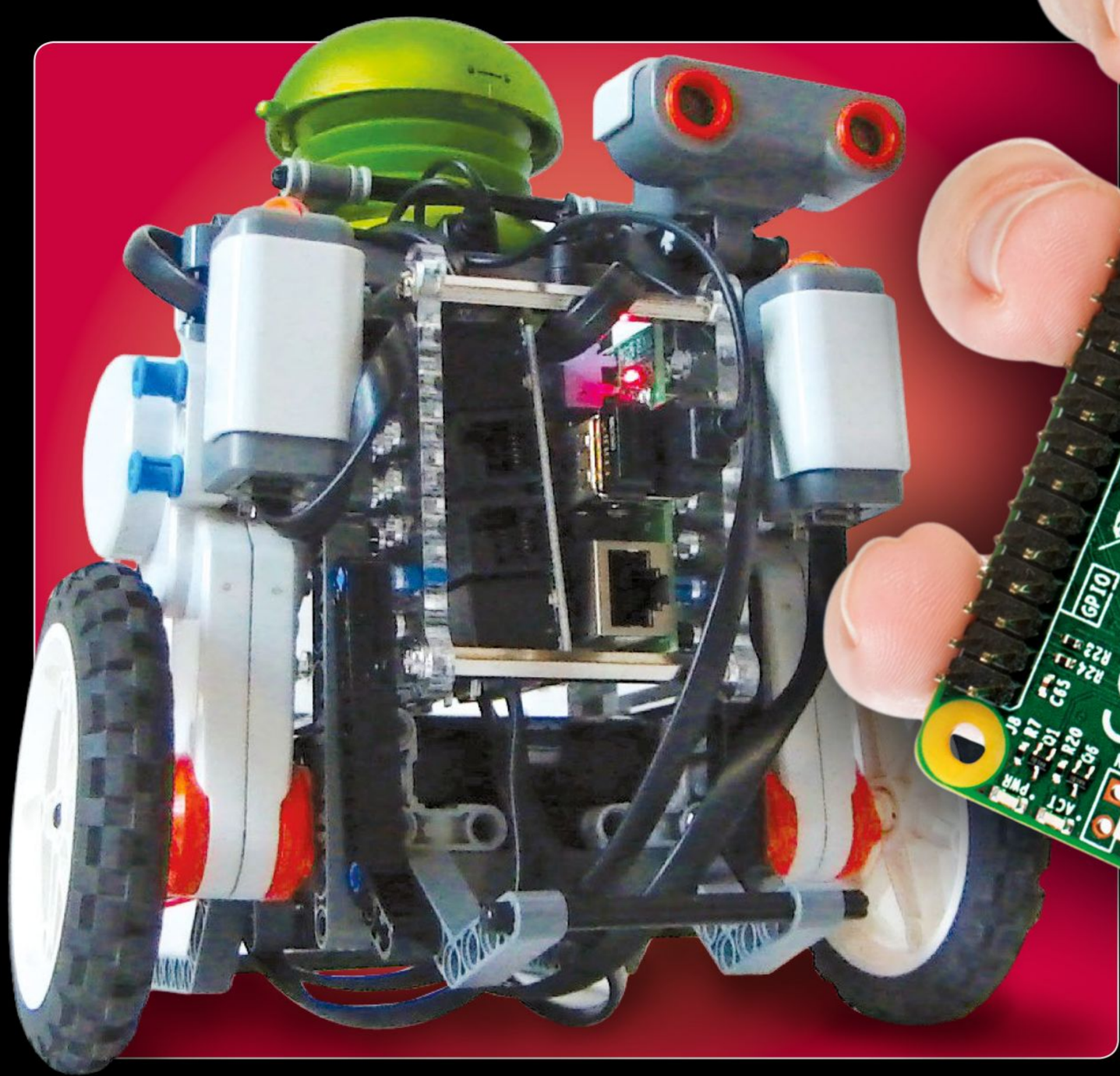
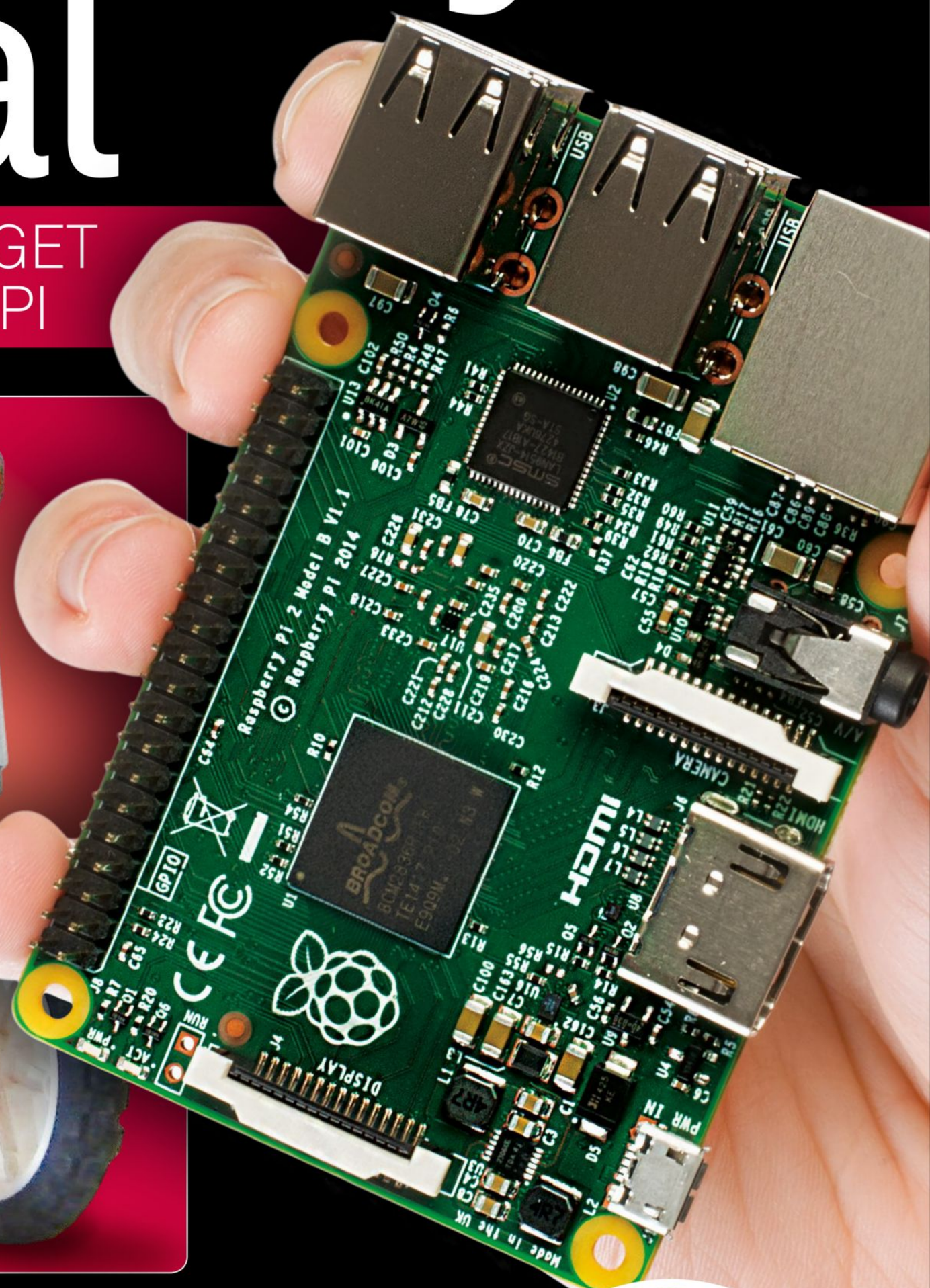
NEW

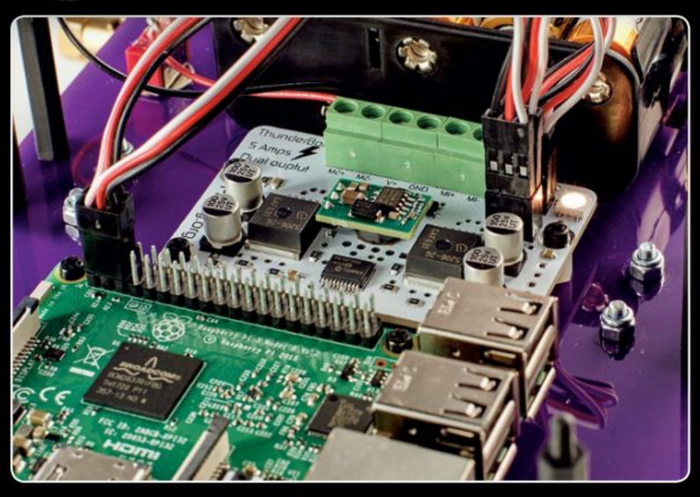
100% INDEPENDENT

Raspberry Pi Annual

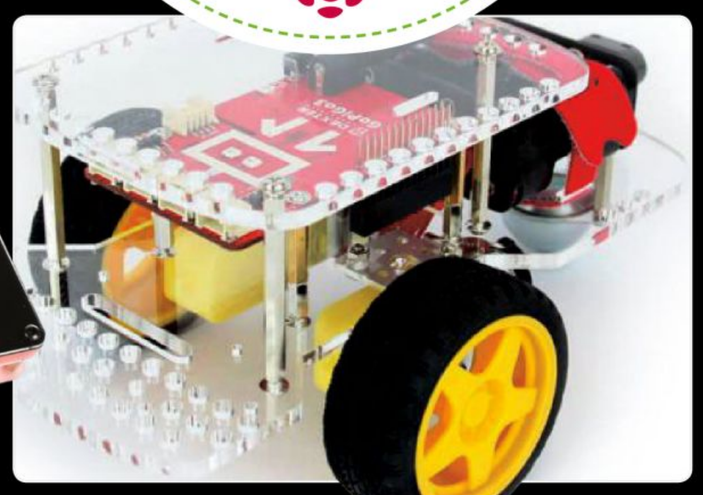
EVERYTHING YOU NEED TO GET THE MOST FROM YOUR RASPI







180 PAGES OF AWESOME PI PROJECTS



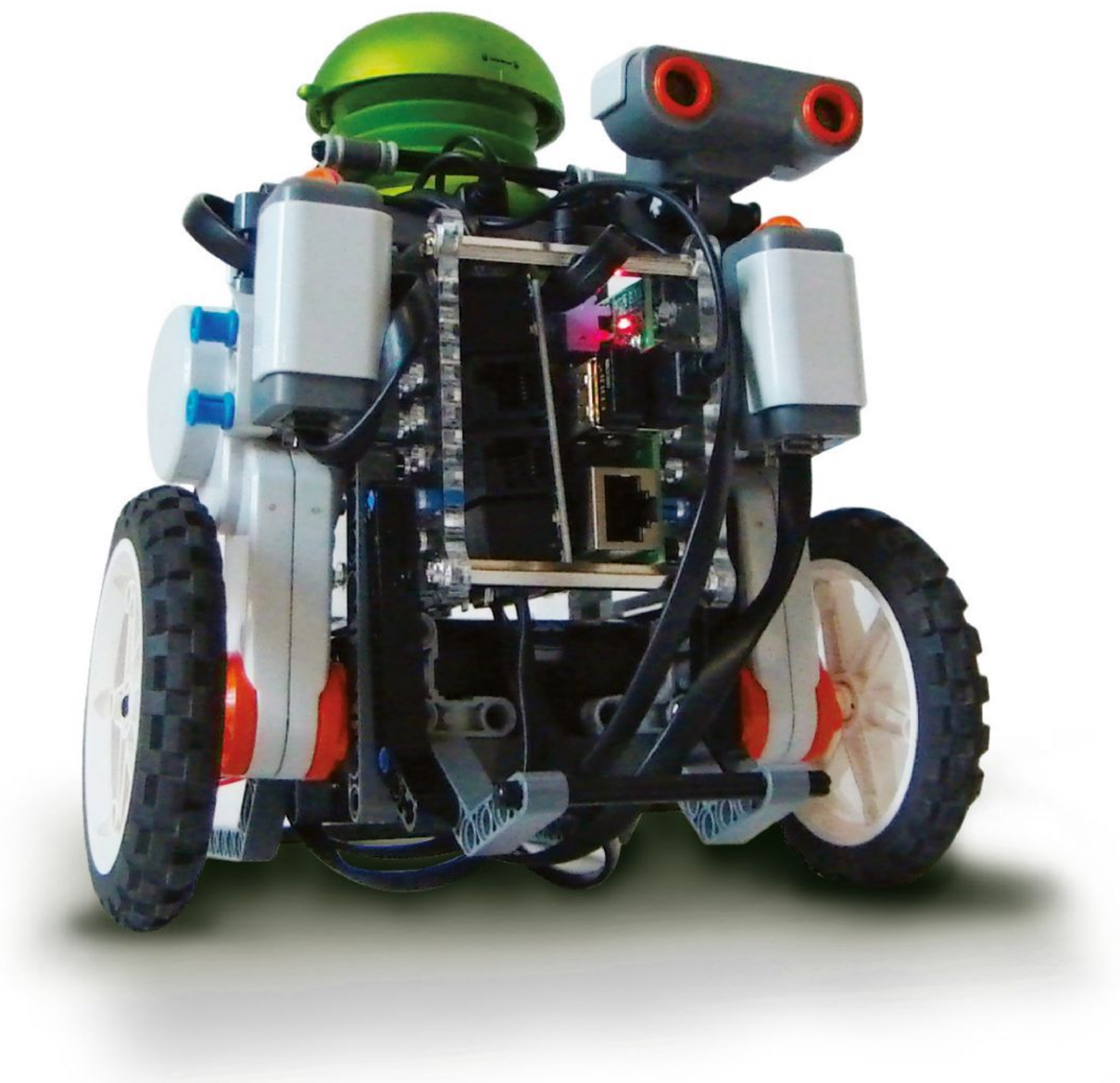
Smart home ideas • Operating systems • Pi gaming

100% UNOFFICIAL

Digital Edition

FUTURE

VOL. 6



Welcome to

Raspberry Pi Annual

The Raspberry Pi was first created in 2012, with the intention of making digital creation accessible to all. Since then, more than 25 million units have been sold to adults and children alike. Pis of different kinds are used in classrooms, libraries, research laboratories, and of course, hackerspaces all over the world. Builders, coders and hackers are using this amazing device to create everything from doomsday switches to fruit ripening detectors. In this new Raspberry Pi Annual, we'll give you some expert tips on how to get the most out of your Pi. We'll also show you the best operating systems, how to create a virtual Pi assistant, plus a ton of other exciting projects – and so much more. Welcome to the Raspberry Pi revolution!

This bookazine is printed on recycled paper. It's important that we care about our planet and make a difference where we can, for us and every generation that follows.

Raspberry Pi Annual

Future PLC Richmond House, 33 Richmond Hill,
Bournemouth, Dorset, BH2 6EZ

Editorial
Editor **Alice Barnes-Brown**
Senior Designer **Adam Markiewicz**
Editorial Director **Jon White**
Senior Art Editor **Andy Downes**

Advertising
Media packs are available on request
Commercial Director **Clare Dove**
clare.dove@futurenet.com

International
Head of Print Licensing **Rachel Shaw**
licensing@futurenet.com

Circulation
Head of Newstrade **Tim Mathers**

Production
Head of Production **Mark Constance**
Production Project Manager **Clare Scott**
Advertising Production Manager **Joanne Crosby**
Digital Editions Controller **Jason Hudson**
Production Managers **Keely Miller, Nola Cokely, Vivienne Calvert, Fran Twentyman**

Management
Chief Content Officer **Aaron Asadi**
Commercial Finance Director **Dan Jotcham**
Head of Art & Design **Greg Whitaker**

Printed by William Gibbons, 26 Planetary Road, Willenhall, West Midlands, WV13 3XT

Distributed by Marketforce, 5 Churchill Place, Canary Wharf, London, E14 5HU
www.marketforce.co.uk Tel: 0203 787 9001

Raspberry Pi Annual, Volume Six
© 2019 Future Publishing Limited

We are committed to only using magazine paper which is derived from responsibly managed, certified forestry and chlorine-free manufacture. The paper in this magazine was sourced and produced from sustainable managed forests, conforming to strict environmental and socioeconomic standards. The manufacturing paper mill and printer hold full FSC and PEFC certification and accreditation.

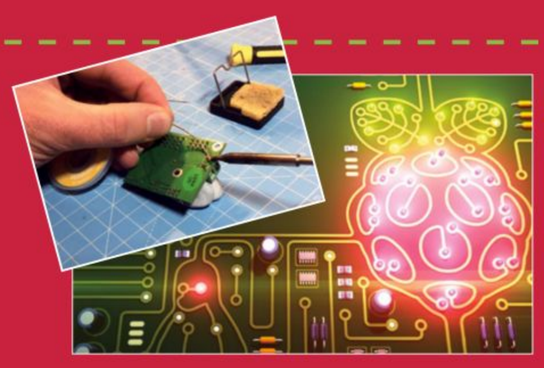
All contents © 2019 Future Publishing Limited or published under licence. All rights reserved. No part of this magazine may be used, stored, transmitted or reproduced in any way without the prior written permission of the publisher. Future Publishing Limited (company number 2008885) is registered in England and Wales. Registered office: Quay House, The Ambury, Bath BA1 1UA. All information contained in this publication is for information only and is, as far as we are aware, correct at the time of going to press. Future cannot accept any responsibility for errors or inaccuracies in such information. You are advised to contact manufacturers and retailers directly with regard to the price of products/services referred to in this publication. Apps and websites mentioned in this publication are not under our control. We are not responsible for their contents or any other changes or updates to them. This magazine is fully independent and not affiliated in any way with the companies mentioned herein.

Future plc is a public company quoted on the London Stock Exchange (symbol: FUTR)
www.futureplc.com

Chief executive **Zillah Byng-Thorne**
Non-executive chairman **Richard Huntingford**
Chief financial officer **Penny Ladkin-Brand**

Tel +44 (0)1225 442 244

CONTENTS

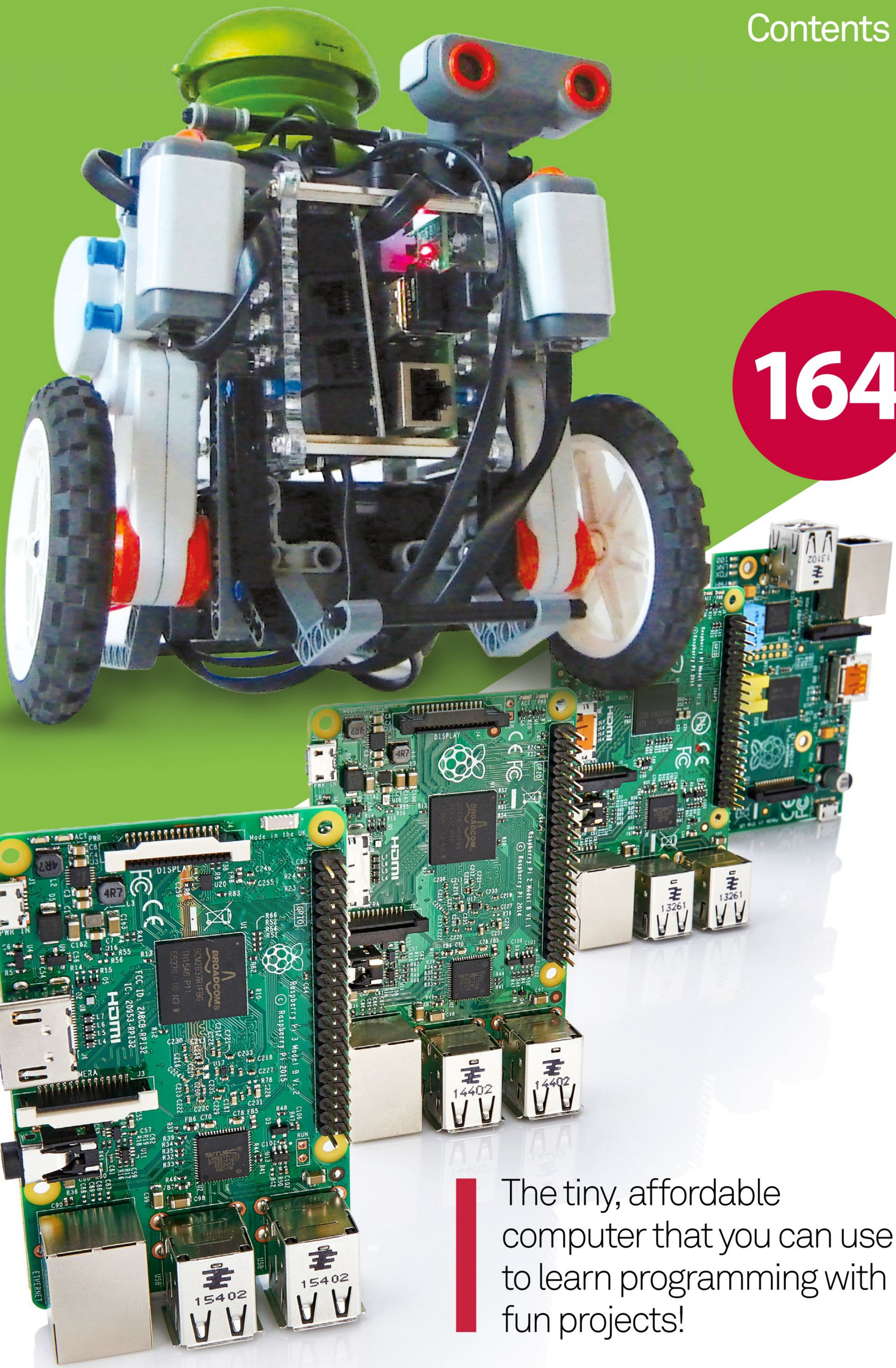


08 The best maker projects

Learn how to make some really cool stuff with this winter project selection box

18 Raspberry Pi 4 B
20 The Raspberry Pi 4 full desktop kit
21 Raspbian Buster
22 Supercharge your Raspberry Pi
26 Copying files from your PC to your Pi
30 Debug serial comms
32 Raspberry Pi 3 A+
33 Compute Module 3+
34 Manjaro Arm v8
35 Raspbian 11.2018
36 Optimise your Pi for better performance
38 Tweak your Raspbian config to perfection
40 Make a Raspberry Pi 2 HTPC
42 Custom embedded Linux images
46 Keep track with Status Board
48 Raspberry Pi Robots
58 4tronix PiBug 2WD
59 4tronix Pi2GoMk2
60 How I made: Robot Arm
62 Remotely control your Raspberry Pi
64 Chromecast: build your own streaming stick
66 Stream live to YouTube with your Pi camera
68 Watch videos easily
70 Build a Raspberry Pi stop-motion studio
74 Create a Raspberry Pi camera booth
78 Make a digital photo frame
82 Build a Pi-powered portable bodycam
84 Turn Pi into a router
88 Protect your identity with the VPiN
90 Monitor your network using a Raspberry Pi
92 Raspberry Pi PoE HAT
93 Picade x HAT
94 Using analogue HAT interfaces
96 Controlling the Pi from a web-based app
98 Get inspiration for your projects with Google
100 Moving servos and flashing LEDs
104 Breakout Garden
105 Thermal camera
106 Sharing data from remote sensors
108 Monitor and archive data
110 Monitor humidity with the I2C bus
114 Build a space station alert system
116 Assemble a night vision baby monitor
120 Velleman VM205
121 MAX30105 Multi Sensor
122 Get and use API access to monitor train times
124 Activate a fan with motion control
126 Build and configure a home security system
128 Create a Raspberry Pi OLED monitor
130 Print wirelessly with your Raspberry Pi
132 How I made: Holographic Audio Visualiser
134 Create your own digital assistant
140 Low-power wireless tagging
145 GameShell kit
146 Create an information radiator with DAKboard
148 Stream PC games around your home
150 Creating a classic system demo
152 Emulate the Atari ST on your Raspberry Pi
156 Build a Raspberry Pi Minecraft console
160 Mod your Minecraft
164 20 Raspberry Pi hacking projects

164



The tiny, affordable computer that you can use to learn programming with fun projects!

THE BEST MAKER PROJECTS

Gather round the warmth of the soldering iron and learn how to make stuff with **Jonni Bidwell's** winter project selection box.

Raspberry Pi and the wider Maker Movement have put a great deal of power into people's hands. With the right knowledge, some tools and several cups of tea it's easier than ever to turn a crazy project idea into reality.

Building a prototype device used to be prohibitively expensive, but now everything you need can be purchased online. Not only are only the components readily available at reasonable prices, you'll find in-depth instructions, detailed pinouts and comprehensive specifications.

Still, when you're starting out it's natural to be daunted by circuit diagrams, soldering, 3D printing and coding. It's also easy to be put off by beginner projects that seem overly simplistic (although you shouldn't – flashing an LED is pretty informative). So we've compiled a collection of our favourite projects that anyone can enjoy. These can be followed to the letter, or adapted to suit your tastes, requirements or whim. Whether you're after utility (like a low-power NAS box), entertainment or flashy LEDs, we've got 10 exciting projects to inspire you.

Taking your first steps...

Start putting your projects together with breadboards and soldering.

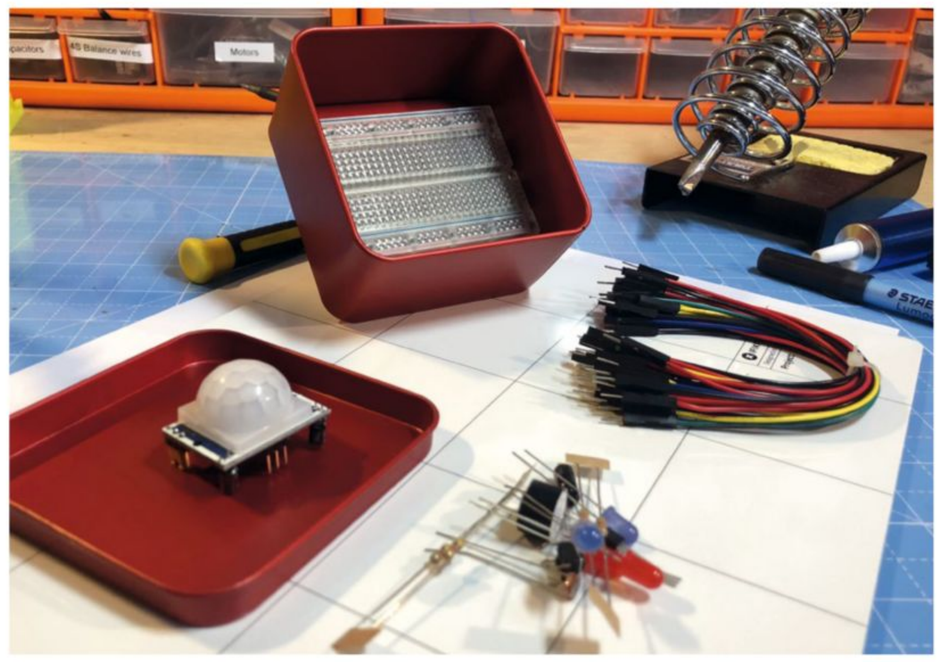
Just as programming often begins with a program that prints `Hello World`, so your first maker project might also be something which appears fairly trivial. But don't worry, these humble beginnings can be built upon. It's also common for first attempts at programming to result in a series of baffling error messages, and it's likewise common for one's first makes to go awry. Don't be put off. Syntax errors will continue to haunt you throughout your programming career, and even seasoned makers melt the wrong thing from time to time. Soldering mistakes can more often than not be undone, and where they can't, it's often only an inexpensive component that needs to be replaced.

A collaboration between ModMyPi and Buyapi.ca has culminated in their YouTube Workshop Kit. It has 10 great Pi projects to get you started, and it comes in a fetching red tin. Inside you'll find a breadboard and a selection of jumper wires and components. Go to **http://bit.ly/pi-youtube-kit** to check the accompanying video tutorials or download PDFs. The workshop takes you through flashing LEDs and buzzing buzzers, to more complex affairs involving temperature and light sensors. The final project connects all the previous ones to make a full-blown motion detector.

Breadboarding enables you to build circuits without having to worry about soldering. It's still possible to make things smoke, but you have to try quite hard. Every hole in each row, and every row in each column of the breadboard are connected, so current travels through your circuit as one navigates a grid system in a town.

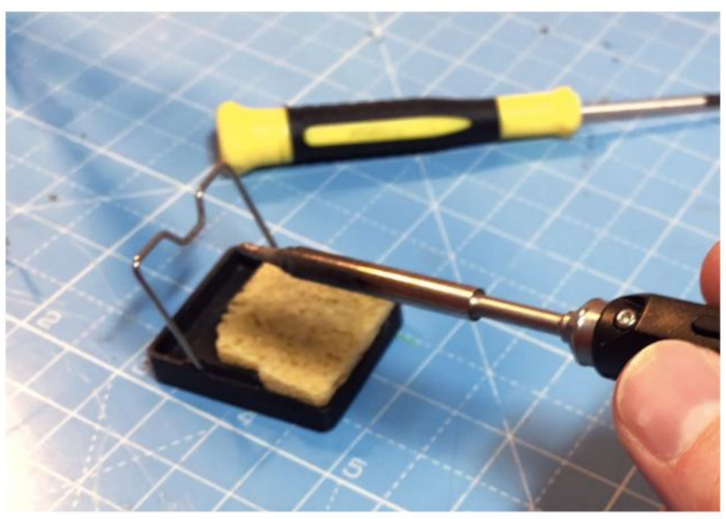
The first things you'll make commonly involve LEDs and resistors. The resistors are there to protect against excessive current going through the former and also to protect against drawing too much current from the source (for example, the Pi's GPIO or 5V pins). LEDs have a positive and negative side. If you connect them the wrong way they probably won't break, but if the voltage across them becomes too great (right way or wrong way) then you will. The positive side of the LED almost always is the one with the longer leg, and circuit diagrams usually depict the resistor going on this side of the LED, although in practice it doesn't matter. On your electronic travels, you'll also encounter buttons, switches, capacitors and many lengths of jumper wire.

The top and bottom two rows of the breadboard are usually labelled positive and negative, although there's nothing special about them. For your first project you'll probably not use the +5V pins on your Pi or Arduino, but rather will use a 3.3V GPIO pin. You'll finish your circuit at a ground pin, and the convention it to usually connect these to a negative row of the breadboard.



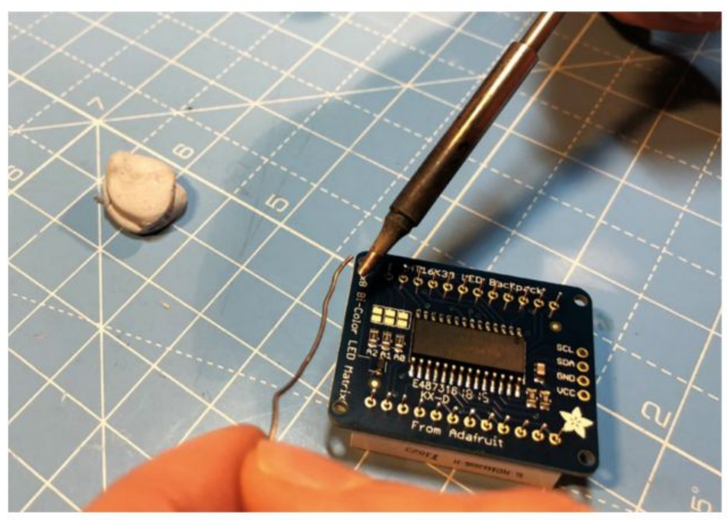
Even if you don't do any of the projects, the lengths of wire and breadboard in a tin will certainly be useful later.

HOW TO SOLDER LIKE A PRO



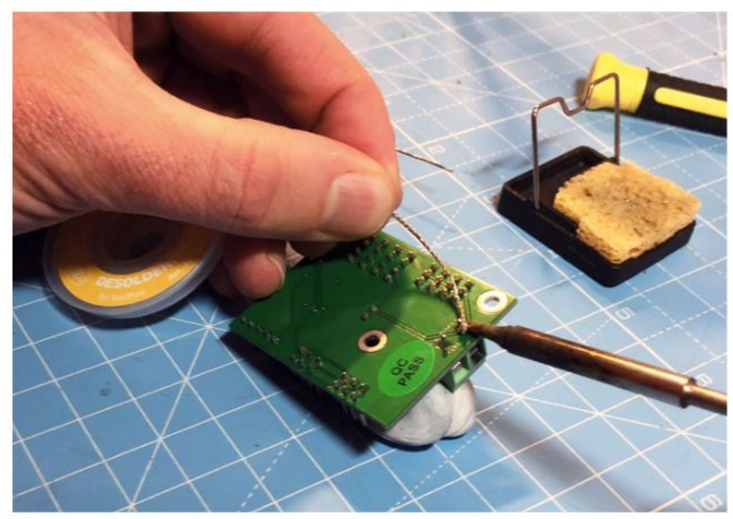
1 Keep things clean

Clear a neat space on a heat-proof surface. Have a solder sponge to hand. You can hold your items together in place either with some elaborate clamp setup, or with good old-fashion Blu tack and gravity. Use the right size of soldering iron for the job in hand. Heat up your soldering iron and when it's hot, clean it on the damp sponge.



2 Solder on

Gently silver the tip of the soldering iron with a little solder. This will help solder flow nicely off it. Carefully hold the solder reel on top of the join and ever so deftly, tap it with the soldering iron. Hold it there until the solder starts to melt. If it doesn't melt nicely, try cleaning and silvering again. If it melts too readily, check the next step.



3 Unsolder any errors

If you're unhappy with your joint then the solder can be removed with a desoldering wick. Heating this up on top of the joint will help remove the solder. Desoldering pens are popular for cleaning up bigger messes. Click these down and then release the vacuum to literally suck the molten solder into the pen.

Pimp your Pi

Turn the lil' Raspberry Pi into a showstopper!

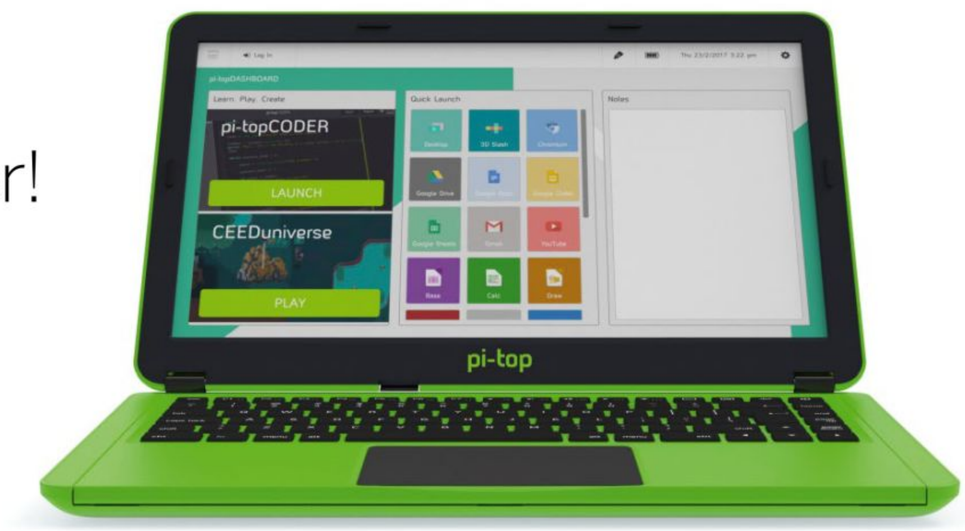
Apart from building wonderful projects, you can invest in some wonderful accessories to make your Pi look pretty. Pimoroni's PiBow cases are extremely popular and come in a range of shapes and sizes. We favour the slimline Coupé models, which allow access to the GPIO pins and have a cutout on the lid so that a heatsink can be installed on the SoC. If you're going to be doing prototyping, you can replace the base with a larger one that accommodates a 400-pin (or two 170-pin) breadboard. This keeps your projects a little neater and tidier.

If you have access to a 3D printer, or a 3D printing service, then you can go one step further and design your own case. The quality and range of materials available have evolved significantly over the past few years. Gone are the days where your print, if it worked, would be a lumpy, off-white insult to your design. These wood-finish materials are impressive, and you'll find all kinds of designs on sites such as **www.thingiverse.com**, so you can start with something having the correct dimensions – always handy!

Hey big spender maker!

Forget Chromebooks, PiBooks is where it's at! Alright, at £260 it's not cheap, but the PiTop puts your Pi (not included) in a neat laptop form factor in a bright green hue. Slide the keyboard forward and you'll find a whole area for prototyping and playing with your Pi. The PiTop comes with an exclusive Inventors' Kit that includes projects to get you started with GPIO, sensors and coding.

A SmartPi Touch LEGO case is also available for GoPros, although that seems a little like overkill.



The Pi Top transforms your Pi into a laptop. It does not glow that colour in the dark, unfortunately. Image credit: ModMyPi

Thanks to the enthusiastic maker community there are a number of things you can make that require little more skills than putting together LEGO. In fact, here's one that you can even incorporate into a LEGO project. For this outing, we're going to transform the Pi into a fully functional tablet, complete with LEGO-compatible mountings on the back, courtesy of the Smart Pi Touch case (available from ModMyPi for £22.99).This project is enabled by the official display kit for the Raspberry Pi.

There have been a few display options for the Pi: three-line LCD displays and small, low-resolution panels have been around for almost as long as the Pi itself, but these are generally not suitable for normal desktop activities. Larger displays eventually became available, but these required complicated soldering on to driver boards, and often required lots of effort to make them work with Linux. This early generation of displays also used lots of GPIO pins, so that if you wanted to use them and a display, then you were out of luck.

Seeing the need, the Raspberry Pi Foundation designed a marvellous seven-inch 800x480 display that's easy enough to put together and, thanks to the DSI Ribbon connection, doesn't need to use any GPIO pins on the Pi. The Pi mounts nicely on the back of the display, and if you don't want to leech power via the Pi's 5V pins, then the display driver board can instead draw power via its own micro-USB connection.

We managed to knock together this wooden Pi case in about three hours (including print time). It's a bit rough around the edges, mind.

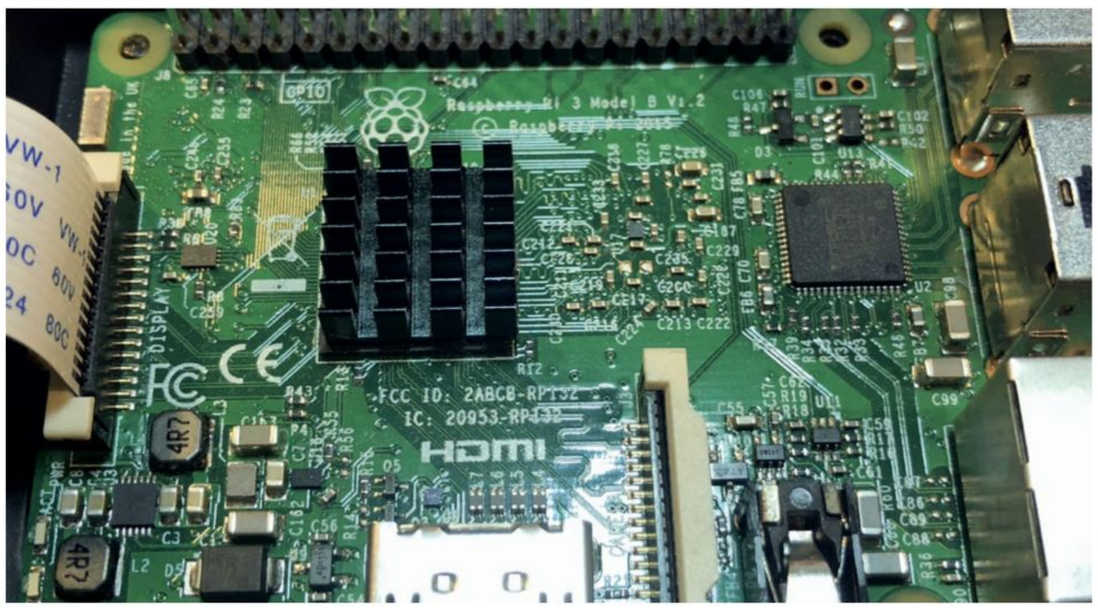
TURN YOUR Pi INTO A TABLET WITH THE SMARTPI TOUCH CASE

1 Use protection.

Affix the transparent rubber feet to the base. Stick the two square foam pads on the Pi trapdoor and near the case corner (where the GPIO side of the Pi will rest). Your display may come with a short ribbon cable, in which case you should replace this with the longer one provided.

2 Secure the base

Fix the case to the base with the generously provided tool. Don't overtighten the bolts because then your hinge will lose its vigour. Place the touchscreen carefully into the case and feed the ribbon cable through the aperture. Secure it in place with the four dark-coloured screws.



3 Connect the DSI ribbon

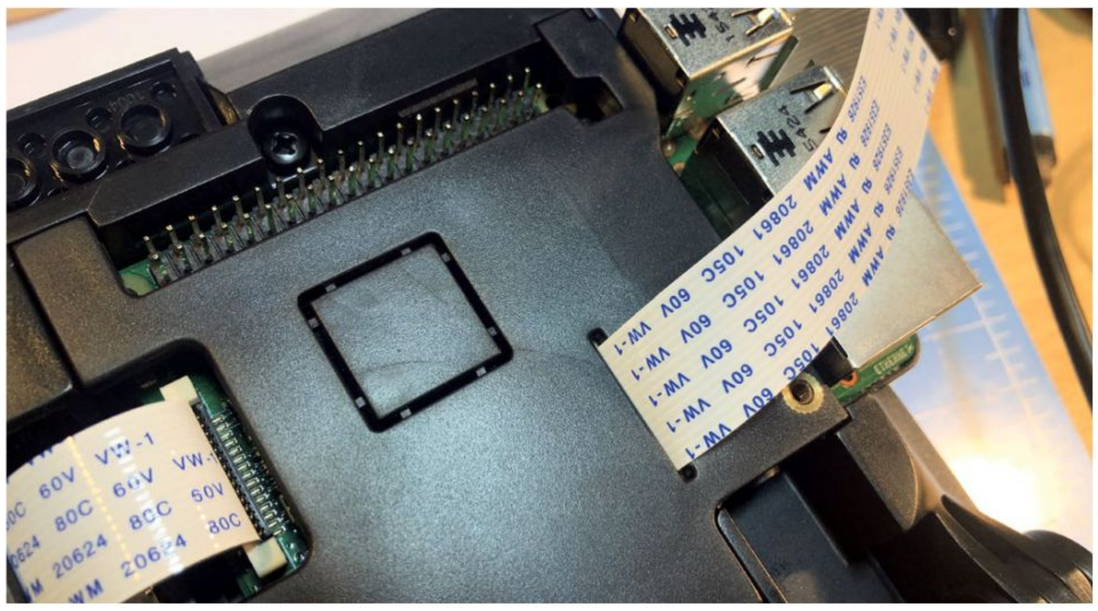
Connect the ribbon connector to the Pi and place it in the cavity. It can be held in place with one or two of the small, silver screws. If you use that second screw by the USB ports then you won't be able to use the trapdoor, which otherwise secures that side of the Pi in place.

4 Connect the power

Use the dual-micro USB adapter to connect both the display's and the Pi's power supplies. It's especially important to have sufficient power to supply both of these appliances. A PC's USB port can typically deliver 500mA, which explains the thunderbolt in the next picture.

5 Enjoy the view

Cross your fingers and plug in the power. An up-to-date Raspbian will have all the necessary drivers and the display should fire into life. Note the power warning in the corner. Also note that to change the micro SD card while the Pi is mounted, you'll need to unscrew it and flip it over.



6 Take pictures

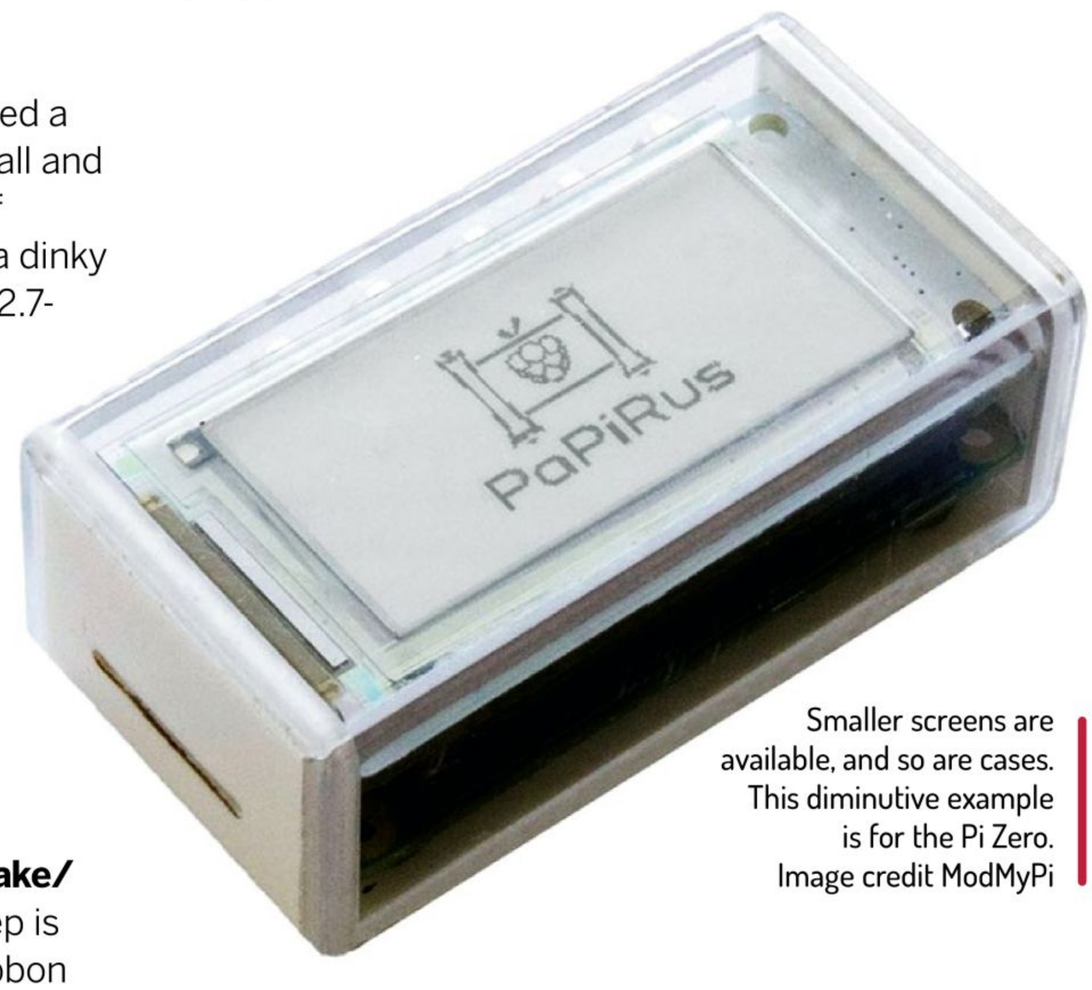
The Smart Pi Touch case includes a LEGO compatible case for the official Pi Cam. Feed the ribbon cable through the slot. Affix the top and clip the case on to the side or on top. Note that connecting the ribbon through the trapdoor is difficult and so you may prefer to remove it.

PaPiRus e-Ink display

Use a low-power display to read books, display graphics and more...

Pi Supply (**https://uk.pi-supply.com**) has designed a range of cheap e-ink screens that are easy to install and have the puntastic moniker PaPiRus. A number of different screen sizes are available, ranging from a dinky 1.44-inch (suitable for Pi Zero projects) to a large 2.7-inch (264x176). It's worth noting that the 2.7-inch model overshoots the edge of the HAT board, and hence the edge of the Pi by a few mm. So the whole shebang won't fit in a standard case. It will sit on top of a Pibow Coupé case or other open case, though. All models are equipped with a battery-backed RTC (real time clock). There are also four input buttons that sit along the edge of the board.

Constructing the kit might be a tiny bit fiddly, depending on how fat your fingers are. For full instructions see **https://learn.pi-supply.com/make/papirus-assembly-tips-and-gotchas**. The first step is easy: connect the screen to the board with the ribbon cable. Just make sure all the connectors line up nicely. The next step is not so easy, but is also optional. It involves soldering a springy pin, known as a pogo pin, on to one of two positions on the board (depending on which model of Pi you're using, the RUN pin on the 3 B has moved to the other side of the board, so use CN15 in this case, and CN13 for all other Pis). If you don't do this, you won't be able to use the wake-on-alarm facility provided by the RTC, but all e-ink-related functions will work as normal. Next, push the buttons onto the board and solder their feet from the reverse. They'll work if you don't solder them, but they'll rattle around and possibly jump off. They are small enough that you could easily lose them forever in this event. Fix the HAT and screen on to the GPIO pins, place the standoffs in between the board and the Pi, and screw them in place.



Smaller screens are available, and so are cases. This diminutive example is for the Pi Zero. Image credit ModMyPi

A number of example Python scripts are included. This one uses the buttons to navigate a menu system.

It's software time

Now we need to install the high-level software. Full instructions can be found at **https://github.com/PiSupply/PaPiRus** you can choose between an easy `curl -sSL https://pisupp.ly/papiruscode | sudo bash` one liner, or you can do things manually. As of press time there's an issue installing the `dateutil` python module, so if you see errors you can rectify them with `$ sudo pip3 install dateutil` and then rerun the set up script. If the SPI and I2C buses weren't enabled already, you'll need to reboot your Pi before you can start playing with it.

Run `sudo papirus-config` and set the appropriate screen size before you do anything else. These screens are quite fragile, but they also behave like they're broken (displaying an odd barcode pattern) if you don't set the dimensions correctly. Now quit the configuration utility and run `sudo papirus-test`. If you see errors about missing **/dev/epd/version**, then the paper driver hasn't loaded. Aside from broken hardware, this could be because the HAT isn't sitting flush on the GPIO pins, which is an easy fix. It could also be because of insufficient power. You can try reloading the driver with `sudo systemctl start epd-fuse`. Hopefully you figure it out. When you do you'll find a whole bunch more examples by typing `papirus-` and pressing Tab. Using handy Python APIs, PaPiRus can display text, graphics, or a combination of both.

» CONSTRUCT AN EREADER

Amazon's Kindle range may have cornered the eReader market, offering devices with lengthy battery lives and the magical ability to beam titles from their giant library in the clouds, but if you don't want to buy into that proprietary ecosystem, you can make your own.

There are a huge number of DRM-free books online. Project Gutenberg hosts thousands of classic titles that are out of copyright. There are also licenced works (for example by Cory Doctorow – *see interview* **LXF222**) that are redistributed there with permission. DRM-free ebooks can also be bought from **humblebundle.com**. Getting PDFs to display on the e-ink screen is likely to be a major effort, but the Epub format is widely used, and Project Gutenberg enables you to download books as plain text too, which will be much easier to parse.

IqAudiO Hi-Fi HAT

Listen to high-definition audio on your Pi, and breathe smart life into your old, dumb hi-fi.

The Pi's multimedia capabilities were pretty impressive when it was released, particularly the Videocore engine and HDMI port. But one area where it's always been sorely lacking is the quality of the analogue audio output. This is a shame as your Pi could otherwise be used to turn your dusty old hi-fi into a full-blown smart speaker system.

The nitty gritty runs something like this (and we encourage audiophiles to write in and correct us if we stray too far from the truth here). The analog signal is generated from digital signals, and those are limited by the PWM (pulse-width modulation) clock to a resolution close to 11-bit (read more about PWM in the box, *below*). This is fine for beeps and bops, but plug it into your fancy amplifier and things will sound hollow, lacking in range. There headphone jack will also pick up noise and static from other components.

Sounds good

To our rescue, then, come a variety of DAC (digital analogue converter) HAT boards that enable faithful audio reproduction for not much money. We tested the IQAudiO DAC+ HAT, which makes glorious 24-bit 192kHz audio reproduction possible. It takes the digital audio signal from the Pi via the I2S protocol and delivers it to its own high-fidelity DAC. There are other manufacturers too, such as HiFi Berry and Allo, which offer a separate reclocking unit to circumvent oddities from resampling audio signals. Some HATs even feature a built-in amplifier, so you can make a tiny 35 watts per channel boombox. We preferred the idea of using our quality 90s amplifiers though, and found IQAudiO's offering produced a decent sound. We also found out that our neighbours don't like psytrance.

Turning your Pi into a smart audio hub is easy with the Volumio distribution. It's available for PCs, Pis and other boards; grab it from **https://volumio.org/get-started**. Once you've flashed the SD Card, fire up your Pi and connect it to the hotspot it sets up from another device. If you're Pi doesn't have wireless (or has an unsupported wireless chip), then use a cable to connect it to your router.

VolumeIO has partnered with Allo to make the Nanosound player, which is made of a Pi and one of Allo's high-grade DACs fitted with a custom OLED display. It's all housed in a nice box (available in a variety of colours) with playback and power buttons. It also comes with a remote control, which can even turn the Pi on and off thanks to wake-up circuitry in the DAC board.

Besides VolumeIO, you could just as well use MPD to serve your music. The *MPDroid* program enables this to be controlled from Android Phones, and a huge number of clients are available for Linux. *Mopidy* (**www.mopidy.com**) combines MPD with a web interface and is another popular choice turning Pis into digital jukeboxes. Better still, extensions make it possible for *Mopidy* to connect to services such as Spotify, Soundcloud and Google Play Music.

Make Pi Audio great again – just attach the HAT, connect to your amplifier with phono (or jack) cables to your amp and enjoy luscious sound.

Our favourite MPD client is the catchily titled ncmpcpp. Music sounds better from the terminal.

» PULSE-WIDTH MODULATION

Pulse-width modulation is a method of encoding information as digital signals. Using the frequency of the clock generator we can encode data based on how much time the signal is low, compared to how it is high. If we send a sequence that sends a 1 then a 0 (for the same time period) and repeats, then that looks like a value of one half. This is known as a 50 per cent duty cycle. By changing the duty cycle we can achieve a variety of in-between values, so we can approximate an analogue signal. This method is used to dim LEDs, to adjust the position of a servo or any number of other things.

PWM can also be used to mimic an analogue signal, for example having an output decrease while the PWM signal is low and increase while it is high. Going back to the Pi example, if we want to play back a 48kHz audio file then the number of different values between high and low we can make, or the number of different shapes we can encode in a single sample works out at 2,267 point something. This is just over two to the power of eleven, so our PWM DAC (assuming it's clocked at the stock 100MHz) gives us, in strictly lay terms, just over 11-bit audio.

Audio hardware has been improved with each Pi generation, in particular the high- and low-pass filtering arrangements. The PWM driver has also been revamped in successive Raspbian releases, most notably in October 2017 which saw output treated to a complex three-stage oversampling procedure followed by a run through a noise-shaping filter.

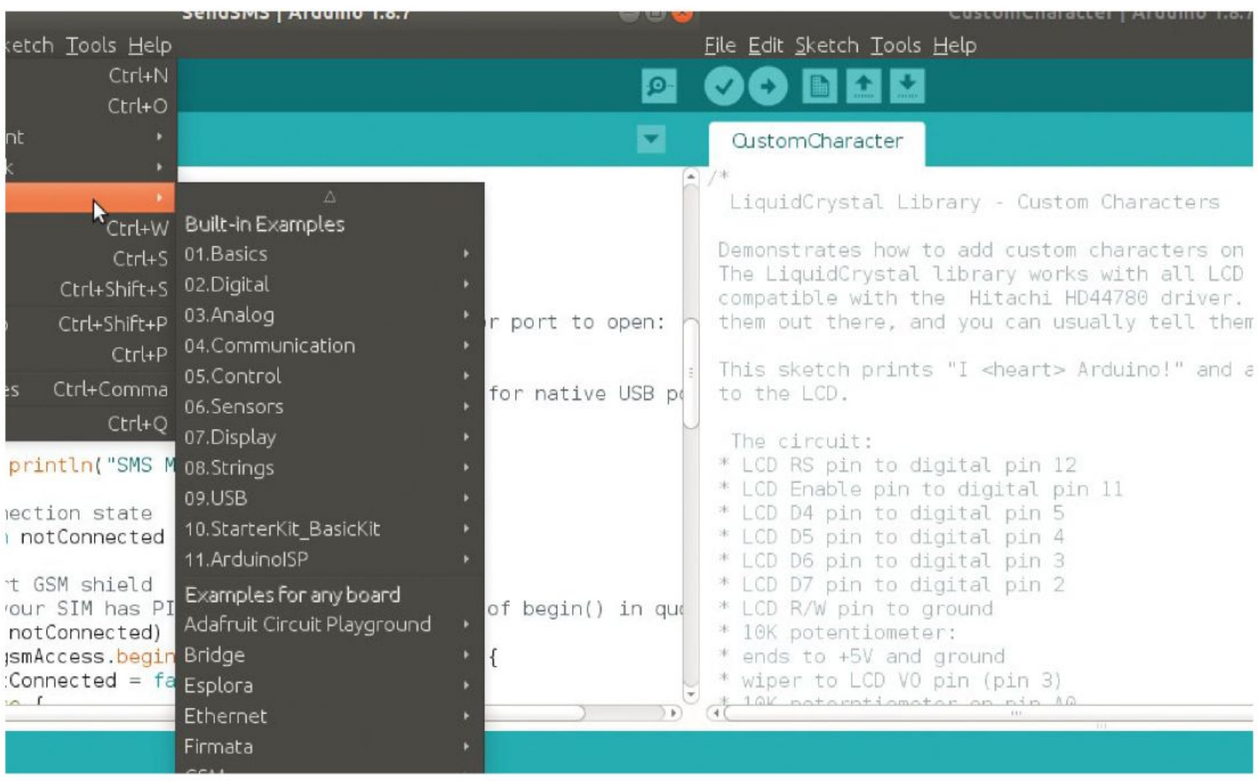
Get started with Arduino

Learn the Arduino language and see what this tiny microcontroller can do.

Occasionally we get chagrinned readers writing in with complaints about too much Raspberry Pi content. So here's something different (*hold on to your hats – Ed*).

Unlike the Raspberry Pi, Arduino devices aren't potent enough to run a full-blown Linux distribution. For example, the popular Arduino Uno is powered by a 16MHz chip and has a total of 32KB of flash memory and only 2KB of SRAM (where your programs, known as sketches, get uploaded). However, they do have a lot of input and output pins, so their main utility is as microcontrollers: taking sensor readings and switching on LEDs, spinning motors or any number of other things in response. More powerful Arduinos exist, but even those rely on other devices for heavy computation, networking and storage.

You can download the Arduino IDE from **https://arduino.cc**, or if you're using Ubuntu you can do it via *Ubuntu-Make*, which will fetch the latest version. Do `sudo apt install ubuntu-make` then `umake ide arduino`. You may need to add your user to the dialout group in order to communicate with the Arduino (*Ubuntu-Make* does this for you, but you'll still need to log out and log back in again for this to take effect). If you're feeling avant garde, you can also use the web editor from any modern browser on any platform by registering at **https://create.arduino.cc/editor**.



There are heaps of categorised examples for you to discover in the IDE. Just head to File>Examples and fill your boots.

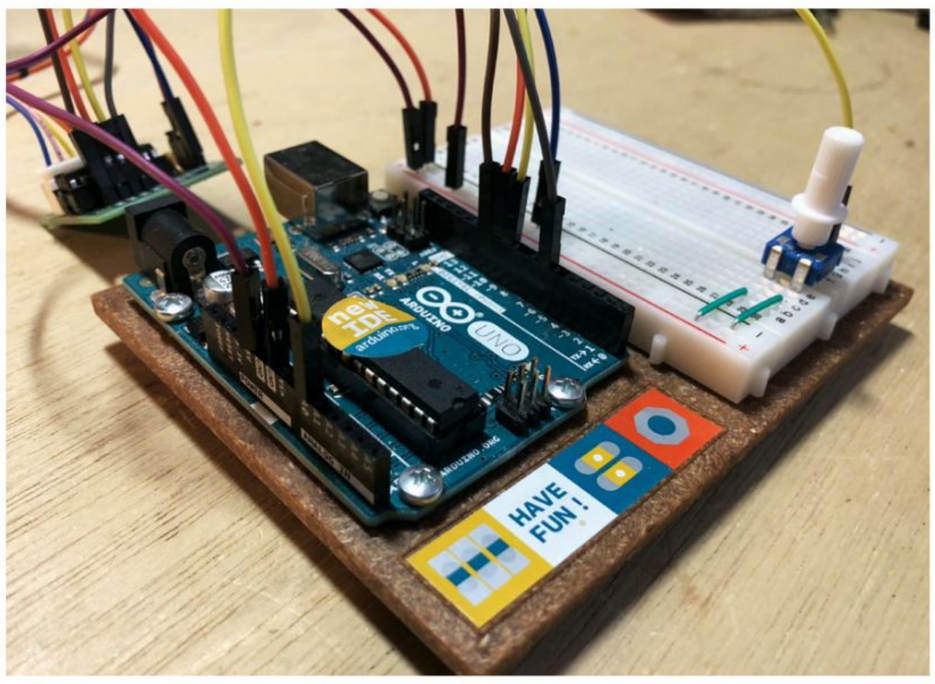
Don't blink

However you do it, let's follow the traditional pathway and load the Blink example. Go to File>Examples> Basics Blink and have a look at the code. We can see it's divided into two functions: `setup()` and `loop()`. In the first we define the LED as an output. It's hard to see how this could be any other way, but we still need to specify such things. LED_BUILTIN is a constant corresponding to pin number to which the LED is connected (usually 13). We need to specify a mode before we can use any pin. Moving on to the loop function, we can see this sets our LED voltage high, waits for one second, sets it low again, waits another second, and continues in this manner indefinitely. Click the Upload button in the toolbar and see that this is indeed what happens.

If we want to read values from our Arduino program on the host machine, we can do so over the serial connection. We need to set the serial connection up by adding the line `Serial.begin(9600);` to the `setup()` function. This means we can send a whopping 9,600 bits per second over the USB cable, which is the same speed as modems of the early 90s sent data over phone lines. Suppose we want to verify the statement above about the LED being connected to pin 13. Then all we need to do is add the following line:

```
Serial.println(LED_BUILTIN);
```

to the `loop()` function. That's a lowercase 'L' in `println` (short for print LiNe) by the way. Now when we upload the sketch the LED will blink as before, but if we go to



Even simple projects with buzzers and switches can look messy, but trust us when we say you'll get used to this.

» TALKING PYTHON TO YOUR ARDUINO

Sometimes it's good to talk to the Arduino from Python, rather than trying to do it all in the IDE. Having a Raspberry Pi work in tandem with an Arduino is useful. We still need the IDE to upload sketches to the Arduino, but we can do more with the data from the Python side.

All we need to do is connect the two boards with a USB cable. We communicate via serial link over USB. On the Python side we need the serial module (installed by default) and the device name of our serial link (check the output of `ls /dev/tty*`). Set things up with:

```
import serial
port = "/dev/ttyACM0"
s1 = serial.Serial(port,9600)
s1.flushInput()
```

We can then read from and write to the Arduino using the object `s1`. The following reads a single byte and then writes a 1. Changes in Python 3 mean we have to coerce the string to a byte:

```
s1.read(1)
s1.write(b'1')
```

Tools>Serial Monitor we can see the value 13 being spat out once every two seconds. If you have an exotic Arduino you may see a different number here.

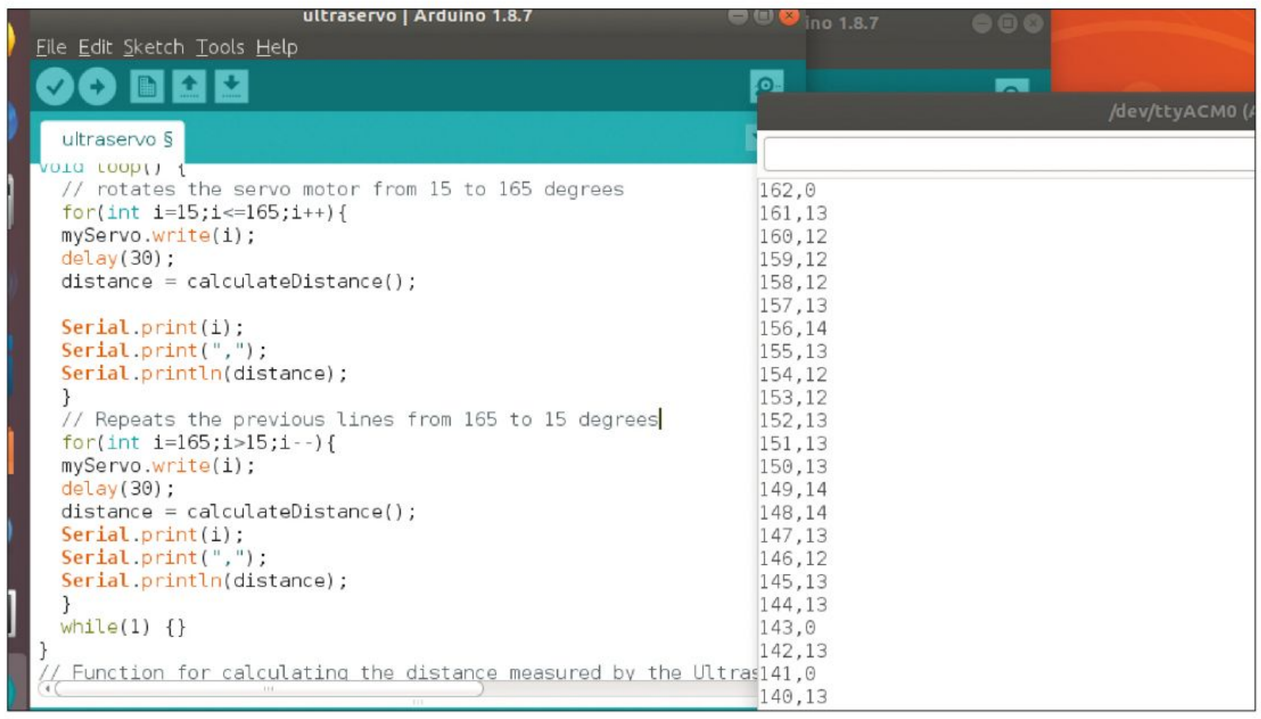
Sensor on a servo

For this project we're using the breadboard just to distribute power. Our servo (Towerpro MG699R) is rated at 500-900mA while moving, which is a fair chunk of power. Its stall current (when it's operating at maximum torque, i.e. its movement is being opposed) is a tremendous 6A, but we won't be doing any stalling. Again though, it's important your Arduino has adequate power and for this project you should connect a PSU to the barrel connector rather than powering by USB. The signal pin on our servo connects to pin 9 on the Arduino

The HC-SR04 ultrasonic sensor is a fantastic piece of kit. Send a pulse on its trigger line, and it emits a high-frequency squeak. If this squeak bounces off a nearby object, then when its echo returns to the sensor the echo pin will sent a 5V pulse lasting as long as it took between sending and receiving the ultrasonic pulse. This timing, combined with the speed of sound (around 330 m/s in air) enables us to determine the distance between the sensor and the object it just squeaked at. This is very much like the echolocation used by bats, except bats work on a much more complicated range of voltages and are furry.

By combining the servo and the ultrasonic sensor, we can do some quite fantastic things. The following code, for example, will sweep the sensor left and right. This makes for a kind of radar contraption:

```
const int trigPin = 10;
const int echoPin = 11;
Servo myServo;
void setup() {
  Serial.begin(9600)
  myservo.attach(9);
}
void loop() {
  for(int i=15;i<=165;i++) {
    myservo.write(i);
    serial.print(i);
    delay(30);
  }
}
```

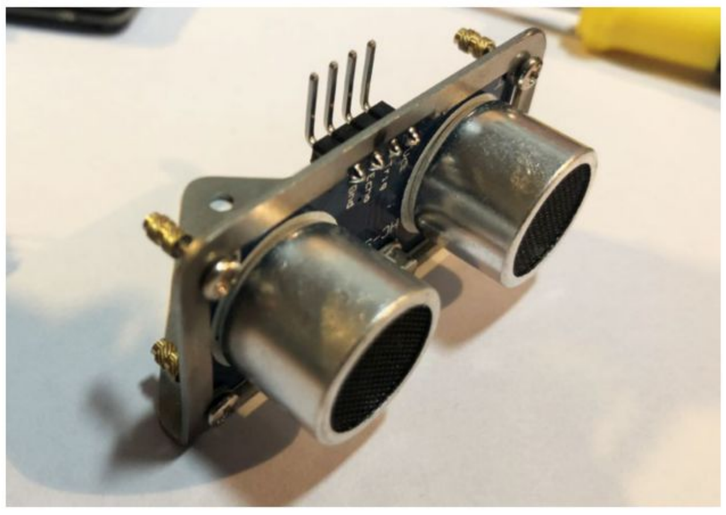


It works – our sensor is sending back a stream of data about nearby objects at various bearings. Whoop!

We use the serial link to get the rotation of the servo and again this can be viewed from the Serial Monitor. From here we can add some code to get data from the sensor information and instead send this, or better yet the distance calculated therefrom as well. We'll leave this as an exercise, or just check the link below.

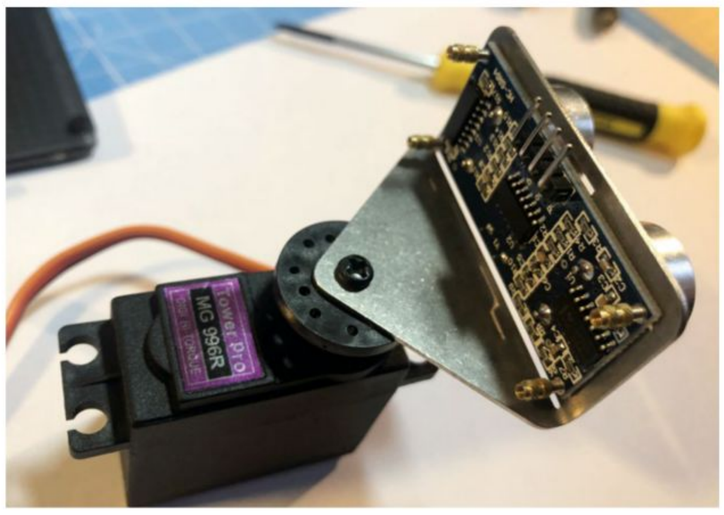
There isn't space to go any further with this, but the possibilities are endless. We were particularly inspired by the Object Detector project described at **http://bit.ly/object-detector**. There, an authentic green radar display is rendered using the Processing language. The drawing part is pretty involved, and features a nice blurring effect as the sensor sweeps left and right, but the underlying hardware is just the same. It's a great example of gathering data on the Arduino, and then processing and visualising it on more capable hardware.

BUILDING THE SERVO-SENSOR FOR THE ECHO DETECTION PROJECT



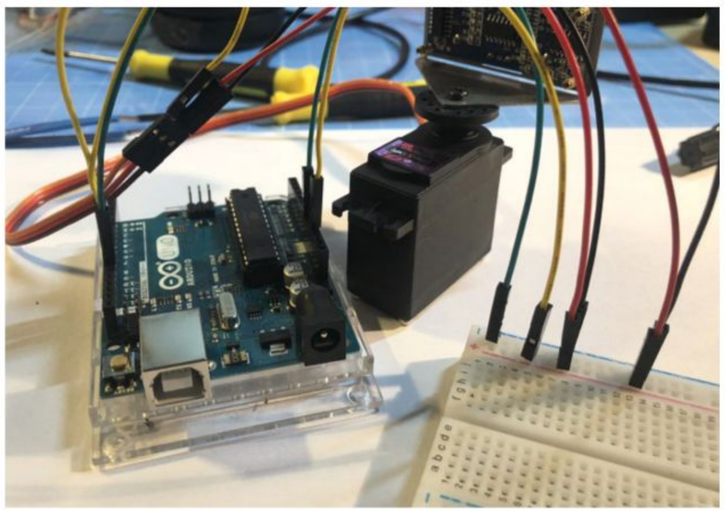
1 Mount the sensor

Carefully bend the bracket to form a 90 degree angle. Try to apply the force evenly so that the bottom edge doesn't twist. You'll be lucky if you get away with bending this back again. Screw in the ultrasonic sensor with the connectors facing upwards, and secure with the four small bolts provided.



2 Fit to servo

Attach the circular fixture to the top of the servo and then screw the mounted sensor in place. Don't worry about it facing straight ahead here. The servo may not be at its resting position and it's relatively straightforward to align it from the Arduino IDE and then adjust the sensor to look dead ahead.



3 Connect the cables

Attach the cable to the sensor and connect the jumper wires to the breadboard as described above. The darkest of the three servo wires (usually brown) is the negative, the middle one (red) is +5V and the last (yellow) is the signal. Besides power, the sensor has trigger and echo pins (connected to pins 10 and 11).

Further making adventures

Let's finish things off with a collection of suggestions for further projects.

Having a portable power supply for your Pi or other devices is a great idea, and there are a few options. Of course, a standard USB powerbank (and they can store a whole lotta Coulombs these days) is fine for many applications, but there are alternatives. If you have a lot of batteries lying around check out the ModMyPi's BattBorg. This enables you to power your Pi with four AA batteries. For more demanding situations eight AA or 9V battery attachments are available. The BattBorg defends against voltage drops, regulating power as batteries become weaker and refusing to turn it on if they're too weak.

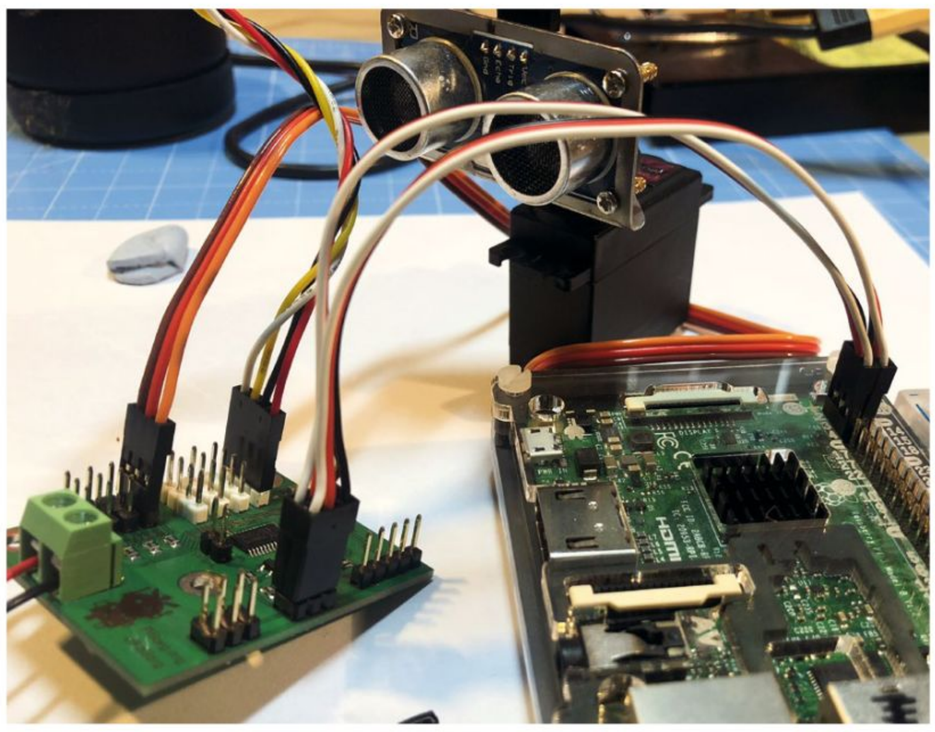
The Ultrasonic sensor we used earlier can also be used with the Pi, but we need to be careful because the Pi's GPIO pins don't take kindly to the 5V pulses from its Echo pin. We need to drop these to under 3.3V to be safe. We can do this with a potential divider: two resistors placed in series with the GPIO pin in between them. You can choose the value of one resistor and then work out the value the second needs to be. In this case a 1K and a 2K resistor will work. Alternatively, check out the UltraBorg board, which makes it possible to connect up to four servos and four ultrasonic sensors straight to the i2C bus of your Pi (SDA and SCL). It can use a separate battery supply which is a good idea if you connect four servos to it. The jumper should be removed if you do this, otherwise you risk damaging your Pi

BattBorg has three components on the PCB and is available as a soldered or an unsoldered kit.

PiJuice is a crowdfunded battery hat for your Raspberry Pi. It comes with a 1,820mAh lithium ion (Li-Ion) battery that's good for up to 5 hours of use, but this can be replaced with a lithium polymer (Li-Po) battery. Further, you can attach a solar panel (a large 40W one is available from ModMyPi and Pi Supply for around £116) to charge the battery, making for a completely isolated solution (unless the sun runs out). Like the PaPiRus e-Ink display it comes with the added bonus of a real-time clock, which will be useful for keeping time while offline. PiJuice's software enables battery levels to be read, and a shutdown script can be set to run when they get low. This makes for a delightfully simple UPS (uninterruptible power supply) solution.

The PiJuice battery pack has an RTC and makes your Pi truly portable. You can even add a solder panel to charge it.

If you can find a small-enough power supply (or a big-enough jar), then connecting it and a camera to a Pi Zero, and then sealing the whole arrangement in a jar, can make for some nice nature photography. The No-IR edition of the camera (which has no IR filter) can even see at night time, which makes it great for capturing badgers, owls (if you're lucky) or other nocturnal



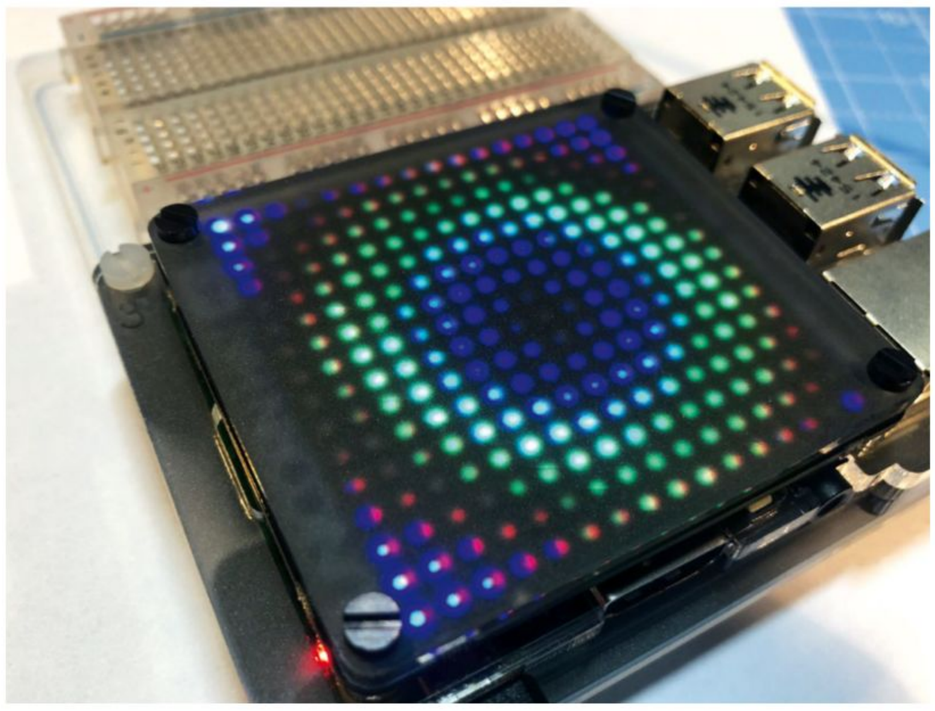
Ultraborg solves several sensor-related problems at once. You can even daisychain multiple Ultraborgs together.

» POWER CONSIDERATIONS

Thanks to most smartphones in the world standardising on micro-USB connections (except those made by a certain fruity company), it's easy to pick up a power supply for your Pi. It's powered by the same thing after all. However, one should be careful: many of the cheaper phone chargers aren't capable of providing the voltage required to reliably power the Pi.

The same thing is true for the USB ports on your computer. This is especially true for the Pi 3s (and newer 3 B+s), whose faster SoCs and wireless chips need extra voltage. Many unsuspecting users of these devices have been traumatised by the appearance of a lightning bolt in the top right of the screen. This signifies a low voltage and that the Pi has throttled its CPU to keep things stable.

The underpower problem is further exacerbated when you start piggybacking other devices (such as motors) onto the 5V and 3.3V power pins. With a good-quality power supply this is fine; the official touchscreen will draw around 1A, and the Pi less than that. We heartily recommend using the official Raspberry Pi Foundation PSU, rated at 2.5A, or some other quality unit. If you need to power a number of devices, hubs that can supply up to 3A per port are also available.



The Unicorn HAT-HD includes some classic effects that hearken back to the 90s demo era. Ah, those halcyon days of computing...

species. Making a rudimentary motion detection program in Python is straightforward, so you don't need to worry about filling up your SD card with the same photo of the eerie, still darkness.

Have we got to 10 yet? We don't know, but we're going to keep going because there's no end to the fun (*and because you have to fill this page – Ed*).

Unicorn HAT-HD

Pimoroni's Unicorn HAT's, which featured an 8x8 matrix of RGB LEDs, proved tremendously popular. So they had to go one better and make an HD edition that quadruples the LED count. It also comes with a nice diffuser whose height can be adjusted to make for different blinkenlight blurring effects.

A neat trick which we're still working on is getting this to display a spectrum analyser for currently playing music. *MPD* can easily be configured to write to a FIFO pipe by adding a section exactly like

```
audio_output {
   type "fifo"
   name "myfifo"
   path "/var/tmp/mpd.fifo"
   format "44100:16:1"
}
```

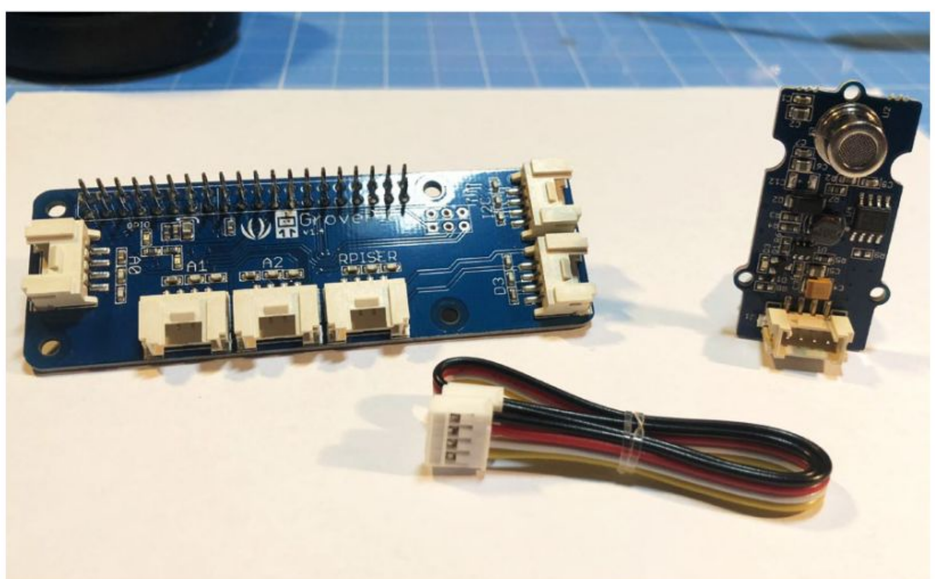
to `/etc/mpd.conf`. We can then access this buffer in Python thusly:

```
fifo = os.open('/var/tmp/mpd.fifo', os.O_RDONLY)
```

and perform the required FFTs to get spectrum data. Or you could use the audioop package to do it for you. A good starting point would be to make a simple VU (Volume Unit) meter, and we would get the peak level from the buffer with the following:

```
data = os.read(fifo, 4096)
stereoPeak = audioop.max(data,2)
```

One important thing to be aware of is that you can't use the 3.5mm headphone jack and the Unicorn HAT at the same time. This is because they both use PWM, so you'll either see glitches on the LEDs or hear interference in your audio. It works fine with HDMI and can also work on top of a DAC hat such as the IQAudio one we tested. Peak blinkenlights is achieved by the LightShowPi project (**http://lightshowpi.org**), which supports all manner of LED boards and can make them blink in all kinds of ways.



This GrovePi sensor measures air quality. It can detect carbon monoxide, formaldehyde and alcohol, a typical **LXF** evening out.

Relax with RetroPie

The RetroPie project (**https://retropie.org.uk**) includes a number of emulators, including SNES and Gameboy (more can be added later). It's available for the Pis as an SD Card image (the recommended approach) or it can be installed manually. Separate images are available for single (Pi 1 and Zero) and multicore (2 and 3) Pis. It's also available on Linux for PCs, which are more capable of emulating the newer platforms.

A big part of RetroPie is RetroArch, a front-end for the Libretro API (a cross-platform effort that enables different emulators ('cores') to be treated uniformly). Go to **https://launchpad.net/~libretro/+archive/ubuntu/stable** to see the huge number of Libretro cores available on Ubuntu. Many retro-styled gamepads are available to help your ring-collecting or mushroom-eating adventures – we favour the old PS2 style.

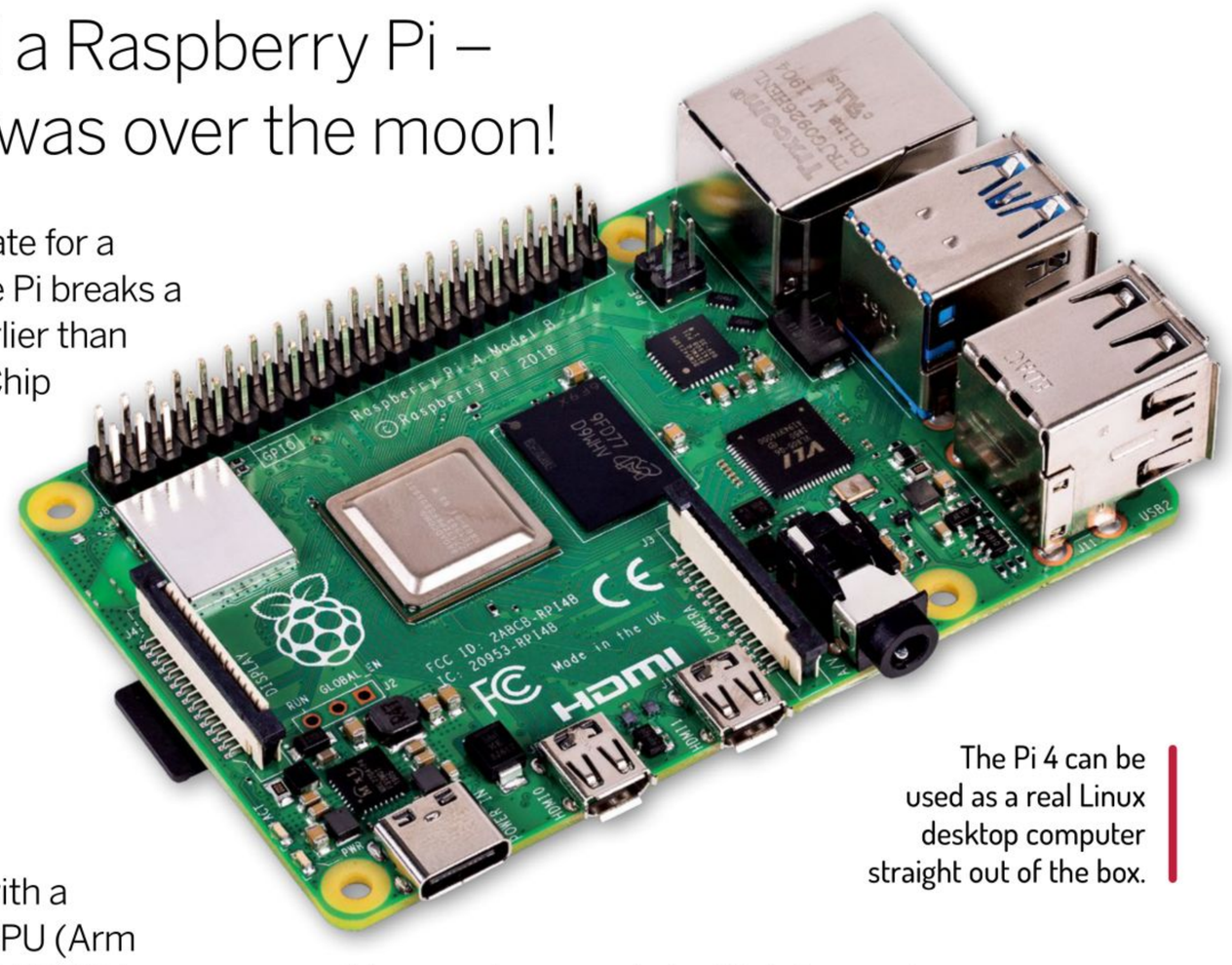
Gamepads and touchscreens and keyboards and rodents are all well and good, but sometimes you just want to control your Pi by shaking your fist at it. And now you can, courtesy of the Flick HAT gesture control interface. More accurately, you're supposed to use swipes, taps and flicks of the wrist, but this is still pretty cool.

And that's all we have space for in this month's maker extravaganza. Special thanks to Jacob and Claire at ModMyPi for generously lending us so many goodies to make this possible. Also thanks to local Bath maker Luke Clifford for letting us use his workshop, its many tools and for helping us 3D print our wooden Pi case.

A controller like this and the RetroPie distro make for many lost hours gaming.

Raspberry Pi 4 B

Les Pounder loves a surprise and a Raspberry Pi – so when the Pi 4 was released, he was over the moon!



The Pi 4 can be used as a real Linux desktop computer straight out of the box.

SPECS

CPU: 1.5GHz 64-bit quad-core Arm Cortex-A72 CPU (ARM v8, BCM2711B0)
RAM: 1GB LPDDR4 (2GB, 4GB options)
Comms: Wi-Fi dual-band 802.11 b/g/n/ac; Bluetooth 5.0; BLE
Ports: 2x USB 3.0; 2x USB 2.0; gigabit Ethernet (PoE support); 40-pin GPIO; 2x micro-HDMI (4Kp60); DSI display port; CSI camera port
Video: H.265 (4Kp60); H.264 (1080p60, 1080p30 encode); OpenGL ES 3.0
Audio: 3.5mm analogue audio; composite video jack
HDD: micro SD card
Power: USB-C
Size: 85 x 56 x 19.5mm, 46g

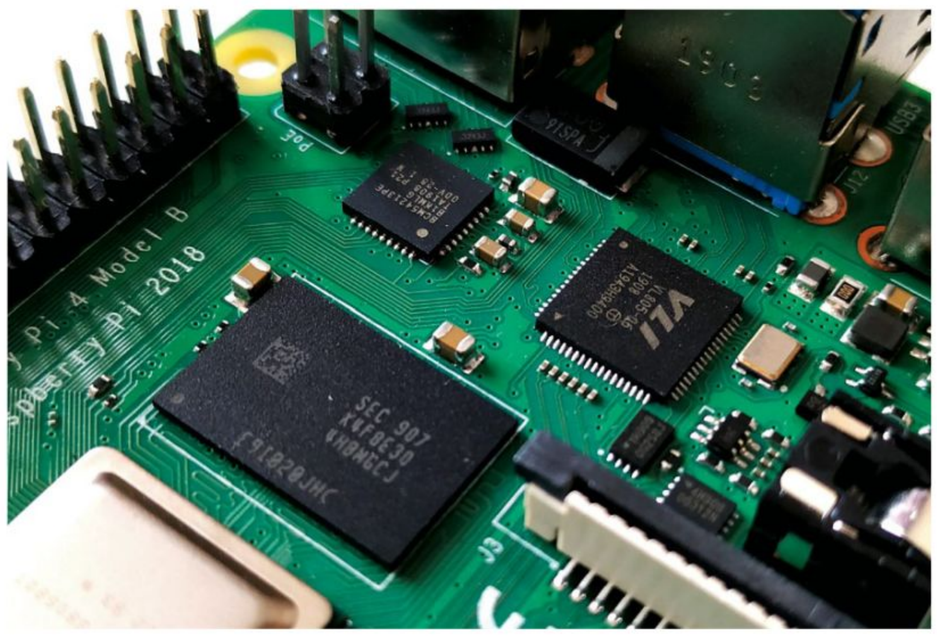
Surprise! June is not a traditional release date for a Raspberry Pi. In fact, this new version of the Pi breaks a few traditions. Firstly, it was released far earlier than predicted due to the BCM2711 System on Chip (SoC) being ready for production some nine to 12 months early. Secondly, there's the price: there is now not just one price for a Raspberry Pi Model B, but three. At £34, the basic model comes with 1GB of RAM; for £44 there is a 2GB model; and for £54 there is a 4GB model. For this review we have chosen to look at the 1GB model as we can directly compare it to previous models of Pi.

No matter the version of Pi 4, all come with a 1.5GHz 64-bit quad-core Arm Cortex-A72 CPU (Arm v8, BCM2711B0) and all of the RAM is now LPDDR4, giving us a much-needed boost in performance. GPU duties are handled by the new VideoCore VI which supports OpenGL ES3.0 and provides another handy boost in performance.

If you are a keen maker/hacker then fear not, as the 40-pin GPIO is present and fully compatible with HAT and pHAT boards. But in a change to previous GPIO configurations, the GPIO of the Pi4 offers four each of I2C, UART and SPI interfaces, enabling more compatible devices to be connected – all by changing the device tree overlay on boot.

For network connectivity, we are truly spoilt. While the Pi 4 retains the excellent 2.4GHz and 5GHz Wi-Fi from the 3B+, we finally have true gigabit Ethernet on a Raspberry Pi thanks to the BCM54213 Ethernet chip, which provided a stonking 931Mbits/sec in our test – around three times faster than the 3B+. No more Ethernet through USB!

Talking of USB, the Pi 4 has four USB ports – two of which are USB 2.0 and the others USB 3.0, which have their own chip that connects to the SoC via PCI-E! Yes, we can now connect high-speed USB drives to the Pi for projects such as media centres and fileservers. We can also boot from USB devices, so adding a cheap SATA SSD via a USB interface will massively speed up your Pi and provide very cheap and plentiful storage for servers.



The BCM54213 handles gigabit Ethernet on the Pi 4, and the VLI chip is used for USB. No shared LAN7514 chip any more.

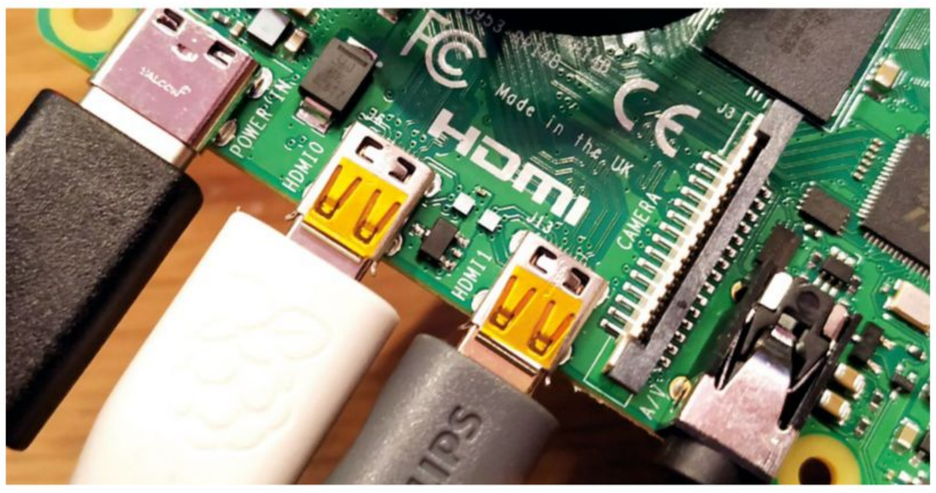
To power the Raspberry Pi 4 there is a USB-C connector in place of the old micro-USB port. The USB-C port provides 5V at 3A (15W), which at idle is approximately 4W and 8W when under load. Since launch a minor flaw has been discovered with the Pi's USB-C implementation, so it can't be powered by e-marked USB-C cables–such as those supplied with Apple laptops–this will require a board revision to fix.

While 15W may seem like overkill, it provides plenty of power for the Pi 4 and plenty of spare power for use by USB 3.0 hard drives. The current draw at idle for a Pi 4 is much greater than a Pi 3B+ – 600mA versus 400mA – and so that means a bigger battery is required. In fact, from cold the Pi 4 consumes around 960mA during the boot process.

If you plan to reuse an older official Raspberry Pi power supply (5V at 2.5A, 12.5W), you can purchase an adapter to change micro-USB to USB-C. Power can also be provided directly to the Pi via the GPIO, but this will bypass any regulators, so check your voltages before connecting. If you need Power over Ethernet (PoE), you'll be happy to know that the PoE HAT is compatible.

With all this new power there has to be heat, and this is where the Pi 4 differs to previous boards. To be clear, the Raspberry Pi 4 runs really warm at idle: around 52 degrees C two minutes after booting up. The temp increased to 66 degrees when under load during a Sysbench CPU stress-test. The board can reach temps of up to 80C and so this means that active cooling is a must. We ran the same idle and stress-test sequence on the Pi with a Pimoroni Fan Shim, and this dropped the idle temp to 36 degrees with the CPU stress-test temp at 46 degrees. We have been advised of an imminent firmware fix that will address these thermals.

Another new feature on the Pi 4 are the dual micro-HDMI ports offering dual screens at 4Kp30, or a single screen at up to 4Kp60. Video is also decoded by a dedicated chip, so there's no hit on CPU performance.



Featuring two micro-HDMI ports, the Pi 4 offers a true dual-display solution – but these ports are too tight for adapters.

But why is this necessary? Well, to be honest it isn't something that most users will take advantage of. But 4GB users who may be using their Pi 4 as a desktop replacement can enjoy an extra screen. Typically coders and writers employ two screens for reference and work.

To use the Raspberry Pi 4 we need to be running the very latest version of Raspbian; older versions will not work with the Pi 4 but the latest Raspbian will work on every model of Raspberry Pi. Based upon Debian Buster (unreleased at the time of writing), the latest Raspbian is impressive and nimble. It booted on our Pi 4 in just under 18 seconds and, after a few hiccups, detected our dual-screen setup, giving us 3840x1080 of screen for projects – lovely stuff!

The Raspbian desktop is very usable with Pi 4 and it feels just like a traditional desktop computer. We ran our usual Sysbench single- and quad-core CPU stress-test, and the Pi 4 shaved almost 30 seconds from the Pi 3B+ single-core test: 119.3 seconds for Pi 3B+, 92.7 seconds for Pi 4. In the quad-core tests the Pi 4 scored 23.3 seconds versus the Pi 3B+ time of 31.8 seconds. The Pi 4 also blazed through our Octane benchmarks with a score of 7274, 2.6 times quicker than Pi 3B+.

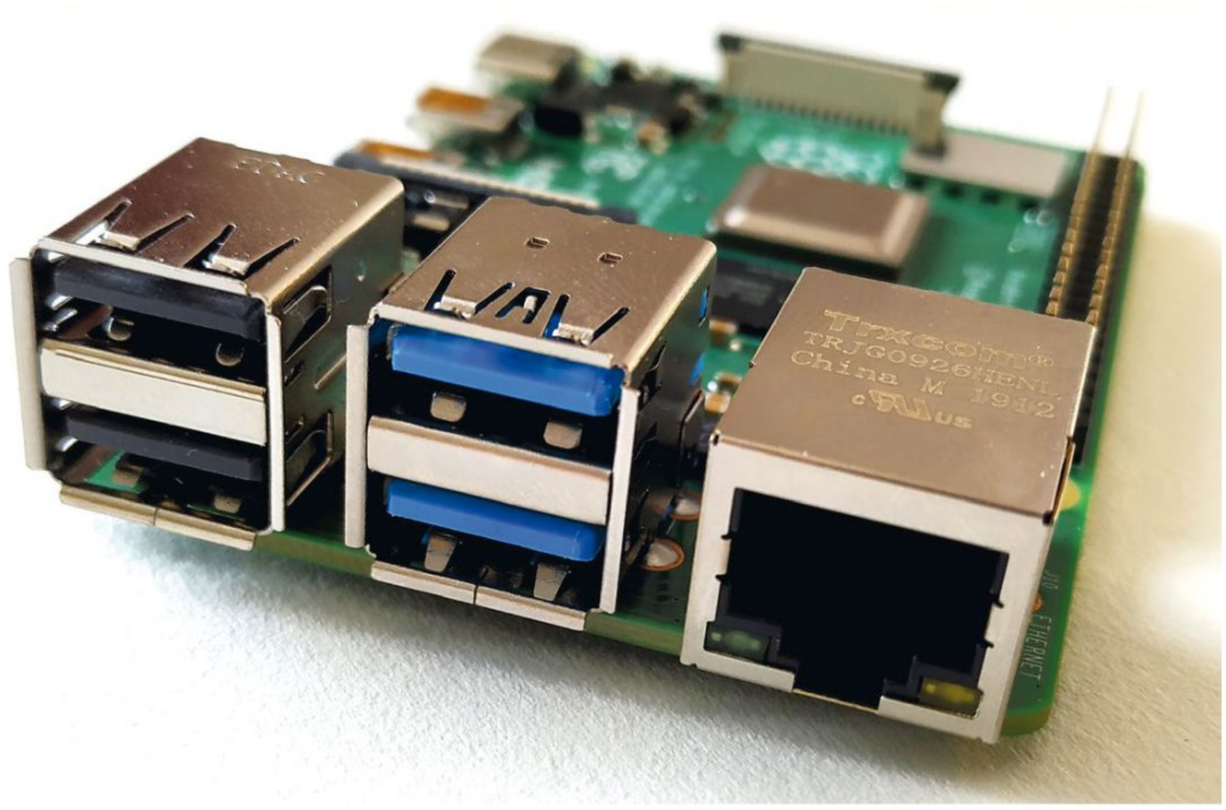
But where the 1GB version will struggle is with resource-heavy apps – *Chromium* is the main culprit. Watching a video at 720P in a window is bearable; make it full-screen and 1080P and it becomes a slideshow. There is a future update planned to address these launch issues, so we shouldn't have to wait long for a fix.

The questions that the Raspberry Pi 4 raises are "Who is this board for?" and "Why do we need this much power?" To be blunt, the Pi 4 does offer a lot of power, but most makers will not take full advantage of it. The Pi 4 offers entry-level desktop performance and for £34-£54 that should be applauded. But for most hobbyists who make GPIO-based projects, robots, or who are learning to code, then sticking to a Pi 3B+ and using the latest Raspbian release will offer all the latest Debian Buster features at no extra cost. Anyone looking to upgrade their Raspberry Pi would be wise to wait for a while until Debian Buster and the Pi 4 software gains a little maturity. Don't get us wrong – the team have done marvellous work with the Pi 4, but the software hiccups need to be ironed out before consumers get their hands on one.

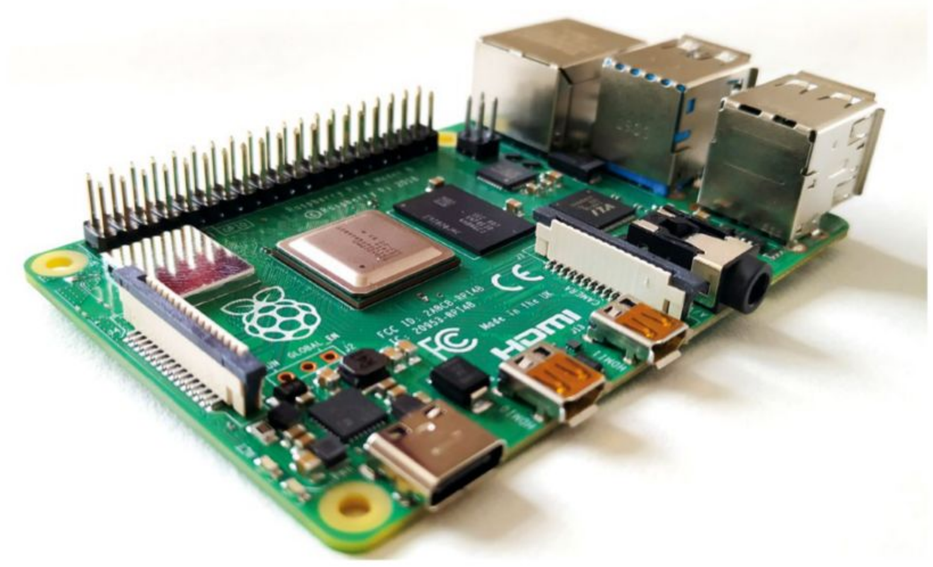
If you have been waiting to run your own low-power Linux home server, then the Pi 4 is the board of your dreams! There's lots of power, gigabit Ethernet, USB 3.0 and Debian stability. The other bonus for all users, no matter what version is purchased, is the immense library of support and documentation provided by the Raspberry Pi Foundation.

» ALL HAIL THE Pi 4

The Raspberry Pi 4 ushers in a new era in the history of the Pi. Coming seven years after the original release, we now see a Linux desktop computer for £55 that can be of real use in the home, office or classroom. Consuming 8W of power when under heavy load, and offering faster IO ports, this is an energy-efficient platform that we are sure many will take advantage of. Lightweight thin clients for offices, VOIP solutions and desktop computers for the home are where the Pi 4 is destined to be used – the power and size of the board is really remarkable. But is this heralding the year of the Linux desktop? No, and is the desktop still the battlefield that Linux even wants to win? Linux is key to the server and mobile industries and those battles are over. Rather, the Raspberry Pi 4 signals that a low-cost computer no longer needs to have caveats on how it performs.



The new USB and Ethernet ports are in different positions to earlier boards, breaking compatibility with older cases. Under the control of "Big Case".



It may look similar to those before it, but looks can be deceiving. This board packs real power!

VERDICT

DEVELOPER: Raspberry Pi Trading
WEB: www.raspberrypi.org
PRICE: £34 for 1GB (2GB £44, 4GB £54)

FEATURES	9/10	EASE OF USE	9/10
PERFORMANCE	8/10	VALUE	9/10

Truly powerful with great support, the Pi 4 offers so much, but it needs a little time to work out some hiccups.

» Rating 8/10

Les Pounder works with groups such as the Raspberry Pi Foundation to help boost people's maker skills.

HOT HOT HOT

June was quite the month! As everyone scrambled to get their latest slice of Raspberry Pi, retailers have been rushing to ensure that they have the latest add-on boards and equipment to support the Pi 4. It is the natural order for a Pi release – a new Pi means new add-ons to further hurt your wallet.

One of the most prominent issues with the Pi 4 has been the amount of heat it can generate, and so this has been the focus of the latest products. The new Pi 4 can idle at around 50°C and can be easily pushed up to 80°C with heavy loads!

Pimoroni has released two cooling products. The Fan Shim was released at the same time as the Pi 4 and has proven to drop temperatures dramatically, by as much as 20°C. This keeps the Pi 4 well under the 80°C thermal-throttling trigger point.

The second product is an aluminium heatsink case which offers a passive, silent cooling option and promises to drop temps just as much. If you need extreme cooling, the ICE Tower CPU cooling fan from seeed is a monster! In fact it looks like a Pentium 4 CPU cooler. The whole unit fits atop the Pi 4 with a custom frame to ensure that it's stable.

But the question is, are they compatible with previous models? And the answer is: yes and no. The Pi 3B+ and the Pi 4 share a very similar chip layout and so there is a high chance of compatibility, but anything older than that is a bit of a gamble, so do your research before spending the money!

The Raspberry Pi 4 full desktop kit

The Pi Foundation has launched a full desktop kit, so you can use your Pi 4 for day-to-day computing.

Despite the initial speed-bumps, there's no doubt that the Raspberry Pi 4 sets the standard for single board computers, and the inclusion of 4GB of onboard memory equips a Pi with the memory needed for a day-to-day desktop system – though we might have a few words about reliable and fast storage...

CREDIT: Raspberry Pi Foundation

Enter the Raspberry Pi 4 Desktop Kit, a box of fun designed to turn the bare-metal Pi 4 into a fully usable desktop computer. As well as a Pi 4, of course, the kit includes all the official add-ons you'd hope for: the Pi 4 case, Pi keyboard and mouse, official USB-C power supply, official beginner's guide, 16GB SD card with NOOBS installed, and not one but two (classic white) HDMI 1m cables.

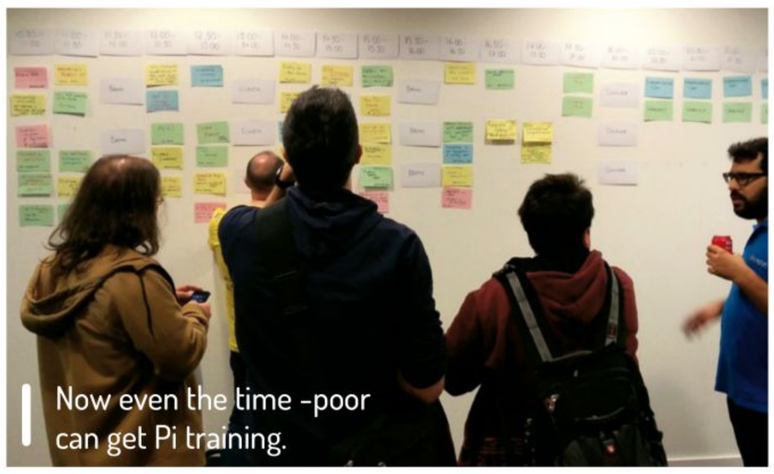
Costing £114 and with regional variants that include the correct power supply and keyboard – UK, US, EU and more – it sounds to us like an ideal Christmas present for any Linux geek in your life (*me?–Ed*). Check it out at **https://www.raspberrypi.org/products/raspberry-pi-4-desktop-kit**.

All your Raspberry Pi kit in one box.

Free physical

Training for all, that is.

Picademy Bytes is a new training scheme launched by the Pi Foundation. Think of it as a mini-Picademy for teachers who can't make the full two-day course. Taking less than 90 minutes, these promise to get everyone up to speed and the community-led scheme is running all around the country. Find the nearest one at **https://community.computingatschool.org.uk/events#**.

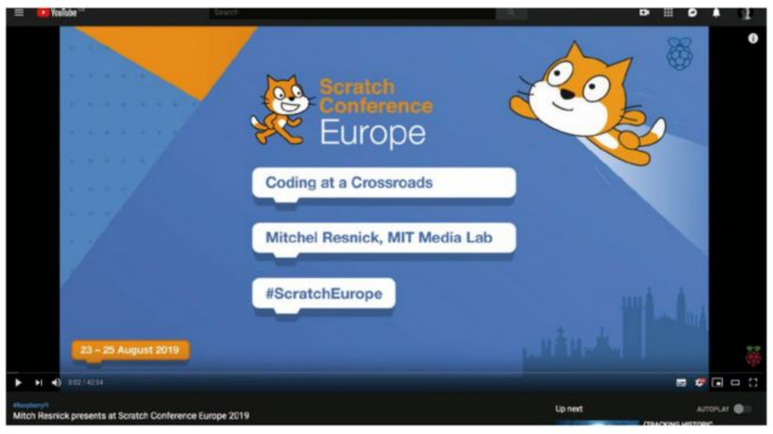


Now even the time -poor can get Pi training.

Scratching itches

Conference talks for all.

Everyone's favourite beginner language (that's Scratch, incidentally) had its yearly conference earlier in 2019 at the Pi's HQ in Cambridge. To help everyone benefit from the talks, the Pi Foundation recorded them all and stuck them up on its YouTube channel. There's a playlist already up, so sit back and have a good scratch at: **http://bit.ly/lxf255scratch**.



Hopefully full VR recordings for next year, right?

CREDIT: YouTube, Raspberry Pi Foundation

Raspbian Buster

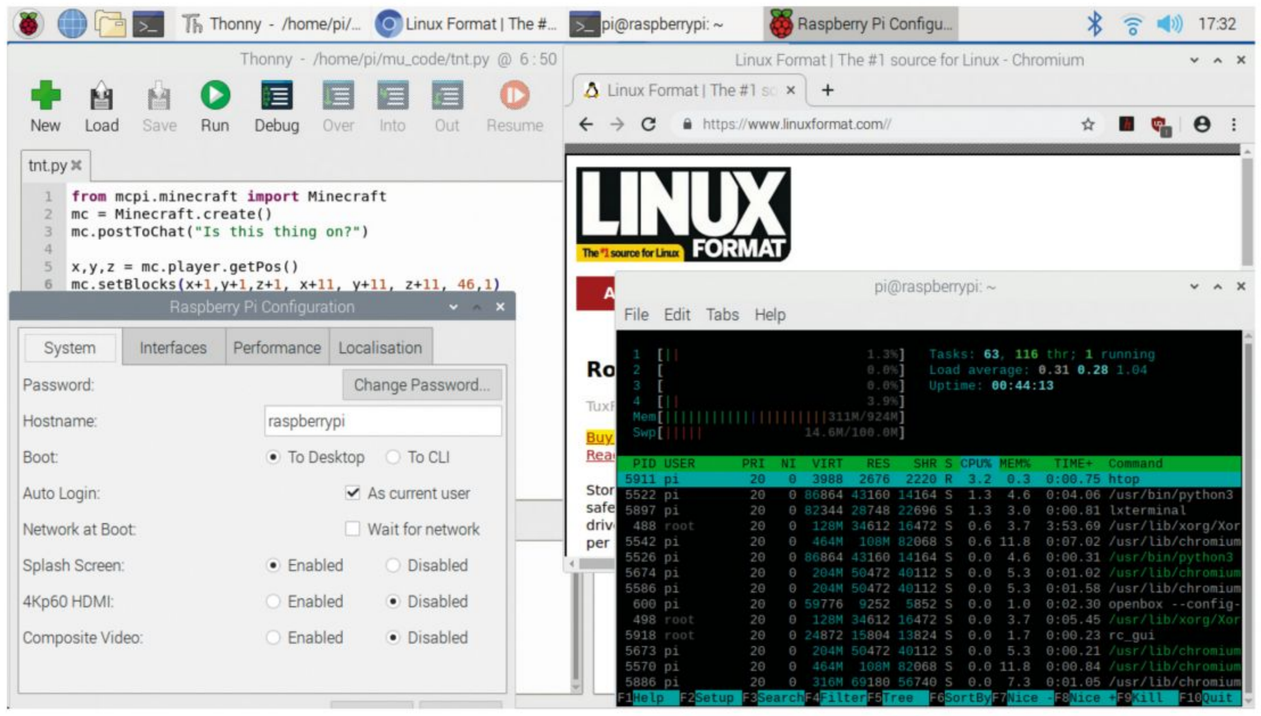
Les Pounder hasn't seen the latest Toy Story 4 movie as yet, but nevertheless he's already met Buster...

IN BRIEF

The latest Raspbian release from Raspberry Pi Trading. Designed for use with the Pi 4 but compatible with most of the older boards, this is a grown-up distribution that can take on most 'desktop PC' tasks.

With the release of a new Raspberry Pi there is always an update of Raspbian, and this time there is one big change. Released on 24 June, Raspbian Buster marks the start of new Raspbian images based on the latest Debian Buster image. In fact Raspbian Buster was released before the official Debian release.

Running Linux Kernel 4.19.50 created on 20 June 2019, Raspbian Buster is not much different to the last release of Raspbian Stretch. Most of the changes are to security and the backend. But Simon Long was keen to refresh Raspbian's interface and so it features a more refined desktop that feels even more like a 'desktop replacement' than before. If you're familiar with previous Raspbian releases then the basics are the same, but it just looks a little nicer.



Refreshed and ready for new challenges, Raspbian Buster offers a great user experience.

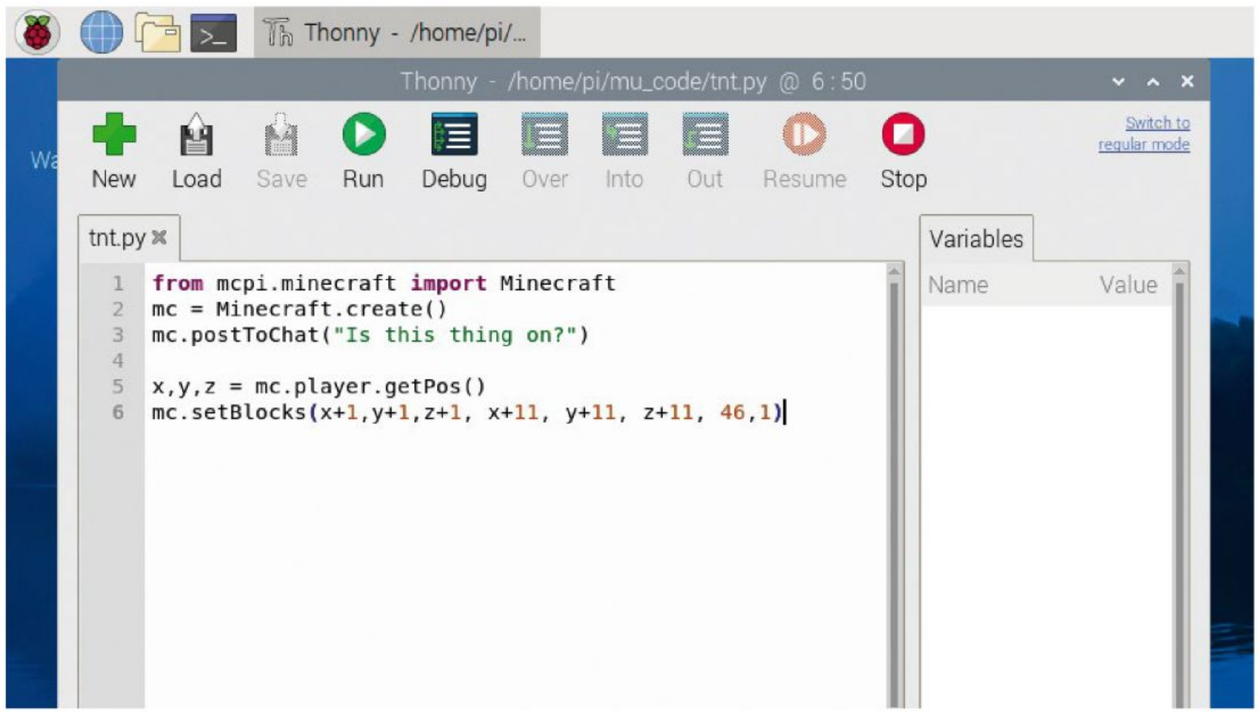
Software choices for Raspbian Buster are *Chromium 74*, which runs reasonably well – and on Pi 4 it runs remarkably well. YouTube playback on anything but a Pi 4 is choppy, and the installed version of *VLC* has issues with audio playback in videos. *LibreOffice 6.1* is installed and runs really well. For Python coders we see Python 3.7.3 installed as standard – it may not be the latest version, but it's recent and well supported. The ageing Python editor IDLE has not been retired from the Raspbian image and instead we see the great *Thonny* editor, an ideal Python editor for beginners. This is now the default editor for Python. There is also an alternative editor called *Mu*, aimed squarely at beginners, which offers compatibility with the CircuitPython, the micro:bit and the game creation library PyGame Zero.

Although Raspbian Buster is meant to work across all versions of Raspberry Pi, there are a few extra features for the Raspberry Pi 4. Raspi-config has an option to use a newer OpenGL driver which offers better performance, and the ability to alter the screen layout, as the Pi 4 supports two 4K screens. There is also a new compositor. If you are not running a Pi 4 these options are not visible nor configurable.

The Raspberry Pi Foundation's Raspbian distribution has always been backwards-compatible, all the way back to the original Pi in 2012. So we tested Raspbian Buster on ageing models of the Pi. If you want to run Raspbian Buster as a desktop OS then stick to Pi 2 and above – in fact, Pi 3 would be better. Anything older that Pi 2 is really only usable with Buster as a terminal-only OS. On the Pi Zero range of boards you can do basic work... slowly. But then the Pi Zero range was never meant to be a desktop device, more embedded into projects.

Is Raspbian Buster any good? Yes it is! We have a great desktop environment for users that will tackle whatever they throw at it. But as we mentioned earlier, this is a release for Pi 2 and above. If you really need to run Raspbian on older hardware then use the last version of Raspbian Stretch, or if you need to run a terminal/server install try something lighter such as DietPi.

Thonny is now the default Python editor and it works well for beginners and experts alike.



VERDICT

DEVELOPER: Raspberry Pi Trading
WEB: www.raspberrypi.org
LICENCE: Mixed, same as Debian

FEATURES	8/10	EASE OF USE	9/10
PERFORMANCE	8/10	DOCUMENTATION	8/10

Grown-up, but it still retains the features that make it the de facto standard for Pi users – especially the Pi 4.

» **Rating 8/10**

Supercharge your Raspberry Pi

Get the most out of your Raspberry Pi with these performance-enhancing tips and tricks.

Your Raspberry Pi is plugged in. Raspbian is installed on the SD card and you are right in the middle of setting up a wireless print server or building a robot to collect your mail from your doormat.

But are you truly getting the most from your little computer? Do the components you're using maximise the potential of your Raspberry Pi or are they holding it back? Perhaps you haven't explored the full set of options in Raspbian, or you're running the entire OS from SD card, something that can reduce SD card lifespan.

Various tools and techniques can be employed to improve performance, from choosing the right hardware to overclocking the CPU. You might even maximise storage space on the Raspberry Pi's SD card or all but replace it with a secondary device to help improve speed.

Use these tips and tricks to reconfigure your Raspberry Pi setup and optimise software and hardware to ensure you get the best performance for your projects.

01 Use better storage hardware

Your choice of storage media can have an impact on your Raspberry Pi's performance, regardless of the operating system. A low capacity SD card with poor error correction, is going to be slower than a larger card with greater resilience, so you need to find the right balance for your project and shop wisely.

Above There's a great guide to SD cards at **elinux.org/RPi_SD_cards**

02 Choosing the best SD card

Various standards of SD card are available, with the more expensive designed for better error correction. For the best performance on your Raspberry Pi, choose an SDHC card with a high rating. The same advice applies to MicroSD cards, which you can use on your Raspberry Pi with an SD card adaptor or directly insert into a Raspberry Pi B+.

```
pi@raspberrypi: ~
login as: pi
pi@192.168.0.4's password:
Linux raspberrypi 3.12.28+ #709 PREEMPT Mon Sep 8 15:28:00 BST 2014 armv6l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
pi@raspberrypi ~ $ sudo raspi-config
```

03 Make the most of your storage

You'll typically need 1-2GB of storage for your chosen Raspberry Pi distro, so any remaining storage on your SD card will be used for updates and data you create or save.

In Raspbian you can open a command line and run the configuration utility to gain more space (only if your SD card's greater than 2 GB):

```
sudo raspi-config
```

04 Expand the Raspbian partition

Maximising the partition affords the full capacity of your SD card, which will increase the media's lifespan (there is more space to write too, so the same sectors aren't being overwritten as often).

With raspi-config running, use the arrow keys to select expand_rootfs in the menu. Then wait briefly while the partition is resized.

05 Write data to RAM

Rather than reading and writing data to your SD card – something that will eventually result in a deterioration of reliability and performance – you can configure Raspbian to write to the system RAM, which will speed things up slightly and improve SD card performance.

This is achieved using fstab (file systems table), a system configuration available in most Linux distros.

```
pi@raspberrypi: ~
  GNU nano 2.2.6              File: /etc/fstab

proc            /proc           proc    defaults          0       0
/dev/mmcblk0p5  /boot           vfat    defaults          0       2
/dev/mmcblk0p6  /               ext4    defaults,noatime  0       1
# a swapfile is not a swap partition, so no using swapon|off from here on, use $

                                [ Read 4 lines ]
^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

06 Enable fstab in Raspbian

This is much like creating a RAM disk in Windows and is almost as easy to setup. In the command line, enter:

```
sudo nano /etc/fstab
```

Add the following line to mount a virtual file system:

```
tmpfs /var/log tmpfs defaults,noatime,nosuid,mode=0755,size=100m 0 0
```

Follow this by saving and exiting nano (Ctrl+X), then safely restarting the Pi:

```
sudo shutdown -r now
```

Buy rated SD cards

It's all too tempting to boot up your Raspberry Pi with an image copied to an SD card that you just pulled out of your DSLR or phone. After all, they're all the same, right? The chances are that your chosen SD card was one that you had lying about when you bought your Raspberry Pi. It might be good enough but if you want the best performance, a high-rated SDHC card with plenty of space is recommended. Such media is inexpensive and can be bought online or in supermarkets. Just make sure you're buying a known brand!

Above Having your filesystem on a USB stick is great for backups as well as performance boosts

Picking an external USB drive

Speeding up your Raspberry Pi by migrating the root filesystem to an external USB drive is a start, but what sort of device should you use for the best performance? With a USB thumb drive you can add flash storage up to 16 GB without running into any significant problems (the larger the drive, the greater the current is required to read/write). Anything larger is expensive and unnecessary. If you're planning to use an external HDD, there are no power issues as it will have its own power supply. As ever, your choice should suit your project.

07 Configure fstab for fast performance

Upon restarting, the virtual file system will be mounted and /var/log on the RAM disk. Other directories that can be moved to RAM include:

```
tmpfs /tmp tmpfs defaults,noatime,nosuid,size=100m 0 0
tmpfs /var/tmp tmpfs defaults,noatime,nosuid,size=30m 0 0
tmpfs /var/log tmpfs defaults,noatime,nosuid,mode=0755,size=100m 0 0
tmpfs /var/run tmpfs defaults,noatime,nosuid,mode=0755,size=2m 0 0
tmpfs /var/spool/mqueue tmpfs defaults,noatime,nosuid,mode=0700,gid=12,size=30m 0 0
```

Add each to /etc/fstab in nano.

08 Move your OS to a HDD

If you're concerned about the lifespan of the SD card, why not reduce your Raspberry Pi's reliance on it? Instead of using the SD card as a sort of budget SSD, change its role and add a HDD or USB stick to run the operating system, leaving the SD card for bootstrapping. This can give a marked performance boost to the SD card.

09 Back up the SD card

Begin by creating a copy of your Raspberry Pi's SD card. Shut down, remove the card and insert it into your desktop computer. In the command line, run:

```
sudo dd bs=4M if=/dev/sdb of=~/backup.img
```

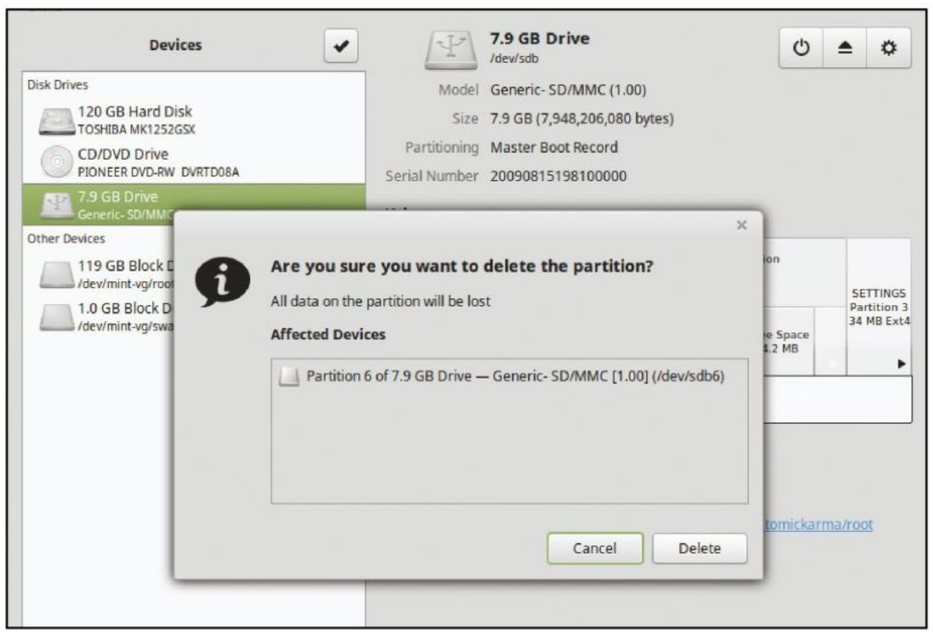
The path /dev/sdb represents the SD card. Copying should take 5-10 minutes. When complete, remove the SD card and connect your USB device.

10 Copy Raspbian to USB

Using a blank Ext4-formatted USB thumb drive (or external HDD) as the destination drive, enter:

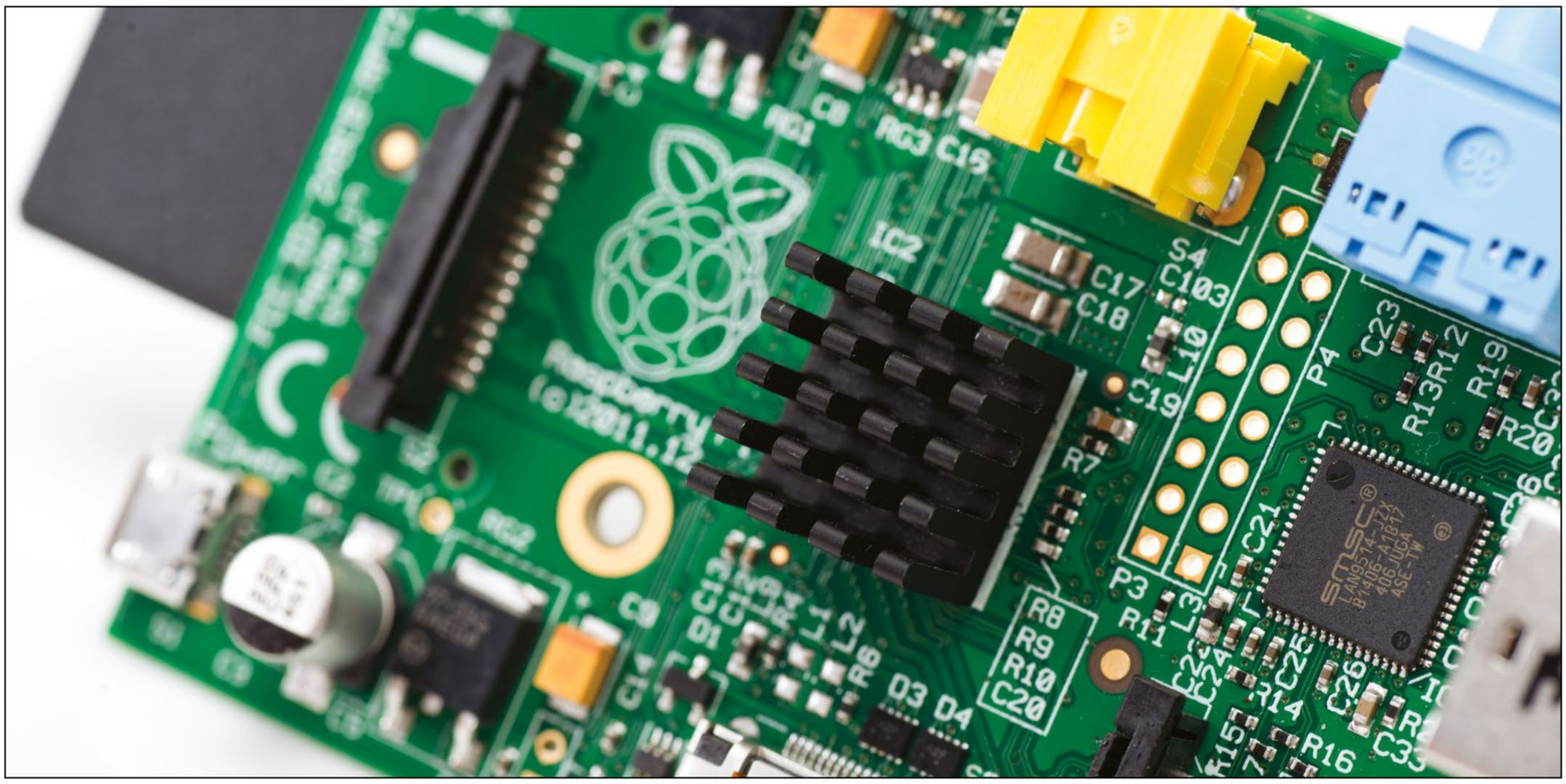
```
sudo dd bs=4M if=~/backup.img of=/dev/sdc
```

Leave the backup on your computer, just in case something goes wrong. With an SD card and USB storage device sharing an identical disk image, it's time to consider what you're going to do next – create a faster Raspberry Pi.



11 Split the Raspbian partitions

Ideally, the boot partition should remain on the SD card while the root filesystem is run from the external HDD or USB thumb drive. Using your preferred partition manager (Disk Utility is in most distros), unmount and delete the root filesystem from the SD card, ensuring you have retained the boot partition. After removing the SD card, connect your USB device and delete the boot partition, taking care to leave the root filesystem intact. Then resize the root filesystem on the USB device, making sure that 10 MB remains.



Above Heat sinks for the Pi are widely available and usually cost less than $10

12 Identify the root filesystem

With this configuration you're going to have the SD card and the external USB storage connected, so you need to tell the Pi where the root filesystem is. Still on the desktop Linux computer with your SD card inserted, run:

```
sudo nano /boot/cmdline.txt
```

Find root=/dev/mmcblk0p2 (or similar) and change that to read root=/dev/sda2 which is your external USB storage. Save and exit.

13 Add other USB devices

You can now restart your Pi with the storage devices attached, but as soon as you connect further USB media you'll suffer problems. Avoid this by installing gdisk:

```
sudo apt-get update
sudo apt-get install gdisk
```

Then run gdisk:

```
sudo gdisk /dev/sdb
```

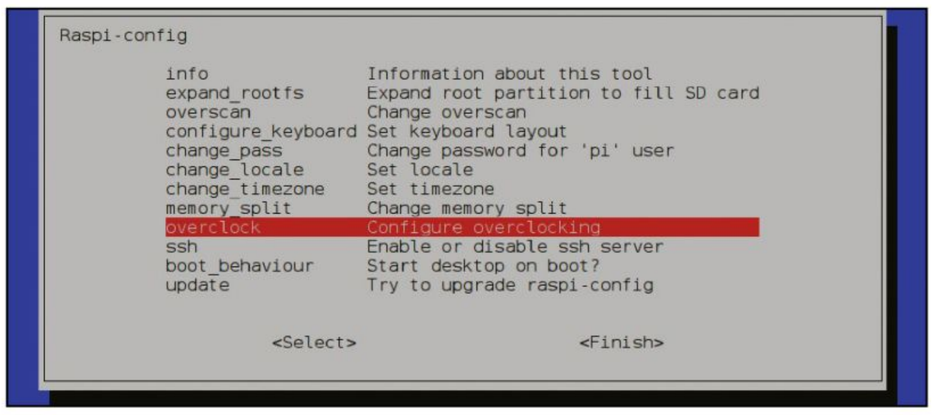
Enter ? to display the options and select Recovery and Transformation options (experts only), followed by Load MBR and Build Fresh GPT. Tap ? one last time and select 'Write Table to Disk' and exit. Remove and replace the USB device and run gdisk again. This time enter I and then 1 to display the Partition Unique GUID.

14 Make your Pi fast & reliable

Make a note of the GUID and then switch to the SD card. Reopen cmdline.txt and change root=/dev/mmcblk0p2 to root=PARTUUID=XXXXXX, where the numerical string from the partition unique GUID should replace the XXXXXX. When you're done, save and exit. You can then start your Raspberry Pi. Congratulations, your Raspberry Pi is now faster and more reliable to use!

15 Boost performance with overclocking

Need more from your Raspberry Pi? It is possible to overclock the computer, although you should be aware of the risks inherent with this activity. You should also ensure that your Raspberry Pi's processor is suitably cooled – heatsinks for the CPU, Ethernet controller and power regulator can be purchased online.



16 Overclock your Raspberry Pi

Overclocking is available through raspi-config. Launch from the command line and arrow down to the overclock option. Four further options are available: Modest, Medium, High and Turbo. With your ideal clock speed selected, exit raspi-config and restart your Raspberry Pi to apply:

```
sudo shutdown -r now
```

Now you will need to perform tests to see how stable it is overclocked. Raspberry Pi founder, Eben Upton, suggests running *Quake 3* as a good stress test. Should the Pi fail to boot, hold Shift to boot without overclocking, run raspi-config and select a more modest overclock.

17 Run Raspbian without the GUI

Despite these changes, you may find that the GUI remains slow. If you find yourself running a lot of commands in bash, the best thing to do is disable launching into X. In raspi-config, choose boot_behaviour and select the first (default) option to ensure your Pi boots to the command line. Should you need the GUI, enter 'startx' in Terminal.

Overclock with a heatsink

Overclocking is potentially dangerous to any computer system, which is why it's great that the Raspberry Pi developers have included the facility in their approved operating system and allowed its use under warranty. If you're using this feature, heatsinks and water cooling systems are available for the Raspberry Pi to ensure you don't bake the CPU and RAM when in use.

Copying files from your PC to your Pi

Swapping data between a Raspberry Pi and your PC can be tricky. **Christian Cawley** shows you all the useful options.

OUR EXPERT

Christian Cawley
A former hardware and software support engineer, Christian has eight Raspberry Pi Model Bs in captivity. He plans to add a Model A.

One of the key aspects of using a Raspberry Pi involves copying data to it. This could be anything, depending on your use; perhaps you have a Bluetooth audio streamer project on the go, or a digital picture frame. You may need to send over a directory of MP3 files to play, or PNG images to view.

Dumping the files to a USB stick on your PC then dragging them from the flash storage on your Pi is one option. It's straightforward, but lacks finesse, and adds too much physical interaction. Instead, it's smarter to use a network option, taking advantage of the local network to beam the data wirelessly or over Ethernet from your PC to your Pi. You might even set up a network share and make your Pi available to move data to and from, using any device on your network. These methods should work with all Raspberry Pi models, although some will be more suited to the later devices.

You can stick it!

Before we look at the high-tech alternatives, however, it's worth going through the basics. Say you have a bunch of files you need to copy from a PC to your Raspberry Pi. All you should need to do is insert the USB drive in to your PC, wait for it to mount, then drag the files to it.

Safely eject the USB stick once the data has been copied, then insert it into your Raspberry Pi. At this point, it should be automatically mounted in the Raspbian operating system – the results may vary with other OSes, though. If the USB stick doesn't automatically mount, open the terminal and find the UUID (Universal Unique Identifier) of the device:

```
ls -l /dev/disk/by-uuid
```

The USB device will probably be labelled **sda1**; keep a note of the name regardless. Next, create a mount point like this:

```
sudo mkdir /media/usb
```

Next, set the Pi user as the folder owner.

