The MagPi ESSENTIALS

THE CAMERA MODULE GUIDE

TAKE PICTURES AND CAPTURE VIDEO WITH YOUR Raspberry Pi

Curated by **Phil King**

WELCOME TO THE CAMERA MODULE GUIDE

One of the most popular add-ons for the Raspberry Pi, the official Camera Module turns your favourite single-board computer into a powerful digital camera. Launched back in 2013, the original Camera Module was succeeded by the higher-spec v2 in April 2016. Even the tiny Pi Zero now has a camera connector, enabling the creation of even more amazing projects. In this book we'll show you how to get started with the Camera Module, taking photos and videos from the command line and writing Python programs to automate the process. We'll reveal how to create time-lapse and slow-motion videos, before moving on to exciting projects including a Minecraft photo booth, spy camera, and bird box viewer. There are just so many things you can do with a Raspberry Pi and Camera Module!

Phil King
Contributing Editor

FIND US ONLINE raspberrypi.org/magpi

GET IN TOUCH magpi@raspberrypi.org

The MagPi

Available on the App Store

Get it on Google play

EDITORIAL
Head of Publishing: **Russell Barnes**
Contributing Editor: **Phil King**
Contributors: **Dan Aldred, Wesley Archer, Richard Hayler & family, Lorna Lynch, James Singleton, Rob Zwetsloot & the Raspberry Pi Foundation education team**

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

DESIGN
Critical Media: **criticalmedia.co.uk**
Head of Design: **Dougal Matthews**
Designers: **Lee Allen, Mike Kay**

THE MAGPI SUBSCRIPTIONS
Select Publisher Services Ltd
PO Box 6337, Bournemouth
BH1 9EH | **+44 (0)1202 586 848**
magpi.cc/Subs1

recycle
When you have finished with this magazine, please recycle it.

In print, this product is made using paper sourced from sustainable forests and the printer operates an environmental management system which has been assessed as conforming to ISO 14001.

This book is published by Raspberry Pi (Trading) Ltd., 30 Station Road, Cambridge, CB1 2JH. The publisher, editor, and contributors accept no responsibility in respect of any omissions or errors relating to goods, products or services referred to or advertised in this product. Except where otherwise noted, content in this magazine is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported (CC BY-NC-SA 3.0). ISBN: 978-1-912047-03-1

The MagPi ESSENTIALS

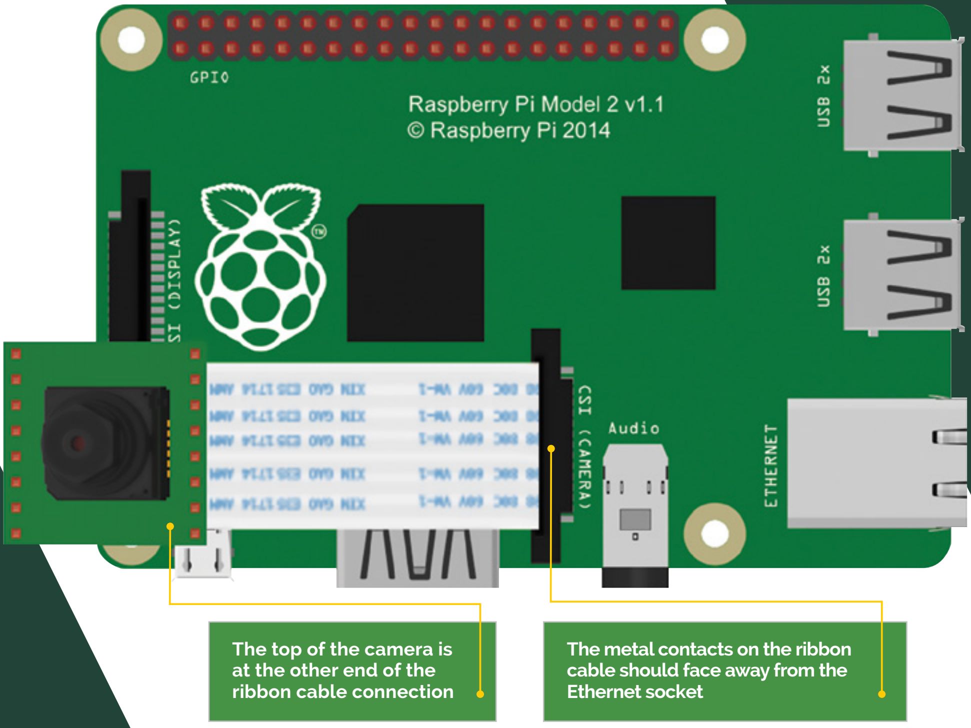
CONTENTS

[PHIL KING]

Phil King is a Raspberry Pi enthusiast and regular contributor to *The MagPi* magazine. Growing up in the 'golden era' of 8-bit computers in the 1980s, he leapt at the chance to write about them in magazines such as CRASH and ZZAP!64. When consoles took over the video games world, he missed the opportunity to program... until the Raspberry Pi came along. Phil is now an avid coder, electronics dabbler, and photographer. He loves to work on projects with his seven-year-old son – which may explain the photos of Playmobil and Lego you'll find in this book!

[CHAPTER ONE] GETTING STARTED

Introducing the Camera Module – find out how to connect it, enable it, and take your first shots



The top of the camera is at the other end of the ribbon cable connection

The metal contacts on the ribbon cable should face away from the Ethernet socket

In this chapter, we show you how to connect the Camera Module to your Raspberry Pi using the supplied ribbon cable, then enable it in Raspbian, before entering some commands in a terminal window to start shooting photos and video. Let's get started…

>STEP-01

Connecting the camera

With the Pi switched off, locate its camera port. On most models, it's the one furthest away from the micro USB power connector, labelled 'CAMERA' or 'CSI'; on a Pi Zero 1.3 or W, it's at the end next to the power port. Take hold of both ends of its plastic slider and pull it away from the Pi gently but firmly; it will move up a short distance, opening up the connector. Insert the ribbon cable, with its blue side (or white tab on a Pi Zero cable) facing the plastic slider. Push the slider back down, putting pressure on both sides, so that it gently clicks into place.

>STEP-02

Camera software

Plug the Raspberry Pi back in and turn it on. Once it has booted into the desktop, click on the Menu (top left) and go down to Preferences. Here you'll find the Raspberry Pi Configuration menu, which you should now click on. On the tab called Interfaces, you'll find an option to enable the camera; if this isn't set to Enabled, do so now. Raspbian doesn't have the camera enabled by default, so this is required. Now reboot your Pi.

>STEP-03

First shots

Point your camera at something interesting, then open a terminal window and type in the following:

```
raspistill -o firstpic.jpg
```

You'll see a red light on the Camera Module, followed by an image of whatever the camera is pointing at appearing on the screen for a moment. If it's you, you can use the screen to arrange your best smile during the five-second pause before the picture is taken. Once snapped, the image can be found in the home directory: **/home/pi**.

>STEP-04

Find the image

You can open the picture and view it from the File Manager, but if you don't have a Pi 2 or Pi 3, you may want to avoid the unnecessary overhead of running that. Just enter this into the terminal window:

```
gpicview firstpic.jpg
```

If it looks rather blurry, check that you remembered to peel the protective plastic from the Camera Module's lens!

>STEP-05

More advanced commands

The **raspistill** command has a list of options so long that it borders on the intimidating. Have no fear, though: you won't need to learn them all, but there are a few that might be useful to you, such as:

```
raspistill -t 15000 -o newpic.jpg
```

The **-t** option changes the delay before the picture is taken, from the default five seconds to whatever time you give it in milliseconds – in this case, you have a full 15 seconds to get your shot arranged perfectly after you press **ENTER**. You can explore more camera options in the next chapter, or by referring to chapter 14.

>STEP-06

A quick fix

One of the problems with a camera at the end of a ribbon cable is getting it positioned properly. You may end up with the camera upside down or slightly askew. Upside-down Camera Modules can be commanded to flip the picture the right way up with **--vflip**, or **-vf** for short. **--hflip** (or **-hf**) handles horizontal flipping, should you need a mirror image. And if your camera is lying on its side, use **--rotation**, or **-rot**, followed by the number of degrees: 90 or 270.

[SHOOTING VIDEO]

For shooting video, **raspivid** is what you need. It can record up to 1080p video at 30fps, a fast enough frame rate for cinema, and 720p at 60fps if you want something smoother. You can do this with:

```
raspivid -t 10000 -o testvideo.h264
```

This records a ten-second video (10,000 milliseconds) at the default 1920 × 1080 resolution. You can also shoot slow-mo video at 640 × 480 by using:

```
raspivid -w 640 -h 480
-fps 90 -t 10000 -o
test90fps.h264
```

Use **omxplayer** in the command line to play the videos back!

The MagPi ESSENTIALS

[CHAPTER TWO] PRECISE CAMERA CONTROL

Use command-line switches to access numerous camera options and effects

So, you've connected your Camera Module to the Raspberry Pi and learned how to take still photos and shoot videos from the command line. Now let's explore the **raspistill** and **raspivideo** commands further, including the many switches and options available. We'll also take a look at the **raspistillyuv** command, which sends its unencoded YUV or RGB output directly from the camera component to a file.

>STEP-01

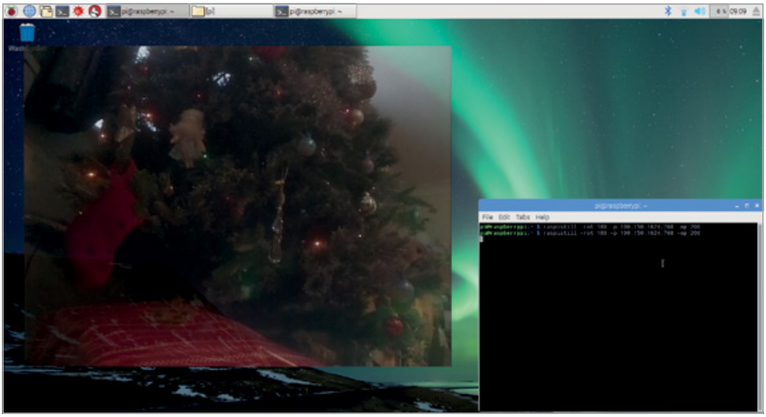
Preview mode

When taking stills or shooting video, one of the first things you might want to alter is the preview window that appears by default on the screen. First of all, if it's upside-down, just add **-rot 180** to your **raspistill** or **raspivid** command to rotate it. As mentioned in chapter 1, **-hf** and **-vf** will flip the image horizontally and/or vertically.

Using the **-p** switch, you can set the window's on-screen position, along with its height and width. The **-p** switch takes four parameters: x coordinate, y coordinate, width, and height. So, for example:

```
raspistill -o image.jpg -p 100,100,300,200
```

...would place the preview window's top-left corner at coordinate (100,100), with a width of 300 pixels and height of 200 pixels.



```
pi@raspberrypi:~ $ raspistill -rot 180 -p 100,150,1024,768 -op 200
pi@raspberrypi:~ $ raspistill -rot 180 -p 100,150,1024,768 -op 200
```

The preview can be resized and positioned manually, and can also have its opacity adjusted

Note that if you only want to see a preview without taking a shot, you can simply omit the **-o image.jpg** part. The **-t** switch sets the duration of the preview: to exit at any point, just press **CTRL+C**.

If you want a full-screen preview, this is easily achieved using the **-f** switch. The **-op** switch can be used to adjust the preview's opacity, from 0 (invisible) to 255 (solid). If you want to disable the preview window completely, use the **-n** switch.

>STEP-02

Camera control options

Like most dedicated digital cameras, the Camera Module offers a range of options to adjust aspects such as brightness (**-br**, from 1 to 100), contrast (**-co**, -100 to 100), sharpness (**-sh**, -100 to 100), saturation (**-sa**, -100 to 100), ISO (**-ISO**, 100 to 800), and EV compensation (**-ev**, -10 to 10).

In addition, there are numerous options for exposure mode for shooting in certain scenarios, akin to the 'scenes' found on most digital cameras. Just use the **-ex** switch followed by one of the following terms: **auto**, **night**, **nightpreview**, **backlight**, **spotlight**, **sports**, **snow**, **beach**, **verylong** (long exposure), **fixedfps** (for video only), **antishake**, or **fireworks**.

Similarly, automatic white balance can be adjusted by following the **-awb** switch with one of the following: **off**, **auto**, **sun**, **cloud**, **shade**, **tungsten**, **fluorescent**, **incandescent**, **flash**, or **horizon**.

You can set the shutter speed in microseconds with the **-ss** switch; the upper limit depends on the exposure mode and other settings. The metering mode – used for preview and capture – can be set with **-mm** to one of the following: **average**, **spot**, **backlit**, or **matrix**.

There's also the option of restricting the region of interest to only part of the sensor, using **-roi** with parameters for x and y coordinates (from top left), width, and height. For example, to set a ROI halfway across and down the sensor, with quarter-size width and height, you'd use: **-roi 0.5,0.5,0.25,0.25**.

>STEP-03

Keypress mode

If you'd like to take a still photo at an exact time, rather than having to wait for the **-t** switch delay time to elapse, keypress mode is your friend.

Just add the **-k** switch to your **raspistill** command, then press the **ENTER** key to take the shot: it acts like a shutter button. To exit the procedure, press **X** followed by **ENTER**.

By adding **%04d** to the end of your file name in the command, you can save every shot you have taken before aborting:

```
raspistill -o keypress%04d.jpg -k
```

Each shot will have a four-digit sequential number added to its file name, so you'll get **keypress0000.jpg**, **keypress0001.jpg**, **keypress0002.jpg** etc. This is a useful technique for time-lapses using the **-tl** switch, too: see chapter 3 for more details.

>STEP-04

Image effects

A whole bunch of effects can be added to the camera in real-time, shown in the preview window. This is achieved by using the **-ifx** switch followed by one of the following terms: **none**, **negative**,

A multitude of real-time effects may be added to images, including emboss, as shown here

The posterise effect is shown here; just use `-ifx posterise` in your command

`solarise`, **`posterise`**, **`sketch`**, **`denoise`**, **`emboss`**, **`oilpaint`**, **`hatch`**, **`gpen`** (graphite sketch effect), **`pastel`**, **`watercolour`**, **`film`**, **`blur`**, **`saturation`** (adjust colour saturation of the image), **`colorswap`**, **`washedout`**, **`colorpoint`**, **`colorbalance`**, or **`cartoon`**.

If you'd like to take monochrome images, you can use the **`-cfx`** (colour effect) switch to achieve this, using the following setting: **`-cfx 128:128`**.

To increase contrast between dark and light areas using DRC (dynamic range compression), use the **`-drc`** switch to turn it on/off (it's off by default).

>STEP-05

Still options

Now let's take a look at some options that are specific to the **`raspistill`** command. As already mentioned, we use **`-o`** followed by a file name to output to a file, and the **-t** switch sets the shutter delay in milliseconds. For example, to save a photo taken after two seconds, use:

```
raspistill -t 2000 -o image.jpg
```

You can set the width and height of the image with **-w** and **-h**, each followed by a value – up to 2592 and 1944 respectively on the original Camera Module, or 3280 and 2464 on the v2.

You can also set the quality of the JPEG image, using **-q**, from 0 to 100 – the latter is almost completely uncompressed. Alternatively, to save it as a lossless PNG (slower than using JPG), use **-e** (encoding) followed by **png**:

```
raspistill -o image.png -e png
```

For a full list of **raspistill** options, see chapter 14. The **raspistillyuv** command works in a similar fashion and offers most of the same options, apart from adding EXIF tags, but sends its YUV or RGB output directly from the camera component to file. To use RGB, add the **-rgb** switch.

>STEP-06

Shooting video

The **raspivid** command is used to shoot video. In this case, the **-t** switch sets the duration in milliseconds. The bitrate is set using **-b**, with a maximum of 25Mbps (**-b 25000000**), while **-fps** sets the frame rate – up to 90fps depending on the resolution, which can be set manually using the **-w** and **-h** switches. For example, to shoot five seconds of video at the default 1080p (1920 × 1080), with a bitrate of 15Mbps and frame rate of 30fps, use:

```
raspivid -t 5000 -b 15000000 -fps 30 -o video.h264
```

See chapter 4 for information on how to shoot slow-motion footage. Many other video options are available, including time delays, keypress mode, and segmenting a stream into multiple files. For full details, see chapter 14.

[CHAPTER THREE] TIME-LAPSE PHOTOGRAPHY

Make a device to capture photographs at regular intervals, then turn these images into a video

Image courtesy of NASA

Time-lapse photography reveals exciting things about the world which you wouldn't otherwise be able see. These are things that happen too slowly for us to perceive: bread rising and plants growing; the clouds, sun, moon, and stars crossing the sky; shadows moving across the land. In this chapter, we'll be making a Raspbian-based device that lets you watch things that are too slow to observe with the naked eye. To do this, we will capture lots of still photographs and combine these frames into a video with FFmpeg/libav, which can then be accessed via a web browser.

>STEP-01

Connect the Camera Module

First, with the Raspberry Pi turned off, connect the Camera Module to the Pi with the included ribbon cable. As mentioned in chapter 1, you need to locate the correct camera socket on the Raspberry Pi, labelled 'CAMERA' or 'CSI'. Carefully lift up its plastic slider and pull it away from

Another amazing example of what's possible with time-lapse photography

the Pi gently but firmly; it will move up a short distance, opening up the connector. Insert the ribbon cable into the socket, with the blue side – or white tab on a Pi Zero camera cable – facing the plastic slider (and the metal contacts facing the other way). Finally, hold the ribbon cable in position and push the slider back down to clamp the cable firmly in place.

>STEP-02

Enable and test the camera

Power the Raspberry Pi up. You now have a choice: boot to the command line, open a terminal window, or establish a secure shell (SSH) connection (to access it from a remote computer). Enable the camera by running this command from a terminal window to launch the Raspberry Pi configuration tool:

```
sudo raspi-config
```

Then select the 'Enable Camera' option. You can test the camera by running the following command:

[OTHER VIDEO FORMATS]

WebM is an open video format that can be displayed directly in most browsers. However, other video formats are available.

```
raspistill -o testimage.jpg
```

If you are using an original v1 Camera Module, its LED should light up for five seconds and a JPEG image will be saved to the current directory. If the camera is mounted upside down, you can use the rotate command-line switch (**-rot 180**) to account for this.

>STEP-03

Install and configure software

Install a web server to access your images remotely. Run this command to install Apache:

```
sudo apt-get install apache2
```

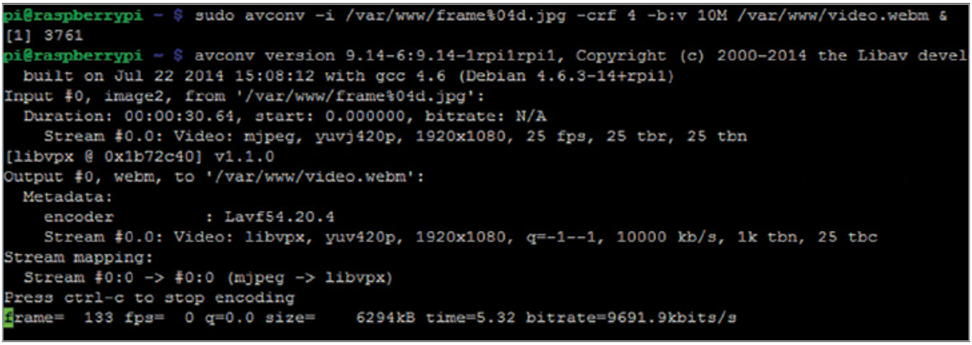
Remove the default page to see the contents of the directory:

```
sudo rm /var/www/index.html
```

Visit the IP address of your Pi (e.g. **http://192.168.1.45** – you can find this by using **ifconfig**) and you should see an empty directory listing. If you run the following command and refresh the page, you should see an image file listed. You run this as a superuser so you can write to the directory.

```
sudo raspistill -o /var/www/testimage.jpg
```

Click on the file link and you'll see the image in your browser.



```
pi@raspberrypi ~ $ sudo avconv -i /var/www/frame%04d.jpg -crf 4 -b:v 10M /var/www/video.webm &
[1] 3761
pi@raspberrypi ~ $ avconv version 9.14-6:9.14-1rpi1rpi1, Copyright (c) 2000-2014 the Libav devel
  built on Jul 22 2014 15:08:12 with gcc 4.6 (Debian 4.6.3-14+rpi1)
Input #0, image2, from '/var/www/frame%04d.jpg':
  Duration: 00:00:30.64, start: 0.000000, bitrate: N/A
    Stream #0.0: Video: mjpeg, yuvj420p, 1920x1080, 25 fps, 25 tbr, 25 tbn
[libvpx @ 0x1b72c40] v1.1.0
Output #0, webm, to '/var/www/video.webm':
  Metadata:
    encoder         : Lavf54.20.4
    Stream #0.0: Video: libvpx, yuv420p, 1920x1080, q=-1--1, 10000 kb/s, 1k tbn, 25 tbc
Stream mapping:
  Stream #0:0 -> #0:0 (mjpeg -> libvpx)
Press ctrl-c to stop encoding
frame=  133 fps=  0 q=0.0 size=    6294kB time=5.32 bitrate=9691.9kbits/s
```

Shell running the rendering process on the Raspberry Pi. This will take some time, so you may prefer to use a faster machine

Some bread dough ready to prove. Watch it rise in your video. Be careful not to move the bowl or camera during filming

>STEP-04

Capture the images

Set up your scene and check the positioning of the camera.

```
sudo raspistill -w 1920 -h 1080 -o /var/www/
testimageFullHD.jpg
```

The width and height have been changed to capture a smaller image in 16:9 aspect ratio. This makes things easier later. The top and bottom are cropped, so make sure that your subject is in frame. Run this to start the capture:

```
sudo raspistill -w 1920 -h 1080 -t 10800000 -tl 10000 -o
/var/www/frame%04d.jpg &
```

This takes a photograph every ten seconds (10,000 milliseconds) for three hours (10,800,000 milliseconds). The ampersand (**&**) at the end runs the process in the background.

