get going with, but you can certainly increase the range of the device with an antenna upgrade down the line.



Figure 3: SX1262 LoRa add-on board for Raspberry Pi Pico family

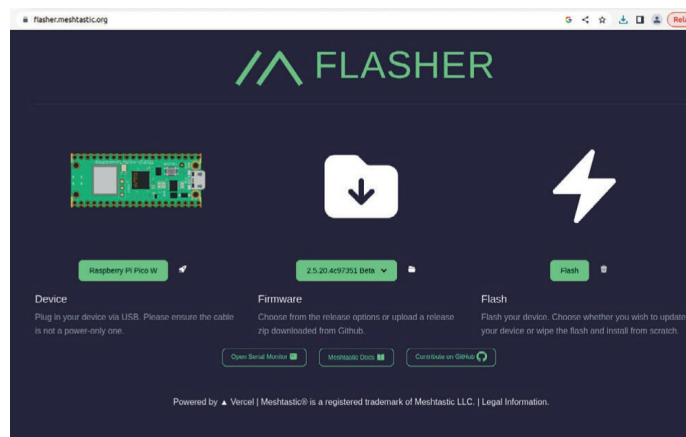


Figure 4: Using the Meshtastic Web Flasher tool makes it really straightforward to flash devices with the correct

firmware

Flashing an RP2040/Pico device with the Meshtastic firmware is really straightforward using the Meshtastic Web Flasher tool ([Figure 4](#)). Navigating over to the page, you can then select Raspberry Pi Pico W from the ‘Devices’ menu. Next, click the Firmware button and select the firmware you want to install. We’ve tended to opt for the latest stable version rather than the beta development versions. Finally, and unlike flashing other Meshtastic devices, when you click the Flash button, you will get the option to download the firmware UF2 file. If you have played with Raspberry Pi Pico before, you’ll recognise this as the drag-and-drop format of firmware that can easily be placed on the Pico for installation. To do this, plug in your Pico to your computer while holding down Pico’s BOOTSEL button. You should see a DFU mode drive appear as a connected drive on your computer. Simply drag and drop, or copy and paste, the UF2 file from your downloads into the connected Pico drive, wait a few seconds, and the drive should disconnect and reboot, running the Meshtastic firmware. The Pico Meshtastic currently has some issues in that, unlike other devices, you can’t connect to the Pico via Bluetooth and you can’t connect to it over Wi-Fi using the web UI due to a lack of HTTP support. However, don’t despair: we eventually want to use this device as a standalone repeater, and there is a simple way to connect to the device for configuration.

Leaving the Pico W device connected to your computer, navigate over to the Meshtastic web client and click the ‘+ New Connection’ button ([Figure 5](#)). In the next window, select the Serial tab from the three available tabs, then click New Device. Select the serial connection that refers to your device in the resulting window and you should now see the device connect and the Meshtastic web client spring into life.

Radio radio

At this point, your Pico node is up and running, but can’t connect to any other nodes since the radio region – and therefore the operating frequency of the radio module – isn’t set. To do this, click Config and then Radio Config; you’ll see in the Mesh Settings window that the first item is Region. Currently the region is set to ‘Unset’; click this drop-down menu and set the region to your locale, so for us that was EU. Of course, you can also edit any other items, the name of your node, channels setup, and more in the web client ([Figure 6](#)).

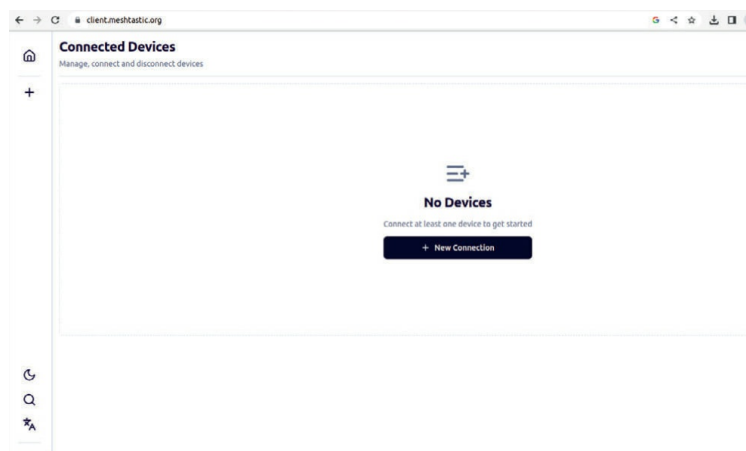


Figure 5: The Meshtastic web client allows you to connect to a Meshtastic device using HTTPS, Bluetooth, or in this

case a USB serial connection

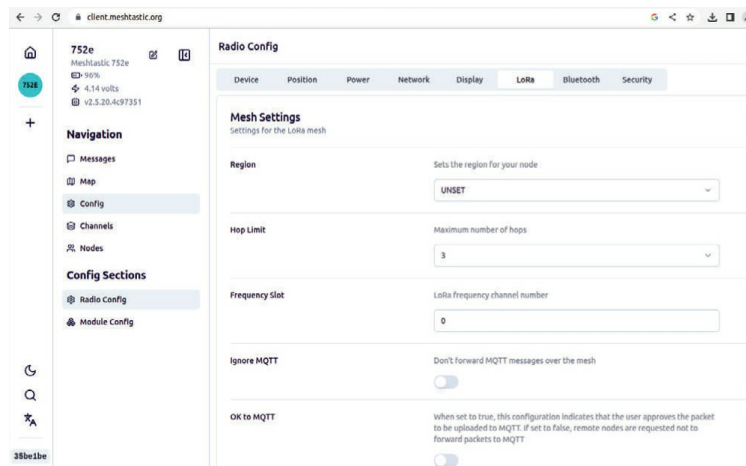


Figure 6: Until the LoRa region is configured, the device won't be able to connect to any Meshtastic devices or network

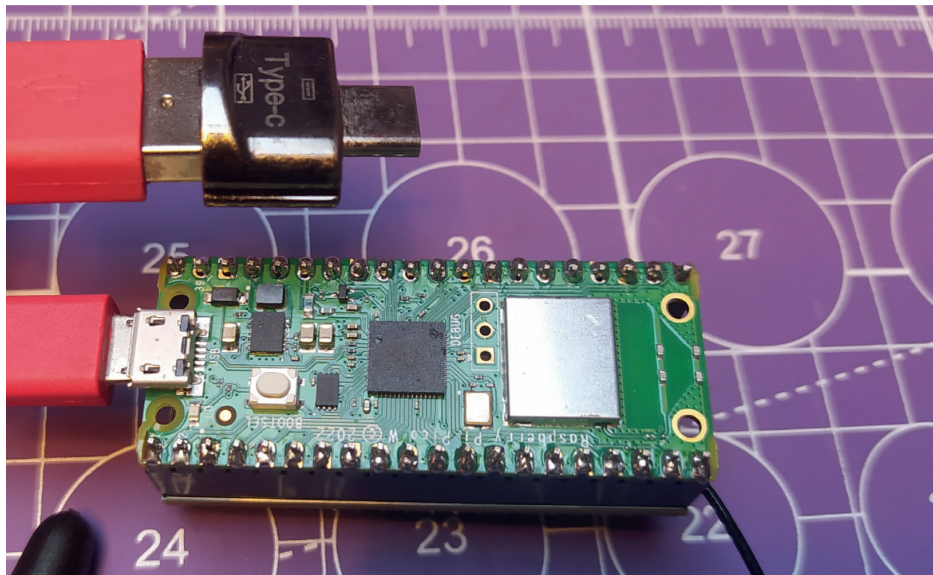


Figure 7: Using a USB-C to USB-A adapter to connect the Pico Meshtastic device to the Android application via USB OTG

Another option for interacting with our Pico W Meshtastic device is using a wired USB OTG connection to our phone and using the Meshtastic app. We needed a USB-C to USB-A adapter to be able to connect to the Pico (**Figure 7**); the device was automatically detected and we found we could select '/dev/bus/usb/001/002' listed in the Meshtastic app connection menu. This is a great option if you want to interact with the device in the field. We'll leave this Pico/Waveshare device at this stage, but in the next issue we're going to look at setting this device up as a standalone repeater with battery and solar charge system.

Another great DIY Meshtastic project is to add GPS to a device so that you can show your device position to other Meshtastic devices or for tracking an item your device is attached to. Of course, again, you could simply buy a Meshtastic device with GPS capability, but where is the fun in that! For this section, we are going to connect a GPS module to a Heltec V3 Meshtastic

device as we don't particularly plan to carry the Pico-based device as a mobile module.

The Heltec Wi-Fi LoRa 32 V3 is a classic entry-level module for Meshtastic. It's affordable, has an on-board screen, and connects well to the Meshtastic app via Bluetooth. Similarly to the Waveshare SX1262 add-on board, the Heltec V3 has a battery connector and you can recharge a connected battery via the USB-C connector. We are going to add a really cheap and cheerful GPS module, a NEO-6m, which is an older GPS module, commonly available, and has been used in many, many hardware projects over the years!

Wire it up

Adding a GPS module to a Meshtastic is as simple as soldering four wires – two for power and two for RX and TX connections – between the Meshtastic device and the GPS module. Beyond that, there is a tiny bit of configuration to do in the Meshtastic app and you should be up and running. It's probably worth noting that we added our GPS module to a Heltec V3 that we had previously set up using the Meshtastic Web Flasher and we had confirmed it as an operational Meshtastic device.

We'll need four wires to solder between the Heltec V3 and the NEO-6m ([Figure 8](#)). On the NEO-6m, connect the VCC pin to the 3V3 pin on the Heltec V3; likewise, the GND pin on the NEO-6m needs to connect to the GND pin. Finally, solder the RX pin on the NEO-6m to pin 48 and then the TX pin to pin 47. After checking your connections, you can then power up the Heltec V3 device and connect to it using the Meshtastic app via Bluetooth. You should see an LED light on the NEO-6m confirming you have power to the board.

QUICK TIP

Always connect an antenna to your Meshtastic device before powering up, as powering a LoRa radio without an antenna attached can damage it.

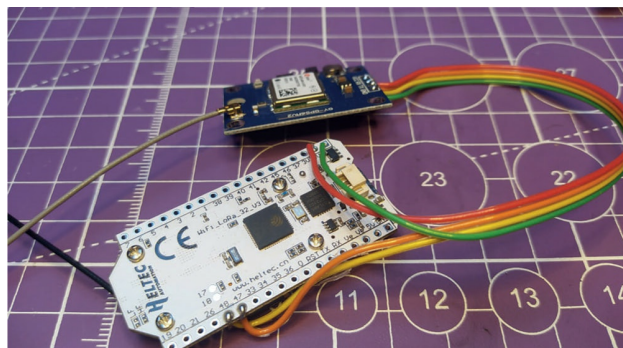


Figure 8: Soldering four wires is all it takes to add GPS capabilities to a Meshtastic device

In the Meshtastic app, click the three-dots menu icon and select the Radio Configuration menu, then the Position submenu. Scroll down this menu and then, under the heading 'GPS mode', use the drop-down box to change the mode from 'NOT_PRESENT' to 'ENABLED'. Then, slightly further down the page, you will find a section where you can redefine pin numbers

([Figure 9](#)). Look for the heading ‘Redefine GPS_RX_PIN’ and then, in the dialog, change the value 0 to 48. Next, do similar for the TX pin by changing the value of ‘Redefine GPS_TX_PIN’ from 0 to 47. Finally, and crucially, scroll down to the very bottom of this page and hit the Send button to push these changes to your device. The device should reboot (disconnecting and reconnecting to the app) and when you are reconnected, after a wait of maybe a few minutes, the device will begin to broadcast its GPS location – well, kind of, as there’s one more thing to consider.

If the primary channel of your Meshtastic device is the default public ‘Longfast’ channel then the Meshtastic firmware, by default, will display an inaccurate GPS location. For example, in [Figure 2](#), the location of this device is around 1 km from my actual position ([Figure 10](#)). The reason for this is that by default Meshtastic doesn’t want to broadcast your exact position, for privacy reasons. To enable a precise position to be broadcast, you need to go into your Channel settings via the Radio Configuration menu, click on the Longfast channel, and turn on the ‘Precise location’ button. If you then check your position in the map tab, you will see, after it updates, that it should be a good match for the device’s position. This means that as you perhaps add private channels on your Meshtastic devices, you can choose channel by channel whether you share your rough location, precise location, or not share location information at all. 🟢

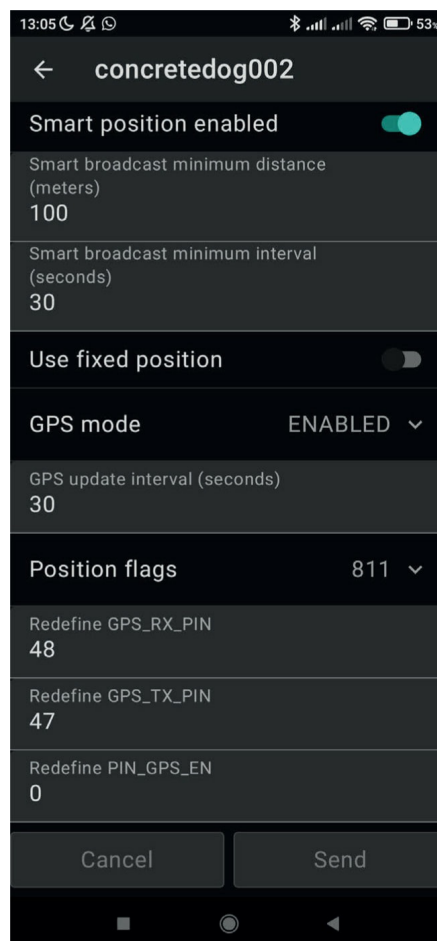


Figure 9: Configuring the RX and TX pins and enabling the GPS device in the Meshtastic app

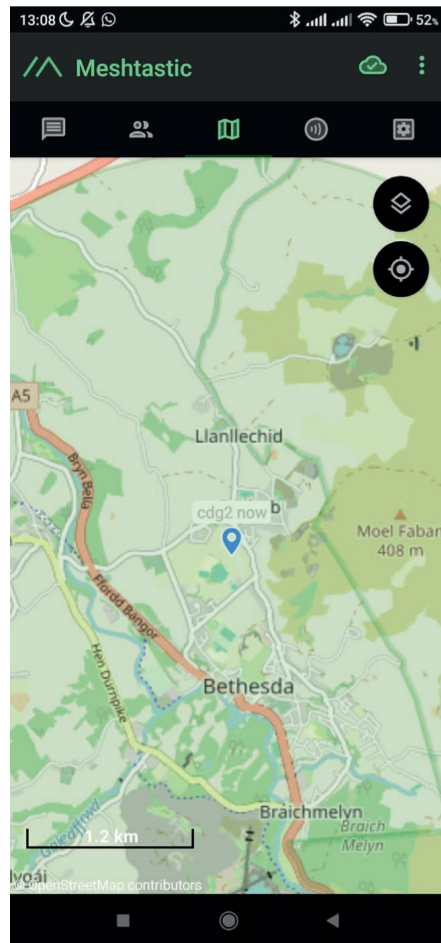
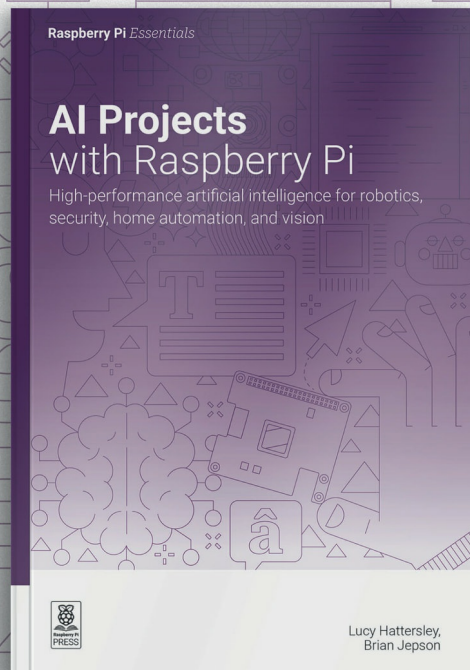


Figure 10: The default GPS settings on the public 'Longfast' channel are deliberately not accurate until a precise location mode is enabled

QUICK TIP

Unless you can place a device in an excellent 'top of a mountain on a mast' position, it's good practice to use client modes in Meshtastic rather than router modes.