```
sudo chown -R pi:pi /media/usb
```

Finally, mount the drive with:

```
sudo mount /dev/sda1 /media/usb -o uid=pi,gid=pi
```

All you need to do now is open the file manager, find the device in the left pane, and click to browse. You can then drag the files from the USB flash device to the intended directory on your Raspberry Pi. When you're done, use the eject button in the file manager to remove the USB stick. If you manually mounted, enter:

```
sudo umount /media/usb
```

With microSD cards, things are a little different. Sure, you can use the **/boot** directory to copy data from your PC without disrupting the main file system, then drag it into place when the Pi is running. But if you want to use SD storage with a USB adaptor, you'll need to install software that lets Raspbian access drives in the exFAT format. SD storage is almost always formatted in this way as the file table has been optimised for SD cards. The tools you need are *exfat-fuse* and exfat-utils. These work together to mount exFAT drives as well as format in exFAT format.

As ever, start with an update and upgrade:

```
sudo apt update
sudo apt upgrade
```

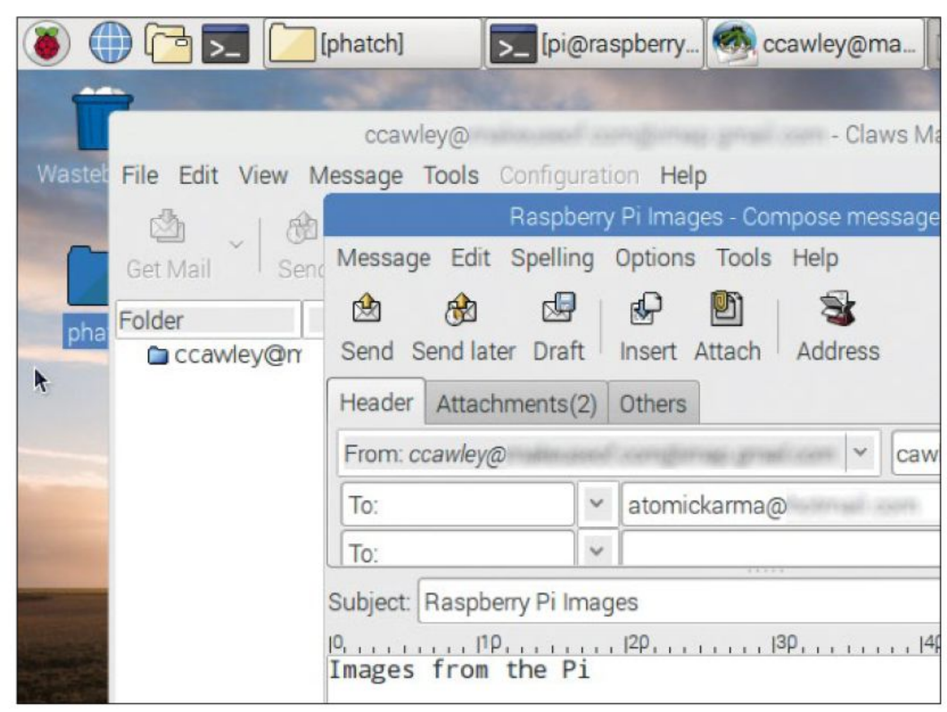
Once complete, install *exfat-fuse* and *exfat-utils*.

```
sudo apt install exfat-fuse
sudo apt install exfat-utils
```

This is particularly useful if you're planning to set up a Samba server or any other project that requires removable flash media.

Browsing by

Another straightforward option for copying files from a desktop computer to your Pi is to use the browser. Two



Email attachment limits will prevent you from sharing too much data from your PC to your Raspberry Pi, so keep an eye on file sizes.

QUICK TIP

Communicating with a Raspberry Pi over SSH from Windows 10 no longer requires *PuTTY*. Instead, simply install the OpenSSH client in PowerShell. Do this in Settings > Apps > Manage optional features. Click 'Add a feature', then select 'OpenSSH Client'. Reboot Windows to install, then open PowerShell to start an SSH session.

easy choices are available: email and cloud storage. Simply attach the files to an email and send it to yourself, or upload to your cloud account from your main PC. Then, on the Raspberry Pi, open the browser and download the attachment or cloud upload. You might not even need to wait for the email to arrive – simply check the Sent Items folder to find and download the file.

This should work without any problem, although it does of course require internet access. You'll also need to be running the *Chromium* browser for the best results, especially if you're using Gmail. If not, you'll find *Chromium* preinstalled in Raspbian Buster. You can also install it on other operating systems with:

```
sudo apt install chromium-browser --yes
```

Although *Chromium* will run on older Raspberry Pi models, the best results from this method can be enjoyed using a Raspberry Pi 3 or later. However, be aware that popular cloud storage options are slow on the Pi, even the Pi 4. This isn't a smart option for larger files, but if you have a few photos you need to quickly move from Pi to PC or vice versa, it's worth it as a simple solution.

Keep it secret, keep it SSH

SSH is one of the key technologies for remote access to Linux-based devices. To use it for transferring data, the Secure Copy Protocol command *scp* can be used.

Start off by ensuring SSH is enabled in Raspbian. The easiest way to do this in advance is to insert the SD card into your PC's card reader, browse to the **/boot** directory, and create an empty text file simply called **ssh**. After safely removing the SD card and replacing it in your Pi, booting up will enable SSH, and you'll be able to start a remote session over SSH.

Another way to enable SSH is on the desktop. Simply open the Menu, then go to Preferences, Raspberry Pi Configuration. Here, select Interfaces then click the Enabled button for SSH. Click OK to confirm.

Finally, you can use *raspi-config* on the command line to enable SSH. Launch with:

```
sudo raspi-config
```

then open Interfacing Options. Use the arrow keys to highlight SSH, tap Enter, then OK to confirm. You'll be prompted to reboot the Pi.

With SSH enabled, SCP is ready to use. Start by establishing an SSH connection:

```
ssh pi@192.168.0.85
```

Enter the password when prompted. To send data from your PC to the Pi, use:

```
scp file.txt pi@192.168.0.85:
```

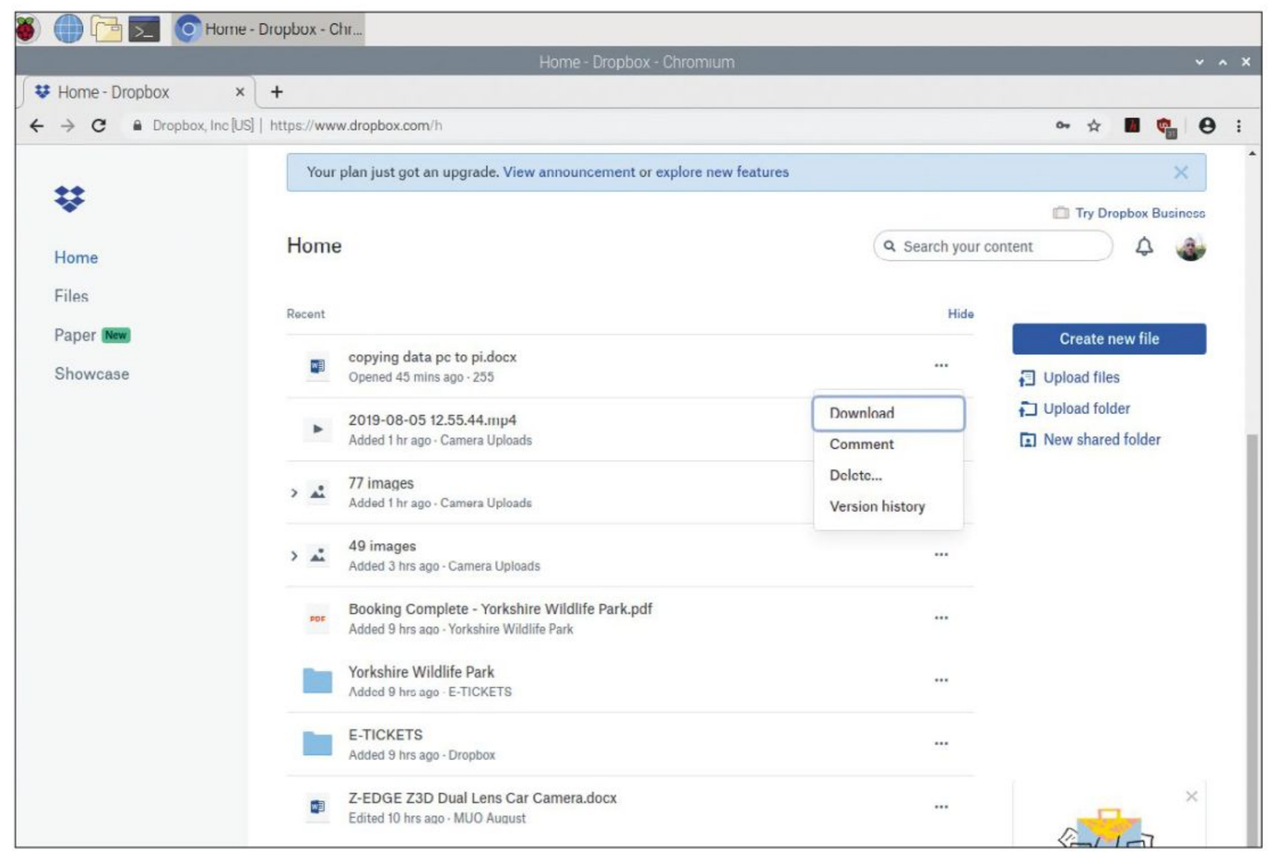
Be sure to include the colon `:` at the end of the command. Note that your Raspberry Pi will typically have a different IP address! This will send the file to the Pi account's **/home** directory. To send it elsewhere, specify a file path:

```
scp file.txt pi@192.168.0.85:<subdirectory>
```

Meanwhile, if you want to pull a file from your Pi to your desktop PC, use:

```
scp pi@192.168.0.15:file.txt
```

Again, the default destination is the **/home** directory for the user. Specify a file path and filename after the colon. Any device with command-line SSH can use this method. You're not limited to desktop operating systems, either; for example, an Android or iPhone with a terminal client app can move data to and from a Raspberry Pi using *scp* in the same way.



Cloud services don't typically have apps for the Pi, but you should be able to access them in the Chromium browser.

Network sharing is caring

With your Raspberry Pi set up on your network (wirelessly or over Wi-Fi), you might consider a Samba share. This is particularly useful if you plan to regularly transfer data between your PC and Raspberry Pi. This method basically establishes a network share on the Pi, accessible from anywhere on your network.

Easy to set up, this option is particularly useful if your main PC is running Windows. For the best results, this

QUICK TIP

By now this should go without saying, but if you're regularly copying data to and from your Raspberry Pi, a high-rated microSD card is needed. Cheaper, low-rated cards are prone to corruption.

» DIRECT ETHERNET CONNECTION

You're not necessarily reliant on USB flash sticks and router-based network connections to share data between your PC and Raspberry Pi. A more straightforward option is available: just hook up an Ethernet cable.

Raspberry Pi Model A and B devices with an Ethernet port can be connected to your PC. Once hooked up, you can communicate with the Pi over SSH, copy files using *scp* or *FileZilla* as described, and disconnect by removing the Ethernet cable.

On Linux, start by installing *avahi-daemon*. This is a package that includes the zeroconf software needed to detect the Raspberry Pi on your computer:

```
sudo apt install avahi-daemon
```

If your main computer is a Mac, zeroconf is baked in. Windows users will need to install *Bonjour* – this is easily done by installing *Skype*, *iTunes (arrg!–Ed)* or *Adobe Photoshop CS3* (or later). Zeroconf will assign a name to your Raspberry Pi (**raspberrypi.local**) which can then be used by any device on your network.

Just boot your Pi and connect it to your PC with the Ethernet cable. Wait a few minutes for it to be detected, then open your SSH tool or SFTP client and connect to **raspberrypi.local**, entering the Pi user and password when prompted.

QUICK TIP

If you plan to access your Pi wirelessly, you'll need to know the IP address. These can be 4–12 digits long, and tricky to recall at times. The smart option? Log in to your router and view the connected devices – your Pi will be listed with its hostname and IP address.

will need to be set up to share a regularly used directory on your Raspberry Pi. It might be the standard **/home** directory, for example, or a shared folder within this.

After updating and upgrading Raspbian, install the Samba software:

```
sudo apt install samba samba-common-bin/
```

Next, if you need a new folder to share, create it with:

```
mkdir /home/pi/shared
```

The destination directory can be specified in the **smb.conf** file. Open this in *nano* to edit:

```
sudo nano /etc/samba/smb.conf
```

Add the following lines to the bottom:

```
[your network share name]
path = /home/pi/shared
writeable=Yes
create mask=0777
directory mask=0777
public=no
```

The conditions are self-explanatory. Setting `writeable` to `yes` means data can be written to the network share. The `create mask` and `directory mask` settings allow read, write, and execute permissions. Ensure `public` is set to `no` to maintain security and ensure that the Pi's username and password are needed to access the folder. Save the file by pressing Ctrl+X, then Y to confirm, and create a Samba user account and password:

```
sudo smbpasswd -a pi
```

Enter the password you wish to use when prompted. Next, restart Samba:

```
sudo systemctl restart smbd
```

Your Pi is then ready to be accessed remotely, a little like a network drive. You'll be able to connect to it in the same way as you attach a network drive in your usual operating system. This method is available Linux, macOS and Windows.

Using more acronyms...

Finally, there's the reliable option of FTP. This method specifically uses the SSH File Transfer Protocol, which you might think of as 'FTP over SSH'. This is the preference of many Pi users, thanks to being so simple to set up.

All you need to do is install an FTP application that supports SFTP, such as *FileZilla* (**https://filezilla-project.org**), ensure SSH is enabled on your Pi, and the job's done. Start by installing *FileZilla* on your PC. For Ubuntu, first upgrade and update:

```
sudo apt update
sudo apt upgrade
```

Next, install the *FileZilla* FTP software:

```
sudo apt install filezilla
```

You should have followed the steps earlier to enable SSH on your Raspberry Pi before proceeding. Once that's done and the Pi has rebooted, run *FileZilla* and open the Site Manager. Here, select New Site, then in the right-hand pane enter the Pi's IP address in the General tab Host field. For Server Type, select 'SFTP – SSH File Transfer Protocol'.

Below this, Set the Logon Type to Normal and input `pi` as the user. Input the matching password for the account, then perhaps leave a note in the Comments field in case you forget what the connection is for (useful if you use *FileZilla* regularly for multiple machines). Finally, click Connect to establish the secure connection to your Raspberry Pi.

Swapping files between machines is straightforward. The left-hand pane represents the local machine (your PC) whereas the right-hand pane displays a browsable file structure for your Raspberry Pi. To copy files to the Pi, browse to them in the left pane and drag them to the desired folder in the right pane. To copy files from your Pi to your PC, move them from right to left.

Using an SFTP client to copy your files is probably the most user-friendly option. Like a Samba share, the Pi can be accessed from any platform. You're not limited to Linux, macOS or Windows either – mobile devices with SFTP clients can access data stored on the Pi too.

» WHY YOU MIGHT COPY DATA

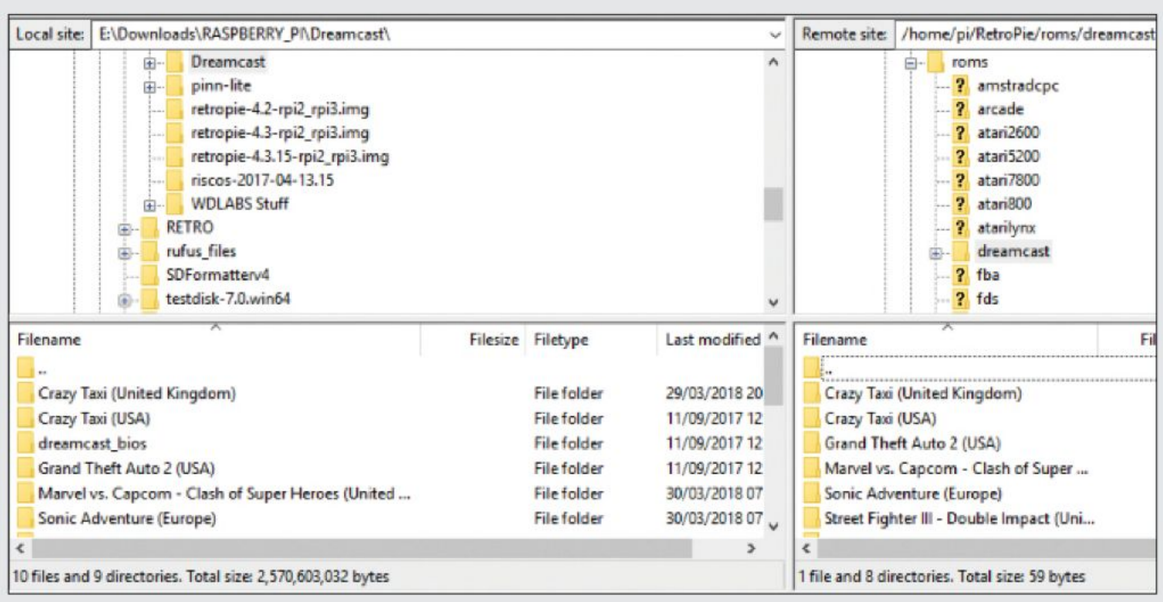
At the top of this tutorial we briefly considered Bluetooth streaming and digital photo frame projects as reasons to copy data to your Raspberry Pi. Of course, these are just two examples; many more reasons exist that you might not have considered.

For example, if you're running *Kodi*, custom add-ons can be added to your Pi. Retro gamers will find SFTP an excellent option for copying game ROMs to their *RetroPie* or *Recalbox* systems. Using your Raspberry Pi for nature time-lapse photography? At some point you'll need to remove the photos from your device.

Hosting a website on your Raspberry Pi? If so, you'll need to copy the files from your PC, unless you build it completely from scratch on the Pi. This can take time, so being able to prepare the files, begin the copying process and then go and do something else is convenient.

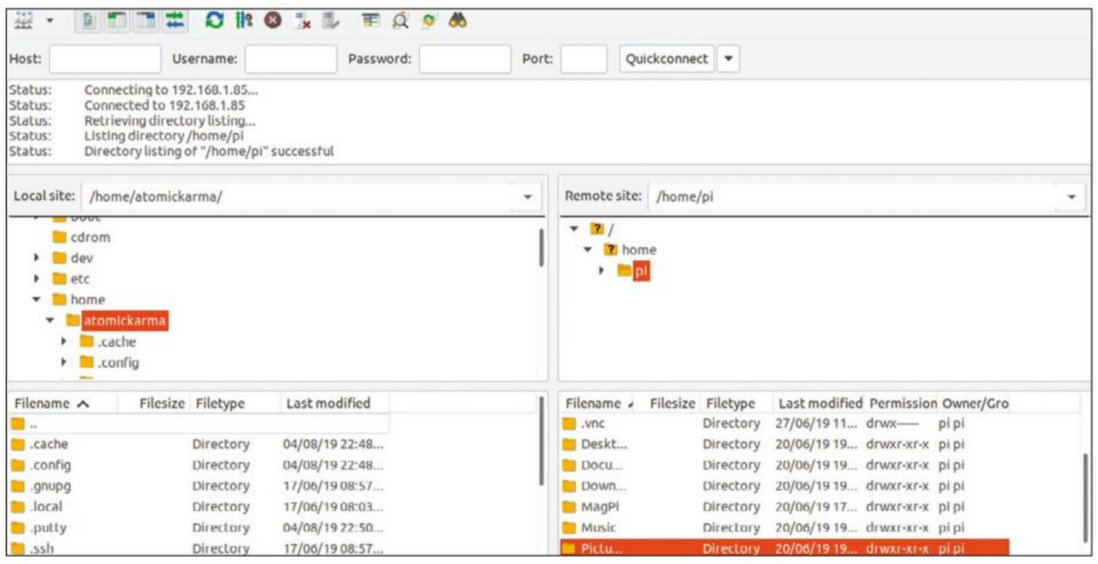
Meanwhile, a Pi set up as an IoT project can benefit massively from complete remote access. This might be to manually update software, add configuration files and so on. Or you may need to access and retrieve a database populated by data collected using the project's sensors, for instance.

Similarly, you might have an older non-wireless Raspberry Pi robot project that needs some files updating from your PC over Ethernet. You'll no doubt find many more reasons of your own.



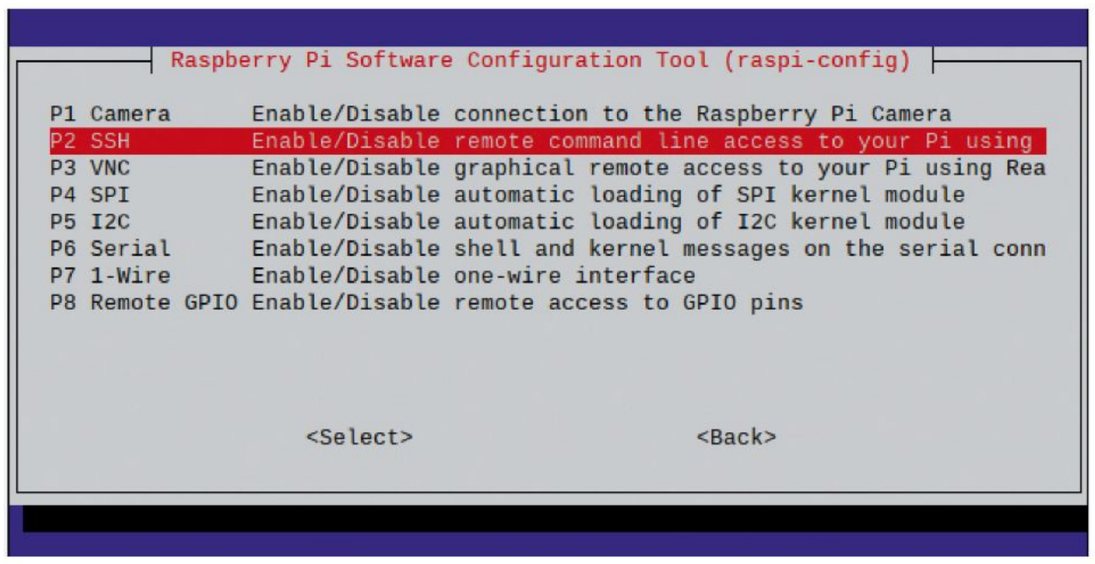
It doesn't matter if it's a media server, web server or IoT project – there's always a reason to copy data to and from a Raspberry Pi.

COPY DATA TO YOUR RASPBERRY Pi USING SFTP



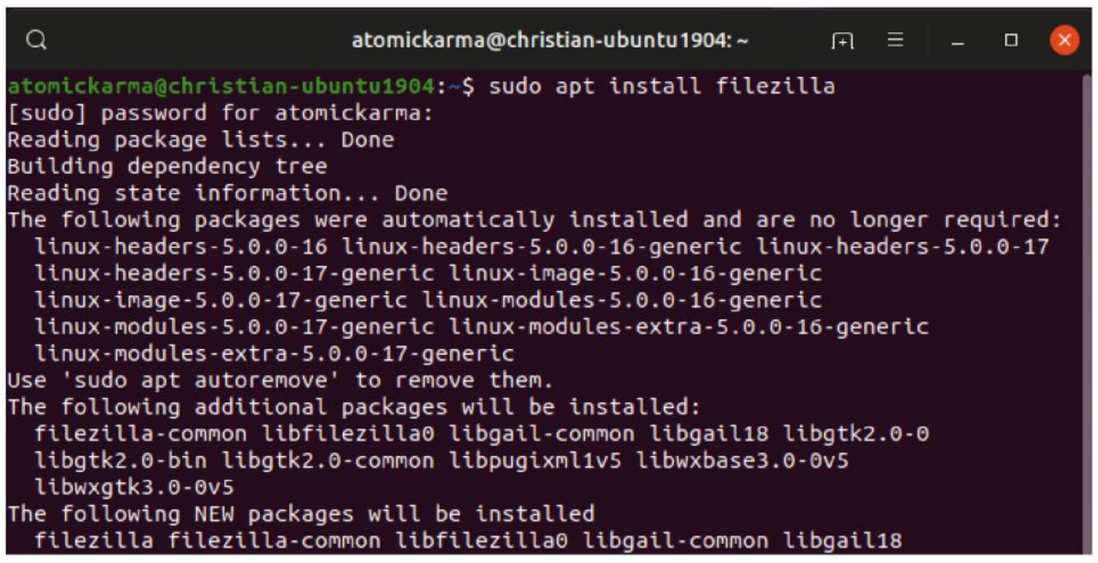
1 The benefits of SFTP

With so many options for copying data to and from a Raspberry Pi, you might wonder which is best. Easiest to set up and compatible with software on all platforms is SFTP (SSH File Transfer Protocol). The best software for this is the *FileZilla* client, available on Linux, macOS, Windows and more. Once installed, simply drag and drop data to and from your Raspberry Pi.



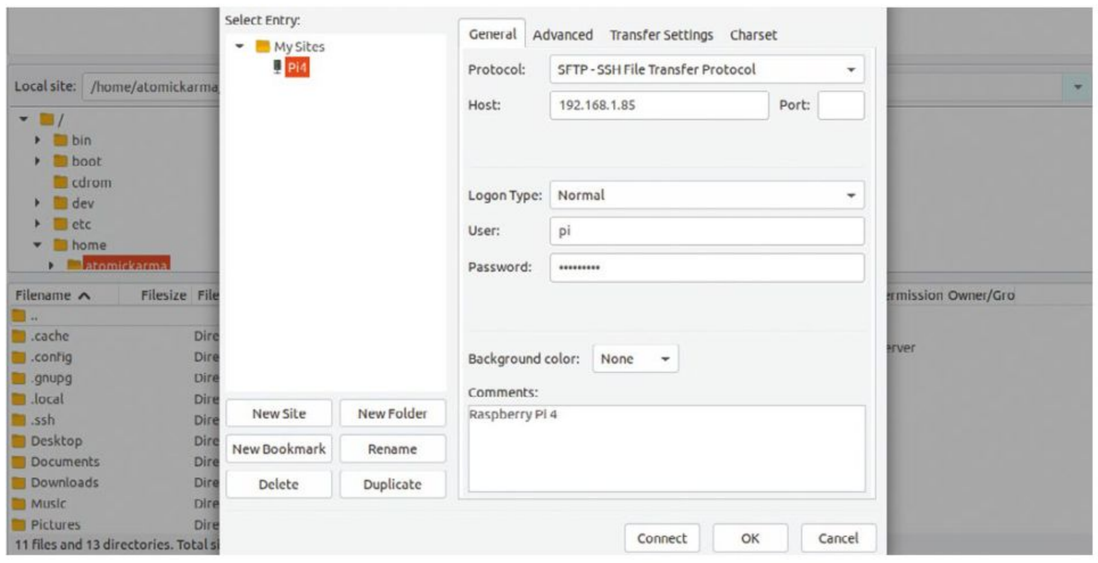
2 Set up SSH

Start off by enabling SSH on your Pi. You might use *raspi-config* (Interfacing Options > SSH) or create an empty text file called **ssh** in the boot directory before booting. But it's easiest via the desktop. Open Menu > Preferences > Raspberry Pi Configuration > Interfaces and set SSH to 'Enabled'. Click OK to finish, then reboot the Raspberry Pi to ensure SSH is set.



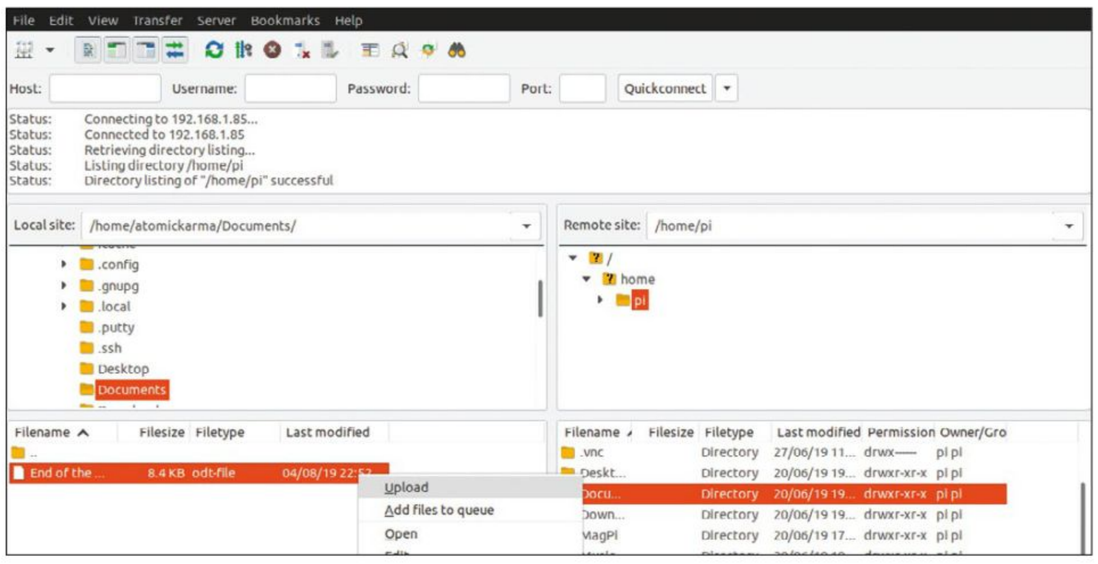
3 Install an SFTP Client

The open source *FileZilla* FTP client features SFTP functionality. This allows you to move data from your PC to your Raspberry Pi using SSH, but with the mouse-driven interface of a desktop app. To install on your desktop PC, head to **https://filezilla-project.org**. On Ubuntu, install with: `sudo apt install filezilla`.



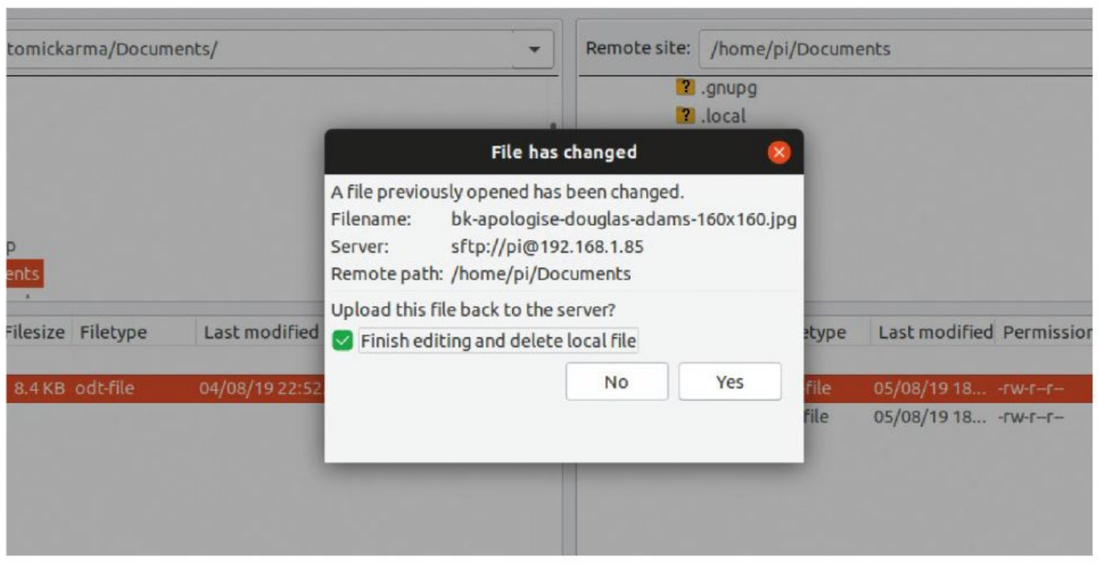
4 Set up a site in FileZilla

To use SFTP, set up your Raspberry Pi as a 'site' in *FileZilla*. Open File > Site Manager, then New Site. On the right-hand pane, add your Pi's hostname or IP address. Below this, set the SFTP option, then input your Pi's username and password. Finally, click Connect. As long as both devices are connected to the same network, a remote connection should be established.



5 Copy data to your Pi

Copying data with *FileZilla* is simple. On the left, you'll see files on your PC. On the right are those stored on your Raspberry Pi. To copy data to your Raspberry Pi, browse for it in the left-hand pane, then select the file you want to copy, and double-click it. The transfer will commence. You can also right-click and choose Upload, or simply drag and drop.



6 Edit remote files locally

As well as copying files from the Raspberry Pi, you can also edit them on your PC. This creates a temporary file that is then re-uploaded once saved. Right-click a file, then select Edit to download and open it in a suitable app on your PC. After saving, *FileZilla* will ask you if you want to copy it back to the Pi. Click OK to proceed.

Debug serial comms

Sean Conway walks you through the process of troubleshooting communication problems between a PC application and a serial device.

Sean D. Conway is a retired IT security specialist who now exercises his brain by mining knowledge through Raspberry Pi projects.

The itch for an article on troubleshooting USB-to-serial connections needed to be scratched, after working on a request for assistance from a local model railroad group (*see* **Linux Format 239**). This tutorial will provide the knowledge to troubleshoot communication (comms) issues from an app on a PC.
to a serial device, using a USB-to-serial cable/interface.

In the diagram shown below, there is a PC running an application that requires a serial interface. The PC is connected to the serial device, shown on the far right side of the drawing, using a USB-to-serial cable/interface. The drawing contains a few additional details that will be introduced as our discussion expands.

Hardware in the PC supported serial comms; at one point during the evolution of the PC, a minimum of two DB9 ports (the 9-pin COM port) was considered standard. The red highlighted text in the diagram are some examples of COM port communication needs. But all things come to an end, and the ubiquitous USB port soon replaced COM ports. To consolidate all the different serial device comms requirements, a USB-to-serial cable/adaptor came into play. Unfortunately, this generic serial comms doesn't always work, and it can be frustrating trying to isolate the root cause of serial communication failure.

After connecting the USB-to-serial cable to the host PC, the first step is to determine if the OS recognises the USB-to-serial adaptor. In Linux, the *dmesg* command reveals information regarding hardware detection, driver assignments and communication port allocation. It is important to take note of what COM port (COM1, 2, 4 or so on) is used – this will be needed to support our efforts, as well as set up the application.

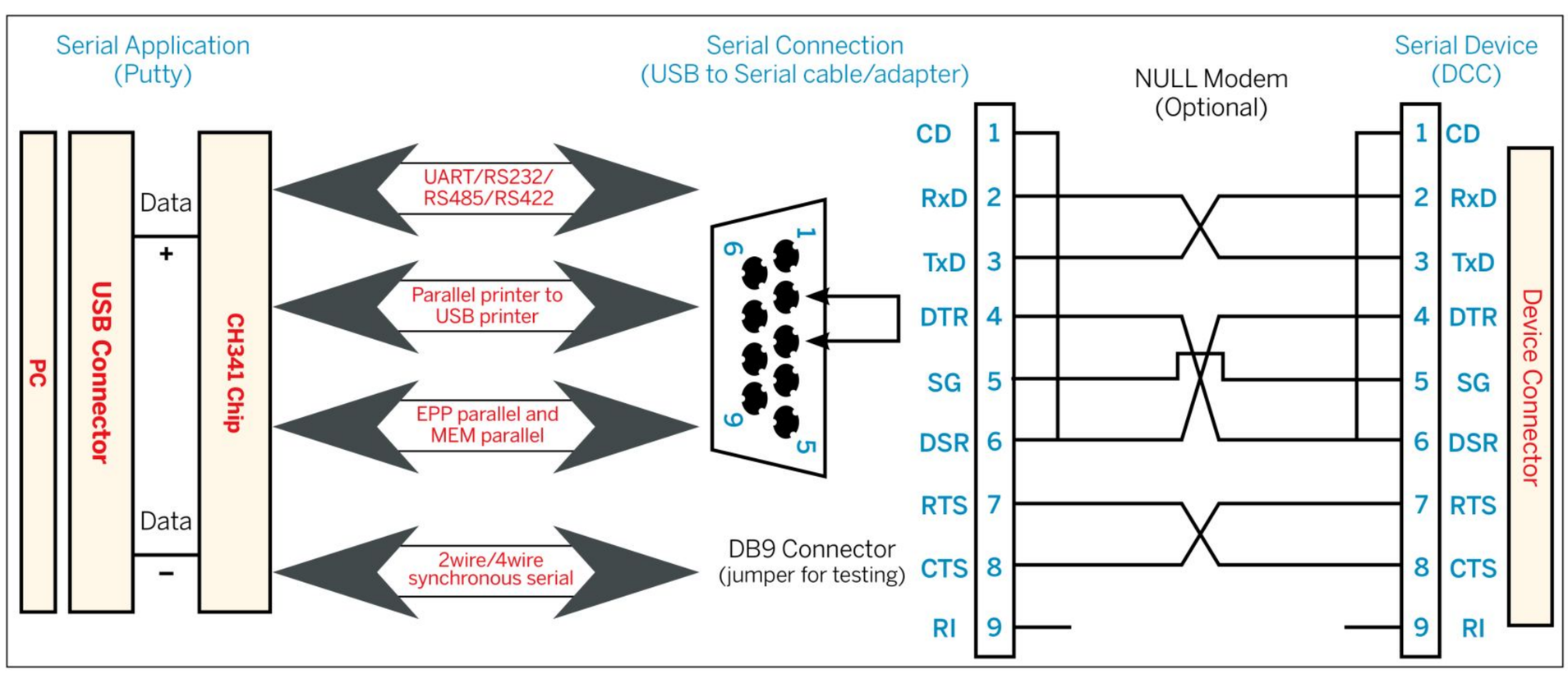
The easiest way to determine if the operating system detects the USB cable/adaptor device is to install the device after a reboot. Installing it before a reboot causes the device details to be consolidated inside other boot message content.

From a command line interface type `dmesg`. If the cable/adapter was plugged into the PC before bootup, the details we want are embedded in the *dmesg* listing. For the novice it's much easier to plug in the cable/adapter after a reboot and then run *dmesg*. The new device details will be at the end of the listing.

From the information returned you can extract which chipset/UART is in the cable/adapter. The details will also identify which communication port has been assigned. You are looking for something like 'ttyUSB', suffixed with an assignment number – for example, `ttyUSB0`.

For the next step in troubleshooting we are going to use an application called *Putty* – its UI makes it easy to configure the ports. The goal is to confirm that software on the host PC can communicate through a USB-to-serial adapter. Once *Putty* is installed, open the application. Select 'Serial' in the 'Connection type' choices. In the 'Serial line' field, enter the COM port details you have recorded previously. Proceed to the serial connection screen. The most common, default setup is 9600baud 8N1. The number sequence translation means: eight bits of data, no stop bits, one parity bit.

Give the session configuration a name and save it. The session configuration is not saved by default when you leave the application; if you don't save the session, you'll need to input the configuration again next time.



Serial connection path mapping, with additional landmarks shown.

Before starting the terminal session configured in *Putty*, let's create a physical connection (technically, a short) between pins 2 and 3 on the nine-pin serial connector. Refer to the DB9 connector details on the diagram: pins 2 and 3 are the serial transmit/receive pins. By connecting the pins together, we are redirecting the output on the transmit pin back to the input on the receive pin.

When the *Putty* session is activated you will be taken to a blank, black terminal-style window. Now select a character on the keyboard. The keyboard character selected should appear on the screen. If the character appears, Putty has transmitted the character data serially out of the transmit pin and, with the short in place, routed the transmission back to the receive pin, to be displayed on the screen.

The real thing

Now that we have confirmed *Putty* can transmit and receive over the USB-to-serial cable/interface, we can replace it with our unknown serial application. The application will no doubt require the configuration details (COM port) used to setup the *Putty* session. If the cable/adapter worked with *Putty* it should work with the application, provided the communication details set up are correct. If not, you have isolated the problem to the application.

The serial application will need to be configured with the serial communication protocol details. The COM port identifier is one piece of information. It is important in serial communication that the transmission rate and the framing of the serial message be compatible between sender and receiver. For *Putty*, we relied on the default serial options, 8N1.

Serial communication relies on both the transmitting end and receiving ends agreeing on how they will frame data for communication. Here is an analogy: two people want to communicate. One speaks French, the other English. Without establishing a language requirement, both parties can transmit, but neither can understand what is received. The same concept applies to serial communication setup.

The serial device manufacturer's documentation should provide the details required to set up the application. The *Putty* serial screen is an example of the values that should be found in the serial device documentation. The application needs to know the speed and bit configuration in order to communicate. You discovered the COM port by looking in the OS. It can take a lot of trial and error to work through the combination to get the serial protocol correct.

Remember that pins 2 and 3 on the serial connector are 'transmit' and 'receive'. When the cable is plugged into the serial device, how does the serial device's manufacturer take care of ensuring the transmit data pin is connected to the serial device's receive data pin? It would be handy if this requirement is provided in the documentation; if not, it's back to our troubleshooting.

There are two methods to achieve the connection goal. The serial device's manufacturer may cross the transmit and receive pins on the serial device end by switching the position of pins 2 and 3 on the connector, or assume that the user has sufficient knowledge to use the option of installing a null modem.

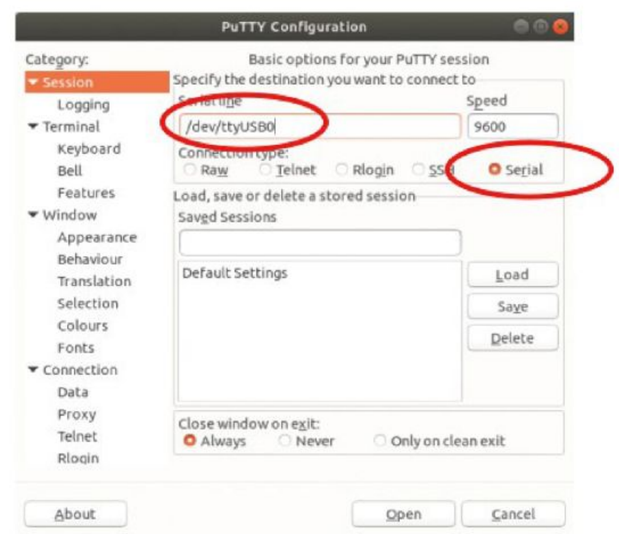
If the two pins are switched in the serial device, transmit and receive pairing will match. Otherwise, a null modem cable will need to be installed to take care of moving the transmit and receive connections. Refer to the diagram for null modem cable details.

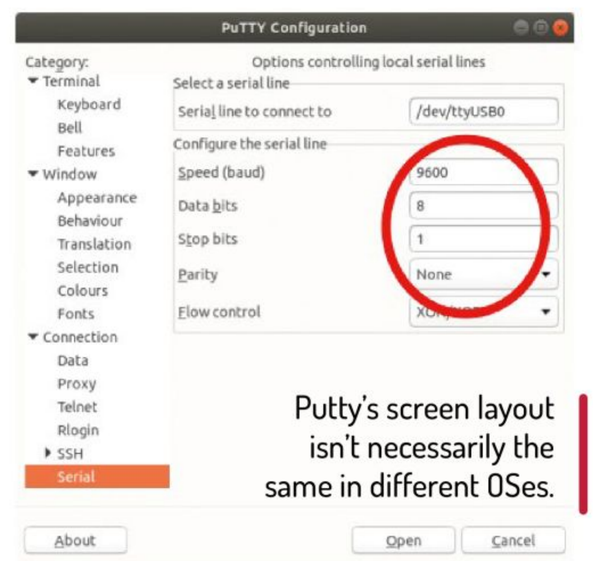
Let's summarise the troubleshooting process we've just developed, to really get it into our minds. First we confirmed the USB-to-serial cable/adapter was supported in the host PC by checking that it was registered in Linux. Next we confirmed that the computer could transmit and receive over the USB-to-serial cable/adapter using *Putty*. We shorted pins 2 and 3 on the cable/adapter's connector, so that the 'transmit' and 'receive' were connected, then transmitted a character and watched it appear on the screen.

Putty was then replaced with the computer application we want to actually test, and the application was configured with the required serial port and communication protocol requirements (rate and framing details) given by the manufacturer.

Finally, the PC was connected with the USB-to-serial cable/interface to the serial device, with the knowledge that transmit and receive was taken care of by the manufacturer's design, or a null modem cable needed to be installed to complete the communication path.

When you experience serial communication problems, what's most important is to determine if the infrastructure for serial communication actually works, without all the overhead of the new applications and devices. Once that is determined you can substitute the known for the unknown, and should be able to isolate what is not working.





Putty's screen layout isn't necessarily the same in different OSes.

QUICK TIP

You can find Simon Tatham's Putty repository at www.chiark.greenend.org.uk/~sgtatham/putty.

» THE LOST ART

You would think that the knowledge that a straight-through connection, or a null modem connection, is required for a bit of kit to work would be provided in the documentation. It's my experience that this knowledge, more often that not, is lacking, however.

With the demise of COM ports as computer hardware, a lot of knowledge about serial communications has been misplaced. The belief now seems to be "Just hook it up and it'll work!" We need to restore the lost knowledge to troubleshoot connection issues.

One caveat to keep in mind is not to apply both connection methods at the same time. If the manufacture has switched the pins and a null modem cable is installed, two rights in this case make a wrong. Think about it: pins 2 and 3 are switched by the manufacture, and then again by the null modem cable.

Raspberry Pi 3 A+

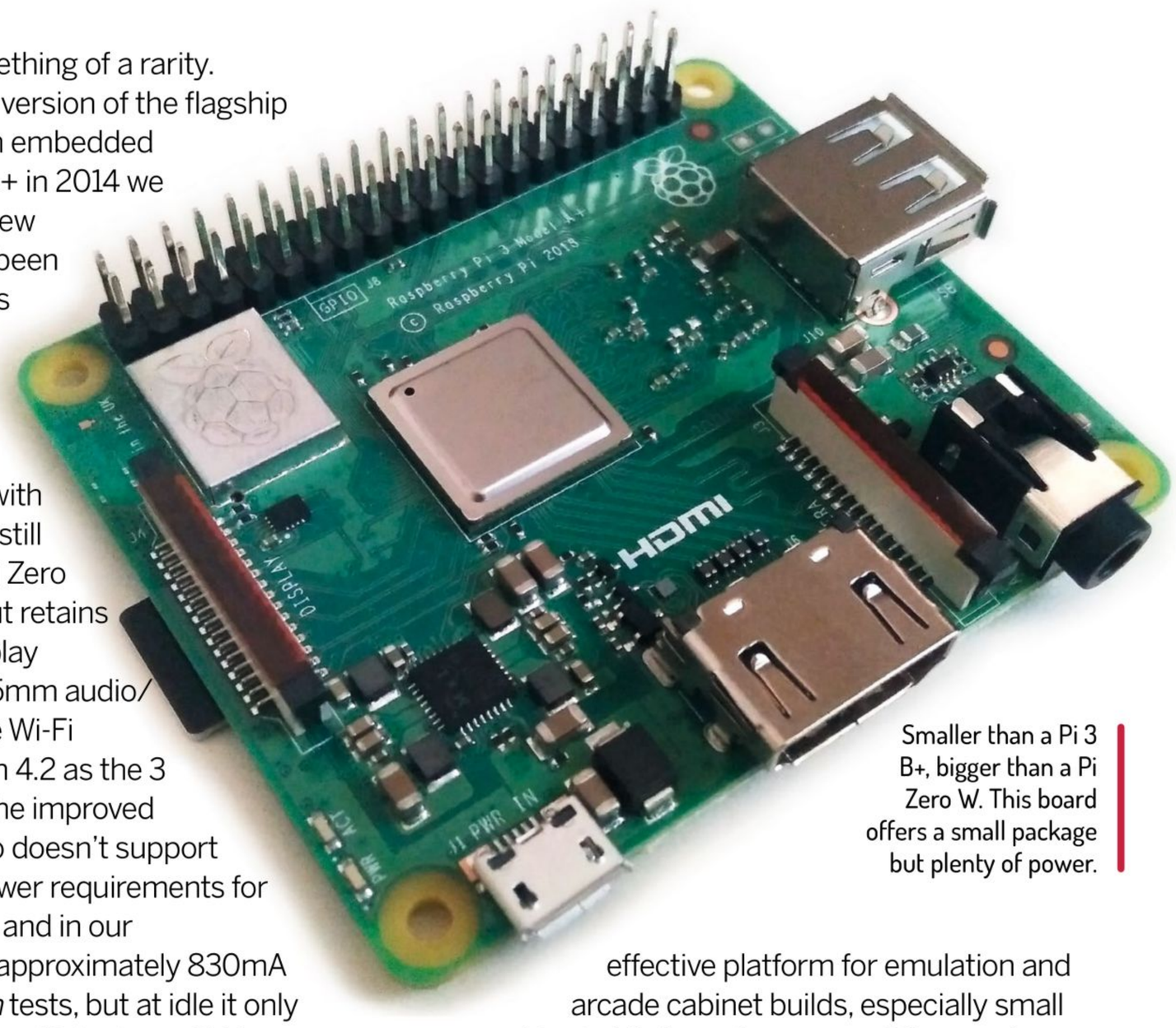
Les Pounder sits down for another slice of "classic" Raspberry Pi, the last of the current generation. Does it still provide the nourishment that we crave?

SPECS

- **CPU:** 1.4GHz quad-core Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit
- **GPU:** VideoCore IV, 400MHz
- **RAM:** 512Mb DDR2
- **Comm:** 2.4/5GHz 802.11b/g/n/ac, Bluetooth 4.2/ BLE
- **Ports:** HDMI v1.3a, 1x USB 2.0, 3.5mm four-pole stereo audio/ composite, CSI camera, DSI display, 40-pin GPIO
- **Storage:** Micro SD
- **Size & weight:** 65x56x11mm, 29g

The Model A Raspberry Pi is something of a rarity. Originally released as a cut-down version of the flagship model B, the A found a home with embedded projects. With the release of the A+ in 2014 we saw the new 40-pin GPIO and a new shape, but since then there have been no more A+ models. So it came as quite a surprise when the Raspberry Pi 3 A+ arrived.

The 3 A+ offers the same 1.4GHz 64-bit quad-core ARM Cortex-A53 CPU as the 3 B+ but with only half the RAM (512MB), yet is still more powerful than the smaller Pi Zero W. It has just a single USB port, but retains the 40-pin GPIO, camera and display connectors plus the combined 3.5mm audio/ video jack. The 3 A+ has the same Wi-Fi (2.4GHz and 5GHz) and Bluetooth 4.2 as the 3 B+. The 3 A+ doesn't come with the improved Ethernet port, which means it also doesn't support the recently released PoE HAT. Power requirements for the 3 A+ are the typical 5V supply and in our benchmarks the 3 A+ consumed approximately 830mA when under load for our *sysbench* tests, but at idle it only used 220mA. So this board isn't as efficient as a Pi Zero, but it has similar efficiency compared to the 3 B+.



Smaller than a Pi 3 B+, bigger than a Pi Zero W. This board offers a small package but plenty of power.

The test results are in...

We ran a few benchmarks and our traditional *sysbench* prime number test came in at 30.0373 seconds. That's a full 1.8 seconds faster than the Pi 3 B+ we tested earlier this year. But this difference will be due to improvements made to the Raspbian operating system. In our *Octane* test which is used to benchmark JavaScript performance we saw a score of 2,538 for the 3 A+. That's less than the 2,780 of the 3 B+ and that will be chiefly caused by the lack of RAM, but not a massive difference. But compared to our Pi Zero W score of 456 the 3 A+ is 5.5x faster! Our network tests connecting via a 5GHz router to an ethernet-connected laptop gave us a transfer rate of 96.6Mbits/sec, compared to 74.7Mbits/sec on the 3 B+. Again, this difference is most likely due to software updates.

To use the Raspberry Pi 3 A+ you'll need to download and flash the latest Raspbian operating system, as older versions of Raspbian won't boot using the new board.

But where does the 3 A+ sit in the range? Well, the Pi Zero is primarily used for embedded projects and robotics, and the model B boards are used for development and advanced projects. But the A+ offers the small size needed for embedded projects and robotics, but the power for advanced projects. So projects such as Google Assistant, Alexa, machine learning and OpenCV have the CPU grunt needed for smooth applications in a much smaller package. The 3A+ would also be an awesome cost-effective platform for emulation and arcade cabinet builds, especially small cabinets. We have the power of the quad core CPU and the small footprint of the 3 A+, but it may take a few days for other distributions to catch up.

The Raspberry Pi 3 A+ offers a lot for £25. Sure, there are compromises but these are outweighed by the benefits of more power in a smaller package.

Eben Upton has confirmed that the 3 A+ will be the last of the "classic" Raspberry Pi platform; future boards will be based on newer and more powerful hardware. The 3 A+ is a fitting swansong for what's been a remarkably disruptive platform. With that in mind we wonder how the Raspberry Pi 4 will continue the disruption.

VERDICT

DEVELOPER: Raspberry Pi Trading
WEB: www.raspberrypi.org/products/raspberry-pi-3-model-a-plus
PRICE: £25

FEATURES	9/10	EASE OF USE	9/10
PERFORMANCE	9/10	VALUE	9/10

The long awaited A+ is back and it is the perfect end to the first chapter of the Raspberry Pi. We finally have power in a compact package – what will you do with it?

» **Rating 9/10**

Compute Module 3+

Les Pounder takes a look at the last of this generation's Raspberry Pi and wonders what makers can do with this much power.

IN BRIEF

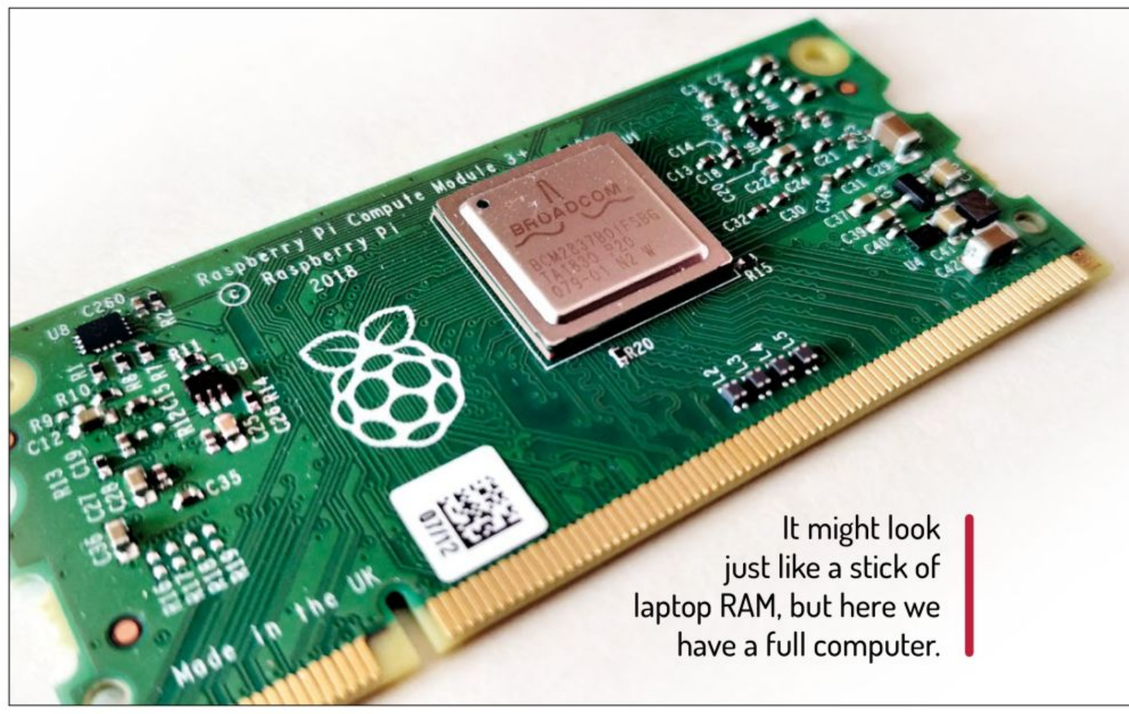
A Raspberry Pi 3 B+ shrunk into the shape of laptop RAM. This tiny board has plenty of power but can be built into bespoke projects designed by the user. If you are building a Raspberry Pi project and want to scale up into a product, or integrate the Pi into an industrial application, this board is for you.

The Raspberry Pi Compute Module is one of those boards seen in the maker community, but rarely used in projects for most makers. It's designed for industrial use and this is evident by the form factor. Measuring only 67.6 x 31mm and looks more like a stick of laptop DDR RAM (but not electrically compatible), this is actually a Pi 3 B+ shrunk down into the smallest form. It even has 1GB of RAM, the current maximum for this generation of Raspberry Pi.

But what do we lose in the process? Well, chiefly wireless connectivity: there is no Wi-Fi or Bluetooth with this board. Another loss is that the CPU for the CM3+ runs at 1.2GHz and not the full 1.4GHz found on the 3 B+.On the other hand, we gain eMMC storage in 8, 16 and 32GB capacities – except for the CM3+ Lite, which relies on offboard micro SD card storage. The eMMC storage is connected directly to the processor and gives us an interface speed similar to a Class 10 micro SD card. This is not lightning fast access, but quick enough and less prone to corruption than an SD card.

The BCM2837B0 System on a Chip (SoC) is the same as found on the 3 B+ and 3 A+, but the CM3+ can offer up all of the GPIO pins present for this SoC – all 120 of them! To do this the CM3+, like its earlier versions, requires the Compute Module IO board. This board provides us with a single USB 2.0 port to connect a hub, keyboard, mouse and so on. There are also connections for the official Raspberry Pi camera and display – in fact, two connections for each device, so dual screens and dual camera projects are possible. The CMIO board also provides a micro SD card slot needed for the CM3+ Lite, and an HDMI connector.

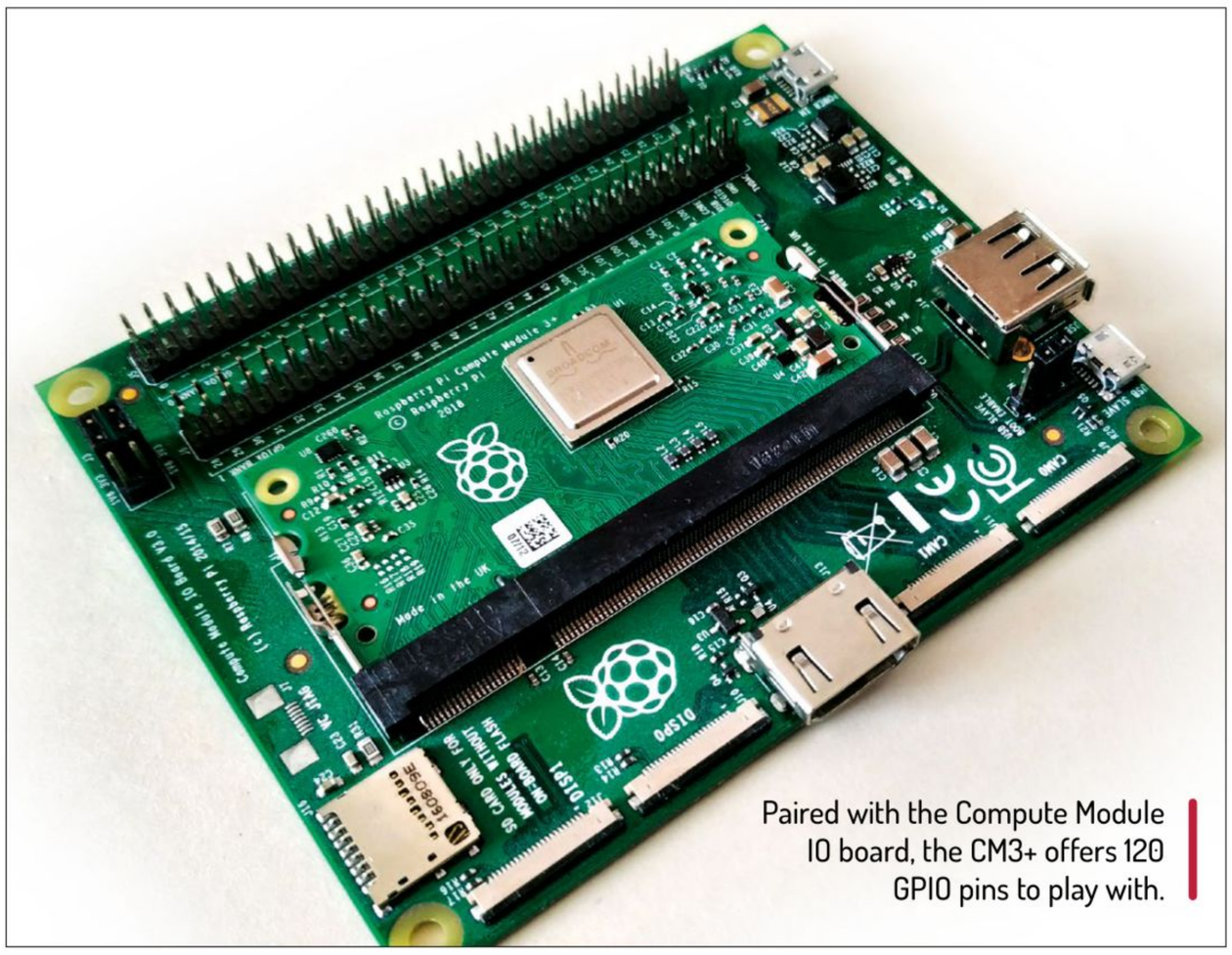
Flashing the operating system to the eMMC is a multi-step process, which is not as simple as a regular Pi, but still straightforward. The first step is to clone a GitHub repository, which includes the tools necessary to perform a USB boot of the CM3+. This enables the CM3+ to be mounted like a regular flash drive. Then we can either use a tool such as *dd* to write a Raspbian image, or in our case *Etcher*, which detected the CM3+ and wrote the image in around 10 minutes.



It might look just like a stick of laptop RAM, but here we have a full computer.

So what do we do with the CM3+? This is where the fun begins. We can design our own circuit boards based around the board. For example we could build a CM3+ powered museum exhibit with touchscreen controls, multimedia playback and motorised elements. We have access to all of the GPIO present on the SoC; all we need is the time and patience to create a carrier board, like the CMIO, that breaks out the connections that we need.

This board isn't for everyone; in fact, it's only really viable for building Internet of Things, robotics and industrial applications. For the average user it's more of an interesting piece of Pi-powered technology. But for the few who want to create their own Pi-powered products without the bulk of the form factor, the CM3+ is an interesting and expansive platform, and cost-effective.



Paired with the Compute Module IO board, the CM3+ offers 120 GPIO pins to play with.

VERDICT

DEVELOPER: Raspberry Pi Trading
WEB: www.raspberrypi.org/products/compute-module-3-plus-32gb
PRICE: £38

FEATURES	8/10	EASE OF USE	7/10
PERFORMANCE	8/10	VALUE	9/10

The swansong for this generation of Raspberry Pi is a capable yet bespoke piece of kit. For those that want to create bespoke products powered by Pi, this is your board.

» **Rating 8/10**

Manjaro Arm v8

Les Pounder takes a look at an alternative operating system for the Raspberry Pi – can it topple the mighty Raspbian?

IN BRIEF

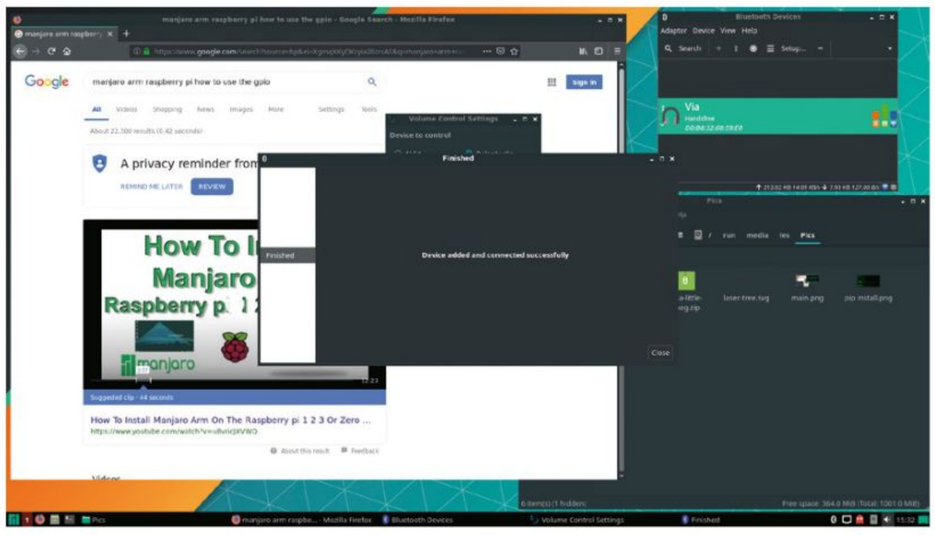
An Arch-based distribution that offers the look and feel of a full-blown Linux distro for single board computers such as the Raspberry Pi. It's easy to install, but offers flexibility and choice during the installation process. Nevertheless this is not a distro for first-time Linux users, which is a shame as it looks so inviting!

We really wanted to love this distro. Installation and configuration was a little more involved than a typical Raspbian install, but it offered so much hope. The first boot was longer than Raspbian's – almost a full minute in fact – but that's OK as it had a lot to do. We waited patiently and then it appeared as if everything had stopped. Was it a failed install? Did we need to flash again? Thankfully not, we thought, as the login screen eventually appeared.

Logging in, we saw the lovely Manjaro LXQt desktop, and everything looked great – the menu and system tray looked just like a proper Linux desktop. We hoped that this trend would continue, so we connected to the internet via Ethernet and opened *Firefox* to a few websites. But we were hit with security warnings for certificates, even for known-working sites. After a little head-scratching we noticed that the time and date were off, and by quite some margin. Setting the time and date using NTP resolved this issue, but it really shouldn't have been an issue to start with.

Firefox then worked quite well and we were able to visit a few sites and watch a YouTube video, which was a little slow but acceptable. For audio, we connected a Bluetooth speaker and after some tinkering in the config – and trial and error with the Bluetooth application – we managed to get audio playback working. Looking in the system tray we noticed that *pacman* was present, and indicated that we had 172 updates to perform using *OctoPi*, a frontend for *pacman*. We triggered the updates, and patiently waited for some time.

Returning to the update after 20 minutes we noticed that it had stalled; in fact, the system was frozen, forcing us to do a hard reset. The system rebooted OK and after logging back in we ran an update from the terminal. This too stalled, forcing us to again reboot. But this time the system failed to start, so we flashed a fresh Manjaro ARM install. This time we skipped the update for fear of wrecking the microSD card…

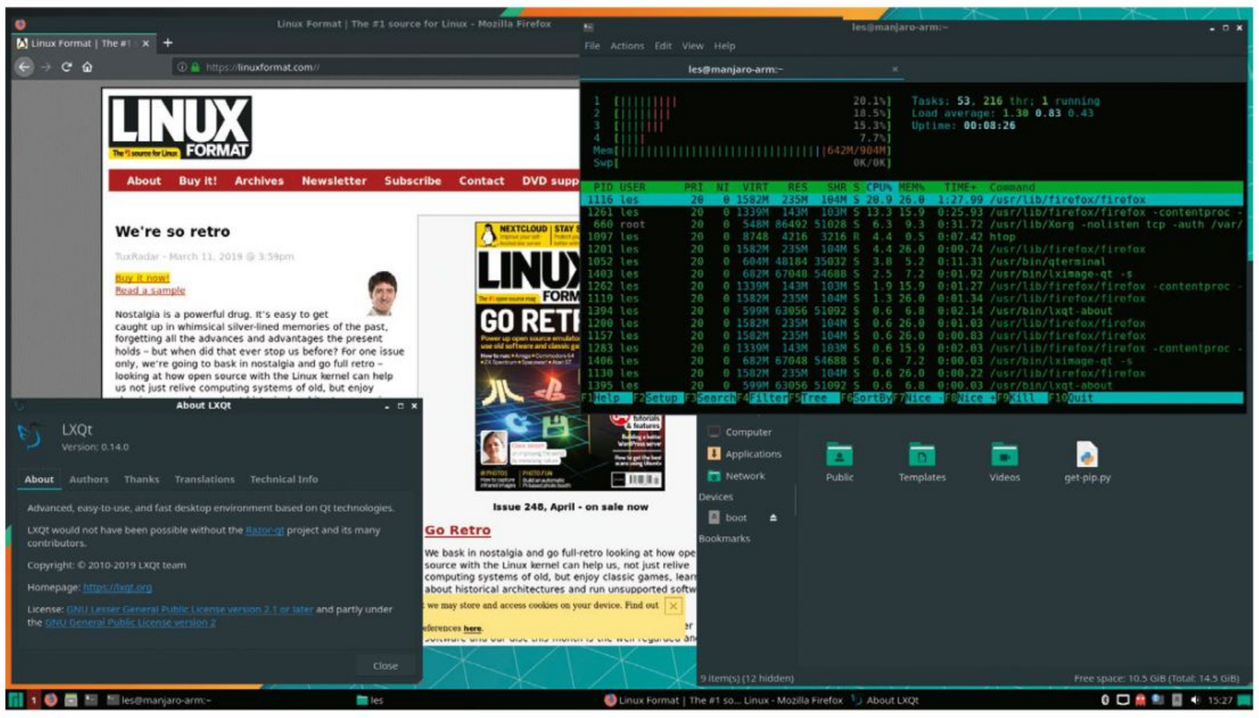


You might eventually get things working after a fashion, but the number of errors we encountered was just frustrating.

We wanted to use Manjaro ARM to control the GPIO of our Raspberry Pi, so with Python 3 already installed we chose to install the GPIO Zero library to control the GPIO. But *pip* wasn't installed, and there was no installation candidate from the *pacman* archives. No problem there: a workaround is to use the *getpip.py* command, and this installed *pip*, enabling us to install GPIO Zero. Hurrah, we thought, and quickly connected an LED to test.

We wrote some test code and then ran the code in the terminal, to a screen full of errors. Turns out that the dependencies were not installed and we had to manually install a pin factory (used by GPIO Zero to control the pins). After tracking one down and trying to install… well, it failed. We abandoned the GPIO test and plugged in our USB flash drive to save screenshots, but noticed that our drive was missing. We found an app in the system tray that mounted our drive and this worked quickly, enabling us to transfer our screenshots. Wi-Fi connectivity is also handled from the system tray and worked well.

Manjaro ARM is a proper Linux distro for the Pi, but the Pi is just too slow (even our Pi 3B+), and there is too little to tempt users to switch from Raspbian.



Manjaro ARM looks stunning and offers a proper Linux distro experience. But the Pi is too slow and the distro too fragile for general use.

VERDICT

DEVELOPER: Manjaro Arm
WEB: http://manjaro-arm.org
LICENCE: GPL

FEATURES	6/10	EASE OF USE	6/10
PERFORMANCE	3/10	DOCUMENTATION	6/10

For general users looking to try something other than Raspbian, look elsewhere.

» Rating 5/10

Raspbian 11.2018

Les Pounder tests the latest version of the official Raspberry Pi operating system. After seven years since its release, does it still warrant praise?

IN BRIEF

The official Pi distro is compatible with every model of Raspberry Pi, including the first model released back in 2012. This latest release sees the distro become "grown up" and multi-purpose while retaining the rich heritage it's built up over the years. A new post-installation setup tool helps new users configure and setup their Pi without using the terminal.

Raspbian in 2012 was a very different experience. Released as a means to interact with the then-new Raspberry Pi, Raspbian was one of two choices, the other being a Pi-centric version of Fedora. But over the years Raspbian has evolved from a barebones hack into a desktop operating system. In November 2018 the latest version was released, and Raspbian has finally come of age while still offering one operating system for all models of Pi.

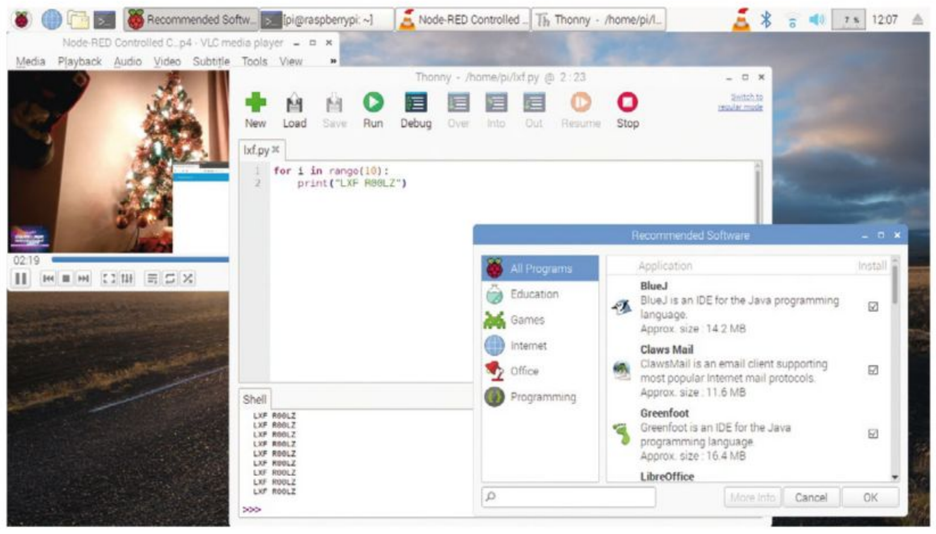
The latest Raspbian offers a helpful first boot setup system, where we can set our country. This is necessary for Wi-Fi and the keyboard layout. We can also change the main password, which is crucial if your Pi is going online and you don't want it to be part of a bot-net. We can set up Wi-Fi and then update the system all without using the terminal. This is a great feature for new users, but don't worry, we can still use the terminal! We can also install software using the recommended software application: an app store for Pi programs.

Another new feature is *VLC*, the venerable Swiss Army Knife of media playback. *VLC* is now the default media player and we're happy to report that it "just works". Better still, videos enjoy hardware acceleration thanks to the VideoCore GPU on the Pi. But if you want to playback MPEG 2 or VC-1 files you'll need to buy a cheap licence from the Raspberry Pi shop.

Get coding with Thonny

While not new to this release we have to mention *Thonny*, an easy-to-use Python 3 IDE that comes in two modes. There's a simple interface for those new to Python, and a regular interface for experienced Pythonistas. These modes can be switched in-app, which is a new feature for Raspbian, but you need to restart the tool. This isn't ideal, but better than in previous releases. *Thonny* is a much better IDE than *IDLE*, especially for newcomers. It even includes tools to install Python modules using *pip*, the Python packaging tool, via a GUI.

We also spotted *SmartSim*, which is a free and open source digital logic circuit design and simulation package.



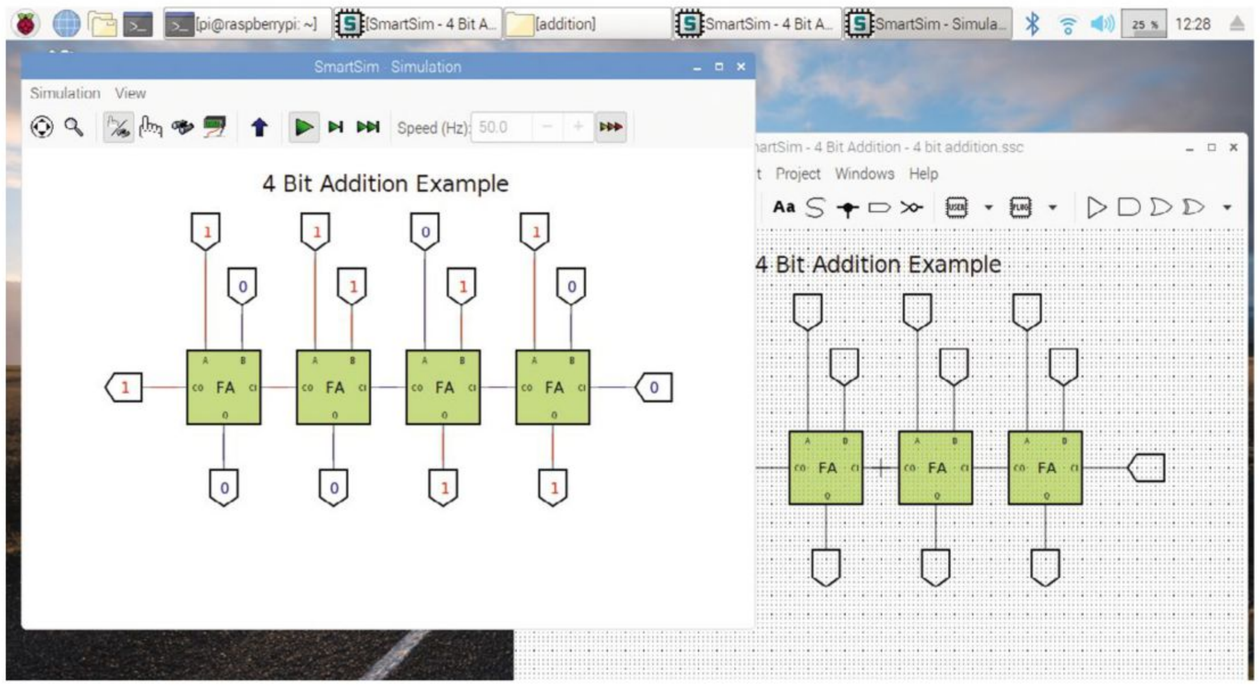
Raspbian is the official Raspberry Pi distro and it's seen many refinements thanks to Simon Long and his team.

In this application it's possible to simulate digital logic and create circuits purely in software – there's no need for any hardware. It can even be used to design and test 32-bit processors.

Something new for this release of Raspbian is that it comes in a new lighter download. Typically, Raspbian Stretch with desktop and all applications is a 1.84GB download, and uses around 7GB of space when written to an SD card. But the new Raspbian Stretch with Desktop is only around 1GB to download and when installed it only uses around 3.5GB of space. So if you need the Raspbian desktop, but not the bloat then you have that option. If you need an ultralight install then Raspbian Stretch Lite offers a terminal-only option, which install in around 1GB of space.

If you were waiting for a fully featured distro for your Pi, then the time has come. Raspbian has been with us since day one, but it is only now that we can consider it a distro for all. If you need a simple distro for YouTube, *LibreOffice* and coding then this is it. If you need a fully featured distro for hardware hacks and tinkering, this is it. If you're building embedded projects for use in the field, this is it. The year (*NO! – Ed*) of the Linux (*Don't say it! – Ed*) desktop (*<sigh> – Ed*) is already upon us.

SmartSim was created by Ashley Newson, created when he was only 16 as part of his A Level computing studies course.



VERDICT

DEVELOPER: Raspberry Pi Trading
WEB: www.raspberrypi.org
LICENCE: Raspberry Pi Bundle License

FEATURES	9/10	EASE OF USE	9/10
PERFORMANCE	9/10	DOCUMENTATION	9/10

Simple to install, use and now configure on boot. This is the best experience for your Raspberry Pi projects. It has many thousands of projects and tutorials based upon its legacy.

» **Rating 9/10**

Optimise your Pi for better performance

Christian Cawley presents his tips and tricks for squeezing as much juice out of your Raspberry Pi as possible.

OUR EXPERT

Christian Cawley has spent the past 15 years telling people how to do the same computing tasks but still has no idea how cars work.

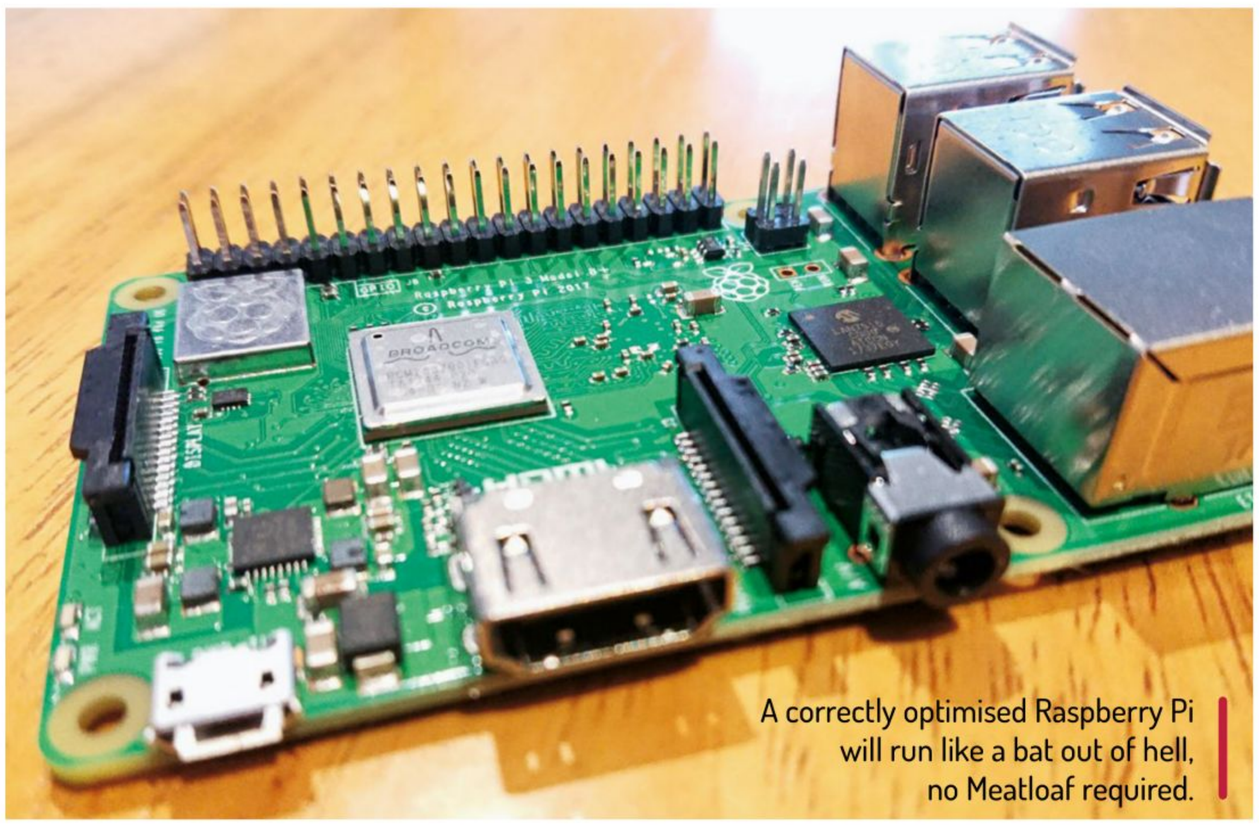
Has your Raspberry Pi slowed down? It's a common problem – it doesn't matter which version of the Pi you're using, performance always suffers if the operating system isn't correctly configured. Using a lightweight operating system such as Raspbian Lite, DietPi, or piCore can help, but there's much more you can do. Performance boosts can be gained from overclocking, changing how RAM is used, and ditching bloatware, among other things.

Power rangers

The wrong power supply can wreak havoc with your Raspberry Pi's performance. With insufficient power to deal with processing requirements, bottlenecks can occur, slowing the computer right down. Low power can also result in read/write issues, which in turn can damage the microSD card (more on that later). Then there's the usual end result of a frozen Raspberry Pi: pulling the power lead and rebooting.

To avoid this, you need to give the Raspberry Pi a reliable power supply. This means using a good-quality PSU (if not the official device) rather than a spare USB port on your PC, TV or power bar.

The official Raspberry Pi power adapter has 100-240V input, and 5.1V 2.5A output. If you're not using the official PSU, you should ensure that your supply matches this, and that the micro USB cable is of good quality, in working order, and fits the Raspberry Pi correctly.



A correctly optimised Raspberry Pi will run like a bat out of hell, no Meatloaf required.

Super-size Pi

Using Raspbian Lite or some other lightweight distro is a good option for a stripped-back experience, but if you're looking for a more feature-packed operating system, the standard Raspbian installation is your best option.

A full gigabyte of storage space can be freed up by removing unused tools such as *LibreOffice*, the *Wolfram Engine*, *Minecraft* and *Sonic Pi*. To benefit from this fully, however, you need to have the full capacity of the SD card. This means using the Expand Filesystem option in the *raspi-config* tool, or the Raspberry Pi Configuration menu on the desktop. You'll find this on the System tab.

Removing what might be unfairly termed "bloatware" on Raspbian is easy. Open a Terminal prompt and type

```
sudo apt purge libreoffice*
sudo apt clean
sudo apt autoremove
```

This will remove *LibreOffice* from Raspbian. Replace the `libreoffice*` command with `wolfram-engine`, `minecraft-pi` and `sonic-pi` to remove those tools.

With these four options removed, you'll have the advantage of increased storage space on your Pi without having the limitations of a lite distro.

Kill JavaScript dead

While it's wise to limit the number of tabs you have open in your preferred browser, further performance boosts can be gained by disabling JavaScript. How you do this will vary. In *Chromium*, use Menu>Settings>Advanced>Privacy and security>Content settings>JavaScript. Here, switch Allowed to Blocked.

Disabling JavaScript can improve overall performance, as well as speed up some any pages that rely on it. While this might mean pages become less functional, it will also help to reduce the load on your Raspberry Pi from adverts and trackers. Note that exceptions can be added to your browser settings to allow JavaScript to run on specific sites.

Underclock, overclock, Pi-clocking's free

Massive performance boosts can be enjoyed by your Raspberry Pi by overclocking the processor. Raspbian

offers this as a built-in feature that can be found in the *raspi-config* tool (other distros may require additional software). Access the overclocking tool by selecting Overclock on the main *raspi-config* menu. Depending on your Raspberry Pi model, you'll have several options here. The default option is None, with the clock running at the basic 700MHz, but you could ramp things up to 1000MHz and above with Turbo mode.

Using the overclocking menu means you can avoid having to worry too much about stress testing as the values on offer have already been tested. However, Turbo mode might cause problems, so it's a good idea to use a cooling solution (see Quick Tip, *right*).

With a Raspberry Pi 3 and above, you can also edit the **config.txt** file in *nano* to specify values for the four overclock settings: arm_freq, core_freq, sdram_freq, and over_voltage. The usual overclocking rules of taking a baseline reading and increasing incrementally apply.

The benefits of overclocking your Raspberry Pi include increased performance for video decoding in *Kodi* or playing more games from the fifth and sixth generation consoles that can currently be emulated on the Pi. Emulation of Windows apps with the *ExaGear* software is also improved.

Ditch the cheapo SD card

Finding the right microSD card for your Raspberry Pi can be tricky, but with the right capacity and speed it can make a huge difference to how your Pi runs.

The secret here is to stick to named brands such as SanDisk, Samsung and Kingston, while keeping an eye on the card's speed and rating. A higher rating indicates better speed and error connection, which will improve your Pi's performance.

Once you've added a quality SD card, it's time to start looking after it. By know you should know how to correctly shutdown a Raspberry Pi. Doing so will prolong the lifespan of the SD card, saving you further outlay (however inexpensive a replacement might be) and the fuss of setting up your Pi again from scratch.

From the desktop environment, be sure to shutdown from the main menu. If you're using a *Kodi* media centre distribution or a retro gaming suite, take the time to find the shutdown option and use this.

If it's quicker, or you're connected to your Pi over SSH, it's easier to shut down from the shell:

```
sudo shutdown -h now
```

You can also schedule a time to shut down the Pi:

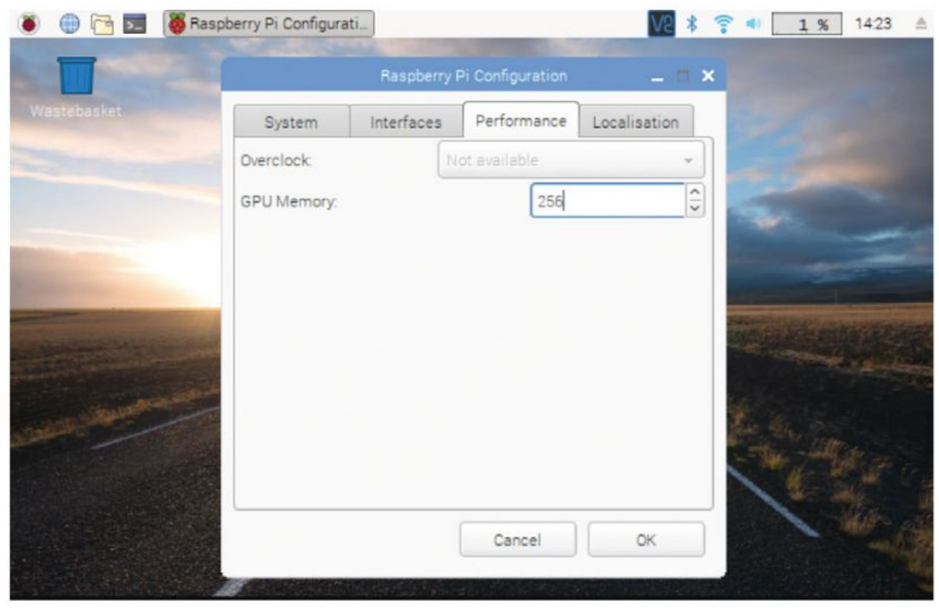
```
sudo shutdown -h 02:46
```

An orderly shutdown will ensure your SD card's integrity is maintained, as well as avoid data loss.

Zippy ZRAM

RAM configuration options for the Raspberry Pi mean that you can easily designate a portion of RAM to the GPU if necessary or reserve it for non-graphics tasks.

Using the Advanced Options sub-menu in *raspi-config*, find the Memory Split option and use this to specify how much memory the GPU should have. For example, if you're not using a desktop environment, then you can set this to 16MB, allowing the Raspberry Pi's processor to benefit from extra memory. For



Tweak the Raspberry Pi's memory split to assign more RAM to the CPU or the GPU, whichever takes your pick.

graphic-intensive tasks, however, your GPU will need more of the RAM. Note that even with a Raspberry Pi 3 B+ (which has 1GB of RAM), you'll be unable to assign more than 256MB to the GPU.

Perhaps the best performance tweak for the Raspberry Pi can be made to the RAM, utilising unused space for swap data.

As explained by YouTuber NovaSpirit Tech, ZRAM can be used as a swap disk. While the data won't persist following a reboot, this can speed your Raspberry Pi up considerably.

Begin by grabbing the script from GitHub:

```
sudo wget -O /usr/bin/zram.sh https://raw.
githubusercontent.com/novaspirit/rpi_zram/master/
zram.sh
```

Next, make the file executable and edit **rc.local** to run the script when you reboot:

```
sudo chmod +x /usr/bin/zram.sh
sudo nano /etc/rc.local
```

Look for the exit 0 line, and above it add:

```
/usr/bin/zram.sh &
```

Hit Ctrl+X to save and exit the file. ZRAM swapping is far faster than relying on even the fastest microSD card, with an instant performance boost.

QUICK TIP

Prolong performance boosts from overclocking by installing heatsinks on the SoC and network controller chips. For the Raspberry Pi 3B+, an onboard fan cooling solution is available for under £10.

» DON'T PUSH YOUR Pi

As versatile as the Raspberry Pi is, it's a good idea to be certain that it's the right device for the project you have in mind.

You might just be using the Raspberry Pi for retro gaming or *Kodi*. It might be acting as a desktop replacement, or as the main component of an IoT project. If your Raspberry Pi is up to the task, then it should work fine. The problem is when you choose a task that the Pi just can't do, even when the hardware appears to be a good fit.

Conversely, the Raspberry Pi might be far too advanced for the task you have in mind, such as flashing an LED over the web, or something equally low-spec.

Perhaps there isn't quite enough RAM; or maybe the issue is with the ARM architecture. Whatever the problem, all the configuration tweaks and tricks in the world won't help if the hardware can't handle the project.

Even the Raspberry Pi 3 B+ isn't powerful enough for some situations. So, what's the alternative? Well, you could look at some of the Pi's nearest rivals, such as the ASUS Tinker Board, Pine64, or even Odroid-XU4. Rather than trying to do everything with the Raspberry Pi, choose the right device for the job.

Tweak your Raspbian config to perfection

Helping you to perfect your Pi project, **Christian Cawley** explains the best way to customise your Raspbian config file.

OUR EXPERT

Christian Cawley has more Pis than you've had hot dinners, probably.

The chances are that when you flash a new image to your Raspberry Pi's microSD card, you're doing it wrong. While the Pi's default desktop experience in Raspbian has improved, the configuration side of things hasn't really matured as much as it could have. The result is you booting up with a keyboard, mouse and monitor attached, and setting up wireless networking, SSH and various other options.

But while the pre-boot configuration could be better, it isn't *completely* absent. A Raspberry Pi running Raspbian Stretch – and some other operating systems, but we're concentrating on Raspbian – can be configured while the microSD card is still inserted in your PC. Wireless networking and SSH can be pre-configured, as can settings for the display, memory and even overclocking.

This is thanks to the Pi's **/boot** directory, where files for the wireless configuration and SSH can be created. It's also where you'll find **config.txt**, a text file that stores configuration information for the Raspberry Pi as it boots the operating system.

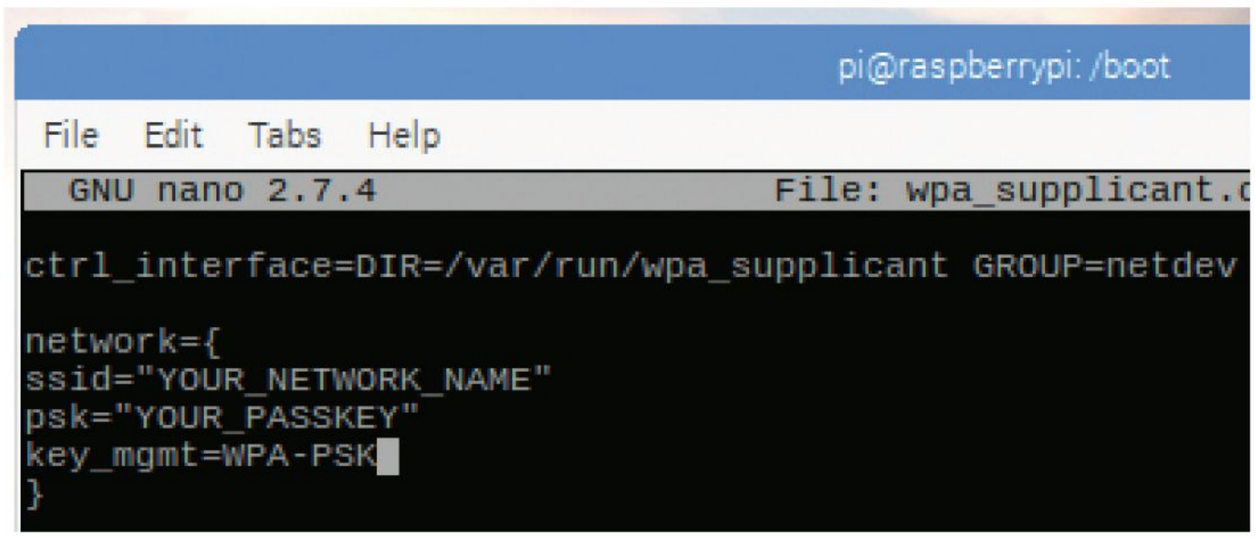
Ready to pre-configure your Raspbian installation? After writing the operating system disk image file to microSD, browse to the **/boot** folder in your file manager or Terminal. Here, create a new file in *nano*:

```
sudo nano wpa_supplicant.conf
```

Copy the following into the file:

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
network={
ssid="YOUR_NETWORK_NAME"
psk="YOUR_PASSKEY"
key_mgmt=WPA-PSK
}
```

The wpa_supplicant.conf file lets you easily set up wireless networking on your Raspberry Pi before you've even switched it on!



This basic script is straightforward. Edit the SSID so that it matches the name of your wireless network. Similarly, set PSK to your network's password, then press Ctrl+X to save and exit the file. When the Raspberry Pi boots up, these details will be detected and automatically loaded into Raspbian's wireless settings. Your Raspberry Pi will be online almost as soon as it boots.

Another easy configuration option for Raspbian is to set up SSH access before you've booted your Pi. The benefits of this are considerable. Being able to boot a Raspberry Pi, have it connected to your network, and be able to remotely access the computer saves a lot of messing about. There's no need to hook up a keyboard, mouse, and display – the Pi will be ready for action.

To enable SSH, once again open your file manager or terminal emulator in the **/boot** directory. Here, create a new file, simply labelled **ssh**:

```
sudo touch ssh
```

That's all you need. When the Raspberry Pi boots, Raspbian will detect the **ssh** file and take this as an instruction to enable SSH access. Any SSH app, whether it's an FTP app with SSH support or a command line tool, can then securely connect to your Raspberry Pi over Wi-Fi.

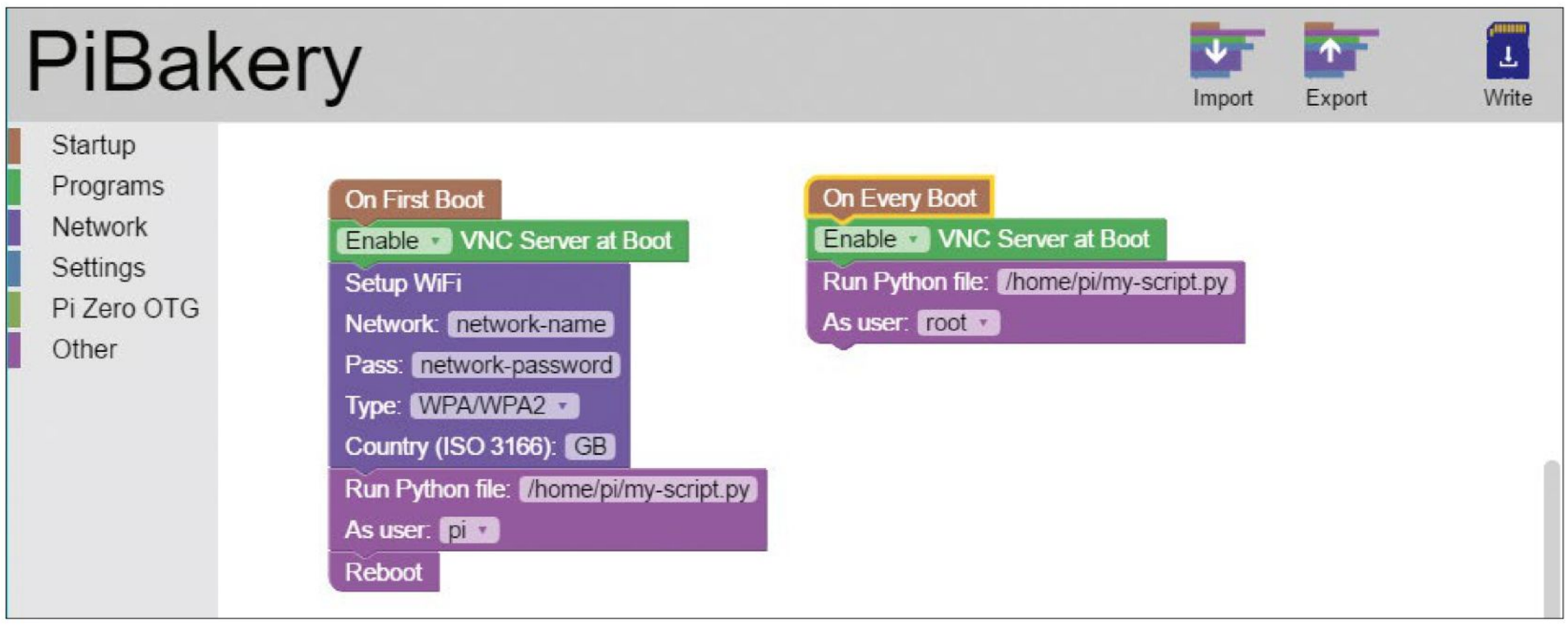
Config your config.txt

If you're looking for a greater selection of pre-boot tweaks, you'll need to look at the **config.txt** file, also in **boot**. This is a list of configurations organised using the basic format of a `property=value` statement, all on one line. For example, to set the HDMI cable to a display as the default output for audio from the Pi, you would use

```
hdmi_drive=2
```

You have three main options for editing **config.txt**. The first is from within Raspbian while your Raspberry Pi is running. However, any changes made to the file will not be applied until you reboot. You may also remotely access **config.txt** via SSH, editing it in *nano* or whatever text editor you prefer. Finally, of course, you can edit it on your desktop computer.

This can be done before you boot Raspbian for the first time, or at any point in future. All manner of settings can be adjusted, from audio and video options



Configure Raspbian with ease using the block-based user interface of PiBakery, before 'baking' your changes to a disk image and writing it to the SD card.

to the camera, GPIO, and beyond. Before proceeding, however, it's a good idea to make a copy of the **config.txt** file. This will ensure that you can overcome any issues that may transpire. For example, specifying the wrong setting when attempting to overclock your Pi will result in the computer not booting. By restoring the original file, you can easily resolve this.

So prior to editing **config.txt**, make a copy with *mv*:

```
mv config.txt backupconfig.txt
```

To restore the backup use *rm* to remove the new config.txt, then reverse the previous command:

```
rm config.txt
mv backupconfig.txt config.txt
```

With the **config.txt** file open, you'll see that many lines of text start with a `#` comment symbol. This prevents the lines being read by the operating system; removing the hash 'activates' the command. It's a useful method of leaving notes in the file, too, to explain what each group of commands do.

You now have a host of settings that can be adjusted. Most of these will not need changing, however; the Raspbian developers have configured the operating system to be as useful as possible for the widest set of parameters. However, some systems need a bit of individuality. For example, you might be building a project that requires the GPIO pins to be reconfigured. Fortunately, this is possible under the GPIO section of **config.txt**.

For example, to configure pins 12 to 16 as inputs, use the `gpio` directive, along with the `ip` command (standing for 'input'):

```
gpio=12-16=ip
```

Bear in mind that changes here can be overridden when Raspbian is running using *raspi-gpio*.

Usually the Pi's memory split is managed in *raspi-config*. However, this can be managed using the `gpu_mem` command in **config.txt**. You need only specify the amount of memory for the GPU: the remainder will be allocated to the system RAM. For example, using `gpu_mem_256` sets the GPU memory to 256MB. You can go as high as `gpu_mem_1024`, but note that if the Pi's total RAM is lower than the figure you're specifying, the command will not work. This does mean, however, that you can specify multiple `gpu_mem` values on an SD card that you plan to swap between multiple models with differing amounts of RAM.

With options for overclocking, kernel instructions, audio visual settings, and even including additional configuration files (useful if you have multiple changes you regularly make to the default **config.txt**), it's worth spending some time browsing the file to get a grip of just what pre-boot tweaks are possible in Raspbian

Baking your config

As straightforward as **config.txt** is, it isn't the most forgiving environment, and changes can have disastrous results. Happily, a tool called *PiBakery* is a user-friendly alternative. Available from **www.pibakery.org**, you can set this up on Linux from source, or install it on macOS or Windows. *PiBakery* comes with Raspbian (and Raspbian Lite) built in, and offers a sensible, Scratch-style block-based UI. Blocks are provided for different types of configuration; for example, Startup actions, Network settings, and more.

These can be dragged into place, individual parameters set, and then the whole thing is baked into Raspbian as the OS is written to your SD card. For example, using the On First Boot block enables easy initialising of your Wi-Fi connection, SSH configuration and even VNC server. These blocks can be dragged, dropped, configured and saved. Configurations can also be imported and exported thanks to XML support, making it the ultimate pre-configuration tool.

QUICK TIP

If you're using SSH to access your Raspberry Pi, it's wise to change the default password. Use the 'passwd pi' command to change the password for the Pi user.

» TRY PIBAKERY V2, TOO

Another option for pre-configuring your Raspberry Pi's installation is *PiBakery v2*, which includes a host of extra features.

For example, you're no longer limited to the *PiBakery* version of Raspbian. Any Raspbian-based IMG file can be used, potentially enabling easier setup of RetroPie, OSMC, and other distros. Like the original *PiBakery*, it supports Raspbian Lite and the full version. *PiBakery v2* can also handle editing of any Raspbian microSD card, rather than being limited to ones created with *PiBakery*.

Also on offer is improved Linux support (the original is limited to Windows and macOS unless you compile) and the ability to import XML files from previous versions. Multiple block sources can be added, enabling you to maintain your own custom blocks.

All of this means that all configuration of a Raspbian-based distro for your Raspberry Pi can be done from the comfort of your PC, meeting the requirements of the project you're planning.

Even if you're only looking for an easier way to set up Wi-Fi, *PiBakery v2* should prove to be the best option for anyone who wants to prepare and tweak Raspbian.

Make a Raspberry Pi 2 HTPC

Finally create a more powerful and capable HTPC using the Raspberry Pi 2 and the excellent OpenELEC project

We know people who just have a Raspberry Pi for XBMC, now called Kodi. It's a great idea and a great use for the Pi – it works just well enough that you can easily play media locally or over the network. The biggest issue came with GUI response on the original Model Bs, and a lack of USB ports for connecting up everything that you want.

While optimisation over the last few years has helped, the leap to Raspberry Pi 2 has basically solved all of these problems by giving you much more powerful hardware to play with. So if you're looking to upgrade or finally take the plunge, this handy guide will help you create the perfect Raspberry Pi 2 HTPC.

01 Choose the software

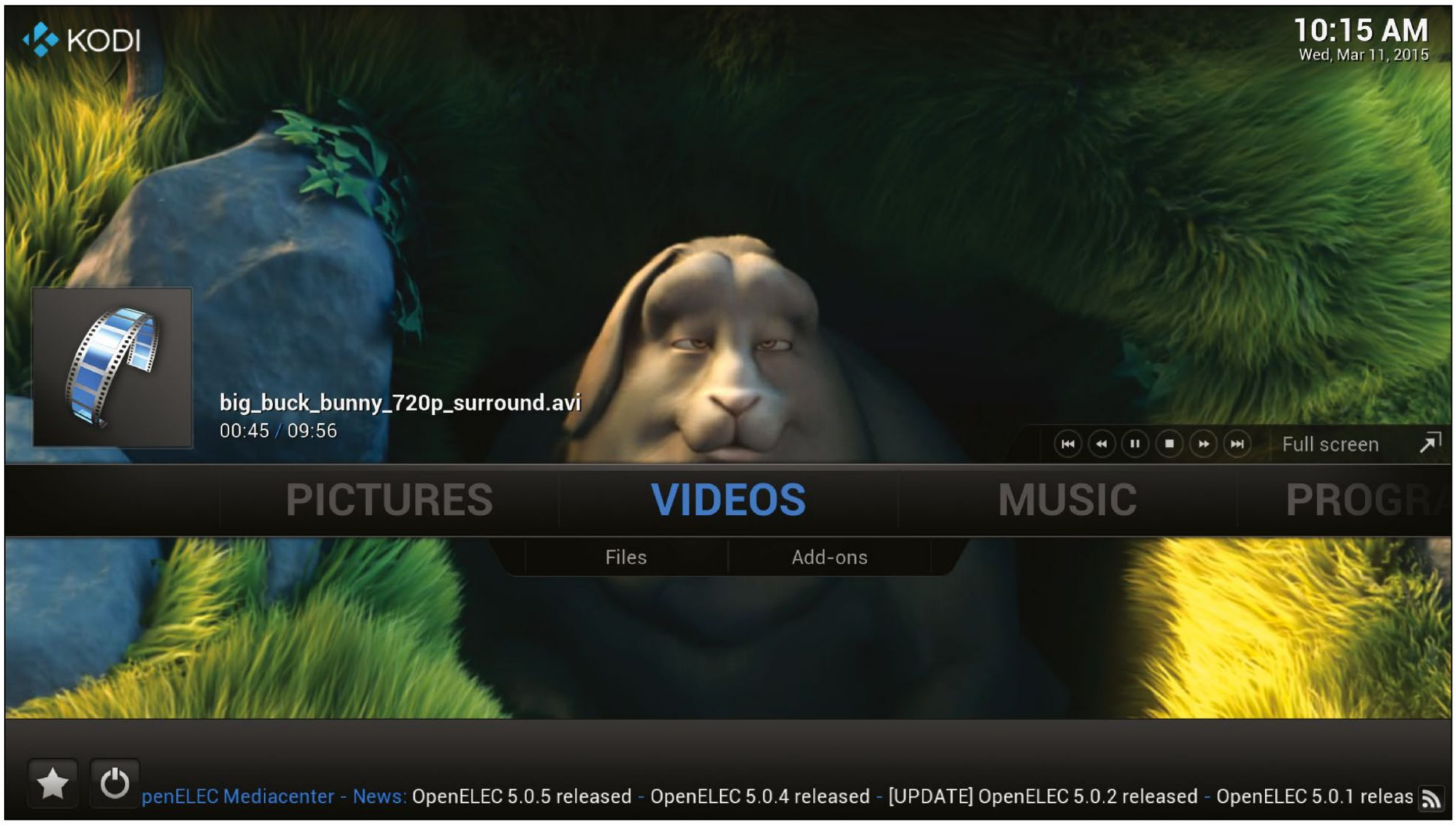
In the past, Pi HTPCs were just a choice between RaspBMC and OpenELEC. However, RaspBMC is on a bit of a hiatus and OpenELEC is your best bet for getting the most up-to-date software. There's not a massive difference between the two, as they both run XBMC.

02 Get the software

Head over to **openelec.tv** and look for the Download section. There's a specific Raspberry Pi section which is split up into original (ARMv6) Pi and the newer Raspberry Pi 2 (ARMv7). Grab the image file from this page for the Pi 2.

What you'll need

- OpenELEC openelec.tv
- HDMI cable
- USB IR receiver
- IR remote
- Case
- Dedicated power supply
- Optional USB storage



03 Install to card

Open up the terminal and use **fdisk -l** to determine where your SD card is located on your system. Something like /dev/sdb or /dev/mmcblk0 will be ideal. Navigate to the image using **cd** and install it with **dd** using:

```
$ dd bs=1M if=OpenELEC-RPi2.arm-5.0.5.img of=/dev/mmcblk0
```

04 First boot

Plug in your Raspberry Pi, either to your TV or to another screen just to begin with, and turn it on. OpenELEC will resize the SD card partitions and write a few extra programs before finally booting into Kodi.

05 Configure Kodi

Go through the basic wizard to get through the interface – if you are connecting via wireless you will need to go to OpenELEC in the System menu and activate the wireless receiver before selecting your network and then entering your password.

Above Kodi really is designed to be used with a remote, and there are some great guides to using them on the OpenELEC site: **bit.ly/1B0AERv**

06 Add network shares

You can stick a portable hard drive or USB stick into the Pi for storage, but the best method is really to stream over the network. Go to File manager under System and Add source. Go to Browse and choose your network protocol to browse the network or alternatively, add it manually.

07 Build your media centre

Placement of your Raspberry Pi is important. As it's going to be out all the time, we highly recommend getting a case for it – the Pibow cases from Pimoroni are quite well suited for this type of use as they are sturdy and can be attached to the rear of some TVs.

08 IR sensors and controllers

Kodi can be controlled with a number of different things – including USB game controllers and compatible IR sensors. We've used FLIRC in the past, but if you have your Pi behind the TV, you'll need a sensor on a wire that can stretch to a useful position.

09 Future updates

OpenELEC has the excellent ability to update itself without needing you to reinstall it every few months, meaning you won't need to do much maintenance on it at all. Now you can sit back and enjoy your media much easier than before.

Live TV

Kodi does have the ability to play live TV via a TV tuner, and you can also record stuff as well as long as you have the local memory. The main thing you'll need to invest in is a compatible TV tuner, a list of these is available here: **bit.ly/1r3mEVj**

Custom embedded Linux images

Tam Hanna guides you through creating a custom – and very small – Linux image for your Raspberry Pi using Yocto.

OUR EXPERT

Tam Hanna is a government certified electrical engineer with a fancy for process computers. One of his current projects involves an embedded Linux system used to track school buses.

The Yocto Project (Yocto) **www.yoctoproject.org** is a system that uses the Linux kernel and packages contributed from the OpenEmbedded software team. The Yocto team points out that its product is not a Linux distribution, but instead builds custom distributions; nevertheless, it can still can be extremely useful for creating a custom image.

The main contribution of the Yocto developers is a build system called *BitBake*. It fetches code from a variety of sources, can manage all kinds of compilation processes and can also take care of exporting the results in a ready-to-burn format.

A Yocto-based system consists of recipes and configuration files. Configuration files have a **.conf** suffix and contribute attributes which govern the build process. **.BB** files are called recipes, and are responsible for creating the actual components of the distribution; ultimately, running a recipe leads to the creation of a package.

Yocto differs from other embedded systems in that it divides its software stack into a set of layers. Images are created by combining layers, with *Bitbake* determining the compilation process. This approach might sound exotic, but it works well. For example, adding a new software stack such as Qt to an existing Yocto project can be accomplished by adding a reference to the **.bb** file to the configuration.

This ability is not only beneficial if you need to change the composition of the image. Projects based on Yocto are easy to maintain, as changes of the kernel and other things happen automatically. This does not mean Yocto is a silver bullet against all ills, though: if a specific program cannot be compiled for the target due to cross-platform problems, there is nothing *BitBake* can do about it.

```
for more information as common configuration options are commented.

You had no conf/bblayers.conf file. This configuration file has therefore been
created for you with some default values. To add additional metadata layers
into your configuration please add entries to conf/bblayers.conf.

The Yocto Project has extensive documentation about OE including a reference
manual which can be found at:
    http://yoctoproject.org/documentation

For more information about OpenEmbedded see their website:
    http://www.openembedded.org/


### Shell environment set up for builds. ###

You can now run 'bitbake <target>'

Common targets are:
    core-image-minimal
    core-image-sato
    meta-toolchain
    meta-ide-support

You can also run generated qemu images with a command like 'runqemu qemux86'
```

Yocto automatically determines when a new configuration has to be created.

Get baking

As with all other build systems, a relatively recent version of Linux is recommended – we'll use Ubuntu 18.04. Start-up involves downloading a batch of packages which are required for running the various tools contained in *Bitbake*:

```
~$ sudo apt-get install gawk wget git-core diffstat unzip
texinfo gcc-multilib \
build-essential chrpath socat cpio python python3
python3-pip python3-pexpect \
xz-utils debianutils iputils-ping libsdl1.2-dev xterm
```

Should you feel like using a non-Ubuntu distribution for compilation, see **http://bit.ly/lxf251yoctobuild** – the Yocto team is glad to provide package installation lists for all commonly used versions of desktop Linux.

In the next step, the actual Poky utilities need to be downloaded. This is best accomplished via GitHub:

```
~/yoctohouse$ git clone -b thud git://git.yoctoproject.
org/poky
```

When *git* has done its job, your workstation is the proud owner of the basic Poky distribution along with compilation tools. In the next step, a configuration needs to be created – it contains the various settings files which decide, among other things, the recipes required. Create a new configuration by sourcing the output of the *oe-init-build-env* utility:

```
~/yoctohouse$ cd poky
~/yoctohouse/poky$ source oe-init-build-env
```

Invoking the program for the first time leads to the output shown in screenshot *(left)*. Note that the current working directory of the terminal also was changed to the **home** folder of the newly created configuration. With that out of the way, the time is ripe for starting the compilation process. Change into the newly created folder, take a quick look around the **conf** folder to find both the configuration and recipe file, and fire off the compile command:

```
~/yoctohouse/poky/build$ bitbake core-image-sato
```

When invoking *BitBake*, a parameter describing the type of image is required. We instruct the creation of an image based on the Sato GUI stack – a simple graphical

user interface intended for resource-constrained devices. Unlike other build systems, *BitBake* is highly optimised, so don't worry if CPU usage is low at first – especially when being run for the first time, the tool needs to invest a bit of time downloading various packages from the internet. Once the initial downloading process has completed, CPU loads of 100 per cent are not unusual, even on machines with eight or 12 cores. On our machine, compiling a basic *BitBake* image takes more than three hours and downloads about 3GB of information.

When done, the product will inform you about the execution results:

```
NOTE: Tasks Summary: Attempted 5794 tasks of which 5 didn't need to be rerun and all succeeded.
Summary: There was 1 WARNING message shown.
```

Freshly downloaded property distributions are configured to create an image intended for the *QEMU* emulator. Taking a look at the spoils of our work is easy:

```
~/yoctohouse/poky/build$ runqemu qemux86
runqemu - INFO - Running MACHINE=qemux86 bitbake -e...
runqemu - INFO - Continuing with the following parameters:
```

Don't worry if *QEMU* asks for your superuser password – a new network interface needs to be mounted so that the virtual machine can access the internet. After about two to three minutes, the desktop shown in the screenshot *(above right)* will appear.

Baked in a Pi

Creating a desktop image for *QEMU* is not particularly interesting. Yocto would be able to demonstrate its strengths if we were able to move to new architectures with minimal effort, so let's download a layer supporting the Raspberry Pi 3:

```
~/yoctohouse/poky$ git clone git://git.yoctoproject.org/meta-raspberrypi
Cloning into 'meta-raspberrypi'...
remote: Counting objects: 6109, done.
```

The drivers and adaptations required to run Linux on a specific target are known as board support package or BSP. In the case of the Raspberry Pi, the Yocto project manages its own board support package which has just been downloaded. Experienced Yocto developers should not send a letter at this point: we intentionally omitted the `-b` parameter to download the latest version of the modules, for learning purposes.

Next, a new configuration must be created:

```
~/yoctohouse/poky/build$ cd ..
~/yoctohouse/poky$ source oe-init-build-env rpiHouse
```

Changing into the home directory of our project, we once again invoke the *oe-init-build-env* utility. This time, we pass in an additional name for the new configuration. When the tool has done its job, the folder contains an additional subdirectory. Here is a quick look at its contents:

```
~/yoctohouse/poky/rpiHouse/conf$ ls
bblayers.conf  local.conf  templateconf.cfg
```

Note the configuration file **local.conf**, which is what we want. Open it in your text editor of choice, and scroll to the following block:

```
#MACHINE ?= "edgerouter"
#
# This sets the default machine to be qemux86 if no other machine is selected:
MACHINE ??= "qemux86"
```

BitBake requires the presence of the `MACHINE` variable – its value is used to determine which target platform the image should have. Lines beginning with an `#` are comments, while the `??=` operator is responsible for setting defaults.

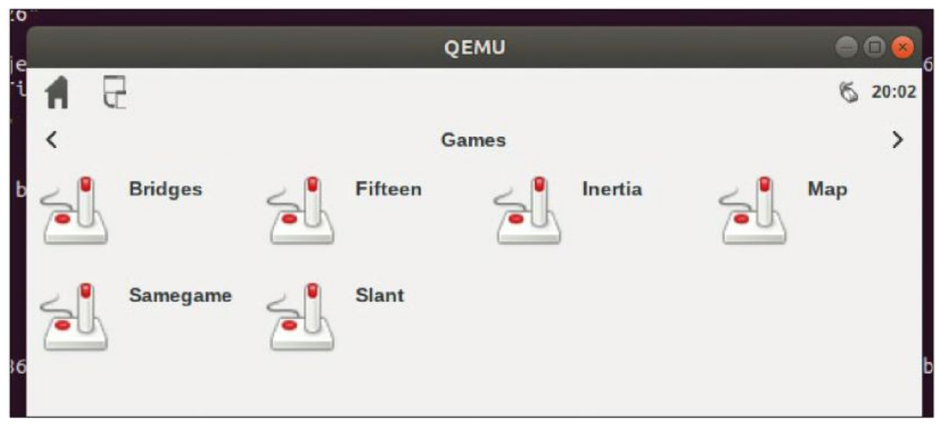
Given that we want to target the Raspberry Pi, the following declaration is needed:

```
MACHINE ?= "raspberrypi"
BBMASK = "meta-raspberrypi/recipes-multimedia/libav|meta-raspberrypi/recipes-core/systemd"
```

In addition to setting the machine variable to target the Arm system, we also passed a `BBMASK`. This is a regular expression for excluding recipes from the build process; the string used here is commonly found in Raspberry Pi builds.

bblayers.conf is the top-level recipe defining the steps *BitBake* needs to take. By default, its content starts out with the following declarations:

```
# POKY_BBLAYERS_CONF_VERSION is increased each time build/conf/bblayers.conf
# changes incompatibly
POKY_BBLAYERS_CONF_VERSION = "2"
```



The Sato GUI stack won't win any beauty pageants.

QUICK TIP

Should you still find yourself on a hard-disk-based machine, stop drinking beer or Red Bull until you can afford an SSD to work with Yocto. Trust us when we say that the productivity gains are more than worth it.

STRESSING OUT

Compiling Yocto images puts extreme levels of strain on your hardware and your net connection. When we started experimenting with Yocto, we purchased a high-performance workstation based on an FX8320 octa-core processor from AMD, along with 16GB of RAM and a 128GB SSD. When work on this tutorial started, the machine had about 90GB of free storage: when done, we were down to zero. RAM usage stayed below 8GB most of the time, however; having an additional 8GB made sense due to our penchant for playing around with virtual machines.

Keep in mind that the Yocto build process is CPU- and disk-intensive, so running desktop applications while the process is taking place in an uncomfortable experience. This makes Yocto ideally suited as a stress-test for your hardware. If a workstation has any kind of thermal issues, a Yocto run is very likely to find them, unless they are caused by the graphics processor.

Developers seeking to experiment with nice values and other scheduling tweaks can also use a Yocto build – it is among the most comfortable ways to load your system quickly while simulating realistic behaviour.

```
BBPATH = "${TOPDIR}"
BBFILES ?= ""
```

The `BBLAYERS` variable is responsible for setting the layers required. Modify it to make our newly downloaded layer part of the project:

```
BBLAYERS ?= " \
  /home/tamhan/yoctohouse/poky/meta \
  /home/tamhan/yoctohouse/poky/meta-poky \
  /home/tamhan/yoctohouse/poky/meta-yocto-bsp \
  /home/tamhan/yoctohouse/poky/meta-raspberrypi \
  "
```

This modification usually doesn't cause problems for experienced Linux developers. The main oddity involves the syntax; each line needs to be terminated with a `\`. Once done, it is time to invoke *BitBake*:

```
~/yoctohouse/poky/rpiHouse$ bitbake core-image-sato
```

BitBake does not limit itself to performing the actual compilation process, but also runs various checks to ensure the plausibility of the image at hand. In the case of our current configuration, the compilation cannot be done as the 'libmms' package is missing:

```
NOTE: Resolving any missing task queue dependencies
ERROR: Nothing PROVIDES 'libmms' (but /home/tamhan/yoctohouse/poky/meta/recipes-multimedia/gstreamer/gstreamer1.0-plugins-bad_1.14.4.bb DEPENDS on or otherwise requires it). Close matches:
  libsm
  libfm
  libcomps
```

While you can force *BitBake* to compile if packages are missing, the resulting images usually will not work on the target system. Instead, open the **readme** file of our newly downloaded layer.

It contains the following statement, informing us that a few additional layers from the main OpenEmbedded distribution are needed:

```
* URI: git://git.openembedded.org/meta-openembedded
* layers: meta-oe, meta-multimedia, meta-networking, meta-python
* branch: master
* revision: HEAD
```

From a technical point of view, you should know what happens next. Change back into the main folder, and obtain the **openembedded** folder via the following command. The disclaimer about missing parameters from the BSP layer also is valid here – we'll find out more about it quite soon. For now, however, simply burden your internet connection with an additional download command:

```
~/yoctohouse/poky$ git clone git://github.com/openembedded/meta-openembedded
```

Enumerate the contents of the folder once the download has completed:

```
~/yoctohouse/poky/meta-openembedded$ ls
contrib COPYING.MIT meta-demo meta-filesystems meta-gnome meta-initramfs meta-multimedia meta-networking meta-oe meta-perl meta-python meta-webserver meta-xfce README
```

The **openembedded** folder contains a set of additional layers, each of which provides packages dedicated to one function. Return to **conf** and add the missing lines to the **bb** file in order to add them to our compile process:

```
BBLAYERS ?= " \
  /home/yoctohouse/poky/meta \
  /home/yoctohouse/poky/meta-poky \
  /home/yoctohouse/poky/meta-yocto-bsp \
  /home/yoctohouse/poky/meta-openembedded/meta-oe \
  /home/yoctohouse/poky/meta-openembedded/meta-multimedia \
  /home/yoctohouse/poky/meta-openembedded/meta-networking \
  /home/yoctohouse/poky/meta-openembedded/meta-python \
  /home/yoctohouse/poky/meta-raspberrypi \
  "
```

Adjust the paths to suit your local situation. In theory, we should be good to go. Return to the project directory and order another *BitBake* run. Sadly, it will fail with an extremely odd error message:

```
WARNING: /home/tamhan/yoctohouse/poky/meta-openembedded/meta-oe/recipes-dbs/postgresql/postgresql_11.2.bb: Exception during build_dependencies for PERL_ARCHLIB          | ETA:  0:00:27
```

Following the stack trace reveals that *BitBake* was unable to satisfy a local variable pointing to the Perl distribution. This is a feature which was added to a later version of Yocto; we intentionally downloaded a slightly older version of Yocto. The two layer download commands, however, fetched the most recent versions from GitHub – this turns out to be a problem, as our version of Poky cannot satisfy their demands.

Layer mismatches are among the most common problems faced by Yocto engineers. Fortunately, fixing it is not hard. First, delete the folders **meta-openembedded** and **meta-raspberrypi**. Next, re-download them while passing a `-b` parameter:

```
~/yoctohouse/poky$ git clone -b thud  git://github.com/openembedded/meta-openembedded
```

One of the most pressing issues when dealing with embedded software involves licensing the various

» YOCTO WILL NOT SAVE YOU MONEY

When optimising the bill of materials for a process computer system, opting for smaller memory card sizes always seem like a good idea. But reducing the amount of memory used in a process computer usually does not lead to significant cost savings.

Three issues are responsible: first of all, permanently falling memory prices caused by Moore's Law and semiconductor industry factors quickly diminish savings.

Secondly, memory card cost is but a small part of the TCO. Optimising other parts tends to give significantly more savings: a good example would be switching the Raspberry Pi 3, which sets you back £35, for an Orange Pi, which can be had for much less.

Finally, the appearance of a pesky customer asking for an extra feature after release is almost preordained. If your system is space-constrained, implementing simple software changes can turn out to be a huge nuisance, requiring a recall or a complete redesign.

This, of course, should not discourage you from using Yocto for any of the reasons mentioned in this article. However, switching to Yocto purely in order to save space on the microSD card almost always turns out to be a waste of time, with costly side effects. Don't say we didn't warn you!

components. When working with automotive companies, developers should not be surprised if they spend a day or two getting audited for compliance. Yocto aims to simplify your life by integrating a licence management system into *BitBake*. The idea behind it is simple: while sifting through the various recipes, Yocto analyses the declared licence flags. Its configuration contains an additional variable specifying the allowed licences; if a recipe has a licence which was not whitelisted, compilation is aborted.

Dealing with licensing

As of this writing, the Yocto Sato version of the Raspberry Pi image includes such a package – when building the image, an error shows up:

```
ERROR: Nothing PROVIDES 'faad2' (but /home/tamhan/yoctohouse/poky/meta/recipes-multimedia/gstreamer/gstreamer1.0-plugins-bad_1.14.4.bb DEPENDS on or otherwise requires it)
faad2 was skipped: because it has a restricted license not whitelisted in LICENSE_FLAGS_WHITELIST
```

For now, save time by working around the problem. Use the following command to create a simpler image which makes do without the GUI stack:

```
~/yoctohouse/poky/rpiHouse$ bitbake core-image-base
```

BitBake contains extensive logic to minimise compilations whenever possible. It will reuse most of the packages built for the GUI-less version when we return to building Sato. While your workstation runs the cross-compile process – it takes longer, as cross-compilation is computationally intensive – let's start tracking down the problem.

Our violation was caused by the package faad2, which resides at **http://bit.ly/lxf251faad**. In it, we find the following code:

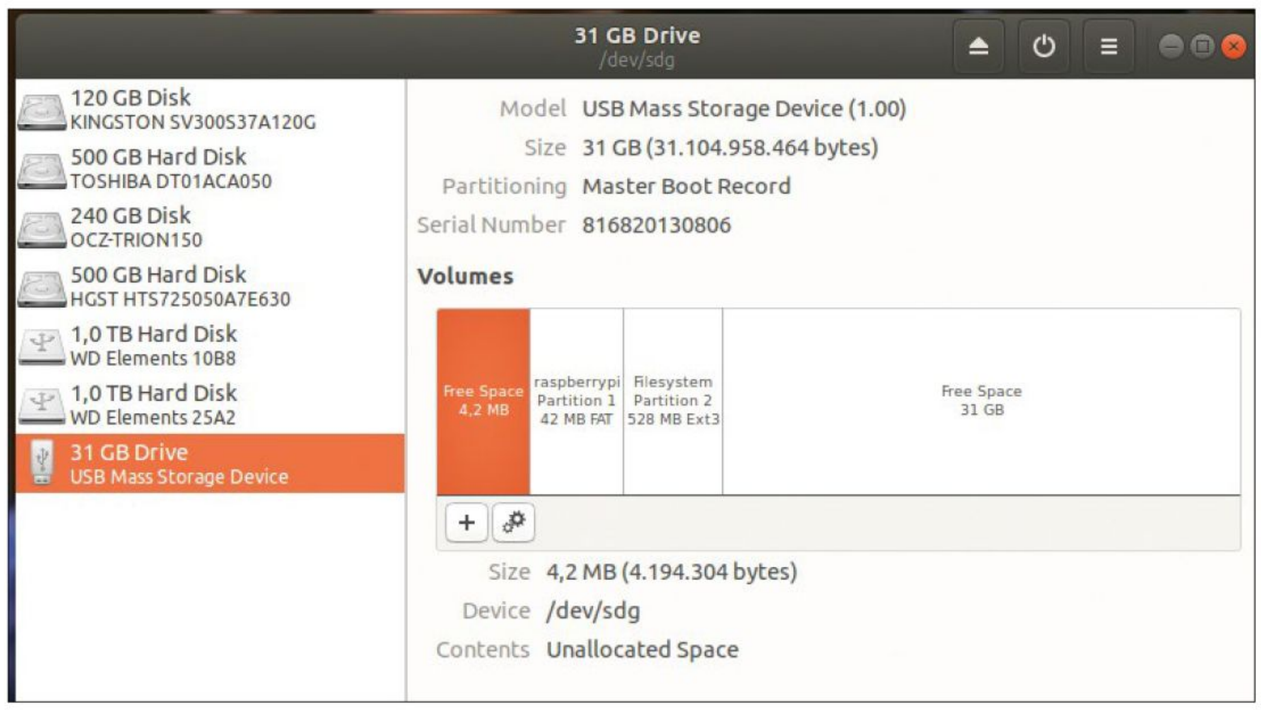
```
SUMMARY = "An open source MPEG-4 and MPEG-2 AAC decoding library"
HOMEPAGE = "http://www.audiocoding.com/faad2.html"
SECTION = "libs"
LICENSE = "GPLv2"
LIC_FILES_CHKSUM = "file://COPYING;md5=381c8cbe277a7bc1ee2ae6083a04c958"
LICENSE_FLAGS = "commercial"
SRC_URI = "${SOURCEFORGE_MIRROR}/faac/faad2-src/faad2-2.8.0/${BP}.tar.gz"
SRC_URI[md5sum] = "28f6116efdbe9378269f8a6221767d1f"
SRC_URI[sha256sum] = "985c3fadb9789d2815e50f4ff714511c79c2710ac27a4aaaf5c0c2662141426d"
inherit autotools lib_package
```

In addition to specifying where the source code is to be obtained, we also find that the `LICENSE` value is set to `commercial`. As we cannot change **.conf** files during a compile, wait until the build process has run.

Whitewashing our problem

Should you have closed down the terminal window by mistake, start out by reconfiguring the various local variables loaded by *oe-init-build-env*. Invoking the command with the name of a 'known' build folder leads to the re-initialisation of the build environment:

```
~/yoctohouse/poky$ source oe-init-build-env rpiHouse
```



The results of building the Sato Raspberry Pi image.

```
### Shell environment set up for builds. ###
You can now run 'bitbake <target>'
```

In the next step, return to **local.conf** and add the following to permit the use of commercial packages:

```
# variable as required.
LICENSE_FLAGS_WHITELIST = "commercial"
```

Running a compile of the 'larger' image in the current state would deplete the storage capacity of our workstation. Fortunately, the image created for *QEMU* is no longer needed; as we passed no filename to *oe-init-build-env*, its contents are located in **/home/tamhan/yoctohouse/poky/build/**. The contents of the **/tmp** folder amount to a impressive 28GB: delete them wholesale. As long as the contents of **/build/conf** remains untouched, *BitBake* can rebuild the image from scratch (burning CPU hours in the process). This, of course, also assumes that the various GitHub repositories remain available – a problem you should keep in mind when working on legally sensitive projects requiring build reproducibility.

At this point, order *BitBake* to run another compilation. The build system will inform you that a few hundred packages were found from the previous run, and will set about finishing the new image. Look for the file **core-image-sato-raspberrypi.rpi-sdimg**, rename it to **core-image-sato-raspberrypi.img** and burn it to a memory card of choice. When done, Ubuntu will reveal the folder structure shown in the screenshot above.

Sadly, the 64-bit processor of the Raspberry Pi will reject the image. Fixing the problem is easy: return to **local.conf**, and adjust the value of the `MACHINE` variable like so:

```
MACHINE ?= "raspberrypi3"
```

This time, the rebuild will work significantly faster – given that the Raspberry Pi 3 shares most of the instruction set with its predecessor, many packages get 'licked at' without compilation.

