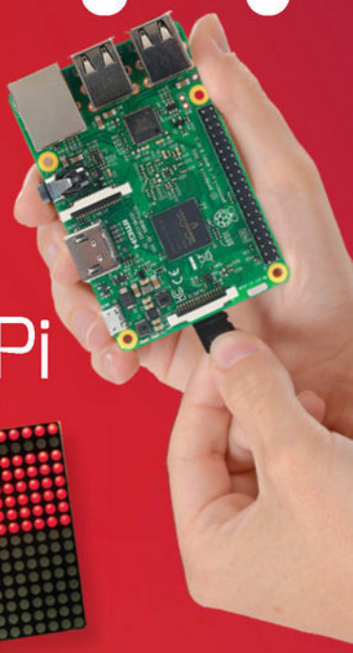
NEW

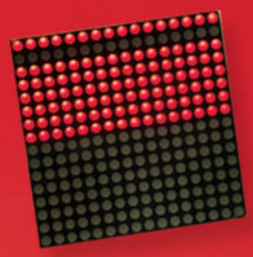
100% UNOFFICIAL

Practical Raspberry Pi Projects

Get hands-on with your Raspberry Pi

Drive a remote control car

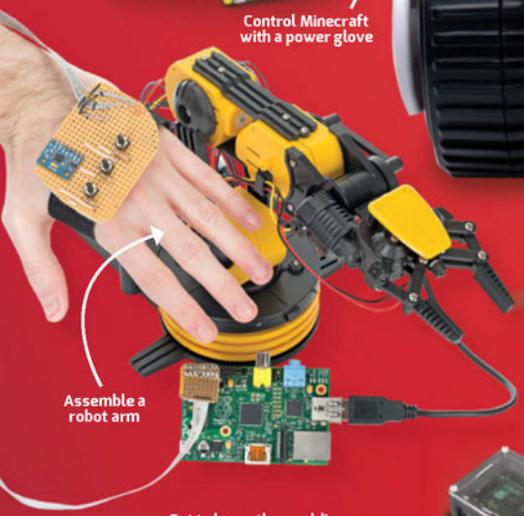




Control Minecraft with a power glove

Build a Bigtrak

Over 60 projects



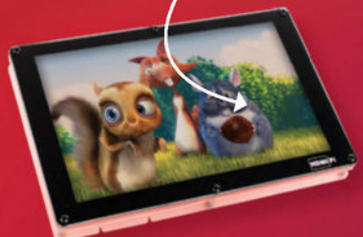
Assemble a robot arm

Fly your own quadcopter

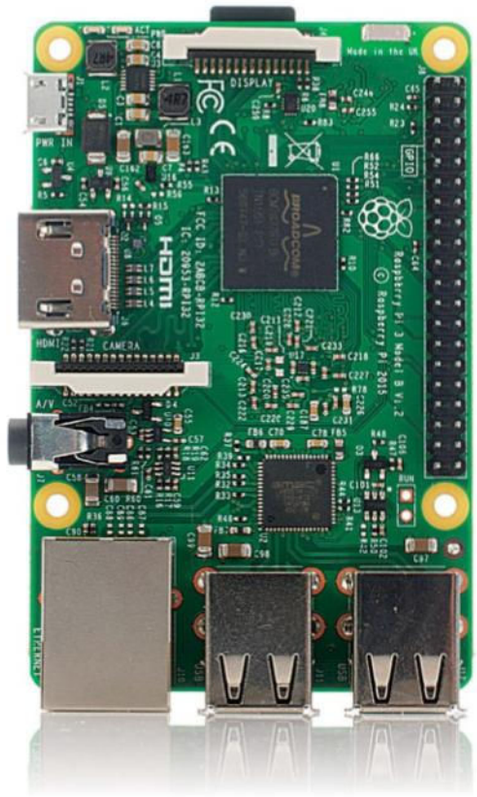
Program a RasPi-powered synth

Get to know the world's smallest computer

Attach screens and other accessories



- SUPERCHARGE YOUR PI
- BUILD INCREDIBLE GADGETS
- PROGRAM AMAZING SOFTWARE



Welcome to

Practical Raspberry Pi Projects

For a device that can fit in the palm of your hand, the Raspberry Pi has had a pretty colossal impact since its launch in 2012. In just a few short years it's changed the way computer science is taught in schools, it's been used in some amazing projects at Raspberry Jam events across the world, and it's inspired a new generation of coders to create and craft new gadgets. No matter your age or experience level, there's a Pi project for you, and in Practical Raspberry Pi Projects we're giving you everything you need to fire up your imagination and unleash your creativity. From hardware-based projects like building a Raspberry Pi-controlled car, through software projects like coding a simple synth, all the way to advanced electronics projects that will see you transforming your Pi into a retro NES, alarm clock robot or quadcopter, we've got plenty here to keep you busy. All you need is your favourite $35 computer and a passion for making things!

Practical Raspberry Pi Projects

Imagine Publishing Ltd
Richmond House
33 Richmond Hill
Bournemouth
Dorset BH2 6EZ
☎ +44 (0) 1202 586200
Website: www.imagine-publishing.co.uk
Twitter: @Books_Imagine
Facebook: www.facebook.com/ImagineBookazines

Publishing Director
Aaron Asadi

Head of Design
Ross Andrews

Editor In Chief
Jon White

Production Editor
Hannah Westlake

Senior Art Editor
Greg Whitaker

Assistant Designer
Steve Dacombe

Photographer
James Sheppard

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

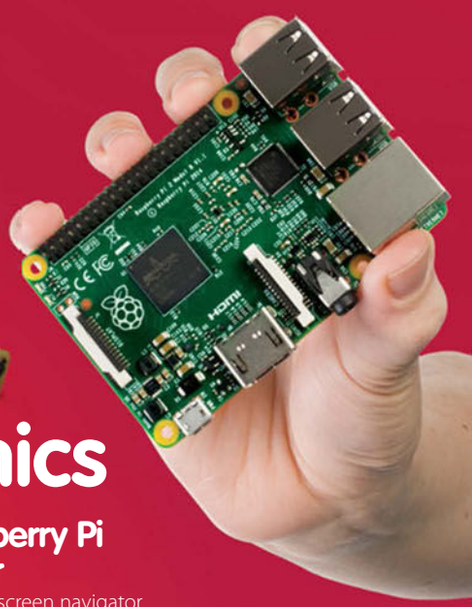
Distributed in the UK, Eire & the Rest of the World by
Marketforce, 5 Churchill Place, Canary Wharf, London, E14 5HU
Tel 0203 787 9060 www.marketforce.co.uk

Distributed in Australia by
Gordon & Gotch Australia Pty Ltd, 26 Rodborough Road, Frenchs Forest, NSW, 2086 Australia
Tel +61 2 9972 8800, www.gordongotch.com.au

Part of the LinuxUser & Developer bookazine series

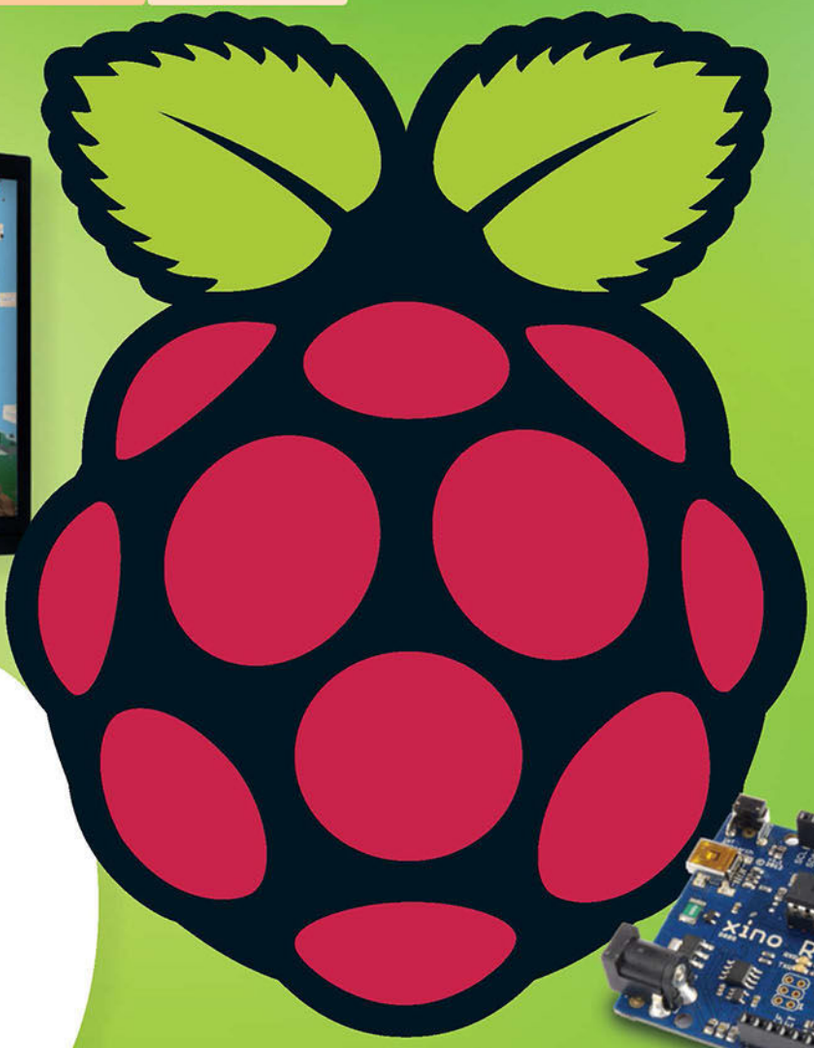
IMAGINE PUBLISHING

Contents

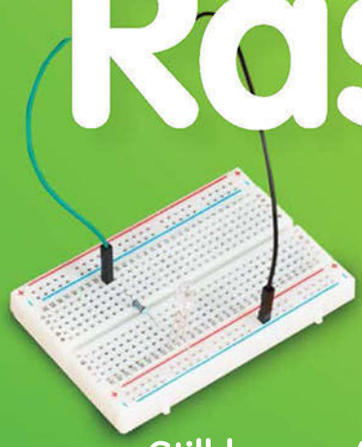


Software

Electronics



10 practical Raspberry Pi projects



Still haven't done anything with your Raspberry Pi? Follow along with our expert advice and kick-start your own amazing Raspberry Pi projects

10 PRACTICAL RASPBERRY PI PROJECTS

Make a stop motion animation

Build a RasPi web server

Create a voice synthesiser

Code your own Twitter bot

From our time covering this incredible credit card-sized computer, it's become clear there are two types of Raspberry Pi owners: those that use theirs and those that don't. Whether it's fear of the unknown, a lack of time or inspiration, when we ask people what they do with their Pi we'll often hear that it's still in the box. If that's you, then you're in the right place. In this feature we've handcrafted ten Raspberry Pi projects practically anyone can enjoy.

These aren't just a random selection of side-projects, though. These are practical ideas designed to help kick-start bigger and better things. Knowledge gained from one project can also be applied to another to create something completely new. For example, you could combine our Twitter and three-colour lamp tutorials to create a desk lamp that changes colour as your Twitter account is retweeted. You could go on to make Pong in Minecraft-Pi or use a button attached to Scratch to take photos with your Raspberry Pi camera module. The list goes on.

All these projects are open source, so you're encouraged to tweak and develop them into something entirely new. If you share your tweaks and changes with the community, you're sure to start benefitting from doing things the open source way...

Make music with the Raspberry Pi

Program your own melodies using Sonic Pi and create musical cues or robot beeps

What you'll need

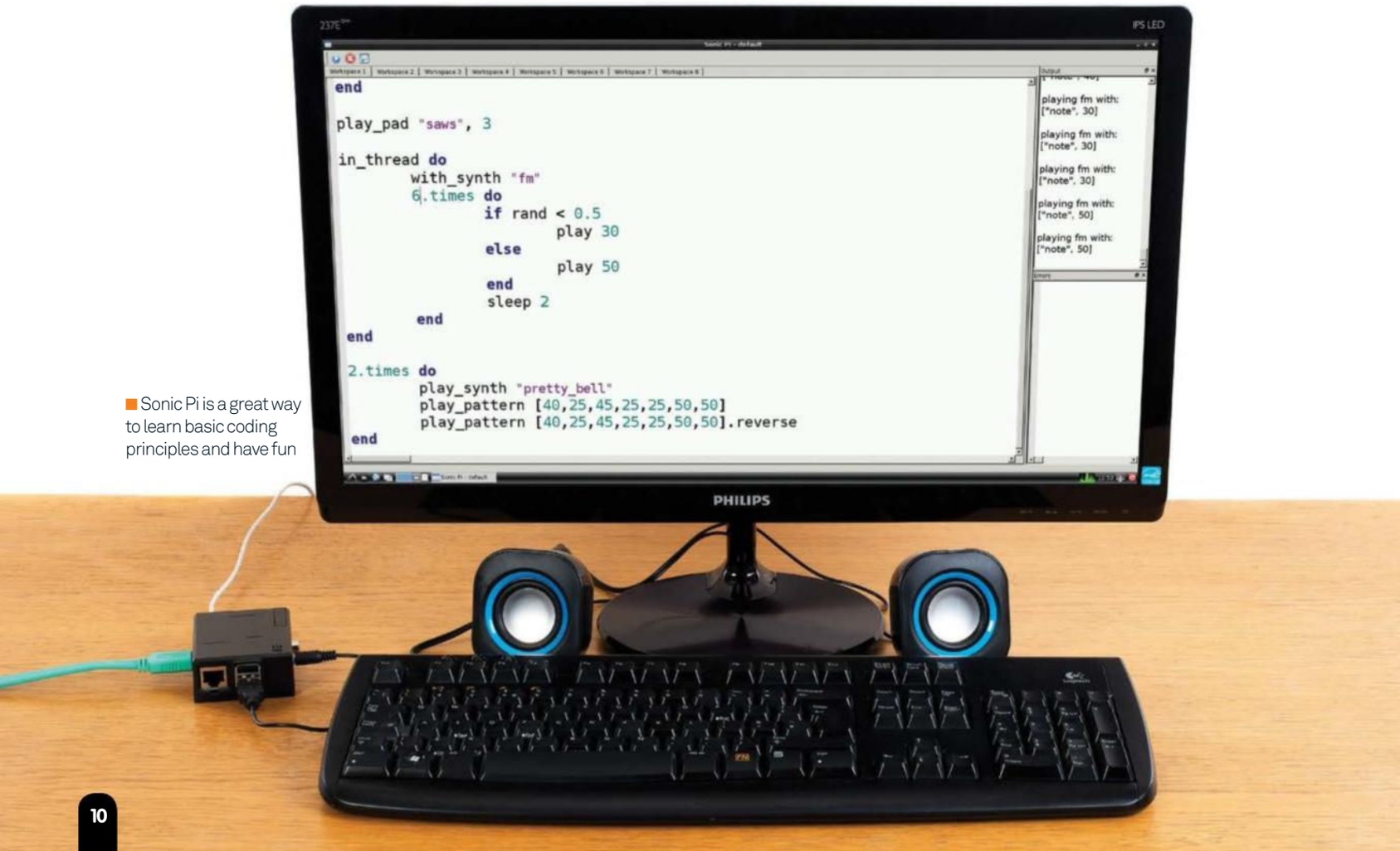
- Portable speakers
- Sonic Pi
  www.cl.cam.ac.uk/projects/raspberrypi/sonicpi/teaching.html

One of the major features of Scratch is its ability to teach the fundamentals of coding to kids and people with no computing background. For kids, its especially appealing due to the way it allows them to create videogames to interact with as part of their learning. In this kind of vein then, Sonic Pi teaches people to code using music. With a simple language that utilises basic logic steps but in a more advanced way than Scratch, it can either be used as a next step for avid coders, or as a way to create music for an Internet of Things or a robot.

01 Getting Sonic Pi

If you've installed the latest version of Raspbian, Sonic Pi will be included by default. If you're still using a slightly older version, then you'll need to install it via the repos. Do this with:

```
$ sudo apt-get install sonic-pi
```



Sonic Pi is a great way to learn basic coding principles and have fun

We can start making more complex melodies by using more of Sonic Pi's functions

02 Starting with Sonic Pi

Sonic Pi is located in the Education category in the menus. Open it up and you'll be presented with something that looks like an IDE. The pane on the left allows you to enter the code for your project, with proper syntax highlighting for its own style of language. When running, an info pane details exactly what's being played via Sonic Pi – and any errors are listed in their own pane as well, for reference.

03 Your first note

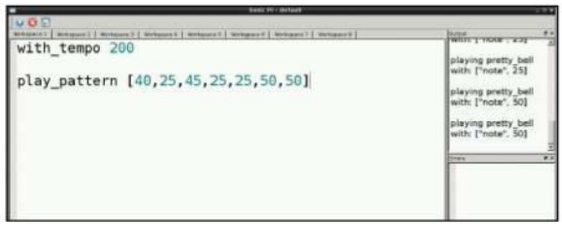
Our first thing to try on Sonic Pi is simply being able to play a note. Sonic Pi has a few defaults preset, so we can get started with:

```
play 50
```

Press the Play button and the output window will show you what's being played. The **pretty_bell** sound is the default tone for Sonic Pi's output, and **50** determines the pitch and tone of the sound.

Full code listing

```
with_tempo 200

play_pattern [40,25,45,25,25,50,50]

2.times do
    with_synth "beep"
    play_pattern [40,25,45,25,25,50,50]
    play_pattern [40,25,45,25,25,50,50].reverse
end

play_pad "saws", 3

in_thread do
    with_synth "fm"
    6.times do
          if rand < 0.5
                play 30
          else
                play 50
          end
          sleep 2
    end
end

2.times do
    play_synth "pretty_bell"
    play_pattern [40,25,45,25,25,50,50]
    play_pattern [40,25,45,25,25,50,50].reverse
end
```



04 Set the beat

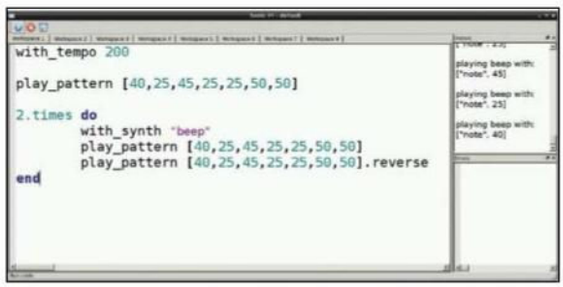
For any piece of music, you'll want to set the tempo. We can start by putting:

```
with_tempo 200
```

...at the start of our code. We can test it out by creating a string of midi notes using play_pattern:

```
play_pattern [40,25,45,25,25,50,50]
```

This will play pretty_bell notes at these tones at the tempo we've set. You can create longer and shorter strings, and also change the way they play.



05 Advance your melody

We can start making more complex melodies by using more of Sonic Pi's functions. You can change the note type by using **with_synth**, reverse a pattern, and even create a finite loop with the **x.times** function; **do** and **end** signify the start and end of the loop. Everything is played in sequence before repeating, much like an if or while loop in normal code.

06 Playing a concert

Using the **in_thread** function, we can create another thread for the Sonic Pi instance and have several lines of musical code play at once instead of in sequence. We've made it create a series of notes in a random sequence, and have them play alongside extra notes created by the position and velocity of the mouse using the **play_pad** function.

You'll learn...

1. How to code

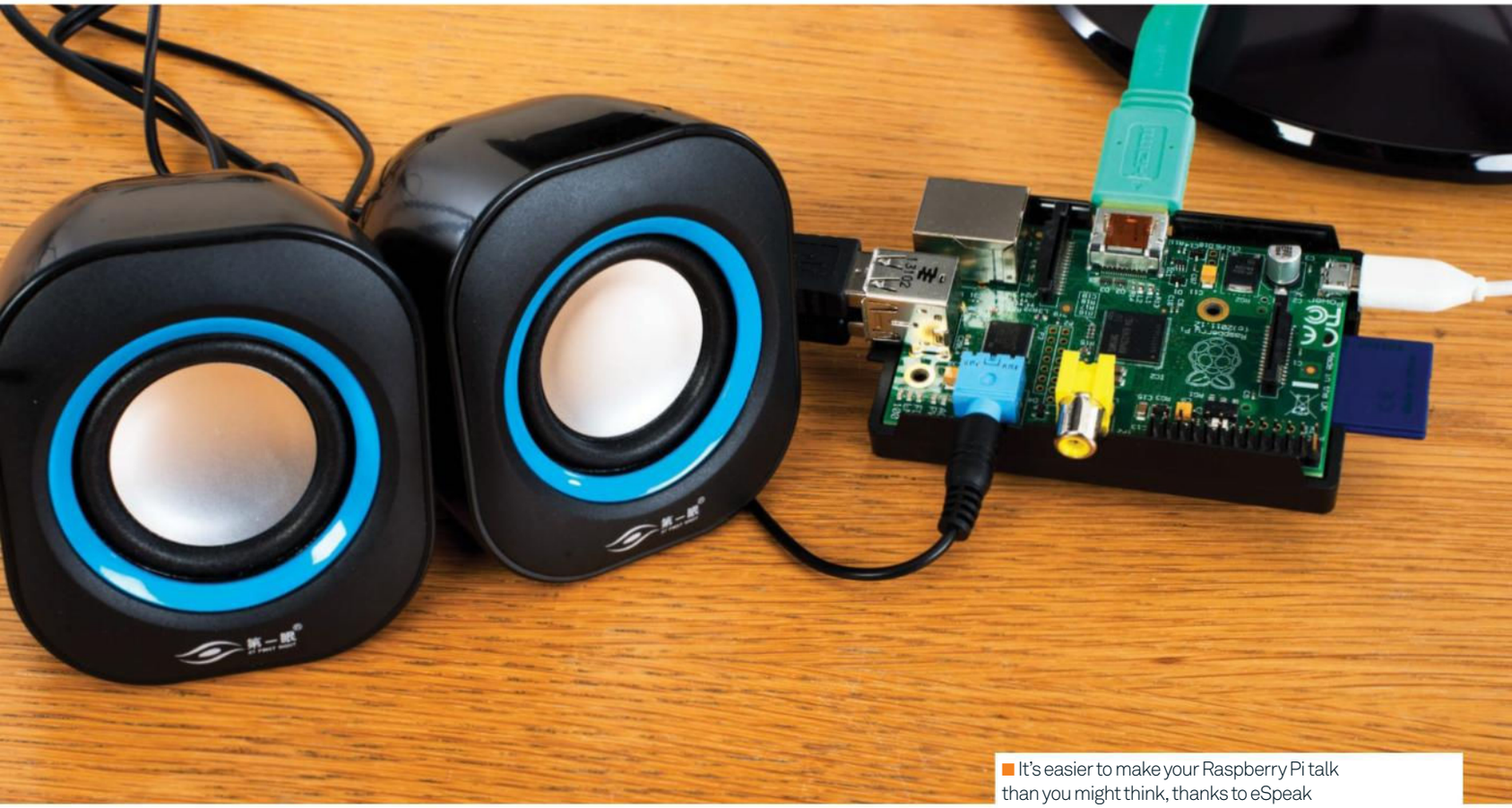
The coding style of Sonic Pi uses concepts from standard programming languages – if statements, loops, threads etc. Whereas Scratch teaches this logic, Sonic Pi teaches their structure.

2. Robotic voice

Employ Sonic Pi to create context-sensitive chirps, chirps and beeps and use it to give a familiar voice while it tootles around.

3. MIDI

The Musical Instrument Digital Interface is a standard for digital music, and the numbers and tones used in Sonic Pi make use of this.



It's easier to make your Raspberry Pi talk than you might think, thanks to eSpeak

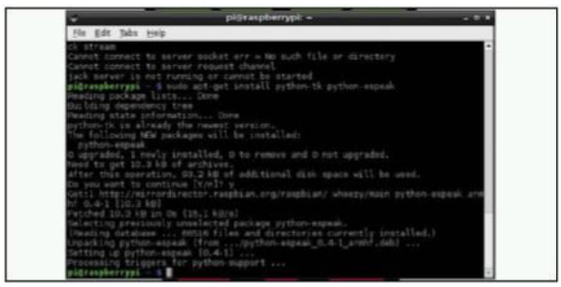
Raspberry Pi voice synthesizer

Add the power of speech to your Raspberry Pi projects with the versatile eSpeak Python library

What you'll need

- Portable USB speakers
- python-espeak module
- eSpeak
- Raspbian (latest image)

We've shown in previous issues how the Raspberry Pi can be used to power robots, and as a tiny computer it can also be the centre of an Internet of Things in your house or office. For these reasons and more, using the Raspberry Pi for text-to-voice commands could be just what you're looking for. Due to the Debian base of Raspbian, the powerful eSpeak library is easily available for anyone looking to make use of it. There's also a module that allows you to use eSpeak in Python, going beyond the standard command-line prompts so you can perform automation tasks.

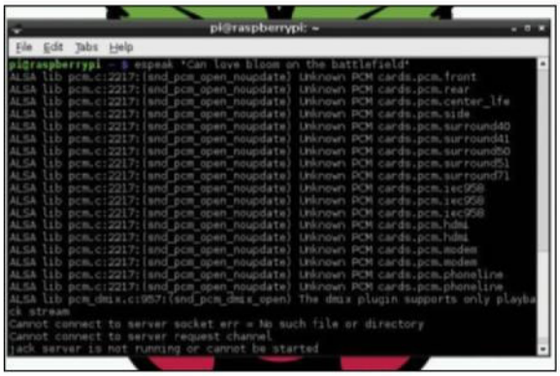


01 Everything you'll need

We'll install everything we plan to use in this tutorial at once. This includes the eSpeak library and the Python modules we need to show it off. Open the terminal and install with:

```
$ sudo apt-get install espeak python-espeak python-tk
```

You can change the way eSpeak will read text with a number of different options



02 Pi's first words

The eSpeak library is pretty simple to use – to get it to just say something, type in the terminal:

```
$ espeak "[message]"
```

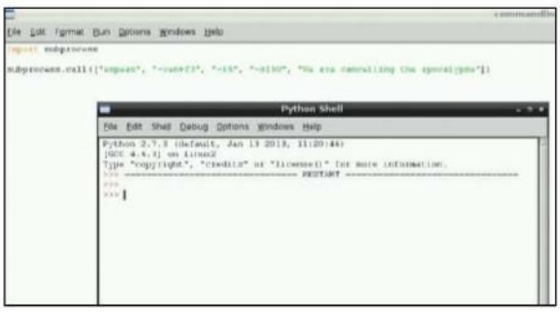
This will use the library's defaults to read whatever is written in the message, with decent clarity.

03 Say some more

You can change the way eSpeak will read text with a number of different options, such as gender, read speed and even the way it pronounces syllables. For example, writing the command like so:

```
$ espeak -ven+f3 -k5 -s150 "[message]"
```

...will turn the voice female, emphasise capital letters and make the reading slower.



04 Taking command with Python

The most basic way to use eSpeak in Python is to use **subprocess** to directly call a command-line function. Import subprocess in a Python script, then use:

```
subprocess.call(["espeak", "[options 1]", "[option 2]",..."[option n]", "[message]"])
```

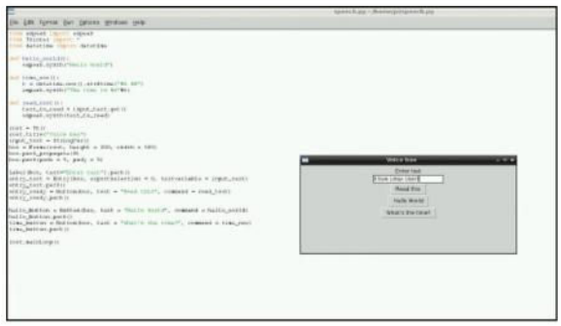
The message can be taken from a variable.

05 The native tongue

The Python eSpeak module is quite simple to use to just convert some text to speech. Try this sample code:

```
from espeak import espeak
espeak.synth("[message]")
```

You can then incorporate this into Python, like you would any other module, for automation.



06 A voice synthesiser

Using the code listing, we're creating a simple interface with Tkinter with some predetermined voice buttons and a custom entry method. We're showing how the eSpeak module can be manipulated to change its output. This can be used for reading tweets or automated messages. Have fun!

Full code listing

Get the code: bit.ly/14XbLOC

Import the necessary eSspeak and GUI modules, as well as the module to find out the time

```
from espeak import espeak
from Tkinter import *
from datetime import datetime
```

Define the different functions that the interface will use, including a simple fixed message, telling the time, and a custom message

```
def hello_world():
    espeak.synth("Hello World")

def time_now():
    t = datetime.now().strftime("%k %M")
    espeak.synth("The time is %s"%t)

def read_text():
    text_to_read = input_text.get()
    espeak.synth(text_to_read)
```

Create the basic window with Tkinter for your interface, as well as creating the variable for text entry

```
root = Tk()
root.title("Voice box")
input_text = StringVar()
box = Frame(root, height = 200, width = 500)
box.pack_propagate(0)
box.pack(padx = 5, pady = 5)
```

The text entry appends to the variable we created, and each button calls a specific function that we defined above in the code

```
Label(box, text="Enter text").pack()
entry_text = Entry(box, exportselection = 0, ↵
textvariable = input_text)
entry_text.pack()
entry_ready = Button(box, text = "Read this", ↵
command = read_text)
entry_ready.pack()

hello_button = Button(box, text = "Hello World", ↵
command = hello_world)
hello_button.pack()
time_button = Button(box, text = "What's the ↵
time?", command = time_now)
time_button.pack()

root.mainloop()
```

Program Minecraft-Pi

Learn to program while playing one of the greatest games ever made!

What you'll need

- Raspbian (latest release)
- Minecraft-Pi tarball
- Keyboard & mouse
- Internet connection

***Minecraft* is probably the biggest game on the planet right now.** It's available on just about any format you can imagine, from PCs to gaming consoles to mobile phones. It should probably come as no surprise that it's also available on the Raspberry Pi. While at first glance Minecraft-Pi is a simplified version of the Pocket Edition (designed for tablets and smartphones), the Raspberry Pi edition is very special, in that it's the only version of *Minecraft* to gives users access to its API (application programming interface).

In this project we're going to show you how to set up Minecraft-Pi and configure it so you can interact with *Minecraft* in a way you've never done before. This small project is just the tip of the iceberg...

01 Requirements

Minecraft-Pi requires you to be running Raspbian on your Raspberry Pi, so if you're not already running that, take a trip to **raspberrypi.org** and get it setup. It also requires you have X Window loaded too. Assuming you're at the command prompt, you just need to type `startx` to reach the desktop.

■ Unlike all other versions of *Minecraft*, the Pi version encourages you to hack it

02 Installation

Make sure you're already in your home folder and download the **Minecraft-Pi** package with the following commands in a terminal window:

```
cd ~
wget  https://s3.amazonaws.com/assets.minecraft.net/pi/minecraft-pi-0.1.1.tar.gz
```

To use it we need to decompress it. Copy the following into the terminal window:

```
tar -zxvf minecraft-pi-0.1.1.tar.gz
```

Now you can move into the newly decompressed Minecraft-Pi directory and try running the game for the first time:

```
cd mcpi
./minecraft-pi
```

03 Playing Minecraft-Pi

Have a look around the game. If you're not familiar with *Minecraft*, you control movement with the mouse and the WASD keys. Numbers 1-8 select items in your quickbar, the space bar makes you jump and Shift makes you walk slowly (so you don't fall off edges). 'E' will open your inventory and double-tapping the space bar will also toggle your ability to fly.

04 Configuring the Python API

To take control of *Minecraft* with the Python API, you next need to copy the Python API folder from within the **/mcpi** folder to a new location. In the terminal, type the following:

```
cp -r ~/mcpi/api/python/mcpi ↵
~/ minecraft
```

In this folder, we want to create a 'boilerplate' Python document that connects the API to the game. Write the following into the terminal:

```
cd ~/minecraft
nano minecraft.py
```

With nano open, copy the following and then save and exit with **Ctrl+X**, pressing **Y** (for yes), then **Enter** to return to the command prompt:

```
from mcpi.minecraft import ↵
Minecraft
from mcpi import block
from mcpi.vec3 import Vec3
mc = Minecraft.create()
mc.postToChat("Minecraft API ↵
Connected")
```

05 Testing your Python script

The short script you created contains everything you need to get started with hacking Minecraft-Pi in the Python language. For it to work, you need to have the game already running (and be playing). To grab control of the mouse while in-game, you can press **Tab**. Open a fresh terminal window, navigate into your **minecraft** folder and start the script with the following commands:

```
cd ~/minecraft
python minecraft.py
```

You'll see a message appear on screen to let you know the API connected properly. Now we know it works, let's get coding!

Full code listing

Get the code: bit.ly/1fo7MQ3

```
# !/usr/bin/env python

from mcpi.minecraft import Minecraft
from mcpi import block
from mcpi.vec3 import Vec3
from time import sleep, time
import random, math

mc = Minecraft.create() # make a connection to the game
playerPos = mc.player.getPos()

# function to round players float position to integer position
def roundVec3(vec3):
    return Vec3(int(vec3.x), int(vec3.y), int(vec3.z))

# function to quickly calc distance between points
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))

def random_block(): # create a block in a random position
    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

def main(): # the main loop of hide & seek
    global lastPlayerPos, playerPos
    seeking = True
    lastPlayerPos = playerPos

   randomBlockPos = random_block()
    mc.setBlock(randomBlockPos, block.DIAMOND_BLOCK)
    mc.postToChat("A diamond has been hidden somewhere nearby!")

    lastDistanceFromBlock = distanceBetweenPoints(randomBlockPos, lastPlayerPos)
    timeStarted = time()
    while seeking:
        # Get players position
        playerPos = mc.player.getPos()
        # Has the player moved
        if lastPlayerPos != playerPos:
            distanceFromBlock = distanceBetweenPoints(randomBlockPos, playerPos)

            if distanceFromBlock < 2:
                #found it!
                seeking = False
            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)

    timeTaken = time() - timeStarted
    mc.postToChat("Well done - " + str(int(timeTaken)) + " seconds to find the diamond")

if __name__ == "__main__":
    main()
```

You'll learn...

Functional, & fun coding

There's nothing too taxing about our code. We've created a couple of simple functions (starting with **def**) and used **if**, **else** and **while** to create the logic.

06 Hide & Seek

As you can see from the code above, we've created a game of Hide & Seek adapted from Martin O'Hanlon's original creation (which you can find on **www.stuffaboutcode.com**). When you launch the script, you'll be challenged to find a hidden diamond in the fastest time possible. We've used it to demonstrate some of the more accessible methods available in the API. But there's much more to it than this demonstrates. Stay tuned – we'll be back with more related guides in future issues.

Scratch can be used to do Internet Of Things projects with a few tweaks

Get interactive with Scratch

Experiment with physical computing by using Scratch to interact with buttons and lights on your Pi

What you'll need

- Breadboard
- LEDs
- Buttons
- Resistors
- Jumper wires
- ScratchGPIO3

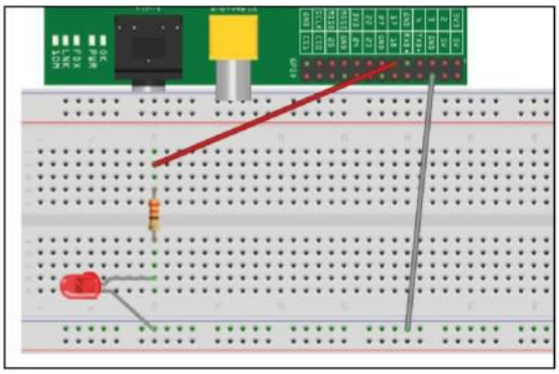
Scratch is a very simple visual programming language, commonly used to teach basic programming concepts to learners of any age. In this project we'll learn how to light up an LED when a button is pressed in Scratch, and then change a character's colour when a physical button is pressed. With these techniques you can make all manner of fun and engaging projects, from musical keyboards to controllers for your Scratch games and animations.

01 Installing the required software

Log into the Raspbian system with the username **Pi** and the password **raspberry**. Start the LXDE desktop environment using the command **startx**. Then open LXTerminal and type the following commands:

```
wget http://liamfraser.co.uk/lud/install_scratchgpio3.sh
chmod +x install_scratchgpio3.sh
sudo bash install_scratchgpio3.sh
```

This will create a special version of Scratch on your desktop called ScratchGPIO3. This is a normal version of Scratch with a Python script that handles communications between Scratch and the GPIO. ScratchGPIO was created by simplesi (**cymplecy.wordpress.com**).



02 Connecting the breadboard

Power off your Pi and disconnect the power cable. Get your breadboard, an LED, a 330-ohm resistor and two GPIO cables ready. You'll want to connect the 3.3V pin (top-right pin, closest to the SD card) to one end of the 330-ohm resistor, and then connect the positive terminal of the LED (the longer leg is positive) to the other end. The resistor is used to limit the amount of current that can flow to the LED.

Then put the negative terminal of the LED into the negative rail of the breadboard. Connect one of the GROUND pins (for example, the third pin from the right on the bottom row of pins) to the negative lane. Now connect the power to your Pi. The LED should light up. If it doesn't, then it's likely that you've got it the wrong way round, so disconnect the power, swap the legs around and then try again.

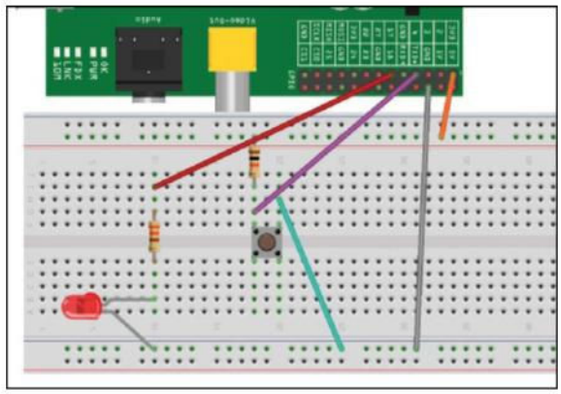
03 Switching the LED on and off

At the moment, the LED is connected to a pin that constantly provides 3.3V. This isn't very useful if we want to be able to turn it on and off, so let's connect it to GPIO 17, which we can turn on and off. GPIO 17 is the sixth pin from the right, on the top row of pins. Power the Pi back on. We can turn the LED on by exporting the GPIO pin, setting it to an output pin and then setting its value to 1. Setting the value to 0 turns the LED back off:

```
echo 17 > /sys/class/gpio/export
echo out > /sys/class/gpio/gpio17/direction
echo 1 > /sys/class/gpio/gpio17/value
echo 0 > /sys/class/gpio/gpio17/value
```

04 Controlling the LED from Scratch

Start the LXDE desktop environment and open ScratchGPIO3. Go to the control section and create a simple script that broadcasts **pin11on** when Sprite1 is clicked. Then click the sprite. The LED should light up. Then add to the script to wait 1 second and then broadcast **pin11off**. If you click the sprite again, the LED will come on for a second and then go off. ScratchGPIO3 uses pin numbers rather than GPIO numbers to identify pins. The top-right pin (the 3.3V we first connected our LED to) is pin number 1, the pin underneath that is pin number 2, and so on.



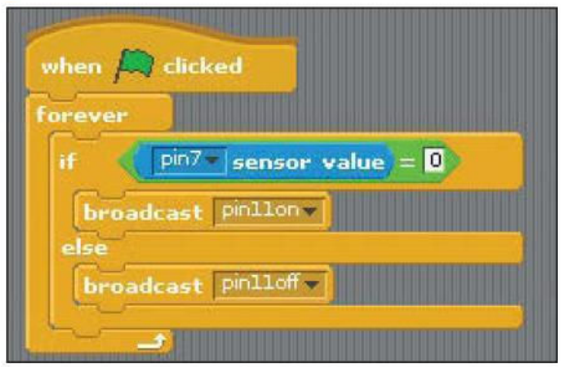
05 Wiring up our push button

Power off the Pi again. This circuit is a little bit more complicated than the LED one we created previously. The first thing we need to do is connect 3.3V (the top-right pin we used to test our LED) to the positive rail of the breadboard. Then we need to connect a 10Kohm resistor to the positive rail, and the other end to an empty track on the breadboard. Then on the same track, add a wire that has one end connected to GPIO 4. This is two pins to the right of GPIO 17. Then, on the same track again, connect one pin of the push button. Finally, connect the other pin of the push button to ground by adding a wire that is connected to the same negative rails that ground is connected to.

When the button is not pressed, GPIO 4 will be receiving 3.3V. However, when the button is pressed, the circuit to ground will be completed and GPIO 4 will be receiving 0V (and have a value of 0), because there is much less resistance on the path to ground.

We can see this in action by watching the pin's value and then pressing the button to make it change:

```
echo 4 > /sys/class/gpio/export
echo in > /sys/class/gpio/gpio4/direction
watch -n 0.5 cat /sys/class/gpio/gpio4/value
```



06 Let there be light!

Boot up the Pi and start ScratchGPIO3 as before. Go to the control section and add **when green flag clicked**, then attach a **forever** loop, and inside that an **if else** statement. Go to the operators section and add an **if [] = []** operator to the if statement. Then go to the sensing section and add a value sensor to the left side of the equality statement, and set the value to pin7. On the right side of the equality statement, enter 0. Broadcast **pin11on** if the sensor value is 0, and broadcast **pin11off** otherwise. Click the green flag. If you push the button, the LED will light up!

You'll learn...

1. Simple circuits

While these are very simple circuits, you'll get a great feel of how the Raspberry Pi interfaces with basic prototyping kit. If you need to buy the bits and pieces, we recommend you check out: **shop.pimoroni.com**

2. Coding principles

If you're new to programming, Scratch is the perfect place to learn the same programming principles employed by all programming languages.

3. Physical computing

There's nothing more magical than taking code from your computer screen and turning it into a real-life effect. Your first project might just turn a light on and off, but with that skill banked, the sky is the limit.

Build a Raspberry Pi web server

Use Google Coder to turn your Raspberry Pi into a tiny, low-powered web server and web host

What you'll need

- Internet connectivity
- Web browser
- Google Coder
  googlecreativelab.github.io/coder/raspberrypi/sonicpi/teaching.html

We're teaching you how to code in many different ways on the Raspberry Pi, so it only seems fitting that we look at the web too.

There's a new way to use the web on the Raspbian-based image that turns your Pi into a web server and web development kit. Accessible easily over a local network and with support for jQuery out of the box, it's an easy and great way to further your web development skills.

01 Get Google Coder

Head to the Google Coder website, and download the compressed version of the image. Unpack it wherever you wish, and install it using **dd**, like any other Raspberry Pi image:

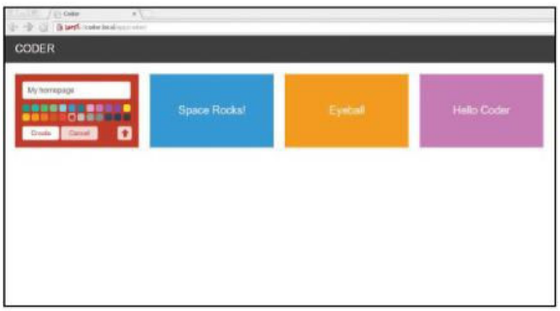
```
$ dd if=[path to]/raspi.img of=/dev/[path to SD card] bs=1M
```

02 Plug in your Pi

For this tutorial, you'll only need to connect a network cable into the Pi. Pop in your newly written SD card, plug in the power and wait a few moments. If you've got a display plugged in anyway, you'll notice a Raspbian startup sequence leading to the command-line login screen.

03 Connect to Coder

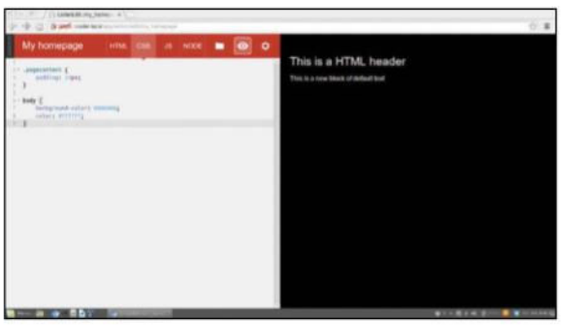
Open up the browser on your main system, and go to **http://coder.local**. You may have to manually accept the licence. It will ask you to set up your password, and then you'll be in and ready to code.



04 Language of the web

Now it's time to create your own app or website. Click on the '+' box next to the examples, give your app a name and then click Create. You'll be taken to the HTML section of the app. Change the Hello World lines to:

```
<h1>This is a HTML header</h1>
<p>This is a new block of default text</p>
```



05 Styled to impress

Click on the CSS tab. This changes the look and style of the webpage without having to make the changes each time in the main code. You can change the background colour and font with:

```
body {
    background-color: #000000;
    color: #ffffff;
}
```

Full code listing

HTML

Some simple HTML code that can point us to some important websites. The h2 tag is used to display the time thanks to Java

```
<h1>Welcome to the internet...</h1>
<h2></h2>
<p><a href="http://www.linuxuser.co.uk">Linux User & Developer</p>
<p><a href="http://www.reddit.com/">Reddit</p>
<p><a href="http://www.linuxfoundation.org/">The Linux Foundation</p>
<p><a href="http://www.fsf.org/">Free Software Foundation</p>
```

Java

We're calling the current time using jQuery in the JS tab so that we can ultimately display it on the webpage

We're going to display the time as a 12-hour clock in the first **if** statement, and use AM and PM to differentiate the time

We make the minutes readable by adding a 0 if it's below 10, then concatenate all the variables and assign to the tag h2

```
var d = new Date;
    var hours = d.getHours();
    var mins = d.getMinutes();
    if (hours > 12) {
        var hour = (hours - 12);
        var ampm = "PM";
    }
    else {
        var hour = hours;
        var ampm = "AM";
    }
    if (hours == 12) {
        var ampm = "PM";
    }
    if (mins > 9){
        var min = mins;
    }
    else {
        var min = "0" + mins;
    }
    var time = "The time is " + hour + ":" + min + " " + ampm;
    $("h2").html(time);
```

Get the code: bit.ly/1Vz5cYv

06 Querying your Java

The third tab allows you to edit the jQuery, making the site more interactive. We can make it create a message on click with:

```
$(document).click(function() {
    alert('You clicked the website!');
    }
);
```

Code your own Twitter bot

Create your very own Twitter bot that can retweet chunks of wisdom from Linux User & Developer

What you'll need

- Internet connectivity
- Latest version of Raspbian
  www.raspberrypi.org/downloads

Twitter is a useful way of sharing information with the world and it's our favourite method of giving our views quickly and conveniently. Many millions of people use the microblogging platform from their computers, mobile devices and possibly even have it on their televisions.

You don't need to keep pressing that retweet button, though. With a sprinkling of Python, you can have your Raspberry Pi do it for you. Here's how to create your own Twitter bot...

01 Installing the required software

Log into the Raspbian system with the username **Pi** and the password **raspberry**. Get the latest package lists using the command `sudo apt-get update`. Then install the Python Package installer using `sudo apt-get install python-pip`. Once you've done that, run `sudo pip install twython` to install the Twitter library we'll be using.

02 Registering with Twitter

We need to authenticate with Twitter using OAuth. Before this, you need to go to **https://dev.twitter.com/apps** and sign in with the account you'd like your Pi to tweet from. Click the 'Create a new application' button. We called our application

Save your mouse button by creating an automated retweeter

'LUD Pi Bot', gave it the same description and set the Website to **http://www.linuxuser.co.uk/**. The Callback URL is unnecessary. You'll have to agree with the developer rules and then click the **Create** button.

03 Creating an access token

Go to the Settings tab and change the Access type from 'Read only' to 'Read and Write'. Then click the 'Update this Twitter application's settings' button. The next step is to create an access token. To do that, click the 'Create my access token' button. If you refresh the details page, you should have a consumer key, a consumer secret and access token, plus an access token secret. This is everything we need to authenticate with Twitter.

04 Authenticating with Twitter

We're going to create our bot as a class, where we authenticate with Twitter in the constructor. We take the tokens from the previous steps as parameters and use them to create an instance of the Twython API. We also have a variable, **last_ran**, which is set to the current time. This is used to check if there are new tweets later on.

> If the tweet's time is newer than the time the function was last called, we retweet it

05 Retweeting a user

The first thing we need to do is get a list of the user's latest tweets. We then loop through each tweet and get its creation time as a string, which is then converted to a **datetime** object. We then check that the tweet's time is newer than the time the function was last called – and if so, retweet the tweet.

06 The main section

The main section is straightforward. We create an instance of the **bot** class using our tokens, and then go into an infinite loop. In this loop, we check for any new retweets from the users we are monitoring (we could run the retweet task with different users), then update the time everything was last run, and sleep for five minutes.

Get the code: bit.ly/1RTgNSH

Full code listing

```
#!/usr/bin/env python2

# A Twitter Bot for the Raspberry Pi that retweets any content from
# @LinuxUserMag. Written by Liam Fraser for a Linux User & Developer article.

import sys
import time
from datetime import datetime
from twython import Twython

class bot:
    def __init__(self, c_key, c_secret, a_token, a_token_secret):
        # Create a Twython API instance
        self.api = Twython(c_key, c_secret, a_token, a_token_secret)

        # Make sure we are authenticated correctly
        try:
            self.api.verify_credentials()
        except:
            sys.exit("Authentication Failed")

        self.last_ran = datetime.now()

    @staticmethod
    def timestr_to_datetime(timestr):
        # Convert a string like Sat Nov 09 09:29:55 +0000
        # 2013 to a datetime object. Get rid of the timezone
        # and make the year the current one
        timestr = "{0} {1}".format(timestr[:19], datetime.now().year)

        # We now have Sat Nov 09 09:29:55 2013
        return datetime.strptime(timestr, '%a %b %d %H:%M:%S %Y')

    def retweet_task(self, screen_name):
        # Retweets any tweets we've not seen
        # from a user
        print "Checking for new tweets from @{0}".format(screen_name)

        # Get a list of the users latest tweets
        timeline = self.api.get_user_timeline(screen_name = screen_name)

        # Loop through each tweet and check if it was
        # posted since we were last called
        for t in timeline:
            tweet_time = bot.timestr_to_datetime(t['created_at'])
            if tweet_time > self.last_ran:
                print "Retweeting {0}".format(t['id'])
                self.api.retweet(id = t['id'])

if __name__ == "__main__":
    # The consumer keys can be found on your application's
    # Details page located at https://dev.twitter.com/
    # apps(under "OAuth settings")
    c_key=""
    c_secret=""

    # The access tokens can be found on your applications's
    # Details page located at https://dev.twitter.com/apps
    # (located under "Your access token")
    a_token=""
    a_token_secret=""

    # Create an instance of the bot class
    twitter = bot(c_key, c_secret, a_token, a_token_secret)

    # Retweet anything new by @LinuxUserMag every 5 minutes
    while True:
        # Update the time after each retweet_task so we're
        # only retweeting new stuff
        twitter.retweet_task("LinuxUserMag")
        twitter.last_ran = datetime.now()
        time.sleep(5 * 60)
```

The Arduino is better at dealing with things like servos and analog input

Program your Arduino with Raspberry Pi

Enjoy all the features and benefits of the Arduino microcontroller on your Raspberry Pi projects

What you'll need

- Arduino Uno
- Internet connectivity
- Nanpy
  https://github.com/nanpy

You might be wondering why you might want to attach an Arduino to your Raspberry Pi. While there are lots of reasons, probably the most poignant is the extra six PWM-capable pins and another six analogue pins that a standard Arduino Uno offers.

You see, while the Raspberry Pi has an excellent array of pins and capabilities, it can't do analogue and it can't do real-time processing out of the box. With an Arduino, additions like servos, potentiometers and a whole array of analog sensors are trivially easy to trigger and control.

The best part is that you don't even have to program in Arduino's quasi-C++ language. All you need is a standard USB connection between your Raspberry Pi and Arduino and a small Python package called Nanpy. Here's how it's done...

01 Grab an Arduino

Before you can do anything, you need an Arduino. We recommend the Uno, since it's the default choice with the best balance of features, convenience and affordability. Since you'll want to put it to use straight away, we recommend investing in a 'starter kit' that includes LEDs, servos and all that fun stuff.

02 Satisfying dependencies

We're assuming you're using Raspbian (recommended), so open your terminal because we need to get

> The best part is that you don't even have to program in Arduino's quasi-C++ language

setuptools so we can install Nanpy. At the terminal, type:

```
wget https://bitbucket.org/pypa/setuptools/raw/bootstrap/ez_setup.py
python ez_setup.py user
```

Once this is complete, you'll be able to use the **easy_install** command to install **pyserial**...

03 Final preparations

Since the communication between the Arduino and Raspberry Pi will happen over the USB serial connection, we need to get the Python-serial library. At the terminal, type:

```
easy_install pyserial
```

We also need to install the Arduino software so the Pi knows how to deal with the device when it's plugged in. In the terminal, type:

```
sudo apt-get update
sudo apt-get install arduino
```

04 Install Nanpy

There are only two steps remaining in the configuration. First, we need to get the Nanpy package downloaded and installed on the Pi. Our preferred way is to clone it with Git. Navigate to your home folder in the terminal (**cd ~**) and do the following in the terminal, one after the other:

```
easy_install nanpy
sudo apt-get install git
git clone  https://github.com/nanpy/nanpy.git
```

05 Configure your Arduino Uno

Why have we cloned the original Git repository? Nanpy relies on an update to the Arduino firmware to function correctly, so you'll need to access the **firmware** folder from the **nanpy** project directory to do it. Before typing the following into the terminal, plug your Arduino Uno into a spare port on the Raspberry Pi. Beware: the following takes some time!

```
cd nanpy/firmware
export BOARD=uno
make
make upload
```

06 Testing Arduino with your Pi

With the installation finally complete, we can test the setup to make sure it works properly. Before we do a proper 'Hello World' application in the code segment to the right, let's first ensure Nanpy is properly installed and the connection between Pi and Arduino is working. From your home folder (**cd ~**), type the following into the terminal:

```
nano nanpy_test.py
```

In the nano editor, simply write:

```
from nanpy imort Arduino
```

Now press **Ctrl+X**, **Y**, then **Enter** to save your new file.

Finally, in the terminal, type:

```
Python nanpy_test.py
```

If you don't see an error, then everything should be working fine. Now you can play with the code across the page to start learning your way around Nanpy.

Full code listing

```
# Like all good hardware-based 'Hello, World' applications, we'll start
# by making the light on the Arduino board flicker off and on.

from nanpy import Arduino
from time import sleep

Arduino.pinMode(13, Arduino.OUTPUT)
for i in range(10):
Arduino.digitalWrite(13, Arduino.HIGH)
sleep(2)
Arduino.digitalWrite(13, Arduino.LOW)
sleep(2)

# This will make the light controlled by pin 13 on the Arduino
# turn on and off every two seconds ten times.

# You can also assign pins a name, to make your code more readable.

light = 13

Arduino.pinMode(light, Arduino.OUTPUT)

# You can also assign multiple pins at the same time:

red_pin = 3
green_pin = 5
blue_pin = 9

for pins in (red_pin, green_pin, blue_pin):
    Arduino.pinMode(pins, Arduino.OUTPUT)

# if you've got an LED screen for your RasPi you'll probably
# find it works out of the box with Nanpy. Just make sure you
# assign the right pin numbers for your screen:

from nanpy import (Arduino, Lcd)

screen = Lcd([7, 8, 9, 10, 11, 12], [16, 2])
screen.printString("Hello, World!")

# If you're using potentiometers, buttons or analog sensors,
# you'll need to assign them as inputs
knob = 0
Arduino.pinMode(knob, Arduino.INPUT)

value = Arduino.analogRead(knob)

for i in range(10):
    print "The value of the knob is:", knob
    sleep(1)

# Sometimes you want to delay what the arduino does.
# This can help you get consistent, solid readings

def get_value():
    value = Arduino.analogRead(knob)
    Arduino.delay(100)
    return value

for i in range(100):
    print "The value is:", get_value()
```

You'll learn

1. Playing to strengths

While the RasPi is much more powerful than Arduino, the latter has the upper hand when it comes to interfacing with the real world. Leverage both their strengths to make better projects.

