ISSUE 19 - DEC 2013

Get printed copies at themagpi.com

The MagPi

A Magazine for Raspberry Pi Users

Quadcopter Lift Off

Environmental Monitor

XLoBorg Paint Brush

GPIO LCD Screen

Remote Sensing

Pi Store Review

Catch Up TV

SonicPi

Win a Raspberry Pi 16GB SD card breadboard & more

ISSN 2051-9990

The MagPi

http://www.themagpi.com

Welcome to Issue 19 of The MagPi magazine.

This year has flown by and we are back with our Christmas issue once again! Is Santa bringing you any Raspberry Pi goodies this Christmas? If so we have plenty of great articles to help you put this clever computer straight to good use.

Are you bored of having your presents delivered by reindeer? If you fancy a change, why not have your own Pi-powered quadcopter air drop them in? Andy Baker begins his series on building this flying machine covering in this issue the parts required, their function and discusses some of the coding used for lift off.... no, it's not powered by Christmas spirit!

If you are too busy attending Christmas parties and are missing your favourite soaps, never fear, we have you covered. Geoff has produced a great article on OpenELEC, bringing you catch up TV on your Raspberry Pi so you never have to miss an episode again! Claire Price continues the festive spirit with a fantastic article on Sonic Pi which will have your Raspberry Pi singing along with a rendition of Good King Wenceslas.

The news in the UK is currently filled with stories of electricity firms putting up their prices this winter. If you want to be savvy with your heating and electricity bills, without turning off the tree lights and turning the thermostat down, why not cast your eye over Pierre's article on environmental monitoring. Alternatively, to warm you up we return to Project Curacao looking at the environmental subsystem used in this remote sensing project.

If that's not enough to keep you busy over the holidays, why not paint an electronic masterpiece with XLoBorg? Andy Wilson looks at scrolling an RSS feed on an LCD via GPIO plus we pay a visit to the Pi Store. Finally, find out about events in your area and PC Supplies are yet again generously offering Raspberry Pi goodies in their monthly competition.

We are also pleased to be able to announce that printed copies of the magazine are now available from various retailers listed at www.themagpi.com.

The MagPi will be taking a short break over Christmas and the first issue of 2014 will be published at the start of February.

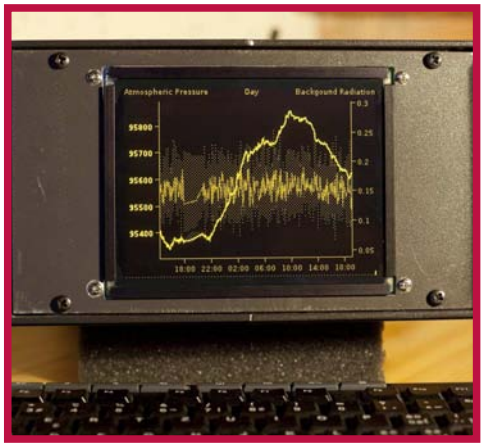
Merry Christmas and best wishes for 2014.

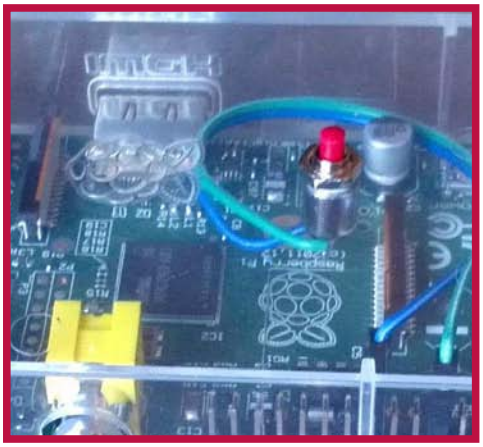
Chief Editor of The MagPi

The MagPi Team

Ash Stone - Chief Editor / Administration / Layout
W.H. Bell - Issue Editor / Layout / Administration
Bryan Butler - Page Design / Graphics
Ian McAlpine - Layout / Proof Reading
Matt Judge - Website / Administration
Aaron Shaw - Layout / Proof Reading
Colin Deady - Layout / Proof Reading
Claire Price - Layout / Proof Reading
Matt Weaver - Layout
Amy-Clare Martin - Layout
Gerry Fillery - Proof Reading / Testing
Paul Carpenter - Tester
Neil Matthews - Tester / Proof Reading
Nigel Curtis - Proof Reading

Contents





Building an airborne autobot

Andy Baker

Guest Writer

SKILL LEVEL : ADVANCED

A quadcopter is a flying machine with four propellers controlled either autonomously (programmed with a fixed flight routine) or via a remote control.

This first article (hopefully of a series) covers a brief overview of how they work, how to build one controlled by a Raspberry Pi, information about where to get all the bits you need and how to bolt them all together physically, electronically and in software. The result should be a quadcopter which can take-off, hover and land autonomously (and with care!)

Future articles will cover more details on testing and tuning this basic quad including code enhancements to allow lateral movement, a Raspberry Pi remote control, and perhaps future developments covering GPS tracking.

Parts of a quadcopter

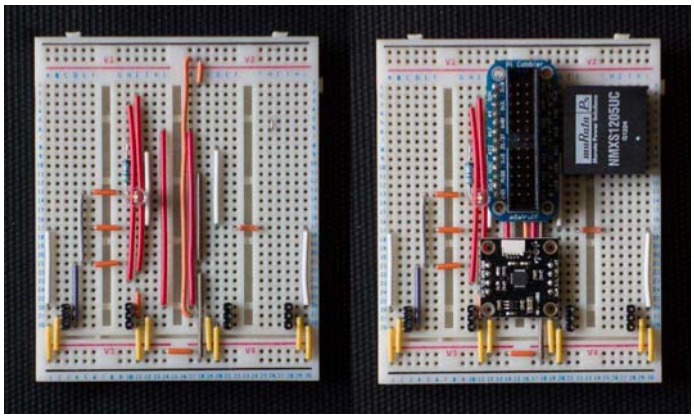
First a quick breakdown of all the parts that make up a quadcopter.

There are four propeller blades. Two of the four are designed to rotate clock-wise; the other two anti-clockwise. Blades which are designed to move the same way are placed diagonally opposite on the frame. Organising the blades like this helps stop the quadcopter spinning in the air. By applying different power to each propeller, and hence different amounts of lift to corners of the quadcopter, it is possible to not only get a quadcopter to take-off, hover and land but also by tilting it, move horizontally and turn corners.

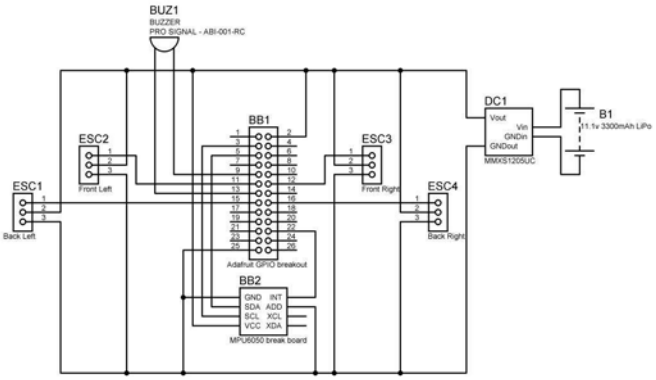
Each propeller has its own DC brushless motor. These motors can be wired to rotate clockwise or anti-clockwise to match the propeller connected to them. The motor has coils in three groups around the body (called the stator) and groups of magnets attached to the propellor shaft (called the rotor). To move the blades, power is applied to one group of the coils and the rotor magnets are attracted to that coil, moving round. If that coil is then turned off and the next one powered up, the rotor moves around to the next coil. Repeating this around the three coils in sequence results in the motor rototating; the faster you swap between the three powered coils the faster the motor rotates. This makes the motor suitable for 'digital' control – the direction and speed of movement of the propeller blade exactly matches the sequence and rate power pulses are applied to the coils. These motors take a lot of power to spin the propeller blades fast enough to force enough air down to make the quadcopter take-off – far more power than a Raspberry Pi can provide – so an Electronic Speed Controller (ESC) bridges that gap. It translates between a Pulse Width Modulation (PWM) control signal from the Raspberry Pi and converts it to three high-current signals, one for each

coil of the motors. They are the small white objects velcro'd under the arms of the quadcopter.

Next there are sensors attached to the breadboard on the shelf below the Raspberry Pi; these provide information to the Raspberry Pi about rocking and rolling in three dimensions from a gyroscope, plus information about acceleration forward, backwards, left, right, up and down. The sensors connect to the Raspberry Pi GPIO I^2C pins.



In the circuit diagram you can see I am considering adding a beeper, so I can hear what the quadcopter thinks it's doing.



The power for everything comes from a single lithium polymer (LiPo) battery which provides 11.1V up to a peak current of 100A, with the full-charge of 3300 mAh thus supplying 3.3A for an hour or 100A for two minutes or anywhere in between. This is a powerful and dangerous beast, yet it only weighs 250 grams. It requires a special charger – if not used, a LiPo battery can easily become a LiPo bomb – beware. There is a regulator on the breadboard to take power from the battery and produce the 5V for the Raspberry Pi and also provide a degree of protection from the vast power surges the motors draw from the battery.

That just leaves the beating heart of the quadcopter itself; the Raspberry Pi. Using Python code it reads the sensors, compares them to a desired action (for example take-off, hover, land) set either in code or from a remote control, converts the difference between what the quad is doing (from the sensors) and what it should be doing (from the code or remote control) and changes the power to each of the motors individually so that the desired action and sensor outputs match.

Creating your quadcopter

First and foremost, flying machines and boats are female and they have names; mine is called Phoebe ("Fee-Bee"). Choose a name for yours and treat her well, and the chances are she'll reciprocate!
Phoebe's body, arms, legs, blades, motors, ESCs and batteries are from kits. Total cost is about £250 – together with a Raspberry Pi, and other accoutrements, the total cost is perhaps £300 – £350. Not cheap for something which certainly at the start has a preference to crash rather than fly!

A complete bill of materials (BOM) is available at http://blog.pistuffing.co.uk/?p=1143

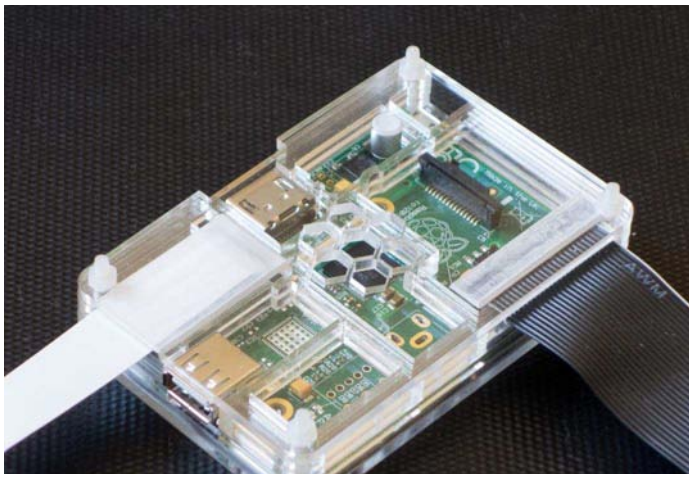
I've actually upgraded my motors to higher power, lighter weight varieties but this is absolutely not necessary – the equipment provided by the kits is excellent. Upgrading components for weight reduction / power efficiency and strength is definitely an afterthought once the basics are working.

The Raspberry Pi is a model A, chosen for lower weight and lower power consumption; these are factors reflected through other pieces of the design. I have removed the audio and video outputs and use a micro-SD card adapter from the guys at Pimoroni – all in the name of weight saving.

The Raspberry Pi case is a variant of the Pimoroni PiBow model B case with a couple of levels removed and some holes sealed for reduced weight and increased strength (protection from crashes!). I've posted the design for these at http://blog.pistuffing.co.uk/wp-content/uploads/2013/10/Pibow002-AB5.pdf. Phenoptix do a great job of laser cutting 3mm acrylic at a very reasonable price.

Talking to Phoebe

Whether Phoebe is autonomous or remote controlled someone needs to talk to her to tell her what to do. To that end, Phoebe runs a wireless access point (WAP) so another computer can join her private network and either SSH in or provide remote control commands. You can see how I did this at http://blog.pistuffing.co.uk/?p=594.



For initial testing the WAP function isn't necessary, any household wireless network will do, but as your quadcopter comes to life you will want to be doing your testing away from animals, children and other valuables you don't want damaged (like yourself). Having a WAP means you can take the testing out into the garden or local park or field.

Presenting Phoebe's Python code

The final step is obviously the most important; once you have a physical quadcopter with associated blades, motors, ESCs, power and circuitry, we use Python code to glue all the pieces together. I'm not going to go into this blow by blow here as the code is available at https://github.com/PiStuffing/Quadcopter and it should be self-documenting. There are more lines of explanatory comments than there are lines of code actually doing something constructive!

The I^2C class provides a programming interface to read and write data from the sensors. Built on that, the MPU6050 class configures the sensors and then provides API access to reading the data and converting the values from the sensor into meaningful values humans would understand (like degrees/sec for rotation or metres/sec^2 for acceleration).

The QUADBLADE class handles the PWM for each blade handling initialization and setting the PWM data to control the propeller blade spin speeds. The PID class is the jigsaw glue and the core of the development and testing. It is the understanding of this which makes configuring a quadcopter both exciting and scary! It is worth an article in its own right – for now there is a brief overview of what they do and how at the end.

There are utility functions for processing the startup command line parameters, signal handling (the panic button Ctrl-C) and some shutdown code.

Last, but not least, there is the big "while keep_looping:" loop which checks on what it should be doing (take-off, hover, land, etc), reads the sensors, runs the PIDs, updates the PWMs and returns to the start one hundred times a second!

PID

The PID (Proportional, Integral, Differential) class is a relatively small, simple piece of code used to achieve quite a complex task. It is fed a "target" value and an "input" value. The difference between these is the "error". The PID processes this "error" and produces an "output" which aims to shrink the difference between the "target" and "input" to zero. It does this repeatedly, constantly updating the "output", yet without any idea of what "input", "output" or "target" actually mean in its real world context as the core of a quadcopter: weight, gravity, wind strength, RC commands, location, momentum, speed and all the other factors which are critical to quadcopters.

In the context of a quadcopter, "target" is a flight command (hover, take-off, land, move forwards), "input" is sensor data and "output" is the PWM pulse size for the motors.

Phoebe has 4 PIDs running currently – pitch, roll, yaw and vertical speed – these are the bare minimum needed for an orderly takeoff, hover and landing.

The PID's magic is that it does not contain any complex calculations connecting power, weight, blade spin rates, gravity, wind-speed, imbalanced frame, poor center of gravity or the many other

factors that perturb the perfect flight modelled by a pure mathematical equation. Instead it does this by repeated, rapid re-estimation of what the current best guess “output” must be based only on the “target” and the “input”.

The “P” of PID stands for proportional – each time the PID is called its “output” is just some factor times the “error” – in a quadcopter context, this corrects immediate problems and is the direct approach to keeping the absolute “error” to zero.

The “I” of PID stands for integral – each time the PID is called the “error” is added to a grand total of errors to produce an output with the intent that over time, the total “error” remains at zero – in a quadcopter context, this aims to produce long term stability by dealing with problems like imbalance in the physical frame, motor and blade power plus wind.

The “D” of PID stands for differential – each time the PID is called the difference in error since last time is used to generate the output – if the “error” is worse than last time, the PID “D” output is higher. This aims to produce a predictive approach to error correction.

The results of all three are added together to give an overall output and then, depending on the purpose of the PID, applied to each of the blades appropriately.

It sounds like magic... and to some extent it is! Every PID has three configurable gain factors configured for it, one each for “P”, “I” and “D”. So in my case I have twelve different gain factors. These are magic numbers, which if too small do nothing, if too large cause chaos and if applied wrongly cause catastrophe. My next article will cover this in much more detail, both how they work and how to tune the gains. In the meantime, use the bill of materials on page 5 and get on with building your own quadcopter. The PID gains in the code I’ve supplied should be a reasonable starting point for yours.

Flying Phoebe

At the moment it is simple but dangerous! Put Phoebe on the ground, place a flat surface across her propeller tips, put a spirit level on that surface and make sure she’s on absolute horizontal by putting padding under her feet – this is absolutely critical if you don’t want her to drift in flight – we’ll fix this in another article with some more PIDs.