Going further

Getting Sato booted on the Raspberry Pi is but part of the job; if the distribution should be used practically, the next step involves adding layers containing recipes for the control task at hand. Yocto simplifies this by letting you create an ADK, a set of compiler and software components for the newly created target.

Sadly, we haven't got the space here to delve into these topics. However, the infrastructure constructed in this article simplifies further experimentation. We hope that you enjoyed the ride!

Keep track with Status Board

Monitor your internet connection, detect who's home and keep track of your finances with this handy Pi add-on

Nate Drake
is a technology journalist specialising in cybersecurity.

Resources

- **Raspberry Pi**
  Use a Pi 3 or a Pi Zero W
- **Code samples**
  http://bit.ly/SBcodesamples
- **Pi Hut tutorial**
  http://bit.ly/SBoardtutorial

The Pi Hut's latest add-on, the Status Board is simple yet powerful, like the Pi itself and with a board, you can monitor anything you want at a glance.

The Status Board has five customisable strips, each with a green and red LED. The options are almost limitless. Do you want to be able to quickly check if your home server is online? Or perhaps you'd prefer to monitor a value, such as your whether your investments have gone up or down? Each strip can be written on with a dry-wipe marker, so you can keep reassigning them.

In this guide, we'll focus on basic setup of your Status Board, along with some example scripts coded by **LU&D** and the folks at the Pi Hut. These sample scripts can be used to check your internet connection is working, ping devices, as well as check if your shares have gone up or down in value. You'll also discover how to customise the scripts to your needs and how to design your own.

This tutorial assumes you have a clean install of Raspbian Stretch on your Raspberry Pi. Make sure to run `sudo apt-get update` and `sudo apt-get upgrade` before proceeding.

01 Connect your Pi Status Board

Head over to the Pi Hut and purchase the Status Board for your either your Raspberry Pi 3 or Pi Zero (**http://bit.ly/SBoardbuy**). Setup is incredibly simple as you only need to connect the Status Board directly to the GPIO pins on the Pi. Disconnect the Pi's power cable while connecting the board.

In order to use the Status Board you'll need the latest version of the GPIO Zero Python library. This should be pre-installed in Raspbian but if not, open a Terminal window and run `sudo apt install python3-gpiozero python-gpiozero -y`.

02 Test Status Board

Open a Terminal on your Pi and launch the Python shell by typing `python3`. First, import the StatusBoard module by entering `from gpiozero import StatusBoard`. Next, enter `sb = StatusBoard()`. This will make it easier to type commands, as they'll be shorter.

Next, switch on all the LEDs on all strips by running `sb.on()`. Switch them off again by typing `sb.off()`. Now let's take a look at how to turn on a single LED. For instance, to illuminate the green LED on strip three, type: `sb.three.lights.green.on()`. Next, try making a light blink: `sb.five.lights.blink()`.

```
File  Edit  Tabs  Help
pi@raspberrypi:~ $ python3
Python 3.5.3 (default, Jan 19 2017, 14:11:04)
[GCC 6.3.0 20170124] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from gpiozero import StatusBoard
>>> sb = StatusBoard()
>>> sb.on()
>>> sb.off()
>>> sb.three.lights.green.on()
>>> sb.five.lights.blink()
>>> sb.off()
```

03 Test naming strips

By default, the strips on the Status Board are named 'one', 'two', 'three' and so on. To avoid confusion, you can name strips individually in Python. Do this by defining the `StatusBoard` values; for instance if you're using it to detect motion in rooms in your house, you could enter: `sb = StatusBoard('kitchen', 'lounge', 'basement', 'study')`. If you're composing a script, you could switch on the red LED for the 'study' strip with the command: `sb.study.lights.red.on()`.

Type `exit()` once you've finished testing.

04 Download ping script

In a Terminal, run:

```
wget https://raw.githubusercontent.com/nate-drake/pistatusboardsamples/master/ping.py
```

```
File  Edit  Tabs  Help
pi@raspberrypi:~ $ wget https://raw.githubusercontent.com/nate-drake/pistatusb
rdsamples/master/ping.py
--2017-09-24 10:49:20--  https://raw.githubusercontent.com/nate-drake/pistatus
ardsamples/master/ping.py
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 151.101.120
33
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|151.101.12
133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 494 [text/plain]
Saving to: 'ping.py'

ping.py             100%[===================>]     494  --.-KB/s    in 0s

2017-09-24 10:49:20 (2.93 MB/s) - 'ping.py' saved [494/494]

pi@raspberrypi:~ $
```

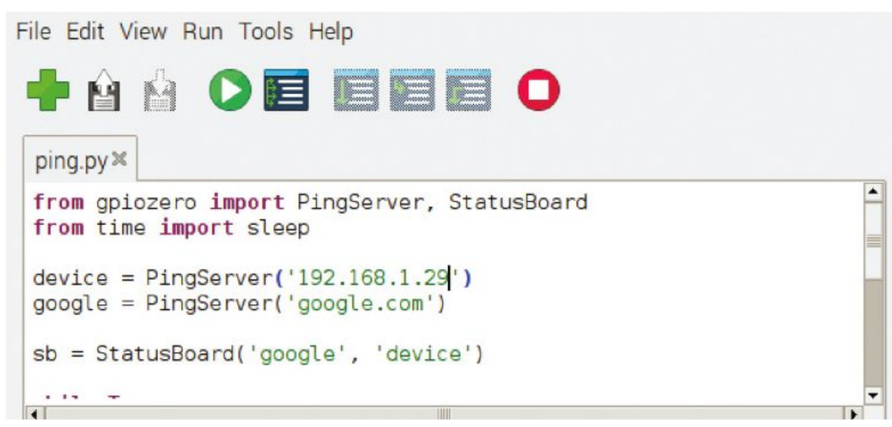
This example ping script uses the first two strips on the Status Board. The first will show a blinking green light if the Pi can connect to the internet. The second checks if the IP address of a certain device such as a mobile phone is connected to your local network. To start editing, run

nano ping.py in the Terminal or better yet, open it with Thonny Python IDE to ensure text is formatted correctly. Note that the first two strips have been named 'google' and 'device'.

05 Configure domain settings

Find the line which which reads `google = PingServer('google.com')`. Under the current configuration, the PingServer module will check if it can connect to http://google.com every three seconds. (You can change this by amending `sleep (3)`.)

If the connection is successful, the script will cause the green LED on the first strip to start blinking, otherwise it will switch off. The advantage of using a domain name here is that PingServer will not only check that your internet connection is working but that your DNS settings are correct too.



06 Configure device settings

The **ping.py** script also tries to locate a device on the same network such as a family member's mobile. To use this feature, you need to know the device's IP (the cross-platform, free mobile app Fing is perfect for this.)

Find the line which reads `device = PingServer('192.168.1.29')` and amend 192.168.1.29 to the IP address of the device you want to detect. If it can be pinged, the green LED on strip two lights up, otherwise the red LED will show. Run the script from Terminal with `python3 ping.py`.

07 Download finances script

Reopen Terminal and run:

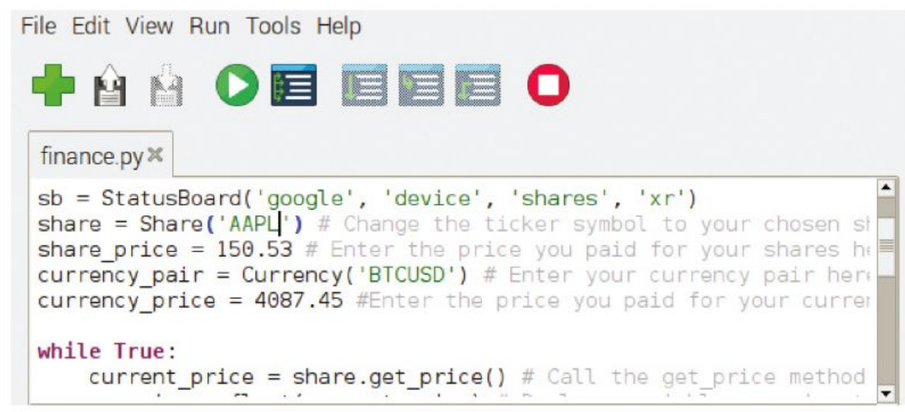
```
wget https://raw.githubusercontent.com/nate-drake/pistatusboardsamples/master/finance.py
```

This second Python example script focuses on the third and fourth strips on the Status Board, and uses the yahoo_finance module to monitor your financial data. The first part of the script checks whether the value of your shares have gone up or down. The second part checks the current exchange rate for a currency pair.

To start editing, run `nano ping.py` in Terminal or better yet, open it with Thonny Python IDE to ensure the text is formatted correctly.

08 Configure share price settings

If you dabble in the stock market, you'll know that shares are identified by a ticker symbol. If you find



the line which reads `share = Share('AAPL')`, you'll see that by default the script will check the value of Apple shares. Amend this to the ticker symbol for the shares you hold. If you're not sure what this is, visit **https://finance.yahoo.com** to search for a company.

Make sure to amend the `line share_price = 150.53` to the actual dollar value you paid per share originally too.

09 Configure currency settings

The script can also check the value of a currency pair. Find the line reading `currency_pair = Currency('BTCUSD')`. By default the script checks the current value of Bitcoin in USD; however, you can change this to another pair if you wish, e.g. EURPLN.

Whichever currency you choose, make sure to amend the line `currency_price = 4087.45` to reflect what you paid for the base currency; e.g. you may have paid 4.2684 euros per Polish zloty. Run the script with `python3 finance.py`. It can run simultaneously with **ping.py**.

10 Further steps

We have intentionally not addressed the fifth strip so that you can use it for whatever purpose has popped into your head while reading this. Head over to **http://bit.ly/SBoardexamples** for a few code examples. These include a script to check the London tube lines, one for checking the weather and even a program to check for names in the news.

Make sure you name your fifth strip in the way outlined above to reflect your chosen purpose. If any rogue LEDs switch on, go to **Step 2** and repeat the commands up to `sb.off()` to turn them off.

Intense pulses

The Status Board can control the intensity of LEDs. You can do this firstly by adding `pwm=True` to `sb` in your scripts; for instance: `sb = StatusBoard('google', 'device', 'shares', 'xr', 'weather', pwm=True)`. You will now be able to control the intensity of LEDs; for instance, `sb.weather.lights.green.value = 0.5` instructs the Status Board to lower the green LED on strip five to half its usual brightness, then switch it on.

You can also use this feature to make LEDs pulse. To do this for the same LED above, run: `sb.weather.lights.green.pulse()`. You can also make both LEDS pulse by running: `sb.weather.lights.pulse()`.

Switch off LEDs in exactly the same way as we've outlined above.

RASPBERRY Pi ROBOTS!

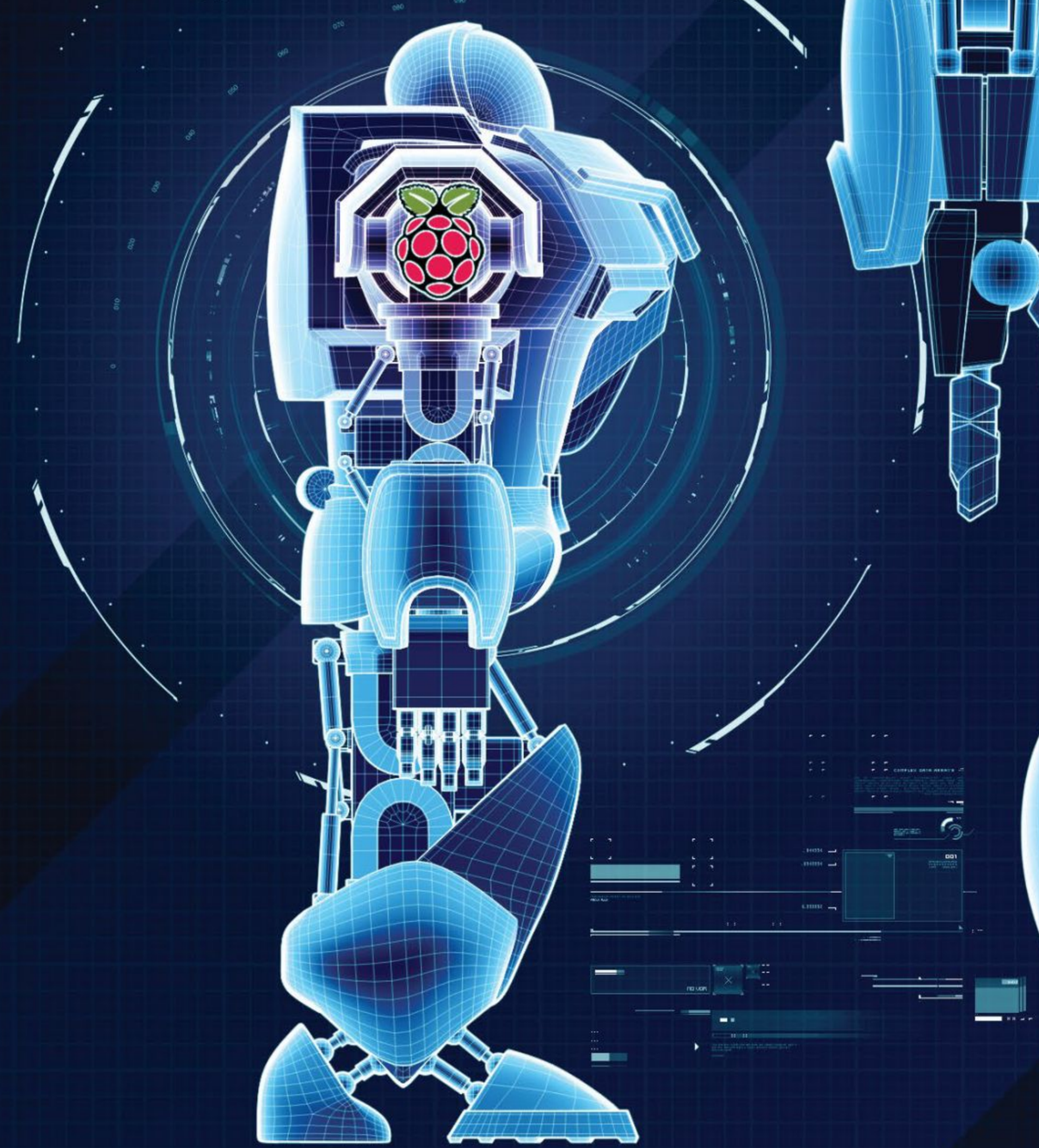
Supervillain Jonni Bidwell is constructing an army of Pi robots. Sneak into his lair and steal his techniques!

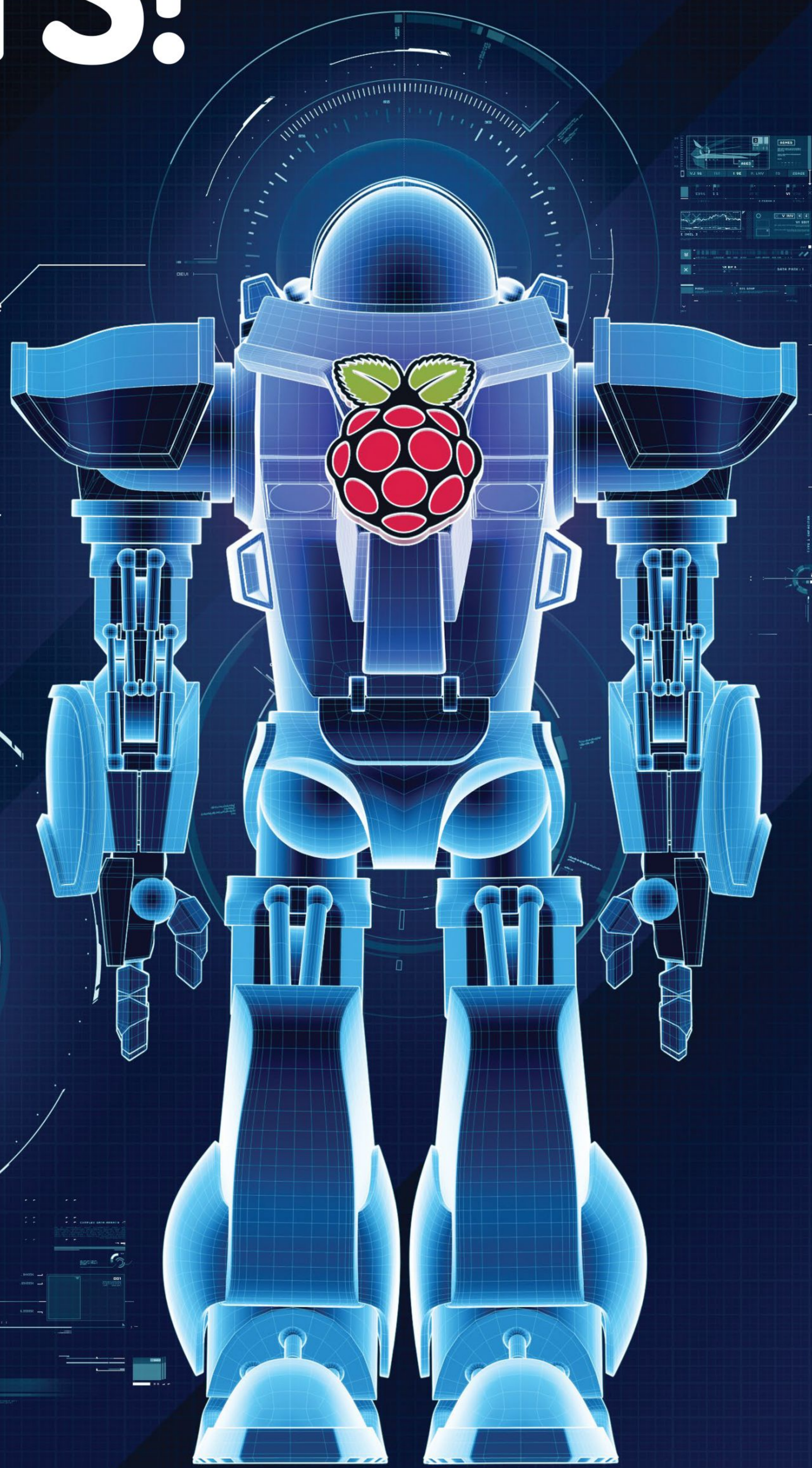
Robotics needn't be about building intimidating humanoid automatons. We can build machines that can perform all kinds of other useful, instructive or fun functions. Even better, you can power them with Linux.

You won't need costly electronics to build these devices. A Raspberry Pi will do just fine. It doesn't need to be any more advanced than constructing a LEGO set, either. There are plenty of off-the-shelf kits suitable for any budding electronics enthusiast.

We'll reveal how to put together your own roving robot, and show you a couple of neat Python programs to get you started. And if you're feeling more adventurous, then why not build your own robotic helpers from scratch? All the components are easy to get hold of, thanks to the growing maker community.

If you want to take it to extremes, then 3D print your own custom parts and add some pseudo-sentience to your creation with some machine-learning voodoo. Or build your own robotic pet and have it follow you around like a puppy. The possibilities are endless. So power up that Pi, get yourself some components and let's start hacking some robots together.





A Pi bot's potential

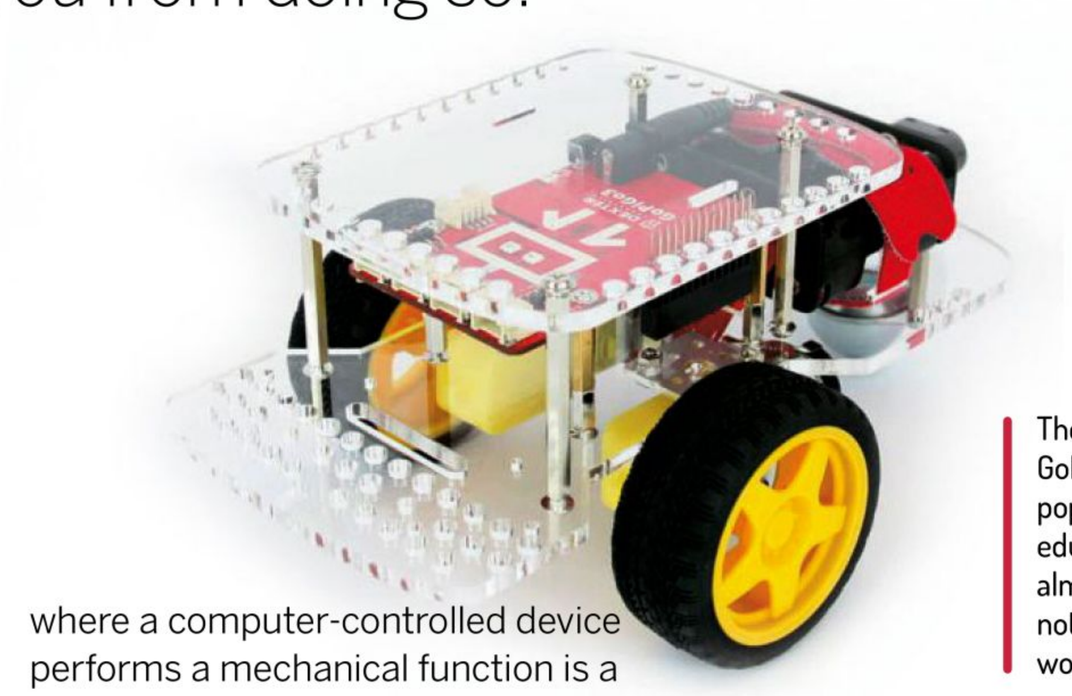
We're not talking about engineering an Arnie-shaped T-101: Pi Edition, but don't let us stop you from doing so.

The Raspberry Pi was launched seven years ago, and since then it has grown into something amazing. We've seen four different major versions (Zero, 1, 2, 3 and 4), and a couple of minor versions.

Pi retailer **ModMyPi.com** was there at the beginning, and was good enough to lend us some treats for this feature. Founder Jacob Marsh offers his take on the Pi's remarkable success. "I didn't realise that the Raspberry Pi would grow to what it is now, I don't think even the Pi Foundation did. ModMyPi was conceived in a university bedroom. Now we have a global e-shop with more than 2,000 product lines, a £3 million turnover, seven full-time staff, and a 300-square-metre warehouse that we're rapidly outgrowing. Long may it continue!"

What makes the Pi so special, and what's contributing to Jacob's real-estate shortage, is the limitless scope for expansion. A tiny general-purpose computer that can be connected to sensors, motors or your central heating system, coupled with the collective imaginations of the open source community, has led to all kinds of wonderful physical computing applications. And one of the most wonderful is robotics.

Nevermind our fanciful illustration. Any situation where a computer-controlled device performs a mechanical function is a robot at heart. Robotic vehicles are particularly popular with Pi hobbyists just now. These might at first appear to be little more than the radio-controlled cars, but they can do much more than scoot around the kitchen table.



The Dexter GoPiGo robot is a popular choice for education, and is almost certainly not bent on world domination.

Having a tiny Linux machine inside enables all kind of tinkering. With a few easy-to-connect components, some open source libraries and just a little bit of Python programming to connect everything together, you're limited only by your imagination. Well, small caveat, you may have to solder a few wires. This is daunting at first, but there are plenty of guides on the internet. Check out the Pi Foundation's director of education Carrie Anne Philbin giving an excellent demonstration at **https://youtu.be/P5L4GI6Q4Xo**.

KEEPING YOUR ROBOT OCCUPIED

"A Pi-powered robot might take pictures, zoom around your house quoting Shakespeare or even take to the skies"

A Pi robot can run the same Raspbian Linux we all know and love. It can connect to wireless networks, so you can SSH in and perform diagnostics while it's on the move. You can even run `apt upgrade` while it's driving around. A Pi-powered robot might also take pictures, zoom around your house quoting Shakespeare or even take to the skies to admire the city below. With the aid of the OpenCV library and a little bit of image processing we can give our robot computer vision, so that it can target and follow objects, or even recognise objects or people. So get dig into our guide, and don't forget to check out **www.modmypi.com** for inspiration.

» ROBOT PLATFORMS

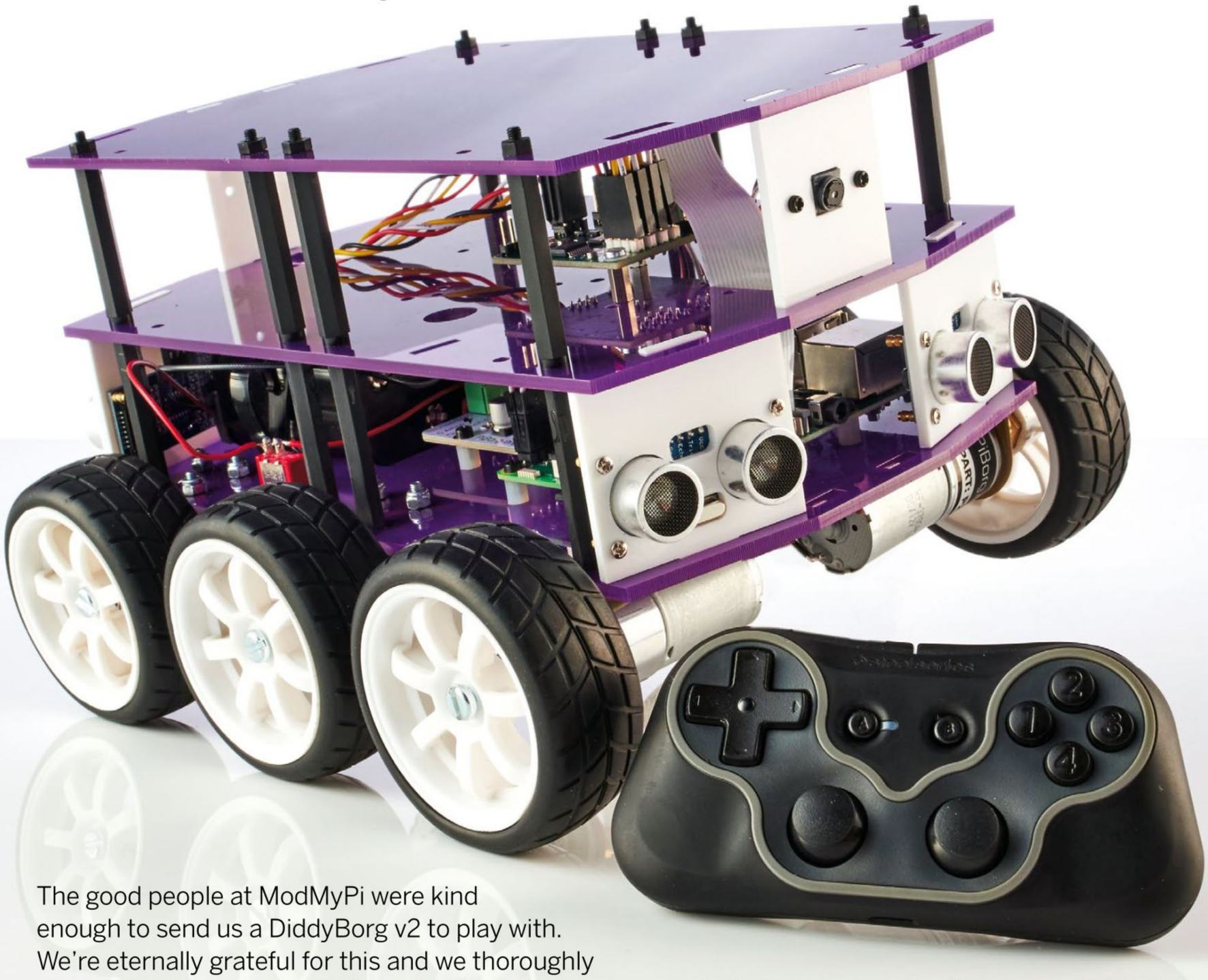
We'll concentrate on bread-and-butter software, namely Raspbian and Python, but there are a number of interesting platforms than can be run on top of, alongside or instead of these.

One such platform is Robot OS (ROS, see **www.ros.org**) which isn't really an operating system at all, but more of a middleware. ROS is a BSD-licensed project with thousands of contributors worldwide. It can be built from source for Raspbian, but since it provides packages for the Pi edition of Ubuntu MATE, that's the preferred base. ROS aims to be truly general-purpose robot software that works on many platforms and encourages collaboration.

We're also impressed with the work of Dexter Industries (**www.dexterindustries.com**), which makes the GoPiGo robot car. The GoPiGo is a kit designed for the classroom, and is programmed using Bloxter, a browser-based language similar to Scratch. Within the Blockly interface it's possible for more advanced students to program in Python, too. Fans of LEGO will enjoy Dexter's BrickPi, which connects the Raspberry Pi with the popular LEGO Mindstorms kit. If that weren't enough, it also makes the Grove Pi, a kit of 12 plug 'n' play sensors for exploring IoT programs.

Building the DiddyBorg

Robot construction is much more straightforward than you think. Let us show you how...



The good people at ModMyPi were kind enough to send us a DiddyBorg v2 to play with. We're eternally grateful for this and we thoroughly recommend you check out their store next time you need anything Pi-related.

The Borg series was designed by PiBorg, brainchild of Timothy Freeburn, who explains "The DiddyBorg is a six-wheel drive robot and is a homage to the Sojourner Mars Rover". PiBorg teamed up with ModMyPi back in October 2017. Jacob explains this collaboration: "Tim and I were both around during the genesis of the original Raspberry Pi B. Where I was off making cases, Tim was designing motor-controllers and LED thingies (**www.piborg.org/ledborg**), so when ModMyPi developed into wider reselling, Tim's products were some of the first that we picked up.

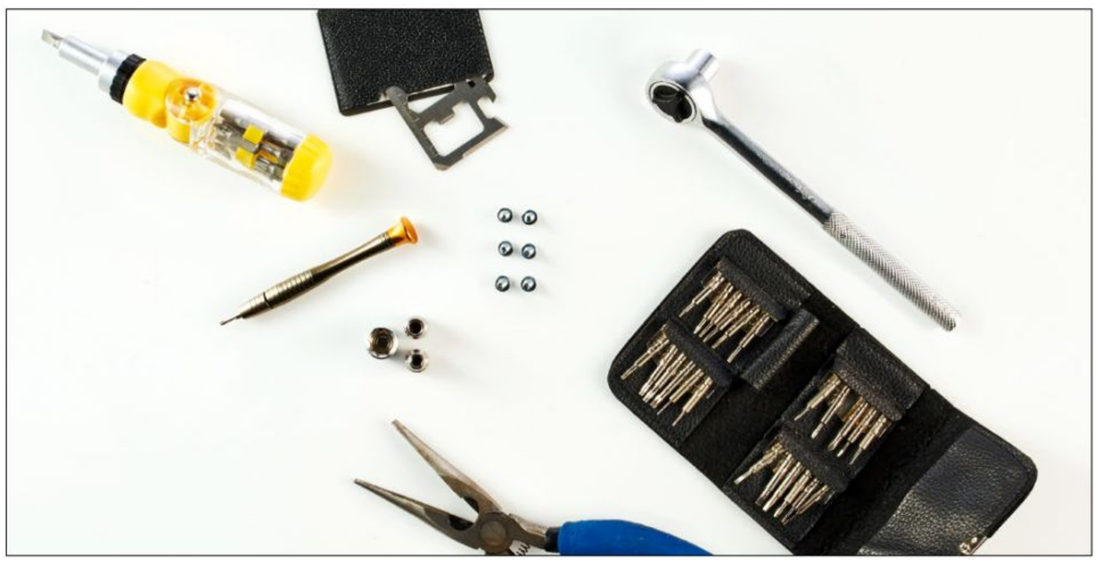
"As the market has progressed over the years we realised the need to bring in more in-house development, and Tim realised the value in utilising our distribution network for his products. He could devote time to making more cool stuff, and we could devote more time to selling and sending. We both work well at our respective goals and get along, so it was a no-brainer to team up!"

Our Diddyborg features six powerful 12V 100RPM motors and is powered by 10 AA batteries. We'll look at alternative power sources later. Power is distributed by the custom ThunderBorg board, which can handle 5A per motor, has a multicolour LED (useful for monitoring battery status) and we're sure you agree looks like the business. Rather than using some complicated servo arrangement for steering, the DiddyBorg rotates by turning its wheels at different speeds.

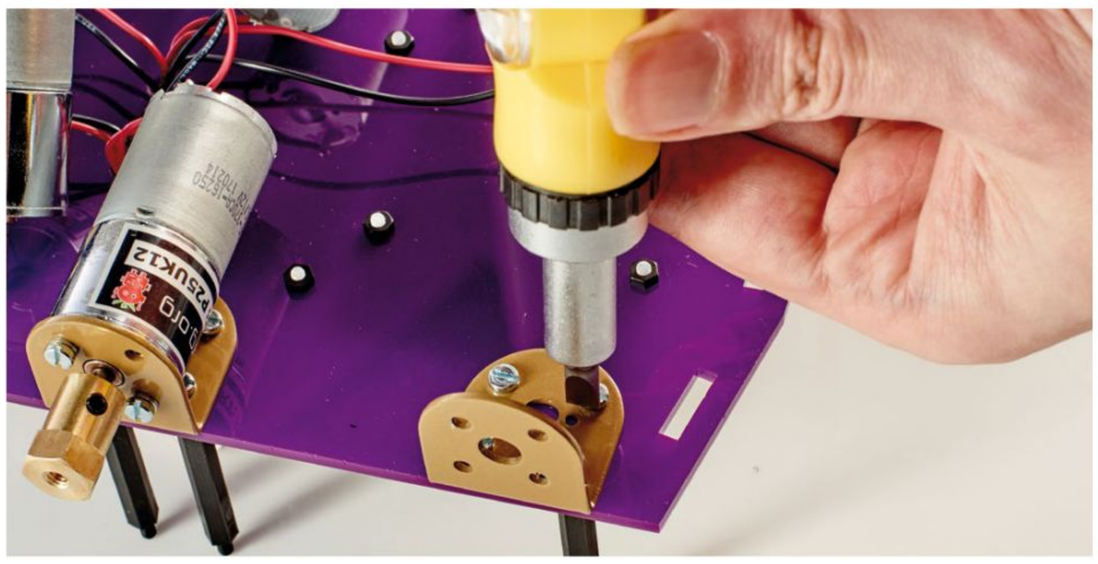
The borgs (see below), and many other robots, are sold as build-it-yourself kits. Depending on the particular kit, this might be quite an involved process, involving small parts, soldering and possibly swearing. Building the Diddy takes a couple of hours from start to finish, and you can find instructions at **www.piborg.org/blog/build/diddyborg-v2-build/diddyborg-v2-build-instructions**. It would be silly to reproduce these in their entirety here, but do check out our general step-by-step robot building guide (*right*). We've based it on the DiddyBorg, because we have one, but construction of other robots will follow a similar prescription.

If you can, it's a good idea to set up the Pi as best you can before starting the build. Of course, it's possible to SSH to the robot once it's built to tweak things (so long as you've told it about your wireless network beforehand), but heavy wireless usage will sap battery power that could otherwise be used for careening around the office annoying lesser magazine teams. If you have another Raspberry Pi and nimble fingers, then you can extract the SD card post-construction and perform any tweaks on the other device.

BUILDING A Pi ROBOT

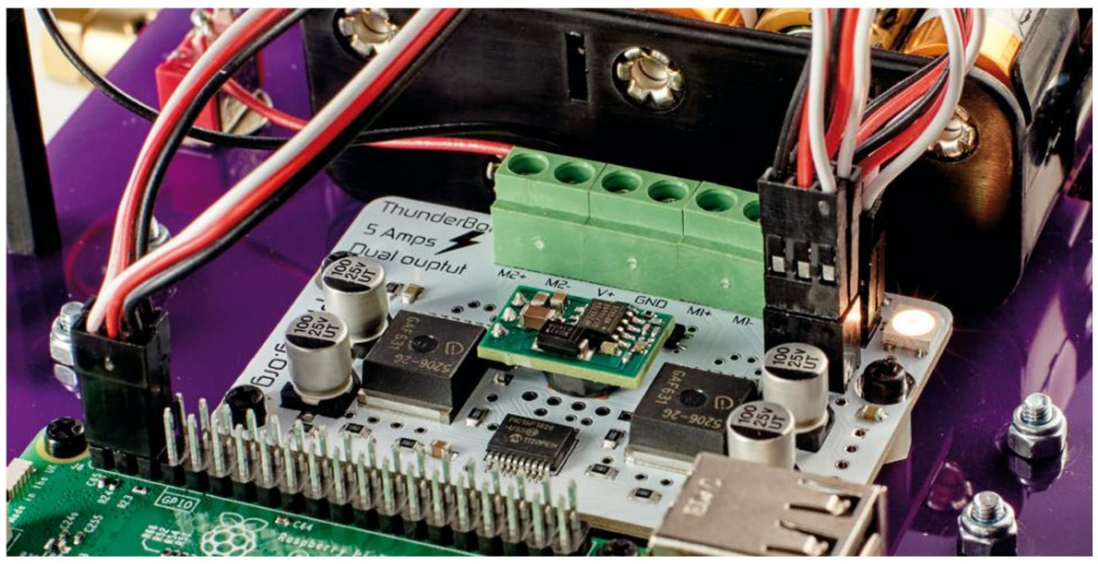


01 Organise your workspace
We know how tempting it is to just dive right in and spill the contents of all the bags over your already messy desk. Don't do this. Instead, clear a nice workspace, read the instructions carefully and place all the tools you'll need to one side. If you have to solder anything then make sure this happens in a well-ventilated area, and have some desoldering wick to hand.

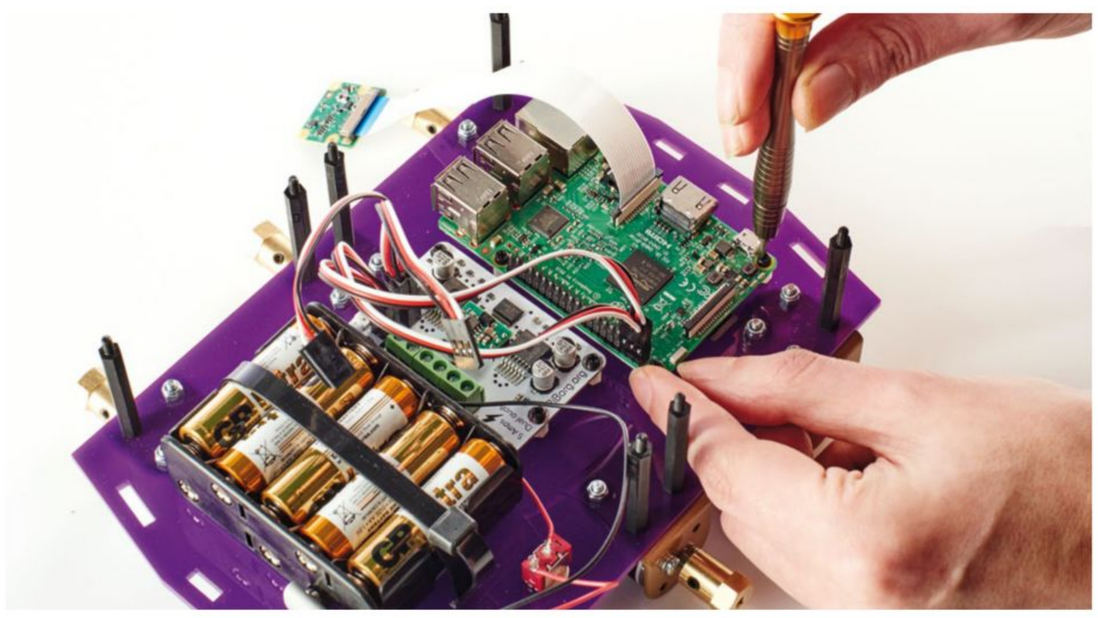


02 Fix the mounts
The motors will usually attach to the underside of the chassis by way of some angled brackets. Find the right holes and attach these brackets with the appropriate nuts and bolts. The DiddyBorg uses metal posts to secure the brackets. Don't forget to add washers if your chassis needs them. These help to distribute force and will protect against chassis fractures.

03 Attach motors and controller
Fix the motors in place with the appropriate screws. The DiddyBorg uses spring-lock washers to ensure friction even if the holding screw is loosened. You really don't want your motors detaching while your robot is in motion. Now attach the motor controller to the chassis, or attach any required standoffs and then do so. Pay attention to the controller's orientation here.



04 Solder and attach
Cut the required lengths of wire and strip the ends. We can't tell you how to solder from the confines of this small box, so be careful. Solder the red wires (if appropriate) to the positive sides of the motors, and the black ones to the negatives. Use a piece of tinfoil to catch any drips and protect the chassis. Connect the wires to the appropriate controller terminals.



05 Attach your Pi and batteries
Affix the battery pack and switch to the chassis, and then connect them to the controller. For more powerful robots this might involve heavy-duty wiring compared to the motors. Ensure that the SD card is in the Pi (and that Raspbian is set up to use your wireless network and Bluetooth controller) and attach the Pi to the chassis. Take care not to overtighten the screws.

06 Finish the job
Connect the controller to the correct GPIO pins on the Pi (ensure that the connectors are oriented correctly). Attach any other robot peripherals (such as sensors or cameras). Fix the wheel hubs to the motors securely with grub screws, and check that there's sufficient clearance for them to turn freely. Finally, put the wheels on the hubs and screw them firmly in place.

Keep your robot busy

What you do with your robot is limited only by your imagination and attention span. But for now, here are a few suggestions to inspire you...

Having constructed our Diddy, we wanted to know more about what it could do. Not content with making hardware, PiBorg also provides some great code examples to get you started. These will need some tweaking to work on other hardware, but should give you some idea of how to talk to the hardware in Python.

Robot kits will provide scripts to deal with all the low-level communication. In our case this is all done through the **ThunderBorg.py** file (see **www.piborg.org/blog/build/thunderborg-build/thunderborg-examples**). This handles all the raw I2C coding, so you don't need to worry about that, and provides much more human-friendly functions such as `SetMotor1()`, which sets the speed of the left hand wheels.

Web control

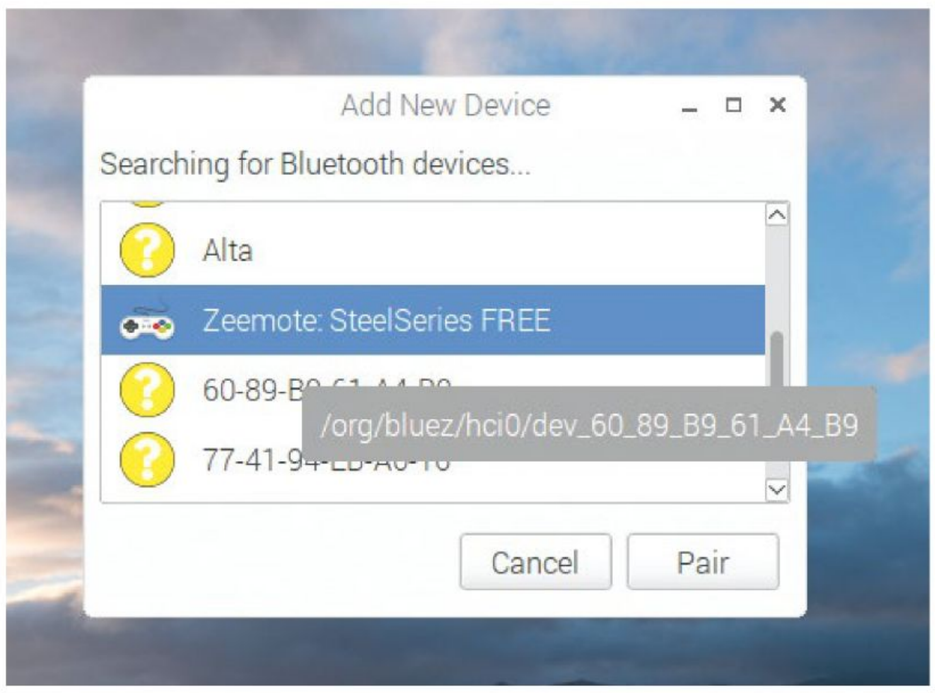
Assuming your Pi is connected to a wireless network, then one slightly roundabout way to control it is to have it run a small webserver with an HTML form to control it. If your robot has a camera attached too, then you can stream the video to this webpage, for a delightful first-person video driving experience.

Creating a streaming video processor in Python is less complicated than you'd think, but more complicated than we'd like to get into in this summary. So study the DiddyBorg web UI example at **www.piborg.org/blog/build/diddyborg-v2-build/diddyborg-v2-examples-web-ui** to see how the magic happens. If you're are lucky enough to own a DiddyBorg then copy that script to it,

Gamepad control

Controlling your robot with a gamepad is a little easier to get along with. However, as we discovered, they might need a little persuasion to work when a GUI isn't running (such as when your Pi isn't connected to a monitor). In theory, you could set this up beforehand, or by removing the SD card from the robot and booting it in another Pi – the settings should be remembered. If not, we can set this up by SSHing into our robot.

There are two challenges to overcome: the actual Bluetooth pairing and the subsequent setting up of the device nodes. The latter is handled by the `joystick` package (or `evdev` which it depends on) and the former by the `bluetoothctl` command (this will be installed as standard). After installing the `joystick` package run `bluetoothctl`. This will start a console where we can scan, pair and connect our controller. First, put the device in pairing mode and then initiate a



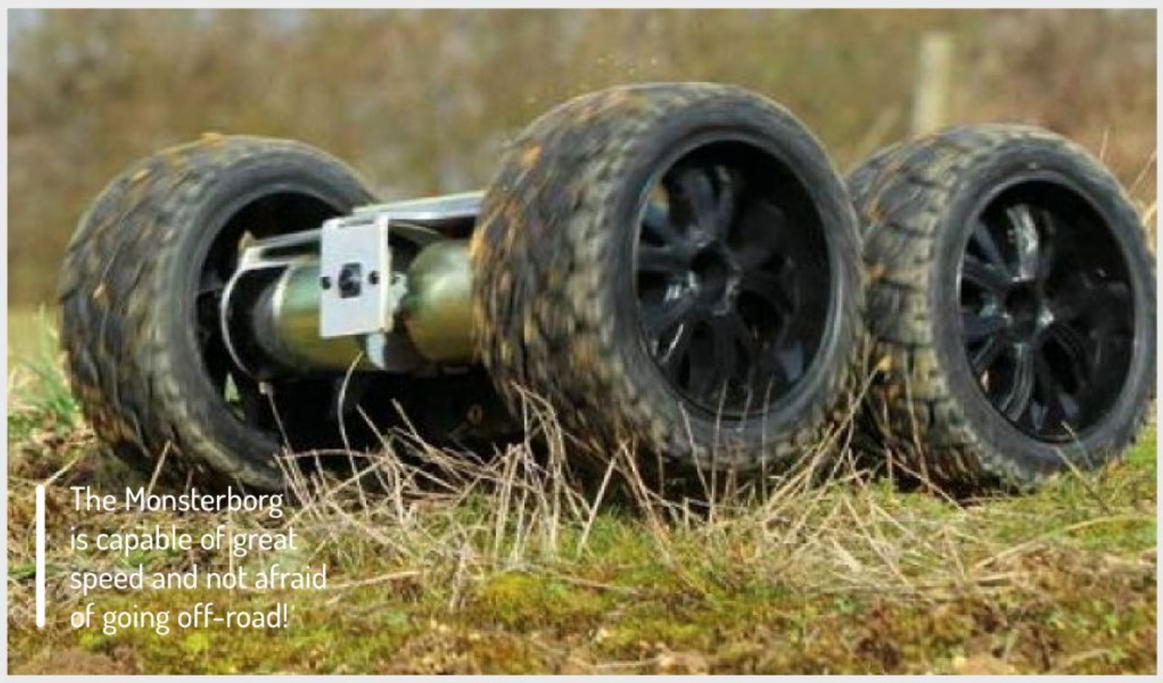
Bluetooth gamepads are easy to set up from the Pixel desktop. If only things were so simple from the command line...

» THE BORG SERIES

Resistance is futile. PiBorg's Borg series consists of three robots: the DiddyBorg (£210), the MonsterBorg (£210) and the YetiBorg (£160). The DiddyBorg's distinctive flat top makes it possible for all kinds of things to be mounted atop it. A terrifying robot claw, for example. It can be reconfigured with the Multifunction Top to fit an additional Pi, or with the Touch Top to fit the official Raspberry Pi touchscreen. It can also be kitted out with front and rear Pi Cameras, as well as ultrasonic sensors for obstacle avoidance.

Then there's the MonsterBorg, a machine not to be trifled with. Its chunky four-inch wheels enable it to chew up off-road terrain. As you may have read in the News section, PiBorg organises the Formula Pi racing event, and racing is what the MonsterBorg does best. Its 3mm -thick aluminium chassis ensures that it can survive all but the messiest of crashes.

Finally, there's the nimble YetiBorg. Powered by a Pi Zero (included) and a single 9V battery, the YetiBorg uses the custom ZeroBorg motor controller. This enables independent control of its four motors. It can be assembled in two different configurations: low-profile and off-road. In low-profile mode, should it accidentally flip over then it's perfectly capable of continuing upside down. If a little more clearance is required, then the off-road mode will keep the chassis clear of bumps and the electronics shielded from puddles.



The Monsterborg is capable of great speed and not afraid of going off-road!

scan with `scan on`. You should see a list of all nearby Bluetooth devices and their MAC addresses. Hopefully your controller is in there, in which case copy the address. Deactivate the scan with `scan off`. Then pair with `pair <MAC address>`, connect with `connect <MAC address>` and take your leave with `exit`.

Now run `evtest`, which will greet you with a list of detected input devices. Select your desired controller and mash the buttons. You should see a different cryptic response for each button. The DiddyBorg includes an example script for joypad control, which uses the PyGame libraries to listen for the relevant button events.

Image recognition

When it comes to machine learning, it's a good idea to run *Tensorflow* on the Pi. This is all thanks to the work of Sam Abrahams, who's provided precompiled wheel files for Python 2.7 and 3.4. This is good news if you're running the second-to-last (Jessie) version of Raspbian, since that includes Python 3.4. If you're running the latest version (Stretch, which uses Python 3.5), however, then you'll need to use the Python 2.7.

Having two different major versions of Python like this is fine (Raspbian ships them both as standard), but one cannot have 3.4 and 3.6 installed concurrently, and the 3.4 wheel won't work with Python 3.6. Before we begin, be advised that Tensorflow models' repository is large and more than once we ran out of space using an 8GB SD card. This can be worked around, by removing larger packages such as *LibreOffice* and *Wolfram Alpha*, but using a 16GB card is recommended. The following commands will set up everything you need:

```
$ wget https://github.com/samjabrahams/tensorflow-on-raspberry-pi/releases/download/v1.1.0/tensorflow-1.1.0-cp27-none-linux_armv7l.whl
$ sudo apt install python-pip python-dev python-pil python-matplotlib python-lxml
$ sudo pip install tensorflow-1.1.0-cp27-none-linux_armv7l.whl
$ git clone https://github.com/tensorflow/models.git
```

This last stage will start a roughly 1GB download, so beware. If you run out of space the process can be resumed, once you've cleared some clutter, by running `git checkout -f HEAD` from the **models/** directory. Once it completes successfully, test it with:

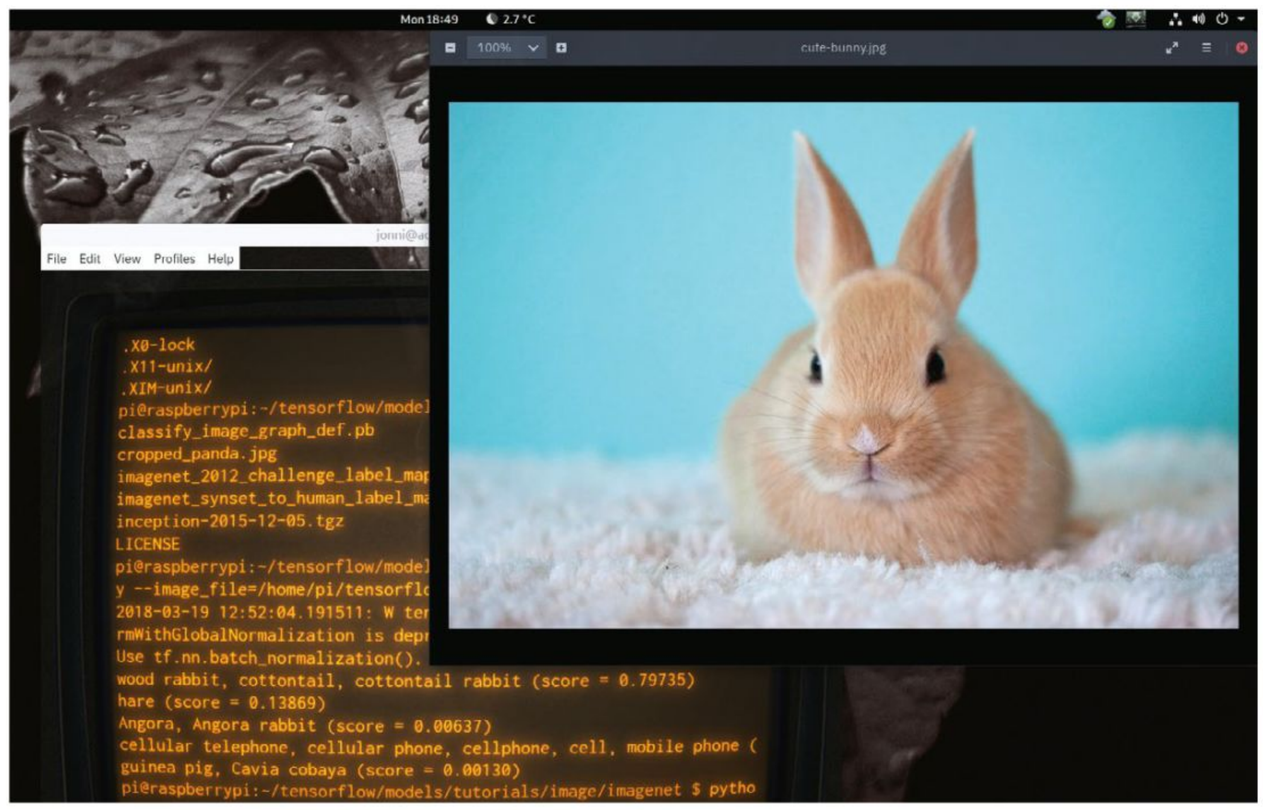
```
$ cd models/tutorials/image/imagenet
$ python2 classify_image.py
```

This should identify the bundled panda image (which was extracted to **/tmp/imagenet/cropped_panda.jpg**). The script can also take an `--image-file` parameter to identify user-supplied images. So with a bit of jiggery-pokery we could adapt things to take a photo and then attempt to identify it. Since the whole process takes about 10 seconds on a Pi 3 (although this could be sped up by keeping the program running in a loop, using C++ or with a more slimline Tensorflow model) we don't really have any hope of classifying things in real time.

Furthermore, it's likely that a photo taken at ground level with the Pi Cam will be tricky for the script to identify. But that's okay, it just adds to the fun. All we need to do is tweak **classify_image.py** a little. Copy this file to **classify_photo.py**, then edit the new file. You'll need to import the `picamera` module early on, then in the `main(_)` function replace the line that begins with `image =` with something like:

```
cam = picamera.PiCamera()
cam.capture('/tmp/picam.jpg')
```

And finally, change the `run_inference_on_image()` call to run on our freshly captured **picam.jpg**. If you're feeling adventurous then why not bind the new script to some button press event? The controller script for the DiddyBorg, which we discussed earlier, could easily be adapted to do this.



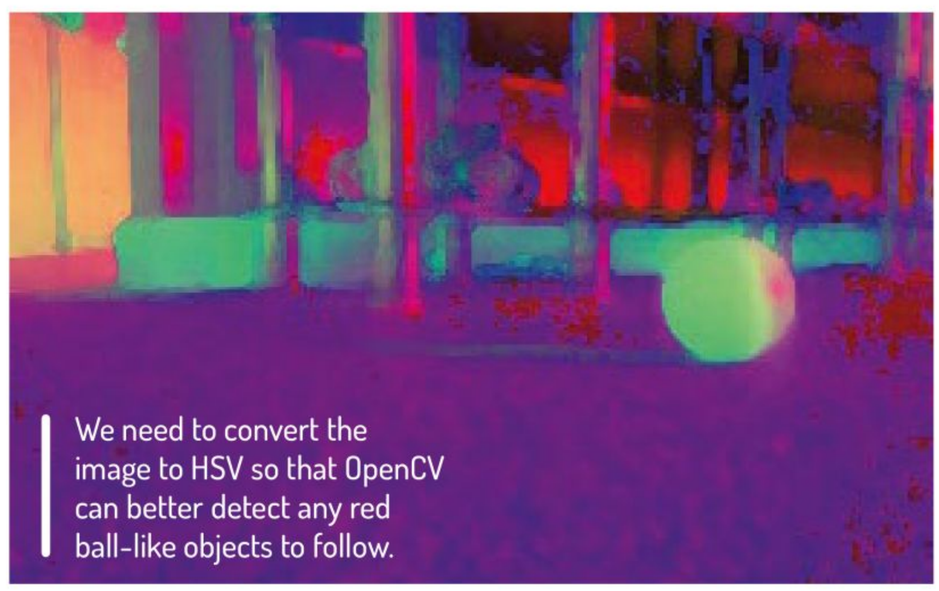
Tensorflow figured that this was a rabbit, but then thought Jonni was a sweet potato, Neil a bow tie and Effy a punch bag.

FOLLOW THE ROBOT ACTION

"If your robot has a camera attached, then you can stream the video for a delightful first-person video driving experience"

Ball following

OpenCV is a powerful computer vision framework that includes Python bindings. It's actually used to draw the image in the earlier Web UI example, but we can use it for more advanced purposes. It's capable of detecting objects within an image, which means we can make our robot drive towards them. If those objects move, then it will follow them. For this to work, said objects need to be fairly distinct, such as a brightly coloured ball. You'll find just such an example at **www.piborg.org/blog/diddyborg-v2-examples-ball-following**.



We need to convert the image to HSV so that OpenCV can better detect any red ball-like objects to follow.

Kit out your robot

There's a whole world of weird and wonderful widgetry that can be fitted to your robot to add functionality and bring the fun!

A simple Pi robot needs little more than a chassis, power source and wheels. But things are more fun when you kit the base construction out with some additional peripherals. We can use well-known Pi peripherals, such as the camera and the touchscreen, in new and innovative ways, or we can use other items – sensors, speakers, servos and more – to really ramp up the excitement levels.

Grab yourself a controller

Controlling a robot via a web interface is all well and good, but even on a fast network you'll experience some degree of latency, which will make driving a little haphazard. A wired controller will work reliably, but obviously this won't be much use if you can't keep up with your robot. With a Bluetooth controller, you can enjoy much more precise control of your robot friend. PlayStation 3 joypads are very popular when coupled with the `sixpair` utility. See **www.piborg.org/blog/rpi-ps3-help** for details. There are a number of other options though, and many of them will work out of the box with Raspbian. Class 1 Bluetooth tooth devices in theory have a range of up to 100 metres, but these are subject to interference and dropouts.

GIVE YOUR ROBOT SIGHT

"If camera flexibility is required then a variety of pan-tilt kits are available for pitching, rolling and yawing to the ideal angle"

We found this handy little Bluetooth gamepad discarded in a dusty corner at Future Towers. It came in surprisingly handy for this feature.

Also, if you don't have a Pi 3 or a Bluetooth adaptor then this isn't an option anyway, so it may be worth considering using an alternative means of transmission and reception. The go-to option here is to use radio frequency (RF) in the 2.4GHz band. This requires a more powerful transmitter, such as what's used in drones and other radio-controlled aircraft.

Fit a Pi Camera

Whatever you want your robot to do, you can't go too far wrong by giving it the gift of sight. This can easily be done with the Pi Camera board, available from all good maker shops. Launched in 2013, the Pi Camera Module was one of the first official add-ons for the Pi and has found its way into all kinds of amazing projects. A new eight-megapixel edition came out in 2016, and a mini-edition for the Pi Zero followed suit.

These cameras connect via a ribbon cable to the bespoke CSI interface, leaving the GPIO pins free for other gubbins. A mount is available so that the camera can be easily and securely attached to wherever it needs to go, and if flexibility is required then a variety of pan-tilt kits are available for pitching, rolling and yawing to the ideal angle. There are also telephoto, fish-eye and wide-angle lens kits available. We saw on the previous pages how libraries such as OpenCV can be used to detect objects using camera input. If you want to incorporate the Pi Camera into your camera projects, then this one-liner will help you:

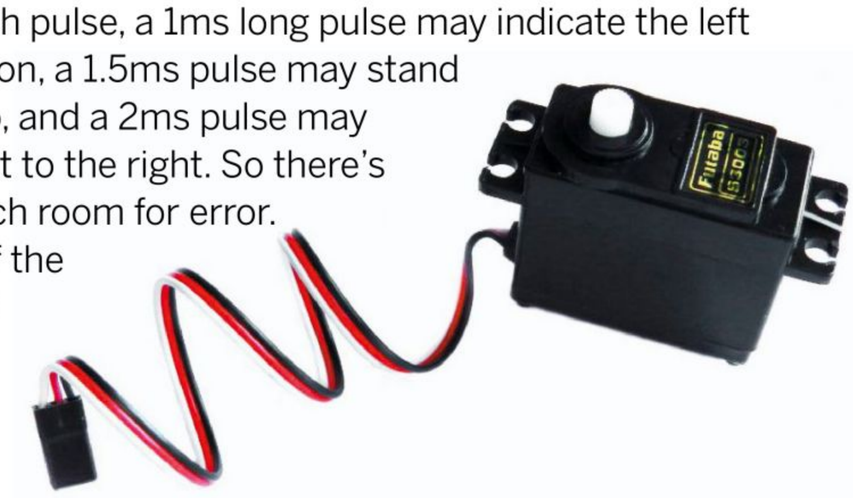
```
$ sudo apt install python-picamera
```

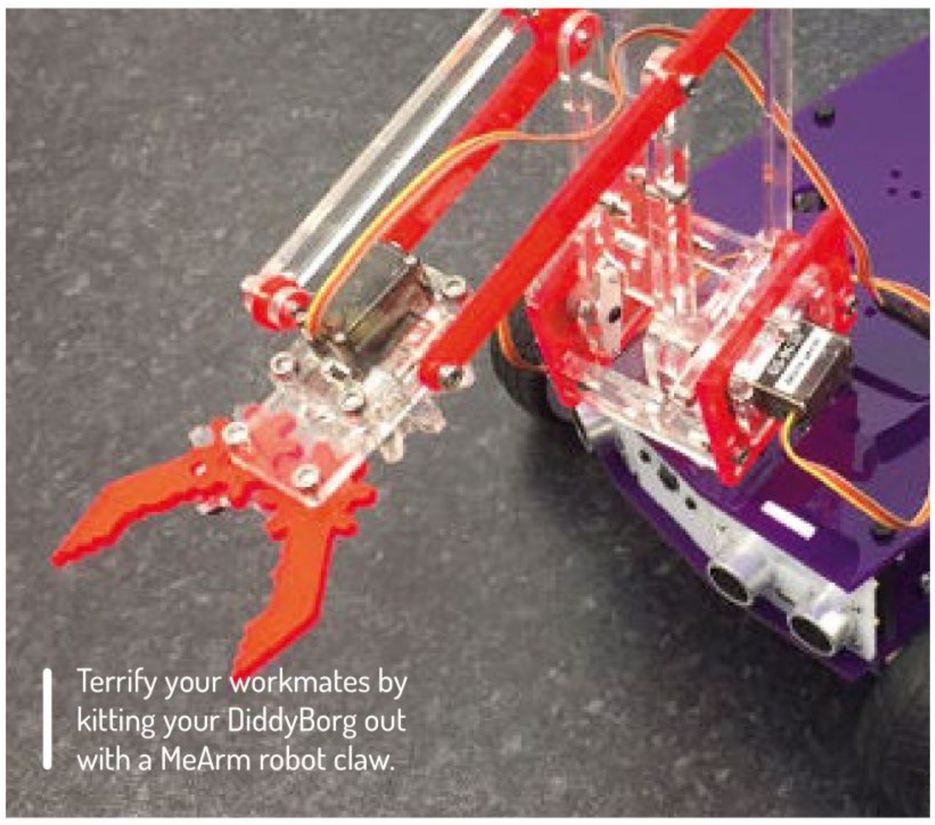
Add motion using servos

Servos are motors (*actually actuators – Ed*) that often have a limited movement range, for example, to move the rudder of a radio-controlled boat, but some can, or can be adapted to, do continuous rotation.

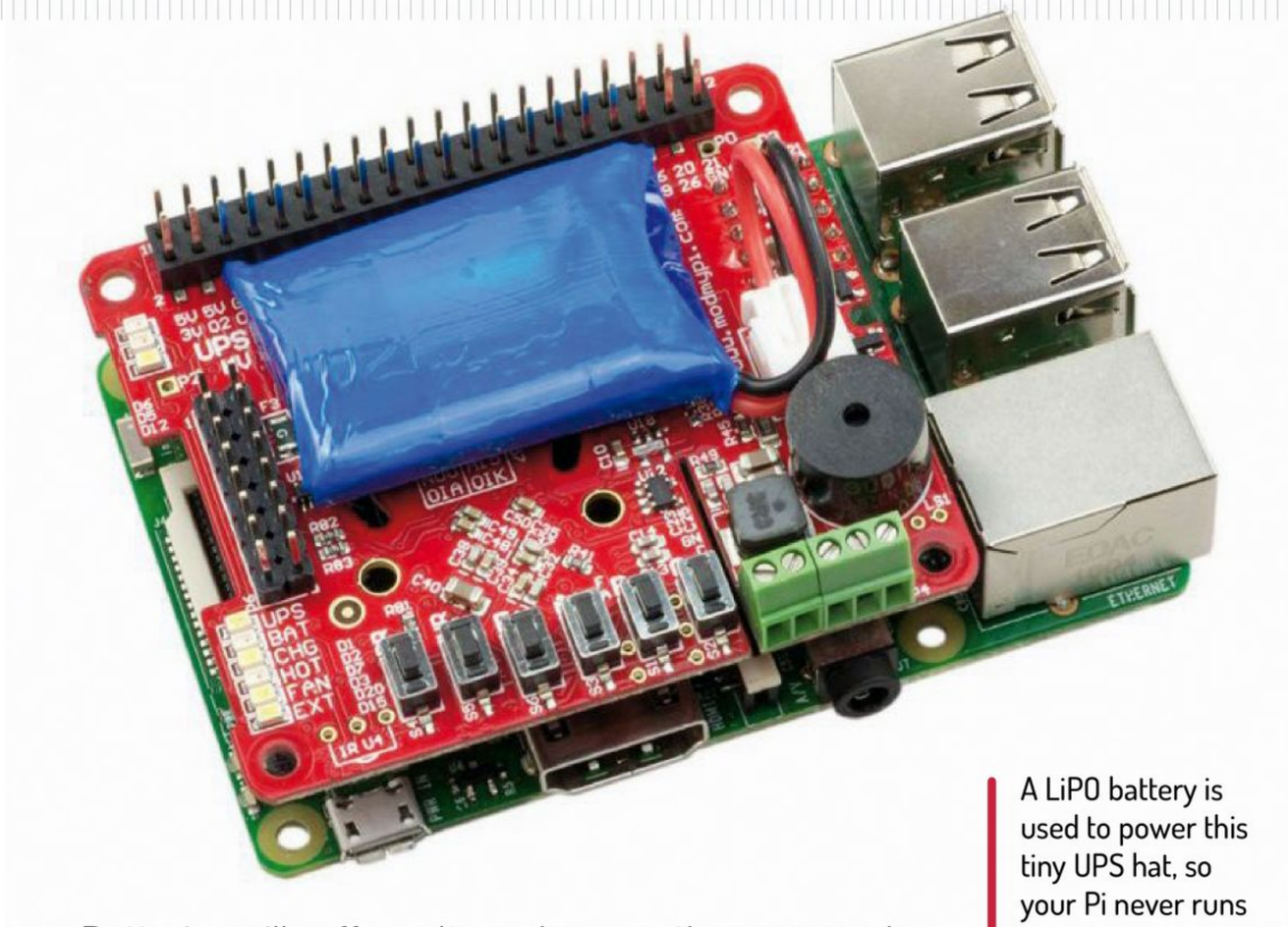
Servos are controlled by pulse-width modulation (PWM). These pulses need to be pretty accurate, otherwise the servos will get confused. For example, a servo may have three different positions: left, up and right, say. The position is determined by the length of each pulse, a 1ms long pulse may indicate the left position, a 1.5ms pulse may stand for up, and a 2ms pulse may move it to the right. So there's not much room for error.

One of the Pi's GPIO





Terrify your workmates by kitting your DiddyBorg out with a MeArm robot claw.

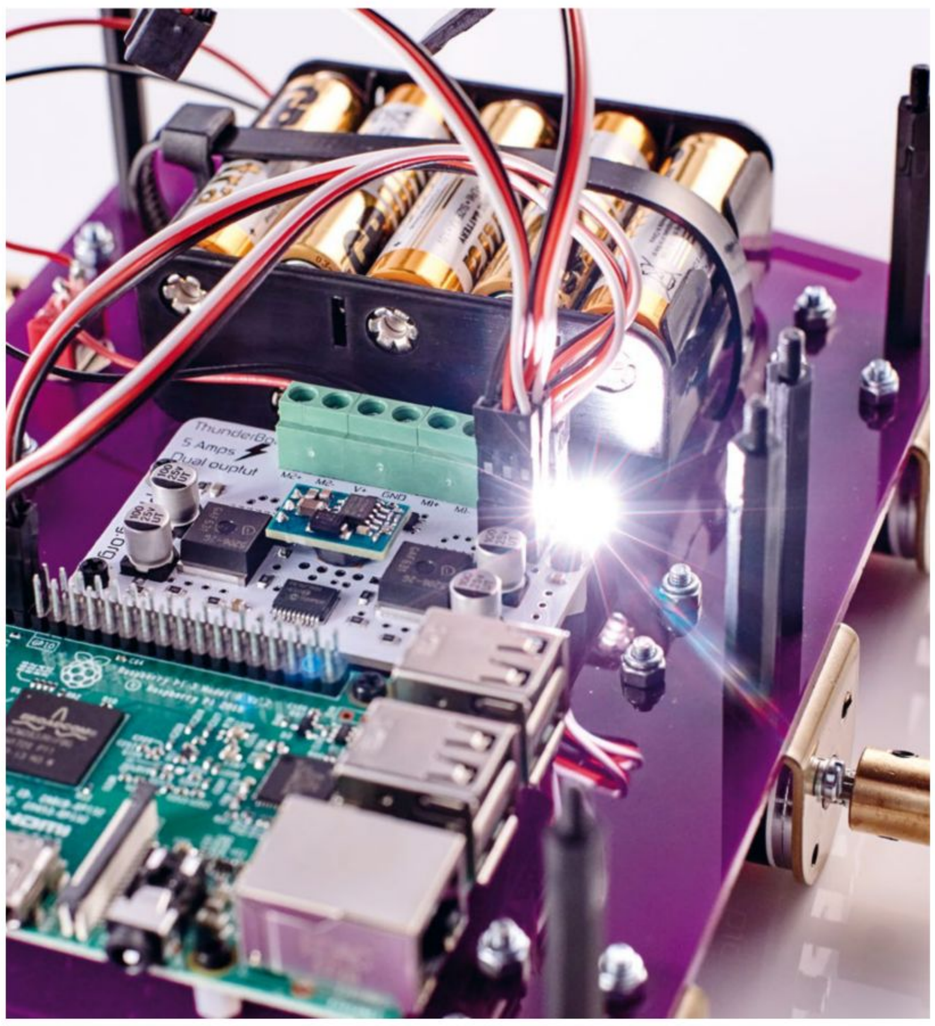


A LiPO battery is used to power this tiny UPS hat, so your Pi never runs out of power.

pins is connected to a hardware pulse generator, so that these pulses can be generated without being interrupted, or elongated by a busy OS. However, sometimes one pulse generator isn't enough and it's common to add a HAT or other expansion board (possibly even a whole new microcontroller such as an Arduino) to handle multiple PWM devices. See how the UltraBorg can achieve this over a web interface at **www.piborg.org/blog/build/ultraborg-build/ultraborg-examples-web-ui**.

Improved battery power

The Pi itself can distribute power to small devices through its GPIO pins. But these are limited to 3.3V and can carry very little current. So powering high-torque motors, lights or weapons (*say what now? – Ed*) through these, with the Pi connected to a standard USB power bank, isn't an option. We've seen that the DiddyBorg gets its juice from 10 AA batteries But the current portable power trend is definitely lithium ion polymer (LiPo) batteries. They provide longer life and higher power than their nickel metal hydride (NiMH) brethren, and also hold a charge for longer.

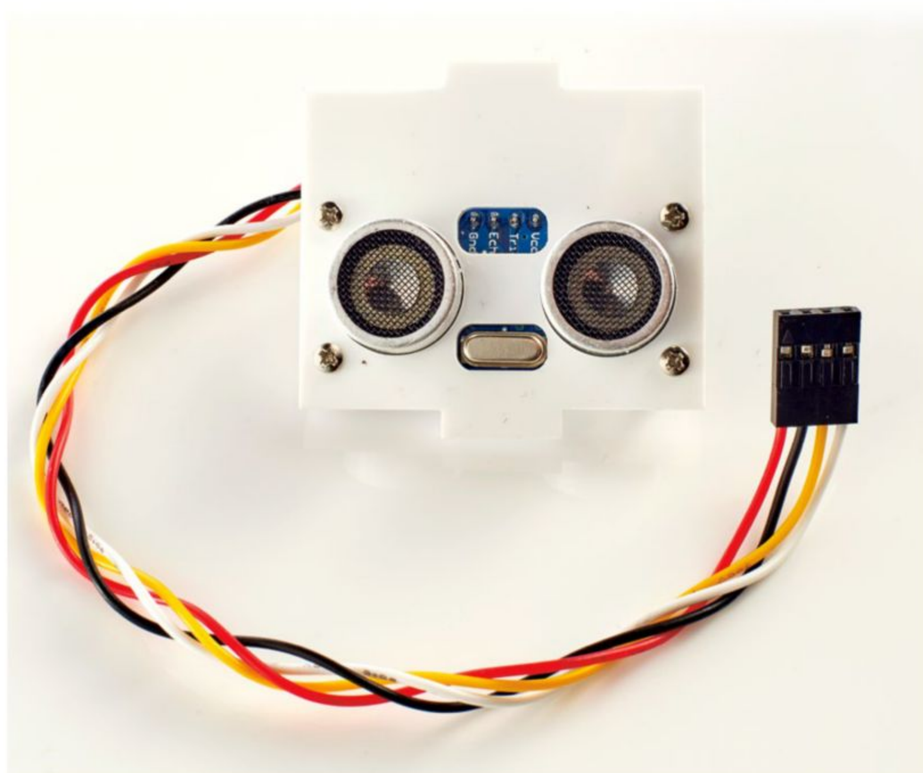


The ThunderBorg has a very bright and optionally colourful LED to gives a clear visual indication of battery status.

Batteries will suffer voltage drops as they approach the end of their lives, and some components (not least of which is the Pi itself) are pretty sensitive to these. So one of the important tasks that the ThunderBorg, which connects the DiddyBorg's battery pack to the Pi, performs is regulating the power supply. If too low a voltage is detected when the board is powered, then it will duly switch itself off. Tim Freeburn told us a bit more about this stylish creation: "We're very proud of ThunderBorg. The wide voltage range was a key part of the design from early on. ThunderBorg has a wide range of protection mechanisms such as undervoltage lockout, short circuit protection, thermal protection and overcurrent protection. Thanks to this, it's pretty hard to destroy one, as long as you make sure you connect the power supply the right way around!"

Ultrasonic sensors

Besides using a camera to assess its environment, our robot can also be like a bat and use ultrasonic chirps to map things out. An ultrasonic sensor, such as the HC-SR04 emits periodic, high-pitched pulses that can detect as they bounce off nearby objects. By measuring the time between emitting the pulse and detecting its reflection, and knowing that the speed of sound is about 340 metres per second, the sensors can judge distance reasonably well. Our DiddyBorg came with a kit for mounting these sensors at each corner. You could use such a sensor array to automate parallel parking.



The DiddyBorg can be kitted out with four Ultrasonic sensors to let it know if obstacles are approaching.

Watch the skies

Psssst, wanna get high with the Pi (in complete compliance with local aviation laws, of course)?

So far we've managed to keep our feet/wheels/caterpillar tracks on the ground, but we think we should finish our robotic journey with a brief foray into the world of drones.

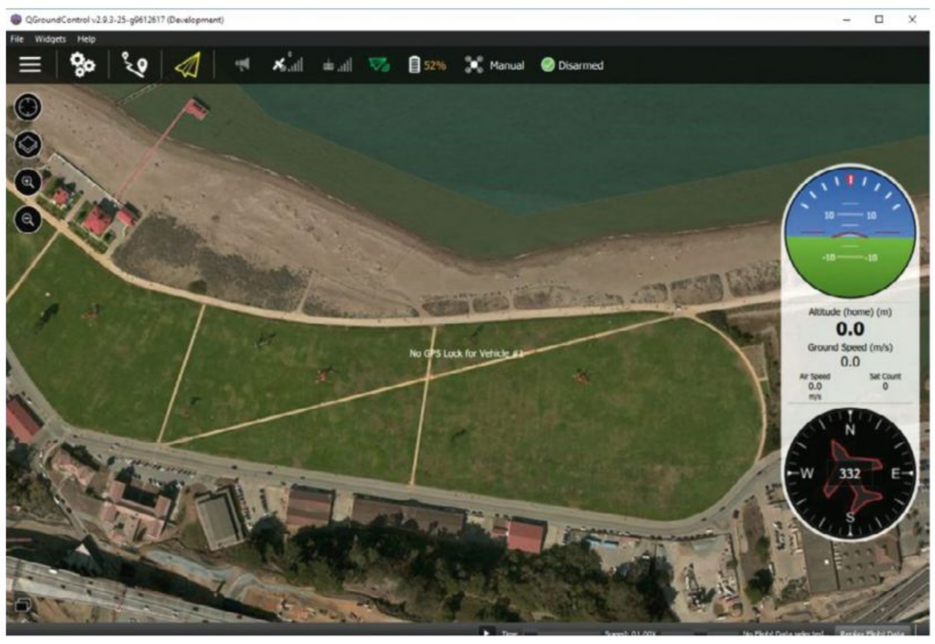
Keeping a drone in the air is a complicated business. Each motor has to be able to respond as close to instantly to changes in orientation. These are detected by a three-axis accelerometer. Other instrumentation is required, too: gyroscopes, servos, and of course motors to keep the propellers turning and generate lift. A vanilla Linux Kernel isn't built to respond within such operational deadlines as this, so it's common for drones to run a real-time OS (RTOS) such as NuttX, or indeed a proprietary OS. The Linux kernel can be tweaked to get closer to such response times though. The PREEMPT_RT patchset is probably the most popular way to achieve this. This patchset, having gone for a long time without funding, is now part of the RTL Collaborative Project, an industry-sponsored initiative under the Linux Foundation. See **https://wiki.linuxfoundation.org/realtime/rtl/start** for more information.

Erle Robotics (**http://erlerobotics.com**) makes the Erle Brain: small Linux computers capable of fulfilling all the functions required to keep a drone in the air. They include all the sensors and expansion ports you could imagine. Erle also makes the PXFMini, a hat for the Raspberry Pi that uses Dronecode's APM Flight stack. Dronecode is an open source platform that's fast becoming an industry standard. The stack consists of the PX4 autopilot, the Mavlink communication protocol, QgroundControl software and runs on a huge variety of hardware. Read more at **www.dronecode.org**.

Even if drone makers often would rather have some other OS flying their aircraft, it doesn't preclude having a separate Linux machine on board. This may even be advantageous, since it enables all of that machine's resources to be devoted entirely to fun tasks, perhaps 360 degree photography via a gimbal-mounted camera.

It feels like we've barely scratched the surface of Pi robotics. PiBorg founder Timothy Freeburn tells us he's "working on Pi-based robots for mining simulations with a big Australian mining company, robots for aircraft intake and exhaust duct inspection, and robots for pipeline inspection. The list is growing rapidly!"

He also points us in the direction of some of his favourite community projects: Pi In The Sky, The Big Hack, and Liam Kennedy's ISS-Above (see **http://issabove.com/iss-above-and-the-raspberry-pi**). Thanks Tim. And thank you, dear reader. Now off you pop and build something amazing!



QGroundControl enables you to keep track of your drone and map out flight paths for your important stealth missions.

» AERIAL FLORA PHOTOGRAPHY

The Pi NoIR camera has no infrared filter, which enables it to 'see' in the dark, making it useful for CCTV applications. The retail package also includes a slightly quizzical blue gel square. When that blue square (Roscolux #2007 Storaro Blue to give it its full designation) is placed in front of the lens, red and green light are filtered out, and the sensor shifts infrared light to the red spectrum, enabling hyperspectral images of plants to be captured.

The actual photo taken is an NGB (Near infrared Green Blue) image, which doesn't reveal very much. But with a bit of post-processing magic, a composite NDVI (near difference vegetative index) image can be generated, which shows how well the plant is photosynthesising, or more crudely, how healthy it is. This technique used to require specialist equipment, mounted on aircraft and satellites. For serious applications it still does, but have a look at the infragram projects at **https://publiclab.org/wiki/raspberry-pi-infragram**, and in particular Matthew Lippincott's Flickr page at **http://bit.ly/raspberry-pi-infragram**, to learn more about what can be achieved with the humble Raspberry Pi.

After post-processing the NGB image (left) to produce an NDVI image (right), we can see where trees are thriving. (Image CC BY-SA 3.0 Public Lab.)

MAKE YOUR OWN PROJECTS WITH THE RASPBERRY PI

Learn the electronics, computing and coding skills you need to make your own projects with the Raspberry Pi, and let your imagination run wild

FUTURE

Ordering is easy. Go online at:

www.myfavouritemagazines.co.uk

Or get it from selected supermarkets & newsagents

4tronix PiBug 2WD

Les Pounder takes a look at a simple robot that can be built in less than 30 minutes without any special tools.

IN BRIEF

Designed for the Raspberry Pi A, A+, B and B+ boards, PiBug is a small and easy to build robot that uses the Raspberry Pi board as a central part of the chassis. No soldering or expensive tools are required – just 30 minutes and a screwdriver.

Robots and the Raspberry Pi go hand-in-hand. There are countless kits on the market and all are vying to be the 'best' or 'easiest' to buildand use. 4tronix has a long history of building robots specifically for the education market, and with PiBug we see their latest robot in its growing army.

PiBug is a little different to other robot builds. Rather than include a chassis onto which we connect our Raspberry Pi, the Pi becomes a major part of the chassis. Coming as a collection of three circuit boards, motors and screws, PiBug is held together via a series of brass columns that connect through the four screw holes around the edge of the Raspberry Pi.

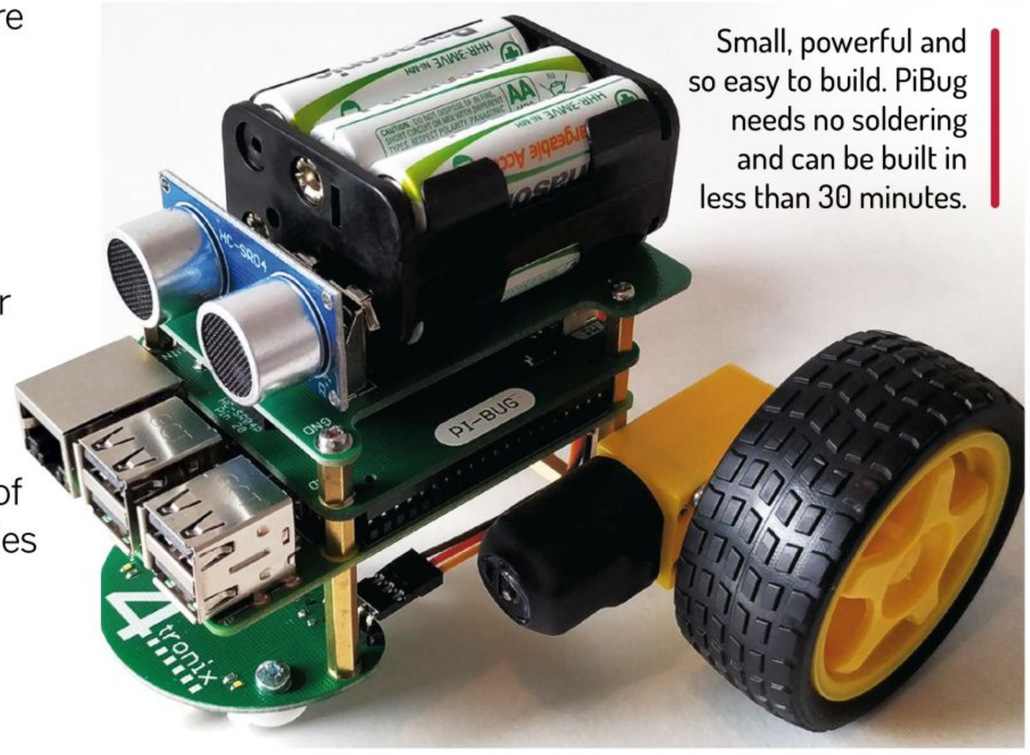
Via a short series of instructions on the website we were able to build our robot in less than 20 minutes, including the wiring of two extra sensors (line following and ultrasonic.) The build required nothing more than a screwdriver and a ruler to measure the length of components. No soldering is required and the kits can be broken down into their parts for reuse in class.

PiBug is designed for Raspberry Pi models A, A+, B and B+. That means your Zero or original Pi will not fit, unless you are determined and have some spare parts. Powering the Pi and the robot are six 1.2V rechargeable batteries, giving us 7.2V of power. This is the recommended configuration, as 1.5V alkaline batteries may introduce a reset bug where the Pi resets when making a turn or rapid change in direction.

The batteries are stored in a holder atop the robot, with a connector to the board locking them into place. Just make sure to use the included Velcro strap to hold the batteries in, as a crash dislodged our batteries, launching them across the room.

The two DC motors are controlled via a DRV8833 motor controller, which is part of the middle PCB in the sandwich. This controller is a staple in the robot community as it offers ease of use at a low cost – a great entry-level motor controller.

Software for the PiBug is provided via an install script on the 4tronix website, and while this may not be the most secure method of installing it, the code is fully examinable and installation is transparent to the user. Once installed we have a directory called **pibug** containing the library **pibug.py**, which has all the functions needed to control the motors and react to sensor input.

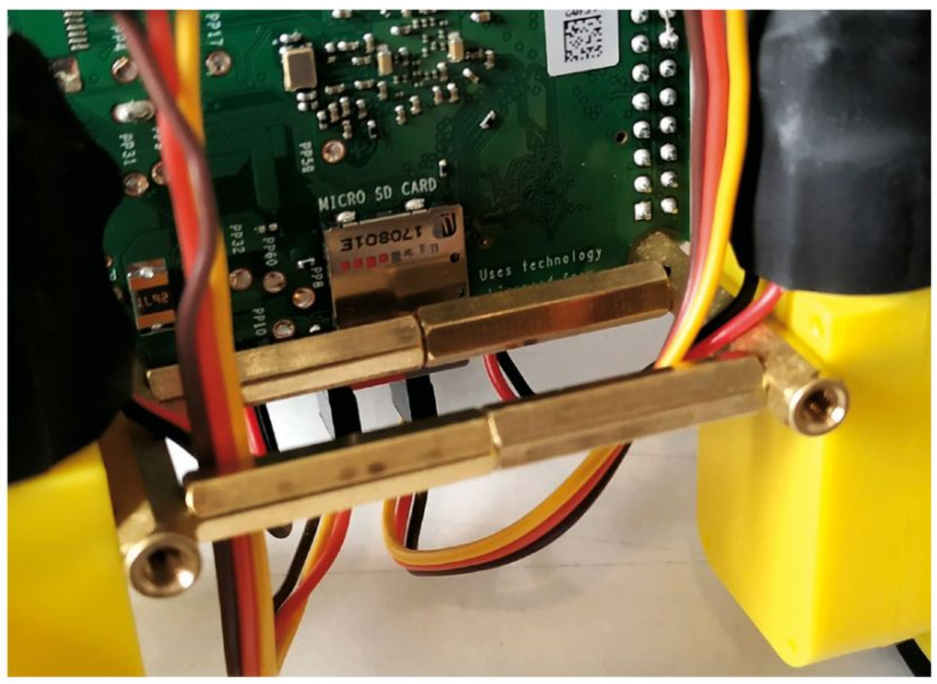


Small, powerful and so easy to build. PiBug needs no soldering and can be built in less than 30 minutes.

The library is written in Python 2, which is a shame as this will lose support in 2020. The code will work with Python 3, but you need to make a few changes to `print` statements, as in Python 3 they are written as functions with parentheses. Once this bug is ironed out the **pibug.py** library works well, and we can clearly see how to retrieve and store data from our sensors and control the direction and speed of the motors.

The PiBug costs £24 for the basic kit. Adding the extra sensors (which we used) takes the price to £32.15, which is a good buy considering how sturdy and easy to build it is. This is a ready-to-go solution for classrooms and those wanting to get started with robotics.

More advanced roboteers will need to look elsewhere, but as the target market for this robot is educational, 4tronix has hit the nail on the head! This is an ideal first robot that offers cost-effective, reusable parts.



Close up of the clever "no chassis" construction method.

VERDICT

DEVELOPER: 4tronix
WEB: https://shop.4tronix.co.uk
PRICE: From £24 (Model reviewed £32.15)

FEATURES	8/10	EASE OF USE	10/10
PERFORMANCE	9/10	VALUE	8/10

Ideal for the classroom, low-cost and easy to use, a solid little robot that offers plenty of functionality in a unique package.

» **Rating 8/10**

4Tronix Pi2Go Mk2

Les Pounder is tired of walking to the shops for his lunch, but with the Pi2Go Mk2 he can have his pies delivered!

IN BRIEF

A feature-packed robot that has sensors for obstacles, light, following lines and ultrasonic! It can drive itself using sensors to avoid obstacles, or be controlled using a Python library via Scratch. This is an intermediate robot kit for those that wish to progress into serious robotics. It also has Neopixels to ensure that everyone will be dazzled by your project.

Another robot enters the Raspberry Pi scene, and with it comes a pedigree. Made by 4Tronix, which has been producing Raspberry Pi robots since 2015, this is a long-awaited revision of the earlier Pi2Go robot from 2014. In fact the original Pi2Go was the first robot that we built, and it rests on a shelf in the office.

Pi2Go Mk2 is made for use with all 40-pin GPIO models of Raspberry Pi, which includes the Raspberry Pi 4. You will need to spend a little more money for the seven AA batteries required or a 18650 board in order to power everything. The Pi2Go can also be used with a Pi Zero W, but it will need a little alteration to make it work.

Building Pi2Go is fun and easy. There is no soldering required – all that's needed is the included screwdriver. The Raspberry Pi is sandwiched into the middle of the robot and it is safe and secure.

The key part of a robot is the motor controller which is controlled by the Pi and controls the high-current circuit for the motors. The controller is a DRV8833, which is a popular, low-cost yet serviceable component. It can even cope with four motors, which is handy as there is a 4WD upgrade pack.

Sensor-y overload

The Pi2Go is packed with sensors: there are two infrared obstacle sensors on the front corners, light sensors on every corner and line-follower sensors. There are two slots for breakout boards and can be used with ultrasonic distance sensors, plus there's a seven-segment display. But even more exciting is that it is compatible with Pimoroni's Breakout Garden add-on boards. Why is that exciting? Well, using Pimoroni's 6DoF motion sensor, we can build an extremely accurate robot.

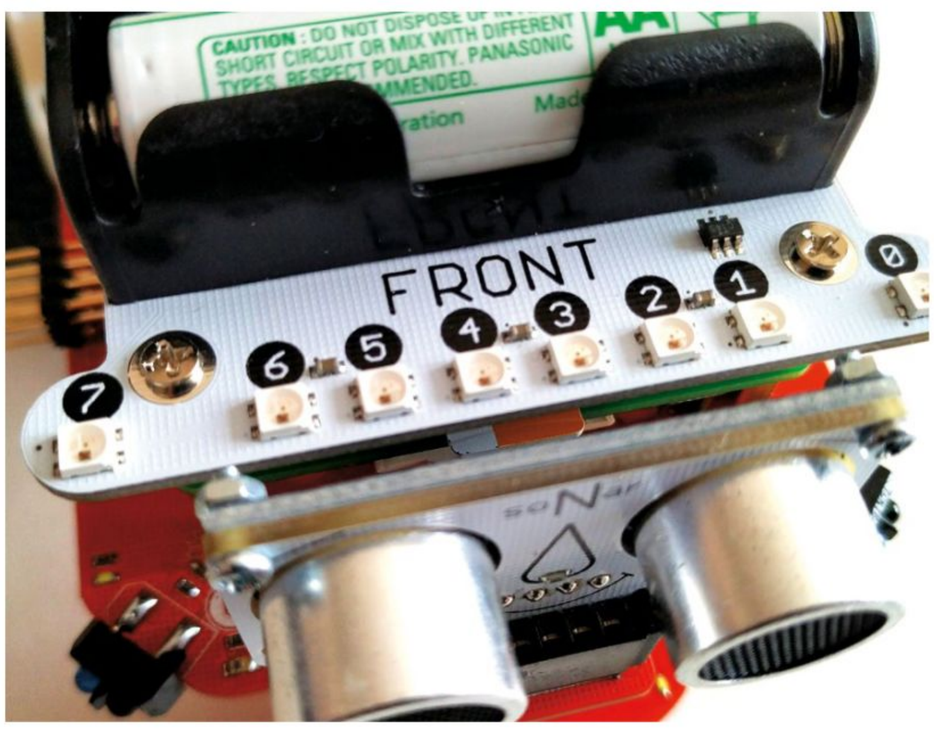
On the subject of hardware, the battery board features 10 SK6812-3535 RGB LEDs which are Neopixel-compatible, so that means your robot can blast out beams of light as it takes over the world.

The software for Pi2Go is all Python – Python 2, in fact. This is a bit disappointing, but after a few tweaks to the example code we managed to make it work with Python 3 – all it took was changing a few print statements to match the new syntax. Otherwise the supplied software is excellent, providing test scripts for every part of the robot, from sensors to motors. We can even monitor the state of the batteries from Python.

Easy to build, strong, powerful and programmable in many different ways. This is a serious robot kit.

When we are ready we can use the Pi2go Python library to create our own projects as the library is impressively high-level and easy to understand. For younger makers, the Pi2Go can also be controlled using Scratch on the Pi thanks to great work by Simon Walters, a key member of the Raspberry Pi community. Because the Pi2Go works with the GPIO, it can also be controlled using Node-RED, and the Pi2Go website has details of which pins are used.

This is a great piece of kit, but there is a downside and that is the price. The basic kit is £72, with extra add-ons retailing for around £11 each. You can build a robot for a cheaper price, but you will not have the same package of sensors as those found on Pi2Go. But if you have already built a robot and want to up your game, this is a kit to invest in. Despite the cost, this is an impressive update to a classic robot kit.



It even comes with twin exhausts. (No it doesn't – Ed)

VERDICT

DEVELOPER: 4Tronix
WEB: https://shop.4tronix.co.uk/products/pi2go2
PRICE: £72

FEATURES	9/10	EASE OF USE	9/10
PERFORMANCE	9/10	VALUE	8/10

We love robots, and the Pi2Go Mk2 is an instant classic and a yardstick from which others will be measured. We just wish the price were a little lower.

» **Rating 8/10**

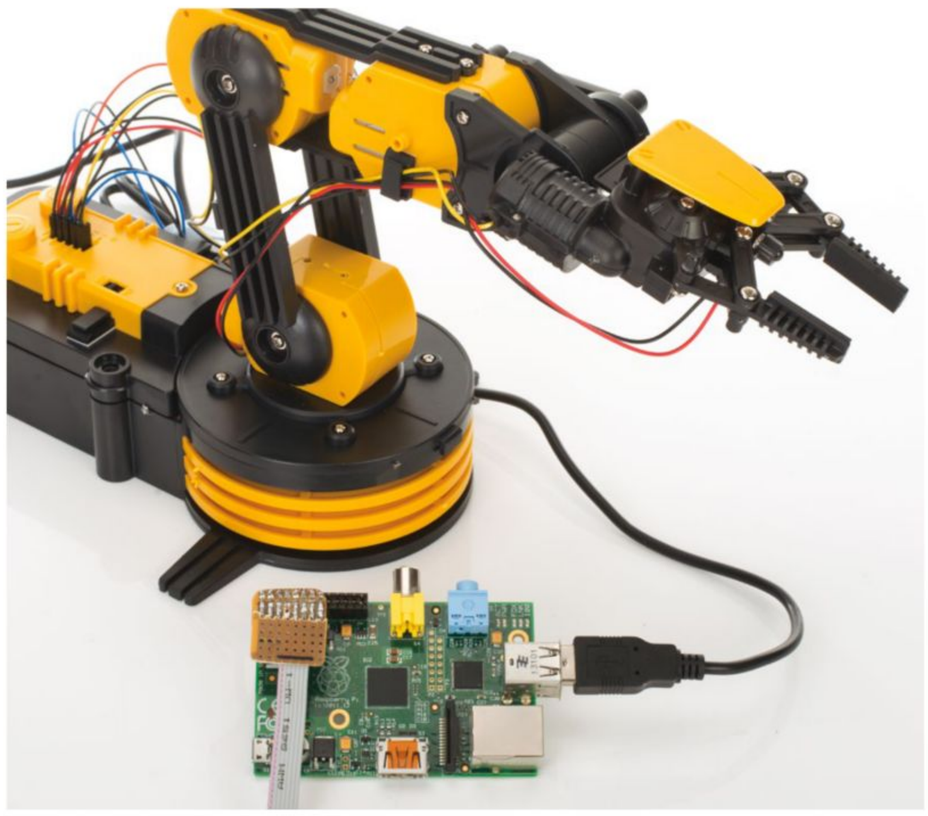
MPU-6050 Containing a MEMS accelerometer and a MEMS gyroscope, this sensor reads the x, y and z axis channels with 16-bit ADC conversion

Robot Arm Available from Maplin Electronics and OWI Robotics, the arm comes with software for control even before you get the MPU-6050 involved

Veroboard Veroboard is great to tidy up wires in projects like this, where they get in the way but can't really be run through a breadboard

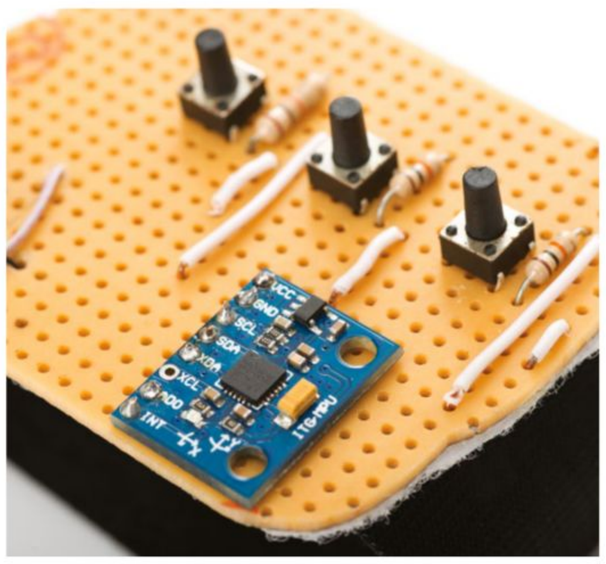
Components list

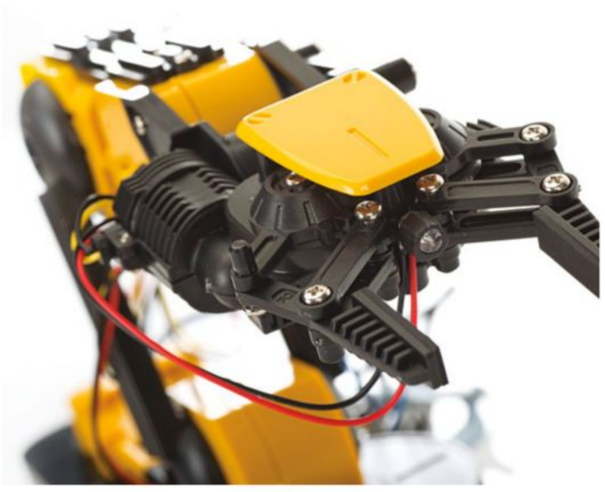
- Raspberry Pi Model B
- Maplin Robotic Arm Kit With USB PC Interface
- MPU-6050 Six-Axis Gyro and Accelerometer
- 3 Mini Push Button Switches
- Veroboard
- Velcro strap
- 1m Ribbon Cable



Left This robotic arm is one of the most used ones and there are tonnes of guides for it

Below One of these buttons controls the light on the end of the robotic arm, while the other two open and close its gripper





How I made: Robot Arm

Get to grips with natural motion control

What first inspired you to begin your robot arm project?

The robot arm itself was one I'd seen years ago and I really wanted it because it's something you can control yourself – it really captured my young imagination. I was volunteering at a science museum down here in Harlow and this club based around the Raspberry Pi sprung up, and I bought the robot arm because I wanted it. So then I had the Raspberry Pi thing going on at the same time and thought, why not meld the two?

I had this complicated system of key presses to get it to do anything, which was a bit boring, and then James Dali (one of the people who helps out with the club) gave me the idea of shoving an accelerometer on the top of it to give an idea of where it is. I took that and thought, 'What if I had the accelerometer on me and sort of used it to mirror the motion of my hand?' So I looked around, searched up the accelerometer he was using (the MPU-6050) and then found it for about £5 on eBay – it's normally about £30 from SparkFun but I'm on a student budget... A lot of the code I've used is borrowed but open source, and people have said it's fine, so then I went through and had two programs – one that could control the arm, one that took the input in from the accelerometer – and kind of just smushed them together. It's not that nice to look at, but it works and that's all that really matters.

So what exactly are you reading with that MPU-6050?

There's the gyroscope and the accelerometer in the code I'd found – you can use one or the other, but the gyroscope is very good for degrees over time and it tends to drift, while the accelerometer is good for sudden turns and for measuring gravity. If you compare the two to each other then you can get a rough angle all of the time, so it's essentially the accelerometer and the gyroscope used together to correct the faults with one or the other. It's got two axes of motion – pitch and roll.

Take us through the code itself.

So in the first bit it finds where the actual I2C interface is and there's a quick setup – I've got three buttons on there to control the gripper and the lights, so it sets those up – and then there's a bit which is using the USB library to find the robot arm, then spitting it out if that's an issue. There are a couple of definitions for some functions to actually move the arm, so it's a little bit easier – each motor direction is a different binary number – and then there are more definitions for setting up reading data from the accelerometer and a bit of maths for making sure the gyro and the accelerometer are both giving the correct angle. Then there's this while loop with a try inside it that is just pulling the accelerometer for data, spitting out the maths stuff , before just checking that the angle given is within a certain range. If it is, move this motor left (for example), or if a button is pressed then it turns a light on. The only problem I've had with it is that to actually move it, it requires a change in angle – so there's not a continuous thing. I have to wave my hand a little bit, but there's that degree angle and if I trip it then it'll move around.

Have you considered adding any more forms of control?

Yeah, I've done a lot of research into this. In terms of other ways to control it, I quite like the intuitiveness of it – to rotate and move this arm you are moving your own arm, so that's something I've been focussing on and trying to get even more intuitive. Trying to get some sort of – I bought an Arduino at some point – trying to build an actual robotic hand and then spreading out from there. Eventually, my big plan – many, many years in the future – is to have an entire sort of human body that is controlled by the movements of the user, but that's a very large plan which I haven't put too much into just yet! But essentially, the prototype that people have done before is sort of having pot sensors – potentiometers – on the fingers just to measure the actual rotation and closing of the fist, then having that represented with servos and then possibly doing that with actual pieces of string to sort of emulate the tendons. So you'd have a single servo, or a couple of servos, in an arm bit that would pull string which would close each finger in turn.

Another idea, which seems to be one of the most viable, is having it completely brain controlled... There's a fair amount of interest in reading brain activity – you can do it with the NeuroSky, for example. There's quite a nice open source project which I might end up using because it has four inputs, so you can measure at least two things at once and that seems to be a fairly interesting place to go. It's expensive though, and if you're going open source then they have a lot of warnings on the websites saying that you do this at your own risk, this is not a medical product, you may fry your brain...

What is the next step then?

Further projects would probably be replacing the motors. Because it's motor-driven, it's timing-based, so having something with servos instead where I can have a definite angle would be a lot more useful, a lot more precise and wouldn't tend to go... one of the problems with it is that if you tell it to keep going in one direction, it will keep going in one direction whether it wants to or not, and there's this awful grinding of gears as it attempts to go in one direction and can't. So that will probably be a new arm, a new robot, trying to get it to be a bit more nice-looking and a bit more precise.

> Another idea is having the arm be completely brain controlled

Joseph Thomas
iis a student helping to run a Raspberry Pi club from a science museum in Harlow, where they have worked on projects ranging from a robot arm to a portable Pi.

Like it?

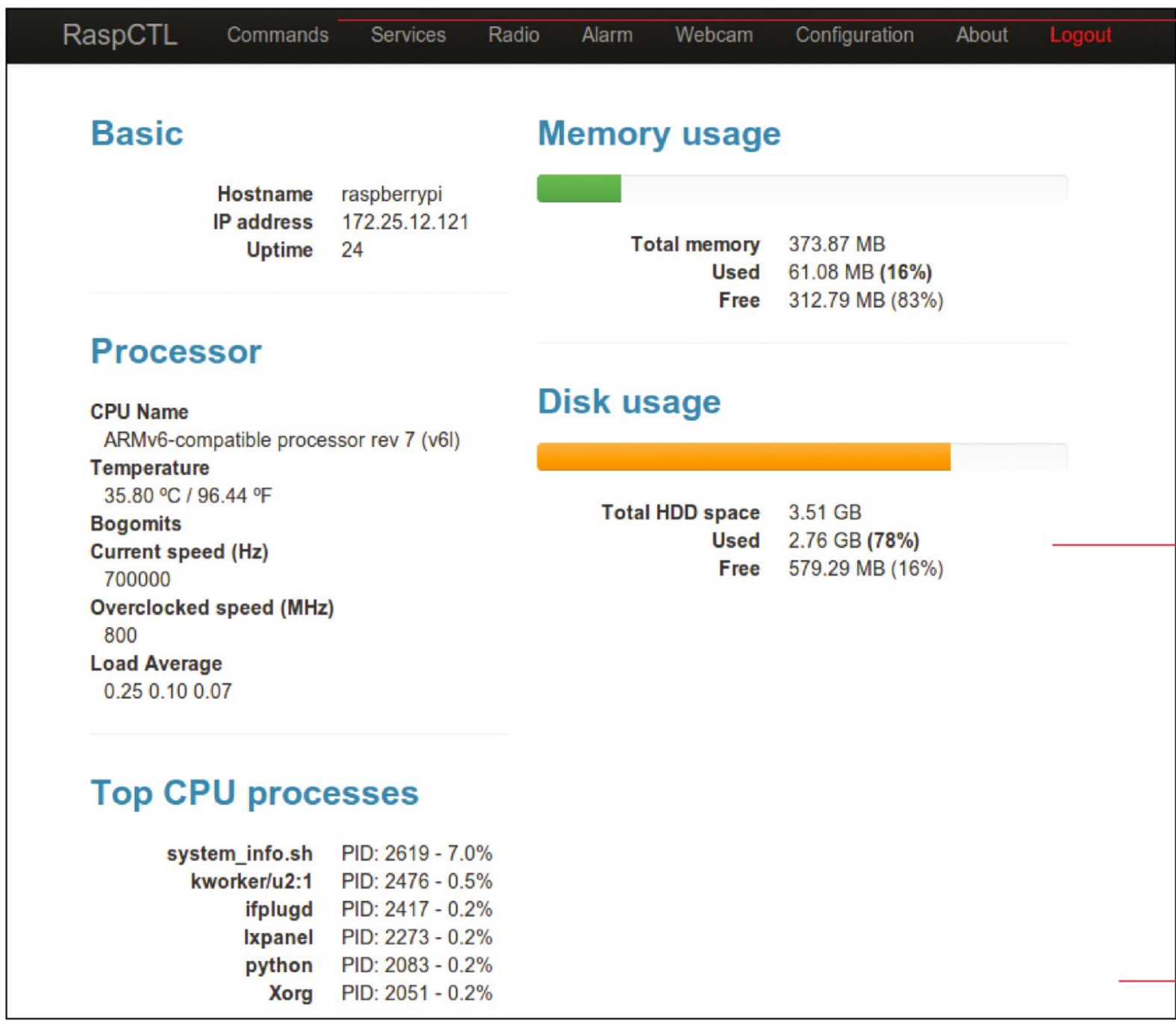
The robot arm that Joseph is using can be bought from Maplins in the UK (**bit.ly/1Da9BrT**) or ordered from Adafruit elsewhere in the world (**bit.ly/1yXlDQt**). There are many guides online to get you up and running, such as this one: **bit.ly/1AKd0OU**.

Further reading

NeuroSky has a whole product family dedicated to EEG and ECG biosensors, including the popular MindWave headsets (**neurosky.com**), and there are a few hacks available too (**bit.ly/1C7w0SP**). OpenBCI is a burgeoning open source project dedicated to brain-computer interfaces (**openbci.com**).

Remotely control your Raspberry Pi

Use a web interface to control your Pi and employ it as a fileserver or media centre from a remote location using any web-connected device.



Commands Create custom commands for running your Raspberry Pi

Other utilities Seeing through your webcam and setting an alarm are just two additional things you can do with your Pi

Main window Get the full details of the currently running system from the web

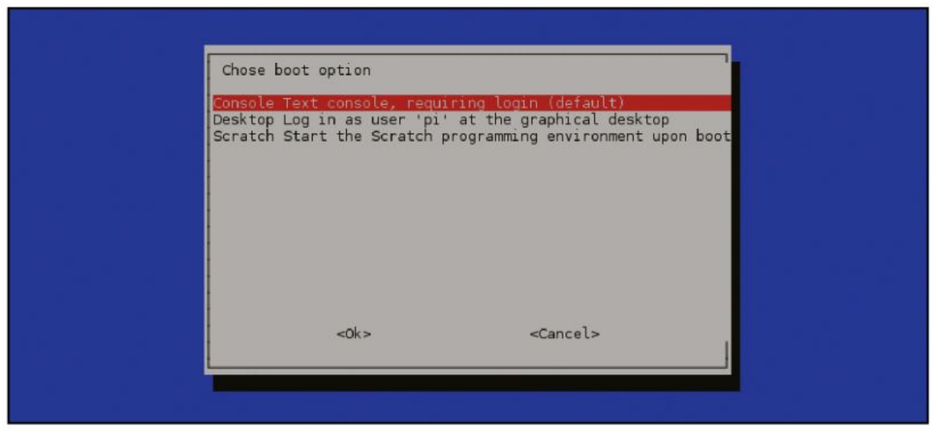
What you'll need

- Raspbian set to command line
- RaspCTL
- Internet connection

Not everyone uses the Raspberry Pi while it's hooked up to a monitor like a normal PC. Due to its size and excellent portability, it can be located almost anywhere that it can be powered and it's widely used as a file server, media centre and for other nontraditional applications as well. Some of these uses won't easily allow access to a monitor for easy updates and maintenance. While you can always SSH in, it's a bit slower than a full web interface that allows for custom commands and a view of the Pi's performance. We're using software called RaspCTL, which is still in development, but works just fine for now.

01 Update your Pi!

To make sure the Raspberry Pi works as best it can, you'll need to update Raspbian. Do this with a `sudo apt-get update && apt-get upgrade`, followed by a firmware update with `sudo rpi-update`. Finally, if you're booting to LXDE, enter `raspi-config` and change it to boot to command line to save power.

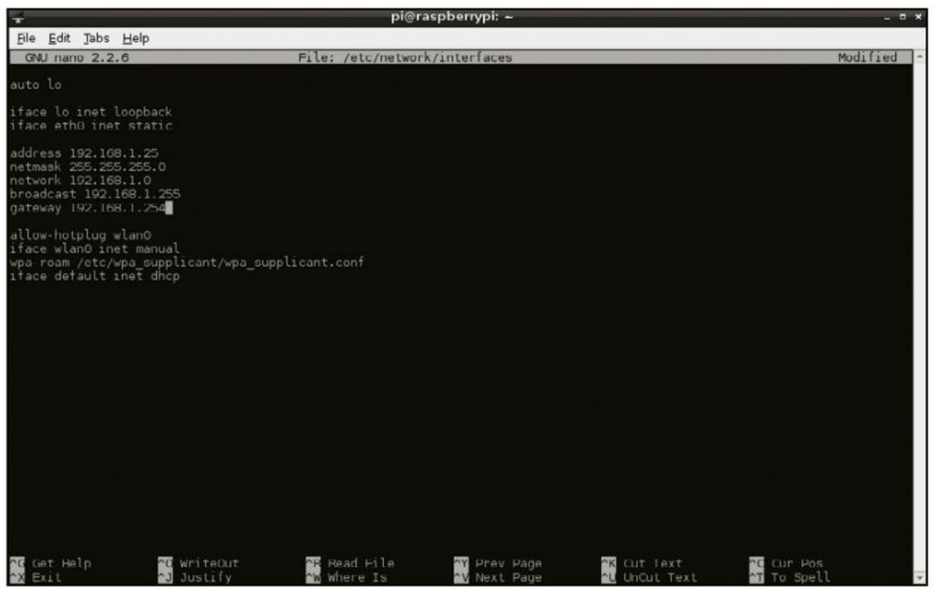


02 Edit the IP

For everything to work more easily, you should set the Raspberry Pi to have a static IP of your choice. To do this, edit the networking config by using:

```
$ sudo nano /etc/network/interfaces
```

...and change iface eth0 inet dhcp to iface eth0 inet static.



03 Set up a static IP

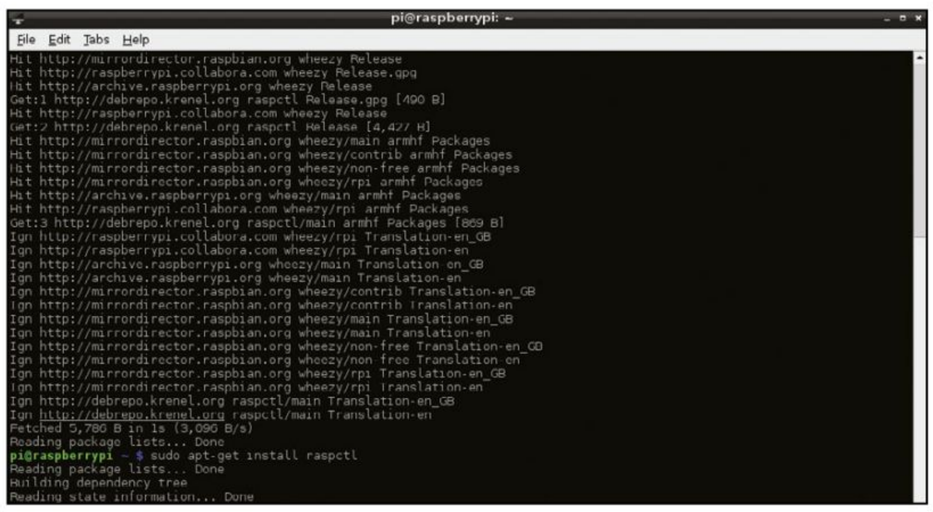
Add the following lines under the iface line with your relevant details:

```
address 192.168.1.[IP]
netmask 255.255.255.0
network 192.168.1.0
broadcast 192.168.1.255
gateway 192.168.1.[Router IP]
```

04 Ready to install

You'll need to grab the public keys for the software we're going to install by using the following commands. The first will take just a moment to download the software, while the other quickly installs it:

```
$ wget debrepo.krenel.org/raspctl.asc
$ cat raspctl.asc | sudo apt-key add -
```



05 Add the repository and install

Add the repository to the source's file with the following command:

```
$ echo "deb http://debrepo.krenel.org/ raspctl
main" | sudo tee /etc/apt/sources.list.d/raspctl.
list
```

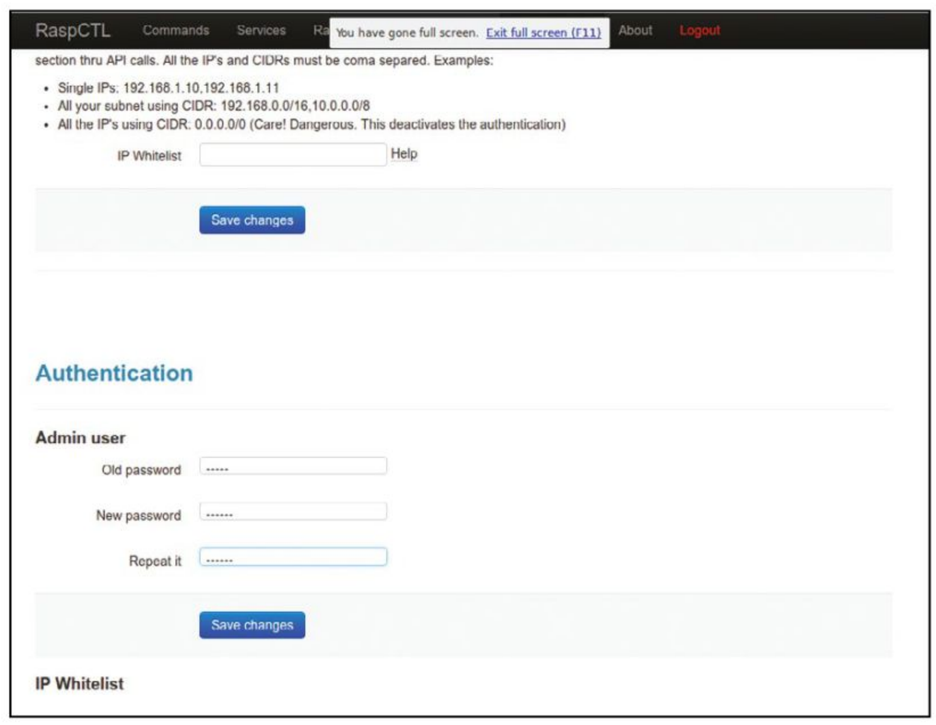
...and finally install the software with:

```
$ sudo apt-get update
$ sudo apt-get install raspctl
```

06 Access your Raspberry Pi

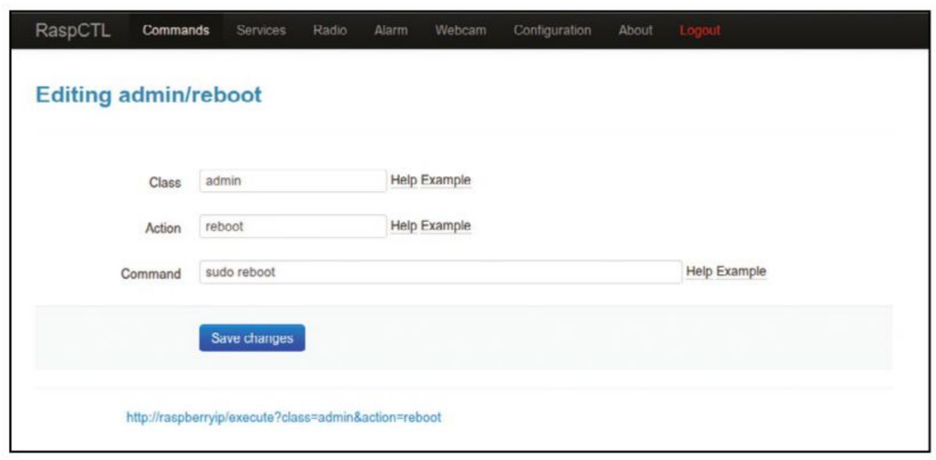
Now the software is installed you can start to access your Raspberry Pi from anywhere on your network. To do this type the following into your address bar, with the IP being the one we set up earlier:

```
http://[IP]:8086
```



07 Change your password

The default username and password is admin for both fields, and you should make sure to change that before doing anything else. Go to Configuration along the top bar and find the Authentication field at the bottom of the page. Input the original password (admin), followed by your new passwords. The username will remain as admin.



08 First command

Go to Commands on the top bar to begin creating commands to run. Here you'll need to add a class – a user-defined way to filter your commands that won't affect the way it's run – a name for the command and the actual command itself. The commands won't necessarily run from the pi user unless you tweak the config files.

09 More functions

The web interface has a few extra functions apart from running commands, such as the ability to view the webcam and connect to radio services. Updating the software every so often will also allow you to make sure it keeps working. Play around with it and see what best suits you.

Build your own streaming stick

No Chromecast? One solution is to use a mobile app and Raspberry Pi to cast media across your network. **Christian Cawley** shows how it's done.

OUR EXPERT

Christian Cawley has spent the past 15 years telling people how to do the same computing tasks but still has no idea how cars work.

Google Chromecast is a great way to get media from your phone to your TV. Unfortunately, it's not ideal in some cases. For example, certain routers such as the Virgin SuperHub 3 won't work out of the box with a first-generation Chromecast. Meanwhile, the Chromecast is low spec and limited in many ways.

Fortunately, the Raspberry Pi is ready to come to the rescue, providing a Chromecast-like experience when paired with a dedicated mobile app. With a just few terminal commands, you can configure the Raspberry Pi to accept video, music, and audio content streamed from your Android phone or tablet, and dress it up with a slick home screen.

While this should work with any of the main Raspberry Pi models, we used a Raspberry Pi 3 B+, which delivered fast, stable streaming of high-definition video from the Android handset.

Streaming video, not apps

It's important to emphasise that the Raspberry Pi plus *Raspicast* setup isn't a complete replacement for the Chromecast. For example, there's no VR support, or Google Home; there's no ambient mode, although it's easy enough to configure a similar home screen, which we'll also look at below.

In short, media streaming is replicated. Everything else you'll need to add yourself, rely on existing streaming options, or quietly overlook.

Enable SSH

Start by ensuring the Raspberry Pi is connected to a display, and you have a mouse and keyboard attached. If this isn't possible, you'll need to set this up remotely, controlling your Raspberry Pi over SSH.

To do this, you'll need to activate SSH on the Raspberry Pi. If this isn't already enabled, you have two options. The first is to connect a keyboard, open the Raspberry Pi Configuration tool in the Preferences submenu on the desktop, click Interfaces and select SSH, clicking OK to confirm. A reboot may be required.

The second option is to insert your powered-down Raspberry Pi's microSD card into your computer, browse to the **/boot/** directory, and create an empty text file called **ssh**, with no file extension. Eject the card, replace in the Raspberry Pi and reboot; SSH will be enabled, ready for remote access. Note that you'll need to know your Raspberry Pi's hostname or IP address for SSH access. You'll find the hostname by entering that word as a command. To see the IP address, either check your router's admin screen or enter `ifconfig`. Look for an entry for eth0 (Ethernet) or wifi0 (wireless), and you'll find the IP address labelled inet.

Fruit upgrade

It's time to update your Raspberry Pi. We're assuming you're using the latest version of Raspbian Stretch for this, although other Pi distros should work fine. Open the Terminal (or connect via SSH) and enter

```
sudo apt-get update
sudo apt-get upgrade
```

With your Pi's repositories refreshed and any upgrades downloaded, it's time to install some prerequisite packages:

```
sudo apt-get install libjpeg9-dev libpng12-dev
```

Two graphics-based packages are installed here: **libjpeg9-dev** and **libpng12-dev**. These enable JPEG and PNG images to be streamed from your Android device to your Raspberry Pi, and displayed on your TV.

Streaming to the (Open)Max

Also required is *OpenMax*, described by its developer as "a GPU accelerated image viewer for the Raspberry Pi." It also includes graphic libraries for GIF and BMP images. Get *OpenMax* by cloning the GitHub directory:

```
git clone https://github.com/HaarigerHarald/omxiv
```

Once downloaded, change directory:

```
cd omxiv
```

Next, use the make command to build the software.

```
make ilclient
make
```

QUICK TIP

Is your streamed video too quiet? You can boost the volume from the Advanced options screen. Look for Audio volume offset and increase the value incrementally until it's loud enough.

Combine a simple slideshow plus weather information tool and media casting to turn your Pi into a Chromecast-like device.

A Chromecast device is nice enough, but it's more satisfying to configure your Raspberry Pi to do the same job.

You'll need to wait a while for this to complete. This is a good time to install the *Raspicast* mobile app, as per the boxout (*below right*).

Once the tool is ready to install use

```
sudo make install
```

Wait while this completes before proceeding.

With everything ready to roll on your computer, and the *Raspicast* app installed on your phone it's almost time to start casting. First, open the app and tap the menu to find the SSH settings. Here, input the hostname or IP address of your Raspberry Pi. You'll also need to add the username and password. Click OK and prepare to start streaming media from your phone.

Two options are available: streaming content stored on your phone from within the app, or by sharing the content from another app. To stream from *Raspicast*, tap the Cast button, and browse the media on your phone. Select the one you want to view, tap the Play button, and your Raspberry Pi will display the content on your TV.

For sharing (perhaps from YouTube), simply cue a video or song, tap the share button, and select *Raspicast*. You'll spot two options: one to play immediate, and one to queue the clip to play later.

As versatile as ever, the Raspberry Pi makes a great Chromecast replacement. *Raspicast* isn't the only method to achieve this, however. If you have Kodi installed on your Raspberry Pi, several remote apps for the media centre, including the official option *Kore*, offer casting from your mobile. This is useful, because *Raspicast* won't work on Raspberry Pi-based Kodi systems anyway.

Take me home

You might have a Raspberry Pi that receives media, Chromecast-style, from your Android device, but it doesn't quite look the part. Chromecast has a slideshow of curated images for you to enjoy, for example, along with weather, and date and time information.

Want to make your Pi more like Chromecast? You need a kiosk app – one that enables custom images to be displayed as a slideshow. Many of these are available, but the simplest to set up is Andrew Delph's *Python slideshow with time and weather* which is on GitHub.

Get started by adjusting the Raspberry Pi's video memory. Open the configuration tool with

```
sudo raspi-config
```

Next, browse to Advanced Options>Memory Split. Here, change the value to 256. This ensures that the Pi's GPU has enough available memory to handle the slideshow images.

Click OK to continue, then follow the instruction to reboot. Upon restarting, open a new terminal window and clone the slideshow files from GitHub:

```
git clone https://github.com/andrewdelph/python-slideshow-with-time-and-weather.git
```

Once this is downloaded, rename the directory to a snappier title, such as **pyss** with the `mv` command.

```
sudo mv python-slideshow-with-time-and-weather pyss
```

You'll also need the *pyowm* tool. This is a wrapper for the OpenWeatherMap, which brings the necessary weather data onto your Raspberry Pi's desktop:

```
pip install pyowm
```

Some additional configuration is required to get this to work. First, you'll need to remove some images. Open the **pyss** directory first and remove the four JPG files starting **WIN_20160220**.

Next, edit the **ss.py** file.

```
sudo nano ss.py
```

Here, look for the line that reads

```
observation = owm.weather_at_place('Ottawa,CA')
```

Change the city and country to match your own location. For example, a Londoner might use

```
observation = owm.weather_at_place('London,UK')
```

Press Ctrl+X to save and exit. In the terminal launch the slideshow:

```
python ss.py
```

A gallery of stunning images, the date and time, and current weather will be displayed on your Raspberry Pi Chromecast replacement!

» INSTALLING RASPICAST

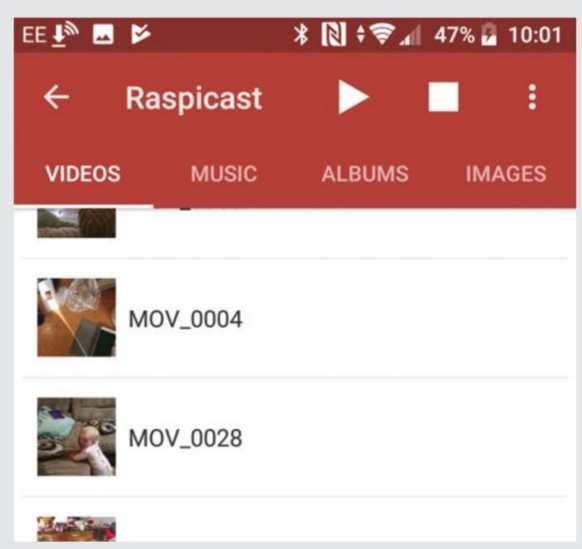
To cast to your Raspberry Pi from your Android phone or tablet (a reliable iPhone or iPad alternative has yet to be released), you'll need *Raspicast*, from the Google Play store. Once installed (with all permissions granted), ensure your phone and the Raspberry Pi are connected to the same network. You can't use *Raspicast* to stream media from your phone to a Raspberry Pi otherwise.

Within the app you'll find standard controls (play/pause, and stop) a file browser, Cast button, and some options that will change how *Raspicast* streams your content.

For example, should you want to replay the current file, open the menu and check the Repeat box. Meanwhile, tap Audio output to ensure that the audio is streamed in the correct way. For video, this will typically be via the HDMI option, but you might prefer the sound to play on your Android device (local) or even through both. You can also use the ALSA option to pipe the file's audio through the Pi's TRRS A/V port.

Several advanced tools can also be found. Use the Advanced options screen to configure these, which include the ability to change the video quality of a YouTube video (switch between HD, SD and even audio only).

You can also use this screen to change the *Raspicast* app's theme, set a playlist directory, configure the appearance of subtitles, and even tweak the delay on slideshow images.



Use Raspicast to browse for videos and then stream them to your telly.

Stream live to YouTube with your Pi camera

Christian Cawley explores how to create a YouTube livestream from your living room, with a Raspberry Pi and Camera Module.

Christian Cawley has more Pis than you've had hot dinners, probably.

Streaming videos on the web is increasingly popular, and YouTube is the chosen platform for many. Almost any device can stream video to YouTube, but what if you want a dedicated streaming solution? The Raspberry Pi is a great choice for this, giving you the option to stream anything you like, just as long as there is an unlimited network connection.

You're limited only by your imagination, with the Pi and its camera module ideal for everything from a YouTube-based baby monitor to running live presentations (*see the box over there for more ideas*).

With a good network connection, the right Raspberry Pi set up, and a YouTube channel stream URL, you can have your Raspberry Pi YouTube streaming video camera up and running in just a few minutes.

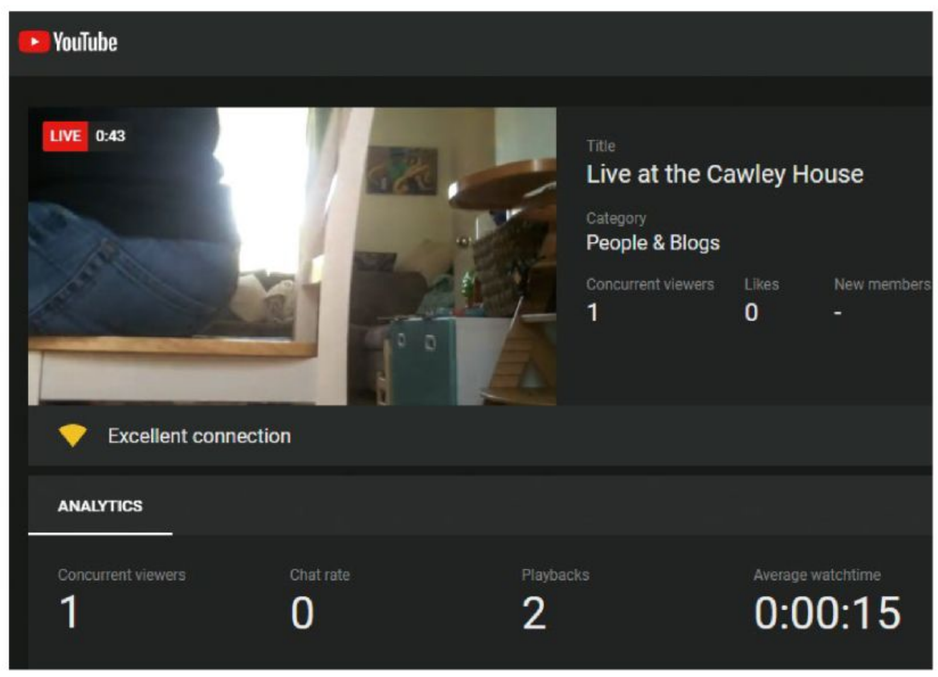
Streaming essentials

Before you even pick up your Raspberry Pi, it's a good idea to ensure your network is up to the task. The optimum bitrate for YouTube videos from the Pi is 400-600kbps. You can change this if necessary, but your router will need to be up to the job. Help things along by moving the Pi closer to the router, using an Ethernet cable if necessary, or employing powerline adaptors to improve signal strength at your Raspberry Pi's location.

While you don't necessarily need a wireless Pi, the best results can be enjoyed with a Raspberry Pi 3 or 3B+. The improved spec of these two over the Raspberry Pi 2 results in superior streaming performance. The Pi 2 will work, but you may run into resource bottlenecks. Tweaking that bitrate (you'll find out how later) will help if you're stuck using one.

Your choice of camera may impact how well this works. USB cameras, while compatible with the Pi, are largely pointless as the computer has such a good official camera. We used the Raspberry Pi Camera Module v2.1 for this project, but the older models should work just as well.

If you're planning to add audio to your camera stream, you'll also need a microphone, of course. The best option here is to use a Raspberry Pi-compatible USB mic, although anything connected to a suitable HAT will work. Don't use the TRRS port, however, as it won't work. Raspberry Pi audio is beyond the scope of this tutorial, so we're continuing with a silent video.



Make sure you have a clear picture and discernible audience for your YouTube stream. 'Man sitting on dining chair' is perhaps a niche too far.

We have a stream

The Raspberry Pi Camera Module must be connected before you power up the computer. If you haven't done this before, find the camera port and lift the catch. Insertion is straightforward: the rule of thumb to remember is that the silver side of the contacts point towards the HDMI port. Before booting up, it's also a good idea to connect the microphone for configuration later on.

With the Raspberry Pi up and running you'll need to configure the camera module. You can do this in the Raspberry Pi Configuration screen in the desktop environment, or via the *raspi-config* tool in the command line.

```
sudo raspi-config
```

Whichever method you choose, you need to display the Interfacing Options and set the Camera to Enabled. Select OK to confirm and reboot the Raspberry Pi when prompted. Next, you'll need to confirm that the camera is working. Use the *raspistill* command in the terminal:

```
raspistill –o image.jpg
```

This creates a file called **image.jpg** in the **/home** directory. You can check it has been created by listing the directory contents:

```
ls
```

In most cases, the camera image will be fine. To check it properly, you'll need to plug your Pi into a HDMI display, or check via VNC.

For streaming to work, you need a YouTube account and a unique key that is created for each stream. If you have a Google account you probably already have a YouTube account, so sign into YouTube and find the camera icon, click it, and select Go live.

In the New stream window, give the feed a title, set a Public/Unlisted/Private option, a description and genre, then click Create Stream. The Stream Preview window appears, displaying the standard Stream URL and the Stream name/key. You'll need this, so click the visibility icon and make a note of the key for later.

Streaming specifics

To create a video stream you should have the *avconv* software installed. This comes pre-installed in Raspbian Stretch, so if you're using an older version it's wise to update first.

```
sudo apt upgrade
sudo apt update
```

If you don't want to do this (perhaps because you've got deprecated software that you don't want to stop using), simply install *avconv* as part of the libav-tools package instead:

```
sudo apt install libav-tools
```

Creating the stream requires the input of a long `raspivid` command, consisting of various switches and instructions such as `fps` to set the frames per second. As it's quite a complicated command, it's worth looking at some of its key elements.

