

## HARDWARE METAPAPER

## Flat-Field and Colour Correction for the Raspberry Pi Camera Module

Richard W. Bowman\*, Boyko Vodenicharski†, Joel T. Collins\* and Julian Stirling\*

The Raspberry Pi camera module is widely used in open source hardware projects as a low cost camera sensor. However, when the stock lens is removed and replaced with other custom optics the sensor will return a non-uniform background and colour response which hampers the use of this excellent and popular image sensor. This effect is found to be due to the sensor's optical design as well as due to built-in corrections in the firmware, which is optimised for a short focal length lens. In this work we characterise and correct the vignetting and colour crosstalk found in the Raspberry Pi camera module v2, presenting two measures that greatly improve the quality of images using custom optics. First, we use a custom "lens shading table" to correct for vignetting of the image, which can be done in real time in the camera's existing processing pipeline (i.e. the camera's low-latency preview is corrected). The second correction is a colour unmixing matrix, which enables us to reverse the loss in saturation at the edge of the sensor, though this requires post-processing of the image. With both of these corrections in place, it is possible to obtain uniformly colour-corrected images, at the expense of slightly increased noise at the edges of the image.

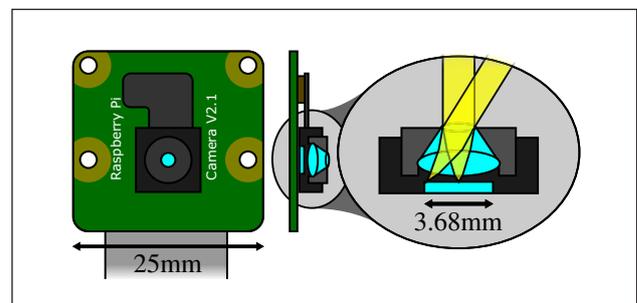
**Keywords:** imaging; image processing; open source; Python

## Metadata Overview

- Main design files: [https://gitlab.com/bath\\_open\\_instrumentation\\_group/picamera\\_cra\\_compensation](https://gitlab.com/bath_open_instrumentation_group/picamera_cra_compensation)
- Target group: microscopists, optical instrument builders, people using custom imaging systems
- Skills required: Python programming – easy/moderate, Arduino – easy, 3D printing – easy;
- See section "Build Details" for more detail.

## Introduction

The Raspberry Pi single-board computer [2] and its accompanying camera module (**Figure 1**) are staple components of many open hardware projects [20, 21, 26]. The ability to embed a small and inexpensive but capable computer enables real-time display and processing of images, and is an excellent basis for scientific instruments that require an imaging component. The Raspberry Pi Camera Module is now in its second version, and is a small (24 × 25 mm) breakout board that connects a Sony IMX219 image sensor [27] to the Raspberry Pi via a MIPI ribbon cable. This is supported by real-time code running in the GPU<sup>1</sup> firmware, that makes it possible to display a high resolution, low-latency video



**Figure 1:** The Raspberry Pi Camera Module v2.1 comprises a Sony IMX219 image sensor and a lens with an effective focal length of 3.04 mm, mounted on a small PCB 25 mm wide. Using the stock lens, light is incident on the edges of the sensor at approximately 30° to the normal.

preview on the Raspberry Pi, and to capture compressed and raw images and video sequences. Popular and well-documented APIs exist in C and Python [17] to control the camera, and there is a strong community of users working with it. There are a wide range of microscopy projects making use of the Raspberry Pi camera including the OpenFlexure Microscope [26], FlyPi [20], a fluorescence imaging system [21], and various others [4, 12, 13, 19]. There have been efforts made in the past to correct the camera for uniform, flat-field response [23]. This manuscript adds real-time flat-field response (referred to as the "lens shading table") and an analysis of colour

\* Department of Physics, University of Bath, GB

† Cavendish Laboratory, University of Cambridge, GB

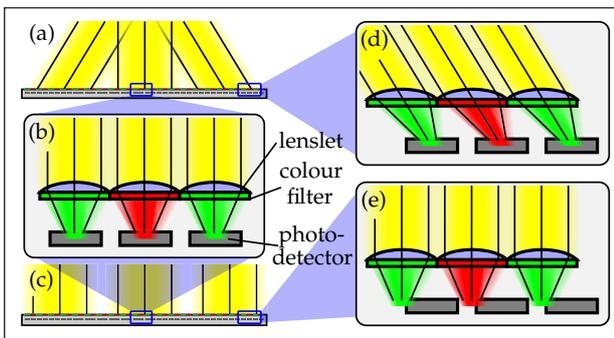
Corresponding author: Richard W. Bowman  
([r.w.bowman@bath.ac.uk](mailto:r.w.bowman@bath.ac.uk))

crosstalk that enables uniform colour response across the sensor.