Connect the LiPo battery. The ESCs will start beeping loudly – ignore them.

Wait until the Wifi dongle starts to flash – that means Phoebe’s WAP is working.

Connect via SSH / rlogin from a client such as another Raspberry Pi, iPad etc. which you have joined to Phoebe’s network.

Use the `cd` command to change to the directory where you placed Phoebe’s code. Then enter:

```
sudo python ./phoebe.py -c
sudo python ./phoebe.py -c -t 550 -v
```

`-c` calibrates the sensors to the flat surface she’s on
`-t 550` sets up the blades to just under take-off speed
`-v` runs the video camera while she’s in flight.

There are other options. You can find them by entering:

```
sudo python ./phoebe.py
```

Enjoy, but be careful.

Raspberry Pi® Starter Kits

Everything but the screen

http://shop.pimoroni.com

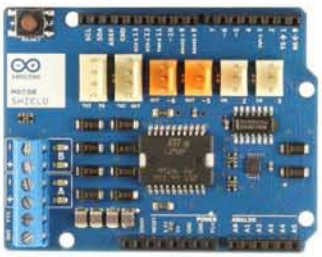
Starter Kit £75
Deluxe Starter Kit £120

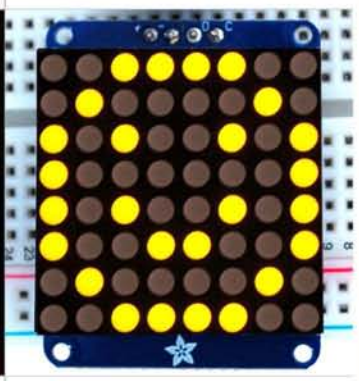
Pimoroni Gift Cards

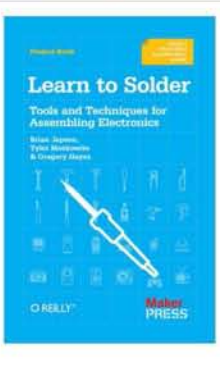
Choosing shiny things is hard!

http://shop.pimoroni.com

PIMORONI







Wearables! Arduinos! Components! Dead Trees!

We now stock a range of shiny things you can make cool stuff with!

http://shop.pimoroni.com

PIMORONI

INTRODUCING.....

THE PI BOT

From a galaxy not too far away comes the Pi Bot
On a mission to entertain earthlings in the way of the Pi

</> PROGRAM AND CONFIGURE YOUR OWN PERSONAL ROBOT

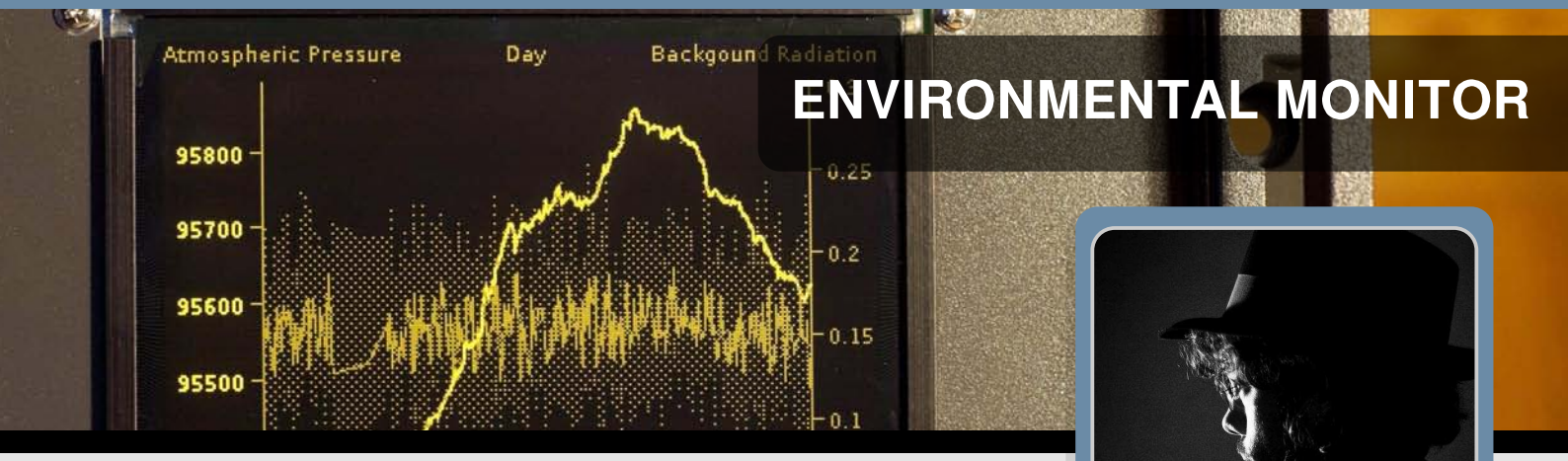
MADE FOR MAKERS

OPEN SOURCE

A UK ENTERPRISE

Find out more online @ WWW.PIBOT.ORG

FOLLOW #THEPIBOT



Using the BMP085 to log temperature and pressure data

Pierre Freyermuth

Guest Writer

SKILL LEVEL : INTERMEDIATE

Data logging systems are often used in industry to allow analysis after data has been read and for monitoring. Studying collected data, which could be a large data sample, leads to a better understanding of the monitored system behaviour. The Raspberry Pi provides an affordable, accessible and low power consumption solution that can log, present and upload data. For example, a Raspberry Pi could be used to monitor the energy consumption of a heating system in a house.

In this article series I will present an independent data logging system that is assembled from several pieces. The components of the system could be modified for other applications. This series uses the Raspbian Raspberry Pi operating system, together with the Java programming language.

Base station requirements

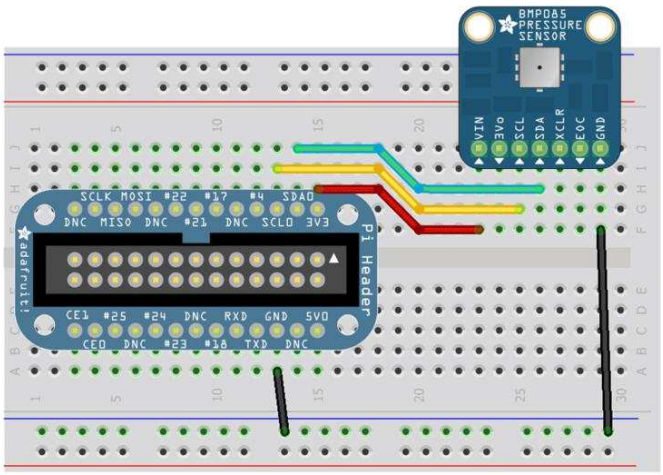
The goal for the project is to run day and night without interruption. Therefore, particular attention needs to be taken when choosing the power supply. It should be efficient and reliable. If only a few low power consumption sensors are connected, a mobile phone charger is probably enough. It is up to you to decide if a screen, keyboard, mouse or remote access is needed to interact with the Raspberry Pi. I chose to record all of the data onto the SD card. However, the data could be written to a USB hard drive connected through a powered USB hub, or to a network mounted file system, or buffered locally and then uploaded. Writing all of the data allows a complete analysis afterwards and reduces the write access to the storage media in comparison to a rotating buffer. Recording sensor data for one year could represent hundreds of megabytes of data. If a log of data needs to be written then either a hard disk drive or network mounted file system may be needed.

Raspbian is a really convenient operating system. The default image now includes the Oracle Java virtual machine, which is very efficient. The default Raspbian image also includes system drivers for the buses (I^2C, SPI, UART) available via the GPIO pins. To install Raspbian on a SD card, follow the official tutorials at http://www.raspberrypi.org/downloads. The buses available on the Raspberry Pi GPIO can be controlled using the Pi4J Java library. This library is discussed later in this article. For those unfamiliar with Java, the Cup of Java MagPi series provides a basic introduction in Issues 14 and 17.

Connecting the BMP085

The BMP085 is a very precise and affordable solution to measure temperature and air pressure. It can be ordered from http://www.adafruit.com or from one of their distributors. It usually comes mounted on a PCB board that can be connected directly to the

Raspberry Pi I²C bus.



Before the BMP085 can be used the i2c kernel module needs to be enabled. From the command line, enter:

```
cd /etc
sudo nano modprobe.d/raspi-blacklist.conf
```

Look for the entry `blacklist i2c-bcm2708` and add a hash '#' at the beginning of the line so it becomes `#blacklist i2c-bcm2708`. Press <Ctrl>+<X> then press <Y> and <Enter> to save and exit.

Next edit the modules file. From the command line, enter:

```
sudo nano modules
```

Add `i2c-dev` on a new line. Press <Ctrl>+<X> then press <Y> and <Enter> to save and exit.

Reboot the Raspberry Pi, open a terminal window and enter:

```
sudo i2cdetect -y 1
```

NOTE: Use 0 instead of 1 for the bus number in the above command if you have a revision 1 Raspberry Pi. The revision 1 Pi does not have any mounting holes on the PCB; newer revisions have 2 mounting holes.

The response should show 77 in the address list. More information is provided in the Adafruit tutorial at http://goo.gl/PDrZGL.

Java data logging program

Oracle has provided a Java virtual machine that has been optimised for the Raspberry Pi. Check your Java version by typing:

```
java -version
```

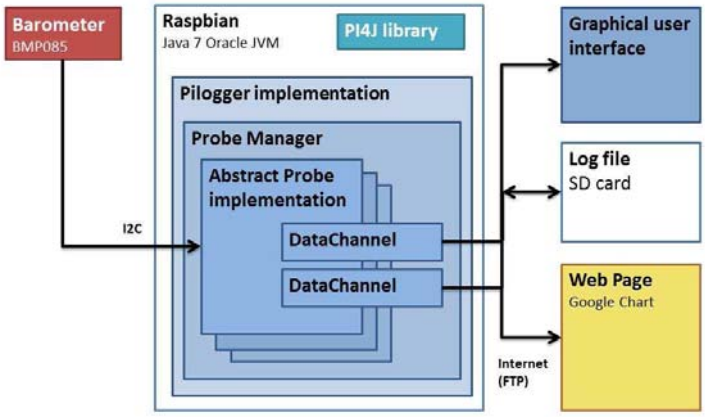
The response should be 1.7.0_40 or higher.

To get the source code needed for this project follow the instructions given at http://goo.gl/KqPH24.
to check out . The source code must be customised according the sensors connected and the online server used. Details of the Pi4J Java library are given at http://pi4j.com.

The Pi4J library provides direct access to the SPI bus and a lot of GPIO facilities.

Goals of the program

• At start-up, configure the communications for the probes and then the probes themselves.



• At start-up, load the previous recorded data from log files.
• Retrieve data from probes and record them to a plain text file on the SD card.
• Perform averaging with different time scales, in order to present historical data in a convenient way.
• Upload averaged data to a web server using FTP to provide access via the internet.
• Use a chart as a graphical user interface, to make local visualisation easy.

By recording all of the data, it is then possible to change the analysis method or analyse the full data.

Structure

Probes connected to the Raspberry Pi can provide several values that may be independent. The chosen solution consists of an abstract class for probes which must be implemented for each sensor connected to the system. A probe implementation of this class must provide one or several DataChannel objects, which represent the data model. In terms of a model-view-controller pattern, the ProbeManager class can be seen as the controller, customising the view.

The DataChannel class includes activities that are common for each type of data - it loads previous data from the log file when it is instantiated, logs new data and performs averaging. The different implementations of AbstractProbe include functionality specific to a given type of sensor. Each derived class should configure the sensor, process or convert data and then add them to the right DataChannel object.

Interfacing with the BMP085

The class BMP085probe extends AbstractProbe and provides access to DataChannel objects. There are two objects: one for temperature and a second one for air pressure. The two arguments of the DataChannel constructor are the name for the display and the name of the data file.

```
private DataChannel pressureChannel = new DataChannel("Atmospheric Pressure",
    "Atmospheric_Pressure");
private DataChannel temperatureChannel = new DataChannel("Room Temperature", "Room_Temperature");
```

The BMP085probe class overides the abstract method that provides access to the DataChannel objects :

```
@Override
public DataChannel[] getChannels() {
  return new DataChannel[]{pressureChannel, temperatureChannel};
}
```

The I2CBus object from the Pi4J library is passed to the BMP085 constructor, since it can be several peripherals on the same I²C bus. With this bus, we can configure our I²C device. The BMP085 has the address 0×77.

```
public static final int BMP085_I2C_ADDR = 0x77;
private I2CDevice bmp085device;
public BMP085probe(I2CBus bus) throws IOException {
  bmp085device = bus.getDevice(BMP085_I2C_ADDR);
  readCalibrationData();
  DataReaderThread dataReaderThread = new DataReaderThread();
  dataReaderThread.start();
}
```

The dataReaderThread object is used to send a request for temperature and pressure information. Then the thread reads two bytes of raw temperature information and three bytes of raw pressure information.

```
bmp085device.write(CONTROL, READTEMPCMD);            //Send read temperature command
sleep(50);                                           //wait the convertion time
rawTemperature = readU16(DATA_REG);                  //retrieve the 2 bytes
bmp085device.write(CONTROL, (byte) READPRESSURECMD); //Send read pressure command
sleep(50);                                           //wait the convertion time
msb = readU8(DATA_REG);                              //retrieve the 3 bytes
```

```
lsb = readU8(DATA_REG+1);
xlsb = readU8(DATA_REG+2);
rawPressure = ((msb << 16) + (lsb << 8) + xlsb) >> (8-OSS); //make raw pressure integer
```

This raw data can be converted into units of Pascals and Degrees by following the BMP085 datasheet at http://goo.gl/CborFs and using the calibration values previously read. Java does not support native unsigned interger types. It is therefore more convenient to replace bit shifting with division and multiplication.

```
double temperature = 0.0;
double pressure = 0.0;
double x1 = ((rawTemperature - cal_AC6) * cal_AC5) / 32768;
double x2 = (cal_MC *2048) / (x1 + cal_MD);
double b5 = x1 + x2;
temperature = ((b5 + 8) / 16) / 10.0;

double b6 = b5 - 4000;
x1 = (cal_B2 * (b6 * b6 / 4096)) / 2048;
x2 = cal_AC2 * b6 / 2048;
double x3 = x1 + x2;
double b3 = (((cal_AC1 * 4 + x3) * Math.pow(2, OSS) )+2) / 4;
x1 = cal_AC3 * b6 / 8192;
x2 = (cal_B1 * (b6 * b6 / 4096)) / 65536;
x3 = ((x1 + x2) + 2) / 4;
double b4 = cal_AC4 * (x3 + 32768) / 32768;
double b7 = (rawPressure - b3) * (50000 / Math.pow(2, OSS));
if (b7 < 0x80000000) pressure = (b7 * 2) / b4;
else pressure = (b7 / b4) * 2;
x1 = (pressure / 256) * (pressure / 256);
x1 = (x1 * 3038) / 65536;
x2 = (-7375 * pressure) / 65536;
pressure = pressure + (x1 + x2 + 3791) / 16;
```

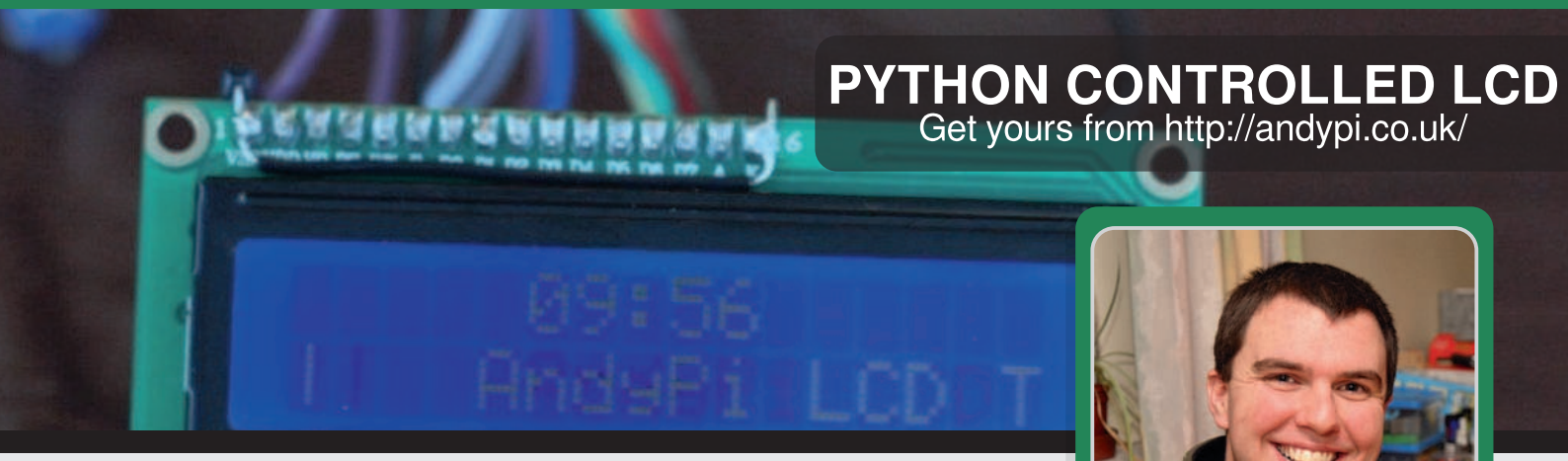
This sensor can provide a high rate of data. To enhance the precision, a thread can take 5 data points, average them and add this new measurement every second to the dataChannel object.