With this essential guide, you'll learn how to create all kinds of AI-powered projects on Raspberry Pi! Combining Raspberry Pi hardware with artificial intelligence gives you new ways to interact with the world: turn your surroundings into a playing field with computer vision models, predict and adjust to changes in environmental conditions, and engage with people in natural ways.

■ **You'll learn how to:**

- *Identify and categorise objects in pictures or video*
- *Recognise and generate speech*
- *Translate from one language to another*
- *Use generative AI models to create images or text*
- *Train models to make predictions from sensor readings, such as temperature and acceleration*

BUY ONLINE: rpimag.co/aiprojectsbook

Meshtastic

Build a standalone solar Raspberry Pi Pico node



Maker

Jo Hinchliffe

With a house and shed full of lathes, milling machines, 3D printers, and more, Jo is a constant tinkerer and is passionate about making. Obsessed with rockets and robots and much more besides, he often releases designs and project as open source.



concretedog.blogspot.com

QUICK TIP

If you want to create a very optimised solar Meshtastic node, other MCUs such as the RAK family of Meshtastic boards are very efficient.

In the first part of this guide, we set up a Raspberry Pi Pico W with a Waveshare SX1262 LoRa Node Module. We're going to add some other simple parts to this device and place it in a simple ABS enclosure so it can be deployed outside. This then can be considered a repeater node that we wouldn't directly connect to with the app or anything, but rather, can be placed so it extends range, or solves range problems like line of sight, increasing our mesh network ([Figure 1](#)).

There are lots of examples online of people building Meshtastic nodes running on batteries and or solar power. Meshtastic on Raspberry Pi RP2040 isn't the most efficient setup, but for this project we aren't aiming for a totally optimised system that can last indefinitely, but rather a device we can deploy quickly that will run for at least a day or two.

As we also aren't too worried about size, having found a pretty large ABS enclosure in a drawer, we can opt for a larger format battery. For a tiny handheld unit we might look at a small single cell LiPo, or perhaps an AA form factor lithium ion battery such as a 14500 cell. We've opted for an 18650 cell. 18650s are very common and come in a range of capacities. Our Ultra Fire cell is listed as 2700mAh. Again, as we have space, we opted to place this cell in a 18650 cell holder rather than spot-welding tabs to the cell permanently ([Figure 2](#)). This also means that

with a spare cell or two, we can increase the uptime of the Meshtastic node by simply replacing the batteries.

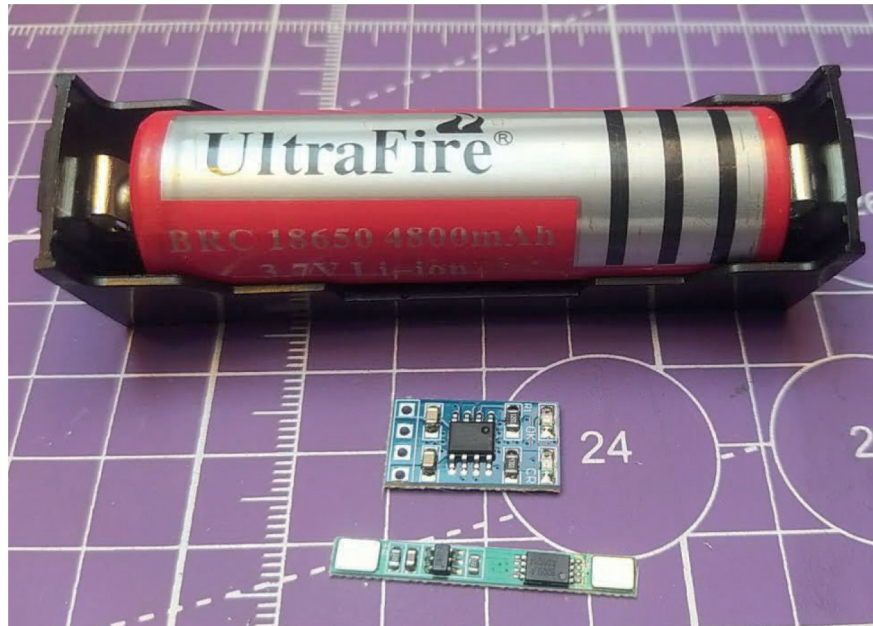


Figure 2: Our 18650 cell, battery holder, cell protection board, and solar controller that form the basis of our power system



Figure 1: Our complete standalone node deployed on a hilltop

In a mountainous region such as here in North Wales, it's easy to walk into valleys or 'cwms' where there is no line of site to any other Meshtastic

Battery power

It's good practice to use a cell protection system when using lithium-ion cells and they are available very affordably. The cell protection system will stop the 18650 from being drained too low, with an overdischarge function kicking in and cutting the power output if the cell reaches 2.4V. Likewise, the cell protection board will protect the cell in an overcharging situation and can detect short circuit scenarios.

We used some very sticky M3 double-sided pads to affix the small cell protection board to the side of our 18650 cell holder. The cell protection boards have some large pads labelled B- and B+ and we simply soldered small wires from the device back to the corresponding pins on the battery holder ([Figure 3](#)).



Figure 3: Using a small piece of double-sided sticky foam, we then soldered the cell protection circuit to the battery holder

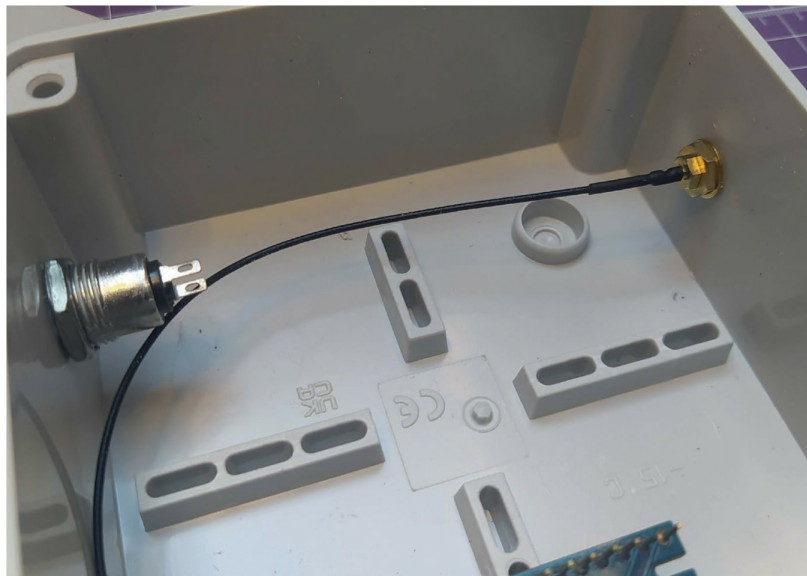


Figure 4: We drilled the enclosure walls to receive a 12mm barrel key switch and the antenna SMA connector

Next, we soldered two pairs of wires to the P+P- terminal on the cell protection board.

Obviously it's a good idea to not have the cell fitted at this point in case any of your stripped wire ends accidentally touch, although the cell protection board should stop any damage to the cell. These two pairs of wires will see one set used as the power supply to our Pico H/ Waveshare boards via a power switch and the other pair as an input from a solar power charge controller.

For a power switch we used a small key switch. This key switch has an off and on position and, importantly, the key can be removed in either position ([Figure 4](#)). We simply soldered the switch (after fitting the switch to the enclosure) into the positive wire between P+ and the pin 39 input on the Pico H/ Waveshare board. This means we can turn the whole node on and off, but also means that if we deploy the node somewhere in the field, it's unlikely that anyone else could turn the device off and, once the key is removed, it's impossible for the device to be accidentally turned off through movement or a knock. As the Pico H/Waveshare stack has header pins fitted, we opted to simply solder some jumper socket connectors to the power supply wires and could then slot them onto the positive pin 39 and the ground pin 38.

The remaining pair of wires connected to the P+ and P- pads on the cell protection board are wired into our SD05CRMA solar charge controller, which is again an affordable and readily available small board. The solar charge controller only has four pins: a positive and negative input to receive power from a 4–6V capable solar cell, and a positive and negative output for charging an attached single cell. We again soldered two header sockets to the positive and negative inputs and then permanently soldered the charge output to the wires we soldered to the P+ and P- pads on the cell protection system. This means that we can easily wire a solar panel to the device to charge the 18650 cell when required and that the cell can be charged regardless of whether the Meshtastic node is powered on or off.

With our wiring largely complete, we began to assemble our device into the ABS enclosure ([Figure 5](#)). The enclosure is perhaps a little large, but that makes it easier to work inside and also would allow us to expand the system down the line, perhaps opting for multiple 18650 cells to increase the available runtime. We drilled a 7mm hole for the antenna SMA connector and a larger 12mm hole for the key switch. An additional 7mm hole was also drilled on the underside of the system which would allow us to connect the solar panel ([Figure 6](#)). When it's not in use, we can cover this hole with a piece of tape, or perhaps make a custom grommet to stop any moisture ingress. The Pico and Waveshare module were mounted to the edge of the box with some double-sided foam, as is the solar charge controller. The battery case was fitted with hot glue as it needs to be robustly connected to resist the forces of inserting and removing the cell.

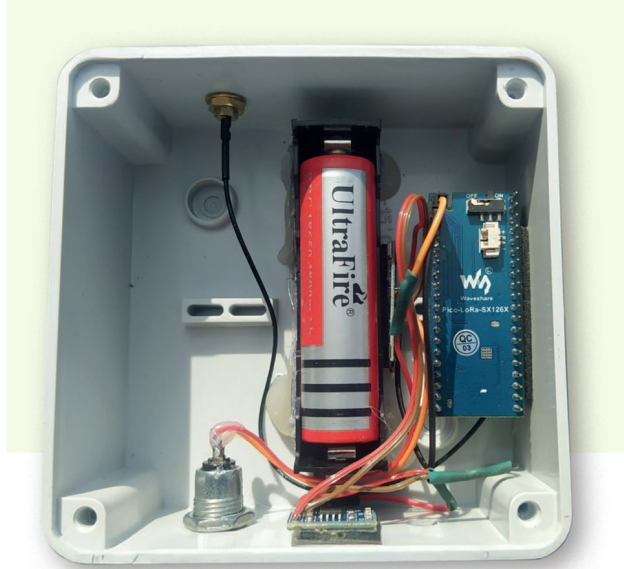


Figure 5: The complete wired-together node assembled in the enclosure

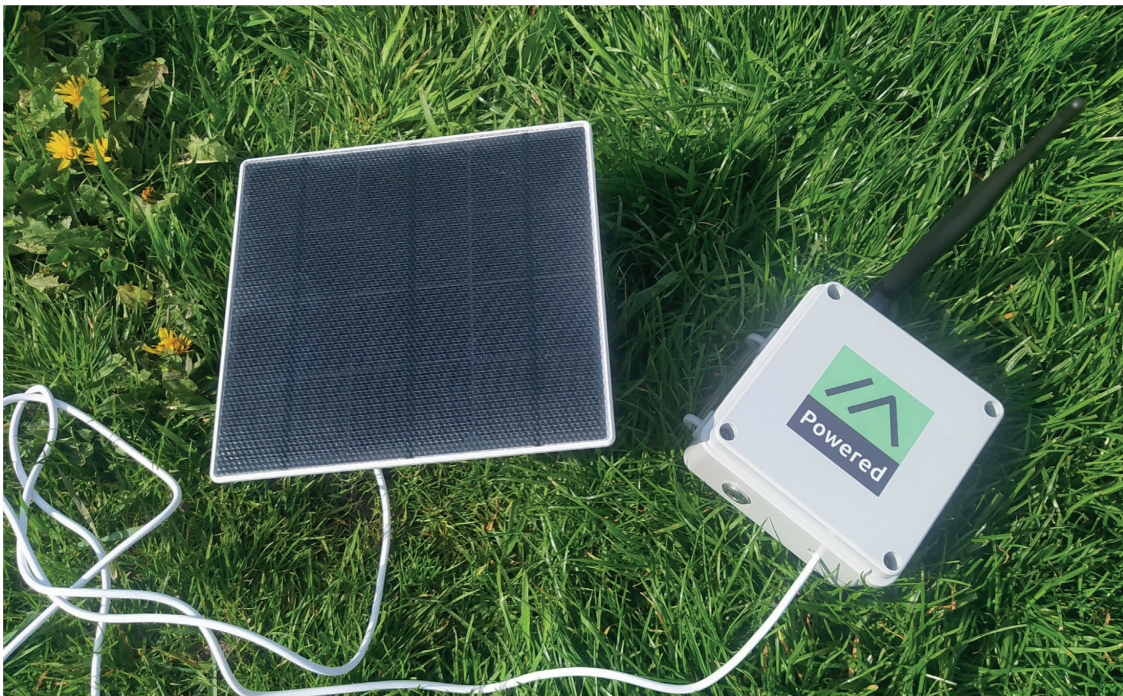


Figure 6: Any 4-6V solar cell or combination of cells can be connected into the node, extending the runtime. Here a 5W solar panel originally for a CCTV camera has been used

We wanted to be able to quickly deploy the node in the field and so some 3D-printed PETG attachment points were printed and glued to the back of the ABS enclosure. Through this we then added some 6mm shock cord ([Figure 7](#)). The shock cord arrangement allows the node to be quickly mounted on any vertical polelike structure – a fence post, for example. In our case, using a trekking pole creates a simple installation for when camping or for impromptu Meshtastic comms sessions taking a break whilst hiking.