`-fps` As noted, this sets the frames per second captured by the camera. Anything over 24fps should be fine; less than this and it may resemble time-lapse. A lower rate can improve streaming, however.

`-w <xxx> -h <xxx>` Specifies the width and height of the video. Without these, *raspivid* defaults to 1920x1080 HD resolution.

`-b` The output bitrate limit, which YouTube recommends should be 400-600Kbps. A lower figure means a lower-quality video, but better streaming.

`-acodec` If you're not using a mic you'll need to include this. YouTube doesn't permit videos without audio (or audio without video), so this command creates a fake audio track. If you have a mic connected and working, you can omit this.

`-f` The output format. For streaming video from YouTube to the Raspberry Pi, FLV is the best option.

Keep on streamin'

If you're setting this up over SSH it's a good idea to take steps to avoid the stream closing when you disconnect. One way of avoiding this is with *screen*.

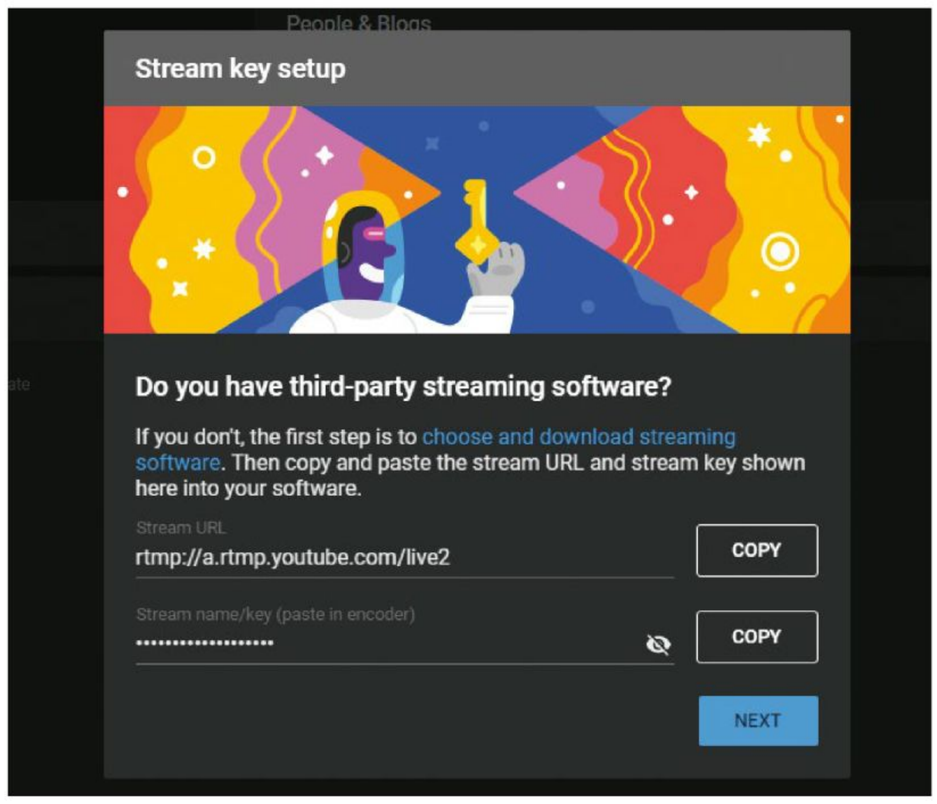
```
sudo apt install screen
```

Wait while it installs, then reboot the Pi.

```
sudo reboot
```

Connect again over SSH, then enter the `screen` command. You can then run the `raspivid` command. Note that this should be entered as a single line, like so:

```
raspivid -o - -t 0 -vf -hf -fps 30 -b 6000000 | avconv -re
-ar 44100 -ac 2 -acodec pcm_s16le -f s16le -ac 2 -i /dev/
zero -f h264 -i - -vcodec copy -acodec aac -ab 128k -g 50
-strict experimental -f flv rtmp://a.rtmp.youtube.com/
live2/[your-secret-key-here]
```



Copy the key for your RMTP stream into the raspivid command to stream footage from your Raspberry Pi to YouTube.

Simply copy the YouTube stream key from the browser window into the terminal, replacing `[your-secret-key-here]`. Press Return and wait while the stream starts. A moment later, you'll see it running in the top-left corner of the YouTube livestreaming page. If nothing happens, double-check all the setup steps as listed above.

Crossing the streams

With the stream running, you can share it or simply watch it. Using a private channel for the feed at the setup stage lets you limit the feed to YouTube apps or browser sessions with the corresponding account details. Otherwise, the feed will be public or unlisted.

With both options, you can share the feed with others. The public feed is open to anyone, whereas the unlisted feed is limited to those who have the feed URL; it won't be displayed in search results. Both options let you share the feed URL on your preferred social networks as well.

QUICK TIP

Want to livestream your life to YouTube via your Raspberry Pi? Try a high-capacity rechargeable portable battery, with enough charge to last a day. Beware: they can get heavy!

» USES FOR LIVESTREAMING

Why might you want to stream video with a Raspberry Pi? Various uses can be considered, although some will be impacted by latency, which can be as much as 30 seconds once the video has been captured, sent and uploaded to YouTube.

For example, you could set up the Raspberry Pi as a baby monitor. Configuring the YouTube feed as private means that you can keep an eye on your little ones from any portable device, just as long as you're signed in with the same account.

Alternatively, you might use the Raspberry Pi as a security camera. Again, a private channel can let you watch your home from any location you can get a mobile signal. While it will miss the benefits of a standard IPTV security camera (such as motion detection), it makes an adequate substitute.

Other problems can be solved with a Raspberry Pi streaming video camera. You could go public with your YouTube video feed, perhaps making short livestreams to share your thoughts about recent topical events. You could demonstrate your job, your work setup, or even a hobby you enjoy; you might even conduct an AMA with your Raspberry Pi as the stream host, direct to YouTube.

» STREAM US TO YOUR DOOR Subscribe now at **http://bit.ly/LinuxFormat**

Watch videos easily

Les Pounder loves watching cartoons, and now he can choose a random video on demand! Now all he needs is a giant bucket of popcorn!

OUR EXPERT

Les Pounder is a freelance maker who works with organisations such as the Raspberry Pi Foundation to promote maker skills. Read his blog at **bigl.es**

Stuck on what to watch this evening with the family? Got some time to kill and not sure what to put on? Fear not – with this project we can have our Raspberry Pi randomly choose a video for us to watch. So let's start building and coding our own random video player! All of the code, and a detailed high-res circuit diagram can be downloaded from **https://github.com/lesp/LXF244-VideoPlayer/archive/master.zip**.

Don't be breadboard

Building the circuit for this project is simple. We only require three buttons, connected to the GPIO at pins two, three and four, and each of these buttons also needs to be connected to GND. For this we can use a single GND pin on the Pi, and a female-to-male jumper wire. Connected to the – rail of a breadboard means that we have multiple GND connections that can be connected via the two male-to-male jumper wires. See the diagram in the download for this project for details.

When the hardware is built, attach the accessories required for your Pi and boot to the Raspbian desktop.

Tune in

Before we can write any Python code, we need to install two libraries that will help us create the project. Open a Terminal, and type the following:

```
$ sudo pip3 install glob
$ sudo pip3 install keyboard
```

We'll talk more about them later. For now open the Python 3 editor, found in the Programming menu and click File>New to create a new blank file. Immediately click File>Save in the new window. Save the code as **VideoPlayer.py** and remember to save often!

So now we start writing the code, and our first act is to import the libraries that we require. The first three libraries are used to detect the button presses (GPIO Zero), stop the code from running once and exiting (pause) and to choose a random video (choice.)

```
from gpiozero import Button
from signal import pause
from random import choice
```

The final three imports are a library that we shall use to list the contents of a directory (glob), run terminal commands (subprocess) and the last is a library to emulate a keyboard (keyboard.)

```
import glob
import subprocess
import keyboard
```

Moving on we create three functions–blocks of code that we can later reuse by calling their name. The first function is called `play_video` and it first creates a list (a data storage object) called `videos`.

```
def play_video():
  videos = []
```

To fill the list with all the videos we can play, we use a `for` loop. This will iterate over every file in a directory, as long as it's an mp4 video file. For this we use `glob` to access the directory **/media/pi/Videos**, which is really a USB stick called Videos full of mp4 files. Change this to match the name of your chosen directory full of videos. Each time an mp4 is found, it's appended to the `videos` list that we've just created.

```
or file in glob.glob("/media/pi/Videos/*.mp4"):
  videos.append(file)
```

Check that the list has been populated with file names by printing the list's contents to the Python shell.

```
  print(videos)
```

Choose a random video from the list and store it in a variable called `chosen`. Print this to the Python shell.

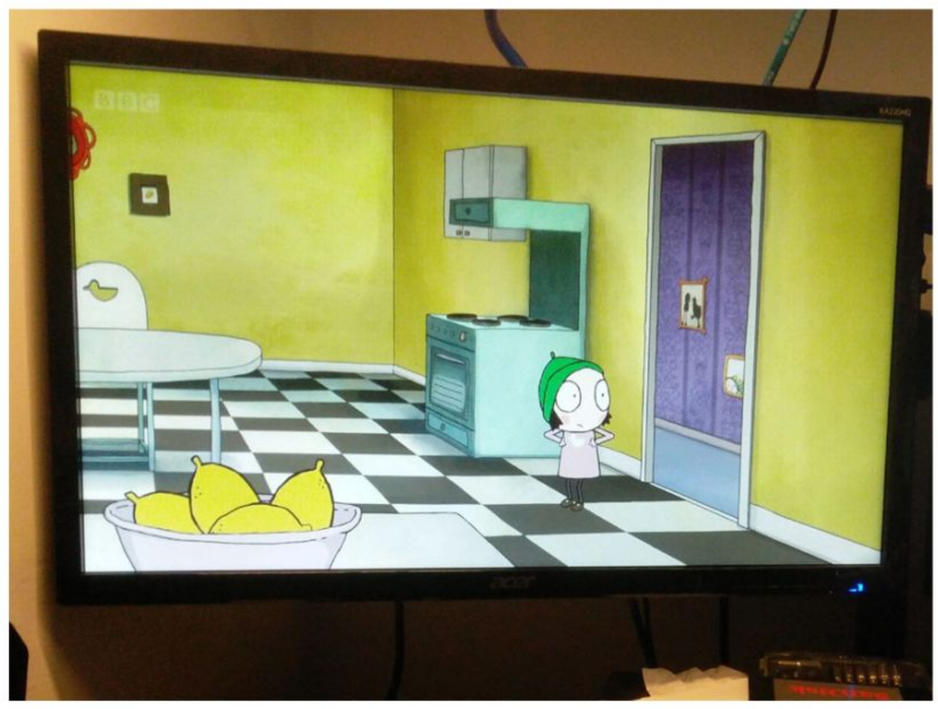
```
  chosen = choice(videos)
  print(chosen)
```

The last line in this function uses the `Popen` class from the subprocess library to run a command as if we were sat at the terminal, and this command is to open the *omxplayer* media player, then play the chosen video.

```
  subprocess.Popen(['omxplayer',(chosen)])
```

The next function is called `stop_video` and as you can guess it will stop the currently playing video. For this we use the keyboard library, specifically the `press_and_release` function to simulate pressing Q.

```
def stop_video():
  keyboard.press_and_release('q')
```



Playing video on our Raspberry Pi is really easy using this script and the omxplayer command, which smoothly runs 1080p video.

YOU NEED

- Any model of Raspberry Pi
- The latest version of Raspbian
- Three push buttons
- Breadboard
- Four female-to-male jumper wires
- Two male-to-male jumper wires

The last function is called `pause_video` and it emulates pressing the space bar on the keyboard, which is how *omxplayer* pauses video.

```
def pause_video():
  keyboard.press_and_release('space')
```

With the functions created, we next need to tell our code where our buttons are connected. We have three buttons connected: randomiser (play), stop and the pause_button. These buttons are connected to the GPIO at pins two, three and four, respectively.

```
randomiser = Button(2)
stop = Button(3)
pause_button = Button(4)
```

Push the button

On to the last part, which looks for a video when a button is pressed, and reacts accordingly. But first we wrap this section in an exception handler. It'll try to run the code, but if the user presses Ctrl+C then the code will exit. So for the `try` section our code will first print three lines to the shell – these are the instructions to the user. The `\n` between each colour is Python shorthand to insert a new line between each instruction:

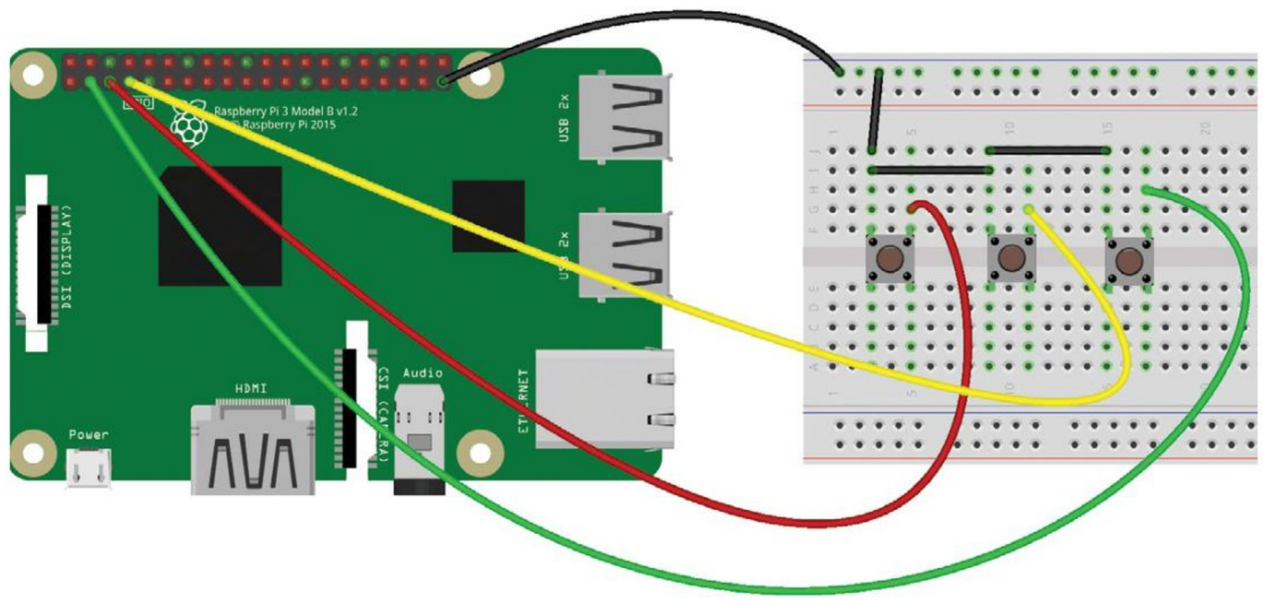
```
try:
  print("Press the GREEN button to start\nYELLOW to
pause\nRED to stop")
```

Our three buttons are waiting to be pressed, and using the GPIO Zero library we call the `when_pressed` class to detect when each of the buttons are pressed. When this happens the corresponding function is run. So when we press the Green randomiser (play) button it'll randomly choose a video from the USB stick and play it. You'll notice that the functions don't have a `()` at the end of the function name. This is because if they did, then the code would run automatically. Right now they're ready to run on demand.

```
  randomiser.when_pressed = play_video
  stop.when_pressed = stop_video
  pause_button.when_pressed = pause_video
```

```
File  Edit  Format  Run  Options  Window  Help
from gpiozero import Button
from signal import pause
from random import choice
import glob
import subprocess
import keyboard

def play_video():
    videos = []
    for file in glob.glob("/media/pi/Videos/*.mp4"):
        videos.append(file)
    print(videos)
    chosen = choice(videos)
    print(chosen)
    subprocess.Popen(['omxplayer',(chosen)])

def stop_video():
    keyboard.press_and_release('q')

def pause_video():
    keyboard.press_and_release('space')

randomiser = Button(2)
stop = Button(3)
pause_button = Button(4)
try:
    print("Press the GREEN button to start\nYELLOW to pause\nRED to
    randomiser.when_pressed = play_video
    stop.when_pressed = stop_video
    pause_button.when_pressed = pause_video
    pause()
except KeyboardInterrupt:
    print("\nEXIT")
```

There are only 34 lines of code in this project, and yet we've created a simple media player that can handle 1080p video playback.



Building the project is simple, take it one step at a time, and check your connections.

The last line in this section is a simple `pause()`, and this is used to keep the code running, and not just exiting after running once.

```
  pause()
```

But what happens if the user presses Ctrl+C? Well this is a `Keyboard Interrupt` and if that occurs we shall simply print an empty line and then Exit on the screen.

```
except KeyboardInterrupt:
  print("\nEXIT")
```

So now save your code, and open a Terminal to the directory where you have saved the code. In order to use the code we need to run it with `sudo` (root powers) because the keyboard library used in the project can only be used as root.

To run the code type:

```
$ sudo python3 VideoPlayer.py
```

When you're all sat on the sofa, press the Green button to play the randomly chosen video, to pause press Yellow, and to stop press Red. Enjoy your evening and be careful not to spill the popcorn!

QUICK TIP

You can embed the project in a tough box, so that little hands don't break the electronics. You can also use arcade style buttons, which are far more robust and less likely to be broken.

» KEYBOARD SHENANIGANS

In this project we used the Keyboard Python library to emulate a keyboard using just the push buttons that controlled playing, pausing and stopping video playback. The Keyboard library can be used to press any keys, so we can automate multiple key presses all from one push button. The library can also be used to write text to the screen.

But the most dangerous function of this library is that it can be used to record every key press on the target machine. These key presses can be recorded to a list, then saved to a file and then used for nefarious purposes. Of course, recording the key presses of a user, without their consent is illegal and can land you in a lot of trouble. So don't do it! With that said, it can be used as a powerful tool when debugging how a user interacts with your code, so for software testing (under consent with the general public and in-house testers) you can see what keys they were pressing right before the code locked up and/or went thermonuclear!

The Keyboard library works with Linux and Windows and it's certainly lots of fun to play with. To read a little more about this library head over to this blog post **https://bigl.es/tuesday-tooling-record-replay-keystrokes-with-python** and see a few examples of how it can be used in day-to-day life.

Build a Raspberry Pi stop-motion studio

Christian Cawley plays with modelling clay and a Raspberry Pi camera to make a movie – although it's not quite that simple...

OUR EXPERT

Christian Cawley
A Raspberry Pi owner since 2012, he's nurtured 13 devices to date, seeing them blossom into media centres, retro gaming systems, and portable radio broadcast stations.

YOU NEED

- **Raspberry Pi 2 or later**
- **Raspbian Stretch**
- **Pi Camera Module**
- **Dual-state push-button**
- **Breadboard**
- **Monitor**
- **Mouse and keyboard**
- **Tripod or secure mount**
- **Something to animate: clay figures, Lego, action figures and so on**

Ever sat guffawing at Terry Gilliam's bizarre subversion of Victorian-era greetings cards in *Monty Python's Flying Circus*, or egging on the unlikely heroes Wallace and Gromit? Then chances are you've spent a few moments thinking about how those animations are made.

There was a time when stop-motion photography was beyond the reach of most people. Overhead cameras mounted on rostrums were expensive, and unless your school or college had one, your best bet was to take some photos and make a simple flick-book instead. But, as with everything else, time and technology has completely revolutionised the production of stop-motion photography, and now making a film with a digital camera or mobile device is child's play.

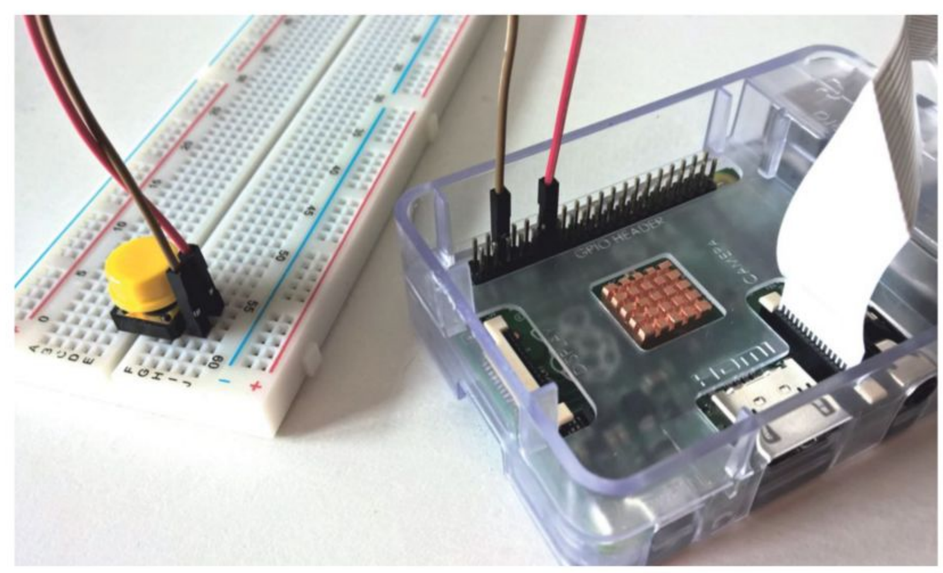
The Raspberry Pi Camera Module offers a superb low-cost digital option. While time-lapse photography is popular, it's easy to overlook its illusory bedfellow. After all, creating a stop-motion animation with figures, household objects or simply toys is intensive work.

Often the results aren't very good, and even getting to the point where you have enough shots to create a short movie clip can be pretty back-breaking. Fortunately, there are some shortcuts you can take, which we'll explain here.

Stop, it's motion time...

It's a good idea to allocate time to your stop-motion project – say a few days. Shoots are famously slow; you're going to be snapping around 24 frames for each second of footage, so having an area that remains undisturbed when you're not using it is vital. This 'hot set' should remain untouched by others, too.

Planning to animate Lego minifigs? Use base pieces or putty to keep the figures stable for each shot.



Be careful connecting the jumper wires to the GPIO – ensure the Raspberry Pi is powered off before hooking up the button.

Your studio area should have enough space for the tripod- or rostrum-mounted Raspberry Pi and Camera module, along with the objects you're filming. In the case of 2D, paper-based animation, this should be an area of table big enough to fill, or almost fill, the lens. Also useful for overhead shooting is a sheet of clear plastic or glass. This keeps the papercraft pieces in place when taking a shot, as well as helping them to look flat and uniform and dealing with any curled edges. If it was good enough for Terry Gilliam, it's good enough for you!

In order to shoot clay models, fruit, toys or whatever 3D objects you plan to animate, you'll need an area large enough to build a set. There should also be space around the table for you to move in to aid with the animation of your characters. Remember that stop-motion animation requires very slight movements in every frame for each of the characters or objects you're shooting. The aim is to get the character from one position to another, while creating the illusion of motion. Access to the character from every angle will help you to achieve this.

Whether you're using an overhead camera or a standard tripod-mounted shoot, the project needs to be well-lit. See 'Lighting balance' box *(over the page)* for some handy lighting tips.

Lights!

To turn your Raspberry Pi into the central component of your stop-motion studio, you'll need to attach a camera. While USB cameras work fine, it's far more satisfying to employ the latest Raspberry Pi Camera Module.

Connecting this device to the Raspberry Pi requires you to insert the ribbon cable into the 'Camera' connector on the SBC. If you're using a Raspberry Pi model B, the shiny side of the ribbon should face away from the Ethernet port.

For Raspberry Pi Zero devices with a camera port, the shiny side of the ribbon should face up. Take care to correctly unclip the catch on these connectors before inserting the ribbon. Similarly, ensure the catch is correctly secured afterwards.

Boot up the Pi and enter the configuration screen on the desktop (Menu > Preferences > Raspberry Pi Configuration) or open a terminal and enter `sudo raspi-config`. Here, select Interfaces (or Interfacing Options in the terminal), then enable Camera. Click OK to confirm and restart your Pi.

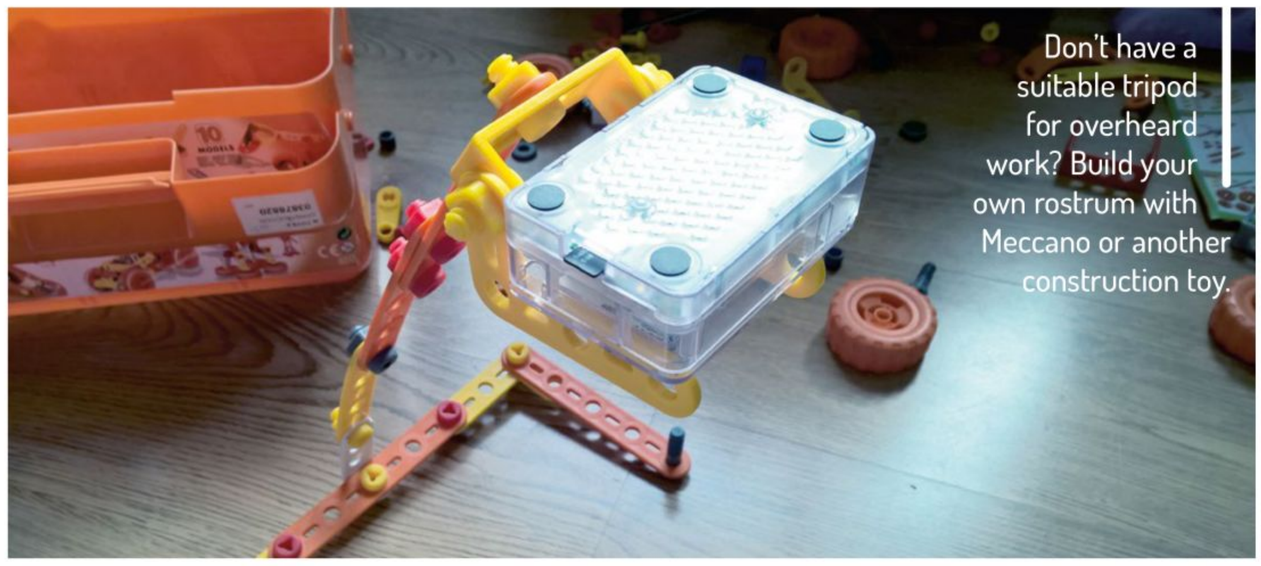
With the Pi restarted, open a terminal window, and enter `raspistill -o image1.jpg`. This should take a photo from the camera, confirming the device is correctly connected and working. You'll notice that *raspistill* is built into the Raspbian operating system. It's part of the *raspicam* software, which we'll come back to later.

Camera!

Setting up the camera on your Raspberry Pi will let you create a stop-motion video, but it can be a bit fiddly entering the same command for every single shot. The solution? Create a shutter button that you can connect to your Raspberry Pi!

Prototype it first using a breadboard, connecting the two wires to pins 6 and 11 on the GPIO header (or GND and GPIO17). With the button correctly mounted to connect to the wires, you can use this setup to capture each frame. Alternatively, connect a wired-up button directly to the pins on your Raspberry Pi's GPIO.

However you connect the shutter button, you'll then need to create a Python script to detect it. Each push of the button will capture and save a new photo to the specified directory. Open Menu > Programming > Python 3 (IDLE) then File > New. In the resulting window, input this script, from raspberrypi.org (**http://bit.ly/LXF249-stopmotion-script**):



Don't have a suitable tripod for overheard work? Build your own rostrum with Meccano or another construction toy.

```
from picamera import PiCamera
from time import sleep
from gpiozero import Button

button = Button(17)
camera = PiCamera()

camera.start_preview()
frame = 1
while True:
  try:
    button.wait_for_press()
    camera.capture('/home/pi/Pictures/stopmotion/
frame%03d.jpg' % frame)
    frame += 1
  except KeyboardInterrupt:
    camera.stop_preview()
    break
```

This script uses the *PiCamera*, *sleep* and *Button* programs, from the *picamera*, *time* and *gpiozero* libraries. It defines a button (pin 17) and camera and waits for a button press.

At this point, it adds a single image into the specified directory. The script ends only when a keyboard interrupt command is detected.

QUICK TIP

Finding something to animate in your stop-motion movie project might seem tough, but really you can use anything. It could be household objects, or it might be toys. Check toy shops for Plasticine character kits or Lego packs specifically designed for stop-motion projects.

For the best results, use a smartphone tripod to mount your Raspberry Pi for improved stability in your stop-motion movie.

» KEEP STEADY AND CARRY ON

One important element to keep in mind is stability: it's all too easy to knock a camera, especially one mounted on a tripod. An overhead camera can also be subject to vibrations and wobbles.

The result of these bumps can be a jarring distortion in your finished movie. While repeating the shot is one solution, it can be time-consuming to arrange. A better solution is to ensure that the camera is secured to a sturdy piece of gear.

A tripod should have a heavy, reliable base with a low centre of gravity to avoid topples. An overhead camera should be securely connected to the base, or whatever rigging is being used to support it. You need to minimise any vibrations that might occur during the capturing of your stop-motion frames. This can be difficult using the breadboard-based shutter button, so it's worth changing this to a direct GPIO pin connection with a long cable.

For this project we opted to use a Raspberry Pi case with a mounting point for the camera module. This meant we could use the Pi itself like a camera, positioning it using a smartphone tripod. You might opt to connect the Pi to a car-windscreen-mounted smartphone holder for similar vibration-free stop-motion video.

QUICK TIP

It's almost always more convenient to use a Raspberry Pi without the display. However, for a camera project like this it's wise to have a monitor connected. Using the preview feature will ensure the shot is lined up and in focus. What type of display you use is up to you, although the larger, the better.

Action!

With everything set up, it's time to make your movie. This could be considered as the difficult stage, depending on how much planning you've done (you *have* made a storyboard, haven't you?).

In truth, however, if the Raspberry Pi is securely positioned, with a reliable power supply, and the shutter button correctly connected, you shouldn't have any trouble. The worst that's going to happen is probably a bit of back- or neck-ache as you make those small adjustments to your characters.

Your recording of the movie should follow this simple procedure: set up scene > take shot > move character/object(s) > take shot > move character/object(s) > take shot. This should continue until your movie is complete, or you're ready to take a break.

As every image is saved into the same folder with an individual file name, stopping and then restarting the following day should not be a problem. Nothing is likely to be overwritten, for example.

When you're ready to stop recording the movie, press Ctrl+C on your Pi's keyboard. This will terminate the *stopmotion.py* program and close the preview. With the movie complete, it's time to compile those hundreds or thousands of individual images into a single movie file. Before you do that, however, it's a good opportunity to look for any images that didn't quite work out.

For example, if you took multiple attempts at a particular shot, you can delete the ones you don't want (or move them to another directory). It's also a good chance to get a look at the overall movie in a sort of 'film strip' view. This will reveal if the lighting is consistent, for example, or if your camera has drifted or been obviously knocked when shooting. In some cases, it might be possible to retake a shot if it is particularly poor. However, it may be easier to simply drop the image, and hope for the best.

When you're ready to compile the images, browse to the **/stopmotion** directory within the terminal (or press F4 from within the desktop file manager to open a terminal). Then run:

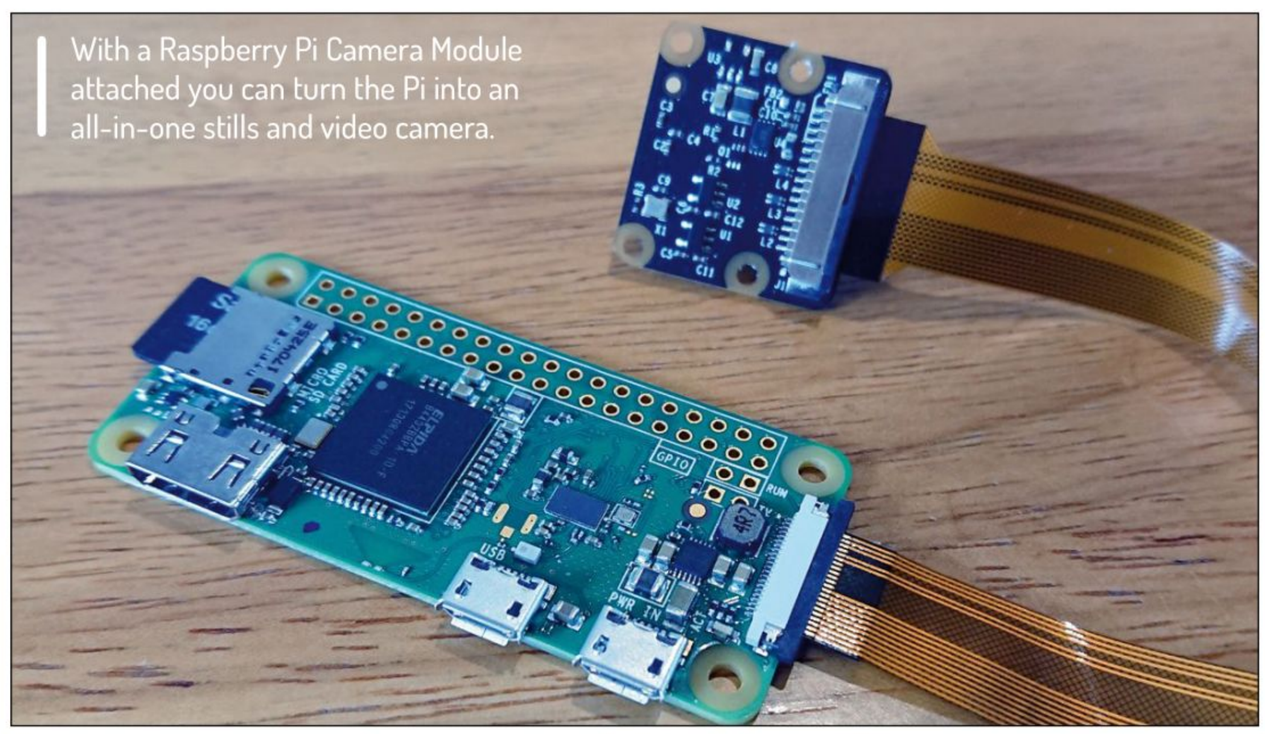
```
avconv -r 15 -i stopmotion/frame%03d.jpg -qscale 2
animation.h264
```

Wait while the process completes. Keep in mind the relatively low specification of your Raspberry Pi, and the fact that it is essentially creating a movie in high definition. This may take a while, so go and recover – maybe have a good shoulder rub.

The completed file **animation.h264** can be run in your preferred media player. For instance, to play in the Raspberry Pi's *omxplayer*, use

```
omxplayer animation.h264
```

Find the video is a bit jerky? Try altering the frame rate and recompiling it, represented in the `avconv` command with the `-r 15` value. The higher the figure, the smoother the animation. Note that the animation will also become shorter as you increase the frame rate.



With a Raspberry Pi Camera Module attached you can turn the Pi into an all-in-one stills and video camera.

Sharing with the world

With your movie completed, you may want to "um" and "ahh" over the problems. Or you might prefer to just get it out there to find out what people think of it.

Pretty much anywhere to which you can currently upload movies is suitable. Better still, as the movie is probably only a few seconds long, it won't take long to upload – so you can think about uploading the stop-motion movie to YouTube, Facebook, Twitter and so on.

Alternatively, you could upload it to your preferred cloud storage account. From here, it should be easy to access it from your mobile device and upload the file to Instagram, say.

Audio!

One shortcoming that you will have noticed with the creation of a stop-motion movie is the lack of sound. You have two main options to get around this.

The first is to load up a desktop video editing suite on your main PC (your Raspberry Pi probably isn't up to it!). Import the stop-motion video and add a soundtrack. This might be audio you already have saved to your PC, whether that's music or sound effects. Alternatively, you could record some custom sound effects or dialogue to accompany the action.

However, there is also the option to add audio when you upload the video to YouTube. The Creator Studio enables you to add sounds from YouTube's library of royalty free songs. The selection is pleasingly vast, so choose carefully!

» LIGHTING BALANCE

Shadows can ruin even the best stop-motion animations, yet this problem can be easily avoided with a bit of forward planning. Lighting is the secret, but you need to avoid relying on daylight or a standard ceiling light. This needn't cause you to break the bank, either; small LED lights perfect for this can be bought for just a few pounds.

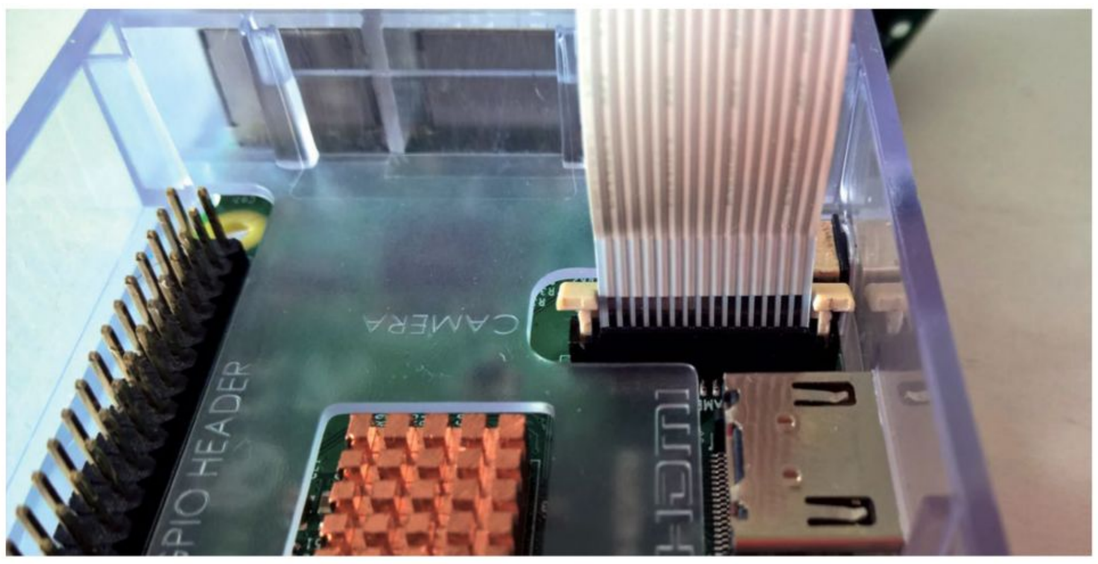
Positioning the lights to the left and right of the animation 'stage' enables you to cancel out any shadows cast by you or the camera. If you're animating with objects, toys or modelling clay, a third light from above will also help to cancel out shadows.

With the shadows banished, you should also ensure the white balance is set correctly. If you're using the recommended *PiCamera* software, auto white balance will probably work.

You can adjust this yourself by altering the `awb_mode` property, which by default is set to `awb_mode=auto`. Alternatives include `sunlight` and `cloudy`, though `tungsten`, `fluorescent` and `incandescent` are set for typical indoor lighting temperatures.

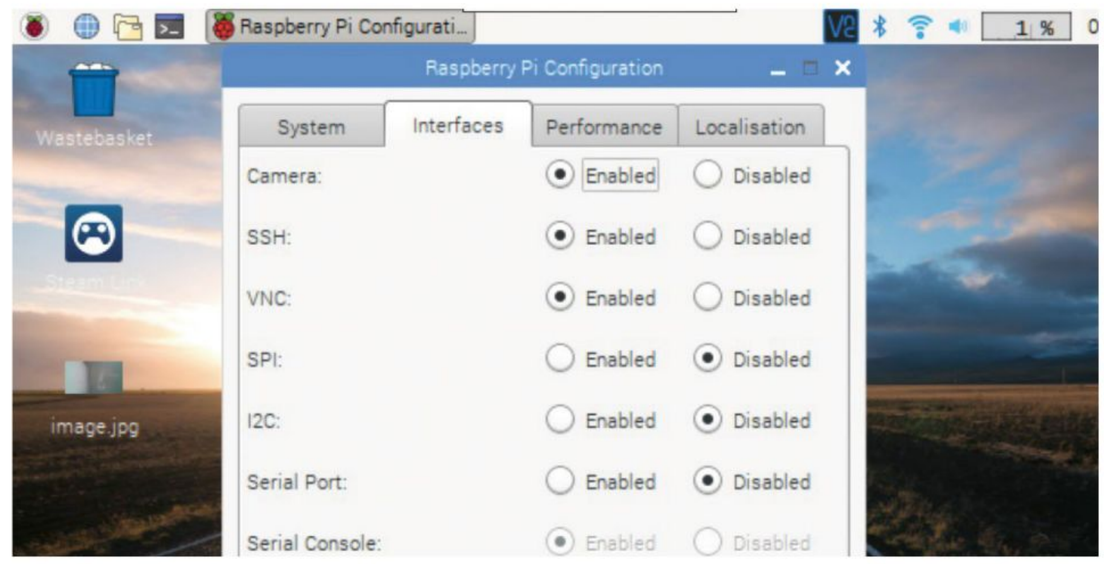
Meanwhile, you can even go as far as disable `awb_mode` with the `off` setting. You can then specify a precise `awb_gains` setting (between 0.9 and 1.9). Consistent lighting will result in a better stop-motion movie, so take the time to set this up accurately.

TURN YOUR RASPBERRY Pi INTO A STOP-MOTION MOVIE STUDIO



1 Connect the Camera Module

Hooking up the Camera Module requires the correct cable for your Raspeberry Pi computer. For Model A and B boards, the standard cable that ships with the Camera Module is adequate. However, for the Pi Zero, a narrower cable is needed. Slot the cable into the port with the shiny side facing away from the Ethernet port (or facing up on the Pi Zero).

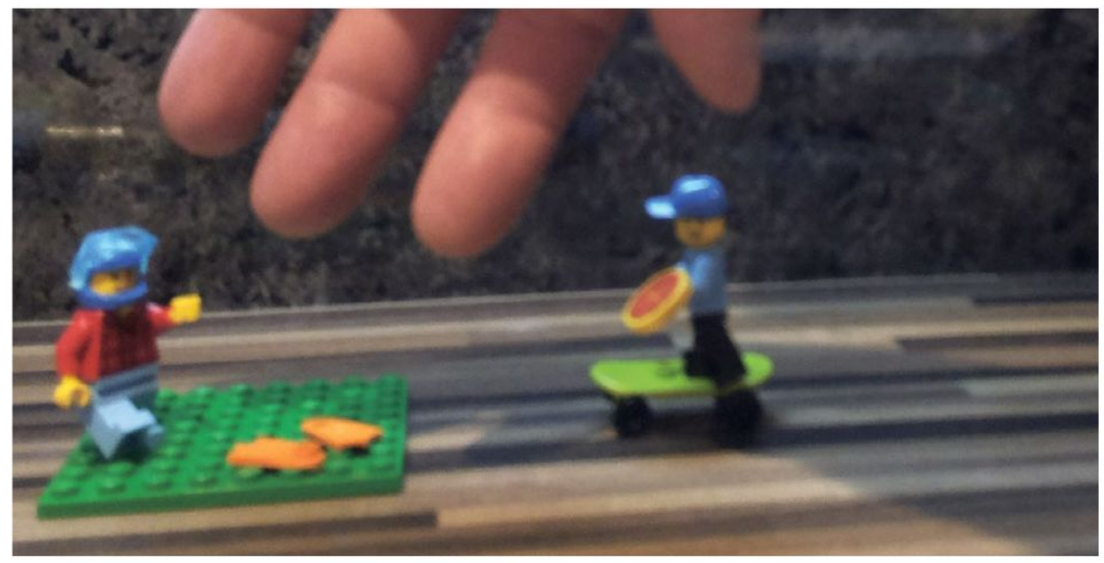


2 Enable the Camera Module

When you boot your Pi, you'll need to enable the camera. On the desktop, do this via Menu > Preferences > Raspberry Pi Configuration. Select Interfaces then select the Enabled button for the Camera, then click OK. Alternatively, open a terminal and use the *raspi-config* app. Select Interfacing Options, then Camera in the next screen. Choose Enable, then OK. Restart.

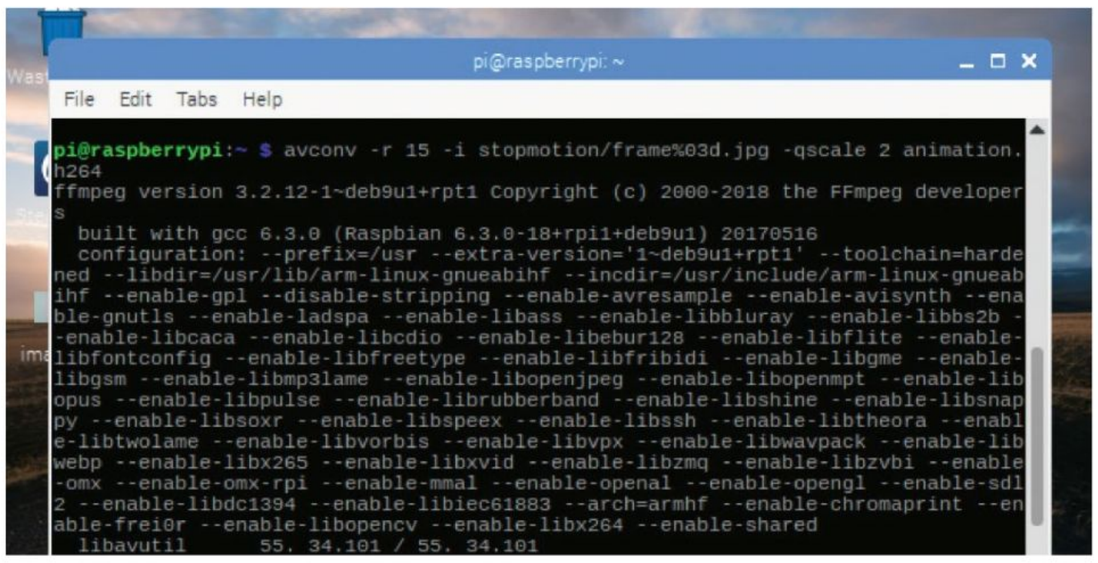
3 Connect a shutter button

The complicated part of this project isn't all that tough, really. If you want to use the *PiCam* script, you'll need a shutter – an on-off (dual-state) plunger button connected to the Raspberry Pi at pins 6 and 11. Once connected to the GPIO, and with the script running, the button will take a single shot from the camera on each press.



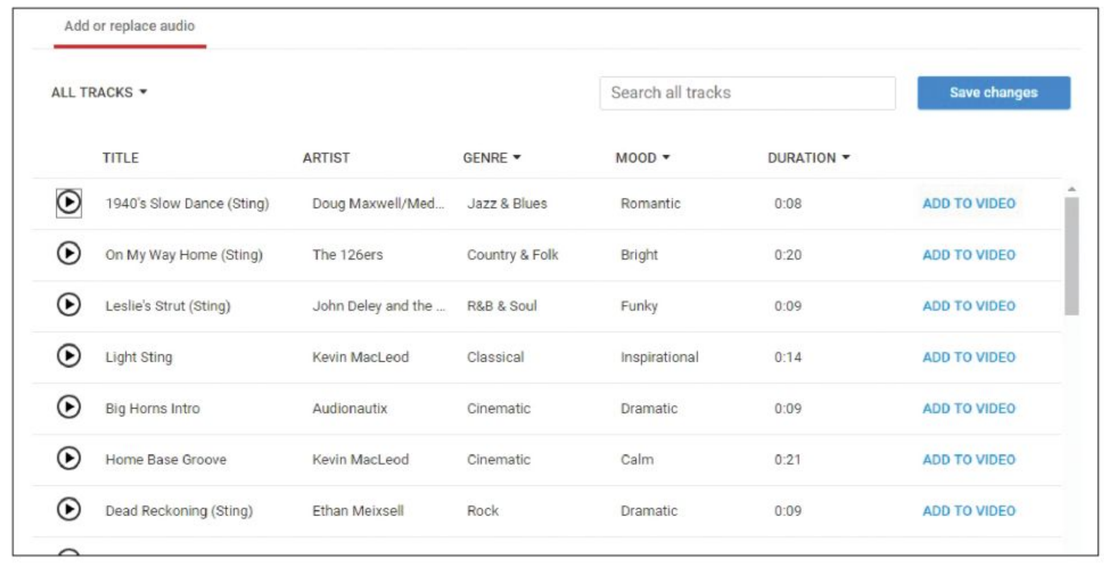
4 Start shooting frames

It's one thing to just plunge the button to make your movie, another thing entirely to get the shots right. With the camera securely placed and the characters and background in place, you'll commence the tiring job of snapping shot after shot to build the scene. Find safe shortcuts. Repeat shots wherever possible, and use as few characters, props and other elements as possible.



5 Create a movie with avconv

The completed movie will appear as a collection of stills in the destination folder. A flick-book effect can be enjoyed by scrolling through the photos in your file manager's preview panel. Using *avconv* to convert the photos into a movie is simple, but time-consuming for longer projects. Be sure to set the right FPS to get the best results first time.



6 Add audio with YouTube

Stop-motion animation is sound-free by default. While it might be fun to let the film play out in total silence, you have a few options to add sound. One is to load the finished video into a movie-editing app and add some custom audio with a mic. The other is to add music to the audio in YouTube when you upload it, using its Creator Studio.

Create a Raspberry Pi camera booth

Dan Frost shares a fun photographic project that will help newcomers get to grips with both coding and the Raspberry Pi.

OUR EXPERT

Dan Frost is Tech Lead in edtech R&D, working on VR, blockchain, voice tech, cloud, web and more. He blogs and talks about technology at **https://thebaseline.co**.

The Raspberry Pi camera module is a great little addition to get people into both coding and hardware. This project has been adapted from a coding club; the school had bought a few Raspberry Pis, a temperature sensor and a camera. The original plan was to record a time-lapse film of plants growing and plot the temperature against this. But the kids in the club were having none of that – they wanted something where they could point the camera at each other, so this is what became plan B.

The project recreates a simple photo booth which takes a photo, applies an effect to the picture and puts a frame around it. This was enough to learn about handling input from devices, storing files and using methods in a library.

This is a great little starter project for kids or adults because it does something tangible but doesn't require a lot of new concepts to be learnt. This tutorial doesn't assume much prior knowledge at all and tries to explain the terms as much as is practical as we go. If you get stuck at any point, you should have enough information to find meaningful help via Google. This might be useful for yourself or as the basis of lessons or coding club projects that you run.

QUICK TIP

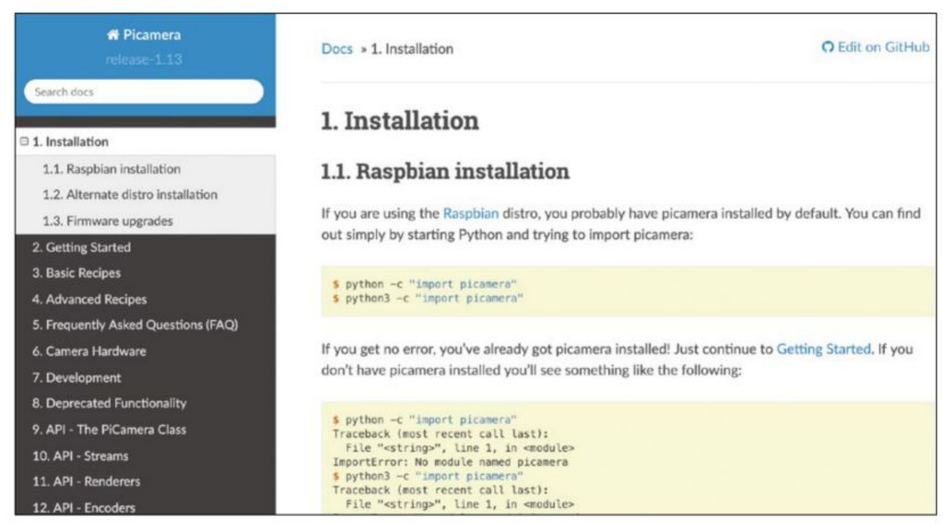
We had such fun turning a small idea into a program the Code Club students could share with their school. Those who didn't want to program worked on the frames or became 'UX people' (though they didn't know it). Be flexible with what you expect, but expect lots!

What you'll need

This has been tested on the Raspberry Pi v1 onwards, so you shouldn't have trouble doing this on any of version of the Pi. If you want to be sure, just search for the camera support for your particular model.

You'll also need a Raspberry Pi camera. If you haven't played with this module before, all you need to know is that's it's much easier to set up than you might expect. The additional hardware is usually pre-configured for the Pi, or the Pi will support it without having to do what we did in the 'old' days: spend days compiling and configuring drivers for each new piece of hardware. The tutorial uses Python, which comes pre-installed on the Pi.

The advantage of using the Pi Camera is that it's built to work with the Raspberry Pi, meaning that you don't have to worry about device drivers (the software which lets the hardware work together). The Pi Camera also comes with a Python module so it's easy to write code to interact with it. Before components such as the Pi Camera and computers like the Raspberry Pi were popular, it was much harder to get started using these kinds of devices, which is why they're a really fun thing with which to get people started in coding.



The PiCamera module docs are your way into understanding what can be done with the Raspberry Pi Camera.

Setting up the Camera

Most of the tutorial doesn't require access to the internet, but for a first step we're going to update the OS, which does of course require internet access. Run these commands:

```
sudo apt-get update
sudo apt-get upgrade
```

Now we need to enable support for the Camera. Run the following command which will launch a command-line configuration tool:

```
sudo raspi-config
```

Navigate down to Camera, select Enable and then Finish. Now reboot your Raspberry Pi:

```
sudo reboot
```

Let's test that the Camera is working. Plug in the camera by lifting up the small plastic handle on the DSI Display Connector. Plug it in, point it at something and open the command line.

```
cd ~/Desktop
raspistill -o image.jpg
```

Now open **image.jpg** from the desktop and you should see the picture you've just taken.

Love, life and running Python scripts

In this tutorial, we want to focus on playing with the Raspberry Pi camera, but to save you Googling too many steps let's give a quick introduction to editing and running Python scripts.

Python scripts are just text files which end in **.py** instead of **.txt**. You can edit them on the Raspberry Pi by opening any text editor and opening the file. Do this now with any text editor you're happy with (although don't use a fully fledged word processor, as that will probably add unnecessary formatting which will confuse Python).

When the file is created and saved, open up the terminal (which you can find through the menu) and navigate to where the file is stored. You can 'execute' (run) the file with:

```
python3 name-of-file.py
```

Now put this line into the file and execute it:

```
print("Hiya, world!")
```

Obviously, change **name-of-file.py** to whatever you named the file. If you're going to use this tutorial with newcomers to computing, I suggest you give a few challenges such as: Change the text in the file. Create another file with a different message and execute it. Google how to create a loop in Python, and put that into a new file.

Having basic control over the file and ability to execute it will, I've found, give most newcomers the independence to solve a lot of basic problems which would otherwise frustrate them.

Starting the project

The project is going to take a picture, like you just did, and then apply a filter and add a frame. Start by creating a directory for our project; we're going to put it on the desktop so we can get to it easily, but you can put it wherever you prefer.

```
mkdir ~/Desktop/photobooth && cd ~/Desktop/
photobooth
touch camera.py
```

We'll also make a directory for the finished photos to live in:

```
mkdir ~/Desktop/pics/
mkdir ~/Desktop/frames/
```

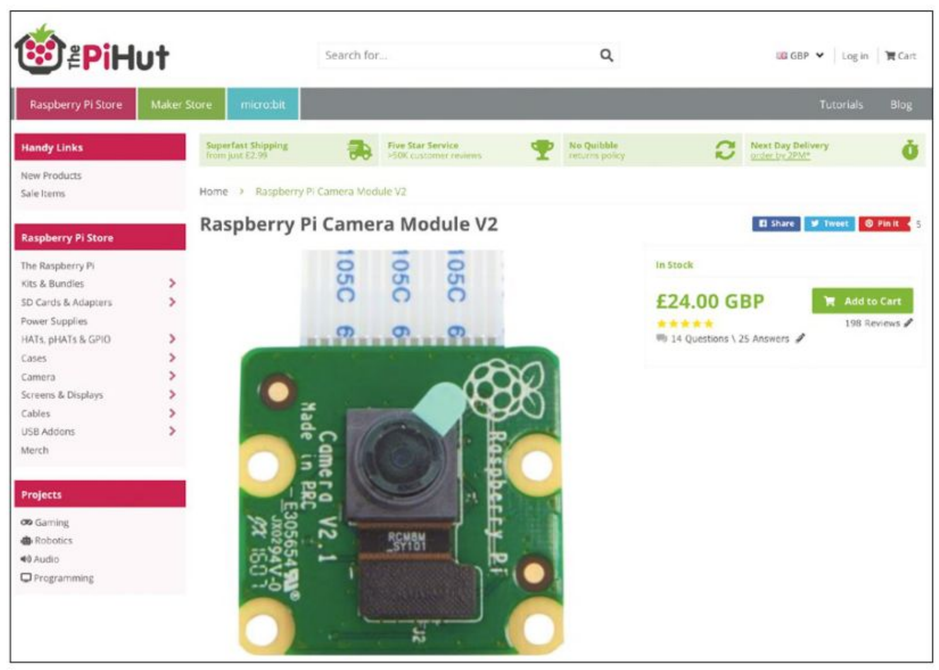
The first version of the script will just take a picture. Add these lines to the **camera.py** file:

```
from picamera import PiCamera
from PIL import Image
print("Starting photobooth")
camera = PiCamera()
camera.rotation = 180
print("\n\nOK... you have 3 seconds to get ready!
Smile!\n")
camera.start_preview()
sleep(3)
camera.stop_preview()
photo = "/home/pi/Desktop/pics/snap.jpg"
camera.capture(photo)
print("All done!")
```

Run this with the line below and then open the picture to see what you captured.

```
python3 camera.py
```

Now of course this hasn't done much more than automate what we did with the test photo we took



Like the Raspberry Pi, the camera is inexpensive and easy to install.

earlier, but it forms the basis for a script that we can use to do much more interesting things once we've got the initial picture.

If you're new to Python or to the *PiCamera* module, there are a few interesting things here to learn. `camera.rotation=180` rotates the picture by 180 degrees to allow for the fact that you probably have the Raspberry Pi on your desk, and the camera ribbon curls over, so the camera ends up upside down. `sleep(3)` pauses the program for three seconds and then `camera.capture` takes a photo and puts it in the location pointed to by the **photo** variable. The camera module is great for learning coding because you can misspell a few words,

» READ THE FLIPPING MANUAL

Many coding manuals are, frankly, awful. They are dense with terminology which you have to learn before the manual becomes meaningful. The *PiCamera* module docs are not like that. They are dry and matter-of-fact, but there are good examples in there. Spend a little time familiarising yourself with them and you'll get much more out of the camera.

For anyone new to manuals like this, look at the sections called 'Getting Started' and 'Basic Recipes'. There are chunks of code which can be copied into your project, modified and used. Newcomers to coding will have to try to understand what's going on, but the syntax and terminology in the examples is not too obscure. For example, the low-light 'recipe' has the lines:

```
...
camera = PiCamera(
  resolution=(1280,720),
  framerate=Fraction(1,6),
  sensor_mode=3)
camera.shutter_speed = 6000000
camera.iso = 800
sleep(30)
camera.exposure_mode = 'off'
camera.capture('dark.jpg')
```

While you might need to know what shutters are and what ISO means, everything around those words will make sense. Similarly, if you wanted to resize the captured image instead of setting the frame to the right size, you could look at the 'Capture resized image' example:

```
camera.resolution = (1024,768)
...
camera.capture('foo.jpg',resize=(320,240))
```

Having understood these examples, you can now look at the documentation of the individual methods and classes further down and learn what else you can do with the camera.

run the script and get errors, but the errors will be pretty self-explanatory because, as a beginner, the correct spelling will be obvious to you. For example, change `capture` to `capure` or `photo` to `phoo`.

Create a frame

Next, let's add a step which the Code Club children being taught really enjoyed: creating a frame for the picture and applying it in the Python script. First, find out the size of the photo you just took by opening it up from the Raspberry Pi desktop. Or you can use the command line tool *file* to get the information:

```
$ cd ~/Desktop/pics
$ file snap.jpg
snap.jpg: JPEG image data, JFIF standard 1.01, aspect ratio, density 1x1, segment length 16, baseline, precision 8, 663x994, frames 3
```

In this example our image has a resolution of 663x994, but you obviously need to use whatever your camera creates. Now create a frame by either drawing it on paper or using *GIMP* or another package. Make it the same size as the picture the script captured. Using *GIMP* or another image editor, delete the centre of the frame and make it transparent. Save it as a transparent PNG in **~/Desktop/frames/frame1.png**.

Now modify the script so that from the capture line onwards you have:

```
camera.capture(photo)
print("Adding the frame")
bg = Image.open( photo )
fg = Image.open( "./Desktop/frames/border1.png" )
bg.paste(fg, (0,0), fg)
bg.save("/home/pi/Desktop/photobooth/finished/snap.png")
print("Done!")
```

When you run this, it will create two files:

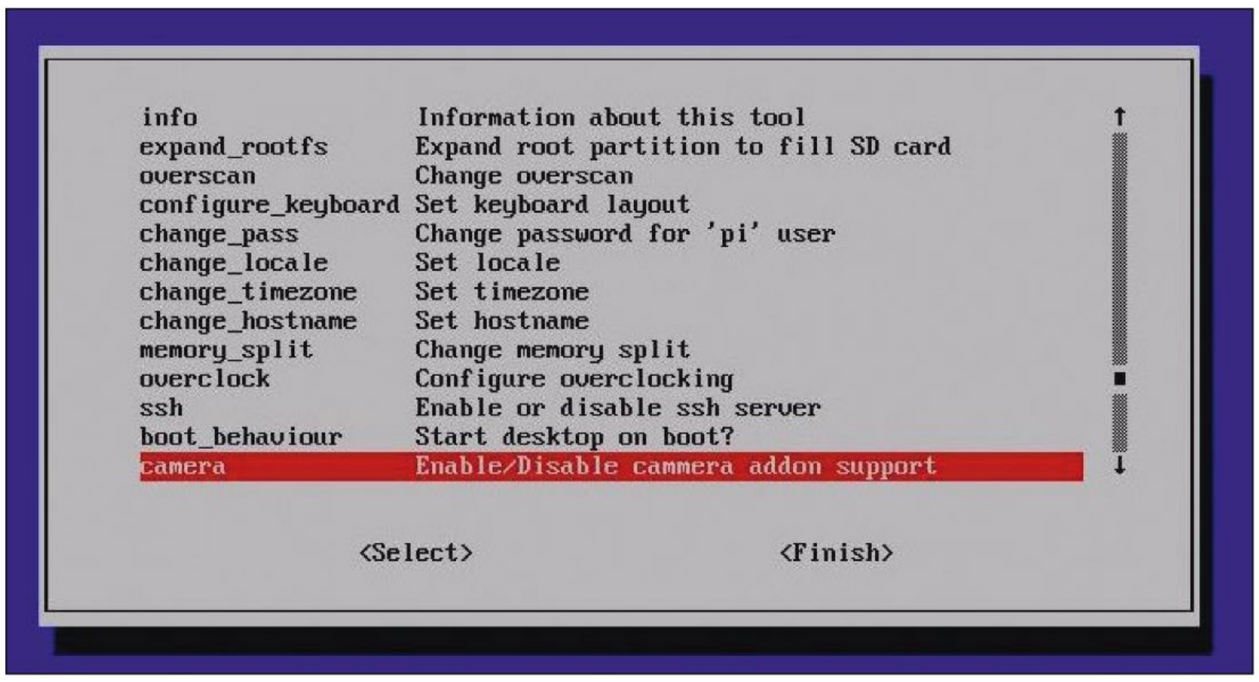
1. The photo, as you're used to doing with the camera.
2. A photo with a frame, using a new *Image* class and `bg.paste`, which pastes one image on top of another.

This code creates the basic photobooth. Run the script again with `python camera.py`. Open up the frame and you should see a nicely framed picture. At this stage, you have enough to start playing around with how the script runs and how it feels for the user.

A really good habit in coding clubs, or any environment where you're teaching or learning coding, is to add extra steps, change the wording, reorganise where the files are stored and replace the frame. These small steps help you to learn the concepts by practising and playing, rather than moving right on to the next thing. Some good tasks are: Create another frame and replace it. Make the welcome message contain more lines and tell the user more of what's going to happen. Make the variables which are used for applying the frame more meaningful.

That last idea moves into thinking about structuring code rather than just getting things working.

Enabling the camera is the only part which feels low-level, but is straightforward.



A choice of frames

It would be fun to have some kind of user input in the script so that you can let the user decide what frame to have. To do this, you will first need to create a few more frames and place them in the frames directory, named **frame1.png**, **frame2.png** and so on.

Next, add a few lines before the `camera = PiCamera()` line:

```
frame = input("Type the number and hit enter. \n>")
frame_id = int(frame)
print("You chose frame %i" % frame)
frame_file = "./Desktop/frames/frame" + frame_id + ".png"
```

Then change the line `Image.open` to:

```
fg = Image.open( frame_file )
```

Now if you run the script again, you'll be given the option to choose your frame. If you're running a coding club, this is a great point to get creative. Make lots of new frames and decide what order to present them in. Or add more output in the script to describe each of the frames so the user knows what they're getting.

Another challenge is to make the script more stable by having it check that the frame file really exists. The challenge we gave the group (but it's optional for this project) was to understand and use the following lines, which will check that the file exists and exit the program if it doesn't.

```
import os
if not os.path.isfile("~/path/to/file.png"):
  raise Exception("File does not exist")
```

As a clue, you should put the `import` line at the start of the file – but the rest is left for you to work out! This gets you thinking about what that code is doing and where to fit it into your program.

Adding effects

We can make the script even more fun by applying some effects which are built into the *PiCamera* module. Start by adding the following line just after the `camera.rotation` line:

```
camera.image_effect = 'colorswap'
```

Run the script again. This is another point to let learners (or yourself) have a play. If you examine the documentation *(see 'Read the flipping manual' on page 87)*, you'll find there are 22 effects to dabble with. Have a play with 'sketch', 'negative' and 'watercolor'.

Just as we gave the user the ability to choose their frame, it would be nice for them to choose their effect from some of those that you've played with. To do this, we introduce a simple `if` statement.

If you're familiar with Python, you'll know that there's a much more elegant way of doing this, using an array to map from numbers to effect names. The reason I chose to do it this way is to make it more readable for first time programmers.

First add the following just after the lines which select the frame:

```
effect = input("Type the number and hit enter. \n>")
effect = int(effect)
```

Now add a couple of `if` statements to select the correct effect:

```
if(effect == 1):
  print("Applying effect 1")
  camera.image_effect = 'colorswap'
elif(effect == 2):
  print("Applying effect 2")
  camera.image_effect = 'solarize'
else:
  print("I didn't understand that. Going ahead with no effect.")
```

This piece of code first asks if the user typed 1 and then if they typed 2. For `1`, the effect is `colorswap` and for `2` it's `solarize`. If you're unfamiliar with this, read it through and if you're using this to teach coding, get your students to read it through.

Then, as a challenge, add some more options. Even for beginner programmers, there is enough information here for them to work out how to make it function. So long as you take a backup of the file before everything goes really wrong (which you should be doing anyway!), you'll get there eventually through experimentation and seeing what works. With this feature added, run the camera script again and test it out.

Setting up the photobooth

What you've built so far is nearly enough to be a really simple working product to show off. The way this has been used in coding clubs in my experience is to set it up as a 'festive photobooth' at the school's Christmas fair – or any fair, come to that.

You can setup a table with the Raspberry Pi on and the camera pointing at a couple of chairs. You can add some fun by providing wigs, feather boas and other frippery. People can pay a donation, run the script, select the frame and the effect they'd like. This creates the picture – but maybe you want to add a second script to send it off to a printer for them to take home.

Be my play thing

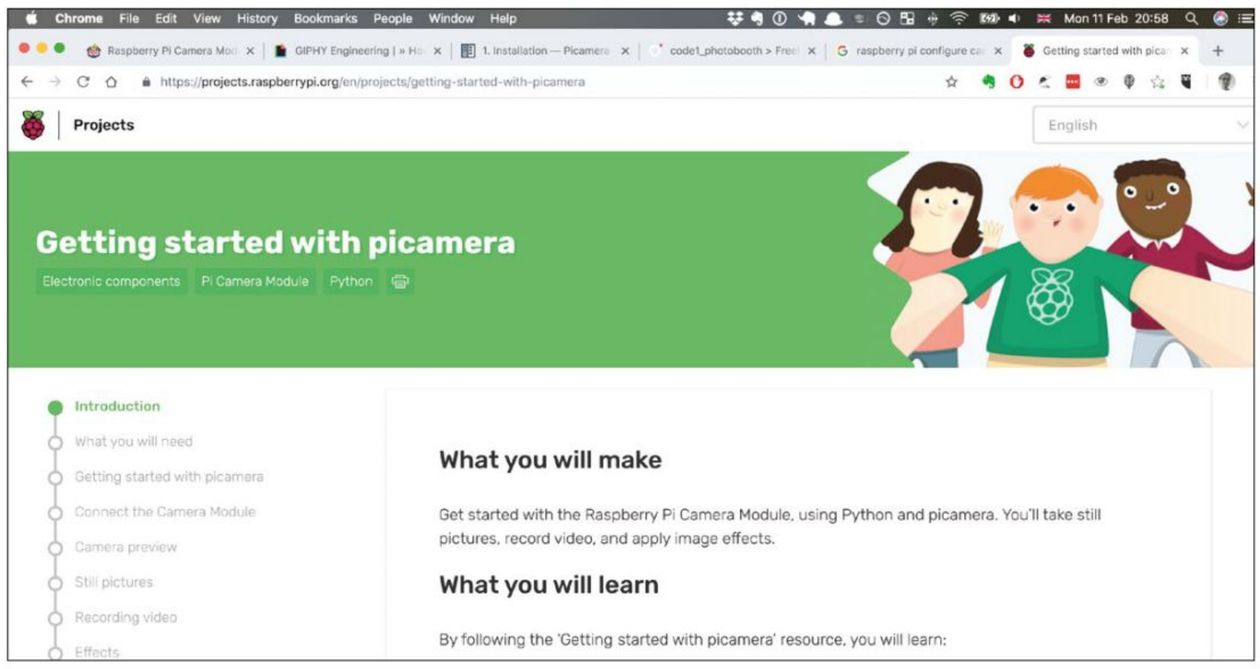
We've only looked at a few examples of the Raspberry Pi camera module, so let's see what else can be done.

In the example we've built, the user doesn't have a chance to preview the scene before the picture is taken, as you would be able to do in a photo booth or selfie app. You could add this using the `start_preview()` function. If you call this, it's going to take over the entire screen, so you'll need to think about using `sleep(x)` to make it sleep for `x` seconds, and then use the corresponding stop function, `stop_preview()`.

If you want to be more advanced, there is an `alpha` option that you can pass to this function to make it partly transparent. Have a look at the documentation for more goodies to play with.

Generate GIFs

Animated gifs are so retro and yet so now. A cool thing to do with the camera output would be to record a short video, convert it to a GIF and display it on a webpage that everyone can see.



Even though this might feel like a big project, there's plenty of support available for the PiCamera module online.

If you did this over time – say, at a school fair – you could end up with loads of funny GIFs being played one after the other. Doing this will take a little reworking, but the core idea is to capture a video instead of a still and generate a GIF from the video.

To capture a video, use the `_recording()` methods, something like this:

```
camera.start_recording('my_video.h264')
camera.wait_recording(60)
camera.stop_recording()
```

This example is lifted directly from the documentation, just to show how easy it can be to piece these things together. Next, after a few minutes of Googling 'convert video to GIF', you'll find an engineering article from Giphy with this line:

```
$ ffmpeg -ss 61.0 -t 2.5 -i input-file.mp4 -f gif input-file.gif
```

If you (or your students) are totally new to those commands then try them out in isolation, and work out what they do by running them. It's really important that when learning, students learn how to try out ideas rather than just copying and pasting code – just as when learning a foreign language, you need to interact with other speakers rather than just learning the word for 'comb' or something.

A learning photobooth

If you're getting pretty confident with the setup so far, here's a project which will be a push: what if the photobooth learnt about people it had seen before, using machine learning? You could store the images of people who had used it, create a model using an open source image categorisation library, and run each image taken past the model to see if the person is recognised.

There are a lot of ways of doing this now. You can use hosted services such as Amazon's AWS, you can find libraries which will hide the details, or you can get into the nitty-gritty with something like *TensorFlow*.

If you have a mix of students – say some who have loads of experience and others who are newcomers – then you could set the newcomers building the photobooth as we've done above and the more experienced building the machine learning model part of this. However you take it from here, there's a lot of coding fun to be had with this simple add-on to the Raspberry Pi.

Make a digital photo frame

Take your Raspberry Pi, HDMIPi and Screenly and turn them into a beautiful digital photo frame.

What you'll need

- Raspberry Pi Model B
- HDMIPi kit
- Class 10 SD Card
- 5.25V Micro USB power supply
- USB keyboard
- Wi-Fi dongle
- Internet connection

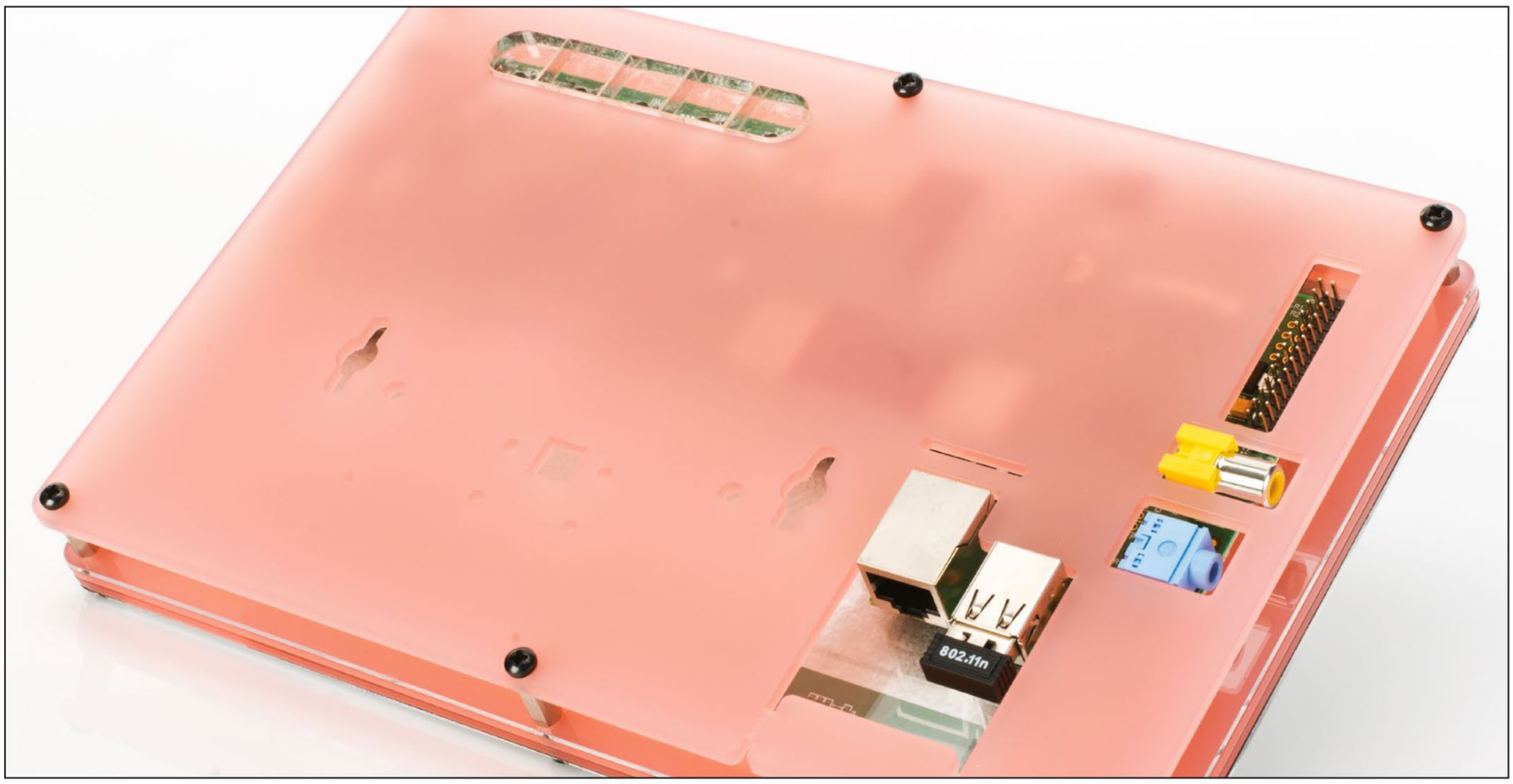
Digital signage is a huge market in modern times, and increasingly we are seeing digital advertising in the public space – be it on street billboards, shopping centres and even in some city taxis. Screenly is an exciting piece of software from Wireload Inc that has set out to reduce the barriers to entry of the digital signage market by employing a Raspberry Pi as its main hardware component for the individual signage nodes. With the powerful VideoCore IV graphics processor at the heart of the Pi and its low electrical power consumption, using it as a digital signage platform is a no-brainer.

Our expert has been using Screenly a lot recently for some projects and it truly is a really great piece of software. He was also one of the first backers of HDMIPi on Kickstarter, and when his reward arrived recently it occurred to him that, together with Screenly, it would make the perfect home-brew digital photo frame and another great Raspberry Pi-based hardware project. In this tutorial, we will show you how to assemble this powerful hardware/software combination for yourself.

01 Order your items

If you haven't already got them in your Raspberry Pi collection, you will need to order all of the items from the "What you'll need" list. The HDMIPi is currently only compatible with Model B of the Raspberry Pi, although a Model B+ version is in the works (the B+ does actually work with HDMIPi, but unfortunately cannot fit into the case). Pretty much any USB keyboard will work with Screenly, including wireless ones, so you do not need to worry about a mouse for this tutorial as it will all be done from the command line.

Finally, a speaker is only necessary if you intend to play videos from the display with sound as well.



Above The reverse view of HDMIPi, showing GPIO and connector cutouts

02 Assemble your HDMIPi

The HDMIPi comes as a do-it-yourself kit rather than a polished product. Not only does this make it cheaper for you to buy, but it also gives it a more hack-like feel in its Pibow-esque acrylic layer case. It is not hard to assemble, but in case you get stuck there is a fantastic assembly video here: **http://hdmipi.com/instructions**.

Getting started

Requirements

Before you begin, you need to make sure you have the following:

- A Raspberry Pi (Model B).
- An SD Card (**>4GB**). Class 10 is highly recommended.
- An HDMI-cable.
- A network connection (with DHCP).
- A keyboard (only required for the installation).
- A monitor/TV that can view full HD (and has HDMI input).

Option 1: Download the custom image

03 Download Screenly OSE

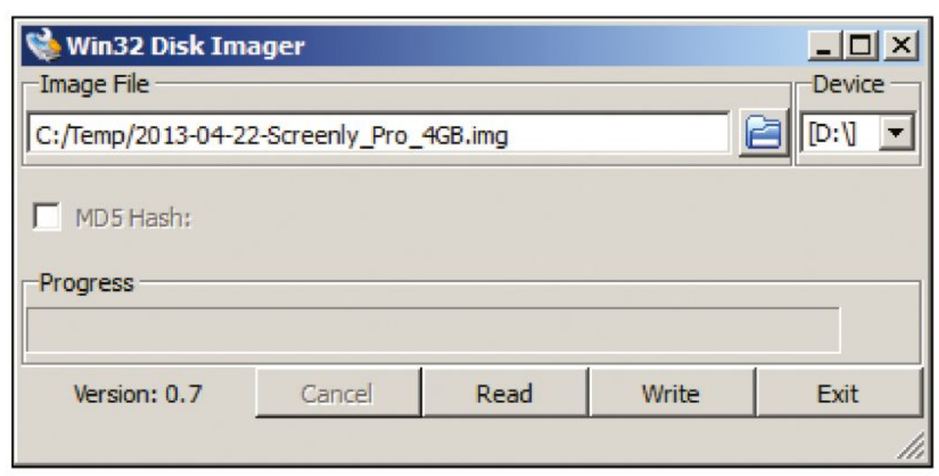
Now that you have the hardware all ready to go we need to download the Screenly OSE SD card image. This is only a 3.7GB image file, however it may not fit on some 4GB SD cards so we would recommend a minimum of 8GB, for extra space for all your pictures and videos. Visit **bit.ly/1wLKIRQ** and download the ZIP file from one of the mirrors under the "Getting started" heading.

04 Flash image to SD Card (Linux)

It's worth noting the value of having a Linux machine at your disposal (or a spare Raspberry Pi and SD card reader) to download the ZIP file in Step 03. This is typically the easiest way to unzip the file and copy the image across to your SD card. Assuming the disk isn't mounted, open a terminal session and type:

```
unzip -p /path/to/screenly_image.zip | sudo dd
bs=1M of=/dev/sdX
```

Make sure that you replace /path/to/screenly_image.zip with the actual path.



05 Flash image to SD Card (Other OS)

If you do not have another Linux machine handy, or a card reader for your Raspberry Pi, you can still do this from other popular operating systems. On Windows you should use Win32 Disk Imager and follow the easy to use GUI. From Mac OS X you have the options of using the GUI-based software packages PiWriter and Pi Filler, or running some code from the command line. Visit **www.screenlyapp.com/setup.html** for more info.

06 Insert SD card and peripherals

Once the Screenly image has been successfully transferred to your SD card, you will need to insert it into the Raspberry Pi within your HDMIPi casing.

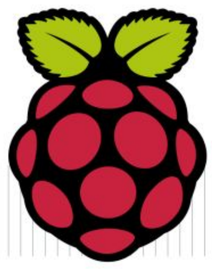
It is a good idea to connect your Wi-Fi dongle and keyboard at this point. Take a look at the image at the top of this page to see where the slots are in relation to the casing. A wired network is also required for the initial setup and for configuring the Wi-Fi connection.

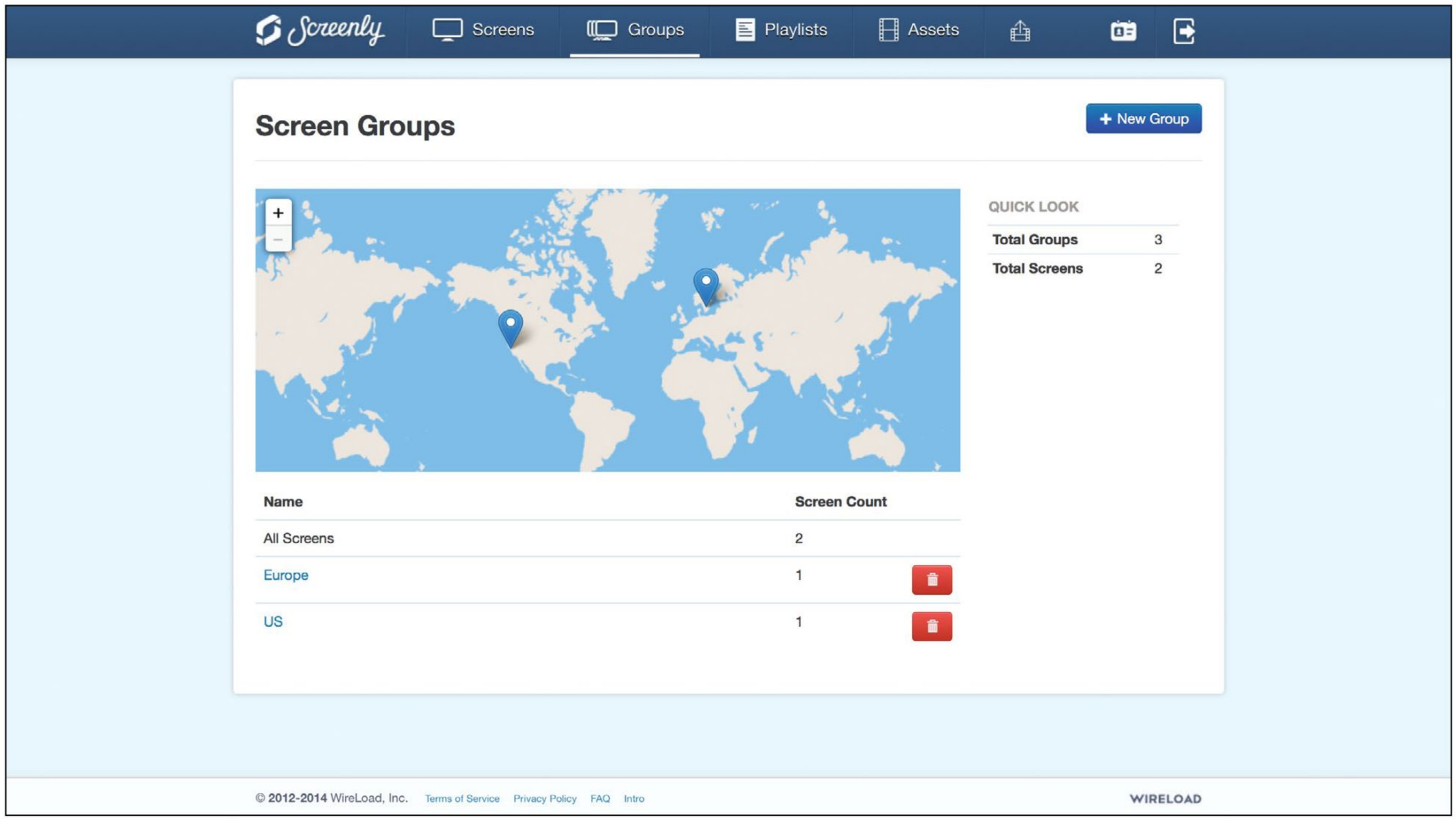
07 Boot up your Raspberry Pi

The HDMIPi driver board has a power output for the Raspberry Pi which means you only need one power supply for this setup. Plug it in and wait for it to boot into the Screenly splash screen. An IP address (of format http://aaa.bbb.ccc.ddd:8080) should be displayed here, which will allow you to gain access to the management dashboard.

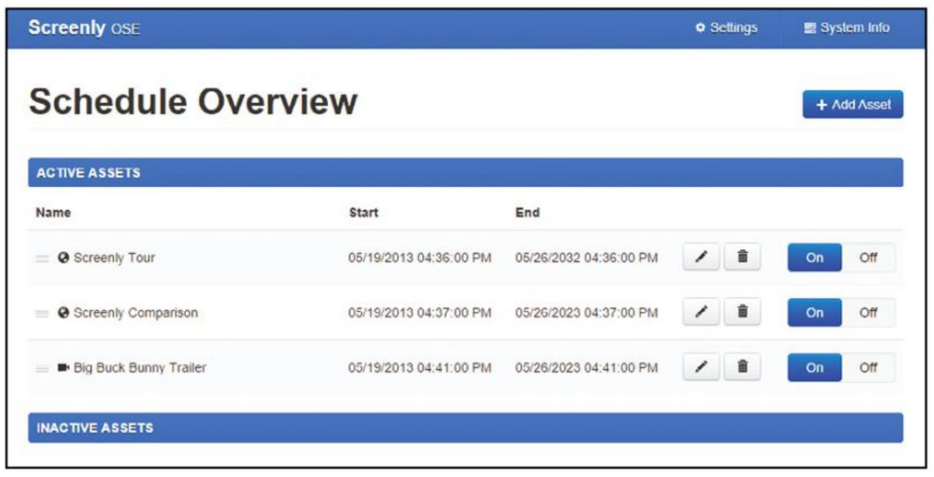
History of HDMIPi

HDMIPi is a collaboration between Cyntech and Alex Eames from RasPi.TV. They wanted to bring a cheap HD resolution HDMI screen to the market that will reduce the cost of having a dedicated monitor for your Raspberry Pi. They took to Kickstarter to launch their idea (**kck.st/17zyaQg**) and there was a huge response to this project from both within and outside the Raspberry Pi community. Over 2,500 people from all over the world enabled them to smash their £55,000 target, and the campaign finished with over £260,000. UNICEF even thought they were good enough to use in their educational projects in Lebanon (**bit.ly/ZDpO8Z**).





Above Screenly Pro can manage multiple screens and has cloud storage too



Screenly Pro Edition

In this tutorial we have used the open source version of Screenly – Screenly OSE. This is a fantastic bit of software and a great addition to the open source ecosystem. At this point, some of you may be dreaming of huge remote-managed display screen networks and the good news is that this is entirely possible with Screenly Pro. This is completely free for a single display screen and 2GB of storage, and it has larger packages for purchase starting at two screens right up to 130+ screens. It also adds a lot of additional features not seen in Screenly OSE – find out more about those on the Screenly website (**bit.ly/1EXl92p**).

08 Disable Screenly video output

Load the IP displayed on the splash screen on the browser of a different computer (you won't be able to do it directly from the same Pi). The Screenly OSE dashboard should now load. Once inside the dashboard, move the slider for Big Buck Bunny to the OFF position or delete the asset entirely.

09 Enter the command line

Once you have disabled the Big Buck Bunny trailer from the web interface, you should now be able to enter the command line easily and you can do this by pressing Ctrl+Alt+F1 on the attached keyboard at any time. Alternatively, you can access the command line over SSH using the same IP address as shown previously on the splash screen.

10 Run the update script

The image file we downloaded from the website is actually just a snapshot release and does not necessarily include the latest Screenly OSE code, as the code updates are made more regularly than the image. It is therefore good practice to run an upgrade to the latest version using the built-in script. You can run the following command:

```
~/screenly/misc/run_upgrade.sh
```

11 Configure Raspberry Pi

Once you are successfully at the command line, you need to type `sudo raspi-config` to enter the settings and then select '1 Expand root file system' to make sure you have access to all of the space on the SD card. Then, change the time zone (option 4 and then 12) so that it is correct for your location. If your screen has black borders around the edge you may also need to disable overscan (option 8 and then A1). We would also recommend changing the default password to something other than raspberry to stop any would-be hackers from easily accessing the Raspberry Pi via SSH (option 2). Once complete, exit without restarting by selecting Finish and then No.

12 Enable and set up Wi-Fi

As Screenly runs on top of Raspbian, the Wi-Fi setup is essentially the same as the standard command line setup within the OS. In order to do this you need to edit the interfaces file using `sudo nano /etc/network/interfaces` and then type in the following code, replacing ssid and password with the actual values:

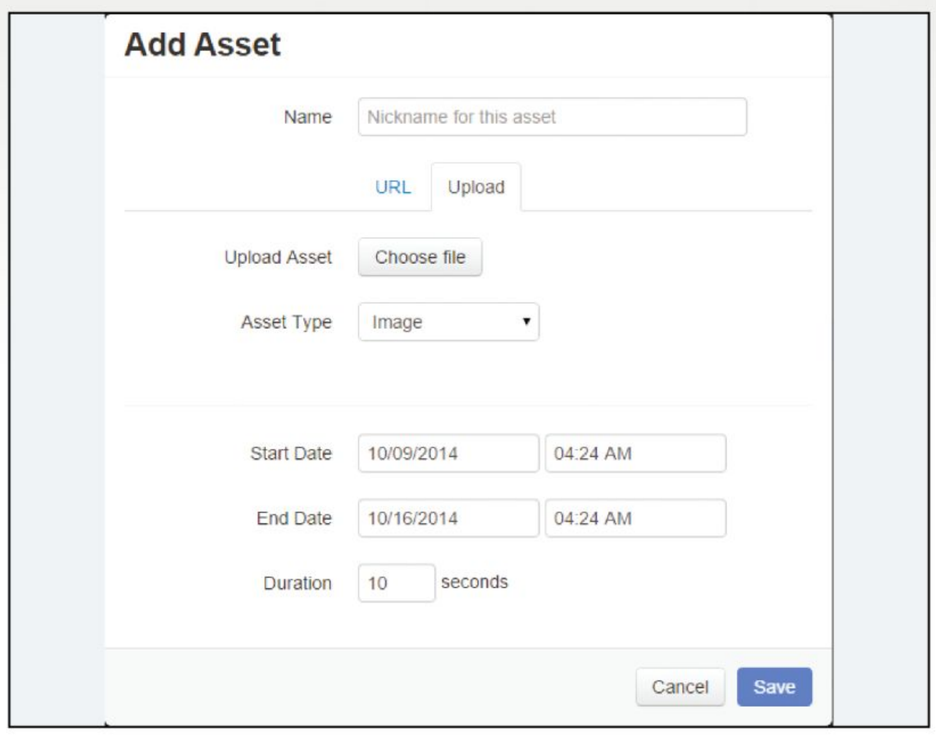
```
auto lo

iface lo inet loopback
iface eth0 inet dhcp

allow-hotplug wlan0
auto wlan0

iface wlan0 inet dhcp
        wpa-ssid "ssid"
        wpa-psk "password"

iface default inet dhcp
```

Then exit and save by hitting Ctrl+X, then Y and then Return.

Above Once fully configured, load your pictures and video to complete your digital photo frame!

13 Test the connection

The easiest way to test the Wi-Fi connection is to shut down the Raspberry Pi using `sudo shutdown -h now` and then remove the wired network connection and reboot the Raspberry Pi by removing and reattaching the microUSB power connector. If the Wi-Fi connection has worked, you should now see the splash screen with IP address again.



14 Upload pictures to Screenly

Once again, you will need to visit the Screenly OSE web interface by entering the IP address into another computer. Since you are now using a wireless connection, the IP address may be different to the previous one. You need to select the 'Add Asset' option at the top right-hand side, which should launch a pop-up options screen. Select Image from the drop-down box and you then have the option of either uploading the image or grabbing it from a URL using the corresponding tabs. Enter the start date and end date of when this image should appear, and how long it should appear on screen for, then press Save. Repeat this step for each of the pictures.

15 Test with video and more

Pictures are great, but Screenly also allows you to display videos (with audio if you wish) and web pages, which really is a huge benefit. This is perfect if you want to enhance your digital photo frame even further or perhaps display the local weather and news to keep yourself informed. Plug in your speaker – we would recommend The Pi Hut portable mini speaker (available from **bit.ly/1xEpBNZ**) or anything similar.

16 Place in situ and enjoy!

Once you have got Screenly all set up and loaded all of your favourite pictures and videos onto it via the web interface, it is now time to enjoy the fruits of your labour! Mould the spider stand (if you have one) into shape by taking the middle two legs at the top and bending them downwards and backwards. Then spread the front-middle legs a bit wider to give a good base and shape the outer legs at the top and bottom to support the screen. You are then ready to give it its permanent home – our expert's is on the mantelpiece over the fireplace!

17 Other project ideas

In this tutorial we have looked at just one fairly basic application of Screenly and the HDMIPi. You could use this powerful open source software to run your digital signage empire, share screens in schools and clubs, or as a personal dashboard using a suitable web page. Whatever you make, please don't forget to take pictures and send them to **Linux User & Developer**!

Access Screenly from afar

The default Screenly image is essentially some additional software running on top of Raspbian OS. This means that SSH is enabled by default (it's why we changed the default password in Step 11) so it's now possible to access the command line, as well as the Screenly dashboard, from outside of your LAN. We recommend setting a static IP for your Screenly-powered Raspi from your router settings and then setting up SSH with keys on your Pi, and port forwarding on your router for ports 22 and 8080. The Screenly dashboard has no login so anyone can access it, but an authentication feature is imminent.

Build a Pi-powered portable bodycam

Christian Cawley wants to record all the evidence that happens on his next family trip. The answer? A Raspberry Pi Zero bodycam.

OUR EXPERT

Christian Cawley has worked for over ten years as a tech blogger and has been covering the Raspberry Pi since it launched.

Bodycams are chest-mounted cameras that have enough storage to record as much, or as little, footage as you like. Regularly used by police around the world, they're also a favourite of certain vloggers. Bodycam footage could be anything from extreme (***LXF*** *Fight Club?–Ed*) sports to recordings of the San Diego Comic-Con and everything in between.

Fancy a bodycam but unsure of whether you would use it? Thanks to the Raspberry Pi Zero, you can build your own. While the Model B boards are more suited to retro gaming and media centre tasks, the Pi Zero and its variants are perfect for camera projects like this.

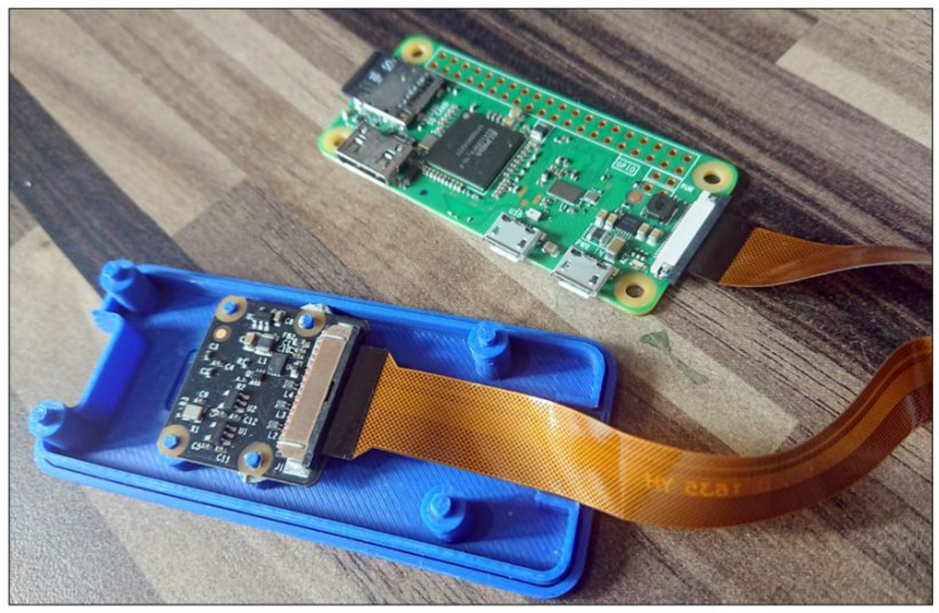
By combining the Pi Zero with a Raspberry Pi Camera Module, rechargeable battery and 3D-printed case, you can set up your own bodycam and record the day's events for posterity. You might even livestream your activity to YouTube (see box on opposite page).

Your bodycam kit

You'll get the best results from this project with the latest version of the Raspberry Pi Zero or Zero W. The only difference between the two is that wireless networking and Bluetooth are available only with the Pi Zero W. Both devices have a 1GHz single-core CPU and 512MB RAM, a microSD slot, and run the latest version of Raspbian.

In addition, you'll need:

- Raspberry Pi Camera Module (any model)
- Camera ribbon cable for Pi Zero
- Portable rechargeable battery
- 3D-printed case



Carefully connect the Raspberry Pi Camera Module to the Pi Zero, ensuring that the ribbon cable is correctly and securely attached.

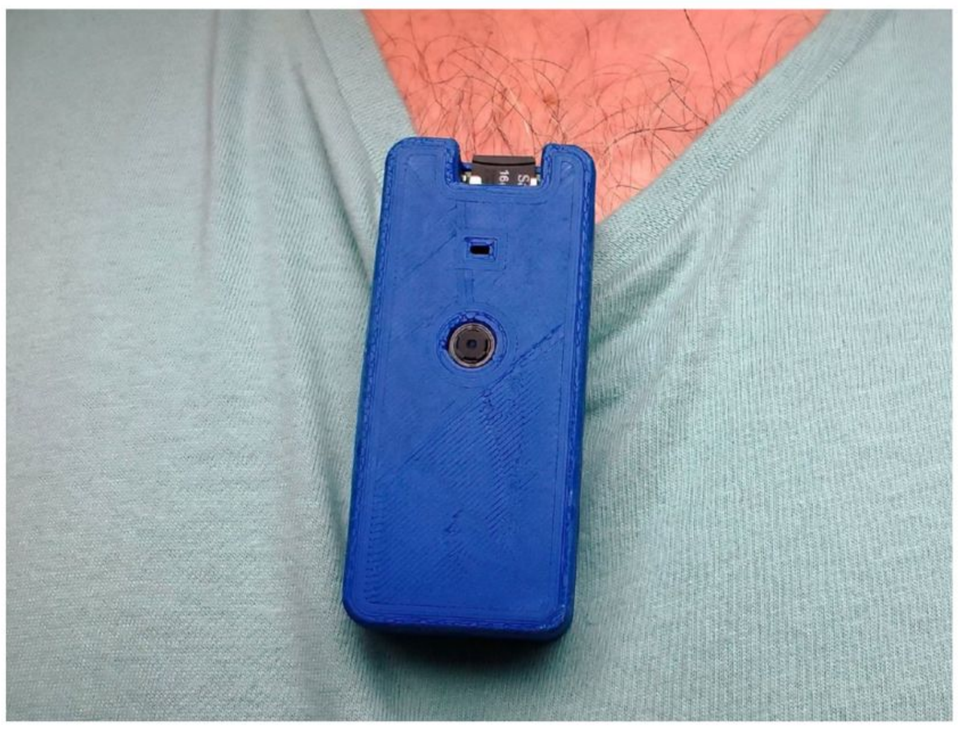
The Raspberry Pi Camera Module you choose will affect the footage. The original camera module was just 5 megapixels, compared with the later 8 megapixels camera. Both have infrared variants, which would result in quite striking 'washed-out' video. These may be more suited to evening bodycam use.

A camera ribbon cable for the Pi Zero is necessary, as the port on the Zero is smaller than that on the Pi Model A and B. Some versions of the camera module come with a suitable cable. If you don't have one, you should be able to order one online for under a fiver.

Your battery option may differ depending on your budget, your chosen case and what you already have handy. The easy solution is a portable smartphone charging battery. For onboard power, however, a 500mAh lithium-ion polymer battery and PowerBoost 500 Charger will do the job.

Keeping the bodycam tidy in a 3D-printed case is a smart choice. Two of the two best options are the One Piece Raspberry Pi Zero + Camera Case and the (**www.thingiverse.com/thing:1595429**) Raspberry Pi Zero Wearable Camera Case by Adafruit (**www.thingiverse.com/thing:1649799**). The Adafruit option comes with a built-in clip and a hole for attaching a cord and wearing as a lanyard. Don't own a 3D printer? Look online for a local or mail order 3D-printing business.

Whichever case you decide upon, and it may be



This 3D-printed bodycam case features a stylish clip for attaching to your clothes.

something completely different, it's wise to assemble the project before installing the boards in a case. Don't do this until you're certain everything is working right.

If you're planning to power the Pi Zero using the li-Ion polymer battery and PowerBoost, you'll need a soldering iron. Connect two wires around 5cm in length to the + and - pins on the PowerBoost, then to the 5V and GND pins on the Pi Zero (pins 4 and 6).

Should you go full Raspbian or Raspbian Lite for your operating system? A browse of the Raspbian downloads page at **www.raspberrypi.org/downloads/raspbian** reveals three options, rather than two. There's the Raspbian Buster download with a bunch of preinstalled applications, a stripped-down Raspbian Buster with Desktop (a version without the apps), and then Raspbian Lite, without a desktop.

For the purposes of demonstrating this project, we used Raspbian Lite. However, for the most flexible option, you might consider Raspbian Buster with Desktop. Given the amount of space you'll need on the SD card, it would be unwise to use the largest version of Raspbian. For quick results, copy Raspbian to the SD card using *Etcher* from **www.balena.io/etcher**, for Linux, Mac and Windows.

With Raspbian installed, connect a keyboard, mouse and monitor. Alternatively, insert the SD card into your computer and create an empty file called **ssh** in the **/boot/** directory to enable SSH mode. For wireless use, you'll also need to create a **wpa_supplicant.conf** file. Add the following lines to it:

```
ctrl_interface=DIR=/var/run/wpa_supplicant
GROUP=netdev
update_config=1
country=US
network={
ssid="SSID"
psk="PASSWORD"
key_mgmt=WPA-PSK
}
```

Take the time to input your own SSID and network passkey to ensure the device connects to your network. With SSH set up, the following steps should be easier.

Connecting the camera is straightforward. Start by attaching the cable to the Pi Zero, then to the camera. Ensure the securing clips on both the camera and the Pi are loosened before inserting the ribbon, silver side facing up. Snap the clips back into place, then power up your Raspberry Pi Zero.

On the command line, open the configuration tool:

```
sudo raspi-config
```

Here, select '5 Interfacing Options > P1 Camera' then Yes to enable the camera. Once enabled, go to '7 Advanced Options > A1 Expand Filesystem'. Reboot the Pi when prompted.

The raspicam software is already installed with Raspbian, so to create a video, all you need is:

```
raspivid -o video.h264 -t 10000
```

This will record 10 seconds (10,000 milliseconds) of footage. You'll probably want more than this, however. As such, some experimentation will be needed. The best results, of course, will be enjoyed with a larger microSD card – the Pi Zero supports up to 256GB. If you're using the bodycam for sports, use the exposure setting:

```
raspivid -o video.h264 -ex sports
```

If you're concerned about the amount of storage being used, the stop-motion bodycam is a smart alternative. Start by checking for package updates and install the WiringPi library:

```
sudo apt update
sudo apt install -y wiringpi
```

Next, create a new file, **timelapse.sh**:

```
sudo nano /boot/timelapse.sh
```

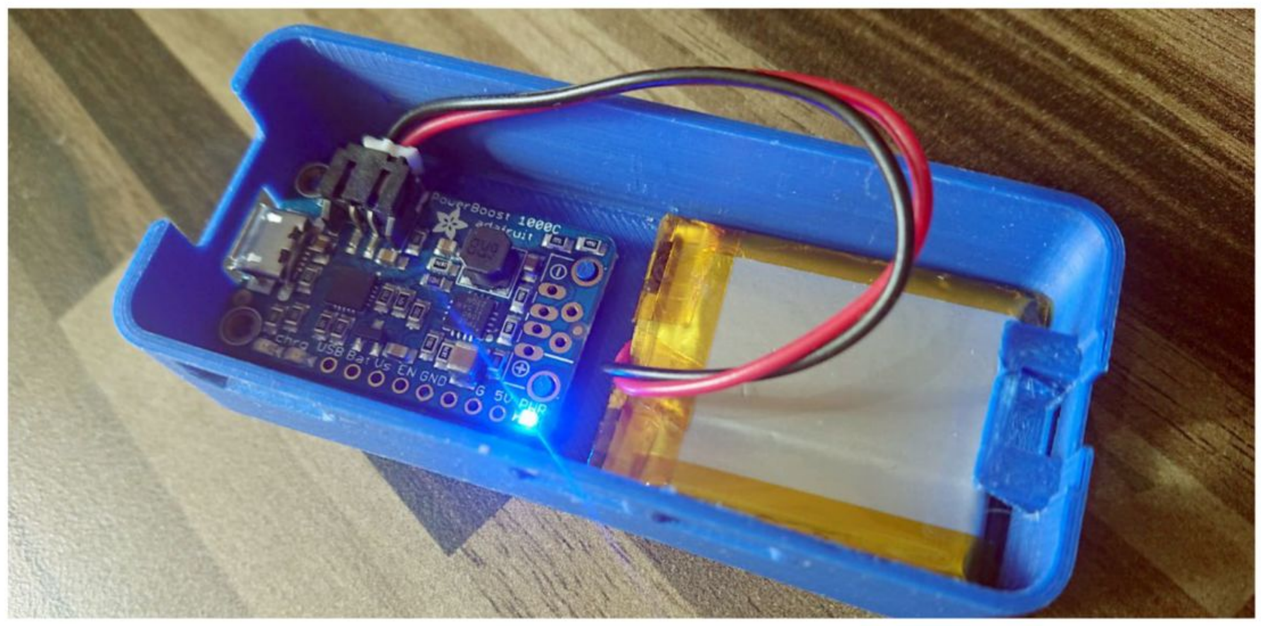
Next, head to Adafruit at **http://bit.ly/LXF256-timelapse** and grab the timelapse script. This is designed to work with an LED. Copy the timelapse script into **timelapse.sh**, then exit *nano* with Ctrl+X and confirm with Y. So that the code launches on startup, edit the **rc.local** file:

```
sudo nano /etc/rc.local
```

Here, add a line before the exit 0.

```
sh /boot/timelapse.sh 2>/dev/null &
```

With the Pi Zero hooked up to the power supply and camera, a script for capturing video or time-lapse frames, and a case, your project is ready to go.



A lithium-ion polymer battery and PowerBoost makes the project more flexible than using a portable mobile phone charger.

QUICK TIP

An LED indicates when the bodycam is recording. Solder two 5cm wires to a mountable LED, then to GPIO5 and the adjacent GND (pins 29 and 30).

» STREAM FOOTAGE TO YOUTUBE

With a Raspberry Pi Zero bodycam set up and running, you'll probably have an eye on the limits of storage. After all, standard high definition video is going to fill up most microSD cards within a few hours. While timelapse is an alternative, it can't be streamed – you'll have to upload the combined video file later.

A smart option, therefore, is to build the project with a Raspberry Pi Zero W (or the pre-soldered GPIO model, the Raspberry Pi Zero HW) and use your mobile device to stream footage directly to YouTube if you're heading out.

You'll need a YouTube account to set this up, along with a channel stream key. You can find this in the Go Live option on YouTube; add it to your *avconv* command. It should look something like this:

```
raspivid -o - -t 0 -vf -hf -fps 30 -b 6000000 | avconv -re -ar 44100 -ac 2
-acodec pcm_s16le -f s16le -ac 2 -i /dev/zero -f h264 -i - -vcodec copy
-acodec aac -ab 128k -g 50 -strict experimental -f flv rtmp://a.rtmp.
youtube.com/live2/[your-secret-key-here]
```

Note that not only is your mobile data plan at risk, but also variations in bitrate might result in the stream ending. As such, be sure to use this option only in places with reliable 4G or 5G.

Turn a Pi into a router

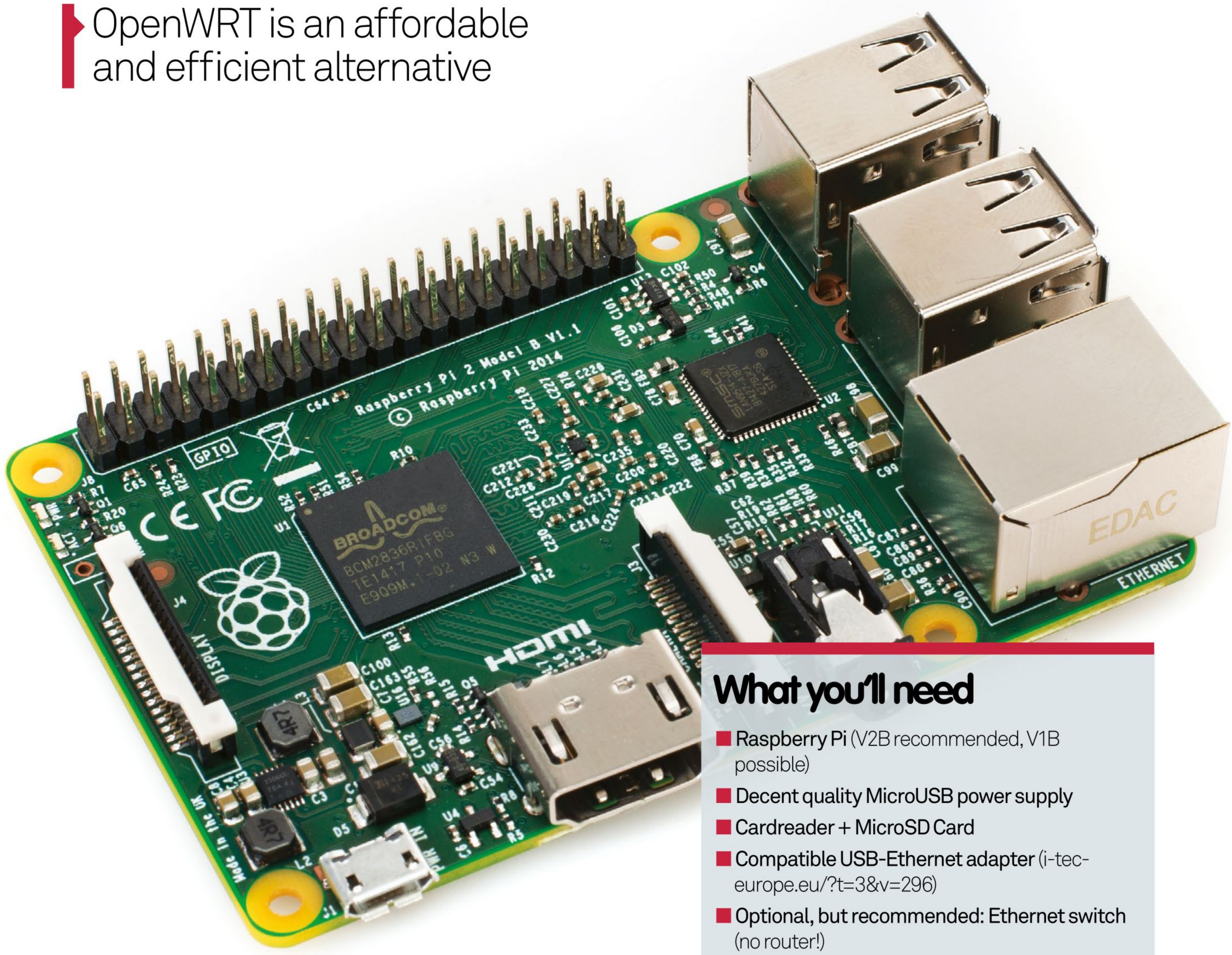
Learn the basics of OpenWRT using a Raspberry Pi as a router.

Controlling the interconnects between various devices is paramount to keeping systems secure and safe. Sadly, most router operating systems are closed source – finding vulnerabilities in them is difficult to impossible. Sadly, running dedicated open-source router operating systems is not a solution for the average user, as they tend to demand high-end hardware with prohibitively high prices.

OpenWRT is an affordable and efficient alternative. It foregoes some of the complexities found in traditional router operating systems, thereby allowing for lower hardware requirements. The community has ported the system to various routers: with a little care, a compatible router can be found for £100 or less. Invest a few hours of your time to transform it into a lean and mean fileserver, torrent client or – configuration allowing – even a system capable of controlling real-world hardware via serial links.

In the following pages we will introduce you to the basics of OpenWRT using a well-known single-board computer. That knowledge can then be applied to a variety of other, more suitable hardware solutions.

OpenWRT is an affordable and efficient alternative

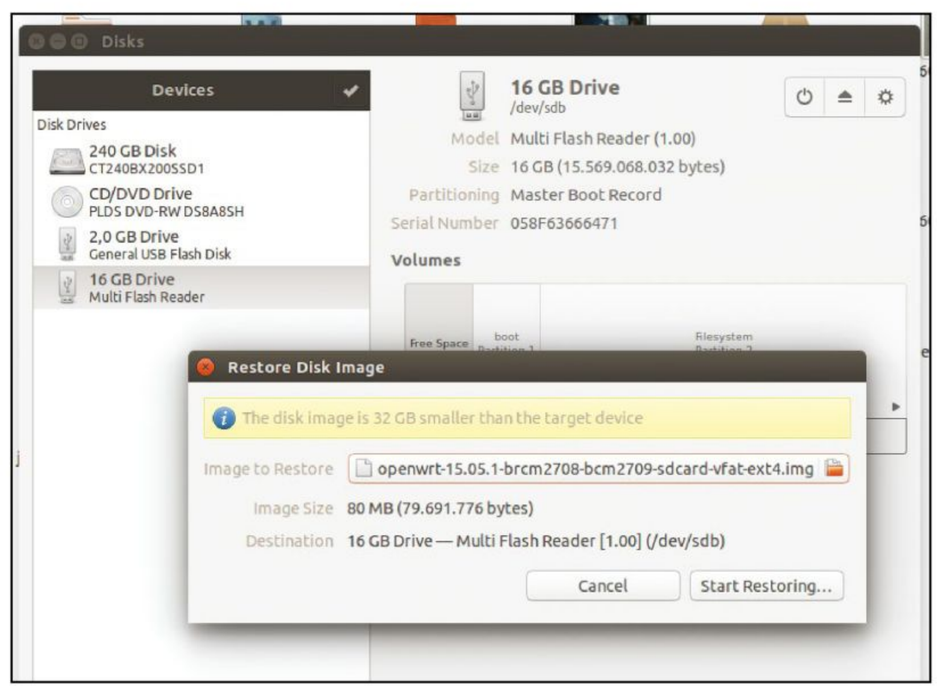


What you'll need

- **Raspberry Pi** (V2B recommended, V1B possible)
- **Decent quality MicroUSB power supply**
- **Cardreader + MicroSD Card**
- **Compatible USB-Ethernet adapter** (i-tec-europe.eu/?t=3&v=296)
- **Optional, but recommended: Ethernet switch** (no router!)

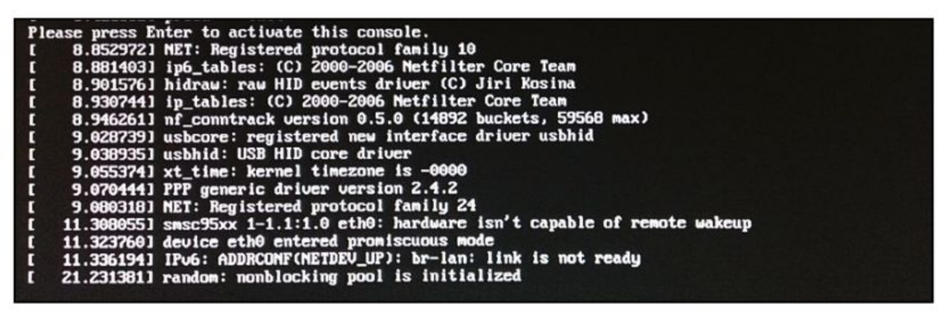
01 Set it up

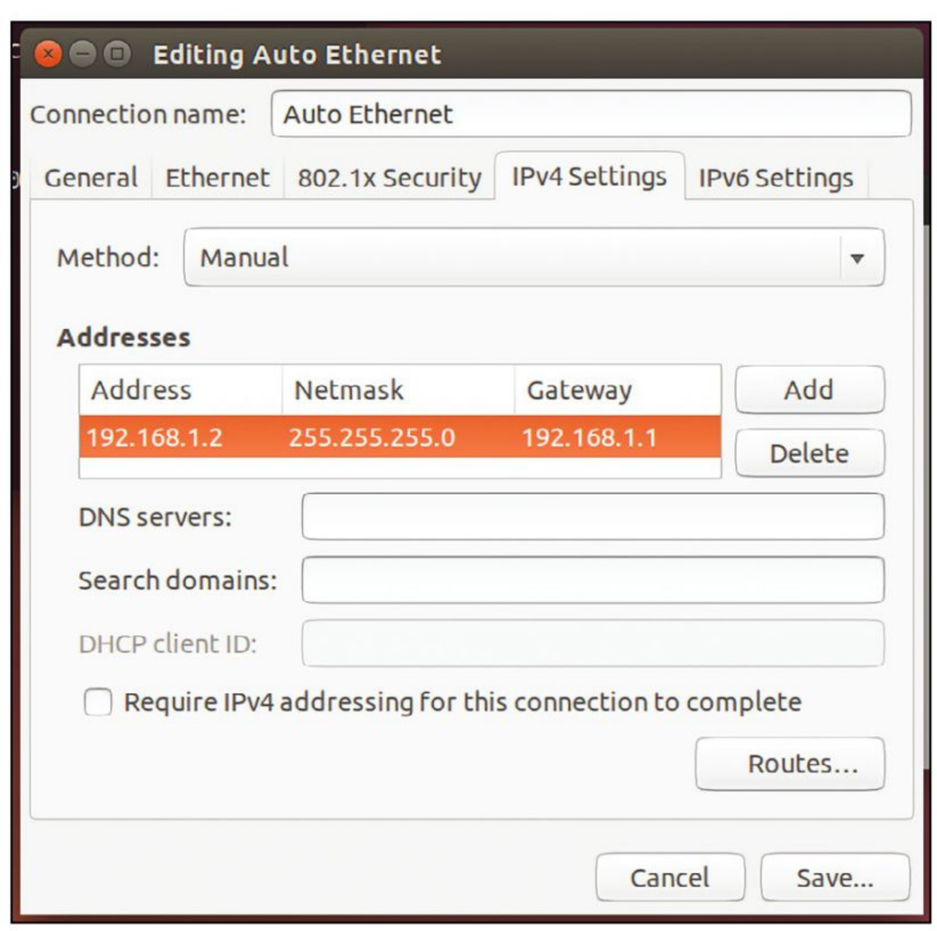
Deploying an operating system requires you to be in possession of a suitable image: due to differences in the hardware, RPi 1 and 2 are targeted with different files which can be downloaded at **http://bit.ly/1T7t4UC**. The following steps are performed on a Raspberry Pi 2 using Chaos Calmer 15.05.1. Burn the image openwrt-15.05.1-brcm2708-bcm2709-sdcard-vfat-ext4.img to the SD card in a fashion of your choice: Ubuntu's Image Writer is the utility shown in the figure. Finally, insert the SD card, connect the RPi's native ethernet port to your PC and power up the contraption. Interested individuals can connect an HDMI monitor in order to see the boot process "live".



02 Get connected

Starting OpenWRT on a Raspberry Pi 2 takes about half a minute: when done, the message shown in the figure will appear. At this point, the ethernet port of the Raspberry Pi 2 will be set to a fixed IP address of 192.168.1.1 and will await network connections from other workstations. Open the "Network connections" applet of the host, and configure it to use a static IP address via the settings shown in the figure.





03 Telnet or SSH?

Chaos Calmer 15.05.1 keeps the Telnet service open on unconfigured instances. The first bit of work involves connecting to the Telnet client: invoke the passwd command to set a new password. Complaints about low password strength can be ignored at your own peril: passwd will not actually prevent you from setting the passcode to be whatever you want, but hackers might be delighted about the easier attack vector.

Once the new root password is set, the Telnet server will disable itself. From that moment onward, your OpenWRT instance can only be controlled via ssh.

```
tamhan@tamhan-thinkpad:~$ telnet 192.168.1.1
Trying 192.168.1.1...
Connected to 192.168.1.1.
Escape character is '^]'.
. . .
root@OpenWrt:/# passwd
Changing password for root
New password:
Bad password: too short
Retype password:
Password for root changed by root
- - -
tamhan@tamhan-thinkpad:~$ ssh root@192.168.1.1
The authenticity of host '192.168.1.1 (192.168.1.1)'
can't be established.
RSA key fingerprint is 11:80:4b:14:cc:b8:9a:a6:42:6a:
bf:8d:96:2a:1b:fa.
Are you sure you want to continue connecting
(yes/no)? yes
Warning: Permanently added '192.168.1.1' (RSA) to
the list of known hosts.
```

04 Let's play nice

The following steps assume that your router will live behind another router. As the activation of USB support requires the downloading of a batch of packages, our first act involves making OpenWRT play nicely with the rest of the network. As the stock distribution includes only vi, open the web interface by entering http://<ip> into a computer of your choice. Next, click Network>Interfaces and click the Edit button next to br-lan. Set the protocol field to DHCP client and select the Switch Protocol button.

Finally, click Save&Apply, close the web page and disconnect the RPi from your PC. Next, connect both PC and Pi to the existing router and run nmap as root in order to find its newly-assigned IP address.

The command shown here is a little nifty in that it instructs nmap to scan the entire 255 addresses of the subnet – be sure to adjust it to your local environment. Furthermore, keep in mind that the IP settings of the PC must be restored to the ones used originally, with a reboot recommended for good practice.

```
tamhan@tamhan-thinkpad:~$ sudo nmap -sn
192.168.1.0/24
```

Limited support for Raspberry Pi 3

On paper, the RPi 3's embedded Wi-Fi module makes it a perfect access point. This is not the case for two reasons: first of all, the range of the module has been shown to be abysmal in lab tests. Second, BroadComm has not released the driver code – at the time of going to press, its use in OpenWRT is unsupported.

Pis make bad routers

Even though the Raspberry Pi makes a great demo and evaluation system, using it in practice might lead to suboptimal performance. This is caused by the unique bus architecture: both ethernet ports must share the USB bandwidth. On the RPi 2, this problem is mitigated by the significantly higher CPU performance.

For large networks, using an X86 based embedded system tends to yield better results. Single-board computers like the BananaPi are another alternative, but tend to crash when confronted with specific ethernet packages.

```
Starting Nmap 6.40 ( http://nmap.org ) at 2016-05-03 21:14 CEST
. . .
Nmap scan report for 192.168.1.104
Host is up (-0.099s latency).
MAC Address: B8:27:EB:53:4E:D9 (Raspberry Pi Foundation)
```

05 Deploy missing USB drivers

At this point, our OpenWRT instance is connected to the internet at large. This allows opkg to download required packages – connect yourself using SSH and the IP address determined by NMAP, and proceed to downloading the packets listed in the code accompanying this step. When all modules are installed, entering dmesg will show that the ASIX ethernet interface has been detected and configured as interface eth1 according to the figure.

```
opkg update
opkg install kmod-usb2 usbutils kmod-usb-core
opkg install kmod-usb-net kmod-usb-net-asix
```

06 Connect

Even though dongles based on the ASIX AX88772B are quite common, not being able to procure one does not condemn your experiment to failure. Connect the USB to LAN bridge to a Raspberry Pi running Raspbian and enter the **lsmod** command. It will provide you with information about the driver modules being used, which can then be tracked down on OpenWRT. Googling <chipset> openwrt or <productname> openwrt can also yield useful results.

07 Open the web interface

After completing the kernel configuration process, our new interface is ready and awaits the deployment of a configuration. As the OpenWRT image provided for the Raspberry Pi restricts us to vi (nano will not install), configuration is best done via the web interface we touched on earlier. It can be accessed by pointing your browser at the URL of the router; log-in can be accomplished via the root password used on the command line.

08 Let's get routing

The newly-created USB ethernet port will be used to connect clients: you can connect either a "dumb switch" or a single device. In both cases, a DHCP server is needed in order to provide IP addresses to the clients.

Click the Add new interface button, and name the new Interface Clients. Next, select the protocol to be Static address and select the newly created interface eth1. Next, scroll to the bottom of the window and click the Setup DHCP Server button in order to fully populate the form.

With that, the IPv4 address and broadcast fields must be set up. Finally, click Save & Apply in order to commit the changes to the network stack. Next, open the network configuration once again and set the Firewall Settings to the firewall zone LAN.

09 Rearrange the interfaces

By default, the LAN interface is bridged. This is not necessary: open its properties, select the Physical Settings tab and unselect the Bridge interfaces checkpoint. Next, open the Firewall settings tab and assign the WAN zone. Finally, another click on Save & Apply makes OpenWRT assign the attributes leading to the configuration shown in the figure accompanying this step (see image on the right).

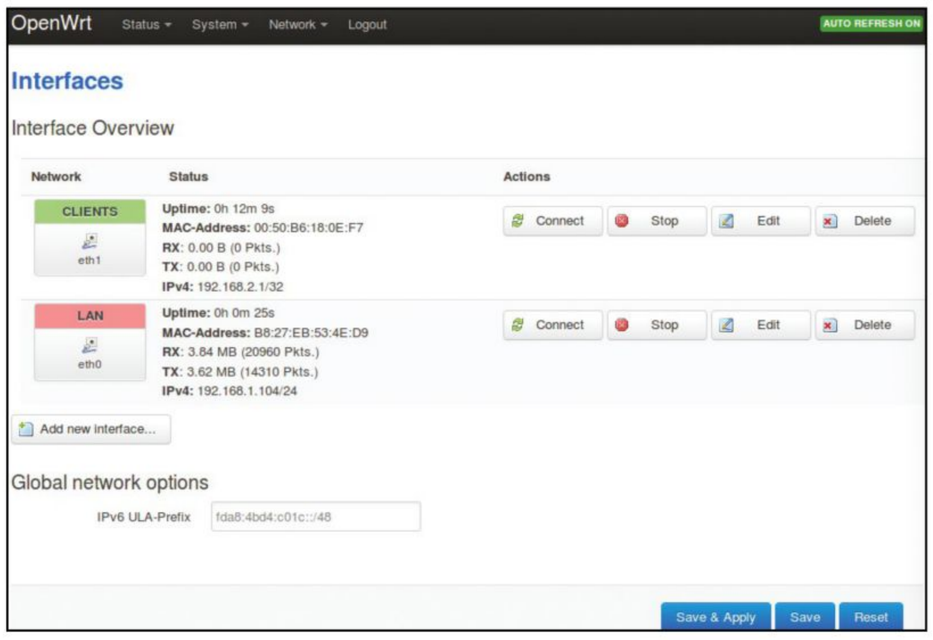
10 Firewall ahoy!

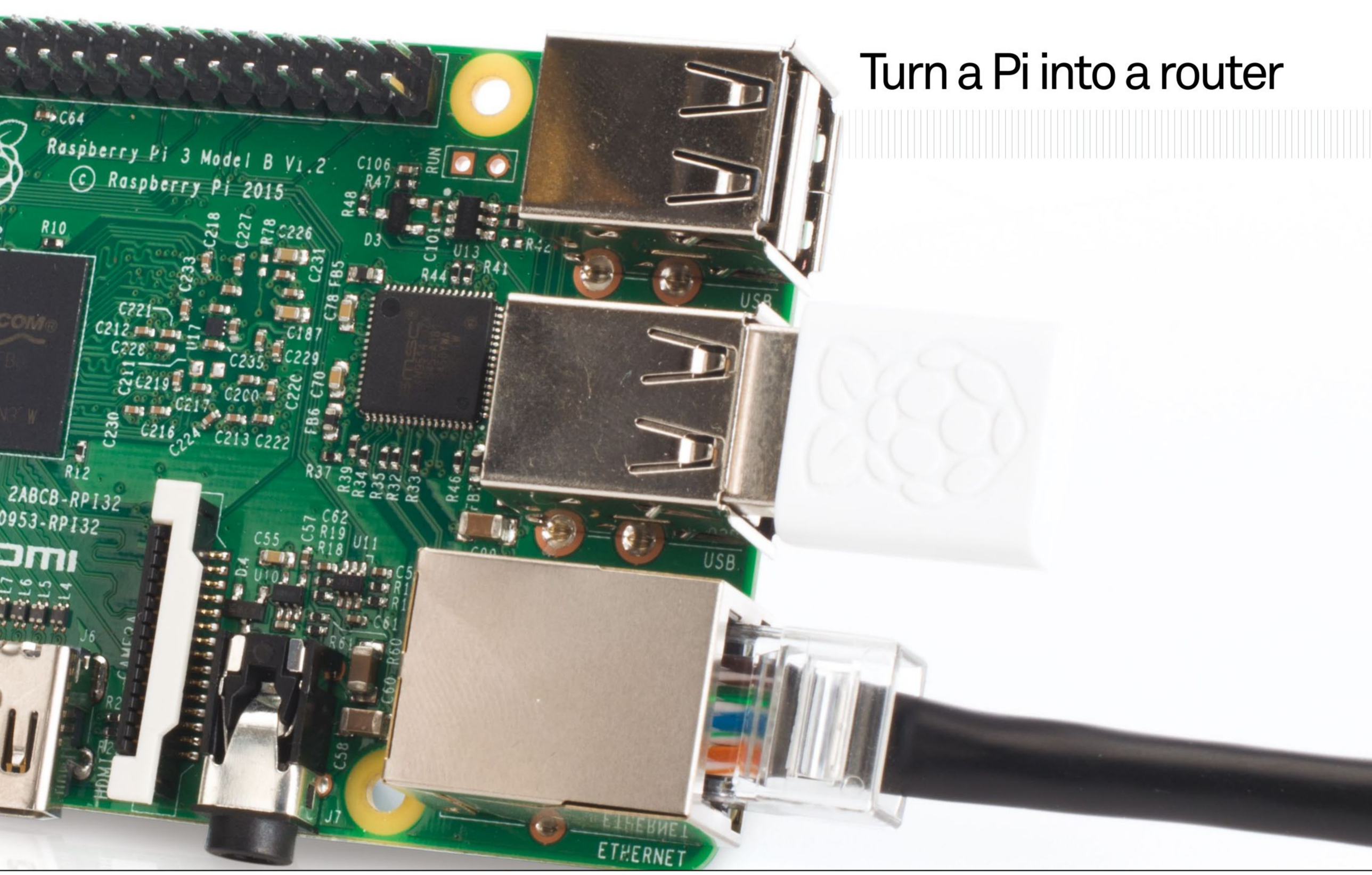
From this point onward, attempting to interact with the LuCI frontend from "outside" of the network will lead to 'Unable to connect' errors – by default, remote configuration is not allowed to make attacks on OpenWRT more difficult.

Solve this problem by disconnecting the workstation from the "outer router", and connect to the Raspberry Pi's USB network interface instead. Then, perform an ifconfig command and connect to the standard gateway in order to open the LuCI interface once again.

Should you find yourself in the situation that no IP adress is assigned to the workstation, reboot the process computer and reconnect the ethernet cable.

```
tamhan@tamhan-thinkpad:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 28:d2:44:24:4d:eb
  inet addr:192.168.2.157  Bcast:192.168.2.255  Mask:255.255.255.0
inet6 addr: fe80::2ad2:44ff:fe24:4deb/64 Scope:Link
```





11 Test the presence of the router

As long as all other network connections are disabled, the workstation can connect to the internet only via the RPi. Enter "mtr www.google.com" in a command line in order to generate the tree structure shown in the figure accompanying this step – from a latency point of view, our OpenWRT access point looks quite good when operating under light load.

12 Analyse the network status

Generating live diagrams with further information about the state of the router is an interesting feature. Open LuCI and select Status > Realtime graph in order to open a set of diagrams telling you more about CPU and network loads.

```
tamhan@tamhan-thinkpad: ~
                          My traceroute  [v0.85]
tamhan-thinkpad (0.0.0.0)                                  Wed May  4 00:03:50 2016
Keys:  Help   Display mode   Restart statistics   Order of fields   quit
                                         Packets               Pings
 Host                                  Loss%   Snt   Last   Avg  Best  Wrst StDev
 1. 192.168.2.1                         0.0%   135    0.5   0.4   0.3   0.7   0.0
 2. 192.168.1.1                         0.0%   135    0.9   0.9   0.8   1.1   0.0
 3. 192.168.0.1                         0.0%   135    1.0   1.0   0.9   1.6   0.0
 4. ???
 5. 84.116.25.33                        0.0%   135   15.8  11.2   7.6  29.2   4.2
 6. at-vie15a-rd1-ae31-2048.aorta.net   0.0%   134   41.6  23.1  19.3  56.3   5.0
 7. 213-46-160-77.aorta.net             0.0%   134   33.3  38.8  20.2  77.0  11.8
 8. de-fra03b-ri1-ae12-0.aorta.net      0.0%   134   20.2  22.8  18.8  51.5   4.9
 9. 213.46.177.42                       0.0%   134   30.1  22.4  19.4  41.6   3.8
10. 216.239.59.60                       0.0%   134   20.7  24.8  19.1  49.9   5.6
11. 216.239.57.125                      0.0%   134   20.9  23.3  19.2  42.1   4.7
12. 108.170.232.77                      0.0%   134   23.1  26.2  22.4  42.5   4.5
13. 74.125.37.88                        0.0%   134   53.3  34.5  30.1  58.7   5.2
14. 66.249.95.142                       0.0%   134   33.0  34.7  30.0  56.7   5.4
15. 216.239.41.129                      0.0%   134   41.0  33.3  30.5  48.5   3.1
16. ber01s15-in-f3.1e100.net            0.0%   134   56.6  34.9  30.1  59.3   5.8
```

13 Deploy file system support

If your router contains a USB port, it can – in theory – be used to access various external USB storage media. Sadly, the required packages are not provided out of the box. This problem can be remedied by deploying the following packages via opkg:

```
kmod-usb-storage required
  kmod-usb-storage-extras
block-mount
  kmod-scsi-core
```

In addition to that, a kmod-fs-* package containing the drivers for the file system is required. One small gotcha awaits all those who want to access FAT filesystems – the relevant package goes by the name "kmod-fs-msdos".