Create a Raspberry Pi three-colour lamp

Use the power of Arduino to do otherwise impossible projects with just a Raspberry Pi alone

What you'll need

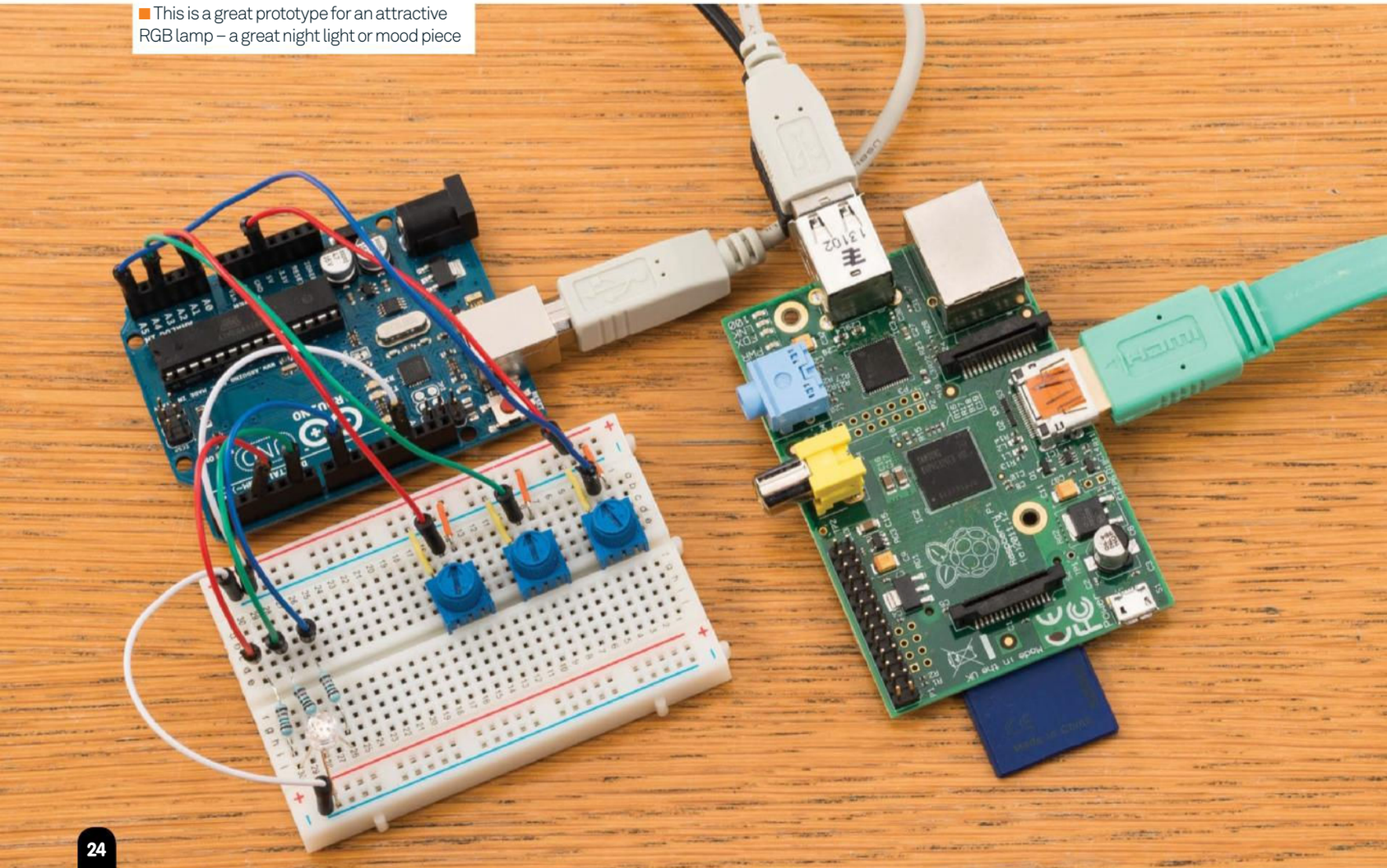
- Arduino Uno
- Breadboard
- Set of prototyping cables
- RGB LED (cathode)
- 3x potentiometers
- 3x 330 Ohm resisters

In the previous project we showed you how you can use an Arduino microcontroller to help the Raspberry Pi become proficient in new skills that can drastically increase the reach of your projects. With the aid of the extra 12 pins capable of PWM and analogue input, you could easily add multiple servos, analogue sensors and even add input devices like joysticks.

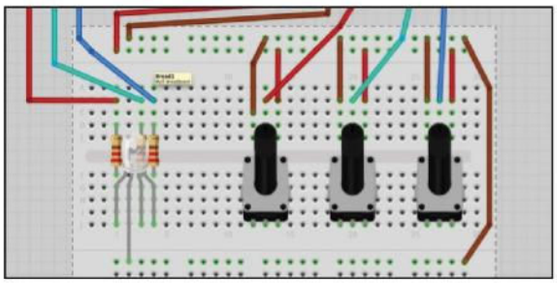
In this project we're going to demonstrate this by creating a three-colour lamp that employs three potentiometers (twisty knobs) to control each of the three colours in an RGB LED light. With it you can make most of the colours of the rainbow. As you'd expect, this would be much more difficult using just a Raspberry Pi alone.

01 Program with Arduino

You'll need to have followed the steps from with the previous project to correctly configure your Raspberry Pi and Arduino Uno. You'll also need to ensure you've got all the components from the list to the left. The resistors should be 330-ohm ideally, but can be of a higher resistance if that's all you have available. Arduinos can be bought as part of 'starter packs' that include exactly these kinds of components, but a quick visit to **www.cpc.co.uk** should fill any holes.



This is a great prototype for an attractive RGB lamp – a great night light or mood piece



02 Populate the breadboard

The circuit for this project might look a little complicated at first glance, but actually there's very little going on. As you'd expect, we want to control the LED light using PWM-enabled pins (to have fine-grained control of the brightness) and the potentiometers (pots) are being read by the analogue pins.

03 Connect the Arduino and Raspberry Pi

Assuming you don't plan to write up the code immediately yourself, you can grab it from the disc or from the website and drop it in your home folder. With the USB cable from the Arduino plugged into the Raspberry Pi, you simply need to run the code with the following:

```
python RGB_Mixer.py
```

Adjust the pots for the corresponding colour of the LED and the colours should change. If the pots are adjusting the wrong colours, just swap them over. You could use a table-tennis ball or plastic mug to diffuse the light to great effect.

04 Setting up the pins

As we demonstrated in the last project, it's easy to name and set the Arduino pins with Nanpy – in our code we've used two simple **for** loops to do the job. The **debug** value below simple prints the values of each pot to the terminal – very useful for debugging or getting a better handle on the code.

05 Functional operation

There are only really three main functions here, written with self-explanatory names. Firstly, **get_pots()** reads in the analogue pin value associated with each pot-pin and returns a tuple of the value for red, green and blue respectively. This is used by the **colour_mixing()** function to assign values to each of the associated PWM pins to change the colours of the LED.

06 Keeping things running

The **main()** function is where the other functions are set to work. Inside the function, we're asking Python to mix the colours (and print the values if **debug** is True) forever, except if we press **Ctrl+C** – initiating the keyboard interrupt. Since we want to clean up after ourselves, this action with trigger **close_pins()** – this turns off the pins attached to the LED, ready to be used next time.

You'll learn...

1. Analogue inputs

It is possible to utilise analogue inputs with the Raspberry Pi using an analogue-to-digital converter (ADC) chip like the MPC3008, but they're much easier to handle with an Arduino using Nanpy.

2. Comment your code!

We've tried to adhere to the best practices for commenting Python code in this project. We're using '#' comments before assignments and quoted comments within functions.

Full code listing

Get the code: bit.ly/1Vz5sGL

```
#!/usr/bin/env python

from nanpy import Arduino
from time import sleep

# set LED pin numbers - these go to the
# Digital pins of your Arduino
redPin = 3
greenPin = 6
bluePin = 9

# set pot pin numbers - these go to the
# (A)nalog pins of your Arduino
pot_r_Pin = 0
pot_g_Pin = 3
pot_b_Pin = 5

#set three coloured pins as outputs
for pins in (redPin, greenPin, bluePin):
        Arduino.pinMode(pins, Arduino.OUTPUT)

# set pot pins as inputs
for pins in (pot_r_Pin, pot_g_Pin, pot_b_Pin):
        Arduino.pinMode(pins, Arduino.INPUT)

# prints values to the terminal when True
debug = False

def get_pots():
        """
        Grab a reading from each of the pot pins and
        send it to a tuple to be read by the colour mixer
        """
        r = Arduino.analogRead(pot_r_Pin) / 4
        Arduino.delay(1)
        g = Arduino.analogRead(pot_g_Pin) / 4
        Arduino.delay(1)
        b = Arduino.analogRead(pot_b_Pin) / 4
        Arduino.delay(1)
        return r, g, b

def colour_mixing():
        """
        Call get_pots() and set
        the colour pins accordingly
        """
        r, g, b = get_pots()
        Arduino.analogWrite(redPin, r)
        Arduino.analogWrite(greenPin, g)
        Arduino.analogWrite(bluePin, b)
        Arduino.delay(1)

def close_pins():
        """
        Close pins to quit cleanly (doesn't work with a 'for
        loop' despite the pins happily initialising that way!)
        """
        Arduino.digitalWrite(redPin,Arduino.LOW)
        Arduino.digitalWrite(greenPin,Arduino.LOW)
        Arduino.digitalWrite(bluePin,Arduino.LOW)

def main():
        """
        Mix the colours using three pots.
        Ctrl+C cleans up the pins and exits.
        """
        try:
                print "Adjust the pots to change the colours"
                while True:
                        colour_mixing()
                        sleep(0.2)
                        if debug:
                                print "Red: {:d} | Green: ↵
{:d} | Blue: {:d}".format(r, g, b)
        except KeyboardInterrupt:
                close_pins()
                print "\nPins closed"

if __name__ == '__main__':
        main()
```

Make a game with Python

We update the retro classic Pong for the Linux generation with a new library called SimpleGUITk

What you'll need

- Latest version of Raspbian
  www.raspberrypi.org/downloads
- Pillow
  https://github.com/python-imaging/Pillow
- SimpleGUITk
  https://github.com/dholm/simpleguitk/

The Raspberry Pi is a fantastic way to start learning how to code. One area that can be very rewarding for amateur coders is game programming, allowing for a more interactive result and a greater sense of accomplishment. Game programming can also teach improvisation and advanced mathematics skills for code. We'll be using the fantastic SimpleGUITk module in Python, a very straightforward way of creating graphical interfaces based on Tkinter.

01 Python module preparation

Head to the websites we've listed in 'What you'll need' and download a zip of the source files from the GitHub pages. Update your Raspbian packages and then install the following:

```
$ sudo apt-get install python-dev python-setuptools tk8.5-dev tcl8.5-dev
```

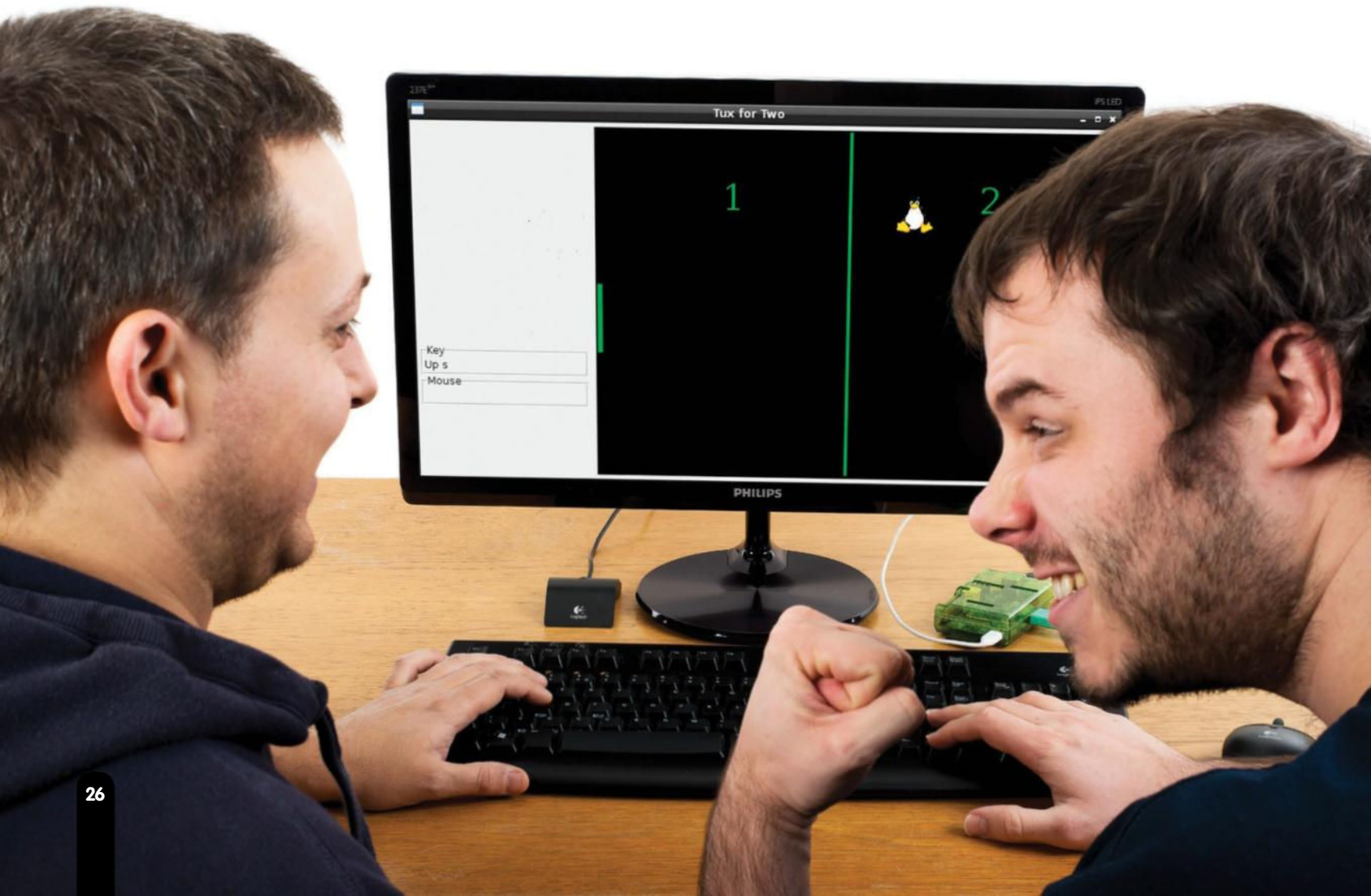
02 Install the modules

Open the terminal and use **cd** to move to the extracted Pillow folder. Once there, type:

```
$ sudo python setup.py install
```

Once that's complete, move to the **simpleguitk** folder and use the same command to install that as well.

Rob got off to a good start, but it was all downhill from there...



03 Write your code

Launch IDLE 2, rather than IDLE 3, and open a new window. Use the code listing to create our game 'Tux for Two'. Be careful to follow along with the code to make sure you know what you're doing. This way, you can make your own changes to the game rules if you wish.

04 Set up the game

There's nothing too groundbreaking to start the code: Tux's and the paddles' initial positions are set, along with the initial speed and direction of Tux. These are also used when a point is won and the playing field is reset. The direction and speed is set to random for each spawn.

05 The SimpleGUI code

The important parts in the **draw** function are the **draw_line**, **draw_image** and **draw_text** functions. These are specifically from SimpleGUI, and allow you to easily put these objects on the screen with a position, size and colour. You need to tie them to an object, though – in this case, **canvas**.

06 SimpleGUI setup code

The last parts are purely for the interface. We tell the code what to do when a key is depressed and then released, and give it a frame to work in. The frame is then told what functions handle the graphics, key functions etc. Finally, we give it **frame.start()** so it starts.

Full code listing

Get the code: bit.ly/1MK2cCy

```
import simpleguitk as simplegui
import random

w = 600
h = 400
tux_r = 20
pad_w= 8
pad_h = 80

def tux_spawn(right):
    global tux_pos, tux_vel
    tux_pos = [0,0]
    tux_vel = [0,0]
    tux_pos[0] = w/2
    tux_pos[1] = h/2
    if right:
        tux_vel[0] = random.randrange(2, 4)
    else:
        tux_vel[0] = -random.randrange(2, 4)
    tux_vel[1] = -random.randrange(1, 3)

def start():
    global paddle1_pos, paddle2_pos, paddle1_vel, paddle2_vel
    global score1, score2
    tux_spawn(random.choice([True, False]))
    score1, score2 = 0,0
    paddle1_vel, paddle2_vel = 0,0
    paddle1_pos, paddle2_pos = h/2, h/2

def draw(canvas):
    global score1, score2, paddle1_pos, paddle2_pos, ↵
tux_pos, tux_vel
    if paddle1_pos > (h - (pad_h/2)):
        paddle1_pos = (h - (pad_h/2))
    elif paddle1_pos < (pad_h/2):
        paddle1_pos = (pad_h/2)
    else:
        paddle1_pos += paddle1_vel
    if paddle2_pos > (h - (pad_h/2)):
        paddle2_pos = (h - (pad_h/2))
    elif paddle2_pos < (pad_h/2):
        paddle2_pos = (pad_h/2)
    else:
        paddle2_pos += paddle2_vel
    canvas.draw_line([w / 2, 0],[w / 2, h], 4, "Green")
    canvas.draw_line([(pad_w/2), paddle1_pos + (pad_h/2)], ↵
[(pad_w/2), paddle1_pos - (pad_h/2)], pad_w, "Green")
    canvas.draw_line([w - (pad_w/2), paddle2_pos + (pad_h/2)], ↵
[w - (pad_w/2), paddle2_pos - (pad_h/2)], pad_w, "Green")
    tux_pos[0] += tux_vel[0]
    tux_pos[1] += tux_vel[1]
    if tux_pos[1] <= tux_r or tux_pos[1] >= h - tux_r:
        tux_vel[1] = -tux_vel[1]*1.1
    if tux_pos[0] <= pad_w + tux_r:
        if (paddle1_pos+(pad_h/2)) >= tux_pos[1] ↵>= (paddle1_
pos-(pad_h/2)):
            tux_vel[0] = -tux_vel[0]*1.1
            tux_vel[1] *= 1.1
        else:
            score2 += 1
            tux_spawn(True)
    elif tux_pos[0] >= w - pad_w - tux_r:
        if (paddle2_pos+(pad_h/2)) >= tux_pos[1] >= ↵
(paddle2_pos-(pad_h/2)):
            tux_vel[0] = -tux_vel[0]
            tux_vel[1] *= 1.1
        else:
            score1 += 1
            tux_spawn(False)
    canvas.draw_image(tux, (265 / 2, 314 / 2), (265, 314), ↵
tux_pos, (45, 45))
    canvas.draw_text(str(score1), [150, 100], 30, "Green")
    canvas.draw_text(str(score2), [450, 100], 30, "Green")

def keydown(key):
    global paddle1_vel, paddle2_vel
    acc = 3
    if key == simplegui.KEY_MAP["w"]:
        paddle1_vel -= acc
    elif key == simplegui.KEY_MAP["s"]:
        paddle1_vel += acc
    elif key==simplegui.KEY_MAP["down"]:
        paddle2_vel += acc
    elif key==simplegui.KEY_MAP["up"]:
        paddle2_vel -= acc

def keyup(key):
    global paddle1_vel, paddle2_vel
    acc = 0
    if key == simplegui.KEY_MAP["w"]:
        paddle1_vel = acc
    elif key == simplegui.KEY_MAP["s"]:
        paddle1_vel = acc
    elif key==simplegui.KEY_MAP["down"]:
        paddle2_vel = acc
    elif key==simplegui.KEY_MAP["up"]:
        paddle2_vel = acc

frame = simplegui.create_frame("Tux for Two", w, h)
frame.set_draw_handler(draw)
frame.set_keydown_handler(keydown)
frame.set_keyup_handler(keyup)
tux = simplegui.load_image('http://upload.wikimedia.org/ ↵
wikipedia/commons/a/af/Tux.png')

start()
frame.start()
```

Raspberry Pi stop motion animation

Fancy yourself as the next Nick Park? Set up this DIY stop-motion studio and see what you can do

What you'll need

- Latest version of Raspbian
  www.raspberrypi.org/downloads
- picamera Python module
  picamera.readthedocs.org
- RasPi camera module
- Pygame
  www.pygame.org

- Pi-Mation is available on GitHub via https://github.com/russb78/pi-mation

The Raspberry Pi camera module opens the door for your Pi projects to incorporate aspects of photography and movie making. We're combining both here to create a fully featured stop-motion animation application, Pi-Mation, which makes it incredibly easy to create impressive HD animations.

We've written this project with Python and it relies on two libraries that you will need to install. Picamera (**picamera.readthedocs.org**) is a pure Python interface to the Raspberry Pi camera module and is a must for all camera module owners. Pygame (**www.pygame.org**), which ensures our images can be displayed on demand.

01 Set up the camera module

First things first, you need to make sure your Raspberry Pi is up to date. In the terminal, type:

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

Next we need to update the Pi's firmare and ensure camera module is activated. Bear in mind that this takes some time.

```
sudo rpi-update
sudo raspi-config
```

02 Install other dependencies

Next we'll make sure Pygame and picamera are installed:

```
sudo apt-get install python-setuptools
easy_install -user picamera
```

Finally, to install Pygame and the video apps, type:

```
sudo apt-get install python-pygame
sudo apt-get install libav-tools && sudo ↵
apt-get install omxplayer
```

03 Final setup

We're going to install Pi-Mation with Git, so let's make sure it's installed:

```
sudo apt-get install git
```

With a terminal open, navigate to your home directory (with **cd ~**) and type:

```
git clone https://github.com/russb78/pi-mation.git
```

If you play with the code and break it, you can revert it back to its original state with:

```
git checkout pi-mation.py
```

04 Running and testing Pi-Mation

Now navigate into the **pi-mation** folder and run the application with:

```
python pi-mation.py
```

Pressing the space bar calls **take_pic()** from the **main()** loop, which saves an image and creates a preview that's loaded by **update_display()**. The Tab button is coded to toggle between two states by asking two variables to switch values.

05 Getting animated

The **main()** loop checks for keyboard events before updating the screen around 30 times per second. Since the camera's live preview is working independently of that loop, **update_display()** only needs to worry about updating the preview image (**prev_pic**) Since **take_pic()** adds to **pics_taken**, only the very latest picture is shown. The **animate()** function is essentially a microcosm of **update_display()**. When the **P** key is pressed, the live preview is suspended and for all of the pictures taken (**pics_taken**), each one will be 'blitted' (updated) on the main window.

Full code listing

```
import pygame, picamera, os, sys

pics_taken = 0
current_alpha, next_alpha = 128, 255
fps = 5

pygame.init()
res = pygame.display.list_modes() # return the best resolution for your monitor
width, height = res[0]
print "Reported resolution is:", width, "x", height
start_pic = pygame.image.load(os.path.join('data', 'start_screen.jpg'))
start_pic_fix = pygame.transform.scale(start_pic, (width, height))
screen = pygame.display.set_mode([width, height])
pygame.display.toggle_fullscreen()
pygame.mouse.set_visible = False
play_clock = pygame.time.Clock()
camera = picamera.PiCamera()
camera.resolution = (width, height)

def take_pic():
    global pics_taken, prev_pic
    pics_taken += 1
    camera.capture(os.path.join('pics', 'image_' + str(pics_taken) + '.jpg'), use_video_port = True)
    prev_pic = pygame.image.load(os.path.join('pics', 'image_' + str(pics_taken) + '.jpg'))

def delete_pic():
    global pics_taken, prev_pic
    if pics_taken >= 1:
        pics_taken -= 1
        prev_pic = pygame.image.load(os.path.join('pics', 'image_' + str(pics_taken) + '.jpg'))

def animate():
    camera.stop_preview()
    for pic in range(1, pics_taken):
        anim = pygame.image.load(os.path.join('pics', 'image_' + str(pic) + '.jpg'))
        screen.blit(anim, (0, 0))
        play_clock.tick(fps)
        pygame.display.flip()
    play_clock.tick(fps)
    camera.start_preview()

def update_display():
    screen.fill((0,0,0))
    if pics_taken > 0:
        screen.blit(prev_pic, (0, 0))
    play_clock.tick(30)
    pygame.display.flip()

def make_movie():
    camera.stop_preview()
    pygame.quit()
    print "\nQuitting Pi-Mation to transcode your video."
    os.system("avconv -r " + str(fps) + " -i " + str((os.path.join('pics', 'image_%d.jpg'))) + " -vcodec libx264 video.mp4")
    sys.exit(0)

def change_alpha():
    global current_alpha, next_alpha
    camera.stop_preview()
    current_alpha, next_alpha = next_alpha, current_alpha
    return next_alpha

def quit_app():
    camera.close()
    pygame.quit()
    print "You've taken", pics_taken, " pictures. Don't forget to back them up!"
    sys.exit(0)

def intro_screen():
    intro = True
    while intro:
        for event in pygame.event.get():
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_ESCAPE:
                    quit_app()
                elif event.key == pygame.K_F1:
                    camera.start_preview()
                    intro = False
        screen.blit(start_pic_fix, (0, 0))
        pygame.display.update()

def main():
    intro_screen()
    while True:
        for event in pygame.event.get():
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_ESCAPE:
                    quit_app()
                elif event.key == pygame.K_SPACE:
                    take_pic()
                elif event.key == pygame.K_BACKSPACE:
                    delete_pic()
                elif event.key == pygame.K_RETURN:
                    make_movie()
                elif event.key == pygame.K_TAB:
                    camera.preview_alpha = change_alpha()
                    camera.start_preview()
                elif event.key == pygame.K_F1:
                    camera.stop_preview()
                    intro_screen()
                elif event.key == pygame.K_p:
                    if pics_taken > 1:
                        animate()
        update_display()

if __name__ == '__main__':
    main()
```

Get the code: bit.ly/1LwoOJA

"A Raspberry Pi is all you need to make some crazy gadgets"

38
Take control of a car

Hardware

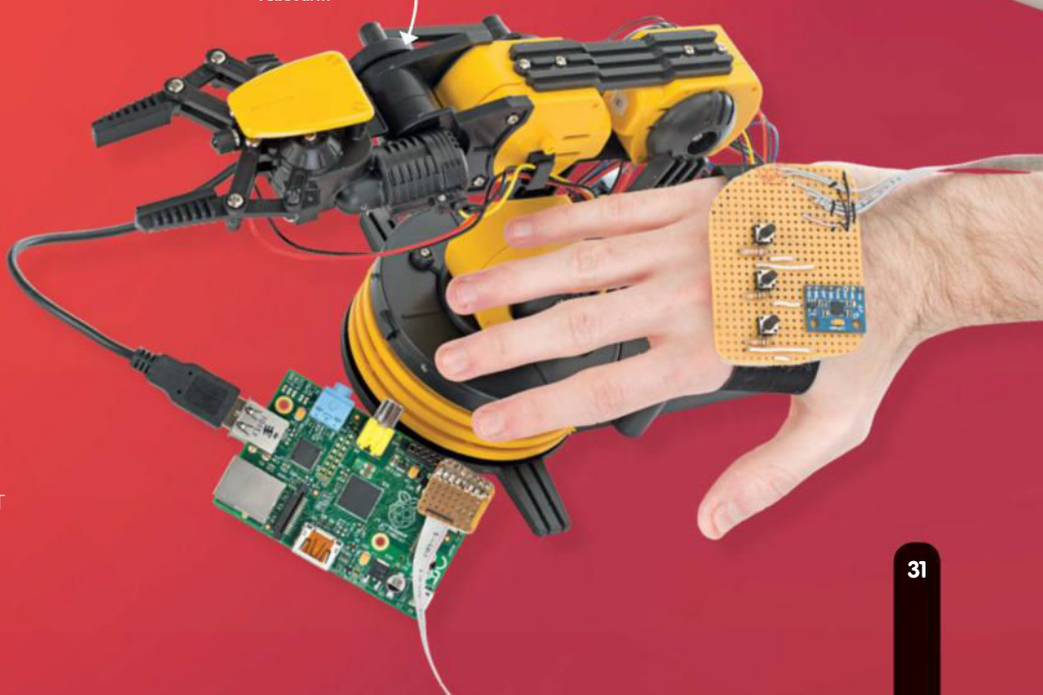
60
Build a RasPi Minecraft console

54
Network a Hi-Fi

36
Read up on this 3D-printed telescope

56
Display your digital photos

44
Check out a crazy robot arm



Turn a Pi into a router

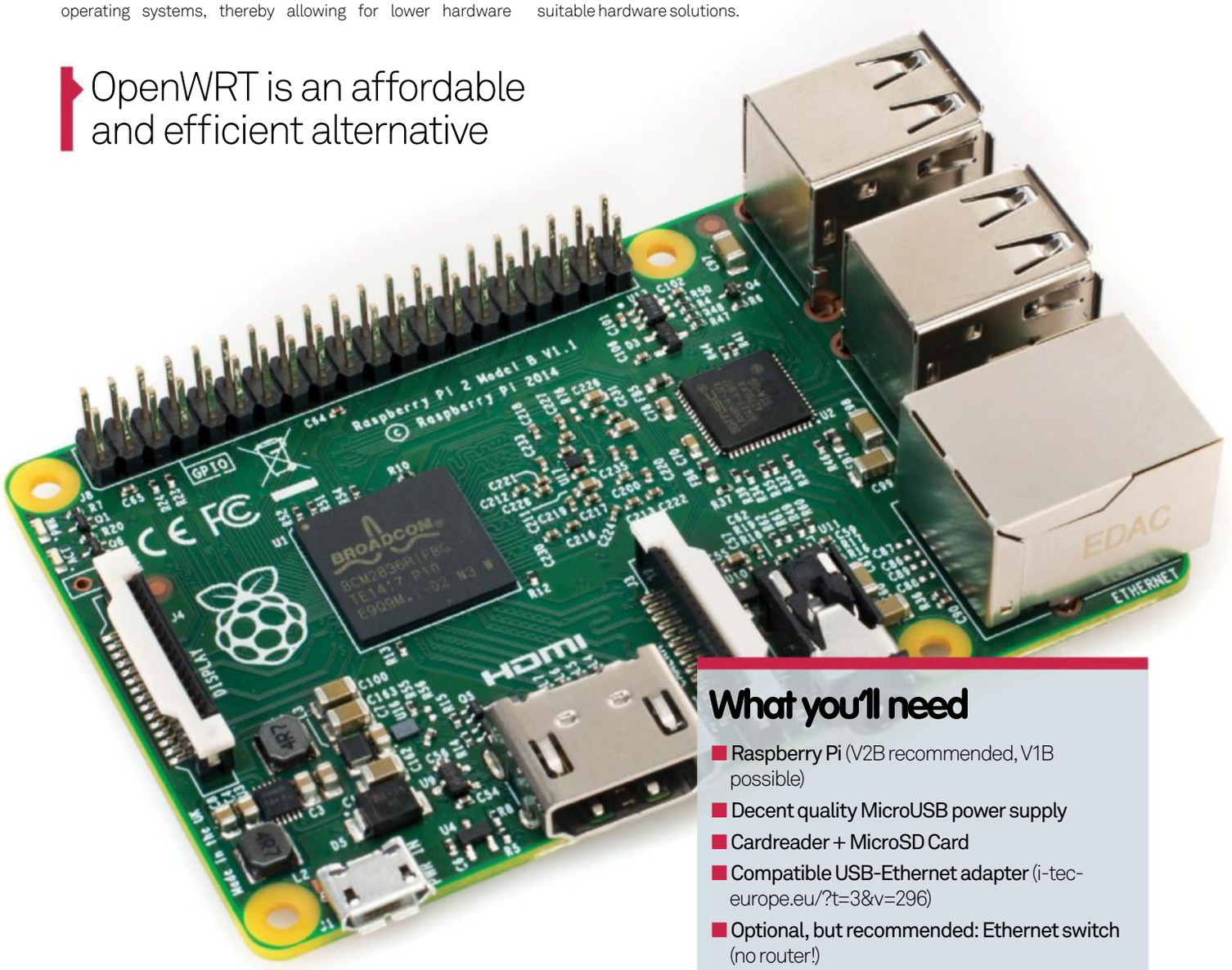
Learn the basics of OpenWRT using a Raspberry Pi as a router

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

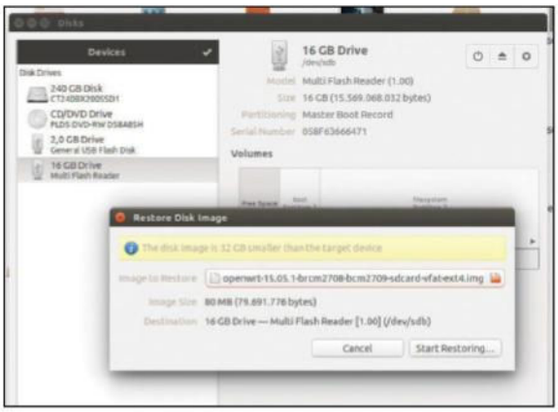


What you'll need

- **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!)

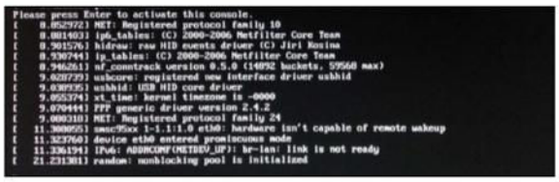
01 Set it up

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.



Be aware that 192.168.1.1 is a popular address for routers: if your Wi-Fi router uses it, the network connection needs to be disabled during the following steps.

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).

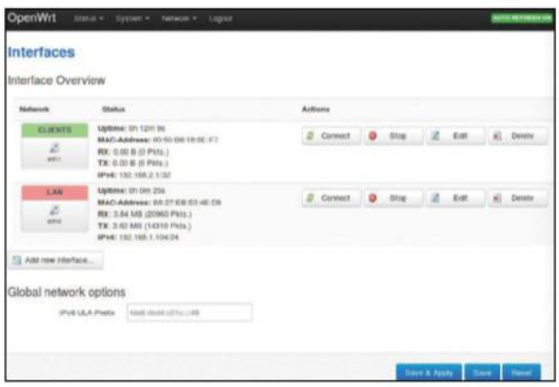
10 Firewall ahoy!

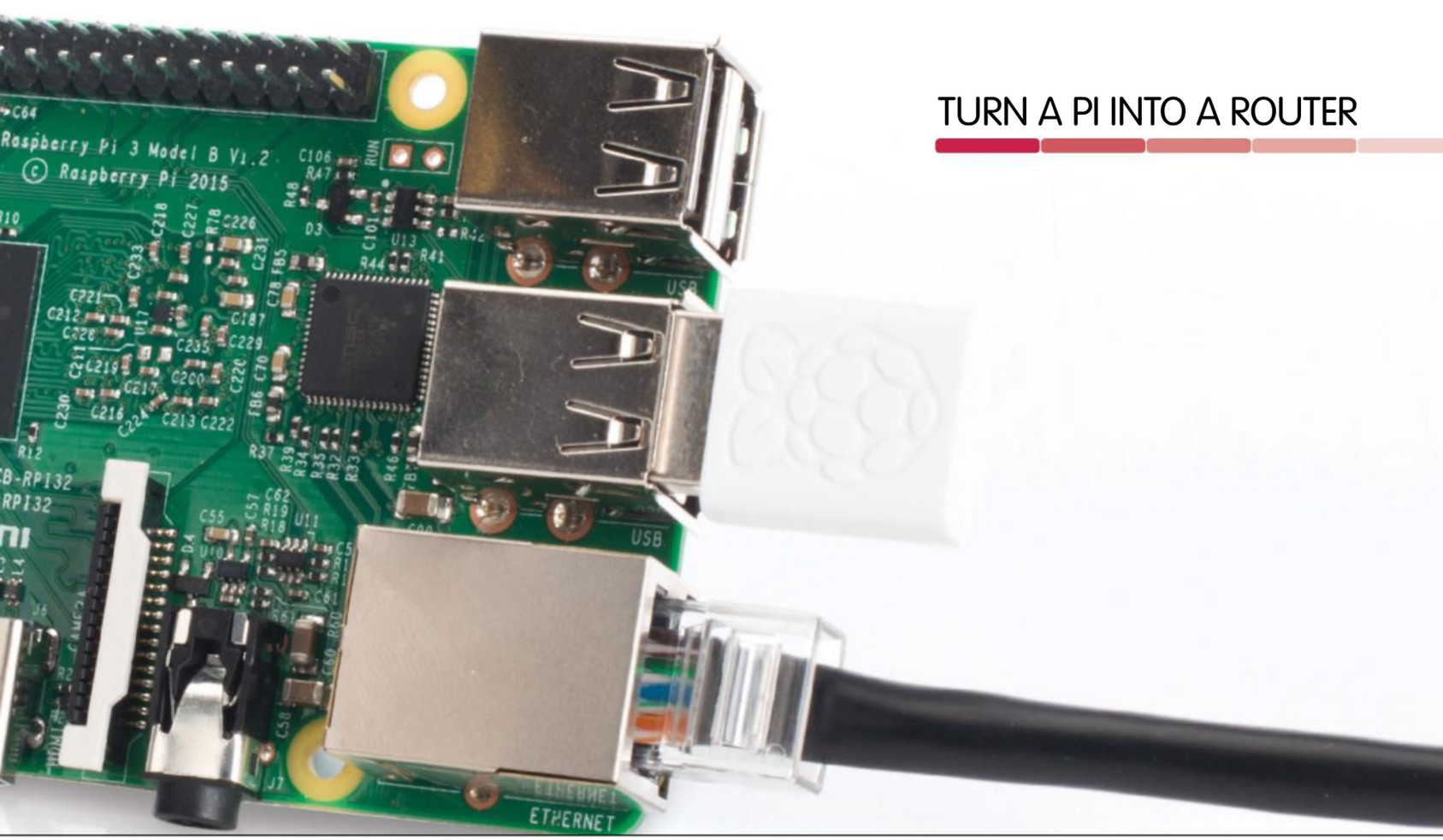
From this point onward, attempting to interact with the LuCI frontend from "outside" of the network will lead to 'Unable to connect' errors – by default, remote configuration is not allowed to make attacks on OpenWRT more difficult.

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
```



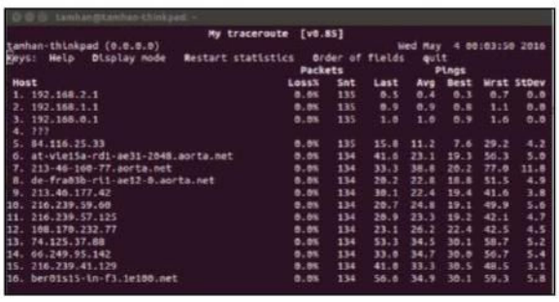


11 Test the presence of the router

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.



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**).

3D-printed If you have already invested in a 3D printer or otherwise have access to one, the material cost of printing the parts is negligible

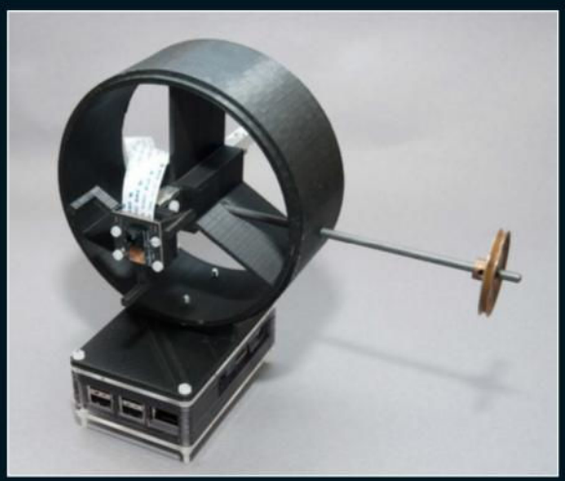
Camera module The camera module's lens has been removed and it is placed below a 4.5-inch mirror, which forms the image

Optical tube 3D-printing this part of the PiKon would have been very inefficient, so Andy and Mark used readily-available venting duct

Focusing Things get a little more complex down towards the base and some Meccano pieces are used to hold everything together

Components list

- Raspberry Pi
- Camera module
- 3D printer
- 5 CAD files to print
  bit.ly/1wfl9a8
- White venting duct
- Small square of aluminium
- Focusing knob
- Meccano pieces
- Tripod



Left The 3D-printed base for the telescope is also composed of pieces of Meccano as well as the small square of aluminium

Below Here is an example of the kind of photo that can be taken with the PiKon telescope: the Moon, with enough detail to be able to see its craters and surface texture

How I made: PiKon

3D-printed telescope meets RasPi camera

Tell us how you began your project

Mark Wrigley I run this small company called Alternative Photonics and like to find out about new technologies. When Sheffield's Festival of the Mind came along I thought of what could we put together in terms of things that are new and disruptive, and that's how the whole project started. The two things I was interested in were using Raspberry Pis for photography and 3D printing. With 3D printers you've got a device in the ballpark of £500, putting it amongst the price you'd pay for consumer items, so manufacturing is something you can do at home. There are companies who will print something for you, so you send in a design and they'll produce it. And there are maker communities – Andy runs a group called Sheffield Hardware Hackers and Makers.

Andy Kirby Sheffield Hardware Hackers and Makers is for anyone off the street to come along to if they want to hack and make things. I set it up as part of the original RepRap project that ran on the Internet quite some years ago [Ed: RepRaps are 3D printer kits that can print parts for more RepRaps]. I found that there were quite a few people building printers who got to a certain point and then got stuck, so I set up this group originally to be a drop-in for people to come along to and get the help they needed.

Andy, what was your role in the PiKon?

Andy I helped Mark pick a 3D printer that was going to be pitched right for his skillset and then, as he was building up the printer, when he got to bits where he got stuck he'd bring it along to the Hardware Hackers group and say 'It doesn't do this right' or 'I can't get it to do that', and we'd work through the problems together. Once he'd got his printer going, I worked through the CAD software with him to see what was available that was open source and within his capabilities. There are various things you can't do with a 3D printer when you design parts, so we had a conversation about that and I gave him the shortcut to get working parts out quicker.

How does the PiKon work?

Mark Most telescopes work on the principle that you have some sort of object lens or mirror which forms an image and then you use an eyepiece to examine that image, so it's usually what's termed as a virtual image. With the PiKon, what we're doing is using the objective device, in this case a 4.5-inch (335mm) mirror, to form an image and then placing the Raspberry Pi camera without its lens – so just a sensor – where the image is formed. So effectively, instead of using an eyepiece we're examining the image electronically by placing the Raspberry Pi sensor there, and the sensor is suitably small so we're able to examine a fairly small area of the image.

What kind of resolution do you get?

Mark At the moment, the setup that we've got is equivalent to a magnification of about 160. If you look at the Moon, the field of view of your eye is about half of one degree; the PiKon's maximum field of view is about a quarter of one degree, so effectively you can see about half the moon in full frame. The next thing I'd like to do is look at planets. In terms of its resolution, the PiKon's sensor is five megapixels, which is 2500x2000 pixels. If you're going to reproduce an image in print you'd normally use something like 300dpi, so 5MP would allow you to reproduce something like an A4 image. On a computer screen all you need is 72dpi, so what I'm quite interested in doing next is seeing how much magnification we can get simply by cropping – so literally throwing away some of the pixels to be able to zoom in on something like a planet. If you read the *Astronomy for Beginners* stuff, they talk about needing a magnification of 200-250 to be able to observe planets, so I'm hoping we can do things like see the rings of Saturn. We're not out to rival the Hubble – we think that what you've got is a reasonable instrument and you can do a few interesting things with it. The other thing is that because it's using a Raspberry Pi sensor instead of an eyepiece, you're immediately into the world of astrophotography rather than doing observations, so that's the sort of way we're going.

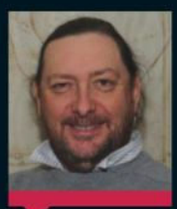
How do you control the PiKon's camera?

Mark We would like to do it better!

Andy At the moment it's done through the command line. I'm not a Raspberry Pi programmer... So we're using raspistill at the moment, which gives you a certain number of seconds of preview and then takes the picture, so it's a bit clunky. I'm talking to a guy who's into Raspberry Pi and is also a photographer, and he's written some programs where you have a shutter button. The next thing to do then is to control PiKon from an input/output device like a shutter button and then give the JPG files you produce a sequential or a date-stamped filename. One thing I'd like to see next is if we could get this hardware out into the public and attract people to actually come along and develop more and more software for it. I tried taking pictures of the International Space Station with an ordinary camera, for example. It's really difficult because suddenly this dot comes flying across the horizon and you have to swing around, get your camera lined up, press the shutter and so on. One thought I had was it would be nice if you could take multiple shots with the PiKon – you press a button and it keeps taking a photograph every five seconds.

> The setup we've got is equivalent to a magnification of about 160

Mark Wrigley is a member of the Institute of Physics and holds a Licentiateship with the Royal Photographic Society



Andy Kirby was an early contributor to the RepRap project and runs the Sheffield Hardware Hackers and Makers group

Like it?

The Raspberry Pi Foundation website featured a project that mounts the Pi and camera onto a telescope and captures great images **bit.ly/1qTp3Pb**

Further reading

If you're interested in astrophotography and developing software for PiKon, check out these astronomy packages: **bit.ly/100wj65**

Build a Raspberry Pi-controlled car

Make use of cutting-edge web technologies to take control of a remote controlled car with a smartphone or tablet...

Web technologies are moving forward at a huge pace, cloud technologies are bringing mass computing to individuals, and hardware has reached a perfect moment in time where sensors, displays and wireless technology have all evolved into efficient and affordable devices. We truly are at a point where nearly anyone can take an idea from nothing to a working product in a week and at very little cost. Just like this project, which is fun, quick and easy to build on and a fantastic way to learn. We're going to grab an old remote-control car, rip off its radio receiver and replace it with the Raspberry Pi, hook it up on the network, fire up a bleeding-edge web server and then get your smartphone or tablet to control it by tilting the device. By the end of this, not only will you have a fun toy – you will have learnt about the basic technologies that are starting to power the world's newest and biggest economy for the foreseeable future. Welcome to tomorrow!

Raspberry Pi-controlled car build process

Components list

- A toy RC car with two channels (steering and drive)
- Adafruit PWM I2C servo driver
- Female-to-female jumper cables
- 5V battery power bank

Estimated cost: £60 / $100

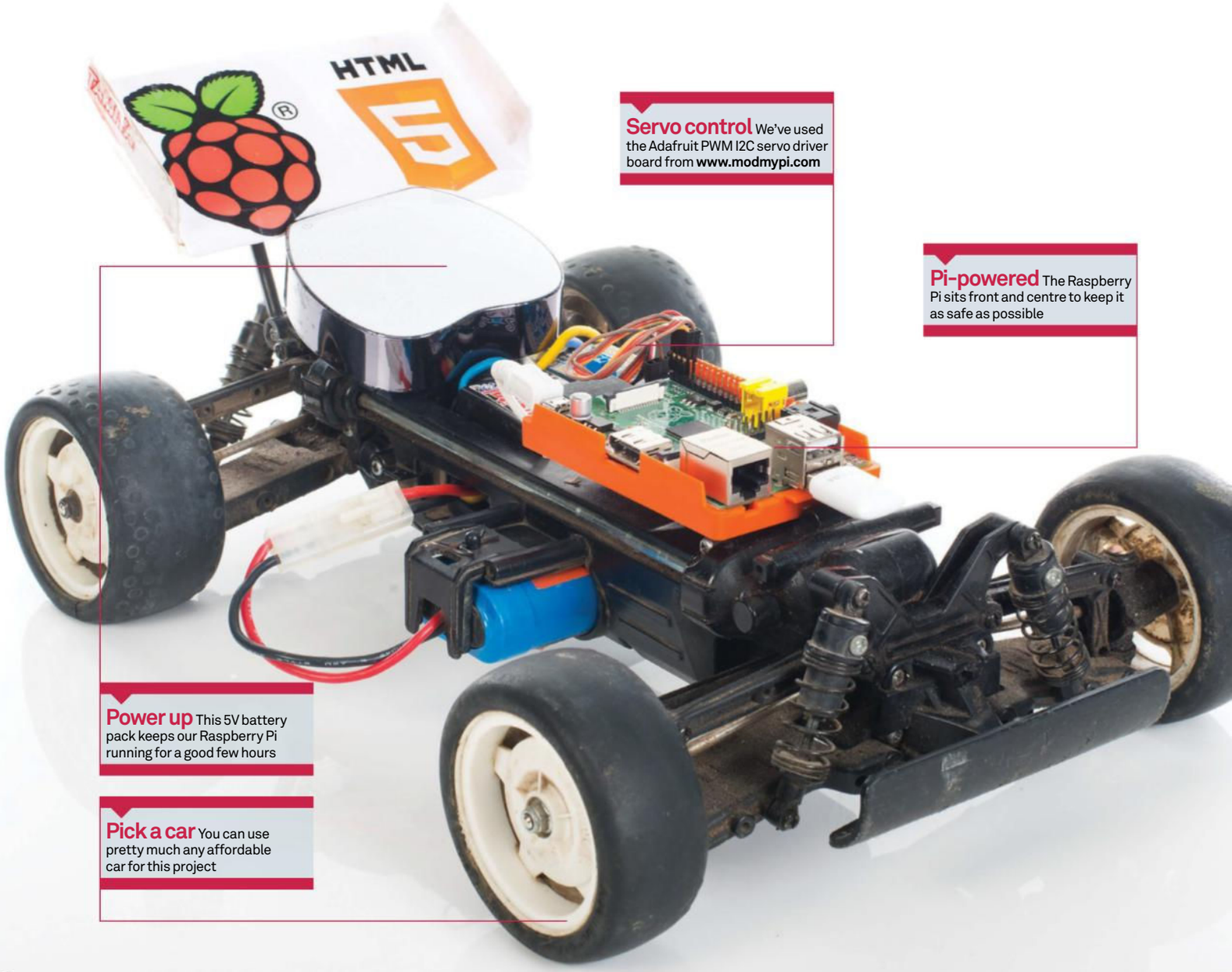
Components from www.modmypi.com

Before you can take control of your car with a smartphone, you'll need to make some significant changes to the chassis

To help our toy car come to life using the latest web technologies and our credit card-sized computer, we're going to need to make some pretty significant changes to its workings. Fortunately, the most complex aspects of the build can be accomplished with a couple of affordable purchases, namely a servo controller board to take care of the steering and throttle, and a 5V battery pack to keep the Raspberry Pi running smoothly.

01 Identify and remove old radio

This project is effectively replacing the car's normal transmitter and receiver. Notice the three sockets on the original receiver: one goes to the motor controller and one to the steering servo. Some remote-control cars also have separate battery for the electronics, but those (especially with an electronic speed controller with BEC) get their 5V power supply directly from the speed controller, saving on components. If you don't have a speed controller with 5V BEC, you'll need to get a 5V supply elsewhere. Many shops sell 5V battery power supplies – often as mobile phone emergency top-ups. **www.modmypi.com** sells a suitable 5V battery power bank for under £20.



We're using the Raspberry Pi's I2C bus to control the servo interface board

02 Attach the servo cables to the new controller

We soldered our 16-channel I2C servo controller board from **www.modmypi.com** as per its instructions and simply plugged channel 0 (steering) and channel 1 (motor) headers onto it. There are six cables in total: the bottom two are ground, the middle two are the power and the top two are the PWM (pulse-width modulation) signals. This is a good time to think of places to mount the extra components and the best fixing method seems to be sticky-back Velcro.

03 Connect the I2C bus to the Raspberry Pi

We're using the Raspberry Pi's I2C bus to control the servo interface board, which only needs four cables – they all go between the Raspberry Pi and the servo controller board as pictured. This month's accelerometer tutorial explains how to set up I2C on the Raspberry Pi.

From top to bottom we need to use the 1. GND, 2. SCL, 3. SDA and 4. VCC, which map directly to the same ports on the Raspberry Pi. Essentially this is power, ground and two communication channels – it's all pretty straightforward so far...

04 Hooking it up to the Raspberry Pi

On a Rev 1 Raspberry Pi, the cables look the same. Though the Rev boards have different labelling, the physical pins are in the same place. Bottom left (closest to the RasPi power connector) is the 3.3V power; next to that is the SDA header, which is the data channel. Next to that in the bottom right is the SCL channel, which controls the clock of the I2C devices. And finally – on the top-right port – is the Ground.

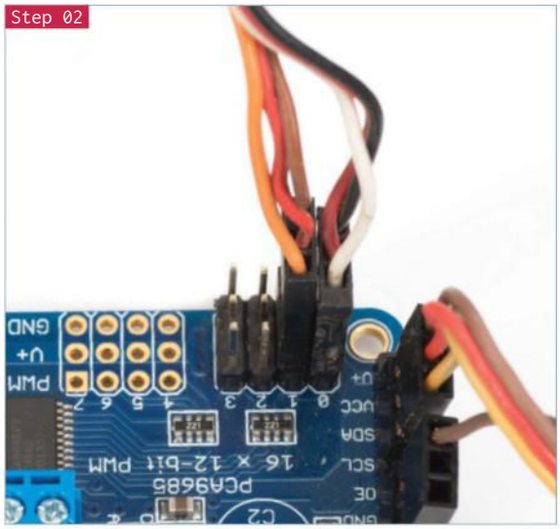
05 Overview of the main components

You should now have the servo board in the middle with the steering servo and speed controller on one side and the Raspberry Pi on the other. The motor is connected to the other end of the speed controller (that end should have much thicker wires); the speed controller also has two thick wires going to the main car's battery – in this case a 7.2V NiCad. We now have two very separate power systems with the high current motors on one side and the low current electronics on the other. Let's make sure it stays that way!

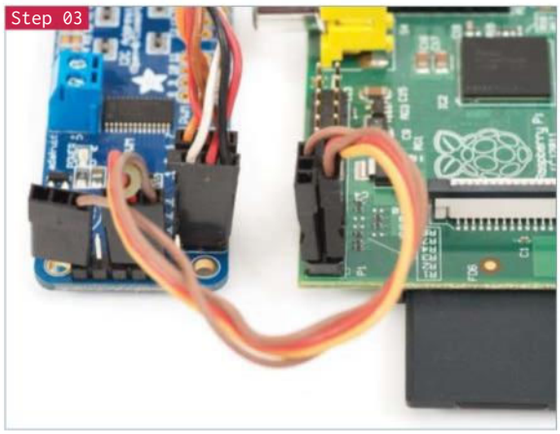
06 Find everything a home

Now it's time to find a home for the new components. Use plenty of sticky-back Velcro, tie wraps or elastic bands to keep everything secure and find spaces in the car's body to hide the wires where possible. While it is possible to stick or screw the Raspberry Pi directly to the car, we recommend to use at least the bottom half of a case for added protection and ease of access. Insert your SD card, network cable or Wi-Fi dongle (if programming from another machine) and power supply. Sit back and admire your hacking. Next we'll tackle the software side of the project...

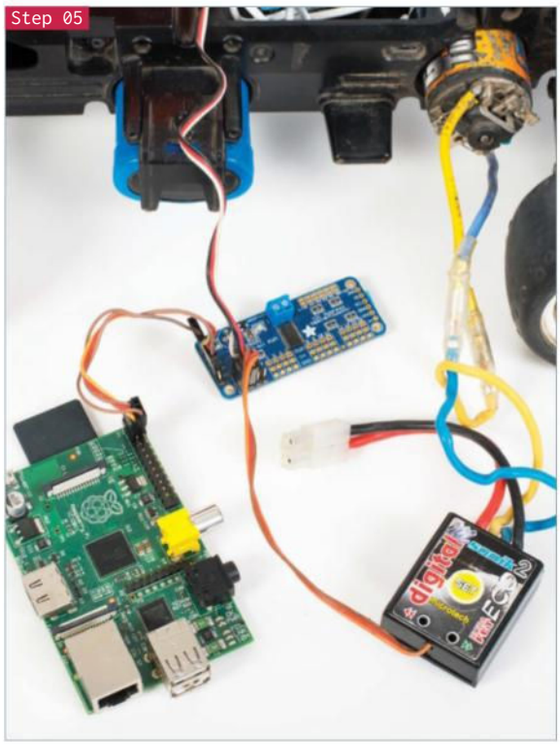
Step 02



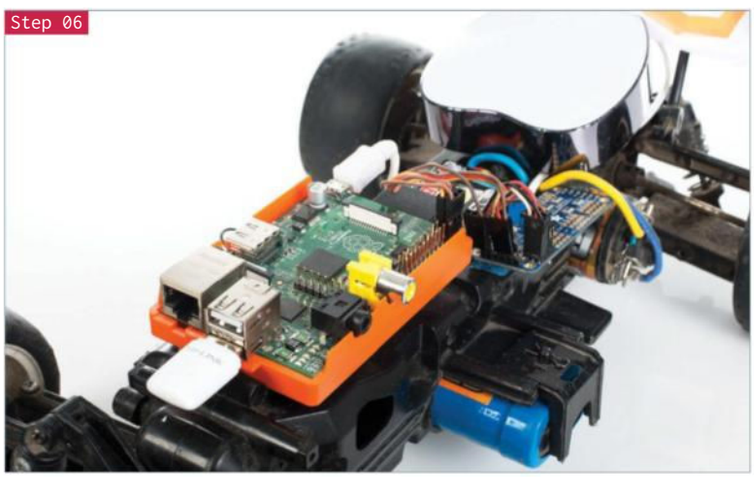
Step 03



Step 05



Step 06



Controlling your Raspberry Pi-powered car

Control a toy car with a smartphone and the latest web technologies

Now we have our fantastic Raspberry Pi-powered car all wired and charged, it's time to make it come alive. We're using the best web technologies that the JavaScript programming language offers, to harness the natural movement of your hand and wirelessly drive the vehicle. Each little movement will trigger an event that calculates what the car should do and then sends it over a socket connection up to 20 times a second.

What you'll need

- A RasPi car, ready to go
- An internet connection
- A reasonably modern smartphone/tablet
- Pi car source code
  github.com/shaunuk/picar

01 Download and install the software

To get the I2C connectivity working, you can follow the steps from pages 64-65. Next we'll need to find a home for our new project code – how about **/var/www/picar**? Type **sudo mkdir /var/www/picar** in the terminal to make the directory and then change into that directory: **cd /var/www/picar**

Now, to download the project using Git, type **sudo git clone http://github.com/shaunuk/picar**. If you haven't got Git, install it with **sudo apt-get install git**.

This will download the custom software for driving the car, but we still need the web server and some other bits before we can start burning rubber...

02 Download and install Node.js

We now need to get the awesome Node.js and its package tool, the Node package manager (npm). Type **sudo wget http://nodejs.org/dist/v0.10.21/node-v0.10.21-linux-arm-pi.tar.gz**. This will download a fairly recent version of Node.js – the version Raspbian has in its repositories is way too old and just doesn't work with the new technologies we're about to use. Extract the node package by typing **sudo tar -xvzf node-v0.10.21-linux-arm-pi.tar.gz**.

03 Configure Node.js

To make it easy to run from everywhere, we will create symbolic links for Node and npm binaries. In the terminal, type **sudo ln -s /var/www/node-v0.10.21-linux-arm-pi/bin/node /bin/node** and then **sudo ln -s /var/www/node-v0.10.21-linux-arm-pi/bin/npm /bin/npm**. Then, to get the extra modules, type **npm install socket.io node-static socket.io adafruit-i2c-pwm-driver sleep optimist**

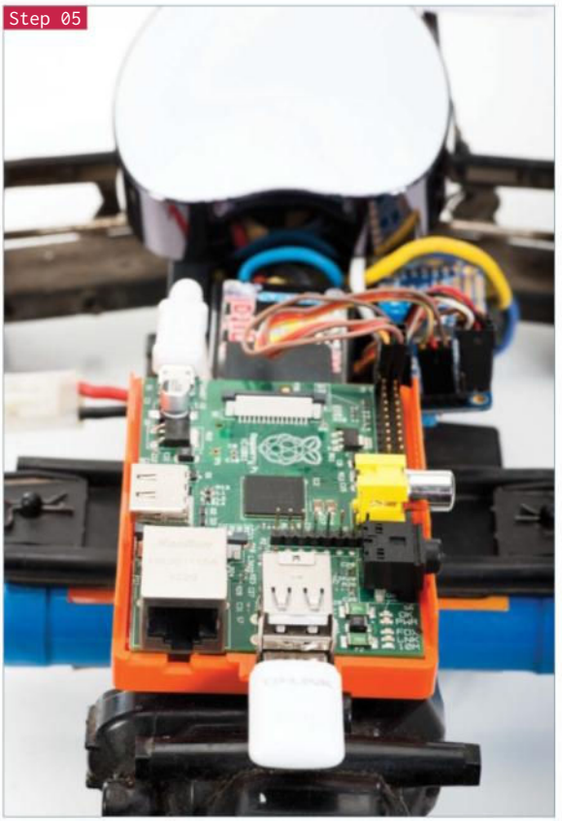
04 Get to know the project

Now we have everything, you should see three files: the server (**app.js**), the client (**socket.html**) and the jQuery JavaScript library for the client. The server not only drives the servos, but it is a web server and sends the socket.html file and jQuery to the browser when requested – it's a really neat and simple setup and just right for what we're trying to achieve.

Step 07

Below All you need to finish off your project is access to a smartphone or tablet

Step 05



Above You need to adjust some of the variables to control your particular remote controlled car set-up

05 Test the servos

Our handy little program (**app.js**) has a special mode just for testing. We use two keywords here: beta for servo 0 (steering) and gamma for servo 1 (motor control). Type **node app.js beta=300**. You should see the front wheels turn. Now the numbers need experimenting with. On our example, 340 was left, 400 was centre and 470 was right. Do the same for the motor by typing **node app.js gamma=400** and take note of the various limits of your car.

06 Configure sensible defaults

Now you know what your car is capable of, we can set the defaults in app.js and socket.html. Edit app.js and find the section that says 'function emergencyStop'. Adjust the two numbers to your car's rest values. Then open socket.html and adjust the predefined values under 'Define your variables here'.

07 Going for a spin

We're almost ready to try it out, but you need to know the IP address of your Pi car, so type **ifconfig** at the terminal. Then fire up the app by typing **node app.js**. Now grab the nearest smartphone or tablet, making sure it's on the same network as your Pi. Open the web browser and go to **http://[your IP address]:8080/socket.html**. You should get an alert message saying 'ready' and as soon as you hit OK, the gyro data from your phone will be sent to the car and you're off!

We'll harness the natural movement of your hand and wirelessly drive the vehicle

Full code listing

socket.html