Figure 7: Some PETG 3D-printed eye mounts were superglued onto the outside of the enclosure. Through these we added some stretchy shock cord, allowing the node to quickly be attached to poles and other objects

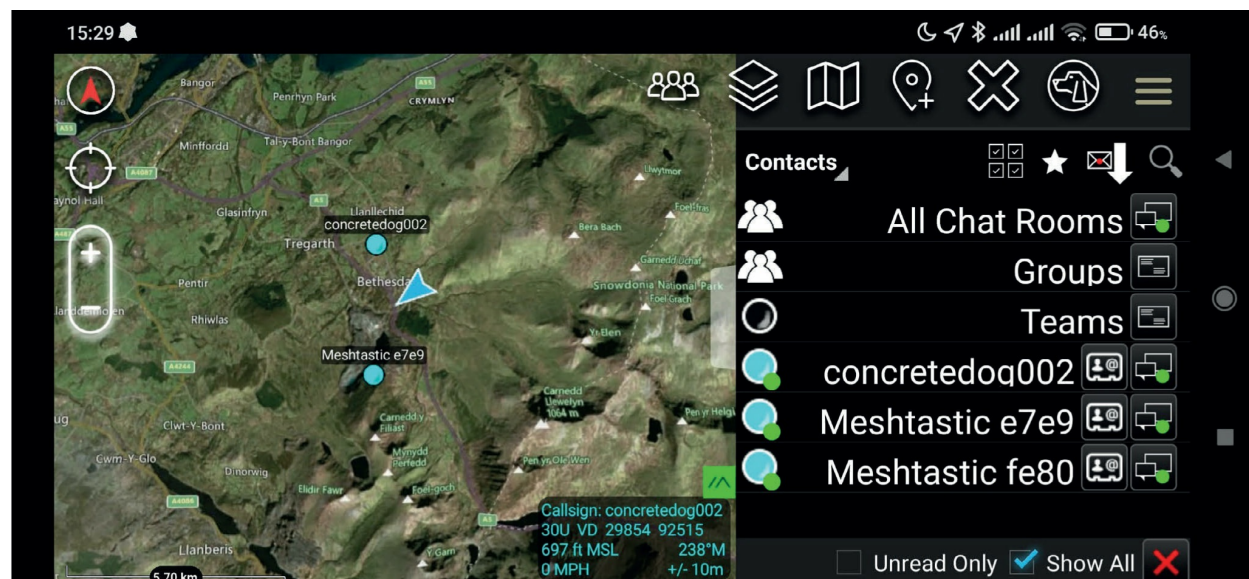


Figure 8: The main page of the ATA-CIV Android app, showing numerous Meshtastic devices laid over a downloaded map

Deep in the valley

Mentioning hiking brings us to a couple of example use cases we want to explore with Meshtastic. In a mountainous region such as here in North Wales, it's easy to walk into valleys or 'cwms' where there is no line of site to any other Meshtastic nodes. With a little forward planning and route consideration, it's often really easy to stealthily place a self-powered node such as this one in a position where it can act as a repeater with line of sight into a valley with limited connectivity, but can also see other nodes in the network.

It can also be used as a short distance repeater. For example, in an exposed position such as in [Figure 1](#), often the weather and wind might preclude you from sitting in the prime summit position for your node. Placing a node like this, then retreating off the summit/hilltop to escape the weather, you can then have an additional smaller node, with a much smaller antenna, to act as

your Meshtastic communication device, knowing that the deployed node is within easy reach.

You can see that a lot of our use cases for Meshtastic are for communications on fun adventures in the hills. This led us to explore another interesting application of Meshtastic, namely plugging Meshtastic into another application call **ATAK-CIV** (Figure 8). The latter is a civilian arm of a tactical military system for managing communications and teams positionally. Whilst it's aimed at military operations, it's a great fit for anything where a team are working in a co-ordinated fashion over a wide terrain. We could see it definitely being of use to mountain rescue teams, but equally, it could be great fun for youth groups, girl guiders or scouts playing wide roaming games in their locality. ATAK-CIV has a huge amount of features and we will only cover the basics, but let's look at getting it installed and setting up Meshtastic as its method of communication.

ATAK-CIV is a civilian arm of a tactical military system for managing communications and teams positionally

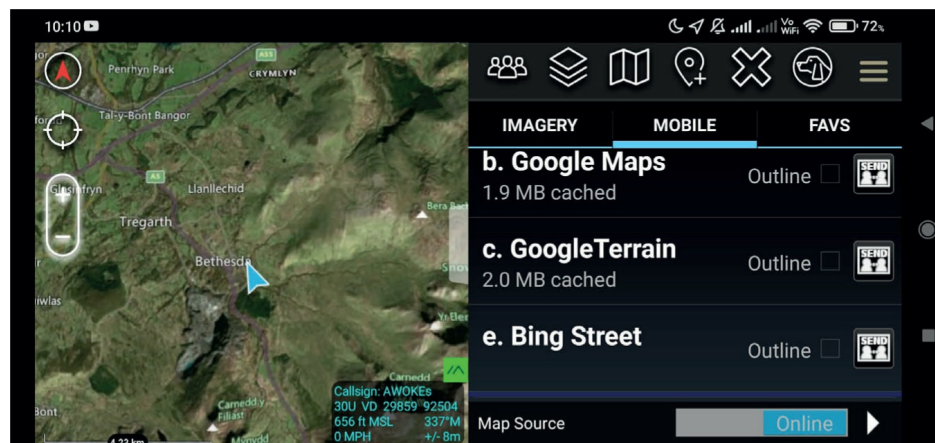


Figure 9: A list of the downloaded, installed, and available maps in our ATAK setup

ATAK-CIV

ATAK-CIV is available for Android only and you can download and install it for free from the Google Play Store. You can also find official Android APK files for ATAK-CIV online; so, if you have a de-googled droid device, this is still possible. Next you need to search for the Meshtastic ATAK plug-in APK. This plug-in acts as an intermediary between the Meshtastic app (we installed this when setting up our device in the first part of the guide); once downloaded, click the APK file to install.

Before launching the ATAK-CIV app and connecting it all together, we'll need to download some maps to install to the app. The default map is just a placeholder and when you zoom in at all, it will remain very blurry. You can easily switch between map types in the ATAK-CIV app, so we downloaded two map selections: the Bing map resource pack and the Google Maps resource pack. To do this, navigate to the links on your Android phone, download the files, note the location of your Downloads folder on your Android device (we have one on both our internal

storage as well as our SD card, so that caused a moment of confusion).

Now we can launch the ATAK-CIV app. It will ask for lots of permissions and will also ask for the location services to be switched on. In the ATAK-CIV app, click the main menu three-bar icon and then scroll down and click the Import button. Navigate to the map content zip files you downloaded and select them. You'll be given the option to move the file to an ATAK directory – copy the file to a directory or use the file from the current location. We opted to move the files into ATAK. Back in the main screen, you should now be able to click the folded map icon and see a list of your added map options and switch between them ([Figure 9](#)).

Finally, we need to enable and configure the Meshtastic plug-in. First, fire up a Meshtastic device that you can connect to with the Meshtastic application and check it is connected. Then, in ATAK-CIV, click the menu button and scroll down and click 'Plugins'; you should see the Meshtastic plug-in listed. Click the tick box to load the Meshtastic plug-in and you should see the listing turn green to show the plug-in is loaded. Finally, return to the ATAK menu, scroll to Settings, then Tool Preferences > Specific Tool Preferences > Meshtastic Preferences. There's a lot of settings to explore and tinker with in here, but just to get started, make sure the first three items in the list are enabled and all the others disabled ([Figure 10](#)).

With all this done, you should now see the ATAK map populate with Meshtastic devices in the area. You will also see an arrow representing you. You can now use the ATAK-CIV app to send messages to other Meshtastic devices, including privately to other Meshtastic nodes with which you have preshared channels. You can also do lots of other fun things. Exploring ATAK, you can do things such as geofencing (so it will alert you if a Meshtastic device in your team moves beyond a specific boundary), allow you to automate sharing map points with all your team members, as well as tracking other publicly viewable Meshtastic devices. We even see that there are ways to connect FPV drones to ATAK and send drones to specific locations! 📍

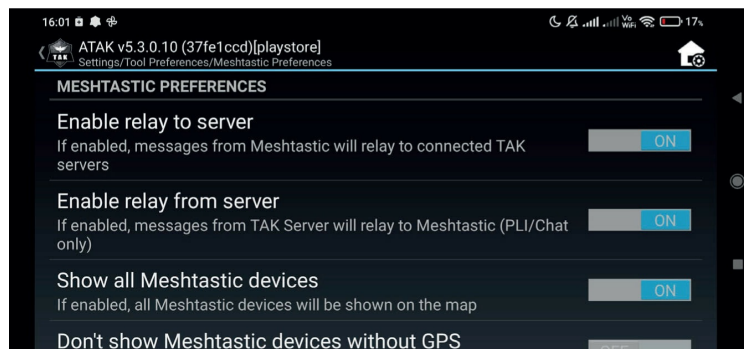


Figure 10: Configuring the Meshtastic ATAK plug-in

Building a capable underwater rover on a budget

Let's look at a common approach to building the chassis of an underwater rover with plastic pipes and 3D printing



Maker

Jo Hinchliffe

With a house and shed full of lathes, milling machines, 3D printers and more, Jo is a constant tinkerer and is passionate about making.

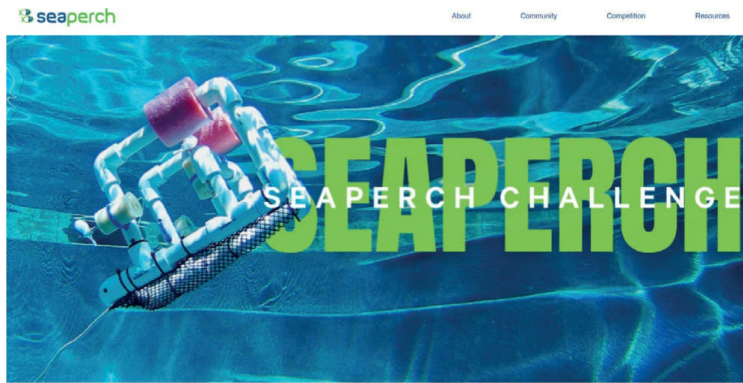


concretedog.blogspot.com

Back in Raspberry Pi Official Magazine issue 157 (rpimag.co/157), we presented the Tiny Open-source Underwater Vehicle (TOUV), a palm-sized underwater explorer built with little more than some small DC motors, 3D-printed frame, and some small float tanks. It's a super-affordable and accessible project, perfect for playing with in small pools or tanks, and a great family fun activity or small STEM project.

However, its tiny stature meant that it had some shortcomings: there's little capacity for any sort of payload on board, and it's only really operable short range where you can maintain line of sight with it in its environment. We're gonna need a bigger boat...

Researching reveals lots of interesting DIY underwater vehicles across a range of complexities. We wanted to build a project large enough to carry cameras and other payloads but also retain a relatively simple drive system built onto an affordable chassis that could be built by most people.



The common approach we went with was to design our chassis as a tubular frame to be made from PVC pipe and pipe fittings. This is a really common approach and there are a lot of examples. A good starting point for research is the Seaperch Challenge, [Figure 1](#). The Seaperch challenge is a STEM program promoting underwater exploration, with international school teams taking part in regional competitions leading potentially to an international final event. If you look around the community, you will find a lot of information and build instructions for all manner of underwater explorers.



Of course, this has led to people increasing the capability of these PVC pipe chassis craft, and there are lots of examples of larger more complex systems based on Seaperch principles. One such source of inspiration for our build was a video series on an enhanced Seaperch build by ‘Agermangineer’, who has an excellent video series on YouTube (rpimag.co/agermangineer).

PVC pipe and fittings come in all manner of shapes and sizes. The commonest pipe family used for Seaperch-style chassis designs is ½" schedule 40. In metric terms, this pipe is 21.3mm outside diameter with a 15.8mm internal diameter. Most DIY stores in the UK and elsewhere

will stock this style of pipe and associated fittings. You can also find it available online and you can even find a range of colours. It's easy to work with: you can cut the pipe with specialist pipe-cutting shear tools that are reasonably affordable, but equally it is not difficult to cut with a small junior hacksaw.

The nice thing about PVC pipes and fittings is you can slot pieces together without gluing them and you can disassemble and reassemble many times, trying out different configurations. Before cutting your pipe into lots of sections, though, you might consider creating a 3D model assembly to try out ideas. We used FreeCAD to create a small toolkit of parts that could then be used in the Assembly workbench to play with different chassis layouts.



Warning!

Water and electricity

This project is low-voltage (12V), but water and electricity don't mix. Make sure to test your waterproofing.

rpimag.co/waterproofing

We wanted to build a project large enough to carry cameras and other payloads

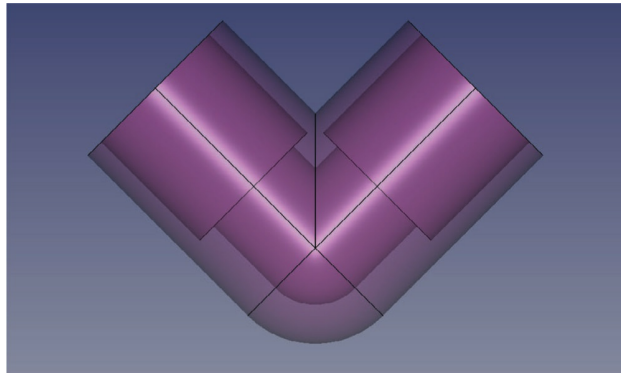


Figure 2: Modelling the 90-degree elbow component in FreeCAD. Once modelled, you can use the part as many times as you like in a 3D assembly

The main aim of the chassis is to provide mount points for three thruster systems and also have room for extra payloads such as cameras and lighting. As a 3D assembly project, it's quite achievable as we essentially only modelled three different parts. We modelled a straight section of tube, a 90° elbow section ([Figure 2](#)), and a T piece. As ½ schedule 40 pipe fittings are common, it is easy to search online and find dimensional drawings to help make the parts accurately. One of the standard Seaperch chassis designs calls for straight pipe sections that are 1 ½", 2 ½", 3", and 4". It's simple to draw a profile sketch of a pipe section using two circles and

then extrude the part to a required length before saving that part.