To add the BMP085 to the final program, in the main class of the program we get an instance of the Pi4J I2CBus, instantiate the BMP085probe and add it to the ProbeManager.

```
private void initI2CandBMP085probe() {
  try {
    final I2CBus bus = I2CFactory.getInstance(I2CBus.BUS_1); //Change to BUS_0 for Rev 1 boards.
    bmp085Probe = new BMP085probe(bus);
    probeManager.addProbe(bmp085Probe);
  } catch (IOException e) {
    e.printStackTrace();
  }
}
```

Now the two DataChannel temperature and air pressure values can be acquired, logged, and displayed as a chart.

In the next article, additional instructions will be given to export the data and setup a web page to view the data. Until then, try putting the code together and try it out with the hardware configuration described.



Scrolling an RSS feed on an AndyPi HD44780 LCD via GPIO

Andrew Wilson

AndyPi

SKILL LEVEL : INTERMEDIATE

For many Raspberry Pi projects, providing visual output is important, but a standard HDMI screen is either too large or unnecessary. For example, you may have a temperature sensor and only need to display the current value, or you may want to display an internet radio station name of a "headless" Raspberry Pi. Alternatively, you could have a standalone display for your latest tweets.

This tutorial explains how you can connect an inexpensive HD44780 type 16 character 2 line LCD (Liquid Crystal Display) to your Raspberry Pi's GPIO port and display the time and the latest news headline from the BBC RSS feed.

The AndyPi website (http://andypi.co.uk/) also contains video and written tutorials on how you can use this LCD to display media information such as mp3 titles using RaspBMC or RaspyFi.

Hardware set-up

A brief description of pin connections is described here but a full list of parts and detailed construction information is available at http://andypi.co.uk/. You can buy the parts individually and make your own, or you can purchase a complete, pre-soldered kit of parts from AndyPi for £12.99 or + P&P (EU only).

On the LCD, wire a 10K potentiometer (for contrast control) between VSS (1) and VO (3), connect K (16) to RW (5), and connect K (16) to VSS (1). LCD to GPIO connections as follows:

LCD pin	LCD pin name	GPIO (P1) pin
1	VSS	GND
2	VDD	+5v
15	A	18
14	D7	14
13	D6	23
12	D5	24
11	D4	25
6	E	08
4	RS	07

Software set-up

Starting with the latest version of Raspbian, make sure your system is up to date, and install some python related tools:

```
sudo apt-get update
sudo apt-get install -y python-dev \
python-setuptools python-pip
```

We need to install some python modules; wiringpi for GPIO control; feedparser to read the RSS feed, and the processing module to allow us to use threading (explained later):

```
sudo pip install feedparser \
processing wiringpi
```

Finally, download the AndyPi LCD python class:

```
sudo wget http://andypi.co.uk/downloads/
AndyPi_LCD.py
```

Python script

The AndyPi LCD class has a number of functions to enable simple control of the LCD. We'll make use of a few of these. Create a new script (in the same folder as AndyPi_LCD.py) as follows:

```
#!/usr/bin/python

from AndyPi_LCD import AndyPi_LCD
from processing import Process
import time
import feedparser

lcd=AndyPi_LCD()
lcd.lcd_init()
lcd.led(512)
msg = feedparser.parse(
'http://feeds.bbci.co.uk/news/rss.xml?edit
ion=uk').entries[0].title
```

After importing the required modules, we set the variable "lcd" as the AndyPi_LCD() class, initialise the LCD, and set the brightness of the backlight (0=off, 512=full brightness, using PWM). Then we can use the feedparser module to set the string variable "msg" to the first title of the BBC world news feed.

To display text, you can use the AndyPi_LCD function static_text(). Here we display text on line 1 and 2, and clear it after 30 seconds:

```
lcd.static_text(1, “c”, “World News:”)
lcd.static_text(2, “l”, msg)
time.sleep(30)
lcd.cls()
```

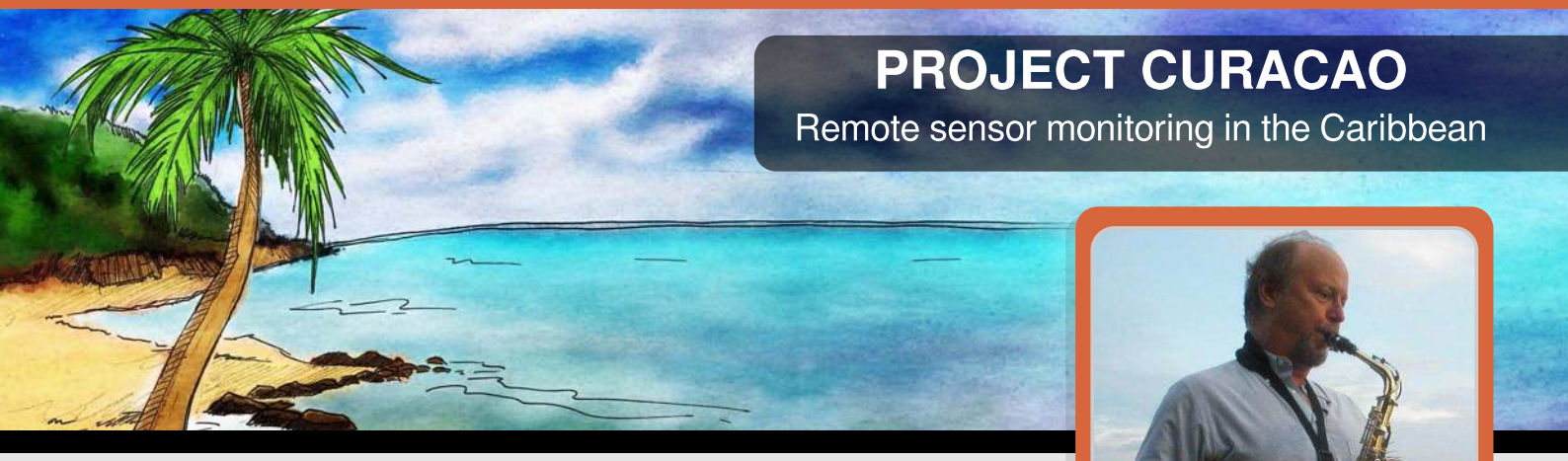
This script is useful for displaying a short static message, but it's not much use for an RSS feed as it only displays the first 16 characters that fit.

Instead we can use the function called scroll_clock, which displays the time on one line, and scrolls the full text along the the second. However, in order to update the LCD by moving one character along at a time, the function loops infinitely - and therefore no code after this is executed. To get around this, we can run this function in a thread (a simultaneously running process) so we can then continue to give further commands (in this case to check for the latest news updates). Here we set up a thread process using scroll_clock, start the thread, wait 60 seconds, update "msg" to the latest RSS feed, and stop the thread. The while loop then repeats to continue scrolling the text. In general, it's bad practice to terminate a thread, but in this case we know that scroll_clock is an infinite loop that will never complete.

```
while True:
  p = Process(target=lcd.scroll_clock,
     args=(1, "c", 0.3, msg))
  p.start()
  time.sleep(60.0)
  msg=feedparser.parse('http://feeds.
bbci.co.uk/news/rss.xml?edition=uk').entri
es[0].title
  p.terminate()
```

The scroll_clock function takes four arguments. Firstly, choose either 1 or 2 to determine which line the clock is placed on. Secondly, choose “l”, “r” or “c” to set the clock alignment. Thirdly, specify the scrolling speed, and the final argument takes any string of characters - here the variable msg (which contains RSS feed text).

RSS feeds of many different topics are widely available on the internet, but there are many other things you could use this display for too - this script is just the start for your own experiments!



PROJECT CURACAO

Remote sensor monitoring in the Caribbean

Part 2: The environmental subsystem

John Shovic

Guest Writer

SKILL LEVEL : INTERMEDIATE

What is Project Curacao?

This is the second in a four part series discussing the design and building of Project Curacao, a sensor filled project that will hang on a radio tower on the island nation of Curacao. Curacao is a desert island 12 degrees north of the equator in the Caribbean.

Project Curacao is designed to monitor the local environment unattended for six months. It operates on solar power cells and will communicate with the designer via an iPad App called RasPiConnect. All aspects of this project are designed to be monitored remotely.

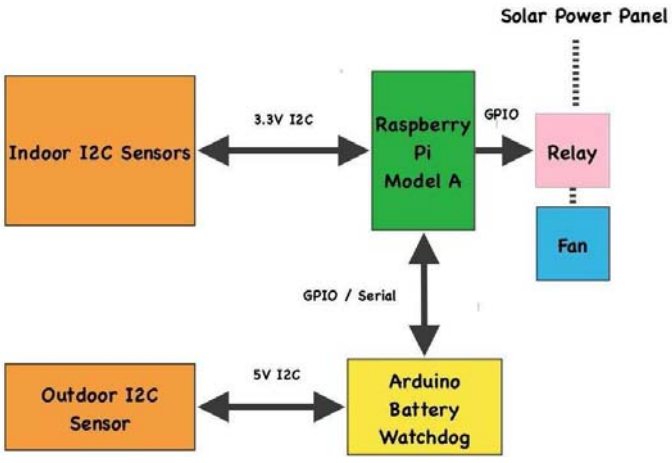
System description

Project Curacao consists of four subsystems. A Raspberry Pi Model A is the brains and the overall controller. The Power Subsystem was described in part 1. In Part 2 we will describe the Environmental Sensor Subsystem.

The Environmental Subsystem

This subsystem consists of an indoor humidity sensor (indoor here refers to inside the box), an indoor temperature and barometric pressure sensor, and a luminosity sensor. An outdoor temperature and humidity sensor is placed on the outside of the bottom of the box. All of these sensors are connected to the Raspberry Pi Model A, with the exception of the AM2315 outdoor temperature and humidity sensor, which is connected to the Arduino based battery watchdog for reasons given on the next page. A small computer fan under Raspberry Pi control is also connected to provide airflow through the box when inbox temperatures get too high or the indoor humidity gets too high.

Environmental Subsystem Block Diagram



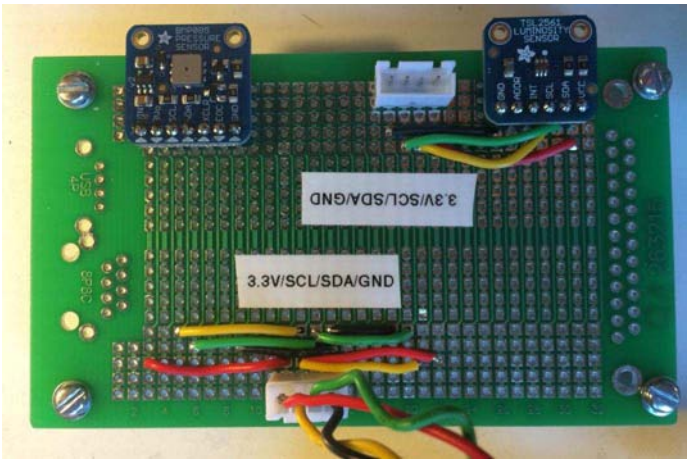
What hardware to use?

We are using the following hardware for the Evironmental subsystem:

1 Adafruit BMP085 Temp/Baro Sensor (I^2C)
1 Adafruit DHT22 Temp/Humidity Sensor
1 AM2315 - Encased Temp/Humidity Sensor
1 Adafruit TSL2561 Digital Luminosity Sensor
1 Akust 5V-12V 8cm Computer Fan
1 Evil Mad Scientist Simple Relay Shield

What to measure?

We want to measure the temperature, humidity and local light levels both inside and outside of the containing box to see what is happening in the local environment. This information will be placed in a MySQL database for later analysis.



Putting in the sensors

The BMP085 and TSL2561 are soldered onto a prototype board with I²C coming in from the Raspberry Pi. There is a plug to further extend the I²C bus to an encased temperature and humidty sensor outside of the box (the AM2315). The AM2315 proved to be problematic (see below), but the other I²C sensors worked like champions. The DHT22 inexpensive indoor temperature and humidity sensor worked well with additional software to filter out bad readings. This caused problems with the RasPiConnect control software because of the unreliable delay to good readings (see below). We control the fan with a relay that connects directly to the output of the solar cells (6.5V on a good day). We figured that the fan would be used on the sunniest days. The fan and relay coil each take 70mA. We are investigating replacing the relay with a lower energized coil current (see Sainsmart: http://goo.gl/aSTU0z) as 140mA really hurts our power budget.

Monitoring the sensors and fan remotely

Project Curacao is monitored remotely through the Internet by the use of SSH and RasPiConnect (www.milocreek.com). Each of the main subsystems has a different display. The environmental display has a graph for temperature/humidity and luminosity/barometric pressure/fan state (shown below). The software for generating this display is also on github.com/projectcuracao.

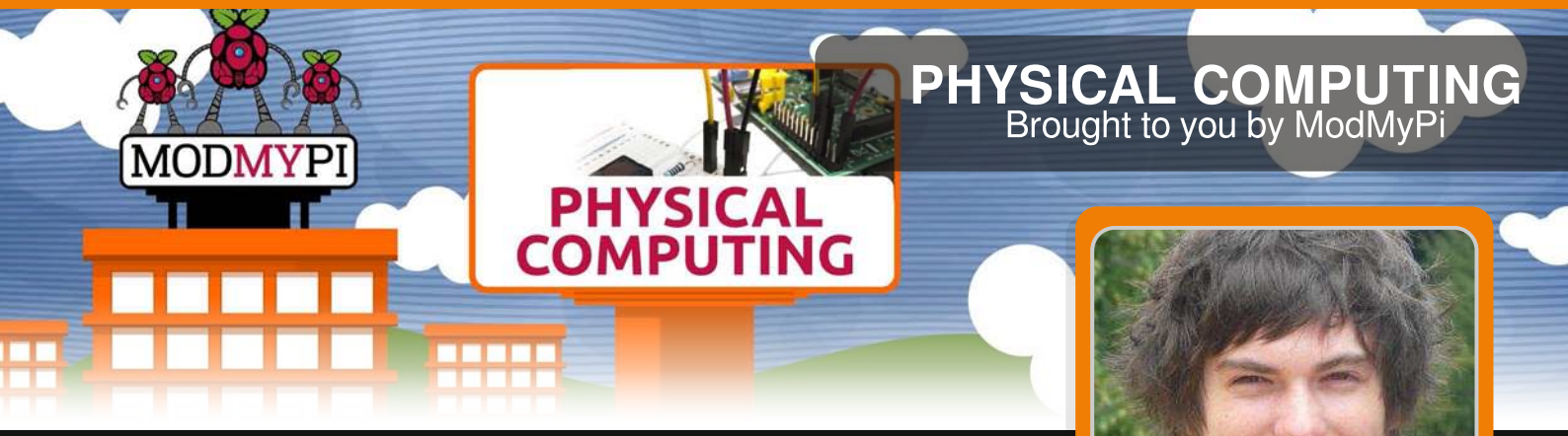
Problems with sensors

There were two major problems with sensors in this design. First of all the AM2315 is an odd duck. It has a power down sequence that requires it to be addressed twice (once to wake up and once to read - for example it takes two invocations of i2cdetect to see the sensor) and secondly, it just isn't reliable using 3.3V for I²C. A solution might be to put a level converter in for the device, but since we had a 5V I²C on the battery watchdog Arduino, we decided to add it to the watchdog. Secondly, the DHT22 has some very tricky timing in order to read it from the Raspberry Pi. Since the Raspberry Pi is a multitasking machine, you can't depend on the timing 100%. This means that readings from the DHT22 fail on a regular basis. The way to fix this problem is to keep reading it until the correct format is received. However, you can only read it every 3 seconds. This plays havoc with an HTTP based monitoring system such as RasPiConnect with 10-15 second timeouts. This problem was fixed by reading the DHT22 in the main software and having RasPiConnect read the last reading from the MySQL database.