>STEP-05

Prepare to make the video

You can render the video on the Raspberry Pi, but it'll be very slow. A better way is to transfer the files to a more powerful computer. Whichever method you decide to use, you will need to install the tools on the rendering machine; for the Pi, enter:

```
sudo apt-get install libav-tools
```

This installs a fork of FFmpeg, but you can also use the original FFmpeg. To copy the images to a remote machine, you can download them from the web server using wget or curl. For example:

```
wget -r -A jpg http://192.168.1.45
```

Or if you don't have wget…

```
curl http://192.168.1.45/frame [0001-0766].jpg -O
```

Change the IP address and numbers accordingly.

>STEP-06

Make the video

The final step is to make the video. Run this command to start the rendering process:

```
sudo avconv -i /var/www/frame%04d.jpg -crf 4 -b:v 10M /
var/www/video.webm &
```

When the rendering process has finished, you'll be able to view the video in your browser. The default frame rate is 25fps. This compresses three hours of images taken at ten-second intervals to about 40 seconds of video. You can adjust this with the **`-framerate`** command-line option. The bitrate (**`-b`**) has been set high, and the Constant Rate Factor (**`-crf`**) has been kept low, to produce a good-quality video.

[MAKE AN ANIMATED GIF]

Instead of video, make an animated GIF with ImageMagick. Use smaller images, captured less frequently.

```
sudo
convert
/var/www/
frame*.jpg
/var/www/
anim.gif &
```

[CHAPTER FOUR]

HIGH-SPEED PHOTOGRAPHY

All you need to make dazzling slow-motion clips of exciting events is your Pi and the Camera Module!

At first glance it seems counter-intuitive, but in order to create a smooth slow-motion movie, you need a high-speed camera. Essentially, a movie is just a collection of still photos, or frames, all played one after the other at a speed that matches the original action. A slow-motion clip is produced by recording more frames than are normally needed and then playing them back at a reduced speed. Normal film is typically recorded at 24 frames per second (fps), with video frame rates varying between 25 and 29fps depending on which format/region is involved. So if you record at 50fps and play back at 25fps, the action will appear to be taking place at half the original speed. It's actually a little more complicated than that with the use of interlaced frames, but you don't really need to consider them here.

Clips can now be recorded at up to 90fps

The original software for the Camera Module was limited in terms of the frame rates it could cope with, but a subsequent update added new functionality so that clips can now be recorded at up to 90fps. There is one slight limitation: the high frame rates are achieved by combining pixels from the camera sensor, so you have to sacrifice resolution. So, although the Camera Module can record at a resolution of 2592 × 1944 (3280 × 2464 on v2 camera), a high-speed mode of 90fps is only possible at 640 × 480 (and 1280 × 720 on v2). This is still good enough to capture decent-quality images, though, and it's perfect for sharing online.

A quick way of getting started is to pick some everyday objects and record them in motion. How about a dropped egg hitting a table top? A pull-back toy car crashing through some Lego blocks? Or even the old favourite of a water balloon bursting? It's best to do the last one outside!

Pick some everyday objects and record them in motion

Once you've chosen your subject, you'll need a way of holding and angling the camera, and some way of lighting the scene. Neither needs to be sophisticated: a normal desk lamp works fine for extra illumination indoors, while a 'helping-hand' work aid is brilliant for keeping the camera stable at tricky angles. You might also want to invest in a longer cable for the camera. You can get a 30 cm ribbon cable for less than £2 or if you want to go even longer, a set of special adaptors allows you to extend using a standard HDMI cable.

Avoid unwanted reflections, and fine-tune your video specifications

If you are using a v1 Camera Module, its red LED will illuminate when recording is taking place. This can cause some undesirable reflections if the camera is positioned close to an object (e.g. a wall of Lego bricks). You can just block the light from the LED off with a blob of modelling clay, or you can turn it off completely by adding the line **`disable_camera_led=1`** to your **/boot/config.txt** file.

The command for capturing video with the Raspberry Pi camera is **`raspivid`**, and this is best run from a terminal window. There are a number of command options that you need to specify:

`-fps` sets the frame rate.

`-w` and **`-h`** specify the width and height of the frames. For the fastest frame rates, set this to 640 and 480 respectively.

`-t` allows you to set how long to record for. If you're working by yourself, the easiest way to avoid missing any of the action is to begin filming for a predefined period, giving yourself plenty of time to start things off manually.

`-o` specifies the file name to use for the saved movie.

`-n` disables preview mode.

So, putting all of that together, the following commands would capture a five-second clip at 60fps and save the resulting movie in the file **test.h264**:

```
raspivid -n -w 640 -h 480 -fps 60 -t 5000 -o
test.h264
```

Right: now that you've recorded your movie clip, how can you play it back? One easy way is to use the free VLC player, which you can install with the following command:

```
sudo apt-get install vlc
```

The Pi version has some handy features which can be accessed by checking the ‘Advanced Controls’ option under the View menu. These include the extremely useful ‘Frame by Frame’ button. You can also alter the playback speed to slow things down even further.

To extend the project, how about connecting a break-beam IR sensor pair via the GPIO pins and using these to trigger the camera recording? The Python Picamera library provides full access to the camera’s functions and could be used with your code.

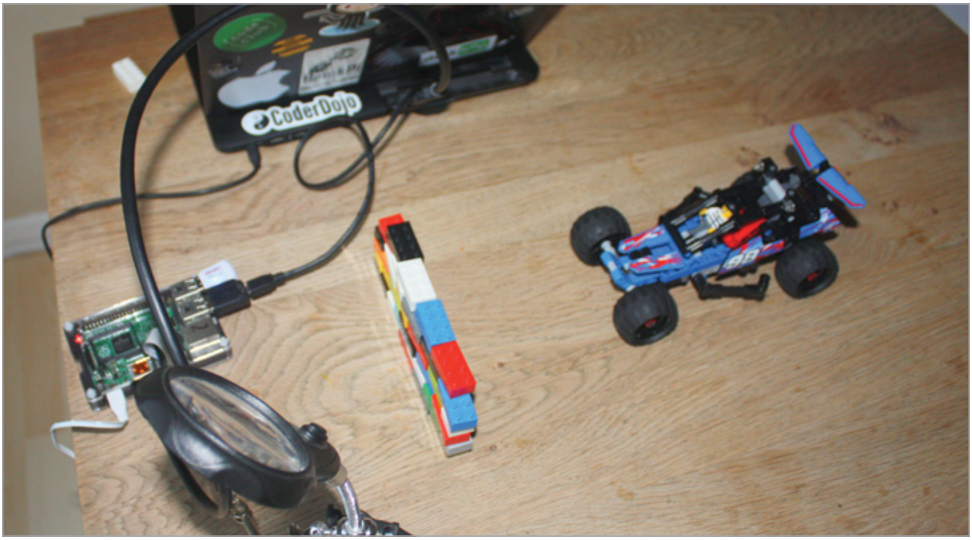
CAPTURING THE CLIP

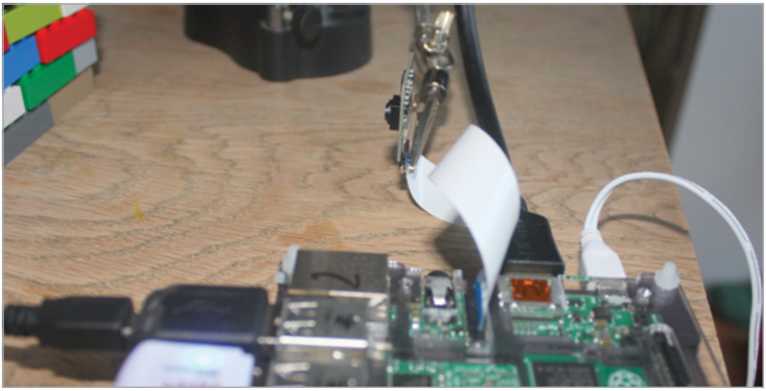
>STEP-01

Lights

Get your scene lined up and test how it looks by using the camera preview mode for five seconds:

```
raspistill -w 640 -h 480 -t 5000
```





>STEP-02

Camera

Type the command, ready for execution (but don’t press **ENTER** yet):

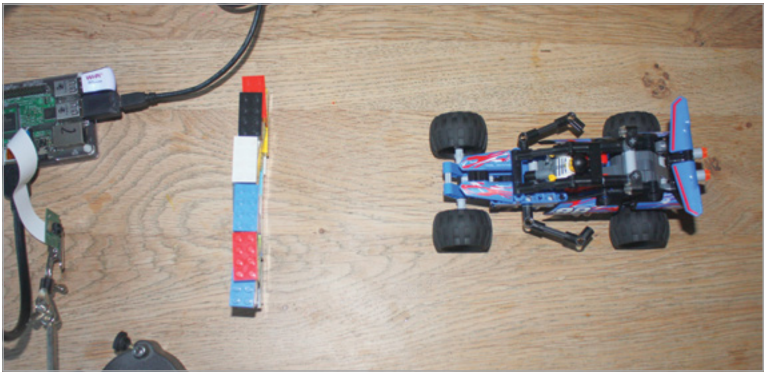
```
raspivid -w 640 -h 480 -fps 90 -t 7000 -o myvid1.h264
```

Once triggered, this will capture a seven-second clip.

>STEP-03

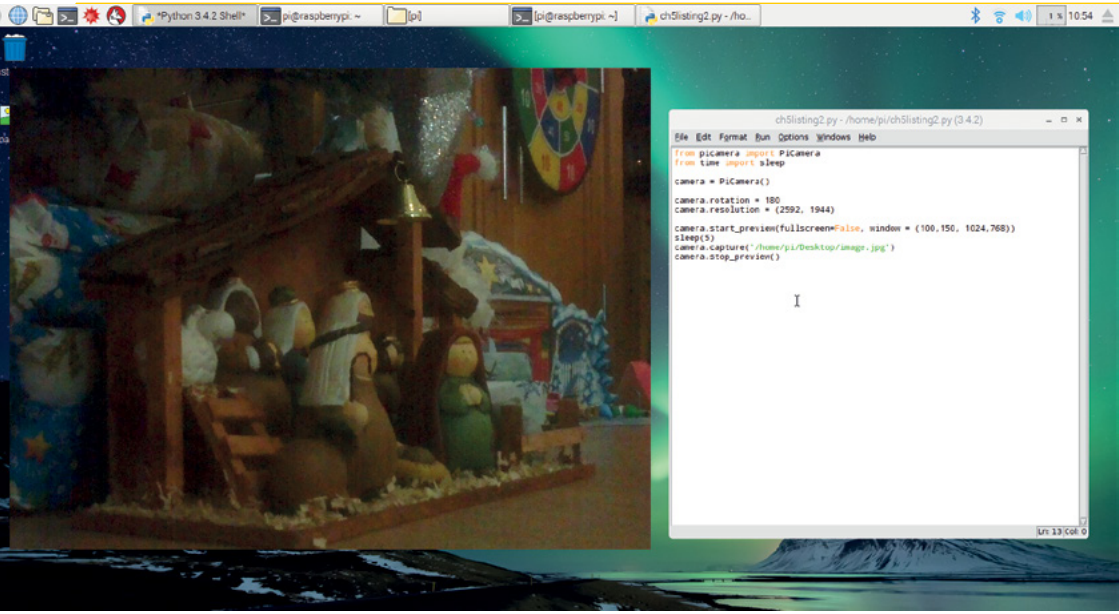
Action

When everything is ready, hit **ENTER** and then release the car/drop the egg/burst the balloon. You’ll have footage before and after the event, which can be trimmed with some post-production editing.



[CHAPTER FIVE] CONTROL THE CAMERA FROM PYTHON

Use the Picamera library to access the camera in Python programs



```
from picamera import PiCamera
from time import sleep

camera = PiCamera()

camera.rotation = 180
camera.resolution = (2592, 1944)

camera.start_preview(fullscreen=False, window = (100,150, 1024,768))
sleep(5)
camera.capture('/home/pi/Desktop/image.jpg')
camera.stop_preview()
```

The camera preview can be resized and positioned to your liking

So far, we've looked at using the Camera Module from the command line. This is all very well and good, but what if you want to control it from a Python program? This is where the Picamera library comes in, enabling you to access all the Camera Module's features in Python. In this chapter, we'll take a look at how to use it to take stills, shoot videos, alter settings, and add effects.

>STEP-01

Getting started

Since the Picamera library doesn't come pre-installed in Raspbian, you'll need to install it manually. In a terminal window, enter:

```
sudo apt-get update
sudo apt-get install python-picamera python3-picamera
```

With your Camera Module already connected and enabled in Raspberry Pi Configuration, open Programming > Python 3 (IDLE) from the Raspbian desktop menu. Create a new file by clicking File > New file. Save it with File > Save, naming it **ch5listing1.py**. Note: Never name a file picamera.py, as this is the file name for the Picamera library itself!

Now enter the code from **ch5listing1.py**. Save it with **CTRL+S** and run with **F5**. The full-screen camera preview should be shown for ten seconds, and then close. Move the camera around to preview what the camera sees. Note that you can only see the preview on a monitor connected to the Pi, not by using a remote access method like SSH or VNC.

If the preview appears upside-down, add the line **`camera.rotation = 180`** just above **`camera.start_preview()`**. Other possible rotation values are 90 and 270.

You can alter the transparency level of the preview by entering an alpha value – from 0 to 255 – within the latter command's brackets. For example: **`camera.start_preview(alpha=200)`**.

It's also possible to change the position and size of the preview. For example, to place its top corner 100 pixels right and 150 down, and resize it to 1024 × 768:

```
camera.start_preview(fullscreen=False, window = (100,150,1024,768))
```

>STEP-02

Take a photo

Now let's take a still photo. We can do this by adding the line:

```
camera.capture('/home/pi/Desktop/image.jpg')
```

...just after the **`sleep`** in our code, so it looks like **ch5listing2.py**. Run the code and after a preview of five seconds (as set by **`sleep`**), it'll capture a photo as **image.jpg**. You may the preview adjust to a different resolution momentarily as the picture is taken. In this example, the resulting image file will appear on the desktop; double-click its icon to open it.

You can alter the file name and directory path in the code, along with the **`sleep`** time. Remember, though, that it should be at least five seconds, to give the camera sensor enough time to adjust its light levels.

>STEP-03

Make a loop

The great thing about using Python with the Picamera library is that it makes it easy to use a loop to take a sequence of photos. In Python 3 (IDLE), create a new file and enter the code from **ch5listing3.py**.

After initiating the camera preview, we add a **for** loop with a range of 5, so it will run five times to take five photos. The **sleep** command sets the time between shots, captured using the line:

```
camera.capture('/home/pi/Desktop/image%s.jpg' % i)
```

Here, the **%s** token is replaced by whatever we add after the **%** following the file name – in this case, the variable **i** set by our **for** loop. Note that i will range from 0 to 4, so the images will be saved as **image0.jpg**, **image1.jpg**, and so on. Once they're all taken, the preview will close. In this example, you'll see the five files on your desktop; double-click to open them.

You can also use a **for** loop to alter camera setting levels such as brightness over time. For more details, see step 4.

>STEP-04

Control camera settings

Brightness is just one of numerous settings available for the Camera Module. Here's a list of the main options, along with their default values (and ranges where applicable):

```
camera.brightness = 50 (0 to 100)
camera.sharpness = 0 (-100 to 100)
camera.contrast = 0 (-100 to 100)
camera.saturation = 0 (-100 to 100)
camera.iso = 0 (automatic) (100 to 800)
camera.exposure_compensation = 0 (-25 to 25)
camera.exposure_mode = 'auto'
camera.meter_mode = 'average'
camera.awb_mode = 'auto'
camera.rotation = 0
camera.hflip = False
camera.vflip = False
camera.crop = (0.0, 0.0, 1.0, 1.0)
```

The resolution of the capture is also configurable. For example:

```
camera.resolution = (1024, 768)
```

By default, it's set to the resolution of your monitor, but the maximum resolution for photos is 3280 × 2464 (v2) or 2592 × 1944 (v1). Note that you may need to increase **gpu_mem** in **/boot/config.txt** to achieve full resolution operation with the v2 Camera Module.

>STEP-05

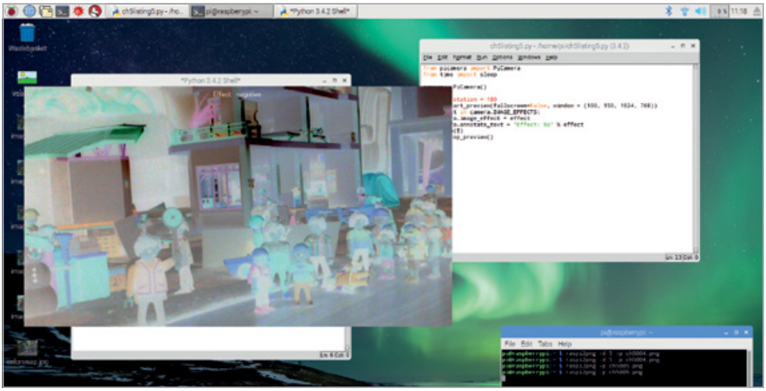
Add image effects

Just as when you are using the command line, a wide range of effects can be added to the camera in real-time, shown in the preview window. The **camera.image_effect** command is used to apply a particular image effect. The options are: **none** (the default), **negative**, **solarize**, **sketch**, **denoise**, **emboss**, **oilpaint**, **hatch**, **gpen** (graphite sketch effect), **pastel**, **watercolor**, **film**, **blur**, **saturation**, **colorswap**, **washedout**, **posterise**, **colorpoint**, **colorbalance**, **cartoon**, **deinterlace1**, and **deinterlace2**.

For instance, to take an image with a colourswap effect, enter the code from **ch5listing4.py** and run it.

You can also run the code from **ch5listing5.py** to loop through the various image effects in a preview. Note that this uses the **camera.annotate_text** command to add a text message to the preview; this can also be applied to captured images (when using the sensor's full field of view).

A variety of image effects are available; here we've used colorswap to alter the colours



```
from picamera import PiCamera
from time import sleep

PiCamera()

otation = 180
art_preview(fullscreen=False, window = (100, 150, 1024, 768))
t in camera.IMAGE_EFFECTS:
a.image_effect = effect
a.annotate_text = "Effect: %s" % effect
(5)
op_preview()
```

Using the ch5listing5.py code, you can view a loop of all the effects on offer

For more details on these effects and other settings, see chapter 14 or the official Picamera documentation at **picamera.readthedocs.io**.

>STEP-06

Shoot a video

To shoot video, we replace the **`camera.capture()`** command with **`camera.start_recording()`**, and use **`camera.stop_recording()`** to stop. Enter the example code from **ch5listing6.py**.

When you run the code, it records ten seconds of video before closing the preview. To play the resulting file, open a terminal window from the desktop and enter:

```
omxplayer video.h264
```

Note that it may well play faster than the original frame rate. It's possible to convert videos to MP4 format and adjust the frame rate using the MP4Box utility (installed with **`sudo apt-get install gpac`**), like so:

```
MP4Box -add video.h264:fps=30 video.mp4
```

All of the image effects and most of the camera settings can be applied while shooting video. You can also turn on video stabilisation, which compensates for camera motion, by adding the following line to your Python program:

```
camera.video_stabilization = True
```

Language

>PYTHON 3

DOWNLOAD:
magpi.cc/2pWtY9l

ch5listing1.py

```python
from picamera import PiCamera
from time import sleep

camera = PiCamera()

camera.start_preview()
sleep(10)
camera.stop_preview()
```

ch5listing2.py

```python
from picamera import PiCamera
from time import sleep

camera = PiCamera()

camera.start_preview()
sleep(5)
camera.capture('/home/pi/Desktop/image.jpg')
camera.stop_preview()
```

ch5listing3.py

```python
from picamera import PiCamera
from time import sleep

camera = PiCamera()

camera.start_preview()
for i in range(5):
    sleep(5)
    camera.capture('/home/pi/Desktop/image%s.jpg' % i)
camera.stop_preview()
```

ch5listing4.py

```
from picamera import PiCamera
from time import sleep

camera = PiCamera()

camera.start_preview()
camera.image_effect = 'colorswap'
sleep(5)
camera.capture('/home/pi/Desktop/colorswap.jpg')
camera.stop_preview()
```

ch5listing5.py

```
from picamera import PiCamera
from time import sleep

camera = PiCamera()

camera.start_preview()
for effect in camera.IMAGE_EFFECTS:
    camera.image_effect = effect
    camera.annotate_text = "Effect: %s" % effect
    sleep(5)
camera.stop_preview()
```

ch5listing6.py

```
from picamera import PiCamera
from time import sleep

camera = PiCamera()

camera.start_preview()
camera.start_recording('/home/pi/video.h264')
sleep(10)
camera.stop_recording()
camera.stop_preview()
```

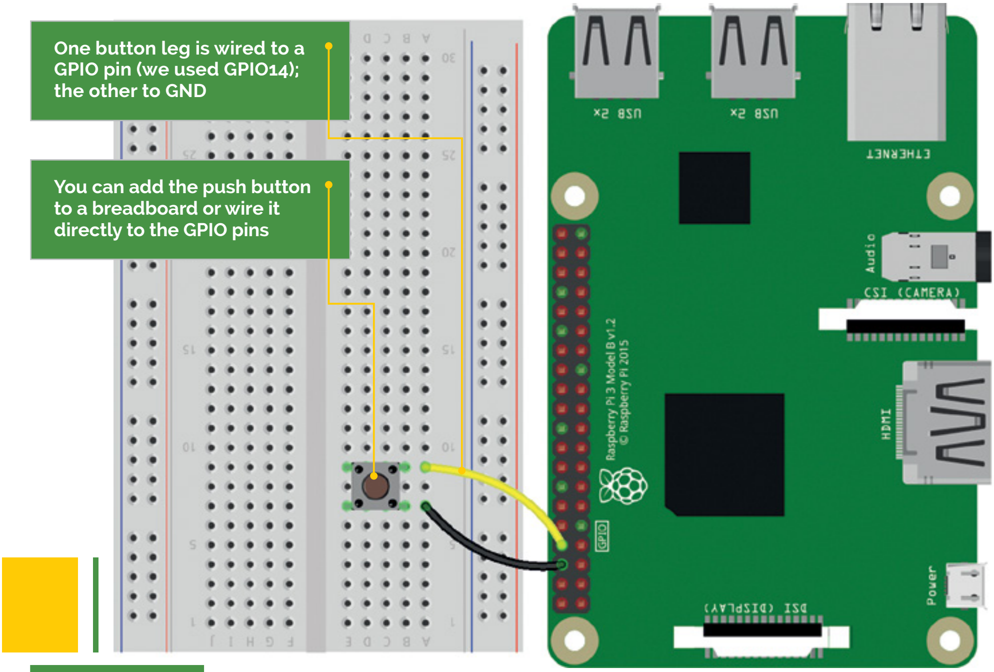
The MagPi ESSENTIALS

[CHAPTER SIX]

STOP-MOTION AND SELFIES

Wire up a physical push button to take photos

Fig 1 Connect the button



One button leg is wired to a GPIO pin (we used GPIO14); the other to GND

You can add the push button to a breadboard or wire it directly to the GPIO pins

[YOU'LL NEED]

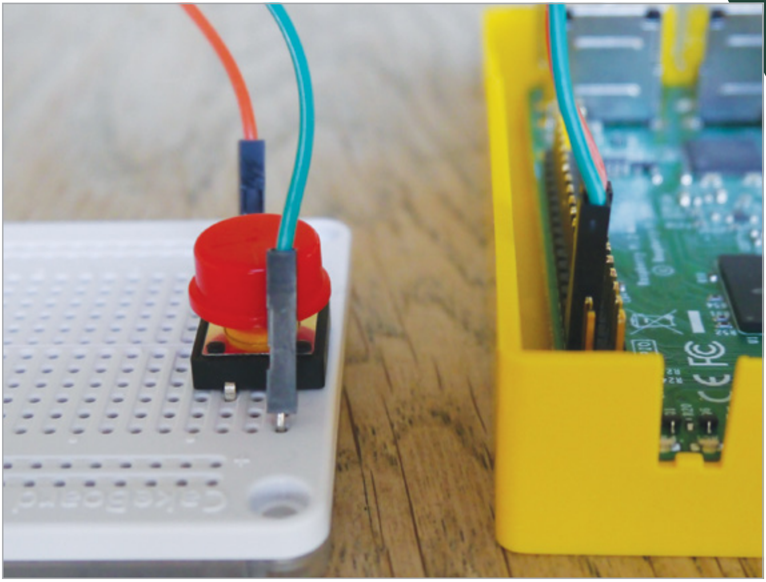
- Camera Module
- Push button
- Breadboard (optional)
- Jumper wires
- Pi case with a hole for the camera cable (selfie stick)
- Long wires (selfie stick)
- A stick, slim metal pole etc. (selfie stick)

Have you been reading the last few chapters and thinking you'd like to take a picture with a Raspberry Pi Camera Module with less hassle? In this tutorial we'll show you how to take a photo with a click of a button, just like a real camera. This could be useful for many projects (for example, time-lapse photography), but in this chapter we are focusing on stop-motion animation. We also show how to create your own selfie stick!