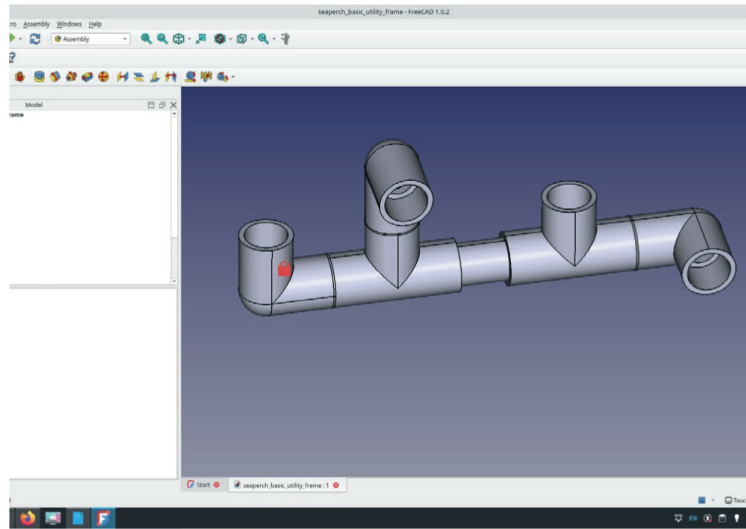


Figure 3: With our three basic parts models, we can import many versions of the parts into the FreeCAD Assembly workbench and begin to prototype

In the newer FreeCAD versions (1.0 and beyond), there is a new built-in Assembly workbench. Whilst a full tutorial on this workbench is beyond the scope of this article, you can insert multiple copies of parts into an Assembly container, then select faces or edges of parts and constrain them together (**Figure 3**). For chassis assembly, we tended to just mate faces together using the ‘Create a fixed joint’ tool, and you can quickly iterate numerous chassis designs for consideration (**Figure 4**). Once happy with a design, you can count up the parts in your Assembly model tree and cut your pipes to length and perhaps order the correct number of elbows and T sections you require.

Turning our attention to thrusters, we decided to go with a common approach where people modify 12V bilge pumps, designed to pump water out of boats, to create powerful water-resistant drive systems. The pumps we used are designed to move 1100 gallons of water per hour drawing around 3 amps in operation. They cost around £10 per pump and therefore are one of the more costly components. The bilge pumps in standard form have a blue plastic cover that can be removed, and then have a white plastic shroud which covers an impeller system that pushes water out of a connector. In their official usage you would place the pump in an area that may become flooded, with a pipe attached to the connector which you would run out of the vessel. To convert to a motor thruster, we need to remove the white plastic shroud and then replace the impeller with a propeller. To remove the shroud, we again used a junior hacksaw to carefully cut through the white plastic a couple of millimetres above the joint with the red plastic section. This sounds harder than it is, but once in hand, it’s pretty easy to feel where you should cut. Once the shroud section was removed, we used a flat-head screwdriver to remove the impeller (**Figure 5**).

One of the advantages of the tubular chassis design is that it can be assembled and disassembled multiple times

We decided it could be a fun project, and a test of our FreeCAD skills, to design and 3D-print the propellers for our thrusters. The propellers are designed with a 4mm hole through the hub, into which we inserted an M4 bolt. We then found we could purchase brass couplers where one end had a 3.17mm hole matching the bilge pump output shaft, and the other end a 4mm hole ([Figure 6](#)). The first props we made had a helical pitch angle of 30 degrees and, when performing a rudimentary test in the bathtub, proved to be far too aggressive! We then adjusted the pitch down to 15 degrees and this seemed to generate a lot of thrust (simply estimated by how much the system pushed against our hand) without too much violent white water thrashing!

Prior to testing in water, we did some bench-testing of the thrusters to see what kind of amperage they draw at 12V. Like most brushed DC motors, there is a larger current inrush as the motor starts, which then settles. Our bench-top variable DC power supply tops out at 5 amps and indeed, the motor did momentarily top out our current before settling to around 3 amps in unloaded running. Obviously, compared to the previous TOUV project which we controlled with a Raspberry Pi Pico coupled to some DRV8833 motor drivers, we are going to need much more capable motor drivers. Of course, it's possible to directly switch the motors to a 12V supply, but it's a much finer experience if you have as a minimum some way of setting the speed of each thruster, and at best variable control of the thrusters.

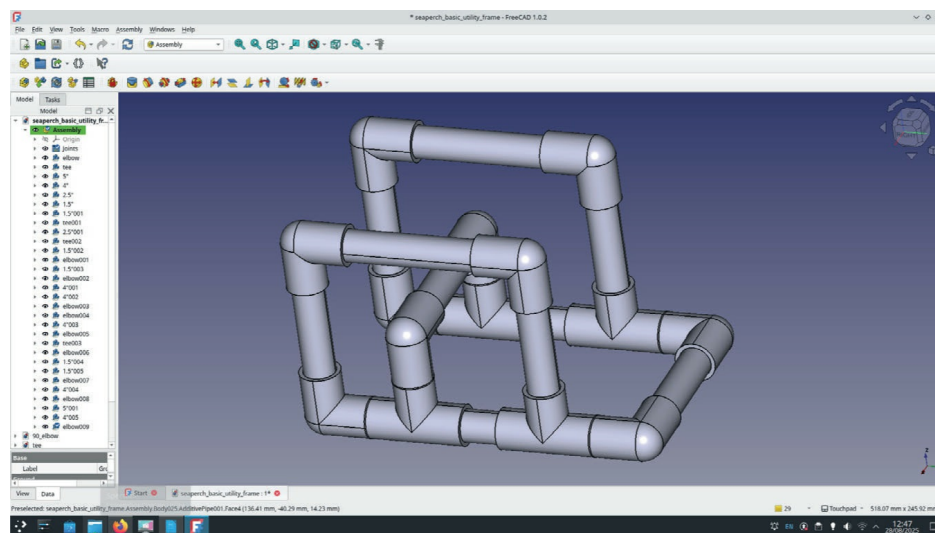


Figure 4: An example of one of our prototype chassis that is close to one of the standard Seaperch designs



Figure 5: Adapting a 1100 gallon per hour 12V bilge pump to become a thruster required cutting some case sections off the device

Looking around, we came across the affordable BTS7960 motor control boards which can drive attached motors via an input PWM signal (**Figure 7**). The BTS7960 supports 12V for the motor driving side and the onboard dual H-bridge ICs can switch up to 43 amps. This offers a lot of headroom and should drive our thrusters with a large safety margin. As a bonus, the BTS7960 modules can output a 5V supply and have protection circuits built in, so we can hook up a Raspberry Pi Pico without an additional power supply.

One of the advantages of the tubular chassis design is that it can be assembled and disassembled multiple times and the friction fit of the plastic components is tight enough that it's not necessary to glue the pieces together. In fact, looking around other builds, some people don't ever glue at all. We can imagine iterating on the design and potentially testing the design in water before maybe using PVC cement to finalise some parts of the chassis. This allows for some different design considerations when it comes to mounting components. As an example, we used FreeCAD to model thruster mount designs, one for the side thrusters and one for the vertical thruster (**Figure 8**). As we can disassemble the chassis, we made a slotted clamp feature allowing them to be slid over an unattached tube section before being bolted into place. If we chose to cement the chassis before fitting these, we would have to design some form of two-piece clamp to allow the mounts to be fitted. Finally, designing with clamps that fit around the standard tube sections allows for flexibility and trimming; we can raise and lower thruster mount points, for example.



Figure 6: A complete thruster with a DIY 3D-printed propeller mounted with an off-the-shelf brass coupler

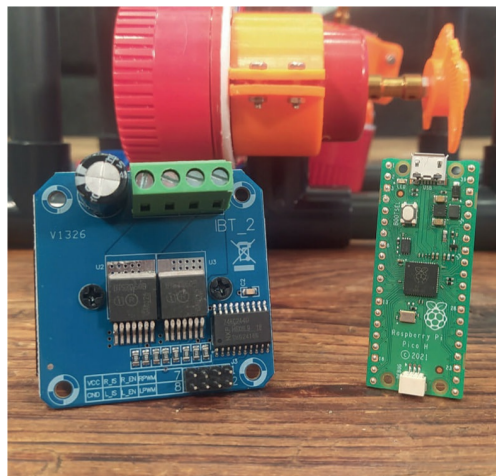


Figure 7: Our plan for motor control primarily consists of pairing a Raspberry Pi Pico with the large and capable BTS7960 motor drivers

As well as designing for trimming and flexibility, when designing somewhat experimental vehicles it's useful to design parts for modularity. For our motor mounts, we envisage that it would be useful to add some form of shroud section that protects the propeller from contact or weeds becoming tangled in it. As this is a part that might need a few iterations, we simply added some holes to the motor mount design to which we can attach designs. This allows for changes in propeller design and any real-world problems encountered in testing and hopefully means we don't have to constantly reprint the basic mount section.

One of the considerations for underwater vehicles, of any scale, is buoyancy. Many examples of Seaperch-style builds add small amounts of foam, often sourced from pool noodles, which are cable-tied to the upper sections of the tubular chassis. This has some merit in that it's pretty simple to do, you can reposition the foam, and also you can add or reduce the amount of foam. One issue reported with this is that if you start to go a little deeper in your underwater explorations, the foam can become compressed due to the increasing pressure and then it becomes less effective.

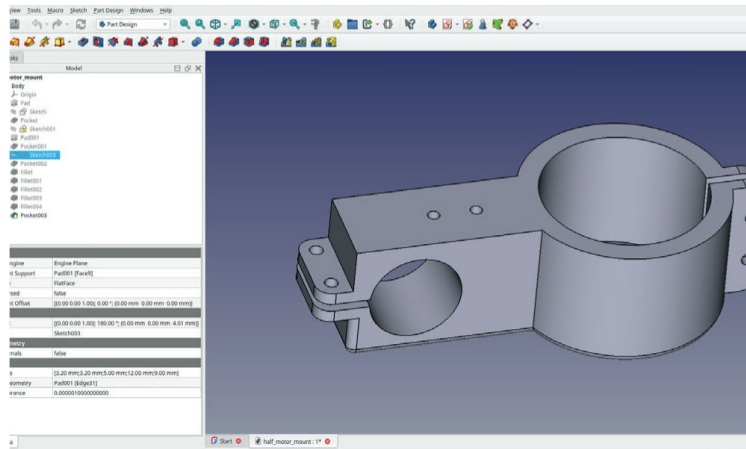


Figure 8: Designing clamp-on thruster mounts in FreeCAD



Figure 9: Adding the longer buoyancy tubes; we again used simply designed 3D-printed clamping mounts

When designing somewhat experimental vehicles it's useful to design parts for modularity

With this in mind, another option we have seen is to add larger plastic pipes, with sealed bulkheads, to create less compressible air chambers. In the TOUV build, this concept was used with the tiny 35mm film canisters and a further advantage was discovered that you could 'trim' the balance and buoyancy of the vessel by slightly flooding the canisters as needed. With this in mind, we opted to add two large tube sections sealed with threaded end caps ([Figure 9](#)). The tubes are 36.5mm diameter and we estimated we might need around 50cm length. Adapter sections are cemented in place as well as the threaded closure receiving sections and then a commercially available end cap with a large rubber washer can be fitted to seal the tubes.

Again, this has some advantages in that we can perhaps use weights and pieces of foam inside the tube to add ballast to specific areas to help balance the vehicle, particularly when loaded with payloads such as cameras, etc. Similar to the motor mount clamps, it's reasonably trivial to create 3D-printed clamping mounts for these tubes and these allow for repositioning of the tubes and clamps relative to the vehicle chassis.

With the mechanical design largely in place, we can now start to look at assembling and testing the thruster drive system, a land-side controller, and some kind of wire tether system. Plans are also in place to create a rugged underwater camera from a Raspberry Pi Zero and we have a cunning idea for how to make it incredibly waterproof. 🟢



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
pinning for a future that could have been.
It's utterly astonishing!"

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



Buy online: rpimag.co/ctmb



Building a capable underwater rover

Last tutorial, we built a pipework chassis for a larger underwater rover. In this section, we look at creating drive control systems and a Raspberry Pi-powered underwater camera



Maker

Jo Hinchliffe

With a house and shed full of lathes, milling machines, 3D printers and more, Jo is a constant tinkerer and is passionate about making.



concretedog.blogspot.com

In the first part of this two-part series, we looked mainly at the chassis design of a larger and more capable underwater rover. We used FreeCAD to prototype a pipework assembly before moving to cutting and test-fitting a chassis and some additional floatation tubes. In this part, we'll look at some subsystems in development for the craft. We'll look at creating a simple drive system with a Raspberry Pi Pico, creating a prototype waterproof Raspberry Pi Zero camera system, and also look at a simple accessory to add a live visual feed from the craft.

Previously, we looked at the common approach of converting 1100 gallon per hour 12V bilge pumps into thrusters for the underwater vehicle. We briefly mentioned that we planned to use a BTS7960 motor driver board in conjunction with some buttons to create a simple drive controller. This essentially is similar in operation to the controller we built for the Tiny Open-source Underwater Vehicle (TOUV) in Raspberry Pi Official Magazine issue 157 (rpimag.co/157). However, of course, the uprated motor drivers are needed to run at the higher current draw and higher voltage.

The motor driver boards we bought claim to have a regulated 5V output so that, on paper, we could run a Raspberry Pi Pico handling the PWM signals to the boards without an extra power supply. However, on receipt of the modules, none of them actually appeared to have this regulated output, so for our prototyping we've provided power to Pico via a USB power bank.

For in-water testing, which will probably now wait till slightly warmer weather next year, we aim to emulate the type of simple control we built for the TOUV project. Namely, three buttons controlling the thrusters but with the Pico using tuned PWM values to set each thruster

speed. As a starting point, this is a very usable system, but also allows for more complex control systems to be developed which might include variable speed with potentiometers or joystick input further down the line.

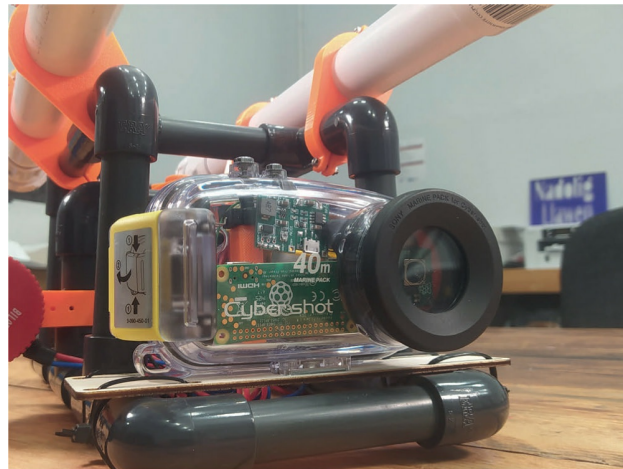


Warning!

Water and electricity

This project is low-voltage (12V), but water and electricity don't mix. Make sure to test your waterproofing.

rpimag.co/waterproofing



A DIY Raspberry Pi dive camera mounted to a front deck on our underwater rover



Figure 2: An affordable 10m waterproof endoscope can make a simple live visual feed back to a phone or tablet

As an example of motor control using the BTS7960 IBT-2 board, we can wire up a Pico to the module as well as adding a momentary press-to-make switch, a pull-down resistor and, of course, our motor/bilge pump thruster. You can see the pin value connections in [Figure 1](#).

With our wiring in place, we created a simple MicroPython script using Thonny. The script is pretty straightforward and we can vary the speed by changing the speed variable value. We've found that it only really makes sense to run the motors in one direction on these three-thruster

underwater craft, but you should see that it's perfectly possible to create bidirectional motor control should you wish to. This BTS7960 script is available in a GitHub repo: rpimag.co/bts7960.