```
<html>
<head>
<script src="jquery-2.0.3.min.js" language="javascript"></
script>
<script src="/socket.io/socket.io.js"></script>
<meta name="viewport" content="user-scalable=no, initial-
scale=1.0, maximum-scale=1.0;" />
<script>

//------ Define your variables here
var socket = io.connect(window.location.hostname+':8080');
var centerbeta = 400; //where is the middle?
var minbeta = '340'; //right limit
var maxbeta = '470'; //left limit
var multbeta = 3; //factor to multiply the raw gyro figure ↵
by to get the desired range of steering
var centergamma = 330;
var ajustmentgamma = 70; //what do we do to the angle to get
to 0?
var mingamma = 250; //backwards limit
var maxgamma = 400; //forward limit
var multgamma = 1; //factor to multiply the raw gyro figure ↵
by to get the desired rate of acceleration
window.lastbeta='0';
window.lastgamma='0';

$(function(){
  window.gyro = 'ready';
  alert('Ready -- Lets race !');
});
window.ondeviceorientation = function(event) {
  beta = centerbeta+(Math.round(event.beta*-1)*multbeta);
  if (beta >= maxbeta) {
    beta=maxbeta;
  }
  if (beta <= minbeta) {
    beta=minbeta;
  }
gamma = event.gamma;
  gamma = ((Math.round(event.gamma)+ajustmentgamma)* ↵
multgamma)+ centergamma;
//stop sending the same command more than once
send = 'N';
if (window.lastbeta != beta) { send = 'Y' }
if (window.lastgamma != gamma) { send = 'Y' }
window.lastbeta=beta;
window.lastgamma=gamma;
if (window.gyro == 'ready' && send=='Y')  { //don't send ↵
another command until ready...
  window.gyro = 'notready';
    socket.emit('fromclient', { beta: beta, gamma: gamma } );
    window.gyro = 'ready'; }}
```

app.js

```
//declare required modules
var app = require('http').createServer(handler)
  , io = require('socket.io').listen(app)
  , fs = require('fs')
  , static = require('node-static')
  , sys = require('sys')
  , PwmDriver = require('adafruit-i2c-pwm-driver')
  , sleep = require('sleep')
  , argv = require('optimist').argv;
  app.listen(8080);

//set the address and device name of the ↵
breakout board
pwm = new PwmDriver(0x40,'/dev/i2c-0');

//set pulse widths
setServoPulse = function(channel, pulse) {
  var pulseLength;
  pulseLength = 1000000;
  pulseLength /= 60;
  print("%d us per period" % pulseLength);
  pulseLength /= 4096;
  print("%d us per bit" % pulseLength);
  pulse *= 1000;
  pulse /= pulseLength;
  return pwm.setPWM(channel, 0, pulse);
};
//set pulse frequency
pwm.setPWMFreq(60);
//Make a web server on port 8080
var file = new(static.Server)();
function handler(request, response) {
  console.log('serving file',request.url)
  file.serve(request, response);
};
console.log('Pi Car we server listening on port 8080 visit ↵
http://ipaddress:8080/socket.html');
lastAction = "";
function emergencyStop(){
  pwm.setPWM(0, 0, 400); //center front wheels
  pwm.setPWM(1, 0, 330); //stop motor
  console.log('###EMERGENCY STOP - signal lost or shutting ↵
down');
}
if (argv.beta) {
  console.log("\nPerforming one off servo position move ↵
to: "+argv.beta);
  pwm.setPWM(0, 0, argv.beta); //using direct i2c pwm module
  pwm.stop();
  return process.exit();
}
if (argv.gamma) {
  console.log("\nPerforming one off servo position move ↵
to: "+argv.gamma);
  pwm.setPWM(1, 0, argv.gamma); //using direct i2c pwm module
  pwm.stop();
  return process.exit();
}
//fire up a web socket server
io.sockets.on('connection', function (socket) {
  socket.on('fromclient', function (data) {
  console.log("Beta: "+data.beta+" Gamma: "+data.gamma);
  //exec("echo 'sa "+data+"' > /dev/ttyAMA0", puts); //using ↵
http://electronics.chroma.se/rpisb.php
  //exec("picar.py 0 "+data.beta, puts); //using python ↵
adafruit module
  pwm.setPWM(0, 0, data.beta); //using direct i2c pwm module
  pwm.setPWM(1, 0, data.gamma); //using direct i2c pwm module
  clearInterval(lastAction); //stop emergency stop timer
  lastAction = setInterval(emergencyStop,1000); //set ↵
emergency stop timer for 1 second
  });
});
process.on('SIGINT', function() {
  emergencyStop();
  console.log("\nGracefully shutting down from SIGINT ↵
(Ctrl-C)");
  pwm.stop();
  return process.exit();
});
```

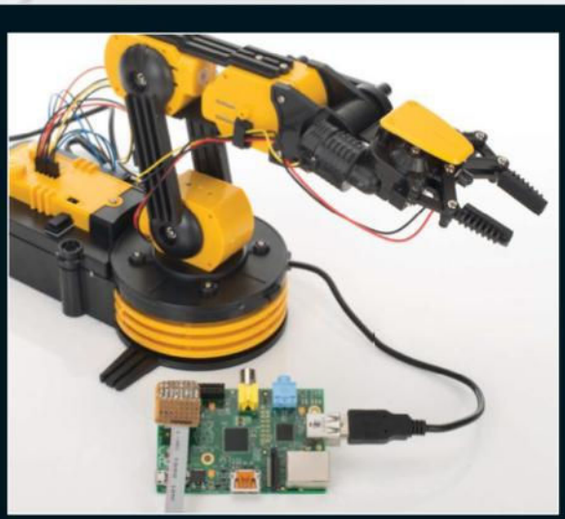
MPU-6050 Containing a MEMS accelerometer and a MEMS gyroscope, this sensor reads the x, y and z axis channels with 16-bit ADC conversion

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

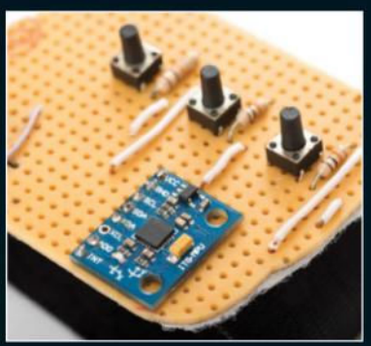
Components list

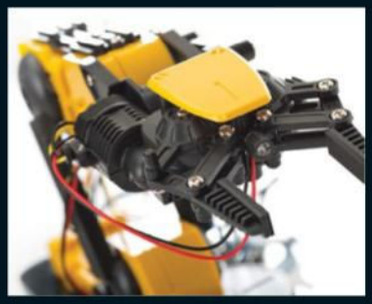
- Raspberry Pi Model B
- Maplin Robotic Arm Kit With USB PC Interface
- MPU-6050 Six-Axis Gyro and Accelerometer
- 3 Mini Push Button Switches
- Veroboard
- Velcro strap
- 1m Ribbon Cable



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 is 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**).

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

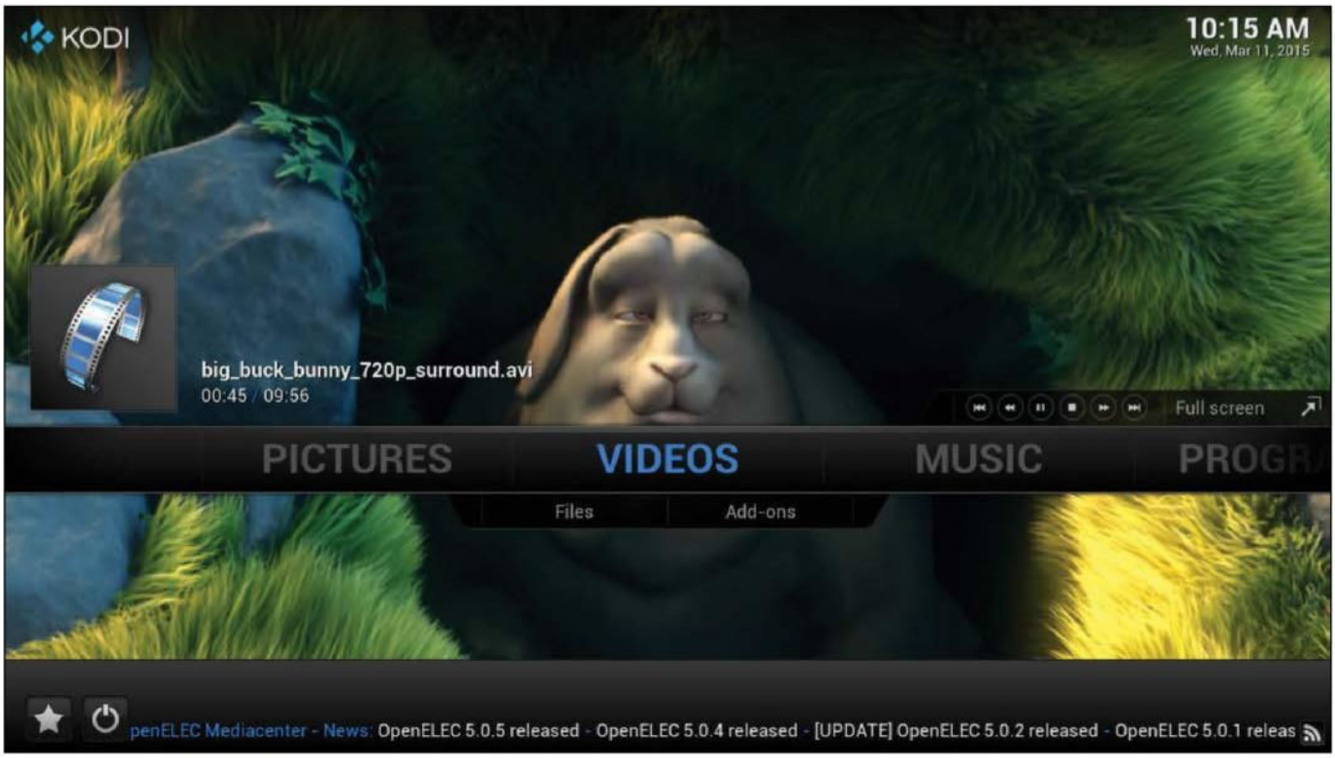
In the past, Pi HTPCs were just a choice between RaspBMC and OpenELEC. However, RaspBMC is on a bit of a hiatus and OpenELEC is your best bet for getting the most up-to-date software. There's not a massive difference between the two, as they both run XBMC.

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:

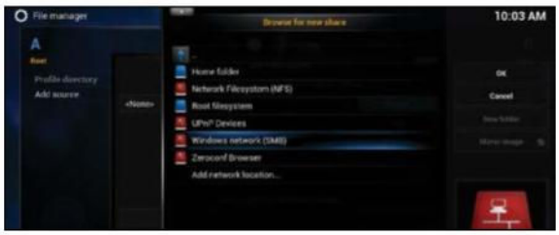
```
$ dd bs=1M if=OpenELEC-RPi2.arm-5.0.5.img of=/dev/
mmcblk0
```

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.



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.

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**

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**

Make a tweeting wireless flood sensor

Flood-proof your basement in just 19 lines of code, or easily tweak the project to create your own personalised alarm system...

Flooding saw hundreds of homes right across the world underwater this year, and many would have benefited from having just that little bit extra warning. In order to be better prepared for floods, we're going to show you how you can prototype your own wireless flood sensor in less than ten minutes. Building it might give you just enough warning to dash home from work, move valuable items upstairs and take the lawnmower, caravan and motorbike to higher ground. Handily, it can also be used to detect toilet flushes, water butt levels or any liquid level rise or fall at all – so it's not just something fun to try out, it's practical too!

Sending tweets

Sending a tweet used to be really easy, if a little on the insecure side. These days you need to register an application with your Twitter account – you do have one, don't you? If not, go create one at **www.twitter.com**. At first this project can look a little daunting, however it can be done painlessly in less than five minutes, if you follow these steps closely!

What you'll need

- **Ciseco** Raspberry Pi Wireless Inventors Kit
  shop.ciseco.co.uk/raswik
- Float sensor
  shop.ciseco.co.uk/float-switch
- DC power supply between 6v and 12v

01 Link Twitter to mobile

Make sure your Twitter account has a mobile phone number associated with it. In your main Twitter account, click the gears icon at the top-right and then 'Mobile' in the list. At this stage, just follow the instructions on screen.

02 Set it all up

With your Twitter username and password, sign in to **https://apps.twitter.com** and click on the button 'Create an application'. In the name field we suggest you use your Twitter account name, add a space and then write 'test'. For the description, just put 'Test poster for Python'. Finally, for the website you can put anything you like. For example, http://www.mywebsite.com – but it's important you don't forget the 'http://'.

03 Enable reading and writing

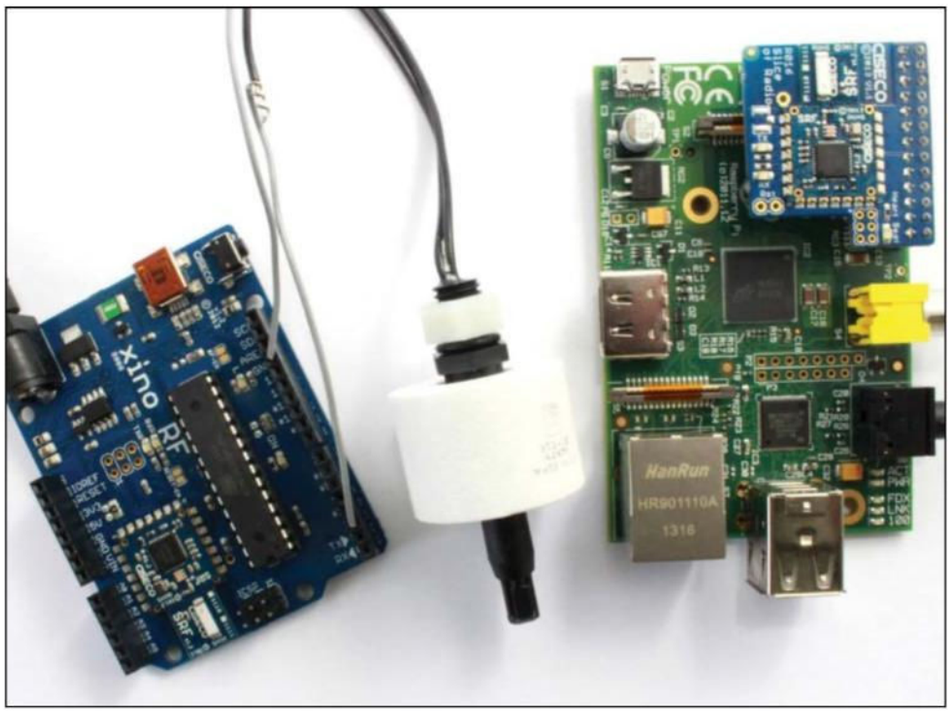
Since you want to be able to send tweets, click on the 'Settings' tab, change to 'Read and Write' and then click 'Update'. This might take a few seconds.

04 Generate codes

Now go back to the 'Details' tab. You will see that an 'API key' and 'API secret' are visible, and that there's a 'Create my access token' button. Click that button to obtain all four of the codes you'll need. If you did this before Step 2, or it still says 'Read', all you have to do is click the button to recreate these codes. It really is straightforward.

05 Remember the codes

Earlier on 'API' was called 'consumer', and you might have come across this before in examples on the web. We suggest copying the following essentials into Notepad so they don't get lost: API key, API secret, Access token and the Access token secret.



Right The Wireless Inventors Kit enables you to connect a Raspberry Pi to an Arduino module from the other side of your house

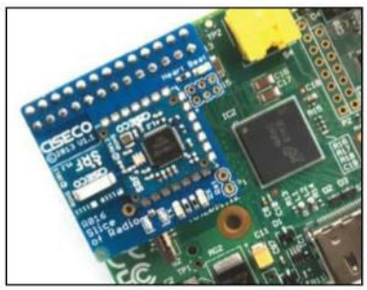
Tweepy is an easy-to-use Python library that works great for accessing the Twitter API

No RasWIK?

Not to worry, using different hardware is always a possibility when playing around with the Raspberry Pi. The reason we chose to use the RasWIK is simply because everything except the float switch is in the box and preloaded with software, making it much easier to get up and running quickly. As a bonus addition, this software is also available to download for free.

To build this with a conventional Arduino or clone, you'll need a USB cable and to leave it 'wired', or use serial-capable radio modules such as the Xbee or APC220. We are, after all, only sending and receiving plain text.

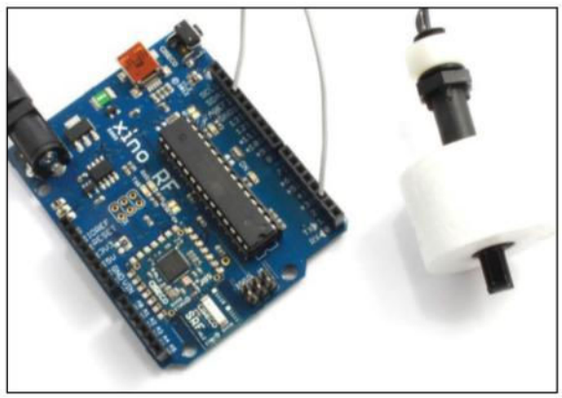
The Arduino sketch used can be downloaded from **http://github.com/CisecoPlc/LLAPSerial**, while the SD image for the OS we used is based on a stock version of Wheezy, which can be downloaded from **http://bit.ly/SfhLLI**.



01 Start simple
To get going with your flood sensor, simply plug in the Slice of Radio to your Pi and insert the preconfigured Raspbian operating system.

02 Go to LX terminal
Power up the Raspberry Pi, log in and type **STARTX** to start the desktop. Double-click the LX Terminal and type the following into the black window:

```
minicom -D /dev/ttyAMA0 -b 9600
```



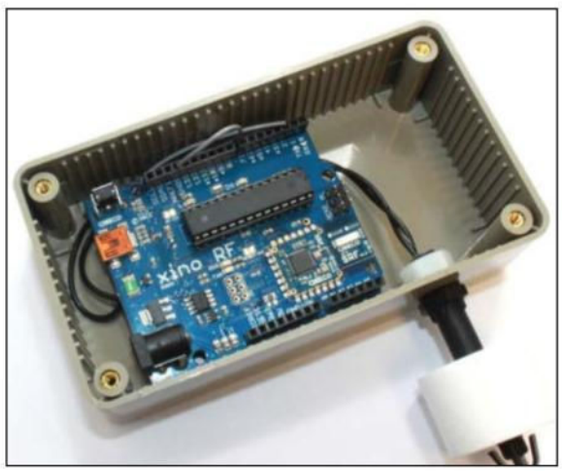
03 Make the connection
Connect the float switch to the XinoRF ground pin (marked GND) and digital I/O pin2. Then, power up the XinoRF (you will see a--STARTED– displayed in minicom)

04 Test the sensor sends messages
Wiggle the sensor up and down (you should get a--D02HIGH– when the sensor position is up) and see the RXR LED on the XinoRF flicker with each message sent.

05 Install Tweepy
Tweepy is an easy-to-use Python library that works great for accessing the Twitter API. For more information or to check out the documentation, visit **https://pypi.python.org/pypi/tweepy/2.2**. Type in a shell window the following:

```
sudo pip install tweepy
```

06 Put the sensor to work
Test your prototype using a regular saucepan of water. If you want to put your flood sensor to real use then place it into a waterproof box and ensure it is mounted securely.



Full code listing

```
#  Let's set up how to talk to Twitter first

import tweepy, serial, time
API_key = "GWip8DF9yf0RYzjIlM6zUg"
API_secret = "Yjipapo56nhkS2SAWtx3M4PAit3HsvWZUYAOghcLn4"
Access_token = "2392807040-19lSoaVOmj8NTvJVteU8x265IPEw2GfY0cS7vuN"
Access_token_secret = "lV4u2ls4oeRZCAxcpaPQNzRDN0lSiUibY0MdCuYKk16Rl"
auth = tweepy.OAuthHandler(API_key, API_secret)
auth.set_access_token(Access_token, Access_token_secret)
api = tweepy.API(auth)

#  Open a com port (yours may differ slightly)

ser = serial.Serial('COM3', 9600)

#  An endless loop checking for sensor alerts

while True:
    SerialInput = ser.read(12)
    if SerialInput == 'a--D02HIGH--':
        TimeNow = datetime.datetime.now()
        DS = TimeNow.strftime('%H:%M:%S %d-%b-%Y')
        AlertText = 'ALERT: LLAP+ device -- flood sensor triggered @ ' + DS
        print (AlertText)
        api.update_status(AlertText)
        time.sleep(10) #stop fast re-triggering
        ser.flushInput()
```

Build a Raspberry Pi-powered Bigtrak

Take a toy, a Raspberry Pi and a PS3 controller; add a dash of Python and some solder for the perfect remote-controlled gadget…

What you'll need

- Bigtrak
  www.bigtrakxtr.co.uk/shop/bigtrak
- Breadboard and cables
- Motor driver bit.ly/1iOnFug
- USB Battery pack
  amzn.to/1h2PBil
- PS3 DualShock controller

The Raspberry Pi is a small, low-cost computer designed to promote an interest in computing and programming – but it doesn't have to be straight-laced computing. In fact, in this article we'll be showing you how you can use it to turn a Bigtrak into a robot. That's educational, right?

The Bigtrak is a toy that takes in a list of straightforward commands (Go forwards, turn left, turn right) and then executes them. To make things more interesting we're going to remove the existing circuitry and replace it with a Raspberry Pi, using a small motor driver to safely control the motors in the Bigtrak, which we'll then set up to be controlled via a PlayStation 3 DualShock controller.

Everything required on the software side comes pre-installed on the latest Raspbian OS images, so all we need to translate changes from the controller to the motors is a small Python script that uses the Pygame and RPI.GPIO modules.

01 Opening up the Bigtrak – the easy bit

Before we can make any changes to the Bigtrak we need to get inside. First, flip the Bigtrak upside down and remove the nine screws from around the edge. These are mostly easy to get at, however the ones on the front may require a more slender screwdriver to reach them.

02 Opening up the Bigtrak – the fiddly bit

The last two screws are located underneath the grey grille on the back. This grille is held in place by four plastic tabs that need to be pushed in while at the same time sliding the grille away from the Bigtrak. This can be quite tricky as there is limited space to get extra hands in to help out. It can help to wedge some thin plastic items (eg a guitar pick) into the sides to keep those two tabs unlocked, while using your fingers to push in the bottom two tabs and slide the grille upwards, allowing you to remove the screws.

03 Removing the top

Put the Bigtrak back onto its wheels then carefully loosen the top and lift upwards. The lid is connected to the base with a ribbon cable and a switch, so only pull the top up far enough for you to tilt it to one side and expose the inside.

With the lid lifted up onto one edge, remove the screw holding the switch in place and detach it from the lid. Next, you need to unscrew the two screws on the PCB that hold the ribbon cable in place and let it slip free.

With the switch and ribbon cable disconnected, the lid should now come free and can finally be completely removed from the base of the Bigtrak.

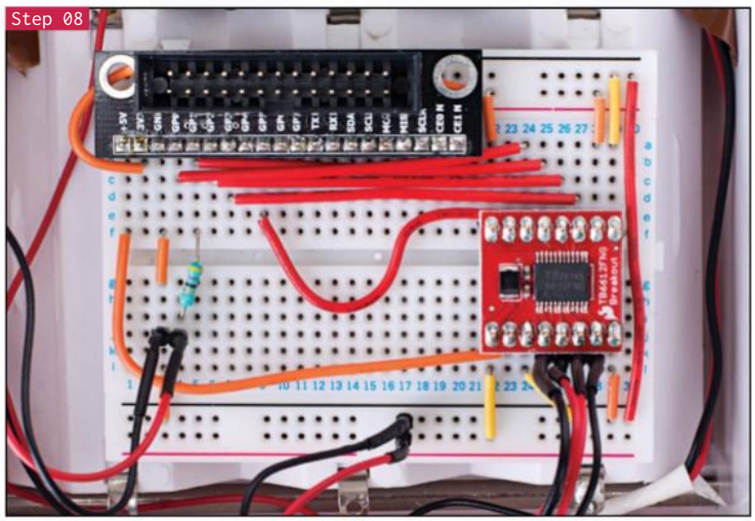
04 Cut the wires

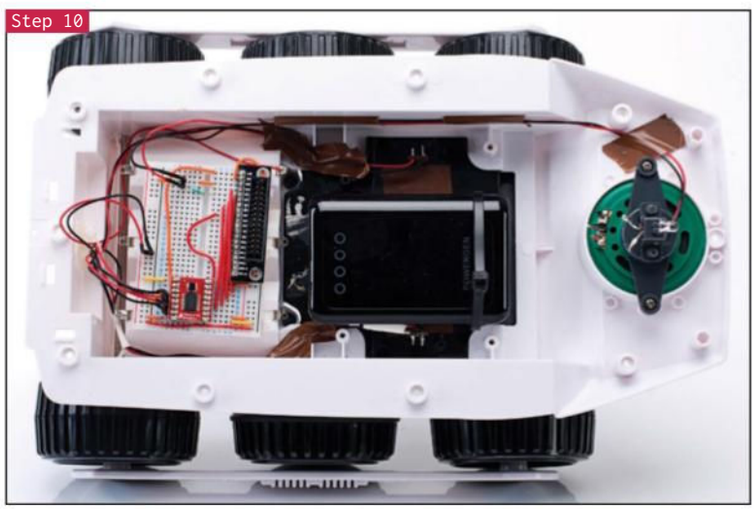
Cut the wires leading to the main PCB. The ones for the switch and power should be cut close to the PCB (so we can reuse them later) whereas the ones to the LED and speaker can be cut wherever you like.

05 Remove the engine

Turn the Bigtrak upside down and remove the four screws holding the engine in place (this will reduce the chance of soldering iron damage to the main body). Carefully turn the Bigtrak back over and lift it up until the engine slips free.

> The wires need to be long enough to reach the back of the Bigtrak, so be generous!





06 Rewire the motor

Remove the solder connecting the PCB to the motors (a solder mop is useful here) and then remove the PCB. With the PCB removed we can now attach wires to the motors in order to drive them from the Raspberry Pi, as opposed to the on-body commands. The wires will need to be long enough to reach the back of the Bigtrak, so be generous – after all, it's far easier to trim long wires to length than replace short wires entirely!

Having installed all of the wires, you can now replace the engine back into the Bigtrak.

07 Connect the motor driver

With the motors back in place we now need to build up a circuit to drive it from the Raspberry Pi. We've used a ribbon cable to connect the GPIO pins on the Raspberry Pi to a breadboard, before connecting it up to a Dual Motor Driver (**http://proto-pic.co.uk/motor-driver-1a-dual-tb6612fng**) to actually drive the motors. This keeps the higher voltage the motors require away from the sensitive GPIO pins.

The connections made on the breadboard are listed in the table below. These values will be needed when writing the software and may be different depending on the breakout board you are using, and the Raspberry Pi revision.

RPi GIPO	Motor Driver
24	AIN2
17	AIN1
18	STBY
21	BIN1
22	BIN2

With the PWMA and PWMB pins directly connected to the 3.3V power rail, the motors will now always run at full speed for as long as they're active.

08 Install the breadboard

The breadboard is going to be installed on top of the battery compartment inside the Bigtrak, so the wires from the motors should be brought to the back to the unit and cable-tied into place. The wires to the batteries can also be brought back to the same place to help keep things tidy.

09 Wire it all together

In order to easily connect the motors and batteries to the breadboard we have soldered some modular connector plugs to the ends of the cable, allowing them to just push into place (these are available from **www.maplin.co.uk/modular-connectors-5348**).

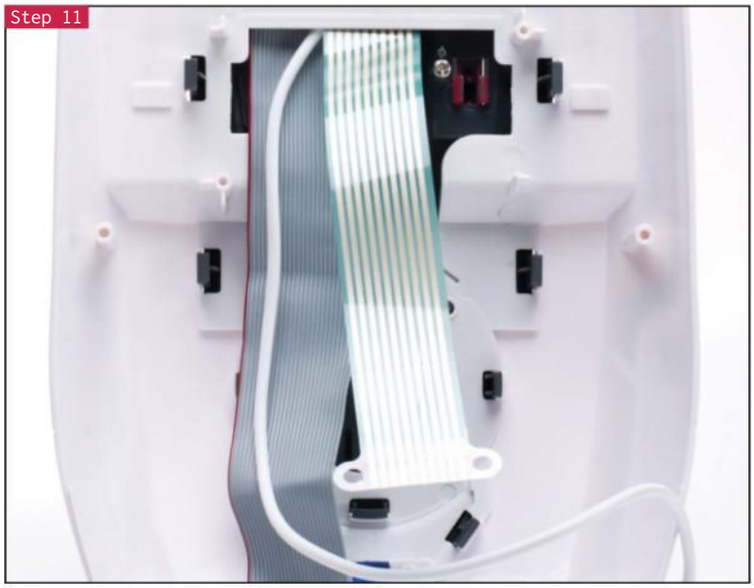
With the breadboard installed (sticking it into place for support) we can now, after double-checking all the connections, plug the motors and power into it. To know when the motors are enabled (and reduce the chance of unexpected movement), the LED can be connected to the breadboard so that it lights up whenever the 'standby' pin is active, using a resistor to ensure it doesn't pull too much current and go 'pop'.

10 Provide power

Power for the Raspberry Pi is supplied via a USB battery pack that is installed on top of the engine and can be held in place by a couple of cable ties or a Velcro strip. This type of battery is typically sold as a portable mobile phone or iPad charger – the one used here is rated at 8000mAh, able to power the Raspberry Pi for up to eight hours.

11 Connect to the Raspberry Pi – adding cables

As the Raspberry Pi will be mounted on the top of the Bigtrak, we need to run the ribbon and power cable through the top of the case. To do this, turn the top of the Bigtrak upside down and release the front two catches that hold the dark grey plastic in place – this provides a big enough gap to feed the ribbon cable and USB power cable through. Make sure that the red edge of the ribbon cable correctly matches up with the connector on the breadboard to save yourself from having to twist the cable inside the case.



12 Connect to the Raspberry Pi – final steps

With the top of the Bigtrak back on, the Raspberry Pi can now be put in place, keeping the GPIO pins towards the front to allow the ribbon cable to easily connect. As for the battery pack, we're holding it in place with cable ties and sticky pads. In theory it's possible to attach the bare Raspberry Pi to the Bigtrak, however this can cause the SD card to press against the edge and bend, so it's recommended to use a case to protect the Raspberry Pi.

Connect the ribbon and power cable to the Raspberry Pi, turn it on and it's now ready to be told what to do. For setting up the software it may be easier to connect up a keyboard and monitor to the Raspberry Pi at this point.

13 Connect the PS3 controller

This should be a simple case of plugging the PS3 controller into one of the USB ports, as all the software to support it is included in the default Raspbian OS image and it will be automatically detected as a joystick. To confirm that the PS3 controller has been detected, run **lsusb** and checked that it appears in the resulting list.

14 Run the software

Now with the system all set up, it should just be a simple case of copying the 'bigtrak.py' file found on this issue's disc onto your Raspberry Pi and running it. As the script accesses the GPIO pins, it will need to be run as the superuser, so launch it using:

```
sudo python bigtrak.py
```

Now we can control the Bigtrak using the analogue sticks! Moving the left stick will control the left motor and moving the right stick will control the right. So, to move forwards push both sticks up, pull both down to go backwards and push one up and one down to rotate on the spot.

If the analogue sticks are not controlling the Bigtrak as expected, double-check the GPIO connections to make sure that they are all as expected.

15 Next steps

Now that you have a solid base for your Raspberry Pi robot, you can make further adjustments to it. Possible next steps could be: add a USB Bluetooth adaptor so the PS3 controller can be connected wirelessly; replace the breadboard with a PiPlate or 'Slice of Pi' add-on board, allowing the Raspberry Pi to be installed inside the Bigtrak; connect up the RaspberryPi camera and a USB WiFi adaptor to stream video as you drive around; or add a robot arm!

Build your own networked Hi-Fi with a Pi Zero

Put the Pimoroni pHAT DAC together with a Pi Zero to create a networked Hi-Fi

We will show you how to create a high-quality networked music player that takes advantage of the UK's online radio stations, Linux's popular Music Player Daemon, and a responsive web-server to control it all. The full-sized Raspberry Pis have two built-in audio outputs: audio over HDMI cable and a 3.5mm headphone jack that can suffer interference and noise. The Pi Zero itself has no audio jacks but Pimoroni has come to the rescue and built a high-quality DAC (digital audio converter) using the same chip as the Hi-Fi berry (PCM5102A).

> The Pi Zero itself has no audio jacks but Pimoroni has come to the rescue

01 Soldering the headers

The pHAT DAC comes with a 40-pin header, which you will need to solder. We consider a flux pen, work-lamp and thin gauge 60/40 solder essential for this. An optional RCA jack can also be bought to give a phono-lead output for older stereos.

02 Install drivers

The DAC relies on I2C, so we have to load some additional kernel modules. If you are running Raspbian then you can type in the following for a one-script installation over secure HTTP:

```
curl -sS https://get.pimoroni.com/phatdac | bash
```

While HTTPS provides a secure download, curious types may want to review the script before running it.

03 Installing Music Player Daemon (MPD)

Now install the MPD package and enable it to start on boot. MPD will be the backbone of the project providing playback of MP3s and internet radio stations. The MPC (client) software is also installed for debugging and setting up initial playlists.

```
sudo apt-get install mpd mpc
sudo systemctl enable mpd
```

04 Clone and install pyPlaylist web-server

pyPlaylist is a responsive (mobile-ready) web-server written with Python & Flask web framework. Once it has been configured, it will give us a way of controlling our networked Hi-Fi through a web-browser. The following code on the next page will install pyPlaylist on Raspbian:

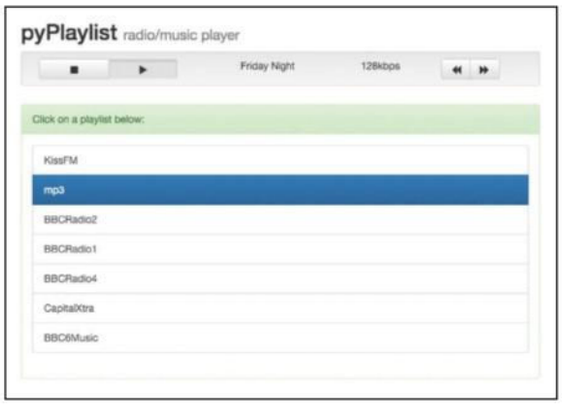
What you'll need

- **Github repository** (http://github.com/alexellis/pyPlaylist)
- **pimoroni pHAT DAC £10-12** (pimoroni.com)
- **Soldering iron, flux & solder**

Auto-starting on Raspbian

In Raspbian/Jessie the controversial systemd software was added, giving a highly modular way of managing start-up scripts amongst other things. While systemd configuration files are now best practice, they can take time to fully understand. For that reason we would suggest using cron to start the script on reboot as a temporary measure.

```
crontab -e
@reboot /usr/bin/python /home/pi/pyPlaylist/app.py
```




```
sudo pip install flask python-mpd2
cd ~
git clone https://github.com/alexellis/pyPlaylist
cd pyPlaylist
./raspbian_install.sh
```

05 Choosing the radio stations

We have put together a list of popular radio stations in the UK which can be run into MPD with the add_stations.sh file. Edit this file or find your own from **http://radiofeeds.co.uk**.

```
cd ~/pyPlaylist
./add_stations.sh
```

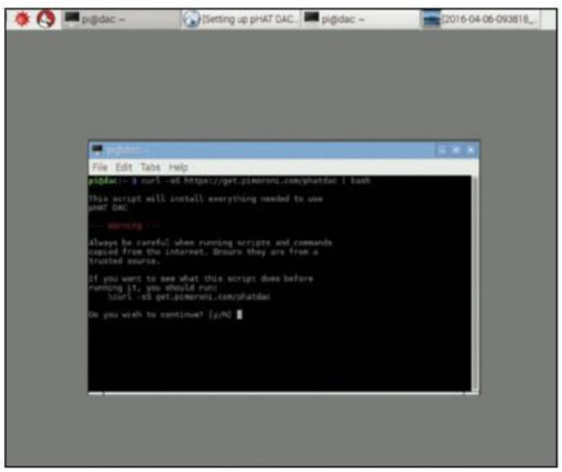
06 Reviewing the stations

Each station is added into its own playlist – the **mpc ls** command shows which playlists are available:

```
$ mpc ls
BBC6Music
BBCRadio1
BBCRadio2
BBCRadio4
CapitalXtra
KissFM
```

If you want to remove one of the stations then type in the following:

```
mpc rm BBC6Music
```



07 Starting the web-server

Now that we have some stations, we can run the web-server from the pyPlaylist directory. Then open up a web browser to start playing a radio station. The following command reveals your IP address on Raspbian:

```
$ ./raspbian_get_ip.sh
192.168.0.20
```

Once you know the IP address, connect to the URL in a web-browser on port 5000, i.e.

```
http://192.168.0.20:5000/
```

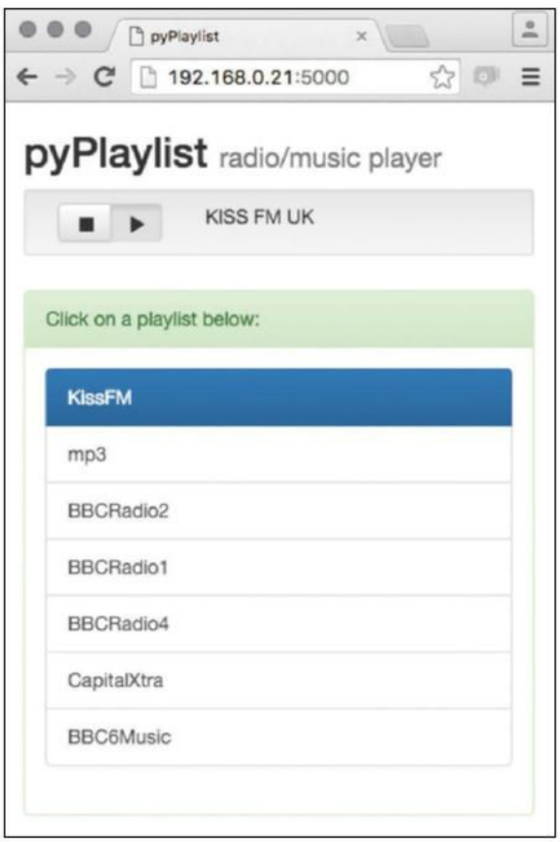
08 Add a custom music playlist

Now put together a sub-directory with your music files under /var/lib/mpd/music/ and ensure that mpd:audio has access to read it. Then we: update mpd's database, clear out the current playlist and add in all the tracks from the new directory (ambient) finally saving it as a new playlist.

```
mpc update
mpc clear
mpc ls ambient | mpc add
mpc save ambient
```

09 Finishing up

Now your music player is functioning, all that's left to do is to add some speakers, obviously! Almost anything with a RCA or 3.5mm input source will work for this purpose. That part we will leave up to you. To take a look at the code here in full, check out **github.com/alexellis/pyPlaylist**. Enjoy the tunes!



pyPlaylist project

We wrote pyPlaylist with the Python flask framework which is an ideal starting-point for simple RESTful websites. The front-end code saves the screen from completely reloading by using jQuery to update the song or radio information. Bootstrap has been employed to make the pages responsive (compatible with your PC, phone and tablet). The code has been released under GPL, so why not fork the code and tweak it to your own needs?

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

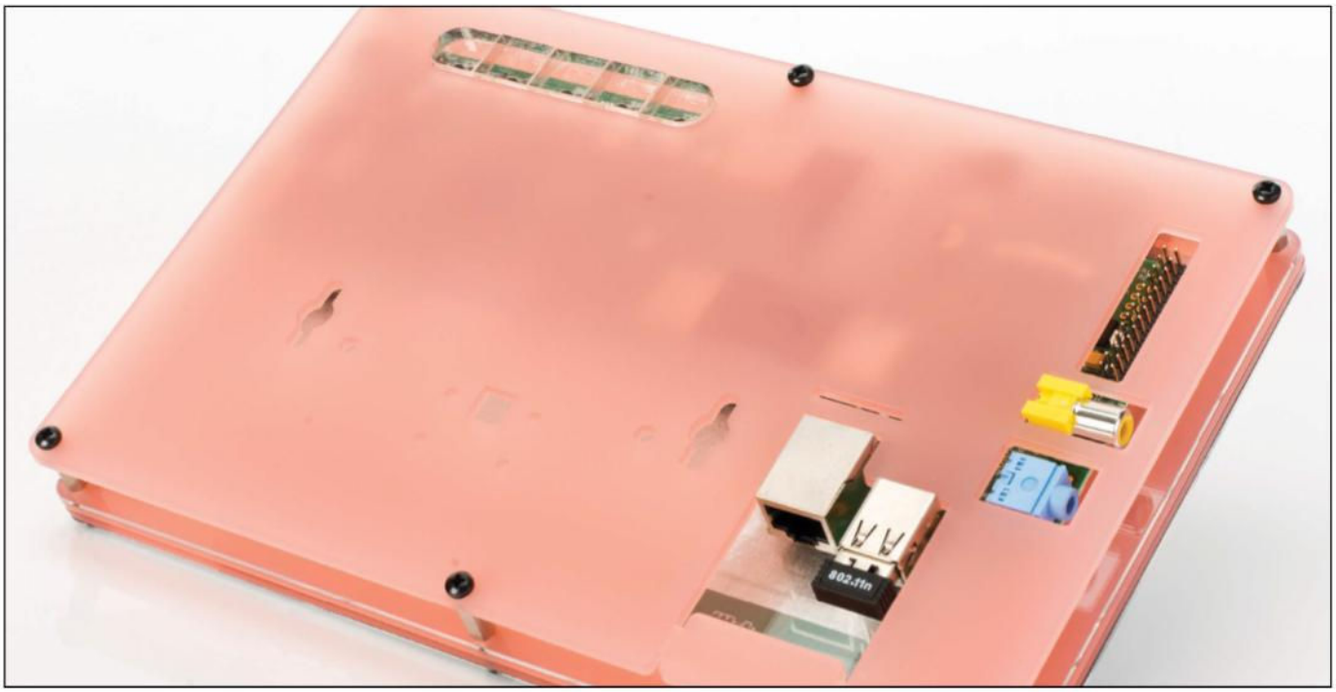
Digital signage is a huge market in modern times, and increasingly we are seeing digital advertising in the public space – be it on street billboards, shopping centres and even in some city taxis. Screenly is an exciting piece of software from Wireload Inc that has set out to reduce the barriers to entry 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.

03 Download Screenly OSE

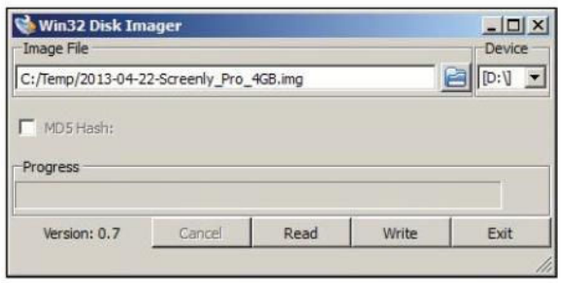
Now that you have the hardware all ready to go we need to download the Screenly OSE SD card image. This is only a 3.7GB image file, however it may not fit on some 4GB SD cards so we would recommend a minimum of 8GB, for extra space for all your pictures and videos. Visit **bit.ly/1wLKIRQ** and download 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.

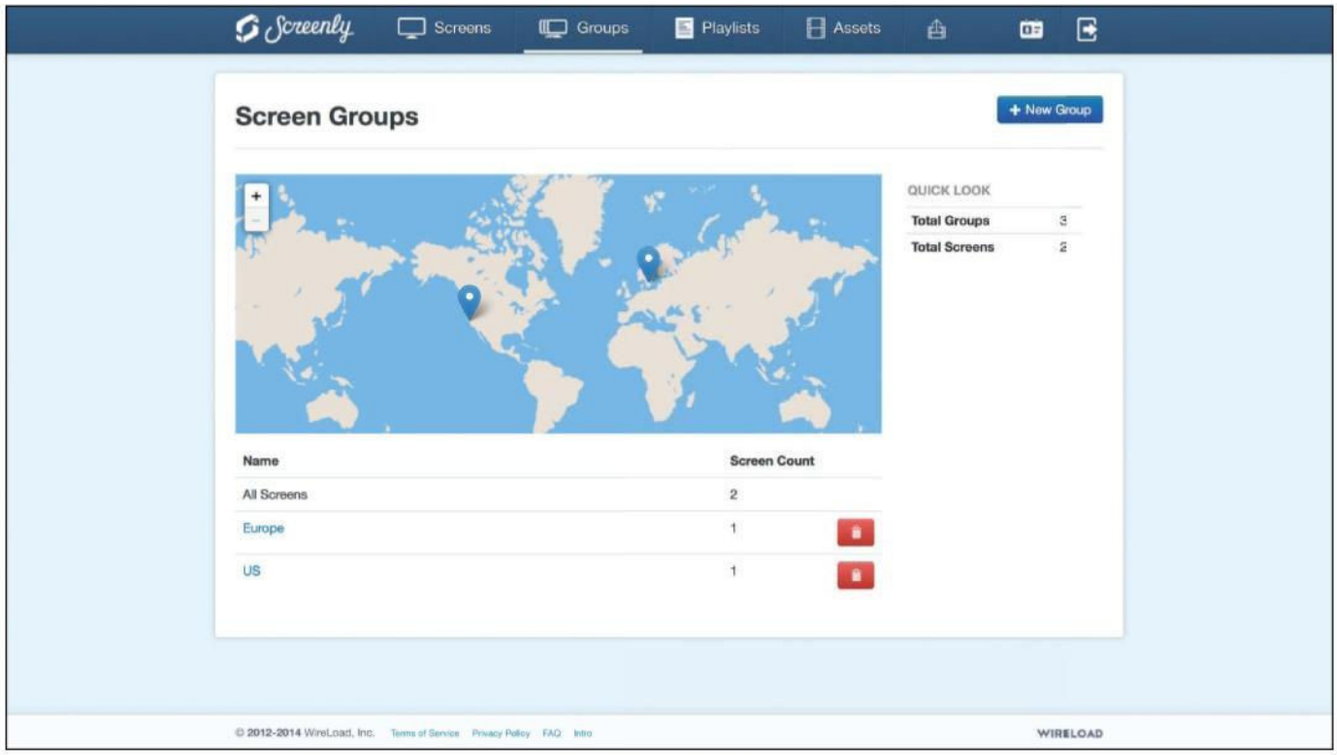
It is a good idea to connect your Wi-Fi dongle and keyboard at this point. Take a look at the image at the top of this page to see where the slots are in relation to the casing. A wired network is also required for the initial setup and for configuring the Wi-Fi connection.

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

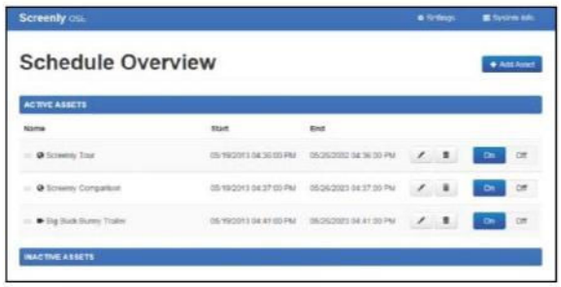
HDMIPi is a collaboration between Cyntech and Alex Eames from RasPi.TV. They wanted to bring a cheap HD resolution HDMI screen to the market that will reduce the cost of having a dedicated monitor for your Raspberry Pi. They took to Kickstarter to launch their idea (**kck.st/17zyaQg**) and there was 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!

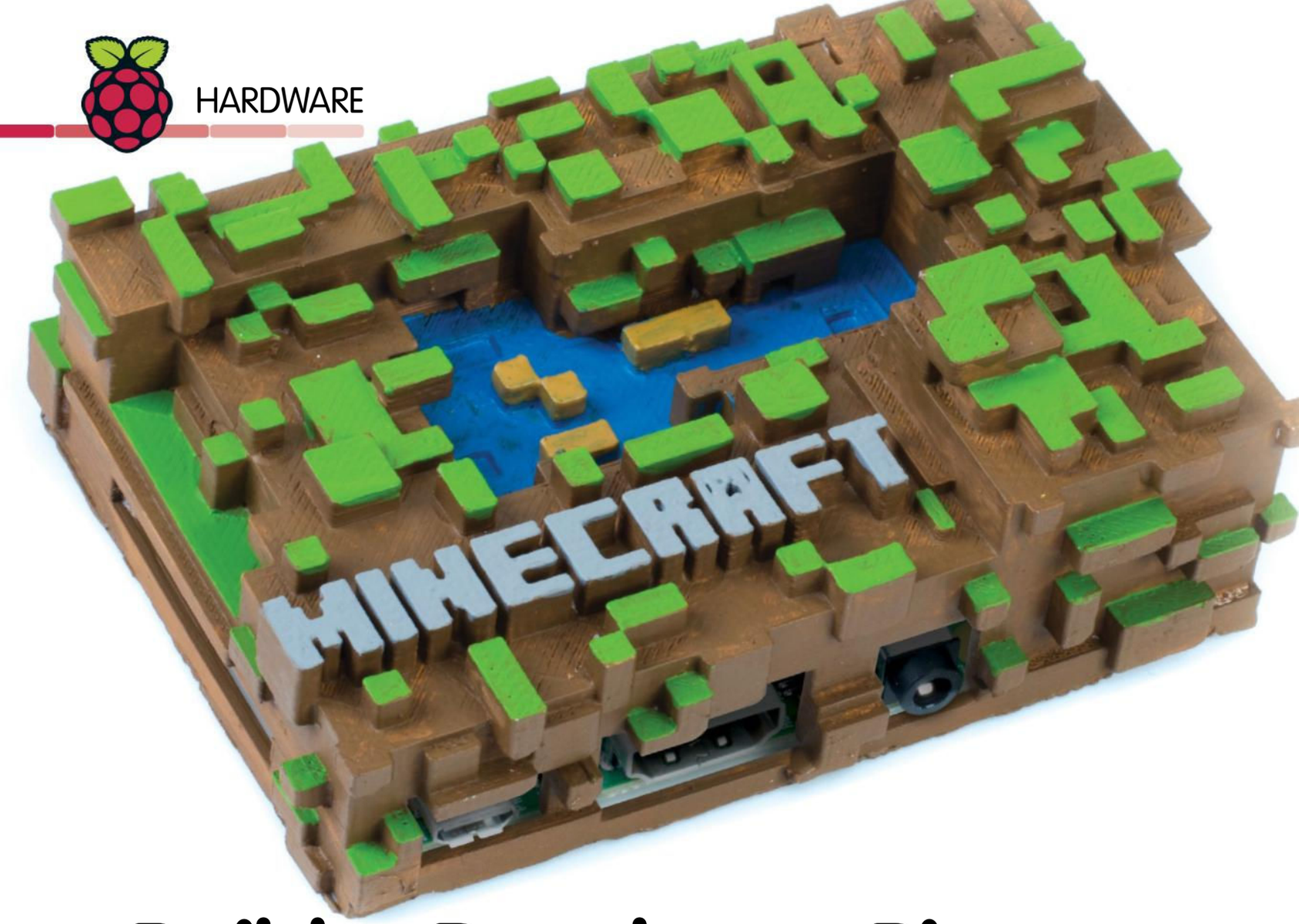
Once you have got Screenly all set up and loaded all of your favourite pictures and videos onto it via the web interface, it is now time to enjoy the fruits of your labour! Mould the spider stand (if you have one) into shape by taking the middle two legs at the top and bending them downwards and backwards. 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 Raspberry Pi Minecraft console

Create a full-functional, Pi-powered games console that you can play Minecraft on and learn how to program too

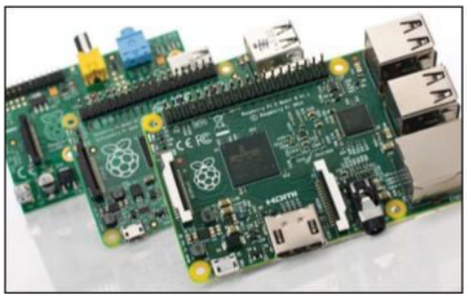
***Minecraft* means many things to many people, and to Raspberry Pi users it's supposed to mean education.** Not everyone knows, though, that you can still have fun and play *Minecraft* as you normally would.

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.

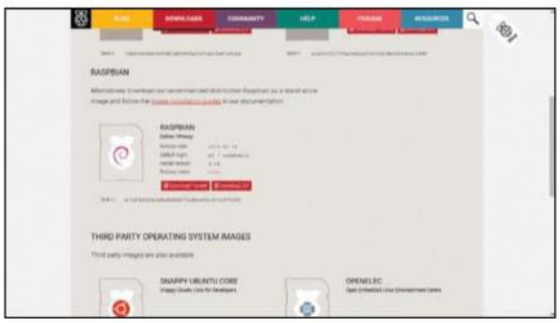
What you'll need

- 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()
```

Visualise music in Minecraft with the PianoHAT

Combine code, Minecraft and the PianoHAT to play music and create a visualisation of the melody

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.

The Raspberry Pi was designed to provide several ways to interact with the world through sensors and activators. In the past, we have looked at using the GPIO interface pins to communicate with several devices at once. This is not the only way to work with the world at large, however. This month, we will look at one of the other mechanisms available, the I2C bus. I2C (Inter-Integrated Circuit) bus was invented by Philips Semiconductor, with version 1 having come out in 1992. The design is for short connection paths, and supports multiple masters and multiple slaves where messages on the bus are delivered using device addresses. Messages have a START section and a STOP section, wrapped around the core of the message. The three types of messages that you can send are a single message where a master writes data to a slave, a single message where a master reads data from a slave, or a combined message where a master sends at least two read or two write messages to one or more slaves. Now that we have a little bit of an idea of what the I2C bus is, how can you use it with your Raspberry Pi? The first step is to activate the bus within the Linux kernel. By default, the relevant kernel modules are blacklisted and not loaded at boot time. If you are using a newer version of Raspbian, you can use the utility 'sudo raspi-config' and select the 'Advanced Options' section to set correct options. If you are using an older version or simply wish to make the changes manually, it is a bit more complex. In order to change this, you will need to edit the file '/etc/modprobe.d/raspi-blacklist.

What you'll need

- Raspbian set to command line
- RaspCTL

01 Getting started

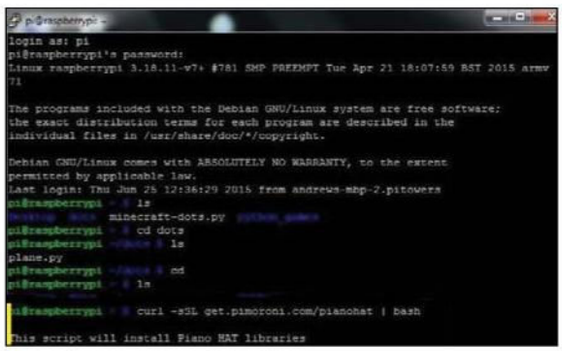
Pimoroni has made it extremely easy to install the software for your PianoHAT. Assuming you have not connected your HAT, simply attach the board and boot up your Raspberry Pi. Load the LX Terminal and update the software; type:

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

Type the following line to install the PianoHat libraries:

```
$ sudo curl -sSL get.pimoroni.com/pianohat | bash
```

Follow the instructions displayed. This will now download the required libraries and a selection of programs to try.