### Chief Ray Angle compensation

The current camera module (version 2) uses the Sony IMX219 camera module – this is an 8 megapixel back-illuminated CMOS<sup>2</sup> image sensor, intended for use in mobile phones and tablets [27]. It is supplied with a 3.04 mm focal length lens, and the sensor measures 4.6 mm across the diagonal which provides the wide field of view typically expected of a mobile phone camera [3]. The low light performance of the IMX219 is improved by two key features – firstly, the sensor is “back illuminated”, meaning that the light enters the sensor and hits the light-sensitive region without having to pass through any of the readout electronics – this enables a greater fill fraction and consequently improves efficiency. Secondly, a lenslet array is bonded to the image sensor chip. This concentrates the light onto each pixel, further increasing the fraction of the available light that is collected [21].

Each pixel is positioned underneath one lenslet, such that light incident on that lenslet (which may be assumed to be roughly collimated) is focused onto the light-sensitive area of the pixel. This is a relatively simple picture for pixels in the centre of the sensor, where the light is normally incident on the sensor. However, at the edges of the sensor, the stock lens (which is typical of mobile phone camera lenses) means that light is not normally incident, and is travelling at an angle. If the lenslets at the edge of the sensor were still directly above the light-sensitive regions of the pixels, the light would not focus in the correct place, which would lead to a significant drop in efficiency [10]. Instead, the lenslets are spaced slightly closer together than the pixels, which means there is an offset between lenslets and pixels at the edges of the sensor, as shown in **Figure 2**. This ensures that light is correctly focused onto each pixel when using the stock lens, and is known as Chief Ray Angle (CRA) compensation [11, 27]. Unfortunately, this comes at the expense of worse response at the edges of the sensor when used with optics that have a different



**Figure 2:** In a typical webcam or mobile phone camera, light is incident at oblique angles on the edges of the sensor (a). A simplified representation of a pixel includes a lenslet, a colour filter, and a photodetector. The lens focuses light onto the light-sensitive photodetector, shown in (b) for normal incidence. Lenslets are displaced laterally at the edges of the sensor (d). If light is incident normally across the sensor (c), this causes a drop in efficiency and a rise in crosstalk (e).

CRA distribution across the sensor, such as microscope optics (where the CRA is nearly constant).

When using a sensor that has lenslets and CRA compensation, any change to the angle of incidence on the sensor may lead to either or both of a loss of efficiency (due to light being focused on non-light-sensitive regions of the camera) or crosstalk (due to light passing through a lenslet but hitting the wrong pixel on the other side). Crosstalk between pixels is particularly problematic because it leads to confusion between the colour channels [15]. Pixels are arranged in a Bayer pattern, alternating between Red, Green, and Blue filters (with twice as many green pixels), so any light displaced from one pixel to an adjacent pixel will lead to a decrease in saturation. The Bayer pattern means that the four nearest neighbors of each red or blue pixel are all green, thus the crosstalk is strongest between green and the other two channels.

### Image processing pipeline

The Raspberry Pi GPU firmware includes routines for handling data from the camera module. This not only handles the data stream from the sensor (which streams raw pixel data directly to the GPU) but also converts the raw images into RGB images for display, via several denoising and correction steps. The processed, corrected images can then either be displayed (with very low latency), compressed as JPEG or H264, or returned to the CPU<sup>3</sup> for storage, further processing, or streaming over the network. The exact details of the algorithm are proprietary, as the GPU firmware is not open. However, the steps in the pipeline are documented, for example in the documentation of the `picamera` Python module [16]. They include:

- Black level correction (using non-light-sensitive pixels)
- Lens shading correction (compensates for vignetting on the sensor/lens combination)
- White balance correction
- Denoising on raw Bayer data
- Demosaicing of the Bayer-patterned image
- Further denoising on the resulting YUV colour image
- Adjusting sharpness, saturation, brightness, and contrast
- Resizing to the requested resolution
- Optionally, conversion from to Red, Green, Blue (RGB) colour channels