One of the great things to do with this larger and hopefully more capable underwater platform is to add some form of live visual feed. We'd also like some method of capturing footage or potentially still images from on board. An easy and common way to approach a live visual feed from underwater is to run a video-carrying cable up through the tether. Indeed, this could be very possible looking at affordable wired CCTV or FPV cameras. However, another option that caught our eye was using a budget drain inspection camera, often referred to as a USB endoscope (**Figure 2**).

These are commonly available for not huge amounts online and they can be connected to your mobile phone with an app installed to allow you to view the video stream. The one we ordered was rated as waterproof and had a 10m cable, so it's certainly enough for us to begin to explore underwater, and adequate for some test missions with the platform when the summer months return.

The video feed from the endoscope can be captured, but we wanted to build something more experimental that could be used in lots of different ways using a Raspberry Pi. As a first experimental payload, we built an 'action camera' equivalent using a Raspberry Pi Zero and Camera Module 3. Obviously Raspberry Pi boards, such as the Zero 2 W we used for this section of the project, aren't waterproof. Please don't set one up and dunk it in water as you won't get much further use from it! So we started to think about waterproof enclosures. We'd also dug out a small action camera as part of the exploration and that prompted an interesting idea we hope others enjoy playing with.

One of the great things to do is add some form of live visual feed

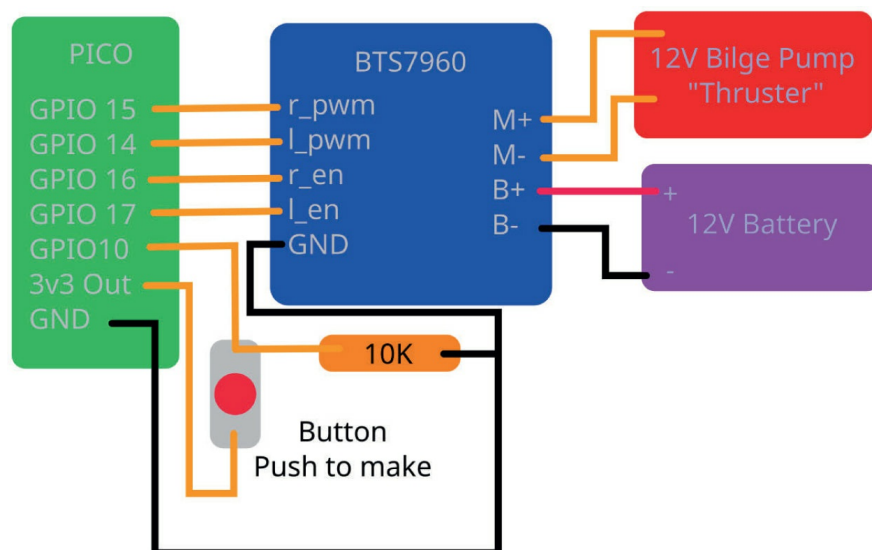


Figure 1: the wiring for a single channel thruster control using a Raspberry Pi Pico and a BTS7960 motor driver board



Figure 3: There are lots of older dive cases for early digital cameras available on second-hand marketplaces

Back in the early noughties, lots of handheld digital cameras started to hit the shelves and became much more common for people to buy and own. At the more capable and expensive end of the market, many manufacturers produced add-on dive enclosures to allow individual camera modules to be used underwater. Of course, these underwater dive enclosures were very well made and pretty expensive at the time, but often had the button layout that matched to a specific model of camera. Of course, this now means that they are very out of date and as such are very affordable on secondhand marketplaces ([Figure 3](#)).

We bought a couple that were branded for different models of Sony camera. Both of the camera dive cases were under £9 and one arrived with a complete spare set of seals and some silicon seal lubricant! The dive enclosures contain lots of waterproof button and dial connectors mounted through the case, which would have aligned with the camera model inside. With some 3D printing and some retrofitting, it's totally possible to mount momentary touch buttons or latching buttons so that they can be actuated by these original buttons ([Figure 4](#)).



Figure 4: There are heaps of waterproof button pushers. We can use 3D printing to place switches and buttons within reach of these case features

As these dive cases are for older handheld cameras, there is lots of space inside and so it's pretty possible to add a Raspberry Pi Zero, a Camera Module, and a power system. Further down

the line, we imagine this could become its own project and there's no reason we couldn't pack in camera lighting and more button controls as well as potentially adding other sensors and experiments. Even aside from the underwater rover, this could become a great project in its own right.

As a first pass at creating an underwater action camera, we designed and 3D-printed a mount for the Camera Module 3; this means that we can swap between the standard module or the NoIR version which may give better results in some situations. It's a challenge to measure the internal section of the lens area of the dive enclosure as it is slightly flexible, in an awkward spot, and is a tapered tube design. We made a couple of attempts and, as it is a relatively small part to 3D-print, we created a best guess version first, test-fitted after printing, and adjusted the CAD a couple of times for it to end up as a reasonably snug fit ([Figure 5](#)).

With a mount for the Camera Module 3 in place, we then created a simple post which in turn held a latching momentary push switch underneath one of the original buttons in the waterproof enclosure. Keeping things simple, we just used this inline with the positive connector of a LiPo cell and we created a system so that when the power button was pressed, Raspberry Pi Zero would boot and then run a small script to record a video file ([Figure 6](#)).

QUICK TIP

Before you mount electronics inside your dive enclosure, it's a good idea to place a piece of tissue paper inside and test the enclosure underwater, checking for any leaks.

Before any final assembly into the dive enclosure, we set up our Raspberry Pi Zero and Camera Module 3 to do some testing and to create the automated scripts. We actually set up our Zero 2 W with the Raspberry Pi HDMI monitor and a keyboard and mouse to do this and used the graphical version of Raspberry Pi OS with the desktop graphical environment. Of course, you could set the following up by SSHing into Raspberry Pi Zero from your desktop or laptop computer and, in turn, you could install lighter versions of the OS without the graphical desktop for a more lightweight responsive design. For ease of prototyping, though, a keyboard, mouse, and monitor can be easier, especially when visually checking the camera.



Figure 5: 3D-printed camera mount

We connected the Camera Module 3 to Raspberry Pi Zero 2 W with the supplied ribbon cable (**Figure 7**). After doing the basic Raspberry Pi setup and booting to the desktop, we opened the terminal application and used some simple `rpikam` commands to check the camera was functional. A good start is to try:

```
$ rpikam-hello
```

This will launch a small preview window with a livestream preview from the camera. Similarly, you can check that the camera can take a still image with:

```
$ rpikam-still -o test_image.jpg
```

This will create an image file called **test_image.jpg** in your home folder.

Finally, you can test and perhaps experiment with changing settings/frame rate/dimensions to create short video clips with a command like:

```
$ rpikam-vid -t 5s -o test_video.h264
```



Figure 6: Using tiny spots of hot glue, we can create a sturdy yet reversible mount post to place a latching button under a button presser

...where the value following `-t` is the duration of the clip in seconds. There's a whole lot more you can do with the Camera Module 3: long exposure photography, intervalometer projects, and more. A great primer for it can be found here: rpimag.co/camera-software.

Next, we created a couple of scripts that formed the system so that when the Raspberry Pi Zero was powered up, it would boot and then record a small video clip. We wanted to be able to increment the file name of the saved video clip so that we could repeat the operation and create numerous test files without overwriting the previous recording. To do this, we created a script called **record_boot_video.sh** in our home directory. Note that both the following scripts we create are available here: rpimag.co/autorecordvideo.

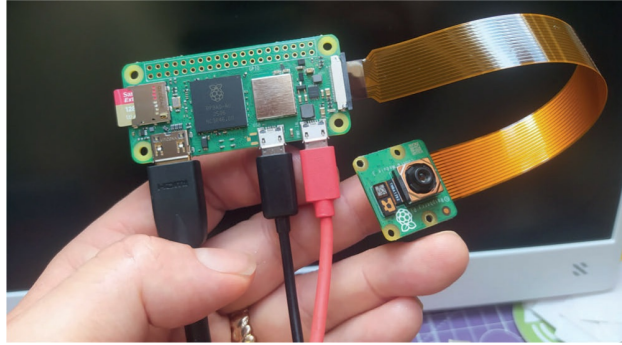


Figure 7: Working with Raspberry Pi Zero 2 W and Camera Module 3 to create the system before mounting into the dive case

To do this, we opened the terminal application and issued the command:

```
$ sudo nano record_boot_video.sh
```

This launches the nano text editor, into which we pasted:

```
#!/bin/bash

sleep 10 # wait 10 seconds after boot, I found 10 seconds the minimum delay that worked. Might be
safer at 15 seconds.

COUNTFILE="/home/cdog/bootcount.txt"

if [ ! -f "$COUNTFILE" ]; then
echo 1 > "$COUNTFILE"
fi

COUNT=$(cat "$COUNTFILE")

rpicam-vid -o "/home/cdog/video_${COUNT}.h264" -t 5000 #this command captures a 5 second video adapt
for your needs

NEXT=$((COUNT +1))
echo $NEXT > "$COUNTFILE"
```

Note that you need to change the username sections of the above script to match your username of your system; so wherever our script says ‘cdog’, you insert your username.

You can see at the top of this script that when it runs, it is going to check for (and in the first run, create) a text file called **bootcount.txt**. Each time this script is run, it will add 1 to the running count value inside this script. It then goes on, in the middle of the script, to record a video file, using the `rpicamvid` command we tried earlier, but this time it will name the file ‘video_’ and then the number detected in **bootcount.txt** will be suffixed to the file name. This means that we can repeatedly try the system and instead of always overwriting the same single video file, we will end up with multiple uniquely named files: **video_1**, **video_2**, **video_3**, etc.

Next, we need to add the following script to the directory **/etc/systemd/system**:

```
#sudo nano /etc/systemd/system/bootrecord.service #add below script

[Unit]
Description=Record video on boot
After=graphical.target

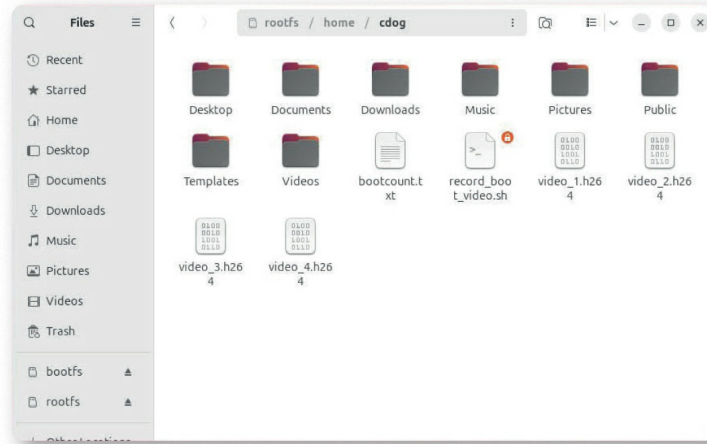
[Service]
```

```
Type=idle
User=cdog
ExecStart = /home/cdog/record_boot_video.sh

[Install]
```

Once you have added the **bootrecord.service** script to systemd, you need to enable the service by running the command:

```
$ sudo systemctl enable bootrecord.service
```



Having booted the test camera system, we can see that it has captured and created numerous sequentially numbered video files

You can then, as a first time test, run the service manually. Normally the service will just run on boot, but for this first run, issue the following command:

```
$ sudo systemctl start bootrecord.service
```

This will make the system run immediately, but the process will fail! This is because the first time the **record_boot_video.sh** script runs, it will create the **bootcount.txt** file, but we will need to change the permissions of that file to be able to write to it. To do this, we can make the **bootcount.txt** writable by any user by using the **chmod** command:

```
$ sudo chmod 664 /home/cdog/bootcount.txt
```

Again, you need to change 'cdog' to your own username in the above command.

You should now find that whenever you reboot the system, it will boot and then automatically record a new video clip.

As the Sony camera dive case is designed as a real camera system, it's pretty adaptable in terms of ways that it can mount. On the underside of the case, there is a standard ¼-inch 20 turns per inch threaded camera mount, so we can use all manner of standard camera mounting systems for it. In the first instance, we've lasercut a small 'for deck' section which is cable-tied to the front of our chassis and we can use a small bolt from our collection of camera mount equipment to mount the DIY underwater camera system.

We're really pleased with the whole build and can't wait to get out testing and exploring with it when the weather is a little warmer. That said, it's a great project to tinker with on your desk in winter, and we are inspired to play more with the idea of retrofitting old camera dive enclosures with fun experiments. 🟢

USA SPECIAL!

**6 ISSUES
FOR \$43**



Subscribe online:

rpimag.co/subscribe

Continuous credit card orders will auto-renew at the same price unless cancelled.

A choice of free Pico 2 W or Pico 2 is included with all subscriptions. This is a limited offer. Not included with renewals. Offer subject to change or withdrawal at any time.

Build a super-simple sensor robot

Affordable robots can still do cool stuff – let's build one from scratch



Maker

Jo Hinchliffe (AKA Concretedog)

With a house and shed full of lathes, milling machines, 3D printers and more, Jo is a constant tinkerer and is passionate about making. Obsessed with rockets and robots and much more besides, he often releases designs and projects as open source.

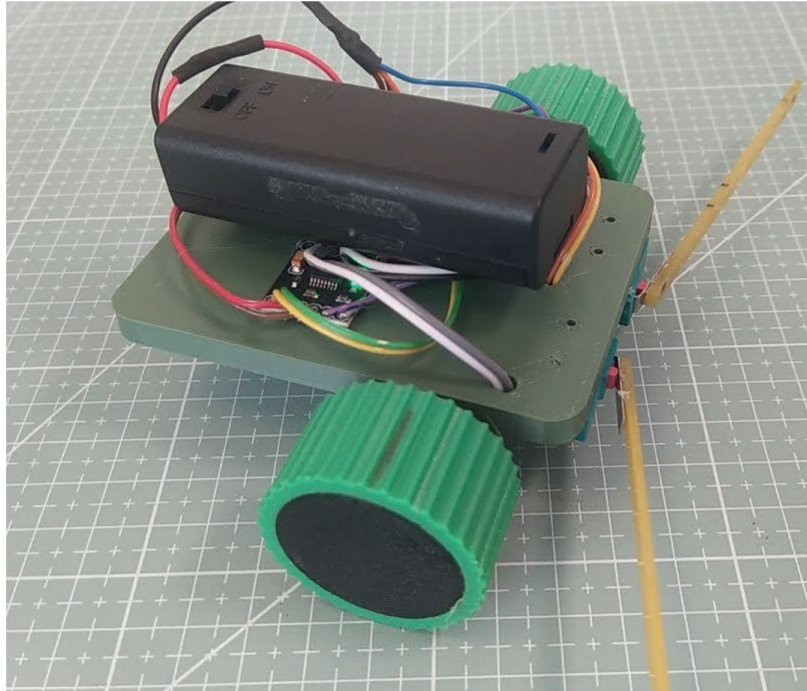


concretedog.blogspot.com

Robots are cool in all shapes and forms, but it's even cooler if you get into building your own. We've featured lots of high-spec robots in Raspberry Pi Official Magazine over the years, but sometimes the old 'keep it simple' adage is worth revisiting. So we wanted to explore building a simple, minimal component robot with the minimum of 'sensors' to give it some kind of autonomy.