What is coming up?

Part 3 goes through the Raspberry Pi Camera Subsystem and Part 4 describes the software system used for Project Curacao. All of the code used in this article is posted on GitHub at github.com/projectcuracao.

More discussion on Project Curacao at: http://switchdoc.blogspot.com



Buttons and switches with the Raspberry Pi - part 3

Jacob Marsh

ModMyPi

SKILL LEVEL : BEGINNER

In our previous tutorial we built a simple button circuit connected to our Raspberry Pi via the GPIO ports and programmed a Python script to execute a command when the button was pressed. We then expanded our circuit with an LED. In this tutorial, we will be expanding our script to include timer and trigger functions for our LED. As the script will be more complicated, it requires a proper clean-up function. Start by reading the previous two tutorials featured in Issues 17 and 18, before trying this one!

Adding LED control code

Now that our LED expansion circuit has been built, we will add some code into our previous program. This additional code will make the LED flash on and off when the button is pressed. Start by booting your Raspberry Pi to the Raspbian GUI (startx). Then start IDLE3 and open the previous example program button.py. Save this file as button_led.py and open it.

Since we want the LED to flash on and off, we will need to add a time function to allow Python to understand the concept of time. We therefore need to import the time module, which allows various time related functionality to be used. Add another line of code underneath line 1:

```
import time
```

Next, we need to define GPIO P18 [Pin 12] as an output to power our LED. Add this to our GPIO.setup section (line 4), below the input pin setup line:

```
GPIO.setup(18, GPIO.OUT)
```

Once GPIO P18 [Pin 12] has been set as an output, we can turn the LED on with the command GPIO.output(18, True). This triggers the pin to high (3.3V). Since our LED is wired directly to this output pin, it sends a current through the LED tha turns it on. The pin can also be triggered low (0V) to turn the LED off, by the command GPIO.output(18, False).

Now we don't just want our LED to turn on and off, otherwise we would have simply wired it to the button and a power supply. We want the LED to do something interesting via our Raspberry Pi and Python code. For example, let us make it flash by turning it on and off multiple times with a single press of the button!

In order to turn the LED on and off multiple times we are going to use a for loop. We want the loop to be triggered when the button has been pressed. Therefore, it needs to be inserted within the if condition 'If input_value == False:', that we created in our original program. Add the following below the line 'print("Who pressed my button!")' (line 9), making sure the indentation is the same:

```
        for x in range(0, 3):
```

Any code below this function will be repeated three times. Here the loop will run from 0 to 2, therefore running 3 times. Now we will add some code in the loop, such that the LED flashes on and off:

```
                    GPIO.output(18, True)
                    time.sleep(1)
                    GPIO.output(18,False)
                    time.sleep(1)
```

The LED is triggered on with the command GPIO.output(18, True). However, since we do not want to immediately turn it back off, we use the function time.sleep(1) to sleep for one second. Then the LED is triggered off with the GPIO.output(18,False) command. We use the time.sleep(1) function again to wait before the LED is turned back on again.

The completed program should be of the form:

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
GPIO.setup(17, GPIO.IN)
GPIO.setup(18, GPIO.OUT)
while True:
  input_value = GPIO.input(17)
  if input_value == False:
    print("Who pressed my button?")
    for x in range(0, 3):
      GPIO.output(18, True)
      time.sleep(1)
      GPIO.output(18, False)
      time.sleep(1)
    while input_value == False:
      input_value = GPIO.input(17)
```

Save the file and open a new terminal window. Then type the following command:

```
sudo python button_led.py
```

This time when we press the button a message will appear on the screen and the LED should also flash on and off three times!

To exit the program script, simply type CTRL+C on the keyboard to terminate it. If it hasn't worked do not worry. Do the same checks we did before. First, check the circuit is connected correctly on the breadboard. Then check that the jumper wires are connected to the correct pins on the GPIO port. Double check the LED is wired the right way round. If the program still fails, double check each line of the program, remembering that Python is case-sensitive and correct indentation is needed.

If is everything is working as expected, you can start playing around a bit with some of the variables. Try adjusting the speed the LED flashes by changing the value given to the time.sleep() function.

You can also change the number of times that the LED flashes by altering the number times that the for loop is repeated. For example if you wanted the LED to flash 30 times, change the loop to: for x in range(0, 30).

Have a go playing around with both these variables and see what happens!

Exiting a program cleanly

When a program is terminated (due to an error, a keyboard interrupt (CTRL+C) or simply because it's come to an end), any GPIO ports that were in use will carry on doing what they were doing at the time of termination. Therefore, if you try to run the program again, a warning message will appear when the program tries to 'set' a pin that's already in use from the previous execution of the program. The program will probably run fine, but it is good practice to avoid these sorts of messages, especially as your programs become larger and more complex!

To help us exit a program cleanly we are going to use the command GPIO.cleanup(), which will reset all of the GPIO ports to their default values.

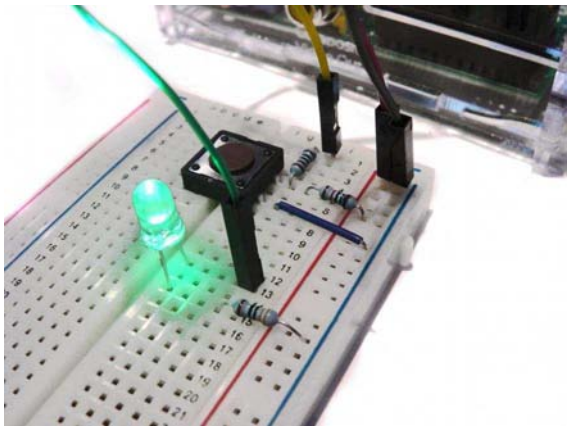
For some programs you could simply place the GPIO.cleanup() command at the end of your program. This will cause the GPIO ports to be reset when the program finishes. However, our program never ends by itself since it constantly loops to check if the button has been pressed. We will therefore use the try: and except syntax, such that when our program is terminated by a keyboard interruption, the GPIO ports are reset automatically.

The following Python is an example of how the try: and except command can be used together to exit a program cleanly.

```
# Place any variable definitions and
# GPIO  set-ups here

try:

# Place your main block of code or
# loop here

except KeyboardInterrupt:
  GPIO.cleanup()

# Program will end and GPIO ports
# cleaned when you hit CTRL+C

finally:
  GPIO.cleanup()
```

Note that Python will ignore any text placed after hash tags (#) within a script. You may come across this a lot within Python, since it is a good way of annotating programs with notes.

After we have imported Python modules, setup our GPIO pins, we need to place the main block of our code within the try: condition. This part will run as usual, except when a keyboard interruption occurs (CTRL+C). If an interruption occurs the GPIO ports will be reset when the program exits. The finally: condition is included such that if our program is terminated by accident, if there is an error without using our defined keyboard function, then the GPIO ports will be cleaned before exit.



Open button_led.py in IDLE3 and save it as button_cleanup.py. Now we can add the code previously described into our script. The finished program should have the form:

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
GPIO.setup(17, GPIO.IN)
GPIO.setup(18, GPIO.OUT)
try:
  while True:
    input_value = GPIO.input(17)
    if input_value == False:
      print("Who pressed my button?")
      for x in range(0, 3):
        GPIO.output(18, True)
        time.sleep(1)
        GPIO.output(18, False)
        time.sleep(1)
      while input_value == False:
        input_value = GPIO.input(17)
except KeyboardInterrupt:
  GPIO.cleanup()
# Program will end and GPIO ports cleaned
# when you hit CTRL+C
finally:
  GPIO.cleanup()
```

Notice that only the loop part of the program is within the try: condition. All our imports and GPIO set-ups are left at the top of the script. It is also important to make sure that all of your indentations are correct!

Save the file. Then run the program as before in a terminal window terminal:

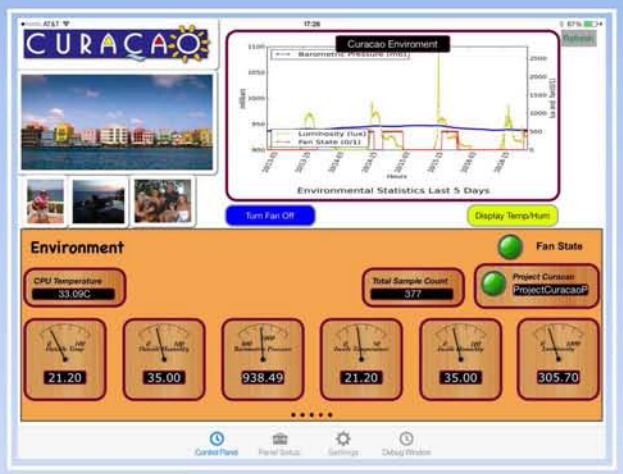
```
sudo python button_cleanup.py
```

The first time you run the file, you may see the warning message appear since the GPIO ports have not been reset yet. Exit the program with a keyboard interruption (CTRL+X). Then run the program again and hopefully this time no warning messages will appear!

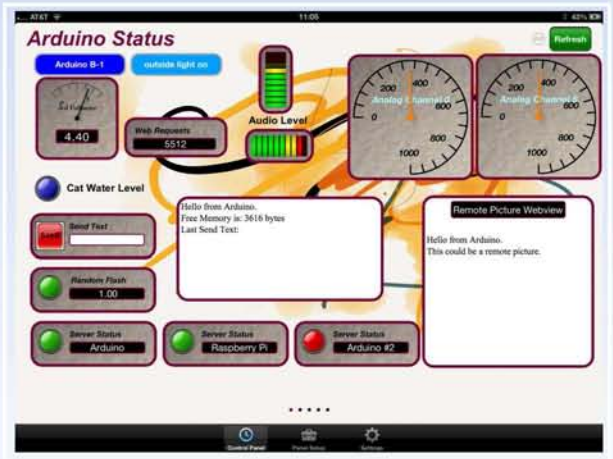
This extra code may seem like a waste of time because the program still runs fine without it! However, when we are programming, we always want to try and be in control of everything that is going on. It is good practice to add this code, to reset the GPIO when the progam is terminated.

RasPiConnect

Connect your Raspberry Pi to the World!



Now Supports Arduino with in-app purchase



Available on the App Store

RasPiConnect connects your Raspberry Pi to the outside world.

It allows you to control virtually anything you connect to your Raspberry Pi from your iPad or iPhone.

- EASY to setup - no syncing required
- Buttons, gauges, webpages, webcam pictures and more!
- Build your pages on your iPad/iPhone
- Supports multiple Raspberry Pis and multiple Arduinos
- Five pages of control panels
- Unlimited Controls
- Exchange your panels with friends
- Supports any computer that supports Python (Windows, Linux, etc.)
- Now allows custom backgrounds

MiloCreek
Where Innovation Flows

Fred Sonnenwald & Hamish Cunningham
Guest Writers

Painting with XLoBorg

SKILL LEVEL : INTERMEDIATE

Maker Culture meets Art

There is a new conjunction emerging around open hardware, maker culture, and art. Remember how pop culture changed with the advent of punk in the late seventies? We seem to be witnessing a similar explosion of 'garageband' creativity in the tennies, and the Raspberry Pi is proudly leading the charge for general purpose open source devices in this context. (The Raspberry Pi Foundation even has an artist in residence — Rachel Rayns.)

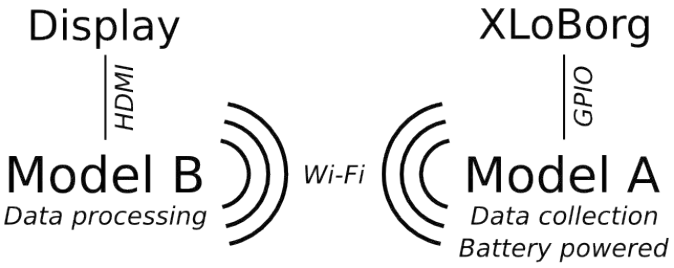
Open-source hardware allows people to make their own robots, cameras or even electrocardiagraph machines by downloading schematics and incorporating any changes they need — and, typically, free open-source software is available to run these projects. 3D printers, laser cutters and CNC routers have helped this adoption of the open-everything ethos.

There are stacks of DIY projects based on the Raspberry Pi, and the flood shows no sign of slowing. The Raspberry Pi is the first general purpose device (in contrast to the magnificent, but more specialised Arduino), which is very easy to cobble together with add-on electronics. The thriving community that has grown up around the Raspberry Pi (including this magazine) is making a huge impact, from changes in the UK school curriculum to the Onion Pi anti-surveillance device and BrickPi's new brains for old lego Robots.

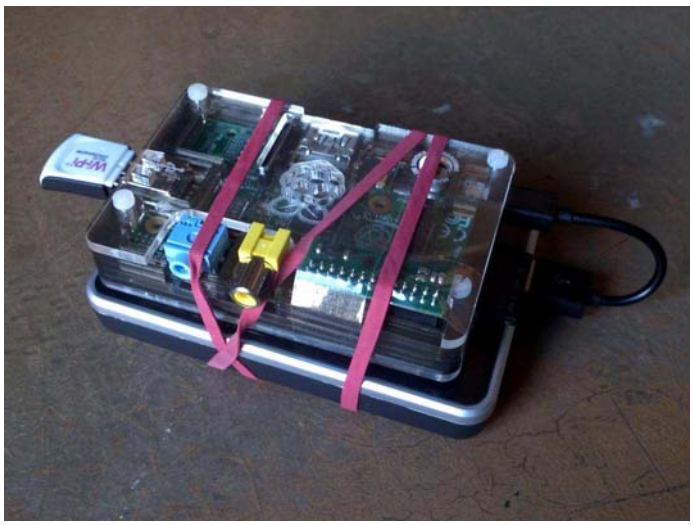
This article describes the PiBrush, a simple on-screen painting system that uses the XLoBorg motion and direction sensor add-on board from PiBorg. The XLoBorg adds an accelerometer and a magnetometer (compass) to your Pi and makes all sorts of motion-based interaction possible — like a Kinect, but free and open. The header graphic for this article was created with the PiBrush.

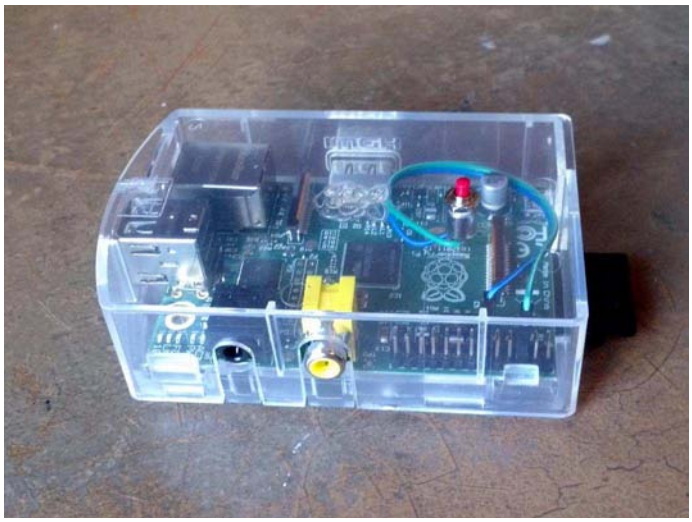
The PiBrush is an interactive art and technology exhibit (a rather grand name for something so small) that simulates flicking paint off the end of a paintbrush onto canvas — as Jackson Pollock famously did in the 1940s and 50s. The setup includes two Raspberry Pis (one Model B and one Model A), one XLoBorg, a battery pack, a display and two Wi-Fi dongles. The Model A is held by the user and waved around. It collects acceleration data with the XLoBorg. These data are then transmitted via Wi-Fi to the Model B, which processes the data collected into paint droplets and displays them on the screen.