The step of most interest for flat-field correction is the “lens shading correction” step, which is the only one that is not applied uniformly over the whole image. This step multiplies each pixel by a gain value, to compensate for non-uniformity in the response of the sensor. These gain values are stored in a “lens shading table” with a resolution that is  $\frac{1}{64}$  of the sensor’s native resolution, and interpolated smoothly to give values for each pixel. The pipeline for version 1 of the camera module used a fixed lens shading table that compensated for the characteristics of the OmniVision OV5647 sensor and lens. With version 2 of the camera module, a more sophisticated adaptive lens shading algorithm is used, which is capable of adapting to different lenses (to the best of our knowledge, the details of this algorithm are proprietary). It is now possible to supply a custom lens shading table thanks to updated `userland`

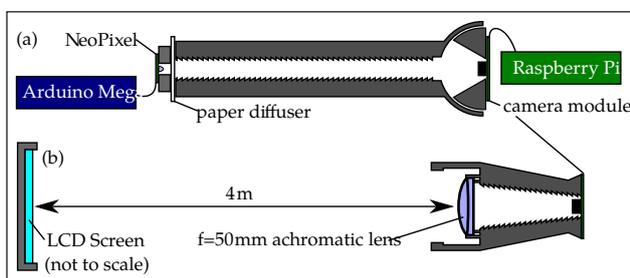
libraries [25], which is what allows us to apply a calibration for vignetting in real time, when not using the stock lens.

### Measuring the sensor's response

As discussed in various forums [1], calibrating a new lens/sensor combination is, in general, a difficult problem requiring careful measurements and specialist equipment. Images must be acquired of a very uniform white target, under even illumination, and care must be taken to avoid artefacts due to stray light or dirt on the target, lens, or sensor. The basic principle, however, is quite simple: acquire a raw image that *should* be uniform and white, then calculate the required gains as the reciprocal of that image – this will then mean that the corrected image of that target will be white. Of course, it also means that any imperfections in the image will be transferred to future images via the lens shading table that is used in subsequent processing.

### Calibration apparatus

In general, the response of an imaging system will depend on the optics used and the sensor. However, there is an important special case, which is that of an imaging system that is very close to uniform, with light normally incident across the whole sensor. As the IMX219 sensor, like the sensor in most mobile phone cameras, is only 4.6 mm across, this assumption is reasonable when using most good quality lenses with a focal length that is much longer than the sensor size. This includes a great many lenses designed for traditional or digital SLR<sup>4</sup> cameras (where the film or sensor is typically 10× larger than our sensor) and most microscopes. If we need only ensure that the sensor is uniformly illuminated, at normal incidence, then it is not necessary to image a perfectly uniform test target – instead we can simply illuminate the sensor with a light source that is spatially quite small, placed some distance from the sensor (to ensure uniform, approximately collimated illumination). Such an arrangement is easy to construct using a “NeoPixel” tri-colour LED<sup>5</sup> and a 3D printed collimation tube, as shown in **Figure 3**. In order to avoid issues due to PWM<sup>6</sup> control of the LEDs causing stripes in the image due to the camera's rolling shutter, the light



**Figure 3:** (a) The calibration jig where a NeoPixel RGB LED illuminates a diffuser, and then provides uniform illumination over the camera sensor. The camera module is able to rotate to vary the angle of incidence. (b) A simple imaging set-up, where a computer screen is imaged using a ThorLabs AC127-050-A achromatic lens. The plastic housing in (b) is an OpenFlexure Microscope infinity corrected optics module, without the objective present. Printable files for both set-ups are available in the project repository [5].

from the NeoPixel was deliberately attenuated by placing a diffuser (white paper) 8 mm from the LED. This meant that the LED could be set to full brightness, without saturating the camera's sensor.

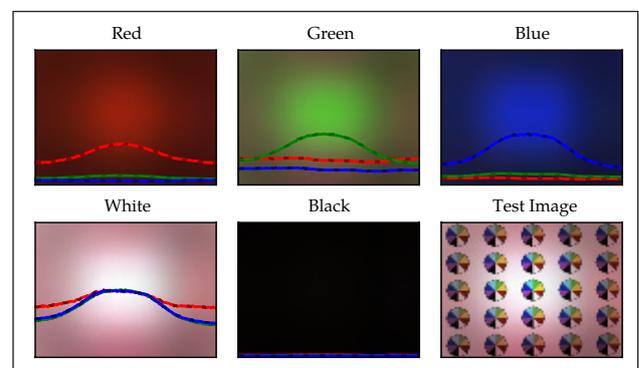
This arrangement allows us to measure images taken under uniform, collimated white illumination, as well as recording individual images under red, green, and blue illumination, and indeed images with no illumination. Python code used to acquire these images, as well as Arduino code used to control the NeoPixel LED unit, is included in the accompanying archive to this paper.