14 Learn more

OpenWRT can be used for a variety of topics not discussed here due to space constraints. The extremely helpful OpenWRT project team provides a set of step-by-step recipes at **https://wiki.openwrt.org/doc/howto/start** – if you feel like implementing something, check whether someone else has already walked the trek for you!

15 Find supported hardware

Our current contraption on these pages – simply made up of a Raspberry Pi and a batch of peripherals – works well for evaluation purposes, but is not particularly well suited to practical deployments. Should you feel like finding a dedicated router, start out by looking at the compatibility list provided at **https://wiki.openwrt.org/toh/start**. Please be aware that router manufacturers tend to change their hardware quite frequently: in some cases, more than twelve revisions with completely different integrated circuits are known.

16 Hardcore debugging

Should you lock yourself out of your OpenWRT router, don't fret: if the memory is not soldered in, simply mount it with a card reader of your choice. Most, if not all, Linux distributions will display the contents of the file systems immediately – accessing some of the files requires that the file manager is run with root rights (**sudo nautilus**).

Protect your identity with the VPiN

Adam Oxford turns an old Raspberry Pi in to a VPN gateway for your home.

OUR EXPERT

Adam Oxford spends his time between writing, teaching, mentoring and camping in the South African landscape.

Caring about your online privacy and actually following best practices to protect it are two different things. As *Linux Format* readers, it's likely that you know the principles of what we should do to avoid the great data collectors and geo-locators in the sky, but maintaining constant vigilance is an effort that soon gets tiresome, if not downright impossible.

For example, you know that you should be using a VPN to avoid ISP-level filters imposed by various pieces of UK law. Setting up and using a log-free VPN service from your Linux desktop is straightforward enough, but what about all the other devices in your home? For example, if you use a games console to watch iPlayer, Netflix or YouTube on your TV, how do you route traffic from there effectively? No consoles have built-in VPN software. There's no app in the PlayStation store.

One solution would be to buy a router that can connect directly to a VPN service, protecting all the traffic on your home network a single stroke. Or, if you have an older router, to modify it with an OpenWRT (*see LEDE in* **LXF234**) firmware and add controls such as these in.

Or, if you want cheaper and more straightforward solution, you could use a Raspberry Pi...

Pi right there

With just a few fairly simple scripts, you can configure any Raspberry Pi to be a headless VPN gateway. This means that when it's connected to your router, you can send traffic to it from other devices before they connect to the outside world – essentially putting them behind a VPN. Here, we're using a first generation Pi – it's a nifty repurposing of a piece of kit that's past its sell-by date for most other common Pi-related projects.

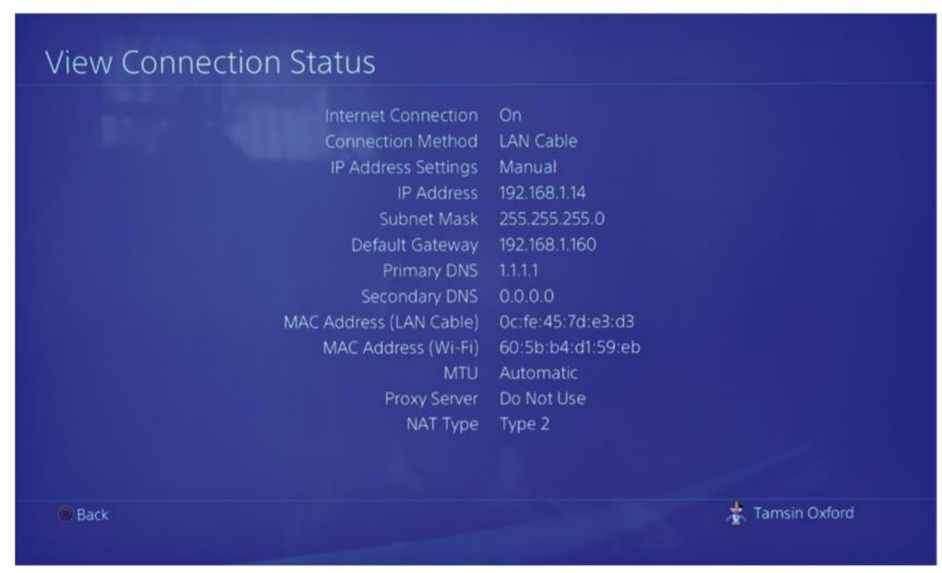
To get started, you'll need four things. A Raspberry Pi, an SD card preloaded with the Raspbian operating system and a subscription to a VPN service of your choice. We're going to use Nord VPN, which assures us that it doesn't log user behaviour or filter for particular activities such as P2P protocols. There are services that promise even more anonymity, or are more affordable, but Nord is a good place to start investigating options.

The fourth thing you'll need is a copy of your provider's OpenVPN configuration files and encryption certificates. There are usually a lot of these – one for each server you can connect to – so pick a handful that you want to be able to quickly access. We opted for two UK and two US servers, choosing one that supports the UDP protocol and one that supports TCP/IP.

You should find these configuration files on your VPN provider's website. Download them and unzip them into a folder on your desktop.

QUICK TIP

Want the VPN to start whenever the Pi boots up? Use this command: `sudo systemctl enable openvpn@example.service`, **where** `example` **is the name of the .conf file you want to connect to (excluding the filetype).**



This is the network setup screen from your games console.

Install packages

We want our Pi to be running headless, in other words without a keyboard and monitor attached, which means that once its up and running we'll need to access it using a remote shell and SSH. For the first run it can be easier to access the Pi directly by plugging in peripherals, at least until you have made sure it has a fixed IP address on your network, which is essential for this walkthough.

To do that, open a terminal on the Pi desktop and type `sudo nano /etc/network/interfaces`. Edit this file to look like this (you can choose any free IP address for the line that ends in 12, bear in mind that the 1 in the third part of the address could be another number).

```
auto lo
iface lo inet loopback

auto eth0
allow-hotplug eth0
iface eth0 inet static
  address 192.168.1.12
  netmask 255.255.255.0
  gateway 192.168.1.1
  dns-nameservers 1.1.1.1 1.0.0.1
```

Press Ctrl+O to write out the file, and then press Ctrl+X to quit *nano*.

Now turn the Pi off, and remove all peripherals leaving just the networking cable in place. Then reconnect the power supply to get it started again.

Once the Pi has booted up, you should be able to access it from another PC using SSH. In our case the command to connect is `ssh pi@192.168.112` and the password is the default, **raspberry**. You'll probably want to change the password.

The next thing you'll need to do is install the OpenVPN packages with the following command: `sudo install openvpn.`

Next, navigate to the folder where you need to keep your OpenVPN configuration files by typing `cd /etc/openvpn`. You need to download the configuration files from your VPN provider, in our case we can do this using *wget* – don't forget to use *sudo* because the **etc/** folder isn't writeable by ordinary users. In our case, the command is `sudo wget https://downloads.nordcdn.com/configs/archives/servers/ovpn.zip`, followed by `sudo unzip ovpn.zip` to decompress it.

A quick `ls` command will show if you have been successful. There should be a list of files ending in .ovpn. Note that some VPN providers may have packaged these files with subdirectories, as an example for connections encrypted with optional 128- or 256-bit protection. You'll need to move the files to the **etc/openvpn** directory using the `mv` command.

You can now open a connection to any of these servers using the command `sudo openvpn example.ovpn –daemon`, where `example` is the filename of the configuration file.

If you try this now, you'll notice that the script asks for your username and password to authenticate the connection. Do test to see if the connection is working by typing `ifconfig`. You should see a connection marked TUN, which is your VPN tunnel.

You shall not password!

So far we have a slightly cumbersome way of connecting our Pi to a VPN via a terminal that requires you to enter your username and password when you want to connect. Good, but it could be better – we're going to create a few scripts to automatically create your credentials.

Take a look at the VPN files you downloaded to your desktop and open one of them. You should see that it starts the client and that there's a list of commands. These include a line that contains `auth-user-pass`. We can alter this line to automatically feed a username and password to our config file when it is called.

Back in your SSH connection to the Pi, navigate back to **/etc/openvpn** and type `sudo nano vpnlogin`. This should open up the *nano* text editor. Create a text document that has nothing except your username for the VPN provider on the first line, and your password on the second. Hit Ctrl+O to write the contents to disk, then Ctrl+X to quit *nano*.

Once you've done that, pick the VPN connection you think you'll use the most and edit the **config** file using `sudo nano example.ovpn`. Change the line that says `auth-user-pass` to `auth-user-pass vpnlogin`.

Now when you start that connection using the `openvpn` command, it should connect directly without the password prompt. (See boxout to change all the config files at once.)

```
studiopc@studiopc:~$ ssh pi@192.168.1.160
pi@192.168.1.160's password:

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue Aug  7 16:32:43 2018 from 192.168.1.59
pi@raspberrypi:~ $ nano uk.sh
pi@raspberrypi:~ $ sudo ./uk.sh
openvpn: no process found
pi@raspberrypi:~ $ Tue Aug  7 16:34:32 2018 OpenVPN 2.3.4 arm-unknown-linux-gnueabih
Tue Aug  7 16:34:32 2018 library versions: OpenSSL 1.0.1t  3 May 2016, LZO 2.08
Tue Aug  7 16:34:32 2018 WARNING: file '/etc/openvpn/vpncred.txt' is group or others
Tue Aug  7 16:34:32 2018 WARNING: --ping should normally be used with --ping-restart
Tue Aug  7 16:34:32 2018 Control Channel Authentication: tls-auth using INLINE stati
Tue Aug  7 16:34:32 2018 Outgoing Control Channel Authentication: Using 512 bit mess
Tue Aug  7 16:34:32 2018 Incoming Control Channel Authentication: Using 512 bit mess
Tue Aug  7 16:34:32 2018 Socket Buffers: R=[163840->131072] S=[163840->131072]
Tue Aug  7 16:34:32 2018 UDPv4 link local: [undef]
Tue Aug  7 16:34:32 2018 UDPv4 link remote: [AF_INET]5.254.112.27:1194
Tue Aug  7 16:34:32 2018 TLS: Initial packet from [AF_INET]5.254.112.27:1194, sid=d3
Tue Aug  7 16:34:32 2018 WARNING: this configuration may cache passwords in memory -
Tue Aug  7 16:34:32 2018 VERIFY OK: depth=2, C=PA, O=NordVPN, CN=NordVPN Root CA
Tue Aug  7 16:34:32 2018 VERIFY OK: depth=1, C=PA, O=NordVPN, CN=NordVPN CA2
Tue Aug  7 16:34:32 2018 Validating certificate key usage
Tue Aug  7 16:34:32 2018 ++ Certificate has key usage  00a0, expects 00a0
```

Your Raspberry Pi can easily handle connecting to a vpn and marshalling your home network's traffic through it.

So far so good, but we don't want to have to type a long command every time we need to connect to a VPN. Remember that you took a note of your most likely used servers right back at the start? Good, because we're going to create a quick script that will enable you to start and switch between those with a simple command.

It begins with an instruction to close any open VPN connections, then starts the OpenVPN daemon filling in the credentials from the text file we just created.

Back in your home folder, type `nano vpn1.sh`:

```
#!/bin/bash
sudo killall openvpn
sudo -b openvpn /etc/openvpn/example.ovpn
```

Repeat this step for the three or four VPN connections that you think you'll use the most, adding one to the number in the filename. Now you can start or switch your connection using by SSHing into the Pi from any computer on the network and typing `sudo ./vpn1.sh`.

Finally, to route traffic via the Pi, you'll need to go back to your PlayStation (or other device) and change the internet settings. Leave everything in its default setting apart from the Gateway and DNS servers.

Change Gateway to the IP address of your Pi, and set the DNS server to 1.1.1.1, 8.1.1.8.

And that's it. Now you can make your PS4 appear in another country without leaving your lounge.

» ONE BIG EDIT FOR MAN

Remember that line in the VPN configuration file for `auth-user-pass`? We need to change that so that credentials file we just created is called automatically. This means you need to change that line to `auth-user-pass vpnlogin`.

You can do that manually for the three or four servers that you think you'll use the most, and then just set-up scripts to start those servers when you need them. But what if you want to pretend you're in Sweden for a day, or South Africa? Can you even remember what the VPN password was?

Nord VPN supplies over two thousand different config files, one for each server with both TCP and UDP protocols. To edit all those files at once open up an SSH connection to your Pi and navigate to **etc/openvpn** again. Now run the following command to use SED for a batch edit of all the .ovpn files.

```
find . -type f -name \*.ovpn -exec sed -i.bak 's|auth-user-pass|auth-user-pass vpnlogin|g' {} +
```

This will also create a backup of the existing files.

Monitor your network using a Raspberry Pi

Christian Cawley scans his network for intruders and unsafe hardware using little more than a Raspberry Pi and Nagios.

OUR EXPERT

Christian Cawley has more Pis than you've had hot dinners, probably.

Network security is vital to the integrity of your data and hardware. Using a router with a firewall and computer security tools is wise, but to get a better idea of what's really happening on your network, you'll need a monitoring tool.

It might seem like the stuff of low-budget cyber thrillers, but a network monitoring tool can be effortlessly set up and put to work pinging the hardware on your network and scanning for intruders. This essentially means that you can learn right away when your devices on your network go offline (perhaps IoT or smart-home applications), and discover unauthorised connections to your router or other networked hardware. All you need is a Raspberry Pi and the *Nagios* software.

Two versions of *Nagios* are available. There's the *Nagios XI* series of premium products, which might be more suited to corporate use. These feature a trial period and technical support. *Nagios* also comes as an open source suite, with four options: *Nagios Core*, *Nagios Core Plugins*, *Nagios Core Frontends* (community-generated UIs) and *Nagios Core Addons* to extend functionality. Once installed on a single device, *Nagios* brings continuous monitoring to your network, regardless of its size and dynamic.

Continuous monitoring is a staggeringly useful concept that all too few businesses take the time to implement. It basically enables the detection of network and server issues almost as they happen. This prompt detection enables fast diagnosis, thus minimising downtime.

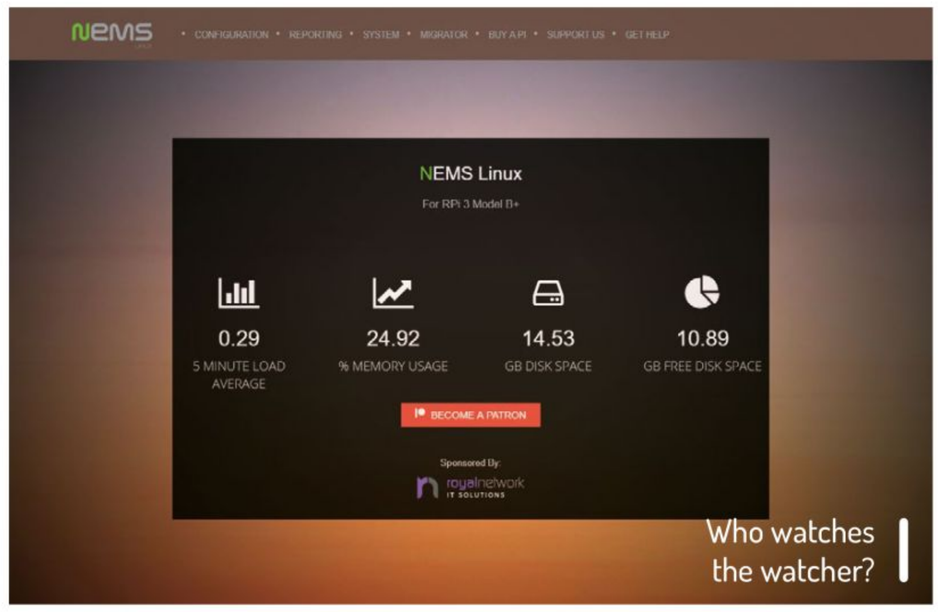
With the open source version of *Nagios*, it's possible for anyone to apply this level of monitoring to a network. It doesn't matter whether you're running a home network, a social club, or you're the system admin for a healthcare provider. Install *Nagios* on a Raspberry Pi, configure monitoring and notifications, and network management is immediately levelled up.

QUICK TIP

Nagios can be manually installed on most versions, including the Pi Zero. Alternatively try this build for the Raspberry Pi 2: http://bit.ly/lxf253nagiospi.

Start being a nagios

As noted, various versions of *Nagios* are available; this is true for the Pi versions too, where different builds have been released over the years. For simplicity, we're going to look at two options: installing a dedicated disk image, and installing manually to your existing OS.



Who watches the watcher?

NEMS for the Raspberry Pi stands for *Nagios* Enterprise Monitoring Server, and is available from **http://bit.ly/lxf253nems** [1]. It requires an SD card of at least 16GB, with 32GB recommended. A Raspberry Pi 3B+ is recommended, although almost any version apart from the Raspberry Pi 1 Model A and Raspberry Pi Compute Modules should work. Note that the NEMS build can only be downloaded using a Bittorrent client. Other *Nagios* disk images are available, however.

Once downloaded, use your preferred disk imaging tool to write the IMG file to the SD card. For simplicity, the open source cross-platform *Etcher* tool from Balena (**www.balena.io/etcher**) is the best option if you're using a desktop environment.

All you need to do is download and install the tool, insert the SD card in your computer, select the disk image, confirm the correct drive is selected, and click Flash. Note that this will reformat and overwrite the SD card. A few moments later, NEMS will ready to run on your Raspberry Pi.

Next, connect your Raspberry Pi to your router via Ethernet. While NEMS can use Wi-Fi, Ethernet is a more reliable medium for network monitoring. Insert the SD card, then boot up the Pi and wait while NEMS configures. This will involve the filesystem automatically resizing, so the initial boot will be longer than usual.

Using SSH, connect to NEMS using `nemsadmin` as the username and password. Then run

```
sudo nems-init
```

This will start the NEMS setup. Here, you'll need to set the correct locale, encoding options, and create an

1) https://nemslinux.com/download/nagios-for-raspberry-pi-4.php

account. This includes adding an email address to which notifications will be sent. In your desktop browser, open **https://nems.local** (or use the Pi's IP address) to start configuration.

Nagios it yourself

Alternatively, you can install *Nagios* manually to your Raspberry Pi's existing operating system, Raspbian or otherwise. For the best results, start with a fresh OS install, then open a terminal or start an SSH session. First update and upgrade:

```
sudo apt update && sudo apt upgrade
```

Be sure to follow this with a reboot.

```
sudo reboot
```

With the Pi restarted, install *Nagios*:

```
sudo apt install nagios3
```

Wait for prompts to create an admin account, keeping note of the password as you'll need this later. The installation will continue for a short while after this. When done, you'll be able to access *Nagios* from another device using the Pi's IP address, such as **http://192.168.1.10/nagios3**. Use the `nagiosadmin` username and the password you created earlier when you're prompted.

Get monitoring

With *Nagios* installed on your Raspberry Pi, you have a continuous monitoring system ready to run. All that's left to do is to configure it for your purposes. If you've set up manually, you can configure a host via the terminal. Start by creating a configuration file; ours is called **monitor.cfg**.

```
sudo nano /etc/nagios3/conf.d/monitor.cfg
```

In this file, add the details that relate to the device you wish to monitor. For example, I have a Raspberry Pi hosting a *Minecraft* server:

```
define host {
  use        generic-host
  host_name    minecraft
  alias       minecraft
  address      192.168.1.22
}
```

The `generic-host` used is a template found at **/etc/nagios3/conf.d/generic-host_nagios2.cfg**. Templates can be used to save time when creating device-monitoring settings. To save time, all you should need to do is create multiple entries based on this definition, changing the `host_name`, `alias` and `address` to the device you're monitoring.

Hit Ctrl+X to save and exit, then reload *Nagios*:

```
sudo service nagios3 reload
```

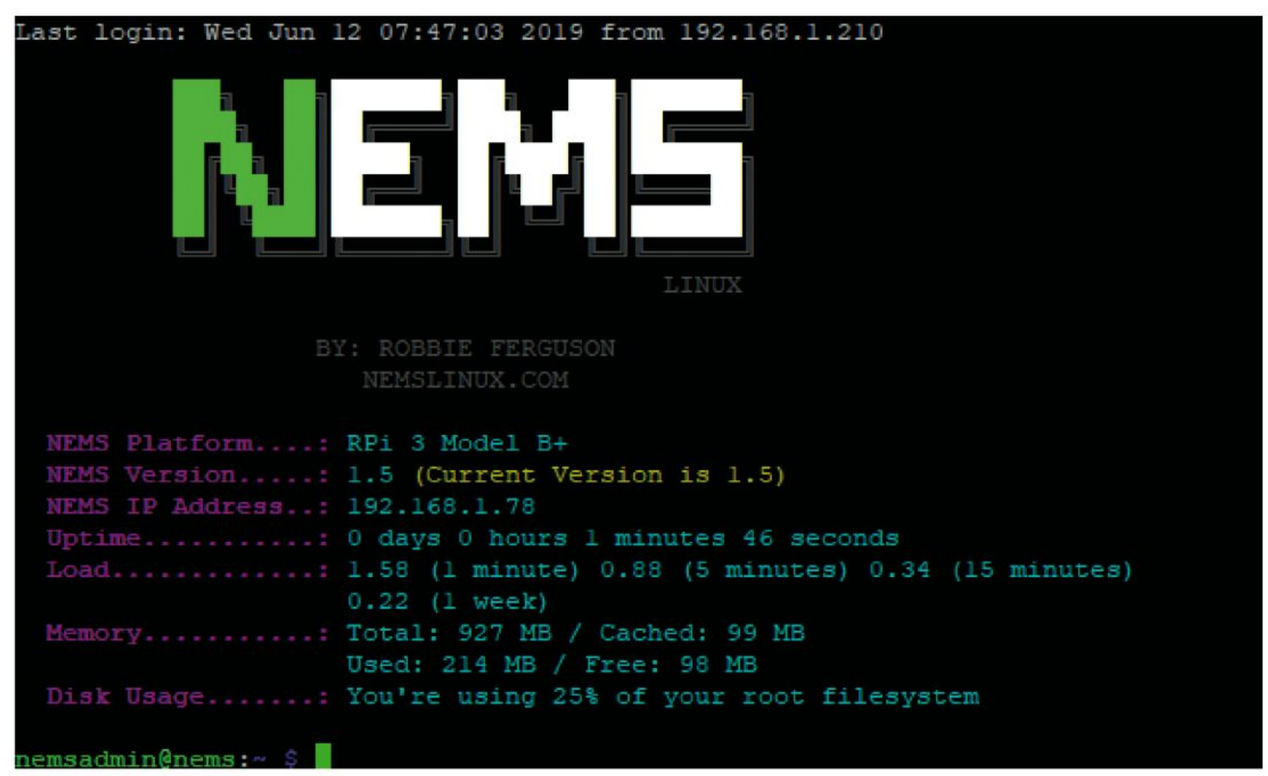
If you run into any problems, check for errors with:

```
sudo /usr/sbin/nagios3 -v /etc/nagios3/nagios.cfg
```

This will ensure your config files are correct. Remember to reload *Nagios* again after checking.

Monitoring can be observed via the browser; indeed, you might consider leaving this open. Or you could simply run it on your Pi's desktop, ready for remote observation via VNC.

Nagios configuration can also be conducted using your web browser. In the browser, open the Configuration menu and find NEMS Configurator. Find Hosts in the left-hand menu and click Add. Next, add the host details with the same credentials as you would manually. Add any necessary notifications, then Submit to save.



The NEMS terminal screen. Pretty, isn't it.

Use the Reporting menu to keep an eye on performance. Two key views are available. Adagios is a modernised user interface, while *Nagios Core* relies on a legacy view that dates back a few years. Both are entirely usable, however.

The full power of *Nagios* is considerable, so take the time to explore the features, add your networked devices, check logs, switch visualisations and more. If you're using NEMS, you can easily configure a host via the browser interface in the same way.

That's all there is to it – your Raspberry Pi *Nagios* box is now set up for monitoring. It can be left running in the background indefinitely, ready to alert you to any problems with hardware connectivity and suspicious activity on your network.

» WHAT IF YOUR SERVER LOCKS UP?

As you might know, the Raspberry Pi is prone to the occasional lock-up. When the system freezes, monitoring will cease. Short of ensuring a constant power supply and using a top-rated SD card, there is little you can do to stop this. You might set a *cron* job to reboot Nagios, or you could rely on a HAT designed to control whether your Pi is powered up, rebooted or shut down.

Compatible with most Pi models (including the Pi Zero) and with support in NEMS, PiWatcher is described as "the best watchdog for your Raspberry Pi" and sits neatly on the first six pins of the Pi's GPIO. Used in place of the board's built-in USB power port, PiWatcher is a programmable watcher circuit that can be configured via the I2C. Where it comes in useful for a locked-up computer is that it can be programmed to reboot the Pi should the computer become unresponsive. This is based on a user-specified time limit, but there's also a built-in switch for manual shutdown (and subsequent reboot) if necessary. As the switch is programmable and the PiWatcher can be easily reconfigured, it has uses beyond basic monitoring.

Note that you'll need to solder the included 2x3 female pin header to the PiWatcher before use. Head to the Omzlo online store to check the documentation and order one for 4.95 euros: **http://bit.ly/lxf253piwatcher**.

Raspberry Pi PoE HAT

Les Pounder has the power! And now it comes over Ethernet. Can he ditch the micro USB power supply and finally live with just an Ethernet cable?

IN BRIEF

A HAT compliant board that delivers power over Ethernet for Pi 3 B+ users. Fitting to the four new PoE pins on the Pi 3 B+ and covering every GPIO pin, the PoE HAT communicates with the Pi via a special configuration overlay, which is configured by the user via a config script. GPIO access is possible but requires additional header pin extensions.

Power over Ethernet (PoE) is nothing new. In fact, it's been with us for many years. But it has eluded the Raspberry Pi... until now.

The PoE HAT is an official Raspberry Pi product and is designed to work only with the new Pi 3 B+, via the PoE pins present on that board. To use the PoE HAT, we need to attach the HAT to all 40 pins of the GPIO, and the four pins of the PoE header. We then supply power using the Ethernet connection, rather than using a micro USB power source. This necessitates using either a PoE-compliant switch/router or a PoE injector that will send 48V of power and data over a single Ethernet cable. Be careful using this with your network, because non-PoE compliant devices may be damaged.

Supplying power via this board requires us to have the latest version of Raspbian installed. But once all that's done, the board just works – and all we need is one cable!

The PoE HAT is dominated by a 25mm 5V fan that's designed to cool the CPU on the Raspberry Pi. When the fan is at full speed it does produce an audible whine. This is slightly annoying, but can be tolerated. However, there's a problem with the PoE HAT, and it's quite a big one.

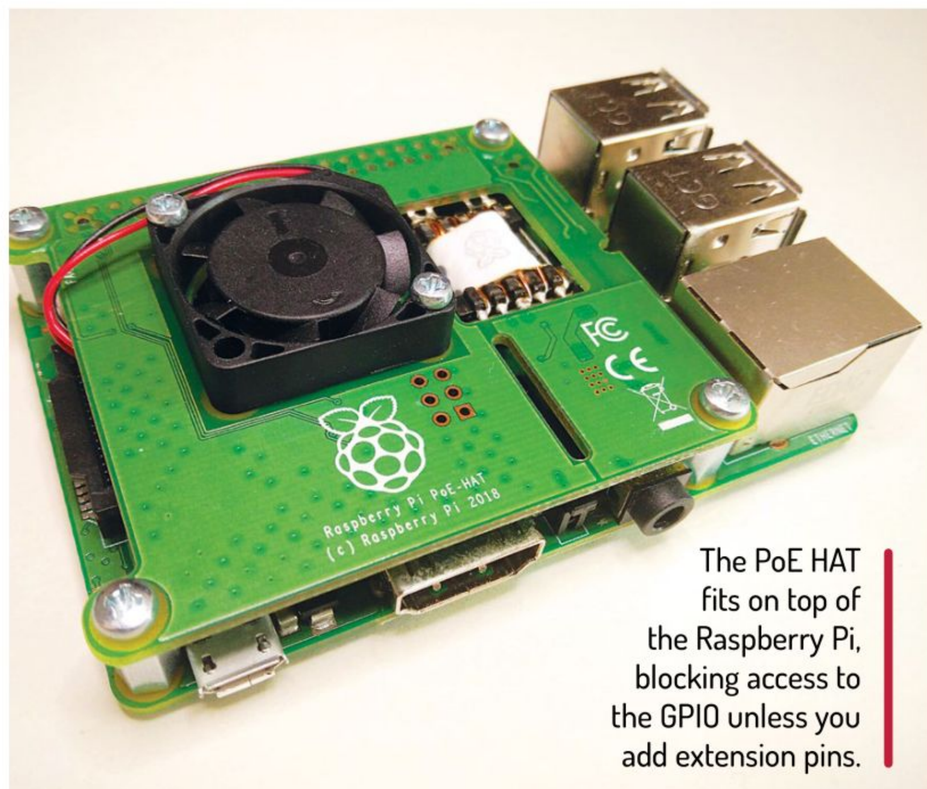
It's a cover-up

Because the board covers all of the GPIO, we can't attach any add-on boards or make direct connections to the GPIO. There's a workaround that involves a 40-pin header extension and a four-pin extension for the PoE header, but the necessary components aren't included. This is a shame, because makers will be using the PoE HAT for powering outdoor projects, typically involving sensors. Adding HATs on top of the PoE HAT is possible, but you'll also need brass spacers to stop the PoE HAT from making contact with the add-on board. Worse still, the clearance between the fan and the add-on board is mere millimetres and generally just heats up the board.

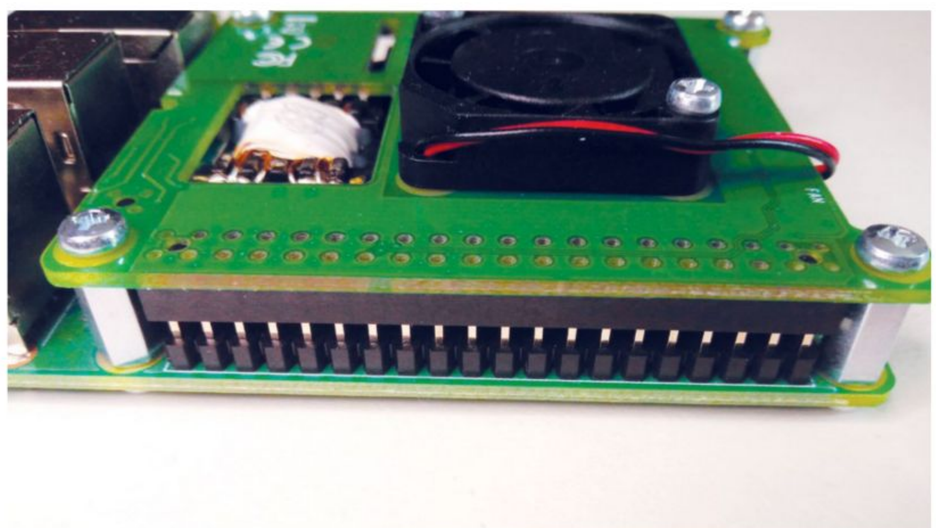
Fan control is automatic, triggered by a temperature threshold set using a config file that communicate with the onboard Atmel chip using I2C. At 45 degrees C the fan will kick in and cool the system. The higher the temperature, the faster the fan will spin.

There's a quirk with the PoE HAT, specifically the USB ports. Connecting keyboards and mice is fine, but any device that draws more than 40mA may trigger the system to shut down power to the USB ports until the current draw is removed. This issue has been addressed by Eben Upton, who said that the cause was due to "an interaction between the fairly low-frequency switching regulator on the HAT, and one of the two brands of USB current limiting switch that we use on the main board." Furthermore, testing of heavy current loads was undertaken with one brand of current-limiting switch, and that "dumb luck" saw the lower loads being tested with this troublesome component. So there's a chance that we can't plug in external USB devices requiring more than 40mA. That's unless we wish to start soldering components to the PoE HAT.

Is the PoE HAT any good? It is, if you need a single cable solution for your project. The problem is that the PoE HAT restricts access to the GPIO, unless we add the extra headers. The GPIO is what makes the Raspberry Pi such a versatile device. Without this functionality, it's really just a low-power Linux computer.



The PoE HAT fits on top of the Raspberry Pi, blocking access to the GPIO unless you add extension pins.



Ask yourself how badly you want to power your Pi with an Ethernet cable, before buying.

VERDICT

DEVELOPER: Raspberry Pi Trading
WEB: www.raspberrypi.org/products/poe-hat
PRICE: £18.35

FEATURES	7/10	EASE OF USE	8/10
PERFORMANCE	6/10	VALUE	6/10

The PoE HAT is easy to use and install. But the fact that it blocks access to the GPIO means that this board interferes with the biggest asset of the Pi. It's a device of two halves.

» **Rating 7/10**

Picade X HAT

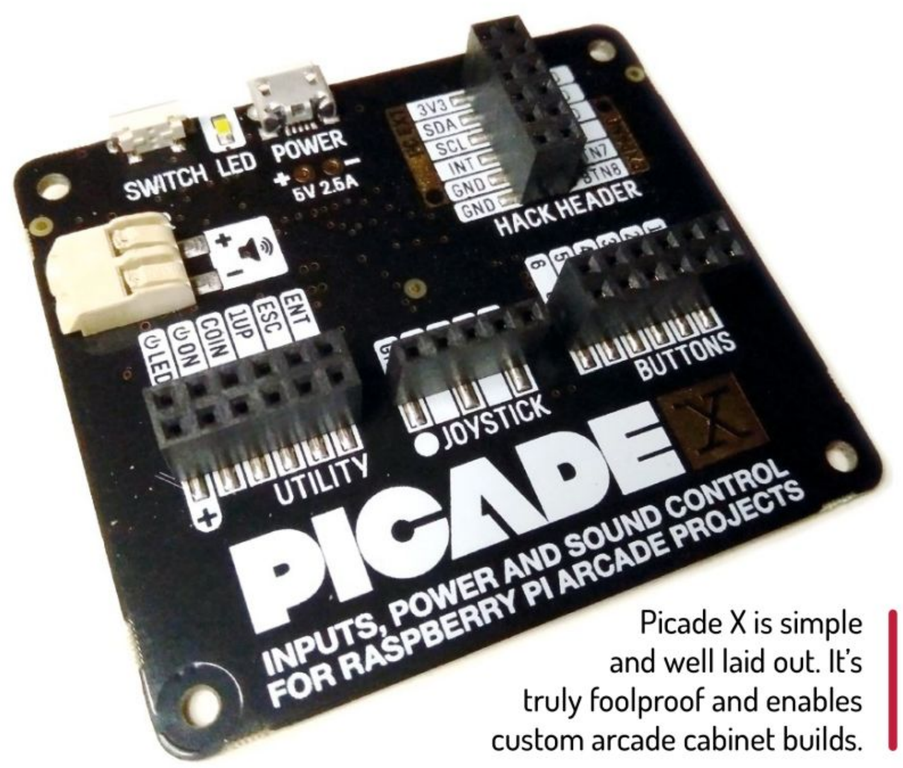
Like a fun-loving Terminator, **Les Pounder** has travelled back in time to the 1980s to play arcade games using his Raspberry Pi.

IN BRIEF

An all-in-one solution for arcade hobbyists using the Raspberry Pi as their development board. It's made up of an audio interface and an interface for your own arcade control inputs. Picade X is a simple way to start your adventures in arcade emulation. The board also offers a power switch to safely power up and down your cabinet.

Coming from a seaside town, the arcade was the pinnacle of entertainment in the 1980s. We cut our teeth playing *Afterburner*, *Space Harrier*, *Bombjack* and spent far too much money to prove our worth. Over the years arcades have seen a downturn with more people electing to play home consoles. But retro is now cool and the Raspberry Pi is now part of the "desktop arcade" experience. You can either buy a complete kit or make your own, and this is where Picade X HAT comes in.

Picade X HAT is an all-in-one control, power and audio solution for 40-pin GPIO models of Pi. Fitting it on top of the GPIO is simple, and the accompanying software install is easy to complete. Picade X HAT is compatible with Retropie which is the de facto standard for Raspberry Pi arcade, computer and console emulation. Aspiring hackers can also use Picade X HAT in their projects, especially games made using PyGame or other game dev libraries.



Picade X is simple and well laid out. It's truly foolproof and enables custom arcade cabinet builds.

Get connected

There are many connections around the board, starting with the audio out connection that offers a mono push fit connector with 3W of power. That's enough for the thumping bass of many classic scrolling beat 'em ups! We also see a micro USB port used for power. The Picade X HAT receives the power before the Pi and then feeds the power to the Pi via the GPIO.

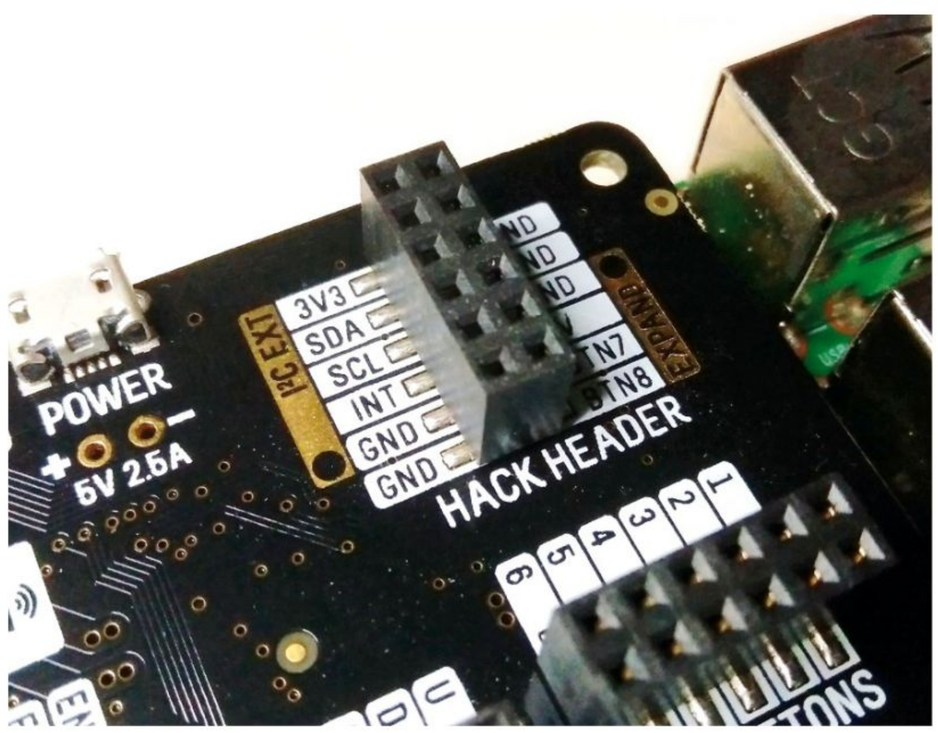
Control inputs are for a four-way joystick, and six player buttons – plenty for *Street Fighter 2* fans! There's also a selection of utility inputs for the simulated coin mechanism, player selection, menu options and power control. The software that this switch triggers can be used to power up the Pi and to soft shutdown the Pi, which preserves the filesystem. There's also a "Hack Header" which offers connections to some of the underlying GPIO pins, and to the I2C interface, so makers can use these pins to drive extra functionality.

Picade X HAT is all about choice. It enables you to source your own controls and inputs for your custom arcade experience. Sure, we can source an arcade to USB interface, use the onboard audio out and write our own software to control it all. But with Picade X HAT we're buying a product that has all the essentials in one board. We just need to bring a screen, controls, speaker and Pi.

Building a cabinet around this board is super simple and anyone can do it. Just place this board on your Pi, install the software and then plug in the buttons or controls where instructed and you too can be playing *Streets of Rage* like a pro. As for cabinets, a quick look on eBay will provide a plethora of kits ready to build.

Picade X HAT is for those of us that want to build their own arcade experience around the Raspberry Pi but have little electronics knowledge. It's easy to use and sensibly priced when we consider how much a USB arcade interface board can be. Typically i-PAC interface boards retail for $40 on their own! The Picade X HAT is best used with Pimoroni's own arcade cabinet, Picade, but you can build your own cabinet for not much money.

Picade X HAT is excellent value for money, and worthy of your time. Relive your youth with the board, and save those coins for penny shuffles and claw games!



The Hack Header feature on the Picade X HAT gives you extra connection options.

VERDICT

DEVELOPER: Pimoroni
WEB: https://shop.pimoroni.com/products/picade-x-hat
PRICE: £15

FEATURES	9/10	EASE OF USE	9/10
PERFORMANCE	10/10	VALUE	9/10

If you have ever wanted to build your own arcade cabinet, then the Picade X HAT is the perfect go-to solution.

» **Rating 9/10**

Using analogue HAT interfaces

Les Pounder shows us two more projects using the deceptively simple Explorer HAT Pro – and a little Python code.

OUR EXPERT

Les Pounder is a freelance maker who works with organisations such as the Raspberry Pi Foundation to promote maker skills.

Here, we'll learn how to use the Explorer HAT Pro to work with analogue electronics by creating two projects. First we create a simple light-detecting nightlight, then we create an electronic candle that we can blow out. But first we need to set up our equipment.

With the power turned off to your Raspberry Pi, place the Explorer HAT Pro so that it connects to all 40 GPIO pins, and so that it fits neatly above the Pi. Now connect your keyboard, mouse, HDMI and so on to your Pi and boot up. For the next step you will need to connect to the internet using Wi-Fi or Ethernet. Once connected, open a terminal, the icon for which is found in the top left of the screen. In the terminal type the following to automatically install and configure the Explorer HAT Pro software:

```
$ curl https://get.pimoroni.com/explorerhat | bash
```

During the install you will be asked to configure I2C and to install example code. Answer yes to these questions. Once installed, close the terminal window.

YOU NEED

- **Any 40-GPIO-pin Pi**
- **Latest Raspbian**
- **Explorer HAT Pro**
- **Electret microphone board**
- **3x male-to-male jumper wires**
- **Light dependent resistor (LDR)**
- **22k ohm resistor (red, red, orange, gold)**
- **Code: http://bit.ly/lxf252pihat**

Project 1: Nightlight

To introduce analogue electronics we shall use the most simple device, a light dependent resistor (LDR). This component changes resistance based on light levels. For this project we shall place the LDR on the breadboard and connect one leg to 3.3V, and the other to GND via the 22K ohm resistor. Then we connect Analog 1 from the board to the same line as our resistor and LDR. We do this as it creates a voltage divider that we use to measure the light level as a voltage. See the diagram in the download for an accurate reference.

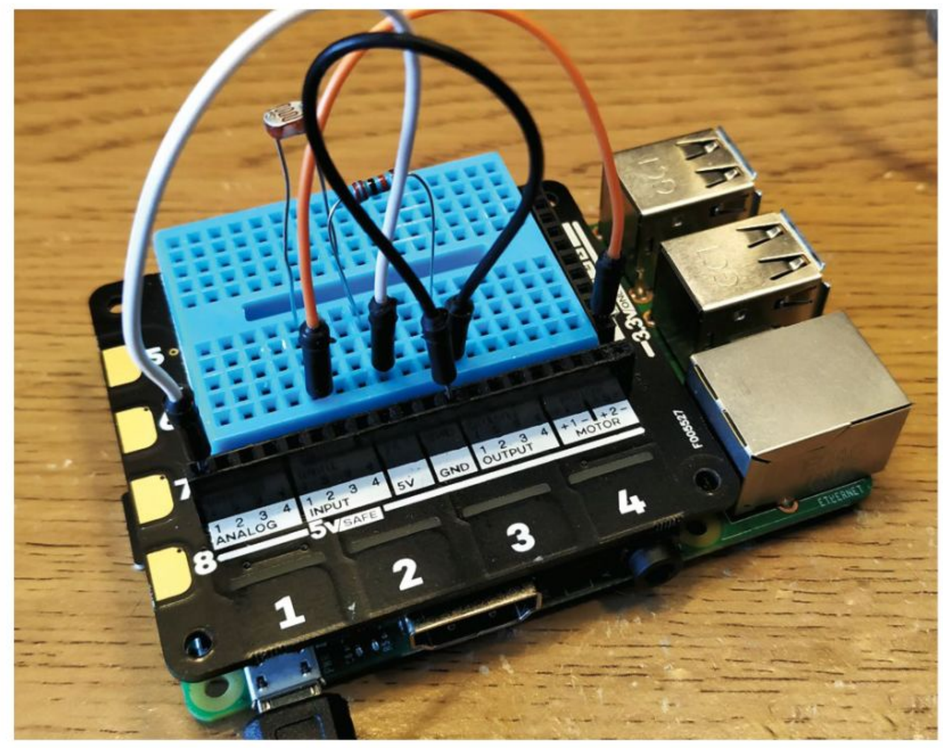
We now move on to the code. Using your favourite editor (*IDLE*, *Mu*, *Thonny*) create a new file and then save that new file as **nightlight.py**. Remember to save often! We start the code with two imported libraries, `time` and `explorerhat`, which we rename to `eh` for better convenience.

```
import time
import explorerhat as eh
```

Then we create a loop to constantly run our code.

```
while True:
```

Let's take a light level reading and store it inside a variable called `light`. This will show the voltage that our circuit is producing based on the resistance level of the LDR. This is printed to the Python shell for debug.



This simple circuit offers a great introduction to analogue electronics. It requires only a few cheap components and is accessible to all.

```
  light = eh.analog.one.read()
  print(light)
```

We then use a conditional test that checks to see if the voltage detected is less than 2V. If that is the case then it must be getting dark, so the four LEDs on the Explorer HAT Pro will turn on.

```
  if light < 2.0:
    eh.light.on()
```

But if it daylight then the voltage will be over 2V and so the `else` condition activates and turns off the Explorer HAT LEDs.

```
  else:
    eh.light.off()
```

Save the code and when ready, run it. Cover the LDR with your hand and the LEDs should light up; move it and the LEDs turn off. If you need to tweak the values, take a look at the Python shell to see what voltage changes there are when the sensor detects darkness.

Project 2: A digital candle

This project is inspired by the Nintendo DS, specifically a section in *The Legend of Zelda: Phantom Hourglass* where you are instructed to 'blow out' fire/torches in the game. To do this we blow on to the console and the game reacts. In reality we are blowing onto a microphone and an analogue signal is being measured

to act as a trigger. To replicate this we shall control the LEDs present on the Explorer HAT Pro using a microphone board. The microphone is connected to our Explorer HAT Pro as: GND > GND, OUT > Analog 1 and VCC > 5V.

We now move on to the code. Create a new file as before and then save that new file as **candle.py**. The first three lines of our Python code import prewritten libraries of code. The first is called `time` and we use that to control the speed of our project. Next we import the `explorerhat` library, but rename it to `eh` for ease of use. Lastly we import the `random` library which will be used for our candle flicker effect.

```
import time
import explorerhat as eh
import random
```

To run our code we need a loop and for that we shall use `while True`, which will run our code until we stop it.

```
while True:
```

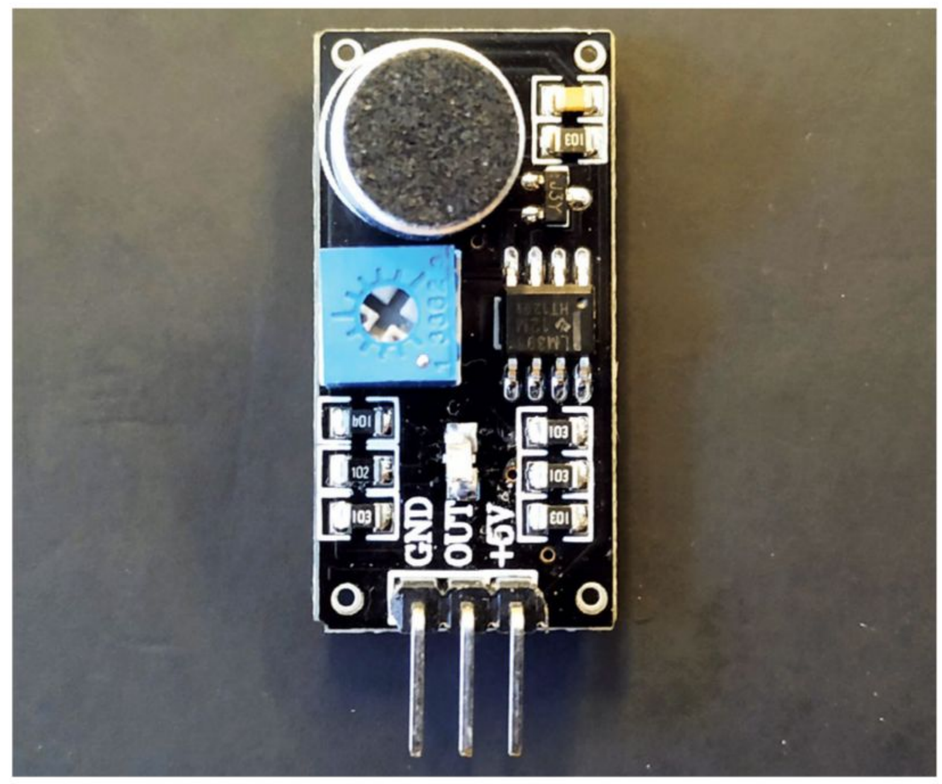
The following code is indented to show that it runs inside of the loop. We create four variables, called `on`, `off`, `fade_in` and `fade_out`. These variables will be later used to control the flicker effect of the LEDs. But how do we create that? Well, we need to introduce a little randomisation. Using the `random.uniform` function we can generate a random float (a number with a decimal point) between 0.0 and 1.0, and then store it in the variable:

```
    on = random.uniform(0.0,1.0)
    off = random.uniform(0.0,1.0)
    fade_in = random.uniform(0.0,1.0)
    fade_out = random.uniform(0.0,1.0)
```

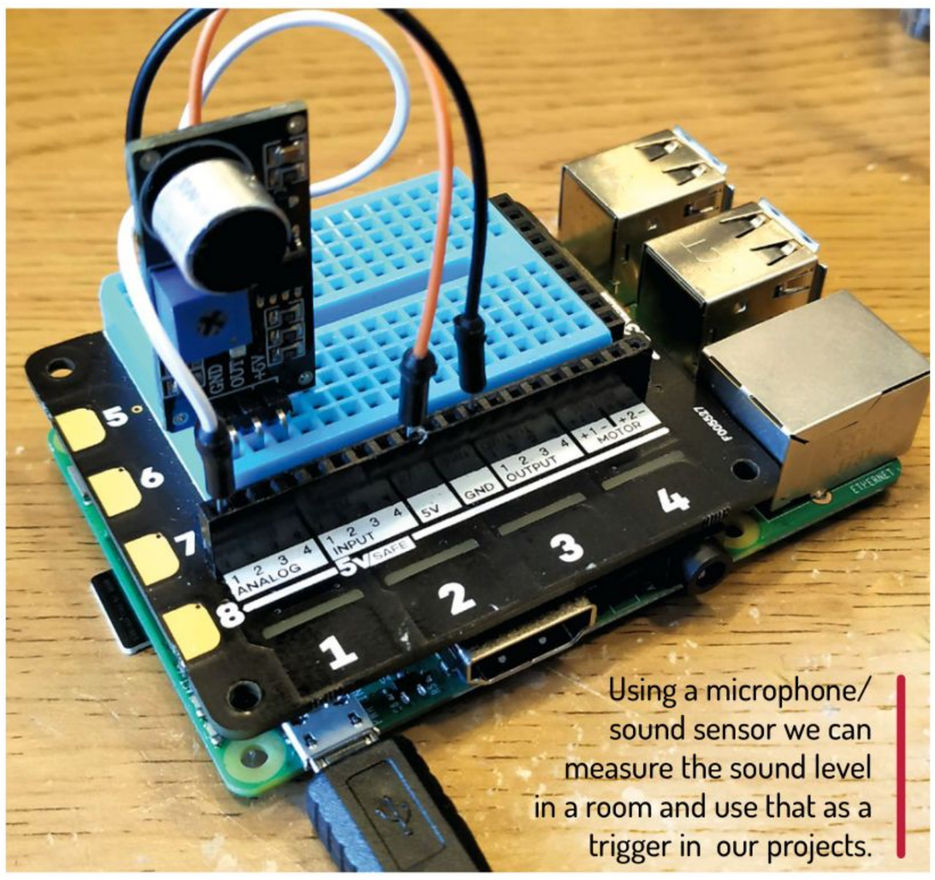
To detect that someone has 'blown out' the candle, we need to read the sensor connected to Analog 1. Using the Explorer HAT library we read the value of the sensor and store it as a variable called `blow`, which is then printed to the Python shell for debug with a small delay for legibility.

```
    blow = eh.analog.one.read()
    print(blow)
    time.sleep(0.1)
```

If we run the code as is, it will print the sensor reading, which in our case was 5.00. This is the voltage



The microphone/sound sensor has a blue trim pot that can be used to alter the output voltage. This is how we can calibrate the sensor for the environment.



Using a microphone/sound sensor we can measure the sound level in a room and use that as a trigger in our projects.

from the sensor. Using the trim pot (blue dial) on the microphone, we can tune the sensor; we chose 1.5 as the trigger point. So if the voltage goes over the trigger point it means that a loud sound (blowing out the candle) has been detected.

```
  if blow > 1.5:
```

Now we need to turn the Explorer HAT lights off. Using the Explorer HAT library we turn off all the lights and then wait for two seconds before the loop repeats.

```
    eh.light.off()
    time.sleep(2)
```

But if we haven't blown out the candle, the `else` condition activates and it uses the `pulse` function to make the lights flicker using the `on`, `off`, `fade_in` and `fade_out` values.

```
  else:
    eh.light.pulse(on, off, fade_in, fade_out)
```

Save the code and then run it. Now try to blow out the candle. If it works, great; if not, adjust the values of your code and the trim pot on the microphone.

So there we have it, two simple projects from one great board.

QUICK TIP

Need to quickly check the sensor data? Open the Python shell and type 'import explorerhat as eh' then 'eh.analog.one.read()'. This is really useful when calibrating the sensor before general use.

» EXPLORER HAT PRO

We have long been fans of this board because it packs so much into a tiny package. In this tutorial we used its analogue inputs to read voltage levels. But we also have four protected inputs, rated up to 5V, and four outputs rated for 5V. There are two motor connections for use with DC motors, so we can easily build robots.

We have four touch buttons that can be used to trigger projects to life. We also have four capacitive touch connections for use with crocodile clips, enabling us to make connections and circuits that react to human skin. There also four LEDs that we used in these projects – these can be on or off, or we can fade their light levels to create subtle glows.

On the right side of the board are connections for I2C and SPI (3.3V only) which means we can connect other sensors and components using their protocols. Lastly, we have a breadboard built into the board and ready for use to build our projects upon it. We get all of this versatility for around £20 – a bargain when you consider how much we can learn from it.

Controlling the Pi from a web-based app

Les Pounder shows us how to create a web app using pure Python to control the Raspberry Pi GPIO.

OUR EXPERT

Les Pounder is a freelance maker who works with organisations such as the Raspberry Pi Foundation to promote maker skills.

We're going to create a web interface using Anvil that uses just Python. This interface will enable us to control real components: a red and blue LED along with a buzzer, connected to a Raspberry Pi.

The hardware build for this project is just two LEDs, one connected to GPIO17 and the other to GPIO27. We used a 330 Ohm resistor to connect each LED to the GND pins of a Raspberry Pi 4. The buzzer is connected to GPIO22 and GND, and requires no resistor. Please see the diagram in the download for this project for detailed information.

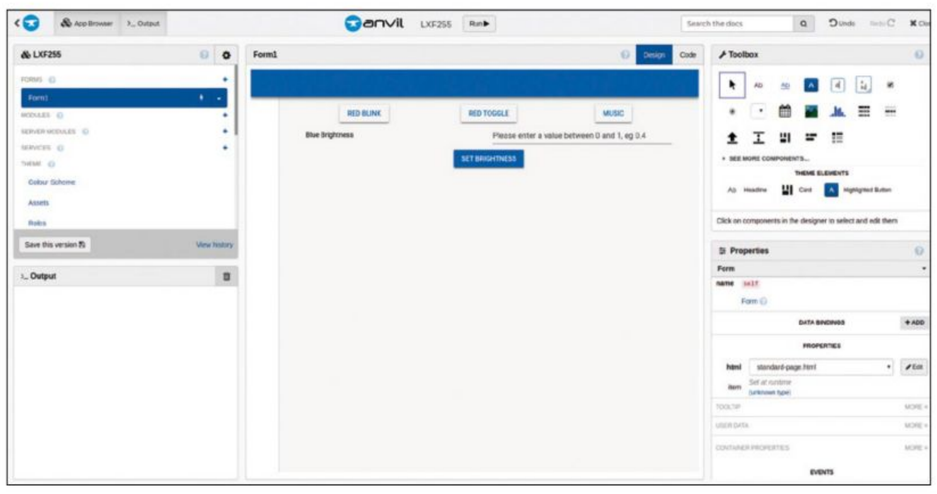
YOU NEED

- **Any model of Raspberry Pi**
- **A desktop/ laptop**
- **Internet connection**
- **Breadboard**
- **2x 330 Ohm resistors (Orange, Orange, Brown, Gold)**
- **2x LEDs**
- **Buzzer**
- **6x female-to-male jumper wires**
- **Code: https://github.com/lesp/LXF255/archive/master.zip**

Writing the code

On a PC visit **https://anvil.works** and create a free account by clicking Start Building. Follow the sign-up instructions, then click Create App,appear then select the theme Material Design. In Anvil the layout is as follows. On the left is an App Browser used for configuration. In the centre is where our app – called a form in Anvil – is designed using elements from the top right of the screen, called the Toolbox. You will see 'Material Design' as the name of the app in the top left of the screen; change this to 'GPIO Control' by clicking the text.

Next we need to add three buttons to the form. The Button tool can be found in the toolbox, just hover the mouse over the icons to identify them. Drag each button over to the form and place them next to each other. For each button change the 'text' field in the properties menu (bottom right of screen) to the following values: RED BLINK, RED TOGGLE, MUSIC. The next step is to place a Label from the toolbox under



The Anvil editor is laid out with an App Browser, Form and Toolbox, which hide a wealth of features and functionality.

Anvil apps are made using a design tool, but accessing the code underneath is just a double-click away.

the buttons, and then change the label text to "Blue Brightness". To go with this label we need a TextBox; drag it so that it's horizontal next to the label and change the placeholder text to "Please enter a value between 0 and 1, eg 0.4". Now add another button, this time placed underneath the previous components. Change the text to "SET BRIGHTNESS".

To enable this app to communicate with our Raspberry Pi we need an API key. In the top left of the screen is a cog settings menu. Click it and select Uplink. There will be a button to get your server API key. Click the button and copy/save/write down this key.

With that complete we can now write some code for the Anvil web app. Double-click the RED BLINK button. This will automatically take us to the appropriate section of code, which is a function. Replace `pass` with the following code, make sure that it is indented to match how `pass` was formatted.

```
anvil.server.call("red_blink")
```

Make a note of `red_blink` as we shall be reusing that on the Pi. To go back to the form, click Design. Now change the code for the two other buttons in the same manner as before.

RED TOGGLE:

```
anvil.server.call("red_toggle")
```

MUSIC:

```
anvil.server.call("music")
```

The final button is SET BRIGHTNESS and for this we need to add a different line of code. Double-click the button to edit.

```
anvil.server.call("blue_brightness",float(self.text_box_1.text))
```

This will call a function on our Raspberry Pi called `blue_brightness` and it will pass a value stored in the TextBox that will control the LED on our Pi. That's all of the code for Anvil, but before we move on to the

Raspberry Pi, press RUN on Anvil to start the app. Nothing will happen until we write the code for the Pi.

Power up the Pi and boot to the desktop. Before we can write any code for the Pi we need to open a terminal and install the `anvil-uplink` Python library.

```
$ sudo pip3 install anvil-uplink
```

To write the Python code open your favourite Python editor and create a new file called **blink.py**. We start by importing the libraries for our code.

```
from gpiozero import LED, PWMLED, TonalBuzzer
from gpiozero.tones import Tone
from signal import pause
from time import sleep
import anvil.server
```

We import the LED, PWMLED and TonalBuzzer classes from GPIO Zero, enabling us to control LEDs and the buzzer. Then we import Tone to control the pitch of the buzzer. `pause` stops our code from exiting, and `sleep` is used to control the pace of the code. Lastly we import the Anvil uplink server. To connect this Pi to the Anvil app, you will need your API key:

```
anvil.server.connect("YOUR API KEY")
```

In order for the LEDs and buzzer to be used we need to tell Python to which GPIO pins they are connected to.

```
red = LED(17)
blue = PWMLED(27)
b = TonalBuzzer(22)
```

The buttons in our Anvil app make calls to functions that we will write in the code, but in order for that to work we need to use a decorator to instruct anvil-uplink. The first function is to toggle the red LED on and off via the RED TOGGLE button in Anvil.

```
@anvil.server.callable
def red_toggle():
  red.toggle()
```

Another function to flash the LED every half a second when the RED BLINK button is pressed:

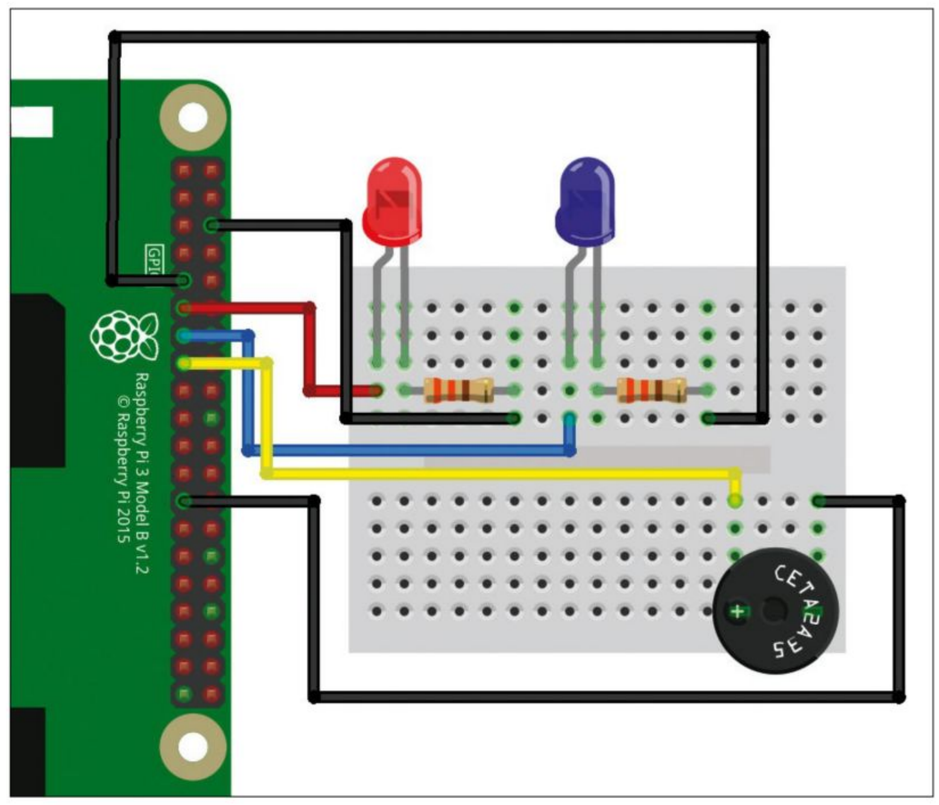
```
@anvil.server.callable
def red_blink():
  red.blink(0.5,0.5)
```

The next function takes the `float` value (between 0 and 1.0) stored in the text box and uses it to control the brightness of the blue LED.

```
@anvil.server.callable
def blue_brightness(brightness):
  blue.value = brightness
```

The last function will play music via the buzzer – the first few notes of *Twinkle, Twinkle, Little Star*. We use the `Tone()` function in GPIO Zero to play specific notes, with pauses between each note to control the tempo. Also note that there has to be a stop at the end, or the note will never stop playing!

```
@anvil.server.callable
def music():
  b.play(Tone("C4"))
  sleep(0.2)
  b.play(Tone("C4"))
  sleep(0.2)
  b.play(Tone("G4"))
  sleep(0.2)
  b.play(Tone("G4"))
  sleep(0.8)
  b.play(Tone("A4"))
  sleep(0.2)
  b.play(Tone("A4"))
  sleep(0.2)
  b.play(Tone("G4"))
  sleep(0.2)
  b.stop()
```



The circuit for this project is kept simple so that we can focus on learning the Anvil application.

The last two lines of code ensure that the project keeps running and waiting even when a button has not been pressed.

```
pause()
anvil.server.wait_forever()
```

With the code completed on the Pi, save and run. You will see the code connect to the Anvil service and this is where we need to switch back to our computer running the Anvil app. Press the buttons to toggle the LED on/off, or to make it flash. To use the blue LED, first type in a value between 0 and 1.0 and then press SET BRIGHTNESS to turn it on. Lastly there is the MUSIC button, which will play *Twinkle, Twinkle, Little Star* using the buzzer.

QUICK TIP

Have you noticed that the function names match what we call from Anvil? The names have to match in order for Anvil to successfully run the function. In the Anvil app they are wrapped in quotation marks and do not have a parenthesis. But the names have to match so "TOGGLE_RED" in Anvil is TOGGLE_RED() in the Python code on your Pi.

» WHAT ELSE CAN ANVIL BE USED FOR?

Anvil is much more than just a tool to control a Raspberry Pi. Anvil has many more features, such as data tables to record data, email, user authentication, Google and Facebook APIs, and Stripe to take payments. What does this all mean? Well, using Anvil we can build an app to enable payments to be made, recorded into a data table and then visualised using the built-in graphing library, plotly.

Anvil apps can be shared using a private URL for internal use, or via a public short URL that can be shared on social media. Anvil can also work with devices that deal with web data, for example ESP8266 and ESP32 boards. These boards are cheap and using low power, they can gather data in the field and then send that data to an HTTP endpoint, for example an Anvil data table.

The free Anvil plan has plenty of features to power most projects and it would be well worth investigating further. We quickly developed a robot control app for the 4Tronix Pi2Go Mk2 robot reviewed in this issue (see page 53).

Anvil is flexible enough to be used simply, but it can also form the heart of more complex projects. So give it a try today!

Get inspiration for your projects with Google

Les Pounder shows you how to use Python to search Google to gain inspiration for your next project.

Les Pounder is a freelance maker who works with organisations such as the Raspberry Pi Foundation.

YOU NEED

- **Any Pi**
- **Raspbian**
- **Code: https://github.com/lesp/LXF249-InsPIration/archive/master.zip**

In each issue of Linux Format, we write a new Raspberry Pi project, and in every issue we try to be inventive and unique. But there are times when inspiration is not with us, and in those moments we turn to searching Google with key phrases or words, and often they are quite ridiculous. These 'mashups' are there to fire the imagination; just as a child given construction blocks can make anything they dream of, we can use these silly searches to think up new and unique ideas.

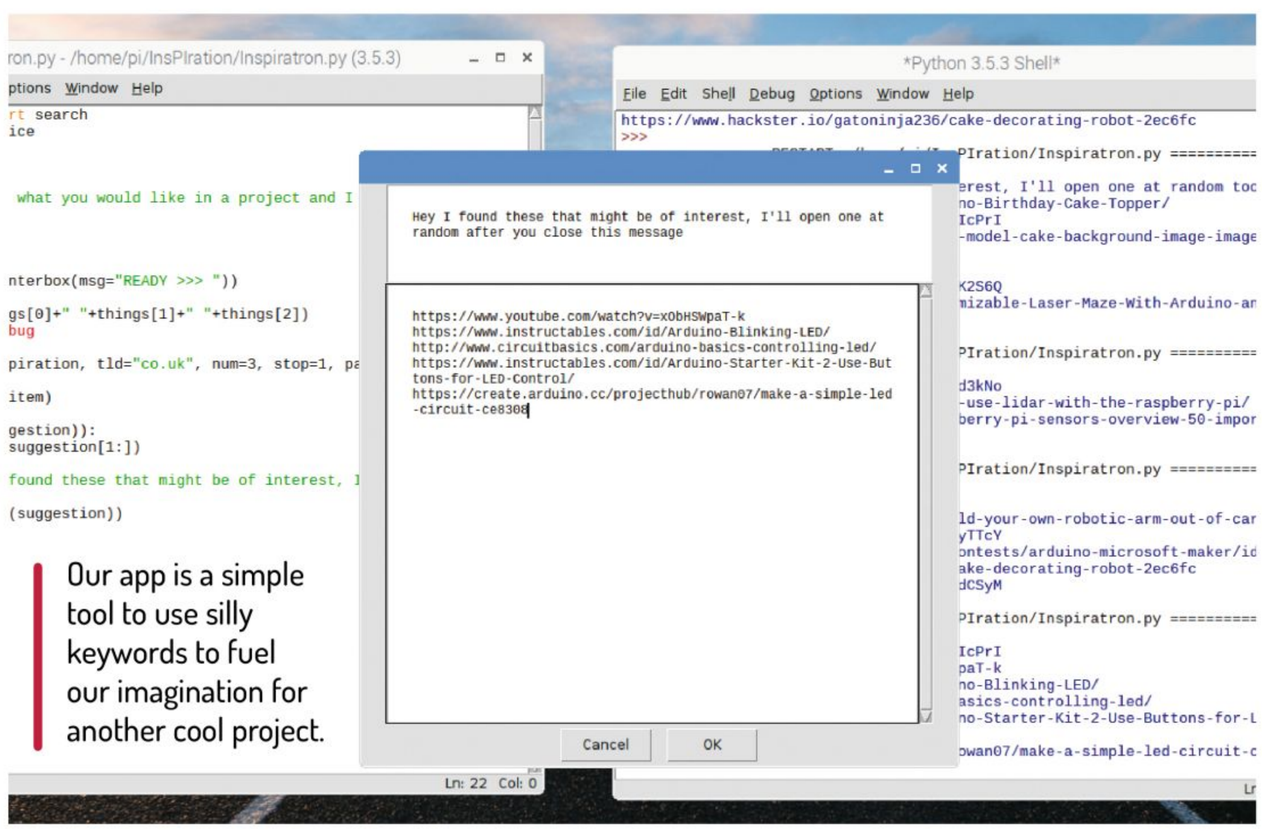
In reality this project is really just three `for` loops and two lists that store and iterate over user input while it searches Google. Then it opens a random page in our web browser.

Let's get mashing!

Before we can write any Python code we first need to install two extra libraries of code. These are called packages or modules, and they contain pre-written code that we can reference in our project. The first library will install a way in which we can search Google from Python. The second library is an easy to use graphical user interface toolkit. In a terminal enter the following command.

```
$ sudo pip3 install google easygui
```

This will take a few moments. Once it's complete, close the terminal window. To write the code we need to use a Python editor – in this project we'll use the Python 3 editor found in the Programming section of the Raspbian menu. When the editor opens, click File > New to create a new blank document. Now immediately save the file using File > Save, call it **inspiration.py** and remember to save often.



Our app is a simple tool to use silly keywords to fuel our imagination for another cool project.

Easy now GUI

We start our project by importing a series of libraries/modules. Firstly we import the `search` function from the *googlesearch* library, then we import the `choice` function from the *random* library. We next import the *webbrowser* library to enable us to open a browser window. Finally we import the *easygui* library, but we shorten the library name to 'eg', as 'easygui' can be tricky to type at speed.

```
from googlesearch import search
from random import choice
import webbrowser
import easygui as eg
```

The first step in this project is to ask the user a message, and to do this we use *easygui*'s `msgbox` (message box) function. Here we can ask a message (`msg`), give the dialogue box a title and change the text in the `ok_button` to 'Inspire Me!'

```
eg.msgbox(msg="Tell me what you would like in a
project and I shall search Google for you. This will
trigger 3 pop up windows", title="InPIration", ok_
button="Inspire Me!")
```

Next we need to create two blank lists. One will contain the `things` that we want to search for, the other will store the suggestions that our Google search finds. Lists in Python are called arrays in other languages. They can contain lots of pieces of data, for example strings, integers, floats and variables, but not exclusively these. We can call data from a list by using its index number. A list starts the index from zero and can go on and on.

```
things = []
suggestion = []
```

In the `msgbox` we instruct the user that we are going to ask them for three things to search for – these are the items that they wish to use as inspiration. To do this we need to use a `for` loop with a range. In this case it will create a variable called `i` and each time the loop goes round it will iterate (add) one to the variable until it

reaches three, then it will stop. Each time the loop goes round it will use an `enterbox` from the *easygui* library, which *is* a dialogue box into which we can type, to capture the user input. This is then appended to our `things` list, which means it is added to the list.

```
for i in range(3):
    things.append(eg.enterbox(msg="READY >>> "))
```

We now have the items to search for, but we need to convert them from list into a string – this is for our Google search later. To do this we create a variable called `inspiration` and in it we store the converted items from the `things` list, starting with 0, then 1 and 2. This is all wrapped inside a string-formatting function that ensures the text is saved correctly. We then print this to the Python shell for debug.

```
inspiration = str(things[0]+" "+things[1]+" "+things[2])
print(inspiration) #Debug
```

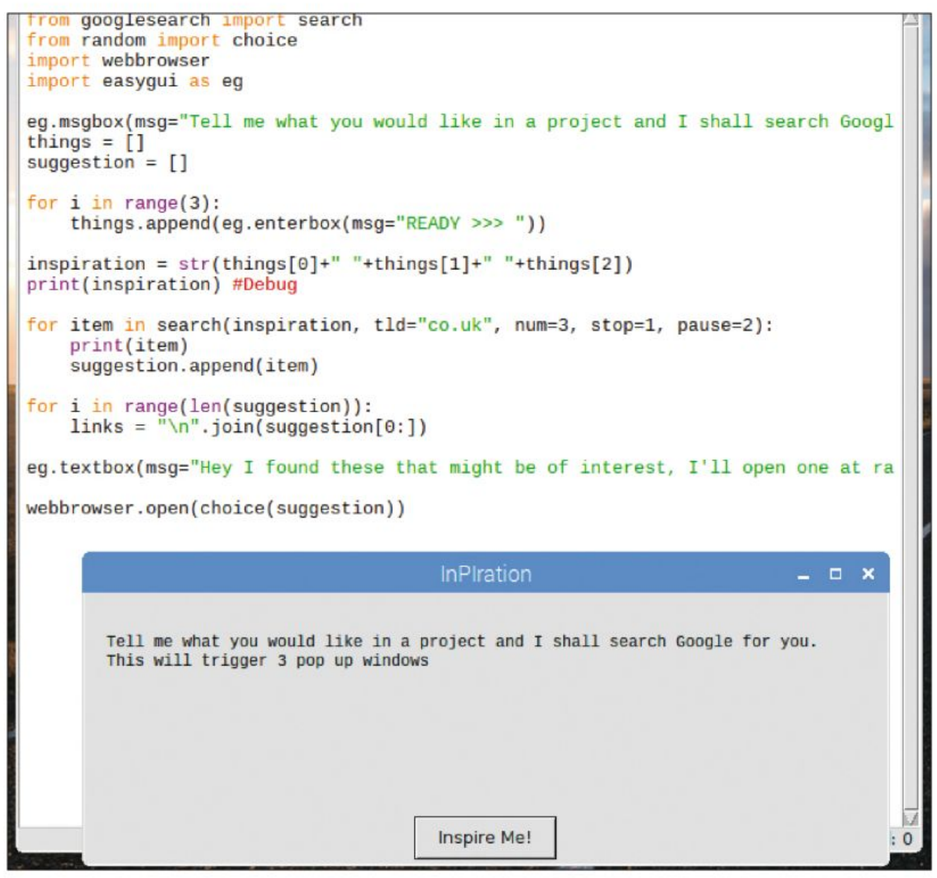
Round and round we go

Time for another `for` loop, and this time we use it to perform a Google search. For this we use the `inspiration` variable, since it contains the search terms. We then instruct the search to use the Google TLD (top level domain) for our country, in our case `co.uk`.

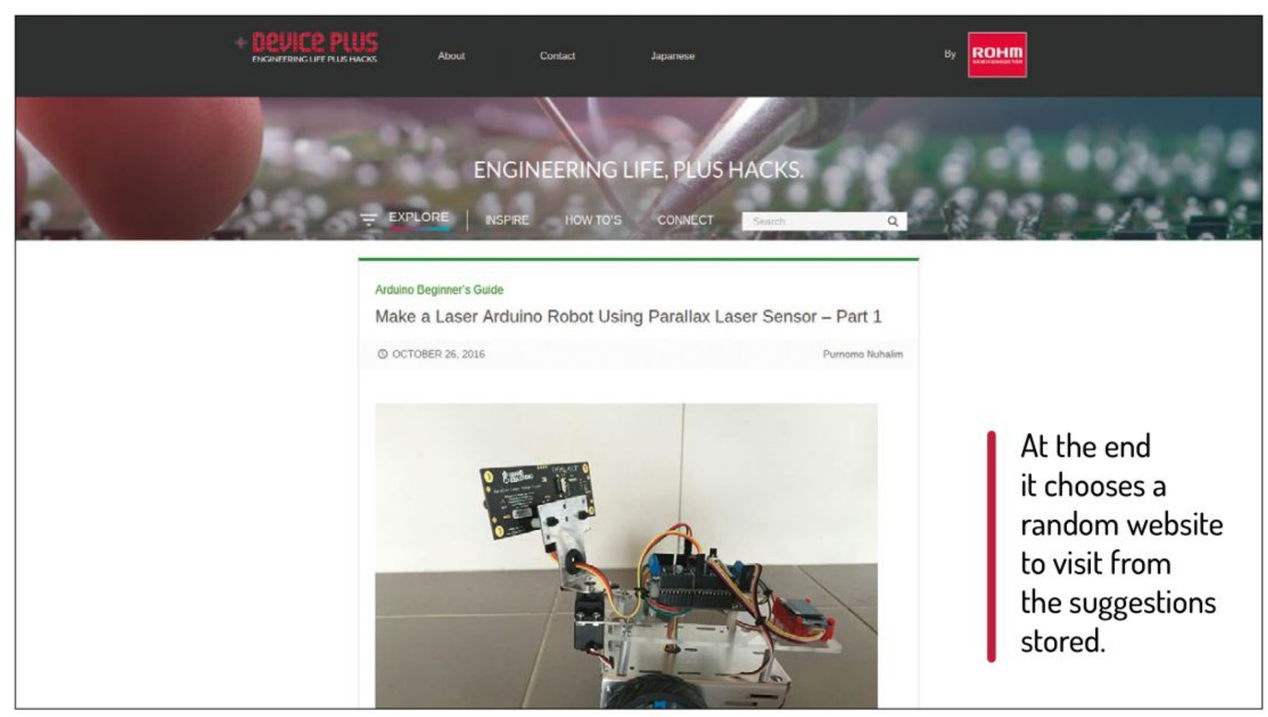
The `num` value limits the number of links returned. The `stop` is a Boolean value tells our search to stop when it has enough links. Lastly, the `pause` value is set to two seconds between each search, otherwise Google may block our computer from using its service. Each time the `for` loop goes round, it prints the returned search item (the URL) and also appends the search item to the `suggestion` list we created earlier.

```
for item in search(inspiration, tld="co.uk", num=3,
stop=1, pause=2):
    print(item)
    suggestion.append(item)
```

Moving along, we have another `for` loop and this time we use it to convert our `suggestion` list into a multi-line string, a paragraph. We do this for the length



When the project starts, it instructs the user on how it will work, using a msgbox.



At the end it chooses a random website to visit from the suggestions stored.

(`len`) of the list. Creating a variable called `links` to store the multi-line string, we instruct Python to insert a new line `\n` and then join the text together, starting from index 0 in our suggestion list. This is called slicing, and by passing the value `0:` we instruct the slice to start at 0.

```
for i in range(len(suggestion)):
    links = "\n".join(suggestion[0:])
```

Now that we have the multi-line string we can use it in *easygui*'s `textbox` function. This function takes a message (`msg`) and the text that we wish to show, in this case the contents of the `links` variable.

```
eg.textbox(msg="Hey I found these that might be of
interest, I'll open one at random after you close this
message", text=links)
```

The final line of code opens the default web browser for Raspbian, but the link it opens will be a random choice from the suggestion list we created earlier.

```
webbrowser.open(choice(suggestion))
```

With the code completed, click Run > Run Module to start. You will now have no excuses when asked to devise a new project. Have fun!

QUICK TIP

When using the easygui library, start typing the name of a function, for example 'eg. enterbox(' and a helpful tool tip will appear to show the syntax for that function. This is common for the majority of Python libraries and is very helpful.

» EASYGUI VS GUIZERO

If you have been hacking around with the Pi for a while, you will know of *guizero*, written by Laura Sach who works for the Raspberry Pi Foundation (**https://lawsie.github.io/guizero**). It's a great library which simplifies the way in which we can create *Tkinter*-based layouts for many types of GUI application. We used it in **LXF242** to create our barcode scanner/Amazon book search app. So you might be thinking, why did we choose *easygui* over *guizero* for this project?

Well, it comes down to our needs. We just needed a few dialogue boxes to capture user input, and display the output from the Google search. We can do this with *guizero*, but it would not be three simple lines – there would be a few more lines, but still with simple code.

We have used *easygui* for a few years, and yes, it is rather old and a trifle too simple for most projects. But for a really simple 'Ask a question, return an answer' project, it cannot be beaten. To learn more about *easygui* and how you can use it via a series of simple tutorials, head over to its Read The Docs page and follow their work: **https://easygui.readthedocs.io/en/latest.**

Moving servos and flashing LEDs

Sean D. Conway provides the theory before using Python with a Pi to control a servomotor, and to change LED brightness.

Sean D. Conway
A retired IT Security Specialist who invests time in Raspberry Pi projects in order to satisfy his passion for continuous learning.

After completing this tutorial, you'll have all the knowledge to understand how to code Pulse Width Modulation (PWM) support for a Raspberry Pi using Python. This knowledge will enable you to control servomotors as well as control the level of brightness of an LED, both of which are crucial for many a project.

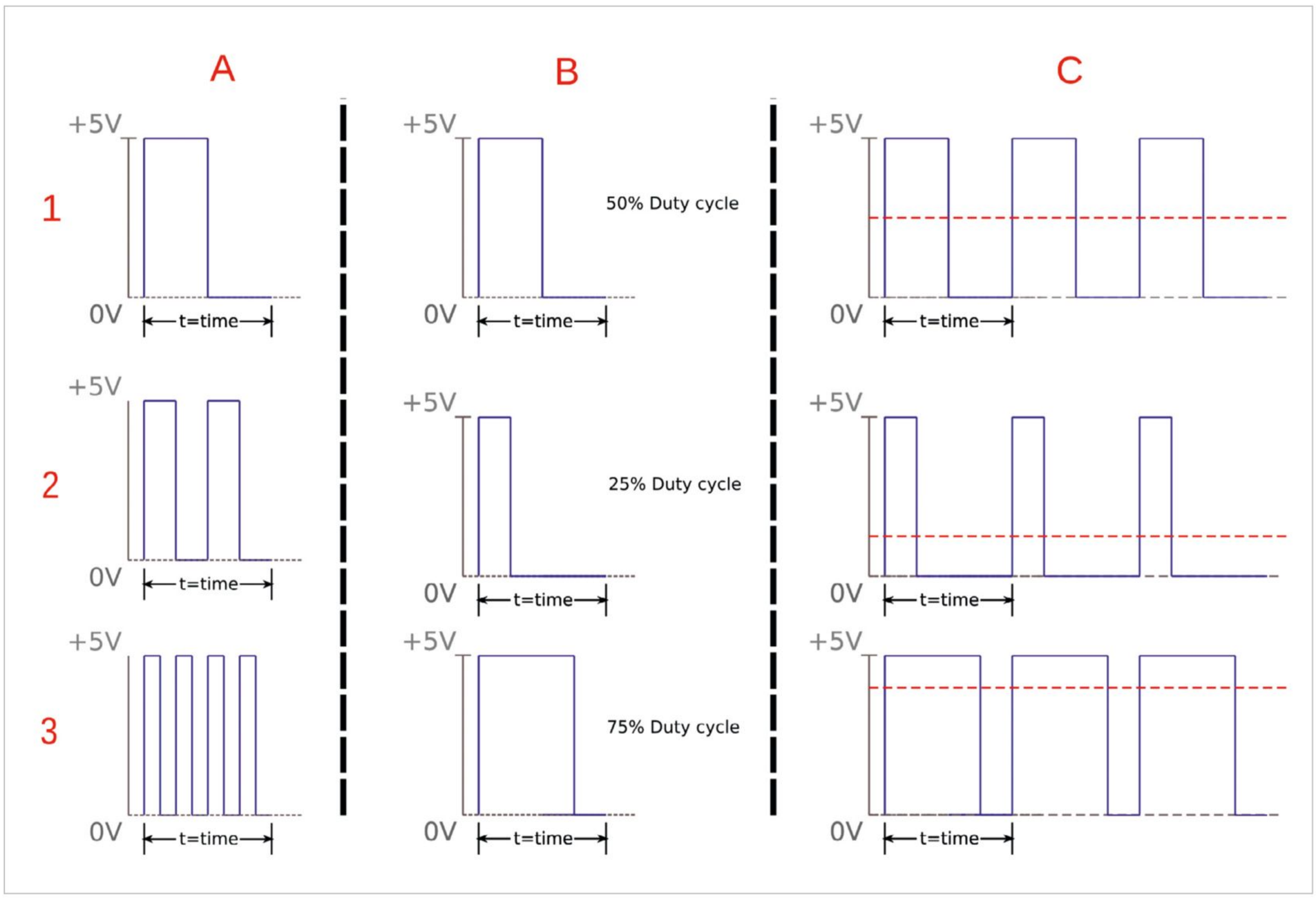
PWM is a digital signal modulated by varying the width of the pulsing signal. First let's explore some terms and concepts to set the groundwork for the tutorial, and we'll include some diagrams to provide visual examples.

In theory...

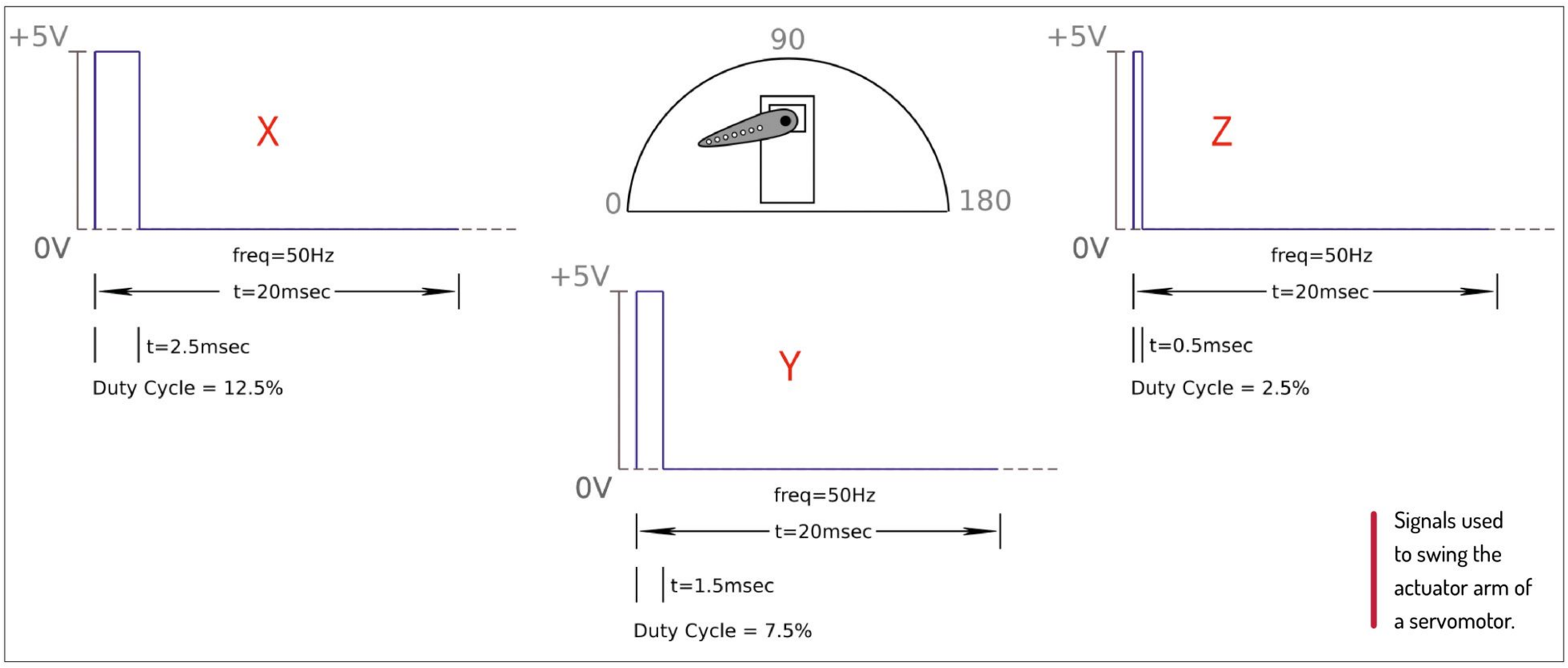
A cycle is an interval of time during which a complete excursion of a value sequence is completed. In the waveform A1 shown below, the value of the digital signal in a time interval starts at 0V, rises to 5V for a period, then returns to 0V to complete the cycle. This complete excursion is one cycle of the digital signal. In waveform A2, in the same time period two excursions of the digital signal have occurred. This would be two cycles. Similarly, waveform A3 shows four cycles of the digital signal occurring.

The time it takes for the digital signal value to complete one cycle is called the period of the digital signal. The number of cycles completed in one second is called the frequency. Until 1965, frequency was indicated in cycles per second. Today, the more common technical term for the unit of measurement for frequency is Hertz, with the symbol Hz.

You may have encountered Hertz in describing mains electrical power, or more likely the speed of a computer's CPU (when measured in GHz). In North America, AC (mains) power has a frequency of 60Hz,



Sample waveforms as mentioned in the main text.



Signals used to swing the actuator arm of a servomotor.

while in the UK and Europe it's 50Hz (*Jonni was explaining there are minor variances across Europe due to a dispute in Serbia that make your microwave clock lose time. FACT!–Ed*). The frequency value of a digital signal is a required parameter when coding a PWM setup in Python for a Raspberry Pi.

The term 'duty cycle' refers to a comparison of the time interval at which the digital signal is high (5V, in this example) versus low (0V). If the digital pulse is high for 50 per cent of the time interval, as in waveform B1, the digital signal has a duty cycle of 50 per cent. In waveform B2, the digital signal is high 25 per cent of the time: a 25 per cent duty cycle. Waveform B3 establishes a digital signal that is high for three quarters of the time interval, so has a duty cycle of 75 per cent. By varying the duty cycle of the digital signal, the pulse width changes. Changing the width of the pulse is referred to as a Pulse Width Modulated digital signal. This can be a method of of controlling an analogue device. Note the word analogue there – digital devices react differently to PWM signals.

PWM is one of the primary means of driving devices such as variable-speed motors, dimmable lights (that is, an analogue light), actuators and speakers. PWM signals apply power in pulses of voltage. The voltage is applied and then removed many times in an interval, as shown in waveform C1. An analogue device driven by PWM reacts to the average value of the pulses, shown by the red lines in the waveforms in column C. The motor doesn't stop instantly when the pulse drops to zero; inertia causes the motor to slow a bit before power is reapplied by the pulse returning high. The average voltage changes as the pulse widths are varied.

As with the frequency value, the width of a pulse stated as a duty cycle is a required parameter when coding a PWM setup.

Servomotors to maximum!

Using this knowledge, let's examine the operation of a digital motor called a servomotor. The movement of a servomotor (determined by its actuator) can be rotary, moving in a circle, or linear, moving in and out. The servomotor uses a feedback sensor coupled to a controller to establish position. Having precise control of the angular or linear position, velocity and acceleration is what makes servomotors unique.

A servomotor can move an arm extension to a specific distance, or rotate a shaft to a specific angle. The movement is a function of the width of the pulse signal applied. By continuously applying a signal of a specific pulse width, the positional feedback to the controller of the servomotor determines the motion – in-out, left-right – and the final position.

Fortunately, we don't have to get too far into the operation of servomotor controllers in order to develop knowledge about how they work. The motor graphic in the centre of the diagram above depicts a SG92R digital servomotor. Its physical orientation is related to angular motion, and this is important when you are trying to determine left and right directions. Facing the servomotor in the opposite direction will change what is 'left' and 'right' when it comes to determining angles.

In order for the servomotor actuator arm to take up the position on the left, at zero degrees (see waveform X), the width of the pulse will need to be 2.5ms, or 12.5 per cent duty cycle at a frequency of 50 times per second – that is, 50Hz. To move the servomotor

QUICK TIP

You can find the Python scripts referenced in this article on the LXFDVD, or head to the LXF archives.

» WHAT'S A SERVOMOTOR, ANYWAY?

The SG92R is an inexpensive, compact, angular rotation servomotor with plastic gears, suitable for use in control applications such as radio-controlled aircraft. It requires a digital pulse signal frequency of 50Hz. The angular control motion and position of the actuator of the servomotor is a function of the pulse width signal applied. By varying the duty cycle (pulse width) of the signal applied, the motion and final position are defined.

The SG92R is a great addition to the Pi.

QUICK TIP

Read more analogue tips for the digital environment at http://bit.ly/lxf250pwm.

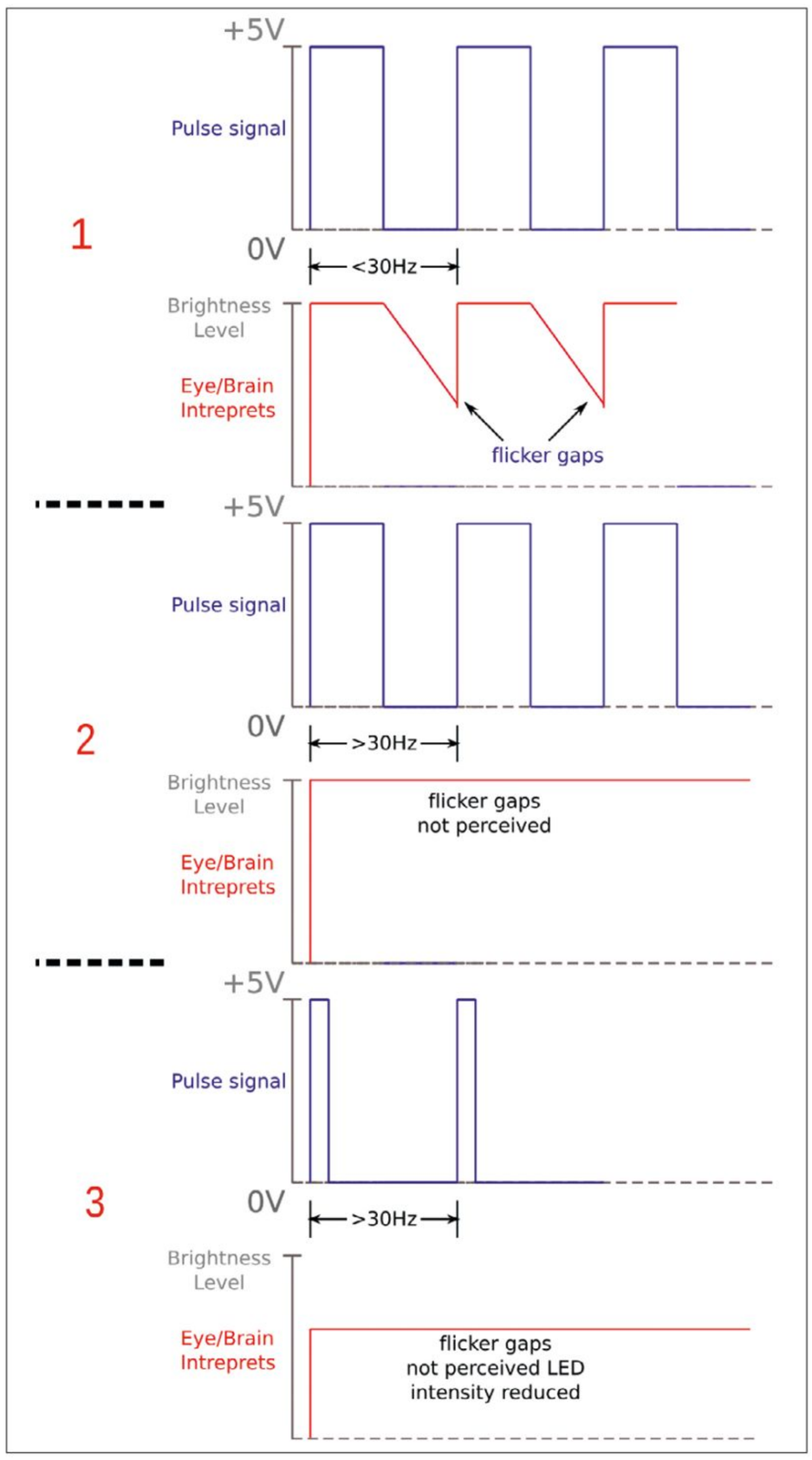
actuator arm all the way to the right at 180 degrees (see waveform Z), the same 50Hz digital signal must have a pulse width of 0.5ms, or a duty cycle of 2.5 per cent. When a pulse signal is applied the actuator moves; when no pulse signal is applied, it's stationary.

The electronic controller inside the servomotor evaluates the width of the incoming pulse and positions the actuator arm to reflect the corresponding pulse width. The controller has the ability to position the arm precisely along the arc, according to the incoming signal. If the actuator was linear rather than angular, the linear movement would also be precise.

LEDs to maximum!

Now let's return to what we've learned so far and apply it to lighting a light emitting diode (LED). Remember that we said that when PWM is applied to an analogue device such as a light, it causes the intensity to change, because the average voltage level varies depending on the duty cycle (that is, the pulse width). Column C in the diagram on page 56 shows some examples.

When applied to a digital device like a LED, PWM provides the same result – lower intensity – but for different underlying reasons. An LED is a device that is either on or off: there's no in-between. The brightness of the LED is part of its design and not a result of the voltage level applied. Apply a voltage to an LED and it's on; take the voltage away and the LED is off. So how do we actually make a LED dimmer?



Waveforms that explain how an LED's brightness appears to change.

If you take a source such as an LED and apply a digital signal that has a frequency of less than approximately 30Hz, the LED will appear to flicker. You can see this in waveform 1 in the diagram below. The absence of LED light during the off cycle causes the flicker in the human eye.

So what happens if the frequency is greater than 30Hz, as seen in waveform 2 below? Think of the eye as having persistence: it retains what it sees for certain amount of time, as can easily be demonstrated if you look at a very bright light and then immediately close your eyes. In this case, then, the LED comes on and the eye registers light. If the next pulse occurs fast enough, the eye and brain persistence 'fills in the gap' to make it seem as if the LED light is still on.

Now let's keep the frequency higher than 30Hz, but reduce the duty cycle (pulse width). In reality, the light is turning on and off. However, thanks to the eye's persistence and the high frequency of the signal, we view the LED as being continuously on, but dimmer. You'll get a chance shortly to see the theory in action by using Python code to demonstrate how changing pulse widths changes the LED's brightness.

The frequency rate of 'greater than 30Hz' to eliminate flicker is not a fixed value. Human eyes and brains are unique; some people may not see a flicker at a frequency less than 30Hz, and others (*me!–Ed*) might.

There's another wrinkle, too, in that the outside of the retina has more sensors to detect dark and light than the center of the retina. You've probably experienced seeing flicker in a fluorescent light in an office only when it's in the periphery of your vision. When you look straight at the light, it seems to be completely stable.

On the periphery, you see the flicker because your eye detects it better. In evolutionary terms, of course, this is because it makes sense to be able to detect potential threats – such as a moving dark shape – at the edge of your viewpoint, even when you aren't looking in their direction.

Putting theory into practice

Anyway, let's put the theory into practice by examining the Python code that will control a servomotor and an LED connected to a Raspberry Pi, as shown in the schematic drawing on page 59.

The Python code you'll find in the archive has been annotated with the reference marker text `#[Callout <number>]`. You can open the scripts from the **LXF** archives in your favourite text editor and then locate the reference markers that correspond to the explanation listed below.

Let's examine the code from the script called **one_servo_continuous_move.py**. When executed, this Python script changes the position of the servomotor actuator arm in three increments from left to right continuously. At `Callout 1` we start with:

```
import time
import RPi.GPIO as GPIO
```

These are external Python code modules that need to be referenced so they can be used within the script. The *RPi.GPIO* module is used to control the Raspberry Pi's GPIO pins. This module contains specific parameters to support PWM. Next, `Callout 2`:

```
freq = 50
```

Here's where the frequency value of a digital signal is defined. This parameter can be found in the servomotor manufacturer's specification. The SG92R requires a digital pulsing signal of 50Hz.

```
pwm = GPIO.PWM(GPIOpin, freq)
pwm.start(7.5)
```

This code segment at Callout 3 contains the parameters defining which GPIO pin(s) will be supporting pulse width modulation (PWM) with the frequency value already defined.

```
    pwm.ChangeDutyCycle(2.5)
    time.sleep(1)
    pwm.ChangeDutyCycle(7.5)
    time.sleep(1)
    pwm.ChangeDutyCycle(12.5)
    time.sleep(1)
```

Here at Callout 4 is where the percentage value of duty cycle for the initial PWM is defined. This value positions the actuator arm in the center of its left-right rotation. In the remaining code, the PWM percentage is changed with `pwm.ChangeDutyCycle(2.5)` to move the actuator arm to left, the `7.5` value to move to centre and `12.5` to full right.

Let's examine the code from the script called **PWM_LED.py**. This script changes the brightness intensity of an LED. The initial value is off, then ramps up from lowest to highest intensity using three increments.

```
freq = 74
```

Callout 1: Here's where we define the frequency of the PWM signal. Remember that the frequency rate will cause a flicker in a light source if the value is less than 30Hz. 74Hz is the frequency value to eliminate this flicker effect.

If you're not fully convinced that the flicker effect results from a PWM frequency of less than 30Hz, modify the script by changing it to `freq = 30` and then start the script. Be aware that a flashing light has the potential to trigger an epileptic seizure in some people. Photosensitive epilepsy is a form of epilepsy in which seizures are triggered by visual stimuli that form patterns in time or space, such as flashing lights; bold, regular patterns; or regular moving patterns. Restore the line to `freq = 74` before continuing with the tutorial.

```
pwm.start(0)
```

Callout 2: We set an initial PWM duty cycle of 0. This will result in the LED being off.

```
    pwm.ChangeDutyCycle(20)
    time.sleep(5)
    pwm.ChangeDutyCycle(60)
    time.sleep(5)
    pwm.ChangeDutyCycle(90)
    time.sleep(5)
```

Callout 3: The PWM duty cycle parameters are set from 20 to 60 to 90, resulting in lower to higher LED brightness intensity. The higher the duty cycle the longer the LED is on, creating what appears to the human eye as being more light.

Pulsing knowledge

Congratulations, you've just used signal waveform drawings to create two basic Python scripts to control entirely different circuits.

In the first the angular or linear position of a servomotor was directly related to the width of the pulse in the frequency of the signal applied. In the second the light emitted from an LED is either on or off, so the PWM signal frequency must be greater than 30Hz to eliminate any flicker. The higher the duty cycle, the brighter the LED appears to the human eye.

And done! Basic theory on PWM, two simple circuits and Python scripts running on a Raspberry Pi. Enjoy.

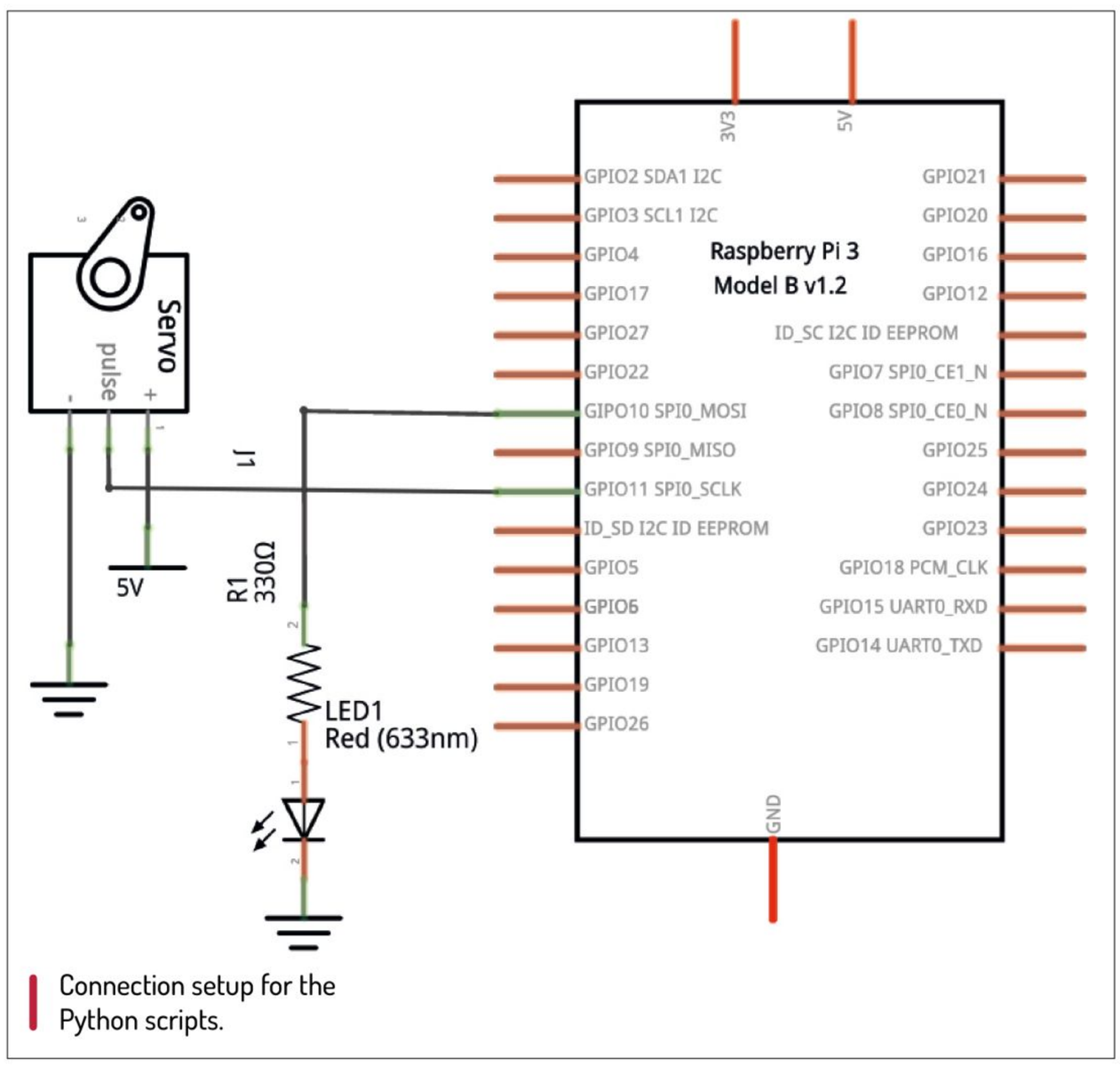
» WHY ALL THE DIFFERENT TERMS?

If you don't have much of a background in electronics, you might find it difficult to follow technical descriptions, especially when different terms are used in the same article to represent the same thing. Two examples would be 'cycles per second' and 'Hertz', and 'duty cycle' referring to 'pulse width'. We've experienced the same frustration and we've made every effort to use the same term or collocate them with its alternate form.

The ideal scenario would be to suggest that the interchange of terms in technical writing shouldn't be accepted. It confuses the reader and can make a complex subject matter worse. With five technical writing courses to his credit, the author agrees, but has seen little change in the industry over his 37-year career.

What is the novice suppose to do? The first step in overcoming the frustration is to realise that the practice of using different terms to represent the same thing is common practice in technical writing. The second step is to read more than one source of material – preferable three sources – in order to fully explore the knowledge.

Understand that the frustration doesn't come from something you have done, or didn't do. Some technical writers – us included – fail to step out of their knowledge level and suitably adapt their writing to accommodate a reader's knowledge level. It's a failure of the writer and not the reader.



Connection setup for the Python scripts.

Breakout Garden

Les Pounder once tried to grow a Raspberry bush by planting a Raspberry Pi in his garden. Hopefully, this type of garden will bear more fruit!

IN BRIEF

An I2C expansion board for all models of the 40-pin GPIO Raspberry Pi. Breakout Garden enables users to easily attach breakout-compatible boards to the Breakout Garden via six slots with built-in reverse polarity protection, and install them using an automated installation script. It's ideal for use in the academic and education markets, where ease of use and robustness are key strengths.

No soil or books are required to grow the Breakout Garden from Pimoroni. It's designed to work with all 40-pin GPIO models of Raspberry Pi, fits on top of the Pi and uses the HAT standard to ensure a secure connection thanks to the provided standoffs and screws. In essence, the Breakout Garden is really just a six-port I2C expander that doesn't require any soldering. For those unfamiliar with I2C, it's a connection protocol where devices can be chained together, yet be accessed independently thanks to an address system that tells the Raspberry Pi where each device is connected.

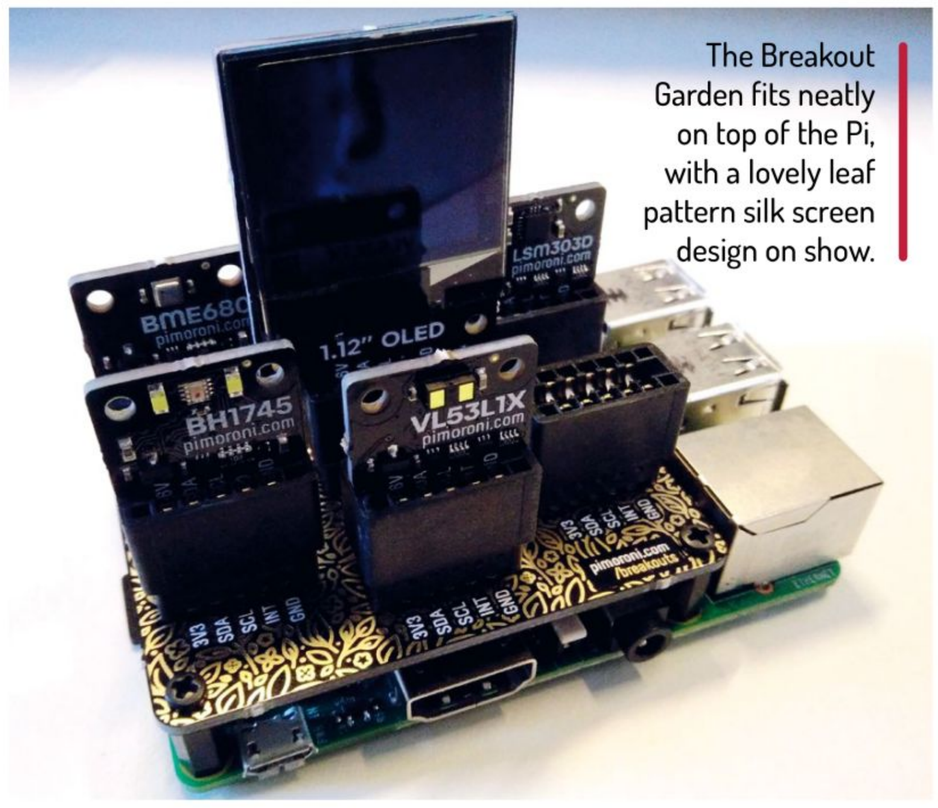
The Breakout Garden has six slots designed to accept one of the new range of Breakout boards from Pimoroni. Each slot has reverse polarity protection, so if a breakout board is inserted incorrectly, it won't damage your Pi. The slots also grip the breakout boards with enough force to ensure they're solid, yet easy to remove. The slots are also rugged enough and not loosened with wiggling or force. Behind these slots are a row of GPIO pins that enable connection using raw GPIO, I2C, SPI and UART, which is handy because this board blocks access to the GPIO.

Software installation for the Breakout Garden is relatively straightforward, after downloading an archive from a GitHub repo. At the time of writing there's no one-line installer, but we've raised this with Pimoroni and it should be addressed in the future.

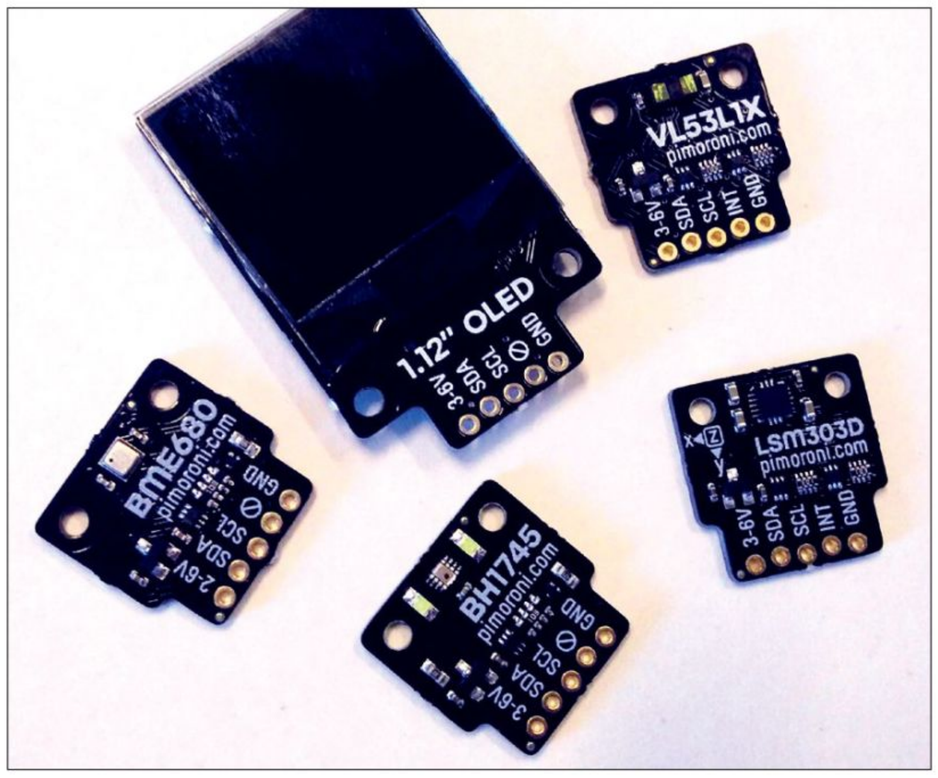
Once installed we next insert the breakout boards into the slots and run the **install.sh** file. This will detect all of the breakout boards in the slot, and then interactively install the necessary Python 3 libraries for each board. In our test the install didn't work for more than one board at a time, an issue we raised via the Github issues tracker for the project and which was fixed in a matter of hours.

So the Breakout Garden has been planted with our breakout boards, and at the time of writing there are six compatible boards on offer. First we see the MLX90640 Thermal Camera (reviewed in **LXF242**), then an OLED screen, which when combined with the VL53L1X laser distance sensor can be used to create a neat parking sensor and display. By using the OLED with the LSM303D eCompass we can create a responsive and educational seismograph, sensitive enough to detect the slightest rumble. The BME680 provides a temperature sensor and air quality sensor which again can be used with the OLED screen to create a basic weather station. The last compatible sensor is a BH1745 colour sensor that can be used to read the colour of an object and this can also be used with the OLED. All of these sensors have been designed to work with the Breakout Garden, and range in price from £9 to £54 for the thermal camera.

At £12 the Breakout Garden is good value, but it's nothing without the breakout sensors and boards. So purchasing breakouts will add to the initial cost. The flexibility of the Breakout Garden, in that we can add and remove boards without powering off the Pi, means that we can try out lots of ideas without downtime. Overall, the Breakout Garden is fun, affordable and easy to use thanks to the recent software updates.



The Breakout Garden fits neatly on top of the Pi, with a lovely leaf pattern silk screen design on show.



Breakout boards can also be used directly with the GPIO by soldering a header pin.

VERDICT

DEVELOPER: Pimoroni
WEB: https://shop.pimoroni.com/products/breakout-garden-hat
PRICE: £12

FEATURES	8/10	EASE OF USE	9/10
PERFORMANCE	8/10	VALUE	7/10

The Breakout Garden is easy to install and provides a simple route to using multiple I2C devices on your Pi. It's a robust method of connecting components without rebooting the Pi.

» **Rating 8/10**

Thermal camera

Les Pounder loves 80s sci-fi film Predator and while a wrist-controlled nuclear weapon is still out of his grasp, he can now see like a Predator!

IN BRIEF

A thermal camera for use with all models of Raspberry Pi and certain Arduino boards. It uses the I2C protocol for minimal pin usage, and can provide a 32x24 resolution live video stream based on temperatures that it can sense in the air.

Adding thermal vision to your Raspberry Pi just got a lot easier, thanks to the MLX90640 Thermal Camera Breakout from Pimoroni. It's sold in two configurations: a 55 degree field of view, much closer to "normal" vision, and 110 degree–useful for either fish eye or wide view applications–and both units feature the Melexis MLX90640 far-infrared sensor array. We reviewed the 55 degree model.

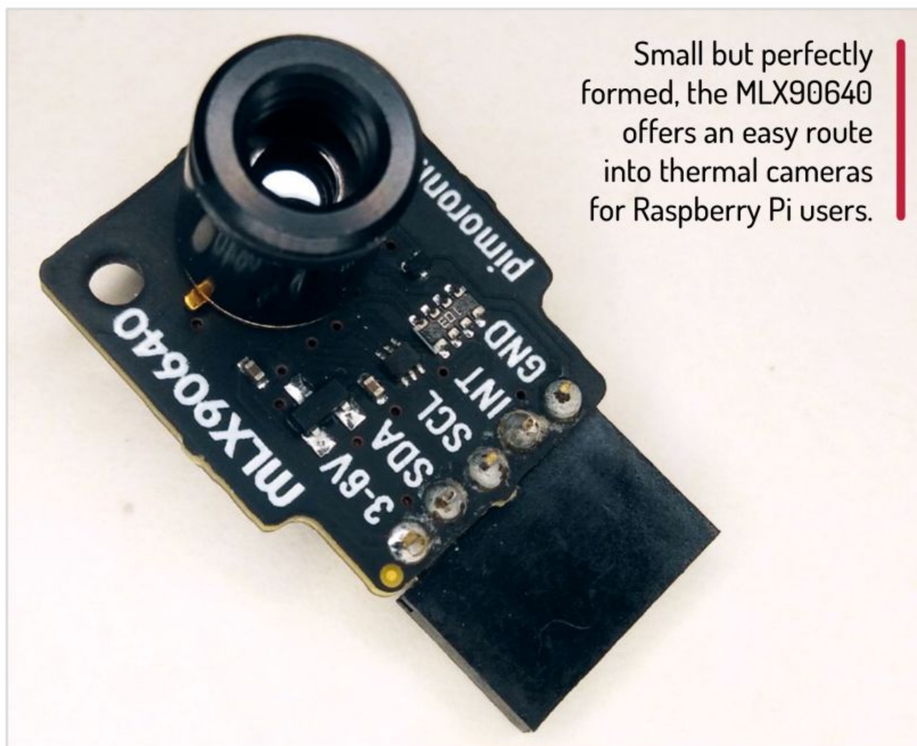
The thermal camera's resolution isn't high; at only 32x24 pixels this isn't a camera for high-resolution work. But the thermal camera can operate at up to 64fps and so can be used to record high-speed experiments in classrooms, for example. The temperature detection range is between -40 and 300°C with 1°C accuracy, so again handy in the classroom but perhaps not for mission-critical projects in the lab.

Installing the thermal camera on our Pi 3B+ test board was a piece of cake. The camera has been designed to fit on top of the I2C pins of a Raspberry Pi. This mean it's also compatible with all models of Pi, including the very first one. However, you'll need to be running the latest Raspbian release. Of course, if you want to use the thermal camera away from a Pi, then you can extend the connections using female to male jumper cables. The thermal camera is compatible with 3.3V and 5V logic, which means it can also be used with certain models of Arduino, too.

Not-so-hot software

With the thermal camera installed, we turned our attention to the software. As usual Pimoroni has provided a one-line installer that downloads the software and installs the files via a Bash script.

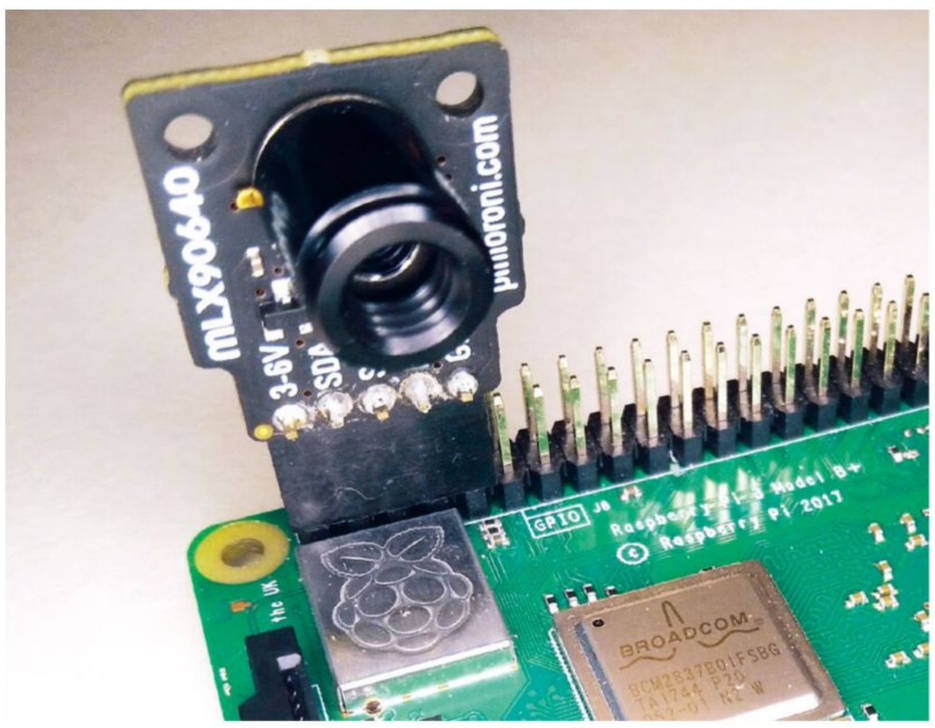
The software for the thermal camera is based upon C, and it includes tools to generate images and video from the thermal camera. Once installed we can navigate to the Python folder and try out the test scripts contained within. Running `test.py` we were quickly able to confirm that the thermal camera was working, and that it could see the temperature of our coffee, visualised as a red splodge on the screen, among cooler-coloured splodges. The included **test.py** script can be modified for use in your projects. At the time of writing there's no Python module from Pimoroni, which typically creates its own package to act as an abstraction layer. This can be installed and imported into your code to handle the thermal camera data and format it for viewing. Hopefully, this will change over time.



Small but perfectly formed, the MLX90640 offers an easy route into thermal cameras for Raspberry Pi users.

The MLX90640 thermal camera is easy to use, but the lack of a Python package will be a barrier to some. If you can get over these barriers then this is a fun and novel sensor to add to your projects. It's easy to install and can provide useful visualisations of temperature data. What's missing, at the time of writing, is a simple scale system that relates the colour shown on screen to a temperature range. This scale is present on FLIR cameras, but they do retail for a significantly higher price.

However, if you'd like to dip your toe into more advanced sensors, then this is a good candidate to start with, and there's even the option of mixing your thermal camera video with another live video stream. This may interest those seeking to gather low-light nature photography, where the thermal camera could act as a trigger and tool to record animal body temperatures.



The thermal camera is designed to fit on to the first five pins of the Raspberry Pi and can be used across all models of Pi.

VERDICT

DEVELOPER: Pimoroni
WEB: https://shop.pimoroni.com
PRICE: £54 (55° or 110° model)

FEATURES	7/10	EASE OF USE	8/10
PERFORMANCE	7/10	VALUE	7/10

The price is quite high, and the software is not quite there, but this board is easy to install and use.

» **Rating 7/10**

Sharing data from remote sensors

Les Pounder reveals how to capture temperature data from remote devices, and how teams can work on a single data source. All for one and one for all!

OUR EXPERT

Les Pounder is a freelance maker who works with organisations such as the Raspberry Pi Foundation to promote maker skills. He blogs at bigl.es

Owing to its small size and easy portability, the Raspberry Pi is ideally suited to working in remote locations. In this tutorial we shall use a remote Raspberry Pi to gather temperature data using a DHT11 temperature sensor and then share that data with our data centre team via Dropbox. Then the data centre team can use their awesome Linux supercomputers to crunch the data and save the world!

All of the code, further screenshots and a detailed high-resolution circuit diagram can be downloaded from **https://github.com/lesp/LXF246-Sharing-data-from-remote-sensors/archive/master.zip**.

Sensor sweep

Connecting the DHT11 to the Raspberry Pi is simple: we used the three-pin version of the DHT11. Connections, looking at the blue plastic cage, are from left to right:

- **Pin 1 to 3V3 GPIO**
- **Pin 2 to GPIO17**
- **Pin 3 to GND**

Use the female-to-female jumper wires to connect the DHT11 to the Raspberry Pi. Now power up your Pi and from the desktop open a Terminal (*LXTerminal*) and type the following command to install the software for the DHT11:

```
$ sudo pip3 install Adafruit-DHT
```

Our next install will be the **dropbox** library for Python. Again in the terminal we type the following:

```
$ sudo pip3 install dropbox
```

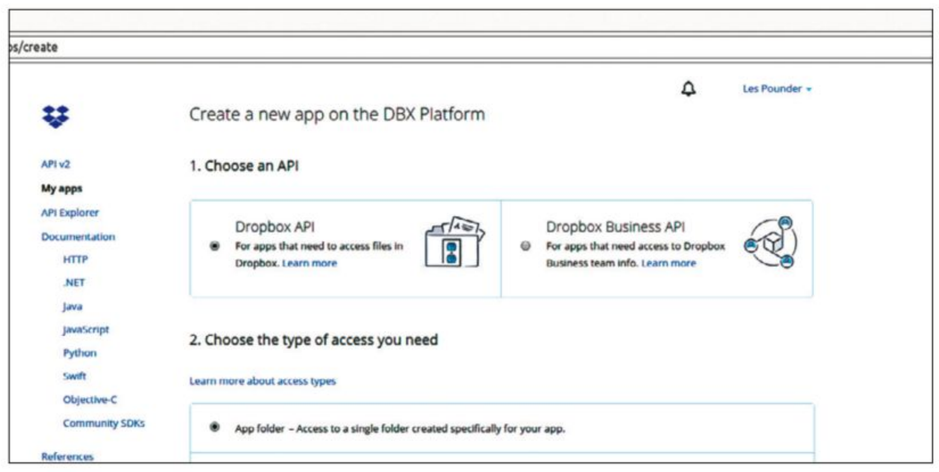
With the software installed, we now turn our attention to setting up our first Dropbox app.

YOU NEED

- **Any model of Raspberry Pi**
- **Internet connection**
- **The latest version of Raspbian**
- **DHT11 temperature sensor**
- **3 female-to-female jumper wires**

Dropbox setup

Dropbox is a cloud storage application that's cross-platform and it has a rather good Python API. To use the API we need to have a Dropbox account. So head over to **www.dropbox.com** and sign up. Once we have an account we next need to login to Dropbox for Developers (**www.dropbox.com/developers**) and from there click Create Your App. This will open a new page that asks us to choose the API. In this case we need the Dropbox API. The required access is "App Folder" and for the final step we need to name the app. We called ours Experiment Data. Click Create App and we shall go to the next screen that gives us a breakdown of the app.



Creating an app gives us lots of choices. The first is to use the Dropbox API, then to save the file to its own app folder. Finally, name the app.

Right now all we need to do is look for Generated access token and click Generate. This will create your access token which our project will use to access Dropbox via Python. Make a note of the token, but don't share the token because it's private to your account.

With Dropbox setup complete, we can now focus on the Python code. Open the Python 3 editor, found in the main Raspbian menu under Programming. Now click File>New to create a new blank document. Immediately save the blank document (File>Save) as **experiment.py** before moving on. Remember to save often.

Our first line of Python code will tell Raspbian where to find the Python interpreter. We'll need this to later run our code autonomously via the terminal:

```
#!/usr/bin/python3
```

The next line of Python code sees us importing libraries to provide extra functionality. In this case we import the Adafruit DHT and Dropbox libraries that we installed earlier. We also import the datetime library used to provide calendar and time data:

```
import Adafruit_DHT, datetime, dropbox
```

To use the Adafruit library we create an object that references the library. By typing "sensor" we can access the entire library without the need to type lots of text:

```
sensor=Adafruit_DHT.DHT11
```

The next object tells the code where to find the DHT11 sensor. The output for that sensor is connected to the GPIO 17 of the Raspberry Pi:

```
gpio=17
```

Our next step is to create an object called `temperature`, which will be a tuple containing two items. First the humidity is saved as item 0 in the tuple.

The next item, at position one is the current temperature in degrees Celsius. The object requires two arguments and these are the type of sensor DHT11 and to which pin it is connected 17:

```
temperature = Adafruit_DHT.read_retry(sensor, gpio)
```

To use Dropbox with Python we need to use the token, which you hopefully recorded earlier. The variable `token` is used to store this long string of characters:

```
token = "<YOUR TOKEN HERE>"
```

Let's make a connection to the Dropbox API by creating an object called `dbx`.

```
dbx = dropbox.Dropbox(token)
```

For debug purposes, we'll print the temperature data to the Python shell. This isn't required, but useful to check that the sensor is working. We use simple positional formatting in the print function. You can see `{}` in the text, and in here Python will drop the temperature date stored in the temperature tuple, at position 1:

```
print("The temperature is {}C".format(temperature[1]))
```

To obtain the current date and time we'll create a variable called `timestamp` that will get the current time, and then use a little string formatting to convert it into something human readable:

```
timestamp = datetime.datetime.now()
timestamp = str(timestamp.strftime("%Y-%m-%d
%H:%M"))
```

With the temperature and time data saved, we now need to write this to a file. For this we use Python's built-in file class to create a file called **data.csv** (if it doesn't exist, Python will create it) and set the file to "append" – this adds the data to the end of the file.

```
f = open("data.csv", "a")
```

We then write the timestamp (date time) to the file, and a trailing `,` as our file is a comma separated value file, used in spreadsheets.

```
f.write(timestamp+",")
```

Then we write the temperature data:

```
f.write(str(temperature[1]))
```

Our next line forces Python to insert a new line, effectively creating a new row in our CSV data ready for the spreadsheet. Then we close the file to ensure that everything is saved correctly.

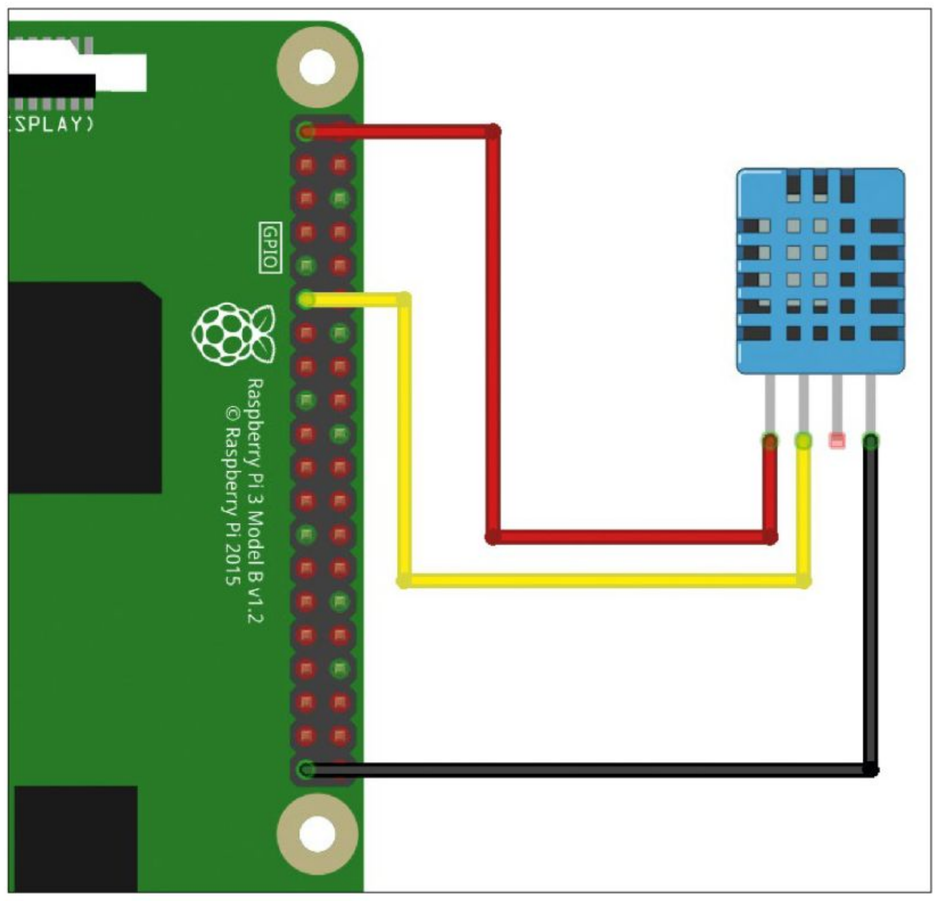
```
f.write("\n")
f.close()
```

The final two lines of Python! Here we open the **data.csv** file in read-only mode and then instruct Python to read it line by line into an object called data. Then we use the Dropbox library to read the file, then save it as **/data.csv** (this is the location in the Dropbox app folder), we mute the output, and then tell Dropbox to overwrite any files matching the name of our file. This will update the file each time the code is run:

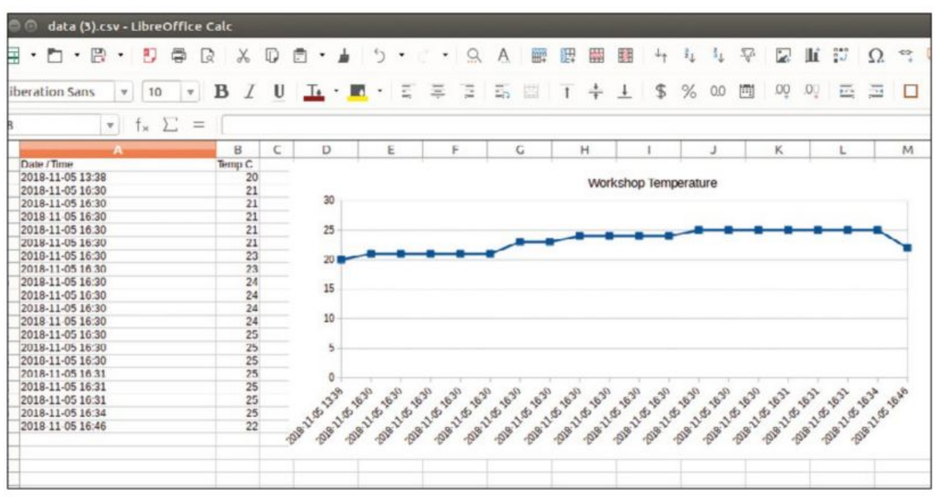
```
with open("data.csv", "rb") as data:
  dbx.files_upload(data.read(), '/data.csv', mute = True,
mode=dropbox.files.WriteMode.overwrite)
```

That's the code! Click File>Save and when ready start the code for a quick test by pressing Run>Run Module from the menu. We can see the Experiment Data folder on the Dropbox website, and via the Dropbox app if used.

So now we can record data, but how can we have it without pressing a button? Well in the boxout (*below*) we show you how to make your code into a application that will start on boot.



Our circuit is simple, thanks to the DHT11 requiring only three connections. In this diagram we have the four-pin version of the sensor. But use the three-pin version instead.