>STEP-01

Wire up the button

If you haven't already switched your Raspberry Pi off, do so now. Next, connect the button to the Pi via a jumper lead, as shown in **Fig 1**. One side of the button will be connected to ground (GND); the other is connected to GPIO pin 14 (but you can choose your favourite pin). We used a breadboard for our stop-motion animation project, but you could wire the button directly to the pins (as you'll be doing for the selfie stick later).



You can use a breadboard for a small button, or connect your jumper wires directly to the pins on a bigger one

>STEP-02

Install Picamera

That’s all the hardware done. Now it’s time for the software. If you haven’t done so already in chapter 5, you’ll need to install the Picamera library. In a terminal window, enter:

```
sudo apt-get update
sudo apt-get install python-picamera python3-picamera
```

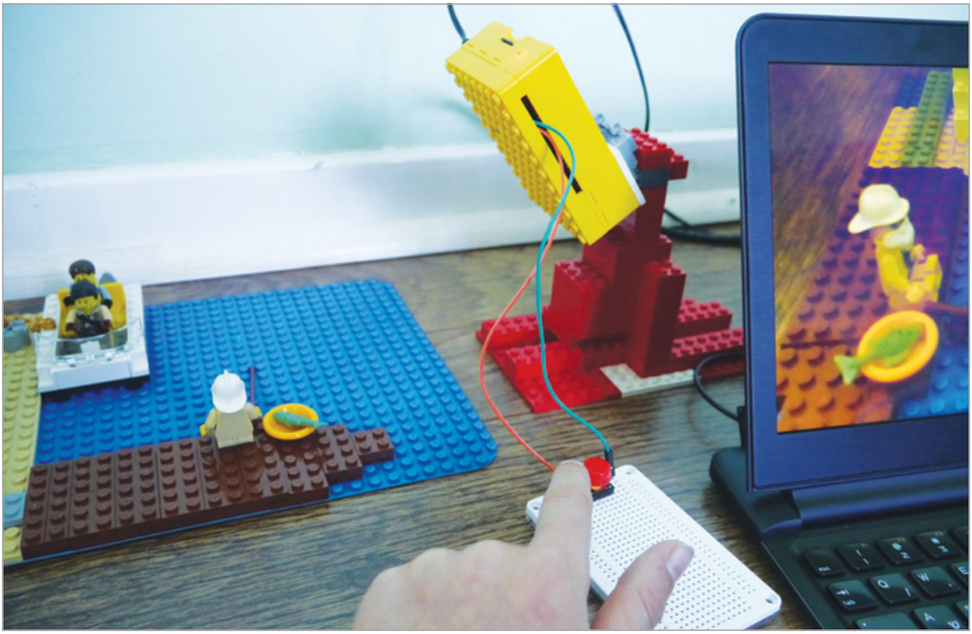
If for some reason you don’t have GPIO Zero already installed (it has come pre-installed in Raspbian for some time), do so with:

```
sudo apt-get install python-gpiozero python3-gpiozero
```

>STEP-03

Stop-motion software

Because we’re focusing on stop-motion for our first project, we’re using the camera’s preview mode so that we can set up our shot



Create your stop-motion scene and use the button to trigger the camera to take pictures and save them to timestamped file

before we take it, to ensure everything is in the frame. Then, only when the button is pressed do we save an image file. Each image file will have a different name based on the date and time at which it is taken. This makes it easy to assemble all the images from the shoot for post-processing.

The wonderful GPIO Zero library is used to capture the button activity; we simply define a function that is run whenever the button is pressed. This function uses the Picamera Python library which allows us to control the camera through code, making all the normal command-line operations available.

Download or type up the code from **ch6listing1.py** and either run it through IDLE or the command line. To quit the program, press **CTRL+C**.

>STEP-04

Other variations

You should be able to use this code as a template to create a program for whatever photography project you have in mind. For example, you could alter the code so that the camera takes continuous photos while the button is held down. Or you could add extra buttons to make a variety of photography modes available.

With this sort of build, you can also start thinking about building a complete, portable, wirelessly connected Pi camera. For this, you can use a case into which you can fit a portable mobile phone battery charger, along with a screen to attach to the Pi. With a bit of modification of the code, you can have it always show the preview of the camera on the screen. Want to record video? More modification of the code will allow for video capturing. The only issue you might have with both of these projects is the lack of a flash or built-in light source, so a well-lit subject would be essential.

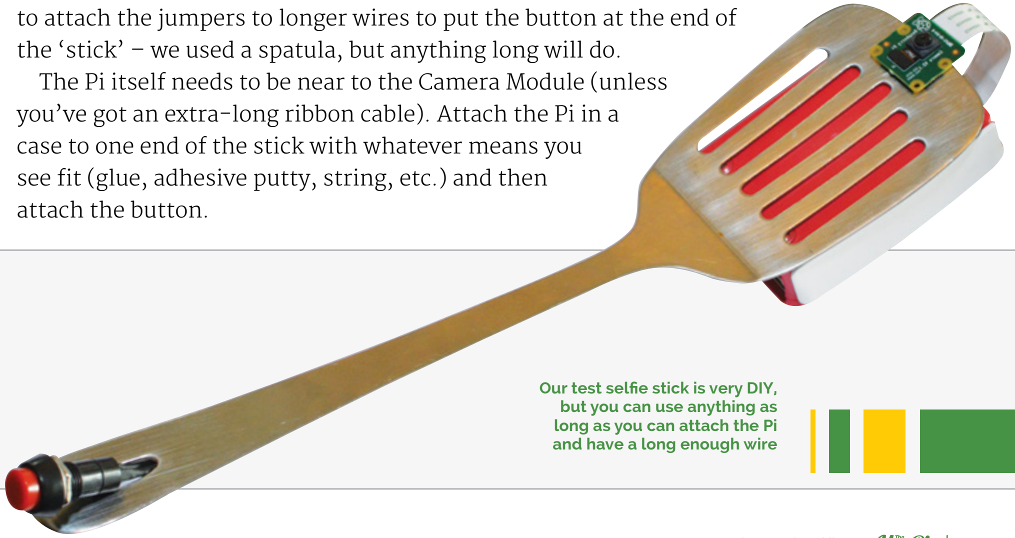
>STEP-05

Selfie stick

Next, we'll look at making a selfie stick. A lot of people roll their eyes and complain about vanity when it comes to the art of the selfie, but we all know it's nothing like that. New outfit? New glasses? Eyeliner wings perfectly symmetrical today? Why not chronicle it? It's a great confidence boost.

Our Pi-powered selfie stick will use a similar hardware and software setup to the stop-motion animation project. As before, we're wiring up a push button to GPIO14 and GND pins on the Pi, but this time we need to attach the jumpers to longer wires to put the button at the end of the 'stick' – we used a spatula, but anything long will do.

The Pi itself needs to be near to the Camera Module (unless you've got an extra-long ribbon cable). Attach the Pi in a case to one end of the stick with whatever means you see fit (glue, adhesive putty, string, etc.) and then attach the button.



Our test selfie stick is very DIY, but you can use anything as long as you can attach the Pi and have a long enough wire

ch6listing1.py

Language

>PYTHON 3

DOWNLOAD:
magpi.cc/2pWilzu

```
#importing the necessary modules
from datetime import datetime
from gpiozero import Button
import picamera
import time

b=Button(14)
pc=picamera.PiCamera()
running = True
#pc.resolution = (1024, 768)
#use this to set the resolution if you dislike the default values
timestamp=datetime.now()
def picture():
    pc.capture('pic'+str(timestamp)+'.jpg') #taking the picture

pc.start_preview() #running the preview
b.when_pressed=picture
try:
    while running:
        print('Active')#displaying 'active' to the shell
            time.sleep(1)
#we detect Ctrl+C then quit the program
except KeyboardInterrupt:
    pc.stop_preview()
    running = False
```

>STEP-06

Add the code

Since the principle is the same – pressing a button to take a photo – we can use the same code, **ch6listing1.py**, as for the stop-motion project. This time we don't need the camera preview, so you can comment out the line **`pc.start_preview()`** if you like, by adding a **#** to the start of it.

Try running the code. Pressing the button will take a photo, but you'll need to practise your aim so you can get yourself in the frame. As before, we add a timestamp to each picture, which helps to organise your pictures later and also results in a slight pause in the code, which at least means you won't take too many pictures with a slip of the button.

The MagPi ESSENTIALS

[CHAPTER SEVEN] FLASH PHOTOGRAPHY USING AN LED

Add an LED flash to shoot images in low light

[YOU'LL NEED]

- Camera Module
- White LED
- Resistor

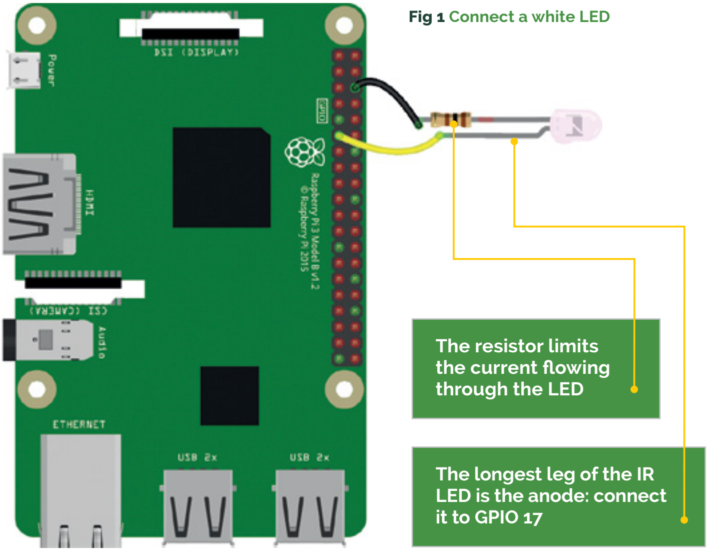


Fig 1 Connect a white LED

The resistor limits the current flowing through the LED

The longest leg of the IR LED is the anode: connect it to GPIO 17

The Raspberry Pi Camera Module works really well in good lighting conditions, but what if there's less light available? Here we show you how to set up a simple LED flash, which will be triggered each time you take a photo, using the Picamera Python library. We also take a look at how to shoot better images in low light when you are not using a flash.

>STEP-01

Download device tree source

Before we can wire up a flash, we need to configure a GPIO pin to use for it. This will then be triggered each time we capture a still using Picamera with the flash mode set to on. To do this, we need to edit the VideoCore GPU default device tree source. First, install device tree compiler with:

```
sudo apt-get install device-tree-compiler
```

Then grab a copy of the default device tree source with:

```
wget https://raw.githubusercontent.com/raspberrypi/
firmware/master/extra/dt-blob.dts
```

>STEP-02

Edit the device tree source

Edit the file using your favourite text editor, such as nano:

```
sudo nano dt-blob.dts
```

You'll need to find the correct part of the code for the Raspberry Pi model you're using; for instance, the part for the Pi 3 v1.2 is found under **`pins_3b2 {`**.

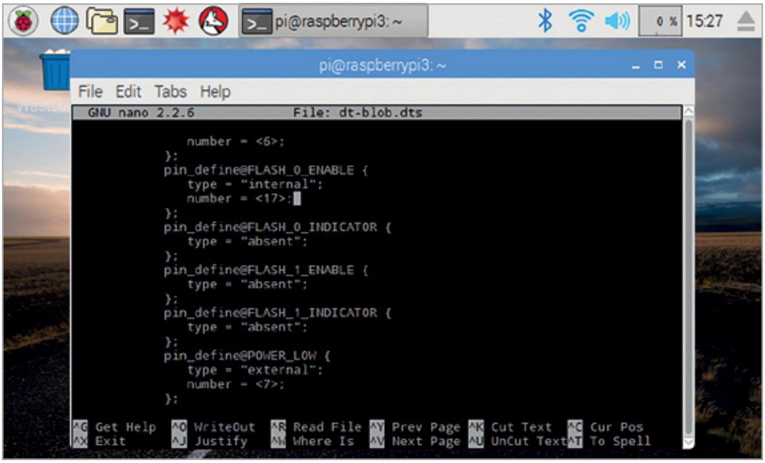
Here you'll find **`pin_config`** and **`pin_defines`** sections. In the **`pin_config`** section, add a line to configure the GPIO pin (we're using BCM 17) that you want to use for the flash:

```
pin@p17 { function = "output"; termination = "pull_down"; };
```

>STEP-03

Enable flash

Next, we need to associate the pin we added with the flash enable function by editing it in the **`pin_define`** section. We simply change **`absent`** to **`internal`** and add a line with the pin number, so it looks like the following (overleaf):



```
  GNU nano 2.2.6              File: dt-blob.dts

            number = <6>;
        };
        pin_define@FLASH_0_ENABLE {
            type = "internal";
            number = <17>;
        };
        pin_define@FLASH_0_INDICATOR {
            type = "absent";
        };
        pin_define@FLASH_1_ENABLE {
            type = "absent";
        };
        pin_define@FLASH_1_INDICATOR {
            type = "absent";
        };
        pin_define@POWER_LOW {
            type = "external";
            number = <7>;
        };

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

You need to edit the device tree source to enable a GPIO pin for the flash

```
pin-define@FLASH_0_ENABLE {
    type = "internal";
    number = <17>;
};
```

Note that it's the **FLASH_0** section that you need to alter: **FLASH_1** is for an optional privacy LED to come on after taking a picture, but we won't bother with that.

>STEP-04

Compile the blob

With the device tree source updated, we now need to compile it into a binary blob, using the following command in a terminal window:

```
dtc -q -I dts -O dtb dt-blob.dts -o dt-blob.bin
```

This should output nothing. Next, you need to place the new binary on the first partition of the SD card. In the case of non-NOOBS Raspbian installs, this is generally **/boot**, so use:

```
sudo cp dt-blob.bin /boot/
```

In you installed Raspbian via NOOBS, however, you'll need to do the following instead:

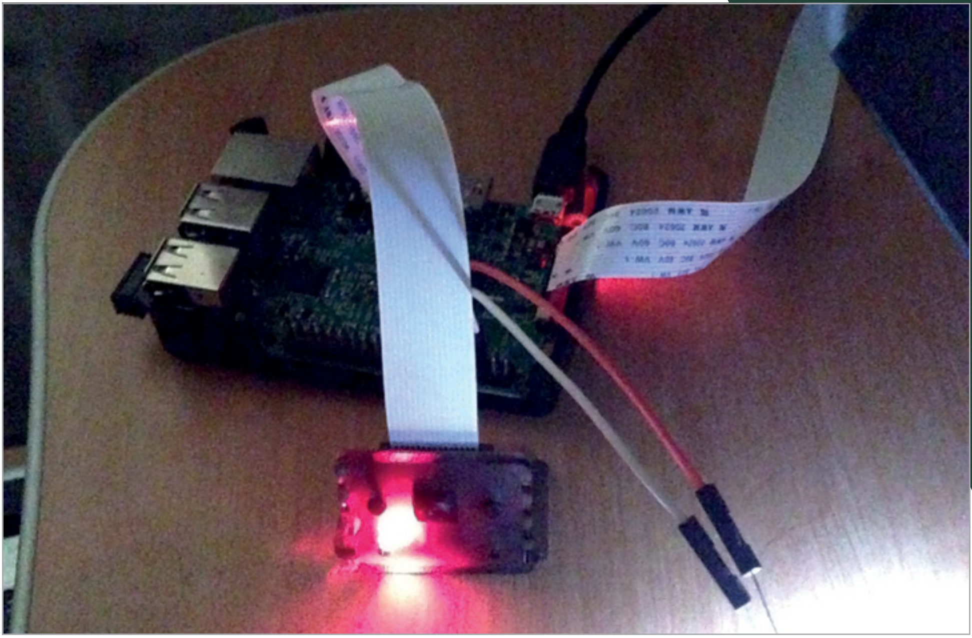
```
sudo mkdir /mnt/recovery
sudo mount /dev/mmcblk0p1 /mnt/recovery
sudo cp dt-blob.bin /mnt/recovery
sudo umount /mnt/recovery
sudo rmdir /mnt/recovery
```

To activate the new device tree configuration, reboot the Raspberry Pi.

>STEP-05

Wire up the LED

Connect a white LED – we used a 5mm one – to your Raspberry Pi as in **Fig 1**. The LED's anode (long leg) is connected to our flash-enabled GPIO pin, BCM 17. To be sure of the LED not burning out from excess



Wire the white LED to GPIO 17 and GND, with optional resistor

current, you should add a low-ohmage resistor (such as 100Ω) between the LED's cathode (short leg) and Raspberry Pi's GND pin. Depending on the maximum forward voltage of your LED (ours was 3.5V), you may be able to get away without using one, but it's best to be safe.

If you want to use higher-powered or multiple LEDs, you'll have to think about powering them via a suitable driver circuit, with a transistor wired to the flash pin. You may also need a separate power supply. Note that, due to the Camera Module's rolling shutter, only an LED or equivalent flash is suitable: you can't use a xenon flash. Alternative flash/lighting methods include NeoPixel sticks and the LISIPAROI light ring.

>STEP-06

Test it out

With the LED connected, we can now test out our flash with a short Python program. In Python 3 (IDLE), create a new file and enter the code from **ch7listing1.py**. The `camera.flash_mode = 'on'` line sets the flash to trigger when we issue the capture command below; the

Even a single LED can provide illumination for close-up photography

LED will light up briefly before the image capture, to enable the camera to set the correct exposure level for the extra illumination, before the flash proper is triggered.

If you want the flash to trigger automatically only when it's dark enough, you can change the penultimate line of the code to **`camera.flash_mode = 'auto'`**.

>STEP-07

Low-light photography

In low-light scenarios where you don't want to use a flash, you can improve capture of images using a few tricks. By setting a high gain combined with a long exposure time, the camera is able to gather the maximum amount of light. Note that since the **`shutter_speed`** attribute is constrained by the camera's frame rate, we need to set a very slow **`framerate`**. The code in **ch7listing2.py** captures an image with a six-second exposure time: this is the maximum time for the v1 Camera Module; if you have a v2 Camera Module, it can be extended to ten seconds. The frame rate is set to a sixth of a second, while we set the ISO to 800 for greater exposure. A pause of 30 seconds gives the camera enough time to set gains and measure AWB (auto white balance).

Using a long exposure, you can shoot stills in very dark settings

Language

>PYTHON 3

DOWNLOAD:
magpi.cc/2pWtlwL

Try running the script in a very dark setting: it may take some time to run, including the 30-second pause and about 20 seconds for the capture itself. Note: if you're getting a timeout error, you may need to do a full Raspbian upgrade with **`sudo apt-get update`** and **`sudo apt-get dist-upgrade`**.

The particular camera settings in this script are only useful for very low light conditions: in a less dark environment, the image produced will be heavily overexposed, so you may need to increase the frame rate and lower the shutter speed accordingly.

If your image has a green cast, you'll need to alter the white balance manually. Turn AWB off with **`camera.awb_mode = 'off'`**. Then set the red/blue gains manually, such as with **`camera.awb_gains = (1.5, 1.5)`**.

ch7listing1.py

```
import picamera

with picamera.PiCamera() as camera:
    camera.flash_mode = 'on'
    camera.capture('foo.jpg')
```

ch7listing2.py