Functionally it looks like this:



and here is the hardware:





We may improve on the elastic band in future models. The Model A (client) top and the Model B (server) bottom, where the red button on the server saves the picture and starts a new picture.

The Code

The hardware for this project is fairly straightforward. The XLoBorg plugs directly into the GPIO pins as a block on the client, and a push button simply connects a ground and GPIO pin on the server. The software is where the challenge lies: intermediate to advanced code is needed, using Python and some basic physics.

The code is available on GitHub at:
https://github.com/hamishcunningham/pi-tronics/tree/master/pibrush/bin
The core is in accel_client.py which runs on the Model A and accel_server.py which runs on the Model B. The former reads from the sensor and sends data across the network; the latter processes the data and displays the output on the screen.

accel_server.py

The server script is the meat of the programming as it handles the physics simulation and display. First, as we're dealing with sensor input we want to use something called a moving average to store accelerometer readings, which is initialised like this:

```
# length of moving average array
AL = 20

# accelerometer storage for moving average
AXa = numpy.zeros((AL, 1))
AYa = numpy.zeros((AL, 1))
AZa = numpy.ones((AL, 1))

# array index for accelerometer data
Ai = 0
```

A moving average is the average of, in this case, the last 20 readings. Some sort of filtering is almost always necessary when dealing with the sort of analog input data that comes from an accelerometer to prevent sudden jumps in the readings from having an undue impact on the processing. A moving average is a convenient and easy way to do it. We simply input the current reading at index Ai and then increment Ai until it exceeds AL, then wrap it back around:

```
AXa[Ai] = float(a[0])
AYa[Ai] = float(a[1])
AZa[Ai] = float(a[2])
Ai = Ai + 1
if Ai == AL:
 Ai = 0
```

This is an array of the acclerometer data, as read over the network. I have used NumPy, because it executes more quickly than standard Python lists and there are convenient functions to make things simpler like .ones() to initialise all of AZa to a value of one — we assume a starting 1G of gravity downwards.

Functions

If you don't know, a function is a bit of code written in such a way that it can be executed from different places in the main program. Each call you can pass different arguments (input parameters) that will be operated on and a new

result returned. There are two quite important functions used in the code that have to do with polar coordinates. If you know X, Y, Z, those are Cartesian coordinates.

```
def polar(X, Y, Z):
  x = numpy.linalg.norm([X, Y, Z])
  if (x > 0):
    y = -math.atan2(Z, X)
    z = math.asin(Y / x)
  else:
    y = 0
    z = 0
  return (x, y, z)
```

Here polar takes a Cartesian X, Y, and Z coordinate and returns the equivalent polar coordinates, where x is R (the radius to the point from the origin), and y is A and z is B. The latter are the angular rotation about the Z and Y axis. Because the Model A may be rotated relative to the screen, we need a convenient mechanism for rotating the acceleration vectors recorded to ones we can use on the screen. In order to do that though we need to subtract the Earth's gravitational field. This is what the polar coordinates are used for. It's an extension of the regular subtraction of vectors. This function was based on this forum thread.

```
def cartesian(X, A, B):
  x = 0 # don't bother - isn't used
  y = X * math.sin(B) * math.sin(A)
  z = X * math.cos(B)
  return (x, y, z)
```

Here cartesian, as you might suspect, does the opposite of polar, taking the distance X and rotations A and B and turns them back into Cartesian coordinates. As this code is only used as part of getting coordinates ready for the screen, x, as the coordinate into the screen, is permanently set to 0. This is an optimization to help the code run better on the Pi. The code here is based on this explanation of Cartesian and polar coordinates.

Main program

The main program consists of an infinitely repeating loop that reads the accelerometer data over the network, applies the earlier moving average, and then processes it.

```
# move time forward
dt = time.time() - last_time
last_time = time.time()
```

The first thing to be done each loop is to move time forwards. Ideally execution time for each loop is identical and dt (the timestep) is constant. However, this isn't the case, so by knowing how long it's been since the last loop, we can check how far to move things, i.e., distance = velocity * time.

```
# moving averages for acceleration
AX = numpy.sum(AXa) / AL
AY = numpy.sum(AYa) / AL
AZ = numpy.sum(AZa) / AL

# combined acceleration for
# working out resting gravity
A = math.fabs(numpy.linalg.norm([AX,
  AY, AZ]) - 1)
```

After reading in the accelerometer data and putting it in the AXa, etc., arrays, we then need to take the average of that array. .sum() adds up the values in an array. Then to perform an average, we divide by the number of elements AL. Therefore, AX, AY and AZ contain the moving average.

The total combined acceleration A is worked out by calculating the Euclidean distance from 0 to the acceleration vector position using linalg.norm(). At rest this should work out to just about 1 (remember we're working in acceleration in G(ravities), which is why we subtract 1. We then use .fabs() so that we always have a positive result which indicates the difference between acceleration due to gravity and the experienced acceleration. At rest this number should be very small.

```
# in a slow moment store most recent
# direction of the gravitational field
if A < 0.02 and (last_time - last_G) > 0.12:
  GX = AX
  GY = AY
  GZ = AZ
  (PGR, PGA, PGB) = polar(GX, GY, GZ)
  last_G = last_time

# rotate to screen coordinates
# and subtract gravity
(PAR, PAA, PAB) = polar(AX, AY, AZ)
(GAX, GAY, GAZ) = cartesian(PAR,
  PAA - PGA + PSGA, PAB - PGB + PSGB)
GAZ = GAZ - PGR
```

Now that we know something about gravity and the current

acceleration, we can act on it. I've pointed out how in order to know which way the Model A is moving, we need to know where gravity is first. We can't know it exactly, but we can estimate it. Since A is the acceleration the Model A is experiencing excluding gravity, it's very low when the Model A isn't moving at all. This means that the only acceleration being experienced is due to gravity. Therefore, we can take the estimate and turn it into polar coordinates.

For every loop we need to actually do the gravity subtraction and rotation for the screen. We turn the current acceleration into polar coordinates. Then on the next line we need to turn them back into cartesian coordinates, while subtracting the estimated rotation of the gravitational field. The GAZ line after this subtracts gravity from its absolute direction.

Paintbrush physics

Perhaps the most interesting bit of what is going on within every loop it the paint brush physics. This is the code that controls what happens on the screen. Everything up to this point has been to define GAY and GAZ, two variables indicating horizontal and vertical acceleration relative to the screen. Now we can interpret this acceleration and make something happen.

```
# acceleration detection for paint strokes
A = numpy.linalg.norm([GAY, GAZ])
```

After this, A is the total acceleration of the Model A, with respect to the screen, ignoring gravity. We can use this number to detect roughly if we are speeding up or slowing down. What happens after that?

```
if fast == 1:
  # accelerate the paint brush
  VX = VX - GAY * dt * 170
  VY = VY - GAZ * dt * 170
  BX = BX + VX * dt * 120
  BY = BY + VY * dt * 120
```

This bit of code is actually responsible for moving the brush, and only when we think it's moving. To get position from acceleration, we have to integrate twice. Imagine you are in a car travelling down the motorway at 100 kph. In one hour you will have travelled 100 km. Easy? That is one simple integration, going from velocity to displacement. Acceleration is the rate of change in velocity, e.g. the speed at which the needle of the speedometer in the car climbs. So now imagine accelerating at 1 kph per second, after 10 seconds you'll be going 10 kph faster. After this acceleration, instead of 100 kph you are now going at 110 kph. To get the distance, you have to to integrate twice. (Fun fact: if you kept up that acceleration, after an hour you'd be going 3,700 kph and would have traveled 36,370 km. Or almost around the Earth.)

We increment the brush velocity by the acceleration, factored by the timestep to keep the animation smooth. I have also added a factor of 170 to scale up the acceleration, so it displays nicely on the screen. (This means that one pixel approximately represents 170 metres.) The next integration increments the brush position by adding on the current velocity, also multiplied by the timestep and scaled, this time by 120. (These values just work, but are physically nonsense.)

```
# add splotches.... high velocity big
# splotches far apart, low small close
if P > 0:
  V = numpy.linalg.norm([VX, VY])
  S = S + V
  d = A * random.randint(3, 5) * 25 + V
```

Now that the paintbrush is moving, here comes the last and most important bit of code: making paint droplets. This bit of code is only run while the paintbrush is moving and only if there is paint on the brush. Vaguely, one expects that the further the brush has been swung, more paint should come off. The faster the brush is moving should also cause more paint to come off. V is calculated as the total velocity and S is a running displacement. d is calculated as a rough estimtate of paint droplet spacing.

Paint droplets fly off due to two factors:

1) Fluid mechanics. Roughly, I'm speaking of the affect of what happens when you move a glass of water too quickly.

2) Air resistance. The force you feel acting on your hand when you stick it out the car window on a drive on a nice summer's day.

Both of these factors are rather complex subjects. Therefore, I have lumped them together as the produce similar results — paint flying off the brush. d is made up of

A (our acceleration) times a random factor which is the fluid dynamics bit, and V is added for the air resistance bit. The random factor makes things a bit more interesting, perhaps taking into account things like globs of paint or hairs in the brush getting tangled. 25 is the scaling factor this time. (This applies to the acceleration term only as velocity was already scaled before.)

```
if S > d:
  S = S - d
  P = P - pow(A*4, 2) * math.pi
  pygame.draw.circle(screen, (COLR, COLG,
    COLB), (int(BX), int(BY)), int(A*45))
  draw = 1
```

If we've travelled further than the expected paint droplet seperation according to our approximate simulation, we need to draw a paint droplet! We calculate the amount of paint that was in the droplet arbitrarily, as the acceleration times four, squared, times π (pi). (This is the area of a circle formula.) This is subtracted from P, the paint on the paintbrush variable. The paint droplet actually drawn on the screen is done using cricles, with the Pygame function draw.circle(). The drawing on the screen takes place with a random colour, at the paintbrush position (BX, BY). A*45 is the paint droplet radius, where 45 is another scaling factor.

KICKSTARTER

Mobile power for the Raspberry Pi - easy on the one hand, hard on the other: it is easy to plug in a 5V battery pack - but when it runs out your Raspberry Pi gets a power cut that might well corrupt the SD cards.

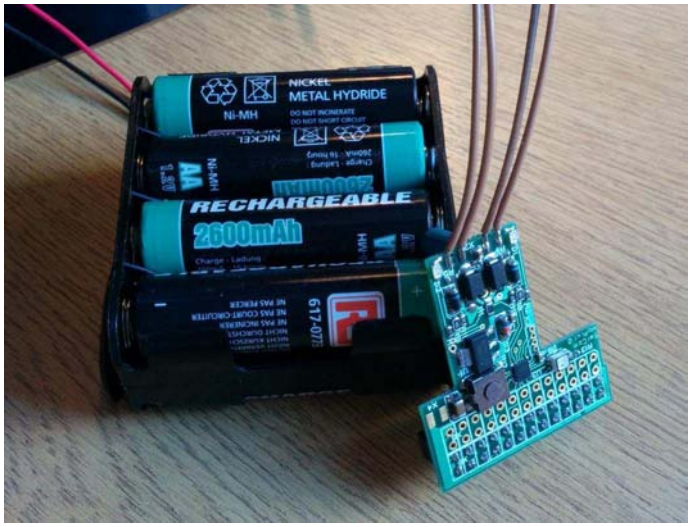
Over the last year we've been designing what we hope is the perfect mobile power solution for the Raspberry Pi, which we're calling MoPi, and we've just taken delivery of the second generation prototype. I think it does pretty much everything you could want for your Raspberry Pi on the go:

• it will accept multiple sources, including standard AA batteries, or your car cigarette lighter, or an old laptop power supply or etc..
• it will let you do hot-swap power replacement without stopping work
• it will shutdown your Raspberry Pi cleanly if the battery charge level gets too low, and it has a neat little power switch on the top to save you logging in to shutdown at other times,
• it behaves like a uninterrupltible power supply (UPS) when the Raspberry Pi is plugged into a mains supply

and it even fits in the Pibow (and other well-known Raspberry Pi cases)

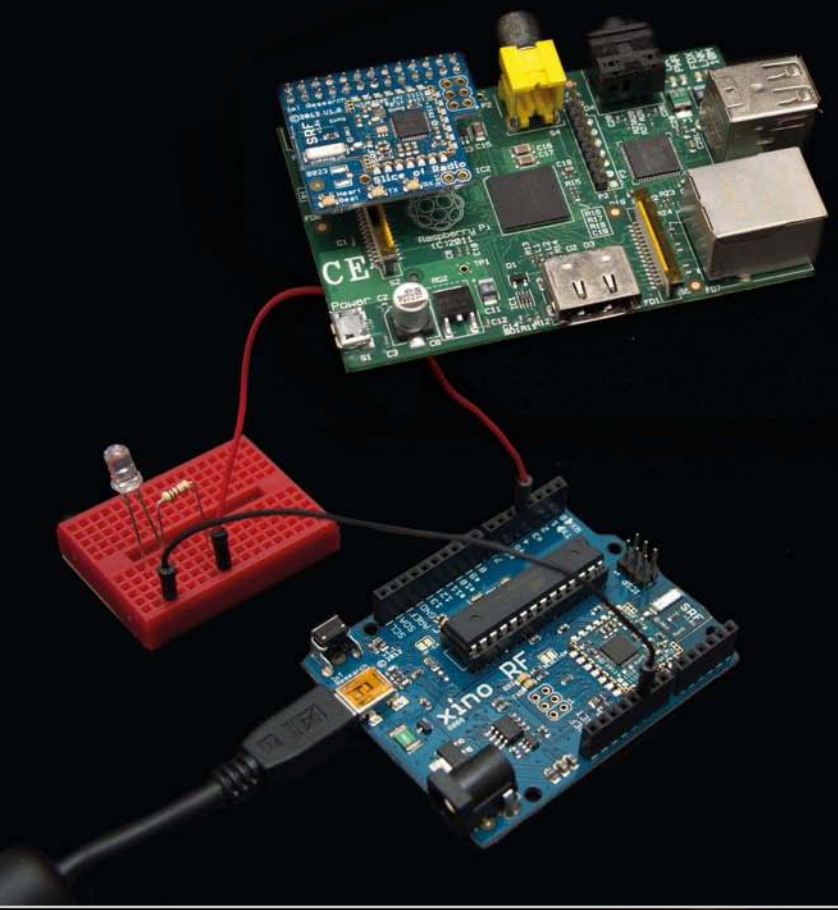
Here's the circuit board, with an eight-pack of AAs (that will get you around 9 hours of mobile time for a Model B Raspberry Pi.

More details of the project can be found at:
http://pi.gate.ac.uk/pages/mopi.html



RasWIK

Wireless Inventor Kit for the Raspberry Pi ™



Contains 88 Parts

- Starter examples require no soldering at all
- Plug in wires and breadboard to make building easy and fast
- A pre installed and configured Raspberry Pi OS makes using your Pi a breeze
- Examples you build, can be mixed with our out the box wireless devices...how cool!
- **"It provides possibly the simplest platform for experimenting with wireless sensor networks I've ever seen."**
  (Gareth Halfacre, CustomPC, Issue 121, Oct 2013)
- Made in the UK

£49.99

www.ciseco.co.uk

Raspberry Pi is a trademark of the Raspberry Pi Foundation

CISECO
Powering creative minds

Pi NoIR - The new high definition infrared camera module from Raspberry Pi

Featuring the same 5 megapixel image sensor as the standard Raspberry Pi camera with the infrared cut-off filter removed to enable IR light frequencies, the Pi NoIR is compatible with Raspberry Pi model A & model B.

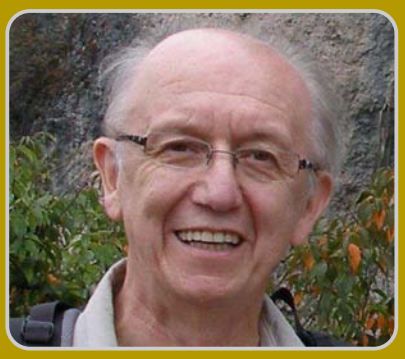
- Extended spectral range
- 5 megapixel image sensor
- Still picture resolution: 2592 x 1944
- Video resolution: 1080p 30fps

Raspberry Pi

Available now at www.rs-components.com/raspberrypi

RS

How to configure OpenELEC so you can watch TV programmes



Geoff Harmer

Guest Writer

SKILL LEVEL : BEGINNER

Do you want to watch TV programmes using your Raspberry Pi?