Downloading the data was easy and thanks to LibreOffice Calc we were able to produce a chart for management to study!

QUICK TIP

The easiest way to get the sensor data from Dropbox is to install the app. This will create a folder structure for the application we have created, and our data.csv file will be in the root of the Apps directory.

» CREATING A BACKGROUND APP

At the start of the project we instructed the code to look for the Python interpreter, and now we need to take two steps to make our app run.

First, we need to make the code run from the terminal rather than open the Python editor. In a terminal we shall make the file executable by changing its permissions:

```
$ chmod +x experiment.py
```

Now give it a quick test by typing this into the terminal:

```
$ ./experiment.py
```

If the code works, great! If not, double-check before moving on. The last step is to tell our Raspberry Pi to run the code on every fifth minute in the hour (5, 10, 15, 20 and so on), for that we need to use *cron*. In a terminal type the following:

```
$ sudo crontab -e
```

It will ask which editor to use, choose *nano* to follow this tutorial. Go to the bottom of the file and type.

```
*/5 * * * * /home/pi/experiment.py
```

Then press Ctrl+O, then Enter, then Ctrl+X to exit. Reboot your Pi and every five minutes the code will run and update the **data.csv** file.

Now all we need to do is load the CSV file into our favourite spreadsheet application and start crunching data!

Monitor & archive data

Alex Cox is sick of craning his neck towards the sky and wondering what to wear outside, so it's time to make more of his weather sensors.

Alex Cox is a tech expert and writer, inexorably drawn to tinkering with, and inevitably breaking, anything put in front of him.

Weather, eh? Who'd have it? Whether it is the relentless onslaught of that burning gasball in the sky or the incessant gushing of the rain, weather never ends – and not only does that make it the perfect topic of conversation with half-acquaintances (*editors?–Ed*) and family members, it makes for a great project: using Linux shenanigans and fancy sensors to feed your obsession with meteorology the right way.

In **Linux Format issue 238**, we looked at the MakerLife Weather Station kit. The fact that we have one hanging around means we're making it (and its Pi Zero W, and included sensors) at least the initial basis of this tutorial. But you're not missing out if you don't have the same kit to work with. This is something you can build off your own back: various Pi-friendly stores sell sensors that hook onto GPIO pins or sit in USB ports. You could also exploit the more commercial end of the market. The Netatmo Smart Home Weather Station (a slightly painful £150 investment) can be tapped into by *weeWX*, which you can find out more about in the 'Your own weather website' box (*below right*).

So we have sensors, but how these work to generate the data isn't important – it's the end result, a stream of data points, which is key. How we interpret that data and what we do with it once we've understood it is what turns a simple live-reading weather station into a more capable archival machine. It will then give us the chance to make predictions based not only on what's happening now, but on what happened in the past.

Pulling Python around

The Makerlife kit (which, it seems, has been discontinued) pulls in its data using a couple of Python libraries. The same is likely true for most Raspberry Pi weather sensors. This is a good thing: Python being the LEGO-esque coding toolkit that it is, it's likely that any sensor you pick up will integrate with any programs you happen to write, as long as you import the appropriate libraries. And this gives us a number of opportunities.

We could, on the most basic level, pipe this raw output not straight onto the screen, but into a text file, which we can process later to spot trends. This is easy enough: taking the code obtained by our weather station, we first add a line outside the main loop to open a file in append mode – something along the lines of `f = open("weather.txt", "a")` and then replace any instances where it would print to screen with code such as `f.write("<weather data>")`, with the data being whatever our code had originally pulled out. Be sure to use `f.close()` to cap off the file when you're done with it.

With more tweaking and a little formatting, and you could turn that output into a comma separated file, suitable for importing directly into a spreadsheet and transforming into graphs. Obviously, you need to be a little careful with your storage here, and consider whether you want to write out with such frequency given that a Raspberry Pi tends to run from a rather fragile SD card.

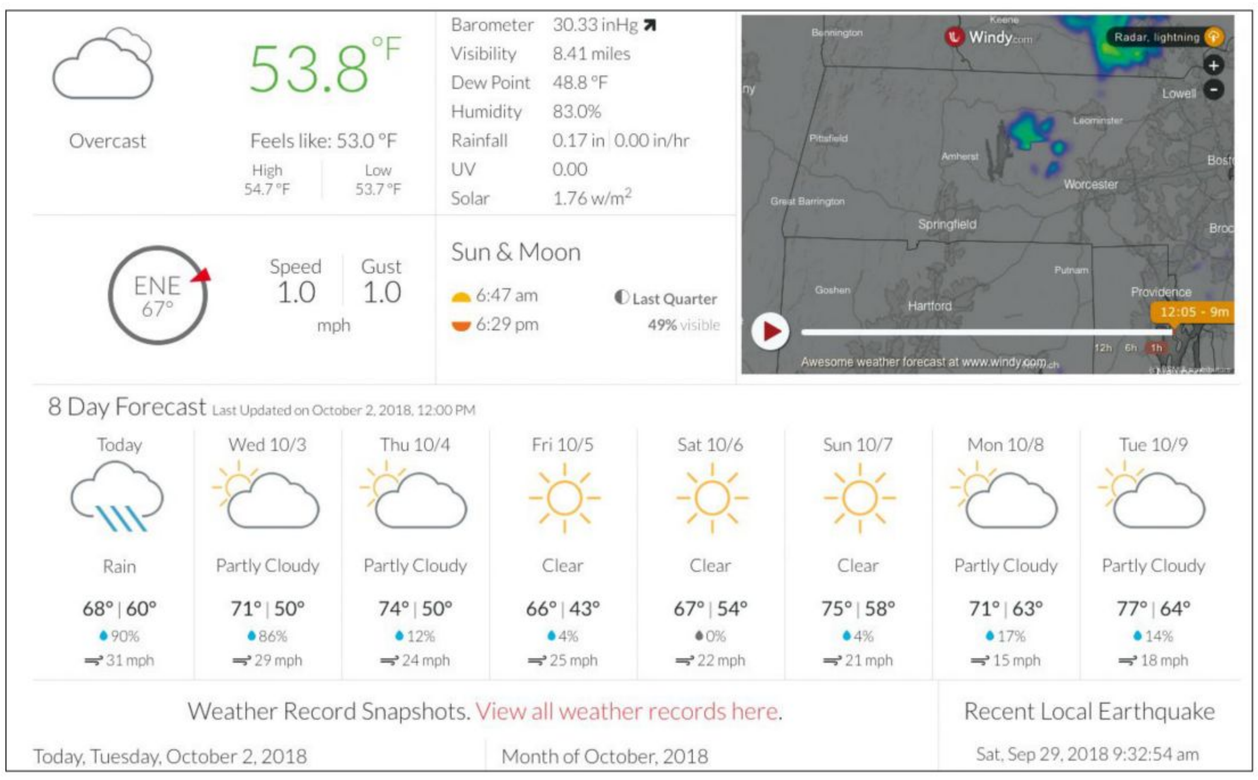
Using weather lists

You could, instead, append that data to a Python list to reduce the number of writes you need to make. Start by creating an empty list variable with `weatherlist = []`. Add a counter variable (you're going to use `i` because, well, that's the done thing) to control both where in that list the data is going to be put, and the frequency that you'll output that data to a file, add one to the counter each time the loop goes through, and switch out your existing `f.write` or `print` statements for `weatherlist.insert(i,<weather data>)` statements which will build your list as it goes.

When your counter reaches an appropriate value, you can then use your preferred write method to output the contents of your list to a file. We'd suggest joining the list items together, dropping it as a single chunk (gracefully closing the file by using the `with` function) then resetting the list and counter once we're done:

```
w_string = ','.join(weatherlist)
weatherlist = []
i = 0

with open('weather.txt', 'a') as f:
  f.write(w_string)
```

This is very basic, inelegant code. You can, no doubt, do better – and you should. It's more a representation



WeeWX – shown here with the Belchertown skin – enables you to combine your personal data with online forecasts.

```
alex@alex-HP-ProDesk-600-G1-TWR: ~
Type "help", "copyright", "credits" or "license" for more information.
>>> import forecastio
>>> api_key = "c3ef6ee3a8d7979cf9733ed5c740e1f9"
>>> lat = -31.967819
>>> lng = 115.87718
>>> forecast = forecastio.load_forecast(api_key, lat, lng)
>>> byHour = forecast.hourly()
>>> print byHour.summary
Breezy starting tonight, continuing until tomorrow morning, and mostly cloudy
starting tomorrow afternoon.
>>> print byHour.icon
wind
>>> print byHour.data
[<forecastio.models.ForecastioDataPoint object at 0x7f650fc27e10>, <forecastio
.models.ForecastioDataPoint object at 0x7f650fc27e50>, <forecastio.models.Fore
castioDataPoint object at 0x7f650fc27e90>, <forecastio.models.ForecastioDataPo
int object at 0x7f650fc27ed0>, <forecastio.models.ForecastioDataPoint object a
t 0x7f650fc27f10>, <forecastio.models.ForecastioDataPoint object at 0x7f650fc2
7f50>, <forecastio.models.ForecastioDataPoint object at 0x7f650fc27f90>, <fore
castio.models.ForecastioDataPoint object at 0x7f650fc27fd0>, <forecastio.model
s.ForecastioDataPoint object at 0x7f650fd3d050>, <forecastio.models.Forecastio
DataPoint object at 0x7f650fd3d090>, <forecastio.models.ForecastioDataPoint ob
ject at 0x7f650fd3d0d0>, <forecastio.models.ForecastioDataPoint object at 0x7f
```

The data section of the DarkSky forecast object contains a bunch of other data objects. Refer to the documentation for how to access these.

of the possibilities than it is a doctrine in Python weather data management.

Now, we're not casting any aspersions here, but it's fair to say that the sensors which your weather station carry are, unless you've gone a little crazy with the spending, likely not dragging in every single weather metric that matters. Maybe you don't even have any weather station gear, you poor thing. Lucky, we can get modular here too – both Yahoo (**https://pypi.org/project/weather-api**) and the excellent *DarkSky* (**https://github.com/ZeevG/python-forecast.io**) offer up live and localised data which you can import and process off your own back.

Dark skies are forecast

Let's look at the latter. Go to **https://darksky.net/dev**, hit the button, and get yourself an API key, which is required to tap into the data. It's free unless you're a heavy user – you get 1,000 API calls per day, and they're charged at a nominal rate after that. Next, jump into a terminal and download the relevant wrapper through *pip* by using `install pip install python-forecastio`.

Once it's installed, pulling that data in is relatively straightforward. At the beginning of your chunk of Python code, add in the relevant libraries with `import forecastio`, then add some variables just below; `api_key = "<your api key>"` keeps that key (which you'll need to pass to the main function) handy, while `lat = <latitude>` and `lon = <longtitude>` store your precise location, useful for making the most of DarkSky's hyper-local forecasting. If you (somehow!) don't already know your latitude and longtitude, **latlong.net** will connect you to an OpenStreetMap instance that'll cough up the goods.

With the prerequisites written in, you can create a forecast object using the following line:

```
forecast = forecastio.load_forecast(api_key, lat, lon)
```

This puts a whole wedge of data, and the relevant methods to extract it, into the forecast object; pull from it by, for example, adding the following (or a more graceful version of it):

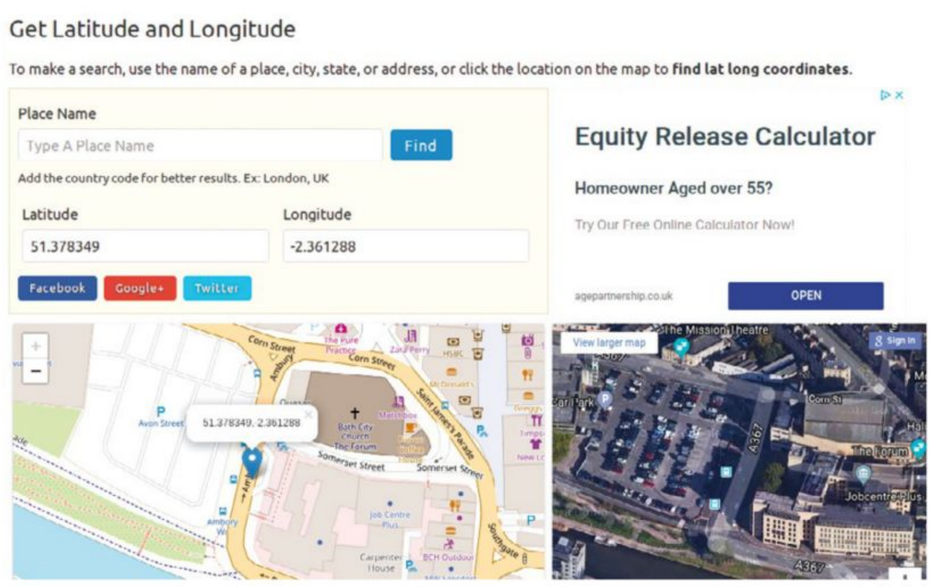
```
weatherHour=forecast.hourly()
print weatherHour.summary
print weatherHour.icon
```

There are tons more options to exploit, if you're after a particular metric, a particular timescale, or anything else. That forecast object also includes `daily()`, `minutely()` and `currently()` methods, and you can simply use `forecast.update()` to refresh the object with the latest info. That raw data is stored in the `data` portion of the DataBlock – do with that what you will.

If you want to play with DarkSky using a different language, you can: there are wrappers for every language from Python 2 and 3 to Ruby, PHP and more to be found at **https://darksky.net/dev/docs/libraries**. Have fun watching the skies!

QUICK TIP

If you'd rather tap in to Weather Underground than DarkSky, there's a python wrapper available for the WU API at pypi.org/project/Wunder Weather/ – or install it using pip with the following: pip install Wunder Weather.



Finding your latitude and longitude is quick and easy, and providing your precise location means ultra-accurate forecasting.

Devices like the Flotilla (which we wouldn't necessarily leave outside) can get you started with weather monitoring for as little as £7.

» YOUR OWN WEATHER WEBSITE

As much as we can play with Python to improve the output of our weather station, there's a lot to be said for at-a-glance personal forecasting. And if you've already dug up an DarkSky key, something like *weeWX* (**weewx.com**) is the perfect tool to get it done. Follow the install guide at **www.weewx.com/docs/usersguide.htm** to get it installed and started for the first time; once it's up you'll be able to access its interface via a local web page.

WeeWX does support a vast number of sensors, although the default list is restricted by the drivers it includes. If it works with your weather gear, that's great: you'll be able to use *weeWX* to join in the swarm of user-run weather stations that contributes to WeatherUnderground's vast forecasting network.

If you're not using any sensors, on the other hand, you can simply add the appropriate latitude, longitude and such into the **weewx.conf** file, and place your API key in the **darksky_secret_key** option to draw the data in directly from there. Make sure *WeeWX* isn't polling too frequently, lest you raise the ire of the DarkSky API limits. And if you want something prettier, don't miss the Belchertown skin at **https://github.com/poblabs/weewx-belchertown**. It's gorgeous.

Monitor humidity with the I2C bus

Tam Hanna feels moist and no one likes that, so he's monitoring his environment.

OUR EXPERT

Tam Hanna whiles away the weekends monitoring his new basement HQ, crafting items on his CnC milling machine and plotting murder most fowl for his tea.

Living under the ground has benefits and disadvantages. While the person who treats himself to a subterranean living space starts to appreciate an absolutely noise-free sleeping experience, managing underground real estate challenges even experienced landladies such as my wife. The inspiration for this story struck when she had to travel to Germany for business: she wanted to keep an eye on the humidity and temperature levels of our headquarters.

The development of the semiconductor industry has led to smart sensors which combine a sensing element with conditioning logic. Output is handled via hardware buses such as I2C, the bus which we'll use in the following tutorial. The Texas Instruments HDC2010 is an excellent temperature sensor; it not only takes care of temperature, but also keeps an eye on humidity. All that is done with a pretty impressive accuracy of within two per cent. While this might not sound like much, it's the absolute high end of what is currently possible in affordable semiconductor sensors.

Sadly, Texas Instruments makes the HDC2010 in an extremely small package. Soldering this by hand is impossible, and reflowing it with the normal reflow oven is difficult at best. The official evaluation kit from Texas Instruments also is quite pricey, leaving unexperienced designers in a bit of a rut.

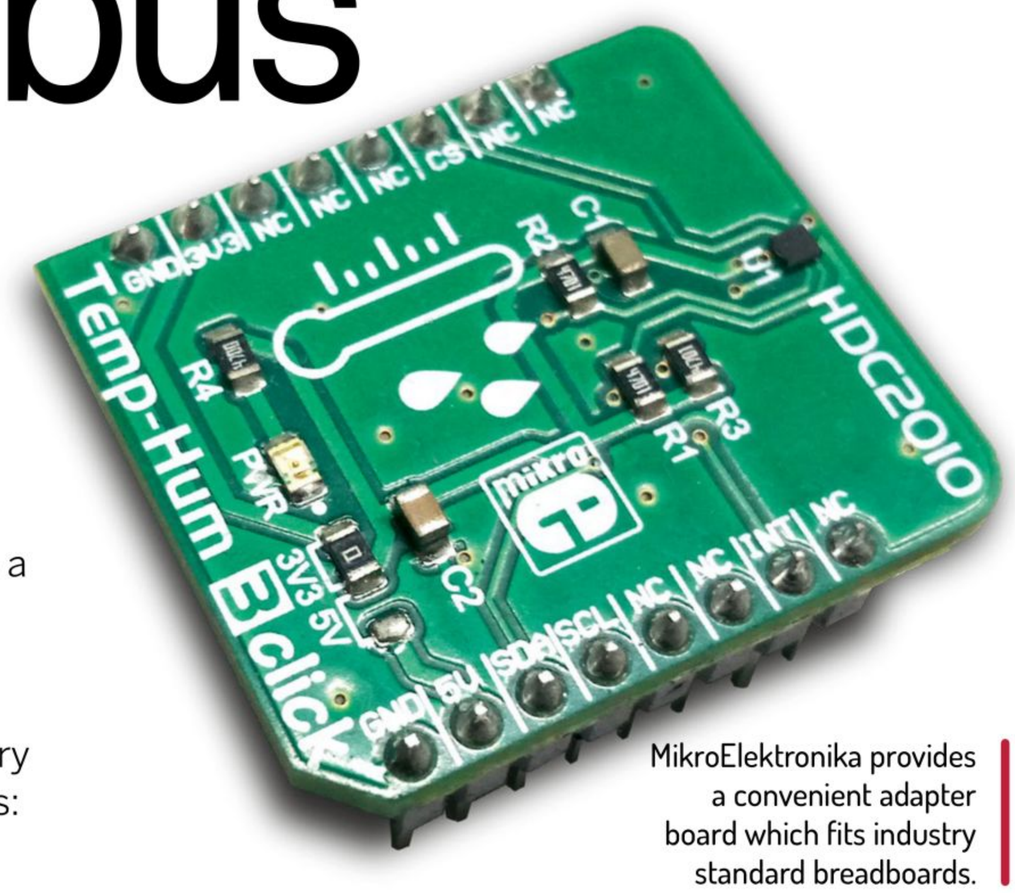
Temp&Hum ahoy

Fortunately, MikroElektronika recently started devoting resources to solving this problem. In the case of the HDC2010, the solution goes by the name of Temp&Hum 3 Click and can be yours for about $13 from **www.mikroe.com/temp-hum-3-click**.

Freshly downloaded versions of Raspbian usually disable the Pi's I2C interface. Fortunately, solving this problem is not hard – open *raspi-config*, switch to Interfacing options and enable the I2C interface. After the obligatory reboot, the I2C interfaces show up in the device tree:

```
pi@raspberrypi:/dev $ ls | grep "i2"
i2c-1
```

Scanning for the presence of devices is best accomplished via a small utility called *i2cdetect*. It's not part of the standard distribution, so download it from the package repository as follows:



MikroElektronika provides a convenient adapter board which fits industry standard breadboards.

```
pi@raspberrypi:/dev $ sudo apt-get install i2c-tools
```

After that, it's time to assemble the actual circuit: see the diagram on page 56. Experienced electrical engineers might wonder why there is no pull-up resistor present. The answer is found in the schematics of the MikroElektronika board *(see page 57)*: the company put a set of pull-up resistors in place.

Let's take a quick look at how I2C actually works. The bus, originally developed by Philips for various hi-fi applications, consists of the master and a set of slaves. The SCL line – short for serial clock – is toggled by the master. It is responsible for setting the speed at which the entity of the bus operates. SDA – short for serial data – is modulated by master or by slave in order to enable actual data transmission.

Master and slave can communicate over one line due to the use of the open drain principle. The above-mentioned pull-up resistors lazily pull SCL and SDA to the positive voltage level. Both master and slaves can pull the line down via a transistor – the interesting bits of this technique are explained in the box on page 55.

For us, however, the next step involves finding out if the Raspberry Pi is able to detect the HDC2010. This is accomplished by *i2cdetect* - the parameter `-y` eliminates the highly annoying warning about potential problems which might occur during the scan process:

```
pi@raspberrypi:~ $ i2cdetect -y 1
     0 1 2 3 4 5 6 7 8 9 a b c d e f
00:         -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: 40 -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

Should your *i2cdetect* run not show a slave presence at address 40, check the wiring. Furthermore, use an oscilloscope to verify the presence of waveforms on SCL; should a logic analyser be on hand, you can also use it to find out more.

Let's code!

At this point, we need to start coding. In theory, opening the data sheet and taking a careful look at it is the action of choice. In practice, coding a driver from scratch tends to be unproductive; almost all semiconductor vendors provide example drivers. In the case of Texas Instruments, simply visit **http://bit.ly/LXF252i2c** and look for the string 'snac075' to download a sample driver for the Arduino platform.

When dealing with manufacturer drivers, that process usually consists of adapting their logic to the API at hand and removing needed parts. For some reason, manufacturers' application engineers insist on the implementation of every useless feature. Like most Arduino examples, the example code provided by Texas Instruments consists of three files. In addition to the header and a **.cpp** file, the semiconductor vendor provides a **.ino** file containing an example demonstrating the API. For convenience's sake, open all three of these files in an editor.

Texas Instruments did an extraordinary job in encapsulating the Arduino-specific parts of the API from the rest of the logic. Because of that, we'll reuse the class provided; in practice, cutting the logic out can often end up being quicker.

Start out by recreating the header file on Raspberry Pi's memory. The contents can be taken one by one from the original file – just make sure to remove the two inclusions of Arduino-specific header files, as they will not be available when compiling C++ code for the Pi. Porting the body of the class is accomplished by using the process set out in the book *Code Reading: The Open Source Perspective* by Diomidis Spinellis. He recommends that code should be considered compilable: make the compiler read it, and use the list of errors to find a working solution.

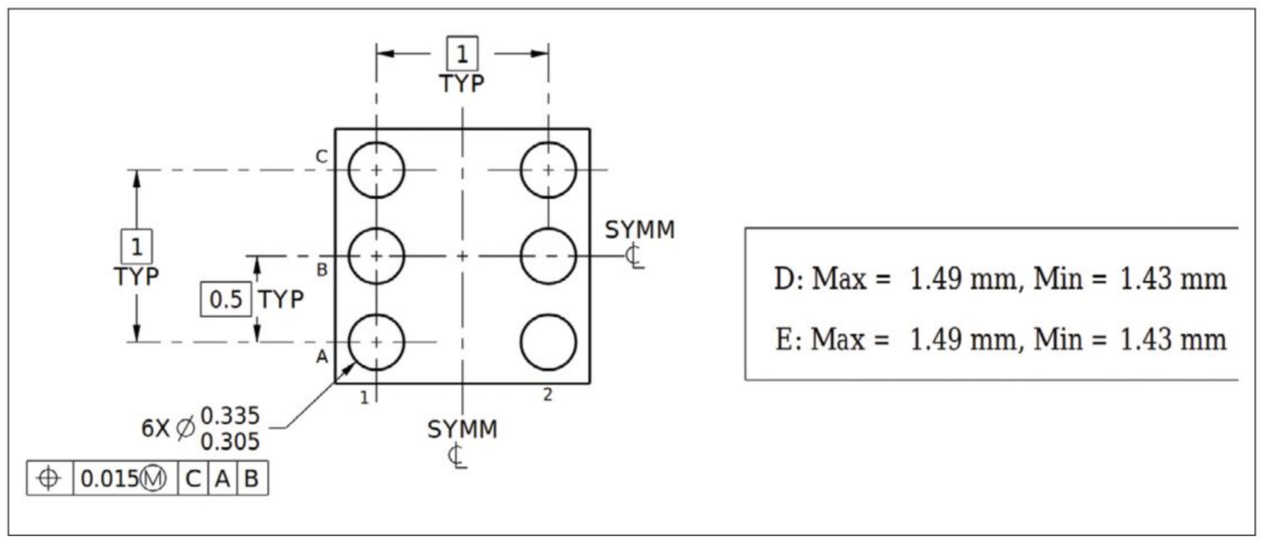
The first step for porting the **.cpp** file involves adjusting its inclusions. Obviously, the Arduino-related library include must be replaced with one intended for the Raspberry Pi:

```
#include "HDC2010.h"
#include <wiringPiI2C.h>
```

On the Arduino, all header files become part of the system library; in the case of our local compilation process, the file must instead be loaded from the current working path. We are ready to order the first compile run:

```
pi@raspberrypi:~ $ gcc HDC2010.cpp -lwiringPi
...
HDC2010.h:98:3: error: 'uint8_t' does not name a type
```

When done, the compiler emits a list of more than a few dozen errors. Glance over them to find low-hanging fruit to eliminate first; eliminating simple problems reduces the length of the error list. We decided to go after those related to the various `uint` types first. Eliminating these first clears the 'fog of war' surrounding the codebase at hand.



The HDC2010. is small and placing it by hand is almost impossible, as the lands on the PCB are really tiny.

A classic nuisance when moving code between platforms involves variable declarations. On the Arduino, a set of types for specific integers is declared to simplify interfacing hardware to code. The Raspberry Pi, obviously, does not have these types, as the system usually is not used for direct hardware interfacing.

Fixing the problem is easy. Open the header file, and add the following defines to bind the tags to variable declarations:

```
#ifndef HDC2010_H
#define HDC2010_h
#define uint8_t unsigned char
#define uint16_t unsigned short int
```

Running the new version of the code leads to a significantly reduced number of errors; another run of *GCC* shows the problems that still need to be tackled.

A careful look at the warning messages shows us that Texas Instruments implemented a set of functions to encapsulate the Wiring API used for I2C communications from the rest of the driver. We need to look at the functions responsible for hardware interaction, and the rest of the code can be used as-is.

Let's start our work with the `begin` function that is responsible for establishing a connection. In case of the Wiring API, creating a connection to the I2C engine is

QUICK TIP

Underground locations can have humidity problems: air, like all gases, can hold a specific amount of water depending on its temperature. As the temperature falls, it holds less humidity – pumping dry, hot air into a dry but cold space always leads to condensation.

» A QUESTION OF (LIMITED) RANGE

One interesting problem of the 'open drain' architecture involves the behaviour of the bus under stress. While the low transition normally is sharp, the charging of the line takes place according to the exponential function known from capacitors.

The capacity of the cable increases with its length and puts limits on the length of I2C buses. When Philips designed the standard, its engineers never expected the bus to extend beyond the confines of a small printed circuit board.

Should you ever find yourself in the situation where a high-range I2C bus is needed, multiple venues are available. First of all, the value of the pull-up resistors can be reduced so that more power is available to charge the 'cable capacitor'. This approach, however, has its limits: keep in mind that the master and the slaves need to sink the current supplied.

Another approach involves reducing the speed of the bus. Most sensors can work with almost any speed ranging from 100 to 400kHz. The slower the bus, the more time is thus given to the resistors to restore the line to the positive voltage level between data exchanges.

simple. Sadly, the wiringPi API takes a different approach in that it returns a file descriptor for the interface. Due to that, we need to add a member to the class definition which holds the file descriptor:

```
    private:
        int myFD;
        int _addr;  // Address of sensor
```

Once this is out of the way, the `begin` method can be adjusted so that connections can take place:

```
void HDC2010::begin(void)
{
    myFD = wiringPiI2CSetup (0x40) ;
}
```

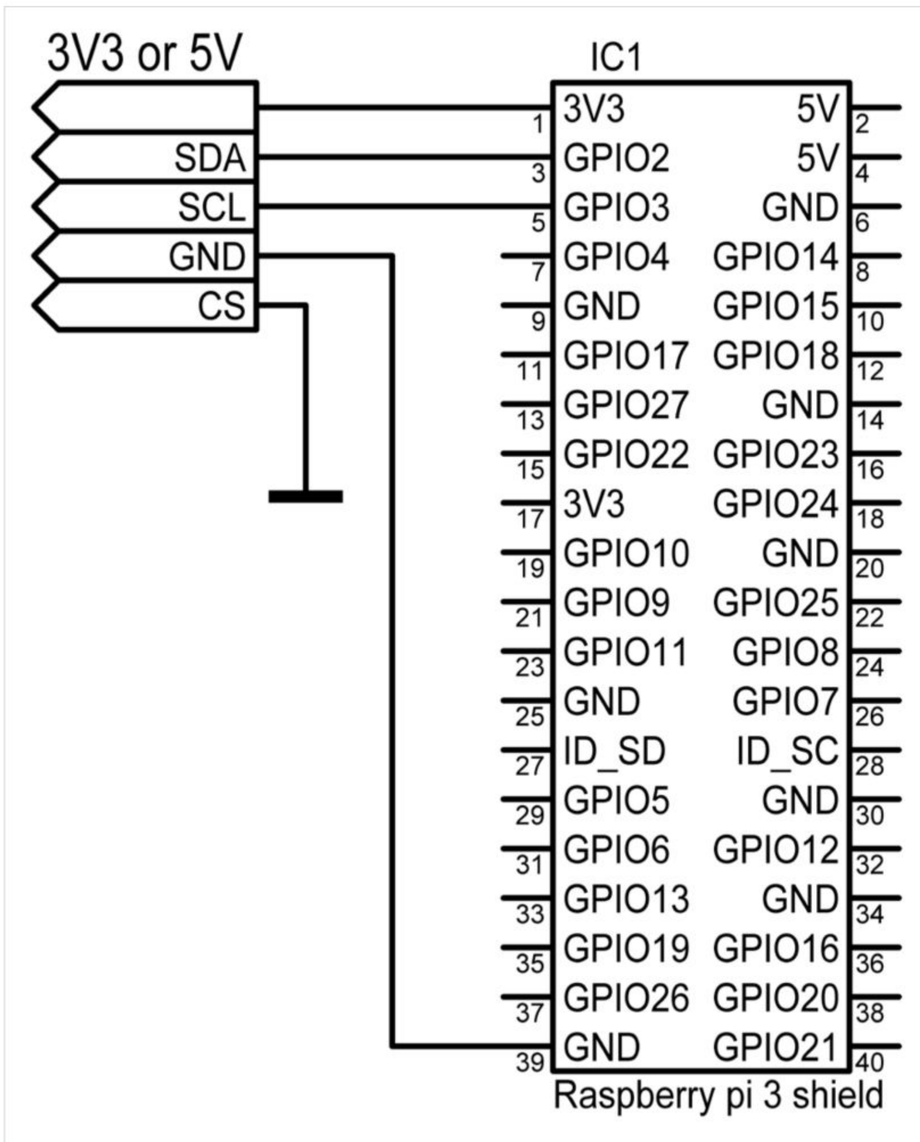
Purists might complain about our practice of not checking the return value carefully; while checking for `-1` is recommended in production code, we omitted it here for clarity reasons.

While finding the best approach to navigate the compile errors is both art and science, we consider the `reset()` function to be the next valuable target:

```
void HDC2010::reset(void)
{
    uint8_t configContents;
    configContents = readReg(CONFIG);
    configContents = (configContents | 0x80);
    writeReg(CONFIG, configContents);
    delay(50);
}
```

While `reset()` looks free of accesses to the Wire API, the use of the `delay` function – normally part of the wiringPi library – causes issues. We limited ourselves to including the header dedicated to the I2C library; add the following declaration to eliminate another problem:

```
#include <wiringPi.h>
```



Connecting the HDC2010 to the Raspberry Pi is not difficult.

A question of registers

While the I2C bus can, in theory, transmit just about any kind of information, most parts work on a register basis. The internals of the part to behave like a classic key value store, which can be addressed, written to and read from by the master.

On the Arduino's I2C API, writing values is a two-step process. The register first needs to be opened for modification, after which the task at hand can be done. In the case of our driver, the design pattern is manifested in function `openReg`:

```
void HDC2010::openReg(uint8_t reg)
{
  Wire.beginTransmission(_addr);  //Connect HDC2010
  Wire.write(reg);  // point to specified register
  Wire.endTransmission();   // Relinquish bus control
}
```

While `openReg` is not needed on the Raspberry Pi, removing the function completely would require significant rewriting of our code. A more comfortable way involves removing the entity of the code and replacing it with no operation comment:

```
void HDC2010::openReg(uint8_t reg)
{
  //NOP - No OPeration on RaspBian
}
```

Readin' and writin'

Now that the individual activation of the bus is out of the way, we need to proceed to reading and writing register information. The original writing function for the Arduino is complex, as the Wiring API requires you to send the register and the data bytes separately:

```
void HDC2010::writeReg(uint8_t reg, uint8_t data)
{
  Wire.beginTransmission(_addr);  // Open Device
  Wire.write(reg);  // Point to register
  Wire.write(data);  // Write data to register
  Wire.endTransmission();  // Relinquish bus control
}
```

I2C bus transactions can involve either 8- or 16-bit values. A careful look at the types used in the methods of Texas Instruments' example show us that the HDC2010 is purely an 8-bit device. Consulting the documentation of the wiringPi API reveals the presence of a dedicated function intended for writing 8-bit

» HUNT DOWN SERIAL PROBLEMS

When selling high-end oscilloscopes, sales people usually jump back and forth about the variety of protocol decoders implemented a model. Their obvious benefit is that the logic content of the waveforms can be shown alongside the physical voltage information gained: you don't only see the actual voltage levels, but also get some kind of interpretation in an overlay or a window shown at the side of the waveform display.

Being admittedly old-fashioned, we don't like this – for a simple reason – experience shows that issues usually are not caused by electronics, but by software or other logical problems such as a misunderstood datasheet or a mistake in the reference driver. Tracking these down usually requires you to look at your code – which lives on the workstation and not on the oscilloscope.

USB logic analysers such as the Saleae tend to be more convenient in this operating mode. Not only are they cheaper than most high-end scopes, the output can be shown comfortably on your workstation's screen right next to the code in question. Finally, also keep an eye on PicoScope: its extremely powerful decoder API lets you adapt the product to new protocols on demand.

register values. This allows us to create a replacement:

```
void HDC2010::writeReg(uint8_t reg, uint8_t data)
{
  wiringPiI2CWriteReg8(myFD, reg, data);
}
```

Reading, in principle, is done along the same lines. The Arduino API shows itself to be extremely verbose:

```
uint8_t HDC2010::readReg(uint8_t reg)
{
    openReg(reg);
    uint8_t reading;  // holds byte of read data
    Wire.requestFrom(_addr, 1);  // Request 1 byte
    Wire.endTransmission();  // Relinquish bus control
    if (1 <= Wire.available())
    {
        reading = (Wire.read());  // Read byte
    }
    return reading;
}
```

As we dealing with 8-bit values, the `readReg` function can also be greatly abbreviated:

```
uint8_t HDC2010::readReg(uint8_t reg)
{
    return wiringPiI2CReadReg8(myFD, reg);
}
```

Save the changes to the file and order another compilation process – it will still fail, but with an extremely interesting message:

```
pi@raspberrypi:~ $ gcc HDC2010.cpp -lwiringPi
(.text+0x34): undefined reference to `main'
collect2: error: ld returned 1 exit status
```

When compiling a C++ program not intended to be a library, the C++ standard requires the presence of the `main` function – so far, we've focused on simply porting the classes. It's now time to create another file and start by including the various headers required:

```
#include <stdio.h>
#include "HDC2010.h"
#include <wiringPi.h>
```

After that, the actual `main` function can be created:

```
int main()
{
  HDC2010 sensor(0x00);
  sensor.begin();
  sensor.reset();
```

Adapting the example to an C++ environment is made difficult. Most importantly, C++ code living in the Arduino environment gets invoked in relatively complex ways. Due to this, our instance of the driver class is obliged to live on the stack.

After creating it, we invoke both the `begin` and `reset` methods. Invoking `reset` might look superfluous at first glance, but makes good sense – when interacting with sensors, ensuring clear start-up situations is a great design pattern.

In the next step, various parameters must be written to the sensor in order to inform it about the way we want our data provided:

```
  sensor.setMeasurementMode(TEMP_AND_HUMID);
  sensor.setRate(ONE_HZ);
  sensor.setTempRes(FOURTEEN_BIT);
  sensor.setHumidRes(FOURTEEN_BIT);
  sensor.triggerMeasurement();
```

After the configuration is complete, it is time for harvesting the actual values. As Texas Instruments' example driver takes care of the conversion for us, the actual information collection can be done like this:

```
  float temperature = sensor.readTemp();
  temperature = sensor.readTemp();
  float humidity = sensor.readHumidity();
  printf("Temp: %f | Humi: %f", temperature,
humidity);
  return 0;
}
```

We invoke the `readTemp` method twice for a reason. The HDC2010 family has a small oddity: after being configured for the first time, the temperature sensor always returns a value of -40°C. This value gets discarded immediately after the first read operation; by invoking the method twice, we can ensure that valid values are available to our program.

At this point, another invocation of *GCC* is needed. This time, both the driver class and the main file need to be parsed in order to prevent linker errors:

```
pi@raspberrypi:~ $ gcc HDC2010.cpp test.cpp
-lwiringPi
```

When done, temperature and humidity values can be harvested from the command line:

```
pi@raspberrypi:~ $ ./a.out
Temp: 21.170044 | Humi: 45.507812
```

Should you not believe our claims about the issue with the -40°C reset, feel free to remove the second invocation of the reading function. After ordering another recompilation, the temperature value returned will no longer be valid:

```
pi@raspberrypi:~ $ ./a.out
Temp: -40.000000 | Humi: 45.782471
```

What now?

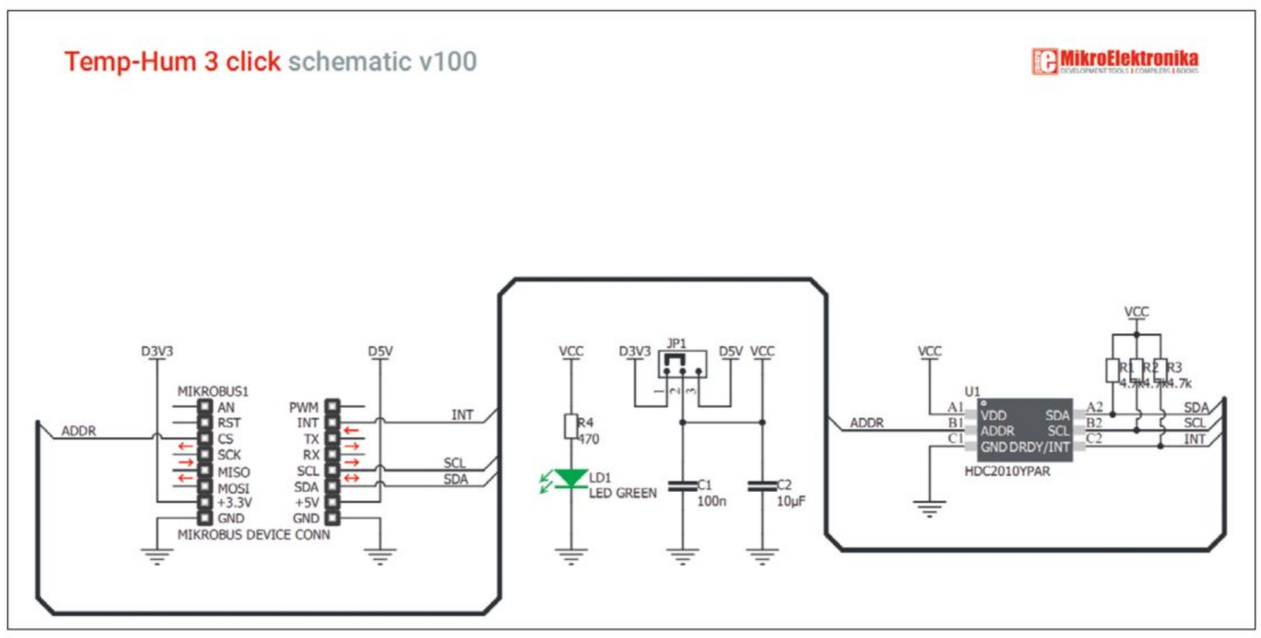
Our Raspberry Pi is able to collect both temperature and humidity information. It can be shared across the internet in a variety of ways – in our case, we used a simple proprietary protocol based on the Berkeley Socket API.

This, however, is a topic better left to software engineers. For the electrical engineering team, another interesting problem arises: given that I2C is intended to cover multiple devices, we should be able to add multiple HDC2010 instances. This raises a set of new questions, the answers to which we will introduce in another issue.

For now, we hope that you enjoyed our little experiments with the world of sensor interfacing on the Raspberry Pi.

QUICK TIP

Should you, for some reason, feel uncomfortable using the wiringPi library, you can also fall back to the I2C library contained in the Linux kernel. More information on this programming approach can be found at http://bit.ly/lxf252i2c.



MikroElektronika's adapter PCB can work off both 3V3 and 5V. As the board does not, however, contain any active voltage regulators, there is no need to move jumpers around.

Build a space station alert system

Les Pounder shows us how to create an International Space Station alert system, so that we never miss it going by overhead.

OUR EXPERT

Les Pounder is a freelance maker who works with organisations such as the Raspberry Pi Foundation to promote maker skills.

Here's an interesting little project in which we'll learn how to get data from the web and use it to tell us when the International Space Station will fly overhead, with the help of Pimoroni's Unicorn HAT LED board (£25.50 from **http://bit.ly/lxf251unicorn**).

Ground control

With the Raspberry Pi unpowered, connect the Unicorn HAT so that it fits neatly on top of the Pi, connecting to all 40 GPIO pins. Connect your keyboard, mouse, HDMI and power to the Pi and boot to the desktop. You will need the Pi connected to the internet for this.

In a terminal, type the following and press Return to install the Unicorn HAT Python library.

```
$ curl https://get.pimoroni.com/unicornhat | bash
```

Follow the install instructions and keep the terminal open, as we need to install more libraries.

```
>>Installing the Python Libraries<<
```

For the project we need two extra Python 3 libraries. The first is called 'requests' and is used to work with data from the web. The second is called 'geopy' and provides us with a location based on our IP address. In the terminal, type the following to install these:

```
$ sudo pip3 install requests geopy
```

After a few moments the libraries will be installed and we can close the terminal.

Using your favourite Python 3 editor on the Raspberry Pi, create a new file and then save the project as **ISS-Alert.py**. We start by importing the libraries for the project, 'requests' and 'unicornhat'.

```
import requests
import unicornhat
```

Then we import two functions from two more libraries. From GeoPy we import `Nominatim`, which is a function to enable our code to look up a location using OpenStreetMap. The second is the `sleep` function, to control the speed of our project.

```
from geopy.geocoders import Nominatim
from time import sleep
```

To use the `Nominatim` location function we shall create an object called `geolocator`:

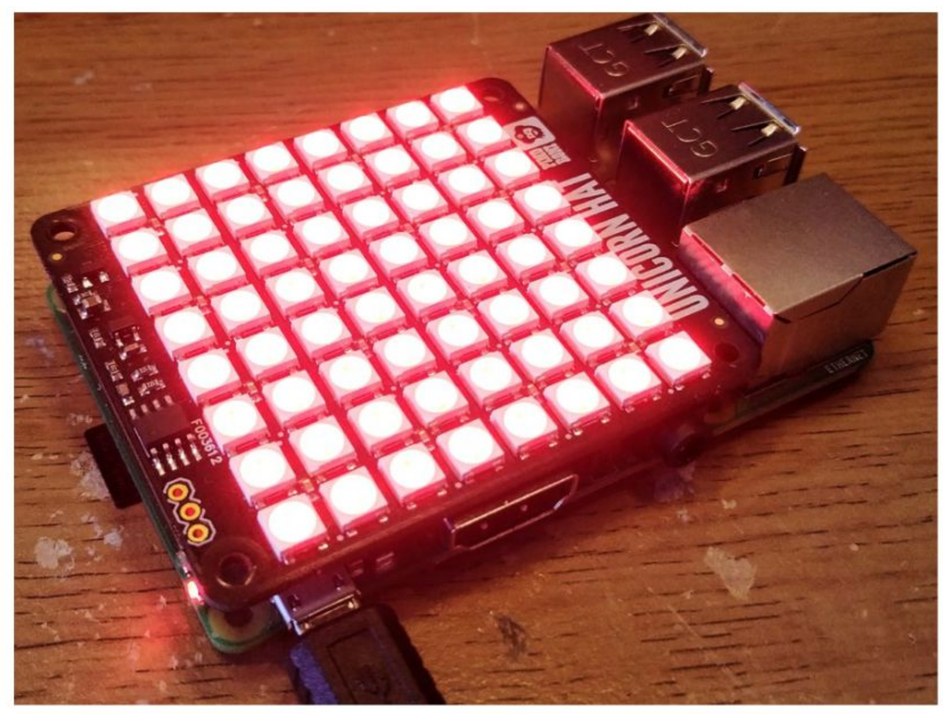
```
geolocator = Nominatim()
```

To access the JSON data we need to provide the URL of where it is located.

```
ISS = "http://api.open-notify.org/iss-now.json"
```

YOU NEED

- A Raspberry Pi 2/3/Zero W
- The latest Raspbian release
- An internet connection
- A Unicorn HAT from Pimoroni
- Code: http://bit.ly/lxf251iss



Alert yourself to the location of the International Space Station using a few lines of Python and 64 ultrabright neopixel LEDs.

How do we find our location? Using geopy, we create an object called `home` and in it we store our geolocation data.

```
home = geolocator.geocode("YOUR TOWN/CITY HERE")
```

To store the latitude data from `home` we need to create a variable called `home_lat`. We then ensure that the data stored is a `float` data type, as we shall be doing a little maths with it later.

```
home_lat = float(home.latitude)
```

We then do the same with the longitude data.

```
home_lon = float(home.longitude)
```

We then print the location data to the shell to check that it is correct.

```
print("HOME")
print(home_lat, home_lon)
```

The main loop of our code is where we use `while True:` to run constantly. The first action in the loop is to get the latest location of the ISS using the JSON file. So we get that data and store it in a dictionary called `r`.

```
while True:
  r = requests.get(ISS)
```

JSON data is contained in `{}` brackets and uses a key: value structure to store data. In Python this is called a dictionary, and we can retrieve data from a dictionary by giving the name of the dictionary and passing the key that we wish to see the value of. So

for the latitude of the ISS we create a variable called `lat` and use that to store the JSON data stored in `r`, specifically the data for `['iss_position']['latitude']`. We also wrap the data in a `float()` helper function that will ensure that our data is a decimal number (floating point number), as we need to use the data later.

```
lat = float(r.json()['iss_position']['latitude'])
```

We do the same for the ISS longitude.

```
lon = float(r.json()['iss_position']['longitude'])
```

Now print the ISS latitude and longitude to the Python shell for debug.

```
print(lat, lon)
```

Here we have a big line of code, but to break it down think of it as a question: "Is the position of the ISS near to my home position?" For this we use an `if` conditional test that checks to see if the latitude and longitude of the ISS is greater than or equal to our home latitude and less than or equal to our home longitude. We subtract 1 from our home latitude and longitude as this will give us approximately 5-10 minutes to get outside and see the ISS.

```
if lat >= (home_lat -1) and lat <= (home_lat) and lon >=
(home_lon -1) and lon <= (home_lon):
```

Sitting in a tin can

So what happens if the ISS happens to be above us? Well, we first print to the Python shell that it is indeed overhead. Then we use two `for` loops, one for `x` and another for `y`. These relate to the x and y axis of our Unicorn HAT board.

```
    print("WARNING: ISS IS ALMOST OVERHEAD")
    for x in range(8):
      for y in range(8):
```

Using these two `for` loops we instruct the Unicorn HAT to set the LED (pixel) at position x,y to red (255,0,0 in RGB colour codes) and then we tell Python to wait for 0.01 seconds before showing the change on the Unicorn HAT.

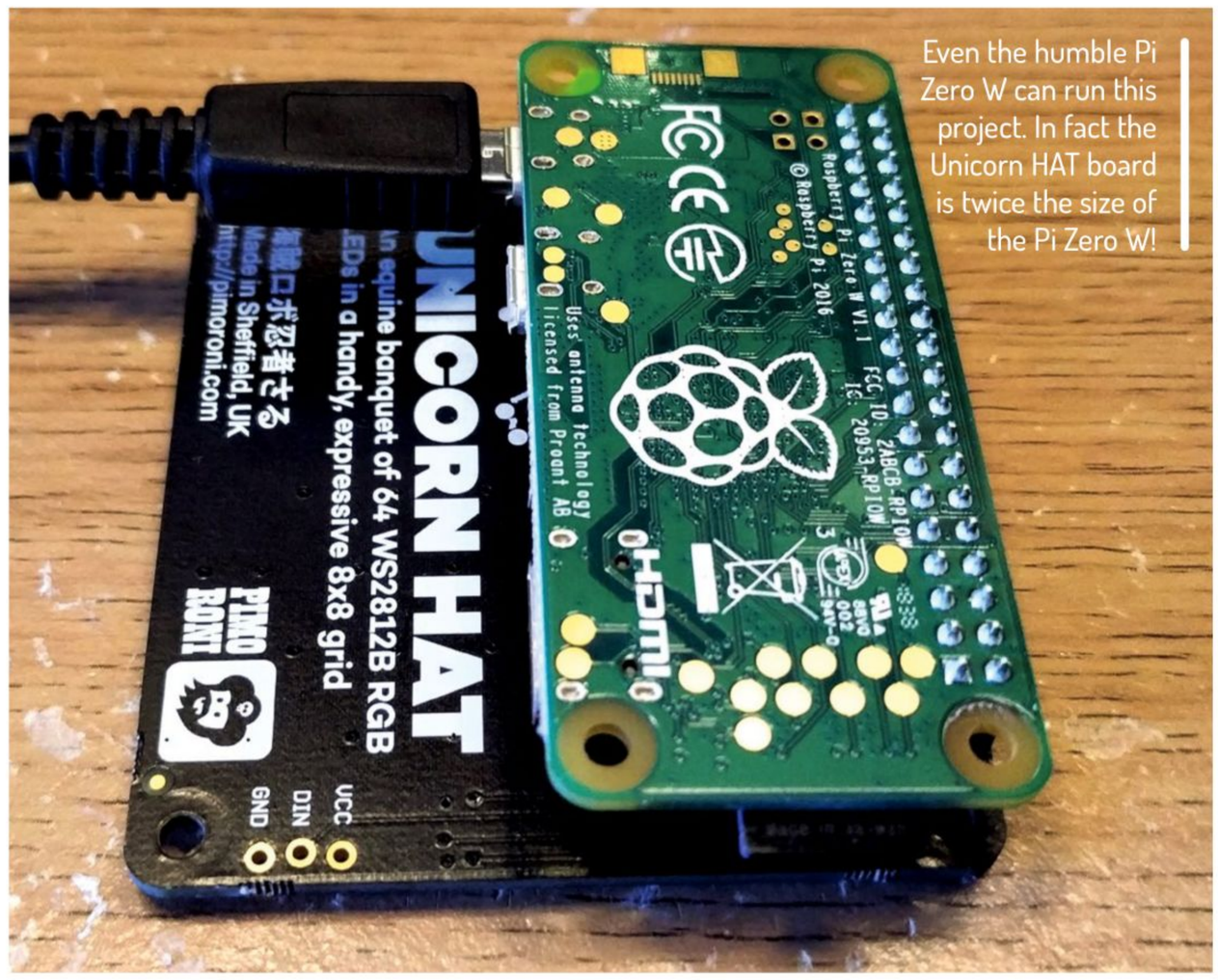
```
        unicornhat.set_pixel(x,y,255,0)
        sleep(0.01)
        unicornhat.show()
```

Breaking out of the two `for` loops we now tell Python to wait for 15 seconds before turning off the LEDs, show the change to the user and than wait for a further 45 seconds before repeating the ISS location check process.

```
    sleep(15)
    unicornhat.clear()
    unicornhat.show()
    sleep(45)
```

If the ISS is not nearby, our `else` condition is activated and we use the exact same `for` loop structure to turn the LEDs green for 15 seconds and then turn them off for 45 seconds.

```
  else:
    print("ISS NOT NEAR")
    for x in range(8):
      for y in range(8):
        unicornhat.set_pixel(x,y,0,255,0)
        sleep(0.01)
        unicornhat.show()
    sleep(15)
```



Even the humble Pi Zero W can run this project. In fact the Unicorn HAT board is twice the size of the Pi Zero W!

```
    unicornhat.clear()
    unicornhat.show()
    sleep(45)
```

Now save the code and open a terminal. The Unicorn HAT can only be used with *sudo*, so to run our code we need to type:

```
$ sudo python3 ISS-Alert.py
```

You should see the Unicorn HAT light up to show the status. If it goes red, run outside and look to the skies for the International Space Station!

» JSON? JSON!

JSON is short for JavaScript Object Notation and was created as a lightweight data-interchange format that is easy for humans to read (*easy for your to say–Ed*) and write. JSON is an open standard and despite mentioning JavaScript in its name, it can be used with many different programming languages. When using Python we can read a JSON file remotely using requests.

```
import requests
ISS = "http://api.open-notify.org/iss-now.json"
r = requests.get(ISS)
print(r)
```

By printing the contents of `r` to the shell we see the entire structure of the JSON file, which is exactly the same as a Python dictionary. We then find the key for the value that we want to use. For example, we found the latitude of the ISS using:

```
r.json()['iss_position']['latitude']
```

We can also create JSON in Python by creating a dictionary and using the `json` library to dump the data into a text file.

```
import json
data = {"January":23.6,"February":24.5,"March":17.1}
with  open("demo.json","w") as output:
  json.dump(data, output)
```

If you would like some interesting datasets to show what JSON can be used for, head to **http://bit.ly/lxf251json** [1] and take a look at earthquake data, movie news, crime statistics and much more.

1) https://github.com/jdorfman/awesome-json-datasets

Assemble a night vision baby monitor

Sean D Conway takes a Raspberry Pi and a night-vision camera and builds a baby monitor for his granddaughter

Our expert

Sean D Conway ihas decided to put his 37 years of experience in electronics and computers into improving on the design of a baby monitor.

When the need for something becomes imperative, you often find different ways of reaching or achieving the end goal. We found the baby monitor that a family member had purchased somewhat lacking. So with thoughts of a Raspberry Pi-based design and a quick search of the internet, we decided to construct our own.

We knew a project without a list of requirements tends to turn into an expensive development exercise with no finished product to show for our time and trouble. With carte blanche spending approval from the women who holds the household purse strings (you can't attach a price tag to something being built for a grand-daughter) a bill/build of materials (BoM) was procured through a series of online purchases.

With the assistance of a "mind map" we built a visual representation of the core design options. We listed the branches for laying down the operating system. Another branch listed the camera software installation and configuration. While a third area held the configuration changes to provide security and enhancements.

The SparkFun Raspberry Pi Zero W Camera Kit contains all the bits and pieces to construct a pan/tilt Raspberry Pi camera. We're assuming that you can assemble this using the kit's instructions.

The 8MP camera module provided in the kit was replaced by the 5MP Infrared IR Night Vision CMOS camera for better low-light support. In addition to the camera, two infrared IR LED boards were attached to enhance the ability to see in the dark.

The Pi Zero case supplied with the kit was replaced with the clear acrylic case listed in the BoM. The author found the design of the case in the kit hampered solid external connections to the on-board connectors. Jiggling cables to resolve intermittent connections was a design feature not desired in the camera install.

HERE'S WHAT YOU'LL NEED...

- SparkFun Raspberry Pi Zero W camera kit
- Clear acrylic case
- 5MP infrared night vision camera module
- Universal gooseneck holder
- 2x AM2302 DHT22 temperature and humidity module

Software

- Raspbian Stretch Lite (2018-03-13)
- Rpi_Cam_Web_Interface.git
- Rpi_PanTilt_Cam.git

A camera mounted to a Pi Zero W with a Raspbian stretch lite OS are the prerequisites for this project. Because the camera is to be a remote installation, we need a wireless setup instead of using a cable for network access. There are a number of resources that describe how to accomplish such a build. If you're unsure where to start, check out the author's article on constructing a drive-in movie theatre in **LXF239**.

Configuring the camera

Log in to the Pi using the default Pi user account. The lite version of Raspbian OS is missing the application to download software from Github repositories, so to obtain files from a git repository we need to use the supporting application:

```
sudo apt-get install git
```

Create a directory called **camera** and then change into that directory. Execute the commands to download the camera application, change to the camera software directory and issue the command to install the software components for the camera video-streaming interface:

```
mkdir camera
cd camera/
sudo git clone https://github.com/silvanmelchior/RPi_Cam_Web_Interface.git
cd RPi_Cam_Web_Interface/
./install.sh
```

The installation script present a dialog screen for making configuration changes. The only default option changed was the credentials for the user. You'll need to recall these credentials once the camera is up and operating, when trying to connect to the web service from a browser on another computer.

After acknowledging "If you desire to start the service page", we're returned to the command prompt. The installation script provides the ability to stop or start the service using the following commands:

```
/home/pi/camera/RPi_Cam_Web_Interface/stop.sh
/home/pi/camera/RPi_Cam_Web_Interface/start.sh
```

If you are curious about the installation, the directory **/home/pi/camera/RPi_Cam_Web_Interface/** will contain the log files that are related to the process, suffixed with the extension **.txt**.

From another computer, open a browser and connect to **http://<IP Address>/html/**. Replace the

<IP Address> attribute in the command with the IP address used by the Pi in your installation (for example, 192.168.2.203).

Success will be apparent if a credential screen is displayed. Recall the user credentials created when the software was installed. The browser should now display the output of the camera.

Moving the camera

Now that the camera video is streaming through the web service, let's establish the configuration to provide pan/tilt functionality. We need to make a small configuration changes to the RPi_Cam_Web_Interface software to support camera movement:

```
sudo mv /var/www/html/pipan_off /var/www/html/pipan_on
```

Now create the channel that enables communication from the browser through the web interface to the servo motors in the pan/tilt mechanism:

```
sudo mknod /var/www/html/FIFO_pipan p
sudo chmod 666 /var/www/html/FIFO_pipan
```

It was discovered during the build that a software module was missing from the installation, which would later prevent the pan/tilt software from working:

```
sudo apt-get install python-smbus
```

Change to the camera directory and issue the command to download the software package. Then change to the directory of the downloaded software to kick off the installation:

```
cd /home/pi/camera/
sudo git clone https://github.com/mhord/RPi_PanTilt_Cam.git
cd /home/pi/camera/Rpi_PanTilt_Cam/
./pantilt.py
```

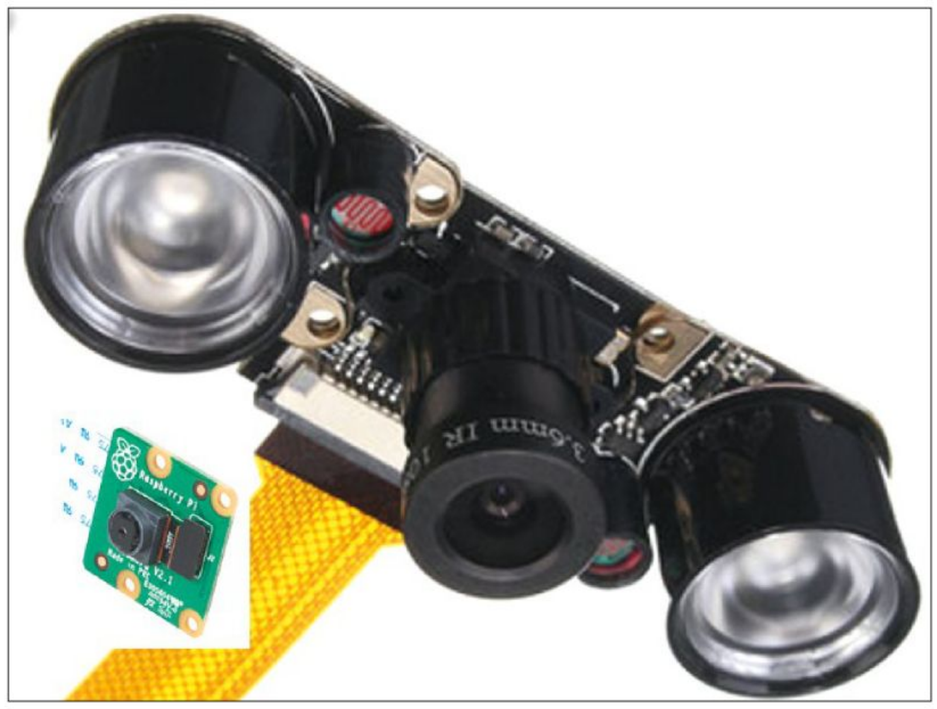
With the camera movement software installation complete, restart the video streaming software *RPi_Cam_Web_Interface* using the commands already provided. From a connected browser, use the arrow keys on the web page (note that this may need a screen refresh) to move the camera.

To ensure that the camera's movement capabilities are available on a reboot, we'll need to make the following configuration changes. The **/etc/rc.local** file is the go-to file to place a call to start an application on boot-up. Using your favourite text editor, add the following line to the file: `python /home/pi/camera/RPi_PanTilt_Cam/pantilt.py` and to the bottom of the file just above the line containing `exit 0`. A quick reboot and we have a camera operating on a pan/tilt platform.

Adding enhancements

Some camera modules have an LED that lights up when the camera is on. Rather annoying for a baby monitor, don't you think? This light can be disabled by using a text editor and adding the following: `disable_camera_led=1 to the bottom of the file /boot/config.txt`. A reboot of the Pi and the LED is duly extinguished.

With a temperature sensor and a short Python script we can measure temperature. This is achieved with a configuration change to a few video streaming software. Now the room temperature can be displayed as part of the data that's obtained by the camera.



We improvised a different camera to improve our night vision capabilities.

Three physical connections are required from the Raspberry Pi board to the temperature module: Pi Board pin 2 to VCC, Pi board pin 6 to GND and Pi board pin 8 to output. The photo (*bottom of page 79*) shows the melted plastic housing of the DHT11 module. It was replaced by the DHT22 sensor during the build because of failure. You can avoid this substitution by not reversing the polarity of the power supply applied to the unit during the construction phase!

Note that the files for this project are downloadable from the Linux Format website. Just head on over to **https://linuxformat.com/archives?issue=240**.

Create a scripts directory and copy the **measure_temp_loop2.py** file from this month's DVD or external archive into the scripts directory:

```
cd \home\pi\camera
mkdir scripts
cp <external source./ measure_temp_loop2.py ~/home/pi/camera/scripts
```

Unix and Unix-like operating systems rely on a program called *cron* that can be used to schedule commands to be executed periodically. We're going to schedule a *cron* job to run the Python temperature script every five minutes to update the reading.

Enter the command `crontab -e in a CLI` and add the following line to the file and save:

QUICK TIP

Keep the PSK plain text password somewhere safe. You can't re-generate the password from the encrypted PSK value.

» JUSTIFYING THE PROJECT

The BoM includes a detailed breakdown of the required kit. The Sparkfun online resources are sufficient to enable a customer to complete the kit build. The kit software is based on a project that has been tracked on a forum at the Official Raspberry Pi site (**www.raspberrypi.org/forums/viewtopic.php?f=43&t=63276**). The Raspberry Pi camera web interface project is fully documented at **https://elinux.org/RPi-Cam-Web-Interface**.

Confused? Why write an article about building a Pi mounted camera if the resources referenced provide all the instructions to do just that? Isn't that redundant? If you recall, the whole camera project was based on the author not being satisfied with what was available. Using the resources as reference, the article tweaks and refines the details to guide the reader to the accomplishing the same task. In addition there are some adds-on such as integrating a temperature display and establishing security.

QUICK TIP

Be aware that turning off certain services without knowledge of their complete use may break specific OS functionality.

```
5 * * * * /home/pi/camera/scripts/measure_temp_loop2.py > /dev/null 2>&1
```

To see what *cron* jobs are scheduled to run on the system, use `crontab -l` (that's an l as in lima) to obtain a listing of the file contents.

The text `/dev/null 2>&1` at the end of the line instructs Linux to store STDERR in STDOUT as well as creating one data stream for messages and errors. If you recall, when a script runs at a command line level there's the potential to generate output. Scripts that are run in *cron* will also report on different levels. There's standard output (STDOUT) and standard errors (STDERR). STDOUT is marked 1; STDERR is marked 2.

Now let's display the temperature reading on the screen. If you delve into the documentation for the streaming software you'll find that what's displayed on the screen can be controlled from attributes configured in the file **/var/www/html/uconfig**. Edit the file to contain the following attributes all on one line:

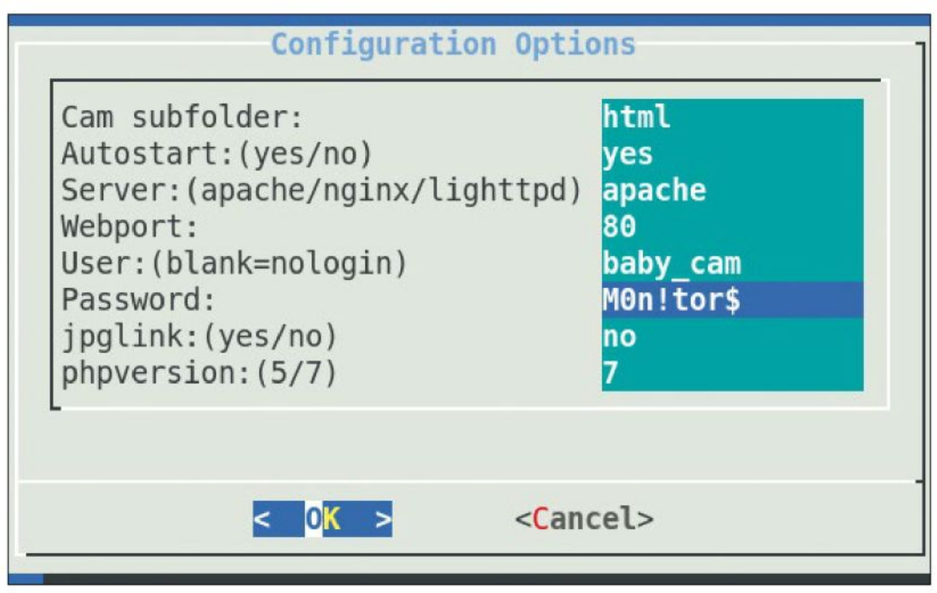
```
annotation %Y.%M.%D_%h:%m:%s %a
```

The last attribute, `%a`, enables the streaming software to read the contents of the file **/dev/shm/mjpeg/user_annotate.txt** and display it on the screen. The **measure_temp_loop2.py** scripts writes the temperature value discovered to this file. After making the configuration edits, stop and restart the streaming software. The temperature reading is the final digits displayed at the end of the text at the top of the screen.

Security

Securely transporting data is something users purchasing commercial products should be aware of. Many IoT offering fall short of security functionality that ensures data security. Security in do-it-yourself projects becomes the builder's responsibility. Let's look at a few things we can do to tighten up security.

Streaming the video content over Wi-Fi to a local area network (LAN) has security risks. The camera content is only intended for consumption within a local area network. Strong security practices need to be in place on the LAN to ensure the content is protected.



Change the defaults for the user and password.

A cool 22.3 degrees Celsius, which is the perfect temperature for reading your copy of Linux Format in the dark.

Installing SSL

Data sent using the HTTP protocol isn't secure. A more secure protocol is HTTPS – it uses SSL, a security protocol for establishing secure connections between clients and servers over a network. SSL uses a certificate for authentication and data encryption. We want to secure the transmitted video data between the browser and video streaming server (the Raspberry Pi) using encryption provided by SSL.

To take advantage of SSL we're going to create a self-signed certificate. The certificate is issued by the same person/company for themselves and doesn't involve purchasing a public SSL certificate from a provider. Most browsers will display a security exception alert when they encounter a self-signed certificate, because the browser is unable to verify the certificate with an external entity. Accepting the security exception notification created by the browser resolves this issue. You can install the self-signed certificate in the browser, but that requires more writing space than provided.

Let's generate a key file and use it to create a self-signed certificate:

```
sudo openssl genrsa -out /etc/ssl/private/webserver.key 2048
sudo chmod 400 /etc/ssl/private/webserver.key
sudo openssl req -new -x509 -out /etc/ssl/certs/webserver.pem -key /etc/ssl/private/webserver.key
```

A number of questions will be asked to complete the process of creating the certificate. To examine your certificate command handiwork you need to issue the following command:

```
openssl x509 -text -noout -in /etc/ssl/certs/webserver.pem
```

Now we need to enable SSL on the apache webserver by setting up the supporting configuration files. Let's start by preserving the web configuration file created for HTTP web service:

```
sudo cp /etc/apache2/sites-available/raspicam.conf /etc/apache2/sites-available/raspicam80.conf
```

Next, establish a base SSL configuration file using some default files provided by apache:

```
sudo cp /etc/apache2/sites-available/default-ssl.conf /etc/apache2/sites-available/raspicam443.conf/
```

» CREDITS FOR THOSE DUE

While the project fell short of its initial aims, that doesn't reduce the need to give credit to the resources that enabled the project. Sparkfun's assembly guide of the camera kit (**https://learn.sparkfun.com/tutorials/setting-up-the-pi-zero-wireless-pan-tilt-camera**) was the initial starting point. It establishes a working camera using the supplies provided in the kit. Second was the RPi Cam Web Interface reference material at **https://elinux.org/RPi-Cam-Web-Interface**, for exploring the workings of the camera application software. Third, the Github pan title resource at **https://github.com/mhord/RPi_PanTilt_Cam**, which provided the application. This link (**https://kamranicus.com/guides/raspberry-pi-3-baby-monitor**) was a resource found during an internet search examining a completely different solution for streaming video. The author introduced ideas that resulted in improvements to this design.

Using a text editor, make the following modification to the HTTPS configuration file **raspicam433.conf**:

Replace the follwing lines:

```
<IfModule mod_ssl.c>
<VirtualHost _default_:443>
        ServerAdmin webmaster@localhost
        DocumentRoot /var/www/html
```

With these lines:

```
<IfModule mod_ssl.c>
ServerName kenzie.inane.ca
        <VirtualHost *:443>
                ServerAdmin webmaster@localhost
                DocumentRoot /var/www
                <Directory /var/www/html>
                        Options Indexes
FollowSymLinks MultiViews
                        AllowOverride All
                        Order allow,deny
                        Allow from all
                </Directory>
                ScriptAlias /cgi-bin/ /usr/lib/cgi-bin/
                <Directory "/usr/lib/cgi-bin">
                        AllowOverride All
                        Options +ExecCGI
-MultiViews +SymLinksIfOwnerMatch
                        Order allow,deny
                        Allow from all
                </Directory>
```

Locate these lines:

```
#SSLCertificateFile        /etc/ssl/certs/ssl-cert-
snakeoil.pem
#SSLCertificateKeyFile /etc/ssl/private/ssl-cert-
snakeoil.key
```

Add the following lines below:

```
SSLCertificateFile /etc/ssl/certs/webserver.pem
SSLCertificateKeyFile /etc/ssl/private/webserver.key
```

Execute the following command to enable apache to use the SSL module:

```
sudo a2enmod ssl
```

Enabling apache to use HTTP (`raspicam80.conf`) or HTTPS (`raspicam443.conf`) service is simply a matter of copying the appropriate configuration file to the apache configuration operational file **/etc/apache2/sites-available/raspicam.conf**. After putting the HTTPS file in play, restart the apache service:

```
sudo systemctl restart apache2.service
```

A good security strategy is to turn off unused services. Vendor OS enhancements make changes that may or may not impact a service. You might find one day you can turn off a service, but after an upgrade the service may be re-enabled because it's required.

The list of services turned off in this project are based on the specific environment and the author's experience. Bluetooth is turned off in this project but if the reader is using a Bluetooth device, such as a keyboard, then this isn't practical.

The command `netstat -plut` shows a list of ports that are providing services. When the services are shut down, run the command again to see the cleanup results:

```
sudo service --status-all
sudo systemctl disable avahi-daemon
sudo systemctl disable bluetooth
sudo systemctl service rsync disable
sudo systemctl service rsyslog disable
sudo systemctl disable rsyslog
sudo systemctl disable motion
```

Recall we can stream video using HTTP and/or HTTPS. Since there are no plans to stream video through HTTP, let's disable the port so it's not supported in apache. Edit the **/etc/apache2/ports.conf** file and comment out support for port 80 by prefixing the line with an octothorp character (to normal humans a #). Apache web services will need to be restarted for this change to take effect.

Removing plain text passwords

The password for the Wi-Fi connection is stored in plain text in the **/etc/wpa_supplicant/wpa_supplicant.conf** file. It could be argued this is not a security risk since the file is only readable by root. Storing password in plain text is frowned upon in the information technology security community. To eliminate the storage of the plain text password execute the command from a CLI:

```
wpa_passphrase "your wifi SSID" "your wifi password"
```

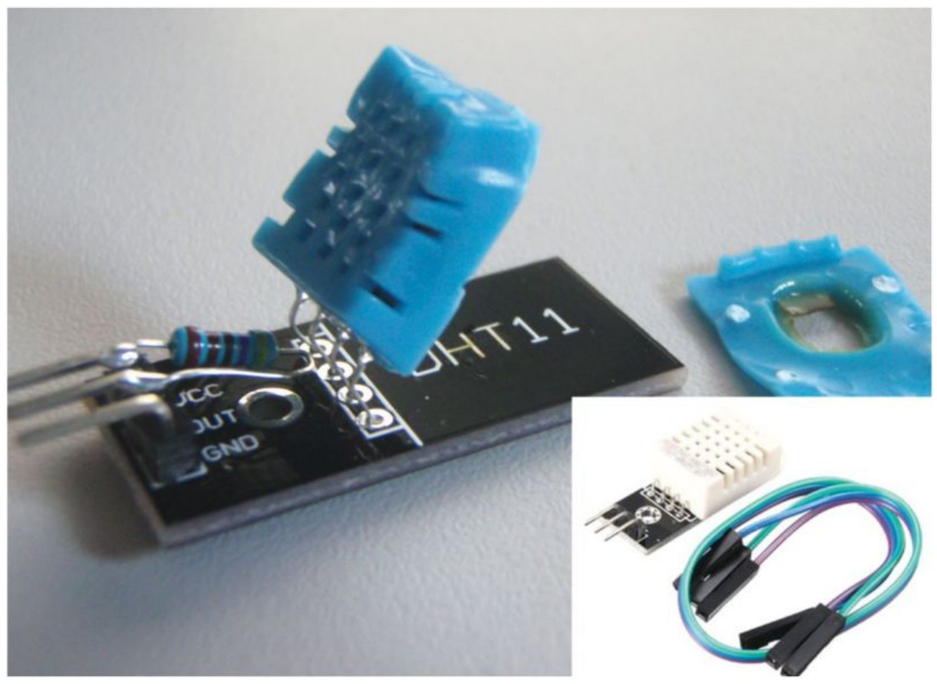
Copy and paste the encrypted PSK text generated into the **wpa_supplicant.conf** file.

```
network={
  ssid="your wifi SSID"
  #psk="your wifi password" remove this entry after
confirming operation.
  psk=131e1e221f6e06e3911a2d11ff2fac9182665c004de
85300f9cac208a6a80531
}
```

The *wpa_passphrase* utility generates an encrypted PSK value that can replace the plain text PSK value. The CLI passphrase tool takes the SSID and the password, and generates the encrypted PSK output. The password can be configured in the **wpa_supplicant.conf** file either as the ASCII representation by enclosing the password in quotes or as a pre-encrypted 32 byte hexadecimal number without quotes.

Final thoughts

You may recall when the project was first suggested there was no clear list of requirements. The lack of audio stream support resulted in the installation being rejected by the grand-daughter's parents. Their commercial monitor provided audio and video. The author failed to include audio support as a design requirement. It was also suggested once the camera platform was mounted to the Universal Gooseneck Phone Stand Holder it was "menacing looking". With the hiss of the servo motors and the dull glow of IR LEDs the "camera creature", as it was referred to, didn't have a place in a kid's bedroom. Who knew aesthetics was a design requirement?



Please note that there's not supposed to be a brown burn hole in the plastic.

QUICK TIP

Default credentials are a high-value attack vector for nefarious individuals that are looking to compromise a system.

Velleman VM205

Les Pounder takes a look at a simple oscilloscope that offers to resurrect that Raspberry Pi sitting in your project box.

IN BRIEF

An introductory oscilloscope and logic analyser for the beginner and hobbyist who needs to see how a project works, but doesn't want to spend a fortune. Designed for the original Raspberry Pi, but compatible with all models, this is a simple digital scope that offers enough to interest you, and possibly lead to a more 'serious' scope. See also the Bitscope review in **LXF194**.

Velleman is well known in the maker world. It sells kits from simple flashing LEDs to dual-extruder 3D printers, and also this oscilloscope shield for the Raspberry Pi.

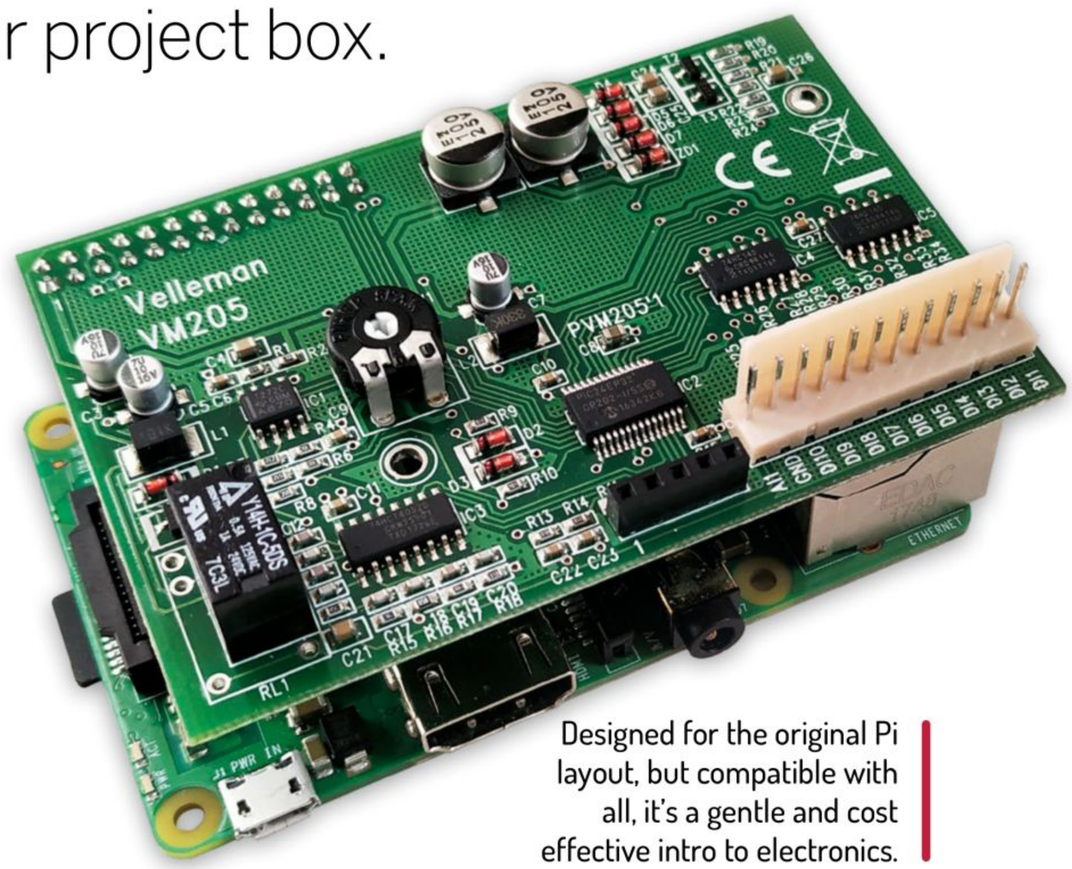
Originally designed for the first model of Pi, with 26 GPIO pins, the Velleman VM205 attaches to all models – but for 40-pin models you need to ensure that the shield does not touch any components, as the included standoffs only match the original Pi layout. The input connections are made via a breakout connector that provides a series of single-core wires which can be inserted into breadboards or screw terminals. We used some Wago 221 connectors to lock the thin wires in place, and then make connections with jumper jerky wires. With the shield attached to the Pi, power up and then set up the SPI interface so that the Pi and shield can communicate. The included instruction book is sparse and out of date, but for the target audience there is enough for the maker to adapt and reference.

Using the VM205 is managed via custom software. The interface is ugly but simple, which enables us to focus on the job. The VM205 has 10 logic analyser channels that means we can peek into 10 components in a circuit, and each of the channels are graphed in the bottom section of the chart. Here we can 'see' how a chip works. For example, we connected our Aneng AN8008 multimeter and used the built-in frequency generator to create a series of waveforms from 50Hz to 5kHz. All of these waveforms were visualised successfully. We then connected a voltage to the VM205 and were able to measure the voltage change as we altered the input voltage. This leads nicely into the input voltage range, which is up to 30V thanks to a part of the circuit that enables the use of such high voltage while protecting the Raspberry Pi's fragile GPIO.

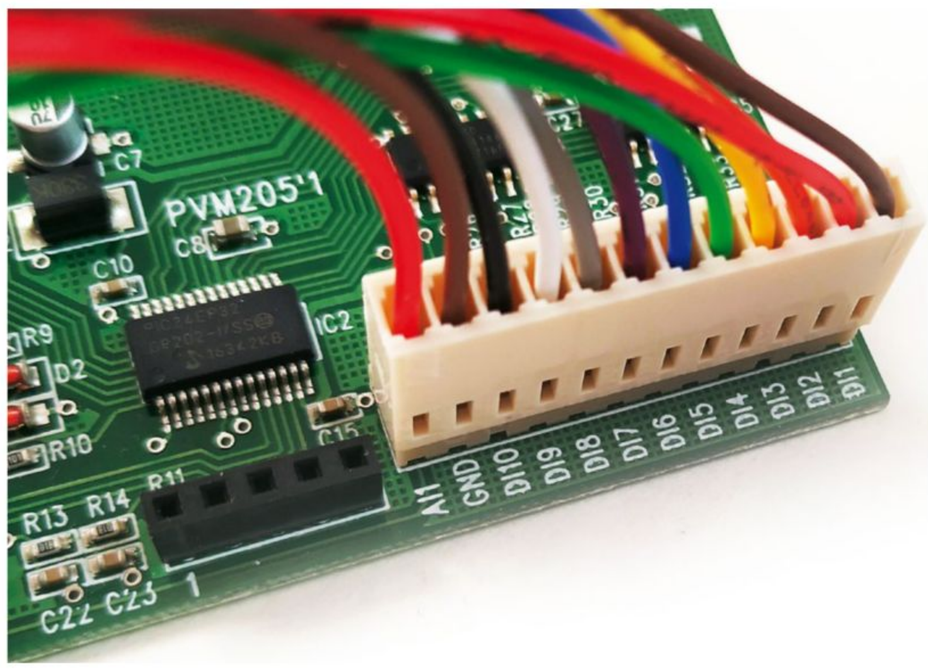
So what can this board detect? Well, it has a sample rate of 1MS/s (megasamples), which is rather low when compared to professional scopes that offer GS/s (gigasamples) – but it's good enough for the price of this unit. The VM205 can measure up to 100kHz signals and can work with AC and DC power. It can record 800 samples and features a simple 8-bit analogue-to-digital converter. So what does that all mean? That this is not a professional oscilloscope – but we kind of guessed that by the price. It's brilliant if you accept that it is a first step into this technology. It can be used to diagnose issues with your projects and other electronic devices – just make sure that you are careful with the voltages!

Oscilloscopes and logic analysers are handy things, but they are expensive and somewhat large. So dipping your toes into the water with the VM205 should prove enlightening and possibly inform purchasing decisions for a hobbyist-grade scope. What we really like about the Velleman VM205 is that it can be used with the older Raspberry Pi models, and keen hackers could quite easily add a small screen and combine the scope into a nice professional package.

This is a fun, gentle and cost-effective introduction to oscilloscopes for new users. For sheer value for money this is a great scope that also revives that old Pi 1 you have in the project box.



Designed for the original Pi layout, but compatible with all, it's a gentle and cost effective intro to electronics.



The Velleman VM205 enables input up to a whopping 30V.

VERDICT

DEVELOPER: Velleman
WEB: www.velleman.eu/products/view/?id=418968
PRICE: £33.59

FEATURES	7/10	EASE OF USE	7/10
PERFORMANCE	7/10	VALUE	9/10

Delightfully simple and easy. It enables the learner to get hands-on with a cheap scope, without a huge price. Not for professionals, but still rather handy for hobbyists.

» **Rating 8/10**

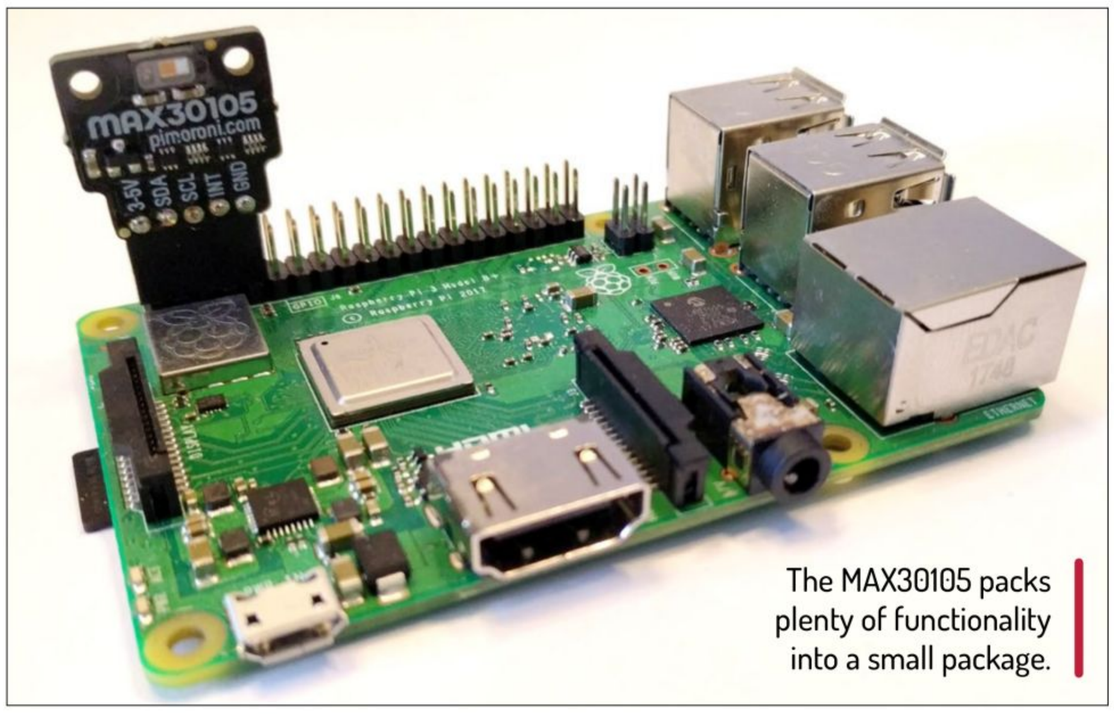
MAX30105 Multi Sensor

Nothing gets **Les Pounder**'s heart racing like a new toy. But how can we monitor his excitement? For that, we need the MAX30105!

IN BRIEF

A tiny sensor that can be used to monitor your heartbeat, blood oxygen level and detect particulate matter from smoke, thanks to an easy to use Python library. Designed to work with Pimoroni's Breakout Garden HAT, but compatible with all models of Pi thanks to an I2C interface, it can also be used with the Arduino.

When you want to collect sensor data, the chances are you'll consider using a Raspberry Pi – largely due to its ease of use and network connectivity. But using sensors can be quite difficult for those new to the idea. Pimoroni released its Breakout Garden HAT board, which we reviewed in **Linux Format issue 243**, to enable anyone to connect a sensor and quickly use it. The MAX30105 board is from the same range, but it doesn't need the HAT board and can be directly connected to the Raspberry Pi's GPIO pins.



The MAX30105 packs plenty of functionality into a small package.

But what does this sensor detect? Well, in a such a small package we have sensors for heart rate, oxygen saturation (an oximeter) and a smoke sensor which measures particulates in the atmosphere. These readings are also reasonably precise at 18-bit accuracy.

The MAX30105 comes as a simple board designed to interface with the Breakout Garden HAT, but by soldering the included male/female header pins we can easily use the board with the standard Pi GPIO. Software installation is remarkably simple thanks to a Python 3 package in the *pip* package manager – this handles installation of the board and any dependencies. Pimoroni's example scripts are available from its GitHub repository.

In use the MAX30105 is pleasantly easy, and within a few minutes we had a test script to monitor our pulse – we're glad to say we averaged around 67 BPM at rest. We then tested the smoke sensor with a little solder smoke, which was easily detected. We also tested the MAX30105 with an Arduino Uno, thanks to a built-in library for the board made accessible via the Manage Libraries dialogue. The included Arduino examples are easy to understand. In fact, pairing this sensor with an Arduino MKR1000 or 1010, which feature Wi-Fi, would enable real-time sensor readings to be sent to remote devices.

So who is this board for? Well, given its simplicity it would be right at home in schools and in the education sector. It provides the ease of use and immediacy that students need to keep their interest. The mixture of sensors provides plenty of options for investigating, monitoring and recording data when used with other Python libraries. That said, the MAX30105 is not a serious medical or life-saving device, so its use in any project that depends on this functionality should be avoided. But for investigating the possibilities of using sensors such as this, the MAX30105 is an entry-level component that serves the learner well.

The MAX30105 is not a bespoke Pimoroni creation, but the form factor is designed to match the company's Breakout Garden. The ability to use the MAX30105 with both the Raspberry Pi and Arduino is fantastic and it means that educators, or those on a budget, can use the same component across platforms. In fact, if you are skilled with I2C you could easily use the MAX30105 with other I2C-enabled devices such as the micro:bit.

In all, this is a cost-effective and feature-packed board that will spark the imagination of those eager to learn. We can now easily monitor the pulse and oxygen saturation of a test subject, and use that data in experiments and projects. All we need are a few lines of code and an idea – plus a willing, uh, volunteer.

You can attach the sensor directly, as here, or use the Breakout Garden HAT board.

VERDICT

DEVELOPER: Pimoroni
WEB: http://bit.ly/lxf250max
PRICE: £13.50

FEATURES	9/10	EASE OF USE	9/10
PERFORMANCE	9/10	VALUE	9/10

The MAX30105's ease of use enables this board to be a great entry to using sensors with your Raspberry Pi or Arduino.

» **Rating 9/10**

Get and use API access to monitor train times

Les Pounder shows you how to use a web API to retrieve data and then use it at the heart of your own monitoring projects, this one train times!

OUR EXPERT

Les Pounder is a freelance maker who works with organisations such as the Raspberry Pi Foundation to promote maker skills.

The Application Programming Interface (API) is an important part of coding. We can use these to power data-driven projects such as displaying weather and news, and in **Linux Format issue 251,** we used one to track the International Space Station. But learning how to use an API can be tricky, so in this project we'll see how to use the free Transport API to create an application that tells us all about departing train times, platforms and operators – using a voice powered by Google's Text to Speech service.

Getting an API key

To use the Transport API service we need to sign up for a free account at **https://developer.transportapi.com.** Write down your App ID and Key; these are personal to you and identify that you are using the service. Keep them safe and do not share them. In order to use the keys in our code, open the Mu Python editor and save a file called **secrets.py**. In this file we'll save two variables, our App ID and Key.

```
app_id = "Your APP ID Goes here"
api_key = "Your Key goes here"
```

Save the file before moving on.

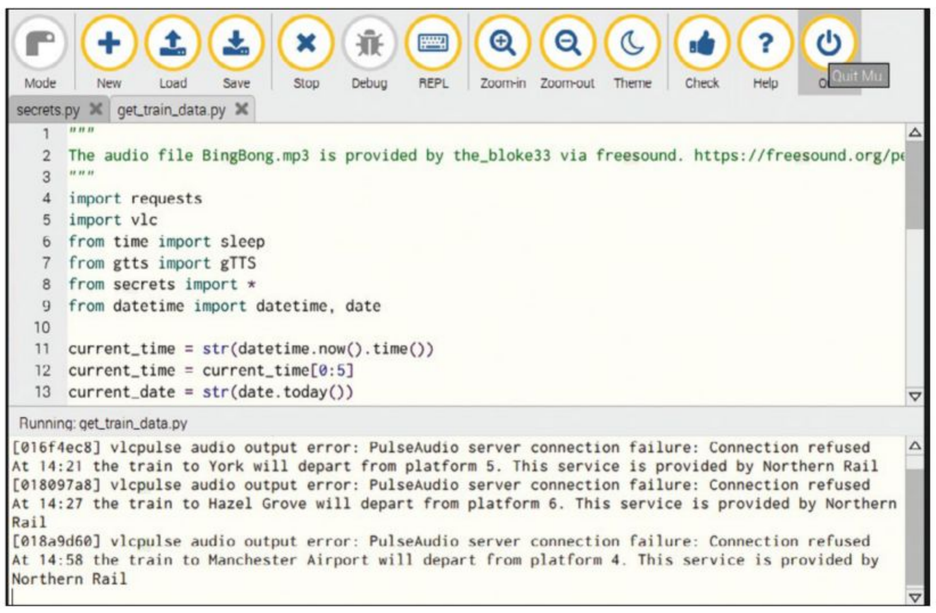
We need to install two Python libraries for this project: gTTs for Google Text to Speech, and vlc, which we use for media playback. To install, open a terminal and type:

```
$ sudo pip3 install gtts python-vlc
```

Using the *Mu* Python editor we create a new file, and then save it as **get_train_data.py**. Remember to save this often!

Our code starts by importing a series of libraries which provide extra features to our code. The first is called `requests`, which enables us to receive and send data over HTTP. We `import vlc` to play the audio using the VLC player, now installed as standard on Raspbian. Then we import the `sleep` function from the `time` library, the `gTTS` class from the `gtts` library and a library called `secrets`, which we created earlier to contain our API details. Lastly we import `datetime` and `date` from the `datetime` library to get the time and date.

```
import requests
import vlc
from time import sleep
from gtts import gTTS
from secrets import *
from datetime import datetime, date
```



You may see errors in the Python shell, but fear not – these will not break the project but act as warnings for audio playback on Raspbian.

```
from secrets import *
from datetime import datetime, date
```

To get the current time and date for our train details, we create two variables, `current_time` and `current_date`. To get the current time we use the datetime library and convert the time to a string. But the returned time is too precise, going down to fractions of a second. So in the second line below we use string slicing to extract the hours and minutes and store them in the variable.

```
current_time = str(datetime.now().time())
current_time = current_time[0:5]
```

To get the current date in YYYY-MM-DD format, we call the date class and ask it for today's date, saving this to the variable.

```
current_date = str(date.today())
```

The last variable in this section is our train station code. This is the code used by rail companies to identify a station. You can find your station via National Rail Enquiries at **www.nationalrail.co.uk/stations_destinations/default.aspx**; here we're using Blackpool North (BPN).

```
station = "BPN"
```

In order to get the train data from TransportAPI, we need to use a carefully crafted web address (`url`) that will get the trains from our station at the current time and date. It will also use our app ID and key. You will see in the URL that there are places where we automatically insert this data from variables in our code.

YOU NEED

- **Any model of Raspberry Pi**
- **The latest Raspbian release**
- **Internet connection**
- **Code: http://bit.ly/lxf253tts**

```
url = "https://transportapi.com/v3/uk/train/station/"+station+"/"+current_date+"/"+current_time+"/timetable.json?app_id="+app_id+"&app_key="+api_key+"&train_status=passenger"
```

With the URL ready we can now use requests to get the data from the URL and save it as an object called `r`.

```
r = requests.get(url)
```

The next line checks for the correct response in the `r` object. If it exists the code will run, else it will fail and skip the code and go to the end. But in this case the code works, so the next three lines trigger the VLC media player, loading the audio file **bingbong.mp3** for train station chimes. It plays the file and then pauses for 4.2 seconds (the length of the audio).

```
if r :
    media = vlc.MediaPlayer("bingbong.mp3")
    media.play()
    sleep(4.2)
```

Moving into more complex code, we use a `for` loop to iterate over every train departure from our chosen station. For this we use `len()` to find the number of trains leaving from the returned data. As the returned data is in JSON format, Python treats it just like a dictionary, so we tell Python to look for the `'departures'`, `'all'` keys as our source.

```
    for i in range(len(r.json()['departures']['all'])):
```

Inside the `for` loop we have four lines of code that look rather intimidating, so let's go through them step by step. The first line sets the destination of our train. In the returned JSON data we have a number of keys that enable us to get values for specific pieces of information. So we ask for all departures; the `[i]` refers to our `for` loop which keeps looping until every departure has been listed. The last key is the destination for that train. So using these keys we get the exact station that our train is heading to.

```
        destination = (r.json()['departures']['all'][i]['destination_name'])
```

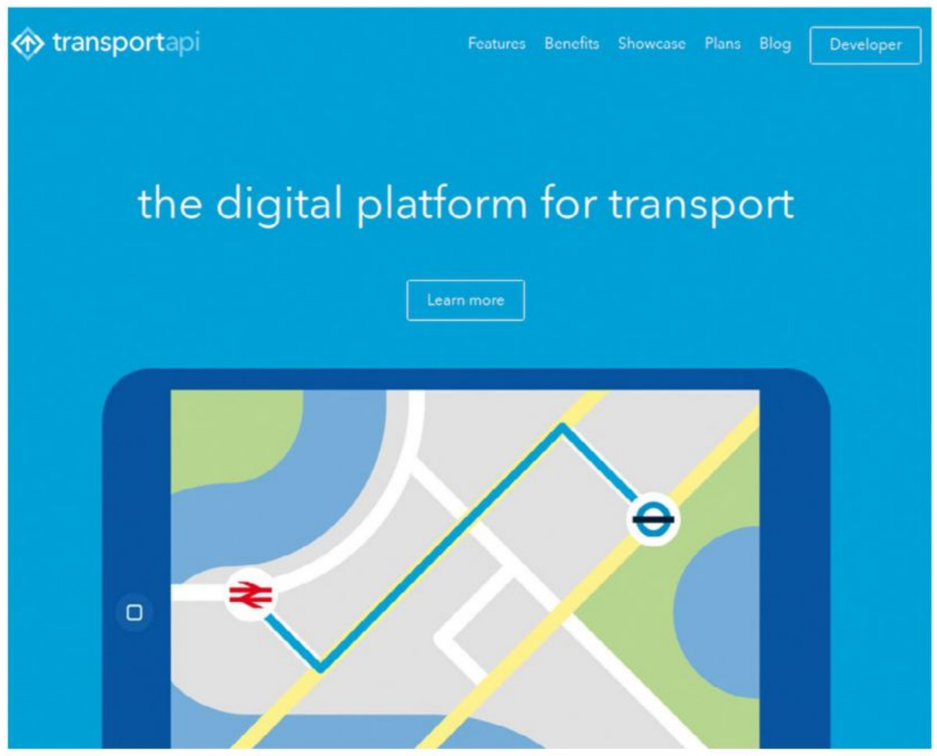
The next three lines repeat this pattern, but the final key covers the departure time, platform and who the operator is. This data is then saved to a corresponding variable for later use.

```
        departure = (r.json()['departures']['all'][i]['aimed_departure_time'])
        platform = (r.json()['departures']['all'][i]['platform'])
        operator_name = (r.json()['departures']['all'][i]['operator_name'])
```

So what do we do with this data? Well first, for debug purposes we print the data to the screen. This is one really long sentence made up of written sections and our variables (`departure`, `destination`, `platform`, `operator_name`) joined together.

```
        print("At "+departure+" the train to "+destination+" will depart from platform "+platform+". This service is provided by "+operator_name)
```

We now create an object called `tts` and inside of this we instruct gTTS to convert the long sentence into speech. At the end of the line we can choose the speech language. Here we have chosen American English (`en-us`) as it sounds more natural.



The Transport API is a great one with which to start learning, and it can provide data for many projects.

QUICK TIP

The URL used in this project can be seen in full via a web browser. Copy the URL into a browser and replace the time, date, station, app_id and app key to match what you are looking for. For example http://bit.ly/lxf253api (would) show the trains that were due to leave Blackpool at 21:00 on 7 June.

```
        tts = gTTS("At "+departure+" the train to "+destination+" will depart from platform "+platform+". This service is provided by "+operator_name, lang="en-us")
```

To save the speech we instruct `tts` to save as **announce.mp3** in the same directory as our code.

```
        tts.save("announce.mp3")
```

To play the audio we use VLC in the same way as the chimes earlier – but this time we add a 10-second delay so that there is enough time for the audio to play.

```
        media = vlc.MediaPlayer("announce.mp3")
        media.play()
        sleep(10)
```

The last section of code is our `else` condition. This is the code which runs if there is no data to read. It will just print that there has been an error.

```
    else:
        print("There has been an error")
```

Great work! In this project we have learned how to use a web API to provide data for an application.

» WHY WAS JSON BORN?

JavaScript Object Notation (JSON) is a lightweight data-interchange format. In other words, it's a really simple way to publish data that can be read by many modern programming languages. JSON is structured using a series of brackets – for example:

```
{"Welcome": "Hi LXF Reader","Tutorial": "Enjoy the tutorial"}
```

Inside this JSON object is a string called "Welcome" and a value. This JSON data can be on a remote server, as it is in this tutorial, or it can be on our Pi. But we need a way to interact with the data, and for that we can use the pre-installed JSON library for Python. Let's say we have a file called **boxout.json** and it contains the above JSON example. We import the JSON library and then open the file, read the contents to a variable and then dump the data to the Python shell.

```
import json
with open('boxout.json') as f:
    data = json.load(f)
    print(json.dumps(data, indent=4)
```

Running this code shows the JSON data. Python has a similar data structure to JSON, called a dictionary. More details on dictionaries at **http://bit.ly/lxf253dictionary** [1] and you can read much more about JSON at **https://docs.python.org/3/library/json.html**.

Activate a fan with motion control

Les Pounder shows how to safely control higher voltages with a Raspberry Pi and two very cheap electronic components.

OUR EXPERT

Les Pounder is a freelance maker who works with organisations such as the Raspberry Pi Foundation. www.bigl.es

YOU NEED

- **Any Pi**
- **Raspbian**
- **Obstacle sensor**
- **2n2222 NPN transistor**
- **Breadboard**
- **1k Ohm resistor**
- **2x two-screw terminals**
- **9V battery and connector**
- **Wago 221 connector**
- **5x female-to male jumper**
- **6x female to female jumper**
- **Relay board**
- **PC case fan**
- **Code: https://github.com/lesp/LXF248/archive/master.zip**

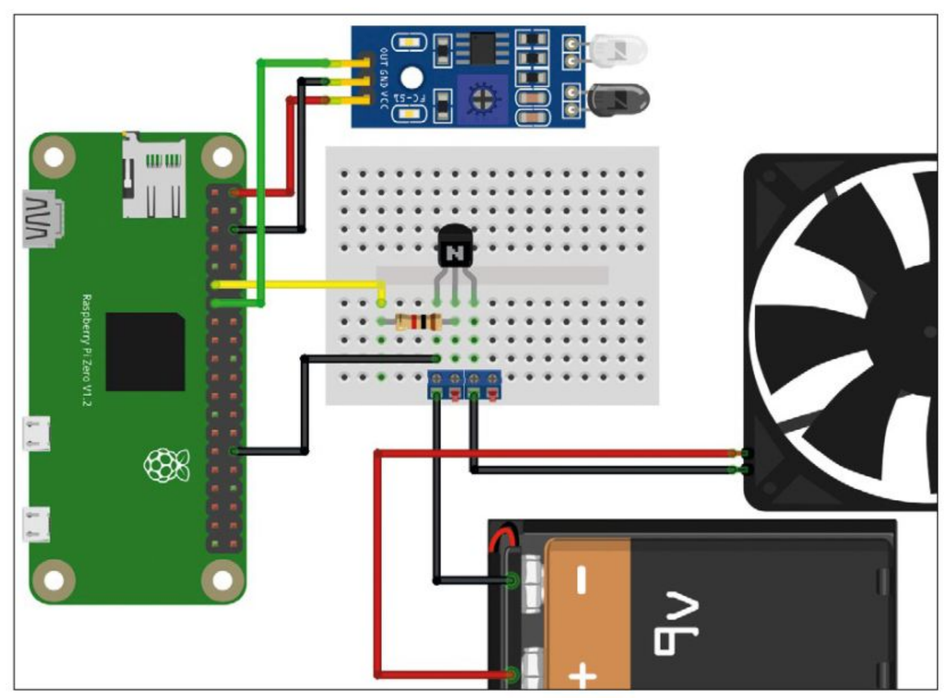
For this project, we'll see how to control higher voltages with our Pi using two different 'switches'. The first is a small component called a transistor, specifically an NPN 2n2222 transistor that is safe to use up to 30V. The second component is a relay which can handle much higher voltages. But no matter what, always be sure to stay safe when working with higher voltages. You may just damage your Pi, but worse still, you might also hurt or kill yourself.

Project 1: NPN transistor

An NPN transistor is basically a switch, and we can use it as one with our Raspberry Pi. Looking at our 2n2222 transistor with the flat side facing us, we see three pins. Pin 1 is the emitter, pin 2 the base and pin 3 is the collector. Our 9V circuit has a broken GND connection between the collector and the emitter – so when the transistor is not activated, the connection remains broken. But when we apply a voltage from our Raspberry Pi GPIO to the base pin of the transistor, we can turn on the transistor and force the switch to close the connection, thus allowing the current to pass from the collector to the emitter. In this project we'll use an obstacle sensor (**https://shop.4tronix.co.uk/products/ir-infrared-obstacle-sensor**), which is really a short-range infrared sensor, to act as a trigger that will cause a fan to turn on.

The hardware build for this project is not too difficult. Consult the diagram and take your time. Try to break the build down into sections, and check your wiring. Your Raspberry Pi will not survive 9V connected to its GPIO, so double-check before connecting. You will notice in the diagram that the 9V + connection from the battery to the fan is directly connected. In the real world this can be achieved by twisting the wires together, or by using a Wago 221 two-port connector (around 50p) to clamp the wires in place. High-resolution diagrams are in the download for this project.

Power up the Raspberry Pi and from the desktop go to the main menu, then the Programming menu and select Python 3 Editor. Once the editor opens, immediately click File > New and from the new window click File > Save. Call the **project fan-control.py** and click Save. Subsequent saves will be much quicker, and remember to save often.



The 2n2222 NPN transistor is tiny, but can control much higher voltages than can be safely controlled with the Raspberry Pi's GPIO.

There is very little code for this project and this is because we are using the *GPIO Zero* library created by Ben Nuttall, Community Manager at the Raspberry Pi Foundation. *GPIO Zero* abstracts away a lot of the complexities of using the GPIO and this means that newcomers can just get started writing code. This enables us to create a gesture-controlled fan in five lines of Python code.

The first line handles importing two classes from the *GPIO Zero* library. These are *DigitalOutputDevice* and *MotionSensor*. The former is a class which enables us to create a new output device, in this case our fan, and control it exactly how we want to. The latter is used with our obstacle sensor; it reads the status of the output pin on the sensor and react to how we program it.

```
from gpiozero import DigitalOutputDevice,
MotionSensor
```

Our connection to the fan is made from GPIO 17 on the Raspberry Pi, and this connects to the base of our 2n2222 transistor via a 1K Ohm resistor. This is present to protect the GPIO from the 2n2222, which may pull a little too much current and damage the pin.

In order to use this connection we need to tell the code that it exists, so we create an object called **fan** and use the *DigitalOutputDevice* class to specify that we are using GPIO17:

```
fan = DigitalOutputDevice(17)
```

Next we create another object for our obstacle sensor, a cheap and commonly used sensor in robotics that ensures a robot doesn't drive into a wall. For this sensor we use the *MotionSensor* class, as this works identically to a PIR (Passive Infrared) sensor.

```
sensor = MotionSensor(27)
```

Now we create a loop that will constantly run our code. For this we use the **while True** loop.

```
while True:
```

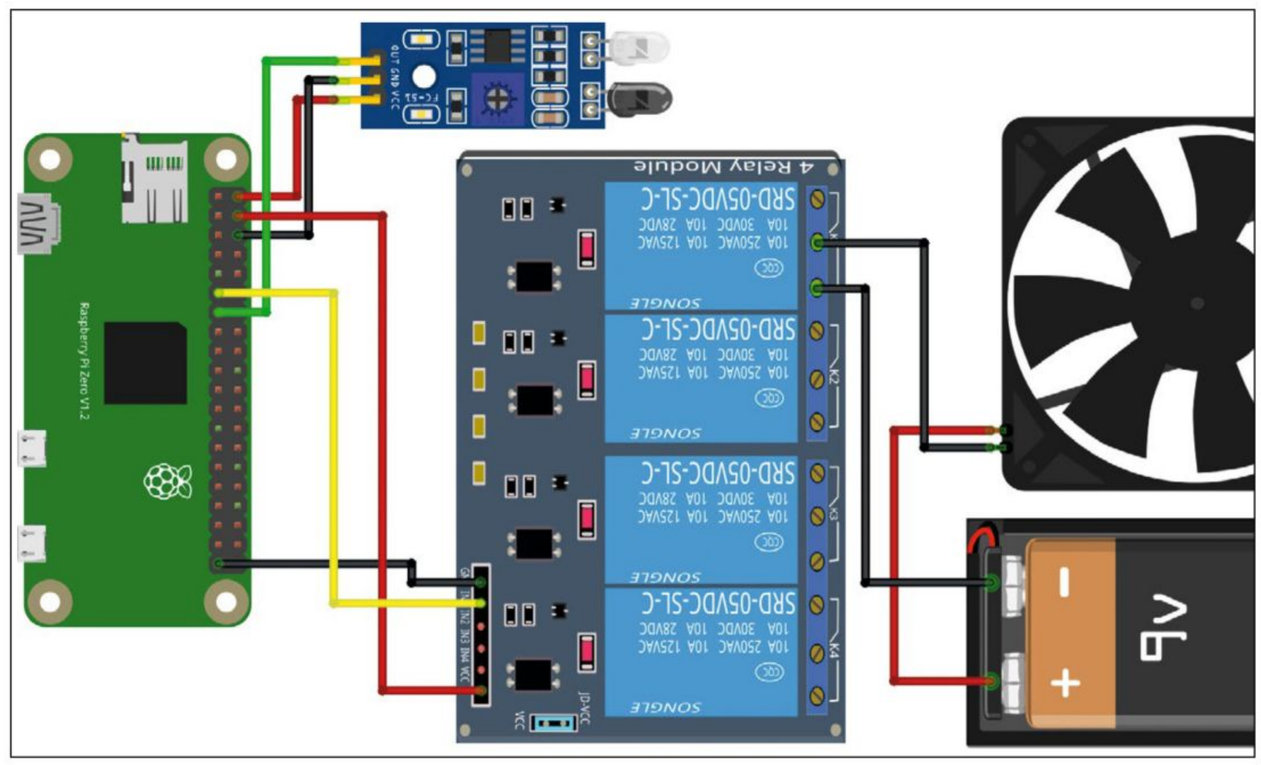
The final line of code is indented four spaces – one Tab key press – automatically by the editor. Indentation in Python can mean many things, but in this case it means that our code belongs inside the loop and should be run when the loop is run. The code that is run checks the status of the obstacle sensor, to which we have created a link using the **sensor** object. When the sensor detects motion the output pin will change from low to high, and this change of state is what the GPIO pin on our Pi is looking for.

The change of state will trigger our code to call the fan object with the **toggle** attribute. In other words, it turns GPIO17 on or off depending on its current state. This toggle will then toggle the base pin of the 2n2222 transistor on or off, and therefore close/open the circuit, enabling the 9V circuit to operate and power our fan.

```
while True:
    sensor.when_motion = fan.toggle
```

Save the code and when you're ready click Run > Run Module to start. Now wave your hand in front of the sensor and your fan should come to life. You may need to tweak the sensitivity of the sensor using the potentiometer on it; you'll need a flat-blade screwdriver for this.

So now we can control a fan using a transistor, but sometimes we need a switch that can handle a higher voltage and current. For that we need a MOSFET or a relay. Relays are mechanical switches that can handle much higher voltages than our NPN transistors (see 'Working with higher voltages', *right*) but they should be used with great care.



Relays enable us to use much higher voltages with our Raspberry Pi, but we must always keep safe and have the knowledge to work with high voltages.

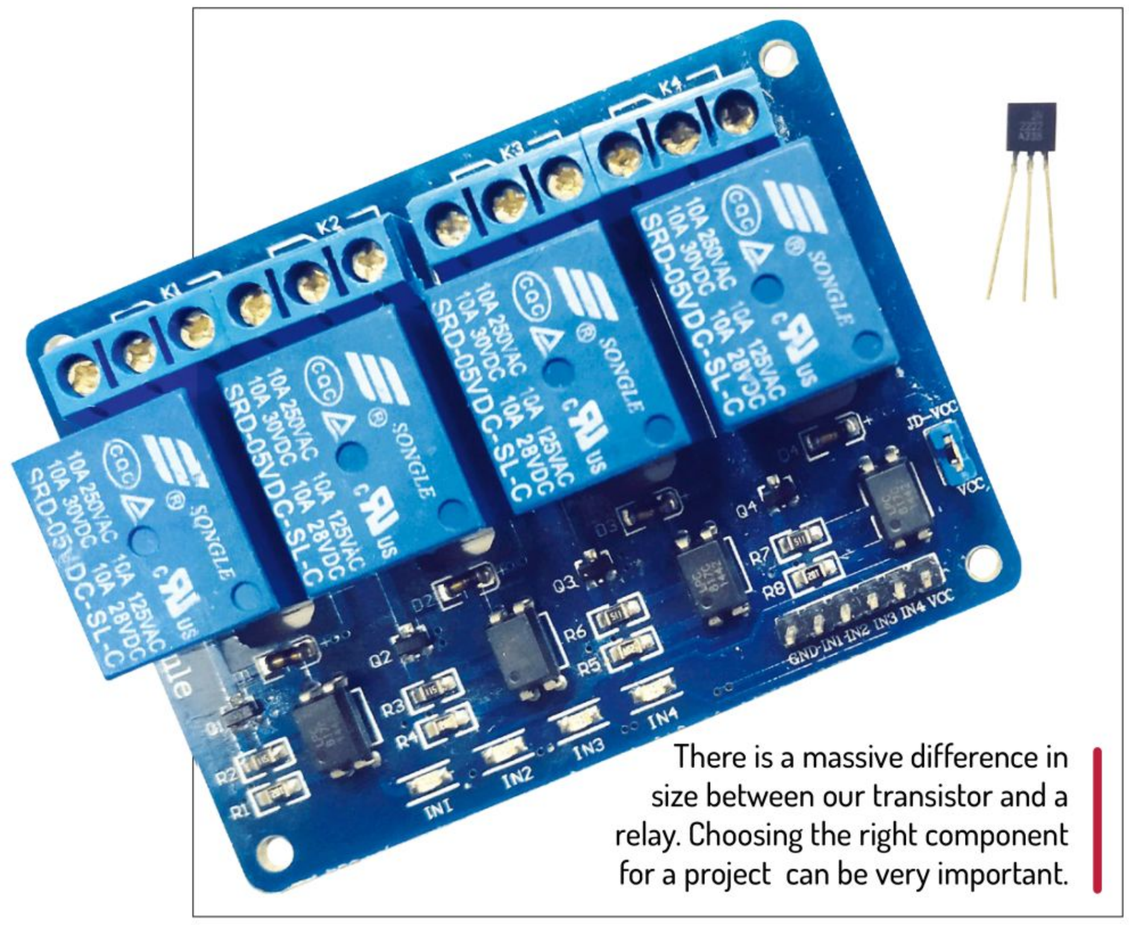
Project 2: Using a relay

To replace our NPN transistor with a relay requires that we alter the circuit. Essentially we remove the breadboard, and using the female to female jumper jerky we connect the 5V and GND pins from the Pi to the VCC and GND pins on the relay board. Then using another jumper jerky, connect GPIO17 to IN on the relay. If you're using a multi-channel relay, ensure that the IN pin used matches the relay being controlled. In our diagram we used IN1 and relay 1.

On the other side of the relay we have the 'high voltage' connections. For our connection we need to connect the two wires from the battery GND and the fan GND so that they are in the screw terminals for Normally Open; these can be identified via the diagram on the relay, or by the letters NO. Connect the wires and start the code once more. You should now hear a click and see/feel the fan start up.

QUICK TIP

Don't buy a fan just for this project! Chances are you will have a spare PC case fan. Snip off the Molex connector, then strip the red and black wires to show 1cm of bare wire. Touch the red wire to + and black to – on a 9V (PP3) battery to test polarity.



There is a massive difference in size between our transistor and a relay. Choosing the right component for a project can be very important.

» WORKING WITH HIGHER VOLTAGES

First, let's make this clear. Only work with high voltages if you know exactly what you are doing as you can be seriously injured or killed!

In this tutorial we worked with higher voltages, in that we used a 9V circuit to power the fan but controlled the circuit using the 3.3V Raspberry Pi GPIO. But how did we know what to use as our switch? Well, we consulted a datasheet.

Datasheets are documentation provided by a company that manufacture components. To find a datasheet for the 2n2222 NPN transistor, we used a search engine and located an appropriate document. Then with a coffee we sat down and read about the voltages and currents that our transistor could tolerate.

We learnt that the emitter could be used with up to 30V and that it could supply up to 800mA. Our fan only draws around 72mA when in use, peaking at 120mA when it starts, so we were well within the safe operation region. If we went over 30V 800mA then we would need something a little beefier, and for that we could use a MOSFET which are designed for much higher currents. Our power source could have beed replaced with a 12V DC source in that case, but we chose to use a relay which can handle up to 250V AC at 10A.

Build and configure a home security system

Les Pounder shows us how to set up our own private security camera, enabling you to monitor an area that you can view on any device.

OUR EXPERT

Les Pounder is a freelance maker who works with organisations such as the Raspberry Pi Foundation to promote maker skills. Read his blog at bigl.es.

Here, we shall use *Pushover* (a notification service for all devices) along with the humble Raspberry Pi to create an intrusion system that will detect movement using an infrared sensor and take a picture of the potential intruder using the official Raspberry Pi camera. And all of this will then be sent to all of our devices using the *Pushover* software.

Our first step in building the hardware is to connect the official camera to the Camera port, located between the HDMI and Ethernet port (Pi Zero W users, your camera connection is on the edge of the board.) With the Raspberry Pi turned off, gently lift the tabs until the port is open. Then insert the ribbon cable of the camera so that the blue section is facing the Ethernet port. Then push the tabs down to lock the cable in place.

Now boot up your Pi and from the Desktop go to the Preferences>Raspberry Pi Configuration. Then click the Interfaces tab and ensure your camera is enabled. You will be asked to reboot, do so and return to the desktop.

To check that the camera works, open a Terminal and type the following:

```
$ raspistill -o test.jpg
```

You will see the screen come alive with a live video feed, then after five seconds it will take a photo. You can view the photo in your Home directory.

YOU NEED

- **Any Pi**
- **Internet**
- **Latest Raspbian**
- **PIR Sensor**
- **3x female-to-female jumper wires**
- **Official Pi Camera**
- **Code: https://github.com/lesp/LXF245-AlarmSystem/archive/master.zip**

Connect the PIR

Our PIR (passive infrared) sensor has only three pins: VCC, OUT and GND. These connect to the Raspberry Pi GPIO as follows:

- VCC to 5V GPIO pin
- OUT to GPIO17
- GND to any GND GPIO pin

With the connections made, the hardware build is now complete.

You're a Pushover

Pushover is a simple notification system that works extremely well. But there's a cost of around $5 per platform, for lifetime access. Of course you can try out *Pushover* before you buy, and there's a seven-day trial period. For this project we purchased a licence for desktop and our Android device. Head over to **https://pushover.net** and sign up for a new account and download the app from the Android Play store.

Using the Chrome web and the Android app we were able to see our intruder live on our devices with no network configuration needed. Our intruder looks familiar!

After setting up the account, create a new application called "Alarm", enter the description, and then tick the box before clicking Create Application. When the application is created, you'll see an API/Token Key. Copy this and keep it safe. Don't share it with anyone or post it online. This code enables the app to send messages and is unique to you. On the main pushover page there will be a User Key. Again, note this down and keep it safe.

For this project we shall be writing Python code using the Python 3 editor, found in the Programming menu. Click File>New and in the new window click File>Save and call the file **Alarm.py**.

We start the code by importing many libraries. The first is "requests" which is used to work with HTTP requests. Then we import "datetime" to get time and date data for our alarm system. Then from four further libraries we import the PIR sensor class from GPIO Zero, the PiCamera class from picamera, pause from Signal which is used to stop our code from exiting. Finally, we import the sleep function from time, used to introduce a delay:

```
import requests, datetime
from gpiozero import MotionSensor
from picamera import PiCamera
from signal import pause
from time import sleep
```

Next we create two objects – one to enable easy use of the camera library, the other tells our code where the PIR sensor is connected.

```
camera = PiCamera()
pir = MotionSensor(17)
```

In order for the PIR sensor to "settle" and reduce the amount of false triggers, we pause the startup for 10

seconds, and inform the user that the system is now armed and ready to use:

```
print("ALARM Starting")
sleep(10)
```

The sequence to detect movement and trigger the image capture and posting to *Pushover* is all contained in a function called send_msg. This function will get the time and date of the intrusion, and store it to a variable called "timestamp" where we then use string formatting to convert the information into Year, Month, Day, Hour, Minutes and Seconds.

```
def send_msg():
  timestamp = datetime.datetime.now()
  timestamp = str(timestamp.strftime("%Y-%m-%d %H:%M:%S"))
```

Intruder alert!

We then print that the alarm has been triggered, and the time of the intrusion before we then use the camera. capture function to take a picture and then use the timestamp as the filename followed by ".jpg"

```
  print("ALARM TRIGGERED: "+timestamp)
  camera.capture("/home/pi/"+str(timestamp)+".jpg")
```

In order to post the alert to *Pushover*, we need to use requests to post the alert to a specific URL in a JSON format. We then pass our token and user keys inside a dictionary. Along with a message that will tell the reader that an alert has been triggered, and the time/date of the intrusion.

```
  r = requests.post("https://api.pushover.net/1/messages.json", data = {
      "token": "API TOKEN",
      "user": "API USER KEY",
      "message": ("Alert: "+timestamp)
  },
```

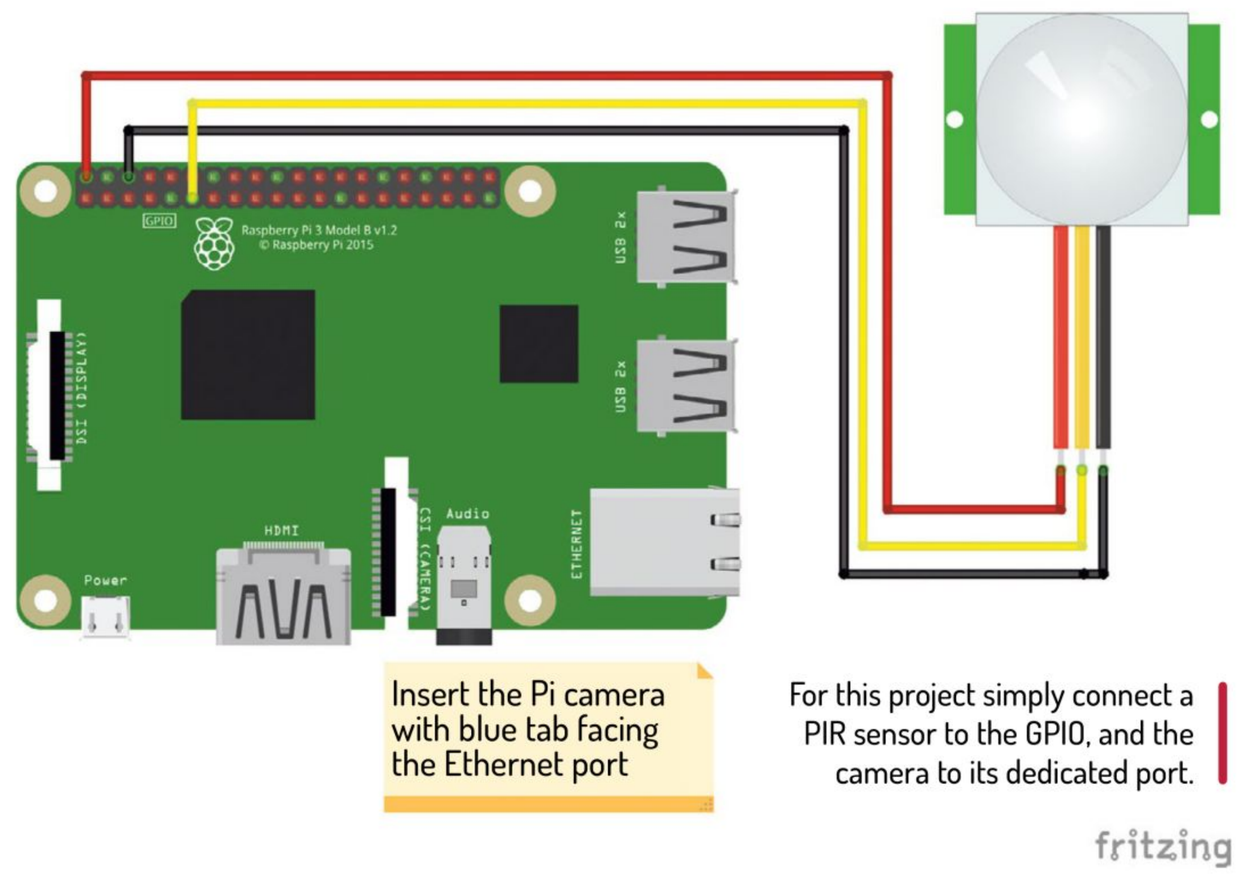
The next step for *Pushover* is to send the image across, and for this we use another dictionary to contain the "attachment" which is used to open the timestamp file as a read-only file. We also need to tell the code that the file is a jpeg image.

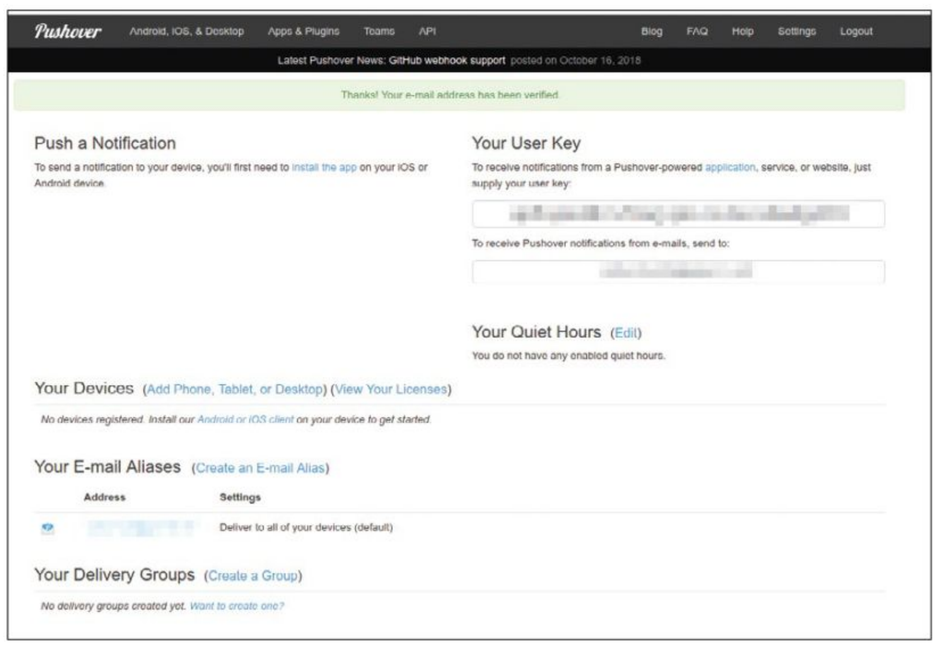
Then we print the response of all the actions, handily contained in the r object we created for our request. This will print in the shell and give us an indication that the code is working. Finally, we sleep for 30 seconds to enable the PIR to settle, although you can change this to meet your own requirements. This is the last part of the function:

```
  files = {
    "attachment": (timestamp, open("/home/pi/"+timestamp+".jpg", "rb"), "image/jpeg")
  })
  print(r.text)
  sleep(30)
```

The last two lines of code start with our PIR sensor sensing motion, and then calling the function that we've created. Then we use `pause()` to prevent the code from exiting, effectively running the code until the next intrusion is detected.

Save the code, and click Run>Run Module to start. Now run around and keep an eye on your Android device, as it will ping messages to your phone showing pictures of you! Now install the unit somewhere safe in a small box, and capture those intruders on camera!





Once we have created a Pushover account we can then easily create apps, such as this alarm project, which can then be used with multiple devices to boost your security around the home.

QUICK TIP

Is your PIR sensor a little "twitchy"? If so, increase sleep duration, to see if the sensor settles correctly. The PIR sensor can see 180 degrees, so using a tube to restrict its view will help settle the sensor.

» PULL ME, PUSH ME

In this project we created our own *Pushover* application that sends alerts to a waiting Android app. But there are other ways to interact with *Pushover*. There are apps for Android, iOS and for *Chrome*, *Firefox* and *Safari*. We tested the *Chrome* application, which can be installed from **https://pushover.net/clients** and it worked flawlessly.

There's also a dedicated system tray application that works pretty well with *Pushover*. The unofficial *Pullover* application is handy to receive the alerts from our device, but sadly it can't display images. That said, the app is handy for receiving text alerts, for example remote sensor triggers such as temperature, pressure and light levels. *Pullover* can be found at **https://github.com/cgrossde/Pullover** and it works with Linux, Mac and Windows devices.

Pushover has an extensive API (Application Programming Interface) that enables us to create groups to receive messages. But *Pushover* can also work with third-party applications, such as IFTTT (If This Then That), the Internet of Things gateway that enables visual programming of triggers or events that cause interesting outcomes. For example, geofencing a supermarket to prompt your device to show your shopping list!

Create a Raspberry Pi OLED monitor

Les Pounder shows us how to monitor a Raspberry Pi 4 with a small screen and a little bit of Python code.

Les Pounder is a freelance maker who works with organisations such as the Raspberry Pi Foundation to promote maker skills.

This project is to create a tiny OLED screen that will provide useful data about a Raspberry Pi at a glance. We will also run our project on boot to monitor the system as we work.

The OLED screen connects to the Raspberry Pi GPIO using an I2C interface. This is a special interface that only needs four pins on the Pi in order to send data. To connect your Pi please refer to the diagram on page 53.

With the physical connection made we now need to enable I2C on our Pi – do this in the Raspberry Pi Configuration tool. We will also need to install two Python libraries for this project. The first library is for our OLED screen and the second enables our code to work with network interfaces. In a terminal enter the following command.

```
$ sudo pip3 install Adafruit-SSD1306 netifaces
```

Writing the code

To write the code we shall be using *Thonny*, the default Python editor on the latest Raspbian. You're free to use any editor you want, but please save the code as **live-data.py** – and save often. We start the code with a line indicating where our Python 3 interpreter is located. Our project will be a background app in the system, so this enables it to run.

```
#! /usr/bin/env python3
```

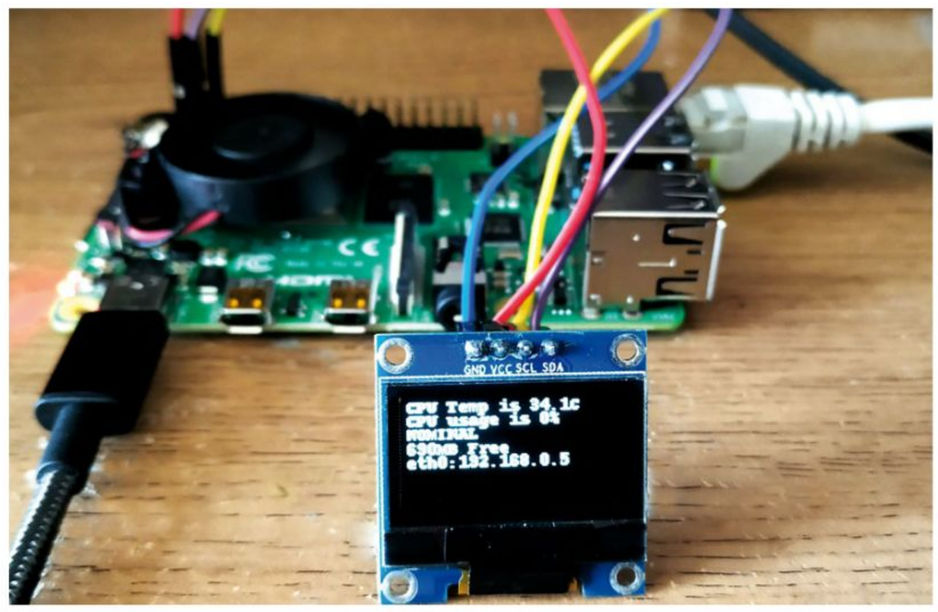
We now import three libraries to add features to our code. The first enables us to use the OLED screen with Python. The second is used for accessing system information in Python. The last is used to access network interface information. Note that we import 'netifaces' but refer to it as 'ni' for ease of use.

```
import Adafruit_SSD1306
import psutil
import netifaces as ni
```

Three more imports, and they are all from the Python Imaging Library (PIL). These will handle drawing content on the screen.

```
from PIL import Image, ImageDraw, ImageFont
```

In order to use the OLED screen, we need to tell Python where it is, and what size it is. For this we create an object called `disp` and in there we use the Adafruit library, specifically the `128_64` function. Note that the screen does not have a reset pin (`rst`) so we set that to `None`. Then we begin the connection to the screen.



This tiny screen is easy to work with and really cheap. It can provide all the data we need at a glance, without firing up a terminal.

```
disp = Adafruit_SSD1306.SSD1306_128_64(rst=None)
disp.begin()
```

To configure the screen we need to know the width and height; luckily, the function handles that. These are saved to variables, and then the screen is set to 1-bit colour (black and white).

```
width = disp.width
height = disp.height
image = Image.new('1', (width, height))
```

Preparing the screen for output, we create an object to which we can draw.

```
draw = ImageDraw.Draw(image)
```

Now we set the confines of the screen. The `padding` variable enables us to stay inside the screen; we set the top to match the padding, and the bottom of the screen as the height minus the padding. We also create a variable, `x`, to keep track of where text will appear on the horizontal axis.

```
padding = -2
top = padding
bottom = height-padding
x = 0
```

Here we gather the data for our screen. The first data collected is the CPU's temperature, and for this we open a file in read-only mode. This file stores the data from the CPU temp sensor, which we save to a variable called `f`.

```
f = open("/sys/class/thermal/thermal_zone0/temp",
"r")
```

YOU NEED

- Any Raspberry Pi
- An i2c OLED screen (128 x 64 pixels)
- 4x female-to-female jumper wires
- The Code: http://bit.ly/lxf254i2c

```
t = f.readline()
```

As the data is stored as a string, in order to do any maths with it we need to convert it to a `float`. The value stored is far too large, so we divide it by 1,000 and round the returned value to one decimal place, giving us the correct temperature.