```
from picamera import PiCamera
from time import sleep
from fractions import Fraction
# Set a framerate of 1/6fps, then set shutter
# speed to 6s and ISO to 800
camera = PiCamera(resolution=(1280, 720), framerate=Fraction(1, 6))
camera.shutter_speed = 6000000
camera.iso = 800
# Give the camera a good long time to set gains and
# measure AWB (you may wish to use fixed AWB instead)
sleep(30)
camera.exposure_mode = 'off'
# Finally, capture an image with a 6s exposure. Due
# to mode switching on the still port, this will take
# longer than six seconds
camera.capture('dark.jpg')
```

The MagPi ESSENTIALS

[CHAPTER EIGHT] MAKE A MINECRAFT PHOTO BOOTH

Create a photo booth in Minecraft that takes photos of the real world. What will you see on your travels?

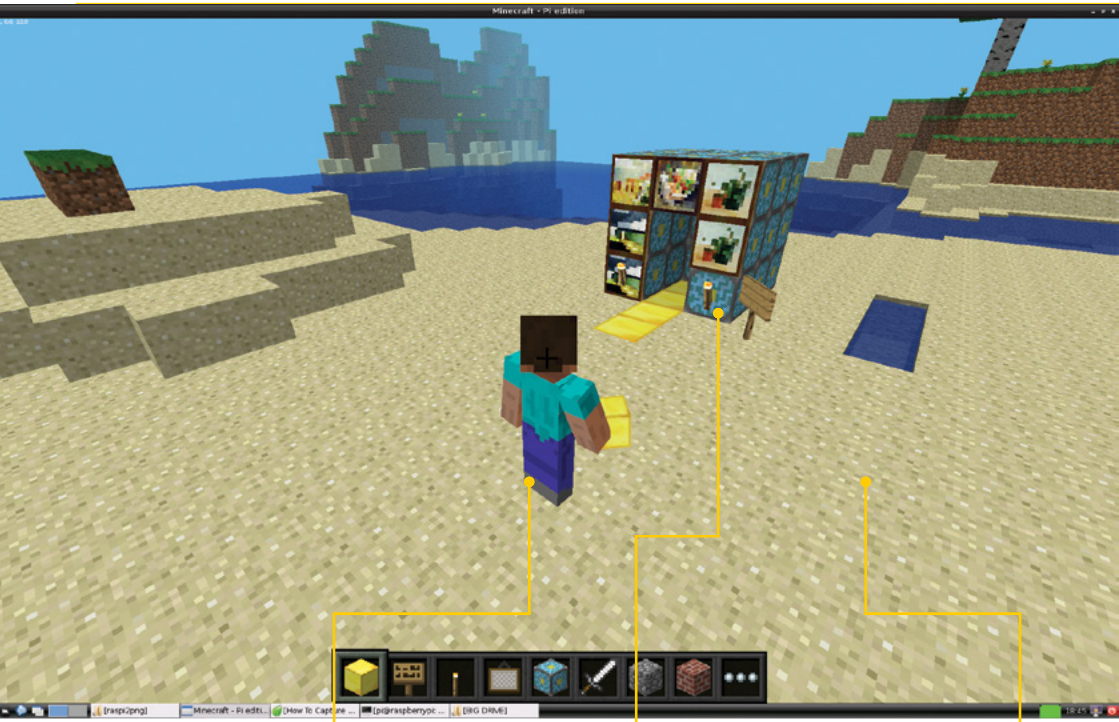
Not only is Minecraft Pi great fun to play around with, you can also use Python programming to manipulate the Minecraft world and create various structures within it. Going beyond this, you can even have it intereact with the real world. In this chapter, we'll be getting Minecraft to trigger the Camera Module with code when the player enters a virtual photo booth.

The first thing you need to do is import the Minecraft API (application programming interface). This enables you to connect to Minecraft and program it with Python. You also need to import Picamera's **PiCamera** class to control the camera, and the **time** module to add a small delay between taking each photo.

Open Minecraft from the applications menu, then enter an existing world or create a new one. Move the Minecraft window to one side of the screen. You'll need to use the **TAB** key to take your mouse's focus away from the Minecraft window to move it. This will be needed later when you switch between the Minecraft and Python windows.

Open Python 3 from the applications menu. This will open up the Python IDLE code editor which you'll use to write the photo booth program; click New > Window to open a new window.

Enter the code from **ch8listing1.py**, or download it. Save with **CTRL+S** and run the program with **F5**. You should see the message



Steve is your 'shutter' in the Minecraft world: move him to the booth to take a photo

Construct your photo booth however you wish; just make sure the code knows where it lives

Place the booth anywhere in your world. Give it a special room in your house, or use it as a trap to see if someone is in your world

'Find the photobooth' appear in the Minecraft world. This is the first part of the code. Stop the program running using **CTRL+C**, and we can explain the rest.

Camera tests

Next, we'll make sure the camera is set up. We've set the camera to show a two-second preview, so that you can strike your pose and smile before the picture is taken. The image is stored as a file called **selfie.jpg** in your home directory.

Now, you need to create a photo booth in the Minecraft environment. This is done manually, and the booth can be built wherever you want to locate it. Using any block type, build your photo booth. It can be any shape you like, but it should have at least one block width of free space inside so that the player can enter.

Once you have created your photo booth, you need to be able to move your player inside and onto the trigger block. This is the block that the player stands on to run the function that you wrote in the first step, which will then trigger the camera. In the Minecraft environment, your position is given in reference to the x, y, and z axes. Look at the top-right of the window and you'll see the x, y, and z coordinates of your player – for example, 10.5, 9.0, -44.3. Assuming you are still in the photo booth, then these are also the x, y, and z coordinates of the trigger block in your booth.

Walk into your photo booth

Note down all three coordinates of your camera trigger block. When you're playing Minecraft, your program will need to verify that you are inside the photo booth. If you are, then it will trigger the **take_the_pic** function and take a picture with the camera. To do this, Minecraft needs to know where you are in the world.

In order to find your position, you use the code **x, y, x = mc.player.getPos()**. This saves the x, y, and z position of your player into the variables **x**, **y**, and **z**. You can then use **print(x)** to print the x value, or **print(x, y, z)** to see them all if you wish, by adding it to the code. Now you know the position of the player, you can test to see if they're in the photo booth.

At this point we have a photo booth, the coordinates of the trigger block, and code to control the Camera Module and take a picture. The next part of the code is to test whether the program knows when you're in the photo booth. To do this, we create a loop which checks if your player's coordinates match the trigger block coordinates. If they do, then you're standing in the photo booth. For this, we use a simple **if** statement, which is known as a conditional.

Change the **if** line in the code to ensure the coordinates you enter are those of your photo booth. Save and run your code to test it: walk into your photo booth and you should see the message 'You are in the photobooth!' in the Minecraft window.

You will note that the **if** statement checks if the x value is greater than or equal to 10.5: this is to ensure that it picks up the block, as it could have a value of 10.6. Remember to replace the x, y, and z values with those from your photo booth. After the message is printed, the same preview and camera snap will happen as before the **`while`** loop. The loop then resets itself so you can enter it again and take another photo!

ch8listing1.py

Language
>PYTHON 3

DOWNLOAD:
magpi.cc/2pWGyFA

```
from mcpi.minecraft import Minecraft
from picamera import PiCamera
from time import sleep

mc = Minecraft.create()
camera = PiCamera()

mc.postToChat("Find the photo booth")

camera.start_preview()
sleep(2)
camera.capture('/home/pi/selfie.jpg')
camera.stop_preview()

while True:
    x, y, z = mc.player.getPos()

    sleep(3)

    if x >= 10.5 and y == 9.0 and z == -44.3:
        mc.postToChat("You're in the photo booth!")
        sleep(1)
        mc.postToChat("Smile!")
        sleep(1)
        camera.start_preview()
        sleep(2)
        camera.capture('/home/pi/selfie.jpg')
        camera.stop_preview()

    sleep(3)
```

The MagPi ESSENTIALS

[CHAPTER NINE]

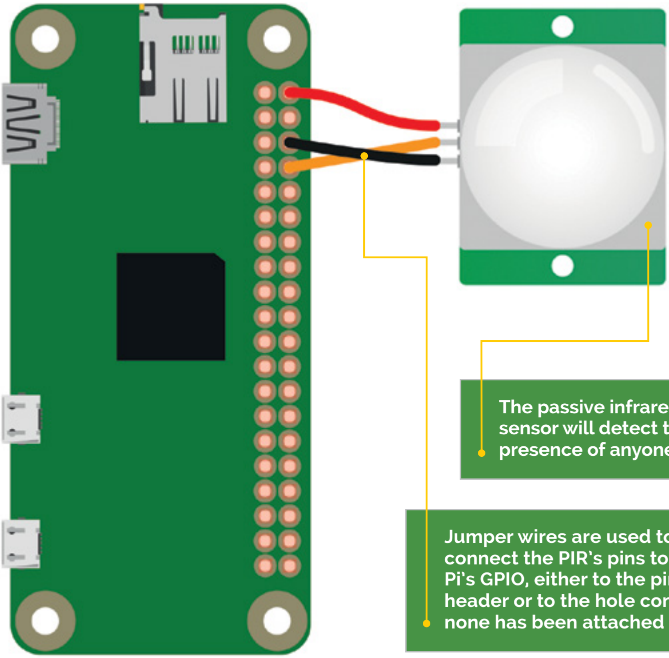
MAKE A SPY CAMERA

Set up a motion-activated spy camera in your room

[YOU'LL NEED]

- Camera Module
- PIR sensor **magpi.cc/2gCQKPjD**
- Pi Zero camera cable adapter (optional) **magpi.cc/2gT2KwE**
- Portable power supply (optional)
- Jumper wires

Fig 1 Connect a PIR sensor



The passive infrared (PIR) sensor will detect the presence of anyone nearby

Jumper wires are used to connect the PIR's pins to the Pi's GPIO, either to the pins of a header or to the hole contacts if none has been attached

We've all been there. You've gone out for the day and you know you closed your bedroom door but you come back and it's slightly ajar. Who's been in there? Were they friend or foe? In this chapter we'll use the Camera Module as a spy camera that takes a picture when anyone's presence is detected by a passive infrared (PIR) sensor. Here we're using a Pi Zero v1.3 – which is easier to hide away due to its size – with a camera adapter cable, but you can use any Raspberry Pi model. Unless you want to power it from the mains, you'll also need a portable power supply such as a mobile phone battery pack.

>STEP-01

Getting started

First, connect your Camera Module to the Pi. Note that if you're using a Pi Zero, you'll need a special adapter cable since its camera connector is smaller: the cable's silver connectors should face the Pi circuit board. You'll also need to have enabled the camera in Raspberry Pi Configuration, as explained in chapter 1.

We'll be using the Picamera Python library to trigger our spy camera, so if you haven't yet installed it, open a terminal window and enter:

```
sudo apt-get update
sudo apt-get install python-picamera python3-picamera
```

>STEP-02

Wire up the circuit

The circuit for this is fairly simple, especially as the PIR does not need a resistor as part of its setup. The PIR comes with three connection pins: VCC, GND, and OUT. If you can't find their labels on the bottom of the sensor, lift off the plastic golf-ball-like diffuser and you should see them on the top of the board. VCC needs to be connected to a 5V power pin, GND needs to go to a ground pin, and then there's the OUT wire which will be our input. We're connecting it to GPIO 14.

If your Pi Zero has GPIO pins attached, you can use female-to-female jumper wires to make the connections, as shown in **Fig 1**. Otherwise you can loop the wire around the GPIO holes and use a bit of putty to keep them in place, or a dab of glue from a glue gun on a low setting. Soldering is an option if you want to create a permanent spy camera device.

>STEP-03

Write the code

Now we've got it all wired up, it's time to start coding our spy camera. In Python 3 (IDLE), create a new file, and enter the code from **ch9listing1.py**. This script uses two libraries: GPIO Zero and the standard Picamera library. GPIO Zero can be used to get a reading from the PIR motion sensor very easily, which can then be tied into the Picamera code so it takes a photo when motion is detected.

At the top, we import **`MotionSensor`** from GPIO Zero and **`PiCamera`** from Picamera. Since we'll be giving each photo a timestamp, we also import **`datetime`**, along with **`sleep`** from the **`time`** library.

In a never-ending **`while True:`** loop, we use GPIO Zero's handy **`wait_for_motion`** function to pause the code until the PIR detects any motion. When it does, we set the photo file name to the current time and date, then take the picture. To enable the PIR to settle, we sleep for five seconds before returning to the top of the loop to wait for motion again.

Camouflaged against Yoshi, the camera will take candid snaps of anyone who comes close to your game collection

>STEP-04

Final preparations

You can run the code first to give it a test. You might want to change the sensitivity and/or trigger time, which you can do by adjusting the little orange potentiometer screws on the side of the PIR board: Sx adjusts sensitivity, while Tx alters the trigger time.

Once that's done, we'll get the program to start automatically whenever we boot up the Raspberry Pi. To do so, open up a terminal window and edit the profile config file with **`sudo nano /etc/profile`**. To the bottom of the file, add this line:

```
sudo python spy.py
```

In addition, to get the Raspberry Pi to boot up slightly faster and, more importantly, to use a little less power so your battery lasts longer, it's best to get it to boot directly to the command line rather than booting to the desktop. The easiest way to change this is to open Preferences > Raspberry Pi Configuration from the desktop; in the default System tab, change Boot to the 'To CLI' option. Alternatively, open a terminal window and enter **`sudo raspi-config`** to open the Configuration Tool; select Boot Options > Desktop / CLI and option B2 – Console Autologin Text console.

Language

>PYTHON 3

DOWNLOAD:
magpi.cc/2pWmGT5

>STEP-05

Hide your camera

Now you need to find a good place to hide your camera. The default cable for the camera is limited by length, while the PIR can have its wires lengthened, so keep that in mind when building your system. Alternatively, you could get a camera extender (**magpi.cc/2ioVgFu**) to link your cable to a standard-width one. Longer standard-width cables – of up to 2m – are also available if you are not using a Pi Zero.

Hiding the Pi and battery behind a plush toy or photo frame can work well (you could even put a dummy photo up and cut a hole in it for the camera to look through). The PIR has quite a wide range, so put it up high where people are unlikely to look.

>STEP-06

Check for intruders

All you need to do now is plug in the power supply and the Pi Zero will turn on and automatically run the script. Do some tests to make sure the camera is facing the right way. Leave it running during the day and then when you get back, plug it into a monitor, stop the script, and run **startx** to get the GUI up. From here you can see the pictures it has taken: crucial evidence to catch your dog or sibling red-handed.

ch9listing1.py

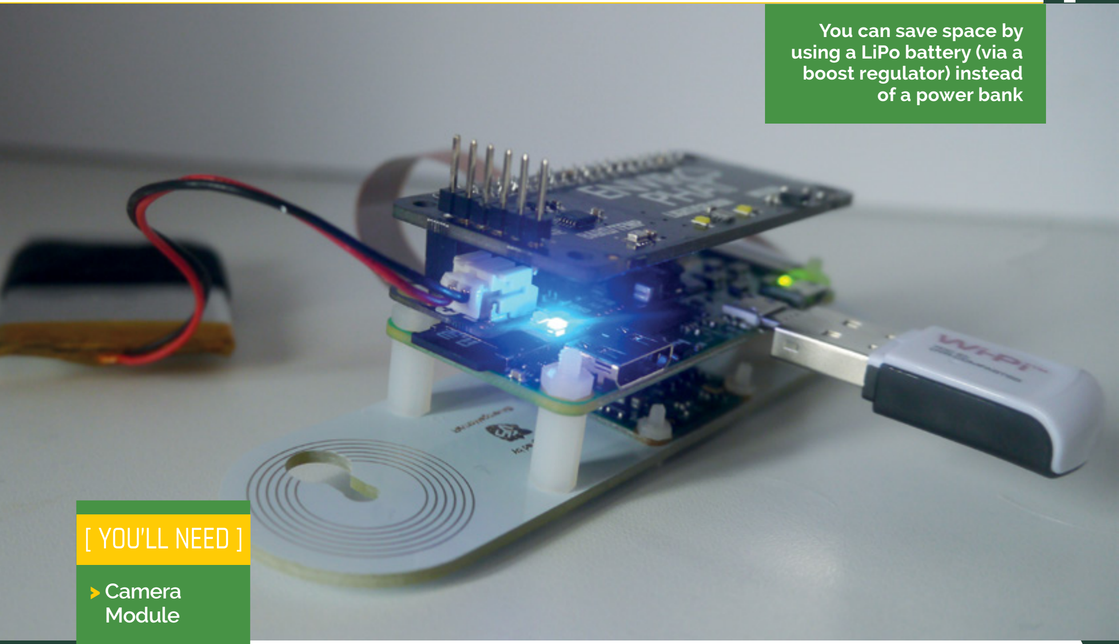
```python
#!/usr/bin/env python

from gpiozero import MotionSensor
from picamera import PiCamera
from datetime import datetime
from time import sleep

sensor = MotionSensor(14)
camera = PiCamera()

while True:
    sensor.wait_for_motion()
    filename = datetime.now().strftime("%H.%M.%S_%Y-%m-%d.jpg")
    camera.capture(filename)
    sleep(5)
```

The MagPi ESSENTIALS

[CHAPTER TEN]

TAKE YOUR CAMERA UNDERWATER

Explore the underwater world with a Camera Module



You can save space by using a LiPo battery (via a boost regulator) instead of a power bank

[YOU'LL NEED]

- Camera Module
- Transparent, waterproof box – **magpi.cc/2e8beBX**
- Portable power source
- hostapd and dnsmasq packages
- Python Flask library
- WiFi dongle (if not using a Pi 3)
- Enviro pHAT (optional) – **magpi.cc/29NHB3T**
- ZeroView (optional) – **magpi.cc/2e8ghWt**

There are plenty of underwater sports cameras available, but they can be quite expensive, especially if you want to be able to control them remotely. In this chapter we're going to use readily available Raspberry Pi add-ons to make a cheaper, customisable camera unit. There are lots of options and alternative sources of components for a project like this. For example, the Pimoroni Enviro pHAT is a really useful option that can report back information about the environment in which the camera is operating, especially how much light is available. There is a fair bit of software configuration involved, but example config files are available in the GitHub repo for this chapter (**magpi.cc/2pWo5Jz**).

>STEP-01

Find a suitable container

Naturally, to protect the electronics inside it, the container for the Raspberry Pi and camera needs to be watertight and to have at least a see-through lid. You can find food container boxes with a very tight seal, but these tend to be translucent rather than transparent. The size of box will probably determine your choice of Pi model and power source. Pi Zeros are great as they are so small, but unless you have a Zero W then

You'll still have to get pretty close to the water yourself

you'll need a WiFi dongle and shim, or a special micro-USB dongle (**magpi.cc/2ilhcyN**). You can also save space by using a LiPo battery instead of a power bank (although you'll need a boost regulator too, such as the Pimoroni Zero LiPo).

>STEP-02

Configure your Pi to be a WiFi access point

Start from a fresh Raspbian Jessie Lite SD card. Open up a terminal window and enter the following commands to update the APT database and install the required packages:

```
sudo apt-get update
sudo apt-get install -y dnsmasq hostapd python3 python3-
dev python3-flask python3-picamera
```

First, configure your wireless interface to have a static IP address by editing the **/etc/network/interfaces** file:

```
sudo nano /etc/network/interfaces.
```

Then set it to not use DHCP by adding the following line to the end of your **/etc/dhcpcd.conf** file:

```
denyinterfaces wlan0
```

Next, create the **/etc/hostapd/hostapd.conf** file using the example in this tutorial's GitHub repository as a template. Change the interface, SSID, and passphrase parameters as needed.

Finally, edit **/etc/dnsmasq.conf**, ensuring that the IP addresses are consistent with your settings in **/etc/network/interfaces**. Then reboot your Raspberry Pi.

>STEP-03

Add the Enviro pHAT

Pimoroni's add-on board enables you to send back environmental data from the camera. You can either solder the Enviro pHAT directly onto the Pi's GPIO pins, or you can use the supplied female header if you want to reuse it in other projects. After that, install the Python library and dependencies using the following command:

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

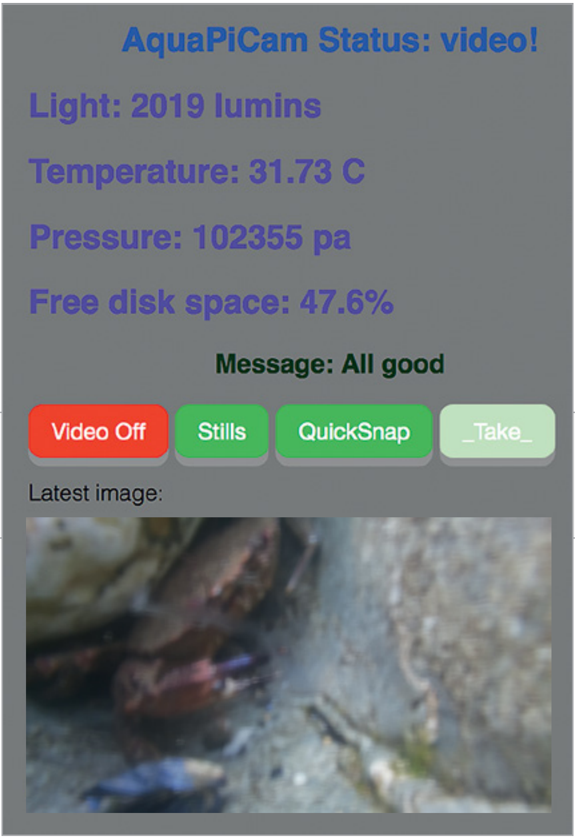
The Enviro pHAT's library comes with some example programs; you should run these to test that everything is working correctly.

Note: If you are not using an Enviro pHAT, you'll need to comment out some of the related code in the main **ch10listing1.py** script.

>STEP-04

Fitting everything into your container

To cut down on reflections and obtain the best possible images, the camera should be as close to the transparent side of your container as possible. The ZeroView from the Pi Hut is a clever mounting plate that uses suction cups and will also hold your Pi Zero securely. Alternatively,



AquaPiCam Status: video!

Light: 2019 lumins

Temperature: 31.73 C

Pressure: 102355 pa

Free disk space: 47.6%

Message: All good

The web interface shows environmental information and lets you control the camera

you could make a mount out of cardboard and glue this to the inside of the container. Velcro tape can be a good solution for power sources (which normally need to be removable for recharging).

>STEP-05

Add some code, HTML and CSS

Clone the project's GitHub repository onto your Pi. In a terminal window, enter:

```
git clone https://github.
com/themagpimag/essentials-
cameraCh10
```

Then use the desktop File Manager to move the **Flask** folder within **essentials-cameraCh10** to the Pi's home directory (or use the **mv** command in a terminal window).

Flask is a small web framework written in Python; it allows you to create simple web services – in this case, a webpage that allows us to see data from the Enviro pHAT and the latest captured images. To see the webpage from another computer, you just have to open a web browser and enter your Pi's static IP address.

Using the on-screen buttons, we can also switch between recording modes (video or continuous still frames) or take photos on demand – by selecting QuickSnap and then clicking the Take button. This control of the camera is achieved via the Picamera library, which is used for the three main functions – **timelapse**, **video**, and **snapstart** – defined in our Python script. You could enhance the project by adding additional exposure and shutter speed controls to your interface if you want.

Note: To see the latest image taken, you need to reload the webpage. If you are using Chrome, you may need to hold **SHIFT** and press **R** to refresh.

>STEP-06

Set the code to run at boot

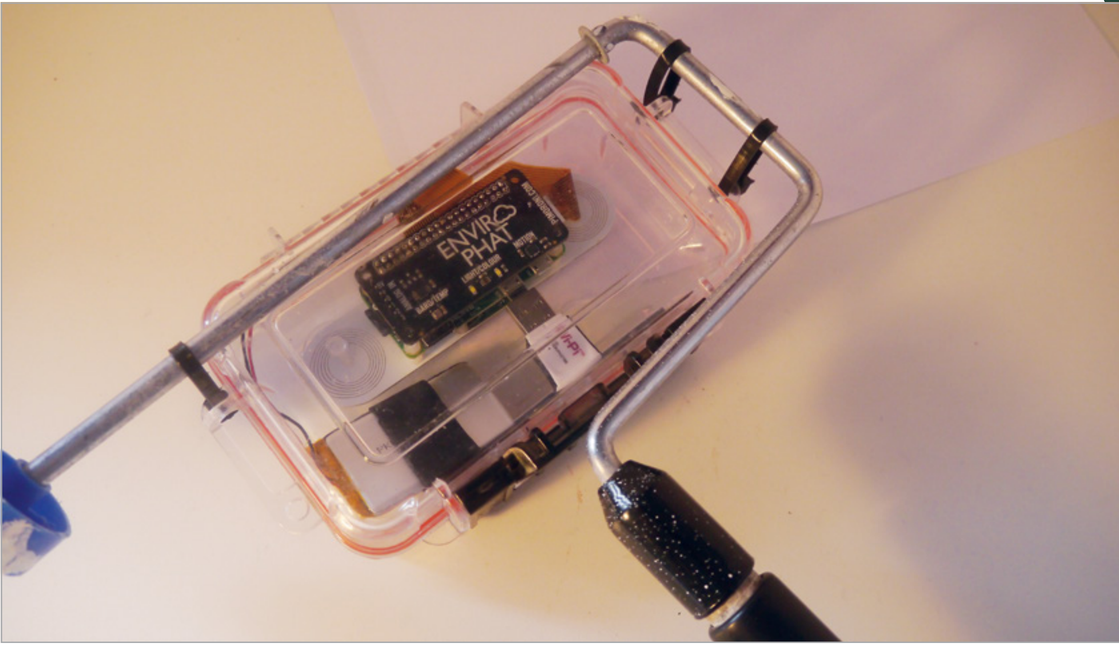
Naturally, we'll want the code to run automatically whenever the Raspberry Pi boots up. To do so, add this line to your **/etc/rc.local** file, immediately above the **`exit 0`** line:

```
python3 /home/pi/Flask/ch10listing1.py &
```

It is also a good idea to configure the Raspberry Pi to only boot to the command line rather than the desktop, as this uses a little less power and prolongs battery life. The easiest way to change this is to open Preferences > Raspberry Pi Configuration from the desktop; in the default System tab, change Boot to the 'To CLI' option. Alternatively, open a terminal window and enter **`sudo raspi-config`** to open the Configuration Tool; select Boot Options > Desktop / CLI and option B2 – Console Autologin Text console.

Now go and find somewhere wet! You might want to run a few tests in the bath before venturing further afield.

A makeshift handle to lower the waterproof box into the water



ch10listing1.py

```
from flask import Flask, render_template,request, redirect, url_for
from envirophat import light,weather
import time, os, shutil
from picamera import PiCamera
from datetime import datetime, timedelta
from threading import Thread

app = Flask(__name__)

def timelapse():  # continuous shooting
    cam = PiCamera()
    cam.resolution = (1640,922)
    for filename in cam.capture_continuous('img{timestamp:%Y%m%d-%H%M%S}.jpg'):
        print('snap taken')
        print(btn1,btn2)
        shutil.copyfile(filename,'/home/pi/Flask/static/latest.jpg')
        if btn1 != 's':
            break
    cam.close()
    print('timelapse thread stopped')

def video(): # record a video
    cam = PiCamera()
    t='{:%Y%m%d-%H%M%S}'.format(datetime.now())
    cam.resolution = (1920,1080)
    cam.start_recording('vid'+t+'.h264')
    while btn1 == 'v':
        print(btn1,btn2)
        pass
    cam.stop_recording()
    cam.close()
    print('video thread stopped')

def snapstart(): # take pictures on demand
    cam = PiCamera()
    cam.resolution = (1640,922)
```

Language

>PYTHON 3

DOWNLOAD:
magpi.cc/2pjHcRm

```
    print('entered snapshot mode')
    global btn2
    while btn1 == 'q':
        time.sleep(0.1)
        if btn2 == 'a':
            print('taken snap: btn2 =' + btn2)
            t='{:%Y%m%d-%H%M%S}'.format(datetime.now())
            filename = 'snap'+t+'.jpg'
            cam.capture(filename)
            shutil.copyfile(filename,'/home/pi/Flask/static/latest.jpg')
            btn2 = 'o'
            print('btn2 =' + btn2)

    cam.close()
    print('exiting snaphot mode')


# we are able to make two different requests on our webpage
# GET = we just type in the url
# POST = some sort of form submission like a button

@app.route('/', methods = ['POST','GET'])
def hello_world():

    status = 'off'
    global btn1
    btn1 = 'o'
    global btn2
    btn2 = 'o'
    message = 'All good '

    # if we make a post request on the webpage aka press button then do stuff
    if request.method == 'POST':

        # if we press the turn on button
        if request.form['submit'] == 'Video':
```

```
    print('BP: Recording video')
    status = 'video'
    btn1 = 'v'
    t2 = Thread(target=video)
    t2.start()
    message = 'All good'
elif request.form['submit'] == 'Video Off':
    print('BP: Video off')
    status = 'Idle'
    btn1 = 'o'
    message = 'All good'
elif request.form['submit'] == 'Stills':
    print('BP: Recording stills')
    btn1 = 's'
    t1 = Thread(target=timelapse)
    t1.start()
    status = 'stills'
    message = 'All good'
elif request.form['submit'] == 'Stills Off':
    print('BP: stills off')
    status = 'Idle'
    btn1 = 'o'
    message = 'All good'
elif request.form['submit'] == 'QuickSnap':
    print('BP: QuickSnap')
    status = 'Ready to snap'
    btn1 = 'q'
    t3 = Thread(target=snapstart)
    t3.start()
    message = 'All good'
elif request.form['submit'] == 'QuickSnap Off':
    print('BP:QuickSnap off')
    status = 'Idle'
    btn1 = 'o'
    message = 'All good'
elif request.form['submit'] == 'Take':
    print('BP:Take')
    status = 'Snapshot mode'
```

```
            btn1 = 'q'
            btn2 = 'a'
            message = 'All good'
        elif request.form['submit'] == '_Take_':
            print('BP:Take error')
            status = 'Error'
            message = 'Enable QuickSnap first'
            btn1 = 'o'
        else:
            pass

    temp = round(weather.temperature(),2) #temperature from Enviro pHat
    press = int(weather.pressure()) # pressure
    lux = light.light() # light levels
    df = os.statvfs('/') # check if we're running out of disk space
    df_size = df.f_frsize * df.f_blocks
    df_avail = df.f_frsize * df.f_bfree
    df_pc = round(100 -(100 * df_avail/df_size),1)
    print(btn1, btn2)

    # the default page to display will be our template with our template
variables
    return render_template('index2.html', message= message,
status=status, temp=temp, press=press, lux=lux, df_pc=df_pc, btn1 = btn1)

if __name__ == "__main__":

    # let's launch our webpage!
    # do 0.0.0.0 so that we can log into this webpage
    # using another computer on the same network later
    # specify port 80 rather than default 5000
    app.run(host='0.0.0.0',port=80,debug=True)
```

The MagPi ESSENTIALS

[CHAPTER ELEVEN] LIVE-STREAM VIDEO & STILLS

Stream video and regular stills to a remote computer

[YOU'LL NEED]

- Camera Module
- WiFi dongle (if not using a Pi 3)
- Remote computer

One of the drawbacks of using SSH or VNC to access your Camera Module-equipped Pi remotely from another computer is that you can't view the camera preview via these methods. To get around this, you'll need to stream live video across the network. While there are various methods available for doing this, in this chapter we'll show you how to create a client-server setup for video streaming using the Picamera Python library. We'll also explore how to send a stream of stills over the network.

>STEP-01

Server-side script

Note: If you are using a Linux machine for playback of the video stream, there is an easier method, explained in step 3.

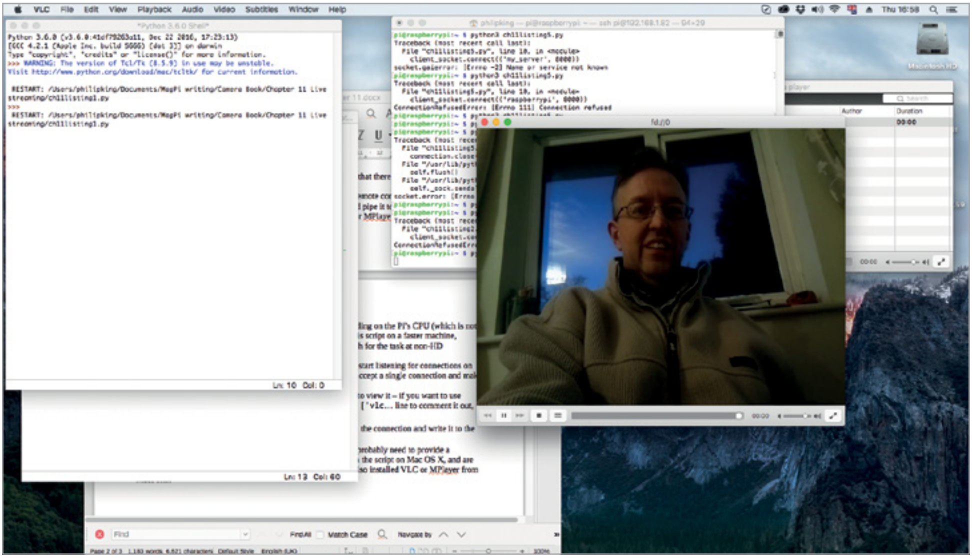
First, we'll write a Python server script, **ch11listing1.py**, for the remote computer that will read the video stream (which we've yet to write to the code to create) and pipe it to a media player. Note that you can't use a Raspberry Pi for playback, since neither VLC nor MPlayer is capable of using the GPU for decoding. They thus attempt to perform video decoding on the Pi's CPU, which is not powerful enough for the task. Therefore you will need to run this script on a faster machine, although even an Atom-powered netbook should be quick enough for the task at non-HD resolutions.

After importing the libraries required at the top of the script, we start listening for connections on 0.0.0.0:8000, i.e. all IP addresses on the local network. We then accept a single connection and make a file-like object out of it.

In the **`try:`** block, we run a media player from the command line to view it – if you want to use MPlayer instead of VLC, add a **`#`** to the start of the **`cmdline = ['vlc`**… line to comment it out, and remove the **`#`** from the **`cmdline = ['mplayer`**… line.

In the **`while True:`** loop, we repeatedly read 1kB of data from the connection and write it to the selected media player's stdin (standard input) to display it.

Note: If you run this script on Windows or Mac OS X, you will probably need to provide a complete path to the VLC or MPlayer executable/app. If you run the script on Mac OS X, and are using Python installed from MacPorts, please ensure you have also installed VLC or MPlayer from MacPorts.



VLC File Edit View Playback Audio Video Subtitles Window Help

```
Python 3.6.0 (v3.6.0:41df79263a11, Dec 22 2016, 17:23:13)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> WARNING: The version of Tcl/Tk (8.5.9) in use may be unstable.
Visit http://www.python.org/download/mac/tcltk/ for current information.

 RESTART: /Users/philipking/Documents/MagPi writing/Camera Book/Chapter 11 Live
streaming/ch11listing1.py
>>>
 RESTART: /Users/philipking/Documents/MagPi writing/Camera Book/Chapter 11 Live
streaming/ch11listing1.py
```

philipking — pi@raspberrypi: ~ — ssh pi@192.168.1.82 — 94×29

```
pi@raspberrypi:~ $ python3 ch11listing5.py
Traceback (most recent call last):
  File "ch11listing5.py", line 10, in <module>
    client_socket.connect(('my_server', 8000))
socket.gaierror: [Errno -2] Name or service not known
pi@raspberrypi:~ $ python3 ch11listing5.py
Traceback (most recent call last):
  File "ch11listing5.py", line 10, in <module>
    client_socket.connect(('raspberrypi', 8000))
ConnectionRefusedError: [Errno 111] Connection refused
pi@raspberrypi:~ $ py
pi@raspberrypi:~ $ py
pi@raspberrypi:~ $ py
Traceback (most rece
  File "ch11listing5.
    connection.close
  File "/usr/lib/pyt
    self.flush()
  File "/usr/lib/pyt
    self._sock.senda
socket.error: [Errno
pi@raspberrypi:~ $ p
pi@raspberrypi:~ $ p
Traceback (most rece
  File "ch11listing2.
    client_socket.co
ConnectionRefusedErr
pi@raspberrypi:~ $ p
```

The remote server script reads the video stream and pipes it to a media player

>STEP-02

Client-side script

Now we'll create a client script, **ch11listing2.py**, on the Raspberry Pi with the Camera Module equipped. This will connect to the network socket of our server (playback) script to send a video stream to it.

After importing the required libraries at the top, we connect a client socket to **my_server:8000** – you'll need to change **my_server** to the host name of your server (the computer that will playing back the stream). If you are using a Linux PC or Mac, just type **hostname** in a terminal window to find it out; in Windows, it's the Computer Name in Control Panel > System.

We then create a file-like object from the network socket before triggering the camera to start recording. In this example we're using a resolution of 640 × 480 with a frame rate of 24fps, but you can adjust these numbers to your requirements. We've also set the camera to record for 60 seconds with **camera.wait_recording(60)**; again, you can change this number to suit your preference.

Run the server script, then the client script. You should see the video stream played in your chosen media player. You may notice some latency; this is normal and due to buffering by the media player.

>STEP-03

Server-side script

As mentioned above, if you're using a Linux PC for playback of the video stream, there is a much quicker and easier way to achieve what we've done in steps 1 and 2. On the server (playback) machine, enter the following command into a terminal window:

```
nc -l 8000 | vlc --demux h264 -
```

Then, on the client – the Raspberry Pi with the Camera Module – issue the following command:

```
raspivid -w 640 -h 480 -t 60000 -o - | nc my_server 8000
```

...replacing **`my_server`** with the server's host name.

>STEP-04

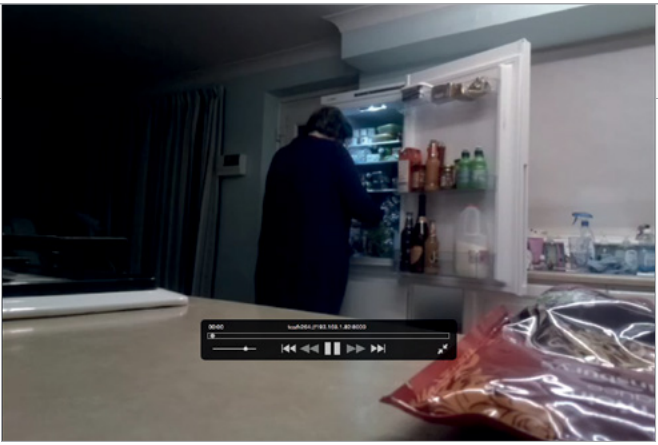
Switch it around

An alternative method is to reverse the direction so that the Raspberry Pi acts as a server. We can then get it to wait for a connection from the client before streaming video. Enter the **ch11listing3.py** example on the Pi and run it.

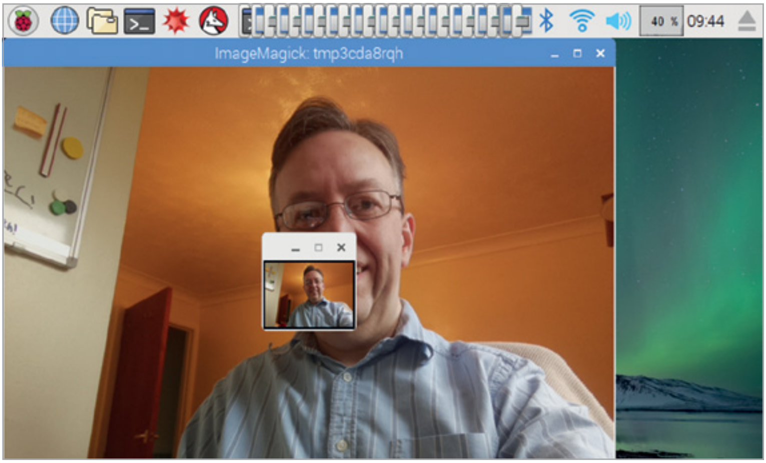
The big advantage of this method is that you then only need to use a single command to initiate playback on the remote computer:

```
vlc tcp/h264://my_pi_
address:8000/
```

...replacing **`my_pi_address`** with your Pi's IP address (discovered using **`ifconfig`**). Or, in VLC running on the desktop, go to File > Open Network and enter the same address: **`tcp/h264://my_pi_address:8000/`**



An alternative is to set the Pi as the server and open the network source in VLC or another media player



The stills are streamed to the remote computer (in this case another Pi) and displayed

>STEP-05

Stream stills

So, we've streamed video over the network. Now let's stream camera stills taken at regular intervals in a variation on a standard time-lapse setup. Entered on a remote computer (which could be another Pi), the server script, **ch11listing4.py**, starts a socket to listen for a connection from the Raspberry Pi with the camera. At the top, we import the required libraries; here we're using PIL to read JPEG files, but alternatives include OpenCV and GraphicsMagick. The script then checks the image length and, if it is not zero, constructs a stream to hold the image data and then reads it from the connection. The **`image.show()`** command will open each image in the default image viewer: it can create a lot of windows if left going for a while! Now to create a client script…

>STEP-06

Stills client script

On the Raspberry Pi with the camera, the client script, **ch11listing5.py**, sends a continual stream of images to the server. We'll use a very simple protocol for communication: first, the length of the image will be sent as a 32-bit integer (in little-endian format), then this will be followed by the bytes of image data. If the length is 0, this indicates that the connection should be closed as no more images will be forthcoming. As before, for connecting the socket, you should replace **`my_server`** in the

Language

>PYTHON 3

DOWNLOAD:
magpi.cc/2pjD97G

script with the host name of the remote computer. We then make a file-like object out of the connection. Before constructing the stream, we start a preview to let the camera warm up for two seconds. Further down, the line **`if time.time() - start > 30:`** limits the streaming period to 30 seconds, though you can alter this to suit your needs.

Note that the server script should be run first to ensure there's a listening socket ready to accept a connection from the client script.

Taking it further, you may want to add a way of closing each image window before the next is generated. Also, rather than simply showing the images, you could use the numerous functions of PIL to process them (see **magpi.cc/2iiDgcC**).

ch11listing1.py

```
import socket
import subprocess
# Start a socket listening for connections on 0.0.0.0:8000
# (0.0.0.0 means all interfaces)
server_socket = socket.socket()
server_socket.bind(('0.0.0.0', 8000))
server_socket.listen(0)
# Accept a single connection and make a file-like object out of it
connection = server_socket.accept()[0].makefile('rb')
try:
    # Run a viewer with an appropriate command line. Uncomment the mplayer
    # version if you would prefer to use mplayer instead of VLC
    cmdline = ['vlc', '--demux', 'h264', '-']
    #cmdline = ['mplayer', '-fps', '25', '-cache', '1024', '-']
    player = subprocess.Popen(cmdline, stdin=subprocess.PIPE)
    while True:
        # Repeatedly read 1k of data from the connection
        # and write it to the media player's stdin
        data = connection.read(1024)
        if not data:
            break
        player.stdin.write(data)
finally:
    connection.close()
    server_socket.close()
    player.terminate()
```

ch11listing2.py

```
import socket
import time
import picamera

# Connect a client socket to my_server:8000
# (change my_server to the hostname of your server)
client_socket = socket.socket()
client_socket.connect(('my_server', 8000))

# Make a file-like object out of the connection
connection = client_socket.makefile('wb')
try:
    camera = picamera.PiCamera()
    camera.resolution = (640, 480)
    camera.framerate = 24
    # Start a preview and let the camera warm up
    camera.start_preview()
    time.sleep(2)
    # Start recording, sending the output to the connection for 60
    # seconds, then stop
    camera.start_recording(connection, format='h264')
    camera.wait_recording(60)
    camera.stop_recording()
finally:
    connection.close()
    client_socket.close()
```

ch11listing3.py

```
import socket
import time
import picamera

camera = picamera.PiCamera()
camera.resolution = (640, 480)
camera.framerate = 24
```

```
server_socket = socket.socket()
server_socket.bind(('0.0.0.0', 8000))
server_socket.listen(0)

# Accept a single connection and make a file-like
# object out of it
connection = server_socket.accept()[0].makefile('wb')
try:
    camera.start_recording(connection, format='h264')
    camera.wait_recording(60)
    camera.stop_recording()
finally:
    connection.close()
    server_socket.close()
```

ch11listing4.py

```
import io
import socket
import struct
from time import sleep
from PIL import Image

# Start a socket listening for connections on 0.0.0.0:8000
# (0.0.0.0 means all interfaces)
server_socket = socket.socket()
server_socket.bind(('0.0.0.0', 8000))
server_socket.listen(0)

# Accept a single connection and make a file-like object out of it
connection = server_socket.accept()[0].makefile('rb')
try:
    while True:
        # Read the length of the image as a 32-bit unsigned int.
        # If the length is zero, quit the loop
        image_len = struct.unpack('<L', connection.read(struct.calcsize('<L')))[0]
        if not image_len:
            break
        # Construct a stream to hold the image data and read the image
        # data from the connection
        image_stream = io.BytesIO()
        image_stream.write(connection.read(image_len))
        # Rewind the stream, open it as an image with PIL
        # and show it in the default image viewer
        image_stream.seek(0)
        image = Image.open(image_stream)
        image.show()
finally:
    connection.close()
    server_socket.close()
```

ch11listing5.py