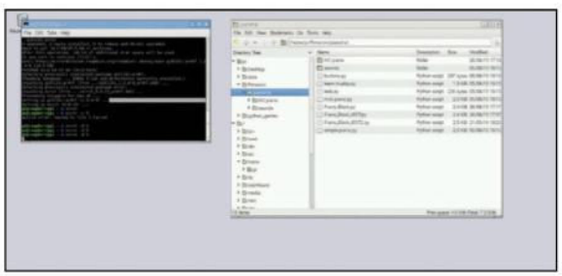
02 Basic events

The software install comes with a set of four example programs to get you started and demonstrate the features and functions of the PianoHAT. In terms of the code for the Piano, there are four basic events that you can control, these are:

on_note – triggers when a piano key is touched and plays a note. on_octave_up – triggers when the Octave Up key is touched and raises the notes by one octave.

on_octave_down – triggers when the Octave Down key is touched and decreases the notes by one octave.

on_instrument – triggers when the Instrument key is touched and changes the sound from a piano to drums.



03 Simple Piano

To get used to the PianoHAT and its features, load the simplepiano program. This is exactly as the name describes: a simple piano, perfect for beginners.

Navigate to the folder home/pi/Pimoroni/pianohat, and press F4 to start a Terminal session (The HAT requires root access and this method provides it). Next, load the piano program, type sudo python simple-piano.py and then press Enter. Wait a while for the program to run and then play yourself a little tune. Use the Octave buttons to move the note range higher or lower, and press the Instrument button to toggle between drums and piano.

Full code listing

```
import pianohat
import pygame
import time
import signal
import glob
import os
import re

from mcpi import minecraft
mc = minecraft.Minecraft.create()

global move
x,y,z = mc.player.getTilePos()
print x,y,z
move = x

BANK      = './sounds/'
FILETYPES = ['*.wav','*.ogg']
samples   = []
files     = []
octave    = 0
octaves   = 0

pygame.mixer.pre_init(44100, -16, 1, 512)
pygame.mixer.init()
pygame.mixer.set_num_channels(32)

patches = glob.glob(os.path.join(BANK,'*'))
patch_index = 0

if len(patches) == 0:
    exit('You need some patches in {}'.format(BANK))

def natural_sort_key(s, _nsre=re.compile('([0-9]+)')):
    return [int(text) if text.isdigit() else text.lower() for
text in re.split(_nsre, s)]

def load_samples(patch):
    global samples, files, octaves, octave
    files = []
    print('Loading Samples from: {}'.format(patch))
    for filetype in FILETYPES:
        files.extend(glob.glob(os.path.join(patch,filetype)))
    files.sort(key=natural_sort_key)
    octaves = len(files) / 12
    samples = [pygame.mixer.Sound(sample) for sample in files]
    octave = octaves/2

pianohat.auto_leds(True)

def handle_note(channel, pressed):
    global move
    channel = channel + (12*octave)

    if channel < len(samples) and pressed:
        print('Playing Sound: {}'.format(files[channel]))
        print channel
        ### Saves the channel number / note as a variable to
compare to block
        Block_number = channel
        samples[channel].play(loops=0)
        ###Sets block infront of you###
        mc.setBlock(move, y+3, z+3, Block_number)
        move = move + 1 ###add one to the x pos to move blocks
along in a line

def handle_instrument(channel, pressed):
```

Make your own sounds

The piano samples are located and stored in the Pimoroni/pianohat/sounds folder. Create your own sounds such as you singing the note or playing it on another instrument and you can create your own personalised piano synth.

05 Teach yourself to play

This neat little program teaches you to play a well known melody (can you guess what it is?). Run the program and the LED for each required note is lit up, indicating that this is the key to press. Press the key and the note is sounded. Once you have done this the next LED lights up; press this key and the note plays, and so on. Follow the LEDs to learn how to play the melody. You can use this program to experiment and create your own melody / song trainer.

06 Minecraft

The new Raspberry Pi OS image comes with *Minecraft* and the required Python library pre-installed. If you are using an old OS version, it will be worth downloading and updating to either the new Jessie or Raspbian image downloadable here: **https://www.raspberrypi.org/downloads/**

Go to the start menus and load *Minecraft* from the programming tabs. Be aware that the *Minecraft* window is a little glitchy when full size and it is recommended to reduce the size so you can view both your Python code and the game at the same time. Let's look at some simple *Minecraft* hacks that will be used in the final Musical Blocks program.

07 Importing the modules

Load up your preferred Python editor and start a new window. You need to import the following module using from mcpi import minecraft and mc = minecraft.Minecraft.create(). These create the program link between *Minecraft* and Python. The mc variable enables you to type 'mc' instead of having to type out the long-winded minecraft.Minecraft.create() each time you want to use an API feature. Next import the time module to add a small delay when the code runs.

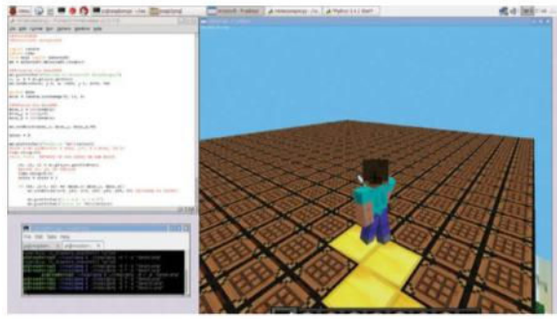
```
from mcpi import minecraft
mc = minecraft.Minecraft.create()
import time
```

Below We've gone for a simple CPU temperature gauge, but the possibilities really are endless

08 Finding your location

When playing *Minecraft* you inhabit a three dimensional environment which is measured by the 'x' axis, left and right, the 'y' axis up and down and the 'z' axis for forward and backwards. As you move along any of these axes, your position is displayed at the top left of the screen as a set of three co-ordinates. These are extremely useful for checking where the player is and can be collected and stored using pos = mc.player.getPos(). This code returns the position of your player and is applied later to the music blocks. Try the simple program below for an example of how the positioning works:

```
from mcpi import minecraft
mc = minecraft.Minecraft.create()
import time
while True:
  time.sleep(1.0)
  pos = mc.player.getPos()
  print pos.x, pos.y, pos.z
```



09 Grow some flowers

Each block in *Minecraft* has its own ID number, for example, flowers have the ID number 38. The code **x, y, z = mc.player.getPos()** gets the player's current position in the world and returns it as a set of co-ordinates: x, y, z. Now you know where you are standing in the world, blocks can be placed using **mc.setBlock(x, y, z, flower)**. Use the code below to place flowers as you walk around the world. Try changing the ID number to place a different block.

```
flower = 38
while True:
  x, y, z = mc.player.getPos()
  mc.setBlock(x, y, z, flower)
  time.sleep(0.1)
```

10 Creating musical blocks

Now you are au fait with the basics of *Minecraft* and the PianoHAT, let's combine them to create a musical block. This uses the ID of each note in the PianoHAT and assigns it to each individual block. For example, the block ID 2 is grass and this corresponds to the note value of C. As you play the piano, the relevant block is displayed in the *Minecraft* world. Open the LX Terminal and type sudo idle to open Python with root privileges. Click file open and locate the simple-piano program, then open it and save it as a different name. You will use this as a template for the musical block program. Now import the modules and *Minecraft* API starting on line 11 of the program.

```
import mcpi.minecraft as minecraft
mc = minecraft.Minecraft.create()
```

11 Finding your positon again

Under the line you just entered and before the line that begins "BANK", line 19, create a global variable called move; this stores the 'x' position of the player. Now find your player's position, line two, using the code you learnt in step 8. On line three, print the position – this is useful for checking that the position and block are functioning correctly. These values are printed to the Python console window. Now you have the position of your player in the *Minecraft* world.

```
global move
x,y,z = mc.player.getTilePos()
print x,y,z
move = x
```

12 Assign a note to a block

Next scroll down to the handle-note function, this begins on line 52 of the final program. After the function name, on the next line, add the global move variable from the previous step. This is the 'x' position of the player. The next line reads channel = channel + (12*octave): 'channel' refers to the number of the note. Move to the If under this line and create a new variable called Block_number which will store the channel number, the number of the note to be played.

```
def handle_note(channel, pressed):
  global move
  channel = channel + (12*octave)
  Block_number = channel
```

13 Set the block

In step nine you learned how to place a block: use this code to place the block that corresponds to the channel number you stored in the previous step. Within the if statement on line 56 under the samples[channel].play(loops=0), add the code to place a block, mc.setBlock(move, y+3, z+3, Block_number) This places the block into the *Minecraft* world.

```
  if channel < len(samples) and pressed:
    print('Playing Sound: {}'.format(files[channel]))
    print channel
    samples[channel].play(loops=0)
    ###Sets block in front of you###
    mc.setBlock(move, y+3, z+3, Block_number)
```

14 The block explained

In the previous step you used the code mc.setBlock(move, y+3, z+3, Block_number) to play a note and place the block. This is achieved by saving the note number, for example, note five, into a variable called Block_number. When the program is run, the code finds your x positon and saves this in a variable called move. This is combined with the set Block code to place the block at your x position. In order for you to view the musical blocks, each block is moved across three and forward three spaces from your original starting position.

15 Moving the block line forward

Once the block is placed, increment the x position by one; this has the effect of moving the next block forward one space. As you play the notes on the Piano, a line of corresponding blocks is built, creating a simple graphical visualisation of the melody you are playing. You will notice that some of the blocks appear to be missing – one of the causes is that there is no block ID number which matches the note ID number. The second reason for a space is that some of the materials are affected by gravity. For example, Sand, Water and Mushrooms all fall down from the line leaving an empty space. Under the line mc.setBlock(move, y+3, z+3, Block_number), line 64, add the code, move = move + 1.

```
mc.setBlock(move, y+3, z+3, Block_number)
move = move + 1
```

16 Posting a message to the MC World

The last step is to post a message to the *Minecraft* world to tell the player that the Piano and musical blocks are ready. On line 86 add the code mc.postToChat("Welcome to musical blocks"). When you run your program you will see the message pop up at the bottom of the world. Try changing your message or use the same code-line to add other messages throughout the game. Once the message is displayed the samples have been loaded and your *Minecraft* Piano is ready.

```
mc.postToChat("Welcome to the music blocks")
```

17 Running the music block

INow that you have completed the code save it. Open *Minecraft* and create a new world. When this has finished loading, press F5 in IDLE to run your program. Press a key on the piano and look out for the block appearing just above your head. Remember that as the player's position is measured only once at the beginning of the program, the blocks will always be placed from the same starting reference position. Play your melody to create a musical visualisation.

Full code listing (cont.)

```
    global patch_index
    if pressed:
        patch_index += 1
        patch_index %= len(patches)
        print('Selecting Patch: {}'.format(patches[patch_
index]))
        load_samples(patches[patch_index])

def handle_octave_up(channel, pressed):
    global octave
    if pressed and octave < octaves:
        octave += 1
        print('Selected Octave: {}'.format(octave))

def handle_octave_down(channel, pressed):
    global octave
    if pressed and octave > 0:
        octave -= 1
        print('Selected Octave: {}'.format(octave))

mc.postToChat("Welcome to music")

pianohat.on_note(handle_note)
pianohat.on_octave_up(handle_octave_up)
pianohat.on_octave_down(handle_octave_down)
pianohat.on_instrument(handle_instrument)

load_samples(patches[patch_index])

signal.pause()
```

Software

72 Supercharge your Raspberry Pi

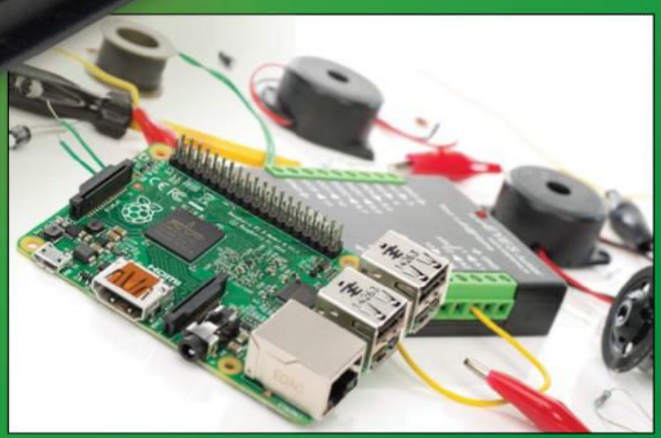
96 Set up a motion sensor

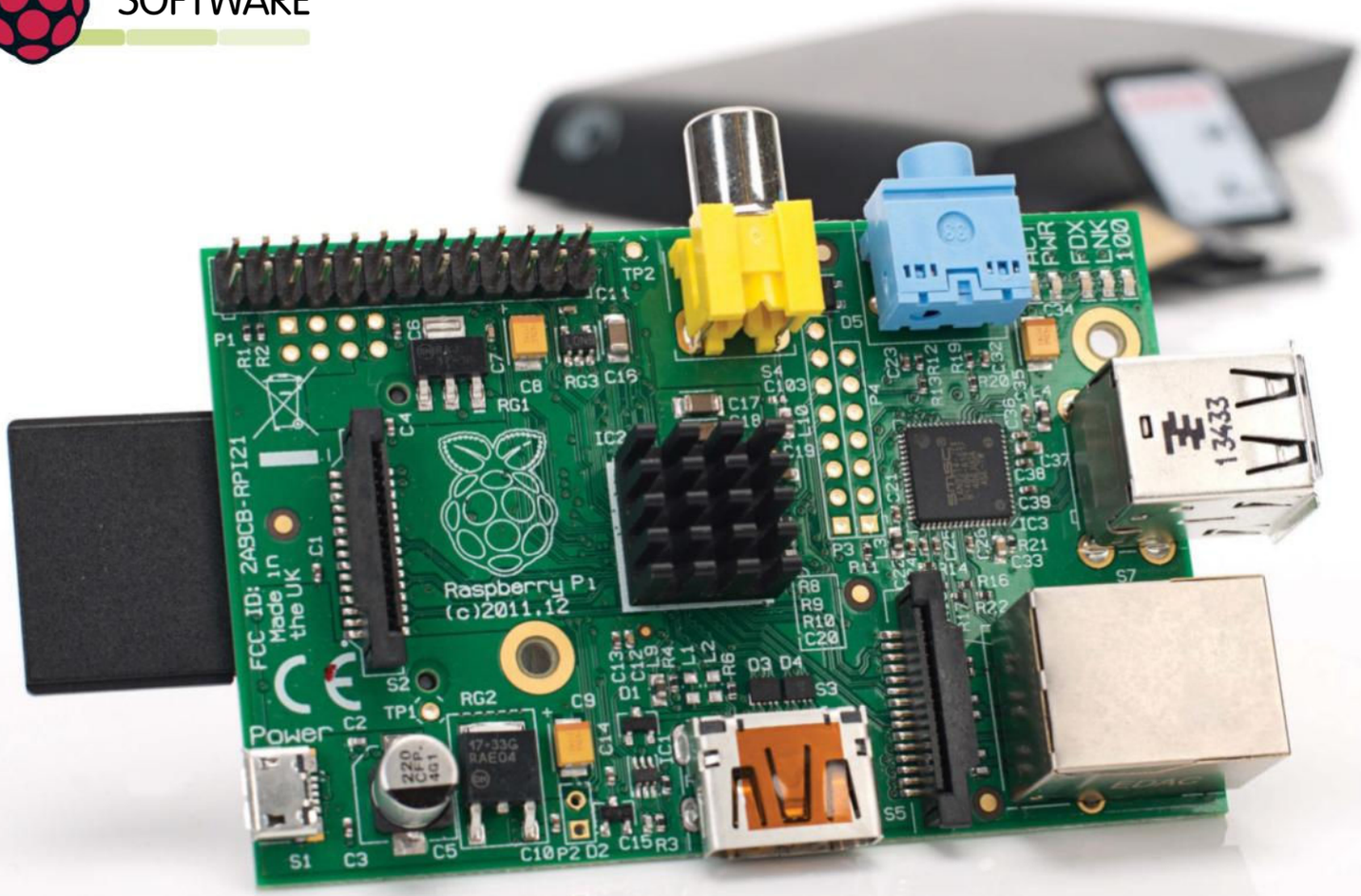
98 Program a synthesiser

86
Monitor CPU temperature

"Use your Raspberry Pi to test out your coding skills and get to grips with programming"

92
Print documents wirelessly





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.

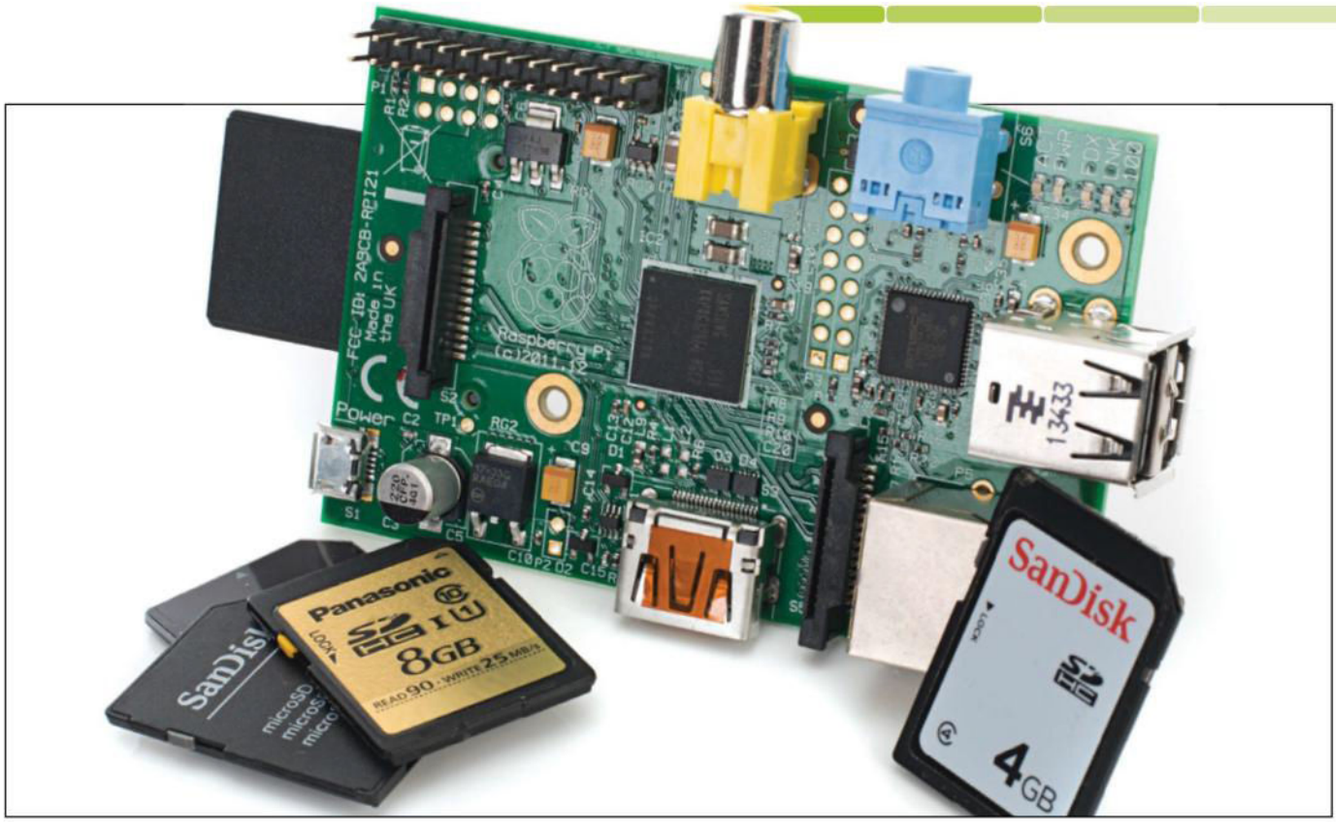
But are you truly getting the most from your little computer? Do the components you're using maximise the potential of your Raspberry Pi or are they holding it back? Perhaps you haven't explored the full set of options in Raspbian, or you're running the entire OS from SD card, something that can reduce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

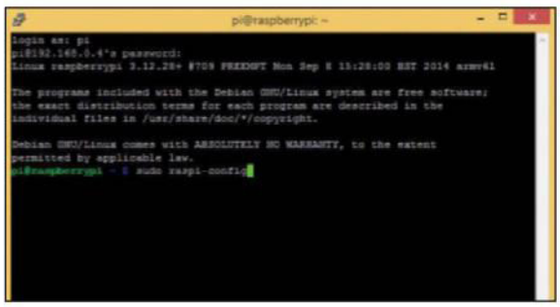
01 Use better storage hardware

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.



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+.



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

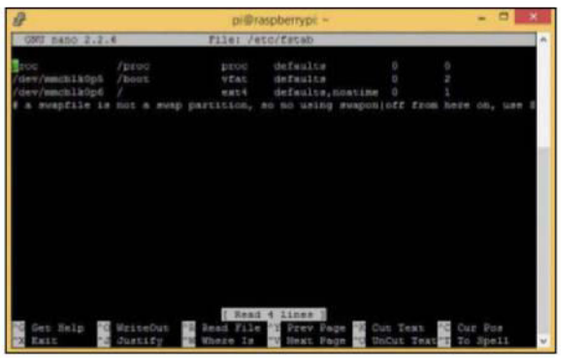
Maximising the partition affords the full capacity of your SD card, which will increase the media's lifespan (there is more space to write too, so the same sectors aren't being overwritten as often).

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.



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
```

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

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=30
m 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,m
ode=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
```

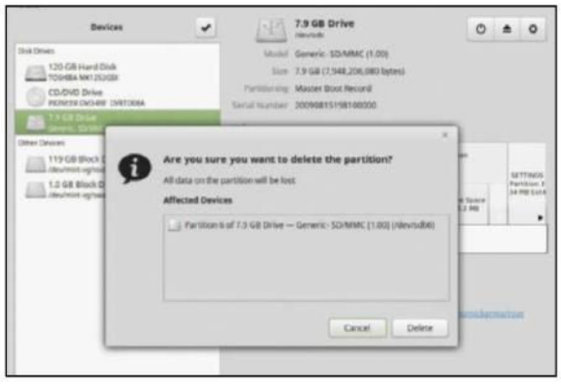
The path /dev/sdb represents the SD card. Copying should take 5-10 minutes. When complete, remove the SD card and connect your USB device.

10 Copy Raspbian to USB

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

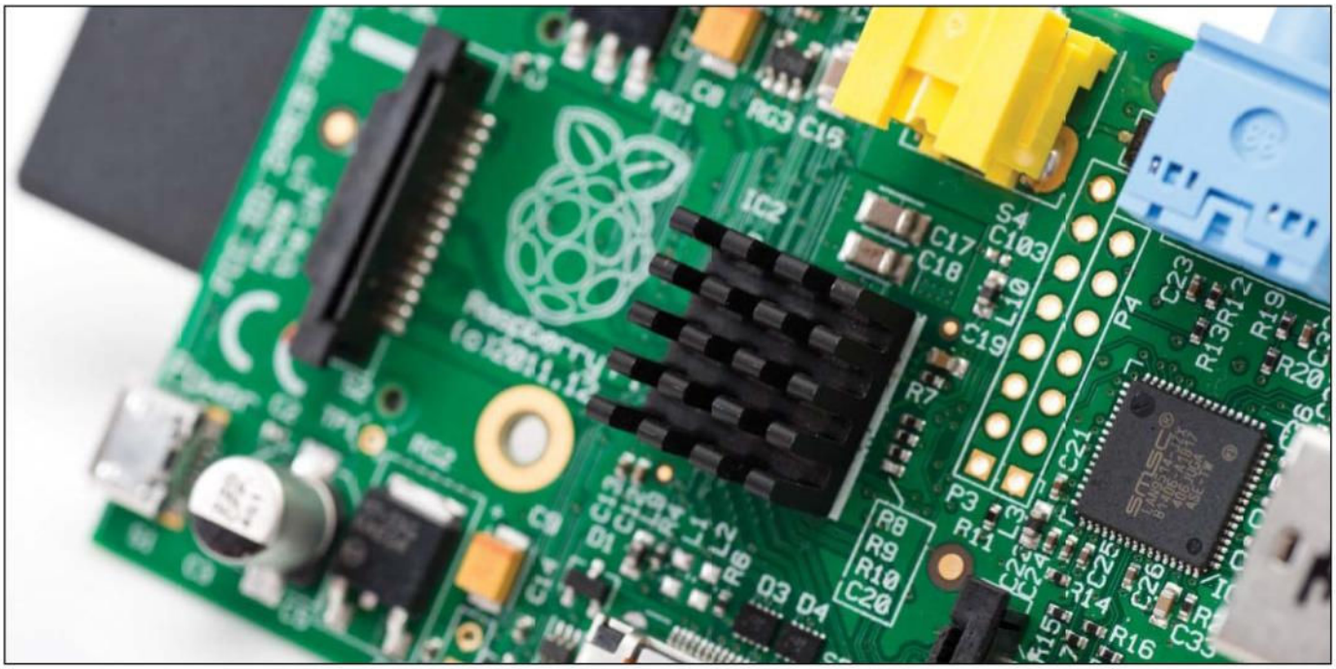
```
sudo dd bs=4M if=~/backup.img of=/dev/sdc
```

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
```

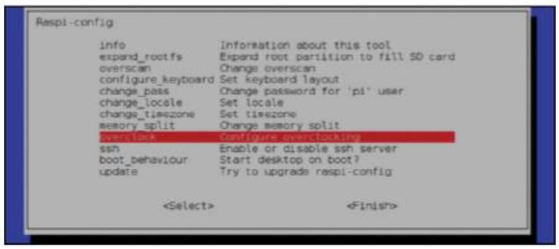
Enter ? to display the options and select Recovery and Transformation options (experts only), followed by Load MBR and Build Fresh GPT. Tap ? one last time and select 'Write Table to Disk' and exit. Remove and replace the USB device and run gdisk again. This time enter I and then 1 to display the Partition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

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.

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.

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

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
```

Right CMUSphinx is used in cross-platform, open source projects like ILA, the Intelligent Learning Assistant

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

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.

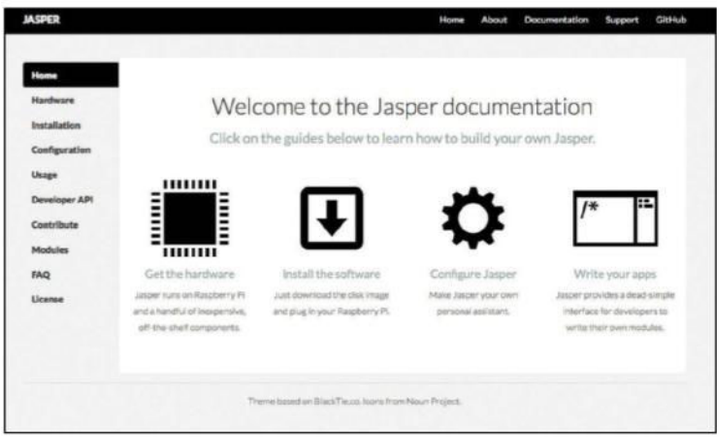
Full code listing

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

What you'll need

- Raspberry Pi Model B
- ExpEYES kit
  bit.ly/1AR15dz

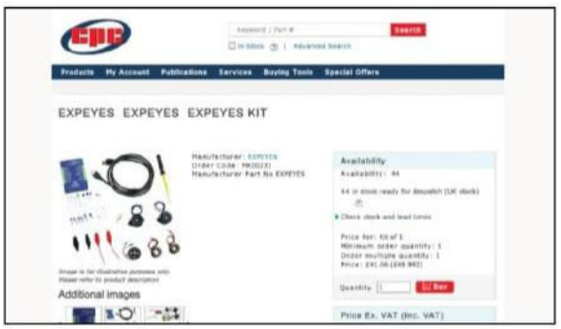
Run science experiments on the ExpEYES kit

ExpEYES is a cheap digital oscilloscope with a signal generator and other features, making it the ultimate tool for electronics

ExpEYES is a relatively unheard of but very impressive hardware and software platform for science and electronics experimentation, as well as a useful electronic probing tool for makers and professionals alike. It is also open source on both the hardware and software sides, which makes it affordable and versatile.

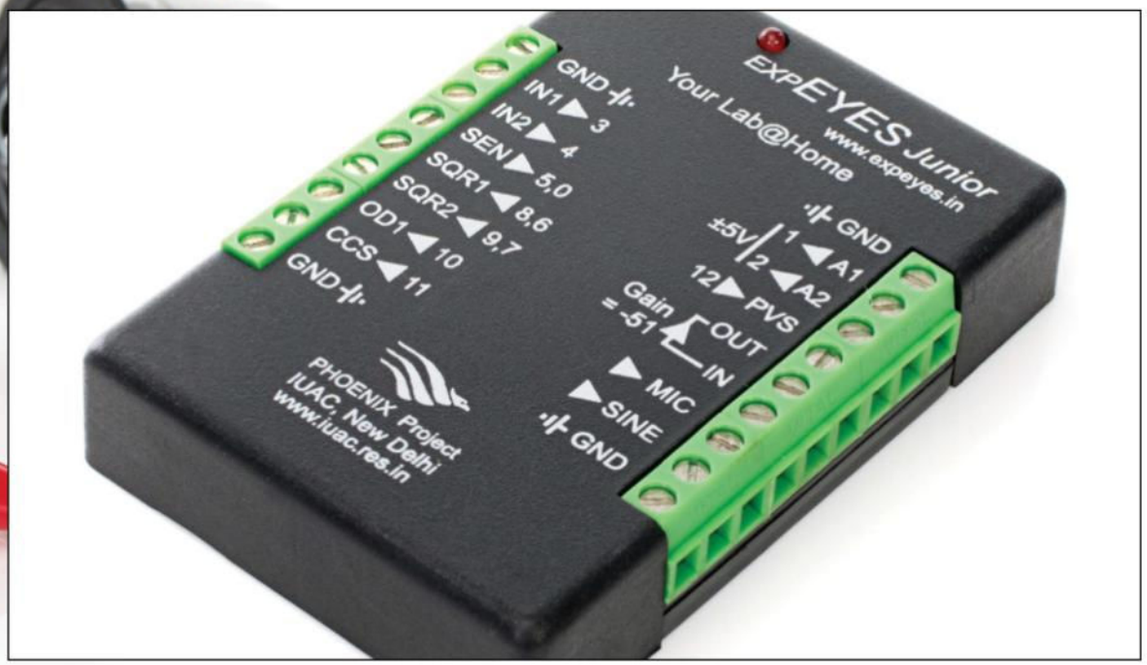
ExpEYES is billed as a science and experimentation kit but really it is much more than that – it is a fully-functioning four-channel digital oscilloscope with an impressive array of features. ExpEYES ships with a wealth of online documentation in a variety of formats (graphics, user guides, web content), including upwards of 50 suggested experiments, and the kit itself contains all of the hardware required to play with the interesting science of electronics contained within the guide material.

The aim is to enable the learning of what can be complex concepts of electronics in an easy and affordable way, without getting bogged down in the arcane details. Paired with our favourite little single-board computer, the Raspberry Pi, you have an extremely powerful and affordable device.



01 Get the parts

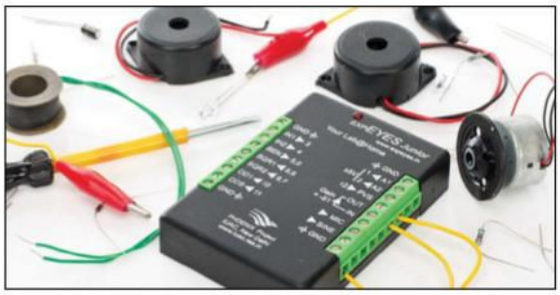
ExpEYES is available to purchase from a variety of online vendors, including CPC (**http://cpc.farnell.com**), for around £50. It is possible to get the kits slightly cheaper from India or China (see **bit.ly/1H38EFC** for other vendors worldwide), however it's likely to end up costing more due to higher shipping rates as well as potential import fees and duties.



Left The kit itself is highly portable and great for taking down to Jams and hackspaces

02 Open it up

The ExpEYES kit contains everything you need to get underway, with over 50 documented experiments from the ExpEYES website. The only other item that may come in handy is a breadboard. You will also need a Raspberry Pi or other computer with a USB port in order to run the digital oscilloscope software and connect to ExpEYES.



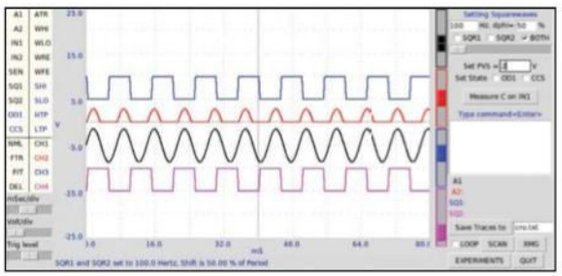
03 What's inside?

As you may have guessed, the ExpEYES kit includes the main ExpEYES USB digital oscilloscope, but it also contains a wide range of other hardware including a DC motor, magnets, LEDs, coils, piezoelectric discs, wiring, a small screwdriver for opening the screw terminals and more. You also get a live CD which contains all the ExpEYES software and documentation ready to go on a bootable disc.

04 What can it do?

The chip at the heart of ExpEYES is an AVR ATmega16 MCU (microcontroller unit), running at 8 MHz coupled to a USB interface IC (FT232RL). These are low-cost but provide good value for money. As we have already mentioned, ExpEYES is therefore capable of acting as a four-channel oscilloscope but also has a built-in signal generator, 12-bit analogue resolution, microsecond timing resolution and a 250 kHz sampling frequency. At this price point, that's an impressive set of features and certainly accurate enough for anything that is not mission critical (like learning, hobby projects, quick readings and so on).

It pays dividends to make sure that your operating system is updated to the latest stable version, as this can save you a lot of hassle



05 Using the live CD

Perhaps the easiest way to get up and running with ExpEYES (if you have a computer with a CD drive) is to use the live CD which is included in the ExpEYES kit. Making sure that you are booting into the live CD from your BIOS boot menu, you should then be greeted with a Linux-based desktop. Plug in your ExpEYES by USB and you can open the software from the menu by going to Applications>Science>ExpEYES-Junior. Alternatively, you can run it from a terminal window using:

```
sudo python /usr/share/expeyes/eyes-junior/croplus.py
```

06 Update your Raspberry Pi

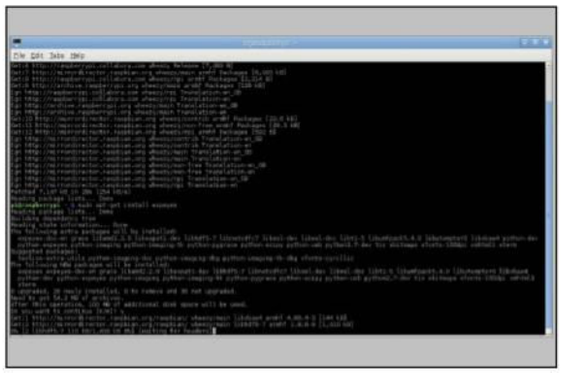
As with almost every project you undertake on the Raspberry Pi, it pays dividends to make sure that your operating system is updated to the latest stable version, as this can save you a lot of hassle further down the line. To do this, open an LXTerminal session and then type **sudo apt-get update**, followed by **sudo apt-get upgrade –y**, and then wait patiently for the upgrade process to complete.

Other supported platforms

The ExpEYES software is mainly written in Python. This means that the core software to run your ExpEYES device is quite platform-agnostic – if the device can run a Python interpreter and has a Python module enabling it to access the serial port then it will work with ExpEYES. If you visit the ExpEYES website, there is a page that explains how to install the software on Linux and Windows – **www.expeyes.in/software-installation**. In addition, there is a native Android app which will enable your ExpEYES to work with any Android device that has USB OTG (on the go) capability.

ExpEYES & PHOENIX

ExpEYES was developed by Ajith Kumar and his team as part of the PHOENIX (Physics with Homemade Equipment and Innovative Experiments) project, which was started in 2005 as a part of the outreach program of the Inter-University Accelerator Centre (IUAC) in New Delhi, India. Its objectives are developing affordable laboratory equipment and training teachers to use it in their lesson plans.



07 Install the software

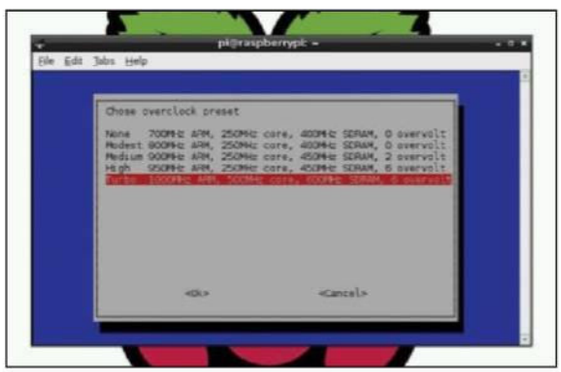
Due to efforts of community member Georges Khaznadar, there are DEB packages available for the ExpEYES software that should work perfectly on Debian, Ubuntu, Linux Mint and, of course, Raspbian. These are also included in the official Raspbian repositories, so all you need to do to install the ExpEYES software is to open an LXTerminal session on the Raspberry Pi and then run the following commands:

```
sudo apt-get update
sudo apt-get install expeyes
```

08 Install dependencies

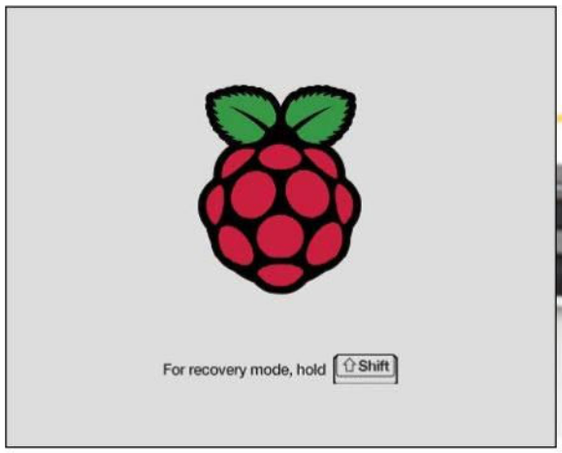
ExpEYES has a number of dependencies that are required for it to run under Linux, as well as a number of other recommended libraries. During the installation undertaken in Step 7, the dependencies should be installed by default. However, to avoid any problems later, you can run the following command in order to make sure that they are all installed:

```
sudo apt-get install python python-expeyes python-imaging-tk python-tk grace tix python-numpy python-scipy python-pygrace
```



09 Overclock your Raspberry Pi (optional)

The ExpEYES software will run fine on a Raspberry Pi with default settings, however it can be slow to respond if you are using a Model A, B or B+. We recommend using a Model 2B, but if you don't have one, overclocking your Pi would be advisable (you can overclock your 2B as well if you want it to run a bit faster). Open an LXTerminal session and type **sudo raspi-config**. In the menu, select the option '7 Overclock'. Click OK on the following screen and then select Turbo. Click OK and you should see some code run. Once this completes, press OK again and then you are brought back to the main raspi-config window. Select Finish in the bottom right and Yes to reboot your Raspberry Pi.



10 Overclocking continued

Overclock can sometimes cause instability on your Raspberry Pi or an inability to boot at all. If this happens you can press and hold the Shift key on your keyboard once you reach the above splash screen to boot into recovery mode. You can then redo Step 7 at a lower overclock setting and repeat until you find the highest stable setting.

11 Resistance of the human body

An interesting experiment for your first time using an oscilloscope it to measure the resistance of the human body over time. This is easy to accomplish with just three bits of wire and a resistor (200 kOhm). On the ExpEYES, connect a wire between A1 and PVS, connect the resistor between A2 and ground, and connect an open-ended wire out of both PVS and A2. Plug in your ExpEYES and open the control panel, then drag A1 to CH1 and A2 to CH2, and set PVS to 4 volts. You can then pick up one of the open-ended wires in each hand and watch the response on the ExpEYES control panel.

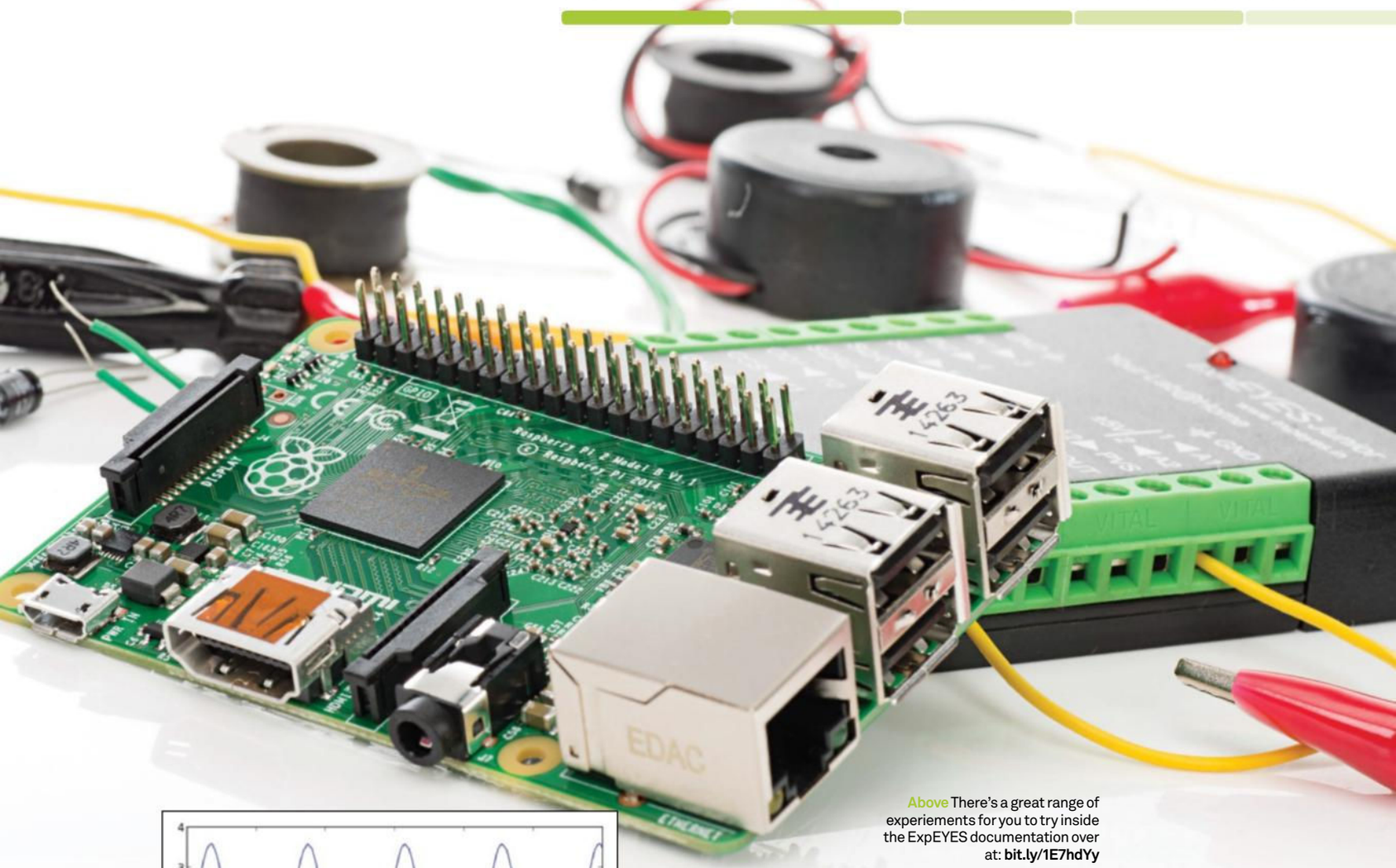
12 Run the maths

From the output plot, you should find that the input on CH1 is coming out at 3.999 volts (which is great because we set it to be 4!). The voltage on A2 (CH2) is showing as 0.9 volts for us, which implies that the voltage across the unknown resistor value (your body) is 4 – 0.9 = 3.1 volts. Using Ohm's law (V=IR), we can then calculate the current (I) across the known resistor value: voltage ÷ resistance = 0.9 ÷ 200,000 = 0.0000045 amps = 4.5 uA (micro amps). Using this value we can then calculate the resistance of the body using the same Ohm's law equation in reverse: voltage ÷ current = 3.1 ÷ 0.0000045 = 688889 ohms = 689 kΩ. This is a surpisingly high value, however the resistance of the human body depends hugely on how dry your skin is and a large number of other factors (body resistance is usually in the range of 1,000 to 100,000 ohms).

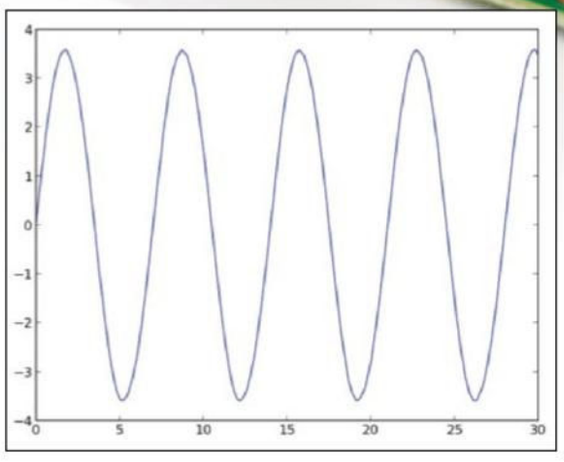
13 Use the Python library

The ExpEYES team have built a custom Python library for the device. This is slightly harder to use than the GUI and not as pretty, but it enables a lot more versatility as well as the capability to use ExpEYES functionality within your Python scripts. If you have followed the installation instructions above, all you need to do is import the Python module and then initialise a connection to the ExpEYES using:

```
import expeyes.eyesj
p=expeyes.eyesj.open()
```



Above There's a great range of experiements for you to try inside the ExpEYES documentation over at: **bit.ly/1E7hdYy**



14 The Python library (continued)

Now we will plot a sine wave using the ExpEYES and PyLab libraries. On the device, connect OD1 to IN1 and SINE to A1 with some wire. Run the following code and you should see that a sign wave has been plotted.

```
import expeyes.eyesj
from pylab import *

p=expeyes.eyesj.open()
p.set_state(10,1)
print p.set_voltage(2.5)
ion()       # set pylab interactive mode
t,v = p.capture (1,300,100)
(plot t,v)
```

A digital storage oscilloscope is a useful tool in any engineer or hacker's toolbox, as it enables you to get insights into your projects that aren't possible with visual checks

15 Further experiments

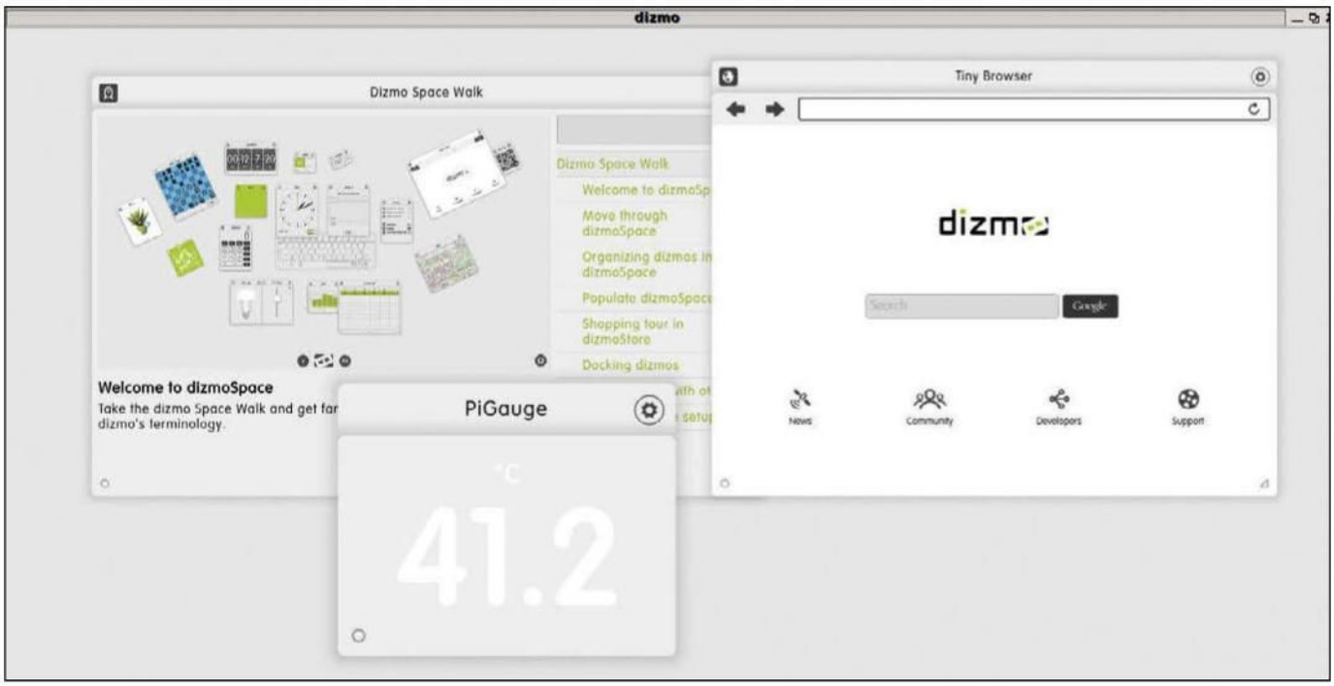
This tutorial has shown you just a single example of the documented ExpEYES experiments available at **http://expeyes.in**. There is a wide variety of different techniques and phenomena explored in those experiments, so it is highly recommended to get your hands on an ExpEYES kit and work through them. Running through those examples as a beginner will give you a much deeper understanding of electronics.

16 The verdict

A digital storage oscilloscope (plus extras) is a useful tool in any engineer or hacker's toolbox, as it enables you to get insights into your projects that aren't possible with just visual checks or using a multimeter. Whilst no £50 oscilloscope will compare to expensive professional units, this is a great entry-level product as well as a versatile, portable USB device with multiplatform support for when you just can't be lugging around a 10 kg, £1000+ scope.

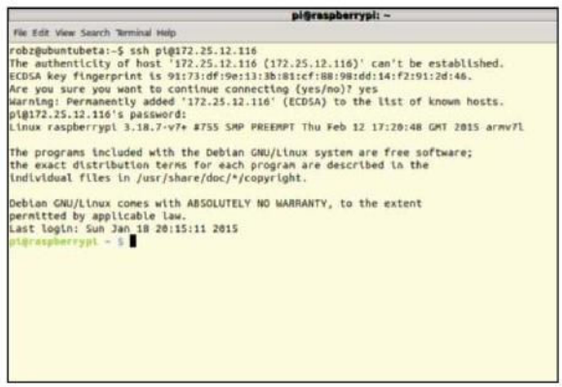
Monitor CPU temperature with Dizmo

Turn your Raspberry Pi into an Internet of Things with this CPU temperature gauge tutorial



The Raspberry Pi is an exciting prospect for people interested in an Internet of Things – size, power and flexibility make it perfect for powering any Internet-connected device around the home or office. Setting up a Raspberry Pi to be the brain of an IoT network isn't exactly a case of selecting the right software in Raspbian, though; there's a lot of custom work you need to do to get one going.

This is where Dizmo comes in, enabling you to control IoT objects using an online API that you can then access remotely. To show you how it works, we're going to have it track the Raspberry Pi's core temperature. In this tutorial we are going to work entirely over SSH, but you can easily do this straight on the Pi – the benefit of SSH though is that for a real IoT, it will be easier to maintain remotely.

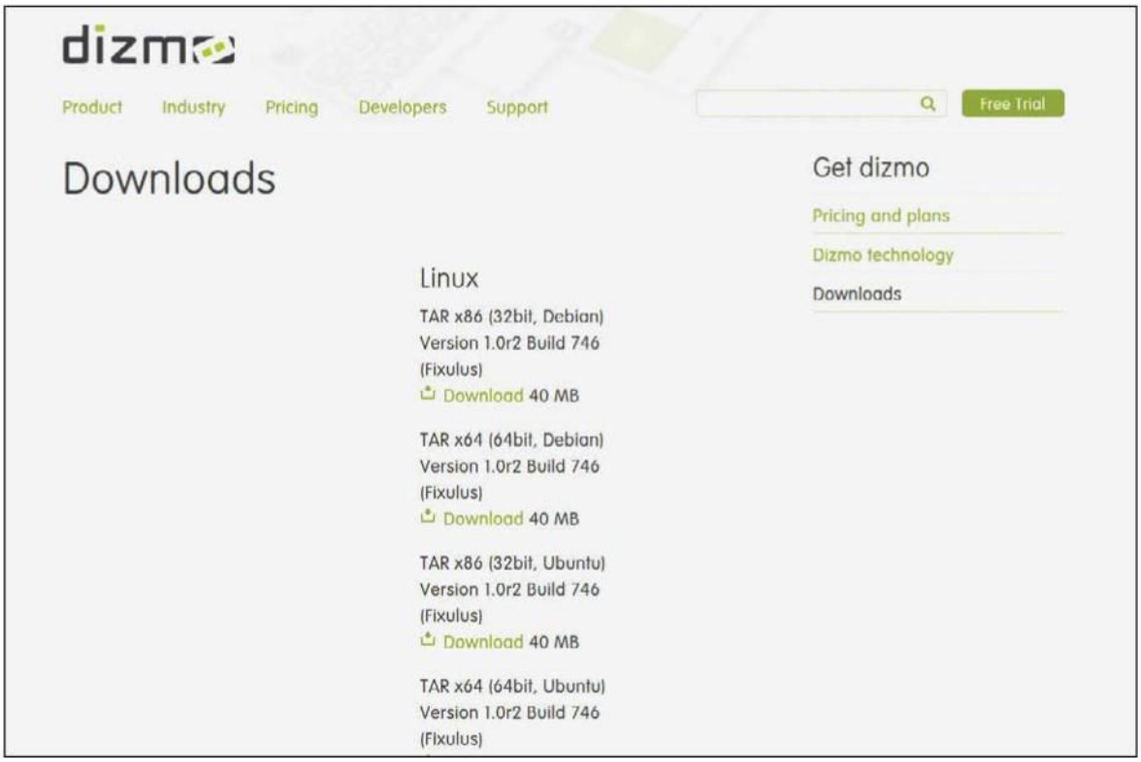


Above Dizmo is designed to be a multi-touch interface

01 Dial into your Pi

Make sure your Raspberry Pi can connect to your network, either via Wi-Fi or ethernet cable, and find out the IP address by using **ifconfig**. Use this IP to dial into the Pi from another system with:

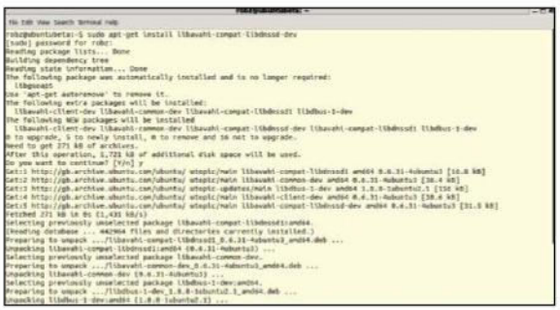
```
$ ssh pi@[IP address]
```



Left Builds are available for various distros on the Download page, and you can also check the pricing plans

02 Install dizmoSpace

If you haven't already, head to **www.dizmo.com**, grab dizmoSpace and install it to the system you plan for it to work with. All you need to do is download the zip and unpack it, then click the Dizmo icon or run it from the terminal.



03 Launch issues?

If Dizmo is complaining about libraries when you try to run it, you'll need to install some extra software. Open the terminal on the PC you're working from and install the extra software with the following:

```
$ sudo apt-get install libavahi-compat-libdnssd-dev
$ sudo apt-get install libavahi-client-dev
```

04 Download node.js

Now, we need to grab the latest version of node.js for the Raspberry Pi. Back in the SSH connection to your Raspberry Pi, use the following:

```
$ sudo wget http://node-arm.herokuapp.com/
node_latest_armhf.deb
$ sudo dpkg -i node_latest_armhf.deb
```

> A Dizmo widget is a HTML file, packaging resources together to create an interface or graphic. Our HTML file uses jQuery

05 Add framework

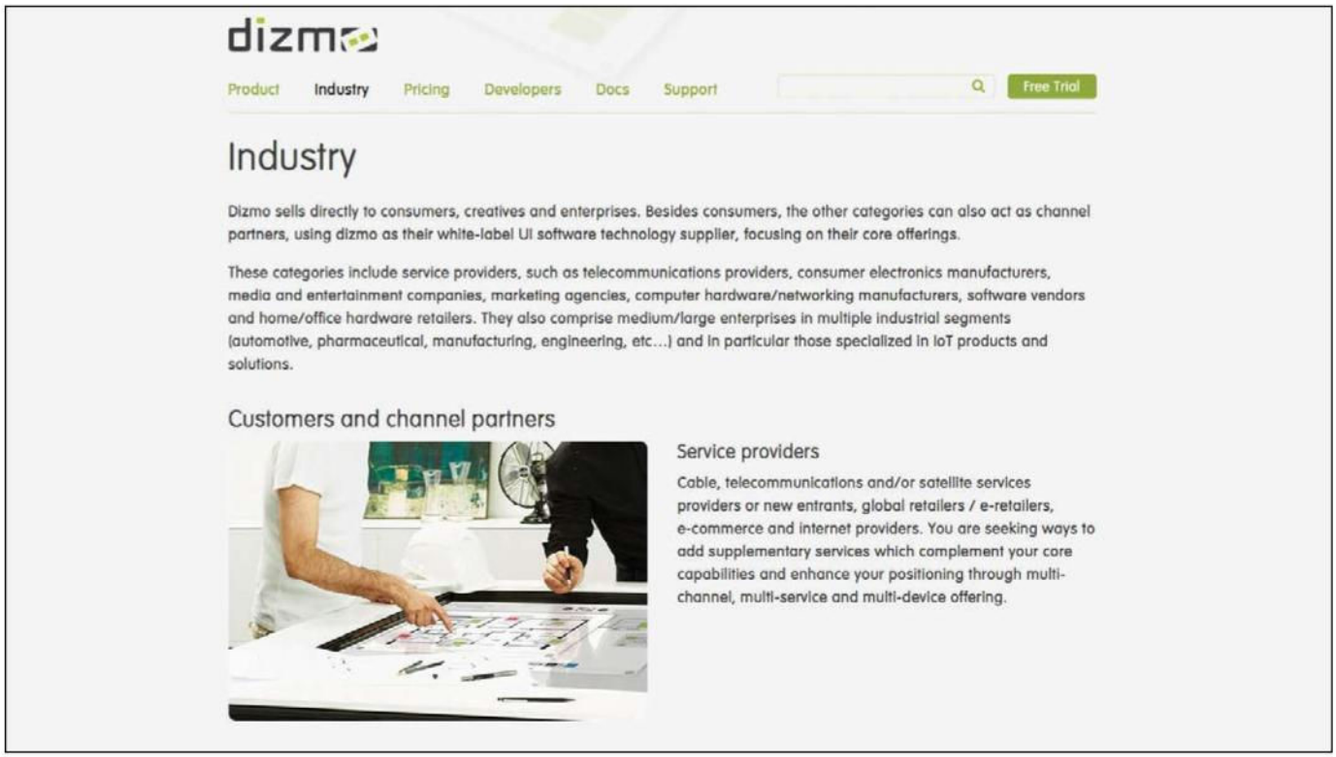
Use **node -v** to check if it's installed correctly – it should spit out a version number for you. Once that's done, install express.js, which will be our web application framework:

```
$ sudo npm install -g express
$ sudo npm install -g express-generator
```

06 Install framework

We'll create the folder www in var and create a symlink for everything to run. Do this by moving to var, creating www and making the symlink with:

```
$ cd /var
$ sudo mkdir www
$ cd www
$ sudo ln -s /usr/local/lib/node_modules/
/node_modules
```



Above As it's multi-touch, Dizmo is perfect for interactive table displays in meetings

Internet of Things

It's not a very descriptive term, but the Internet of Things can be almost anything. Any item that is or can be connected to the internet or networks, such as modern automated lights, can be connected up to Dizmo and the Raspberry Pi.