We have some prior experience designing robots. A few years ago, we created the Modular Tracked Vehicle (MTV) Robot ([Figure 1](#)). This used an off-the-shelf tracked chassis system and two pretty powerful brushed geared motors to create a really capable robot platform. In more recent times we published the *Design an RP2040 Board with KiCad* book (rpimag.co/kicad2040) which, in one chapter, led to the creation of the open-source StoRPer robot ([Figure 2](#)).

The StoRPer robot is great fun. It was designed with four motor driver systems on board so that you could create an all-wheel-drive system; this is great for Mecanum wheels and Mecanum drive systems. We later added a breakout board (or 'bumper'), which added the ability to easily add ultrasonic sensors and bump switches. Of course, all of this adds complexity, but it serves as some inspiration for a cheaper, simpler build. One area we can simplify from StoRPer is we can build a robot with half the number of motors. Of course, the MTV robot only had two motors, but the tracked system is too complex to make ourselves and requires a higher-rated power supply. The N20 motors have options available that can be powered with the same supply as the Pico, so they are a simpler choice. We can go for a simple configuration of two motors to drive two wheels, with a third passive wheel to help the robot balance. This will make for a fun robot, at reduced cost, complexity, and number of components.



Our simple two-wheeled robot features two motors and a basic 3D-printed chassis

Look out!

Having played with bump switches with the StoRPer robot, we know that we can make a simple system for collision detection and avoidance which will create a crude form of autonomy in our robot. So with this in mind, we'll make a start.

First of all, we need to think about a chassis. A chassis in its simplest form is simply a base for all the robot components to be attached to. Whilst we have used FreeCAD to create a simple design and then used a 3D printer, it's fair to say that the chassis could be any piece of flat material into which we could drill holes or use hot glue to attach parts. Common materials are plastic boards, or a piece of wood, or even a couple of layers of thick cardboard glued together. Got an old CD? ... that could be easily used as a chassis.

It's a great idea to prototype your robot system using a breadboard and indeed, we connected the entire circuit on a breadboard and then tinkered with code on the Pico to help us get to our desired design. However, it's worth noting that with some double-sided sticky foam pads, it's totally possible to attach the motors and bump switches to the underside of a breadboard. If you can make or buy a small castor wheel for the back end of the robot, then why not consider making the whole robot on a breadboard? See [Figure 3](#).



Geared motors

The motors we used on this build, and on the StoRPer robot are called N20 geared motors. They are widely available but you need to take a little care to choose the correct option. They are available in a lot of different gear ratios and in a lot of different voltage ratings. Whilst you can choose the gear ratio yourself, make sure to buy a pair that are the ones rated for up to 6V. This means they will run well whilst testing with a USB cable supplying power to the Pico, but also later when we use a single-cell Li-ion battery.

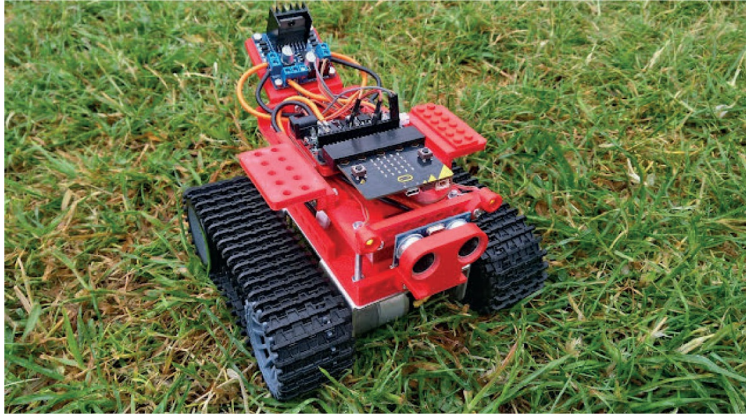


Figure 1: The open-source Modular Tracked Vehicle 'MTV' robot

It's a great idea to prototype your robot system using a breadboard

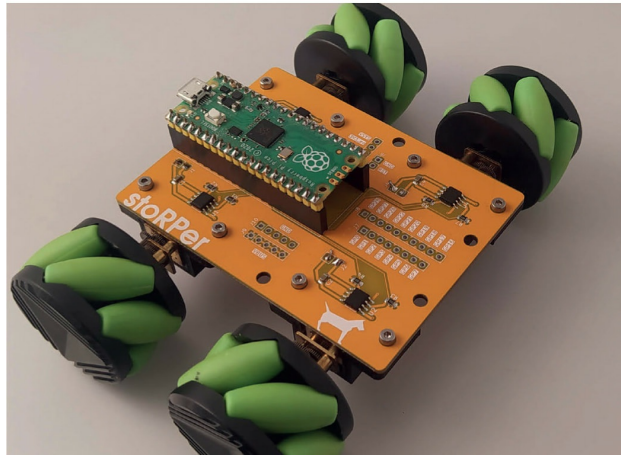
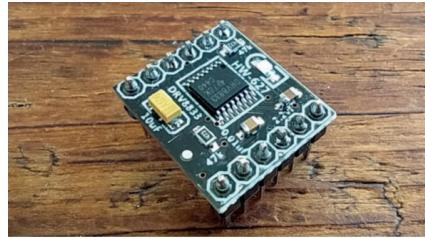


Figure 2: The open-source StoRPer robot is built around a Pico and a KiCad-created PCB chassis

All the 3D design files and STLs are here: rpimag.co/sensorbot.

Motor driver



To drive our motors, we've used a DRV8833 motor controller module. There are numerous similar DRV8833 motor controller modules out there that all operate in a similar fashion. One note is that the DRV8833 has a pin which needs to be tied to a logic high (so a connection to our VSYS, pin 39) that can be called different names on different modules! The pin is often referred to as 'Sleep' or sometimes 'nSLEEP', but on some modules this is labelled on the silkscreen as 'SLP'; on others it's 'EEP'. Whichever you have, the module won't send power to the motor output pins unless this pin is held high.

Mounting momentary microswitches

We also know that we want to add some momentary microswitches to the robot. Almost any microswitches you may already have will do. Most microswitches will have some mount holes. We have a stock of branded Omron SS5GL microswitches as well as some non-branded cheaper clones of this classic switch. The SS5GL has two mount holes that are a little over 2mm in diameter and spaced at 10mm between the centres. This useful dimensional info allows us to quickly add two pairs of holes, slightly under 2mm diameter, so we can self-tap some small M2 bolts into the chassis and hold the switches firmly ([Figure 5](#)).

Another nice feature of the SS5GL-type microswitches (and others) is that they can be connected in either a 'normally closed' or 'normally open' configuration. Normally closed means that the two connections to the switch are electrically connected until the switch lever arm is pressed and then the disconnect. Vice versa, normally open, means the two connections aren't connected until the switch is pressed. As our Raspberry Pi Pico has internal pullup resistors, it makes sense to use the switch connections for normally closed. This means that we will connect each switch to one GPIO pin on the Pico and ground. We'll define the GPIOs as inputs and we'll set the resistors to pull up. In this configuration, when the switches are not pressed, the GPIO will be connected to ground. When the switches are pressed, and the ground is disconnected, the pull-up resistors ensure the GPIO pin is high. We can then write code to say what happens when the pins are low and what happens if one or more of the pins goes high. Conveniently, using the internal pull-up resistors rather than adding external resistors means we have even fewer components to find, sticking with the theme of this robot build!

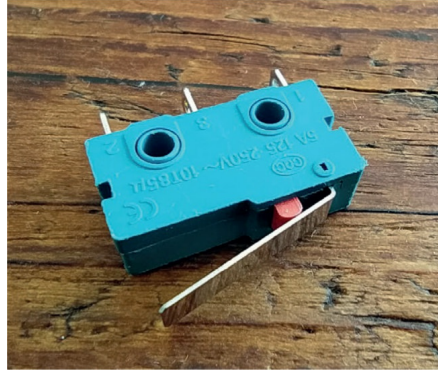


Figure 5: Almost any microswitch can work; we used some affordable SS5GL-style switches

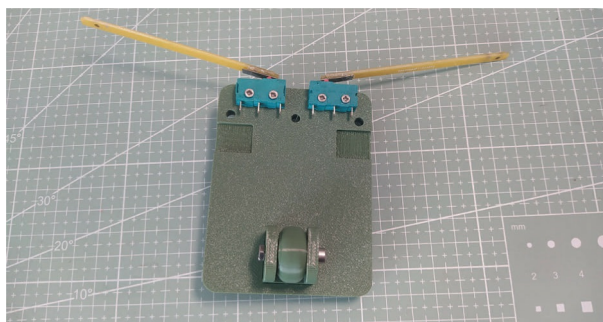
With some ideas and decisions for the chassis and the switches made, it's a good idea to prototype the system and to perhaps iterate on your code. As an example, we began by placing a Pico H onto a breadboard. We connected it up to Thonny and then, as it was a while since we had used this Pico, updated the MicroPython version to the most recent official Raspberry Pi version. We then added our DRV8833 module to the breadboard.

We connected VBUS (pin 40) on the Pico to VCC on our DRV8833 module. We then connected GND (pin 38) on the Pico to the GND pin on the DRV8833. Next, we connected the 'EEP' pin on the DRV8833 (on some modules marked 'SLP' or 'nSLEEP') also to VSYS on the Pico to tie this pin high, enabling the module to supply power to the motors.

We then connected Pico GPIO 14 and GPIO 15 (pins 19 and 20) to the IN1 and IN2 pins on the DRV8833 module and then finally connected one of our 3–6V N20 geared brushed motors to the OUT1 and OUT2 pins on the DRV8833. We are then ready to do our first little code test to check our motor and motor driver are functional using the following code.

Bumper arms

Adding bump switches to a robot is great, but you'll find that, in their standard form, they aren't big enough to detect all objects. The common modification is to attach some kind of lightweight arm to the switch's lever. This in effect makes a system like a cat's whisker! You can experiment with all manner of add-on whiskers for your bump switch mechanism. A popular easy-to-find choice is a length of lollipop (popsicle) stick or a wooden drink stirrer. You could try some small thin stiff wire, or you could even experiment designing and 3D-printing an object to add to your switch. We have a big pile of small offcuts of PCB boards which are the perfect size and length. To attach these to our switch arms, we used a small blob of hot glue, being really careful not to accidentally glue our switch!



```

from machine import Pin, PWM

IN1 = PWM(Pin(14))    #set up PWM for GP14 which connects to IN1 on the DRV8833
IN2 = PWM(Pin(15))    #set up PWM for GP15 which connects to IN2 on the DRV8833

IN1.freq(1000)        #set PWM Frequency
IN2.freq(1000)

IN2.duty_u16(0)        #setting duty cycle of one pin to zero fixes the direction of rotation
IN1.duty_u16(60000)    #setting duty cycle of this pin sets speed of rotation 0-65535

```

All the code examples in this article are online (rpimag.co/sensorbot), so you can download or copy-paste them from there. The above code is **simplest motor_test.py**. Stepping through the above code, you should see that the first line is importing the functions `Pin` and `PWM` from the `machine` library. The next two lines set up PWM on both GP14 and GP15 and give these the name `IN1` and `IN2`, which are our inputs on the DRV8833 module. Next, we set the frequency for the `IN1` and `IN2` PWMs both of which are set to 1000, which is 1000 times per second or 1kHz. To create a PWM signal, we need to set the duty cycle of the PWM at each of our motor input pins. If we set either (but in the example above, `IN2`) to zero, this then fixes the motor to only rotate in one direction, but we can vary the speed of rotation by adjusting the value of the duty cycle of `IN1`. So by setting the value to `IN1`, you can press the ‘Run current script’ button in Thonny to check that your motor rotates, then adjust the value of `IN1.duty_u16(45000)` to a value between 0 and 65535 to play with the speed.

There are a couple of things of note at this stage. If you press the ‘Stop/Restart back end’ button in Thonny, it’s likely your motor will keep running! Your only options are to either set `IN1.duty_u16()` to 0 or, the more likely option, pull the USB cable out of your Pico! In the next stage of developing the code, we will add a system to avoid this and we’ll also add a reverse function using a small trick in the code.

We aren’t going to add the entire code, but rather look at what we will add to the previous motor test example. This code is on the repo, called `basic_n20_motor_test.py`. We import the `time` module for use later in the script, but the main changes begin after setting the `IN1` and `IN2` PWM frequencies. Instead of just setting up the `IN1` and `IN2` duty cycles manually, we will write some small functions to control speed and direction of the motor. The function looks like the code below:

```

def set_motor(speed) :

    if speed > 0:
        # Forward
        IN1.duty_u16(speed)
        IN2.duty_u16(0)

    elif speed < 0:
        # Reverse
        IN1.duty_u16(0)
        IN2.duty_u16(abs(speed)) # absolute
value used

    else:
        # stop

```

```
IN1.duty_u16(0)
IN2.duty_u16(0)
```

As you should see, we begin by defining a function called `set_motor`, which receives an input called `speed`. We are still going to set the speed of rotation of the motor by setting the `IN1` and `IN2` duty cycle values. In the first part of the function, we set up an `if` conditional statement where if the value of `speed` is set to a number larger than 0, we will create a forward motion by setting the `IN1` duty cycle to the value of `speed` and the `IN2` duty cycle to zero. In the next `elif` (else if) statement, we set up conditions for reverse. This time, the `elif` is looking for whether the value of `speed` is less than 0. So if the value of `speed` is a negative number, then we will set `IN1` to be zero and `IN2` to be `speed`. However, PWM values are expected to be between 0 and 65535! We are going to allow the speed to be set as either a positive or negative number between 0 and 65535 and then you'll see in the code for reverse that we've used `(abs(speed))`. The `abs` is short for absolute and basically creates an absolute value that is the distance the value is from zero. So if you input -12345 as `speed`, it will actually use the number 12345. A really quick way to see how `abs` works is to use the REPL in Thonny, input `print(abs(-10))`, then press **RETURN** and it should print '10'. Finally, we can finish this function with an `else` statement where we set the duty cycle of both `IN1` and `IN2` to zero, meaning all motors stop any rotation.

We've then added in this script a small `while` loop which allows us to set up a choreographed sequence of motor moves. You should be able to see in the following code snippet how we have set the motor to go forward by setting `speed` in the `set_motor` function to a positive value, then the motor stops with it set to zero, and then reverses when we set it to a negative value. Wrapping the `while True` loop in `try:` and then the `except KeyboardInterrupt:` is a quick way of avoiding the earlier problem where when we stop the script in Thonny, the motors keep turning. When you run this example, you can then press **CTRL+C** on the keyboard and it will interrupt the script and stop the motor.