```
t = float(t)
t = round(t/1000,1)
```

To get the CPU usage percentage, we create a variable called `cpu` and in it store the output from the `psutil.cpu_percent()` function. Then we round the value down, with no decimal place for ease of reference.

```
cpu = psutil.cpu_percent(interval=1)
cpu = round(cpu)
```

To get the RAM usage we again use `psutil`, specifically the virtual memory function. This will return a list of data, and from the list we need the second value, which is available memory. To get the second item we use `[1]`, as a list starts from 0. Then we do a little maths on the value in memory, which represents bytes. We need to convert it to megabytes so we divide the value by 1,024, twice.

```
memory = psutil.virtual_memory()
memory = memory[1]
memory = round(memory / 1024 / 1024)
```

The last piece of data is the internal IP address of the Ethernet port. For this we create a variable called `ip` and in it we store the data obtained from netifaces `ifaddresses` function. This requires the interface name as an argument, and in the returned data we need to go into a series of nested lists: first at position 2, then inside that list, position 0. Then we use the `addr` key to retrieve the data stored in a dictionary.

```
ip = ni.ifaddresses('eth0')[2][0]['addr']
```

The next step is to add an alert for our CPU. If the temp (`t`) is over 50C, it will change the text on the screen. Otherwise we get `NOMINAL`.

```
if t > 50:
  ALERT = "CPU TEMP ALERT"
else:
  ALERT = "NOMINAL"
```

We now start writing text to the screen. The first two lines write the CPU temp and usage.

The first line is our CPU temp and you will see that we start writing in the top left (`x`), two pixels from the top of the screen (top is -2). We write a string of text, and convert it to a string `t`. Then we use the default font, and set the colour to `255` (white). The second line is the same, but we set the move down eight pixels.

```
draw.text((x, top),     "CPU Temp is "+str(t)+"C",
font=ImageFont.load_default(), fill=255)
draw.text((x, top+8),   "CPU usage is "+str(cpu)+"%",
font=ImageFont.load_default(), fill=255)
```

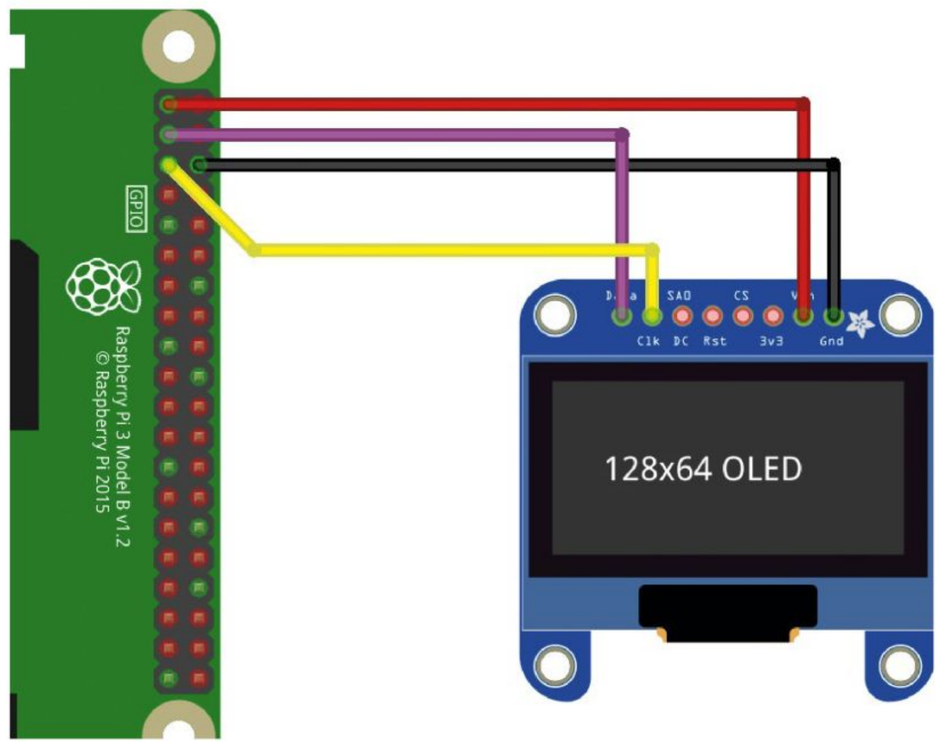
The next three lines are our Alert status, available RAM, and IP address.

```
draw.text((x, top+16),    ALERT, font=ImageFont.load_
default(), fill=255)
draw.text((x, top+25),   str(memory)+"MB Free",
font=ImageFont.load_default(), fill=255)
draw.text((x, top+33),   "eth0:"+ip, font=ImageFont.
load_default(), fill=255)
```

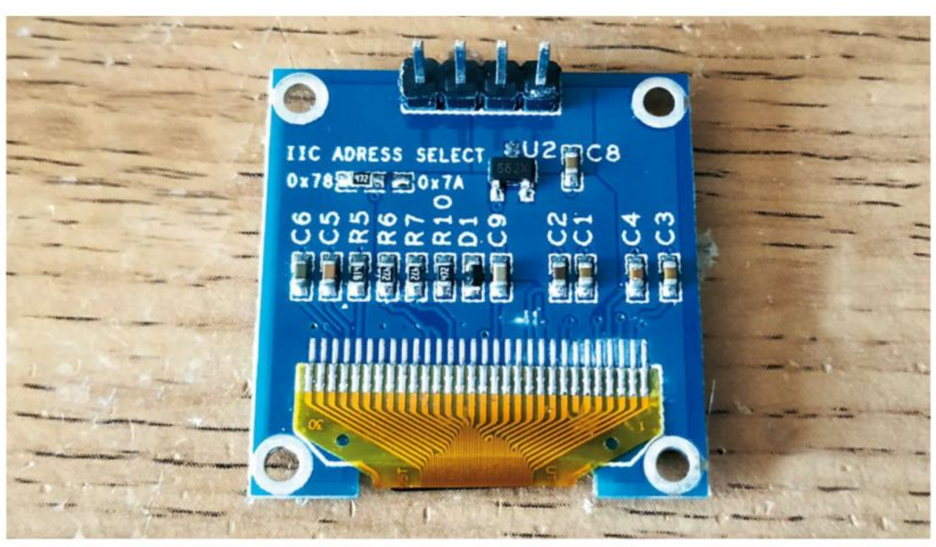
The very final two lines of this code will update the display and show the live data.

```
disp.image(image)
disp.display()
```

Save the code and click Run; your OLED screen will come to life and show the latest data for your machine. See below to have this code run on boot.



Only four wires are needed for this project. It's easier to wire up than an LED!



If you need to change the I2C address for your screen, desolder the connection to 0x78 and make a new one to 0x7A.

QUICK TIP

Sometimes OLED screens come in I2C and SPI versions. If in doubt, always look for SDA and SCL. But sometimes SCL can be referred to as SCLK or CLK. No matter what, SDA is data and SCL is 'Clock', used for precisely timing the interface.

» RUNNING THE CODE ON BOOT

Our project is best used on boot, so here are the extra steps to make that happen. First we need to make our code executable, and for that we need to open a terminal and type

```
$ chmod +x live-data.py
```

Now to make the code available to everyone as a command, we need to copy it to **/usr/bin**, so in the terminal type

```
$ sudo cp live-data.py /usr/bin/live-data
```

Now give the command a try. In the terminal type the following and watch the OLED screen.

```
$ live-data
```

So now we have the command, let's use it on boot. In the terminal we shall use *crontab* and edit *cron* to start the command on boot, and run the command every minute that the Pi is on.

```
$ sudo crontab -e
```

In *crontab*, go to the bottom and add this line.

```
* * * * * live-data >/dev/null 2>&1
```

Now save by pressing Ctrl+X and pressing Y, then Return. That's it! Now you can reboot your Pi and watch your OLED screen as every minute your Pi is on, it will give you live data. The lengthening autumn evenings will just fly by...

Print wirelessly with your Raspberry Pi

Breathe new life into an old printer by using your Raspberry Pi as a wireless print server.

What you'll need

- Latest Raspbian image
  raspberrypi.org/downloads
- USB printer
- USB wireless card

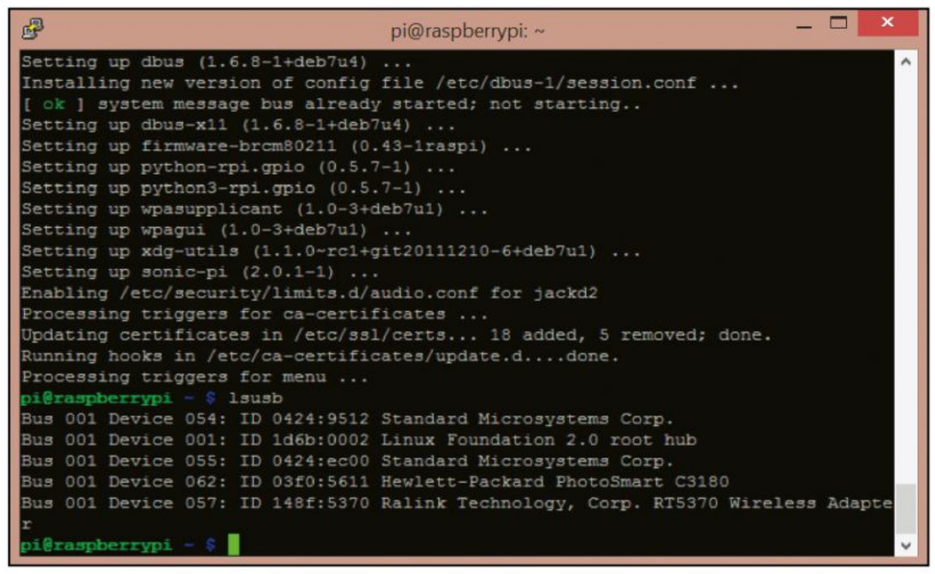
Wireless printing has made it possible to print to devices stored in cupboards, sheds and remote rooms. It has generally shaken up the whole process of printing and enabled output from smartphones, tablets, laptops and desktop computers alike. But you don't have to own a shiny new printer for this to work; old printers without native wireless support don't have to end up in the bin, thanks to the Raspberry Pi.

The setup is simple. With your Raspberry Pi set up with a wireless USB dongle, you connect your printer to a spare USB port on the computer. With Samba and CUPS (Common Unix Printing System) installed on the Raspberry Pi, all that is left to do is connect to the wireless printer from your desktop computer, install the appropriate driver and start printing.

CUPS gives the Raspberry Pi a browser-based admin screen that can be viewed from any device on your network, enabling complete control over your wireless network printer.

01 Check your printer works

Before starting, check that the printer you're planning to use for the project still works and has enough ink. The easiest way to do this is to check the documentation (online if you can't find the manual) and run a test print.



02 Detect your printer

With your Raspberry Pi set up as usual and the printer connected to a spare USB port, enter:

```
lsusb
```

This will confirm that the printer has been detected by your Raspberry Pi. In most cases you should see the manufacturer and model displayed.

03 Install Samba and CUPS

Install Samba to enable file and print sharing across the entire network:

```
sudo apt-get install samba
```

Next, install CUPS:

```
sudo apt-get install cups
```

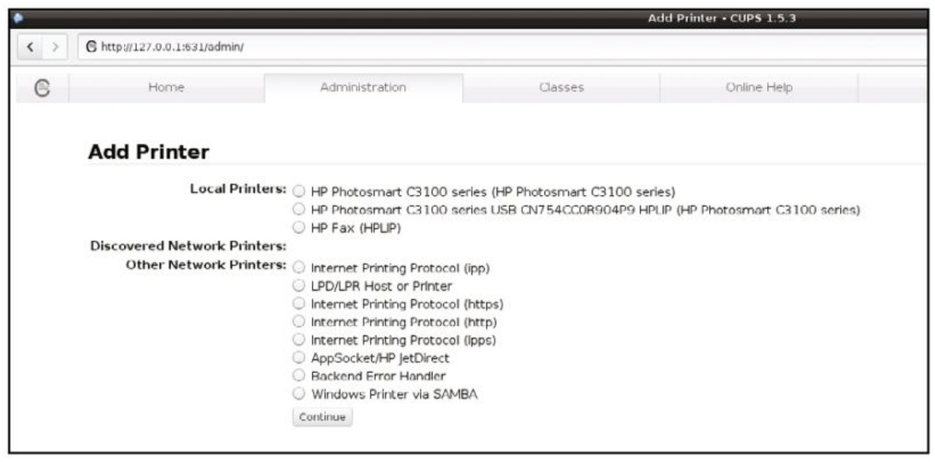
With a print server created, begin configuration by adding default user 'pi' to the printer admin group:

```
sudo usermod -a -G lpadmin pi
```

04 Set up print admin

Set up the CUPS print admin tool first. Boot into the GUI (startx) and launch the browser, entering 127.0.0.1:631.

Here, switch to Administration and ensure the 'Share printers' and 'Allow remote administration' boxes are selected. Next, select Add Printer and enter your Raspbian username and password when prompted.



05 Add your printer

A list of printers will be displayed, so select yours to proceed to the next screen where you can confirm the details, add a name and check the Share This Printer box. Click Continue to load the list of printer drivers and select the appropriate one from the list.

06 Configure Samba for network printing

Using a Windows computer for printing? Samba will need some configuration. Open '/etc/samba/smb.conf' in nano, search (Ctrl+W) for '[printers]' and find 'guest ok' which you should change as follows:

```
guest ok = yes
```

Next, search for "[print$]." Then change the path as follows:

```
path = /usr/share/cups/drivers
```

Begin configuration by adding the default user 'pi' to the printer admin group

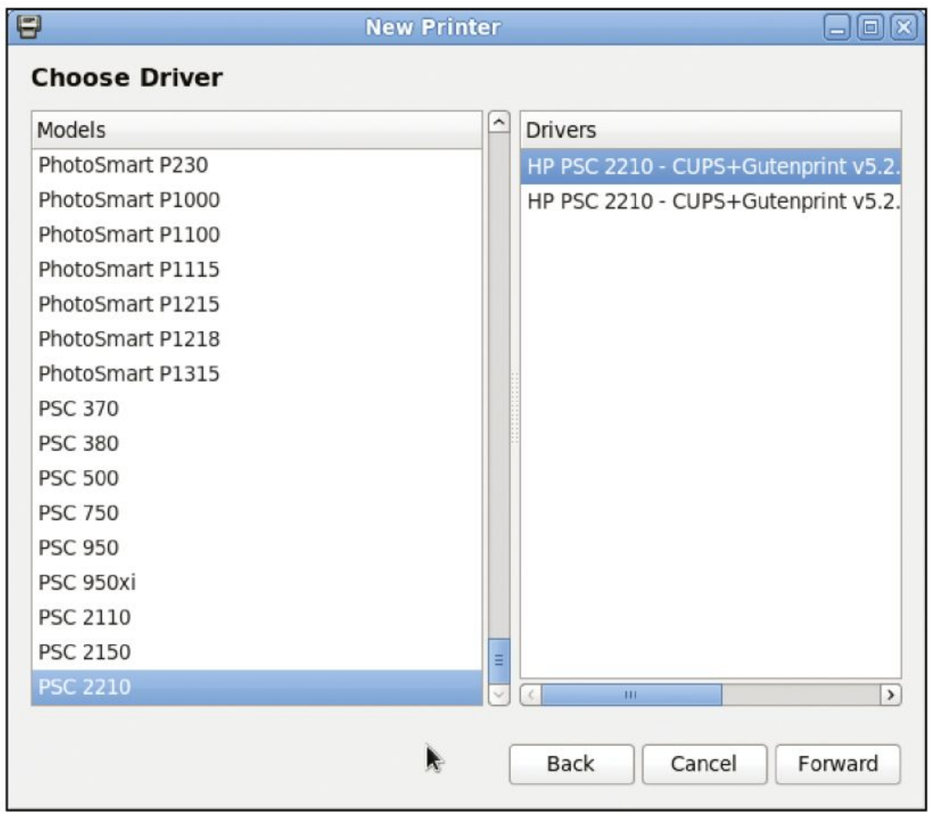
07 Join a Windows workgroup

With these additions made, search for "workgroup" in the configuration file and add your workgroup:

```
workgroup = your_workgroup_name
wins support = yes
```

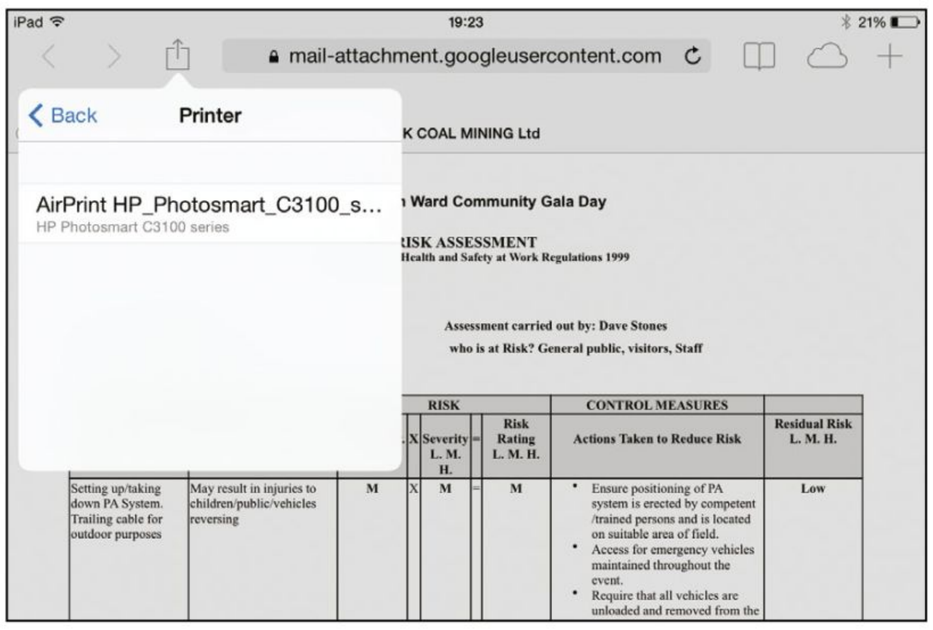
Make sure you uncomment the second setting so that the print server can be seen from Windows. Save your changes and then restart Samba:

```
sudo /etc/init.d/samba restart
```



08 Accessing your printer on Linux

Meanwhile, it's a lot easier to access your wireless printer from a Linux, Mac OS X or other Unix-like system, thanks to CUPS. All you need to do is add a network printer in the usual way and the device will be displayed.



09 Add AirPrint compatibility

It's also possible to print wirelessly from your iPad using Apple's AirPrint system. To do this, you need to add the Avahi Discover software:

```
sudo apt-get install avahi-discover
```

Your wireless printer will now be discoverable from your iPad or iPhone and will be ready to print.

How I made: Holographic Audio Visualiser

Make your music come alive with this futuristic holographic-style visualiser and track changer.

Aaron Tainter & Davis Lau

Aaron writes the code for the projects while Davis works on hardware and designs parts for the builds.

Like it?

Aaron is a software developer for eBay and Davis is a mechanical engineer at an aerospace company.

Further reading

They run a Youtube channel (**www.youtube.com/c/HackerShack**) and a website called Hacker Shack (**www.thehackershack.com**) where they share interesting DIY builds.

With some acrylic sheets mounted on a wooden frame, the hacker duo at Hacker House have managed to transform a regular LCD monitor into a 3D 'holographic' music visualiser that's a sure head-turner at any party.

The device projects an image from a monitor down onto an acrylic pyramid, or frustum, which then creates the holographic parallax effect. To add to the futuristic aura of the device, Aaron and Davis have connected a Flick board from Pi Supply, which enables them to use Jedi-like hand gestures to control playback, volume and visualisations. The Flick board is connected to a Raspberry Pi 3 that processes the gestures and sends them to the visualiser, which is running on a more powerful computer that can smoothly run WebGL graphics at a decent frame rate.

The enterprising pair have done an excellent job of documenting the entire build with a detailed construction video and a thorough illustrated guide (Hackster.io, **http://bit.ly/HoloVisualiser**). At first glance, the physical construction of the project might seem quite daunting, since creating the frame requires several different lengths of wood and the acrylic sheets need to be scored to exact dimensions dependent on the size of the monitor projecting the visualisation. The construction part of the project also requires a slew of 3D-printed parts to hold the frame, the monitor and the Flick board, along with the Raspberry Pi. However, the duo has shared the .STL files for the custom parts along with the code.

In fact, their instructions are so easy to follow that a group of school pupils have been able to successfully replicate the project and build their own holographic visualiser (Instructables, **http://bit.ly/HoloProjector**) as part of a school project. Because the visualiser is essentially controlled by web requests, the students have even managed to change some of the Pi code to get the device to work with buttons instead of the Flick gesture-control pad.

What was the inspiration behind the project?

We acquired a gesture-control board and we were brainstorming things to use it for. I had done some 3D audio-visualiser graphics with WebGL in the past for a personal project. At the same time, we saw some videos online about how to make mini hologram-projectors with a CD case and a phone. We thought it would be interesting to merge those things together to make a really futuristic visualiser.

What was the most challenging part of the project? Is there anything you would have done differently?

The construction was pretty straightforward, but the code took quite a bit of time to write. I would have liked to add more animations or a simple interface for anyone at home to add their own visualiser.

Any particular reason for using a computer to run the visualiser instead of the Pi?

We initially tried running the WebGL graphics on the Pi, but it wasn't able to run them at a smooth frame rate. Ideally, we'd power everything from one device, but the control board didn't have an SDK that could be run on a normal UNIX machine.

Do you have any plans to extend the project?

I was originally going to use the Spotify API for music, but I discovered that it only streams 30-second sound clips. We had to use the Soundcloud API because it was the only thing that I could find that can stream full songs, but it's mostly independent stuff. In the future, I might try to stream audio from local files or a Bluetooth connection with my phone, so that I can play more popular music.

This project involves a lot of physical construction. What things should people keep in mind before trying to build it?

The Plexiglass is very difficult to cut, so I would make sure to use a thin piece and create some paper templates to verify that everything fits properly. If they don't fit perfectly in place, you can always use some clear tape at the bottom to hold them together.

What kind of projects interest you? What does the Hacker House have for us in the pipeline?

I'm really interested in the practical applications of AI and computer vision. I'd like to make more smart-home appliances with AI, either a custom AI or Alexa/Google Home integration.

In the past, we also wrote code for an autonomous cooler, but it navigated based on GPS waypoints. We want to extend that code with some computer vision to give a few other objects some basic autonomous vehicle capabilities.

Components list

- Raspberry Pi 3
- Pi Supply Flick gesture breakout
- 24-inch computer monitor
- 0.093-inch acrylic sheet
- 0.118-inch black foam PVC sheet
- Various sizes of wooden planks
- Handful of screws
- Hand drill
- Spray paint
- Some 3D-printed brackets

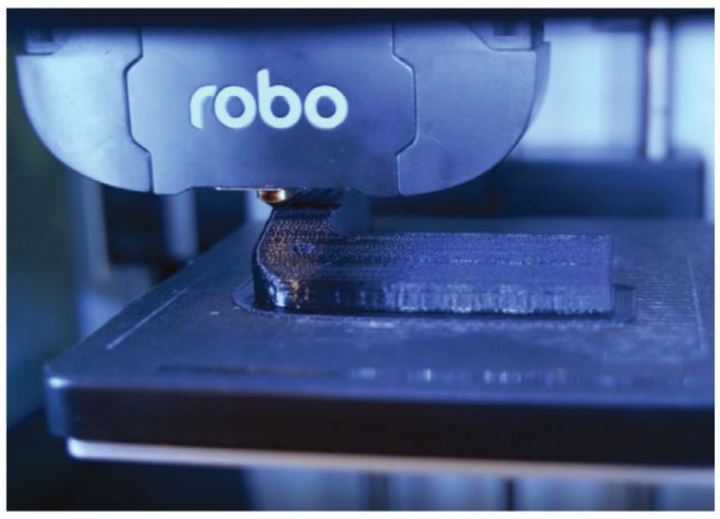
Monitor
A 24-inch monitor located at the top of the device is used to generate the 'holographic' display. It's connected to the Raspberry Pi via an HDMI cable, running alongside a power cable, hidden behind the frame.

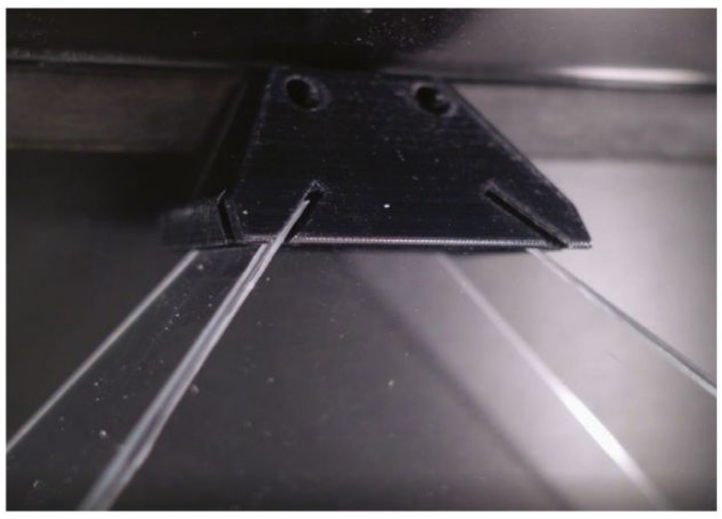
3D projection
The pseudo-holographic projection is created by reflecting the audio-visualiser animation playing on the monitor at 45 degrees. This has a parallax effect and makes the reflection seem 3D.

Acrylic pyramid
The most challenging part of the build, the three-sided pyramid with a 45 degree slope is made up of three pieces of acrylic sheets.
A 3D-printed frustum joint holds the sheets together at the correct angles.

Wooden frame
The top frame holds the monitor and is just a rectangle with a lip on the inside. The bottom frame has two wooden columns at the back to hold the top frame.

Track changer
The Flick gesture-control board is connected to the Raspberry Pi placed inside the 3D-printed housing. You can control volume, music playback and change the animation of the visualiser using simple hand gestures over this motion control board.





1 3D-printed parts
The project uses several 3D-printed parts, including the frustum joints holder, brackets for the monitor and the housing for the Raspberry Pi. You can find the .STL files for these components on the project's hackster.io page.

2 Frustum joints holder
The creation of the frustum was one of the trickiest parts of the build. A 3D-printed holder, bolted to the centre of the frame, was used to slide in the three acrylic sheets at the proper angles for the projection effect to work.

3 Black PVC
The base and the back side are covered with black PVC sheets that block light and also help the 'holographic' effect stand out more strongly in front of the dark background. The sheets have small notches to make them fit between the 3D-printed brackets.

Voice control

In this and further issues, we will look at the parts needed to make your own voice control software for your projects. If you want a virtual assistant, one project is the Jasper system (**jasperproject.github.io**). The documentation on the main website has a description of hardware to attach to your Raspberry Pi and a full set of instructions for installation and configuration. There is a set of standard modules included to allow interaction with various services. Use the time, Gmail or even the joke module, and there are also third-party modules for you to access. There is even a developer API and documentation to help you add your own functionality to Jasper.

Create your own digital assistant, part 1

Everyone would like to tell their computers exactly what to do. Well with Python and a Raspberry Pi, now you can.

Everyone who has watched the *Iron Man* movies has probably dreamt of having their own artificially intelligent computer system to do their every bid and call. While Jarvis has massive amounts of computing power behind him, you can construct the front-end with very modest resources. With a Raspberry Pi and the Python programming language, you can build your own personal digital assistant that can be used as a front-end to whatever massive supercomputing resources that you use in your day-to-day life as a playboy, philanthropist genius. We will go over the basics that you will need to know over the next few pages, so that by the end of the series you should be able to build your own rudimentary, customised agent.

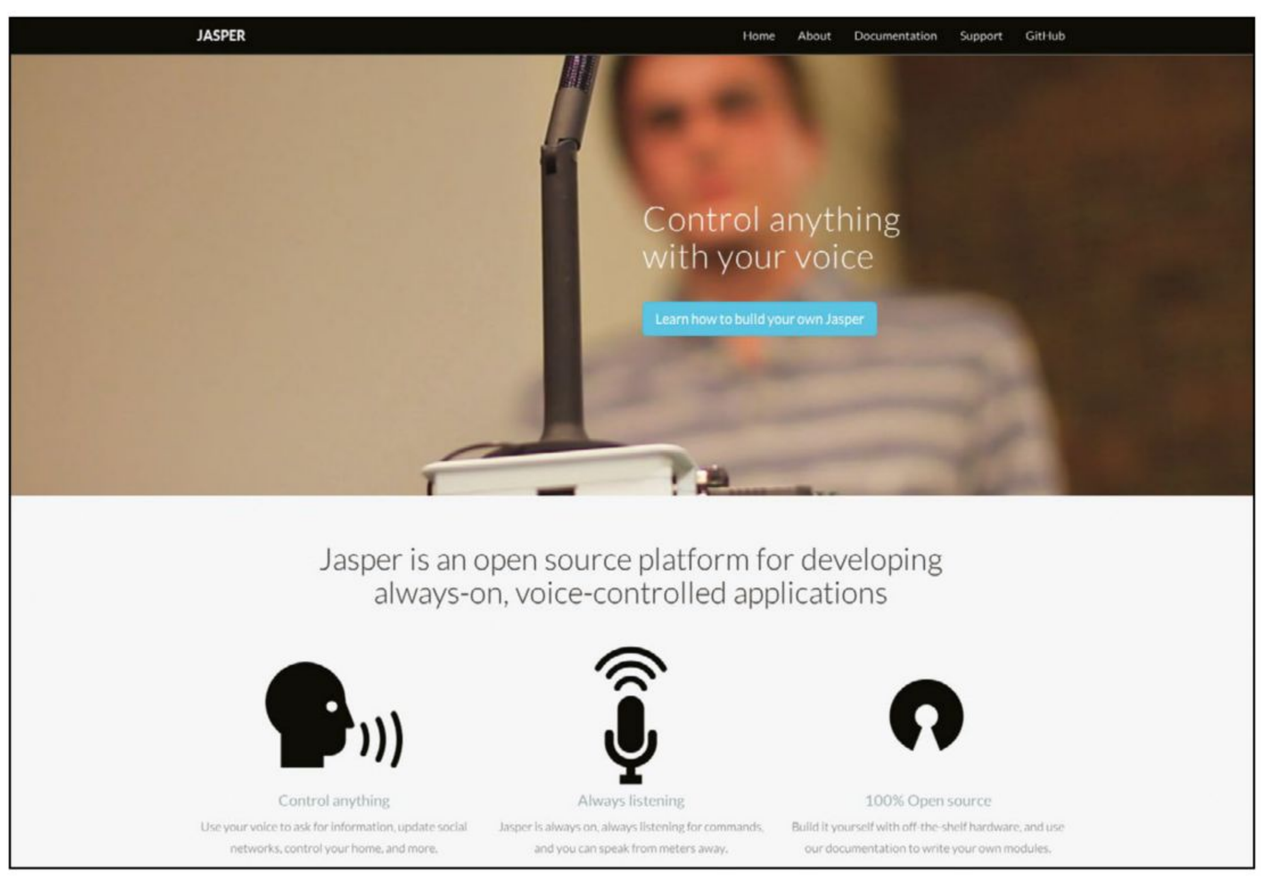
The first step to interacting with the humans around us is to listen for verbal commands so that we know what we need to process. You have several options available to handle this task. To keep things simple, we will be dealing only with devices that are plugged into one of the USB ports. With that stipulation you can talk directly with the USB device at the lowest level. This might be necessary if you are trying to use something that is rather unusual to do the listening, but you will probably be better off using something that is a bit more common. In this case you can use the Python module PyAudio. PyAudio provides a Python wrapper around the low level cross-platform library PortAudio. Assuming that you are using something like Raspbian for your distribution, you can easily install the required software with the command:

```
sudo apt-get install python-pyaudio
```

If you need the latest version you can always grab and build it from source. PyAudio provides functionality to both read in audio data from a microphone, along with the ability to play audio data out to headphones or speakers. So we will use it as our main form of interaction with the computer.

The first step is to be able to read in some audio commands from the humans who happen to be nearby. You will need to import the 'pyaudio' module before you can start interacting with the microphone. The way PyAudio works is similar to working with files, so it should seem familiar to most programmers. You start by creating a new PyAudio object with the statement p = pyaudio.PyAudio(). You can then open an input stream with the function p.open(...), with several parameters. You can set the data format for the recording; in the example code we used format=pyaudio.paInt16. You can set the rate in Hertz for sampling. For example, we are using rate=44100, which is the standard 44.1KHz sampling rate. You also need to say how big a buffer to use for the recording – we used frames_per_buffer=1024. Since we want to record, you will need to use input=true. The last parameter is to select the number of channels to record on, in this case we will use channels=2. Now that the stream has been opened, you can start to read from it. You will need to read the audio data in using the same chunk size that you used when you created the stream – it will look like stream.read(1024). You can then simply loop and read until you are done. There are then two commands to shutdown the input stream. You need to call stream.stop_stream() and then stream.close(). If you are completely done, you can now call p.terminate() to shutdown the connection to the audio devices on your Raspberry Pi.

The next step is to be able to send audio output so that Jarvis can talk to you as well. For this you can use PyAudio, so we won't have to look at another Python module. To make things simple, let's say that you have a WAVE file that you want to play. You can use the 'wave' Python module to load it. Once again, you will create a PyAudio object and open a stream. The parameter 'output' should be set to true. The format, the number of channels and the rate is all information that will be derived from the audio data stored in your WAVE file. To actually hear



Right Check out the documentation to see what Jasper can do: **bit.ly/1MCdDh4**

Full code listing

```
# You need to import the pyaudio module
import pyaudio

# First, we will listen
# We need to set some parameters
# Buffer chunk size in bytes
CHUNK = 1024
# The audio format
FORMAT = pyaudio.paInt16
# The number of channels to record on
CHANNELS = 2
# The sample rate, 44.1KHz
RATE = 44100
# The number of seconds to record for
RECORD_SECS = 5

# Next, we create a PyAudio object
p = pyaudio.PyAudio()

# We need a stream to record from
stream = p.open(format=FORMAT, channels=CHANNELS,
                rate=RATE, input=TRUE, frames_per_buffer=CHUNK)

# We can now record into a temporary buffer
frames = []
for i in range(0, int(RATE / CHUNK * RECORD_SECS)):
   data = stream.read(CHUNK)
   frames.append(data)

# We can now shut everything down
stream.stop_stream()
stream.close()
p.terminate()

# If we want to play a wave file, we will need the wave module
import wave

# We can open it, give a filename
wf = wave.open("filename.wav", "rb")

# We need a new PyAudio object
p = pyaudio.PyAudio()

# We will open a stream, using the settings from the wave file
stream = p.open(format=p.get_format_from_width(wf.getsampwidth()),
                channels=wf.getnchannels(), rate=wf.getframerate(),
                output=True)

# We can now read from the file and play it out
data = wf.readframes(CHUNK)
while data != '':
      stream.write(data)
      data = wf.readframes(CHUNK)

# Don't forget to shut everything down again
stream.stop_stream()
stream.close()
p.terminate()
```

the audio you can simply loop through, reading one chunk of data from the WAVE file at a time and immediately writing out to the PyAudio stream. Once you're done you can stop the stream and close it, as you did above.

In both of the above cases, the functions block when you call them until they have completed. What are the options if you want still be able to do processing while you are either recording audio or outputting audio? There are non-blocking versions that take a callback function as an extra parameter called stream_callback. This callback function takes four parameters, named in_data, frame_count, time_info, and status. The in_data parameter will contain the recorded audio if input is true. The callback function needs to return a tuple with the values out_data and flag. Out_data contains the data to be outputted if output is true in the call to the function open. If the input is true instead, then out_data should be equal to None. The flag can be any of paContinue, paComplete or paAbort, with obvious meanings. One thing to be aware of is that you cannot call, read or write functions when you wish to use a callback function. Once the stream is opened, you simply call the function stream.start_stream(). This starts a separate thread to handle this stream processing. You can use stream.is_active() to check on the current status. Once the stream processing is done, you can call stream.stop_stream() to stop the secondary thread.

Now that we have covered how to get audio information into and out of your Raspberry Pi, you can start by adding this functionality to your next project. In the next step, we will look at how to convert this audio information into something usable by the computer by using voice recognition modules. We will also look at the different ways to turn text into audio output using TTS modules.

Digital assistant, part 2: speech recognition

In this second instalment, learn how to decode your audio and figure out what commands are being given by the humans around you.

Offload tasks

You can offload the audio data processing to Google, accessing the API directly over HTTP by posting your audio data to the appropriate URL. First install the Python module SpeechRecognition:

```
pip install SpeechRecognition
```

Now create an instance of the Recognizer object. A Helper object, called WavFile, will take an audio file and prepare it for use by the Google API. Then process it with the record() function and hand this processed audio in to the function recognize(). When it returns, you will get a list of pairs of possible texts, along with a percentage confidence level for each possible text decoding. Be aware that this module uses an unofficial API key to do its decoding, so for anything more than small personal testing you should request your own API key.

Previously we looked at how we could have our Raspberry Pis listen to the world around them. This is the first step in building our own version of the J.A.R.V.I.S system made famous in the *Iron Man* movies. The next step is to try and make sense of what we may have just heard. In general, this is called speech recognition and it is a very large and active area of research. Every major mobile phone operating system has applications trying to take advantage of this mode of human interaction. There are also several different Python modules available that can do this speech-to-text (STT) translation step. In this second article, we will look at using Pocket Sphinx to do all the heavy lifting. Sphinx was developed by Carnegie Mellon University and is licensed under a BSD licence, so you are free to add any extra functionality that you may need for specific tasks. Because of the activity in this field, it is well worth your time to keep track of all the updates and performance improvements.

While you can download the source code for all of these modules and build it all from scratch, we are going to assume that you are using one of the Debian-based distributions, like Raspbian. For these you can simply use:

```
sudo apt-get install python-pocketsphinx
```

...to get all of the required files for the engine. You will also need audio model files and language model files in order to get a translation in you language of choice. To get the files needed for English, you can install the packages:

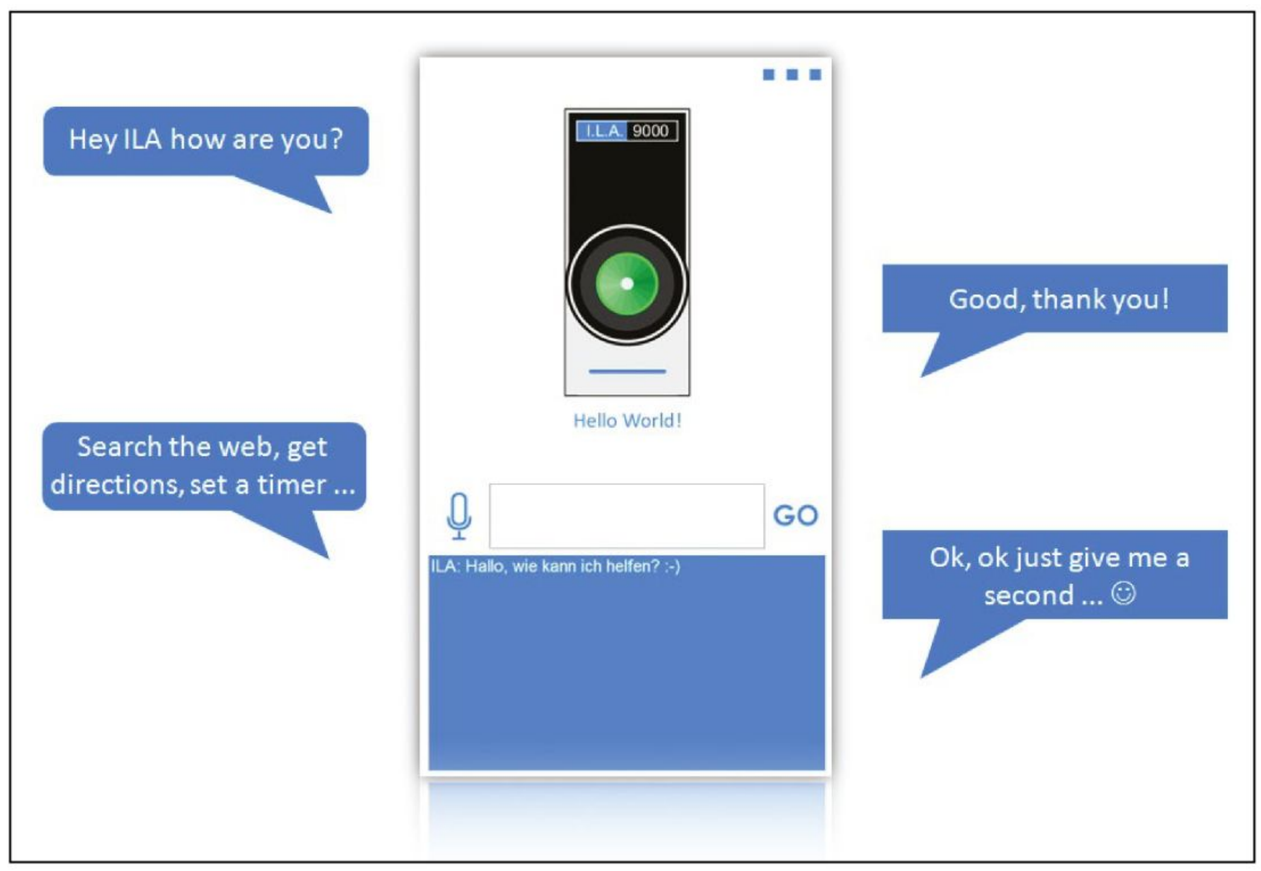
```
sudo apt-get install pocketsphinx-hmm-wsj1 pocketsphinx-lm-wsj
```

You may need to go outside the regular package management system if you want to process other languages. Then you can simply start writing and using your code straight away. To start using these modules, you will need to import both pocketsphinx and sphinxbase with:

```
import pocketphinx as ps
import sphinxbase
```

These modules are actually Python wrappers around the C code that handles the actual computational work of translating sounds to text. The most basic workflow involves instantiating a Decoder object from the pocketsphinx module. The Decoder object takes several input parameters to define the language files it is allowed to use. These include 'hmm', 'lm' and 'dict'. If you use the above packages used to handle English, then the files you need will be in the directories /usr/share/pocketsphinx/model/hmm/wsj1 and /usr/share/pocketsphinx/model/lm/wsj. If you don't set these parameters, then it tries to use sensible defaults which usually work fine for English language speech. This newly created Decoder object can now be given WAV files with data to process. If you remember that previously, we saved the recorded speech as a WAV file. In order to have this audio recorded in the correct format, you will want to edit the code from the first tutorial and ensure that you are recording in mono (using one channel, for example), and recording at 16kHz with 16-bit quality. To read it properly you can use a file object and load it as a binary file with read permissions. WAV files have a small piece of header data at the beginning of the file that you need to jump over. This is done by using the seek function to jump over the first 44 bytes. Now that the file pointer is in the correct position, you can hand the file object in to the Decoder object's decode_raw() function. It will then go off and do a bunch of data crunching to try and figure what was said. To get the results, you would use the get_hyp() function call. You get a list with three elements from this function: a string containing the best guess at the spoken text, a string containing the utterance ID and a number containing the score for this guess.

So far, we've looked at how to use the generic language and audio models



Right CMUSphinx is used in cross-platform, open source projects like ILA, the Intelligent Learning Assistant

for a particular language. But Pocket Sphinx is a research-level language system, so it has tools available to enable you to build your own models. In this way, you can train your code to understand your particular voice with all of its peculiarities and accents. This is a long process, so most people will not be interested in doing something so intensive. However, if you are interested, there is information available at the main website (**cmusphinx.sourceforge.net**). You can also define your own models and grammars to tell pocketsphinx how to interpret the audio that it is processing. Once again, effectively carrying out these tasks will require more in depth reading on your part.

If you want to process audio more directly, you can tell Pocket Sphinx to start processing with the function start_utt(). You can then start reading audio from your microphone. You will want to read in appropriate sized blocks of data before handing it in to pocketsphinx – specifically to the function process_raw() – and you will still need to use the function get_hyp() to actually get the translated text. Also, because your code can't know when someone has finished a complete utterance, you will need to do this from within a loop. On each pass of the loop, read another chunk of audio and feed it into pocketsphinx. You then need to call get_hyp() again to see if you can get anything intelligible from the data. When you are done doing this real-time processing, you can use the function end_utt().

So far, we have covered how to record your speech and how to turn that speech into text. In the next tutorial, you will learn how to take that translated speech and actually take actions based on how the system has been configured. But even with only these two steps, you could build yourself a nifty little dictaphone or vocal note-taking system.

Full code listing

```
# You first need to import the required modules
import pocketsphinx as ps
import sphinxbase

# Next, you need to create a Decoder object
hmmd = '/usr/share/pocketsphinx/model/hmm/wsj1'
lmd = '/usr/share/pocketsphinx/lm/wsj/wlist5o.3e-7.vp.tg.lm.DMP'
dictd = '/usr/share/pocketsphinx/lm/wsj/wlist5o.dic'
d = ps.Decoder(hmm=hmmd, lm=lmd, dict=dictd)

# You need to jump over the header information in your WAV file
wavFile = file('my_file.wav', 'rb')
wavFile.seek(44)

# Now you can decode the audio
d.decode_raw(wavFile)
results = d.get_hyp()

# The most likely guess is the first one
decoded_speech = results[0]
print "I said ", decoded_speech[0], " with a confidence of ", decoded_speech[1]

# To do live decoding, you need the PyAudio module
import pyaudio
p = pyaudio.PyAudio()

# You can now open an input stream
in_stream = p.open(format=pyaudio.paInt16, channels=1, rate=16000,
                   input=True, frames_per_buffer=1024)
in_stream.start_stream()

# Now you can start decoding
d.start_utt()
while True:
   buf = in_stream.read(1024)
   d.process_raw(buf, False, False)
   results = d.get_hyp()
   # Here you would do something based on the decoded speech
   # When you are done, you can shut everything down
   break
d.end_utt()
```

Digital assistant, part 3: run other programs

This third and final article will cover how to actually run the commands you are giving to your Raspberry Pi.

This is the last in our trilogy of articles to help you build your own voice control system. The first article looked at how to listen for incoming commands. This involved listening on a USB device and also outputting audio feedback to a user. The second article looked at how to interpret those commands. This involved using speech recognition libraries to translate the recorded audio into text that can be processed. This time, we will look at how to actually run the commands that were given. We will look at a few different options to execute tasks and get work done based on the interpreted speech.

If you have put together a system based on the suggestions from the first two articles, you should have a string containing the text that was spoken to your Raspberry Pi. But, you need to figure out what command this maps to. One method is to do a search for keywords.

> A more Pythonic method is to use classes and objects. You can write a script that defines a class that contains methods for you to call when you need it

If you have a list of keywords available, you can loop through them and search the heard string to see if any one of those keywords exist within it as a substring. Then you can execute the associated task with that keyword. However, this method will only find the first match. What happens if your user accidentally includes a keyword in their spoken command before the actual command word? This is the auditory equivalent to having fat fingers and mistyping a command on the keyboard. Being able to deal with these errors gracefully is an ongoing area of research. Maybe you can create a new algorithm to handle these situations?

Let's say that you have a series of Python scripts that contain the various tasks you want your system to be able to tackle. You need a way to have your system be able to run these scripts when called upon. The most direct way to run a script is to use **execfile**. Say you have a script called do_task.py that contains Python code you want to run when a command is given; you can run it with:

```
execfile("do_task.py")
```

Using this form, you can add command line options to the string being handed in. This will look in the current directory for the script of that file name and run it in the current execution context of your main program. If you need to rerun this code multiple times, call execfile each time you do. If you don't need the script to run within the same context, use the subprocess module. Import it with:

```
import subprocess
```

You can then execute the script like so:

```
subprocess.call("do_task.py")
```

This will fork off a subprocess of the main Python interpreter and run the script there. If your script needs to interact with the main program, this is probably not the method that you should use. Collecting output from a call to do_task.py with subprocess isn't straightforward, so another way of achieving the same thing is to use the import statement. It also runs the code in your script at the point the import statement is called. If your script only contains executable Python statements, these get run at the point of importation. In order to rerun this code, you need to use the reload command. The reload command doesn't exist in version three – so if you're using that particular Python version, a better option is to encapsulate the code contained in the script within a function. You can then import the script at the beginning of your main program and simply call the relevant function at the correct time. This is a much more Pythonic method to use. If you have the following contents for do_task.py:

```
def do_func():
  do_task1()
  do_task2()
```

You can then use it with the following code within your main program:

```
import do_task
....
....
do_task.do_func()
....
```

An even more Pythonic method is to use classes and objects. You can write a script that defines a class that contains methods for you to call when you need it.

What are the options if you want to do something that isn't achievable with a Python script? In these cases, you need to be able to run arbitrary programs on the host system. The host

Social media

You may want your system to check your social media accounts on the Internet. There are several Python modules available to handle this. Let's say that you want to be able to check your Facebook account. Install the following Python module:

```
sudo apt-get install python-facebook
```

You can then use **import facebook** to get access to the Facebook API. If you're a Twitter user, install the python-twitter Debian package to use the Twitter API. Email is easier as long as your email provider offers IMAP or POP access. You can then import emails and get voice control to read unread emails out to you. For the Google fans, Google has a Python module that provides access to the APIs for almost everything available; work with your calendar, email or fitness data.

system in this case is your Raspberry Pi. As a toy example, let us say you need to download some emails using the Fetchmail program. You can do this in a couple of different ways. The older method is to use the **os.system()** command where you hand in a string. In our example, this would look something like the following:

```
os.system("/usr/bin/fetchmail")
```

You need to explicitly use **os.wait()** to be told exactly when the task has finished. This method is now being replaced by the newer subprocess module. It gives you more control over how the task gets run and how you can interact with it. A simple equivalent to the above command would look like this:

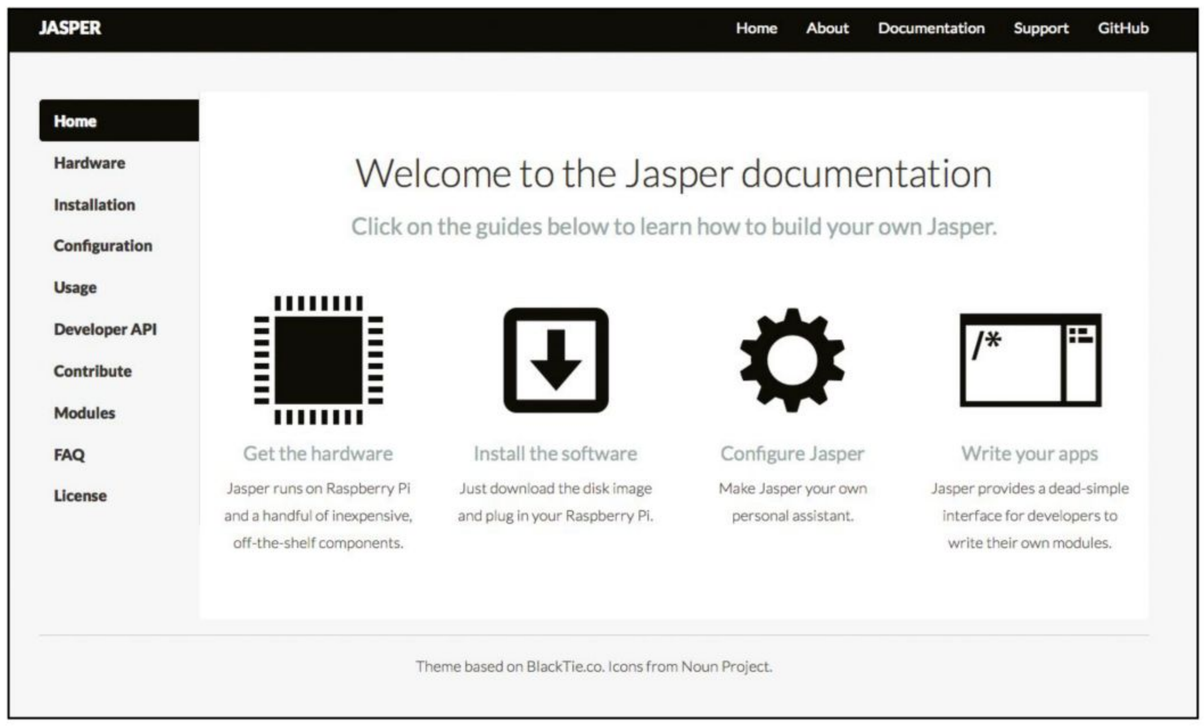
```
subprocess.call("/usr/bin/
fetchmail")
```

It waits until the called program has finished and returns the return code to your main Python process. But what if your external program needs to feed in results to your main program? In this case, you can use the command: **subprocess.check_output()**. This is essentially the same as **subprocess.call()**, except when it finishes, anything written out by the external program to stdout gets handed in as a string object. If you also need information written out on stderr, you can add the parameter **stderr=subprocess.STDOUT** to your call to **subprocess.check_output**.

After reading these three articles, you should have enough of the bare bones to be able to build your own version of the J.A.R.V.I.S system. You will be able to fine-tune it to do basically anything that you command it to do. So go forth and order your machines around, and have them actually listen to what you are saying for once.

Full code listing

```
do_task.py
----------
def do_func():
   print "Hello World"

main_program.py
---------------
# You can import your own module to do tasks and commands
import do_task

# You can then go ahead and run any included functions
do_task.do_func()

# You can run system programs directly
import os

# The exit code from your program is in the variable returncode
returncode = os.system("/usr/bin/fetchmail")

# The subprocess module is a better choice
import subprocess

# You can duplicate the above with
returncode = subprocess.call("/usr/bin/fetchmail")

# If you want to get the output, too, you can use
returned_data = subprocess.check_output("/usr/bin/fetchmail")
```



Left The Jasper project has some great documentation that might help guide you further in terms of hardware and software choices

Low-power wireless tagging

Tam Hanna taps into RFID technology and explains how it's possible for your Raspberry Pi to identify objects and persons passing by.

Tam Hanna started out in the embedded space when the PIC 16F84A was hot. After a few years of consulting in the mobile phone space, he is back in black.

Wireless technology is all the rage, whether it's proprietary protocols, Bluetooth 5.0 or LoRA and 5G. RFID and NFC tags lost their lustre as Nokia went belly-up. A well-placed RFID tag can make your Raspberry Pi's life easier, so let's take a closer look.

Placing the old whodunnit question in regards to RFID is a fruitless endeavour. Dozens of companies claim "inventions", with the common trend being logistics benefits. This makes sense from a technical point of view. A smart implementation makes boxes talk with the machines handling them. Removing human intervention eliminates one of the most common source of trouble (to put it politely).

Selecting RFID hardware is a complex task. Enter the four magic letters into a search engine of choice, and you'll drown in device suggestions. The boxouts spread throughout this article provide an overview of the technical trade-offs. The following steps assume 125KHz technologies, but also apply to other readers. Our choice of the Raspberry Pi is but a template. Yours truly's company has done extensive field trials using Shenzhen Xunlong's OrangePi series.

Serial murder

RFID readers are a dime a dozen. The following steps use SeeedStudio's 113020002, which is readily available from various distributors. The screenshot above provides an overview of prices at the time of publication.

Our kit comes with SeeedStudio's standard four-wire interface. It provides both power and data communications via a serial interface. The red wire handles 5V, while the black wire handles ground. Because our unit isn't able to write information to the card, the white pin is not in use. Finally, the yellow pin provides a serial output stream containing card data.

Serial interfaces are a common method of connecting hardware to process computers. The idea behind them is simple. They transmit information as a stream of bits, which the receiver reassembles. These processes take place at a speed called baud rate. Most process computers contain so-called UARTs, which handle timing and protocol issues.

Unixoid operating systems have used serial terminals since the days of dumb terminals. During startup, the kernel emits a stream of logging messages to one or more serial ports. Embedded developers harvest these to find errors in their custom kernels.

Price Comparison — Explore our API option

Supplier	Product	Quantity	Manufacturer	MOQ	Per Unit	Total	DataSheet	PCB	Buy Now
Arrow North America	113020002	26	Seeed Studio	1	€9.7030	€9.70			Buy Now
chip 1 stop	113020002	26	Seeed Studio	0	€9.7030	€9.70			Buy Now
verical	113020002	26	Seeed Studio	1	€9.7030	€9.70			Buy Now
Rapid	113020002	37	Seeed Studio	0	€10.9864	€10.99			Buy Now
MOUSER ELECTRONICS	113020002	21	Seeed Studio	1	€11.2978	€11.30			Buy Now
Digi-Key	113020002	40	Seeed Technology Co., Ltd		€11.5255	€11.53			Buy Now
DISTRELEC	113020002	20	Seeed Studio	1	€12.1600	€12.16			Buy Now
reichelt	113020002	In Stock	SEEED	0	€12.4400	€12.44			Buy Now

Using a price comparison engine such as OEMSecrets could save you a lot of money, when it comes to 'Adding to Basket'.

This situation isn't ideal because the Raspberry Pi's serial port usually isn't available. Raspbian dedicates it to the kernel debugger, but only one function can be assigned at any time. Fortunately, fixing the problem is relatively straightforward.

Start off by opening the *raspi-config* program from your terminal. Next, select Interfacing Options using the cursor keys and choose the Serial option to start the serial interface configuration process. Raspbian first asks if you want a login shell at the serial port – decline this choice. The next question concerns the activation of serial hardware. Confirm it to obtain a general-purpose serial port.

```
| The serial login shell is disabled |
| The serial interface is enabled |
```

Modifying this hardware configuration requires a reboot. After restarting Linux, a few new serial devices pop up in the **/dev** tree:

```
pi@raspberrypi:~ $ ls /dev | grep serial
serial0
serial1
```

Under Raspbian, serial ports are a somewhat touchy topic. As discussed at **www.raspberrypi.org/forums/viewtopic.php?f=28&t=159483**, current versions of Raspbian provide a set of abstraction objects called **/serial***. Be careful to ignore the tty serial ports used on other process computers. They don't cause the program to crash, but don't lead to measurable output at the serial port pins, either.

QUICK TIP

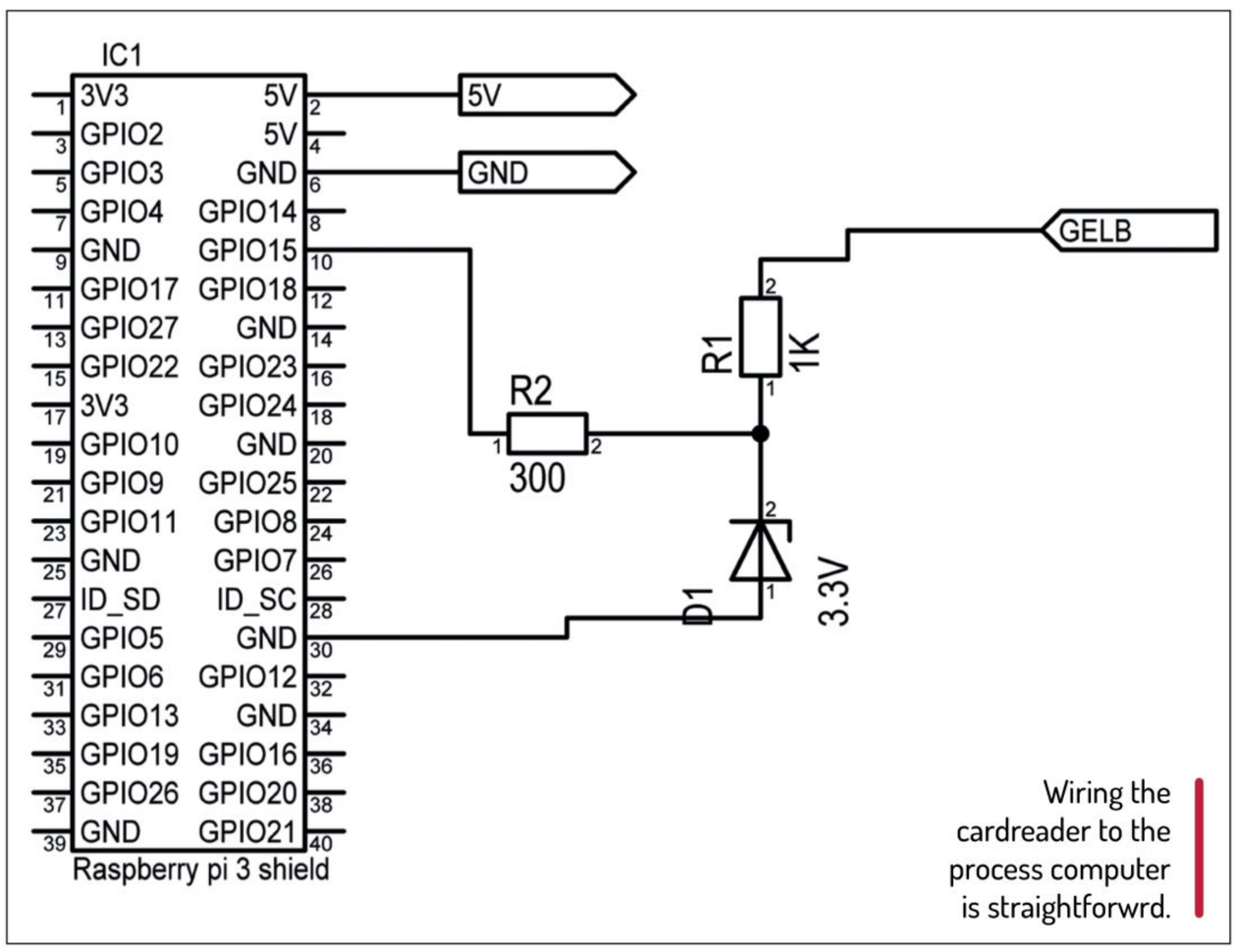
Long data lines act as "magnets" for all kinds of unwanted effects. A few zener diodes or similar protective elements are a good investment as the range increases.

The next step involves connecting the cardreader to the process computer. The diagram (*below*) shows a sample schematic that's proven itself reliable. GELB, incidentally, stands for yellow – SeeedStudio's evaluation boards work with colour coding.

R1 and D1 form a classic zener diode attenuator. D1 always drops its nominal voltage, with R1 limiting the current flowing into the diode. R2 is a little trick to increase the circuit's resilience to overvoltage scenarios.

Understanding its role requires a deeper look into the Raspberry Pi's internal circuitry. Each input pin comes with two protection diodes connected to Vcc and GND. If too high or too low voltages are applied then a balancing current flows through them.

Due to the small size of the diodes, their ability to sustain large balancing currents is limited. This is where our 300 Ohm resistor comes in. In normal operation, minimal current flows into the input, and no voltage falls off across it. As balancing currents flow, the voltage drop raises according to R= U / I. This protects the transceiver from damage.



Wiring the cardreader to the process computer is straightforwrd.

With that out of the way, it's time to start programming the part at hand. We'll stick to C for ease of use. Keep in mind that compiling code requires the inclusion of a library:

```
pi@raspberrypi:~ $ gcc serial.c -l wiringPi
```

In the next step, it's time to set up the entry point and a few Raspberry Pi-specific headers are required:

```
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <wiringPi.h>
#include <wiringSerial.h>
```

The WiringPi library provides a set of helper functions for serial interactions. Loading them saves you from the hassle of having to interact with the UART via the normal file system APIs.

The library can't shield you completely from the underlying operating system. Starting the serial port requires the presence of a string pointing to the UART. Furthermore, the baud rate must be specified. It defines how fast the port works, and must be the same on both sides of the connection.

Speed specifies our reader's baud rate at a comfortable 9,600bps, leading to the following initialisation sequence:

```
int main(){
  int fd ;
  int count = 0;

  if ((fd = serialOpen ("/dev/serial0", 9600)) < 0)
  {
    fprintf (stderr, "Unable to open serial device: %s\n",
strerror (errno)) ;
    return 1 ;
  }
```

Unix's base philosophy states that everything is but a file. As such, accessing the serial port takes place via the file system API. It uses so-called descriptors to describe open files – in our case, the value is stored in the variable **fd**. The rest of the Wiring Library gets initialised after that:

```
  if (wiringPiSetup () == -1)
  {
    fprintf (stdout, "Unable to start wiringPi: %s\n",
strerror (errno)) ;
    return 1 ;
  }
```

Endless loops constitute a classic mistake in desktop programming, but are used commonly when interacting with hardware. In the case of our program, a for loop with empty conditionals is used:

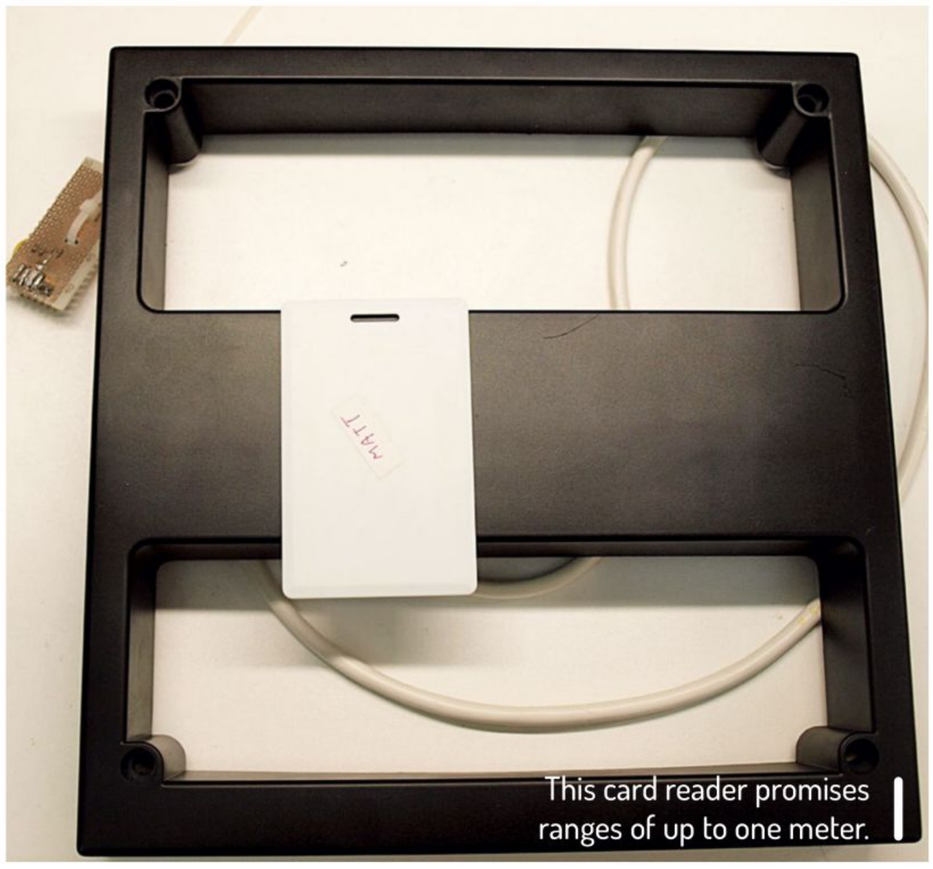
```
  for(;;){
    if(serialDataAvail (fd)>0){
      printf("CARD: ");
      while(serialDataAvail(fd)){
        printf ("%3d", serialGetchar (fd)) ;
        fflush(stdout);
        delay(2);
        count++;
```

» NOT EXACTLY SAFE...

Wiegand-based card readers must not be used in security-sensitive settings. Their word length of 26 bits can be brute-forced easily. This is especially important if your application has to handle large amounts of data and can't implement a back-off delay – if the product at hand doesn't require fast reading, then enforcing a five-second delay after two or three failed authentication attempts should be enough to thwart the average hacker.

This, however, is just part of the problem. RFID readers are a dime a dozen, while RFID cards can be copied and emulated by microcontrollers. An attacker who has gained access to a card can "clone" it, and perform a replay attack to secure access to the goods at hand, even when the card isn't present. This problem can be solved via smart cards. They perform complex cryptographic computations on the data transmitted, thereby requiring a different approach for each challenge-response pair.

Further information on the security concerns involved awaits you at **www.cs.tufts.edu/comp/116/archive/fall2017/bchung.pdf**.



This card reader promises ranges of up to one meter.

» ACTIVE OR PASSIVE?

The active tags used in this article are not the only form of RFID tag. There are active versions that contain a small battery, which provides additional power to the transmitter. This directly leads to a greater range, because the card can contain a readback amplifier and doesn't need to be powered from the reader. Many, if not most high-range systems use active tags.

Sadly, using active tags causes maintenance hassles. Prepare to see many "empty batteries" if untrained or unmotivated workers are involved. A client of the author experimented with active tags in school kid transportation – the project was, as expected, a catastrophic failure because parents forgot to exchange the batteries regularly. Hey ho...

Furthermore, active tags tend to be more expensive than their passive colleagues. This investment can, however, pay for itself in the right scenario. A classic example would be tags with a display – they can, for example, be used to replace address or price labels.

```
    }
  }
```

The baud rate of 9,600 tells us that data is moved into the UART at a speed of 9,600 Bits per second. Delaying the program by two milliseconds ensures that the card reader is faster. Once we detect the presence of a byte via `serialDataAvail`, we can move the data to the command line with the `serialGetchar` command.

Our reading loop terminates if the UART runs out of data. In that case, our program checks if it managed to read data. If that is the case, we send a carriage return to "space up" the results. A 500 millisecond delay then ensures that CPU cycles are not wasted:

```
  if(count>0)
  {
    count=0;
    printf("\n");
    fflush(stdout);
  }
  delay(500);
 }
}
```

We must call `fflush` frequently due to the presence of heavy caching. Omitting these calls can lead to non-working command line programs – in some cases, a delay of up to 10 seconds occurs before information appears on-screen.

At this point, we can run the program for the first time. Compile it with *gcc*, and touch the reader's red antenna coil with a programmed RFID card. The green LED on the reader will light up, and the card's serial number shows up in the command line:

```
pi@raspberrypi:~ $ ./a.out
CARD:  2 48 49 49 48 57 49 69 65 67 68 65 55  3^C
```

Note that each card starts out by sending a byte with the value 2. After that, various ASCII values from 0 to 9 and A to F are sent. Finally, a terminator consisting of a byte with the value 3 is transmitted. When done, press Ctrl+C to terminate the reading loop.

Verification of data input

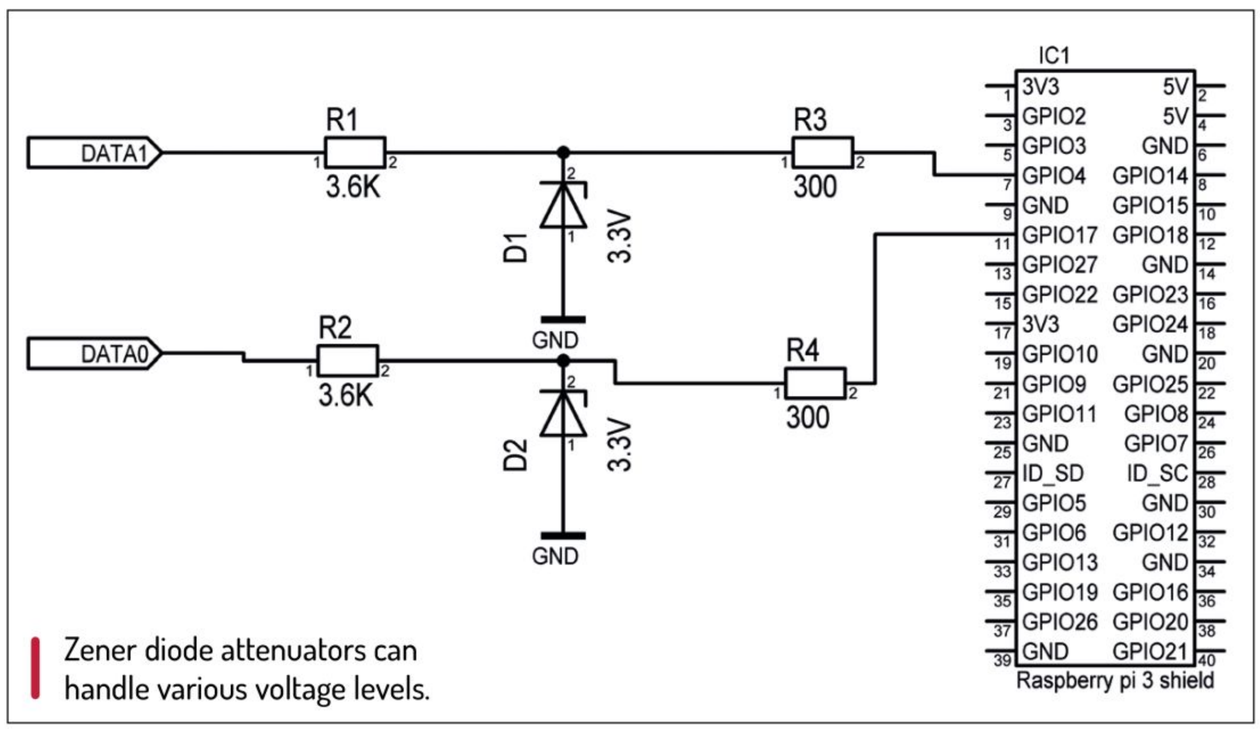
Ensuring the correctness of data transmissions is vital. Your RFID reader might find itself in surroundings polluted with EMI. A careful look at the output above shows that our reader sends 12 bytes. Ten of them are card data, while the remaining two provide a checksum.

Checking the correctness of the transmission requires us to compute a checksum of the data at hand. We compare it to the checksum sent from the reader – mismatches indicate errors.

Our checksum method goes by the name of `processFields`. It takes a data buffer and a read counter indicating the amount of information stored:

```
char processFields(unsigned char* store, unsigned
char readCounter) {
    int aggregate[14];
    int xorGrande=0;
    unsigned char i, j;
    for(i=0;i<readCounter;i++){
        if ((store[i] >= '0') && (store[i] <= '9')) {
            store[i] = (char) (store[i] - '0');
        } else if ((store[i] >= 'A') && (store[i] <= 'F')) {
            store[i] = (char)(10 + store[i] - 'A');
        }
    }
```

Serial numbers stored in RFID cards consist of a sequence made up of 0123456789ABCDEF. The checksum algorithm expects 0 to correspond to numeric zero, with A treated as 10. Transforming the values is best done via character arithmetic. The numbers from 0 to 9 are stored in sequential fields, as are A to F. Moving a ASCII zero to a real zero is as easy as subtracting the value of '0' from it. A to F are handled the same way, with the value 10 added as an offset.



Zener diode attenuators can handle various voltage levels.

Next, we join the four bit words into eight bits. The "higher-value" nibble is shifted to the left, and the lower-value-nibble is added to complete the process:

```
    for(i=1, j=0;i<readCounter;i+=2,j++){
        aggregate[j]= (char)(store[i+1] | (store[i]<<4));
    }
```

Now we use the XOR operator to compute the CRC checksum. Simply run it multiple times, adding values to the accumulator over time:

```
    for(int i=0;i<5;i++){
        xorGrande^= aggregate[i];
    }
```

After computing the checksum, the method compares it to the chechsum obtained from the reader:

```
    if(xorGrande==aggregate[5]) {
        return 1;
    }
    return 0;
}
```

Creating the CRC function, sadly, is but part of the game. Instead of outputting data to the command line, we must redirect it to an array. As the actual start-up code doesn't differ much from the one used in the preceding program, we won't print it again:

```
int main(){
 int fd ;
 . . .
 char cardInput[20];
```

Collecting data isn't particularly difficult. Instead of outputting the bytes to the console, an incrementing index addresses cardInput sequentially. Information is then stored there using the delayed loop:

```
 for(;;){
  if(serialDataAvail (fd)>0){
    printf("CARD: ");
    while(serialDataAvail(fd)){
      cardInput[count++] = serialGetchar (fd);
      delay(2);
    }
  }
```

After having acquired a card's worth of data, the CRC method discussed above checks its validity. We send a warning to the command line if the card is malformed – practical systems are well advised to take more defensive measures:

```
  if(count>0)
  {
    if(processFields(cardInput, count)){
        printf(" CRC OK \n");
    }
    else{
        printf(" CRC FAIL \n");
    }
    count=0;
    fflush(stdout);
  }
  delay(500);
 }
}
```

Simple RFID readers are vulnerable due to the relatively short length – if the system responds fast, an attacker can brute-force all possible card codes. Prevent this via a back-off algorithm: the first few failed attempts don't cause a delay. The delay increases with the amount of attempts, with logging or live alerting thrown in for good measure.

Running the program once again – usually – leads to successful executions of the processFields function:

```
pi@raspberrypi:~ $ ./a.out
CARD:  CRC OK
```

Go long-range!

Our wireless transmitter is efficient as long as the card comes close. This works as long as targets drop by "on foot", so if a person is in a car, higher range is desirable. A good example is a £100 unit sold by AliExpress. It differs from the 113020002 by the use of a different communication protocol. If a Wiegand-based version is ordered, the reader connects to the process computer as shown in the diagram on page 57.

Wiegand is an ancient protocol developed for magnetic stripe RFID tags. It transmits either 26 or 32 bits, and should not be used for high security applications (see the boxout on page 56). Implementing the protocol poses an interesting challenge. The photograph *(below right)* shows the attenuated output on the two data lines on an oscilloscope.

A major problem involves supplying power to the reader. The vendor prefers 12V of supply voltage, with tests showing the reader to work from about 10V. Its current draw overwhelms the Raspberry Pi even with an efficient boost converter. In standby, more than 260mA are needed – during start-up, spikes of more than 500mA occur. The best way to solve this problem involves an external power supply. Connect its GND pin to one of the grounds of the Raspberry Pi, and use the output to power the card reader via its red line.

You can harvest information via polling or by using an interrupt routine. Because our Raspberry Pi has more than ample CPU power, let's settle for a polling implementation. We base it on the state machine pattern, which is well-suited to tasks such as this one.

Directly addressing GPIO pins is inefficient. The preprocessor enables you to use sensible names in the rest of the program, and is highly recommended:

```
#include <wiringPi.h>
#define pin1 7
#define pin0 0
```

QUICK TIP

Housing antenna coils can be difficult. Prefabricated housings such as the ones by Hammond provide a nice way around the problem – simply tack the coil with a bit of hot glue. A small hole in the housing's frame makes passing the antenna cable in and out an easy task.

» WHAT'S THE FREQUENCY, KENNETH?

RFID tags exist in three different frequency bands. Our example uses 125KHz cards, which are the oldest and most common format. The long wavelength makes them extremely resilient – they can also be attached to humid bodies – think (*human?–Ed*) cattle. Sadly, the reading range is extremely limited: achieving more than a meter or so is difficult even with very large antennas.

13.56MHz cards are commonly known as NFC or HF tags. Their reading range is a bit higher, but the lower wavelength makes them less robust against interruptions. Furthermore, 13.56MHz cards are extremely common – bank cards, fitness membership tags and other items usually work in this frequency range. This is problematic because it leads to a higher probability of "foreign" objects showing up in your reader's range.

Finally, UHF systems, which work from 860 to 920MHz, reach extremely high ranges. They are used in stores for "inventory administration". When bought in bulk, their tags are cheap, but placement isn't easy because of the extreme impedance sensitivity.

In the next step, we need a set of fields. Two clock types enable us to track time passed, while the enum stores the current execution state of the program:

```
enum {inStasis, read0, read1, waitloop} currentState;

clock_t delta;
clock_t wartezeit;
```

Efficient state machines live off endless loops. `for(;;)` is a neat way to create one. Inside, one usually finds a switch statement iterating over the cases. Our program starts out with the idle state, which is entered if no card data transaction takes place:

```
for (;;){
        switch(currentState)
        {
        case inStasis:
            if(digitalRead(pin0)==LOW)
            {
              printf("0");
              currentState=read0;
            }
            if(digitalRead(pin1)==LOW)
            {
              printf("1");
              currentState=read1;
            }
            break;
```

State handling usually starts out by checking causes for a state change. In our case, the card reader will pull one of the data lines down to transmit a bit. We react by calling `printf()` and modifying the `currentState` variable to enter the "reading" state.

Our next job involves waiting for the read to end. As a zero and a one end with the same condition, we can handle `read0` with a fall-through to `read1`. The comment isn't necessary, but makes the program's intent clearer to people maintaining the code later:

```
        case read0:
            /*FALLTHROUGH*/
```

Wiegand's defining attribute is that data lines are pulled high if no transmission takes place. Should our program detect the return to idle after a transmission, the next job is finding out whether the entire card's worth of data was transferred:

```
        case read1:
            if(digitalRead(pin0)==HIGH &&
digitalRead(pin1)==HIGH)
            {
              currentState=waitloop;
              wartezeit=clock();
            }
            break;
```

We accomplish this via a special state called `waitloop`. It's activated only after a bit of information was obtained from the card reader, and terminates after 55msec of inactivity. Accomplishing this behaviour requires us to store the current time in a variable via the `clock()` function as our program enters it.

`Waitloop` is a state machine of its own. It starts out by checking if the card reader sent another byte – if that's the case, we return to the read states:

```
                case waitloop:
                    if(digitalRead(pin0)==LOW)
                        {
                          printf("0");
                          currentState=read0;
                        }
                    if(digitalRead(pin1)==LOW)
                    {
                      printf("1");
                      currentState=read1;
                    }
```

If waitloop doesn't find the transmission of additional information, we proceed to checking the amount of elapsed time. If it exceeds the 55msec "dead time" threshold, we send a carriage return and return to the idle state for good:

```
                    delta = clock()-wartezeit;
                    delta = ((float)delta)/
CLOCKS_PER_SEC *1000;
                    if(delta>55){
                        printf("\n");

currentState=inStasis;
                    }
                    break;
        }
}
```

With that out of the way, the two input pins are ready for action:

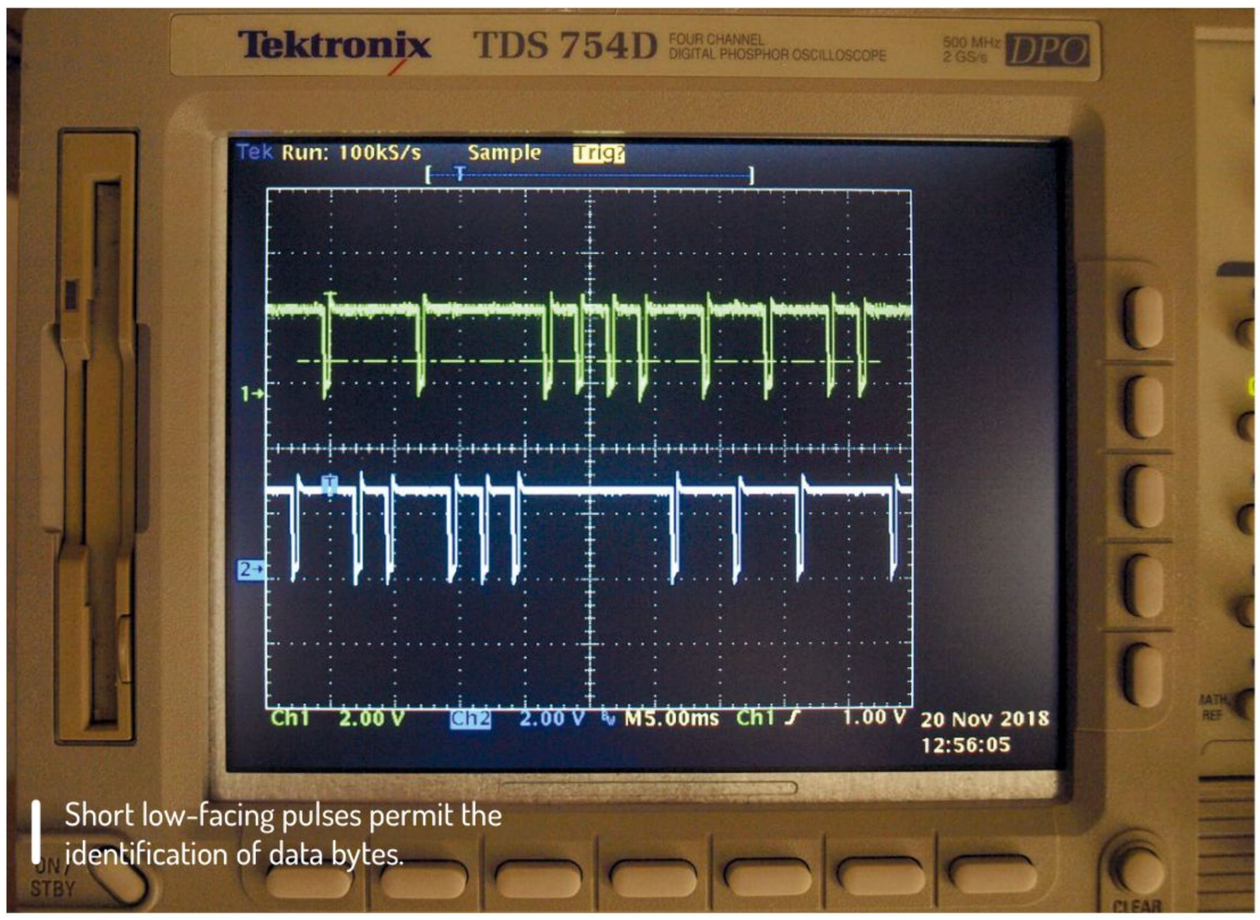
```
int main (void)
{
 wiringPiSetup () ;
 pinMode (pin1, INPUT) ;
 pinMode (pin0, INPUT);
```

At this point, the third example program is ready for action. Our listing shows the result of two scan operations of the same card:

```
pi@raspberrypi:~ $ gcc wiegand.c -l wiringPi
pi@raspberrypi:~ $ ./a.out
01001000111101010110011010
01001000111101010110011010
```

Conclusion

Adding an RFID reader is a cheap way to spruce up all kinds of hardware. Our two smart-home related examples show just a small range of the possibilities. Even more functionality awaits those who master RFID tag writing. Reprogramming tags on the fly enables you to store dynamic information – the sky's the limit.



Short low-facing pulses permit the identification of data bytes.

QUICK TIP

Housings for antennas must not consist of ferrous materials. Using a steel or a cast iron case is a sure-fire way to greatly reduce your RFID systems range.

GameShell kit

Need to take your love of retro gaming everywhere you go? Check out the open source handheld ClockworkPi GameShell, says **Christian Cawley**.

SPECS

OS: Clockwork OS (based on Debian 9)
Display: 2.7-inch 320x240 pixel TFT
CPU: Quad-core Cortex-A7 CPU
GPU: Mali GPU
RAM: 1GB DDR3
Slot: MicroSD 16GB
Output: Micro HDMI
Comms: Wi-Fi and Bluetooth
Sound: Two-channel audio, 3.5mm jack
Size: 85 x 135 x 12mm
Battery: 1,200mAh

The retro gaming explosion seems to be centred around two types of device: dedicated, sometimes licensed, revivals of original machines built to modern specifications, and open source SBCs like the Raspberry Pi.

Veering away from this is the GameShell from ClockworkPi, which despite the similar name has no connection to the Raspberry Pi. It's a modular device that can be bought in kit form only, but the result is a Nintendo Gameboy-like pocket-sized gaming machine that runs a host of retro emulation software.

A successful Kickstarter campaign got the ball rolling with $290,000 worth of backing in 2017, and the console is now available to buy online. But do you really want a device that you have to build yourself? Of course you do!

Featuring a Gameboy-style case, the GameShell is available in three colours, and the case can be easily opened to access the microSD card or, potentially, upgrade or replace a component. The modular nature of the GameShell means that construction can be time-consuming, but once each component is sitting in its own module, the whole thing can be connected and booted inside five minutes.

The GameShell's display, mainboard, speaker, D-pad and battery each sit in their own clear plastic case. These modules are placed into the main case, connected with wires, and the case closed securely. No screws or soldering are needed; instead, the case is secured using a pair of tightening wheels. The mainboard itself features a Quad-core Cortex-A7 CPU, Mali GPU and 1GB of DDR3 RAM. There's also a GPIO port. The console comes with wireless and Bluetooth support, which is useful as there is no USB OTG.

Booting the GameShell launches you into Clockwork OS, a Debian-based OS with a compact UI that is reminiscent of some 1990s mobile-phone interfaces. It comes pre-installed on the microSD card, although you download and install the latest version if you prefer. Alternatively, you can update it over Wi-Fi. Our review device experienced some problems following this, however, requiring some tinkering via SSH to fix it. Fortunately, the ClockworkPi community forum is a friendly place.

As compact and pocketable as the GameShell is, you might prefer to play on a larger device. As such, a Micro HDMI port is also included. Note that the resolution of this will depend on the capabilities of the emulator you're using. In testing, the GameShell worked effortlessly on a high definition TV, but rather than it stretching or upscaling the picture, it was unobtrusively framed with GameShell illustrations.

While its lineage is obvious, the GameShell plays more than just old GameBoy titles.

Software controls are included for brightness and sound; a headphone jack is also provided, alongside the Micro HDMI and Micro-USB power connector. An MP3 player is included, and there's even a version of *Kodi* available for Clockwork OS. But what about the gaming?

Several titles are included to get you started, including *FreeDoom*. MAME, NES and PC Engine emulators are preinstalled, along with a collection of indie game titles. Adding ROMs to the GameShell is best done via your desktop web browser (by enabling the Tiny Cloud feature) over Wi-Fi, although using the SSH feature of your desktop FTP also works. If you prefer to use SSH via a terminal session, this is also supported.

For fans of retro platforms not included in Clockwork OS, RetroArch is also preinstalled. Note, however, that devices with keyboard controls are less suitable for the GameShell. While a Bluetooth keyboard might be connected, you'll find some keymapping is required.

VERDICT

DEVELOPER: ClockworkPi
WEB: www.clockworkpi.com
PRICE: $199 (international shipping from $26)

FEATURES	7/10	EASE OF USE	7/10
PERFORMANCE	8/10	VALUE	7/10

If you don't mind building the console yourself, the GameShell is lightweight, affordable, and programmable.

» **Rating 7/10**

Create an information radiator with DAKboard

Les Pounder shows us how to create a low-cost, Raspberry Pi-powered family calendar using an old monitor.

OUR EXPERT

Les Pounder is a freelance maker who works with organisations such as the Raspberry Pi Foundation to promote maker skills.

The term 'information radiator' is a typically obscure tech way of saying 'device that displays information on a screen'. This information can range from simple news feeds and adverts to complex service status. They are typically used in hotel lobbies to display news, weather and adverts, and in tech offices where developers need to see the status of their work at a glance.

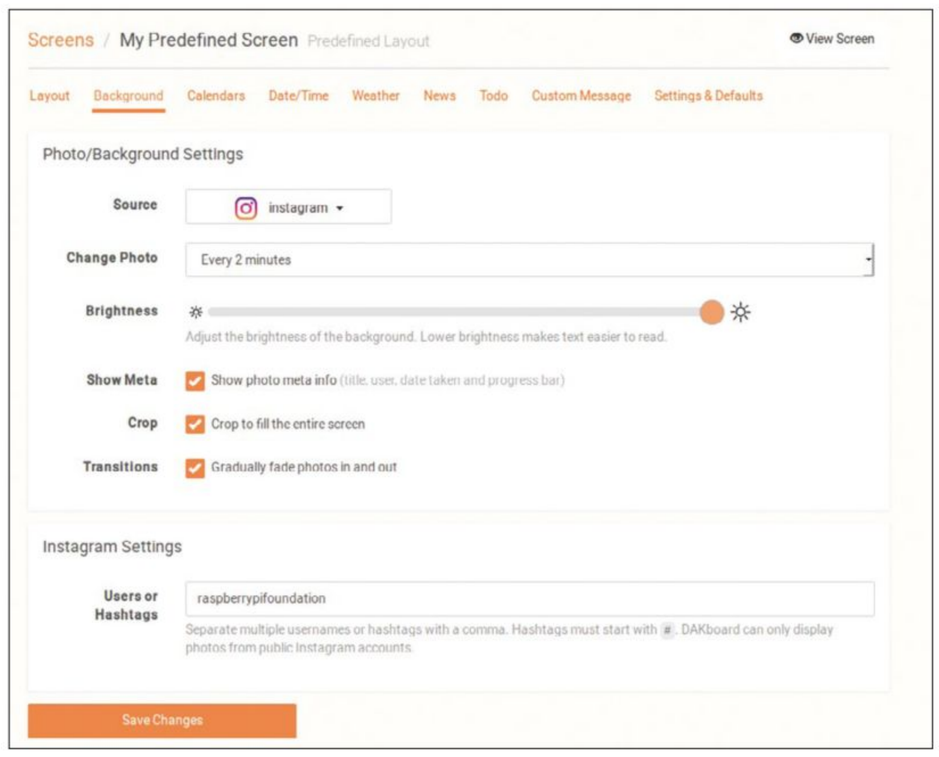
In this project we're going to make our own simple information radiator using *DAKboard* and a Raspberry Pi. The best type of Pi for this project is any of the Pi 3 models or a Pi Zero W, as they all come with built in Wi-Fi connectivity – but if you have an older model you can easily add your own Wi-Fi dongle.

DAKboard is a customisable display – available as a physical device, or just as a service – that can show photos, RSS feeds, weather and calendars. To start using *DAKboard*, sign up for a free account at **https://dakboard.com** and then log in. The first screen we encounter is called Screens, and here we can manage a number of them. For this tutorial we'll edit 'My Predefined Screen' so click on it to start editing.

The first task is to set the layout of the screen, so try out a few options. Simply click an option and then Save Changes. In the top right of the screen is 'View Screen': open this in a new tab and it will display your screen. There should be a large image on the left and a black space on the right. In the bottom left is the weather. All of these options will be edited later; for now, just change the layout to 'Big Calendar'. The next step is to alter the background image, so click Background and in the new screen click Source to choose the source of your images. We're going to use Instagram as the image source, specifically images from the Raspberry Pi Foundation. In Instagram Settings, add the user `raspberrypifoundation` and click Save Changes. Now go back to the tab we opened earlier and click refresh to see images from the Raspberry Pi Foundation's Instagram feed.

Moving on, we can set up our calendar. In this case it is the **LXF** Family Calendar, reminding the team to make the tea, drink beer and do work *(not in that order though – Ed)*. First we enable the calendar and set it to show seven days' of content at once. In the Calendars section, we provide an *iCal* URL, which in this case is from a Google Calendar we set up for this tutorial.

To get the *iCal* URL for a Google Calendar, open Google Calendar and then click the three dots next to the calendar. Click 'Settings and Sharing', scroll down to 'Access Permissions' and click 'Make this calendar public'. Now scroll down to 'Public Address in iCal format' and copy the URL below it. Paste this URL into the Calendars section in *DAKboard*, and give the calendar an appropriate name. Click on Save Changes



The ability to use live images from your Instagram feed means that your family's screens can see images from around the world, as they are posted to Instagram!

YOU NEED

- Any Pi
- Latest Raspbian Lite
- Internet
- PC monitor

DAKboard is really simple to use and the free tier has just enough functionality for home users. Professional users may want to upgrade to the paid tiers.

and then go back to the other tab and refresh to see the calendar appear.

In the next section we can alter the time and date according to where we are in the world – this is pretty self-explanatory. Next, we configure the Weather tab, where can set the source of our weather data to one of three providers. For this tutorial we'll set it to OpenWeatherMap and then set the location to where we live – *DAKboard* will help you find the nearest location. We set the units to those required, in our case Celsius. We lastly set the extended forecast to five days. Click Save Changes and then go back to the screen tab, click refresh to see the results.

Feed me

The next step is to add a news feed to the screen and for this we go into News and enable 'RSS'. In the News Feed URL, we paste the 'Top Stories' news feed from BBC News, which can be found at **www.bbc.co.uk/news/10628494** – but feel free to use any RSS feed you want. Save Changes and then go back to the other tab, refresh and watch as the news is displayed in the bottom right of the screen.

In the next section we can set up a To-Do list; for this we've used our *Todoist* account, which was easily linked to *DAKboard* via a short wizard in this section. Then we chose the correct list to show, in this case our to-do list for today. Click Save Changes and then go back to the other tab and refresh. You can see the list appear at the left of the screen, just under the time and date.

With all of the configuration complete, go to 'Settings & Defaults', give the Screen a name and click Save Changes. Now write down the Private URL link as we will need this later. Also, do keep this link private...

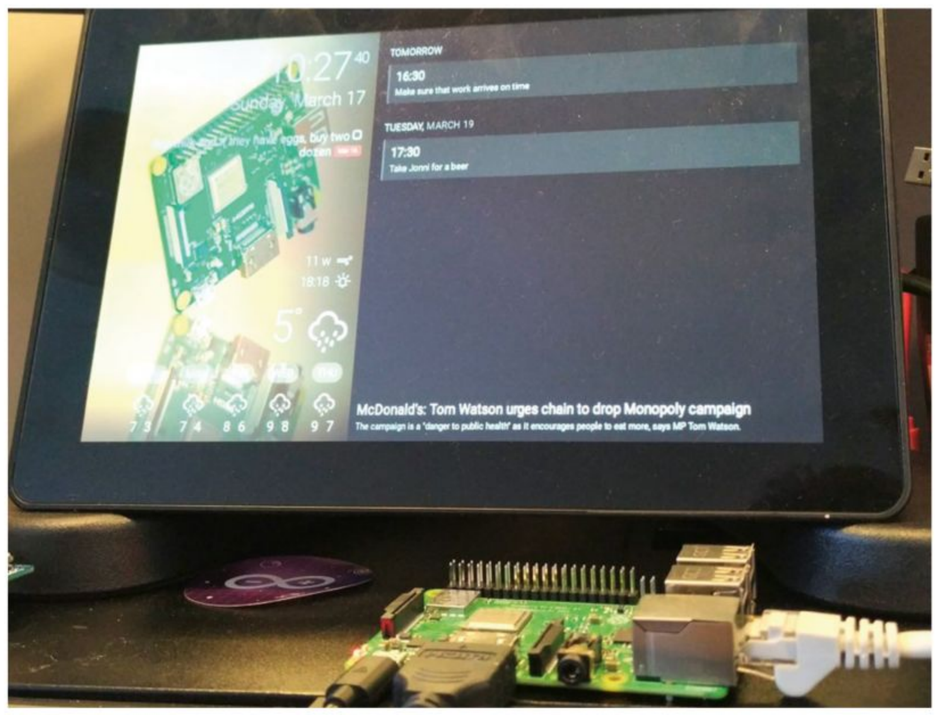
Pi Lite

Our Raspberry Pi will need the latest Raspbian Lite release. This gives us the bare operating system and a web browser, ideal for our needs. We also need an internet connection in order to load the *DAKboard* screen. On the Raspberry Pi we open a terminal and edit a file that enables the web browser to automatically open when the operating system starts. In the terminal, type the following command and press Return.

```
$ sudo nano /etc/xdg/autostart/RPI-LXF.desktop
```

We are going to create a config file that creates a desktop entry for an application. This will open the *Chromium* browser when the desktop loads, set the window size to 1920x1080, and specify fullscreen kiosk mode to prevent anyone exiting the application. It will also suppress error messages and run in incognito mode. Remember your *DAKboard* private URL from before? Replace `PRIVATE URL FROM DAKBOARD` with that address in the following:

```
[Desktop Entry]
Type=Application
Name=RPi-LXF
Comment=Information Radiator
NoDisplay=false
Exec=chromium-browser PRIVATE URL FROM
DAKBOARD --window-size=1920,1080 --start-fullscreen
--kiosk --incognito --noerrdialogs --disable-translate
--no-first-run --fast --fast-start --disable-infobars
--disable-features=TranslateUI --disk-cache-dir=/dev/
null
NotShowIn=GNOME;KDE;XFCE;
```



Information radiators come in all sizes. Here's our test setup using a 10-inch screen that handily reminds us of key information.

If you want to know more about what these command line arguments do, especially in the `Exec=` line, have a look at the Raspian Lite documentation – there are some very handy commands in there.

To save press Ctrl+O, then press Return, then Ctrl+X to exit. Still in the terminal, our next step is to edit a file that will prevent the display from blanking after five minutes. In the terminal, type:

```
$ sudo nano /etc/lightdm/lightdm.conf
```

In the file look for the line

```
#server-command=X
```

Change it to

```
server-command=X -s 0 dpms
```

To save press Ctrl+O, then press Return, then Ctrl+X to exit. Now reboot the Pi and once it has successfully rebooted to the desktop, it will automatically load our *DAKboard* screen!

There you have it: our own information radiator for the home, office and datacentre.

QUICK TIP

You can check how your screen looks by clicking View Screen at the top-right of the *DAKboard* interface. The only way to share this screen with the Pi is to use the Private URL which is in the Settings & Defaults menu. Remember, do not share this URL with others as it contains your private information!

» ROTATING THE DISPLAY

Your information radiator will most likely be in landscape orientation, just like the monitor on your desk. But what if you want to show off your *DAKboard* using a portrait display? Well, rotating the screen is quite simple – perhaps not as simple as on Ubuntu/Fedora desktops, but all it requires is a little basic terminal work. In Raspbian, open a terminal and type:

```
$ sudo nano /boot/config.txt
```

Scroll to the bottom of the file and add the following two lines:

```
#ROTATE HDMI
display_hdmi_rotate=X
```

Change the `X` to `1` for 90 degrees, `2` for 180 or `3` for 270 degrees. To save press Ctrl+O, then press Return, then Ctrl+X to exit.

Reboot the Raspberry Pi and your display should now be rotated and ready for use. If the effect is not as desired, change the value of `X` to another number between 0 and 3. To turn off the rotation, just add a `#` to the start of the line and the system will ignore it.

Stream PC games around your home

Christian Cawley discovers that you don't need Steam Link to stream games from your PC to a Raspberry Pi.

OUR EXPERT

Christian Cawley recently had a crown fitted (the tooth kind) after attempting to eat his Raspberry Pi. This is a single blight (bite?) on a technical support and writing career of 15 years.

Raspberry Pi computers have become increasingly popular since their first appearance, but remain far too low a spec for serious gaming. However, thanks to game streaming software like *Parsec*, you can play games on any TV in your home, streaming the title from a dedicated gaming PC through a suitable Raspberry Pi. The result is a slick gaming experience that to all intents and purposes looks as though it is running on the Pi.

While a similar experience can be enjoyed with the *Steam Link* software from Valve (which turns the Raspberry Pi into a version of the now-discontinued Steam Link box), *Parsec* adds some useful features to enhance the gaming experience.

By the time you've set this up, you'll be re-evaluating just how you game on your PC. Whether you're using a Linux PC for gaming, Windows or even macOS, there's a strong chance you might be considering locking it away into a dedicated, ventilated area. Relying on streaming via the Raspberry Pi is a realistic alternative thanks to *Parsec*.

There's almost no catch with this: you can play and stream any PC game, overcoming the Pi's hardware limits. The only unavoidable downside is that a certain amount of lag will be introduced to gameplay, as data needs to be encoded and decoded at each end, as well as travel to and from each computer.

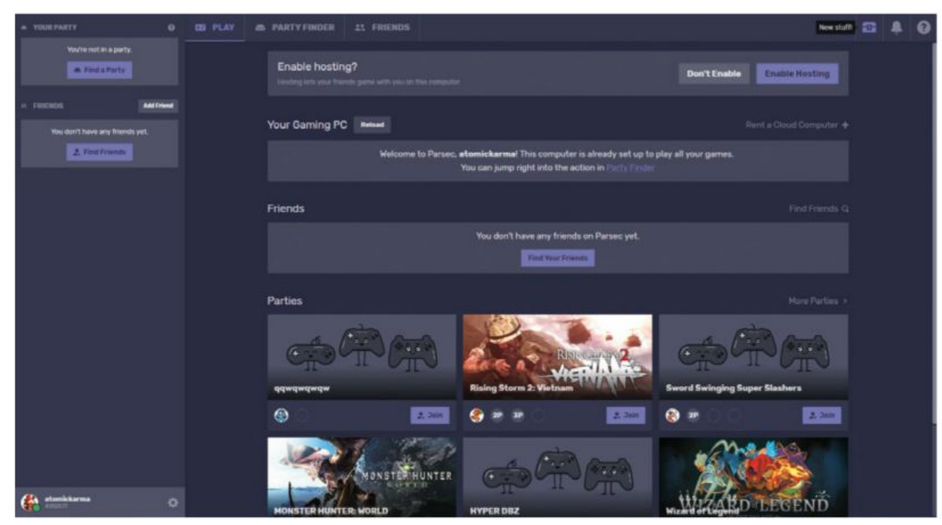
Nothing unusual is required to get *Parsec* running on your computer and a Raspberry Pi. A fresh copy of Raspbian Stretch is a smart idea, written to your Raspberry Pi the easy way with *Etcher* (**www.balena.io/etcher**). We used a Raspberry Pi 3B+, but any Raspberry Pi 2 or later should work fine. For the best results, an Ethernet cable is needed, along with a USB mouse and keyboard, HDMI cable and a suitable USB game controller. Your PC with the games installed on it should be connected to the same network that you plan to stream across.

QUICK TIP

It's wise to be able to disconnect your Pi from your *Parsec* gaming PC. To do this, hit Ctrl+Alt+~. Check the Keybindings screen for more shortcuts.

Pi-sec

Installing *Parsec* on your gaming PC turns it into a game streaming server. It's compatible with Ubuntu 18.04 LTS, Windows 7 and later (32-bit and 64-bit) and macOS 10.9 and later, and you'll find a version suitable for your OS at **http://bit.ly/LXF251-parsecDL**. While it downloads, take the time to visit the *Parsec* account creation page at **http://bit.ly/LXF251-parsecsignup**. After installation (remember to agree to installing controller support), run *Parsec* and login with the account you created on the website. Click Enable Hosting, then 'Add a computer'. In the next screen, choose 'Add your own gaming computer', then wait for the ID to be created. Click Play then Manage to see the ID code; make a note of this, as you'll need it later.



Parsec offers a number of additional game streaming features, including online parties.

Your computer is now connected to the *Parsec* network, awaiting the connection of a target device: your Raspberry Pi. With your Pi booted up, open a terminal (or connect via SSH) and update its sources and software.

```
sudo apt update
sudo apt upgrade
```

Next, open the configuration tool:

```
sudo raspi-config
```

Here, open Advanced Options > Memory Split and set the value to 128. Use the arrow keys to select OK, then exit the configuration tool. Raspbian will prompt you to reboot, so select Yes to proceed.

With the Pi rebooted, use *wget* to download the *DEB* file **parsec.rpi-deb**:

```
cd ~
wget https://s3.amazonaws.com/parsec-build/package/parsec-rpi.deb
```

This can take a while, so it's worth taking this time to ensure the games you intend to play are up to date on your gaming PC. Waiting for an update when you're attempting to try out streaming can prove frustrating!

After downloading the *DEB* file, install it with:

```
sudo dpkg -i parsec-rpi.deb
```

Parsec is now installed, but before running you will need to ensure the correct configuration. For this, you'll need the keyboard and display connected to the Pi (rather than a remote SSH connection), with a terminal session running. Enter the `parsec` command and when prompted enter the username and password you created earlier on your gaming PC.

A Server List will appear, displaying your gaming PC by name. Input the corresponding number to connect to the server. In the event no server is listed use this command instead:

```
parsec server_id=<YOUR_ID_CODE>
```

Substitute `<YOUR_ID_CODE>` with the ID code you kept a note of when setting up *Parsec* on your PC. It shouldn't take long for the gaming PC and Raspberry Pi to connect. Once this happens, the display connected to your Pi will present the desktop environment of your gaming PC. Any input devices connected to your Raspberry Pi will control your PC, including USB game controllers.

Although *Parsec* delivers a game streaming experience with minimal lag, you may want to tweak your mouse settings slightly. This will speed up the mouse on the remote PC, useful for launching games or playing mouse-driven titles.

Open a terminal to edit **cmdline.txt**:

```
sudo nano /boot/cmdline.txt
```

Next, look for the line that starts:

```
dwc_otg.lpm_enable=0
```

Use the arrow keys to skip to the end of the line and add the following:

```
usbhid.mousepoll=8
```

Save and exit with Ctrl+X, then restart the Pi:

```
sudo reboot
```

Any mouse lag issues should now be resolved.

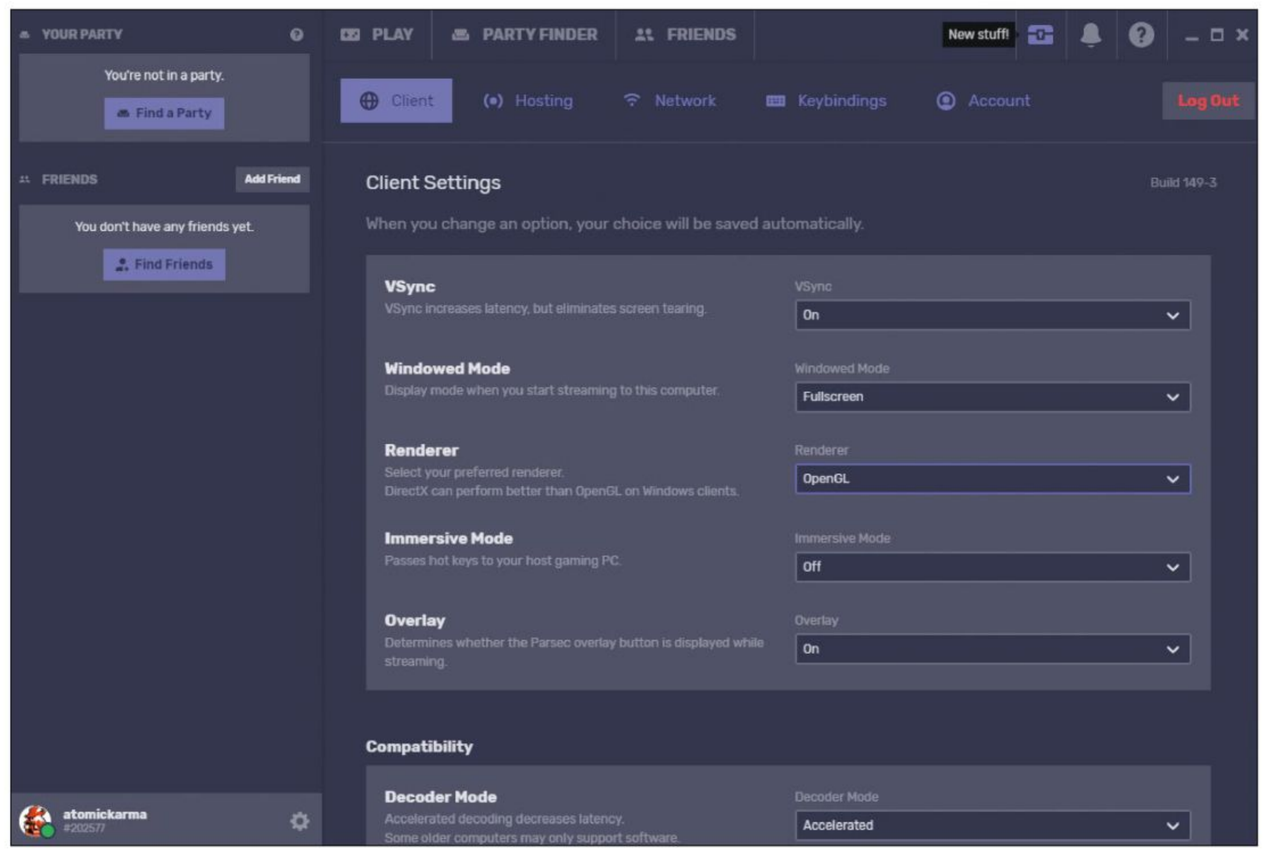
Tweak away

As noted, an Ethernet cable connection is recommended for this project, both for the gaming PC and the Raspberry Pi, via your router. If this isn't possible, at least one of the computers should be connected via Ethernet, and the other over Wi-Fi. Relying on Wi-Fi for both devices isn't recommended unless they're both in the same room and your router has a 5GHz band.

Failure to do this will result in a gaming experience that isn't worth the bother. Slow, stuttery gameplay is a waste of time, which is why ensuring *Parsec* is customised for the best performance can help considerably, so we'll do that now.

On your gaming PC, open the *Parsec* server app and take the time to check out the options in the Client settings. Find the Settings cog to access this, then click Client. You'll see options for VSync (default: On) and whether you should use Windowed Mode. Fullscreen is the norm here, but there might be scenarios in which Windowed works better.

There's also the option to switch your renderer; OpenGL is an option, although Windows gaming PCs will almost certainly run better with DirectX. Meanwhile, the Accelerated Decoder Mode is a default option that you should leave alone. Further settings can be tweaked in Hosting. Here, you can set local hosting options, but there is also the option to Enable Hosting for multiplayer gaming *(see box, below)*.



Get the client settings right for trouble-free game streaming from a gaming PC to Raspberry Pi.

Under Basic Settings, you'll see a Bandwidth Limit. The default is based on your network speed, but you can spend some time finding the right option for this once you have settled upon locations for your gaming PC and Raspberry Pi. If they're likely to move around the house – perhaps because your gaming PC is a laptop – then leaving this set to the default is wise.

Resolution, meanwhile, defaults to Autodetect, but several options are available. In most cases, it probably doesn't need to be set to 4K unless you're hardcore about gaming. You'll need the best Ethernet cables available to get the best results from this – not to mention a suitably powerful PC.

In the same screen, you can set Approved Apps for remote connections during multiplayer, although you might be more interested in Advanced Settings. Here, set H.265 to Off, and set Display Adaptor to use your PC's video card. All you need to do now is launch any game on your PC (from Steam or elsewhere) and start playing via your Raspberry Pi.

» BEYOND BASIC GAME STREAMING

'Party' and 'friends' are found on almost every screen in the *Parsec* server app. The reason is simple: not only does the software stream games to other computers such as the Raspberry Pi, it lets you join multiplayer gaming parties with online friends.

As you'll spot in the Account screen, there's even support for the Discord voice and text chat system with *Parsec*. You'll also spot the option to sign up to *Parsec Warp*, which adds a theme to the *Parsec* environment and gives early access to Beta features.

Online multiplayer parties can be accessed via the Party Finder tab, where a collection of games can be found. Prefer to host your own party? Ensure you've enabled hosting in the Hosting tab, then click 'Create a party' in the Party Finder screen or left-hand panel. Running games will be displayed here, enabling you to quickly set up the game party. Keep the Private switch off if you don't want your game party to be publicly listed.

As if that wasn't enough, *Parsec*'s ability to stream any PC game means that you can even stream emulated games to your Raspberry Pi. This is great if you've been exploring the Pi's possibilities as a retro-gaming machine, but found it can't handle a particular platform.

Creating a classic system demo

Join the intrepid **Les Pounder** as he takes you back to the late '80s, when the Amiga 500 reigned supreme and demos were an art form.

Les Pounder is a freelance maker who works with organisations such as the Raspberry Pi Foundation to promote maker skills. He blogs at bigl.es

For this tutorial we shall take our first steps in creating a demo. These were popular in the '80s and '90s on the Amiga and Atari computers. They showcased the power of the computer and the prowess of the coder. Typically, these demos (**http://ada.untergrund.net**) were used by rival "crack teams" who broke copy protection on games. But they grew into an art form that led to many bedroom coders becoming big names in the computing industry. We shall be using Pygame Zero to create our own demo using a little Python code and some assets created in image editors and music applications. More on that later.

To write the code for this project we need to install Mu, a simple Python IDE for beginners. We install Mu because it's a great environment in which we can easily write the Pygame Zero code.

To install we need to open a terminal, and type

```
$ sudo apt update && sudo apt install mu-editor
```

With *Mu* installed you can find the application in the Programming menu. Open *Mu* and in the top left look for the Mode button. Click this and select Pygame Zero.

Before we get into the code, note that in your home directory **/home/pi/** there'll be a directory called **mu_code** containing directories that will be used to store the images, sounds and music for this project. Sprites and background images should go into the **images** directory, sound effects into **sounds** and any music into **music**. *Pygame Zero* requires these files in the right place so that we can easily reference them in the project.

Start coding

We start the code by importing a single library and this is the random integer function from the random library. We shall use this to randomly place our sprite:

```
from random import randint
```

Sprites are objects that move on a screen, for example Mario is a sprite in 8- or 16-bit *Mario* games. *Pygame Zero* calls sprites "Actors" and so to use a sprite we need to call the actor by their name. This relates to the name of an image in our Images directory. In this case we used "biglesp" but you should change this to your chosen sprite:

```
sprite = Actor('biglesp')
```

Using the `randint` function we can specify a random place for our sprite to start. In this case we ensure that the sprite is 10 pixels from the screen edge, and then use `randint` to randomly appear at a different height in the window. We use 0,200 as a range, but change this to reflect the height of your window:

```
sprite.topright = 10, randint(0,200)
```

Talking of windows, the next two variables control the width and height of the window and are based on the dimensions of the background image. Ensure that these are changed to match the image that you wish to use:

```
WIDTH = 1096
HEIGHT = 739
```

Next we tell *Pygame* where to find our background music. This is an OGG file in the Music directory. Using the `play() function` without passing a number means that the music will loop forever.

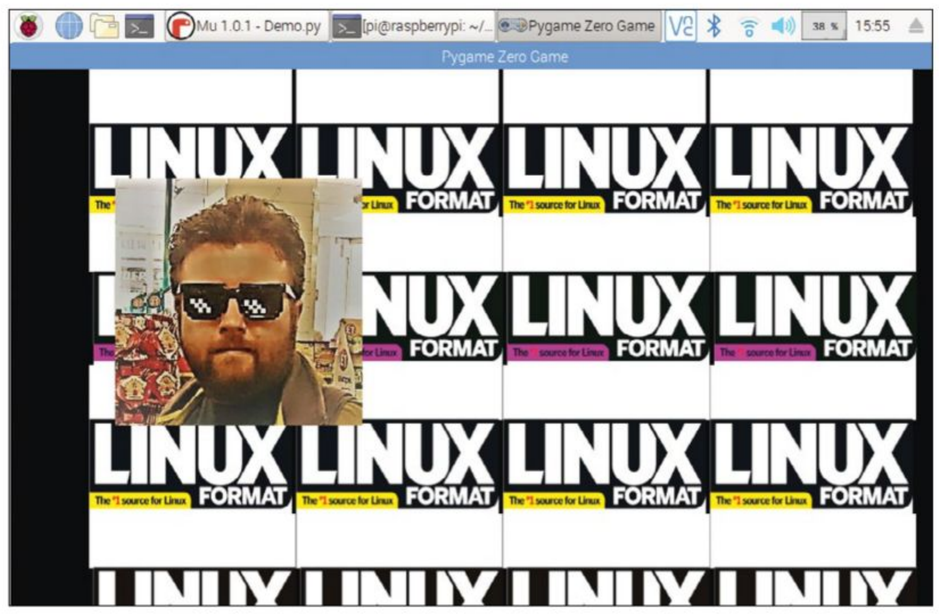
```
music.play('bg_music.ogg')
```

The first function we create is for our screen. This will create the window, clear the contents of the window, then place our background image in the top left (0,0) of the window. This is then drawn on the screen:

```
def draw():
  screen.clear()
  screen.blit('lxf_group.png',(0,0))
  sprite.draw()
```

The next function is called `update` and it's used to update our sprite (actor) so that it can scroll left to right in the window:

```
def update():
  sprite.left += 2
```



Our demo is relatively simple, but it demonstrates many core concepts needed to build a video game using Python and Pygame Zero.

YOU NEED

- Any Pi
- The internet
- The code: https://github.com/lesp/LXF247-Pygame-Zero/archive/master.zip

Still inside the function, let's add some interactivity using a series of conditional statements that will check to see if we've pressed a key on the keyboard. The first two statements control the height of the sprite in the window. This uses the W key to move upwards and the S key to move downwards.

```
if keyboard[keys.W]:
    sprite.y -=1
elif keyboard[keys.S]:
    sprite.y += 1
```

We then use the A and D keys to rotate the sprite by changing the angle of the sprite:

```
elif keyboard[keys.A]:
    sprite.angle += 1
elif keyboard[keys.D]:
    sprite.angle -= 1
```

The next two conditions to assess are firstly a test to see if the sprite has gone beyond the boundary of the window. If that's the case then the sprite will move back to the far left using the `sprite.right` object. The last condition to test checks if the user has pressed the ESC key to close or quit the window:

```
if sprite.left > WIDTH:
    sprite.right = 0
if keyboard[keys.ESCAPE]:
    quit()
```

Click the sprite

The next function is designed to react to mouse input. If our sprite is clicked using the mouse, then it's "collided" with the pointer and so it'll print to the Python shell. Then it will call the function `set_sprite_click()`, which we shall create later. But if the sprite isn't touched then it prints another message to the Python shell. These print statements are optional, but are handy when debugging:

```
def on_mouse_down(pos):
    if sprite.collidepoint(pos):
        print("Got me!")
        set_sprite_click()
    else:
        print("You can't catch me!")
```

Next, we create the function that will be called when the sprite's been clicked. In this function we swap the sprite image for an inverse of the original image, which gives us visual feedback to show that the sprite has been clicked. For audio feedback we use the *Pygame* sounds class to play a cowbell sound effect. The last line in the function is a callback to a function that will return our sprite to `normal` after one second. We use *Pygame's* `clock.schedule` function because it won't block the code from running, unlike using the time library. Using `clock.schedule` enables the rest of the code to carry on while we wait for the sprite to return to normal:

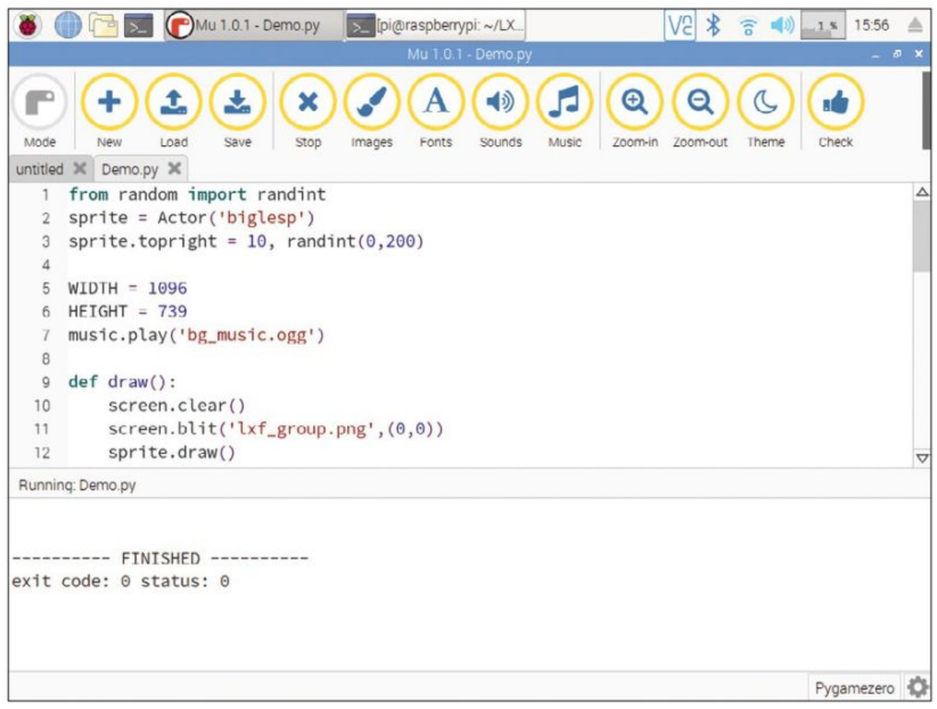
```
def set_sprite_click():
    sprite.image = 'biglesp_inv'
    sounds.cowbell.play()
    clock.schedule_unique(set_sprite_normal,1.0)
```

The final function resets the sprite to normal after it's been clicked:

```
def set_sprite_normal():
    sprite.image = 'biglesp'
```

So now we can save the code as **Demo.py** and then click Play to run the code. We should see a new window appear, and our demo music should be playing.

Pygame Zero is an impressive introduction to game development using Python and this quick project should have whetted your appetite enough to learn more and via its website. Do just that by visiting **https://pygame-zero.readthedocs.io/en/stable/index.html** today!



Mu is an easy-to-use Python editor that removes the "sting" from learning to code. It's ideal for those of us who want to "just get started" writing code to make games.

QUICK TIP

Pygame Zero can also be run from the terminal. Type pgzrun followed by the name of the file you wish to open. This must be run from inside the same directory as the project file.

The Pygame Zero website has a detailed guide on how to use the framework and a reference on all of the objects used to create a game.

» CREATING THE DEMO'S ASSETS

Our demo uses two images and one audio file. But where can we get them from?

Well, if we need images we can use the artwork at **openclipart.org** because all of the images there are royalty free and in the public domain. For audio, take a trip to YouTube, which has a selection of copyright-free music tracks and sound effects. Go to **www.youtube.com/audiolibrary/music** to see the vast selection on offer. You can also visit **https://cchits.net**, which is a curated list of the top 15 Creative Commons music tracks.

But if you want to create your own assets then consider using *Inkscape* or *Gimp* to create images. *Inkscape* is a vector image editor and is excellent for accurate drawings. *Gimp* is similar to *Photoshop* and can be used to enhance an existing image.

For music we found that *Sonic Pi*, a music editor that uses code to make music, is a great place to start if you want to compose a simple tune. You can then output the audio and edit it with *Audacity* to trim and add effects to the loop.

So there are options open for those of us that need a quick solution to asset creation, and those that prefer to hand-craft their own assets.

Emulate the Atari ST on your Raspberry Pi

Pull on a pair of baggy jeans and stripy top, and travel back in time to reveal how to transform your Pi into a fully fledged Atari machine.

You'll almost certainly remember the late 80s when the eight-bit computing revolution gradually gave way to the rise of 16-bit machines. Atari and Commodore stood at the forefront of this revolution, with the Amiga eventually winning out. And thanks to the brilliant Hatari (**http://hatari.tuxfamily.org**), you can now resurrect the ST (and its successors) on your Raspberry Pi.

What you'll need

- Raspberry Pi 2 or better
- *Hatari* **http://hatari.tuxfamily.org**

01 First steps

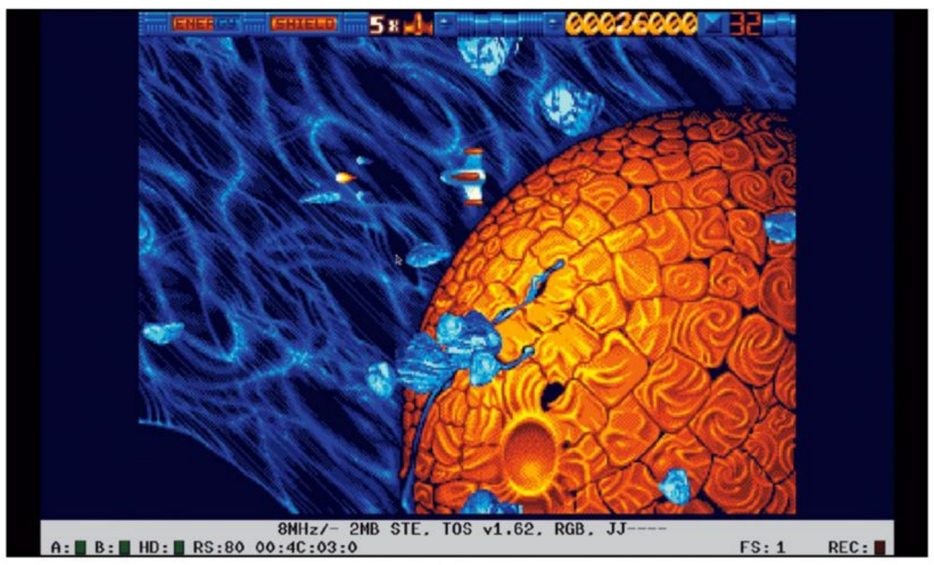
You'll need a desktop environment to run Hatari – we're using Raspbian PIXEL. The latest version of Hatari uses SDL2 to render graphics, so you'll need to enable the OpenGL driver, noting that it's still experimental and may be slightly unstable. It also means Hatari requires a Pi 2 or better to run optimally.

To enable the driver, open a Terminal window, type sudo raspi-config and hit Enter. Select Advanced Options>GL Driver>GL (Full KMS) OpenGL desktop drive with full KMS. Reboot when prompted.

From here, installation is simple: open Pi>Preferences>Add/Remove Software, type 'hatari' into the Search box and you'll see one hit: tick this and click OK to install it. Once installed, open the Pi>Games menu – you'll see two entries: Hatari and Hatari UI. Choose the former option, where you'll immediately come across a message about not being able to load a TOS file.

TOS is the Atari's operating system, and comes in a range of revisions and versions depending on which 16-bit Atari machine you're trying to emulate. Technically, you should supply this yourself, but thankfully the hard work of obtaining any version of TOS has been done for you.

Before proceeding, a quick note about legalities: when running emulator software, it's assumed in most cases that you own the original computer as well as any software you play on it. From a legal standpoint, you should really honour this, but practically speaking there's nothing but your conscience preventing you from running whatever you like, even if it's no longer buried away in your loft somewhere.



Right The Atari STE was capable of better graphics as this game – Stardust – clearly shows

Let's begin by obtaining TOS for Hatari. Open your web browser and head to **www.atariworld.org/tos-rom** where you'll find every single version of TOS from the original 1.0 release up to TOS 4.92 for the Atari Falcon. Which one do you need? It depends on what you're planning to emulate. The original STFM model, which covers most bases, runs best on TOS 1.4, but you may want to download TOS 1.02 for some older games. Those emulating the STE want TOS 1.62, while TOS 2.06 (both STFM and STE) is a good choice when copying lots of files via the ST's GEM desktop.

A word to the wise – the TOS 2.06 ROM file has the wrong permissions assigned to it – you need to right-click the IMG file after extracting it and choose Properties>Permissions. Make sure View content is set to Anyone and Change Content is set to Only owner before clicking OK and copying the file across.

If you're determined to stay the right side of the law and have no Atari gathering dust in the attic, try EmuTOS instead (**http://emutos.sourceforge.net/en**). It's is a free version of TOS that can run on physical computers via floppy disk as well as your new Hatari emulator, but note it'll be hit and miss what games and software you'll be able to run on it.

When it comes to downloading each version, choose the English UK link. A zip file will be saved to your Downloads folder, ready for use shortly. You now have all the elements that you need to start configuring Hatari itself. The walkthrough will reveal the key steps that you need to follow to complete this awesome project.

02 Download floppy disk images

As things stand, your Hatari setup is now a fully functional Atari machine – the only thing missing is software. Most games come on floppy disks, and once again the painstaking process of converting ST floppy disks into image files recognisable by your Pi has been done by others. All you need to do is locate and download these images, which come in MSA, ST or STX format.

There are a wide range of sites offering images – Google "Atari ST ROMs" for a long list. If you can cope with the fact it's French (Chromium will offer to translate the pages for you),

then we like **www.planetemu.net/roms/atari-st-games-st** in particular. Here you'll find some games are listed several times based on who's submitted the images – try one and then download another if it doesn't work.

As with ROMs, floppy disk images are downloaded in zip format, so you'll need to extract them first, where you'll be presented with files in .ST or .STX format (thoughtfully associated with Hatari, so you'll see a nice floppy ST icon to help identify them). These can be stored anywhere on your hard drive, but we recommend setting up a dedicated folder – say atarist – inside your home folder for easy access.

03 Play ST games

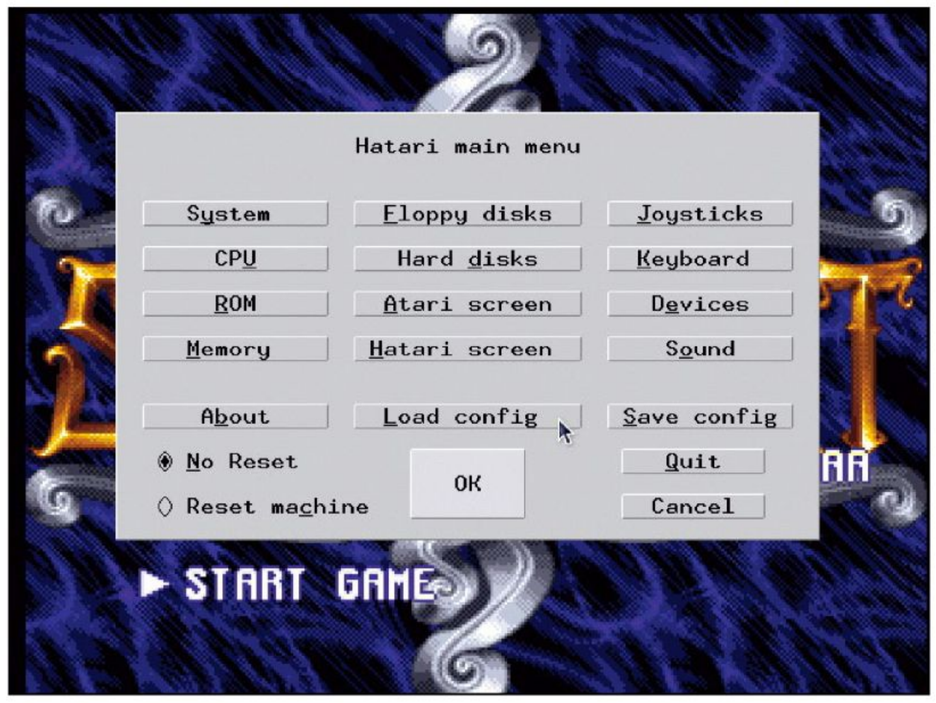
There are two ways to load floppy disks into Hatari. The simplest thing to do is simply browse to the folder containing your disks and double-click the .ST or .STX file. Hatari will launch using the configuration you've saved under hatari.cfg and then – after a pause that can be quite lengthy – the familiar green ST desktop will appear or the game will automatically load. Double-click the A drive icon if necessary, followed by the game file to start loading it.

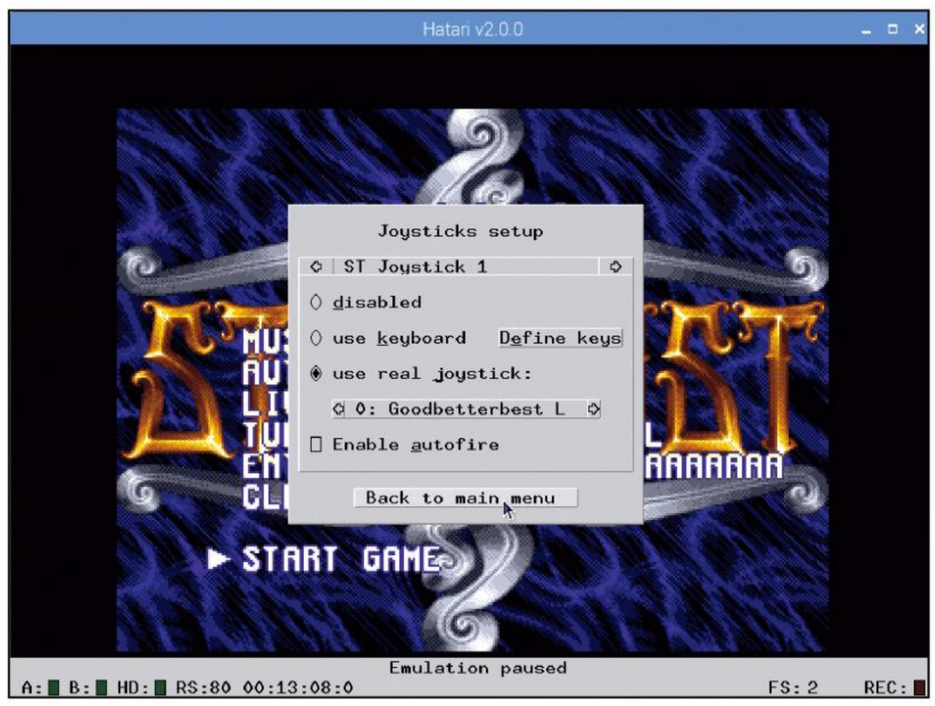
If you plan to play a game that comes on two floppy disks, it's easier to start things off inside Hatari itself: press F12 to bring up the configuration screen and click Floppy disks. Click Browse next to Drive A to load disk one into drive A, then repeat with Drive B for disk two. Leave Auto insert B selected, then click Return to main menu followed by OK.

Keep an eye on the status bar at the bottom of the screen, which should hopefully indicate the floppy disk is still loading. It can be a slow process, but check out the Quick Tip for some possible tweaks to try. Once done, the game's main screen should appear and it's then a case of navigating it with your mouse, keyboard or – if you've got one plugged in and configured – game controller.

To set up the latter, press F12 and click Joysticks. From here you can select whether to disable the port, use a "real" joystick or define the keys used to simulate the joystick, namely up, down, left, right and fire. Up to four ports can be configured – two regular ST joysticks plus two additional STE-only joypads.

When it comes to using a real joystick, any game controller that's plugged in and recognised by your Pi should be immediately visible – our wired PC/PS3 Gioteck controller was detected instantly, for example. As you'll have seen when defining keys to simulate a joystick, ST joysticks were basic and only supported four directions of movement and one fire button – use the analogue stick and the top fire button to emulate the joystick, while the secondary fire button emulates the space bar key.





Top left You can connect a real game controller to Hatari, or use keyboard emulation to play games that require a joystick.

04 Create your own blank floppies

If you're trying software that appeared on magazine covermount disks – yes, we're talking ST Format – then chances are they were compressed to enable more content to be placed on the disk. You'll need to extract these to a separate "disk" – we'll cover hard drives in a moment, but if you simply want to run them from their own virtual floppy image, click Create blank image on the Floppy Disks configuration screen. Leave the default settings – 80 tracks, nine sectors and two sides – as they are, give your floppy a label and click Create.

Next, choose where to save the floppy – your default folder is best – and again name it for the game you plan to copy to it. Click OK, then Back followed by Drive B: to insert it into drive

Quick tip

By default, your Atari machine emulates a colour RGB monitor, which enables you to switch between low and medium resolutions. To run high-resolution programs such as DTP software, press F12 and click Atari screen and select Mono

SOLVE COMPATIBILITY ISSUES

It all sounds so simple, and if you're lucky you'll get to play a number of games before coming across one that doesn't appear to work. Start by visiting **http://bit.ly/hatari-compat** for a non-exhaustive list of ST programs that are known to cause problems. Failing that, check the website you downloaded the disk from to see if there's any specific configuration you should use. If you're running an older game, try switching to ST mode with TOS 1.02 for compatibility purposes.

Avoid allocating more than 2MB of RAM to your virtual ST, because timing issues can cause problems, too. 1MB should be ample for most games. Some games may require a NTSC (US English) TOS rather than the default PAL TOS you've downloaded – simply swap them out and see if you're now lucky.

Some games and demos are very particular about floppy emulation – press F12, click Floppy Disks and untick Fast floppy access to see if they're now able to load successfully. Other settings to tweak include unticking Boot Faster and Patch Timer-D under System. If there's screen corruption, also try different Video timing settings.

Switching settings manually can be fiddly, but Hatari makes it possible to save multiple configuration files by changing the name of the .cfg file, enabling you to set up multiple setups (say STFM, STE and Falcon) and switch between them. Hatari will always load the hatari.cfg file by default on startup, but you can quickly switch by pressing F12 followed by Load config to choose another.

Bottom left You can press F12 at any time to pause Hatari and access its comprehensive configuration options

Quick tip

One way to speed up load times is to tick Fast floppy access under Floppy disks in Harari's configuration. Another is to switch to Fast Forward mode – press AltGr + X to speed things up slightly. Press it again to return to normal speed

B. You can now either copy the compressed file across before double-clicking it to unpack the game or use the covermount's own program to extract the files to the new disk. Be warned, it's a slow process, and you may end up having flashbacks, or perhaps that's just this magazine's former disk editor talking. Speaking of which, visit **www.exxoshost.co.uk/atari/STF** for a complete listing and download links for STF's disks.

05 Beyond the basics

As things stand, you now have a fully working Atari machine, running at decent speed. If all you're interested in is reliving some past glories, you should have everything you need to play games on your Atari. But you can push the envelope further, too.

First, look beyond the ST as a mere gaming machine – it was also renowned as a music-making tool and as the games market waned, major applications continued to be developed, from video editing and word processing to desktop publishing. Visit **www.planetemu.net/machine/atari-st** where you'll have access to a wider range of titles, including demos, magazine cover disks, public domain releases and applications.

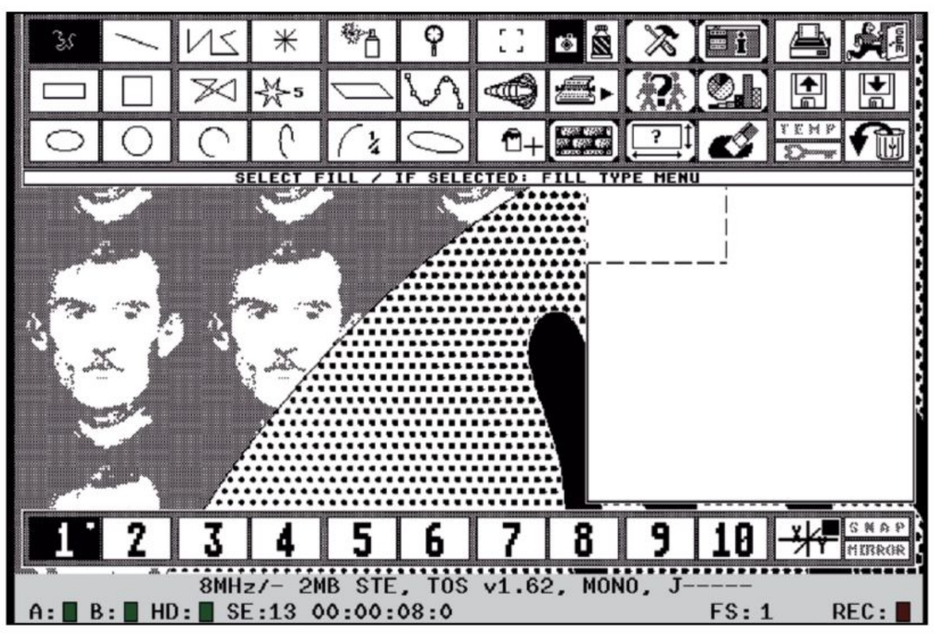
As the number of titles grows, you may get tired of having to press F12 to swap floppy disks around. Use the AltGr+D to swap disks, followed by AltGr+R to perform a warm reset or AltGr+C for a cold reset. Alternatively, why not connect a virtual hard disk to your ST – the box reveals the simplest way to add a single 80 MB ACSI hard disk. You can duplicate this blank image file to create multiple hard disks, but you can only connect one at a time.

Another option is GEMDOS, which enables you to mount a folder on your Pi's storage as a virtual drive, making it easier to copy files directly to the drive from your Pi storage. Note, however, that you'll either need a dedicated hard disk driver or access the drive using EmuTOS, which supports GEMDOS natively.

If you find yourself getting into an old game or two, you may want to know how to save your progress. If the game itself has a save option, you could always create a blank floppy image as outlined earlier and insert it into drive B. Alternatively, you could use Hatari's own memory snapshots feature, which are obviously quicker.

Start by pressing F12 and selecting Memory. Click Save, type a suitably descriptive filename (with the all-important .sav extension) and click OK. Return to your game, pressing AltGr+K at any time to update the saved file with your latest progress, or AltGr+l to reload your last saved state.

You can also snap screenshots at any point – press AltGr+G to take a screen grab in PNG format. Press AltGr+A to record an AVI movie (you'll see the Rec button turn red in the Hatari status bar), and press it again to stop recording. Finally, press AltGr+Y to start and stop recording your ST's sound. All recorded material can be found in your home folder.



Bottom right Your virtual Atari ST can emulate the old high-resolution mono display, too – two monitors for the price of one!

CREATE A VIRTUAL HARD DRIVE

Floppy disks are slow. Attach a virtual hard drive and loading times are quicker and you can switch between programs without having to swap out disks. Hatari supports three type of emulated hard drives: ACSI, IDE and GEMDOS. Of these, ACSI is the most compatible option and works with all versions of TOS from 1.02 up.

ACSI emulated drives exist physically as an IMG file and can be fiddly to set up, plus they require drivers placed on the virtual hard disk to work correctly. Thankfully, Hatari provides you with an empty 80MB hard drive image, which you can download from https://hatari.tuxfamily.org/hd80mb.image.gz. Extract the hd80mb.image file to a suitable folder, then open Hatari's configuration, select Hard disks and click Browse next to ACSI HD image to select the file.

Reboot your ST and you'll see a new drive C appear, which you can copy files to. Note that many commercial games will refuse to run from hard disk. For those that do (such as Civilization), just keep your games in separate folders and make sure you save your configuration, so the hard disk is always accessible whenever you start your virtual ST.

06 MIDI for musicians

The ST was a big draw from musicians with its built-in MIDI ports and was used by major artists including the likes of Queen for The Miracle. If you have a USB-to-MIDI cable then you can hook up a MIDI keyboard to use with the likes of Cubase.

First, you need to verify your cable has been detected by Raspbian. Open a Terminal window and type the following:

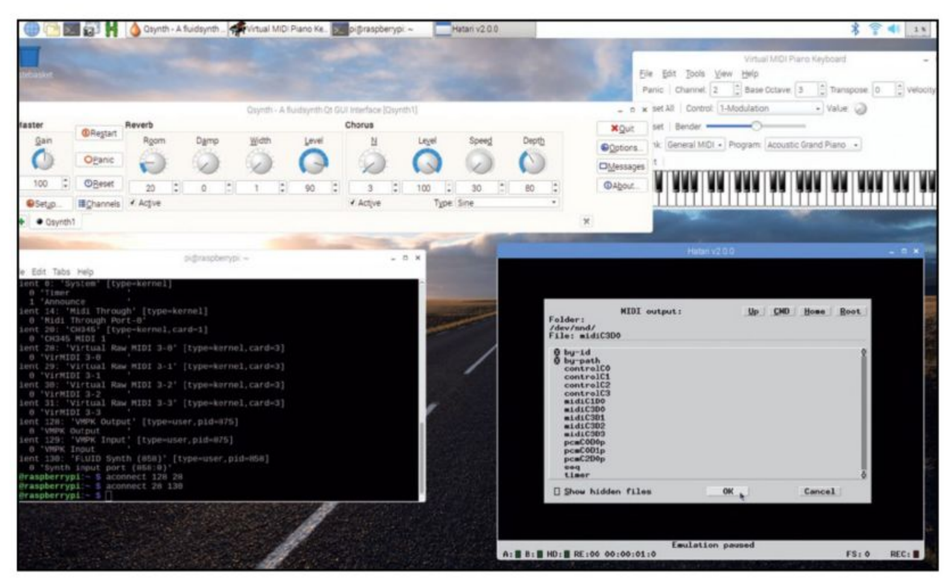
```
$ lsusb
```

You should see your MIDI adaptor listed. Now type the following, which should verify at least one MIDI port is available.