Dizmo space walk Enjoy some pre-installed projects to see exactly what Dizmo can do

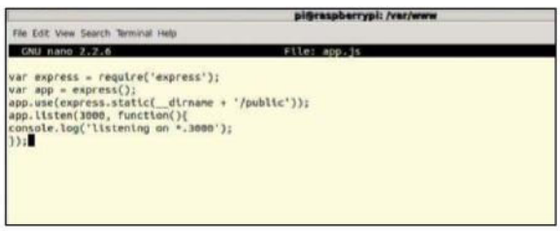
PiGauge Create a custom app to monitor the temperature of your Raspberry Pi, and then go even further

Browser Create an entire custom display using a variety of information that can connect to and through the Pi

07 Package file

First, create the file package.json with `sudo nano package.json`, then enter:

```
{
"name": "ServeSysinfo",
"version": "0.0.1",
"dependencies": {"express": "4.x"}
}
```



08 App node

Now, create a file called app.js and enter the following:

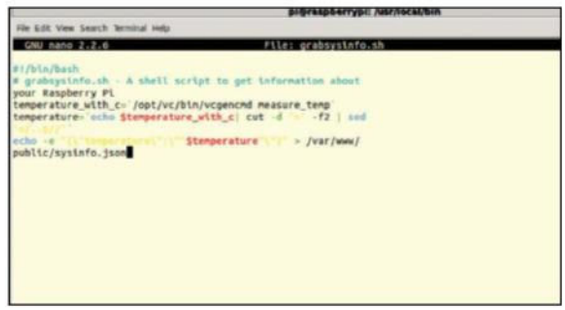
```
var express = require('express');
var app = express();
app.use(express.static(__dirname + '/public'));
app.listen(3000, function(){
console.log('listening on *.3000');
});
```

09 Start node.js

You can now start the node server by typing in:

```
$ node app.js
```

It will say it's listening on *.3000. Start up a new terminal, **ssh** in, and create the folder /public with **mkdir /public** to save all of the CPU data in.



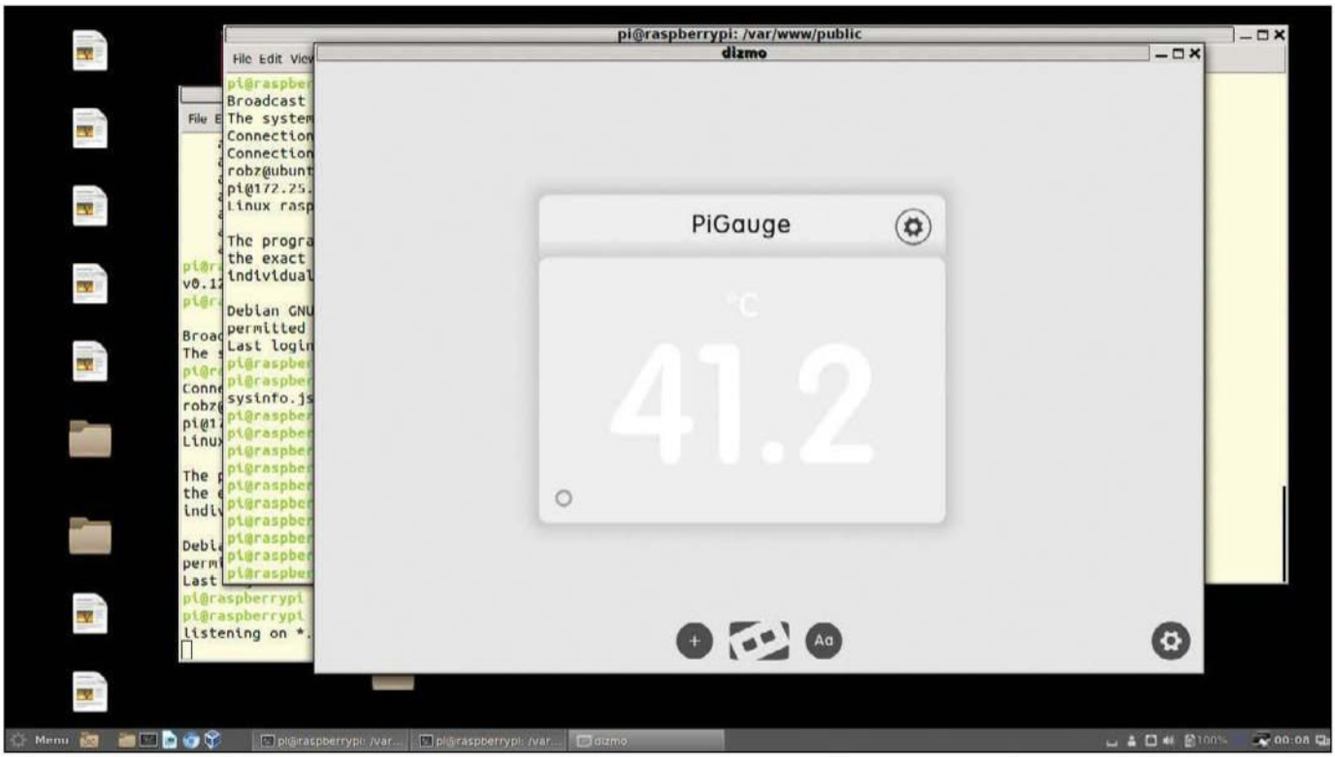
10 CPU information

We are going to use the **vcgencmd** command to get the CPU information from the Raspberry Pi. We will write a script that will do this and then write the info to sysinfo.json. Download the file grabsysinfo.sh from FileSilo and put it in /usr/local/bin.

11 Make a cronjob

We will make it so that the temperature is updated every ten minutes. You can make it update much faster if you want, but have a play around with that. Open up cron with `sudo crontab -e` and add this at the end:

```
*/10 * * * * /usr/local/bin/grabsysinfo.sh
```



12 Start creating the widget

It is time to actually start building the widget. First of all, create a folder on your local machine called Gauge and **cd** to it. Now you need to download the first file called info.plist into here by using the following:

```
$ wget x/info.plist
```

13 Index file

A Dizmo widget is basically a HTML file, packaging resources together to create an interface or graphic. Here, we have the main HTML file that uses jQuery, which helps display the temperature. Still in the Gauge folder, download it with:

```
$ wget x/index.html
```

> With these building blocks, you can now start doing more interesting IoT things – controlling the GPIO ports, getting more information

14 Style guide

Now we'll add the CSS style sheet for the Dizmo widget. As usual, this styles up the display on the page that will become our widget. Download it with:

```
wget x/style.css
```

15 Final application

The final step is to create the application.js file, which will call the temperature from the Raspberry Pi using Ajax. You can download it using:

```
wget x/application.js
```

Change the IP address to the one on your Pi.

Once that's done, you can test out the widget – compress the Gauge folder to a .zip and then change the .zip to a .dzm. Launch dizmoSpace and drag the dzm file onto it for it to start.

16 Get coding

With these building blocks, you can now start doing more interesting IoT things – controlling the GPIO ports, getting more information, having it connect to other objects to control them as well. Check out the Dizmo website for more details on projects that you can do.

Above We've gone for a simple CPU temperature gauge, but the possibilities really are endless

Talking on the I2C bus

There are several ways that the Raspberry Pi can talk to the world. Here, learn about the I2C bus

The Raspberry Pi was designed to provide several ways to interact with the world through sensors and activators. In the past, we have looked at using the GPIO interface pins to communicate with several devices at once. This is not the only way to work with the world at large, however. In this tutorial, we will look at one of the other mechanisms available, the I2C bus. I2C (Inter-Integrated Circuit) bus was invented by Philips Semiconductor, with version 1 having come out in 1992.

The design is for short connection paths, and supports multiple masters and multiple slaves where messages on the bus are delivered using device addresses. Messages have a START section and a STOP section, wrapped around the core of the message. The three types of messages you can send are a single message where a master writes data to a slave, a single message where a master reads data from a slave, or a combined message where a master sends at least two read or two write messages to one or more slaves.

> I2C (Inter-integrated Circuit) bus was invented by Philips Semiconductor, with version 1 having come out in 1992

Now that we have a little bit of an idea of what the I2C bus is, how can you use it with your Raspberry Pi? The first stephere is to activate the bus within the Linux kernel. By default, the relevant kernel modules are blacklisted and not loaded at boot time. If you are using a newer version of Raspbian, you can use the utility 'sudo raspi-config' and select the 'Advanced Options' section to set correct options. If you are using an older version or simply wish to make the changes manually, it is a bit more complex. In order to change this, you will need to edit the file '/etc/modprobe.d/raspi-blacklist.conf' and comment out the line about the I2C module. The line in question is

```
blacklist i2c-bcm2708
```

This line should be changed to

```
#blacklist i2c-bcm2708
```

Once you have removed the I2C module from the blacklist, you can add the I2C module to the list of modules to be loaded at boot time. This file is '/etc/modules', and you should add the following to the end of the file contents

```
i2c-dev
```

Rebooting at this point will now make the I2C bus accessible to the kernel. Because it is a low level interface, your user will need to be added to the I2C access group. If you are still using the default Pi user, you can do this with the command

```
sudo adduser pi i2c
```

In order to do anything useful, you will want to install the available command line tools and the Python module with the command

```
sudo apt-get install i2c-tools
python-smbus
```

A simple test to verify that everything is working correctly is to use the command 'i2cdetect -y 0' to query the bus and see if anything is connected. You should see that nothing is there, since we haven't connected anything yet. If you are using a newer Raspberry Pi, the I2C bus is set to using port 1, rather than 0, so you would need to use the command 'i2cdetect -y 1' instead. You are now ready to connect your devices to the Raspberry Pi.

The pins used are part of the GPIO header, with two of those pins used for I2C communications. There are modules available to detect magnetic fields, or ultrasonic range finders, among many others. The devices that you attach to the I2C bus all need to have unique addresses so that only one of the devices will receive messages to some particular address. The address of the device is set during manufacture, so you will need to read the specification documents to see what the address is for any particular device. Now that everything is set up and connected, we can start to look at how to write some Python code to actually do something useful with the devices on the bus. The first step in this process is to import the required module with

```
import smbus
```

You may have noticed that we didn't import something with I2C in the name. This is because the hardware on the Raspberry Pi uses a subset of the full I2C specification, called SMBus (System Management Bus), defined by Intel in 1995. This is also the protocol used in I2C interfaces for desktop computers. Before doing anything else, you will need to instantiate an SMBus object with

```
bus = smbus.SMBus(0)
```

The parameter handed in within the constructor is the port to open a connection on. So, for a newer Raspberry Pi, you would use 1 rather than 0. Once you have a new SMBus object you can start doing some basic reading and writing to the devices on the I2C bus. The most basic boilerplate code looks like

```
i2c_addr = 0x20
# Write a byte to the device
bus.write_byte(i2c_addr, 0xFF)
# Read a byte from the device
val = bus.read_byte(i2c_addr)
```

Since we are dealing with individual bytes, it is easiest to use hexadecimal numbers in your code. The common parameter in both reading and writing is the bus address for the device. This address is a 7-bit number, which may be given to you as a binary number within the documentation for the device. You can convert it to a hexadecimal pair by adding an extra 0 to the beginning of this 7-bit address.

These simple commands write to the first register of your device. But, it may be more complex and have multiple registers available for reading and writing data to. In these cases, you can explicitly pick which register to use with the functions

```
# Writing to a specific register
reg = 0x10
val = 0x01
bus.write_byte_data(i2c_addr, reg, val)
# Reading from a specific register
return_val = bus.read_byte_data(i2c_addr, reg)
```

For larger chunks of data, you can read and write 2-byte words, as well. The code to do this looks like

```
# Writing a full word
word_val = 0x0101
bus.write_word_data(i2c_addr, reg, word_val)
# Reading a full word
return_word = bus.read_word_data(i2c_addr, reg)
```

For most devices, this is probably the most that you will need to use. There will be cases, however, when you need to read and write even larger chunks of data to and from your device. In these cases, you can read and write entire lists of values to and from your device. Because of the specification differences between I2C and SMBus, there are two sets of reading and writing functions. If you want to use the SMBus, the functions look like

```
# Writing a full list
list_val = [0x01, 0x02, 0x03, 0x04]
bus.write_block_data(i2c_addr, reg, list_val)
# Reading a full list
return_list = bus.read_block_data(i2c_addr, reg)
```

The problem with these methods is that they are limited to a maximum of 32 bytes of data. If you need to transfer more than this, you need to use the underlying I2C protocols. When you write a list, you can simply hand in the list. When reading, however, you need to tell the library how many bytes to read in as part of the function call. A basic example of the code would look like

```
# Writing a full list
list_val = [0x01, 0x02, 0x03, 0x04]
bus.write_i2c_block_data(i2c_addr, reg, list_val)
# Reading a full list of 5 values
return_list = bus.read_i2c_block_data(i2c_addr, reg, 5)
```

There's also the concept of a process call within the SMBus protocol. This function both sends a block of data and reads a block of data from a device on the bus. The python function call looks like

```
result_list = bus.block_process_call(i2c_addr, reg, list_val)
```

This lets you interact with the device in a single function call, which can help clean up your code a bit.

The last two functions we will look at are shortcut functions, designed to allow for quick interactions with your I2C device. The first is the function

```
bus.write_quick(i2c_addr)
```

This function writes a single bit to the first register of the device at the address you give it. For some devices, this may be enough interaction to get some useful work done. The second shorthand function is

```
bus.process_call(i2c_addr, reg, val)
```

This function call executes the process call transaction of the SMBus protocol, similar to the 'block_process_call()' function from above. The purpose is to send a chunk of data to your device and receive a resultant set of data back from it, as a single function call.

Hopefully, this article has been able to provide a jumping off point in using I2C and SMBus. Now, you can start adding a whole suite of devices to your Raspberry Pi and create a complete sensor platform for your projects.

SPI is available too

The Raspberry Pi has another communication bus available for you to use, called the SPI bus (Serial Peripheral Interface). It is similar to the I2C interface, except it only allows for a single master. The SPI bus is also not active by default.; you will need to activate it, either manually or by using the 'raspi-config' utility. You will also need to install the relevant python module with

```
sudo apt-get python-spidev
```

Once you have SPI activated and the spidev module installed, you can initialise the bus with the code

```
import spidev
spi = spidev.SpiDev()
```

The next step is to open a connection to the device of interest. To do this, you need to use the function

```
spi.open(0, 0)
```

The two parameters in the open function are the bus and device IDs for the device you want to talk to. When you are done, you will need to explicitly close the connection with

```
spi.close(0, 0)
```

To do basic reading and writing, you can use the following two functions

```
# Read X bytes  spi = spidev.SpiDev()
vals = spi.readbytes(X)
# Write X bytes
inputs = [0x01, 0x02, 0x03]
spi.writebytes(inputs)
```

For larger chunks of data, there are two other functions available. These are 'xfer()' and 'xfer2()'. The first one transfers the data at once, keeping the CE line asserted the whole time. The second one de-asserts and re-asserts the CE line after each byte is transferred. There is a low-level function available, called 'fileno()', that returns a file descriptor for the SPI device. This file descriptor can then be used with low-level file interfaces, like 'os.read()'. This provides yet another way of talking with peripheral devices.

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

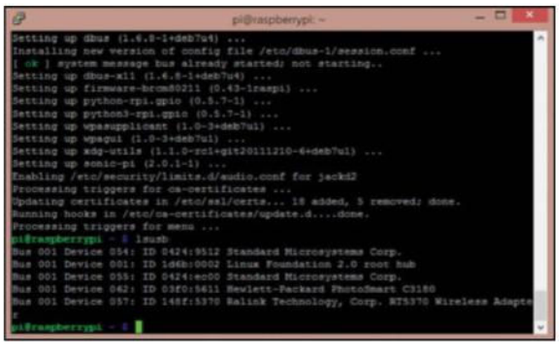
Wireless printing has made it possible to print to devices stored in cupboards, sheds and remote rooms. It has generally shaken up the whole process of printing and enabled output from smartphones, tablets, laptops and desktop computers alike. But you don't have to own a shiny new printer for this to 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
```

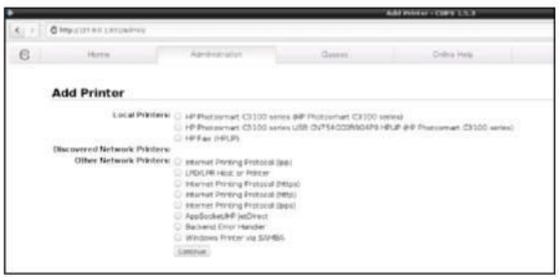
With a print server created, begin configuration by adding default user 'pi' to the printer admin group:

```
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

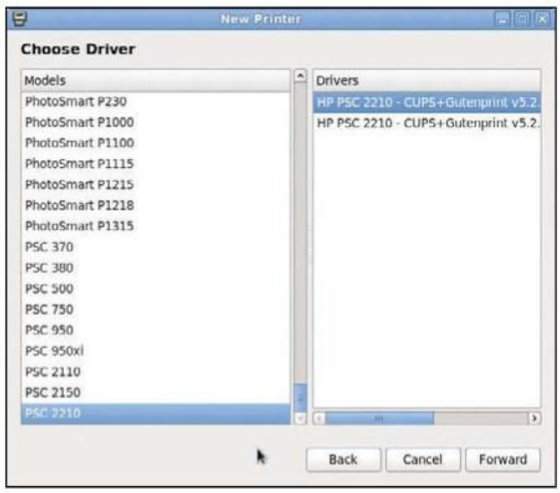
07 Join a Windows workgroup

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

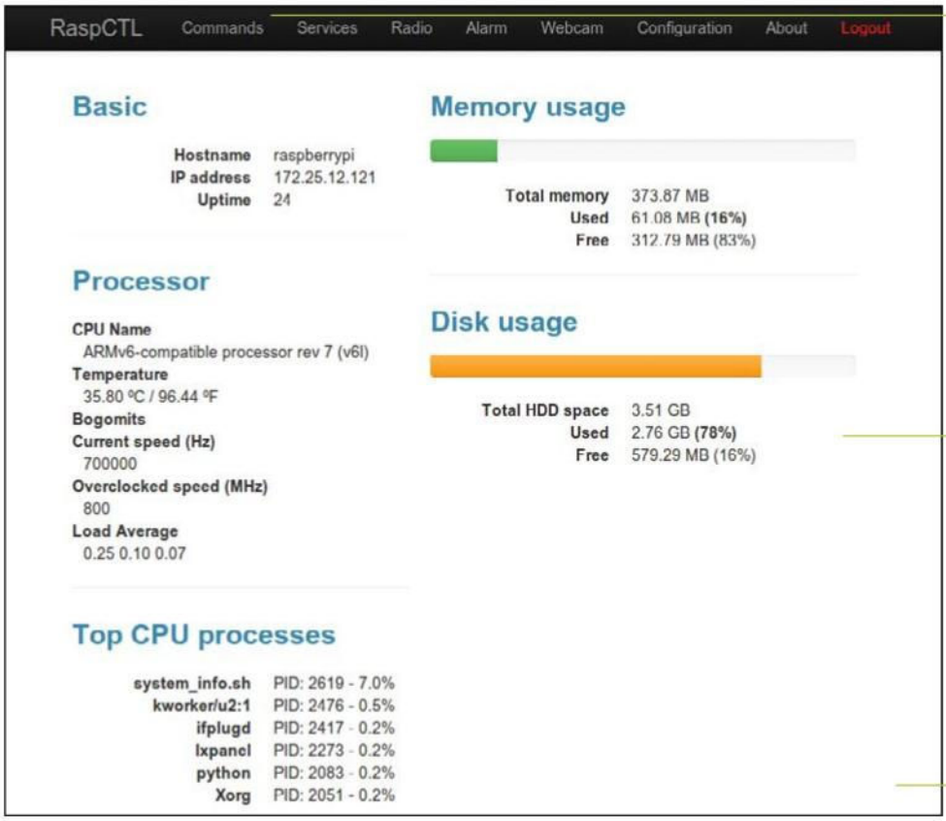
It's also possible to print wirelessly from your iPad using Apple's AirPrint system. To do this, you need to add the Avahi Discover software:

```
sudo apt-get install avahi-discover
```

Your wireless printer will now be discoverable from your iPad or iPhone and will be ready to print.

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

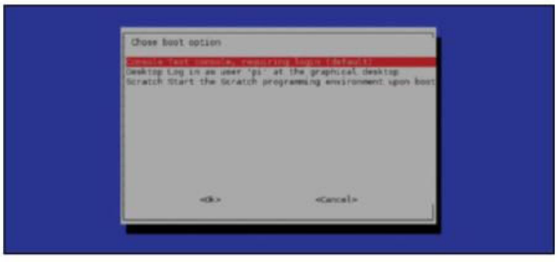
What you'll need

- 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.

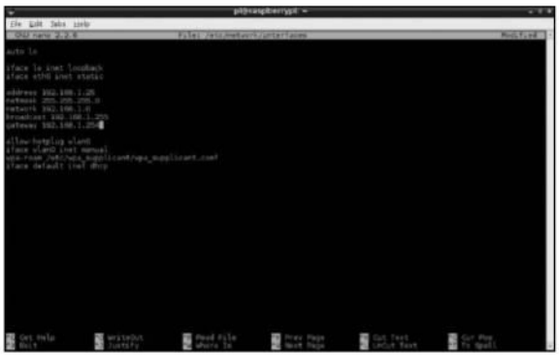


02 Edit the IP

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

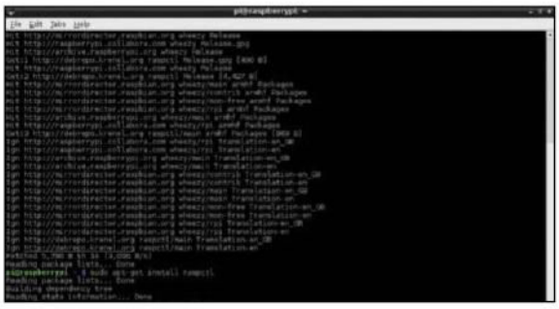
Add the following lines under the iface line with your relevant details:

```
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
```

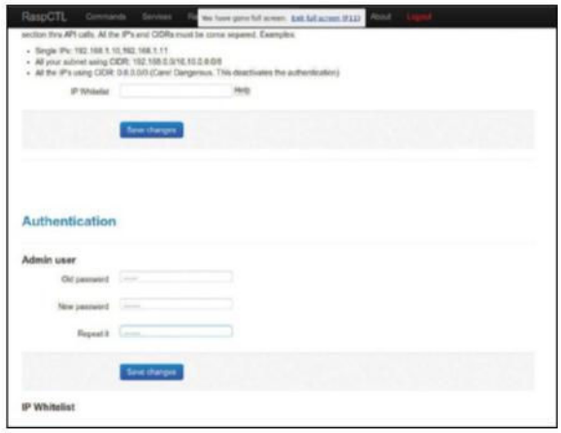
...and finally install the software with:

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

06 Access your Raspberry Pi

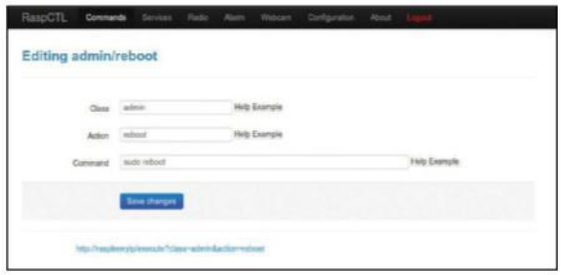
Now the software is installed you can start to access your Raspberry Pi from anywhere on your network. To do this type the following into your address bar, with the IP being the one we set up earlier:

```
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.

Turn your Pi into a motion sensor with SimpleCV

Learn how to implement facial recognition into your Raspberry Pi using Python and a webcam

Why Python?

It's the official language of the Raspberry Pi. Read the docs at **python.org/doc**

The Kinect has proven a popular piece of tech to use with the Raspberry Pi. But not everyone has access to this kind of hardware. Another class of project that is popular with Raspberry Pis is using USB cameras to create monitors of one form or another. A lot of these projects use command line applications to talk to the USB camera and generate images or movies that are used as part of the system. But what if you are writing your own program in Python and you want to add some form of image system to your code? Luckily, there are several modules available for you to choose from. In this article, we will take a look at using SimpleCV to get your program to talk with the USB camera. SimpleCV is built on top of OpenCV, making it easier to use for common tasks. Assuming you are using Raspbian, you can go to the main page for SimpleCV (**www.simplecv.org**) and download a DEB file. To install it, you can simply run:

```
sudo dpkg -i SimpleCV-1.31.deb
```

Before you do, however, you will want to install all of the dependencies. You can do that with the command:

```
sudo apt-get install python
python-support python-numpy
python-scipy ipython python-
opencv python-pygame python-
setuptools
```

You can check that everything worked by running the command 'simplecv' at the command line. This will start Python up and run the interactive shell that is provided by the SimpleCV module. You can then try connecting to your USB camera and pulling images from it.

Now that everything should be up and running, how do you actually use it in your own code? You can load all of the available functions and objects into the global scope with the command:

```
from SimpleCV import *
```

Making sure that you have your USB camera plugged in, you can now create a camera object with:

```
cam = Camera()
```

This will load the required drivers, and initialise the camera so that it is ready to start taking pictures. Once this object creation returns, you can grab an image from the camera with:

```
img = cam.getImage()
```

At least in the beginning, when you are experimenting, you may want to see what this image looks like. You can do this with:

```
img.show()
```

You will, of course, need to have a GUI up and running in order to actually see the movie. Otherwise, you will get an error when you try and call 'img.show()'. Don't forget that you can always pull up documentation with commands like:

```
help(cam)
help(img)
```

> SimpleCV is built on top of OpenCV, making it easier to use for common tasks

With the 'Image' object, you can do some basic processing tasks right away. You can scale an image by some percentage, say 90%, with 'img.scale(90,90)'. You can also crop an image by giving it a start location and saying how many pixels across and how many up and down you want to crop to. This looks like 'img.crop(100,100,50,50)'. SimpleCV has the location (0,0) as the top-left corner of an image.

The really interesting functionality in SimpleCV is the ability to find features within an image and to work with them. One of the clearest features you can look for is blobs, where blobs are defined as continuous light regions. The function 'img.findBlobs()' will search the captured image for all blobs and return them as a FeatureSet. You can set the minimum number of pixels to consider a single blob, the maximum number of pixels, as well as a threshold value. If you are looking at a region that has some hard edges, you can use the function 'img.findCorners()'. This function will return a FeatureSet of all of the corners within the captured image. A very simple monitor program could use one of these functions to see if there is any motion happening. If there is, then the set of blobs or corners will change from one frame to another. Of course, a little more reading will lead you to the 'img.findMotion()' function. This function will take two subsequent images and see if any motion can be detected going from one to the other. The default method is to use a block matching algorithm, but you can also use either the Lucas-Kanade method or the Horn-Schunck method.

The above methods will let you know some features of the captured images, and if any kind of motion has occurred. But what if you are more interested in identifying whether people have been moving around? Maybe you have an area you need to secure from espionage.

Above Any basic USB webcam or surveillance monitor will do for this

You can look for blobs – continuous light regions

In this case, you can use the function 'img.findSkintoneBlobs()'. You can use a binarise filter threshold to set what constitutes a skin tone. If you need to do more, you have access to all of the underlying OpenCV functionality. One of these more advanced functions is face recognition. You can use the function 'img.findHaarFeatures()' to look for a known type of object. If you wanted to look for faces, you could use something like:

```
faces = HaarCascade("./SimpleCV/Features/HaarCascades/face.xml","myFaces")
img.findHaarFeatures(faces)
```

When you start developing these types of programs, one thing that might come into play is timing issues. You want to be sure that your code is fast enough to catch everyone that may be moving through the field of the camera. In order to figure out what is costing time, you need to be able to profile your code. The shell in SimpleCV provides a feature called 'timeit' that will give you a quick and dirty profiling tool that you can use while you are experimenting with different algorithms. So, as an example, you can see how long the 'findBlobs()' function takes on your Raspberry Pi with something like:

```
img = cam.getImage()
timeit img.findBlobs()
```

Once you find and fix the bottlenecks in your code, you can create the end product for your final version.

With this article, you should now have enough to start using cameras from within your own programs. We have only been able to cover the bare essentials, however, so don't forget to go check out the documentation covering all of the other functionality that is available in the SimpleCV module.

Full code listing

```
# SimpleCV provides a simple interface to OpenCV
# First, we will import everything into the local
# namespace

from SimpleCV import *

# Make sure your USB camera is plugged in,
# then you can create a camera object
cam = Camera()

# Getting an image from the camera is straightforward
img = cam.getImage()

# You can rescale this image to half its original size
img2 = img.scale(50,50)

# There are several features that you may want to look at

# You can extract a list of blobs
blobs = img.findBlobs()

# You can draw these blobs and see where they are on
# the image
blobs.draw()

# or a list of corners
corners = img.findCorners()

# If you want to identify motion, you will need two
# frames
img2 = cam.getImage()

# You can get a FeatureSet of motion vectors with
motion = img2.findMotion(img)

# Face recognition is possible too. You can get a list of
# the types of features you can look for with
img.listHaarFeatures()

# For faces, you can generate a Haar Cascade
faces = HaarCascade('face.xml')

# Now you can search for faces
found_faces = img.findHaarFeatures(faces)

# You can load image files with the Image class
my_img = Image('my_image.jpg')

# You can save images to the hard drive, too
img.save('camera.png')
```

Importing

SimpleCV is built on top of OpenCV and provides a simplified set of functions. But what can you do if you have more complicated work to do? You always have the option of using OpenCV directly to gain access to the full set of functions. You can import the module into the local namespace with:

```
from cv2 import *
```

Not only do you have the usual image manipulation functions and the feature recognition tools, but you also have the ability to process video. You can use meanshift and camshift to do colour based motion detection. There are functions to look at optical flow. These look at apparent motions in a video, from one frame to the next, that are caused by either the object moving or the camera moving. You can even subtract the background from a moving foreground object. This is a common preprocessing step in vision systems. You can even construct 3D information from a set of stereo images gathered by a pair of cameras. With OpenCV, you really can deal with almost any vision problem you might be tackling.

What you'll need

- Raspberry Pi 2
- **USB sound card** (we used a Behringer UCA202)

Code a simple synthesiser

Learn how to write a simple polyphonic synthesiser (and the theory behind it) using Python and Cython

We are going to take you through the basics of wavetable synthesis theory and use that knowledge to create a real-time synthesiser in Python. At the moment, it is controlled by the computer keyboard, but it could easily be adapted to accept a MIDI keyboard as input.

The Python implementation of such a synthesiser turns out to be too slow for polyphonic sound (ie playing multiple notes at the same time) so we'll use Cython, which compiles Python to C so that you can then compile it to native machine code to improve the performance. The end result is polyphony of three notes, so this is not intended for use as a serious synthesiser. Instead, this tutorial will enable you to become familiar with synthesis concepts in a comfortable language: Python.

Once you're finished, try taking this project further by customising the mapping to better fit your keyboard layout, or tweaking the code to read input from a MIDI keyboard.

01 Install packages

Using the latest Raspbian image, install the required packages with the following commands:

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install python-pip python2.7-dev
portaudio19-dev
sudo pip install cython pyaudio
```

The final step compiles Cython and PyAudio from source, so you might want to go and do something else while it works its magic.

02 Disable built-in sound card

We had issues getting the Raspberry Pi's built-in sound card to work reliably while developing the synthesis code. For

Cython

Cython is a tool that compiles Python down to the C code that would be used by the interpreter to run the code. This has the advantage that you can optimise some parts of your Python code into pure C code, which is significantly faster. This is achieved by giving C types, such as int, float and char, to Python variables.

Once you have C code it can then be compiled with a C compiler (usually GCC) which can optimise the code even further. A downside to using Cython is that you can't run Cython optimised code with a normal Python interpreter. Cython is a nice compromise because you get a similar simplicity to Python code but higher performance than usual. Cython has a profiler which you can run using:

```
cython -a synth.pyx
```

The profiler outputs a html file which shows where any optimisations can be made, giving you an insight into just how much overhead using Python introduces. For more details you can go to **http://cython.org**.

that reason, we are using a USB sound card and will disable the built-in card so that the default card is the USB one:

```
sudo rm /etc/modprobe.d/alsa*
sudo editor /etc/modules
```

Change 'snd-bcm2835' to '#snd-bcm2835' and save, then:

```
sudo reboot
```

03 Test sound card

Now we can test the USB sound card. Type **alsamixer** and then ensure that the volume is set to a comfortable level. If you're plugging speakers in, you'll probably want it set to 100%. Then type **speaker-test**, which will generate some pink noise on the speakers. Press Ctrl+C to exit once you are happy that it's working.

04 Start project

Start by creating a directory for the project. Then download one cycle of a square wave that we will use as a wavetable, like so:

```
mkdir synth
cd synth
wget liamfraser.co.uk/lud/synth/square.wav
```

05 Create compilation script

We need a script that will profile our Python code (resulting in synth.html). Generate a Cython code for it and finally compile the Cython code to a binary with GCC:

```
editor compile.sh:
#!/bin/bash
cython -a synth.pyx
cython --embed synth.pyx
gcc -march=armv7-a -mfpu=neon-vfpv4 -mfloat-
abi=hard -O3 -I /usr/include/python2.7 -o synth.
bin synth.c -lpython2.7 -lpthread
```

(Notice the options that tell the compiler to use the floating point unit.) Make it executable with:

```
chmod +x compile.sh
```

Full code listing

```
#!/usr/bin/python2

import pyaudio
import time
from array import *
from cpython cimport array as c_array
import wave
import threading
import tty, termios, sys
```

Step 07

```
class MIDITable:
    # Generation code from
    # http://www.adambuckley.net/software/beep.c

    def __init__(self):
        self.notes = []
        self.fill_notes()

    def fill_notes(self):
        # Frequency of MIDI note 0 in Hz
        frequency = 8.175799

        # Ratio: 2 to the power 1/12
        ratio = 1.0594631

        for i in range(0, 128):
            self.notes.append(frequency)
            frequency = frequency * ratio

    def get_note(self, n):
        return self.notes[n]
```

Step 08

```
cdef class ADSR:
    cdef float attack, decay, sustain_amplitude
    cdef float release, multiplier
    cdef public char state
    cdef int samples_per_ms, samples_gone

    def __init__(self, sample_rate):
        self.attack = 1.0/100
        self.decay = 1.0/300
        self.sustain_amplitude = 0.7
        self.release = 1.0/50
        self.state = 'A'
        self.multiplier = 0.0
        self.samples_per_ms = int(sample_rate / 1000)
        self.samples_gone = 0

    def next_val(self):
        self.samples_gone += 1
        if self.samples_gone > self.samples_per_ms:
            self.samples_gone = 0
        else:
            return self.multiplier

        if self.state == 'A':
            self.multiplier += self.attack
            if self.multiplier >= 1:
                self.state = 'D'
        elif self.state == 'D':
            self.multiplier -= self.decay
            if self.multiplier <= self.sustain_amplitude:
                self.state = 'S'
        elif self.state == 'R':
            self.multiplier -= self.release

        return self.multiplier
```

Full code listing (Cont.)

Step 09

```
cdef class Note:
    cdef int wavetable_len
    cdef float position, step_size
    cdef c_array.array wavetable
    cdef public float freq
    cdef public object adsr
    cdef public int off

    def __init__(self, wavetable, samplerate, freq):
        # Reference to the wavetable we're using
        self.wavetable = wavetable
        self.wavetable_len = len(wavetable)
        # Frequency in Hz
        self.freq = freq
        # The size we need to step though the wavetable
        # at each sample to get the desired frequency.
        self.step_size = self.wavetable_len * \
                         (freq/float(samplerate))
        # Position in wavetable
        self.position = 0.0
        # ADSR instance
        self.adsr = ADSR(samplerate)
        # Is this note done with
        self.off = 0

    def __repr__(self):
        return ("Note: Frequency = {0}Hz, "
                "Step Size = {1}").format(self.freq,
                                          self.step_size)

    cpdef int next_sample(self):
        # Do the next sample
        cdef int pos_int, p1, p2, interpolated
        cdef int out_sample = 0
        cdef float pos_dec
        cdef float adsr

        adsr = self.adsr.next_val()
        # Need to turn the note off
        # synth will remove on next sample
        if adsr < 0:
            self.off = 1
            return out_sample

        pos_int = int(self.position)
        pos_dec = self.position - pos_int

        # Do linear interpolation
        p1 = self.wavetable[pos_int]
        p2 = 0

        # Wrap around if the first position is at the
        # end of the table
        if pos_int + 1 == self.wavetable_len:
            p2 = self.wavetable[0]
        else:
            p2 = self.wavetable[pos_int+1]

        # Inerpolate between p1 and p2
        interpolated = int(p1 + ((p2 - p1) * pos_dec))
        out_sample += int(interpolated * adsr)

        # Increment step size and wrap around if we've
        # gone over the end of the table
        self.position += self.step_size
        if self.position >= self.wavetable_len:
            self.position -= self.wavetable_len
```

06 Start to code

Our code file is going to be called synth.pyx. This extension tells Cython that it is not plain Python code (and as such, can't be ran in a normal Python interpreter). Create the file with your favourite editor and add the imports.

07 MIDI Table

To synthesise the standard note of a piano, we need a table of MIDI values. MIDI notes range from 0-127. MIDI note 60 is middle C on a piano. The MIDI Table class has a 'get note' function that returns the frequency of a note when you give it a MIDI note number.

Above A visual representation of an Attack, Decay, Sustain, Release curve

08 Attack, Decay, Sustain, Release

The ADSR class applies a volume curve over time to the raw output of an oscillator. It does this by returning a multiplier to the note that is a multiple between 0.0 and 1.0. The version we provide has an attack time of 100 ms, a decay time of 300 ms and a release time of 50 ms. You can try changing these values to see how it affects the sound.

The ADSR class does a lot of maths (44,100 times per second, per note). As such, we want to give types to all of the variables so that the maths can be optimised into a raw C loop where possible, because Python has a massive amount of overhead compared to C. This is what the cdef keyword does. If cdef public is used, then the variable can also be accessed from inside Python as well.

09 Generate notes

The **note** class is the core of our synthesiser. It uses the wavetable to generate waves of a specific frequency. The synthesiser asks the note class for a sample. After generating a sample, the ADSR multiplier is applied and then returned to the synthesiser. The maths of this are explained in the synthesis theory boxout on the opposite page.

The note class does as much maths as the ADSR class, so it is optimised as much as possible using cdef keywords. The cpdef keyword used for the next_sample function means that the function can be called from a non-cdef class. However, the main synth class is much too complicated to give static types to absolutely everything.

10 The audio flow

This synth class is the main class of the application. It has two sample buffers that are the length of the buffer size. While one buffer is being played by the sound card, the other buffer is being filled in a different thread. Once the sound card has played a buffer, the callback function is called. References to the buffers are swapped and the buffer that has just been filled is returned to the audio library.

The smaller the buffer size, the lower the latency. The Raspbian image isn't optimised for real time audio by default so you may have trouble getting small buffer sizes. It also depends on the USB sound card used.

Full code listing (Cont.)

Step 09

```
        return out_sample

class Synth:
    BUFSIZE = 1024
    SAMPLERATE = 44100

    def __init__(self):
        self.audio = pyaudio.PyAudio()

        # Create output buffers
        self.buf_a = array('h', [0] * Synth.BUFSIZE)
        self.buf_b = array('h', [0] * Synth.BUFSIZE)
        # Oldbuf and curbuf are references to buf_a or
        # buf_b, not copies. We're filling newbuf
        # while playbuf is playing
        self.playbuf = self.buf_b
        self.newbuf = self.buf_a

        self.load_wavetable()
        self.notes = []
        self.notes_on = []

        # The synth loop will run in a separate thread.
        # We will use this condition to notify it when
        # we need more samples
        self.more_samples = threading.Event()
        self.exit = threading.Event()

        # MIDI table of notes -> frequencies
        self.midi_table = MIDITable()

    def stop(self):
        print "Exiting"
        self.exit.set()
        self.stream.stop_stream()
        self.stream.close()

    def stream_init(self):
        self.stream = self.audio.open(
                    format = pyaudio.paInt16,
                    channels = 1,
                    rate = Synth.SAMPLERATE,
                    output = True,
                    frames_per_buffer = Synth.BUFSIZE,
                    stream_callback = self.callback)

    def load_wavetable(self):
        # Load wavetable and assert it is the
        # correct format
        fh = wave.open('square.wav', 'r')
        assert fh.getnchannels() == 1
        assert fh.getframerate() == Synth.SAMPLERATE
        assert fh.getsampwidth() == 2 # aka 16 bit

        # Read the wavedata as a byte string. Then
        # need to convert this into a sample array we
        # can access with indexes
        data = fh.readframes(fh.getnframes())
        # h is a signed short aka int16_t
        self.wavetable = array('h')
        self.wavetable.fromstring(data)

    def swap_buffers(self):
        tmp = self.playbuf
        self.playbuf = self.newbuf
        self.newbuf = tmp
        # Setting the condition makes the synth loop
```

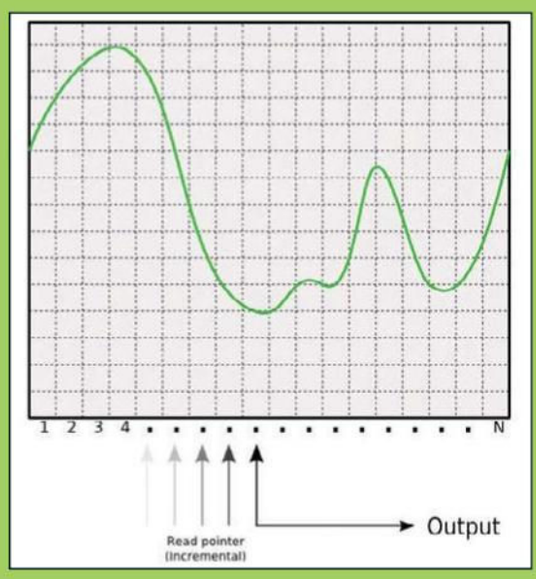
Synthesis theory

Wavetable synthesis is where you use a single cycle of a wave as a lookup table to synthesise sound. In this case we have a square wave, but you can load any wave shape you like. CD-quality audio has a sample rate of 44,100 Hz, which is what we used in our implementation. At each sample, the synthesiser outputs a value from the wavetable and then increments a position pointer to the next value in the table. However, if the wavetable has a frequency of 440 Hz then we need to be able to step through it at arbitrary sizes (ie non-integer values). To achieve this, we use linear interpolation. Assuming the table had a frequency of 440 Hz and we wanted a frequency of 220 Hz, we'd need to step through the table at a step size of 0.5. This can be thought of as drawing a line between two values in the table and picking a value on the line as your output. As an example, if element 0 is 5 and element 1 is 10 then element 0.5 would be 5 + ((10-5) * 0.5), which gives us a value of 7.5. When you reach a position that goes over the end of the table, you wrap around and start again. There is no discontinuity as you're storing a single cycle of the wave in the table. The equation for step size is:

```
step_size = table_size * (note_frequency / sample_rate)
```

The wavetable oscillator gets us a note at the desired frequency, but it's always at maximum amplitude and will sound rough and unnatural. If you cut off a wave in the middle of a cycle there will be a pop or click, so this is where Attack, Decay, Sustain and Release envelopes help. These change the amplitude of the raw oscillator output over time to sound more like an instrument. This is done by applying a fractional multiplier to the original sample point returned by the wave table oscillator. Having a release time from 100% volume to 0% means that a note will fade out smoothly when it's turned off. With the right ADSR curves and the correct wavetable, a synthesiser can sound very similar to real instruments.

More information can be found at: bit.ly/1Kgl9dp.



Above Here's one cycle of a wavetable oscillator

Performance issues

Python introduces a number of performance issues compared to a native synthesiser implementation that is written in C or C++. Cython has been used in our implementation to try and mitigate these issues but it is nowhere near enough. As a rough comparison, our expert worked on a synthesis project targeting 100 Mhz ARM processors that were programmed in C and could get around 30 notes of polyphony, compared to three in this implementation on a 900 Mhz ARM core.

A major issue is that the sound card uses 16-bit signed integers to represent a sample. However, Python doesn't natively support this type. To pass the data to the audio library it needs to be encoded from an array of integers into a byte string. Then at the other end, the Python that talks to the audio library will decode this byte string back into an integer array. If it was written in C or another lower-level language like C++ or Rust, the sample could be passed almost directly to the audio hardware.

Another issue is that Python has a large function call overhead. In compiled languages, this can be optimised out by compiling function calls in line with the caller (effectively, copying the code from the function into the caller). Variable access also has overhead because of all the type checking required. There is also the overhead of the garbage collector, which destroys objects when there are no longer references to them.

A major issue is that the sound card uses 16-bit signed integers to represent a sample. However, Python doesn't support this type

Full code listing (Cont.)

```
        # generate more samples
        self.more_samples.set()

    def callback(self, in_data, frame_count,
                 time_info, status):
        # Audio card needs more samples so swap the
        # buffers so we generate more samples and play
        # back the play buffer we've just been filling
        self.swap_buffers()
        return (self.playbuf.tostring(),
                pyaudio.paContinue)

Step 11
    def do_sample(self, int i):
        cdef int out_sample = 0
        # Go through each note and let it add to the
        # overall sample
        for note in self.notes:
            if note.off:
                self.notes.remove(note)
            else:
                out_sample += note.next_sample() >> 3

        self.newbuf[i] = out_sample

    def synth_loop(self):
        cdef int i

        while self.exit.is_set() == False:
            # For each sample we need to generate
            for i in range(0, Synth.BUFSIZE):
                self.do_sample(i)

            # Wait to be notified to create more
            # samples
            self.more_samples.clear()
            self.more_samples.wait()

    def start(self):
        self.stream_init()
        # Start synth loop thread
        t = threading.Thread(target=self.synth_loop)
        t.start()

Step 12
    def freq_on(self, float freq):
        n = Note(self.wavetable, Synth.SAMPLERATE,
                 freq)
        print n
        self.notes.append(n)

    def freq_off(self, float freq):
        # Set the ADSR state to release
        for n in self.notes:
            if n.freq == freq:
                n.adsr.state = ord('R')

    def note_on(self, n):
        self.freq_on(self.midi_table.get_note(n))
        self.notes_on.append(n)
```

11 Synth loop

The start method of the synth class initialises the audio hardware and then starts the synth_loop method in its own thread. While the exit event is set to false, the do_sample function is called.

The do_sample function loops through the notes that are currently turned on and asks for a sample from each one. These samples are shifted right by three (ie divided by 2^3) and added to out_sample. The division ensures that the output sample can't overflow (this is a very primitive method of adding notes together, but it works nonetheless).

The resulting sample is then put in the sample buffer. Once the buffer is full, the more_samples condition is cleared and the synth_loop thread waits to be notified that the buffer it has just built has been sent to the audio card. At this point, the synth can fill up the buffer that has just finished playing and the cycle continues.

12 Turn on notes

There are both note_on/off and freq_on/off functions that enable either MIDI notes or arbitrary frequencies to be turned on easily. Added to this, there is also a toggle note function which keeps track of MIDI notes that are on and turns them off if they are already on. The toggle note method is used specifically for keyboard input.

13 Add keyboard input

For keyboard input, we needed the ability to get a single character press from the screen. Python's usual input code needs entering before returning to the program. Our code for this is inspired by: **https://code.activestate.com/recipes/577977-get-single-keypress**.

There is a mapping of letters on a keyboard to MIDI note numbers for an entire keyboard octave. We have tried to match the letter spacing to how a piano is laid out to make things easier. However, more innovative methods of input are left as an exercise to the reader.

14 Put it all together

The main function of the program creates an instance of the synth class and then starts the audio stream and synth loop thread. The start function will then return control to the main thread again.

At this point we create an instance of the KB input class and enter a loop that gets characters and toggles the corresponding MIDI note on or off. If the user presses the Q key, that will stop the synth and end the input loop. The program will then exit.

15 Compile the code

Exit your editor and run the compile script by typing the following command:

```
./compile.sh
```

This may take around 30 seconds, so don't worry if it isn't instant. Once the compilation has finished, execute the synth.bin command using:

```
./synth.bin
```

Pressing keys from A all the way up to K on the keyboard will emulate the white keys on the piano. If you press a key again the note will go off successfully.

Full code listing (Cont.)

Step 12

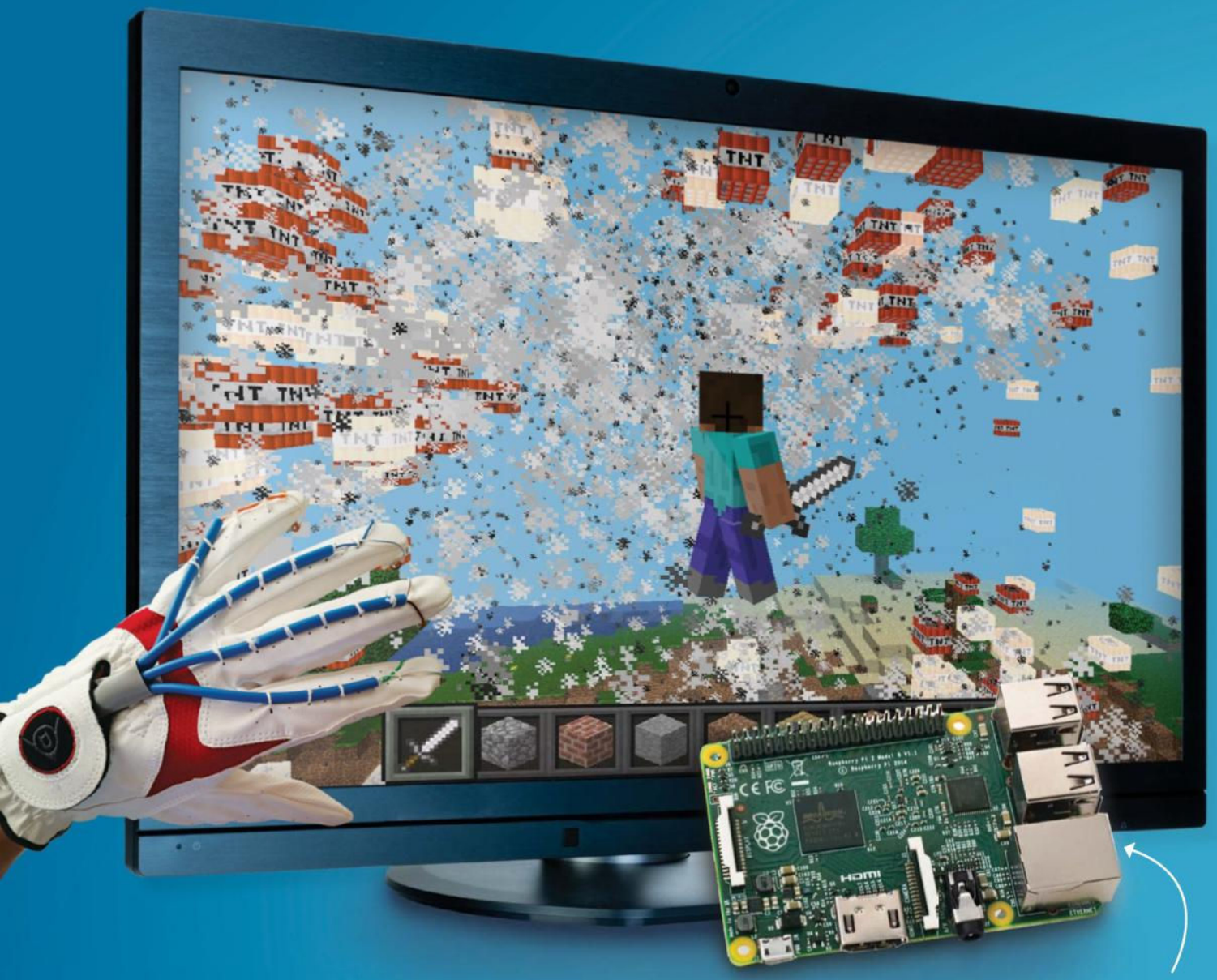
```
    def note_off(self, n):
        self.freq_off(self.midi_table.get_note(n))
        self.notes_on.remove(n)

    def toggle_note(self, n):
        if n in self.notes_on:
            print "note {0} off".format(n)
            self.note_off(n)
        else:
            print "note {0} on".format(n)
            self.note_on(n)
```

Step 13

```
class KBInput:
    def __init__(self, synth):
        self.synth = synth

        self.keymap = {'a' : 60, 'w' : 61, 's' : 62,
                       'e' : 63, 'd' : 64, 'f' : 65,
                       't' : 66, 'g' : 67, 'y' : 68,
                       'h' : 69, 'u' : 70, 'j' : 71,
                       'k': 72}
        self.notes_on = []

    @staticmethod
    def getch():
        fd = sys.stdin.fileno()
        old_settings = termios.tcgetattr(fd)
        try:
            tty.setraw(fd)
            ch = sys.stdin.read(1)
        finally:
            termios.tcsetattr(fd, termios.TCSADRAIN,
                              old_settings)
        return ch

    def loop(self):
        while True:
            c = self.getch()

            if c == 'q':
                self.synth.stop()
                return

            if c in self.keymap:
                n = self.keymap[c]
                self.synth.toggle_note(n)

if __name__ == "__main__":
    s = Synth()
    s.start()
    kb = KBInput(s)
    kb.loop()
```

```
note 60 on
Note: Frequency = 261.625640869Hz, Step Size = 0.599187970161
note 60 off
note 64 on
Note: Frequency = 329.627685547Hz, Step Size = 0.754929602146
note 64 off
note 65 on
Note: Frequency = 349.228363037Hz, Step Size = 0.799820065498
note 65 off
note 67 on
Note: Frequency = 391.995574951Hz, Step Size = 0.897767663002
note 67 off
Exiting
```

Above The simple user interface. Notice how the step size in the wavetable varies with frequency



148
Modify a retro radio

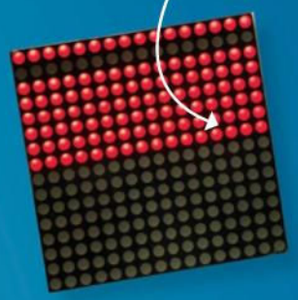
126
Enhance Minecraft with your Pi

Electronics

134
Add gesture control to your Pi

120
Turn your Pi into an oscilloscope

130
Assemble a LED display



142
Fly a Pi-powered quadcopter

148
Build a motorised alarm clock

148
Try out a light gun game


Queens Park
Talbot Village
Bennett Road
Wenner Way
Wenner Way
Wenner Way
Wenner Way
Garfield Avenue
Lincoln Avenue
Washington Avenue
St Mary's Road
Curzon Road
Spring Road
Spring Road
Windham Road
Windham Road
Wilson Road
Boscombe Grove Road
South Road
Ashley Road
Sound BLASTER
Raspberry Pi Model B+ V1.2
© Raspberry Pi 2014
GPIO
HDMI
Micro USB Power

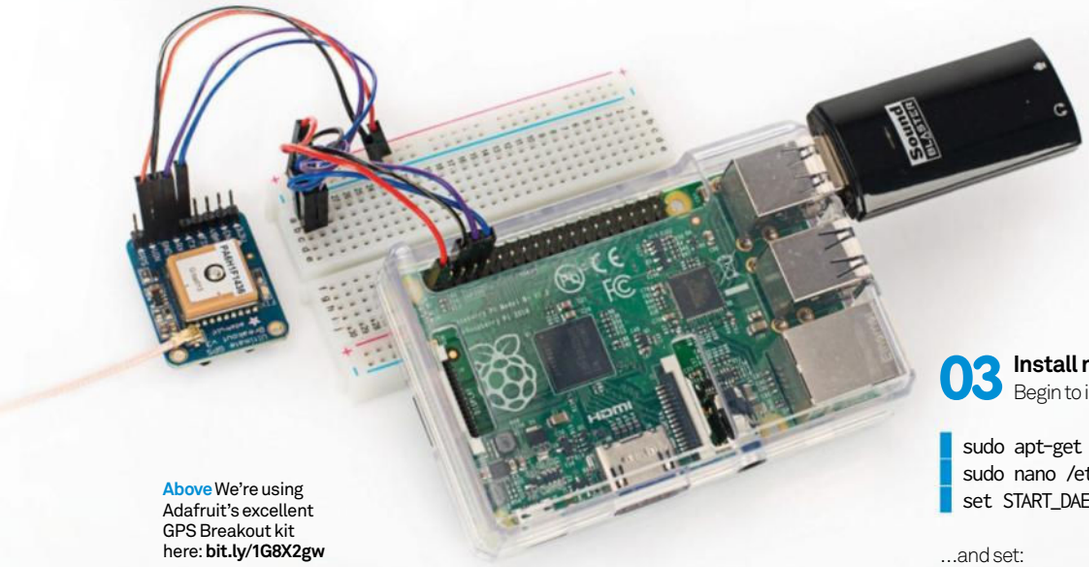
Build a Raspberry Pi car computer

Make your own touchscreen navigation system that gives directions, local weather reports and plays music

Cars are getting clever. These days, with smart navigation interfaces built into new cars, you don't need to go out and buy yourself a TomTom to get help with directions. But if you've got a Raspberry Pi then you don't even need to buy that – let alone a new car!