• No need to buy a smart TV to get your HDTV to connect to catch-up TV services such as BBC iPlayer and ITV Player in the UK, or CTV and Space and Bravo in Canada, or RTE1 and RTE2 in Ireland or Network10 in Australia.
• No need to have your Raspberry Pi keyboard and mouse and their cables spoiling the look of your HDTV.
• Use a Raspberry Pi with a Wifi USB dongle running OpenELEC - a compact, high performance Linux distribution that has XBMC media centre software installed - and then hide the Raspberry Pi behind the HDTV.
• Use a remote control app on your Android, iOS or Blackberry smartphone to control XBMC via your home Wifi so you can select and watch catch-up TV programmes on your HDTV.
• No Linux commands are needed to install or use OpenELEC with the Raspberry Pi.

My style of television watching over the last few years has switched from live viewing and recording, to predominantly using catch-up TV. Catch-up TV is particularly helpful when a missed programme gets a good review the next day from friends or from TV reviews in newspapers or on the web.

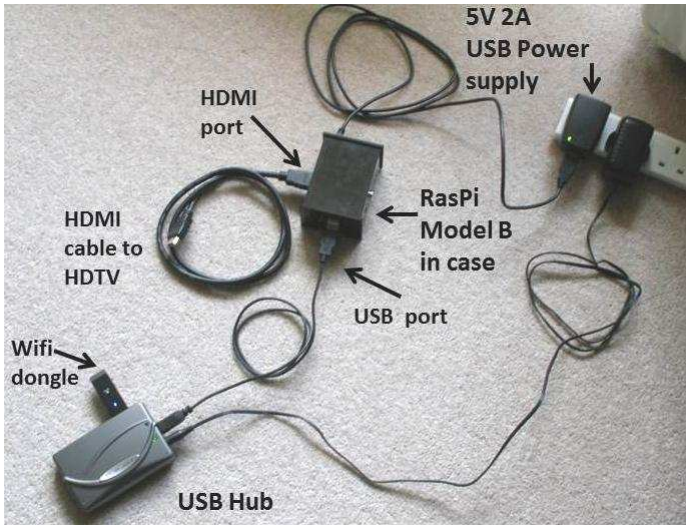
I am going to show you how to install catch-up TV services using XBMC and the OpenELEC distribution on a Raspberry Pi to work with your HDTV. It takes about 60 minutes to set up and requires no use of Linux commands either!

Permanent components

• Raspberry Pi Model A or Model B.
• Good quality 5V power adapter for the Raspberry Pi (minimum 700mA or greater).
• HDMI cable to connect the Raspberry Pi to your HDTV.
• Wifi nano USB dongle if you do not have easy access to a wired internet connection (the Wifi dongle from http://www.thepihut.com uses built-in drivers and does not require a powered USB hub).
• 4GB+ SD card loaded with the latest NOOBS software from http://www.raspberrypi.org.
• Android, iOS or Blackberry 10 device with a downloaded app to remotely control XBMC, e.g. XRMT for Blackberry 10 OS, or Official XMBC Remote for Android, or Official XMBC Remote for iOS.

Note: The Raspberry Pi USB ports may not be able to supply enough current for certain Wifi dongles. This can result in video stuttering. Assuming you have a good connection, the solution is to use a powered USB hub with a minimum 2A power supply. Some hubs will also back power the Raspberry Pi so you only need the one power supply. Check http://elinux.org for a list of known good hubs.

Installation instructions



Step 1. Get NOOBS

Copy the latest NOOBS distribution onto a 4GB+ SD card. When finshed insert the SD card into the Raspberry Pi, plug in the Wifi USB dongle and use a HDMI cable to connect the Raspberry Pi to your HDTV. Plug in the power adaptor and boot up the Raspberry Pi. The first time you start NOOBS you will be asked to choose what operating system to install. Choose OpenELEC. This will take a few minutes to install, after which you can reboot your Raspberry Pi.

Step 2. XBMC and networks

Note: If you are using a Raspberry Pi Model A then for this section you will temporarily need to use a powered USB hub so you can connect both the Wifi dongle and a mouse.

After your Raspberry Pi has started, XBMC will present its main screen known as Confluence (blue bubbles background) followed by an initial setup wizard. Here you choose your region, provide a friendly network name then choose from a list of available wireless connections. You will need to know your wireless connection name and security passphrase.

By default your network IP address and DNS servers will be automatically configured. However you may want to change the DNS servers to watch overseas programming using services like http://www.unblock-us.com, or you may want to change the IP address to use a static IP. If you want to change the network settings later, from the main screen select SYSTEM, OpenELEC and then click on Connections. Click on your connection and choose Edit from the pop-up menu.

A static IP address tends to be more reliable because it remains fixed and so the remote app on your smartphone will always connect. The IP address created using DHCP can vary depending on what other devices are connected to the broadband router.

Setting up a static IP (optional)

Choose to Edit your connection, as described above and click on IPv4. You will see the IP Address Method is currently set to DHCP. Click on this and change it to Manual. Now click on IP Address and enter an unused address within the range permitted by your broadband router e.g. 192.168.1.50. You can leave the Subnet Mask (e.g. 255.255.255.0) and Default Gateway (e.g. 192.168.1.254) at their default settings.

To change the DNS settings for your connection choose DNS servers. You will see options to enter IP addresses for up to three nameservers. By default these will already have the IP addresses of the DNS servers from your internet provider, but you can choose to use others. For example OpenDNS provides enhanced security using its own free to use DNS server IP addresses at http://www.opendns.com. [*Ed: I*

changed these to the Unblock-Us DNS servers so, as an ex-pat, I can still watch BBC iPlayer content in Canada].

At this point it is a good idea to click the Power button on the main screen and reboot the Raspberry Pi. When OpenELEC restarts you should have a working network connection.

Step 3. Other important XBMC features

1. Is the display too large for your screen or do you see a black border? To change the screen size select SYSTEM then Settings and click on Appearance. In the Skin section, change Zoom (up and down keys) to set an appropriate reduction.

2. Stop RSS feeds across the bottom of the Confluence screen. From the Confluence screen select SYSTEM then Settings and click on Appearance. In the Skin section click on Show RSS news feeds to turn it off.

3. To allow remote wireless control of XBMC select SYSTEM then Settings and click on Services. In the Webserver section click on Allow control of XBMC via HTTP to turn this option on. Note the default username is xbmc with no password. You can optionally change these. In the Remote control section click on Allow programs on other systems to control XBMC.

4. To ensure OpenELEC has your home workgroup from the main screen select SYSTEM then Settings and click on Services. In the SMB client section the Workgroup option is set to WORKGROUP by default. If your home network has its own workgroup name then you must change WORKGROUP to your home workgroup name. You can find out your home workgroup name from your Windows PC by opening the Control Panel, open the System and Security category then click on System. The workgroup is shown near the bottom.

Step 4. Install TV catch-up files

Add-ons need to be added to XBMC in order to use catch-up TV. Here are some examples that have been tested to work.

UK:
BBC iPlayer
http://code.google.com/p/xbmc-iplayerv2/downloads/list
ITV player
http://code.google.com/p/xbmc-itv-player/downloads/list
Ireland:
http://mossy-xbmc-repo.googlecode.com/files/plugin.video.irishtv-2.0.11.zip
Canada:
http://goo.gl/j6Fw5K
Australia:
http://xbmc-catchuptv-au.googlecode.com/files/plugin.video.catchuptv.au.ten-0.4.0.zip

The Canadian add-in is particularly good as it covers 20 stations, but just like the BBC and ITV add-ins, it is region locked. Fortunately Australia's Network10 and Ireland's RTE stations appear to be freely available to all.

Using your PC, copy the latest zip files onto a USB memory stick without unzipping them. Now plug the USB memory stick into the USB port of your Raspberry Pi and power-up the Raspberry Pi. To install these add-ons, from the main menu select VIDEOS then Add-ons and click on Get More. At the top of the list click on .. (i.e. two dots). At the top of the next list again click on .. (i.e. two dots). Now click on Install from zip file. A new window will pop-up. Select your USB memory stick. If it is not visible remove it and insert it again.

Navigate to the first add-on you wish to install and click it. It gets installed at this point within about 30-40 seconds but it is not obvious. You will be returned to the Install from zip file menu and you may momentarily observe (bottom

right of the screen) that the new add-on has been installed. Check by returning to VIDEOS and selecting Add-ons and you should see the name of the add-on you just installed. If it is absent, wait a few minutes for it to appear and if still absent reboot your Raspberry Pi. Repeat for each add-on, one at a time.

Step 5. XBMC remote control

Download and install the Official XBMC Remote app to your Android or iOS device, or install XRMT for your Blackberry 10 OS device.

As an example, here is how to configure XRMT on your Blackberry 10 OS device. Once installed open the app and click the three dots icon (bottom right) and select Add Server. For Server name enter openelec (lower case). For Server IP address enter the address that you set up for the Raspberry Pi in step 2 on page 23. Leave the Port code as 9090 then click the icon at bottom left.

Now scroll down from the top of the app, click on Settings and change the Auto connect by clicking on openelec below it. Close the settings screen. The app is now ready to use.

Once you have installed an XBMC remote app on your smartphone you are ready to control XBMC without a mouse or keyboard. Simply use the remote control app to navigate the XMBC Confluence screen, select VIDEOS and then select Add-ons. Your add-ons will be displayed. Simply click on an add-on such as iPlayer to run it. Enjoy the catch-up TV programmes.

Interesting extra facts and tips

HDMI connections

Are all the HDMI sockets on your TV in use with other devices such as a PVR, bluray player and games console? Not enough HDMI sockets on your TV for your Raspberry Pi? The solution is to use a 2-in/1-out HDMI switch (e.g. Belkin). Thus both the Raspberry Pi and the other device are connected to the HDTV. The Belkin product has a button on it for selecting which of the 2 inputs to currently use.

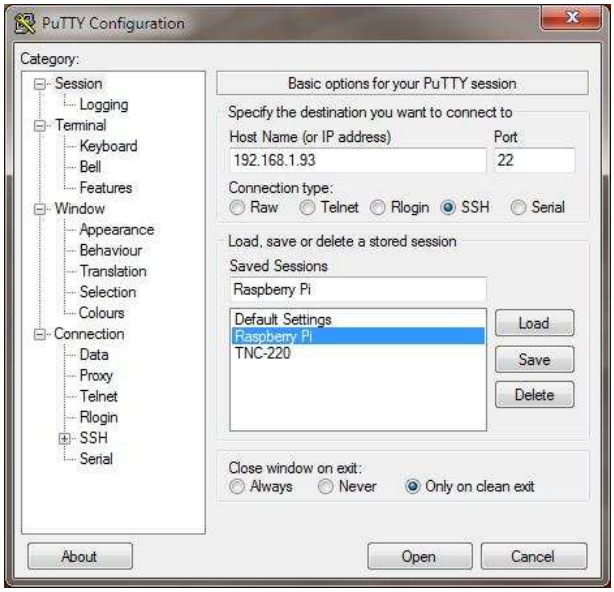
XRMT connection issue

If you have been using a mouse to control XBMC, you may find that your Blackberry XRMT won't connect. Unplug the mouse and reboot the Raspberry Pi and then XRMT should connect.

Command line access

Do you want to access the OpenELEC operating system with Linux commands from your Windows PC? You can do this using a program called PuTTY. PuTTY uses the SSH protocol to connect to the Raspberry Pi. Download the latest version of PuTTY from http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html.

You also need to set OpenELEC to allow a SSH connection. From the XBMC main screen select SYSTEM then OpenElec and click on Services. Select the SSH option and click to enable it. Reboot the Raspberry Pi. Once PuTTY is installed on your Windows PC simply run the file putty.exe. After the PuTTY screen has opened enter the IP address of your Raspberry Pi. Leave the Port as 22 and choose SSH as the connection type. Click on Open then enter the username and password. For OpenELEC the default username is root and the default password is openelec.



Expand your Pi

Stackable Raspberry Pi expansion boards and accessories

IO Pi

32 digital input/output channels for your Raspberry Pi. Stack up to four IO Pi boards to give you 128 I/O channels.

£16.99

RTC Pi

Real-time clock with battery backup and 5V I²C level converter for adding external 5V I²C devices to your Raspberry Pi.

£9.75

ADC Pi

8 channel analogue to digital converter. I²C address selection allows you to add up to 32 analogue channels to your Raspberry Pi.

£17.99

Com Pi

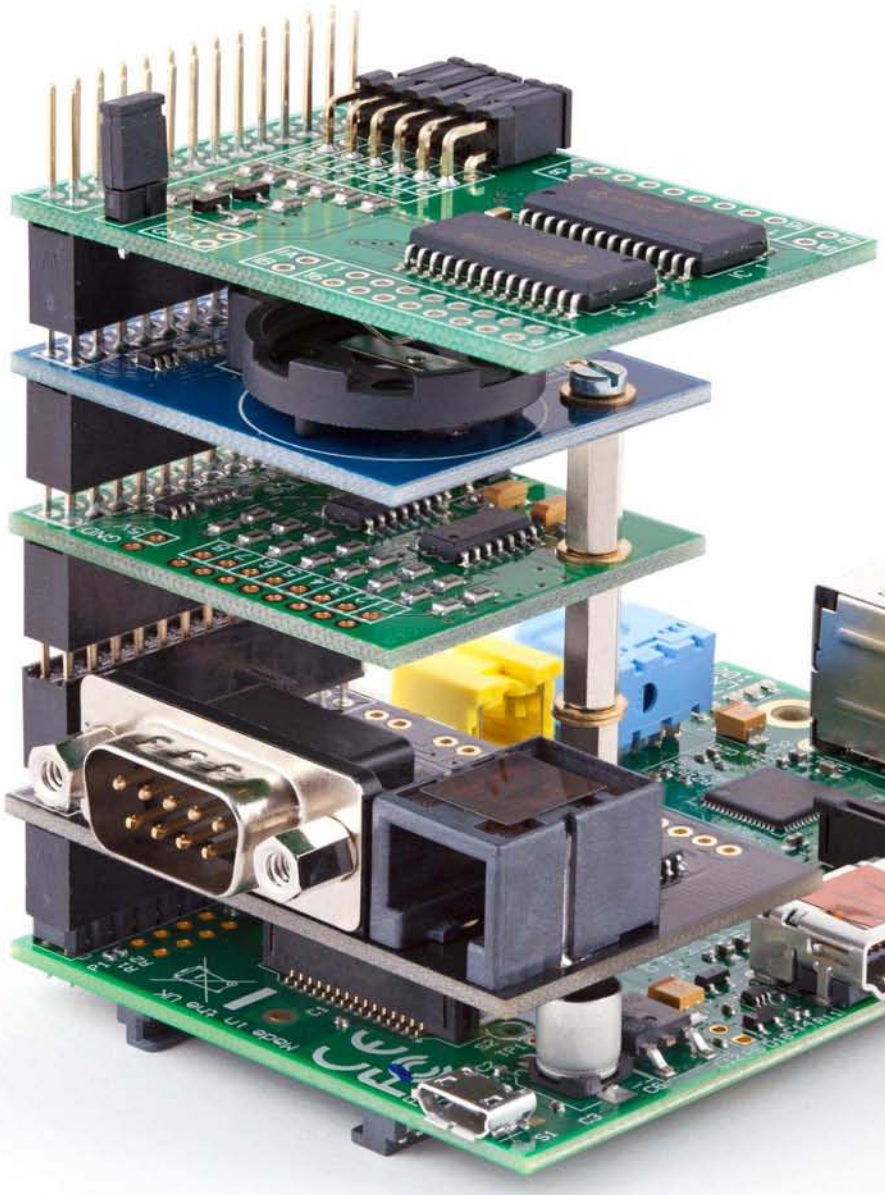
RS232 and 1-Wire® expansion board board adds a serial port to your Raspberry Pi. Ideal for the Model A to enable headless communication.