```
import io
import socket
import struct
import time
import picamera

# Connect a client socket to my_server:8000
# (change my_server to the hostname of your server)

client_socket = socket.socket()
client_socket.connect(('my_server', 8000))

# Make a file-like object out of the connection

connection = client_socket.makefile('wb')
try:
    camera = picamera.PiCamera()
    camera.resolution = (640, 480)
    # Start a preview and let the camera warm up for
    camera.start_preview()
    time.sleep(2)
    # Note the start time and construct a stream to
    # hold image data temporarily (we could write it
    # directly to connection but in this case we want
    # to find out the size of each capture first to keep
    # our protocol simple)

    start = time.time()
    stream = io.BytesIO()
    for foo in camera.capture_continuous(stream, 'jpeg'):

        # Write the length of the capture to the stream
        # and flush to ensure it actually gets sent

        connection.write(struct.pack('<L', stream.tell()))
        connection.flush()

        # Rewind the stream and send the image data over the wire

        stream.seek(0)
        connection.write(stream.read())

        # If we've been capturing for more than 30 seconds, quit

        if time.time() - start > 30:
            break

        # Reset the stream for the next capture

        stream.seek(0)
        stream.truncate()

    # Write a length of zero to the stream to signal we're done

    connection.write(struct.pack('<L', 0))
finally:
    connection.close()
    client_socket.close()
```

[CHAPTER TWELVE]

SET UP A SECURITY CAMERA

Protect your home using motionEyeOS

[YOU'LL NEED]

- Camera Module
- motionEyeOS
- WiFi dongle (if not using a Pi 3)
- Pushover app
- Remote computer

The specialist motionEyeOS distro turns your Raspberry Pi and Camera Module into a fully fledged security camera that can stream a live view, detect motion, and capture video and stills. In this chapter we'll show you how to install it, get started using it, and even send custom push notifications to your phone when motion is detected!

>STEP-01

Install motionEyeOS

motionEyeOS is a Linux distribution that turns a single-board computer into a video surveillance system. To see the list of supported devices and download the relevant distro image, go to **magpi.cc/ 1UCw1Jk**. Note that there are three different versions for Raspberry Pi, so make sure you download the correct one for your model.

With the image downloaded, you can write it to an SD card in a similar fashion to Raspbian, by using dd or Etcher, for example. However, motionEyeOS's creator provides a write utility for Linux and Mac OS X. The advantage of using this is that you can preconfigure the wireless network connection so that you don't have to connect the Pi to your router via Ethernet at first. This is particularly useful if you are using a Pi Zero, which lacks an Ethernet port. To download the utility and make it executable, enter the following commands in a terminal window:

```
curl https://raw.githubusercontent.com/ccrisan/
motioneyeos/master/writeimage.sh
chmod 775 writeimage.sh
```

To write to the SD card and preconfigure the wireless connection, use:

```
./writeimage.sh -d /dev/yoursdcard -i "/path/to/
motioneyeos.img" -n 'yournet:yourkey'
```

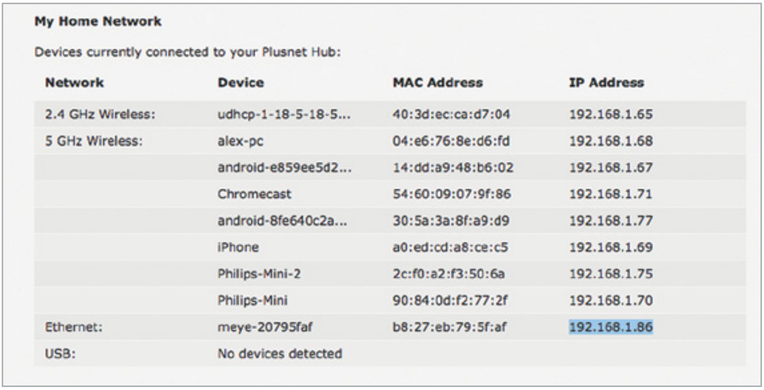
...replacing the generic elements with your own details.

>STEP-02

Alternative wireless method

Another way of preconfiguring wireless connectivity, if you're using a standard card writing method, is to create a file called **wpa_supplicant.conf** containing these lines (with your router's SSID and password):

Find the IP address of the motionEyeOS Pi by visiting your router's homepage



My Home Network

Devices currently connected to your Plusnet Hub:

Network	Device	MAC Address	IP Address
2.4 GHz Wireless:	udhcp-1-18-5-18-5...	40:3d:ec:ca:d7:04	192.168.1.65
5 GHz Wireless:	alex-pc	04:e6:76:8e:d6:fd	192.168.1.68
	android-e859ee5d2...	14:dd:a9:48:b6:02	192.168.1.67
	Chromecast	54:60:09:07:9f:86	192.168.1.71
	android-8fe640c2a...	30:5a:3a:8f:a9:d9	192.168.1.77
	iPhone	a0:ed:cd:a8:ce:c5	192.168.1.69
	Philips-Mini-2	2c:f0:a2:f3:50:6a	192.168.1.75
	Philips-Mini	90:84:0d:f2:77:2f	192.168.1.70
Ethernet:	meye-20795faf	b8:27:eb:79:5f:af	192.168.1.86
USB:	No devices detected		

```
update_config=1
ctrl_interface=/var/run/wpa_supplicant

network={
    scan_ssid=1
    ssid="your_network"
    psk="your_password"
}
```

You will need to turn the file into an executable with **`chmod +x wpa_supplicant.conf`** before moving it to the boot partition of your SD card (alongside start.elf etc.). Note that you must do so before attempting to boot it on the Pi for the first time.

>STEP-03

Remote access

Insert the SD card into your Pi and boot it. There is no need to attach it to a monitor as it won't show much and it's intended to be run headless. Assuming you've preconfigured the wireless connection, it should connect to the router after a couple of minutes. If you have any problems, check the wireless details you entered; if your router syncs 2.4GHz and 5GHz on the same SSID, you may need to split this into separate SSIDs to get a connection. If you're still having trouble, you can connect to the router via an Ethernet cable and set up a wireless connection from the remote web interface later.

Either way, to find the Pi's IP address, just visit your router's homepage (e.g. 192.168.1.254) and view the list of attached devices; your Pi will appear as **meye-** followed by a hex number. Enter the IP address for it in a web browser on a remote computer. You will be presented with a login screen: just enter the default **admin** user name without a password. You can add a password later, as well as a standard user.

>STEP-04

Camera features

Once you're logged in, you will be able to see the live view from the camera, which you can also expand. Open the options menu on the left (the icon is three horizontal parallel lines) to access numerous options; change Layout Columns to 1 to enlarge the standard camera view.

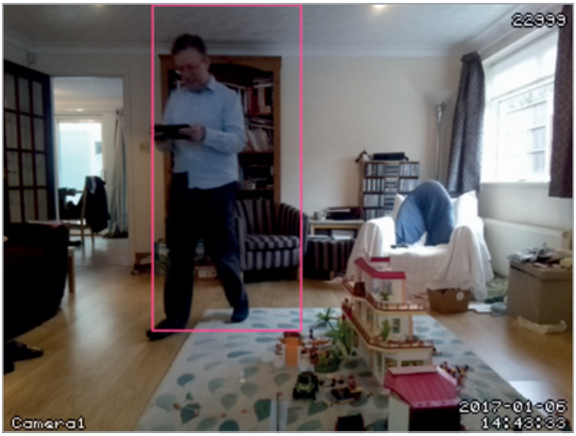
Turning on Advanced Settings reveals a host of additional options. These include camera settings such as video resolution and rotation. You can also adjust motion detection settings and options for capturing stills and movies, which can be viewed via the icons shown on the camera view after you click on it.

The Motion Notifications panel enables you to send yourself an email whenever motion is detected, or call a web hook, or run a command. This last option is what we'll be using for our custom notifications using the Pushover service…

>STEP-05

Create Pushover app

Pushover has a great, easy-to-use API. Before we start, we need to register an application with it. Click on Register Application under the Your Applications heading on the Pushover website (**pushover.net**). Give your app a name – something like RaspiMotion – and then make sure the type is Application. Give your app a quick description (e.g. 'Push notifications sent by my Raspberry Pi') and, if you are



Intruder alert! Any change in the camera view will be detected, triggering stills or video capture feeling creative, upload a custom icon which will show in your Pushover client app whenever a notification is sent.

Once you have created your application, you should have access to an API token/key. This is a unique combination of numbers and letters: keep it a secret! You'll also need your user key, which is shown once you log into Pushover's website. Now you have an app and your API and user keys. The next thing is to write a Python script to tell your Raspberry Pi to work its magic once the script is called upon by motionEyeOS.

>STEP-06

Write Python script

You'll need to SSH into the Pi from a terminal window on a remote computer (or PuTTY on Windows) to do this, using **`ssh admin@IP-address`**. The default user is **admin**, with no password. Our script needs to live in the **data** folder, so go there and create **ch12listing1.py** using nano:

```
cd /data
nano ch12listing1.py
```

Once here, you'll need to type in the code listing, while also including your API token and user key where required. As with any script, we need to make sure it can be executed, otherwise it's nothing more than a fancy collection of text! From the command line, make sure you're in the **data** folder and then type:

```
chmod +x ch12listing1.py
```

Or, if you are using WinSCP, select the **ch12listing1.py** file in the **data** folder, then press **F9**. In the window that appears, change the permissions to 0755 and then click 'OK' to confirm.

>STEP-07

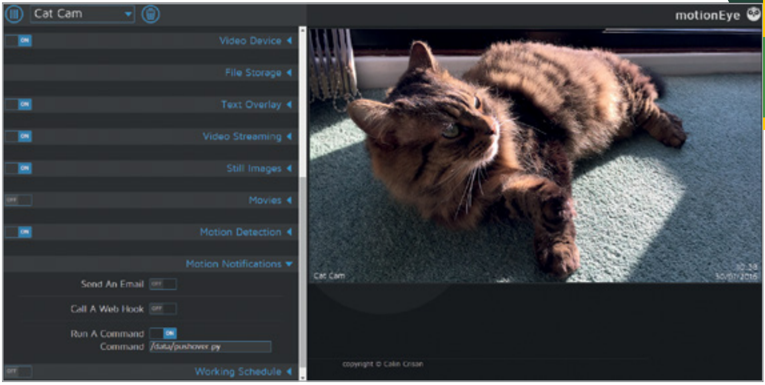
Trigger the script

Now that we have our script, we need to tell motionEyeOS to use it when it detects motion. To do this, log in, go to the Motion Notifications menu and turn on the 'Run A Command' option. You then need to specify which command to run, which will be the Python script you just created – this is **/data/ch12listing1.py**. Then click on Apply to confirm the changes.

Language

>PYTHON 3

DOWNLOAD:
magpi.cc/2pjKhB1



Video Device: ON
File Storage
Text Overlay: ON
Video Streaming: ON
Still Images: ON
Movies: OFF
Motion Detection: ON
Motion Notifications
Send An Email: OFF
Call A Web Hook: OFF
Run A Command: ON
Command: /data/pushover.py
Working Schedule: OFF

In the Motion Notifications menu, set Run A Command to the path of your script

To test it out, you'll need the Pushover app installed on your smartphone or tablet. Wave your hand in front of your camera (or you can do a dance if you're feeling energetic!) and then shortly afterwards you should receive a notification via Pushover, warning you that motion has been detected. Feel free to experiment with the script to customise the message displayed and sound played in Pushover.

ch12listing1.py

```python
import httplib, urllib

conn = httplib.HTTPSConnection("api.pushover.net:443")
conn.request("POST", "/1/messages.json",
  urllib.urlencode({
    "token": "APP_TOKEN",                       # Insert app token here
    "user": "USER_TOKEN",                       # Insert user token here
    "html": "1",                                # 1 for HTML, 0 to disable
    "title": "Motion Detected!",                # Title of the message
    "message": "<b>Front Door</b> camera!",     # Content of the message
    "url": "http://IP.ADD.RE.SS",               # Link to be included in message
    "url_title": "View live stream",            # Text for the link
    "sound": "siren",                           # Define the sound played
  }), { "Content-type": "application/x-www-form-urlencoded" })
conn.getresponse()
```

The MagPi ESSENTIALS

[CHAPTER THIRTEEN]

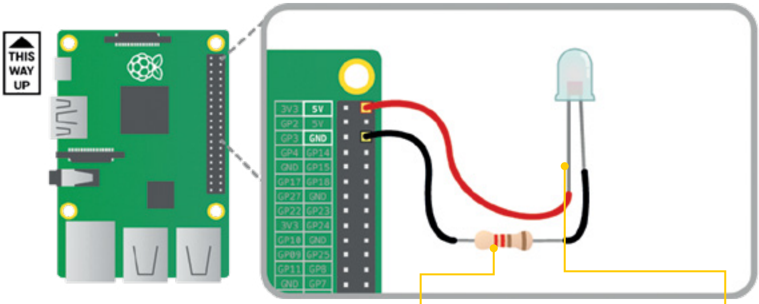
INSTALL A BIRD BOX CAMERA

Observe nesting birds without disturbing them

[YOU'LL NEED]

- Pi NoIR Camera Module
- Bird box
- IR LED
- Female-to-female jumper wires
- Sharp tool and tweezers
- Power source
- WiFi dongle (if not using a Pi 3) or Ethernet cable

Fig 1 Connect an infrared LED



The 222-ohm resistor limits the current flowing through the LED

The longest leg of the IR LED is the anode: connect it to a 5V pin

While it's simple enough to set up a Camera Module in a weatherproof box to observe wildlife in your garden, for this project we'll be installing a camera inside a bird box. Since it'll be dark inside, and we can't use a standard light source, we'll need to use a Pi NoIR Camera Module. 'NoIR' stands for 'no infrared', as it omits the IR filter found in the standard camera. This enables you to use an infrared light source to see in the dark. Note that we'll need to adjust the fixed focus of the camera by unscrewing the lens.

>STEP-01

Set up the Pi NoIR

We can't use a standard light source inside the bird box, since this could attract insects and predators, and so would deter any birds from nesting there. So we need to use a Pi NoIR Camera Module. Apart from the omission of an infrared filter, this works exactly the same way as the standard camera, so you can connect it up to your Raspberry Pi as in chapter 1 and use all the same terminal commands. So, for instance, you can obtain a video preview with:

```
raspivid -t 0
```

You'll notice that everything looks a little strange; this is because you're looking at a combination of visible light and infrared light. To test it out in darkness, turn the lights off, aim a TV remote control at your face and press the buttons to produce an IR light source. You should see your face illuminated in the darkness. The image will be black and white (greyscale), because there are no wavelengths of light from the visible spectrum being detected. However, a black and white image is good enough to allow you to watch what's happening inside a bird box, and it doesn't disturb or interfere with the birds in any way. Press **CTRL+C** to exit the preview.

>STEP-02

Wire up an IR LED

We'll need a suitable infrared light source in the bird box. In this example we're using a single IR LED, but alternatives include small IR lamps and the IR version of the LISIPAROI (**lisiparoi.com**). Our 890nm IR LED is an identical component to the ones found inside TV remote controls; the only difference is that we're going to keep it on constantly when shooting video or stills in the bird box.

As usual, you should turn off the Raspberry Pi before connecting anything up to it. If you've wired up an LED to the Pi GPIO pins before, then please note that this LED needs to be done slightly differently. Since an infrared LED requires more current than the GPIO pins can provide, it needs to be connected directly to the 5V supply of the Pi with a 220-ohm resistor inline; without the resistor, the current will be too high and the LED will burn out after about ten seconds.

Fig 1 (page 85) shows how the LED should be wired up. You'll notice that the LED has two legs, one slightly longer than the other. The longer of the two is called the anode and the shorter is the cathode. The LED needs power to flow into the anode and out of the cathode; if you get the polarity wrong then nothing will happen.

Use a couple of female-to-female jumper wires to make the following connections. Connect the anode (long leg) to 5V, which is the first pin on the outside row on the Pi. Connect the cathode (short leg) to the 220-ohm resistor. Connect the other side of the resistor to ground, which is the third pin in on the outside row on the Pi

>STEP-03

Test the LED

With everything wired up correctly, turn the Pi back on. You'll notice that the infrared LED doesn't appear to be working, but in fact it is.

Your human eyes can’t see it, but the Pi NoIR camera can. Turn on the camera preview again with **`raspivid -t 0`**. Hold the LED in front of the camera and you should see that it is lit. If not, then you may have mixed up the polarity of the anode and cathode. Double-check your wiring against the **Fig 1** diagram. Try turning out the lights and aiming the LED at yourself; don’t look directly into it, however, as infrared light can still cause harm to your eyes. You’ll see from the Pi NoIR camera preview that it will illuminate you quite well. Press **CTRL+C** when you want to exit.

>STEP-04

Remove lens glue

By default, the Pi NoIR Camera Module has a fixed focal length of 50cm and depth of field of 50cm to infinity. This means that objects will only appear in focus if they’re at least 50cm away from the lens of the camera. The bird box which we are using in this example has an interior height of 18cm, and yours may be similar, so we’ll definitely need to know how to shorten the focal length.

Like the standard Camera Module, the Pi NoIR has a lens that can rotate to adjust the focus. It’s held in place by three blobs of glue, however, marked as A, B, and C in **Fig 2**. With the camera

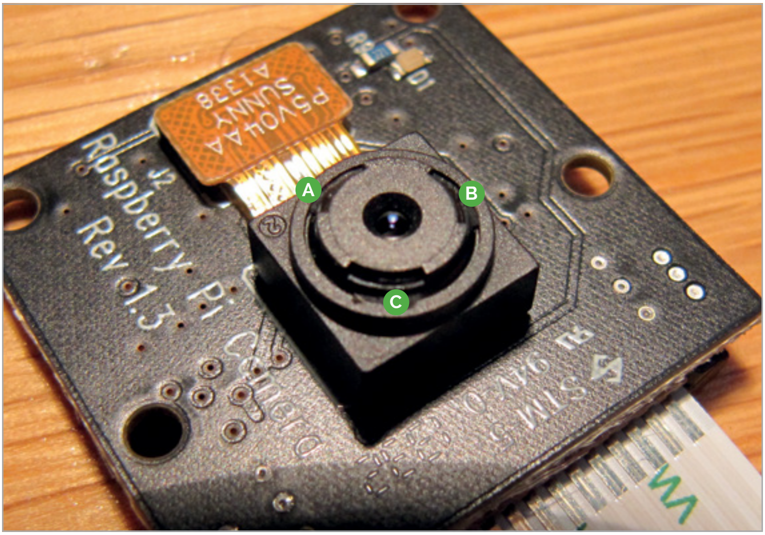


Fig 2 The three blobs of glue securing the lens are found at positions A, B, and C

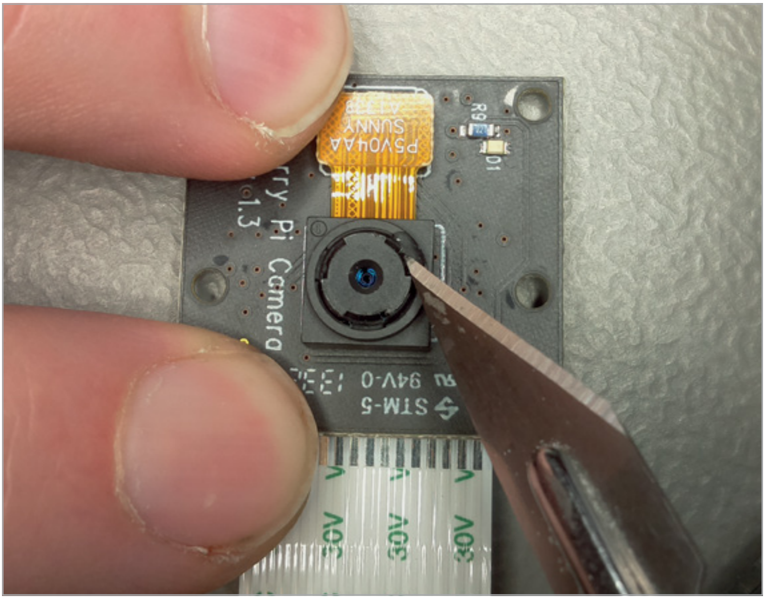


Fig 3 Be careful not to damage the lens, or your fingers, while scraping away the glue

disconnected, you'll need to dig these out using a sharp tool like a needle, scalpel, or a dental pick, as in **Fig 3**. It's easier than it sounds and only takes about five minutes, but be careful not to cut your fingers. Children should only do this under adult supervision for safety, especially if a scalpel is being used.

The orange connector with the word SUNNY printed on it can pop out when you're scraping the glue away; don't worry, though, because it pops right back in. While the camera may look a bit scruffy afterwards, it's unlikely that you'll damage it unless you're very heavy-handed, in which case it's your own responsibility!

>STEP-05

Adjust the focus

Once you're satisfied that you have removed all of the glue, use a pair of tweezers or jewellery pliers to grip the inner section of the camera as shown in **Fig 4**; you should then be able to turn it. Carefully rotate it anticlockwise a few times; be careful not to rotate the lens too far, otherwise it will pop out, and it can be a bit tricky to get it back in and on the thread. If this does happen, though, just put it back in gently

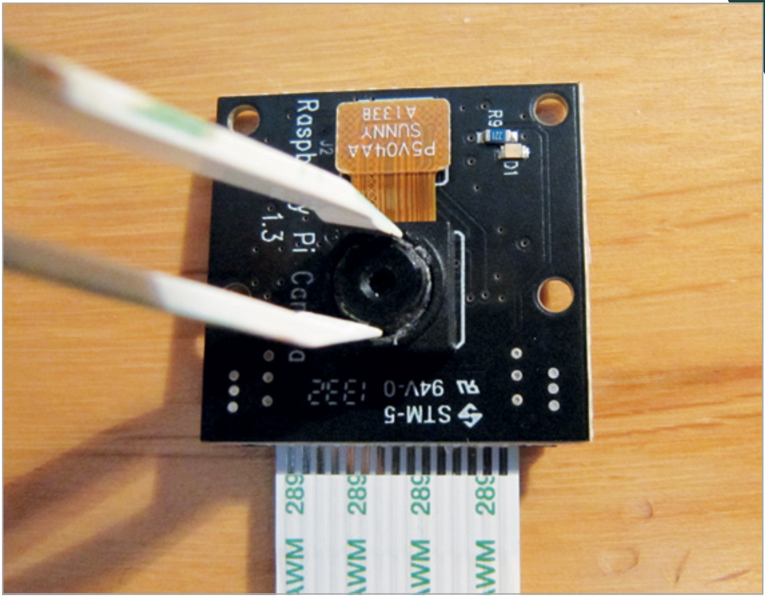


Fig 4 Use tweezers to rotate the lens anticlockwise a few times

and rotate clockwise until it catches. You won't need to re-glue the lens after adjusting it, as it should stay in place.

Now connect the camera back up to the Raspberry Pi. Place a test object with some fine detail – such as a watch or business card – in the bottom of the bird box, then remove its roof (remove the screw), hold the camera at the approximate height of the roof, and look at the camera preview. You may wish to put something under the object at this point to simulate the height of a nest, to make doubly sure that the birds will be in focus. Remember that once birds move in, you can't come back and adjust the camera if the focus is wrong.

>STEP-06

Install the camera

Place your finger on the roof, approximately above the centre of the main body of the bird box. Lift up the roof and place your thumb directly below your finger, so that you're pinching the lid as shown in **Fig 5** (overleaf). Your thumb is now where the camera needs to be. Take a pen and mark this spot with a cross. Cut out a rectangle of cardboard approximately 4cm × 2cm (1.5″ × 0.75″) and fold it in half

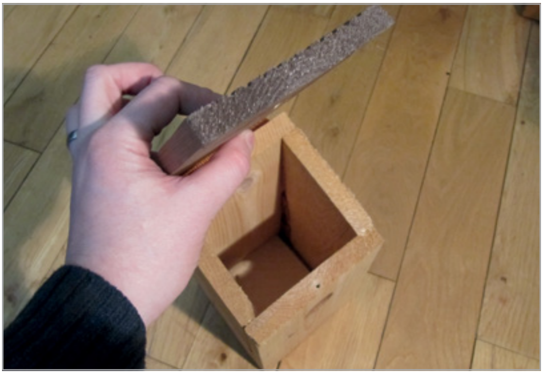


Fig 5 Pinch the lid and then use a pen to mark a cross where your thumb is

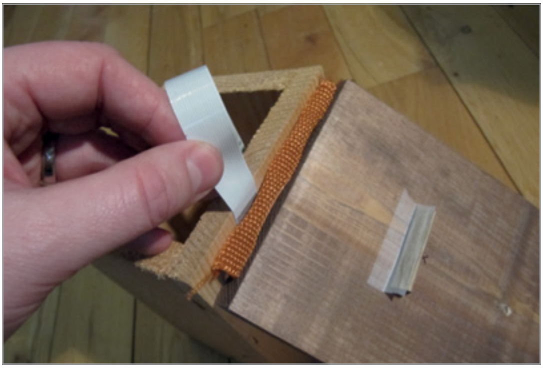


Fig 6 Slide the camera cable between the roof hinge and the back wall of the box

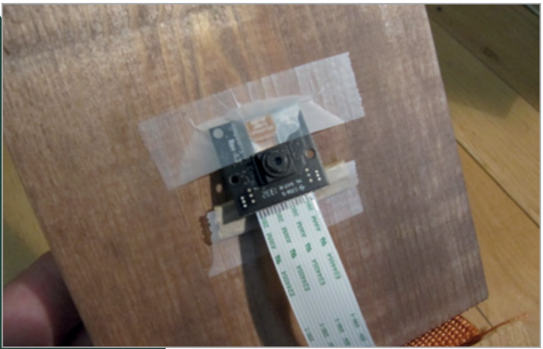


Fig 7 When taped in place, the camera should sit at an angle to compensate for the roof slope

lengthways. Use some tape to secure it to the underside of the roof so that it's a few millimetres below the cross. This is going to be used to compensate for the angle of the roof, so that the camera points directly into the middle of the bird box.

Next, take the Pi NoIR and slide the flexible cable down between the roof hinge and the back wall of the box as shown in **Fig 6**. Do this with the tin connectors facing away from the back wall. In our example, we removed the two middle staples holding the hinge in place, as this enabled the flex to exit the bird box more neatly.

Take some tape and put it across the top of the Pi NoIR board: do not cover the camera lens! Secure the camera in place so that the central lens is directly over the cross that you drew earlier. The camera should sit at an angle as in **Fig 7**. Close the lid and inspect the camera angle from the side: it needs to point directly at the centre of the base of the box. If it doesn't look right, go back and adjust it until you're happy. An alternative to taping it in place would be to use the four mounting holes to screw it to the lid using a wedge of wood instead of the cardboard.

>STEP-07

Add the LED

Secure the infrared LED to the underside of the roof. Don't attach

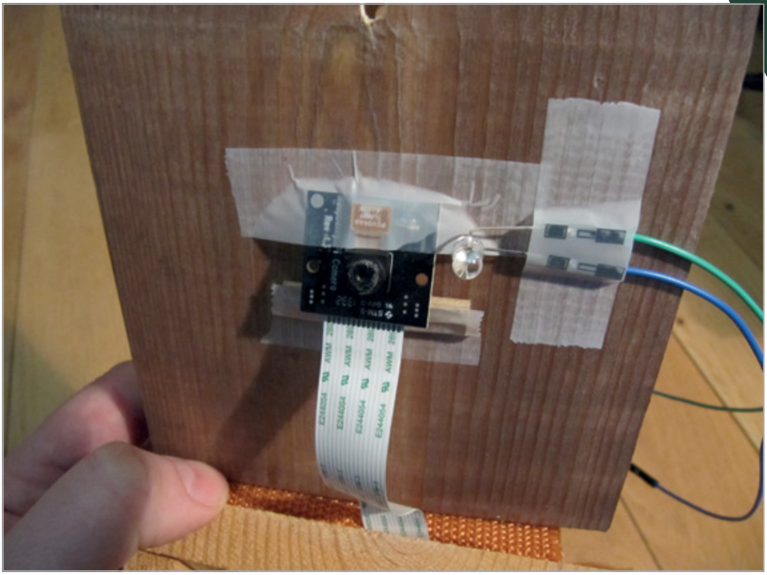


Fig 8 The IR LED is taped to the underside of the roof, not too close to the camera

it too close to the camera, or you'll see a lot of glare on the video. The LED can go anywhere, but it can help to bend its legs by 90 degrees, as shown in **Fig 8**, and secure it to the roof that way. You may also wish to blank off the end of the LED with correction fluid or by filing it down with a nail file. This will prevent any spotlight effect on the video and give a more diffuse light.

>STEP-08

Test it again

Now reconnect the Raspberry Pi and test the focus once again. We recommend connecting the camera flex coming from the back of the bird box to the Pi first. Then connect the LED and resistor, followed by the screen, keyboard, and finally the power supply. When testing this setup, it can be helpful to rest the Raspberry Pi upside-down on the roof of the bird box, but do whatever works best for you.

Boot up the Raspberry Pi as usual and then start the video preview with **`raspivid -t 0`**. With the roof of the bird box closed, you should be able to see the inside in black and white. This shows that the infrared illumination is working; you should even be able to cover the hole and still see the inside. It will look similar to **Fig 9** (overleaf), but will be

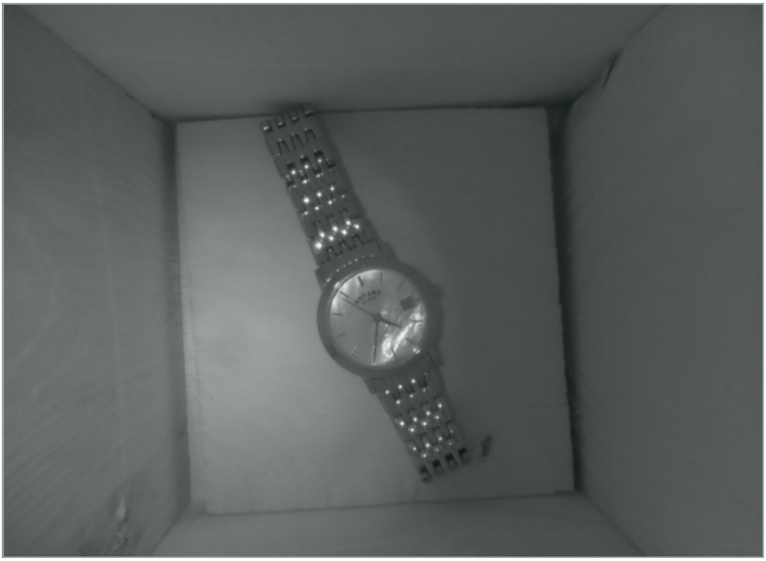


Fig 9 Make sure that the test object is raised up slightly and the text is in focus

slightly more zoomed in. This is because this image was taken using the **raspistill** command and not **raspivid**. If you can't see anything at all, then it's likely the LED is not wired up correctly: double-check the wiring and the polarity of the anode and cathode.

It's now helpful to use an object with some black-on-white text inside the bird box to verify the focus, which is where the watch or business card comes in. Ensure that the text is in focus and readable; adjust the camera focus again as necessary before continuing. Remember to compensate for the nest height. Press **CTRL+C** when you want to stop the camera preview.

>STEP-09

Turn off red LED

Another aspect to consider is the red LED on the (v1) camera. By default, it comes on whenever the camera is on. This will be a huge deterrent to birds moving in, so you should disable it. This can be done by editing the Raspberry Pi configuration file. Enter the command:

```
sudo nano /boot/config.txt
```

Add the following line to the end of the file:

```
disable_camera_led=1
```

Press **CTRL+X**, then **Y** and **ENTER** to save and quit. The changes will only take effect after a reboot: **`sudo reboot`**.

>STEP-10

Weatherproof it

While you can attach the Raspberry Pi directly to the outside of the bird box, an alternative is to use a longer camera cable. Either way, you'll need to put the Pi inside a weatherproof box. Preventing water getting into the bird box should also be a priority. The roof could be sealed using silicone sealant, which is often used to seal the edges of windows and bathroom sinks. Choosing a site which is beneath the overhang of an existing roof will help a lot, as this means that the bird box will not be rained on directly.

Lastly, you need to consider how you will get power and an internet connection to the bird box? You could use a wireless USB dongle, or the built-in wireless LAN of a Raspberry Pi 3, but Ethernet is more reliable for streaming video, especially in built-up areas that have a lot of wireless traffic.

>STEP-11

Obtain images

With everything installed, connected, and powered up, you can SSH into your Pi from another computer (see **magpi.cc/1GULmTr** for details) to control it remotely. You are then able to enter standard terminal commands such as **`raspistill`** and **`raspivid`** to obtain stills (including time-lapses – see chapter 3) and video footage. You could also write one or more Python scripts using the Picamera library.

Note that you can't view the live camera preview via SSH (or VNC). However, you are able to live-stream video from the bird box. This could be achieved using a client-server setup, as described in chapter 11, to pipe the output to a video player on the client computer. Alternatively, you could make use of an internet video service offering live streaming, such as YouTube (see **magpi.cc/2j8rWnv** for details).

The MagPi ESSENTIALS

[CHAPTER FOURTEEN]

QUICK REFERENCE

To help you get to grips with the Camera Module, here's a reference guide to the hardware, commands, and the Picamera Python library

01. CAMERA HARDWARE

Find out all about the Camera Module hardware and its sensor input modes

The first thing to note about the Raspberry Pi Camera Module is that it's akin to a smartphone camera, most notably because it features a rolling shutter. So, when capturing an image, it reads out the pixels from the sensor one row at a time. Unlike the global shutter on a DSLR camera, it also lacks a physical shutter that covers the sensor when not in use.

In addition, the Camera Module acts more like a video camera than a stills camera, as it is rarely idle. Once initialised, it is constantly streaming rows of frames down the ribbon cable to the Raspberry Pi for processing. Numerous background tasks include automatic gain control, exposure time, and white balance. That's why it's best to give it a couple of seconds or more once activated, to adjust the exposure levels and gains before capturing an image.

For more details on how the camera hardware works, see the Picamera documentation at **magpi.cc/2kdHDql**.

[FIXED FOCUS]

The Camera Module has a fixed focal length of 50cm and depth of field of 50cm to infinity. This means that objects will only appear in focus if they're at least 50cm away from the lens of the camera. However, it is possible to alter this by carefully scraping away the blobs of glue holding the lens in place and then unscrewing it slightly to shorten the focal length – this is done at your own risk! See steps 4 and 5 in chapter 13 for more details.

Camera Module versions

The original Camera Module features a 5MP OmniVision sensor, while the later v2 has an 8MP Sony IMX29. Here are the key specs for both…

	Camera Module v1	Camera Module v2
Sensor	OmniVision OV5647	Sony IMX219
Sensor resolution	2592 × 1944 pixels (5MP)	3280 × 2464 pixels (8MP)
Sensor image area	3.76 × 2.74 mm	3.69 × 2.81 mm
Pixel size	1.4 µm × 1.4 µm	1.12 µm × 1.12 µm
Optical size	1/4	1/4
Video modes	1920 × 1080, up to 30fps 1280 × 720, up to 60fps 640 × 480, up to 90fps	1920 × 1080, up to 30fps 1280 × 720, up to 90fps 640 × 480, up to 90fps

Sensor input modes

By default, the camera switches automatically between sensor input modes according to parameters of the **`raspistill`** or **`raspivid`** command given. However, you can force the sensor into any of seven

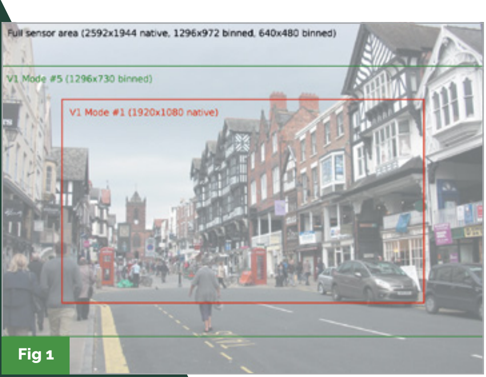

Full sensor area (2592x1944 native, 1296x972 binned, 640x480 binned)
V1 Mode #5 (1296x730 binned)
V1 Mode #1 (1920x1080 native)

Fig 1

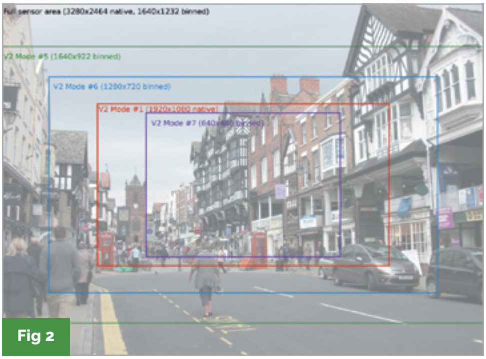

Full sensor area (3280x2464 native, 1640x1232 binned)
V2 Mode #5 (1640x922 binned)
V2 Mode #6 (1280x720 binned)
V2 Mode #1 (1920x1080 native)
V2 Mode #7 (640x480 binned)

Fig 2

discrete modes, as shown below for each Camera Module version, by using the **-md** switch (or **sensor_mode** constructor in Picamera).

Note that you'll still need to specify the resolution and frame rate manually, which should be within the stated range. Modes with a partial field of view are captured from the centre of the sensor, as shown in **Fig 1** and **Fig 2** for Camera Module v1 and v2 respectively.

Camera Module v1

Mode	Resolution	Aspect Ratio	Framerates	Video	Image	FoV	Binning
1	1920 × 1080	16:9	1–30fps	•		Partial	None
2	2592 × 1944	4:3	1–15fps	•	•	Full	None
3	2592 × 1944	4:3	0.1666–1fps	•	•	Full	None
4	1296 × 972	4:3	1–42fps	•		Full	2 × 2
5	1296 × 730	16:9	1–49fps	•		Full	2 × 2
6	640 × 480	4:3	42.1–60fps	•		Full	4 × 4
7	640 × 480	4:3	60.1–90fps	•		Full	4 × 4

Camera Module v2

Mode	Resolution	Aspect Ratio	Framerates	Video	Image	FoV	Binning
1	1920 × 1080	16:9	0.1–30fps	•		Partial	None
2	3280 × 2464	4:3	0.1–15fps	•	•	Full	None
3	3280 × 2464	4:3	0.1–15fps	•	•	Full	None
4	1640 × 1232	4:3	0.1–40fps	•		Full	2 × 2
5	1640 × 922	16:9	0.1–40fps	•		Full	2 × 2
6	1280 × 720	16:9	40–90fps	•		Partial	2 × 2
7	640 × 480	4:3	40–90fps	•		Partial	2 × 2

02. COMMAND-LINE OPTIONS

A guide to the options available when controlling the Camera Module from the command line

Common options

When using **raspistill** or **raspivid** from the command line, you have access to an array of useful switches to change numerous parameters...

Preview window settings

Preview position/size

--preview or **-p** (x,y,w,h)

Allows the user to define the size of the preview window (with **w** and **h** values) and its location on the screen (**x** and **y**). Note that this will be superimposed over the top of any other windows/graphics. For instance, to set its top-left corner at (100, 100) and give it dimensions of 300 × 200, use: **-p 100,100,300,200**.

Fullscreen preview mode

--fullscreen or **-f**

Forces the preview window to use the whole screen. Note that the aspect ratio of the incoming image will be retained, so there may be bars on some edges.

No preview window

--nopreview or **-n**

Disables the preview window completely. Note that even though the preview is disabled, the camera will still be producing frames, so it will be using power.

Preview opacity

`--opacity` or **`-op`**

Sets the opacity of the preview window; 0 = invisible, 255 = fully opaque.

Camera control options

Image width

`--width` or **`-w`**

Sets the width of the resulting image or video. For stills, up to 2592 (Camera Module v1) or 3820 (v2). For video, up to 1920.

Image height

`--height` or **`-h`**

Sets the height of the resulting image or video. For stills, up to 1944 (Camera Module v1) or 2464 (v2). For video, up to 1080.

Image rotation

`--rotation` or **`-rot`** (0 to 359)

Sets the rotation of the preview and saved image. Note that only 0, 90, 180, and 270 degree rotations are supported (other values are rounded down).

Horizontal flip

`--hflip` or **`-hf`**

Flips the preview and saved image horizontally.

Vertical flip

`--vflip` or **`-vf`**

Flips the preview and saved image vertically. Note: Using **`-hf`** and –**`vf`** together is equivalent to a 180° rotation.

Output to file

`--output` or **`-o`**

Specifies the output file name. If this is not specified, no file is saved. If the file name is '–', then all output is sent to stdout, which is handy when using another application that expects image or video data through a standard input.

Timeout

`--timeout` or **`-t`**

The program will run for this length of time; the default is five seconds. If output is specified, it will then take a capture with **`raspistill`**. If using **`raspivid`**, this is the length of the recording.

Verbose information

`--verbose` or **`-v`**

Outputs verbose debugging information during the run.

Sharpness

`--sharpness` or **`-sh`** (−100 to 100)

Sets the sharpness of the image. 0 is the default.

Contrast

`--contrast` or **`-co`** (−100 to 100)

Sets the contrast of the image. 0 is the default.

Brightness

`--brightness` or **`-br`** (0 to 100)

Sets the brightness of the image. 50 is the default. 0 is black, 100 is white.

Saturation

`--saturation` or **`-sa`** (−100 to 100)

Sets the colour saturation of the image. 0 is the default.

ISO

`--ISO` or **`-ISO`** (100 to 800)

Sets the ISO to be used for captures. In effect, this adjusts the light sensitivity of the sensor.

EV compensation

`--ev` or **`-ev`** (−10 to 10)

Sets the EV compensation of the image. Default is 0.

Exposure mode

`--exposure` or **`-ex`**

Sets the exposure mode to any of the following: **`auto`**, **`night`**, **`nightpreview`**, **`backlight`**, **`spotlight`**, **`sports`**, **`snow`**, **`beach`**, **`verylong`** (long exposure), **`fixedfps`** (for video only), **`antishake`**, or **`fireworks`**.

Note that not all of these settings may be implemented, depending on camera tuning.

Automatic white balance (AWB)

`--awb` or **`-awb`**

Set the AWB mode to any of the following: **`off`**, **`auto`**, **`sun`**, **`cloud`**, **`shade`**, **`tungsten`**, **`fluorescent`**, **`incandescent`**, **`flash`**, or **`horizon`**.

Image effect

`--imxfx` or **`-ifx`**

Sets an effect to be applied to the image. Choose from the following: **`none`**, **`negative`**, **`solarise`**, **`posterise`**, **`sketch`**, **`denoise`**, **`emboss`**, **`oilpaint`**, **`hatch`**, **`gpen`** (graphite sketch effect), **`pastel`**, **`watercolour`**, **`film`**, **`blur`**, **`saturation`** (colour saturate the image), **`colourswap`**, **`washedout`**, **`colourpoint`**, **`colourbalance`**, or **`cartoon`**.

Colour effect

`--colfx` or **`-cfx`** (U:V)

The supplied U and V parameters (range 0 – 255) are applied to the U and Y (colour) channels of the image. For example, **`--colfx 128:128`** will result in a monochrome image.

Demo mode

`--demo` or **`-d`**

Cycles through the range of camera options. No capture is taken, and the demo will end at the end of the timeout period. The time between cycles should be specified in milliseconds.

Metering mode

`--metering` or **`-mm`**

Specifies the metering mode used for the preview and capture. Choose from **`average`**, **`spot`**, **`backlit`**, or **`matrix`**.

Sensor region of interest

--roi or **-roi** (x,y,w,h)

Allows the specification of the area of the sensor to be used as the source for the preview and capture. This is defined as x,y for the top-left corner, and a width and height, with all values in normalised coordinates (0.0 to 1.0). So, to set a ROI at halfway across and down the sensor, and a width and height of a quarter of the sensor, use: **-roi 0.5,0.5,0.25,0.25**.

Shutter speed

--shutter or **-ss**

Sets the shutter speed to the specified value (in microseconds). The upper limit is approximately 6000000µs (6000ms, 6s) for the Camera Module v1, and 10000000µs (10000ms, 10s) for the v2.

Dynamic range compression (DRC)

--drc or **-drc**

DRC changes images by increasing the range of dark areas, while decreasing the brighter areas. This can improve the image in low light settings. Choose from: **off** (default), **low**, **medium**, or **high**.

Image statistics

--stats or **-st**

This displays the exposure, analogue and digital gains, and AWB settings used.

AWB gains

--awbgains or **-awbg**

Sets red and blue gains (as floating point numbers) to be applied when **-awb off** is set. For instance, **-awbg 1.5,1.2**.

Sensor input mode

--mode or **-md**

Sets a specified sensor mode, disabling the automatic selection. See ‘Camera Hardware’ section (page 97) for more details.

Annotate flags/text

`--annotate` or **`-a`**

Adds some text and/or metadata to the image. Metadata is indicated using a bitmask notation, so add them together to show multiple parameters. For example, 12 will show time(4) and date(8), since 4+8=12. Text may include date/time placeholders by using the '%' character, as used by strftime (**magpi.cc/2kiJGt5**).

Value	Meaning	Example Output
-a 4	Time	20:09:33
-a 8	Date	02/14/17
-a 12	4+8=12 Show the date(4) and time(8)	20:09:33 10/28/15
-a 16	Shutter Settings	
-a 32	CAF Settings	
-a 64	Gain Settings	
-a 128	Lens Settings	
-a 256	Motion Settings	
-a 512	Frame Number	
-a 1024	Black Background	
-a "ABC"	Show some text	ABC
-a 4 -a "ABC %Y-%m-%d %X"	Show custom formatted date/time	ABC 2017-02-17 20:09:33
-a 8 -a "ABC %Y-%m-%d %X"	Show custom formatted date/time	ABC 2017-02-17 20:09:33

Extra annotation parameters

`--annotateex` or **`-ae`**

Specifies annotation size, text colour, and background colour. Colours are in hex YUV format. Size ranges from 6 to 160; default is 32. Asking for an invalid size should give you the default.

Examples:
`-ae 32,0xff,0x808000 -a "Text"` gives size 32 white text on black background.

`-ae 10,0x00,0x8080FF -a "Text"` gives size 10 black text on white background.

[TWO CAMERAS]

Since the Raspberry Pi has only one CSI connector for a camera, the use of two cameras is only possible with a Compute Module (see magpi.cc/2klAggv). In this case, the following commands may be used for stereoscopic images and video.

`--camselect` or `-cs` – Selects which camera to use. Use 0 or 1.

`--stereo` or `-3d` – Selects stereoscopic mode.

`--decimate` or `-dec` – Half width/height of stereo image.

`--3dswap` or `-3dswap` – Swaps camera order for stereoscopic.

Photo options

The following options are only available when using the **`raspistill`** command (and most of them also when using **`raspiyuv`**).

Time-lapse mode

`--timelapse` or **`-tl`**

The specific value is the time between shots in milliseconds. Note that you should specify %04d at the point in the file name where you want a frame count number to appear. So, for example, the following code will produce a capture every two seconds, over a total period of 30 seconds, named image0001.jpg, image0002.jpg and so on, through to image0015.jpg:

```
-t 30000 -tl 2000 -o image%04d.jpg
```

If a time-lapse value of 0 is entered, the application will take pictures as fast as possible. Note that there's a minimum enforced pause of 30ms between captures to ensure that exposure calculations can be made.

Image quality

`--quality` or **`-q`**

Sets JPEG quality, from 0 to 100.

Raw data

`--raw` or **`-r`**

Adds raw Bayer data to JPEG metadata.

Link latest frame

`--latest` or **`-l`**

Links latest frame to file name specified.

Thumbnail parameters

`--thumb` or **`-th`** (x:y:quality)

Allows specification of the thumbnail image inserted into the JPEG file. If not specified, defaults are a size of 64×48 at quality 35. If **`--thumb none`** is specified, no thumbnail information will be placed in the file; this reduces the file size slightly.

Encoding for output file

`--encoding` or **`-e`**

Valid options are jpg, bmp, gif, and png. Note that unaccelerated image types (GIF, PNG, BMP) will take much longer to save than JPG, which is hardware accelerated. Also, the file name suffix is completely ignored when deciding the encoding of a file.

EXIF tag

`--exif` or **`-x`** (format as 'key=value')

Allows the insertion of specific EXIF tags into the JPEG image. You can have up to 32 EXIF tag entries. This is useful for tasks like adding GPS metadata. For example, to set the longitude to 5 degrees, 10 minutes, 15 seconds, use:

```
--exif GPS.GPSLongitude=5/1,10/1,15/1
```

See EXIF documentation for more details on the range of tags available. Setting **`--exif none`** will prevent any EXIF information being stored in the file; this reduces the file size slightly.

Full preview mode

`--fullpreview` or **`-fp`**
Runs the preview window using the full-resolution capture mode. Maximum frame rate in this mode is 15fps, and the preview will have the same field of view as the capture. Captures should happen more quickly, as no mode change is required.

Keypress mode

`--keypress` or **`-k`**
The camera is run for the requested time (**`-t`**), and a capture can be initiated throughout that time by pressing the **ENTER** key. If you are using **`raspivid`**, this will pause or resume shooting video.

Pressing **X** then **ENTER** will exit the application before the timeout is reached. If the timeout is set to 0, the camera will run indefinitely until exited.

With **`raspivid`**, the timeout value will be used to signal the end of recording, but is only checked after each **ENTER** keypress.

Signal mode

`--signal` or **`-s`**
The camera is run for the requested time (**`-t`**), and a capture can be initiated throughout that time by sending a USR1 signal to the camera process; or, with **`raspivid`**, it toggles between paused and recording. This can be done using the **`kill`** command:

```
kill -USR1 <process id of raspistill or raspivid>
```

To find the camera process ID, use **`pgrep raspistill`** or **`pgrep raspivid`**.

Burst mode

`--burst` or `-bm`

Enables burst capture mode, to capture a sequence of images (using time-lapse, `-tl`) without switching back to preview mode between them. This helps to prevent dropped frames when using a short delay.

[RASPIYUV OPTIONS]

The `raspiyuv` command uses most of the same options as `raspistill`. Unsupported ones are `--exif`, `--encoding`, `--thumb`, `--raw`, and `-quality`.

One extra option is `--rgb` or `-rgb`. This forces the image to be saved as RGB data with 8 bits per channel, rather than YUV420.

Note that the image buffers saved in `raspiyuv` are padded to a horizontal size divisible by 32, so there may be unused bytes at the end of each line. Buffers are also padded vertically to be divisible by 16, and in the YUV mode, each plane of Y,U,V is padded in this way.

Video options

The following options are specific to the **`raspivid`** command for shooting video.

Bitrate

`--bitrate` or `-b`

Sets the bitrate for the video. Use bits per second, so 10Mbits/s would be `-b 10000000`. For H.264, 1080p30 a high-quality bitrate would be 15Mbits/s or more. Maximum bitrate is 25Mbits/s (`-b 25000000`), but much over 17Mbits/s won't show noticeable improvement at 1080p30.

Frame rate

`--framerate` or `-fps`

Specifies the frames per second to record. This varies depending on the camera mode used. The maximum is 90fps, when using a resolution of 640 × 480. See the Camera Hardware section for more details.

Video stabilisation

--vstab or **-vs**

Turns on video stabilisation, which attempts to account for camera shake when it is moving.

Preview after encoding

--penc or **-e**

Displays the preview after compression, to show any artefacts. In normal operation, the preview will show the camera output prior to being compressed.

Intra refresh period

--intra or **-g**

Sets the intra refresh period (GoP) rate for the recorded video. H.264 video uses a complete frame (I-frame) every intra refresh period, from which subsequent frames are based. This option specifies the number of frames between each I-frame. Larger numbers here will reduce the size of the resulting video, while smaller numbers make the stream less error-prone.

Quantisation

--qp or **-qp**

Sets the initial quantisation parameter for the stream. Varies from approximately 10 to 40, and will greatly affect the quality of the recording. Higher values reduce quality and decrease file size. Combine this setting with a bitrate of 0 to set a completely variable bitrate.

H.264 profile

--profile or **-pf**

Sets the H.264 profile to be used for the encoding. Options are: **baseline**, **main**, or **high**.

Insert PPS & SPS headers

`--inline` or **`-ih`**

Forces the stream to include PPS and SPS headers on every I-frame. Needed for certain streaming cases, e.g. Apple HLS.

Timed pause/record periods

`--timed` or **`-td`**

Allows video capture to be paused and restarted at specified time intervals. Two values are required: the on time (for recording) and the off time (for paused). The total time of the recording is defined by the timeout option. For example:

```
raspivid -o test.h264 -t 25000 -timed 2500,5000
```

...will record for a period of 25 seconds. The recording will be over a time frame consisting of 2500ms (2.5s) segments with 5000ms (5s) gaps, repeating over the 20s. So the entire recording will actually be only ten seconds long.

Initial state on startup

`--initial` or **`-i`**

Defines whether the camera will start paused or will immediately start recording (when using the **`-td`** option). Options are **`record`** or **`pause`**.

Segment stream

`--segment` or **`-sg`**

Rather than creating a single file, the file is split into segments of approximately the number of milliseconds specified. In order to provide different file names, you should add %04d or similar at the point in the file name where you want a segment count number to appear. For example:

```
--segment 3000 -o video%04d.h264
```

...will produce video clips of approximately 3000ms (3s) long, named video0001.h264, video0002.h264 etc.

Maximum segment number

`--wrap` or **`-wr`**

When outputting segments, this sets the maximum the segment number can reach before it's reset to 1, giving the ability to keep recording segments, but overwriting the oldest one. So, if set to 4 in the previous segment example, the files produced will be video0001.h264, video0002.h264, video0003.h264, and video0004.h264. Once video0004.h264 is recorded, the count will reset to 1, and video0001.h264 will be overwritten.

Initial segment number

`--start` or **`-sn`**

When outputting segments, this is the initial segment number, giving the ability to resume a previous recording from a given segment. The default value is 1.

Circular buffer

`--circular` or **`-c`**

Runs encoded data through circular buffer until triggered, then saves.

Inline motion vectors

`--vectors` or **`-x`**

Outputs inline motion vectors when used with **`-o`**.

Intra refresh type

`--irefresh` or **`-if`**

Sets intra refresh type: **`cyclic`**, **`adaptive`**, **`both`**, or **`cyclicrows`**.

Flush buffers

`--flush` or **`-fl`**

Flushes buffers in order to decrease latency.

Timestamps

`--save-pts` or **`-pts`**

Saves timestamps to file for mkvmerge.

Codec
`--codec` or `-cd`
Specifies the codec to use: H264 (default) or MJPEG.

H.264 level
`--level` or `-lev`
Specifies H.264 level to use for encoding: 4, 4.1, or 4.2.

Raw video
`--raw` or `-r`
Outputs raw video to file when used with `-o`.

Raw format
`--raw-format` or `-rf`
Specifies output format for raw video: `yuv`, `rgb`, or `gray`.

03. PICAMERA PYTHON LIBRARY

Control the Camera Module from Python programs using the Picamera library

PiCamera class

Here are some of the most commonly used methods and options of the PiCamera class.

`start_preview(**options)`
Displays the preview overlay. Options include `fullscreen` (True or False), `window` (x,y,w,h for position and size), `layer`, and `alpha`.

`stop_preview()`
Hides the preview overlay.

```
capture(output, format=None, use_video_port=False,
resize=None, splitter_port=0, bayer=False, **options)
```

Captures an image from the camera and stores it in **output**. If the latter is a string, it's treated as the name of a file; if an object, it's treated as a file-like object and the image data is appended to it.

```
start_recording(output, format=None, resize=None,
splitter_port=1, **options)
```

Starts recording video from the camera, storing it in **output**. If the latter is a string, it's treated as the name of a file; if an object, it's treated as a file-like object and the video data is appended to it.

```
wait_recording(timeout=0, splitter_port=1)
```

Pauses recording for the number of seconds specified in **timeout**. This method is recommended over the standard **time.sleep()**, since it checks for errors during recording and will immediately raise an exception.

```
stop_recording(splitter_port=1)
```

Stops recording video from the camera. The optional **splitter_port** parameter specifies which port of the video splitter the encoder you wish to stop is attached to. Valid values are 0 to 3 (default 1).

```
close()
```

Stops all recording and preview activities and releases all resources associated with the camera; this is necessary to prevent GPU memory leaks. It should always be called once you are finished with the camera (e.g. in the finally section of a try...finally block).

```
capture_continuous(output, format=None, use_video_
port=False, resize=None, splitter_port=0, burst=False,
bayer=False, **options)
```

Captures images continuously from the camera as an infinite iterator. If **output** is a string, each image is stored in a file named after it with the substitution of two values: **(counter)** (a simple incrementer starting at 1) or **(timestamp)**. For example, setting output to **'image(counter).jpg'** would result in image1.jpg, image2.jpg, image3.jpg, etc.

`capture_sequence(outputs, format='jpeg', use_video_port=False, resize=None, splitter_port=0, burst=False, bayer=False, **options)`

Captures a sequence of consecutive images from the camera, as fast as is possible.

`record_sequence(outputs, format='h264', resize=None, splitter_port=1, **options)`

Records a sequence of video clips from the camera. The caller can control how long to record to each item by only permitting the loop to continue when ready to switch to the next output.

`split_recording(output, splitter_port=1, **options)`

Continues the recording in the specified output. When called, the video encoder will wait for the next appropriate split point (an inline SPS header), then will cease writing to the current output (and close it, if it was specified as a file name), and continue writing to the newly specified output.

`add_overlay(source, size=None, **options)`

Adds a static overlay to the preview output.

`remove_overlay(overlay)`

Removes a static overlay from the preview output. The **overlay** parameter specifies the PiRenderer instance that was returned by **add_overlay()**.

`request_key_frame(splitter_port=1)`

Requests the encoder, running on the specified **splitter_port**, to generate a key-frame (full-image frame) as soon as possible.

analog_gain

Retrieves the current analogue gain of the camera.

annotate_text

Retrieves or sets a text annotation for all output.

`awb_gains`
Gets or sets the auto-white-balance gains of the camera, as a tuple (red, blue) – values are between 0.0 and 8.0. This attribute only has an effect when **`awb_mode`** is set to '**`off'`**.

`awb_mode`
Retrieves or sets the auto-white-balance mode of the camera. Possible values are: **`'off'`**, **`'auto'`** (default), '**`sunlight'`**, **`'cloudy'`**, **`'shade'`**, **`'tungsten'`**, **`'fluorescent'`**, **`'incandescent'`**, **`'flash'`**, or **`'horizon'`**.

`brightness`
Retrieves or sets the brightness setting of the camera, as an integer between 0 and 100 (default 50).

`color_effects`
Retrieves or sets the current color effect applied by the camera, as a (u, v) tuple – values are between 0 and 255. Default is None. When set to (128, 128), it results in a black and white image.

`contrast`
Retrieves or sets the contrast setting of the camera, as an integer between -100 and 100 (default 0).

`digital_gain`
Retrieves the current digital gain of the camera.

`drc_strength`
Retrieves or sets the dynamic range compression strength of the camera. Valid values are: **`'off'`** (default), **`'low'`**, **`'medium'`**, or **`'high'`**.

`exposure_compensation`
Retrieves or sets the exposure compensation level of the camera, as an integer between -25 and 25 (default 0). Each increment represents 1/6th of a stop.

`exposure_mode`
Retrieves or sets the exposure mode of the camera. Valid values are: **`'off'`**, **`'auto'`** (default), **`'night'`**, **`'nightpreview'`**, **`'backlight'`**, **`'spotlight'`**, **`'sports'`**, **`'snow'`**, **`'beach'`**, **`'verylong'`**, **`'fixedfps'`**, **`'antishake'`**, or **`'fireworks'`**.

`exposure_speed`
Retrieves the current shutter speed of the camera, in microseconds. The default is 0 (auto).

`flash_mode`
Retrieves or sets the flash mode of the camera. Valid values are: **`'off'`** (default), **`'auto'`**, **`'on'`**, **`'redeye'`**, **`'fillin'`**, or **`'torch`**'.

Note: You must define which GPIO pin the camera is to use for flash (and optional privacy indicator). This is done within the device tree configuration, as detailed in chapter 7.

`frame`
Retrieves information about the current frame recorded from the camera.

`framerate`
Retrieves or sets the frame rate at which video-port based image captures, video recordings, and previews will run. It can be specified as an int, float, or fraction. The default is 30.

Note: The actual sensor frame rate and resolution used by the camera is influenced – but not directly set – by this property.

`hflip`
Retrieves or sets whether the camera's output is horizontally flipped. Default is False.

`image_denoise`
Retrieves or sets whether denoise will be applied to image captures. Default is True.

image_effect
Retrieves or sets the current image effect applied by the camera. Valid values are: **'none'** (default), **'negative'**, **'solarize'**, **'sketch'**, **'denoise'**, **'emboss'**, **'oilpaint'**, **'hatch'**, **'gpen'**, **'pastel'**, **'watercolor'**, **'film'**, **'blur'**, **'saturation'**, **'colorswap'**, **'washedout'**, **'posterise'**, **'colorpoint'**, **'colorbalance'**, **'cartoon'**, **'deinterlace1'**, or **'deinterlace2'**.

image_effect_params
Retrieves or sets the parameters for the current effect, as a tuple of numeric values up to six elements long.

iso
Retrieves or sets the apparent ISO setting of the camera, which represents its sensitivity to light. Lower values tend to produce less 'noisy' images, but operate poorly in low light conditions. Valid values are: 0 (auto), 100, 200, 320, 400, 500, 640, or 800.

led
Sets the state of the camera's LED (v1 only) via GPIO. If the RPi.GPIO library is available and the Python process is run as root via sudo, this property can be used to set the state of the camera's LED as a Boolean value (True is on, False is off).

Note: This doesn't work on the Raspberry Pi 3, due to a GPIO reconfiguration.

meter_mode
Retrieves or sets the metering mode of the camera. Valid values are: **'average'** (default), **'spot'**, **'backlit'**, **'matrix'**.

recording
Returns True if the **start_recording()** method has been called, and no **stop_recording()** call has been made yet.

`resolution`
Retrieves or sets the resolution at which image captures, video recordings, and previews will be captured. It can be specified as a (width, height) tuple, a string formatted 'WIDTHxHEIGHT', or as a string containing a commonly recognized display resolution name (e.g. 'VGA', 'HD', '1080p', etc). The camera must not be closed, and no recording must be active when the property is set.

`rotation`
Retrieves or sets the current rotation of the camera's image. Valid values are: 0 (default), 90, 180, and 270.

`saturation`
Retrieves or sets the saturation setting of the camera, as an integer between -100 and 100 (default 0).

`sensor_mode`
Retrieves or sets the input mode of the camera's sensor. By default, mode 0 is used, which allows the camera to automatically select an input mode based on the requested resolution and framerate. Valid values are currently between 0 and 7. See the Camera Hardware section for more details on modes.

`sharpness`
Retrieves or sets the sharpness setting of the camera as an integer between -100 and 100 (default 0).

`shutter_speed`
Retrieves or sets the shutter speed of the camera in microseconds. Default is 0 (auto). Faster shutter times require greater amounts of illumination and vice versa.

Note: In later firmwares, this attribute is limited by the value of the **`framerate`** attribute. For example, if frame rate is set to 30fps, the shutter speed cannot be slower than 33,333µs (1/fps).

`timestamp`
Retrieves the system time according to the camera firmware.

`vflip`
Retrieves or sets whether the camera's output is vertically flipped. The default value is False.

`video_denoise`
Retrieves or sets whether denoise will be applied to video recordings. The default value is True.

`video_stabilization`
Retrieves or sets the video stabilisation mode of the camera. The default value is False.

Note: The built-in video stabilisation only accounts for vertical and horizontal motion, not rotation.

`zoom`
Retrieves or sets the zoom applied to the camera's input, as a tuple (x, y, w, h) of floating point values ranging from 0.0 to 1.0, indicating the proportion of the image to include in the output (the 'region of interest'). The default value is (0.0, 0.0, 1.0, 1.0), which indicates that everything should be included.

NOTES

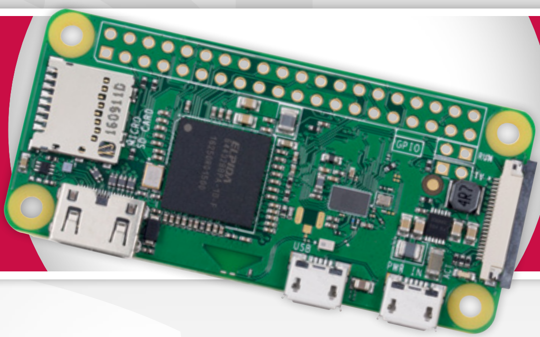
NOTES

The MagPi Magazine

raspberrypi.org/magpi

SUBSCRIBE TODAY AND RECEIVE A

FREE PI ZERO W



Subscribe in print for 12 months today and receive:

- A free Pi Zero W (the latest model)
- Free Pi Zero W case with 3 covers
- Free Camera Module connector
- Free USB and HDMI converter cables

Other benefits:

- Save up to 25% on the price
- Free delivery to your door
- Exclusive Pi offers & discounts
- Get every issue first (before stores)

NEW RASPBERRY PI ZERO W OUT NOW!
The MagPi
RETRO GAMING BLISS
THE OFFICIAL RASPBERRY PI MAGAZINE
BUILD A MAGIC MIRROR
CODING

SAVE UP TO 25%

PLUS

AN OFFICIAL PI ZERO CASE WITH 3 COVERS

AND FREE CAMERA MODULE CONNECTOR AND USB / HDMI CONVERTER CABLES

Pricing:

Get six issues:

£30 (UK)

£45 (EU)

$69 (USA)

£50 (Rest of World)

Subscribe for a year:

£55 (UK)

£80 (EU)

$129 (USA)

£90 (Rest of World)

FREE PI ZERO W!

Get three issues:

£12.99 (UK) (Direct Debit) | $37.50 (US) (quarterly)

How to subscribe:

- magpi.cc/Subs-2 (UK / ROW)
- imsnews.com/magpi (USA)
- Call +44(0)1202 586848 (UK/ROW)
- Call 800 428 3003 (USA)

Search 'The MagPi' on your app store:

GET IT ON Google Play

Available on the App Store

The MagPi ESSENTIALS

raspberrypi.org/**magpi**