In this project we will show you how to build your own car computer with your Pi, a quality touchscreen like the 9-inch model from SainSmart that we're using here, and a few other bits like a GPS module and USB 3G modem. Your CarPi will be able to use open source navigation software Navit to show your route map on screen, plus speech synthesis to read out directions, and it will also be able to check your location and give you weather reports. It'll work as a music player too, of course.

It's an ambitious project, but you will gain a solid understanding of custom-made interfaces, navigation software and geolocation data, touchscreen calibration, speech synthesis and more. While you don't have to use the same SainSmart screen as us, we do recommend it for this project as it is one of the few large touchscreens out there for the Pi. There are more improvements at the end too, so check the components list, make sure you've got everything and let's get started!

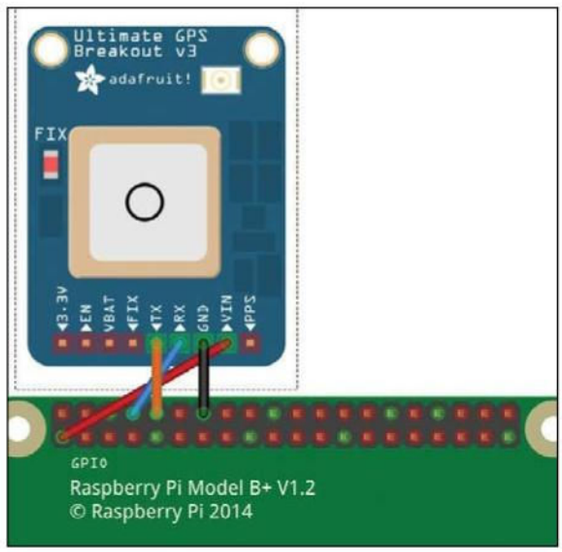


Above We're using Adafruit's excellent GPS Breakout kit here: **bit.ly/1G8X2gw**

01 Basic configuration

Boot up your Raspberry Pi and expand the filesystem using **raspi-config**. Go to Advanced Options and disable the Serial connection – you'll need this to talk to the GPS module later. In raspi-config, enable X at boot as the pi user. Say Yes to reboot. Once rebooted, ensure your packages are up to date with:

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



02 Connect GPS module

Solder the pin headers onto the Adafruit GPS module. You can also solder the battery connector which is used to keep the device partially active, giving a faster fix. You only need to use 4 pins: 3.3V, ground, serial transmit and serial receive. Power the Pi off again before connecting anything.

As we are using GPS, the antenna will have to go outside or under a window to gain signal. Connect the antenna to the board and power everything back on. The light on the GPS module will flash frequently while finding a fix. Once it has one, it will blink every 15 seconds.

03 Install navigation software

Begin to install the Navit navigation software by entering:

```
sudo apt-get install navit gpsd gpsd-clients espeak
sudo nano /etc/default/gpsd
set START_DAEMON="true"
```

...and set:

```
DEVICES="/dev/ttyAMA0"
```

Start the GPS daemon with:

```
sudo /etc/init.d/gpsd start
```

You can check it's working by looking at the GPS data with:

```
cgps -s
```

04 Connect the screen

The SainSmart screen doesn't come with any written instructions. Instead there is a YouTube video on their website with details about how to put it together: **bit.ly/1DF6eJJ**. The important part is that the DC power supply should be 12V.

05 Set the screen resolution

We will have to force the correct resolution (1024x600) for the screen by editing /boot/config.txt with sudo. To do so, add the following options:

```
framebuffer_width=1024
framebuffer_height=600
hdmi_force_hotplug=1
hdmi_cvt=1024 600 60 3 0 0 0
hdmi_group=2
hdmi_mode=87
```

For the changes to properly take effect you will need to reboot with `sudo reboot`.

06 Download kernel source

To start the touchscreen, you need to compile an extra kernel module to support it. The program rpi-source (**github.com/notro/rpi-source/wiki**) will find the source of your kernel. Install rpi-source with:

```
sudo wget https://raw.githubusercontent.com/notro/
rpi-source/master/rpi-source -O usr/bin/rpi-source
&& sudo chmod +x /usr/bin/rpi-source && /usr/bin/
rpi-source -q -tag-update
```

Then run `rpi-source` to get the source of the running kernel.

07 Update GCC

Recent Raspberry Pi kernels are compiled with GCC 4.8. Raspbian only comes with 4.6 so you will have to install 4.8 to continue with the following steps. Do this by entering:

```
sudo apt-get install -y gcc-4.8 g++-4.8 ncurses-dev
```

Then you have to set GCC 4.8 as the default:

```
sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-4.6 20
sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-4.8 50
sudo update-alternatives --install /usr/bin/g++ g++ /usr/bin/g++-4.6 20
sudo update-alternatives --install /usr/bin/g++ g++ /usr/bin/g++-4.8 50
```

08 Pick the module to compile

Rpi-source puts the kernel source in a folder called 'linux'. To choose the USB Touchscreen Driver, enter the following:

```
cd linux
make menuconfig
Device Drivers -> Input device support -> Generic input layer (needed for keyboard, mouse, ...) -> Touchscreens (press space to include) -> USB Touchscreen Driver (press M to make module)
```

Once you've done that, you then need to make sure you save your changes as '.config' and run scripts/diffconfig to see the differences.

09 Compile and install the module

Now you need to compile and install the module. Do so by entering:

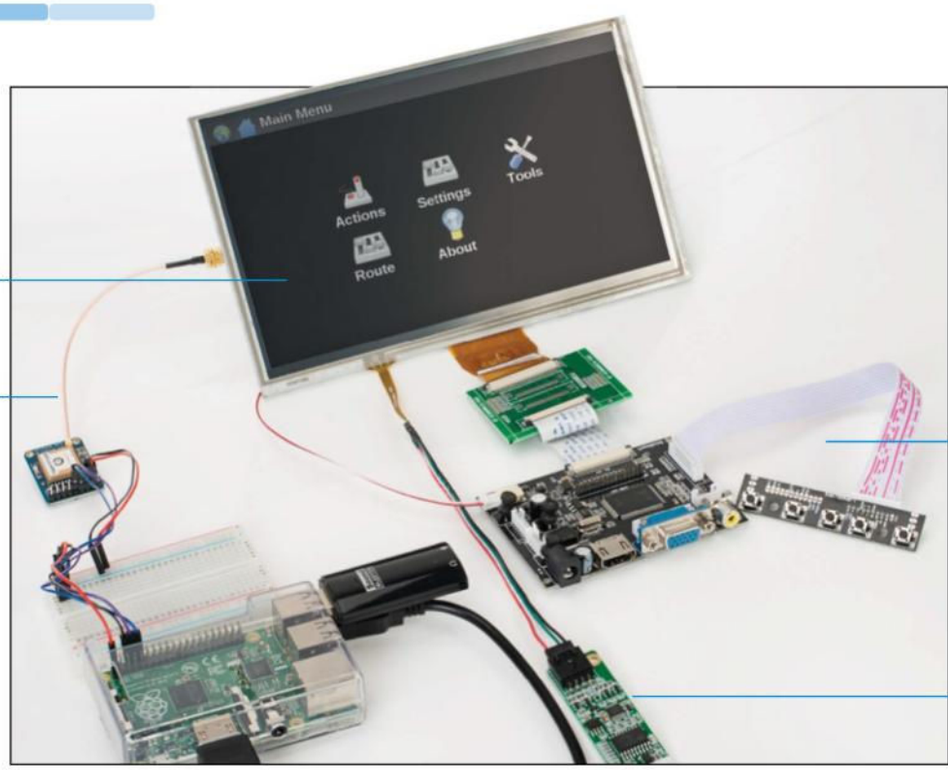
```
make prepare
make SUBDIRS=drivers/input/touchscreen modules
sudo make SUBDIRS=drivers/input/touchscreen modules_install
sudo depmod
```

If you unplug and reconnect the touchscreen, it should work fine but it will probably need calibrating.

Full code listing

```
#!/usr/bin/env python2

import os, sys, requests, pygame
from gps import *
from pygame.locals import *

class WeatherClient:
    apikey = "7232a1f6857090f33b9d1c7a74721"

    @staticmethod
    def latlon():
        gpsd = gps(mode=WATCH_ENABLE)

      # Needs better error handling
        try:
            while True:
                report = gpsd.next()
                if report['class'] == 'TPV':
                    gpsd.close()
                    return report['lat'], report['lon']
        except:
            return None, None

    @staticmethod
    def usefuldata(j):
        # Returns a string of useful weather data from a LOT of json
        d = j['data']['current_condition'][0]
        out = "Now - Temp: {0}C, Feels Like: {1}C, Description: {2}\n"\
              .format(d['temp_C'],
                      d['FeelsLikeC'],
                      d['weatherDesc'][0]['value'])

        hourly = j['data']['weather'][0]['hourly']
        hour_count = 1
        for h in hourly:
            out += ("+{0}hr - Temp: {1}C, Feels Like: {2}C, Chance of Rain:"
                    " {3}%, Description: {4}\n")\
                    .format(hour_count,
                            h['tempC'],
                            h['FeelsLikeC'],
                            h['chanceofrain'],
                            h['weatherDesc'][0]['value'])
        hour_count += 1

        # Rstrip removes trailing newline
        return out.rstrip()

    @staticmethod
    def update():
        errstr = "Error getting weather data"

        lat, lon = WeatherClient.latlon()
        if lat == None or lon == None:
            return errstr

        api_req = ("http://api.worldweatheronline.com/free/v2/weather.ashx"
        "?q={0}%2C{1}&format=json&key={2}").format(lat, lon,
                                                  WeatherClient.apikey)
        r = None
```

SainSmart's 9-inch HDMI/VGA touchscreen (**bit.ly/1Ciu4H9**) has a fantastic display and is perfect for all sorts of Pi projects



The screen control panel that comes with the SainSmart screen enables you to easily change the display settings (i.e. brightness, contrast, etc) as well as the input (i.e. HDMI, VGA, AV1, etc)

Adafruit's Ultimate GPS Breakout kit provides Navit and the weather function with the location data that they require

As well as the main controller board, the touch screen is connected to a four-line USB controller which then plugs into the Pi's USB port

10 Calibrate the touchscreen

At this point, you can easily calibrate the touchscreen by entering the following:

```
cd /etc/X11
sudo mkdir xorg.conf.d
cd xorg.conf.d
sudo nano 99-calibration.conf
```

...with the following content:

```
Section "InputClass"
    Identifier "calibration"
    MatchProduct "eGalax Inc. USB TouchController"
    Option    "SwapAxes"    "1"
    Option    "InvertX"     "1"
EndSection
```

Invert X actually inverts Y because the axes have been swapped around. Reboot again for these changes to occur. Now the calibration is roughly correct, download an input calibrator that Adafruit have packaged already.

```
wget http://adafruit-download.s3.amazonaws.com/
xinput-calibrator_0.7.5-1_armhf.deb
sudo dpkg -i xinput-calibrator_0.7.5-1_armhf.deb
DISPLAY=:0.0 xinput_calibrator
```

DISPLAY=:0.0 is useful because you can run the program from any terminal (including an SSH session) and have it appear on the touchscreen. Touch the points on the screen as prompted. Once the program is finished, you should get an output that is similar to the following:

```
Option    "Calibration"    "84 1957 270 1830"
```

Add it to the '99-calibration.conf' file that we created earlier just below the other Option entries.

11 Download maps

Navit needs maps; download them from **maps.navit-project.org**. You can either use the web browser on the Pi or download the map from another machine and copy it using **scp**. Use the predefined area option to select where you live. The smaller the area that you pick, the less data you will have to process. Here the UK has a map size of 608 MB. Now move the map to the navit folder:

```
mkdir -p /home/pi/.navit/maps
mv /home/pi/Downloads/$your_map /home/pi/.
navit/$country.bin
```

For example:

```
mv /home/pi/Downloads/osm_bbox_-9.7,49.6,2.2,61.2.bin
/home/pi/.navit/maps/UK.bin
```

12 Navit configuration

Sudo-edit /etc/navit/navit.xml with your favourite editor. Search for openstreetmaps. Now disable the sample map above, enable the openstreetmap mapset and set the data variable to where you just moved your map. In this case it looks like this:

```
<!-- Mapset template for openstreetmaps -->
<mapset enabled="yes">
<map type="binfile" enabled="yes" data="/home/
 pi/.navit/maps/UK.bin"/>
</mapset>
```

Then search for osd entries similar to:

```
<osd enabled="yes" type="compass"/>
```

...and enable the ones you want – we recommend enabling them all. You may want to zoom in closer than the default map layout. A zoom value of 64 is useful.

Embed the screen

We've looked at the PiTFT and the HDMIPi before, but the SainSmart touchscreen we're using here is uniquely suited to many embedded projects. It's larger than the PiTFT but also without the large bezels of the HDMIPi – and it's incredibly thin –so it's the kind of thing that is really useful for installation projects, whether that's something as simple as a photo slideshow in a real picture frame or a home automation control interface embedded into a cupboard door.

13 Sound configuration

Before configuring speech support for Navit, configure the external sound card. You have to stop the Broadcom module from loading and remove some Raspberry Pi-specific ALSA (Advanced Linux Sound Architecture). To do this, sudo-edit /etc/modprobe and comment out (i.e. prefix with a #):

```
snd-bcm2835
```

Then run:

```
sudo rm /etc/modprobe.d/alsa*
```

Reboot for the changes to take effect. Use alsamixer to set the volume on the if it's too quiet.

14 Download a voice

The speech synthesis software needs a voice and a proprietary binary. You can get both by completing the following steps:

```
sudo mkdir -p /usr/share/mbrola/voices/
wget http://www.tcts.fpms.ac.be/synthesis/mbrola/dba/en1/en1-980910.zip
unzip en1-980910.zip
sudo cp en1/en1 /usr/share/mbrola/voices
wget http://www.tcts.fpms.ac.be/synthesis/mbrola/bin/raspberri_pi/mbrola.tgz
tar zxvf mbrola.tgz
sudo mv mbrola /usr/local/bin/
```

15 Create speech script

Navit supports speech by running an external script and passing the text to speak as an argument. Create one using:

```
cd /home/pi/.navit
wget  http://liamfraser.co.uk/lud/carpi/chime.wav
touch speech.sh
chmod +x speech.sh
```

Now edit speech.sh:

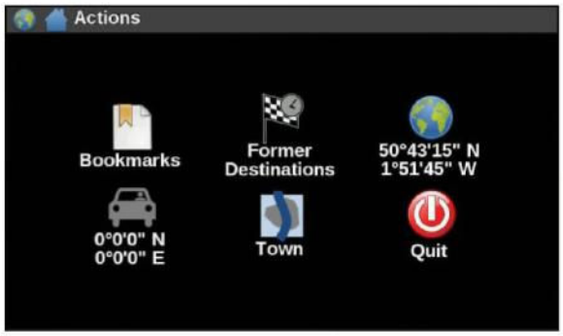
```
#!/bin/bash
aplay -r 44100 /home/pi/.navit/chime.wav
espeak -vmb-en1 -s 110 -a 150 -p 50 "$1"
```

Finally, test it with:

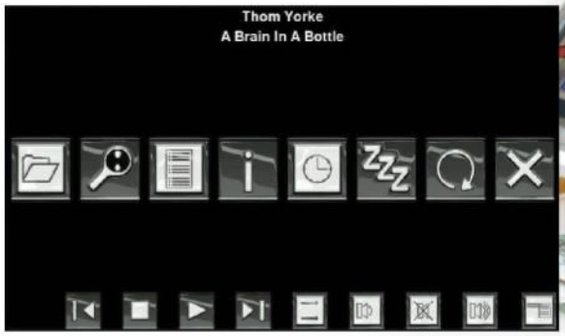
```
./speech.sh "Hello World"
```

Full code listing

```
        try:
            r = requests.get(api_req)
        except requests.exceptions.RequestException as e:
            return errstr

        return WeatherClient.usefuldata(r.json())

class CarLauncher:
    def __init__(self):
        pygame.init()
        pygame.mixer.quit() # Don't need sound
        screen_info = pygame.display.Info()
        self.screen = pygame.display.set_mode((screen_info.current_w,
                                               screen_info.current_h))
        pygame.display.set_caption('Car Launcher')
        self.titlefont = pygame.font.Font(None, 100)
        self.wfont = pygame.font.Font(None, 30)
        self.w_text = None # Weather text

    def clean_background(self):
        background = pygame.Surface(self.screen.get_size())
        self.background = background.convert()
        self.background.fill((0, 0, 0))

        # Render title centered
        text = self.titlefont.render("CarPi Launcher", 1, (255, 255, 255))
        textpos = text.get_rect()
        textpos.centerx = self.background.get_rect().centerx
        self.background.blit(text, textpos)

        self.screen.blit(self.background, (0,0))
        pygame.display.flip()

    def main_menu(self):
        # btns maps Text -> Rectangles we can do collision detection on
        self.btns = {'Music' : None, 'NAV' : None, 'Weather' : None}

        item_num = 1
        for key in self.btns:
            text = self.titlefont.render(key, 1, (255,255,255))
            textpos = text.get_rect()
            max_width = self.background.get_rect().width / len(self.btns)
            center_offset = max_width * 0.5
            # This y pos puts buttons just below title
            textpos.centery = self.background.get_rect().centery / 2
            textpos.centerx = (max_width * item_num) - center_offset
            self.btns[key] = textpos
            self.screen.blit(text, textpos)
            item_num += 1

        pygame.display.flip()

    def select_rect(self, rect, text):
        # Colour a rect the user has clicked in green
        surface = pygame.Surface((rect.w, rect.h))
        surface.fill((0, 255, 0))
        # Now we have to draw the text over it again
        t = self.titlefont.render(text, 1, (255,255,255))
        surface.blit(t, (0,0))
        self.screen.blit(surface, rect)
        pygame.display.flip()
```



Above The Navit software comes with a host of options built into its menu hierarchy



Above The pympdtouchgui front-end for the music player is surprisingly featureful

You will need to write your own launcher for CarPi

Make it mobile

It is definitely best to put this project together in a clean workspace so that you can clearly see what you're working with and ensure everything is correctly wired and soldered, but the point of the project is to make this setup portable so that you can put it in your car and use it on the road. You could install everything into a single, hand-made enclosure or customise a large bought one, or you could secure the various parts inside, for example, your glovebox or car doors. You'll also need to power both the screen and your Pi with a power pack and ensure that the GPS antenna is fastened into a good spot for signal.

16 Configure Navit for speech

The last part is simple. Edit the Navit config file again (/etc/navit/navit.xml) and replace the following line:

```
<speech type="cmdline" data="echo 'Fix the speech tag in navit.xml to let navit say:' '%s'" cps="15"/>
```

...with:

```
<speech type="cmdline" data="/home/pi/.navit/speech.sh %s" cps="10" />
```

Now you can run Navit with **DISPLAY=:0.0 navit** and have fun experimenting.

17 Install the music player

MPD is the music player back-end and pympdtouchgui is the front-end that needs installing manually:

```
sudo apt-get install mpd ncmpcpp
wget http://www.spida.net/projects/software/pympdtouchgui/pympdtouchgui-0.320.tgz
tar zxvf pympdtouchgui-0.320.tgz
cd pympdtouchgui-0.320/
sudo python setup.py install
# Fix hard coded path in software
sudo ln -s /usr/local/share/pympdtouchgui/ /usr/share/pympdtouchgui
```

18 Copy music

Scp (secure copy protocol) was used here to copy music. First get the Pi's IP address by running **ip addr**. Then run **sudo passwd** to set a password for root. From a computer with music on, run:

```
scp -r music_folder root@pi_ip_address:/var/lib/mpd/music/
```

Then on the Pi, change the ownership of the music that you just copied:

```
sudo chown -R mpd:audio /var/lib/mpd/music
```

19 Update mpd music library

Ncmpcpp is a command line client for mpd. Type **ncmpcpp** and press U to update the library. Press 3 to browse the library and check the music is there, and press Q to quit. Pressing 1 will select the help screen if you want to do more.

20 Install awesome window manager

Now you will need to write your own launcher for CarPi, which will run full-screen. To ensure every application is forced to full-screen, use awesome window manager in full-screen mode.

```
sudo apt-get install awesome
sudo rm /etc/alternatives/x-session-manager
sudo ln -s /usr/bin/awesome /etc/alternatives/x-session-manager
```

When changing the default x-session-manager, awesome will be auto-started at boot instead of LXDE. If you reboot the Pi, awesome should then load up automatically.

21 Install the requirements for your launcher

The launcher is going to use a weather API combined with location data from the GPS receiver to give weather updates when requested. The nicest HTTP API for Python is requests, which you can install by doing the following:

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

22 Write the launcher code

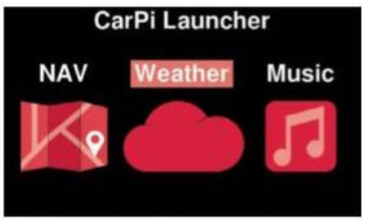
Creating the code itself is pretty self explanatory, but you can use our ready-made version by downloading the CarPi package from **FileSilo.co.uk** and extracting carlauncher/carlauncher.py.

23 Start the launcher automatically

Sudo-edit /etc/xdg/awesome/rc.lua and move awful.layout.suit.max.fullscreen to the top of the layouts list. Add the following to the bottom of the file:

```
awful.util.spawn_with_shell("/home/pi/carlauncher/carlauncher.py")
```

Now reboot again and the launcher should come up automatically.



24 Future improvements

There are a number of improvements that could be made to the base project at this point:

- Make the launcher switch between applications rather than start them again each time
- Make the launcher look better aesthetically with icons
- Use Mopidy instead of MPD so you can use Spotify
- Further Navit configuration to make it more featureful
- An SSD or USB flash drive for storage to make things quicker

Full code listing

```
    def reset(self):
        self.clean_background()
        self.main_menu()
        self.render_weather()

    def execute(self, path):
        os.system(path)
        # os.system blocks so by the time we get here application
        # has finished
        self.reset()

    def render_weather(self):
        if self.w_text == None:
            return

        # Get y starting at the bottom of the nav button
        margin = 10
        y = self.btns['NAV'].bottomleft[1] + margin

        for t in self.w_text.split("\n"):
            line = self.wfont.render(t.rstrip(), 1, (255,255,255))
            line_rect = line.get_rect()
            line_rect.centerx = self.background.get_rect().centerx
            line_rect.y = y
            self.screen.blit(line, line_rect)
            y += margin + line_rect.height

        pygame.display.flip()

    def handle_events(self, events):
        for e in events:
            if e.type == QUIT:
                sys.exit()
            elif e.type == MOUSEBUTTONDOWN:
                pos = pygame.mouse.get_pos()
                # Check if it collides with any of the buttons
                for btn_text, rect in self.btns.iteritems():
                    if rect.collidepoint(pos):
                        self.select_rect(rect, btn_text)
                        if btn_text == "NAV":
                            self.execute("/usr/bin/navit")
                        elif btn_text == "Music":
                            self.execute("/usr/local/bin/pympdtouchgui")
                        elif btn_text == "Weather":
                            self.w_text = WeatherClient.update()
                            # Reset will render weather if string is populated
                            self.reset()

    def loop(self):
        clock = pygame.time.Clock()
        self.reset()

        while 1:
            self.handle_events(pygame.event.get())
            # 5 fps is plenty
            clock.tick(5)

if __name__ == "__main__":
    cl = CarLauncher()
    cl.loop()
```

How I made: Terrarium controller

Keep plants or reptiles at optimal temperature with this automated terrarium controller project

Tom Bennet
by day is an SEO consultant based in London, but in his spare time he loves pushing the boundaries of the Raspberry Pi.

Like it?

Tom's project is fairly unique, but it's very much possible to build your own Raspberry Pi heating monitor. While the finished product doesn't include the complexity of the terrarium controller, it certainly gets the job done. Head across to http://bit.ly/1g3AhR6 for the full tutorial.

Further reading

Tom has gone to the trouble of putting a detailed tutorial of the project on his blog over at www.bennet.org for anyone interested in building their own controller. You can also find all the necessary code available at his GitHub page: https://github.com/tombennet

When did you decide you wanted to build a terrarium?
I'd built my terrarium to grow tropical plants, but soon realised that it couldn't maintain a stable temperature. The bulbs provided enough heat, and I had a cooling system for when it got too warm – the problem was, the cooling needed to run almost continuously on hot days but hardly at all on cold days.

All the terrarium controllers on the market were either mechanical timers or overkill (huge units designed for greenhouses). I needed a simple controller to monitor the conditions, switch mains-powered devices on or off as required, and plot my data onto the Internet Of Things service ThingSpeak.

Could you give us an insight into the development process?
It took me several weekends to order all the components, write the Python scripts, and assemble a working prototype. All the required software is available via the Python Package Index or GitHub. Adafruit maintains a library for reading DHT sensors, the GPIO Zero library can be used to control Energenie mains sockets, and the Requests HTTP library can be used to push your data to the cloud.

I initially wired everything together using a breadboard, duct-taped it to the terrarium, and only soldered it together after a successful trial run. It was then that I realised the same design might be useful for a snake vivarium, so I decided to write the tutorial and stick my scripts up on GitHub.

Did you face any problems?
For a couple of weeks the project became a bit of an obsession, and I think my wife worried I was going crazy. I'd ordered lots of the terrarium components from an online hydroponics store, and I think my neighbours suspected I was rigging up a system for growing drugs!

Other than that, it was plain sailing. The main technical issue was that the transmitter for the Energenie remote sockets is designed to sit directly on the Raspberry Pi's GPIO, occupying almost the entire header and leaving no power pins for other components, like the temperature sensor. To avoid this problem, you can use an extra-short ribbon cable of jumper wires to connect the GPIO pins which are actually needed directly to their respective sockets on the transmitter. This way, you've still got room for your sensor and for any other components you want to add in the future.

What role does the Raspberry Pi play in this project??
The Raspberry P is essentially the brains of the whole thing. In terms of the raw computational power that's required, the Raspberry Pi 3 is hugely overpowered for this project. An Arduino or Raspberry Pi Zero would probably have served just as well. I chose the Pi 3 primarily because I wanted the flexibility to add new components in the future. You could build a small alphanumeric screen to display a read-out of the current conditions; install a soil sensor to monitor your plants' moisture levels; design a web interface or app to control your mains sockets remotely; or wire a spare GPIO pin to an ultra-bright LED mounted in the lid for a night-time moonlight effect.

How was i implement ing things like temperature and humidity control?
The AM2302 temperature and humidity sensor – which is a wired version of the DHT22 – has been great so far. It costs less than £10, provides accurate readings, and uses minimal power. I've heard that prolonged exposure to the high moisture levels common to many terrariums and amphibian enclosures may eventually reduce the accuracy of the humidity reading, but it's been fine.

As for controlling temperature and humidity, this can be done using any mains-controlled components. Heating pads designed for snake enclosures and cheap computer fans can be positioned and then activated/deactivated when the temperature or humidity reaches your chosen thresholds. It isn't actually as difficult as you might think.

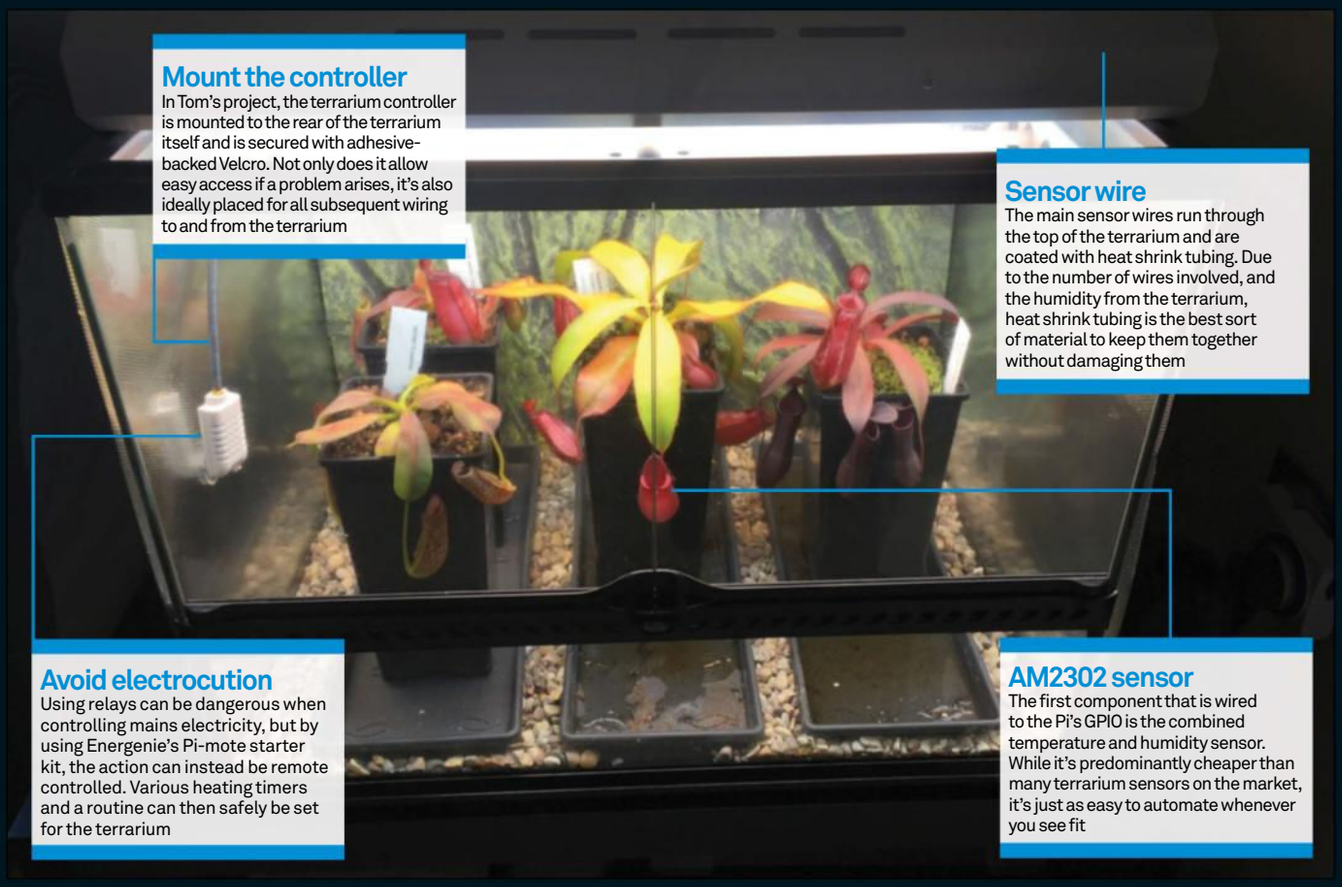
> I chose the Pi 3 because I wanted the flexibility to add new components

Do you have any advice for anyone interested in recreating your project or working on something similar?
I'd say it's a great project for anyone who owns a terrarium or reptile enclosure. It should also be a good first serious project for children, especially given the number of ways it can be extended. I certainly learnt a lot while developing it.

The basic design could be adapted for any situation where you want to activate or deactivate mains-powered devices depending on ambient conditions. There are also many things you could do with the data being pushed to the cloud –ThingSpeak offers a range of MATLAB visualisations and if-this-then-that style actions. There really are countless possibilities for what can be done here.

Have you got any other Raspberry Pi projects that you are currently working on? What do you think of it?
I love working with the Raspberry Pi. It's a great device, and it has an incredibly active and helpful community. My next project is a remote-controlled cat feeder. Most of the cat feeders on the market are flimsy – no match for my two cats, who are fat and strong – and based on timers.

I want to build a device that dispenses a portion of food based on a signal from my phone. The goal is for cat owners to be able to feed their pets remotely when they're working late or away travelling for a day.



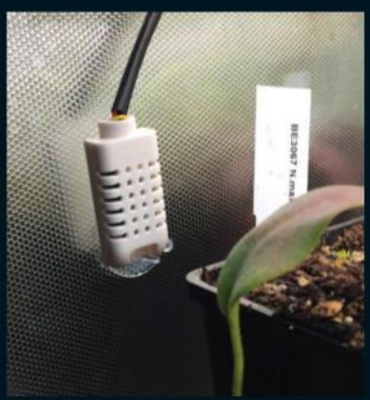
Mount the controller
In Tom's project, the terrarium controller is mounted to the rear of the terrarium itself and is secured with adhesive-backed Velcro. Not only does it allow easy access if a problem arises, it's also ideally placed for all subsequent wiring to and from the terrarium

Sensor wire
The main sensor wires run through the top of the terrarium and are coated with heat shrink tubing. Due to the number of wires involved, and the humidity from the terrarium, heat shrink tubing is the best sort of material to keep them together without damaging them

Avoid electrocution
Using relays can be dangerous when controlling mains electricity, but by using Energenie's Pi-mote starter kit, the action can instead be remote controlled. Various heating timers and a routine can then safely be set for the terrarium

AM2302 sensor
The first component that is wired to the Pi's GPIO is the combined temperature and humidity sensor. While it's predominantly cheaper than many terrarium sensors on the market, it's just as easy to automate whenever you see fit

Components list
- AM2302 temperature and humidity sensor
- Energenie Pi-mote starter kit
- Male-female jumper wire
- Solid core prototyping wire
- Heat shrink tubing
- Small suction cup
- Adhesive backed Velcro

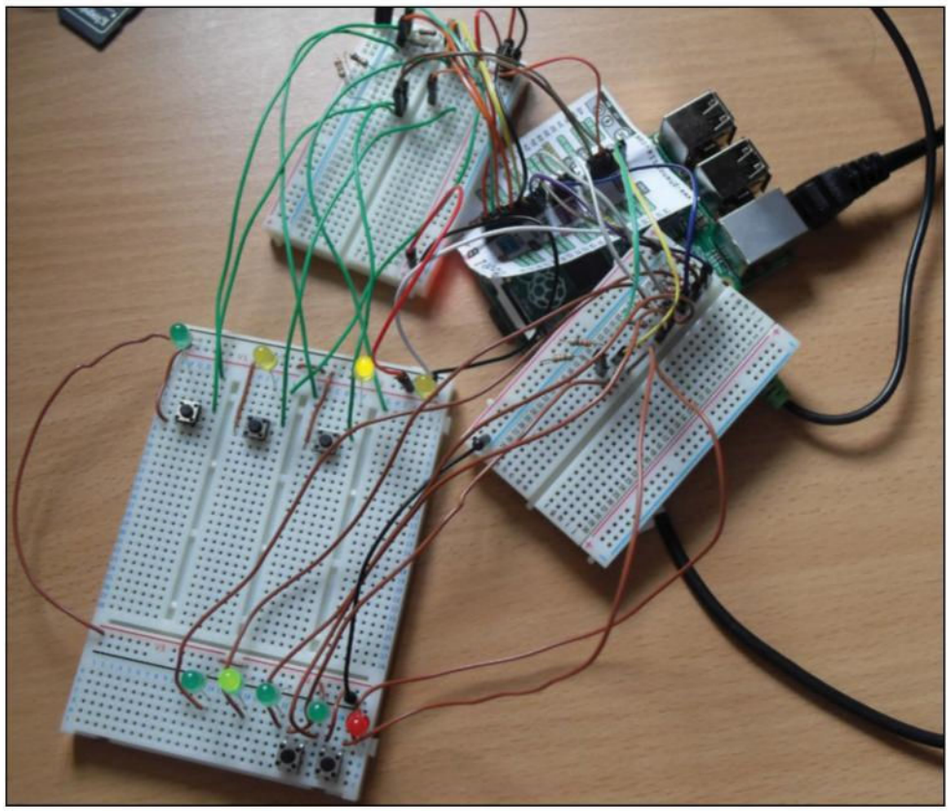


Left A small suction cup helps fasten the sensor to the side of the terrarium. While industrial tape can be used, the humidity may cause it to unwrap quickly

Right The controller is fixed to the back of the terrarium for both easy access and its convenient location for sensor wire placement

Make a Raspberry Pi sampler

Build your own looping drum machine with only 200 lines of code!



Left Extra breadboards are used here to keep the main breadboard as free from wires as possible

What you'll need

- Latest Raspbian image raspberrypi.org/downloads
- At least one breadboard
- Push buttons
- LEDs
- Female-to-male GPIO jumper cables
- Male-to-male GPIO jumper cables

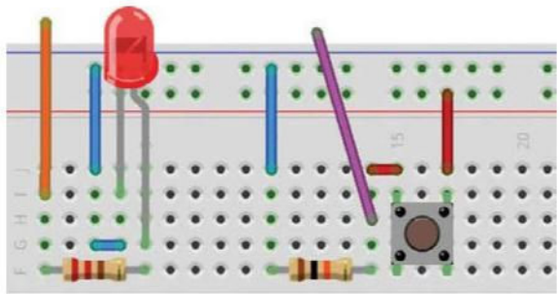
In this tutorial we combine electronics, music and programming knowledge to create a simple sampler with looping capabilities. The implementation used in this article has three drum sounds as the samples, but it is trivial to add more until you run out of GPIO pins.

Before we start, I'll cover some basic musical terms. Music is split into bars. There are a certain number of beats in a bar. This sampler uses the 4/4 time signature, which means there are 4 beats in each bar. Tempo is the speed at which music is played, and it is measured in beats per minute (bpm). A metronome is an audible tone that is heard at the start of every beat.

Quantization is the process of aligning notes to beats, or exact fractions of a beat, and a quantization value is usually given in the form 1/8. This means that there are eight possible places in a bar where a note can be played. When the sampler is recording and a sample button is pressed, we store the sample at the current position in the bar with the accuracy of the quantize value. There's a lot to cover, so let's get started.

01 Connect LEDs

The circuit diagram is an LED that can be turned on and off with a GPIO output. The orange wire is the connection from the GPIO output. This then goes through a 220Ω resistor to limit the current draw to a safe level. This current flows through the positive leg of the LED and then back to ground. We need nine LEDs for this project.



02 Wire up buttons

The second circuit we need is a push button. The purple wire goes to a GPIO input. There is a 10KΩ pull down resistor to ground, which represents a logical 0. When the push button is pressed, the 3.3V supply representing a logical 1 is connected to the purple wire. The electricity takes this path because it has less resistance than the path to ground. We need two buttons for record and undo, and then as many buttons as you like for samples (three drum samples are provided).

03 Download samples

Create a few folder for the project called pisampler. Then download and unzip the sounds:

```
mkdir pisampler
cd pisampler
wget http://liamfraser.co.uk/lud/pisampler/sounds.zip
unzip sounds.zip
```

There will now be a folder called sounds with some samples in. The file format for samples is .wav audio, Microsoft PCM, 16 bit, stereo 44100 Hz. Mono will work too. Any samples can be converted to this format with Audacity by exporting them as a .wav file.

04 Import required libraries

Create a file called pisampler.py in your favourite editor. The first thing we need to do is import the required libraries and set some configuration values. A key option is debounce: the time to wait before a button can be pressed again to stop accidental presses from contact bounce.

05 Create a sample class

We're going to use a class to represent each sample. It's going to need a few things: the pin that the sample button is connected to, the name of the sound file, and a reference to the instance of the sampler class. We haven't created the sampler class yet, but the sample will need to be able to tell if the sampler is recording or not, and have access to the data structure that recordings are stored in to add itself to it if necessary.

The other thing that we need to do is set the GPIO pin to an input, and add an event listener for when the button is pressed. We set callback (the function to be executed when the button is pressed) to a function called self.play_btn, which will play a sound

Full code listing (Cont. on next page)

Step 04

```
import RPi.GPIO as GPIO
import time
import pygame
import os

beat_leds = [2, 3, 4, 17]
bar_leds = [27, 22, 10, 9]
record_led = 11
record = 19
undo = 26
debounce = 200 # ms
```

Step 05

```
class Sample(object):
    def __init__(self, pin, sound, sampler):
        self.sampler = sampler
        self.name = sound
        self.sound = pygame.mixer.Sound(os.path.join('sounds', sound))
        self.pin = pin
        GPIO.setup(pin, GPIO.IN)
        GPIO.add_event_detect(self.pin, GPIO.RISING, callback=self.play_btn,
                              bouncetime=debounce)

    def play_btn(self, channel):
        self.sound.play()
        s = self.sampler
        if s.recording:
            s.recording_data[s.bar_n][s.quantize_n].append({'loop' : s.loop_count,
                                                            'sample' : self})
```

Step 06

```
class PiSampler(object):
    def __init__(self, tempo=80, quantize=64):
        pygame.mixer.pre_init(44100, -16, 1, 512)
        pygame.init()

        self.quantize = quantize
        self.tempo = tempo
        self.recording = False
        self.record_next = False

        self.metronome = False
        self.met_low = pygame.mixer.Sound(os.path.join('sounds', 'met_low.wav'))
        self.met_high = pygame.mixer.Sound(os.path.join('sounds', 'met_high.wav'))
        self.met_low.set_volume(0.4)
        self.met_high.set_volume(0.4)

        self.samples = []

        self.recording_data = []
        for i in range(0, 4):
            bar_arr = []
            for i in range(0, quantize):
                bar_arr.append([])

            self.recording_data.append(bar_arr)

        GPIO.setmode(GPIO.BCM)
        for pin in beat_leds + bar_leds + [record_led]:
            GPIO.setup(pin, GPIO.OUT)

        GPIO.setup(record, GPIO.IN)
        GPIO.add_event_detect(record, GPIO.RISING,
                              callback=self.record_next_loop,
                              bouncetime=debounce)
```

Full code listing (Cont.)

Step 06

```
        GPIO.setup(undo, GPIO.IN)
        GPIO.add_event_detect(undo, GPIO.RISING,
                              callback=self.undo_previous_loop,
                              bouncetime=debounce)
```

Step 07

```
    @property
    def tempo(self):
        return self._tempo

    @tempo.setter
    def tempo(self, tempo):
        self._tempo = tempo
        self.seconds_per_beat = 60.0 / tempo

        self.quantize_per_beat = self.quantize / 4
        self.quantize_seconds = self.seconds_per_beat / self.quantize_
per_beat
```

Step 08

```
    def add(self, sample):
        self.samples.append(sample)

    @property
    def recording(self):
        return self._recording

    @recording.setter
    def recording(self, value):
        self._recording = value
        GPIO.output(record_led, value)

    def record_next_loop(self, channel):
        self.record_next = True
```

Step 13

```
    def play_recording(self):
        for sample_dict in self.recording_data[self.bar_n][self.
quantize_n]:
            if sample_dict['loop'] != self.loop_count:
                sample_dict['sample'].sound.play()

    def undo_previous_loop(self, channel):
        if len(self.last_recorded_loop) == 0:
            print "No previous loop to undo"
            return

        print "Undoing previous loop"

        loop = self.last_recorded_loop.pop()

        for bar in self.recording_data:
            for quantize in bar:
                removes = []
                for sample in quantize:
                    if sample['loop'] == loop:
                        removes.append(sample)

                for sample in removes:
                    quantize.remove(sample)
```

Step 11

```
    def do_leds(self, leds, n):
        count = 0
        for led in leds:
            if count == n:
                GPIO.output(led, True)
            else:
```

and add it to the recording data if we are recording. It will become abundantly clear how this works once we've written the sampler class. Note that the GPIO event handler passes the pin that the event handler was triggered on, hence the channel variable that is present but never used.

06 The sampler init method

Here's the start of the sampler class. The last value in the Pygame mixer init is the buffer size. You might need to increase this to 1024 or higher if you have audio dropouts. We create some variables to store recording state. Metronome sounds are then added and their volume lowered. We also create a list to hold our samples in.

We create nested arrays to represent recorded sample presses. There is an array for each bar. Each bar has an array for each possible quantize value. The default value of 64 gives us 64 possible places to store a sample hit per bar.

Finally, we set up the LED pins, and the pins for the record and undo buttons.

07 The tempo property

The tempo variable is actually a property with a custom setter. This means when a value is assigned, it does a custom action. In our case, we need to calculate how often we need to check for recorded notes to play in the main loop that we'll write later.

08 Helper functions

There are a few helper functions in the class. One of them simply adds a sample to the list of samples. Another sets a variable to trigger recording at the start of the next loop. There is also a function which turns the red LED on when the recording variable is set to true. Now we'll jump forward and take care of the main loop towards the end of the full code listing.

09 Start the main loop

The main loop doesn't actually have to do any work at all to play sounds, as that's done by the GPIO event handlers. The main loop is used to play the metronome, update the state about which bar/beat/quantize we are currently on, update the LEDs and deal with recording if necessary.

Before the loop, we create variables to track the state. The last recorded loop is a list that we will use as a stack. A stack is a last in/first out data structure, allowing us to undo recordings multiple times by removing each sample that was recorded when the loop count was the value on the top of the stack.

If we're at the start of a new beat then we use a function called do_leds that we haven't created yet. As the LEDs work in the same way (a block of four LEDs where only one is turned on), we can use the same function twice and just pass a different set of pins, and the index of the LED we want to turn on. We then call the do_metronome function which will play the appropriate metronome sound.

We then do some recording logic which starts recording if we should be recording, and stops recording if we have just been recording, adding the

loop number to the last_recorded_loop stack. We increment the loop count after doing this.

10 Main loop continued

This code is at the indentation level after the "while True:" statement. After dealing with the recording logic, we need to play any notes that have been previously recorded. We'll work out how to do that later on. After that, we can sleep until the next quantize change is due. Once this happens, we have to do logic that deals with the quantize and any related variables such as the beat or bar if necessary, either incrementing them or resetting them if necessary.

11 Lighting LEDs

The LED code is simple. It simply goes through each pin in the list you provide it with and lights up the appropriate LED, ensuring that all of the others are turned off.

12 The metronome

The metronome simply plays a high tone on the first beat or a lower tone on the remaining beats if the metronome variable is set to true.

13 The recording code

Looking back at the sample class we created at the start, you can see that if recording is enabled, and a note is pressed, then we add a dictionary to the list of samples for the current bar at the current quantize point. The dictionary contains a reference to the sample so that it can be played, and also the loop that it was added on so that it can be removed if necessary. The code for playing and undoing recordings can be seen below.

Note that we directly play the sound rather than using the btn_play function so that we don't trigger the recording logic when playing recorded sounds.

The pop function in undo_previous_loop removes the last thing that was added to the stack, which will be the loop count. We then go through every possible recording data point and remove anything recorded on the loop we want to remove.

14 Finishing it off

To finish it off, we need to add a main function where we load some samples in and then start the main loop. Remember that you need to run the code with sudo python2 pisampler.py because we need sudo to access the GPIO. Happy jamming!

15 Possible improvements

There are a number of improvements that could be made to the sampler. Here are a few to get you started:

- A button to turn the metronome on and off
- The ability to time stretch samples (such as chords) to fit with the tempo
- The ability to pitch shift samples on the fly
- Using a shift register to use less pins when lighting the LEDs, allowing more inputs
- The ability to save recorded beats so that they can be loaded and played back

Full code listing (Cont.)

Step 11

```
                GPIO.output(led, False)

            count += 1
```

Step 12

```
    def do_metronome(self):
        if not self.metronome:
            return

        if self.beat_n == 0:
            self.met_high.play()
        else:
            self.met_low.play()
```

Step 09

```
    def run(self):
        self.loop_count = 0
        self.last_recorded_loop = []
        self.bar_n = 0
        self.beat_n = 0
        self.quantize_beat_n = 0
        self.quantize_n = 0

        while True:
            if self.quantize_beat_n == 0:
                self.do_leds(beat_leds, self.beat_n)
                self.do_leds(bar_leds, self.bar_n)
                self.do_metronome()

                if self.quantize_n == 0 and self.bar_n == 0:
                    if self.record_next:
                        self.recording = True
                        self.record_next = False
                    elif self.recording:
                        self.recording = False
                        self.last_recorded_loop.append(self.loop_count)

                    self.loop_count += 1
```

Step 10

```
            self.play_recording()
            time.sleep(self.quantize_seconds)

            if self.quantize_beat_n == self.quantize_per_beat - 1:
                self.quantize_beat_n = 0
                self.beat_n += 1
            else:
                self.quantize_beat_n += 1

            if self.quantize_n == self.quantize - 1:
                self.quantize_n = 0
            else:
                self.quantize_n += 1

            if self.beat_n == 4:
                self.beat_n = 0
                self.bar_n += 1
                if self.bar_n == 4:
                    self.bar_n = 0
```

Step 14

```
if __name__ == "__main__":
    sampler = PiSampler(tempo=140)
    sampler.add(Sample(05, 'kick01.wav', sampler))
    sampler.add(Sample(06, 'snare01.wav', sampler))
    sampler.add(Sample(13, 'clhat01.wav', sampler))
    sampler.metronome = True
    sampler.run()
```

Transform your Pi into a micro oscilloscope

Prepare to turn your Raspberry Pi into a fully functional micro oscilloscope, logic analyser and waveform generator with the BitScope Micro

> Requiring no external power source, the BitScope Micro is also water resistant

The Raspberry Pi has been used in a plethora of applications in hardware, software and some quite unbelievable combinations of the two. From record-breaking space flights to automated bartending devices and much more, there are thousands of Raspberry Pi projects that, over the last two and a half years, have shown what a capable little Linux box this is.

The BitScope Micro is certainly no exception and when you couple it with your Raspberry Pi you have a very powerful, pocket-sized oscilloscope that also features a whole host of other functionalities, such as a waveform and clock generator as well as a spectrum and logic analyser. Best of all though, the whole setup (including the Raspberry Pi itself) comes in at well under £150.

Requiring no external power source, the BitScope Micro is also water resistant and so is perfect for either home or lab use. It is fully configurable and user programmable in Python, C++ and more, and can even continuously stream data to disk.

01 Grab your BitScope

If you have not already done so, you need to go and order your shiny new BitScope Micro (directly from BitScope or from one of their worldwide retailers). If you are serious about electronics then you need a good oscilloscope, so it is truly worth every penny! Once it arrives, you should be greeted with the neatly packaged box pictured above.

02 Open up the box

Once you have received your BitScope Micro and opened up the box for the first time you should find all of the pictured items inside (if you get any extras, then it is obviously your lucky day). The main contents are the BitScope Micro itself (with mini USB cable preattached) and a set of ten test clip grabbers. There is also a variety of documentation containing a large amount of product info and guidance.

03 Update your Raspberry Pi

As with almost every project you undertake on the Raspberry Pi, it pays dividends to make sure your operating system is updated to the latest stable version, as this can save a lot of hassle further down the line. To do this, open an LXTerminal session and then type:

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

Then wait patiently for the upgrade process to complete.

04 Locate the BitScope Software

Now your Raspberry Pi is all up to date you need to acquire the BitScope DSO (digital storage oscilloscope) software. This is not yet available as a Raspbian package, but it is very easy to install anyway using the downloadable DEB file. Visit **www.bitscope.com/pi** and click on the Download link at the top.

Above The BitScope Micro comes complete with test clip grabbers and a whole lot of documentation

05 Download the software

The previous step should have brought you to the BitScope Downloads page. From here you need to download the top item in the list, BitScope DSO 2.7 (beta), and save it to the /home/pi directory on your Raspberry Pi so you know where to find it later. On some browsers the file will automatically download to the /home/pi/Downloads directory.

06 Install the software

Now we have downloaded the package, the easiest way to install the software is to open an LXTerminal session and then run the following code...

```
sudo dpkg -i bitscope-dso_2.7.EA17H_armhf.deb
```

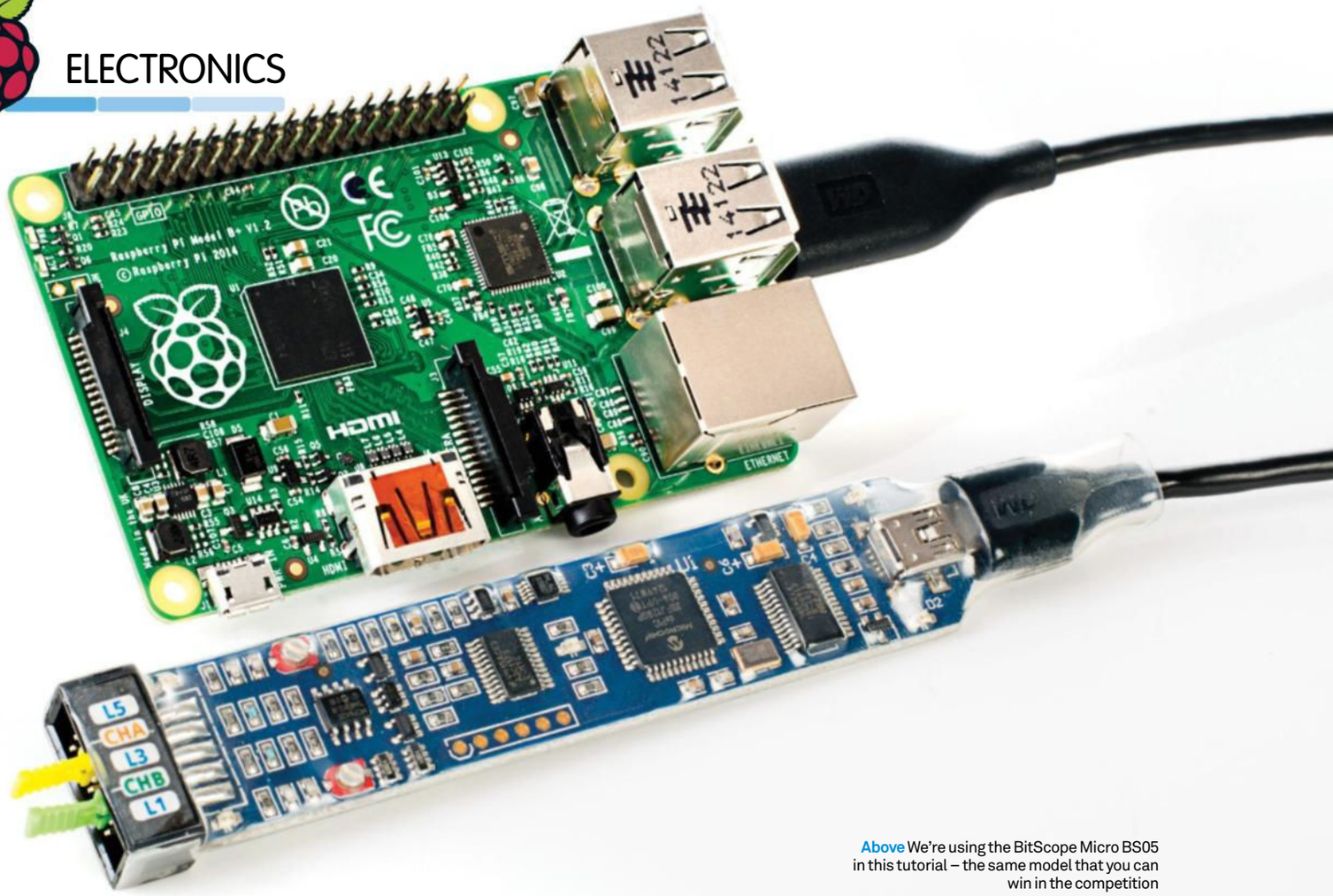
...or the equivalent version for newer software. You should see lines of output as the installation progresses. The BitScope DSO software then appears in the main menu under Other.

07 Overclock your Raspberry Pi (optional)

The BitScope software will run fine on a Raspberry Pi with default settings, however it can be a bit slow to respond. Open an LXTerminal session and type sudo raspi-config. In the menu, select option 7 Overclock. Click OK on the following screen and on the next one select Turbo. Click OK and then you should see some code run. Once this completes press OK and then you are brought back to the main raspi-config window. Select Finish at the bottom right, and then select Yes to reboot your Raspberry Pi.

Multiple platform support

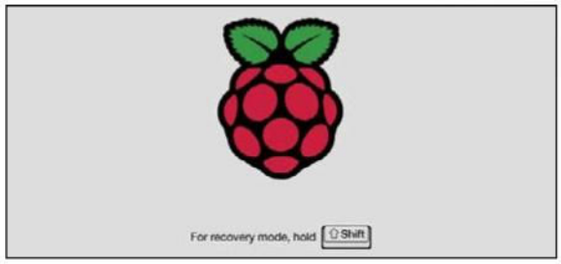
One of the best things about the BitScope Micro (as well as its big brother, the BitScope BS10U) is that it's capable of running on a Pi and on any Linux, Windows or Mac OS X device with a USB port. The graphical UI is identical on each of these devices so it's easy to switch between them. The BitScope Micro should also work with smartphones capable of USB on-the-go connections, but there is no software available to take advantage of this yet.



Above We're using the BitScope Micro BS05 in this tutorial – the same model that you can win in the competition

Hardware upgrades

One of the best things about the BitScope Micro is that it runs on exactly the same software as the more capable hardware in the range. This means if at some point in the future you feel the BitScope Micro is not enough for your needs, you can quickly and easily upgrade to better hardware with no hassle, and no need to learn any new software!