£19.99

Buffer Pi

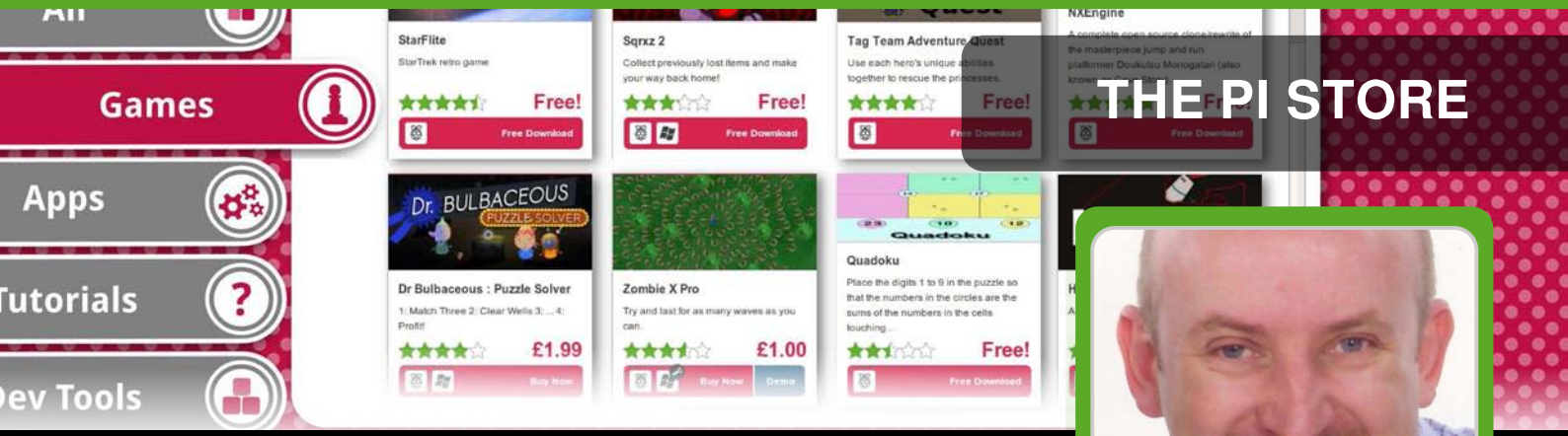
Bidirectional Voltage-Level Translator and GPIO protection. Use 5V devices devices on your Raspberry Pi GPIO port.

£13.99



AB electronics UK

www.abelectronics.co.uk



THE PI STORE

A look at the diverse range of applications in the Pi Store

Ian McAlpine

MagPi Writer

SKILL LEVEL : BEGINNER

Twelve months ago the Pi Store was launched with just 24 titles. Today there are over 100 titles and the vast majority are free. The MagPi takes a look at some of the diverse content in the Pi Store, especially the non-free content, to help you determine what is worthy of your time and money.

Digital downloads

Apple was one of the first companies to encourage the mass adoption of digital downloads with its iTunes store for music and movies. This was later followed with the App store, the iBooks store and finally iTunes U for students. The App store was the only way you could get applications for Apple mobile devices and later it was extended to Apple's MacBook and iMac computers.

Amazon pioneered the digital download of books and today most major bookstores have digital download affiliations. Of course the instant gratification advantages of digital downloads were not lost on game console manufacturers with Microsoft, Sony and Nintendo all introducing digital download stores for their respective game consoles. Digital app stores are less prevalent for Windows based PCs with notable exceptions being game centric Steam and Origin.

Introducing the Pi Store

Ultimately all of the previously mentioned digital download stores have a commercial interest for the host company. But this was not the case for the Raspberry Pi Foundation when they launched the Pi Store. Instead of thinking how much money could be made, they saw all the advantages of having an app store dedicated to the Raspberry Pi:

- a place where developers of all ages can share their creations with the Raspberry Pi community.
- a place where complete beginners can install and remove great Raspberry Pi titles without having to go near a command line.
- a place where every Raspberry Pi user can discover new content for their computer.

Unfortunately some did not see this and criticised the Foundation for teaching kids capitalism! But is this any different from giving kids pocket money for doing chores? Having said that, there are currently 100+ titles and at the time of writing only six cost money... and you can buy them all and still have change from US$12.00/£8.00.

The Pi Store content is divided into five categories; Games, Apps, Tutorials, Dev Tools

and Media. Before we explore each of these areas here are some navigation tips that will help you find the content you want.

In the Explore tab of the Pi Store there is an option to filter the content by status and also an option to specify the sort order. If you want to explore content which is still work-in-progress then change the status filter to In Progress. You can also set it to Show All to see all Pi Store content.

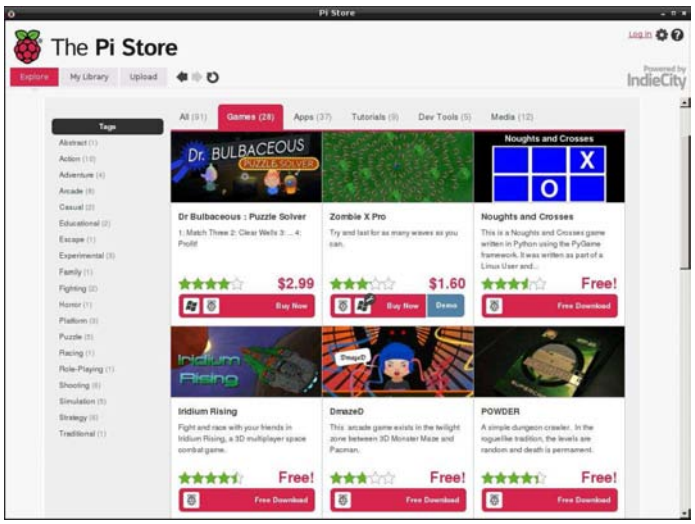
There are many sort options, but the most useful are Most Played, Newest First and Top Rated. Note that currently the Pi Store does not store these settings so the next time you open the Pi Store the status will likely be reset to Finished and the sort order will be reset to Price - Highest.

Several programs in the Pi Store do not run on LXDE and require a reboot before they start. Although there is a warning on the application page in the Pi Store, there is no final "Are you sure?" after you click on Launch. So make sure you have everything saved first.

Don't forget to tip!

When you receive good service you leave a tip. Almost everything in the Pi Store is free. If you enjoyed playing a game or found a title useful, consider showing your appreciation by leaving a tip for the developer. It's easy to do. In your library select the title you want to tip then click View Details. In the details page click Tip this Project to send $1.00/£1.00 by default.

Games



Surprisingly this is not the largest category in the Pi Store, but with 30 titles at the time of writing there is something for all gamers here.

In **Dr Bulbaceous : Puzzle Solver** (US$2.99/£1.99) you drop different coloured objects and when 3 or more are touching they disappear. The goal is to clear the screen before the objects reach the top. Although I found the game too easy, kids will enjoy it. It is colourful and well implemented. Tip: To swap your current object for something more potent, press the Enter key. It looks like the mouse should work, but it didn't for me.

Alas **Zombie X Pro** ($1.60/£1.00) refused to run correctly for me so I cannot comment on it. There is a demo option you can try to see if it works for you.

DmazeD is a lot of fun. You search a maze for the key to the exit, but it is dark so you have a limited field of vision. Listen out for the monster that is also in the maze with you!

Another game I found myself thoroughly enjoying is **The Little Crane That Could**. Through dexterous use of your crane and its many movements you complete a series of increasingly difficult tasks.

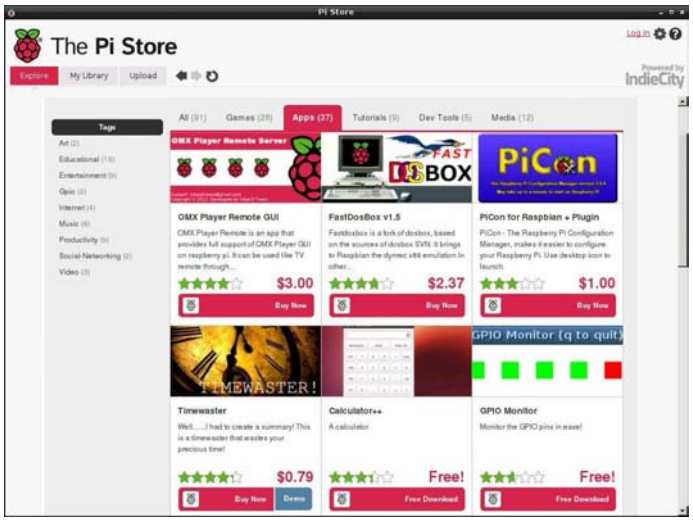
Many folks of a certain age (like myself!) use their Raspberry Pi to relive the golden age of home computing and there is no shortage of

games inspired from that decade. In the Pi Store these include **Abandoned Farmhouse Adventure** and **King's Treasure** - classic text based adventures, **Chocolate Doom** - play all versions of Doom, **NXEngine** - a clone of the "Cave Story" platformer, **Star Flite** - a Star Trek retro game plus an emulator for the **Atari800**.

Other excellent games include **Freeciv** - an empire building strategy game, **Sqrxz 2** and **Sqrxz 3** - platformers, the impressive **Open Arena** - first person shooter, **OpenTTD** - a simulation game based on Transport Tycoon Deluxe and **Iridium Rising** - a 3D space game but currently suffering a "Servers Full" issue.

The first commercial game on the Pi Store was **Storm In A Teacup**. It remains one of my favourite Raspberry Pi games, but unfortunately it was recently removed from the Pi Store. Hopefully it will be back soon.

Apps



By far the largest category in the Pi Store, there is an incredible selection of titles ranging from utilities to emulators to media programs to 'heavy-weight' applications. Examples of the latter are **Asterisk** - a PBX system and the brilliant **LibreOffice** - a Microsoft Office compatible and equivalent office suite.

Four of the six paid applications can be found here. **OMX Player Remote GUI** (US$3.00/£1.80) provides a web UI that can be used on a smart phone or tablet device to control OMX Player running on the Raspberry Pi. Simply enter the given URL in your web browser, specify your media folder and then have full control of playlists, volume and all the usual media controls.

FastDosBox (US$2.37/£1.35) is a fast 386 based PC emulator and is a great way to run 90's era DOS based games. Another similar program on the Pi Store is **RPix86**.

PiCon (US$1.00/£0.60) is a very flexible configuration manager for hardware tweaking. It provides a GUI for overclocking every aspect of the Raspberry Pi, but for me its real strength is with its video configuration. Visually get the perfect monitor setup quickly and easily. No more black bars! Multiple presets can be saved so if you take your Raspberry Pi to different locations (e.g. school, Raspberry Jam, hackspace) you can get the best display configuration quickly without changing your 'home' settings.

Timewaster (US$0.79/£0.49) wastes not only your time, but also your money, if you are stupid enough to buy it - which I did so you won't! It is disappointing that this was even allowed in the Pi Store in the first place.

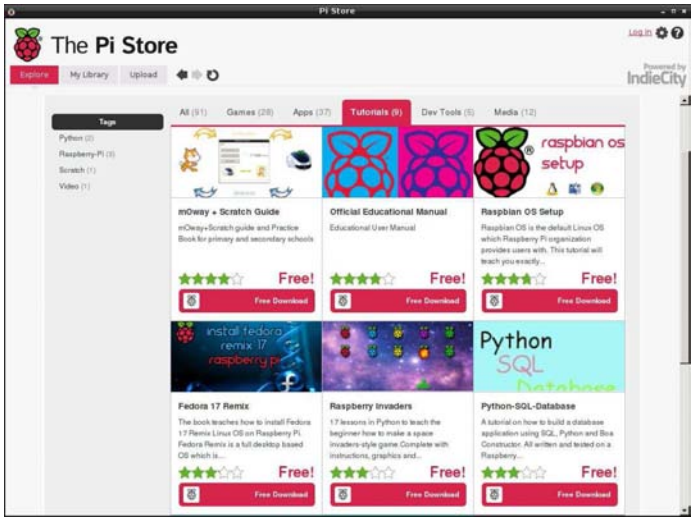
I was not familiar with **XIX Music Player**, but it is pleasantly surprising and is definitely worth downloading. I am using it to listen to internet radio (Absolute Radio from the UK, Hit FM from Belgium, MacJingle Heartbeat from Austria...) while I layout this article using Scribus and all running smoothly on the Raspberry Pi.

Staying with the music theme there are two music creation applications in the Pi Store. **Schism Tracker** is both a MOD file player and a music composition tool. Basic instructions explaining its operation are in issues 2, 12 and 13 of The MagPi.

With **PXDRUM** you will have a lot of fun creating cool beats. You can simultaneously play eight

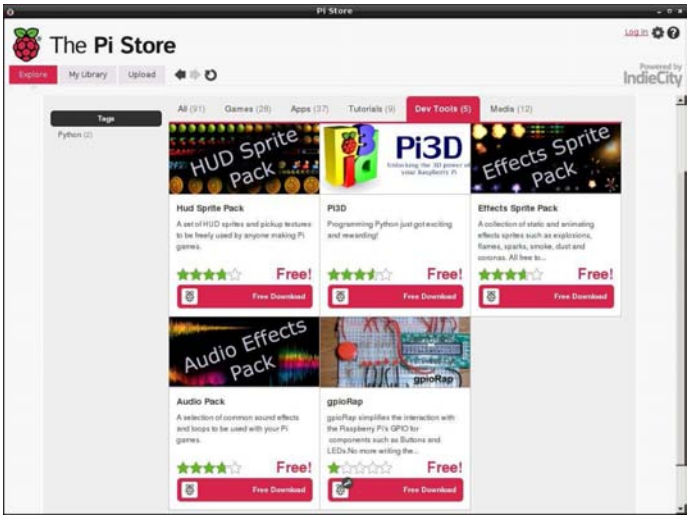
different percussion instruments and change their sound by loading different drum kits. Start with one of the demo songs and get your groove on!

Tutorials



The Tutorials category contains several very useful guides including the **Official Raspberry Pi Educational User Manual**. Other guides include **Python-SQL-Database** - a tutorial on how to create database applications using SQL and Python plus **Raspberry Invaders** - a Python programming course with 17 lessons where you learn how to create a space invaders style game using Python.

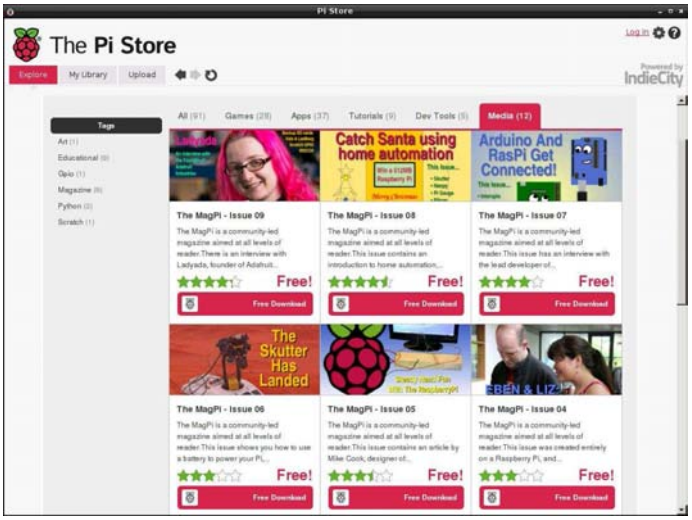
Dev Tools



This is the smallest category in the Pi Store yet it contains useful collections such as **HUD Sprite Pack**, **Effects Sprite Pack** and **Audio Effects Pack** for your Python and Scratch games... and of course all free to use.

There is also the very excellent **Pi3D**, a Python module that simplifies developing 3D worlds and provides access to the power of the Raspberry Pi GPU. In addition to both 3D and 2D rendering, Pi3D can load textures, models, create fractal landscapes and offers vertex and fragment shaders. There are also 19 impressive demos.