We've added a small while loop which allows us to set up a choreographed sequence of motor moves

```
try:
    while True:

        set_motor(30000) #forward
        time.sleep(2)

        set_motor(0)     # stop motor
        time.sleep(2)

        set_motor(-30000) # backward
        time.sleep(2)

        set_motor(0)     # stop motor
        time.sleep(2)

except KeyboardInterrupt:
    set_motor(0)         # stop motor
```


Next, we added one of our microswitches. Again, the code example, called **N20_motor_with_one_button.py**, is on the repo. In this example, you can see we added the line:

```
button = Pin(16, Pin.IN, Pin.PULL_UP)
```

This adds and defines button as being attached to GPIO 16 (pin 21), sets it as an input, and makes sure it is using the internal pull-up resistor. The other wire from the switch is connected to the Pico ground/GND (pin 38). This means, as it is a normally closed switch, when the switch is not pressed, the pin is grounded. Whenever the switch is pressed, the pin will be pulled high via the internal pull-up resistor. This means that we can then change our while loop to look out for button presses and respond accordingly.

```
FORWARD_SPEED = 30000
REVERSE_SPEED = -30000
try:
    while True:

        set_motor(FORWARD_SPEED)

        if button.value() == 1:

            time.sleep(0.05) #debounce

            if button.value() == 1:

                set_motor(0) #stop
                time.sleep(1)

                set_motor(REVERSE_SPEED) #reverse

                time.sleep(2)

                set_motor(0) #stop
                time.sleep(1)

                while button.value() == 1:
                    pass

            time.sleep(0.01)

except KeyboardInterrupt:
    set_motor(0) # Stop motor safely
```

So in the above code snippet, we have set up a couple of variables, FORWARD_SPEED and REVERSE_SPEED, and given them a value. In the loop we set up the motor to always run forwards when the button is not pressed. We've then set up an if statement for what happens when the switch is pressed and returns a high value (1). We've added a little debounce time and check the switch is still high and then it goes into a small choreographed routine with the motor. First, the motor stops for a second, then the motor reverses for a second, then stops again, and then we return out of the if statement to the main loop with the motor travelling forward continuously again. This final single motor with a button should now give you all the clues as to how we are going to set up a double motor and switch system to form our robot. We won't step through that example, but essentially the idea is that you add a second motor using the IN3 and IN4 on the

DRV8833 module and add a second switch to the Pico. You can create a script where the robot drives forwards continuously, but when either switch it pressed, the robot stops, reverses, turns, and then tries to move forward again! If that sounds too complex, we have of course included the script in the examples, called `n20_bump_robot_basic_code.py`.

This could be a great and more affordable robot project for a Code Club

Rear castor-wheel

A common approach for a two-wheel drive robot is to deploy some form of castor-wheel system at the back of the robot. This non-driven 'wheel' tends to act more to balance the robot so it isn't dragging the rear of the chassis. A common and affordable off-the-shelf component is the ball-bearing castor. This is a plastic part that can be screwed or glued onto your chassis that contains a free-rotating metal ball bearing. This is a nice solution as the ball can roll in any direction.

As we had access to a 3D printer, we made a couple of designs of rear wheels to fit into the 20mm square area we defined on our chassis earlier. We went for a more standard wheel on a shaft, but made the edges of the wheel very rounded. Whilst this doesn't rotate in the direction of travel like a true castor, the contact point of the rounded wheel on the travelling surface is small, so offers very little friction and allows the robot to turn and rotate freely.



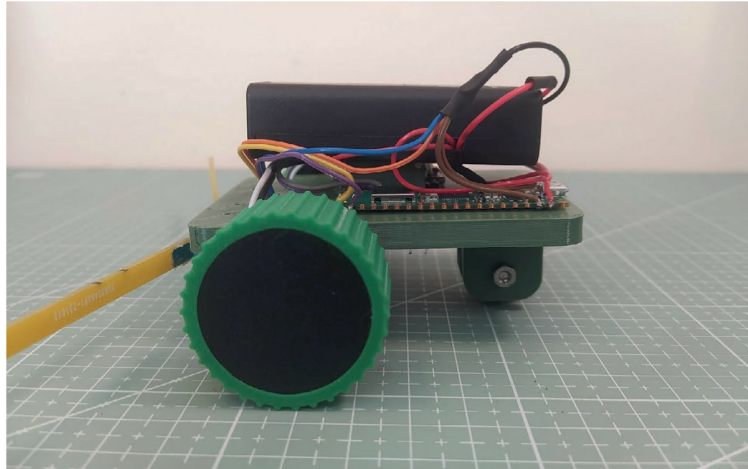
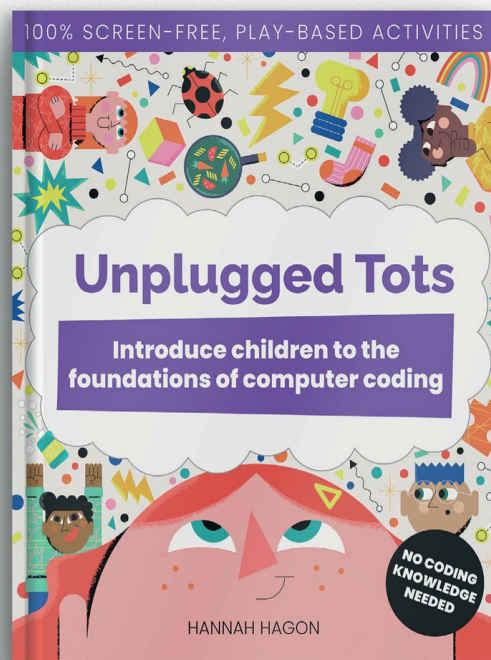


Figure 6: Our simple robot with StoRPer wheels, ready to roll!

We went on to build our prototype onto the 3D printed chassis. We used a separate Pico and DRV8833 module for this build as we wanted to directly wire/solder to the devices rather than use header pins. This made it slightly more compact, but again, there's no reason you couldn't mount a Pico with pin headers with the pins pointing upward, etc. You'll notice our chassis has a small fin on the back. This is a little trick for being able to expand a chassis working area: you can glue a small platform on top of the fin to create an upper area. On this we have used a dab of hot glue to mount an AA battery holder (**Figure 6**). Of course, we aren't using an AA battery, but we are using a 14500 Li-ion battery which puts out 4.2V at full charge. We've wired this to our VSYS and GND connections and it acts as a super-neat little power supply for this robot. For wheels, we've 3D-printed some from the StoRPer design, but there are many wheels available that fit the N20 motor shaft. The StoRPer design files are all available here: rpimag.co/storper.

This has been a fun project and whilst we've been making it, it's struck us numerous times that this could be a great and more affordable robot project for a Code Club or just for someone looking to only spend a pocket-money level of cash to explore the world of Pico robotics! 🟩

Features 100% screen-free
activities — no computer, tablet,
or mobile phone needed



rpimag.co/unpluggedtots

Build a wordsearch clock

Show the time in a unique and very customisable way



Maker

Rob Miles

Rob has been playing with software and hardware since almost before there was software and hardware.



robmiles.com

Create a stylish and ever-changing time display using a Raspberry Pi Pico, a 16×16 NeoPixel panel and multicoloured 3D printing. You can extend this to make an alarm clock which plays suitably sarcastic messages to encourage you to get out of bed. You can find all the resources to create and customise your wordsearch clock in the GitHub repository here: rpimag.co/wordsearchclock.



Figure 1: The time is drawn in sequence, so you know the order of the values

Figure 1 shows a clock created as a wordsearch which shows the time. It uses a 16×16 array of NeoPixels with a light behind each letter. A Raspberry Pi Pico in the clock running a MicroPython program lights pixels behind letter shapes in sequence to spell out the time in words. The colours for each word are chosen randomly, producing a very pleasing display. Or at

least that is what the author thinks. He has always enjoyed a good wordsearch, so he thought he might have a go at creating his own and putting it in a device. So, he started by writing a Python program to make some wordsearches.

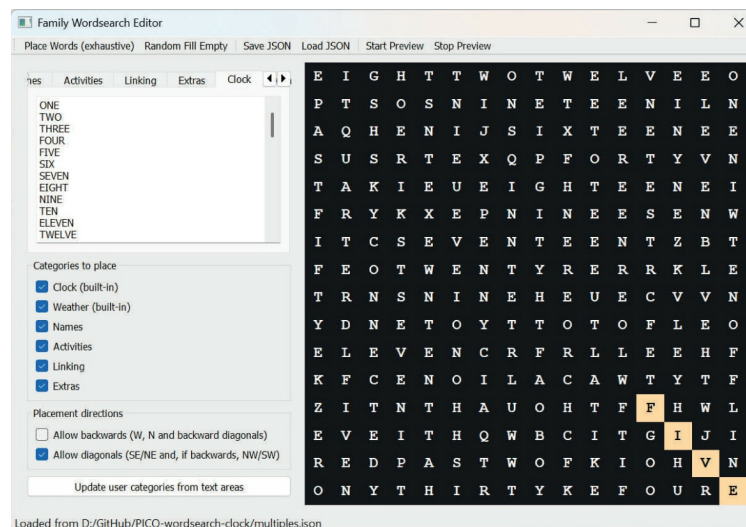


Figure 2: You can add your own words to the wordsearch

Searching for words

The Wordsearch Editor shown in [Figure 2](#) builds wordsearches from lists of words. The words are arranged into categories so that the clock can display other information. The time category contains multiple copies of some of the words to make the time display more interesting. The program builds a wordsearch by placing each word in turn, picking a random location for the first letter and checking that the rest of the word fits. It will also try to fit words inside words if it can. The program will clear everything and start again if it can't place a word. The current arrangement took 15 attempts to find one that worked. It turns out that there is nothing difficult about making complex wordsearches if you are happy to leave your program running for a while. When all the words have been placed, the program generates a JavaScript Object Notation (JSON) file that describes the word arrangement:

```
"grid": {
  "rows": 16,
  "cols": 16,
  "letters": [
    "EIGHTTWOTWELVEEO",
    "PTSOSNINETEENILN",
    "AQHENIJSIXTEENEE",
    "SUSRTEXQPFORTYVN",
    "TAKIEUEIGHTEENEI",
    "FRYKXEPNINEESENW",
    "ITCSEVENTEENTZBT",
    "FEOTWENTYRERRKLE",
    "TRNSINEHEUECVVN",
    "YDNETOYTTOTOFLEO",
    "ELEVENC RFRLLEEHF",
    "KFCENOILACAWTYTF",
    "ZITNTHAUOHTFFHWL",
    "EVEITHQWBCITGIJI",
    "O NYTHIRTYKEFOURE"
  ]
}
```

```
"REDPASTWOFKIOHVN",
"ONYTHIRTYKEFOURE"
]
```

The program builds a wordsearch by placing each word in turn, picking a random location for the first letter and checking that the rest of the word fits

YOU'LL NEED

- Raspberry Pi Pico W or 2 W
- RGB panel WS2812B 16×16 pixels flexible module
- DuPont connecting wires, 10cm male-to-male and male-to-female
- Micro USB / USB-C cable for Pico W / 2 W
- M2 Philips pan head screws, 5mm long, to hold the Pico and the speaker in the case
- Two colours of PLA filament (black and white work very well together)
- **If you want to make the alarm clock version, you will also need:**
- DFPlayer – search for 'DFPlayer Mini mp3'
- MicroSD card – between 4GB and 16GB
- Four non-latching push-buttons which fit through a 13mm diameter hole
- 40mm speaker with mounting holes spaced at 32.5mm

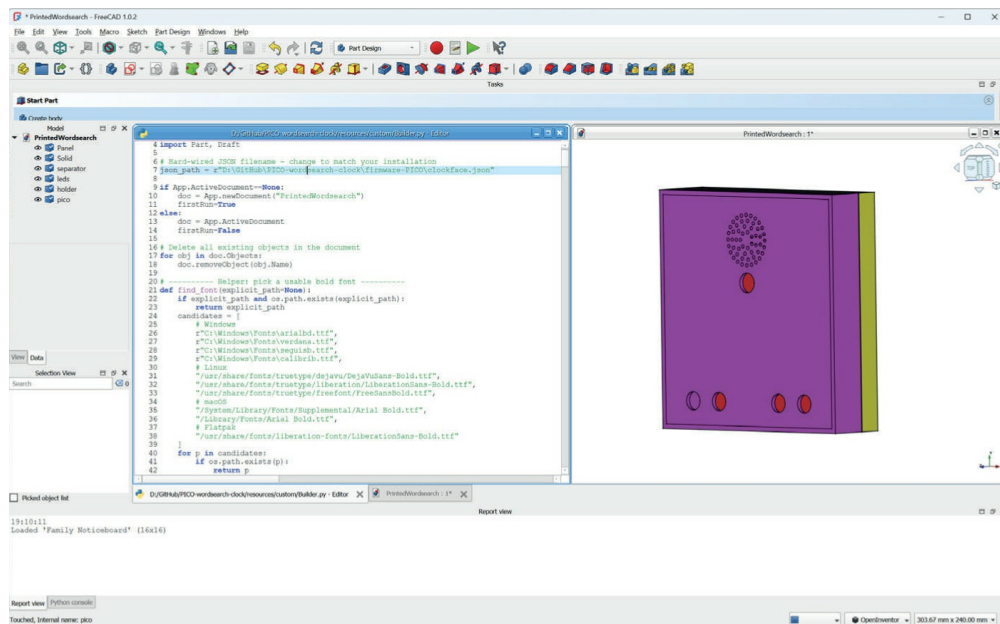


Figure 3: The program uses fonts on the local machine for the character designs

This code above shows the start of the wordsearch description which gives the size of the grid and the letter arrangement. This is followed by an array of word descriptions:

```
"words": [  
  "word": "ONE",  
  "category": "clock",  
  "direction": "SE",  
  "cells": [  
    {  
      "row" : 1,  
      "col" : 3  
    },  
    {  
      "row" : 2,  
      "col" : 4  
    },  
    {  
      "row" : 3,  
      "col" : 5  
    }  
  ],  
],
```

The code above describes the position of the word 'ONE' in the wordsearch. It gives the location of each of the letters in the word. The MicroPython program that displays the clock uses this to create an index giving the positions of the words it needs to display the time. The same JSON file is also used by another Python program that runs inside the FreeCAD design tool to create the clock case.

Making a case

[Figure 3](#) shows the **Builder.py** program running inside FreeCAD. The program creates the front and back of the clock along with a separator that fits between the LED panel and the front of the clock. It also generates objects representing the LED panels and the Pico inside the clock. This helps a lot with the design process. The two halves of the case contain grooves and ridges so that they snap together. The alarm clock version of the case contains fittings for the speaker and DFPlayer device as well as the Pico. The front of the case is printed using two different filament colours. The builder program creates two output files which are merged when the object is sliced for 3D printing.