08 Overclocking – part two

Overclocking can sometimes cause instability on your Raspberry Pi or an inability to boot at all. If this happens, you can press and hold the Shift key on your keyboard once you reach the above splash screen to boot into recovery mode. You can then re-do step 7 at a lower overclock setting and repeat until you find the highest stable setting.

09 Plug in the BitScope

Now that the software has been successfully installed on your Raspberry Pi, we can get started with the BitScope. If you are using a Model A or B Raspberry Pi without a USB hub then I would recommend turning the Raspberry Pi off before attaching the BitScope or it may crash. The B+ should be fine with plug and play.

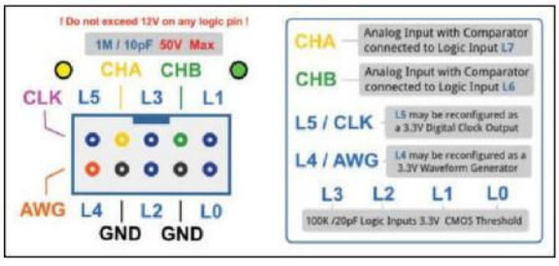
10 Load the BitScope DSO

Now you can navigate to the BitScope DSO software in the menu. This should load a splash screen with two options – POWER and SETUP. Click POWER and then OK on the pop-up information box. After a minute or less, the BitScope DSO main screen will load and you should see some lights on the BitScope start to flash.

Overclocking can sometimes cause instability

11 Familiarise yourself with the software

The image on page 33 shows the screen layout of the BitScope DSO software. It is fairly intuitive, and is similar to other physical or virtual oscilloscopes. The largest part is the main display window. To the top-left is the trigger window (this changes to wave control if selected). Under the main window you have the analog input channels and various trim adjustments.

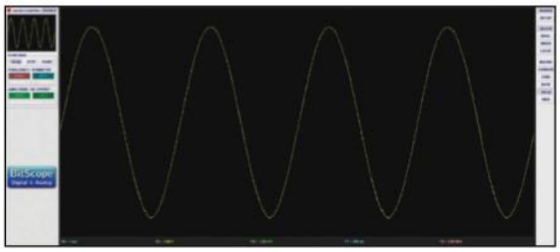


12 Familiarise yourself with pinout

The image above shows the BitScope Micro pinout diagram. There are a total of ten pins with two of them being ground pins (GND). The remaining eight pins can be configured as logic pins and four of them also have different functions – L4 is also a waveform generator (AWG), L5 is a clock generator, (CLK) and L7 and L6 relate to CHA and CHB respectively.

13 Perform a sample experiment

The easiest way to test whether or not your BitScope is working correctly is to connect one of the test clip grabbers to the analog input CHA on the BitScope. Connect the other end to physical pin two on your Raspberry Pi and adjust the scale of the y axis to 2V/div. You should then see an output in the main window of around five volts.



14 Use different waveforms

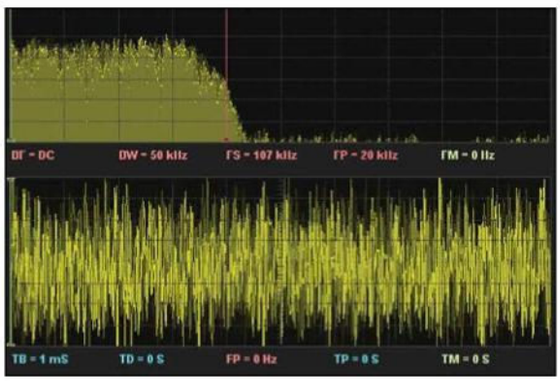
The BitScope can generate its own waveforms. Connect a female-to-female jumper cable between CHA and L4 (AWG). On the right-hand side of the DSO interface, select Wave and then a wave should appear in the main screen. Change the x axis to 100us/Div and the y axis to 500mV/Div. Right-click on the yellow OFFSET button and select MEDIAN. The wave should now fill the main window as in the above screenshot. You can adjust various parameters of the waveform in the wave control panel top-left and can also change to step (square) or ramp (saw-tooth) waves instead of tone (sinusoidal).

15 Experimenting with your body

Another interesting (but fairly simple) thing to try is measuring electrical signals from your body. Set the vertical axis to 1V/Div and horizontal to 20ms/Div. Then plug in one of the probes to CHA, pull back the grabber end and touch it with your finger. You should then see a sine wave on the screen. Bizarrely, this wave is actually radio waves emitted by your mains electrical wiring which are then being picked up by your body (which is acting as an antenna). This is why the wavelength of the signal you see is approximately 50 to 60 Hz.

16 Programming your BitScope

The BitScope DSO and other available software (BitScope Logic and BitScope Chart) are very powerful applications for a lot of experimentation. However, in some more complex monitoring environments they may not offer enough flexibility. The good news is that you can program the BitScope in C/C++, Python or Pascal using their programming API.



17 Further experiments to try

This tutorial has shown only a small fraction of what the BitScope Micro is capable of. As seen in the above image it can also be used as a spectrum analyser along with a whole host of other functionality. Perhaps for your next experiment you could measure the resistance of your body by comparing the voltage drop across your body with that of a known resistor. Or you could try probing your I2C or SPI connections to see how they work. If you ever run out of ideas, then why not visit the BitScope website and start experimenting further!

Above The experiment uses your body as an antenna to pick up radio emissions from your house

Mains electrical frequency

We looked at a signal from your body caused by radio emissions from the mains power supply in your home. In Europe the mains frequency is 50 Hz and typically with a voltage of 230 or thereabouts. In USA and some parts of Asia, the mains frequency is 60 Hz with a typical voltage of 110. Most modern electrical equipment is therefore capable of operating at either voltage or frequency.

Popper buttons Beneath the fingers are small metal discs that can be activated just by a touch. Each button has been given a new function

Camera module This has been embedded into the fabric on the back of the glove, so photos can be taken by raising a hand and touching a button

Power pack As well as the Model A+, the USB charger used to power the Pi can be mounted on the glove by sliding it inside on the palm side

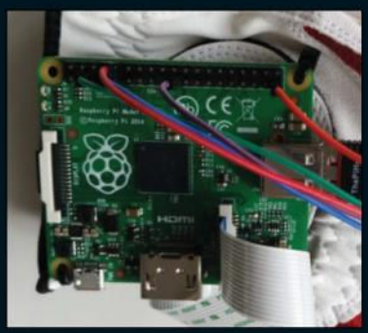
Raspberry Pi Using the Model A+, the Ras Pi could be mounted on the back of the glove. The old CAT5 cables were replaced with fewer wires

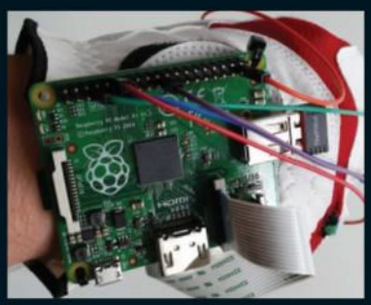
Components list

- Raspberry Pi Model A+
- Raspberry Pi camera module
- 5 Short cables
- 5 Popper buttons
- USB mobile phone battery
- USB Wi-Fi dongle
- Energenie add-on board
- Golf glove

Left For the next version, Dan may add a palm button for a hierarchical menu structure to navigate the functions on each finger-button

Below Now he's done the social media (see issue 148) and the home help modules, Dan will make a start on the fitness module





How I made: Pi Glove 2

Check out Dan Aldred's new home help module

What physical modifications have you made since we last spoke?

The glove is more portable – previously, the Raspberry Pi was in the wearer's pocket and you had the long CAT5 cables attached the glove. This has all been stripped back and several steps enabled this. Firstly, was the use of fabric clothes poppers to replace the tactile switches. These are metal and when one makes contact with a ground popper on the thumb, it creates the circuit. It has meant that the same functionality is achieved with five wires as opposed to the previous ten wires. Secondly, I have moved from a B+ model to the A+ model, which has meant that the Raspberry Pi is now small enough to be mounted on to the glove itself. Now the wires only need to run from the fingertip to the wrist. The camera module is also embedded within the glove. The lens is exposed through the glove but the rest of the camera is now housed within the fabric of the glove. You have to be a little bit more careful when you take the glove off, but the overall pay-off is that the glove is now lighter and more compact. The power comes from a small USB mobile phone charger which gives about six hours running time, depending on how much you use it for.

What new functions does the rebuilt glove have?

It was always the plan to develop the Pi Glove with 'modules' for various uses, starting with home assistance. Imagine if you did or maybe do struggle with a disability and you wake up in the night – the Pi Glove now enables you to check the time, which is read out, and a light can be turned on with a simple finger movement. If required, an emergency text can be sent to a carer, family member or the provider of other medical assistance. The fourth button enables the Pi camera, which takes a picture of a sign, for example. OCR is then used to recognise and store the text, which is then read back to you.

I decided to add the Pi camera around the back of the hand area – this linked in well, enabling a more mobile use of the camera; it can now be positioned in a direction that the user wants, is more accessible and you could even take a selfie! The main reason for this change was to enable 'on the fly' optical character recognition. I installed a Python OCR library and, combining this with the image taken from the Pi camera, the software can identify the text within the picture. This text is then fed back to the console and easy-Speak reads out the text. I tried various file formats – JPG seemed to work well. Also, changing the picture to black and white to pick up detail and differentiate between the text, had improved results. There were issues with it not identifying text in low light, and also if the text was the wrong way round or upside down. Finally, changing the saturation and increasing the sharpness produced usable results.

The emergency text on the second button makes use of Twilio, the web-based communication API, which enables the user to send a pre-written message requesting assistance. This could be adopted by others, such as the police or fire brigade, for use in dangerous situations. The current time is also added to the text.

To turn the lights on I used an add-on board by Energenie. Once installed you can use it to control up to four simple Energenie radio-controlled sockets independently, using a small program. The add-on board connects directly to the GPIO, which can be controlled as either input or output lines under your software control. A Python library is also available to program the sockets. I simply mapped the 'on' state to the click of the button and it turned the light on – from your fingertips!

> The camera takes a picture of a sign. OCR recognises and stores the text, which is read back to you

Are you currently developing any new modules for the future?

The current module I am working on is fitness-related and will allow the wearer to take their heart rate, change their music and upload the details of a run to their chosen social media site. This will be on the fly with no need to stop running or whatever sporting activity you are doing. Just touch the buttons and your workout music changes, or your heart rate is read and then converted to a string and read back to you through your headphones. I find that current apps on phones and watches disrupt your workout – I don't want to have to stop running to check my pulse or change the music.

I was looking at the idea of linking the glove functionally to a smartphone but this, I feel, would be moving away from the original aim, which was to remove the cumbersomeness of the phone – having to unlock it, load an app, wait for the camera to prepare itself. The glove enables you to click and go.

What would you like to do for the third iteration of Project New York?

I was introduced to the Micro Python Pyboard (**micropython.org/live**), which has a range of features built in and is smaller, making it more suitable as a wearable component. The Micro Python board is a small electronic circuit board that runs Micro Python on the bare metal, and gives you a low-level Python operating system that can be used to take control of all kinds of different electronic projects.

The battery size is also an area that could be reduced – I am looking into this. The smaller that all these components are, the more natural the glove will feel when it's being worn.

Dan Aldred is a curriculum leader for Computing and IT at a comprehensive school and the lead co-ordinator for Computing At School. As a Raspberry Pi Certified Educator, Dan promotes the use of the Pi in education and as a tool for learning.

Like it?

To learn more about the redesigned Raspberry Pi Glove and the current home help module, check out Dan's YouTube video (**bit.ly/1HVQTYA**) and the project write-up (**bit.ly/19xgQyC**).

Further reading

If you're interested in setting up optical character recognition for your own sign-reading Python projects, check out Dan's guide over at TeCoEd (Teaching Computing Education): **tecoed.co.uk/python-ocr.html.**

Assemble a Minecraft power move glove

Create a piece of wearable tech with power moves assigned to each button to enhance your Minecraft game

Many of you will be avid fans of the game *Minecraft*. In schools it is fast becoming a motivational teaching and learning tool, useful in areas such as programming, creating logic gates and setting up a network.

This project is framed around creating a simple networked *Minecraft* game where one player chases the other and tries to hit the block they are standing on. The real hack is programming a 'power glove' that enables you to assign power moves to each button. These powers can then be deployed and used to slow down the other player and get yourself out of sticky situations – check out the video at bit.ly/1CQSmHS! The real beauty of this hack is that you can then create and customise your own power moves. The possibilities are endless, limited only by your imagination. If you're confident with GPIO pins and setting up buttons, jump straight to Step 8.

01 Update the Raspberry Pi

This project is designed for the Raspberry Pi 2 which requires the updated operating system, although it is compatible with the Raspberry Pi B+ too. First ensure that the software is up to date – open the LX Terminal type:

```
sudo apt-get update
sudo apt-get dist-upgrade
sudo apt-get install raspberrypi-ui-mods
```

02 Connect a button to the Raspberry Pi

Take one of the buttons and connect a wire to each contact, then take the other two ends and connect to the Pi. You may find this easier using a female-to-female connector. To set up a test button, use GPIO pin 17 – this is physical pin number 11 on the board. The other wire connects to a ground pin, shown by a minus sign (the pin above GPIO pin 17 is a ground pin).

What you'll need

- Raspberry Pi 2
- 4 x 6mm tactile buttons
- Female-to-female jerky
- Terminal blocks
- A glove
- Router
- 2x CAT5 cables

03 Test that the button works

Use this test code to ensure the button is functioning correctly. Once it is, the same setup method can be used throughout this project. To ensure the buttons are responsive, use the pull-up resistor with the code GPIO.PUD_UP – this will ensure that multiple touches aren't registered on each button. Using Python 2.8, type in the code below, then save and run. If working correctly, it will return the message 'Button works'.

```
import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BCM)

GPIO.cleanup()
GPIO.setup(17, GPIO.IN, GPIO.PUD_UP)

while True:
    if GPIO.input(17) == 0:
        print "Button works"
```

04 Create a power move

Now to create your first power move. The glove and code are structured in such a way that once the basic code template is set up, you can create your own power moves and assign them to each button, keeping both your ideas and your gameplay fresh. The first power move you will program enables you to place a wall of TNT between you and the player trying to catch you. They have a choice to go around the wall or blow it up, but it will slow them down. In a new Python window, type the code below and save the program into the home folder:

```
import time
def Firewall():
    mc.postToChat("Firewall Placed")
    TNT = 46,1
    x, y, z = mc.player.getPos()
    mc.setBlocks(x-6, y, z+2, x+6, y+10, z+3, TNT)
    time.sleep(10)

while True:
    Firewall()
```

05 Open Minecraft

The updated version of the Raspbian OS comes with *Minecraft* pre-installed, it can be found under Menu>Games – so load it up. If you have used the *Minecraft: Pi Edition* before you will be aware that it runs better in a smaller-sized window, so don't make it full screen. You may prefer to adjust and arrange each window side-by-side to enable you to view both the Python code and the *Minecraft* game at the same time.

PiGlovePowerMoves.py

```
import time
from mcpi import minecraft

mc = minecraft.Minecraft.create("192.168.1.211")
#Replace with the other players IP address

import RPi.GPIO as GPIO

#Set up the GPIO Pins
GPIO.setmode(GPIO.BCM)

#Sets the pin to high
GPIO.cleanup()
GPIO.setup(17, GPIO.IN, GPIO.PUD_UP)
#11 on the BOARD GREEN Firewall
GPIO.setup(18, GPIO.IN, GPIO.PUD_UP)
#12 on the BOARD BROWN Lava
GPIO.setup(9, GPIO.IN, GPIO.PUD_UP)
#21 on the BOARD BLUE Mega Jump
GPIO.setup(4, GPIO.IN, GPIO.PUD_UP)
#7 on the BOARD ORANGE Puddle
GPIO.setwarnings(False)    #switch off other ports

#Builds a wall of TNT which if the player hits will explode
def Firewall():
    mc.postToChat("Firewall Placed")
    TNT = 46,1
    x, y, z = mc.player.getPos()
    mc.setBlocks(x-6, y, z+2, x+6, y+10, z+3, TNT)
    time.sleep(1)

#Lays Lava to slow down the other player
def Lay_Lava():
    Lava = 10
    check = 1
    mc.postToChat("Lava Deployed")
    while check == 1:
        time.sleep(0.2  )
        x, y, z = mc.player.getPos()
        mc.setBlock(x-1, y, z, Lava)
        check = 0

#Peforms a Mega Jump to lose players
def Mega_Jump():
    time.sleep(0.1)
    mc.postToChat("Mega-Jump")
    x, y, z = mc.player.getPos()
    mc.player.setPos(x, y+15, z)
    time.sleep(1)

#Creates a Puddle to slow down your player
def Mega_Water_Puddle():
    mc.postToChat("Mega Water Puddle")
    time.sleep(0.2)
    WATER = 9
    x, y, z = mc.player.getPos()
    mc.setBlocks(x-5, y, z-4, x-1, y, z+4, WATER)
    time.sleep(1)
```

GPIO pins

GPIO pins are a physical interface between the Pi and the outside world. At the simplest level, you can think of them as switches that you can turn on or off (input) or that the Pi can turn on or off (output). The GPIO.BCM option means that you are referring to the pins by the "Broadcom SOC channel" number. GPIO.BOARD specifies that you are referring to the pins by the number of the pin and the plug – the numbers printed on the board.

IP address

Most home networks will use IP addresses that start with 192.168.1 and then a number up to 255. An old router will assign these IP addresses automatically. If the router has Wi-Fi then players can also connect to multiplayer using a USB Wi-Fi dongle; you are setting up a LAN (Local Area Network).

06 Engage the firewall

Start a new *Minecraft* game, then go to the Python program and press F5 to run it. You may need to press the Tab key to release the mouse from the *Minecraft* window. The program will place a 12 x 10 wall of TNT behind the player every ten seconds – you can blow them up but the Pi might lag.

07 Assign the power move to the button

Now you have a working button and a power move, you can combine these two together. Basically, when you press the button the power move will deploy. Once this is working you can then set up the other three buttons with the same method. Open a new Python window and type in the code below – save the file in the home folder. Start a *Minecraft* game as shown in Step 6 and then run the new program. Every time you press the button you will place a TNT firewall.

```
import time
from mcpi import minecraft
mc = minecraft.Minecraft.create()
import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BCM)

GPIO.setup(17, GPIO.IN, GPIO.PUD_UP)
#11 on the BOARD

def Firewall():
    mc.postToChat("Firewall Placed")
    TNT = 46,1
    x, y, z = mc.player.getPos()
    mc.setBlocks(x-6, y, z-2, x+6, y+10, z-1, TNT)

while True:
        if GPIO.input(17) == 0:
            Firewall()
```

08 Set up further buttons

Once you have one button working the next step is to set up the other three using the same method from Step 2. Take the button and connect the two wires to either side. At this point you can add all three buttons and test them individually for connectivity. Connect each set of button wires to the GPIO 17 and run the firewall program. If the firewall power move builds on each click of the button, the connections and buttons are working and you're ready to attach the buttons to the glove.

Below If you want to get fancy, try using the popper buttons that Dan added to his Pi Glove 2 project

09 Create the glove

Take your glove and attach the four buttons to the fingers on the glove. There are a number of ways to do this: glue the button on, sew them in, stick them with double-sided tape – the choice is up to you. Wires can be hidden or on display, depending on your preferences and how you want the glove to look.

10 Connect the wires

Now you are ready to connect the wires to the Raspberry Pi to enable all four power moves to be used. Take one end of each wire and connect them to the respective pins, as shown below. The other end of the wire is placed into a ground pin. Note that the RPi.GPIO pin numbering system is used here rather than the physical pin number on the board – for example, GPIO pin 17 is physical pin number 11. The following pins are used:

```
GPIO 17, pin 11 on the board
GPIO 18, pin 12 on the board
GPIO 9, pin 21 on the board
GPIO 4, pin 7 on the board
```

11 Set the network up

To interact with other players in the *Minecraft* world you will need to set up a networked multiplayer game. This is simple and can be achieved using an old router. Connect each Raspberry Pi via an ethernet cable to the router and then back to the ethernet port on each Raspberry Pi. Turn on the router and wait about 30 seconds for the IP addresses to be assigned. Now load up *Minecraft* and one of the players will start a new game.

12 Run the game!

After the game has loaded, the other connected player will see an option to "connect to a multiplayer game", usually called Steve. The player selects this option and the networked game will begin. You will be able to see two 'Steves' and the *Minecraft* chat will inform you that 'Steve has joined the game'. Open Python and the glove code, press F5 to run and you will be able to see the power moves being deployed.

13 Interact in another player's world

You'll notice under the current setup that if the Pi Glove is connected to the game and you use one of your power moves, it will only appear in your *Minecraft* world and not in the other players. This isn't great, as the power move is supposed to stop the other player! To resolve this, find the IP address of the other

Raspberry Pi, then enter this IP address into this line of code: **mc = minecraft.Minecraft.create( )**. For example, **mc=minecraft.Minecraft.create("192.168.2.234")**, filling the empty brackets with the IP address of the other Raspberry Pi within your game. Remember that this is the IP address of the other player's Raspberry Pi and not your IP address.

14 Find your IP addresses

To find the IP address of a Raspberry Pi, load the LX terminal, type **ipconfig** and press Enter – the address will be displayed on the line that begins **int addr:**. This is the number that you enter into the **mc = minecraft.Minecraft.create("192.168.2.234")**. Remember on the Glove Raspberry Pi to enter the IP address of the other player's Raspberry Pi, not yours.

15 Run both programs

No game would be complete without some healthy competition and strategy. A second program is deployed by the other player on the network which tracks and registers if they catch or hit you. The program checks the block that they have hit and compares it to the player's location.

16 Test for hits

To check if the other player has hit you, run the second program on the Raspberry Pi of the player who is doing the chasing. The program basically finds the other 'glove' players current position and stores it in a variable. It then compares the position that you hit with your sword, recording and storing this too. The program then compares the values, if they match then you have hit the other player and have won the game. If not, get ready for a tirade of power moves. Note that in order to monitor where the other player is, you must set the code line **mc1 = minecraft.Minecraft.create( )** to the IP address of the Glove Raspberry Pi; for example, **mc1 = minecraft.Minecraft.create("192.168.1.251")**.

17 Game on

Now you are ready to play, check again that the IP addresses are set for the other Raspberry Pi and not your own. Build a new *Minecraft* world and start a new game on the Raspberry Pi with the player who is chasing. When loaded, the glove player joins the multiplayer game – this will be called Steve (see Step 11). When loaded, you should see both players in the world. Then run the 'Pi Glove power moves' program, and on the other Pi run the 'You hit me program'. Don't forget to set the IP addresses to each other Raspberry Pi.

Once set up, you can modify the power moves, use different blocks and add new moves. You could create a timer and a scoring system to track which player can survive the longest. If you are feeling adventurous, you may want to make another Power Glove, one for each player.

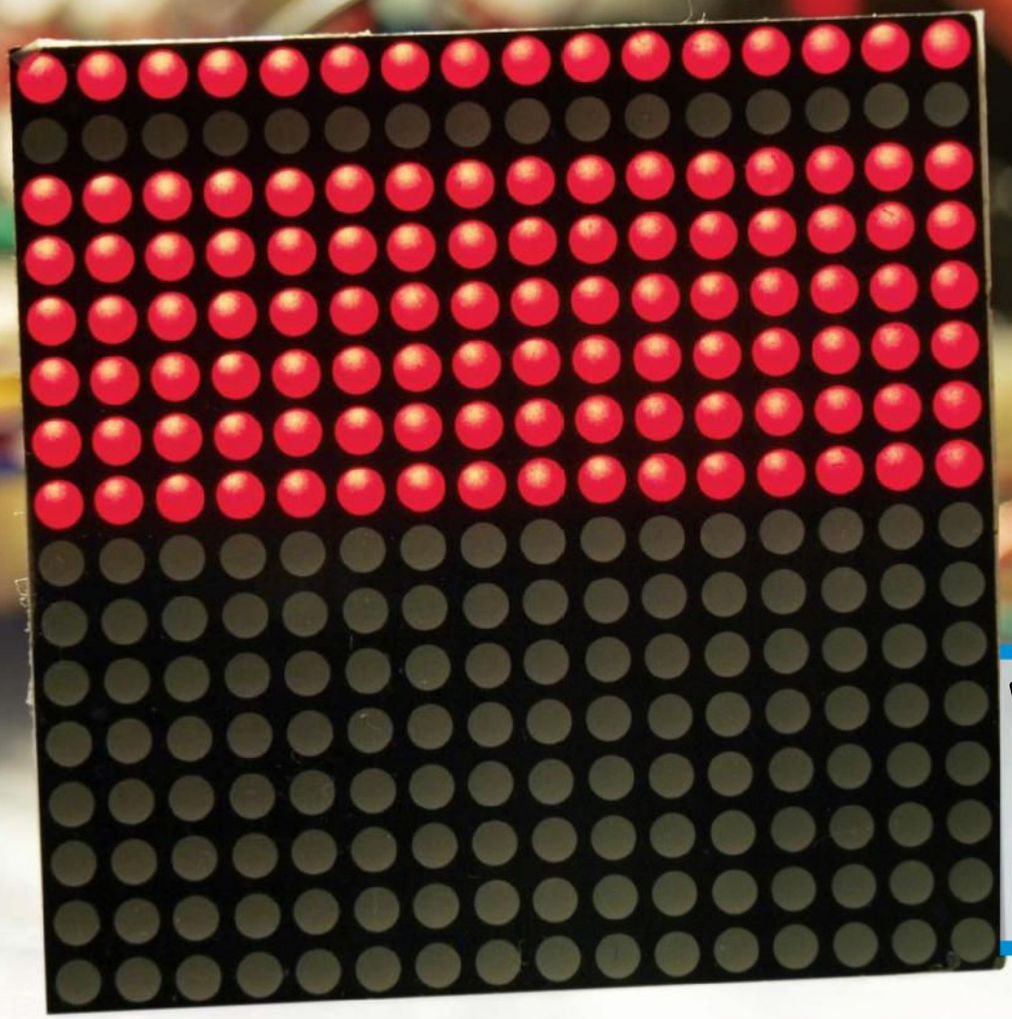
> The program compares the values, if they match then you have hit the other player and have won the game. If not, then get ready for a tirade of power moves

PiGlovePowerMoves.py (Cont.)

```
#Main code to run
try:
    lava_check = 0
    mc.postToChat("Minecraft Power Glove Enabled")
    while True:
        if GPIO.input(17) == 0:
            Firewall()
        if GPIO.input(18) == 0: #needs to stop
            Lay_Lava()

            #GPIO.output(18, GPIO.LOW)
        if GPIO.input(9) == 0:
            Mega_Jump()
        if GPIO.input(4) == 0:
            Mega_Water_Puddle()

except:
    print "Error"
```

YouWereHit.py

```
import time
from mcpi import minecraft

mc1 = minecraft.Minecraft.create("192.168.1.245")
#The other players IP address goes here

mc = minecraft.Minecraft.create()
mc.postToChat("###Here I come")
Hit = 1

while Hit == 1:

#Find the block stood on
  stood_x, stood_y, stood_z = mc1.player.getTilePos()
  time.sleep(3)

  blockHits = mc.events.pollBlockHits()
  if blockHits:
    for blockHit in blockHits:
      if stood_z == blockHit.pos.z and stood_y == blockHit.pos.y+1:
        mc.postToChat("I got you")
        time.sleep(2)
        mc.postToChat("###GAME OVER###")
        time.sleep(1)
        Hit = 0
        mc.postToChat("###Restart Hit Code")
```



What you'll need

- Breadboard & wires
- 16x16 LED Matrix
- 2x 74HC238
- 2x 74HC244
- 16x 220 Ohm Resistor

Build a complex LED matrix

LED matrix display systems find use everywhere from gaudy kebab shops to impressive steampunk-styled systems

Driving LEDs in an efficient fashion is a science of its own. The common availability of single-board computers has put the necessary technology within reach of everyone.

When dealing with LED displays, two different systems must be considered. We will focus on traditional matrix-based systems made up of one or more LEDs. Their affordable nature makes them ideally suited to classic display applications: they communicate currency prices, provide stock-brokers with updates from the trading floor and have even been used as basic displays for primitive oscilloscopes.

Finally, we will also provide you with an overview of electronic basics. This tutorial is a bit more advanced than the ones we've featured up until now, and it's also worth noting that we're going to be programming with C rather than Python. Follow along using the code listing annos.

01 Think about LEDs

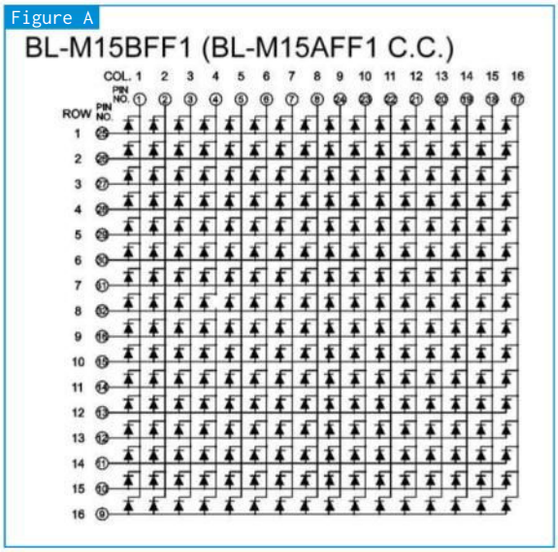
Standalone LEDs are primitive – they light up once current flows through them. Driving a few LEDs is as easy as connecting them to GPIO pins along with a resistor. Sadly, this method becomes wasteful once more than a few of them get involved – driving 16 diodes ties up 16 pins.

02 Arrange your diodes

Methods were devised to reduce the number of pins needed. Matrix-based systems are resilient to individual diode failures, and provide a pin-to-LED ratio of n=(n/2)^2. The following steps assume a 16x16 LED matrix which is made up according to Figure A. Since LEDs permit current in only one direction, you can enable a single LED by bringing the corresponding pins high and low.

Our LED model has a total of 32 inputs, which overwhelms older versions

Figure A



Above The extended version of this schematic is inside **FileSilo.co.uk** – just sign in and head to the issue #151 page

03 Harness the MUX

Our LED module has a total of 32 inputs, which overwhelms older versions of the RPi. The first way to restrict their number comes in the shape of the 74HC238, a component described as a 3-to-8 line decoder/demultiplexer. Its function is described in the Figure B image on the next page.

04 Separate concerns

Chip two goes by the name of 74HC244, which is described as an octal buffer with tri-state capability. Tri-State outputs can physically disconnect themselves from the bus line. This permits you to tie their outputs together without fear of short circuits. As long as all but one chip are in tri-state mode, no current can flow between high and low output pins.

05 Round them up

Four GPIO pins control the currently-enabled 'line' of the display. Three pins configure the address which is to be emitted, while the signal emitted from the fourth pin is connected to the activity inputs. This ensures that but one IC is active. The 74HC244 ensures that but one of the two groups is active at any given time.

06 Configure the pins

We used a library from Hussam Al-Hertani's Hertaville blog (**hertaville.com/2014/07/07/rpimmapgpio**). The first step involves setting output functions. As the GPIOs are set to outputs, the tri-state feature might connect the internal state to the output pins of the IC. This could lead to internal shorting if the output is not turned off.

Full code listing

Step 12

```
#include "€ammapGpio.h"
#include <unistd.h>
#include <stdio.h>

#define PINA0   2 // 3
#define PINA1   3 // 5
#define PINA2   4 // 7
#define PINA3   14// 8
#define PINCS0  17// 11
#define PINCS1  18// 12

#define PIND0   23// 16
#define PIND1   24// 18
#define PIND2   10// 19
#define PIND3   9 // 21
#define PIND4   25// 22
#define PIND5   11// 23
#define PIND6   8 // 24
#define PIND7   7 // 26
```

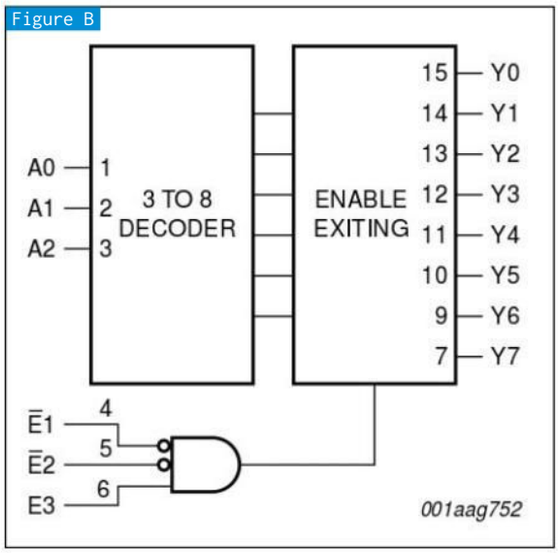
Step 07

```
void setAddress(unsigned char _which, mmapGpio* _where)
{
  if(_which&1)
  {
    _where->writePinHigh(PINA0);
  }
  else
  {
    _where->writePinLow(PINA0);
  }
  if(_which&2)
  {
      _where->writePinHigh(PINA1);
  }
  else
  {
      _where->writePinLow(PINA1);
  }
  if(_which&4)
  {
      _where->writePinHigh(PINA2);
  }
  else
  {
      _where->writePinLow(PINA2);
  }

  if(_which&8)
  {
      _where->writePinHigh(PINA3);
  }
  else
  {
      _where->writePinLow(PINA3);
  }
}
```

Step 08

```
void safelySetRow(unsigned char _which, mmapGpio* _where)
{
  _where->writePinHigh(PINCS0);
    _where->writePinHigh(PINCS1);
  if(_which==0)
  {
    _where->writePinLow(PINCS0);
  }
  else
  {
```

Figure B



07 Power the MUX

Create a convenience function taking an address ranging from zero to 15. It is converted into pin outputs for our 3-to-8-demultiplexer. The effect of this is that all but one of the sixteen rows is to be supplied with energy.

08 Select a row

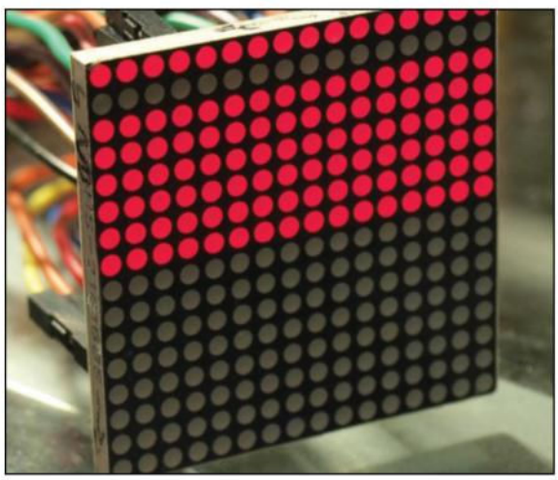
In the 74HC244, we first disable both units and proceed to turning on the one which is needed. This sequence prevents ghosting during the switching process.

09 Do the main loop

The outer part of the loop consists of logic that manages the addressing of the individual rows. Our program must flash the individual LED groups one after another using the building blocks described in the next step.

10 Complete the loop

Writing out data is accomplished in a sequence of three commands. We select the row, configure the column and then write out the data bits that are to be displayed. A small pause is observed in order to give the LEDs some time to 'burn into' the viewer's eyes.



Above Digital LED matrices like this one give you far more control over each individual 'pixel' in the display

Full code listing

Step 08

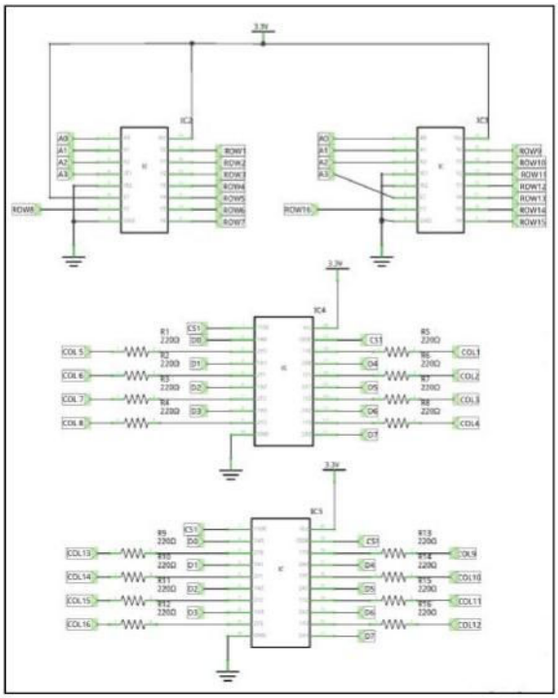
```
        _where->writePinLow(PINCS1);
    }
}
```

Step 11

```
void setData(unsigned char _which, mmapGpio* _where)
{
    if(_which&1)
    {
        _where->writePinHigh(PIND0);
    }
    else
    {
        _where->writePinLow(PIND0);
    }
    if(_which&2)
    {
        _where->writePinHigh(PIND1);
    }
    else
    {
        _where->writePinLow(PIND1);
    }
    if(_which&4)
    {
        _where->writePinHigh(PIND2);
    }
    else
    {
        _where->writePinLow(PIND2);
    }
    if(_which&8)
    {
        _where->writePinHigh(PIND3);
    }
    else
    {
        _where->writePinLow(PIND3);
    }
    if(_which&16)
    {
        _where->writePinHigh(PIND4);
    }
    else
    {
        _where->writePinLow(PIND4);
    }
    if(_which&32)
    {
        _where->writePinHigh(PIND5);
    }
    else
    {
        _where->writePinLow(PIND5);
    }
    if(_which&64)
    {
        _where->writePinHigh(PIND6);
    }
    else
    {
        _where->writePinLow(PIND6);
    }
    if(_which&128)
    {
        _where->writePinHigh(PIND7);
    }
    else
```

LED stripes

Two versions of LED strips are offered. 'Primitive' ones are based on analogue technology. In it, an entire strip of diodes has the colour set by the three input pins. Systems such as the mega-display shown in the left-hand image require the use of the digital version. They are based on the concept of the shift register. Your system inputs individual colour values which are then pushed on along the strip.



Above This is the full schematic of the LED matrix that we're working with here (you can also view it at its full size on FileSIlo)

11 Energy control

LEDs light up if current flows through them. SetData pulls the pins of the 74HC244 low to ensure that the energy supplied from the 74HC238 can flow through the diode.

12 Avoid GPIO trouble

The Raspberry Pi Foundation has a tendency to change the layout of the expansion header regularly, a habit which professional manufacturers of process computers abhor. It's recommended to handle the mapping between pins and functions via a set of defines. Our code is optimised for a Rev2 Raspberry Pi with a 'short' header – 40-pin variants will require readjustments making sure the physical pin numbers correspond to the logical GPIO numbers.

13 Add example data

Test the code by setting the datastore to a value of your choice. Setting 64 to all fields will disable one row in each part of the display.

14 Kick it off

Check all connections between the planar and the single-board computer, and proceed to starting the compiled app. Don't forget to use the sudo command – direct memory access is restricted to root in order to prevent apps from causing havoc in the physical memory. Users are accustomed to this, so requiring them to put a sudo in front of the command doesn't cause concern.

15 Notice a flicker

Sharp-eyed readers will notice an occasional flicker where one line appears brighter than the others. This is caused by the stalling of the program – if the kernel does other work, the switching routine can't run. We could solve this problem by using a real-time Linux kernel.

Full code listing

```
        {
            _where->writePinLow(PIND7);
        }

    }
```

Step 06

```
    int main(void)
    {
      mmapGpio rpiGpio;
      //Set outputs
      rpiGpio.setPinDir(PINA0,mmapGpio::OUTPUT);
      rpiGpio.setPinDir(PINA1,mmapGpio::OUTPUT);
      rpiGpio.setPinDir(PINA2,mmapGpio::OUTPUT);
      rpiGpio.setPinDir(PINA3,mmapGpio::OUTPUT);
      //TURN OFF ASAP!
      rpiGpio.setPinDir(PINCS0,mmapGpio::OUTPUT);
      rpiGpio.writePinHigh(PINCS0);
      //TURN OFF ASAP!
      rpiGpio.setPinDir(PINCS1,mmapGpio::OUTPUT);
      rpiGpio.writePinHigh(PINCS1);
      rpiGpio.setPinDir(PIND0,mmapGpio::OUTPUT);
      rpiGpio.setPinDir(PIND1,mmapGpio::OUTPUT);
      rpiGpio.setPinDir(PIND2,mmapGpio::OUTPUT);
      rpiGpio.setPinDir(PIND3,mmapGpio::OUTPUT);
      rpiGpio.setPinDir(PIND4,mmapGpio::OUTPUT);
      rpiGpio.setPinDir(PIND5,mmapGpio::OUTPUT);
      rpiGpio.setPinDir(PIND6,mmapGpio::OUTPUT);
      rpiGpio.setPinDir(PIND7,mmapGpio::OUTPUT);
```

```
    unsigned char dataStore[2][16];
```

Step 13

```
    for(int j=0;j<2;j++)
    {
    for(int k=0;k<16;k++)
    {
      dataStore[j][k]=64;
    }
    }
```

Step 09

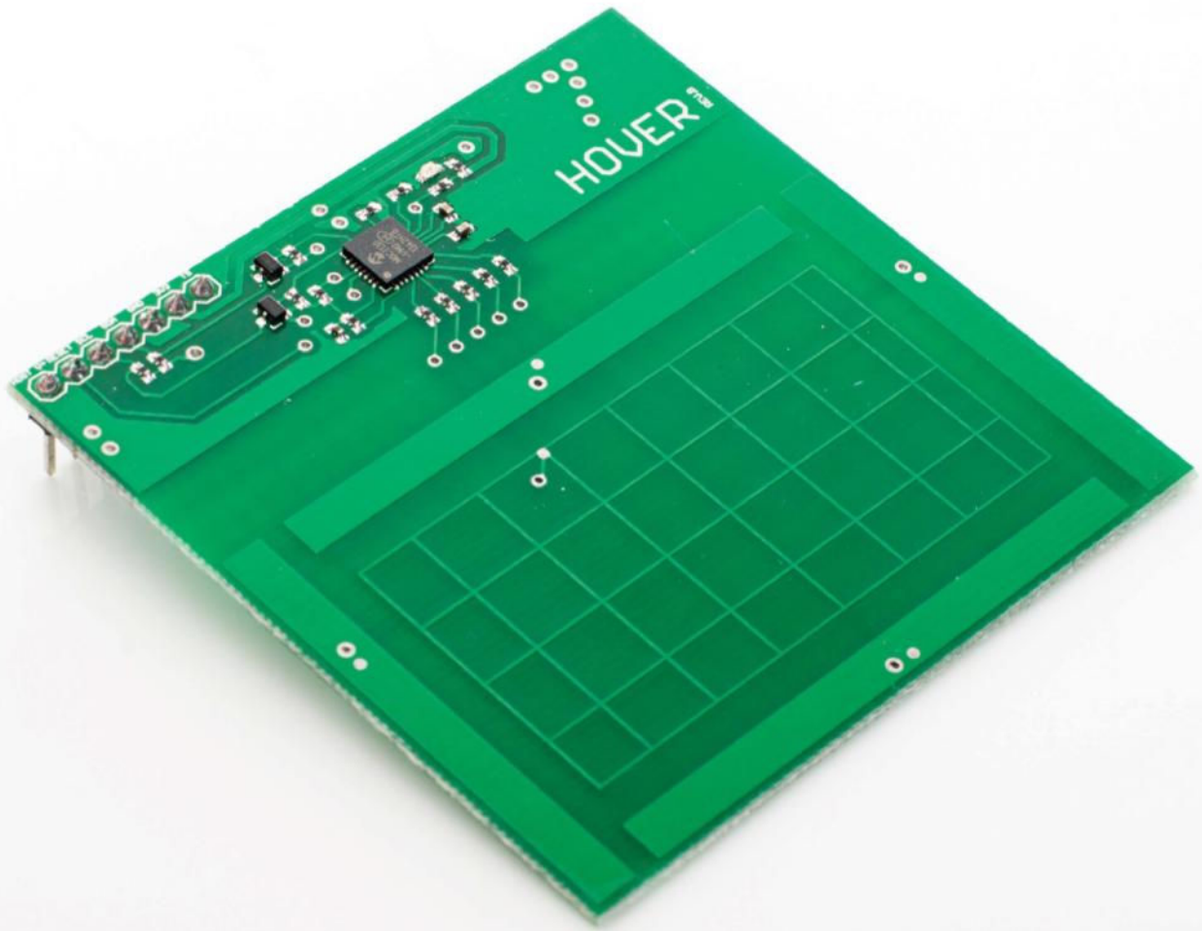
```
      int blockCounter=0;
      int rowCounter=0;
          while(1)
        {
        blockCounter++;
        if(blockCounter==16)
        {
          if(rowCounter==0)
          {
            blockCounter=0;
            rowCounter=1;
          }
          else
          {
            blockCounter=0;
            rowCounter=0;
          }
        }
```

Step 10

```
        safelySetRow(rowCounter, &rpiGpio);
        setAddress(blockCounter, &rpiGpio);
        setData(dataStore[rowCounter][blockCounter], &rpiGpio);
        usleep(50);

        }

      return 0;
    }
```

Add gesture control to your Raspberry Pi

Hover is an impressive add-on board for your Raspberry Pi that allows you to easily add touch and gesture control to any project

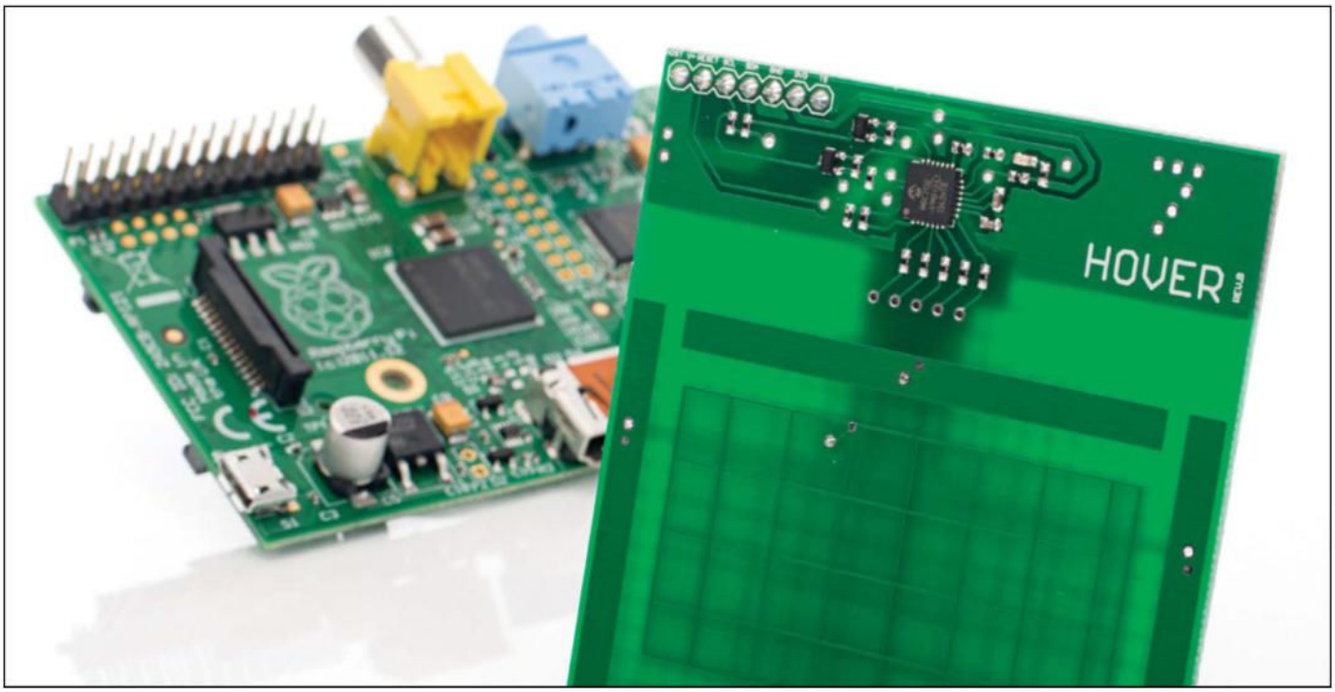


What you'll need

- Raspberry Pi
- Hover
- Breadboard
- Male to female jumper cables
- Speaker or headphones

People often ask what the best way is for them to get started with Raspberry Pi. Obviously this does depend on the individual user and what they want to achieve and get out of any project, but in a more general sense it's often the hardware projects that win out for getting to grips with it. They teach a variety of skills (including programming, circuit building, soldering, hardware design and much more) and are also varied enough to both keep beginners interested and allow them to work out for themselves exactly what aspect they love best. Even a seasoned professional will get a serious kick out of a bit of physical computing and automation! This is one of the unique features of the Pi compared to traditional "black box" computers; you can break out of the usual boundaries and interface with everyday objects like never before.

One of the most important aspects of a hardware project is often the user input mechanism, and as technology is refined we see new and more intuitive ways to accomplish this task. Gesture and touch control is now present in a large number of consumer devices and even the biggest technophobes are starting to embrace the ease of this technology. It is time to bring your Raspberry Pi projects into the 21st century with Hover!



Above You can tap the Hover or swipe in four directions just above it

The physical pins you should be using on the Raspberry Pi are 1, 3, 5, 6, 16 and 18

01 Get the gear!

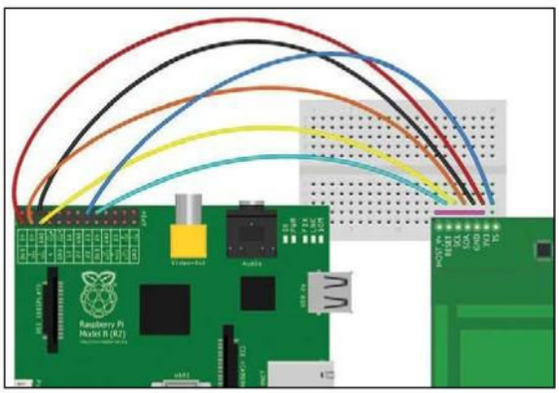
The Hover add on board is available to purchase direct from Hover (**http://www.hoverlabs.co/#shop**) for $39 (£25), however this will ship from Northern America and therefore if you are based in the UK or Europe it will likely be quicker and cheaper to order from one of the other retailers listed via the above link. The added benefit of ordering from a retailer is that if you need any of the other items you can likely get those at the same time!

Hover will work perfectly with any Raspberry Pi, including both the new plus versions and the older models – just make sure your system is fully up to date with:

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

02 Update GPIO and I2C

When making use of GPIO and I2C (or any other interfacing technique on the Raspberry Pi) it is always good practice to update to the very latest software versions possible. Newer versions typically have bug fixes and additional futures which can come in very handy. GPIO and the RPi.GPIO Python library are installed by default on Raspbian, but you may need to enable I2C if you haven't already. This is a fairly standard process and has been covered many times so we won't go into it here. We would, however, highly recommend the brilliant I2C setup tutorial from Adafruit (**https://learn.adafruit.com/adafruits-raspberry-pi-lesson-4-gpio-setup/configuring-i2c**).



03 Set up the hardware

Make sure your Raspberry Pi is powered down and not connected to power before starting this step, to avoid any unnecessary damage to your Raspberry Pi. Pick up your Hover, breadboard and wires and connect the as shown in the Fritzing diagram. The physical pins you should be using on the Raspberry Pi are 1, 3, 5, 6, 16 and 18. Whilst a Model B Pi is shown, this will be the same connection on a Model A, B, A+ or B+ of any revision. Once completely set up like the image, reconnect the power cord and open an LXTerminal session.

04 Check the connection

Hover connects to the Raspberry Pi through the I2C interface located on the main 26 or 40 pin GPIO bank (depending on which version of the Raspberry Pi you are using). There is a very easy way to check if your Raspberry Pi is correctly connected to Hover using the simple command line I2C tools. Issue the following command:

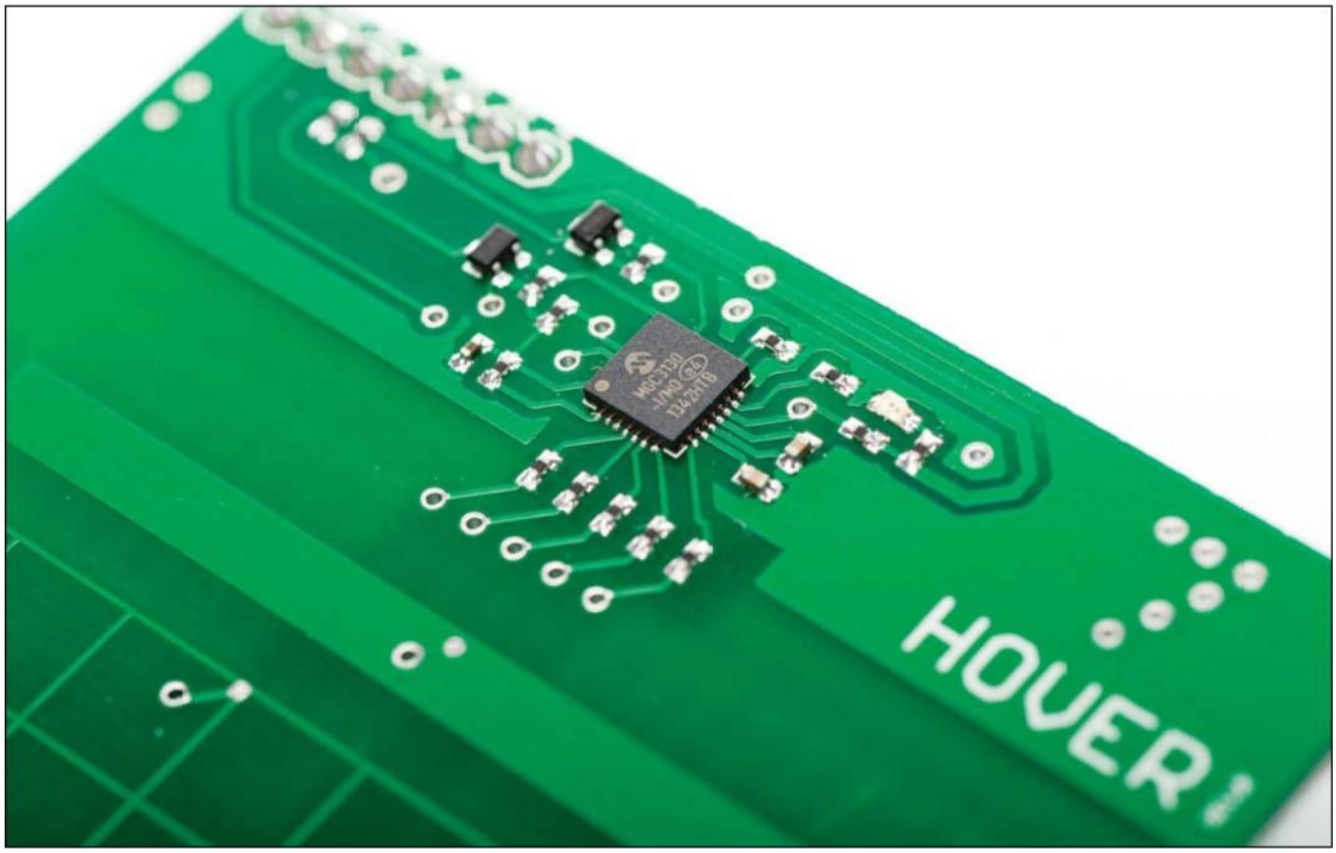
```
sudo i2cdetect -y 1
```

If you see 42 in the response then you are successfully connected to Hover!

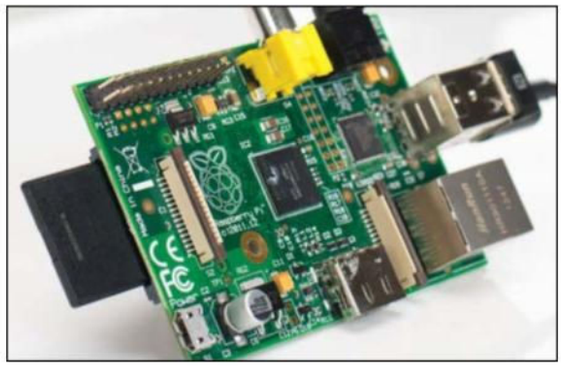
Plenty of platforms

The Hover board has intelligent on-board level shifting, meaning that it can be used with either 3.3V or 5V logic levels which means it can be used with pretty much any microcontroller your heart desires. There are connection examples and code snippets available for Arduino, Sparkcore and PCduino on the Hover website (**hoverlabs.com**) and these can also be adapted to suit other devices fairly easily.

If you decide to create your own example with another device then why not submit a pull request to the Hover GitHub (**github.com/jonco91**) if you are happy to share!



Above This MGC3130 chip works as the 3D tracking and gesture controller



Why Python?

Python is extremely useful for beginners due to its easy-to-understand syntax, fairly prose-like formation and the flexibility and ease of acquiring existing software libraries to help your projects. It is also the official programming language of the Raspberry Pi and is therefore very well supported within the community. That is not to say that Hover will not work with other programming languages; simply that the creators of Hover have not yet released any code libraries in other languages.

05 Using a Rev 1 Pi?

In the code, we have passed an option "-y 1" which tells the operating system which I2C bus to look at (there are two on the BCM2835 processor on the Pi). The first revision Raspberry Pi (the one that initially launched in February 2012 with 256MB of RAM) made use of I2C bus 0, whereas all other versions of the Raspberry Pi since have used I2C bus 1. So the above code would change to:

```
sudo i2cdetect -y 0
```

And you should expect the same output (42) as in step 7. Additionally you will need to edit line 27 of the Hover_library.py file, changing bus = smbus.SMBus(1) to bus = smbus.SMBus(0). A patch that automatically detects the Raspberry Pi version and makes this change for you has been submitted, but not yet accepted into the master branch so this may not be necessary in future versions.

06 Download the sample code

Now you have everything hooked up correctly and your Raspberry Pi is fully up to date, it is time to get the Hover Python library, which makes using the board from Python scripts extremely easy. You can get this using the following command:

```
git clone https://github.com/jonco91/hover_
raspberrypi.git
```

This should download a folder called hover_raspberrypi to your /home/pi directory containing all of the files needed for this article. Alternatively you can download the zip file from https://github.com/jonco91/hover_raspberrypi/archive/master.zip.