```
$ aplaymidi -l
```

If they're available, open Hatari and press F12, then click Devices. Click the two Browse buttons under Enable MIDI emulation and point them to whichever midi entry appears under /dev/snd. Tick Enable MIDI emulation and restart your virtual ST.

If you don't have a MIDI device, try a software MIDI synthesiser instead. First, install Virtual MIDI Piano Keyboard (search Pi>Preferences>Add/Remove Programs for 'vmpk'). Next, launch QSynth from Pi>Sound & Video followed by VMPK. In VMPK select Edit>Connections, click the Output MIDI



Bottom right Armed with the right knowledge you can set up MIDI emulation on your Atari ST – either with proper hardware or virtual software

connection and choose FLUID Synth to connect the two. Hit the virtual keys and you should hear sound to verify that it's working correctly.

Now open a Terminal window and type the following:

```
$ sudo modprobe snd-virmidi
$ aconnect -i -o
```

Make a note of the client numbers and then type the following:

```
$ aconnect <sender> <receiver>
```

Substitute <sender> with the client number of VMPK Output and <receiver> with the client number of the first Virtual Raw MIDI port. Now repeat the command, this time substituting <sender> with the Virtual Raw MIDI port, and <receiver> with Synth input port – for example, like this:

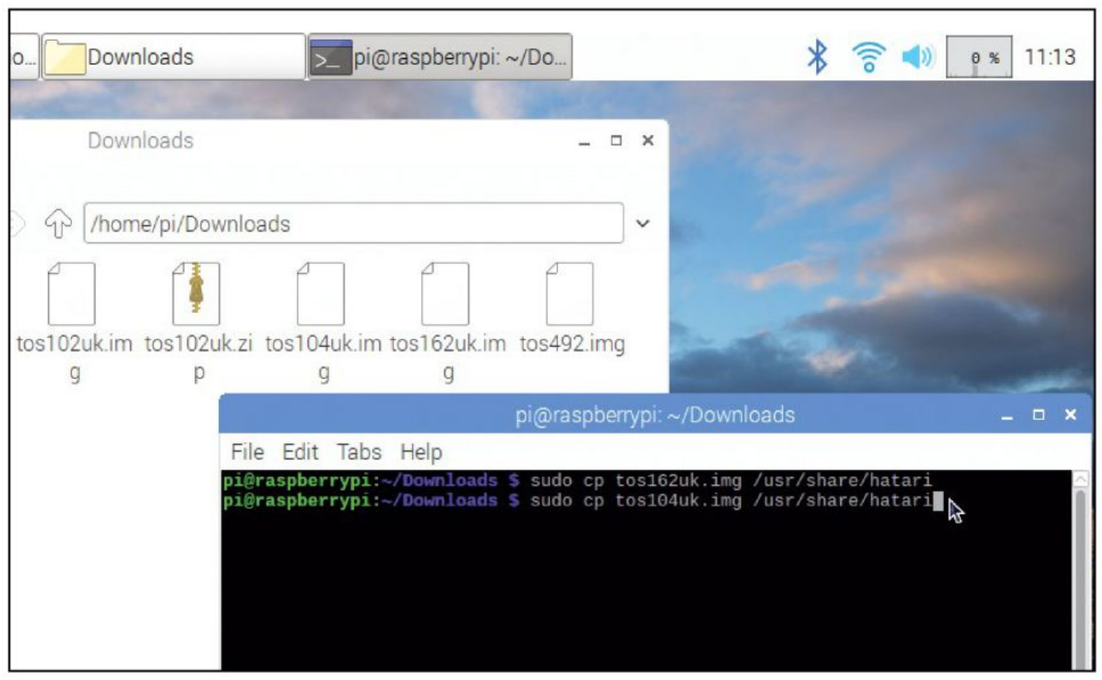
```
$ aconnect 128 28
$ aconnect 28 130
```

Now open Hatari, press F12 and click Devices, then point it towards the correct connection. Tick Enable MIDI emulation, and you should be able to use your virtual synthesiser to input music into sequencing software like Cubase while listening back through Qsynth. Job done!

Quick tip

Click Hatari screen and tick Fullscreen to run it full-screen by default or press AltGr+F to switch between full screen and windowed mode. Press AltGr+M to capture the mouse in windowed mode

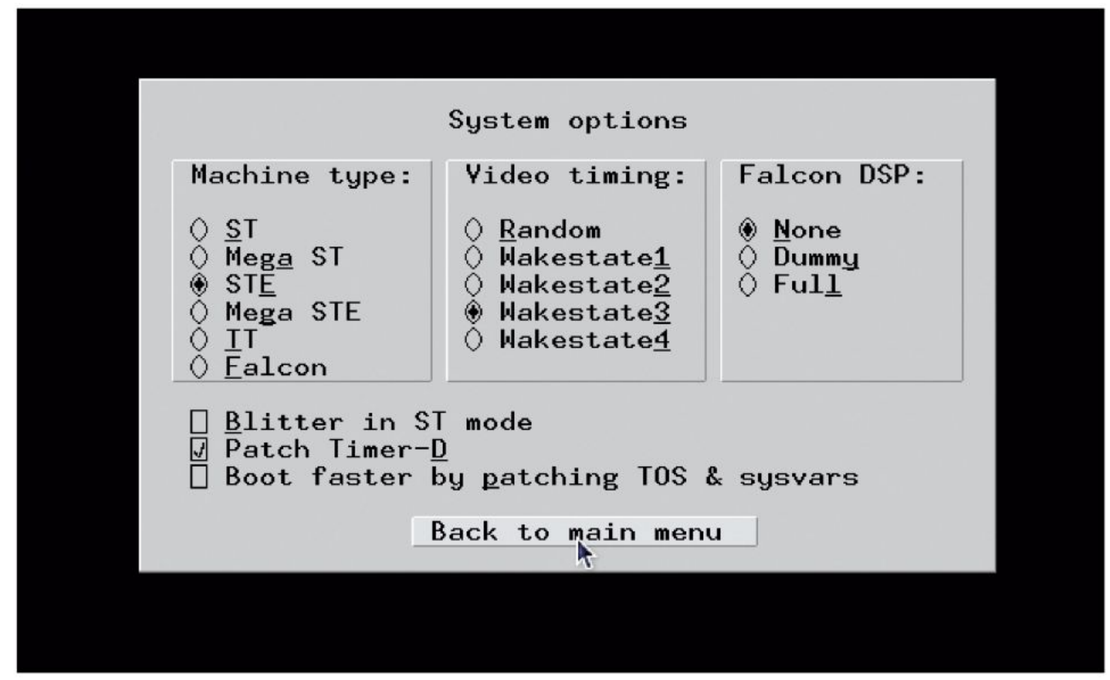
CONFIGURE YOUR VIRTUAL ATARI ST



1 Copy TOS files

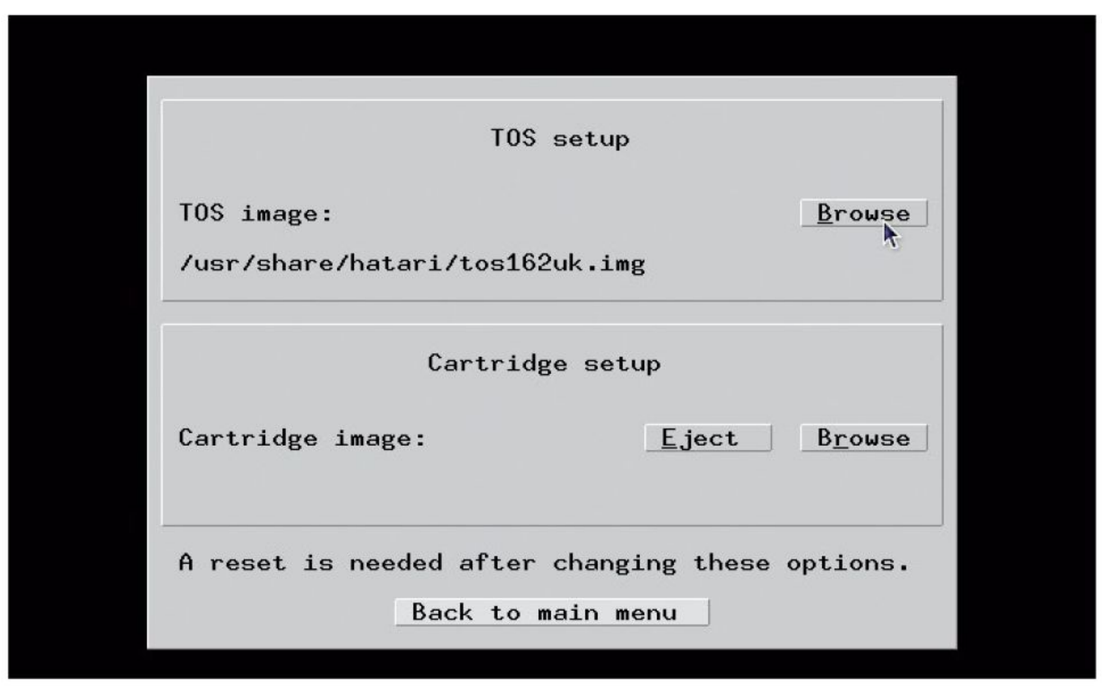
Open File Manager. Double-click the downloaded TOS zip file to open it in Archive Manager, then drag the .img file into your Downloads folder. Once done, press F4 to open a Terminal window pointing to the Downloads folder and type the following (substitute tos102uk.img with the image file):

```
$ sudo cp tos102uk.img /usr/share/hatari
```



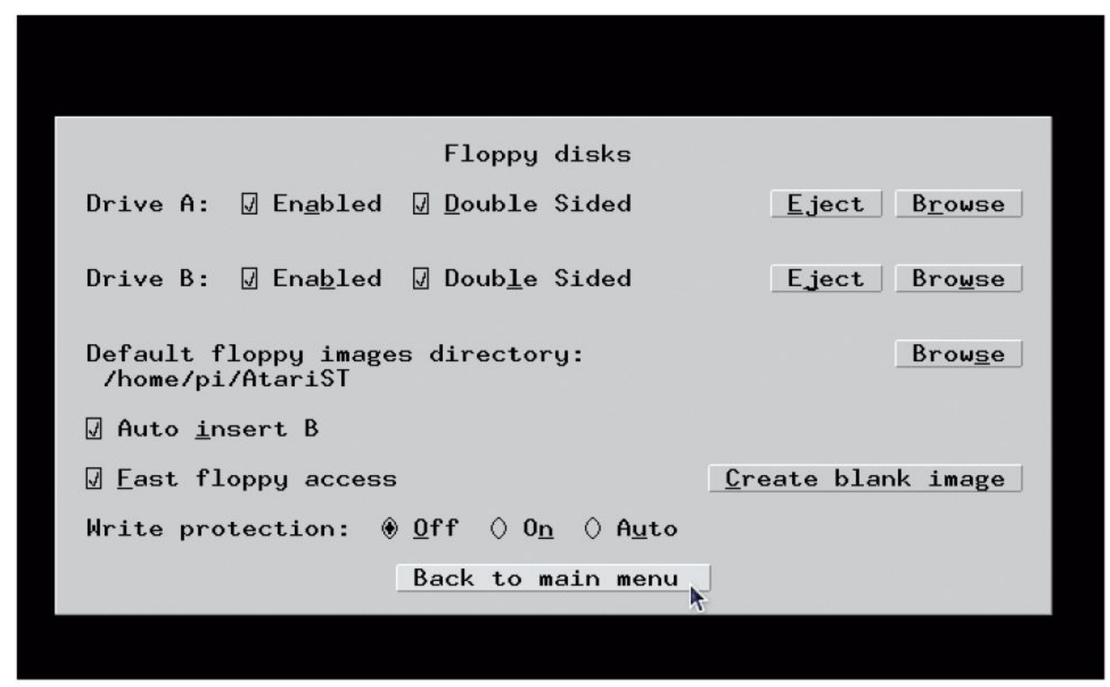
2 Set system type

Now open the Pi menu and select Games>Hatari. Click OK at the Could not load TOS file screen. When the main menu appears, click System. By default, Hatari is configured to emulate the STFM, but you can select a different Atari machine from the list. Tick Boot faster by patching TOS & sysvars regardless of which machine you choose to speed loading times slightly.



3 Select ROM and RAM

Return to the main menu and click ROM. Click Browse to select the TOS image you copied across in step one – make sure it matches the Atari machine you're emulating. Click Back to main menu again, then click RAM. By default, 1MB is allocated, which is sufficient for gaming – drop this to 512K for some older games if they refuse to load.



4 Select floppy disk and save configuration

Now click Floppy disks followed by Browse next to Default floppy images directory to select the folder you've stored your disk images in. Tick Fast floppy access to help speed up loading times, but note that some disks may not work (in which case unticking this may help). Click Back to main menu followed by Save config to save your setup.

Build a Raspberry Pi Minecraft console

Create a full-functional, Pi-powered games console that you can play Minecraft on and learn how to program too.

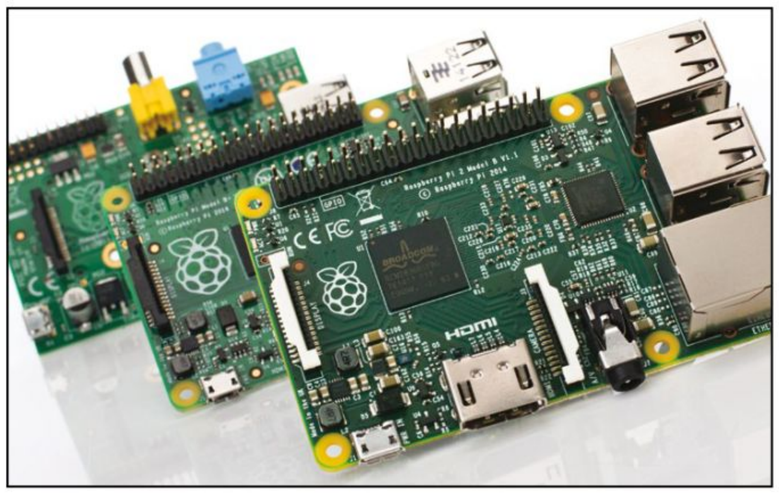
Minecraft means many things to many people, and to Raspberry Pi users it's supposed to mean education. Not everyone knows, though, that you can still have fun and play *Minecraft* as you normally would.

Using Raspberry Pi, it is also the cheapest way to get a fully-functional version of *Minecraft* up onto your TV. However, in its normal state, just being on a TV isn't the end of it. Using all the features and functions of the Pi, we can take it to a state more fitting of a TV by making it into a hackable, moddable *Minecraft* console.

In this tutorial, we will show you how to set it up in terms of both software and hardware, how to add a game controller to make it a bit better for TV use, and we'll even give you some example code on how to mod it. Now, it's time to get building, so head to Step 1.

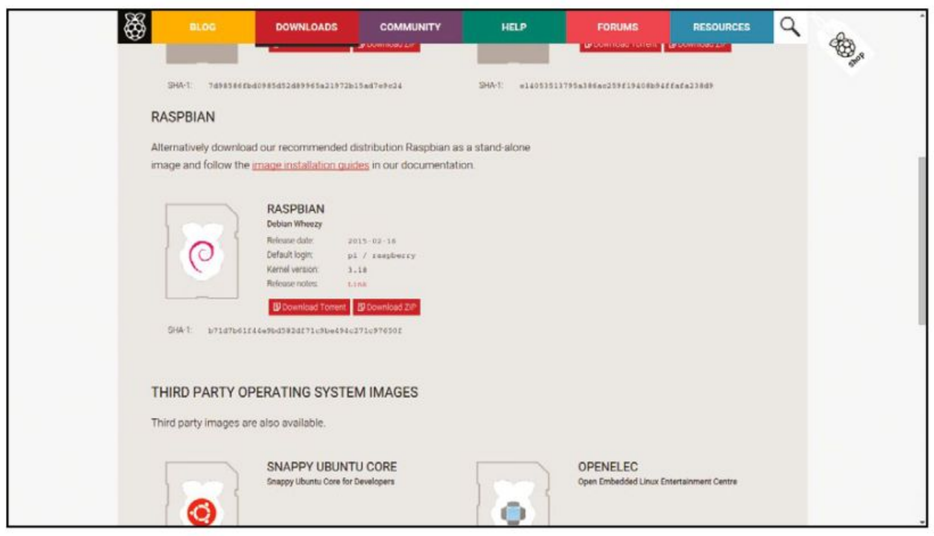
What you'll need

- Raspberry Pi 2
- Latest Raspbian image
  raspberrypi.org/downloads
- Minecraft Pi Edition
  pi.minecraft.net
- Raspberry Pi case
- USB game controller (PS3 preferable)



01 Choose your Raspberry Pi

Before we start anything, everything we plan to do in this tutorial will work on all Raspberry Pi Model Bs with at least 512 MB of RAM. However, *Minecraft: Pi Edition* can chug a little on the original Model Bs, so we suggest getting a Raspberry Pi 2 to get the most out of this tutorial.



02 Prepare your Raspberry Pi

Minecraft: Pi Edition currently works on Raspbian. We recommend you install a fresh version of Raspbian, but if you already have an SD card with it on, the very least you should do is:

```
sudo apt-get update && sudo apt-get upgrade
```

03 Prepare Minecraft

If you've installed Raspbian from scratch, *Minecraft* is actually already installed – go to the Menu and look under Games to find it there ready. If you've just updated your version of Raspbian, you can install it from the repos with:

```
$ sudo apt-get install minecraft-pi
```

04 Test it out

If you've had to install *Minecraft*, it's best just to check that it works first. Launch the desktop, if you're not already in it, with **startx** and start *Minecraft* from the Menu. *Minecraft: Pi Edition* is quite limited in what it lets you do, but it does make room for modding.

05 X setup

If you have a fresh Raspbian install and/or you have your install launch into the command line, you need to set it to load into the desktop. If you're still in the desktop, open up the terminal and type in **raspi-config**. Go to Enable Boot to Desktop and choose Desktop.

Above Give *Minecraft: Pi Edition* a quick test before you start building the console

> If you've installed Raspbian from scratch, Minecraft is actually already installed – go to the Menu and look under Games to find it

06 Set up Python

While we're doing set up bits, we might as well modify *Mincecraft* using Python for a later part of the tutorial. Open up the terminal and use:

```
$ cp /opt/minecraft-pi/api/python/mcpi ~/minecraft/
```

07 Minecraft at startup

For this to work as a console, we'll need it to launch into *Minecraft* when it turns on. We can make it autostart by going into the terminal and opening the autostart options by typing:

```
$ sudo nano /etc/xdg/lxsession/LXDE-pi/autostart
```

08 Autostart language

In here, you just need to add **@minecraft-pi** on the bottom line, save it and reboot to make sure it works. This is a good thing to know if you also want other programs to launch as part of the boot-up process.

09 Turn off

For now, we can use the mouse and keyboard to shut down the Pi in the normal way, but in the future you'll have to start turning it off by physically removing power. As long as you've exited the *Minecraft* world and saved, that should be fine.

Updates to Pi Edition?

Minecraft: Pi Edition hasn't received an update for a little while, but it was previously limited by the original Model B. Now with more power, there may be an update that adds more to it, but right now there's no indication of that. If it does come though, all you need to do is update your Pi with: **sudo apt-get update && sudo apt-get upgrade**.

Getting power to the Raspberry Pi 2 so that it runs properly can be tricky if you're using a USB port

3D-print a case

Aaron Hicks at Solid Technologies designed this *Minecraft* case for the Raspberry Pi and shared it on GrabCAD. We've uploaded our slightly modified version to **FileSilo.co.uk** along with your tutorial files for this issue. All you need to do is send the STL file to a 3D printing service – many high street printing shops have at least a MakerBot these days – and they will 3D-print the case for you. It should only cost around £15.

10 The correct case

In this scenario, we're hooking this Raspberry Pi up to a TV, which means it needs a case so that there's less chance of damage to the components from dust or static. There are many good cases you can get – we are using the Pimoroni Pibow here as you can mount it to the back of the TV. Alternatively, you could get really creative and 3D-print your own case, as you can see on page 58. Check out the boxout just to the left.

11 Find the right power supply

Getting power to the Raspberry Pi 2 so that it runs properly can be tricky if you're using a USB port or a mobile phone charger – the former will be underpowered and the latter is not always powerful enough. Make sure you get a 2A supply, like the official Raspberry Pi one.

12 Go wireless

We understand that not everyone has an ethernet cable near their TV, so it may be a good idea to invest in a Wi-Fi adapter instead. There is a great list of compatible Wi-Fi adapters on the eLinux wiki: **elinux.org/RPi_VerifiedPeripherals**.

13 Mouse and keyboard

Now that we have the Raspberry Pi ready to be hooked up, you should look at your controller situation – do you want to be limited by the wires or should you get a wireless solution instead? We will cover controller solutions over the page, but it's worth considering now.

14 Get ready for SSH

It will be easier to create and apply scripts to *Minecraft* by uploading them via the network rather than doing it straight on the Pi. In the terminal, find out what the IP address is by using **ifconfig**, and then you can access the Pi in the terminal of another networked computer using the following:

```
ssh pi@[IP address]
```

15 Have a play

At this stage, what we have built is a fully-functional *Minecraft* console. Now, at this point you could start playing if you so wish and you don't need to add a controller. You can flip over to page 62 now if you want to begin learning how to mod your *Minecraft* and do a bit more with it to suit your needs. However, if you do want to add controller support then carry on and take a look at Step 16.

Controls

Here's the full layout of the buttons used by the PS3 controller by default – you can change them in the script that you download in Step 18

16 Add controller support

Make sure the controller input functions are installed on the Raspberry Pi. To do this, **ssh** into the Raspberry Pi like we did in Step 14 (where 'raspberry' is the password) and install the following package:

```
$ sudo apt-get install xserver-xorg-input-joystick
```

17 Controller mapping

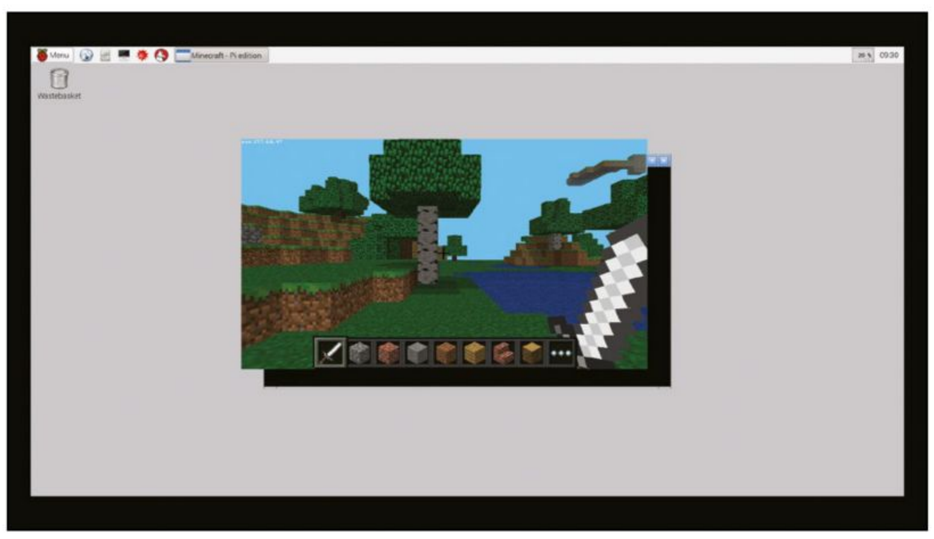
We have a controller map for the PS3 controller that you can download straight to your Pi, and with a bit of tweaking can fit most USB controllers as well. Go to the controller configuration folder with:

```
$ cd /usr/share/X11/xorg.conf.d/
```

18 Replace the controller mapping

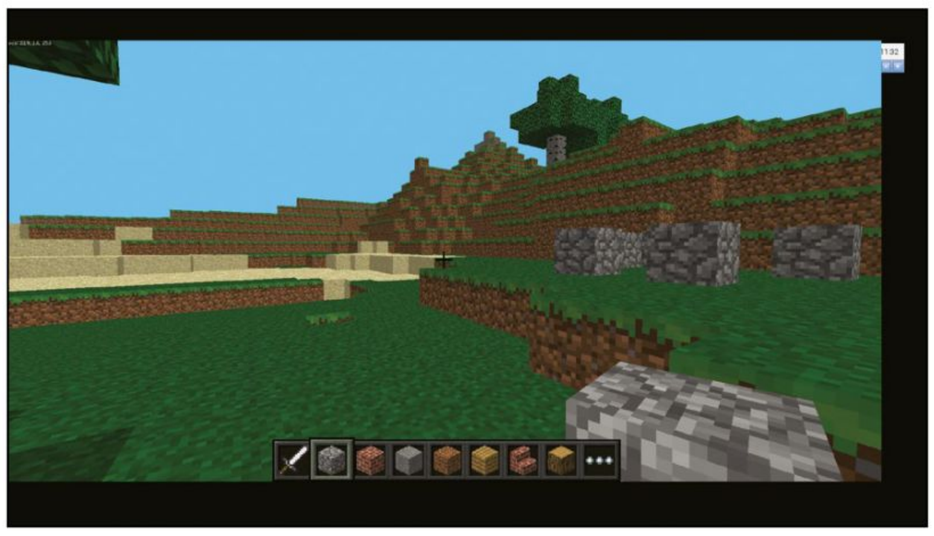
We'll remove the current joystick controls by using **sudo rm 50-joystick.conf** and then replace by downloading a custom configuration using:

```
$ sudo wget http://www.linuxuser.co.uk/wp-content/
uploads/2015/04/50-joystick.conf
```



19 Reboot to use

After a reboot to make sure everything's working, you should be able to control the mouse input on the console. R2 and L2 are the normal mouse clicks and can be used to navigate the *Minecraft* menu to access the game.



20 Go full-screen

So far you may have noticed that *Minecraft* is running in a window – you can click the full-screen button to make it fill the screen, however you then heavily limit your mouse control. Thanks to the controller, you can get around that. As soon as you load the game, make sure you use the sticks for movement and the d-pad for selecting items in the inventory.

Xbox controllers

Unfortunately, Xbox 360 controllers work slightly differently with Linux. As they use their own drivers that are separate to the normal joystick drivers we used for the PS3 pad and other USB controllers, a 360 controller doesn't work as a mouse and is harder to assign specific functions to. This makes it tricky to use in a situation such as this.

Mod your Minecraft

Here is some example code, and explanations for it, so that you can learn how to program in Python and mod Minecraft Pi.

We program *Minecraft* to react in python using the API that comes with *Minecraft: Pi Edition* – it's what we moved to the home folder earlier on. Now's a good time to test it – we can do this remotely via SSH. Just **cd** into the Minecraft folder in the home directory we made, and use **nano test.py** to create our test file. Add the following:

```
from mcpi.minecraft import Minecraft
from mcpi import block
from mcpi.vec3 import Vec3
mc = Minecraft.create()
mc.postToChat("Hello, Minecraft!")
```

Save it, and then run it with:

```
$ python test.py
```

"Hello, Minecraft!" should pop up on-screen. The code imports the Minecraft function from the files we moved earlier, which allows us to actually use Python to interact with *Minecraft*, along with the various other functions and modules imported. We then create the mc instance that will allow us to actually post to *Minecraft* using the postToChat function. There are many ways you can interact with *Minecraft* in this way – placing blocks that follow the player, creating entire structures and giving them random properties as they're spawned as well. There are very few limits to what you can do with the Python code, and you can check out more projects here: **https://mcpipy.wordpress.com**.

Over the page, we have a full listing for a hide and seek game that expands on the kind of code we're using here, where the player must find a diamond hidden in the level, with the game telling you whether you're hotter or colder. You can write it out from scratch or download it to your Pi using the following commands:

```
$ wget  http://www.linuxuser.co.uk/tutorialfiles/
Issue134/ProgramMinecraftPi.zip
$ unzip ProgramMinecraftPi.zip
$ cp Program\ MinecraftPi/hide_and_Seek.py ~/minecraft
```

Check out the annotations to the right to see how it works.

> We program Minecraft to react in Python using the API that comes with Minecraft Pi – it's what we moved to the home folder earlier

Below You can see the hidden diamond just to the left of the crosshair at the centre of this screenshot

Full code listing

Import

Here we're importing the necessary modules and APIs to program *Minecraft*. Most importantly are the files in the mcpi folder that we copied earlier

```
from mcpi.minecraft import Minecraft
from mcpi import block
from mcpi.vec3 import Vec3
from time import sleep, time
import random, math
```

Locate

We connect to *Minecraft* with the first line, and then we find the player's position and round it up to an integer

```
mc = Minecraft.create()
playerPos = mc.player.getPos()

def roundVec3(vec3):
  return Vec3(int(vec3.x), int(vec3.y), int(vec3.z))
```

Range finding

Calculate the distance between the player and diamond. This is done in intervals later on in the code, and just compares the co-ordinates of the positions together

```
def distanceBetweenPoints(point1, point2):
  xd = point2.x - point1.x
  yd = point2.y - point1.y
  zd = point2.z - point1.z
  return math.sqrt((xd*xd) + (yd*yd) + (zd*zd))
```

Creation

Create a random position for the diamond within 50 blocks of the player position that was found earlier

```
def random_block():
  randomBlockPos = roundVec3(playerPos)
  randomBlockPos.x = random.randrange(randomBlockPos.x - 50, randomBlockPos.x + 50)
  randomBlockPos.y = random.randrange(randomBlockPos.y - 5, randomBlockPos.y + 5)
  randomBlockPos.z = random.randrange(randomBlockPos.z - 50, randomBlockPos.z + 50)
  return randomBlockPos
```

Start

This is the main loop that actually starts the game. It asks to get the position of the player to start each loop

```
def main():
  global lastPlayerPos, playerPos
  seeking = True
  lastPlayerPos = playerPos
```

Notification

This part sets the block in the environment and pushes a message using postToChat to the *Minecraft* instance to let the player know that the mini-game has started

```
  randomBlockPos = random_block()
  mc.setBlock(randomBlockPos, block.DIAMOND_BLOCK)
  mc.postToChat("A diamond has been hidden - go find!")
```

Checking

We start timing the player with timeStarted, and set the last distance between the player and the block. Now we begin the massive while loop that checks the distance between the changing player position and the fixed diamond. If the player is within two blocks of the diamond, it means they have found the block and it ends the loop

```
  lastDistanceFromBlock = distanceBetweenPoints(randomBlockPos, lastPlayerPos)
  timeStarted = time()
  while seeking:

    playerPos = mc.player.getPos()

    if lastPlayerPos != playerPos:
      distanceFromBlock = distanceBetweenPoints(randomBlockPos, playerPos)
      if distanceFromBlock < 2:
        seeking = False
```

Message writing

If you're two or more blocks away from the diamond, it will tell you whether you're nearer or farther away than your last position check. It does this by comparing the last and new position distance – if it's the same, a quirk in Python means it says you're colder. Once it's done this, it saves your current position as the last position

```
      else:
        if distanceFromBlock < lastDistanceFromBlock:
          mc.postToChat("Warmer " + str(int(distanceFromBlock)) + " blocks away")
        if distanceFromBlock > lastDistanceFromBlock:
          mc.postToChat("Colder " + str(int(distanceFromBlock)) + " blocks away")

      lastDistanceFromBlock = distanceFromBlock

    sleep(2)
```

Success

It takes a two-second break before updating the next position using the sleep function. If the loop has been broken, it tallies up your time and lets you know how long it was before you found the diamond. Finally, the last bit then tells Python to start the script at the main function

```
  timeTaken = time() - timeStarted
  mc.postToChat("Well done - " + str(int(timeTaken)) + " seconds to find the diamond")

if __name__ == "__main__":
  main()
```

Playing with blocks Minecraft style

Les Pounder spent the weekend hacking Minecraft in a classroom full of kids, and here are the hacks that the kids wanted!

OUR EXPERT

Les Pounder is a freelance maker who works with organisations such as the Raspberry Pi Foundation to promote maker skills. He blogs at bigl.es

YOU NEED

- **Any model Pi**
- **The latest Raspbian**
- **https://github.com/lesp/LXF243-mczero/archive/master.zip**

In the Python shell type commands() and press Enter to see all of the commands. This works for blocks() showing all the blocks that we can build with, and their respective blocktype (number.)

As a learning tool, *Minecraft* is awesome. Kids play the game and learn how to create blocks of lava, diamond and water. But for younger players, the syntax of the commands is difficult. So for our project this issue we shall create an easy-to-use series of commands, cheekily called *Minecraft Zero* after GPIO and GUI Zero libraries, that enable kids to simply have fun, and learn a little Python along the way.

On your Raspberry Pi, open the Python 3 editor, found in the Programming menu. Once opened, click File>New to create a new blank document. Now click File>Save and call the file **mczero.py**. Remember to save your progress regularly.

Our project is made up of a series of functions, blocks of Python code that we can use by calling their name. The first of these is the most simple, and that's to post a message to the *Minecraft* chat window. As with all functions we need to define a name, and we shall call it `chat`, but you will see `msg` in parenthesis. This `msg` is the text we want to display in the chat window.

```
def chat(msg):
```

Inside the function we then use *Minecraft's* library, which we shall later import and rename to `mc`, `postToChat` function. Note that we also use a `str()` helper to ensure that any data used for messages is converted to a string because the `postToChat` function only works with strings:

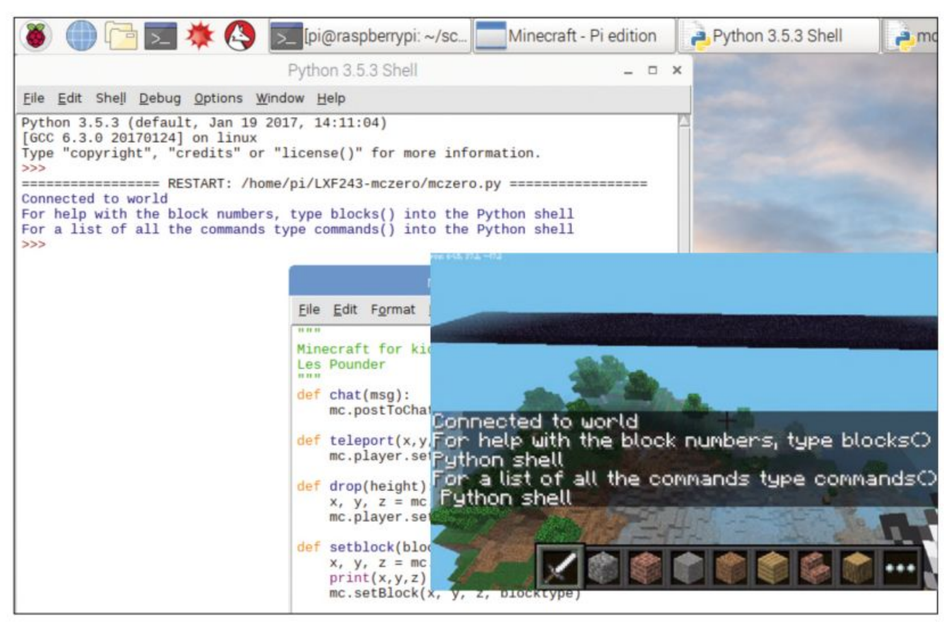
```
    mc.postToChat(str(msg))
```

Take a trip

The next function is called `teleport` and it's used to send the player to any point in the world. All it needs are x, y, z coordinates, found in the top left of the screen. This function works in a similar manner as `chat` in that we invoke the function with a parameter which is the x, y, z position. Then in the function we set the position of the player `(setPos)` to those coordinates:

```
def teleport(x,y,z):
    mc.player.setPos(x, y, z)
```

Kids love to soar high into the sky, and then fall back to the world. Well, with the `drop` function we can do that. This function takes a parameter, the height (in blocks) to drop Steve from. This is then used in the function after we first learn of the players current position using `getPos` to update the x, y, z coordinates.



When the mczero library is first used, it'll print helpful messages to the Python shell and the Minecraft chat window.

Then using `setPos` we change the value of y so that it's increased by the value of `height`.

```
def drop(height):
    x, y, z = mc.player.getPos()
    mc.player.setPos(x, y+height, z)
```

The next important part of *Minecraft* is changing blocks. Using Python we can edit any part of the world, changing grass to flowing lava, diamond or gold. To do that we need to alter the `blocktype` and our `setblock` function handles this by changing the block on which we're stood to anything that we pass as a parameter. If you don't know the blocktypes (numbers) don't worry, we cover that in our `blocks()` function later.

```
def setblock(blocktype):
    x, y, z = mc.player.getPos()
    mc.setBlock(x, y, z, blocktype)
```

Cubes on demand!

Our cube function takes two parameters: the size of the cube (in blocks) and the type of blocks it's made from. So we can create cubes of diamond and use them to obstruct our enemies! In this function we use `setBlocks` and give two x,y,z positions. The first being the player's position plus one block on each axis, and the second position being the player's position plus the size of the cube. *Minecraft* then fills in the values between them to create a solid cube, like so:

```
def cube(size,blocktype):
    x, y, z = mc.player.getPos()
```

```
mc.setBlocks(x+1, y+1, z+1, x+size, y+size, z+size, blocktype)
```

But what if you want a big, hollow cube? Well, our `hollow_cube` function will do that for you. It creates a solid cube of diamond blocks (blocktype 57), and then we create a smaller hollow cube inside it using a cube of air blocks (blocktype 0):

```
def hollow_cube(size):
    x, y, z = mc.player.getPos()
    mc.setBlocks(x+1, y+1, z+1, x+size, y+size, z+size, 57)
    size = size - 2
    mc.setBlocks(x+2, y+2, z+2, x+size, y+size, z+size, 0)
```

Tick, tick, BOOM!

This function creates a trail of active TNT that follows your every step. When you hit it with a sword, it explodes! For this we need to get the players position, then set the block at their feet to TNT (blocktype 46), but we need to also pass the value `1` so that the TNT can be explosive. Then we sleep for `0.1` to slow down the number of TNT blocks on screen. This function is best used inside a while `True: loop` so that it continuously runs.

```
def tnt_trail():
    x, y, z = mc.player.getPos()
    mc.setBlock(x, y, z, 46,1)
    sleep(0.1)
```

Our `tnt_cube` function creates a cube that's full of TNT! Just pass the size parameter, hit the cube and run! Note that on older Pi, the maximum size should be 10x10x10. For Pi 3 you can risk 20x20x20, but bear in mind that it may lock up the Pi:

```
def tnt_cube(size):
    x, y, z = mc.player.getPos()
    tnt = 46
    mc.setBlocks(x+1, y+1, z+1, x+size, y+size, z+size, tnt ,1)
```

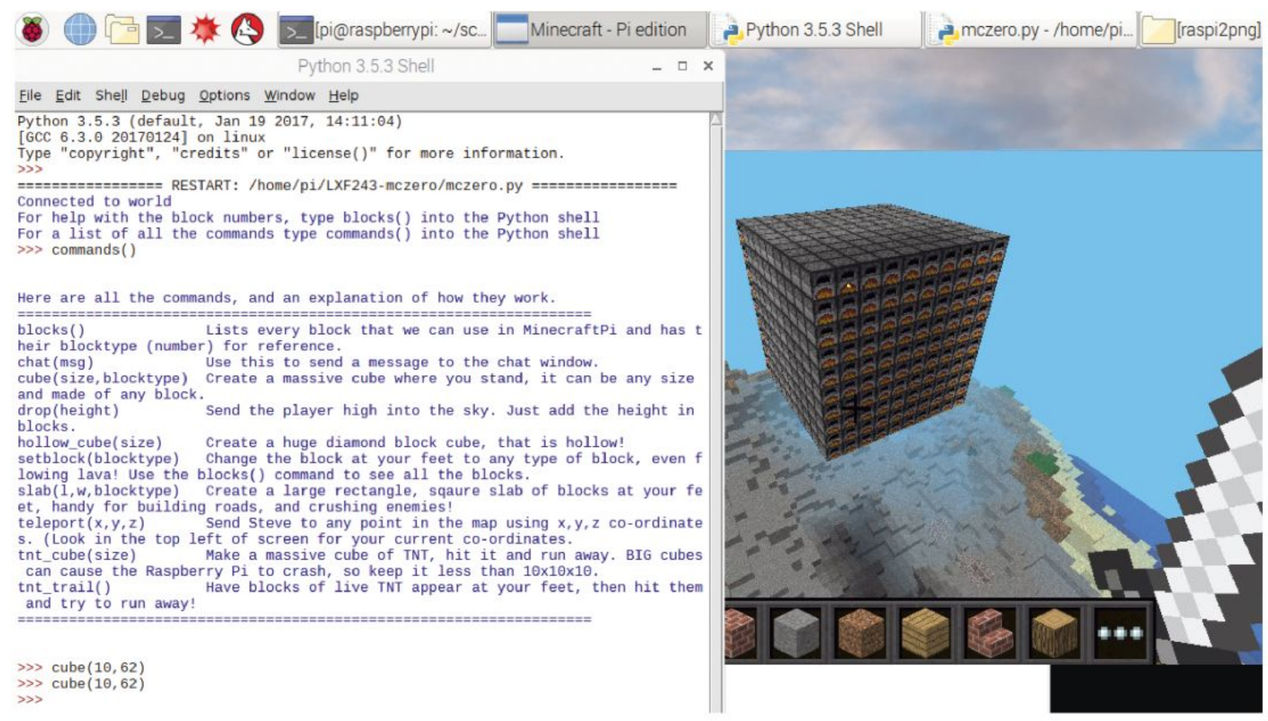
Building long lines of blocks is boring! But with our new `slab` function we can create large slabs of blocks by passing the length (l), width (w) and the blocktype. Essentially, this is the same as our cube function, but the value of y never changes, which forces the slab to be built in only two axes:

```
def slab(l,w,blocktype):
    x, y, z = mc.player.getPos()
    mc.setBlocks(x+1, y+1, z+1, x+w, y+1, z+l, blocktype)
```

The last in-game function is called `fire_feet` and it places flaming torches (blocktype 50) at your feet wherever you go. This is best used with a `while True` loop, just like the `tnt_trail` function.

```
def fire_feet():
    x, y, z = mc.player.getPos()
    mc.setBlock(x+1, y, z+1, 50,1)
    sleep(0.1)
```

The next two functions are built-in help tools. The first is called `blocks()` and it stores a dictionary (called `block_dictionary`), a data type to store large amounts of data that are referenced via a key:value relationship. In this case the keys are the names of the blocks, and the values are the blocktype numbers. So, for example the key AIR return the blocktype 0. There are loads of different blocks in the dictionary, and to show them all we use a for loop that will iterate over each of them and print the key and value, with a 0.1 second pause



Using the commands() function we found the function to generate a cube, so we made a 10x10x10 of furnaces, because you can! Try it with flowing water for utter mayhem!

between each line. You can find the code listing for this function in the download for this project.

The final function is called `commands()` and it simply prints out all of the commands and what they do. Handy to remind ourselves when in the middle of battle! You can find the code listing for this function in the download for this project.

To run all of the code we use an exception handler, that will try to run the code. In this case it will import the *Minecraft* library, the block library and import sleep from the time library. Then it creates a connection to the running *Minecraft* game. If this works then text is displayed in the Python shell and *Minecraft* chat window advising the player of the help functions (`blocks()` and `commands()`), but if the connection fails then an exception is raised and an error is printed to the shell.

Save the code, launch a *Minecraft* game and when you're in the world, click Run>Run Module in the Python editor. Now try out the commands and prepare to cause havoc and mayhem!

» ADDING THE CODE TO A PROJECT

Our **mczero.py** code can be used via the shell, for an interactive battle of skill against other players. But it can also be used as part of a project in the same manner as we import other libraries of code, for example when we import the Minecraft (mcpi) library into our **mczero.py** code. To do this we need to ensure that any code that we create is saved in the same directory as the **mczero.py** file. To start we import the mczero library:

```
import mczero
```

Then we can use any of the functions in our code by calling their names, preceded by the name of the library, in this case mczero. For example let's pop a message in the chat window:

```
mczero.chat("Hello Linux Format")
```

Then let's create a loop for our fire_feet function:

```
while True:
    fire_feet()
```

Any code that we wish to continuously run can be placed inside the loop, so our `tnt_trail()` and `fire_feet()` functions can be run together. However, for the larger functions, that create cubes, slabs and so on, they should ideally be run once, or on demand using a conditional statement (`if..elif..else`) For example, one that checks for the type of block that the player is standing on.

Build your own games console

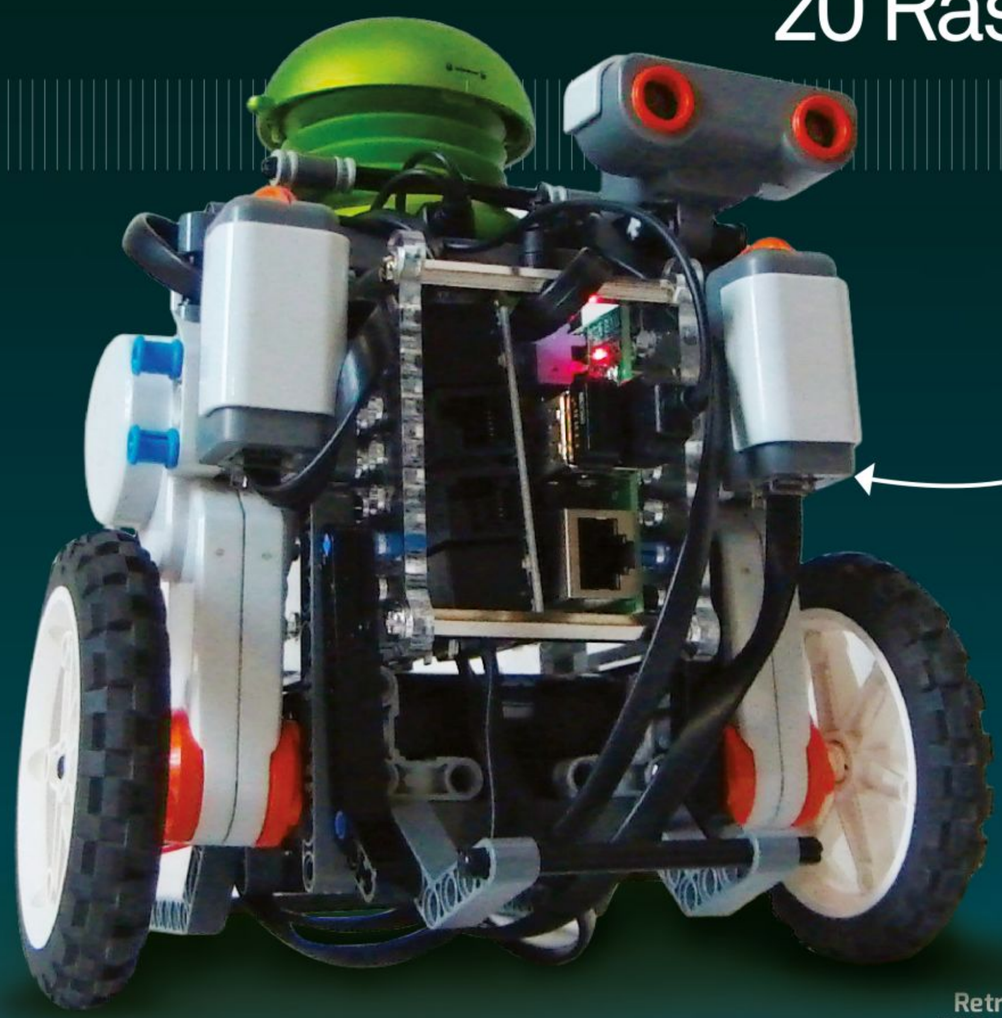
Simulate a spaceship launch

Reinvent a classic camera

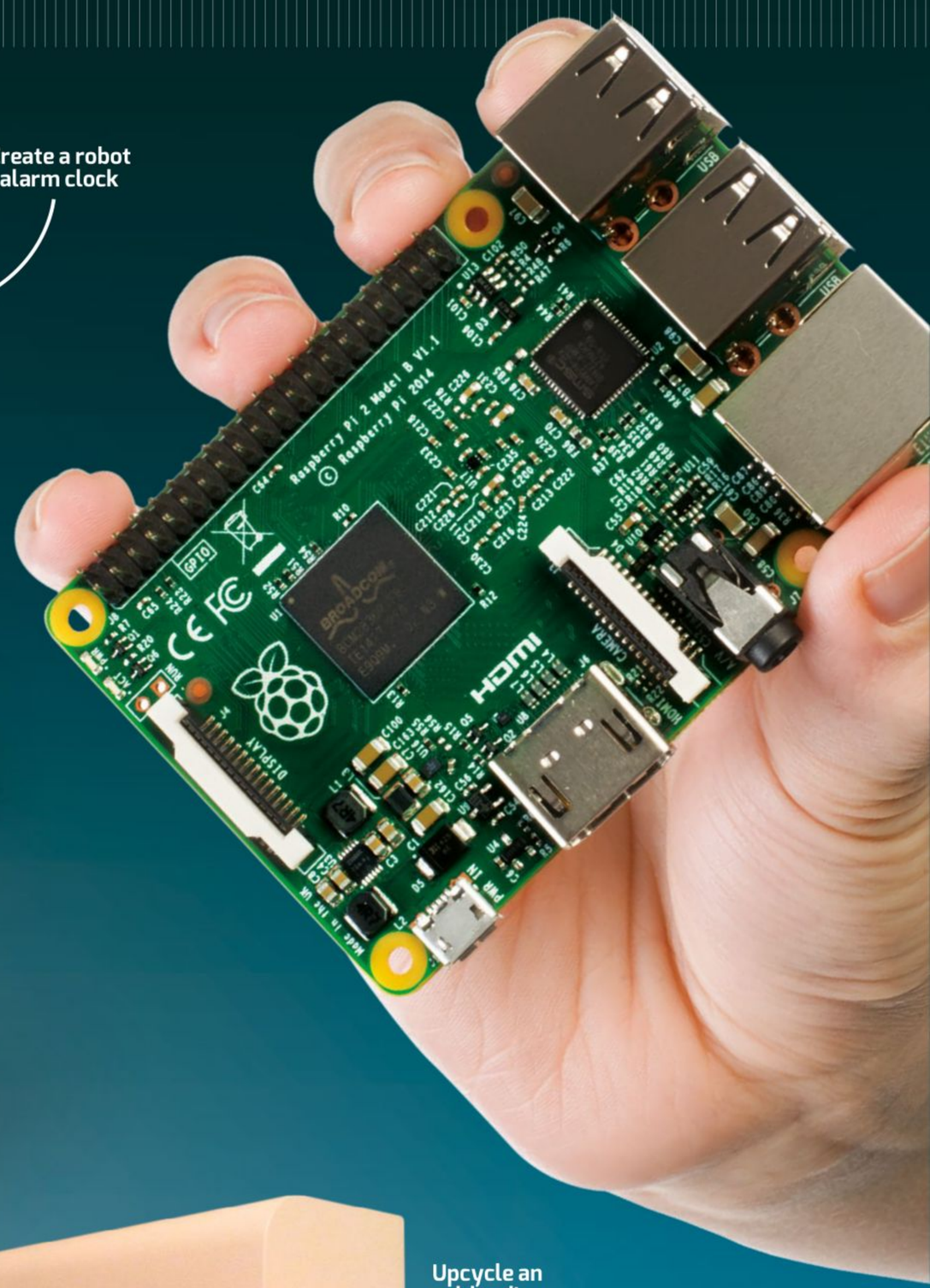
20 Raspberry Pi hacking projects

Get the inside story on how to replicate the greatest Raspberry Pi hardware hacks

20 Raspberry Pi hacking projects



Create a robot alarm clock



Retrofit a NES console

Upcycle an old radio

Build an Android-powered TV

Play an AR light gun game

Turn your Pi into a telephone

There are now over five million Raspberry Pi models out in the wild, and some of the things that you, the Raspberry Pi community, have made with them truly are wild. From elegantly crafted scripts that chain together a series of web services to homebrew Rube Goldberg machines, they are as creative as they are diverse. And through the crowd of new projects bubbling up online every day, if there's one word that's guaranteed to get everyone's attention then it's the word 'hack'.

But what exactly is a hack? Well, for the purposes of this feature, we decided that a hack has to have some sort of hardware base. It's the kind of project where you take one device and, with a little Raspberry Pi magic, transform it into something wholly new and original. These are the projects that get us excited and make us want to learn more about electronics, engineering and programming.

Over the next few pages we're going to introduce you to some of the greatest Raspberry Pi hacks we've discovered. Projects where vintage hardware has been torn apart and the components repurposed into something amazing, or where the hardware has been puzzled over, fiddled with, and brought back to life after years spent in a garage. These hacks inspire us, with each maker striking the right balance between passion, skill and virtuosity, and we hope they inspire you too. Read on as we hear how you can launch a satellite from a bedroom spaceship, transform an analogue camera into a digital one, make a classic Apple Pi and more.

Camera Pi

Power up a regular point-and-shoot DSLR camera

Maker Profile

Dave Hunt

Photographer and maker

David has been making projects for the Raspberry Pi since the early days.

Find out more: **davidhunt.ie**

Dave Hunt is well known to Raspberry Pi fans, and here he's back with his Camera Pi.

"I needed to transmit photos to an iPad as they were taken," explains Dave, "but the commercial solutions were £500. I had a broken battery grip big enough to fit my Raspberry Pi and a battery, so it went from there.

"The battery grip holds two batteries. Once I'd stripped out the battery compartment, I set about filing down all the mounting holes inside the grip so I could get the Raspberry Pi inside.

"The next task was to fit a camera battery and DC-DC converter inside. I was able to use part of the removed internals of the grip, and before long I had a slot to insert a camera battery into. It's capable of powering the Pi for about four hours.

"Making it wireless was a case of plugging in a USB Wi-Fi adapter. A few lines of Perl later and I was able to poll the camera with gphoto2, pull the new files off and send them via FTP to ShutterSnitch on my iPad."

Above Read up on the full build process and check out Dave's video at **bit.ly/1BxEMbC**

Pi Telephone

Revive a ringing phone with C# circuit wizardry and voltage manipulation

Maker Profile

Stuart Johnson

Managing director

Stuart runs Logic Ethos, an IT company in Southampton providing network services and cloud computing help to developers.

Find out more: **logicethos.com**

Stuart Johnson is bringing a classic GPO 746 handset back to life, and while the project isn't yet complete, he has finished the lion's share of it.

"I took out the main circuit board inside the phone and squeezed the Raspberry Pi in there," says Stuart. "I was then faced with two challenges – the biggest one was getting the bell to ring. I found a solution by raising the voltage to 19 volts and dropping it down to 5 for the Ras Pi using a very small DC-DC converter (the OKI-78SR), with the rest then being used for the bell. I was surprised by how well it worked.

"The bell is using one of the I/O ports, and there's an available C# library (raspberry-sharp-io) which lets you monitor and control those ports. So I linked one of the I/Os to the pulse dial and connected another to a relay using transistors. Then with the software I put in a timer to measure the pulse clicks. I managed to write some code to time those pulse clicks and determine the number dialled."

Car Computer

Ours was good, but Derek Knaggs really has built the real deal

Maker Profile

Derek Knaggs

Managing director

Derek Knaggs runs Flamelily IT, an IT supply and support company, and is studying Computing at the University of Worcester.

Find out more: **flamelily.co.uk**

Remember the car computer that we made earlier? Well Derek Knagg already beat us to it, and he's embedded the display in his dashboard and extended the setup to include screens for the rear passenger seats too.

"I removed the DVD player, which was a standard Ford head unit," explains Derek, "and then purchased a head unit from Xtrons. It's designed for the Ford Focus so it was a straight swap. The Xtrons radio has an S-Video input and that goes into the radio, so the Raspberry Pi displays as Auxiliary Input. There's two Auxiliary Outputs on the radio, so the Raspberry Pi sends a video to the main radio which then sends it back out to the screens in the passenger seats. What I've done is put in a device – like a VGA adaptor: it takes one input and puts seven out – that gives me the ability to have the Raspberry Pi running to the back screens on their own, so the radio can then control itself. I can have my kids watching movies at the back with the Raspberry Pi using an audio splitter (they've got headphones on), and we can be at the front using the normal radio controls, like the satnav for example. So that works well."

Above In this setup, the Pi is one of the inputs for the Xtrons head unit

Left It's difficult to see, but there are some difference to this NES compared to an original

RetroNES

Now you're playing with power. Raspberry Pi power to be exact, situated inside an old game console

Maker Profile

Chris Crowder

Programmer and database administrator

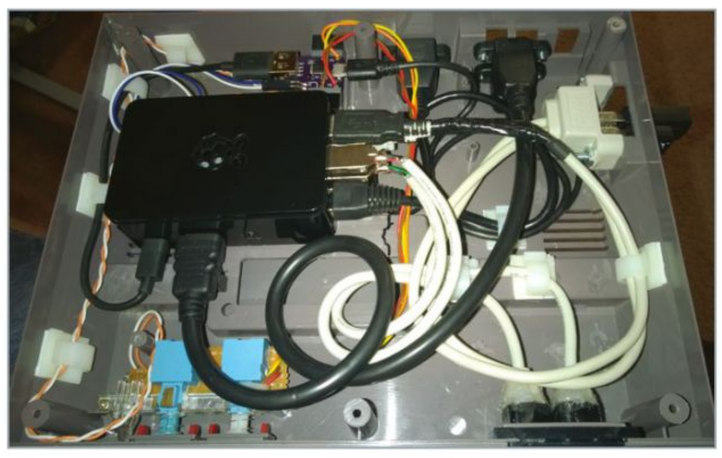
Working in the car industry, Chris develops manufacturing systems for production floor systems using .Net and SQL. In his down time, he likes to play videogames and tabletop games, but was previously limited to his PC for the former.

Find out more: **imgur.com/a/KPi2n?gallery**

> It was a little intimidating at first as I wanted to make sure that this project looked and felt like an NES

"It all started when my wife asked me what I wanted for Christmas", said Chris over email. "I had absolutely no idea but I had been wanting to mess around with a Raspberry Pi since it came out, so she got me a starter kit.

"While waiting for Christmas I started narrowing down ideas and found the RetroPi project. I thought that I would just install that, load some ROMs and call it good, then I remembered that I had some old NES and SNES controllers in storage. I went to get them and found my old childhood NES console along with the controllers. Once I got the NES back in the house and started looking at it, I found that almost all of the internals were damaged due to insects and moisture. All of the connectors were corroded and some of the boards had traces that were peeling. That is when I decided that I would use the Raspberry Pi to 'resurrect' the NES."



Above Everything is packed inside the original case, without needing to open it up to use it

Chris completely gutted the case and replaced the insides with a Raspberry Pi, hooking up I/O ports to the original connectors for the controllers and the AV cables and such. What's it like taking on a project with one of the most revered consoles in videogame history?

"It was a little intimidating at first as I wanted to make sure that this project looked and felt like an NES but with more flexibility. The biggest issue I ran into was that I wanted it to be able to work like an NES, meaning that if someone wants to play a game that they just turn it on, select a game and then they are playing. When they are finished all they have to do is press the power button to turn the console off. We can't do that easily with a Raspberry Pi since there is no ATX-style power switch. I was able to solve this issue with a Mausberry Circuit and a Python script. When the power button is pressed it communicates with the Pi via a GPIO connection and it runs the shutdown command. Once the Pi is shut down, the circuit cuts the power to the Pi."

The final product works great, with Chris reporting he can play on Atari, Sega and Nintendo games just fine. He's now looking to upgrade it with a Raspberry Pi 2 and increase the number of games he can play.

Refitting a NES

Do you fancy taking on the challenge of bringing your old console back to gloriously pixelated life? Thought so. In that case, you'll be needing this – here's a list of the equipment that Chris used to repair and revive his childhood NES console:

- A broken NES console – please don't do this to a working console
- Replacement NES Door
- Canakit Starter B+ – soon to be replaced with a Raspberry Pi 2
- Panel Mount Ethernet Cable
- Panel Mount HDMI Cable
- Panel Mount USB Cable
- USB A Male Connectors 10pk
- SNES USB Controllers
- Anker 13000 mAh 3 watt Battery – this is for when there is no power outlet nearby or you want to be portable
- Mausberry Circuit – shutdown circuit that uses your own switch, USB
- LEGO – to hold the Mausberry Circuit
- Gorilla Glue
- Blue 3-volt LED – the original NES LED was 12 volts, plus all of the other items in Chris' entertainment console are blue

Above The Mission Control desk groups the various functions into 'station' panels

Right Outside of playtime, this is just an ordinary homework desk. Almost

Mission Control Desk

Astronaut training begins early in Jeff Highsmith's home, with his sons running launches from their home-made Mission Control

Maker Profile

Jeff Highsmith

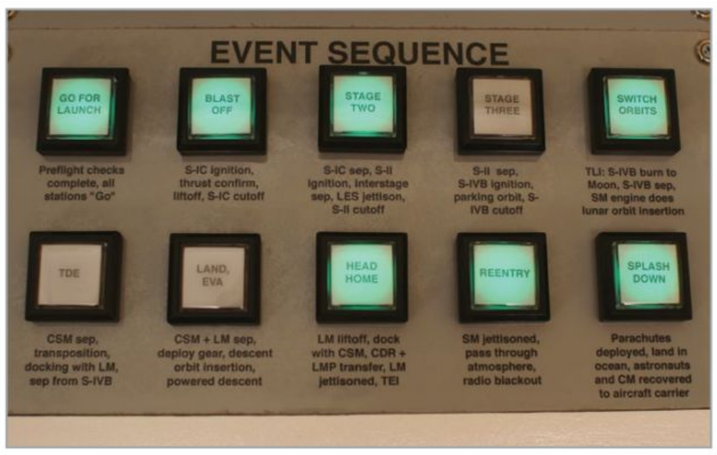
Tinkerer extraordinaire

Jeff Highsmith loves to make new and novel things. The medium isn't important, and he enjoys scrounging for materials and making do with what's at hand.

Find out more: **jeffhighsmith.com**

Astrocarpentry

The panels are assembled from bare components. Jeff ordered the switches and then designed the panel layout and labels on his computer, printing them on inkjet transparency – clear acetate – and then gluing those onto some fibreboard that he had spray-painted a metallic grey. He estimates that the whole budget was $700.



Above These reflect the real stages of a NASA mission and play authentic recorded sounds

Jeff Highsmith is probably the best Dad in the world. Not content to just build his son a desk for his room, he modified it so that space adventures can start with the push of a button.

"My eldest son was starting kindergarten and he needed a desk to do his homework on," Jeff tells us, "and since I like to build stuff I thought I would make him a desk rather than buy one, and I was thinking, 'What would make a really awesome desk?' Well, having lots of buttons and switches like a mission control desk! Carpentry-wise it was pretty simple to build.

"The Raspberry Pi is up in the front-centre behind a piece of flexiglass, next to the Arduino that takes care of reading the inputs. The Pi handles all the sounds and the logic – the gameplay aspect. That was my first Python experience and it was pretty good.

"The desk has got several modular panels and each has a different function. So in the real mission control at NASA there's a desk that controls the retro stage, and for this desk I made a retro booster panel and put a bunch of rocket noises on it. There's a capcom (capsule communicator) panel, so you put on a little headset and you can talk to the astronaut that is in the spaceship in the other room.

"There are a couple of panels that have numerical displays: one reads out some attitude numbers, like x, y and z in space, and there's one that monitors the astronauts' vital signs (supposedly). There's one that does mechanical spaceship noises, like pumps, heating elements, buzzing noises, fans etc. I wanted it to be like you're turning things on and off, not just pushing a button that plays a sound – that's why I have the toggle switches as well as push buttons. There's a spot for the iPad in the middle too – you can watch videos of rocket launches.

"There are homages to actual NASA emergencies, like the stirring of the oxygen tanks that led to the Apollo 13 explosion. I have a switch that makes it sound like it's stirring the tanks, then it makes an explosion sound and plays the audio from the astronauts talking: 'Houston – we have a problem'. There's a sequence panel too that has the different mission stages on it and each of those plays a real NASA soundbite, all the way from the launch to the landing on the Moon to the splashdown."

Spaceship Bedroom

After successfully accomplishing his desk mission, Jeff Highsmith set his sights higher

Mission control was but one small step. Next was the mission itself, as Jeff explains: "So my boys would hit the buttons on the desk and go through all those mission stages and run around with their toy rockets, but having the actual spaceship, I thought, would be cool. The spaceship has some panels similar to the desk, but it also has a small screen in there that goes to a video camera in the cargo bay. There's a motorised hatch on the side that you can open up by flicking a switch, and then the camera shows you a cargo bay with a robot arm inside it. You can't see the cargo bay when you're inside because you're laying on your back, but looking at the screen you can see it and the controls are in front of you. It really feels quite fun – I've got a little toy Hubble space telescope in there and I hung a piece of fishing line from the ceiling with a little bit of metal on it, and then I've got a magnet in the space telescope. So you take the telescope from the cargo bay with the arm, move it over and snap it onto the string that hangs from the ceiling – we call that orbit. Once it's in orbit you can pull the arm back in the cargo bay, close the hatch and your mission is complete, you can return to Earth. And then there's the inevitable mission to go and fix the Hubble...

Jeff's going to upgrade this awesome setup further. "Eventually," he says, "I've got some ceiling satellites planned, so I'll have them orbiting a track in one of the bedrooms and the iPad can monitor the different sensors on the satellites. The track will be a thin metal rod under the ceiling in an ellipse, and then each satellite will have a tiny wheel extending from the top of it, which has a very small gear motor on it, so it'll hang from the track on that wheel. The idea is that the kids can build satellites out of Lego, put them in the cargo bay, then winch them up into orbit."

You can see the robot arm reaching out of the cargo bay door to the left

> Having an actual spaceship, I thought, would be cool

Automatic Roller

Nearly a Rube Goldberg machine, we hope it plays 'Powerhouse' when used

Maker Profile

Emil Jaworski

Maker

Finding himself in a rented apartment for a few months, Emil has decided to upgrade it himself.

Find out more: **imgur.com/a/OYdPo**

There are many schools of thought regarding your sleeping environment to help aid better and more restful sleep. No electronics in the bedroom, try and relax before going to sleep, take a cool or hot shower depending on the time of the year. Some people require pitch darkness to get a good night's sleep though, while others like to wake up when the sun rises as part of a natural body reaction. Emil likes to do both of these things with his Raspberry Pi that automatically shuts and opens his blinds at specific times of day:

"7:30: Press up button, wait 10 seconds (I have smaller windows on the bottom and bigger above them, so after these 10 seconds, shutters are going to be open only on the bottom ones), press stop button. 8:00: Press up button. 22:00: Press down button."

Unfortunately, it means he has some electronics in his bedroom, but whatever works for him.

RasPi Terminal

We've had lots of Raspberry Pi over the years, but what about Apple Pi?

Maker Profile

Austen Barker

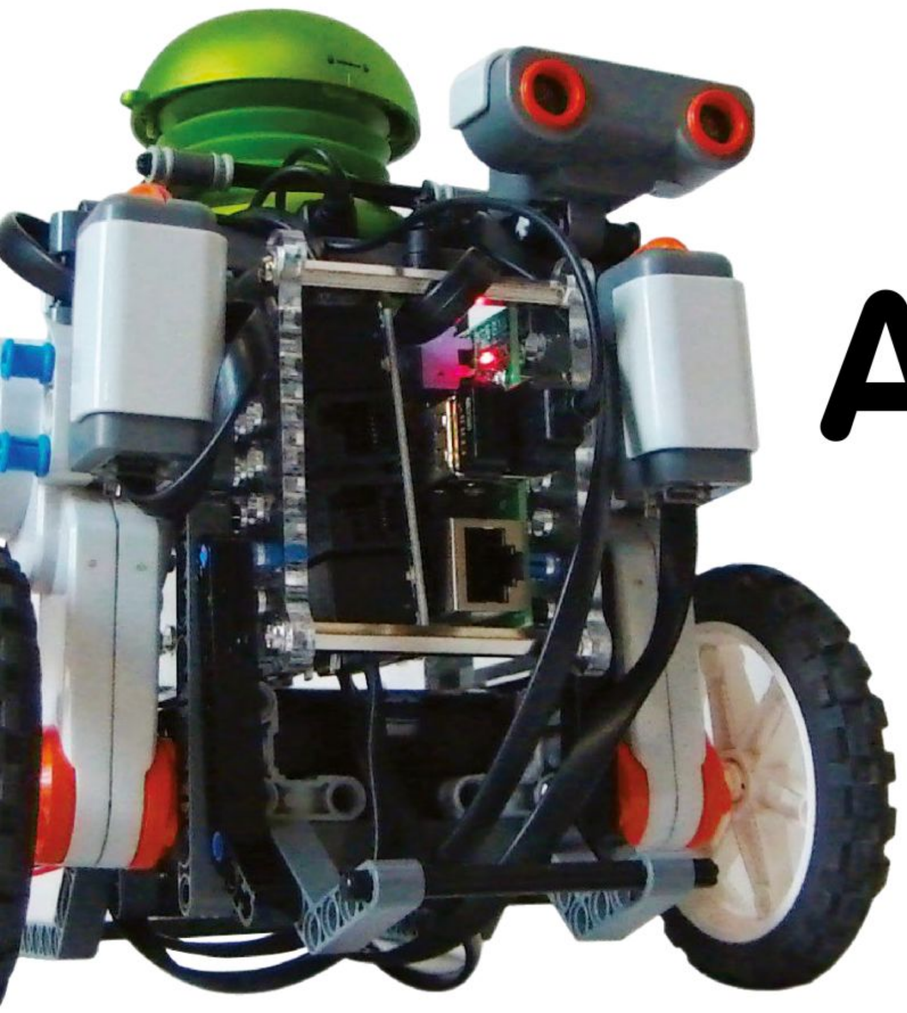
Engineering student

A Californian student that has a habit of messing around with any old computer hardware that he can get his hands on.

Find out more: **imgur.com/a/vOsML**

"In my spare time I like messing around with older computer hardware that I come across. Originally I found the need for a terminal to connect over SSH to a server that I was using for an internship. It was more efficient than devoting a more powerful machine to it. I took a server to school with me to continue working for the same company as I studied.

I had a Raspberry Pi also sitting around and I started hooking it up to a monitor from an Apple IIc that I found at my university's electronics surplus. This became valuable when I started having to log in to a school server over SSH to compile assignments for my programming classes. Eventually, I found a keyboard from a Macintosh 512k and made it work over USB with a microcontroller and a custom wired key matrix, and I paired it with the monitor and Pi."



Alarm Clock Robot

Chasing your alarm clock may sound like a nightmare to some, but here it is a reality

When we talked about Rolly the alarm clock robot around the office, most people burst forth with a string of expletives not fit for print. It's a delightfully evil invention – an alarm clock you need to work for to turn off. It sounds like a great invention, getting people who have trouble waking up to actually get up out of bed and start the morning.

Like any other Dexter Industries robot, it runs on BrickPi, the LEGO Mindstorms adapter for the Raspberry Pi that enables it to interface with LEGO kits via programming.

"Today almost everyone uses their phone as an alarm clock, which has a range of benefits," the website explains. "Phones are easy to set, easy to adjust, play custom songs and can even sense when is the best time to wake you up. The problem is, unless your phone is across the room, we use our phones so much we can literally use them in our sleep. Why not build a robot that is able to do all these things, but won't stop till you get up and start moving! Our robot will be able to easily move randomly around the room over any surface, playing a custom alarm tone.

"In order to set the alarm, simply link the program on the BrickPi to your Google account and it will search events with the title 'wake1' and automatically start the alarm at the event's time. This means the alarm time can very easily be adjusted using any device that can access your Google Calender."

Maker Profile

Taryn Sullivan

Advisor

Taryn is an international businesswoman. As well as flying between Shanghai and DC for her own engineering business, she now works with Dexter Industries to promote robotics education.

Find out more: **bit.ly/1BTYljv**

Robotics education

There are many ways to get kids excited with coding, like making games in Scratch, modifying *Minecraft* and teaching via robotics. The latter is a new concept but still has the same merits – a physical creation that children are excited about and react to the programming they've done on it. Visible results and instant gratification is a great way to get imaginations fired up.

PiFM Radio

Sometimes the simplest hacks can open up whole worlds of possibility

Maker Profile

Oskar Weigl

Electronics engineer

Oskar is an electronics professional and hobbyist, as well as an avid forward and reverse engineer.

Find out more: **bit.ly/1om6BQE**

When studying at Imperial College London, Oskar Weigl and Oliver Mattos turned the Pi into an FM radio by connecting a wire (antenna) to GPIO 4 and using a custom Python module.

"There is a clock generation module in the hardware of the Raspberry Pi chip that lets you output a clock signal at a user-selected frequency," Oskar explains. "We used the DMA controller to send commands to the clock module to change the frequency and achieve frequency modulation. We had to overclock the clock generation module by a factor of 20. The sound is 14 bits per sample, enhanced to a higher number using delta sigma modulation and the range is at least 50 metres."

Ras Pi Smart TV

Is your smart TV not smart enough? Open the case and put a Pi inside

Maker Profile

Tony Hoang

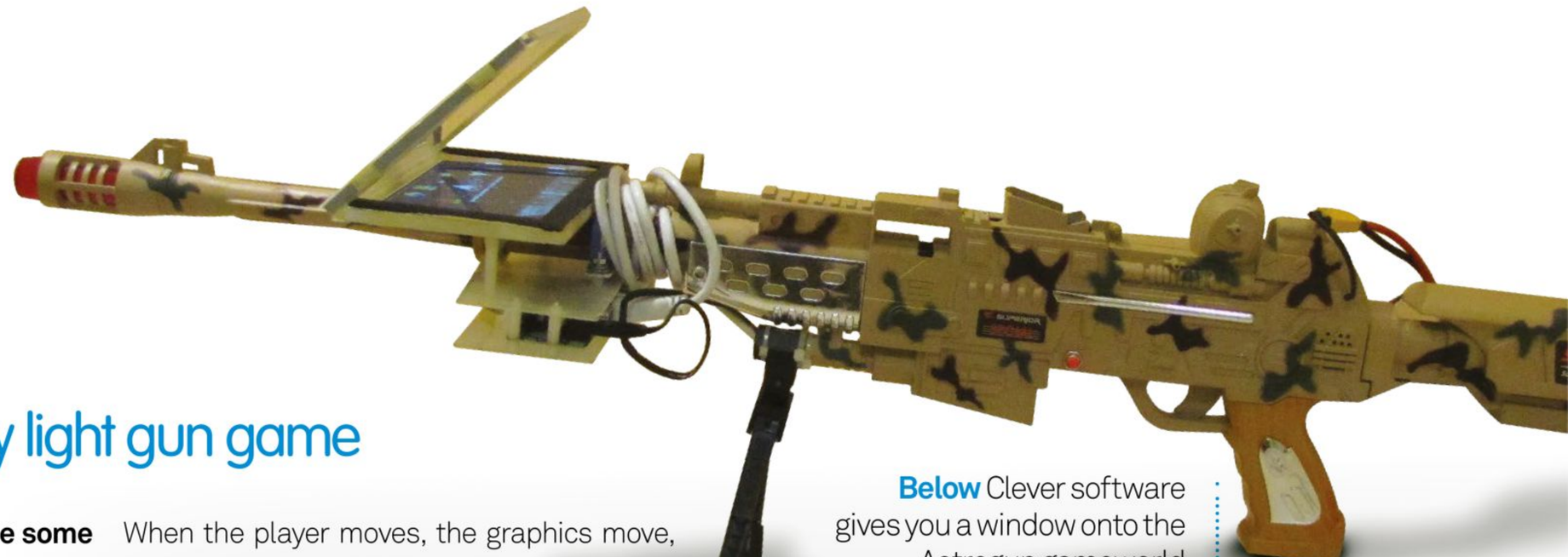
Graduate researcher

Tony Hoang is a PhD student studying computational biophysics and single molecule research at SUNY-Albany in Albany.

Find out more: **linkedin.com/in/tonyphoang**

"There's plenty of room for additional electronics inside the Hisense LED smart TV," begins Tony Hoang. "There's a large flat area for electronic parts in the centre of the TV where I placed my Raspberry Pi. The dual down-facing speakers were quite loud, so I removed one and replaced it with a USB hub. The back panel was mostly flat, so finding a spot for the LAN port and HDMI output wasn't too hard.

"The Raspberry Pi is powered by the logic board of the Hisense. There were the obvious 5v-500 mAh outputs from the 2x USB 2.0 ports, which I tried but I found out that the logic board shuts off the power to these ports when the TV turns off. To keep the Raspberry Pi turned on, I probed the logic board with a multimeter and found one from an unused GPIO."



Astrogun

An augmented reality light gun game

Walking around Maker Faire, you see some weird and wonderful things. If you'd been present at the Jerusalem Maker Faire you may have seen people wield a giant toy gun to shoot down virtual asteroids in Avishay's AR motion game Astrogun.

"In the Astrogun lies a Raspberry Pi computer," Avishay explains. "An IMU card connected to it (Sparkfun's MPU-9150 breakout board) gives it the ability to sense the unit's orientation. The Pi is then able to draw the elements seen from that angle. When the player moves, the graphics move, giving the 'object in the room' sense."

Why the Raspberry Pi? It was due to time, according to Avishay: "I had a short time to bring it to a working thing, so I had to pick a platform that was capable of the task and easy to use. The RPi fits that criteria. I used many software components designed for the RPi or tested on it – the Pi3D and RTIMULib. The combination of Pi as a hardware platform and Python as a programming language is the fastest way to materialise ideas."

Below Clever software gives you a window onto the Astrogun gameworld

Maker Profile

Avishay Orpaz

Electronics engineer

Find out more:
bit.ly/1AYPSqg

Pirate TV

Go all the way and totally rebuild Android TV!

Maker Profile

Donald Derek Haddad

Software engineer

Donald is an open source hacker.

Find out more:
donaldderek.com

The perfect companion to the Smart TV hack, Donald Derek Haddad's project is a custom TV interface you can make yourself.

"Pirate TV is a smart television application that runs on the Raspberry Pi with the Raspbian OS," Donald tells us. "It's built with open source tools and shipped with a free remote controller, your mobile device. At its core lies a Node.js application that runs a web server with Express.js/Socket.io to handle users requests from the remote and trigger shell scripts. The TV user interface is rendered on a Chromium instance in kiosk mode. Videos are streamed from YouTube or other channels played on OMXPlayer, and cached including 1080P HD content. This project is a work in progress and it's not going to be able to tap into a lot of the content, which makes a Google (now Android) TV or other commercial platforms so valuable." Check out Derek's tutorial: **bit.ly/1l6YKpj**.

Pye Radio

Old meets new in this modified radio

Maker Profile

Tony Flynn

Senior embedded systems design engineer

Find out more:
bit.ly/19zrgPl

Upcycling is a great concept: recycling a product using new technology to make it relevant in the modern world. While standard analogue radio isn't dead yet, its nice to have options when listening to music. This is where Tony's idea came in:

"I'd been working on a Raspberry Pi to play music streams through my stereo. Once this was running I integrated the Raspberry Pi into an old radio named a Pye! With a background in woodwork and engineering this seemed like the perfect project."

However, this project wasn't a walk in the park for Tony: "The hardest part of the conversion is linking the tuner knob to the rotary encoder. For this radio I used the spring from an old biro as a drive train to link the tuner knob spindle to the rotary encoder through 90 degrees."

Did Tony have qualms about heavily modifying such a classic design? "None whatsoever! Some people don't like the modern style now stamped on this old radio, I think it's a new era, new look!"

Right The body has also been cleaned and repainted

Digital Camera Conversion

Classic aesthetics with modern convenience, this old-school camera has been upgraded with a Raspberry Pi

Maker Profile

Pete Taylor
Web manager

Pete works for a charity and has always tinkered with computers, ever since he got a hand-me-down BBC computer. He likes that the Raspberry Pi returns to a time when you could hack your own computer without making an expensive mistake.

Find out more: **bit.ly/1MKRASw**

> The Raspberry Pi is a maker's dream – it's cheap and cheerful, and the community built around it is a brilliant resource

Old cameras have a very specific design aesthetic that it seems has been lost to time, although nostalgia for them is still very strong in certain circles. Unfortunately, while nostalgia, desire and working cameras still exist in the 21st Century, usable film is quickly dwindling in supply. So if you like the aesthetic and aren't too bothered about using the old photo process, why not upgrade the insides with more modern technology?

"I wanted a suitable case for the Raspberry Pi camera board," Pete tells us, "and the Holga seemed a perfect fit. Most of the available cases are either a bit ugly or suited more towards stationary webcam-type applications."

Why the Raspberry Pi, though?

"The Raspberry Pi is a maker's dream – it's cheap and cheerful, and the community that's built up around the Pi makes a brilliant resource when you're stuck with a problem or want to find out more."

The entire build doesn't require a massive amount of components either. As well as the actual camera and Pi itself, Pete used a Raspberry Pi camera board to actually take photos, a Wi-Fi module for connecting to it remotely, a battery and a switch for it, and a few buttons and resistors to wire up the camera's control buttons to the Pi.

"It works better than I expected!" Pete said about the quality of the finished product. "Although it seems a bit daft to build a camera that's about as good as you'd get from a cheap camera phone, it's changed the way I take pictures. By removing the instant replay – most people seem to view the world through the displays on their phones – I can concentrate on taking the photo. Only seeing the pictures when I've taken the camera home and downloaded them to my PC adds a bit to the film nostalgia and I'm often surprised by the photos I've taken. Plus I've received some nice feedback about how the camera looks."

This is only the first iteration, and Pete has plans to make the next build easier and also use the original lens with the Pi camera.

Do it yourself

Version two of the project will result in a kit that people can use to convert their own cameras "that doesn't require you to take a Dremel to the insides of a Holga!". He's not decided yet on whether to make a kit that converts a Holga, or a kit that builds a Holga-esque case around the Raspberry Pi itself. Either way, the whole thing should also have a better photo-taking capability, which is the ultimate goal.

Top-left The camera of the past, updated for today

Left The Pi is the perfect size to fit inside the camera case

Bitcoin Pool Table

Modern pool pros pay digitally by scanning a QR code to insert Bitcoins

Maker Profile

Stuart Kerr

Technical director

Stuart Kerr is the technical director of Liberty Games, which specialises in classic table and bar games. You might have heard of some of their hacks.

Find out more:
libertygames.co.uk

Liberty Games loaded a Raspberry Pi into the side of a pool table that enables people to make payments via a Bitcoin app on their phone to release the balls.

"We did this with the Raspberry Pi 1", Stuart tells us. "We tried to install the entire Bitcoin client on the Pi but it was struggling, and downloading the whole blockchain caused issues too. It was going to be connected to the Internet, so we offloaded the heavier work to a server able to handle the blockchain. The server receives the Bitcoin payment and then communicates to the Pi securely that the payment has come in, and it syncs up the prices as well.

"We connected a PiFace via a breakout board that's monitoring on WebRCT for the go-ahead from the server, and once it gets that, it sends the physical voltage to the electrical ball release mechanism."

Above The price of a game is displayed, in Bitcoins, on the PiFace display module

Fireball Pinball

Ian Cole and his sons enter the high score boards with Fireball HD

Maker Profile

Ian Cole

Maker and geek raiser

Ian Cole is a keen maker, hacker and inventor, and regularly blogs about his family projects with his two sons. Their Fireball project, for example, grew out of an innocent 'Can you make it playable?'

Find out more:
raisinggeeks.com

At raisinggeeks.com, Ian Cole and his two sons love a challenge. So the chance to repair a game table gave them the perfect Raspberry Pi primer.

"We've taken an existing pinball machine playfield," says Ian, "and built a new game from it. This required learning the underlying hardware first. Then we learned how to use the Raspberry Pi, pygame for sound and text graphics, and omxplayer for HD video, and we connected the software tools with the hardware of the pinball machine.

"We built a MOSFET circuit on a breadboard to test a single solenoid. When that worked, we duplicated it onto a hand-soldered protoboard and extended it to control the five solenoids. The Raspberry Pi handles graphics, audio, scoring rules, saving scores, etc. One Arduino drives the lamp and switch matrix, another drives the solenoids. The three are connected with an I2C bus."

LED Voice-Controlled Coffee Table

Light up your living room with a voice-controlled light show for a coffee table

Maker Profile

Mikel Duke

Software developer

Mikel loves programming, photography, biking and hiking, and sharing his projects.

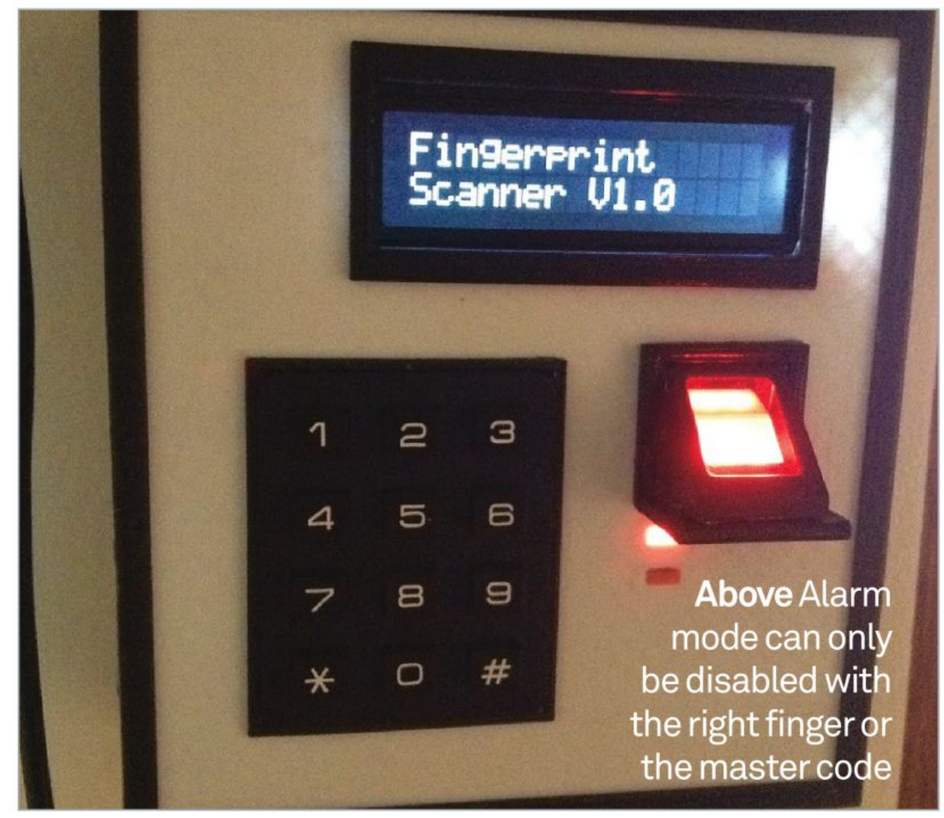
Find out more:
bit.ly/1F0HDkW

Mikel found himself in a quandary: he didn't have a coffee table. However, instead of just heading to Ikea and picking up a LACK, he decided to make one himself. Not just any coffee table though, one that lit up using a Raspberry Pi to control the sequence.

"I used the Raspberry Pi because it was easy to work with and has a great community. Whenever I run into an issue, there is always documentation on how to fix it. I wanted to have network support and do more complex operations, like loading images, than I could do just using an Arduino with Ethernet. The combination of the two really made development faster. Now that Pi4J has added support for the Raspberry Pi's integrated SPI, I am working on controlling the LEDs directly, without the Arduino."

Recently, Mikel added voice control to it using Google Now. "After you say a command to Google Now, Autovoice, a plugin for the Android app Tasker, intercepts the response back from Google. If the response matches a Tasker task, the task will be launched. I have a few tasks set up to make web service calls, basically calling a URL with parameters. This URL connects to a Java based web service I created, running on my Raspberry Pi."

Mikel also encourages users not to use coasters, as drinks light up well when the table is switched on.



Above Alarm mode can only be disabled with the right finger or the master code

Project Jarvis

The 1950s future today with your voice-activated smart home automation system

Maker Profile

Mayur Singh

Computer systems engineering student

Located in South Africa, Mayur's expertise is in embedded systems and he is familiar with microcontrollers and systems that enable outside hardware interaction, such as the Raspberry Pi and the Beaglebone Black.

Find out more: **bit.ly/1Evlmci**

Left The Jarvis interface is gorgeous, but you can control the system from anywhere using your smartphone or your voice

Home automation has been a thing for years now, with a fair few intrepid engineers and DIYers modifying their home. With wirelessly controlled lights, heating, garage doors and many other household items, it's amazing what some have achieved. Science fiction has always portrayed houses with advanced systems and Mayur has been busy creating such a system.

"The major inspiration came from watching the *Iron Man* movies, which is where the name Jarvis comes from. I decided to build a simple home automation system, however over the many years and many reinstalls of Windows, I lost my program. I built a new and improved home automation system, which features an AI assistant and more functionality, after I saw the third *Iron Man* film."

While home automation is one part of the project, convenience isn't the only factor.

"The main function of the system is to help save energy in homes," Mayur tells us. "Jarvis can read and monitor the electricity usage per light or appliance and this lets the AI perform certain tasks. These tasks are determined by outside factors, like whether or not a light should be switched on if there is adequate natural light in the room. This is a basic example but other factors influence the determination of the AI, which has control over the power to each light or appliance. The data is logged throughout the month and the system uses that information to achieve better results the following month. This is smart home automation."

> The Raspberry Pi is a maker's dream – it's cheap and cheerful, and the community built around it is a brilliant resource

The Jarvis AI is a major part of it, and you can talk to it. "He can control your home using voice interaction and make basic decisions about energy savings in a room. I have also recently built a wall-mounted tablet system that links to a local online grocery store. It uses voice control to identify what products you need and adds the items to a list until you specify the delivery."

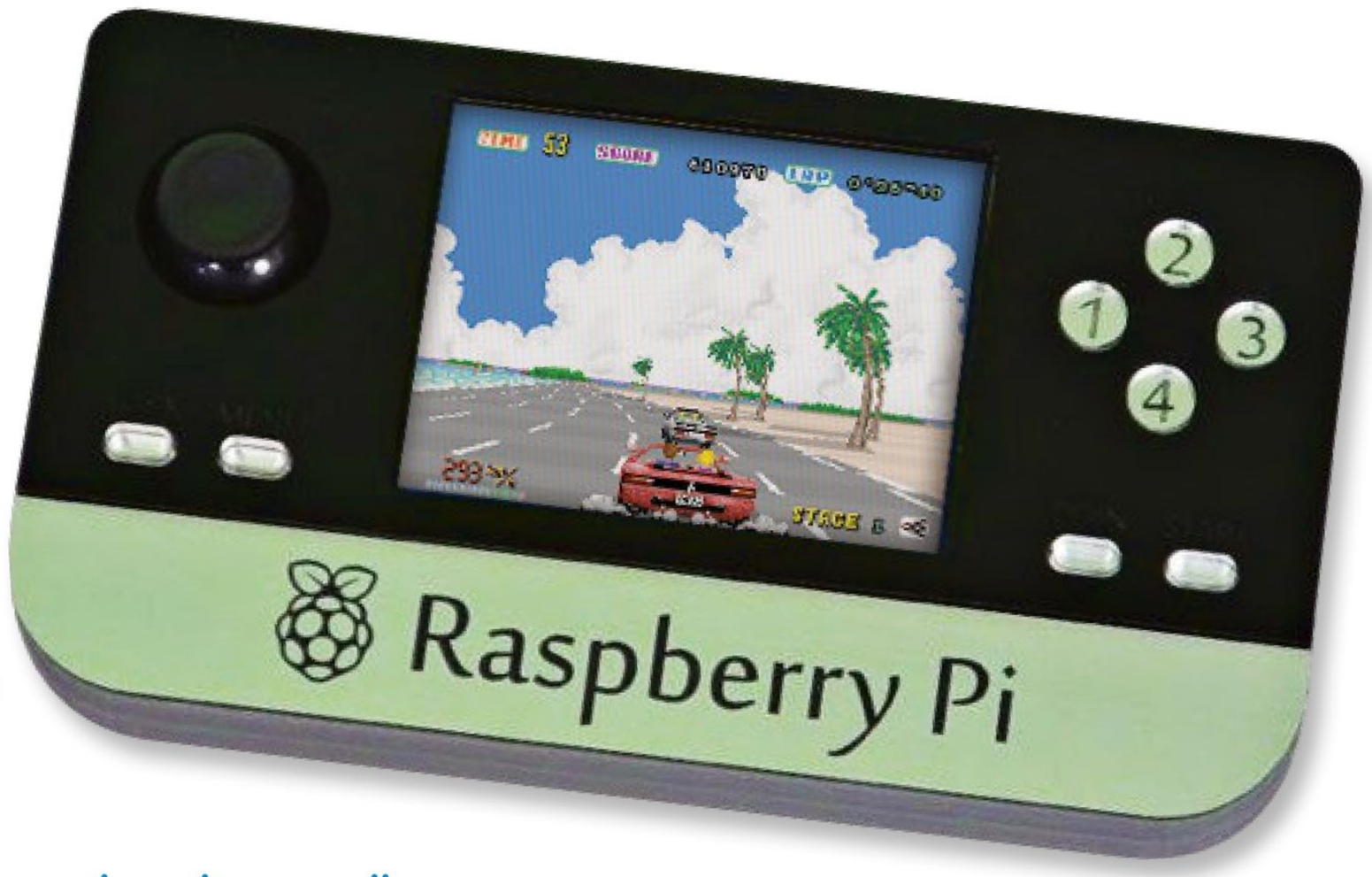
With a year and a half of work on the project, everything Mayur has completed works reliably, but there are still more functions he wants to add.

The voice

The Jarvis that this project is named after is well-known for its snarky remarks and tone of voice. Choosing an actual voice that works though is a hard enough task without going for a certain style, as Mayur explains:

"It's been difficult trying to find voices with a good API and price but I have settled for a free API which offers cloud conversion and just sends an MP3 sound file back. It takes longer but it ensures that the voice works on all OSs."

If you're interested in having a go at creating your own Jarvis-esque home automation system then thankfully you're in luck. You can check out Joey Bernard's extensive three-part series (beginning on page 76 of this very book) that explains how to set up a digital assistant that uses Pyaudio and a few voice recognition modules in order to parse your spoken commands.



Portable Pi Arcade

Love old arcade games? With Ben Heck's hack, you can play them all on a single hand-built, hand-held console

Above Install MAME and you will never run out of arcade games to play

Ben Heck has built two versions of the Portable Pi Arcade. The first was the original Portable Raspberry Pi project (**youtube.com/watch?v=dUZjzQuTNX4**), where he hacked the Pi to reduce its size and opened up a USB game controller to extract the circuits. With a new assembly in place, he 3D-printed a custom-designed case, put the new device together, and booted up MAME (the Multiple Arcade Machine Emulator) to play old-school games.

His recent revival of this earlier project was even more home-made. For the Raspberry Pi MAME portable gaming device (**youtube.com/watch?v=zrEj1aQRbpw**), Ben made a circuit board from scratch, fitting all the components into a new 3D-printed case that, rather than resembling a Game Gear, looks pretty close to a Game Boy.

Above Watch the video to see how Ben build and tests the custom circuit board

We asked Ben to take us through the original version of his project: "My first portable Pi project was a small, battery-powered unit for gaming," he begins. "It had a single USB port for Wi-Fi or external storage and we featured it back on season three of the show. The screen came from a cheap NTSC LCD screen that Amazon sells for use as a car's backup camera. The buttons I laser-cut myself and the case was 3D-printed." And in terms of physical modifications to the Pi? "Mostly I removed the taller through-hole components," he replies, "and attached the Teensy HID (used for controllers) directly to it. I also moved the secondary USB port."

As you can see above, the case is very well made. "I did the initial layout in Adobe Illustrator, for the laser-cut portions," explains Ben, "then transferred the whole design to Autodesk 123D to create a 3D-printable file. Hand-writing the buttons for the controls was the most challenging part of this project. It was the most time-intensive part and required a lot of precision and attention to detail."

Ben is no stranger to taking apart consoles and controllers for his Pi hacks – but he also makes one-handed accessibility controllers. "In addition to all of the other projects and hacks, we modify gaming controllers for people who have difficulty using existing ones," Ben tells us. "On the show we've featured a few of them – Xbox One, PS4, even the Wii. Now we build these controllers by request and they can be ordered off my website, though we only do Xbox 360/Xbox One controllers as those use PCBs throughout (instead of silk screen circuits like the PS4). Recently I trained Felix (an assistant on element14's *The Ben Heck Show*) on how to do it, so he's been helping and working on them in his spare time as well."

Maker Profile

Ben Heck

Master hacker, creator of **The Ben Heck Show**

Ben Heck is an online sensation and a pillar of the maker community, putting out amazing how-to videos for games console hacks and all kinds of different Pi projects. He's done it all on *The Ben Heck Show*.

Find out more: **benheck.com**

Heck's hacks

Pi Point and Shoot: A Raspberry Pi camera module, PiTFT from Adafruit, PlayStation 3 controller battery and additional parts were all made into a point-and-shoot camera.

Pi Retro Computer: A tribute to the BBC Microcomputer from the 1980s, Ben mounted a Raspberry Pi to a self-made wooden case, HDMI port, on/off switch and USB hub for an 'old-school' feel computer and carrying case.

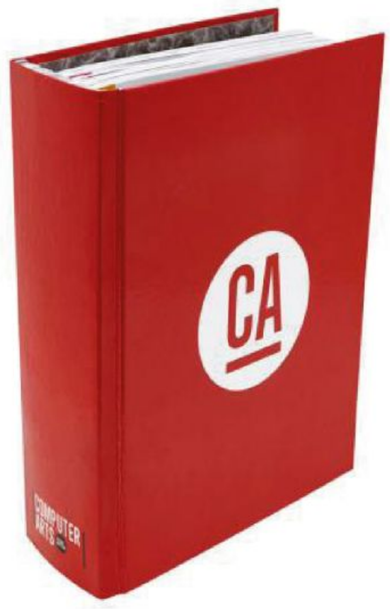
Handheld Pi Console: Ben hacked a Raspberry Pi single board computer to make it smaller. Combined with a composite LCD wireless keyboard, lithium battery power source and USB joystick, he created a handheld gaming console.

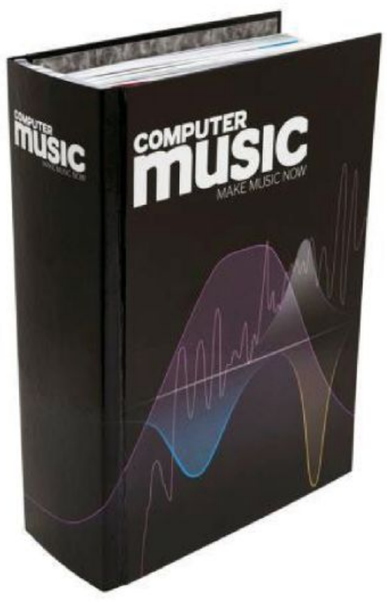
SUBSCRIBE AND SAVE UP TO 74%

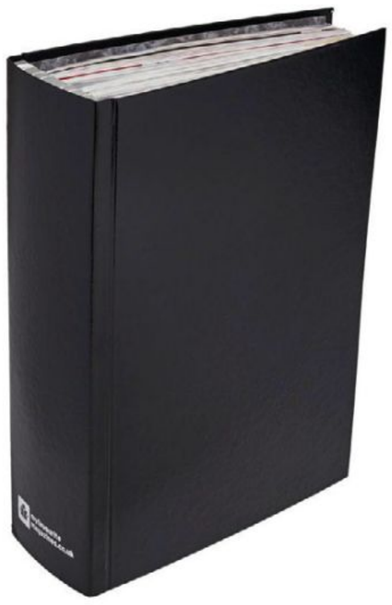
Every issue of your subscription, delivered direct to your door. Print & digital editions available.

NEAT STORAGE

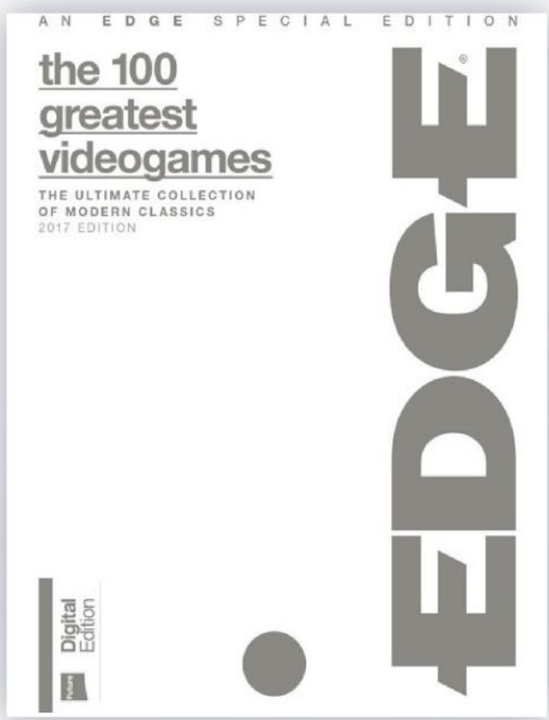
Store up to 13 issues of your magazine subscription in a coordinating slip case or ring binder.

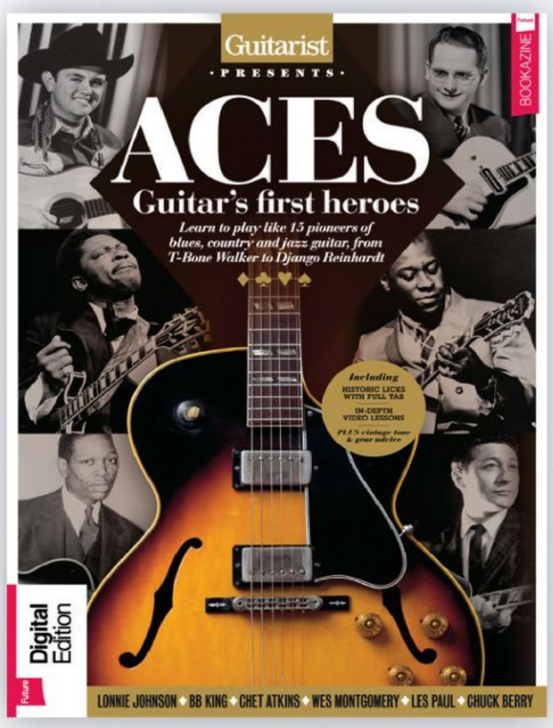






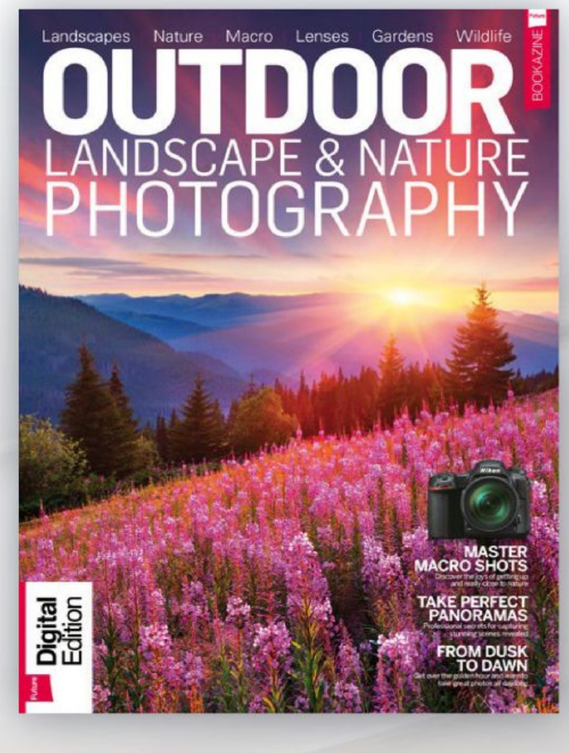
myfavouritemagazines.co.uk

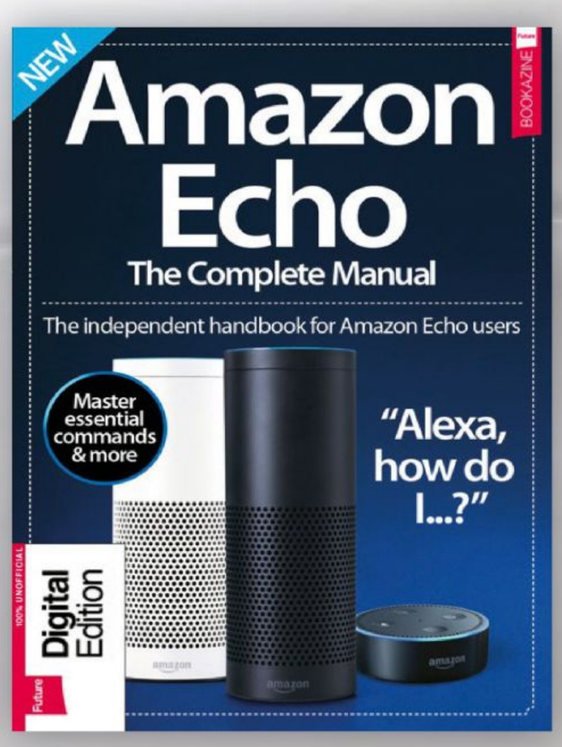


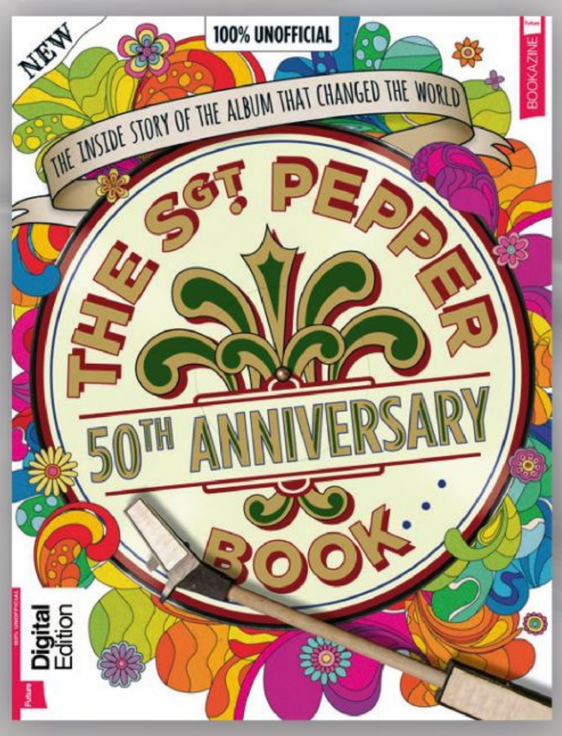


DISCOVER MORE GREAT GUIDES & SPECIALS

From photography to music and technology to field sports, there's something for everyone.







Our magazines and binders make perfect gifts too. And don't worry if you're not quite sure which titles they'll enjoy the most because we have a great choice of gift vouchers available to buy.

No hidden costs | Shipping included in all prices | We deliver to over 100 countries | Secure online payment

FUTURE

myfavouritemagazines
Official Magazine Subscription Store

myfavouritemagazines.co.uk

HOW TO USE FileSilo

EVERYTHING YOU NEED TO KNOW ABOUT ACCESSING YOUR NEW DIGITAL REPOSITORY

To access FileSilo, please visit **www.filesilo.co.uk/bks-2095**

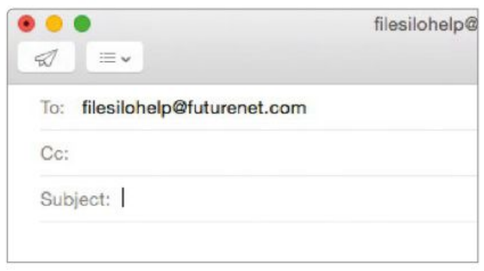
01 Follow the on-screen instructions to create an account with our secure FileSilo system, log in and unlock the bookazine by answering a simple question about it.

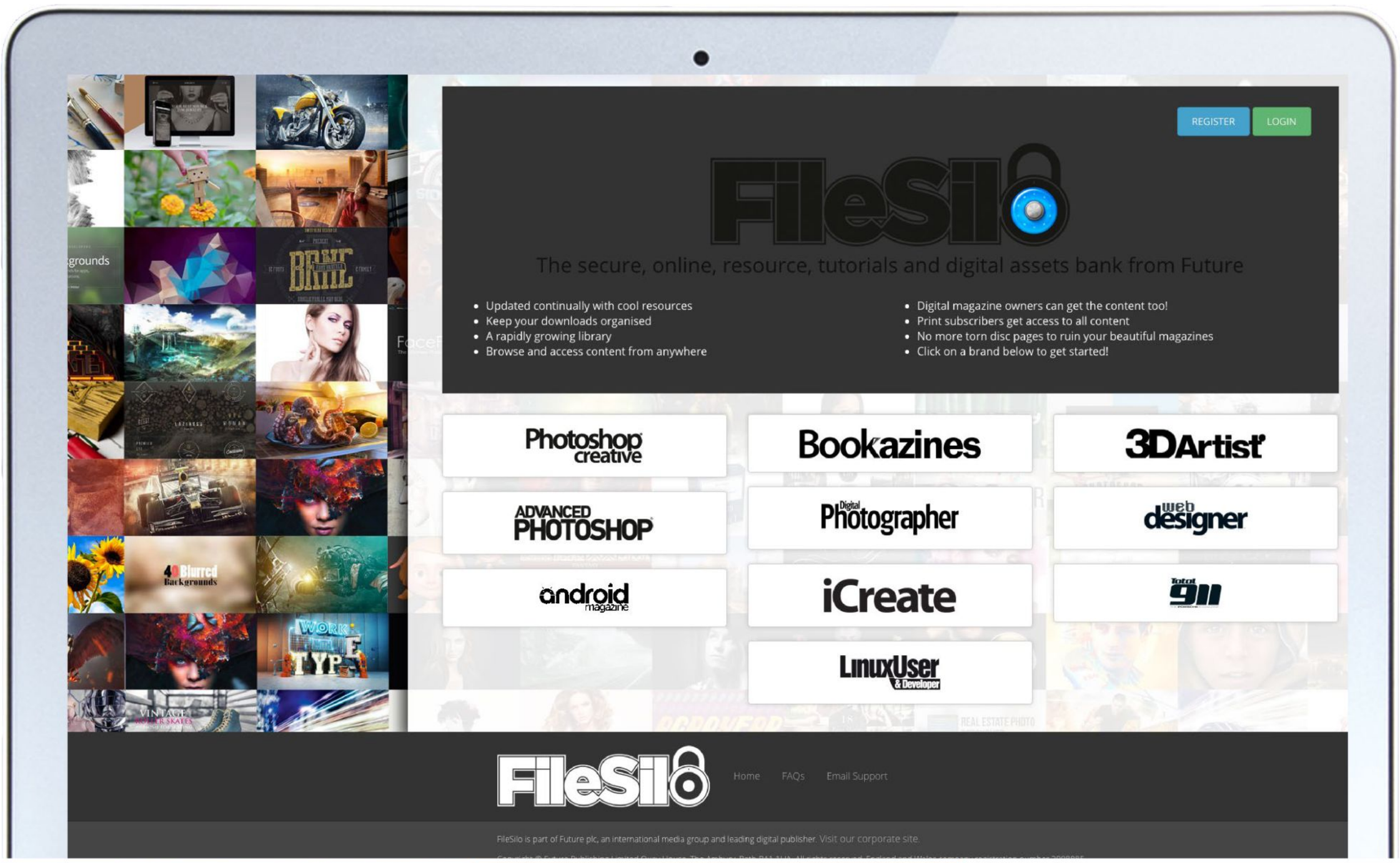
You can then access the content for free at any time, and download it to your desktop.

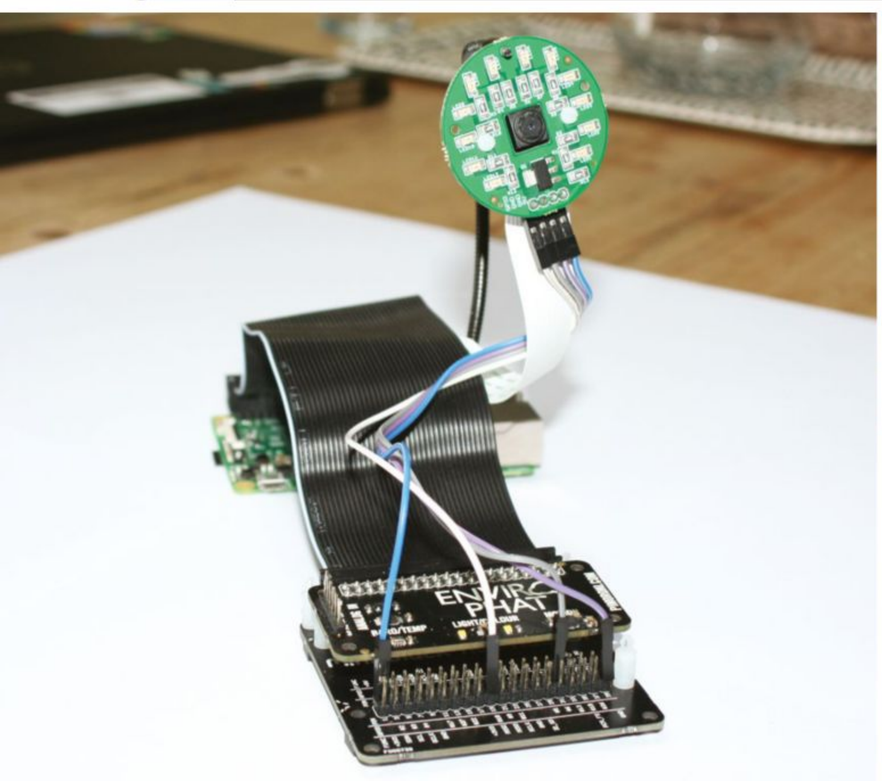
02 Once you have logged in, you are free to explore the wealth of content available on FileSilo, from great video tutorials and exclusive online guides to superb downloadable resources. And the more bookazines you purchase, the more your instantly accessible collection of digital content will grow.

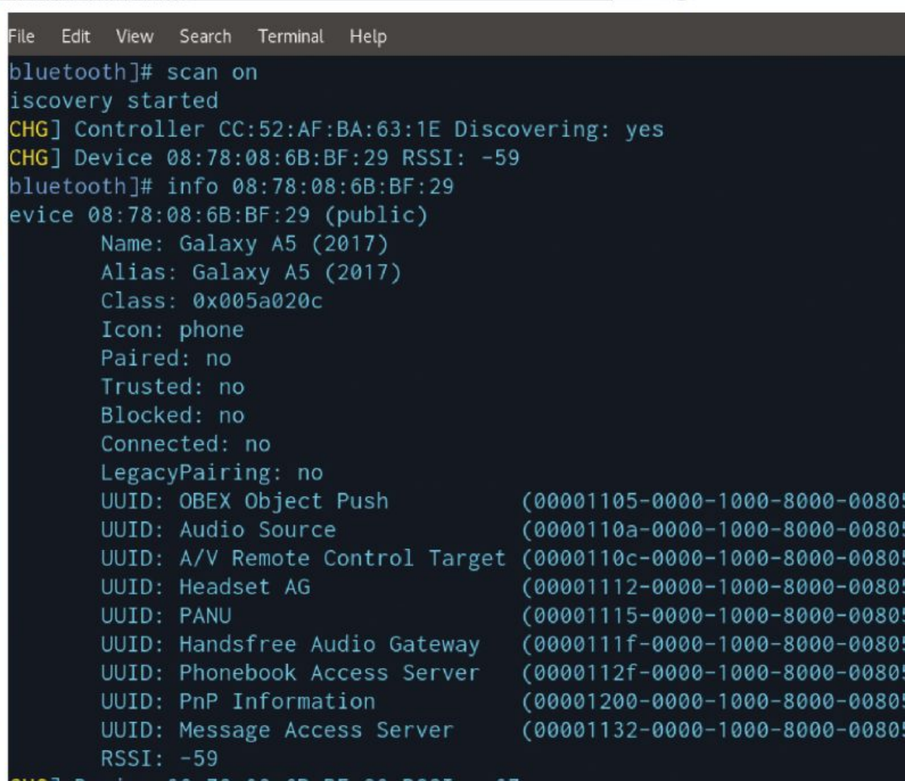
03 You can access FileSilo on any desktop, tablet or smartphone device using any popular browser (such as Safari, Firefox or Google Chrome). However, we recommend that you use a desktop to download content, as you may not be able to download files to your phone or tablet.

04 If you have any problems with accessing content on FileSilo, or with the registration process, take a look at the FAQs online or email **filesilohelp@futurenet.com**.





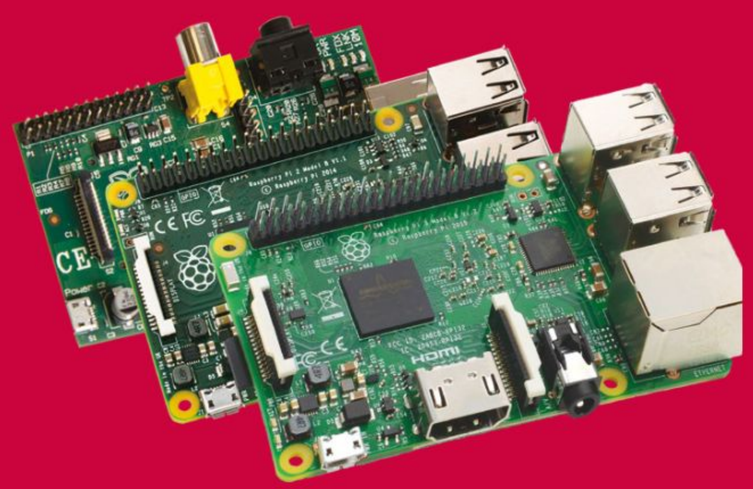


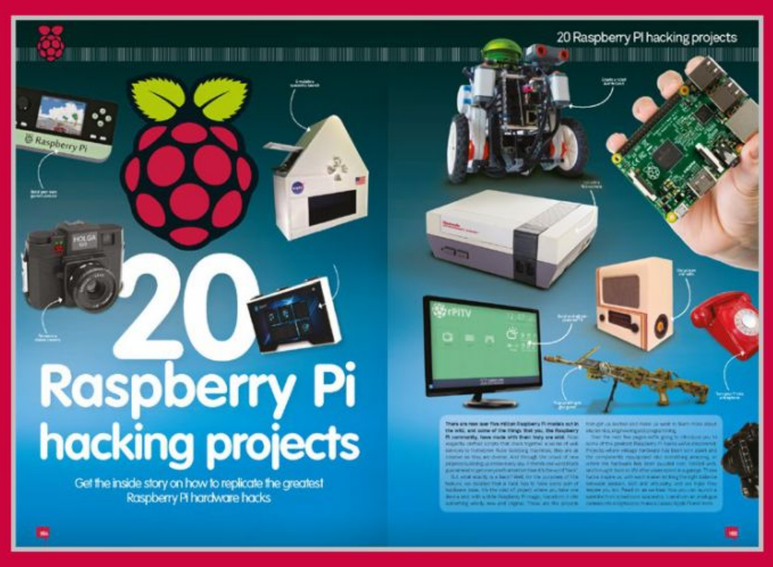


100% INDEPENDENT

Raspberry Pi Annual

Everything you need to get the most from Raspberry Pi





Get help with hacking

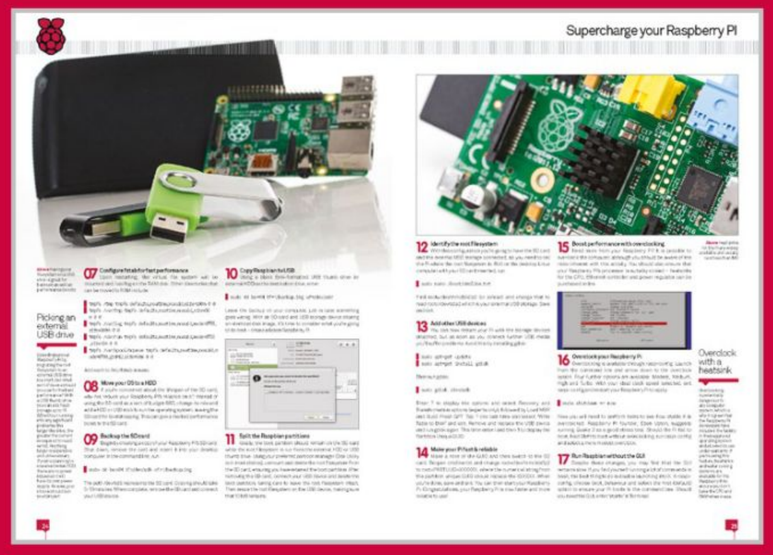
Discover the essential electronics out there right now for Pi hackers

Upgrade your gaming

From controllers to games, use your Pi to have some fun

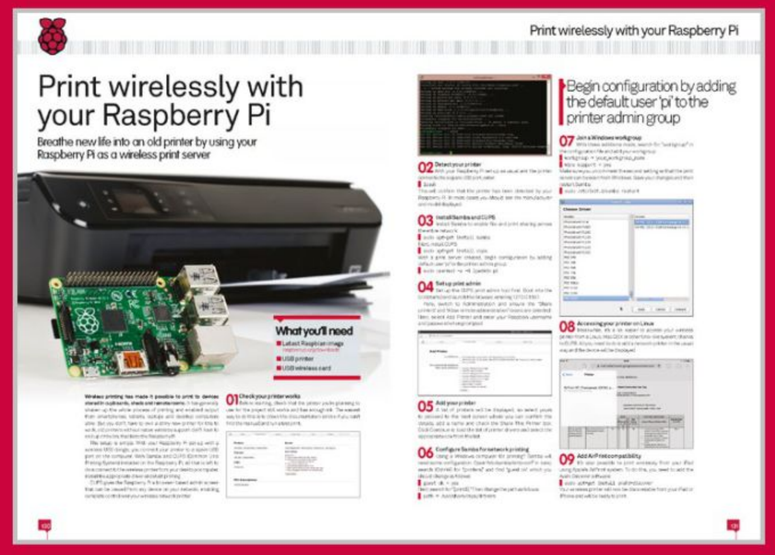
Take the next step

Further your Pi knowledge with tried and tested tutorials



Supercharge your Pi

Help your Pi reach its full potential with our handy guide

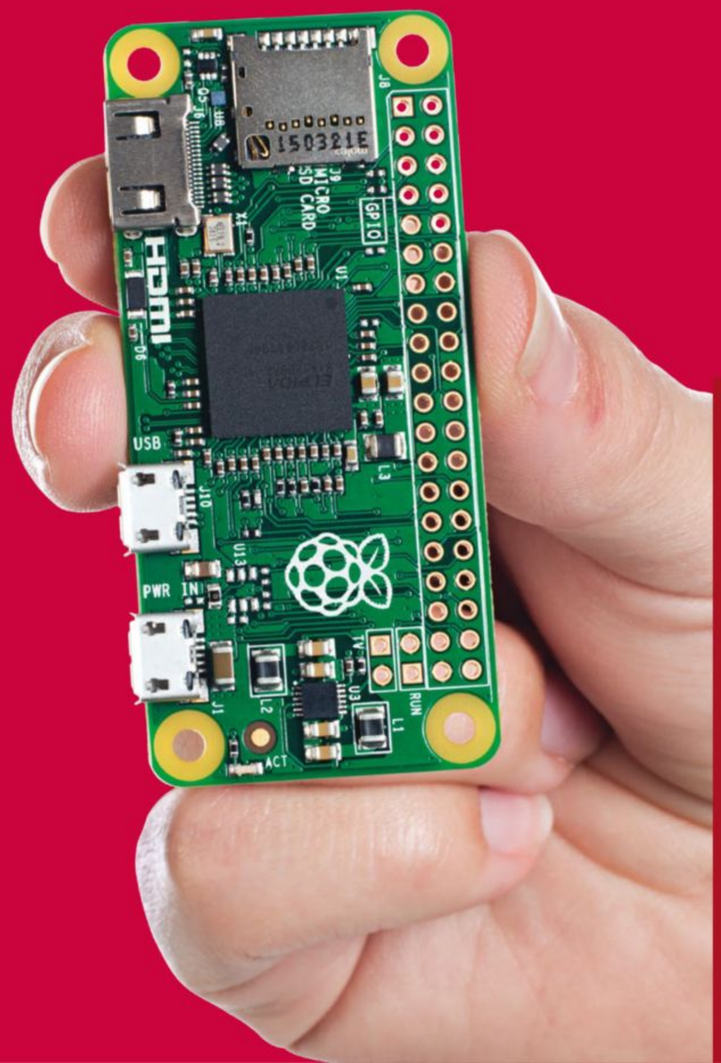


Build smart machines

Learn how to combine your Pi with household tech

Cool starter projects

Discover a variety of fun Pi Projects for you to create



BOOKAZINE