Figure 4: The letters have been moved back so that they are visible in the picture

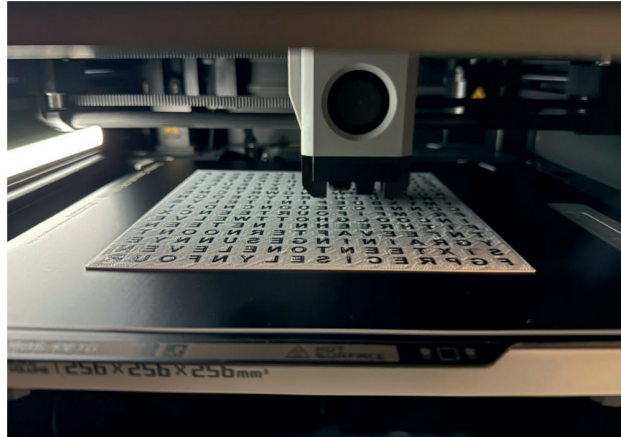


Figure 5: The first few layers must be printed very slowly so that the individual letter elements have time to stick to the print bed

Figure 4 shows the two components that are 3D-printed together to make the clock face shown in **Figure 1**. The black outer case contains holes which are filled by the white letters. Light shines through the text to display the letter for that square. The case is printed face-down on a smooth print bed.

Figure 5 shows a clock face being printed. This clock will have black letters on a white background. You can also make a clock with white letters on a black background.

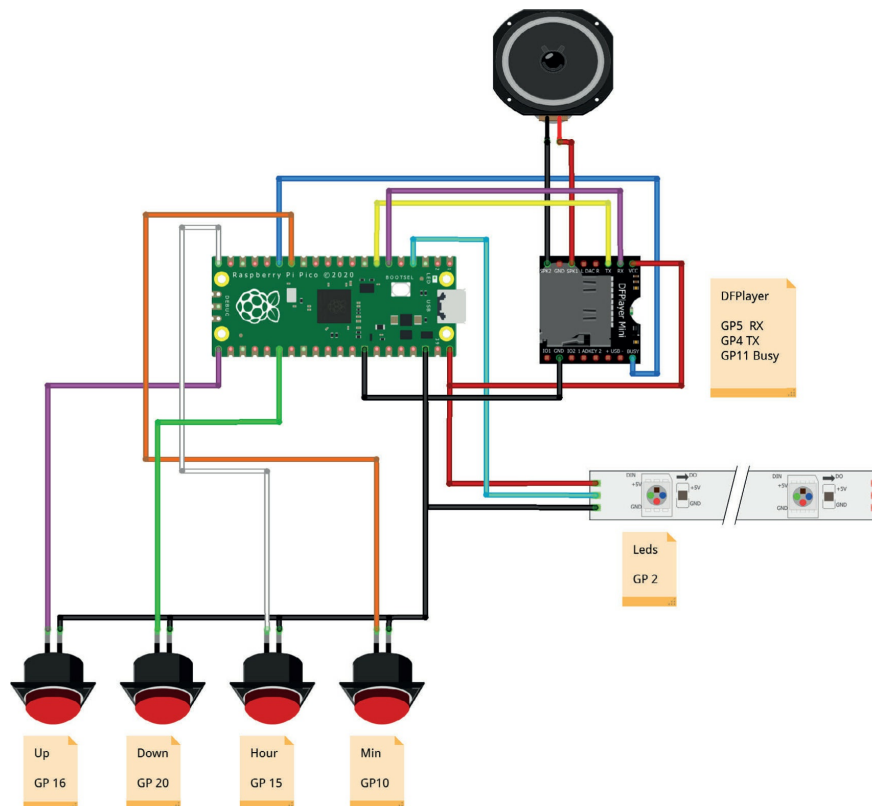


Figure 6: You only need the Pico W and the LEDs if you are not making the alarm clock

Wiring up

[Figure 6](#) shows the circuit of the clock. The four buttons are used to set the alarm time. The DFPlayer is fitted with a microSD card that contains the alarm sounds encoded as MP3 files. The DFPlayer is controlled via a serial connection to the Pico. There is also a connection between the DFPlayer and Pico which is used to detect when a sound sample has finished playing. This is read by GP11 on the Pico. The LEDs are connected to a single pin and powered from the USB input to the Pico. A standard USB 5V adapter can be used to power the clock.

[Figure 7](#) shows the internals of the completed clock. The connections are made using DuPont male-to-male cables. The pins at the ends of the cables are soldered directly into the Pico and onto the switch connections. The DFPlayer is supplied with pins which are connected using male-to-female DuPont cables. The power supply output from the VBUS pin on the Pico is split between the pixels and the DFPlayer.

Managing managers

The clock is controlled by MicroPython code that is structured as a set of managers. Each manager is a software object that has setup, update, and teardown behaviours. Starting the clock involves reading a settings file to determine which managers are to be loaded, creating the manager objects, and initialising them. When the clock is running, each manager is updated in turn. There are managers for the Wi-Fi connection, the network clock, the wordsearch application, the pixels, the MP3 player, and the input buttons. A manager can generate events and provide services to other managers; for example, the button manager will tell the wordsearch application when the user presses a button. The managers are configured in the **settings.json** (JavaScript Object Notation) file stored with the code.

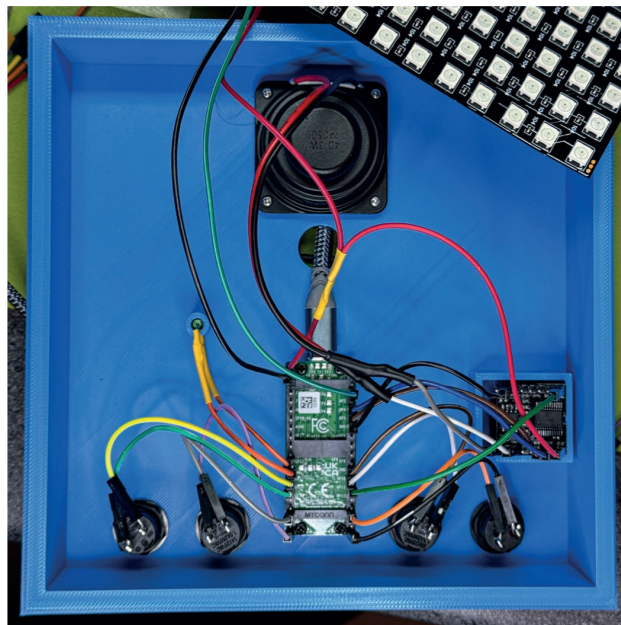


Figure 7: This version of the clock also contains a tilt sensor which is not presently used by the software

The settings info for the Wi-Fi manager in **settings.json** includes the network SSID and password values.

```
"App_wordsearch_clock" : {
  "key_repeat_interval_ms" : 500,
  "start_audio_track_no" : 3,
  "end_audio_track_no": 20,
  "run_on_power_up": true,
  "enabled" : true,
  "wordsearch_file" : "clockface.json",
  "wordsearch_letter_delay_ms": 250,
  "alarm_timeout_ms" : 30000,
  "key_repeat_delay_ms": 1000,
  "wordsearch_word_delay_ms": 1000,
  "dependencies": [
    "clock",
    "gpio"
  ],
  "alarm_sample_interval_ms": 1000,
  "wordsearch_display_gap_ms": 5000
}
```

The JSON above configures the wordsearch clock manager. It provides all the settings information for the clock, including which managers the wordsearch is dependent on. There are similar entries for all the managers that are to be used. The device contains a range of managers for different devices. Any manager with a name starting with 'App_' is an application.

```
App_wordsearch_clock_manager.py
base_manager.py
clock_manager.py
dfplayer_manager.py
display_manager.py
gpio_manager.py
hullos_manager.py
mqtt_manager.py
pixel_manager.py
rotary_encoder_manager.py
stepper_manager.py
uart_manager.py
wifi_manager.py
```

The code above shows some of the currently available managers. They make it very easy to use different devices together. You just create a **settings.json** file that brings the managers together and configures them.

QUICK TIP

The settings for the alarm time are stored in a small extra file so that the entire settings file does not have to be rewritten when the alarm time is changed.

Eventful events

Once the managers start running, they can connect to events and services that they provide for each other:

```
evt = self.clb.get_event("clock.minute")
if evt:
    evt.subscribe(self.on_minute_tick)
```

The code above runs when the wordsearch manager starts up. It connects the method `on_minute_tick` in the wordsearch manager to the minute event provided by the clock. When the clock is active, it will call this method every minute.

```
def on_minute_tick(self, event, data) :
    if self.first_run:
        if self.settings["run_on_power_up"]:
            self.first_run=False
            self.start_show_time()
```

The code above shows the method that runs each minute. It checks a setting to see if the clock should display the time when it is powered up. If `run_on_power_up` is true, the code checks to see if this is the first time that `on_minute_tick` has been called. If it is, the `start_show_time` method is called to start displaying the time. This makes sure that the clock will start showing the time when it has valid clock data.

At your service

A manager can use services provided by another. The `get_service_handle` method returns a reference to an object which exposes all the services provided by the specified manager.

```
self.dfplayer = self.get_service_handle("dfplayer")
```

This statement runs inside the wordsearch clock manager when it starts up. The clock needs to use the MP3 player to generate alarm sounds, and this is how it gets access to the player. The variable `self.dfplayer` is set to refer to an object which has all the behaviours of the `dfplayer` manager. Code in the wordsearch clock manager can now call methods on `self.dfplayer` manager (the one that plays audio samples for the alarm). It can then call methods on the player to tell it what to do:

```
def play_alarm_sample(self):
    self.dfplayer.play(random.randint(self.start_audio_track_no, self.end_audio_track_no))
```

The `play_alarm_sample` method above plays a random alarm sound by calling the `play` method on the `dfplayer`. Each manager also exposes all its available methods as commands you can enter via the serial console when the clock is running.

```
[dfplayer]
dfplayer.play      - Play track n
dfplayer.stop      - Stop playback
dfplayer.volume    - Set volume (0-30)
```

If you want to play a track, you can enter a command at the serial console to do this:

```
dfplayer.play 3
```

The above command would start track 3 playing. Being able to send commands to managers as the system is running is very useful when creating and debugging a device.

Manager magic

You might be starting to think that using all these managers, service objects, and events makes creating applications rather complicated. Well, it turns out that the author has discovered that not using these techniques makes things even harder. Spreading behaviours across multiple managers and providing well-defined ways that they can communicate makes it much easier to create complex devices.

Data-driven for the win

This is an example of a ‘data-driven’ application. The starting points for the design and the program behaviours are JSON files of structured data. There are no ‘hard-wired’ values in the program code: everything is loaded from data when the programs run. The **clockface.json** file is used by the wordsearch clock program and in FreeCAD to make the clock body; the **settings.json** file is used to specify the behaviours that are required and how they should be configured. If we want to make a new application, we just need to create a new manager and bring together the required behaviours for it.

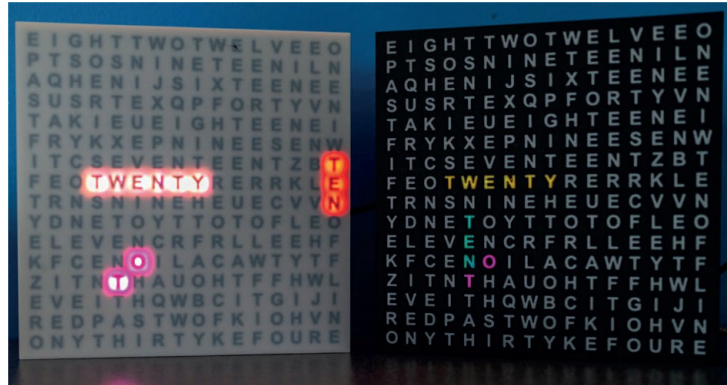
The present version of the clock just displays the time. The next version of the clock will add Message Queue Telemetry Transport (MQTT) support and additional words so that the clock can display other information as well. 

QUICK TIP

You can play different ranges of alarm sounds by changing the start and end track numbers.

Reading, reading, and reading

You can make two versions of the clock case, as shown below. On the left-hand clock, the text is displayed by lighting the square around a black letter shape. On the right, the text is displayed by shining a light through a white letter shape. It turns out that when you shine light through letters, you need extra diffusion behind the letter (otherwise light shines through gaps around the letter), so you can't change from one version to another by just swapping the black and white filaments. You tell the **Builder.py** program the version that you want. 



You could use other colour combinations, but make sure that one of them is fairly transparent

Free eBook!



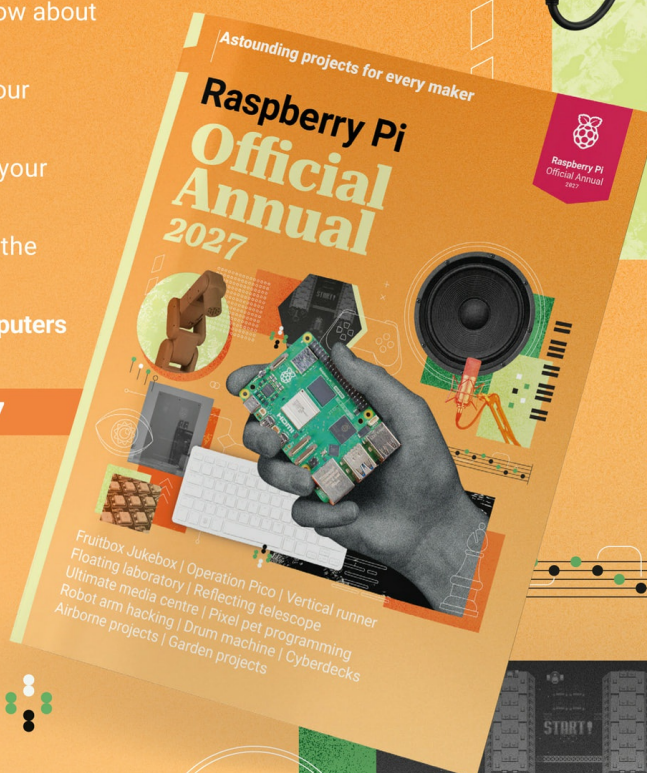
Download your copy from
rpimag.co/freecadbook

Raspberry Pi Official Annual 2027

200 pages of Raspberry Pi

- A **get started guide** covering every Raspberry Pi
- **Everything you need** to know about Raspberry Pi 500+
- **Troubleshooting tips** for your latest build
- **Inspiring projects** to fuel your next project idea
- **Tutorials on how to build** the ultimate media centre
- **Pushing Raspberry Pi computers** to extreme limits

rpimag.co/annual27



Raspberry Pi

BOOK OF MAKING 2027

Technology isn't just something you buy — you can build it yourself! With a bit of know-how you can 3D print moulds to make personalised soap bars, control a robot arm, add sensors and reactive LEDs to an old quilt, or anything else you put your mind to.

Whether you want to build an underwater rover, a wordsearch clock, or a digital jukebox, there's something for you here. We've rounded up a few of our favourite makes previously published in Raspberry Pi Official Magazine to keep you entertained for an afternoon, a weekend, or longer.

180 pages of hacking and making

Build a simple sensor robot from scratch

Experiment with long-range comms platform Meshtastic

Turn a toy skeleton into a jumping, dancing robot

Design, prototype and manufacture a custom PCB