07 Run the example file

The current Hover library is simply a Python file with all of the necessary functions included within it, rather than an installable package (however, this may change in the future). In order to use the functions contained within the Hover_library.py script discussed above, it is therefore necessary to make sure that the Hover_library.py script is located in the same folder as any script you have written that makes use of any of the Hover functions. In a terminal session, navigate to the folder containing the Hover_example.py file and run it using:

```
sudo python Hover_example.py
```

The Hover board will initialise and you will then see a message "Hover is ready", meaning you are good to go.

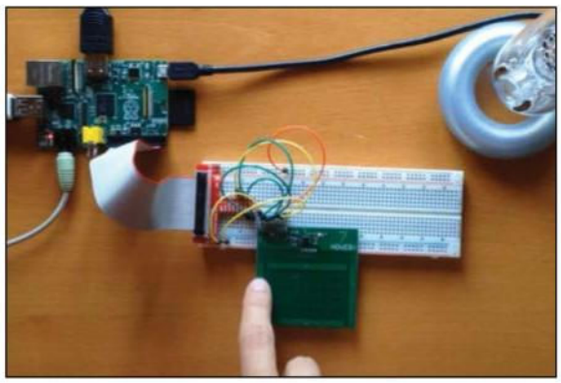
08 Investigate the output

Once you have completed step 7, if you touch the Hover board or make gestures above it you will begin to see output in the terminal which is a bunch of 0s and 1s and then a description of what it has seen – right swipe, north tap, etc. The way the Hover works is that it can sense any one of nine different actions and these are sent to the Raspberry Pi over I2C as an 8-bit binary value. The first three bits describe whether it was a touch or gesture event and the remaining five bits describe the specific type or direction of the event. The exact breakdown can be seen in the code listing to the right.

09 Enable 3.5mm audio

Grab your speakers and plug them in to the 3.5mm jack plug on the Raspberry Pi. You will then need to route audio to the 3.5mm jack using the following command (you can skip this step if you are using an HDMI display, which has in-built audio):

```
sudo amixer cset numid=3 1
```



10 Make a drum machine

In the hover_raspberrypi folder is another folder called examples that contains code and sounds to turn Hover into a drum machine! Navigate to the hover_raspberrypi directory and then copy the Hover_library.py file into the examples folder by using:

```
cp Hover_library.py examples
```

You can then move into the examples folder and run the Hover_drum.py file using:

```
cd examples
sudo python Hover_drum.py
```

Make some gestures and taps on and around Hover and you will have your own basic drum machine!

11 Create your own responses

The great thing about having a Python library available is that it is easy to integrate this device into any of your existing or future projects. The code shown is all you need to get started with Hover. You will see that on line 15 and onwards there are comments saying "code for ... goes here". Essentially all you need to do is insert the actions you want to occur on the particular event mentioned in the comment and you will be up and running... it really is that easy!

Full code listing

```
import time
from Hover_library import Hover

hover = Hover(address=0x42, ts=23, reset=24)

try:
  while True:

    # Check if hover is ready to send gesture
    # or touch events
    if (hover.getStatus() == 0):
      # Read i2c data and print the type of
      # gesture or touch event
      message = hover.getEvent()
      type(message)
      if (message == "01000010"):
        # code for west touch goes here
      elif (message == "01010000"):
        # code for centre touch goes here
      elif (message == "01001000"):
        # code for east touch goes here
      elif (message == "01000001"):
        # code for south touch goes here
      elif (message == "01000100"):
        # code for north touch goes here
      elif (message == "00100010"):
        # code for swipe right goes here
      elif (message == "00100100"):
        # code for swipe left goes here
      elif (message == "00110000"):
        # code for swipe down goes here
      elif (message == "00101000"):
        # code for swipe up goes here

      # Release the ts pin until Hover is
      # ready to send the next event
      hover.setRelease()
    time.sleep(0.0008)   #sleep for 1ms

except KeyboardInterrupt:
  print "Exiting..."
  hover.end()

except:
  print "Something has gone wrong..."
  hover.end()
```

12 Other project ideas

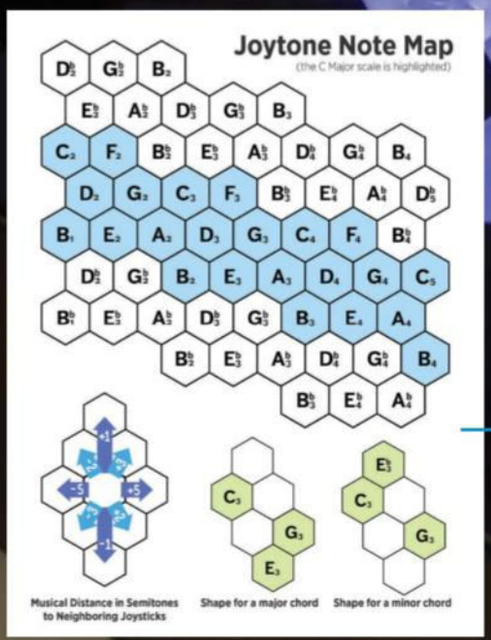
Most of you are probably now wracking your brains for projects you could use Hover in, but let's face it – pretty much any project that requires physical interaction would be made better with touch and gesture control. If you think it is cool but are lacking inspiration, we recommend looking at the projects section of the Hover website at **http://www.hoverlabs.co/projects**, where there are projects by the creators and community alike. If you make something cool, be sure to send us the pictures!

Where are the hoverboards?

Did you come here looking for information on how to build your space age transportation device? We can't help you with that, but we don't want to leave you disappointed! Hoverboards were first popularised as a fictional personal transportation method in the 1989 film *Back To The Future Part II* and took the appearance of a levitating skateboard with no wheels. 25 years later and it seems we might be getting close to turning this dream into a reality. Hendo Hoverboards have created a $10,000 hoverboard which uses a principle similar to that of maglev trains to generate lift (**kck.st/ZMd9AA**), and more recently Ryan Craven has created a much cheaper alternative using four leaf blowers and some other cheap parts (**mrhoverboard.com/about**).

Chords You can play chords by manipulating three thumbsticks at once – how you move them affects the sound of the notes in the chord

Thumbsticks Push forward/backward to switch between triangles and reverse sawtooth waves or push left/right to affect the oscillators



Isomorphic All the chords and scales you can play have the same geometric shape, no matter what key you're playing them in

Hexagonal grid The Joytone uses a hexagonal layout for the notes like electronic musical instruments such as Rainboard or apps like Musix

Above (inset) Getting to know the musical distances between adjacent thumbsticks is the key to learning to play the Joytone

Right There's a marathon lab session behind this six-note demo unit that Sharples and Glanzman presented at the CIS Senior Design Fair

Components list

- Raspberry Pi Model B
- Cypress PSoC 4
- Arduino Micro
- 57 Xbox 360 thumbsticks
- 57 NeoPixel Diffused 8mm Through-Hole LEDs (Adafruit)
- 8 16-channel analogue multiplexers
- 5 16mm Illuminated Pushbuttons (Adafruit)
- 200mm SoftPot Membrane Potentiometer

How I made: Joytone

Reinventing the keyboard with a grid of joysticks

What inspired the Joytone?

David Sharples The Joytone is an expressive isomorphic musical instrument which came out of my frustration trying to learn to play musical instruments. I was trying to teach myself piano and was wondering why there are white and black keys – it makes things so much harder. I learned a bit of music theory and came to realise that a major scale is just a well-defined musical structure; the gaps between the notes in a major scale are the same regardless of which major scale it is. But, because there are white and black keys on a piano, you can play a C major scale just by running up all the white notes. If you want to play a C# major scale you have to throw in a bunch of black keys. It's hard to remember and you have to build up muscle memory. So one of the goals with this project is to build an instrument that doesn't have that bias based on the keys – so it's isomorphic; that's what that means.

And you're using analogue thumbsticks?

David Glanzman They're Xbox joysticks...

Sharples That's the second big goal of the project. When I was researching this, I noticed there were some instruments that had these isomorphic keyboards – a grid of hexagons – but the issue was that they were just buttons, they didn't have the same richness and depth as an actual musical instrument. When you're playing a guitar there are a million ways to affect the sound – how hard you're pushing on the string, how hard you pluck it, where on the fret you're holding it, if you bend the string or not – and you can get all these rich sonic qualities. So we wanted to make it isomorphic and we wanted to make it expressive. We used these thumbsticks because you get two channels of analogue control in this familiar little interface. One axis changes the waveform of the synthesised sound from a triangle wave (has a pure, bell-like quality) to a reverse sawtooth wave (has a buzzy, bright sound, like a trumpet). There are two oscillators creating each note and if you push the thumbstick to the left, those oscillators are exactly in tune, making a very soft sound. If you push it all the way to the right then they're offset by a couple of hertz, which makes a wide, rich sound. Then the amount that you rotate the joystick gives the volume. So you have two and a half dimensions of control there, which adds some depth to the sound.

What is the role of the Raspberry Pi?

Sharples There's a two-brain system going on – we have the Raspberry Pi and then we have the Cypress PSoC 4. The Cypress PSoC 4 does all the heavy lifting with the data reading.

Glanzman It does all the measurements for the joysticks. It's got ADCs in it that convert analogue to digital, and then basically looks at each axis for each joystick in sequence, records it, and then waits for the Raspberry Pi to ask it for data values for each of the joysticks.

Sharples There's 57 thumbsticks and each one has two analogue channels, so that's 114 analogue channels total. So what we did was we had eight 16-channel multiplexers hooked up to the PSoC and then the PSoC sends a signal to all of them that says 'give me channel one'. Then it reads the eight channels, and then it says 'give me channel two' and it reads the eight channel twos. After it does that for all 16 channels it then has this full bank of current data. The Raspberry Pi periodically says 'give me all your most recent data', so the PSoC forwards the data to the Raspberry Pi, which then does a little bit of processing in Python and then sends commands to PureData, which is our synthesiser program.

What's the Arduino component doing?

Sharples Each thumbstick also has an RGB LED in its hexagonal cell, and our intention was to use these to show which nodes are in key or out of key. We also wanted to guide the user through a scale – or even a song, showing the next note that they're supposed to play – but we ran into some technical difficulties. The ideal setup for this is that you daisy-chain all these lights in a big line and then hook them up to an Arduino Micro, which is really good at controlling these specific lights from Adafruit, and then it can just push all of this data down the line and you should just be able to put whatever colours you want on each light individually. But we had some problems with the signal and could only get about four lights to behave.

Is it easy to learn to play the Joytone?

Sharples The barrier to entry is much lower. It's funny, when we first got it working, David was playing with it, wearing headphones, and he sort of stopped talking and was pushing a few of the joysticks, like 'Wait, wait...', and then he just played back Bach. So the key to learning it is just learning a little, tiny bit about the structures in music theory. There's a shape you can hold your fingers in that will play a major chord; once you learn that shape, that's it – that's how all of the Joytone's major chords are played.

Glanzman When it comes to learning the Joytone, you have to attack musical instruction differently than you would with another instrument. When you learn something like the piano, you learn that this is D major, this is F# minor – you learn different things based on the note, the actual class. But with the Joytone, the pitch class is totally irrelevant because we hear everything in relevant terms, and you play everything in relative terms. So to learn the instrument, you don't even have to discuss pitch classes – you just talk about relative distances. So major thirds or minor thirds, fifths, fourths – it's distances between notes instead of the actual note values. I think if you phrase musical instruction in those terms, in terms that we experience music in rather than the terms we normally go through to create music, it becomes a much more natural interface because it's built on that type of instruction, making it simple to learn.

David Sharples is an interaction designer and graduated from the University of Pennsylvania's Digital Media Design programme

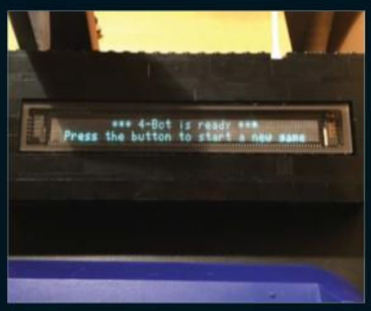
David Glanzman is a sophomore in Computer Engineering at the University of Pennsylvania. David has worked on microprocessor design, audio electronics, medical devices and more

Like it?

Interested in learning more about isomorphic instruments? Check out David Sharples' senior design blog: **davesharpl.es/blog**

Further reading

Here's the final demo video of the Joytone, comprising David Glanzman's virtuoso Bach performance: **bit.ly/1vfXnlw**



How I made: Connect 4 robot

Think you can outsmart a robot? David Pride's 4-Bot will put you to the test

What inspired you to make this?
I took part in Pi Wars, hosted at Cambridge University last December, and following this I was approached to supply some robots to the Raspberry Pi foundation for their 'Robot Pod' as part of the BETT show in January. My wife bought me the brilliant MeArm kit and I used it to build a LEGO block sorter, as you do. This used the Pi camera module and a colour recognition script I wrote in Python to identify the different coloured blocks and then used the arm to drop them in the correct 'buckets'. You can find a video over at: **https://youtu.be/FJ8WV1uLhFA**

Based on this I was then looking for other uses for the colour-capture code and Connect 4 seemed like a really good choice. Research soon led me to find that the game, and the logic behind it is far from simple! There is good information online; however, while I found many versions of Connect 4 for Python, few of them ran successfully on the Pi.

How long has the development of the robot taken you? Did you encounter many problems during its development?
It took approximately two months of evenings and weekends to complete the bot. The trickiest part was undoubtedly capturing the game board accurately every time. It is extremely light-dependant as the Python module works by capturing the RGB value of the 42 spaces on the board. These values however do change dramatically depending on the lighting.

I wrote a 'testcard' script that can be run with counters in known locations. This script then reports back what it thinks it sees, and the tolerances for the RGB components can then be adjusted until the result matches what is actually there on the real game board. This made the game more portable as it can be adjusted to its surroundings each time.

What role does the Raspberry Pi play in this project?
The Raspberry Pi is integral to the bot; the game program is written in Python and is configured to run at boot. The servos that control the arm and the LCD display are controlled by the PiXi board designed by Mark Cantrill at Astro Designs (@AstroDesignsLtd). The board is a monster of an add-on board with built-in FPGA and the ability to control 50+ servos! I was fortunate enough to be given an early production version and Mark is currently working on a Raspberry Pi HAT version, available later this year.

We saw that you used the Minimax algorithm for this project, how does that integrate with the robot?
In regards to intelligence the Minimax algorithm that the game uses is well known – although it wasn't known to me when I started this project! There is a Python class for this algorithm written by Erik Ackermann and Charlene Wong that I adapted and wrote an interface for.

The interface turns the captured Pi camera image into an array of RGB values. These values are then converted in to the matching counters and the result of this is then saved in an array as the 'game state.' This is then passed to the algorithm, which then returns the 'best move' based on the game state. This move is then passed to the control logic for the arm, which picks up a counter and drops it in the correct slot.

> The trickiest part was capturing the game board accurately

How intelligent is the robot? We're assuming you've been able to beat it a few times?
In terms of how well it plays, Connect 4 is a 'perfect' game in mathematical terms. There is a huge but finite number of solutions and they can all be calculated with enough processing power. The trade-off is in the depth of search and therefore the time taken to calculate each move. With a Pi 2 the calculation time is around 5-7 seconds, using a Pi 3 this drops to 2-3 seconds. If you increase the search depth this massively increases the calculation time so I selected a middle ground where the bot plays a pretty mean game but the total time per move is still acceptable. With capturing and processing the image, calculating the next move and delivering the counter the total time per turn is around 25 seconds.

I took 4-Bot to the Raspberry Pi Fourth Birthday party in February where it took on all comers – including Eben Upton himself. We had a proper 'Inventor vs Invention' showdown – which ended in an honourable draw. Over a hundred games were played by children and adults alike over the course of the two days. I suspected that the bot would prove popular so to limit the time taken for each game I altered the code so each player had only ten counters.

Even with this limit, three extremely smart youngsters managed to beat it fair and square. Truly impressive – and congratulations again to James, Louis and William, who each won a new Raspberry Pi 3 for their efforts. My personal tally is roughly 50 per cent lose, 40 per cent draw and 10 per cent win when playing a full game. I am still yet to beat it playing with only ten counters!

So what's next for you?
In terms of what comes next, I am currently a full time MSc Computer Science student. I am just finishing the taught modules, soon to start on my final project and dissertation. I of course want to find something Pi-related. Additionally I was recently sent a very cool educational robot head called OhBot (**www.ohbotrobot.com**) that uses an Arduino and has software that runs under Windows and talks to the bot via USB-to-serial connection. I am currently working on converting that to run on the Ras Pi.

Like it?
Puzzle-solving Pi robots are always well received, and if you like the look of David's Connect 4 robot, go ahead and check out this Rubix cube solving robot from Maxim Tsoy. It'll be a while before it can beat the world record, but it isn't far off: **http://bit.ly/1SvrlQ6**

Further reading
A large part of the development of the Connect 4 robot stems from the Minimax algorithm. It's fairly complex, but adaptable to a wide number of projects and useful when mixing motors and servos with the Raspberry Pi. A complete guide to the algorithm can be found here: **http://neverstopbuilding.com/minimax**

Python imaging library
A complex imaging library works behind the scenes to help 4-Bot identify the colours and location of each game piece. Based on these calculations, it can then work with the MeArm claw to provide the optimal move for that turn

Second hand claw
The rail from a disassembled 3D printer helps move the on-board claw into different positions. It works in tandem with the Python interface designed by David, and has been aligned to make sure counters aren't dropped in the wrong columns

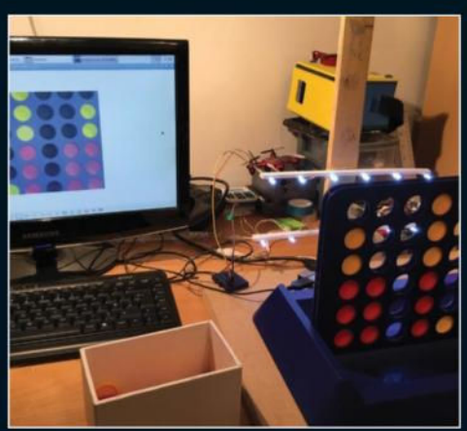
Complete servo control
The primary servos used to control the various arm movements are powered by a PiXi board; one of the key benefits of this board in particular is that it can support up to 50 individual servos, allowing for more fluid movements from 4-Bot

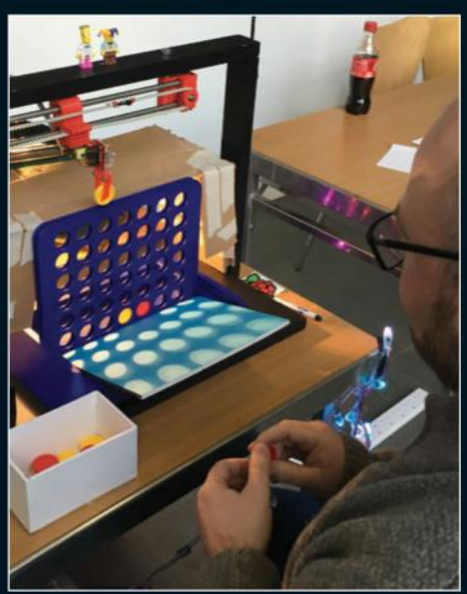
Minimax algorithm
Every move that 4-Bot makes is based around the Minimax algorithm, which is ideal for keeping calculation times down to a minimum. Despite each move being mathematically correct, there's room for error and 4-Bot can be beaten

Components list
- Raspberry Pi board
- Disassembled 3D printer
- Wood
- MeArm claw
- PiXi controller board
- Stepper motor
- LCD message screen

Right Once connected to a Raspberry Pi 3, the 4-Bot takes around 25 second to program and complete its move

Below While each decision that 4-Bot makes is mathematically correct, it can be beaten from time to time





Program a Raspberry Pi quadcopter

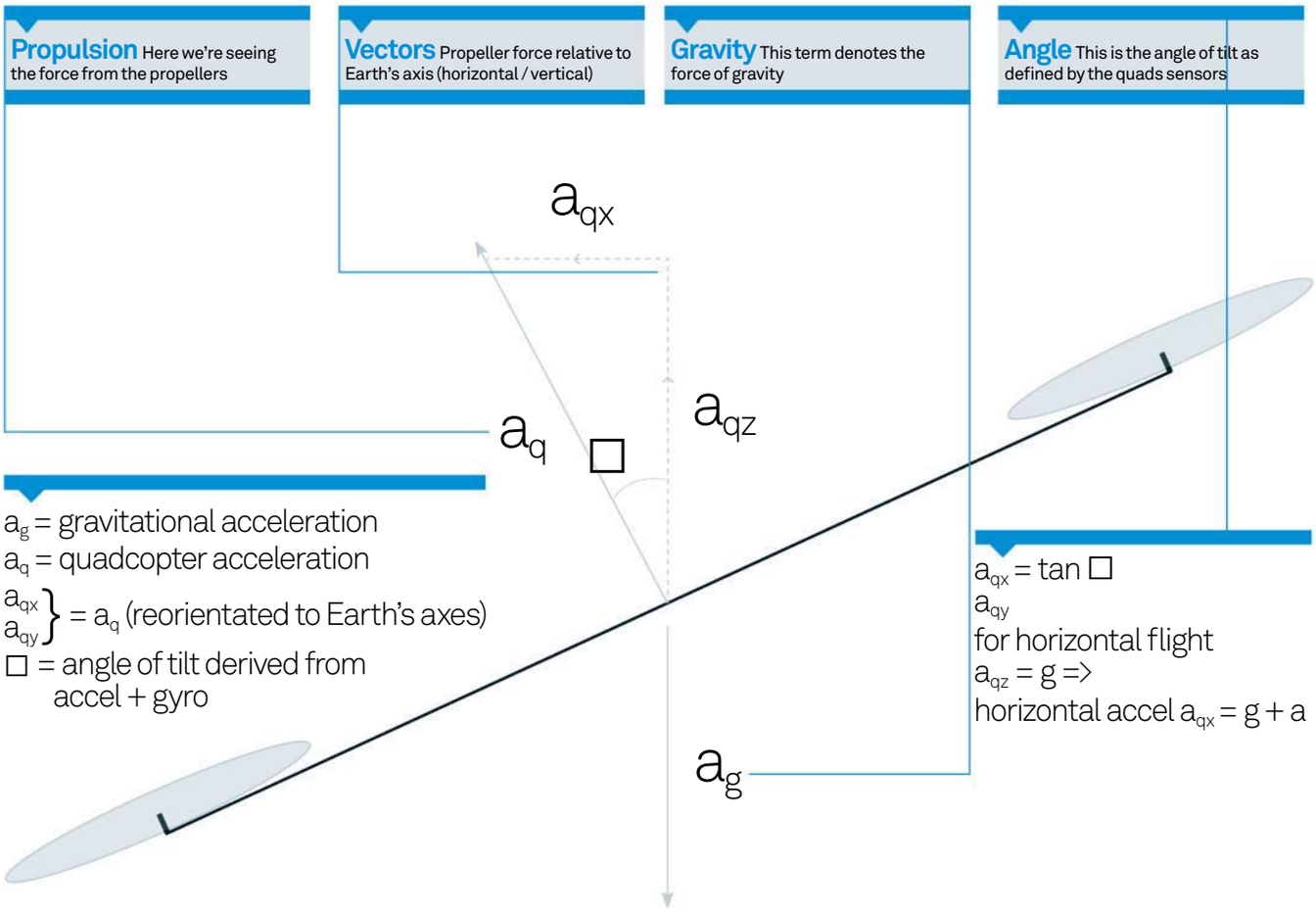
How do you beat a Raspberry Pi robot? You give it wings. Andy Baker shows us how to code our way into the clouds

The Raspberry Pi is a fantastic project board. Since we love a challenge, we set out looking for something to really take advantage of its capabilities in the most spectacular way possible, something that would really push the limits of our hacking and coding skills. We chose a quadcopter.

Kits are available as ready-to-fly (RTF) if you just want the joy of flight, but where's the challenge in that? We started with an almost-ready-to-fly (ARF) kit – the DJI Flame Wheel F450 – all the hardware, but none of the control electronics or software. Many enthusiasts have created DIY quadcopters using Arduino microcontrollers, so we knew a DIY build was possible, but very few, if any, have successfully used the Raspberry Pi.

This article uses the Python code on the disc as a guide through what's needed to build a quadcopter, metaphorically bolting it together so that by the end, you don't just understand the code but also the interaction with the real-world to enable you to build your own quadcopter with confidence.

As you read the article, you can follow the corresponding code by searching for an equivalent tag comment; for example, to find the code related to the '# Angles' section of the article, simply search the code for '# Angles'.



Interpreter

The command interpreter converts a series of commands either from a radio control or programmed into the code itself. The commands combine the direction and speed compared to the horizon that the user want the quadcopter to follow. The code converts these commands into a series of targets for vertical speed, horizontal speed and yaw speed – any command from a pair of joysticks can be broken down into a set of these targets.

Inputs

The inputs to the quadcopter come from a series of electronic sensors providing information about its movement in the air. The main two are an accelerometer which measures acceleration force (including gravity) in the three axes of the quadcopter, and a gyroscope which measures the angular speed with which the quadcopter is pitching (nose/tail up and down), rolling (left/right side up and down), and yawing (spinning clockwise and anticlockwise around the central axis of the quadcopter itself).

Axes

The accelerometer is relative to the orientation of quadcopter axes, but the command targets are relative to the Earth's axes – the horizon and gravity. To convert the sensor output between the quadcopter axes and the Earth axes needs a little trigonometry

Understanding quadcopters...

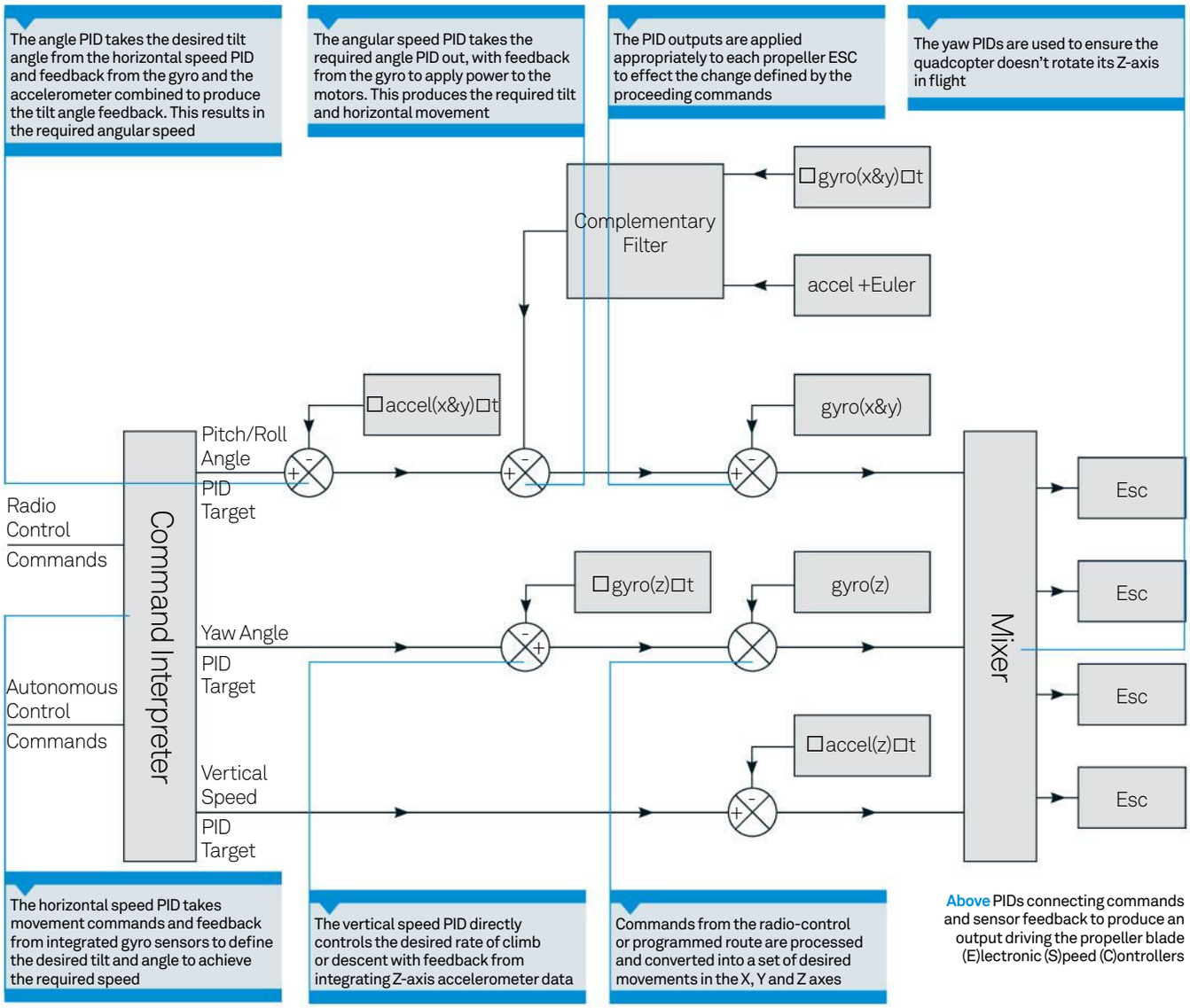
Although this article focuses on software, a very basic background in the hardware from the kit is necessary to provide context.

A quadcopter has four propellers (hence the name) pointing upwards to the sky, each attached to its own brushless DC motor at one of the four corners of (usually) a square frame. Two motors spin clockwise, two anticlockwise, to minimise angular momentum of the quadcopter in flight.

Each motor is driven independently by an electronic speed controller (ESC). The motors themselves have three sets of coils (phases), and the ESCs convert a pulse-width-modulation (PWM) control signal from software/hardware to the three phase high-current output to drive the motors at a speed determined by the control signal.

The power for the ESCs and everything else on the system comes from a Lithium Polymer battery (LiPo) rated at 12V, 3300mA with peak surge current of 100A – herein lies the power!

Above How sensors in the quadcopter point of view are converted to the Earth (horizontal/vertical) viewpoint to provide horizontal motion



Above PIDs connecting commands and sensor feedback to produce an output driving the propeller blade (E)lectronic (S)peed (C)ontrollers

trigonometry and knowledge of the tilt angles in pitch and roll axes of the quadcopter with respect to the Earth.

Angles

Both the accelerometer and gyro can provide this angle information, but both have flaws.

The accelerometer output can be used to calculate the angle by using the Euler algorithm. However, the accelerometer output is plagued by noise from the motors/propellers, meaning a single reading can be hugely inaccurate; on the plus side, the average reading remains accurate over time.

In contrast, the gyro output does not suffer from the noise, but since it is the angular speed being measured, it needs to be integrated over time to find the absolute angle of the quadcopter in comparison to the horizon. Rounding errors in the integration lead to ever increasing errors over time, ultimately curtailing the maximum length of a flight.

Filter

Although independently they are both flawed, they can be merged mathematically such that each compensates for the flaws in the other, resulting in a noise-free, long-term accurate reading. There are many versions of these mathematical noise/drift filters. The best common one is by Kalman; the one we've chosen is slightly less accurate, but easier to understand and therefore to code: the complementary filter.

Now with an accurate angle in hand, it's possible to convert accelerometer sensor data to inputs relative to the Earth's axes and work out how fast the quadcopter is moving up, down, left, right and forwards and backwards compared to the targets that have been set.

PIDs

So we now have a target for what we want the quadcopter to do, and an input for what it's doing, and some motors to close the gap between the two; all we need now is a way to join these together. A direct mathematical algorithm is nigh on impossible – accurate weight of the quadcopter, power per rotation of each blade, weight imbalance etc would need to be incorporated into the equation. And yet none of these factors is stable: during flights (and crashes!), blades get damaged, batteries move in the frame, grass/mud/moisture changes the weight of the 'copter, humidity and altitude would need to accounted for. Hopefully it's clear this approach simply won't fly.

Both the accelerometer and gyro can provide the angle information, but both have flaws

Instead, an estimation method is used with feedback from the sensors to fine-tune that estimate. Because the estimation/feedback code loop spins at over 100 times a second, this approach can react to 'errors' very quickly indeed, and yet it knows nothing about all the factors which it is compensating for – that's all handled blindly by the feedback; this is the PID algorithm.

It takes the target, subtracts the feedback input, resulting in the error. The error is then processed via a Proportional, Integral and a Differential algorithm to produce the output.

Blender

The outputs are applied to each ESC in turn: the vertical speed output is applied equally to all blades; the pitch rate output is split 50/50 subtracting from the front blades and adding to the back, producing the pitch. Roll is handled similarly. Yaw too is handled in a similar way, but applied to diagonal blades which spin in the same direction.

These ESC-specific outputs are then converted to a PWM signal to feed to the hardware ESCs with the updated propeller/motor speeds.

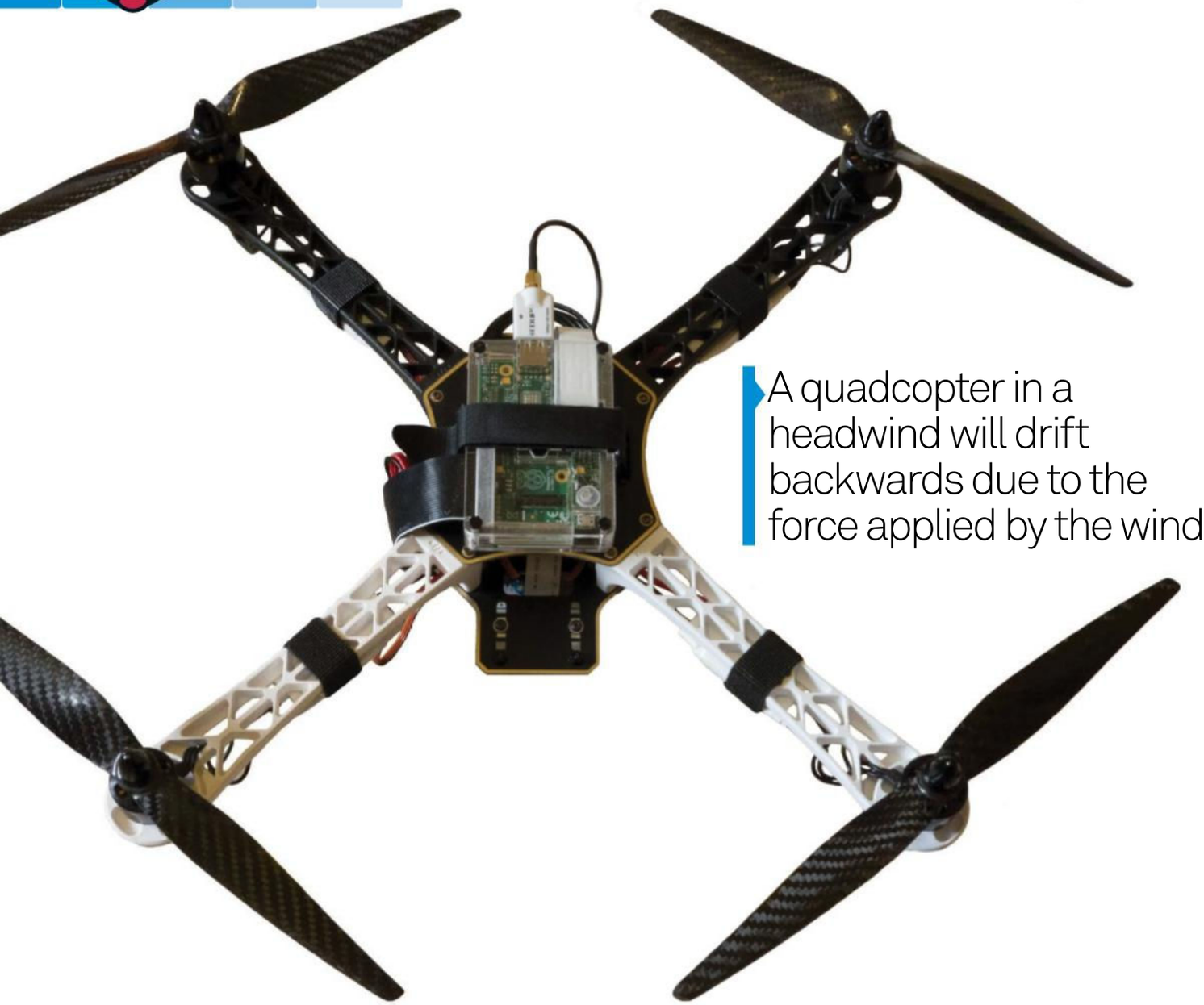
Code and reality

In this code, there are nine PIDs in total. In the horizontal plane, for both the X and Y axes, the horizontal speed PID converts the user-defined desired speed to required horizontal acceleration/angle of tilt; the angles PID then converts this desired tilt angle to desired tilt rate which the rotation speed PID converts to changes in motors speeds fed to the front/back or left/right motors for pitch/roll respectively

In the vertical direction, a single PID converts the desired rate of ascent/descent to the acceleration output applied to each plate equally.

Finally, prevention of yaw (like a spinning top) uses two PIDs – one to set the desired angle of yaw, set to 0, and one to set the yaw rotation speed. The output of these is fed to the diagonally opposing motors which spin in the same direction.

The most critical of the nine are pitch/roll/yaw stability. These ensure that whatever other requirements enforced by other PIDs and external factors, the quadcopter is stable in achieving those other targets; without this stability, the rest of the PIDs cannot work. Pitch is controlled by relative speed differences between the front and back propellers; roll by left and right differences,yaw by clockwise/anticlockwise differences



A quadcopter in a headwind will drift backwards due to the force applied by the wind

differences from the corresponding PIDs' outputs. The net outputs of all three PIDs are then applied to the appropriate combination of motors' PWM channels to set the individual pulse widths.

With stability assured, some level of take-off, hover and landing can be achieved using the vertical speed PID. Placing the quadcopter on a horizontal surface, set the target to 0.5 m/s and off she zooms into the air, while the stability PID ensures that the horizontal attitude on take-off is maintained throughout the short flight, hover and landing.

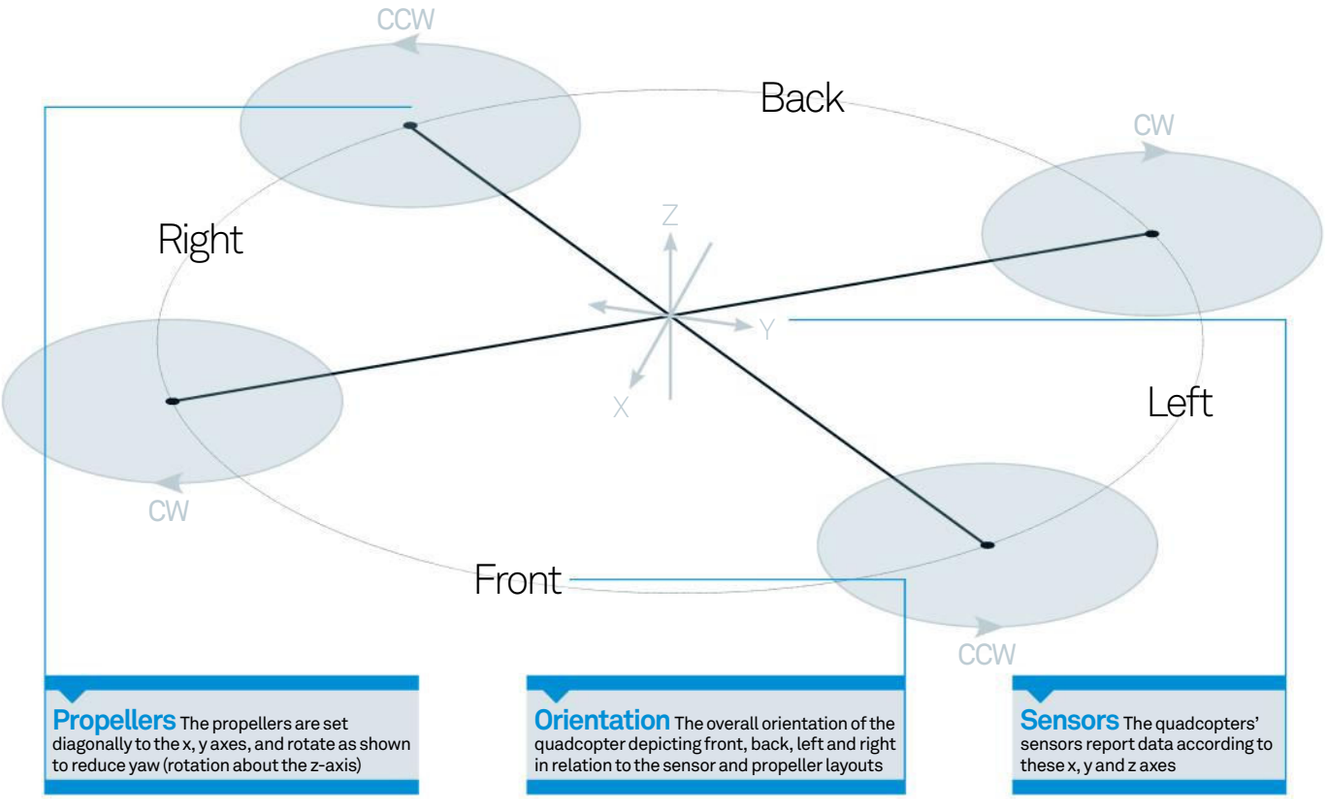
Up to this stage, the PIDs are independent. But what about for horizontal movement target, and suppression of drifting in the wind?

Taking the drift suppression first, a quadcopter in a headwind will drift backwards due to the force applied by the wind. To compensate, it must tilt nose down at some angle so that some of the propellers' thrust is applied horizontally to counteract the wind. In doing so, some of the power keeping the 'copter hovering at a fixed height is now battling the wind; unless the overall power is increased, the 'copter will start descending.

Horizontal movement is more complex still. The target is to move forwards at say 1 metre per second. Initially the requirement is similar to the headwind compensation – nose down plus increased power will apply a forward force leading to forward acceleration. But once that horizontal speed is attained, the quadcopter needs to level off to stop the acceleration, but at the same time, friction in the air will slow the movement. So there's a dynamic tilting fore/aft to maintain this stable forward velocity.

Both wind-drift suppression and controlled horizontal movement use nested PIDs; the X and Y axes horizontal speed PIDs' outputs are used as the pitch and roll angle PIDs targets; their output feeds the pitch and roll rate PIDs to ensure stability while meeting those angular targets. The sensor feedback ensures that as the desired horizontal speed is approached, the horizontal speed PID errors shrink, reducing the targets for the angular pitch PID, thus bringing the quadcopters nose back up to horizontal again.

Hopefully it now becomes clearer why accurate angle tracking is critical: in the nose-down, headwind example, the



input to the vertical speed PID from the sensors is reduced by the cosine of the measured angle of 'copter tilt with respect to the horizon.

Similarly, X and Y axis speed PID sensor inputs need compensating by pitch and roll angles when comparing target speeds against accelerometer readings.

Experimentation and tuning

While the code accurately reflects everything we've described here, there's one critical set of steps which can only be found through live testing; these are the PID gains. For each PID running, there is an independent Proportional, Integral and Differential gain that can only be found with estimation/experimentation. The results for every quadcopter will be different. Luckily there is a relatively safe way to proceed.

First, find the PWM take-off speed: this is done by sitting your quadcopter on the ground and slowly increasing the PWM value until she starts looking light-footed – for your expert, this was about the 1590us pulse width (or 1000us + 590us, as shown in the code).

Next, sorting out the stability PIDs – assuming your quadcopter is square and its balance is roughly central, then the result of pitch tuning also applies to yaw tuning. For pitch tuning, disable two diagonally opposed motors and rest these on a surface – the quadcopter sits horizontal in between. Power up the dangling motors' PWM to just under take-off speed (1550us pulse width in our expert's case). Does the quad rock manically, wobble in some pretence of control, self-right when nudged, or do nothing? Tweak the P gain accordingly. Once P gain is good, add a touch of I gain – this will ensure return to 0 as well as stability. D gain is optional, but adds firmness and crisp response. Tapping a D-gain stable quad is like knocking on a table – it doesn't move.

Vertical speed PID can be guesstimated. 1590us is taking off; desired take-off speed is 0.5m/s so a P gain of 100 is okay. No I or D gain needed.

With that a real take-off, hover and landing are safe, which is good as these are the only way to tune the directional PIDs. Just be cautious here – excessive gains lead to quadcopters slamming into walls or performing somersaults in mid-air before powering themselves into the ground. Best executed outside in a large open field/garden/park where the ground is soft after overnight rain!

There isn't a shortcut to this, so just accept there will be crashes and damage and enjoy the carnage as best you can!

Assuming all the above has gone to plan, then you have a quadcopter that takes off, hovers and lands even in breezy conditions. Next step is to add a remote control, but that's for another article...

Above The orientation of the quadcopter compared to the direction of travel, the rotation of the propellers and the axes used in the code

Build your own games console

Simulate a spaceship launch

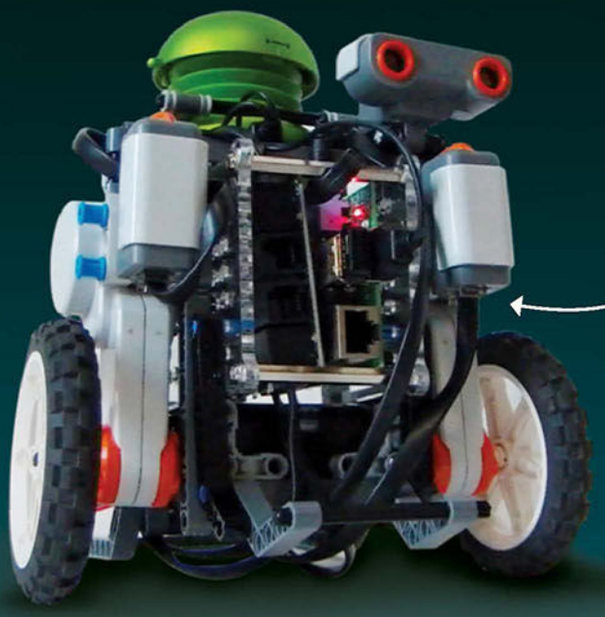
Reinvent a classic camera

Activate Project Jarvis

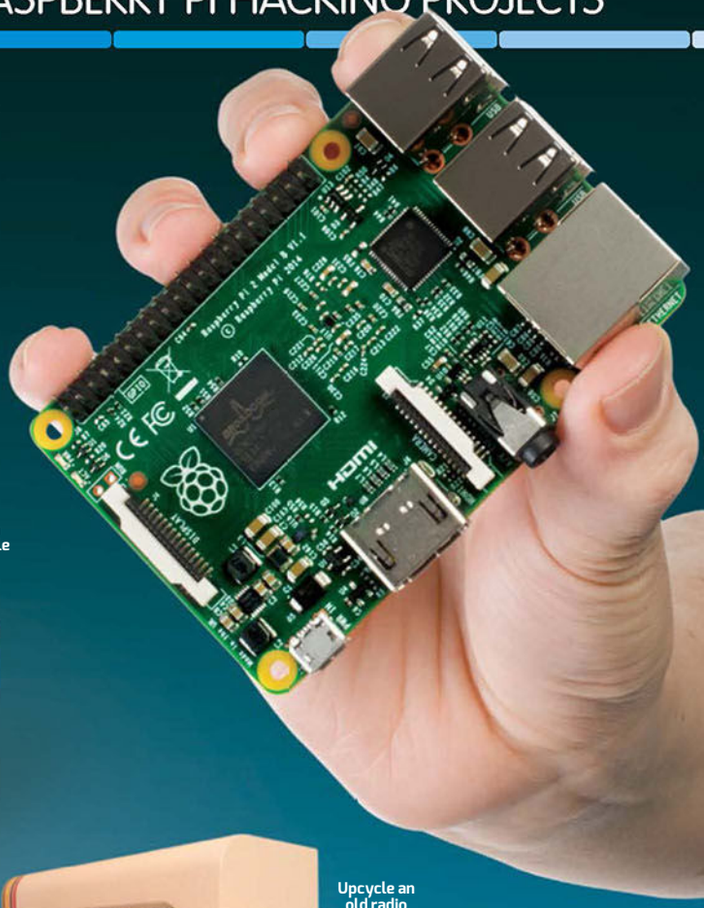
20 Raspberry Pi hacking projects

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/0YdPo**

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

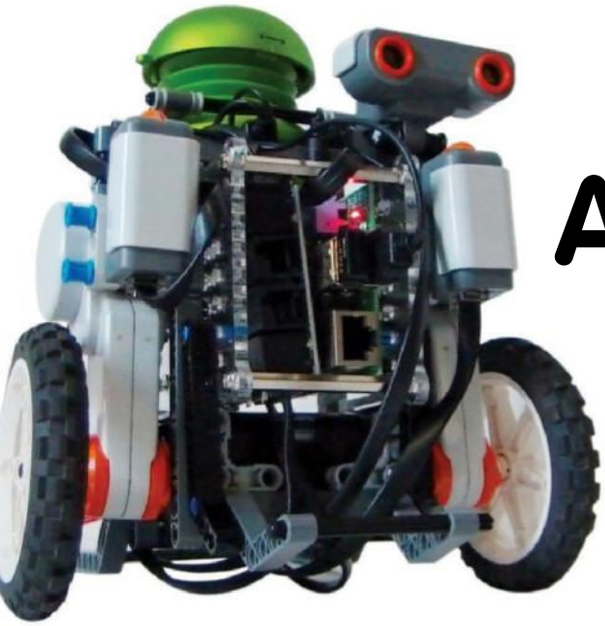
Engineering student

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.

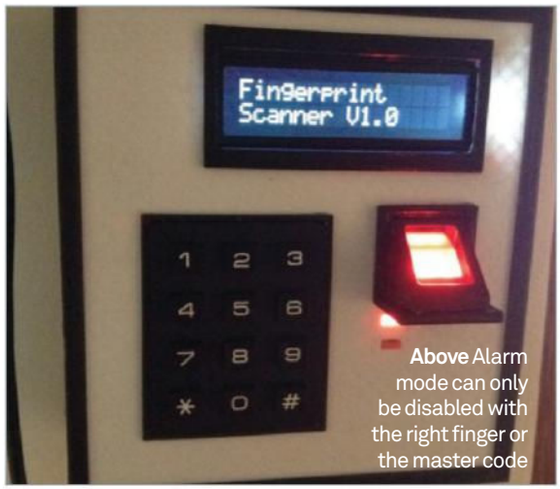
Find out more:
bit.ly/1F0HDkW

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.

About the mag

The magazine for the GNU generation

Written for you

Linux User is the only magazine dedicated to advanced users, developers & IT professionals

In-depth guides & features

Written by grass-roots developers & industry experts

Jam-packed with Ras Pi

A Practical Raspberry Pi section brings you clever tutorials, inspirational interviews and more

subscribers to...

LinuxUser & Developer™

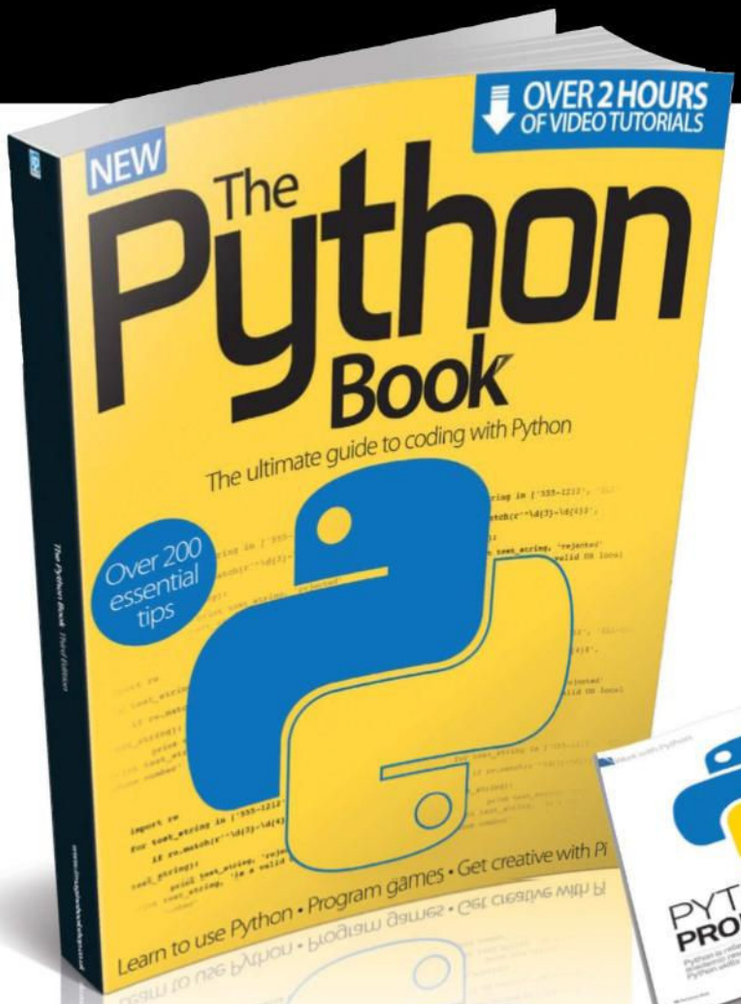
Try three issues for £5 in the UK*
or just $7.85 per issue in the USA**
(saving 54% off the newsstand price)

For amazing offers please visit
www.imaginesubs.co.uk/lud
Quote code ZGGZINE

Or telephone UK 0844 249 0282+ overseas +44 (0) 1795 418 661

* Calls will cost 7p per minute plus your telephone company's access charge

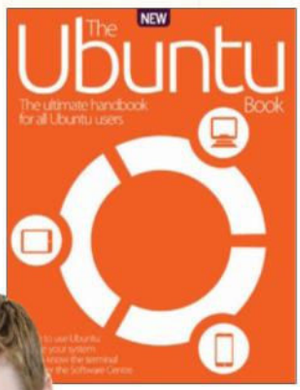
From the makers of LinuxUser & Developer

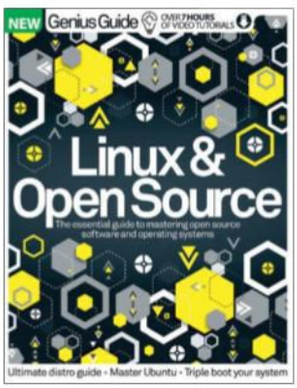


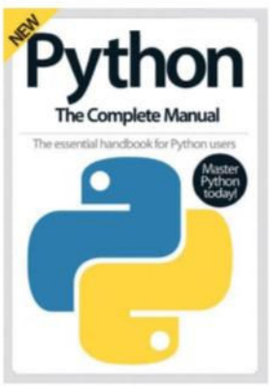
The Python Book

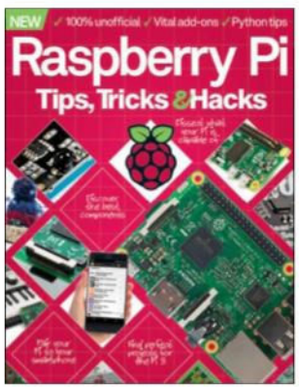
Discover this exciting and versatile programming language with The Python Book. You'll find a complete guide for new programmers, great projects designed to build your knowledge, and tips on how to use Python with the Raspberry Pi – everything you need to master Python.

Also available...









A world of content at your fingertips

Whether you love gaming, history, animals, photography, Photoshop, sci-fi or anything in between, every magazine and bookazine from Imagine Publishing is packed with expert advice and fascinating facts.

BUY YOUR COPY TODAY

Print edition available at www.imagineshop.co.uk

Digital edition available at www.greatdigitalmags.com

Print Tablet Phone Desktop

Practical Raspberry Pi Projects

Experiment with a robot alarm clock

Build your own light gun

Get hands-on with your Raspberry Pi

Build brilliant machines

Create everything from quadcopters, to synths, to a games console and more

Program your Pi

Run a huge range of software with a Raspberry Pi and some simple coding

See crazy RasPi creations

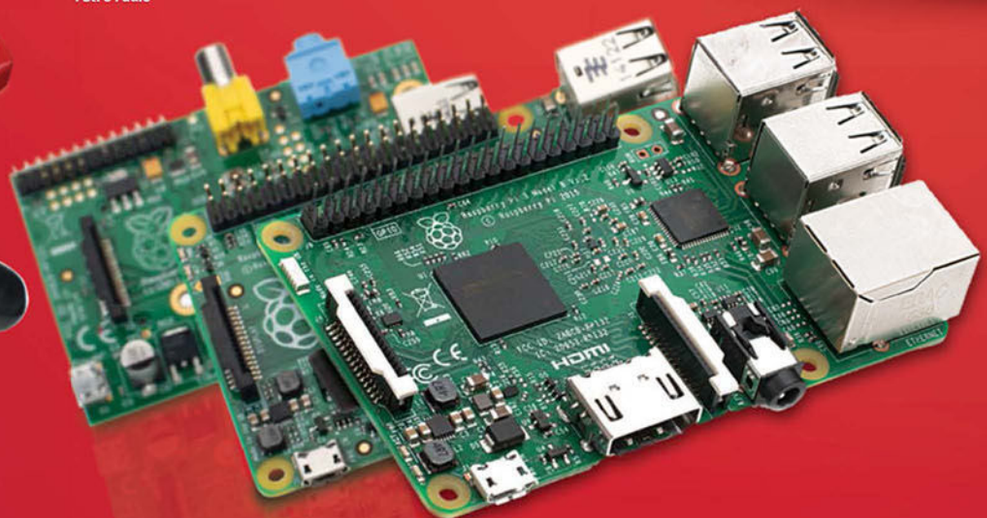
Create a mini Raspberry Pi tank

Modify a retro radio

Customise the circuitry

Tweak the innards of your Raspberry Pi and unlock some amazing features

Turn your RasPi into a telephone



Take arcade games on the go

Augment a telescope