Media

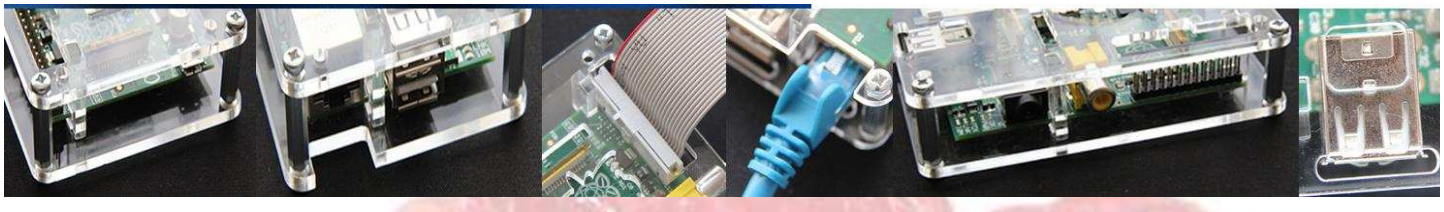


Last, but certainly not least, is the Media category. This is where you will find every issue of **The MagPi** - the first, and in my biased opinion the best, magazine for the Raspberry Pi... and of course it is free! There will typically be a short lag between The MagPi being released at the start of each month and it appearing in the Pi Store. That is because the Pi Store version is the same version that we use for creating the printed version of the magazine so we are doubly thorough with the quality control.

Conclusion

With over 2 million Raspberry Pis sold worldwide and the Pi Store being located on the desktop of Raspbian, it has the potential to be an incredible resource and the "go to" place for Raspberry Pi content. It is easy to upload so why not share your original programs or tutorials with others? The Pi Store is there for your benefit and the benefit of the community, so go use it.

DECEMBER COMPETITION



Once again The MagPi and PC Supplies Limited are proud to announce yet another chance to win some fantastic Raspberry Pi goodies**!**

This month there is one MASSIVE prize!

The winner will receive a new Raspberry Pi 512MB Model B, an exclusive Whiteberry PCSL case, 1A PSU, HDMI cable, 16GB NOOBS memory card, GPIO Cobbler kit, breadboard and jumper wires!

For a chance to take part in this month's competition visit:
http://www.pcslshop.com/info/magpi

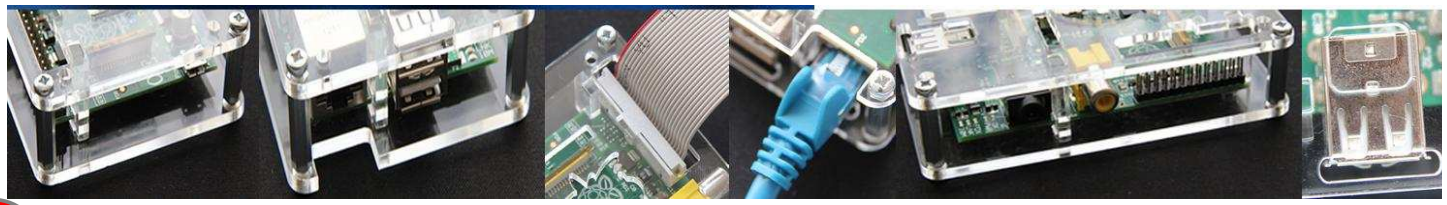
Closing date is 20th December 2013. Winners will be notified in the next issue and by email. Good luck!

To see the large range of PCSL brand Raspberry Pi accessories visit
http://www.pcslshop.com

November's Winner!

The winner of a new 512MB Raspberry Pi Model B plus an exclusive Whiteberry PCSL case, 1A PSU, HDMI cable, 16GB NOOBS memory card, GPIO Cobbler kit, breadboard and jumper wires is **Lena Pellow (Cardiff, UK).**

Congratulations. We will be emailing you soon with details of how to claim your prizes!



The MagPi What's On Guide

Want to keep up to date with all things Raspberry Pi in your area?

Then this section of The MagPi is for you! We aim to list Raspberry Jam events in your area, providing you with a Raspberry Pi calendar for the month ahead.

Are you in charge of running a Raspberry Pi event? Want to publicise it?
Email us at: editor@themagpi.com

Raspberry Pi Bake Off

When: Thursday 12th December 2013, 9.00am until 3.30pm
Where: North Herts College, Monkswood Way, Stevenage, UK

Event for students to attend with their teachers. Workshops on networking and interaction. BYOP (Bring your own Pi). http://setpointbakeoff-es2.eventbrite.co.uk

Winchester Raspberry Jam

When: Thursday 12th December 2013, 2.00pm to 5.00pm
Where: Winchester Science Centre and Planetarium, SO21 1HZ, UK

Teachers are invited to bring along 3 interested students to turn their Raspberry Pi's into Twitter Decks, tweeting from the Raspberry Pi using Python. http://www.eventbrite.co.uk/event/8754828929

Pi and Chips, Computing at School

When: Thursday 12th December 2013, 9.30am until 12.30pm
Where: Nottingham Trent University - Clifton Campus, Clifton Lane, Nottingham, NG11 8NS, UK

Hands-on experience for teachers new to coding control using using the Picaxe and Arduino chip systems in addition to the Raspberry Pi. http://www.eventbrite.co.uk/event/9274928561

Torbay Raspberry Jam

When: Saturday 14th December, 1.00pm until 3.00pm
Where: Paignton Library and Information Centre, Great Western Road, Paignton, TQ4 5AG, UK

The first Torbay Raspberry Jam. Scratch, Python, Minecraft. http://dcglug.drogon.net/torbay-pi-jam.

Raspberry Jam Sweden, in Helsingborg

When: Saturday 4th January 2014, 12.00pm until 4.00pm
Where: Bredgatan 12, at Mindpark (next to Campus), Helsingborg, Sweden

Everybody is welcome free of charge. Sign up for free tickets: http://www.raspberryjam.se. We wll serve hot Raspberry Pie's with cold custard. Bring your own project to show or get help from other Jammers.

Claire Price

MagPi Writer

Good King Wenceslas

SKILL LEVEL : BEGINNER

Sonic Pi was developed by Sam Aaron and is an excellent, easy and creative way to learn programming, especially as Ruby is embedded. It uses the MIDI note numbering system to play sounds and this means you don't have to be a musical genius to make some music! A good MIDI reference table can be found at http://www.midikits.net23.net/midi_analyser/midi_note_numbers_for_octaves.htm

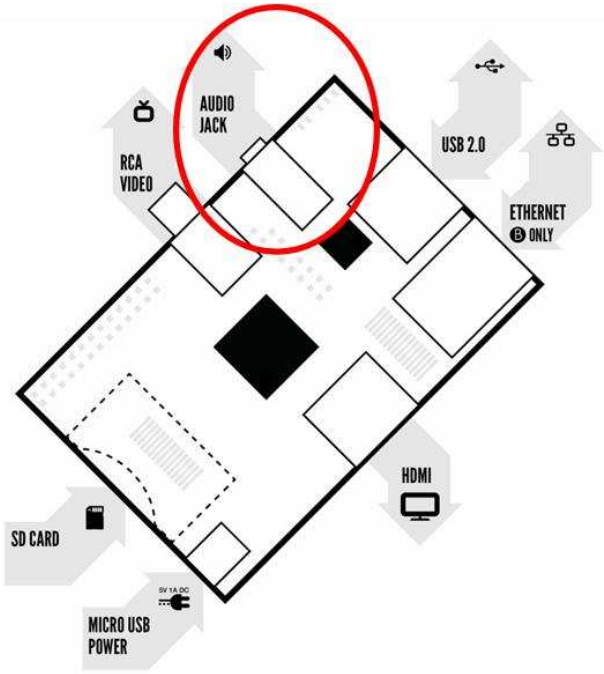
Getting Started

If you are using the latest version of NOOBS or Raspbian Wheezy then Sonic Pi should already be installed. You can check this by looking to see if you can find Sonic Pi under "Programming" in the main menu. However, if you have older versions of these you either need to download the latest image files or you could type in the following into your terminal:

```
sudo apt-get update ; sudo apt-get install sonic-pi
```

It is important you update before downloading Sonic Pi otherwise Sonic Pi won't work.

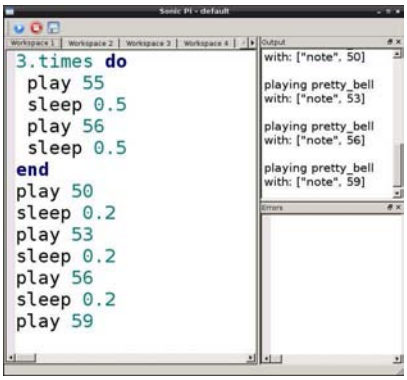
To hear the sound generated, you will need speakers or a pair headphones, if you can't get sound through your monitor/screen, and plug them into the sound jack port on your Raspberry Pi (see Rapberry Pi diagram below).



Now we have everything set up, let's make some music!

Making Sounds

Open up Sonic Pi. You will see 3 panes (see screenshot on next page).



In the large pane type in

```
play 48
```

By typing this we are telling the program you want to play MIDI note 48, which is the equivalent of playing a c on a piano. To hear it, we need to press play. This is the triangle button at the top of the page. We should hear a note and in the output pane (which is found on the top right of the screen) we will see what is happening (i.e. note 48 is playing). If you make a mistake while typing in the code in, don't worry. The bottom right hand pane will show you your mistake. Try typing in

```
ploy 48
```

and see what happens when you press play.

To create a tune we will want to play multiple notes. Let's type in the following:

```
play 48
play 52
play 55
```

Press play. You will notice the three notes are played together. This is great if we want to play a number of notes at the same time, but no good if we want to play separate notes. So to make sure the notes play one after another we need to type `sleep` followed by the number of seconds we want to wait between notes. For instance, if we want a 1 second wait, we would type `sleep 1`. Let's try putting this into what we have just written.

```
play 48
sleep 1
play 52
sleep 0.5
play 55
```

Press play. Can you hear how changing the timings between the notes changes how the melody sounds?

Writing chords

We may want to introduce chords into our tunes. The easiest way to do this is to use the `play_chord` function:

```
play_chord [48,52,55]
```

This is telling the program we want to play notes 48, 52 and 55 at the same time. By adding the sleep function after `play_chord` we can change the timings between notes (see previous section). Try this out by typing:

```
play_chord [48,52,55]
sleep 1
play 52
```

Using loops

If we wanted to repeat this section, we could rewrite everything we had just typed or we could use a loop. This is achieved by using `times do`. By writing a number in front of times do we can get the program to repeat the section that many times. So if we write `2.times do` everything will be repeated twice, if we write `3.times do` everything will be repeated 3 times and so on. Adding end at the bottom of this section tells the program we only want to repeat this section.

```
2.times do
   play 48
   sleep 1
   play 52
   sleep 0.5
end
```

Press play. You should hear all the notes played twice in the loop.

Playing patterns

In our tune, we may have a series of notes requiring the same timings in between, e.g.

```
play 48
sleep 1
play 52
sleep 1
play 55
sleep 1
```

There is nothing wrong with writing the code as in the example, but we could write it another way:

```
play_pattern_timed [48,52,55],[1]
```

This is a great way to write sequences as it is more concise and reduces the chance of error as you do not need to repeatedly type `play` and `sleep`. We can also use this format to shuffle, reverse and sort the notes we want to play.

By adding `.shuffle`, the program will play the notes in a random order:

```
play_pattern_timed [48,52,55].shuffle,[1]
```

Note the `.shuffle` comes after the notes. If it is placed after the sleep command, i.e. [1], then the notes will be played in the exact order we have written them. Try it and see:

```
play_pattern_timed [48,52,55],[1].shuffle
```

We can also reverse the order of the notes we have asked the program to play by using `.reverse`

```
play_pattern_timed [48,52,55].reverse,[1]
```

Again if we put `.reverse` after the [1] the notes will be played in the order they have been typed.

We can also use `.sort` to order the pattern of notes we have written. This works particularly well if we have written a sequence of notes in any sequence and then decide what we really want is for the notes to be played in numerical order. For instance, we might type

```
play_pattern_timed [52,48,55],[1]
```

but we actually want is for the notes to be played 48,52,55. By adding `.sort` this can be achieved:

```
play_pattern_timed [52,48,55].sort,[1]
```

Playing two tunes at once

When writing a tune we may want to have multiple melodies playing at the same time, like when someone is playing a piano with both their right and left hand. To do this we use `in_thread do`.

```
in_thread do
   play 48
   sleep 1
   play 52
   sleep 1
end

in_thread do
 2.times do
    play 55
    sleep 1
 end
end
```

The sections we want to use, we encapsulate in `in_thread do` and end so the program knows where to start and finish. The program will then play these two sections at the same time. We can also use other functions within the `in_thread do` too, in this case `2.times do`.

Changing the synth

`pretty_bell` is the default synth, but this can be changed. There are a number of different of synth sounds to choose from and these are: `dull_bell`, `pretty_bell` (the default

synth), fm, beep and saw_beep. To change the synth, we use with_synth followed by the name of the synth we want to use (in quotation marks, “ “), e.g.

```
with_synth “dull_bell”
```

Anything that follows this command will be played with this synth. So let’s try it out:

```
with_synth “dull_bell”
play 48
```

You can hear the note sounds very different even though the same note is being played. Try out the other synths and see which you prefer.

Now let’s write a tune in Sonic Pi!

Good King Wenceslas

As it is the Christmas season, it seems like a good idea to start by writing a Christmas carol. Good King Wenceslas is a good carol to choose as we can use some of the functions we have discussed.

The first section of Good King Wenceslas is repeated twice so let's start with

```
2.times do
```

A lot of this section is the same pattern, i.e. there is a sequence of notes to be played with the same timings so we can use play_pattern_timed. Remember the indents.

```
   play_pattern_timed [55,55,55,57,55,55],[0.5]
   play 50
   sleep 1
   play_pattern_timed [52,50,52,54],[0.5]
   play 55
   sleep 1
   play 55
   sleep 1
```

Now we need to tell the program to only repeat this section so we type

```
end
```

Now we can finish off the rest of the tune.

```
play_pattern_timed [62,60,59,57,59,57],[0.5]
play 55
sleep 1
play_pattern_timed [52,50,52,54],[0.5]
play_pattern_timed [55,55],[1]
play_pattern_timed [50,50,52,54,55,55],[0.5]
play 57
sleep 1
play_pattern_timed [62,60,59,57],[0.5]
play_pattern_timed [55,60],[1]
play 55
sleep 2
```

So the finished tune should look something like this:

```
2.times do
   play_pattern_timed [55,55,55,57,55,55],[0.5]
   play 50
   sleep 1
   play_pattern_timed [52,50,52,54],[0.5]
   play 55
   sleep 1
   play 55
   sleep 1
end
play_pattern_timed [62,60,59,57,59,57],[0.5]
play 55
sleep 1
play_pattern_timed [52,50,52,54],[0.5]
play_pattern_timed [55,55],[1]
play_pattern_timed [50,50,52,54,55,55],[0.5]
play 57
sleep 1
play_pattern_timed [62,60,59,57],[0.5]
play_pattern_timed [55,60],[1]
play 55
sleep 2
```

Congratulations you have written your first Christmas carol!

For more information on Sonic Pi see http://www.cl.cam.ac.uk/projects/raspberrypi/sonicpi/ and have fun creating your next tune!

Feedback & Question Time

I bought a Raspberry Pi a while ago, and other things got in the way of my plan for playing with it. Now that winter is on the way and the evenings are drawing in, I have time to devote to returning to playing with my Raspberry Pi. I have downloaded and read through some of the back issues of your magazine, and really like the format and contents.

Keep up the good work,
Robin

Great magazine, especially at the price!

As an electronics engineer with a bit of a background in programming, I was amazed when I saw the Raspberry Pi launched at such a low price.

I am now spending all my spare time (which isn't that much) building bits and coding the Raspberry Pi. Takes me back to the 80's, ZX80, Acorn Electron etc.

Your magazine is providing loads of info. for me, as well as ideas and enthusiasm.

Sincere Thanks
Nigel

Thank you for all your hard work on The MagPi magazine. You guys are doing a fantastic job!

Thanks again for this brilliant resource.

Roeland Schumacher

Just wanted to inform you that the article (Pi Vision, Issue 18) looked great! My kids are very proud!

B.J. Rao

I was pointed towards your magazines for help with LEDs and motors (issue 2).

I am a complete noob to all this and your magazine and personal help has helped me greatly. I will now try the Python programs too.

I am going to be making a YouTube video on how to set up everything for complete noobs and will also post a link to your website and issue.

You've been a great help.

Thanks once again!
Chad Murphy

Many thanks for your wonderful magazine. It's without question the 'must have' accessory for the Raspberry Pi.

Martin Hodgson

Merry Christmas and a Happy New Year!

Don't forget our first issue of 2014 will be available online in February.