### Measurements

Our experiment consisted of acquiring five images to calibrate the sensor, and a further image containing colour wheels at different points in the camera's frame to test our calibration. At the start of the experiment, the sensor was run in full auto mode for a few seconds, to adjust the exposure time, gain, and white balance to produce an image that was not saturated and had neutral colour balance, under white illumination. After this had been achieved, the settings were frozen, the illumination was changed through different values, and images were saved with each of white, red, green, blue, and black illumination. Finally, the test image was acquired and saved, using the same camera settings. An example set of images is shown in **Figure 4**.

### Loading raw images

The Raspberry Pi camera firmware is now capable of returning raw images as well as processed ones, and does so by embedding the raw data into a JPEG<sup>7</sup> image file. It is important to note that the image represented by the JPEG data has generally been through most of the pipeline described earlier, and is thus not the same as the raw image that appears later in the file. We use a modified version of the `picamera.array` submodule to strip the raw data from the end of the JPEG file, and reconstitute it into a `numpy` array [22]. Pixels are assigned to the appropriate colour channel, but no demosaicing or smoothing is performed. Each pixel thus has only one non-zero value



**Figure 4:** A typical set of 5 images, acquired under uniform illuminations of different colours, plus a test image with multiple colours. Note the vignetting, uneven colour response, and low saturation at the edges of the test image. Sections through the centre of each image in Red, Green, and Blue colour channels are overlaid as lines across the middle of each plot.

in its red, green, and blue channels. Binning the image in blocks of  $16 \times 16$  pixels then yields one RGB value for each block.

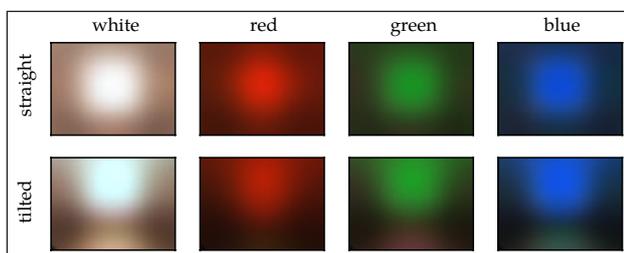
A very important consideration, particularly when the light level is low, is the black level of the image. Multiplying the sensor's output by the inverse of a nominally white image makes the assumption that the sensor's output will be zero in the absence of light. The IMX219 chip includes non-light-sensitive pixels, which allow it to measure and correct for the black level, which varies with temperature. The raw 10-bit image data is adjusted such that pixels read 64 when there is no light incident on the sensor, so when reading the raw data it is important to correct for this.

### Analysis

#### Vignetting

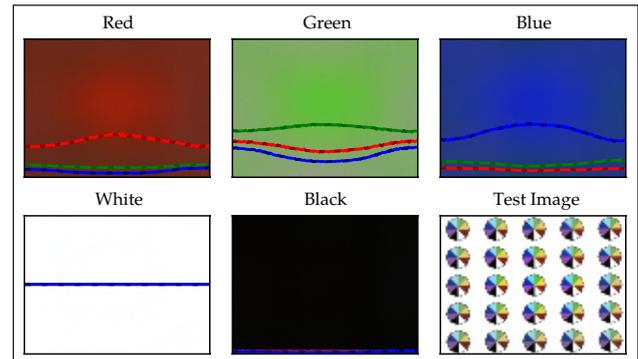
The most prominent effect noticeable in the raw images is vignetting; the image is significantly less bright towards the edges of the sensor. This is not uniform across the colour channels, in part because the lenslets do not focus the light achromatically, so we expect a more diffuse focus (and thus a gentler fall-off with distance from the centre of the sensor) using red light. When using optics that are free from vignetting, such as a lens designed for a much larger sensor, or the calibration jig in **Figure 3(a)**, the Raspberry Pi camera module still exhibits significant vignetting. To substantiate our assertion that this is due to the CRA compensation of the sensor, **Figure 5** shows a set of images with normally incident light, and light incident at an angle. The vignetting pattern shifts noticeably, demonstrating that this is an angle-dependent effect.

A reasonable model of vignetting might be a smooth 2D image, which may be different for the different colour channels, representing the efficiency as a function of position on the sensor. Indeed, this is how the correction is applied in the GPU processing pipeline. Vignetting can be corrected by dividing through by a reference image to compensate for the difference in efficiency between the centre and edges of the sensor. If we use a white image to

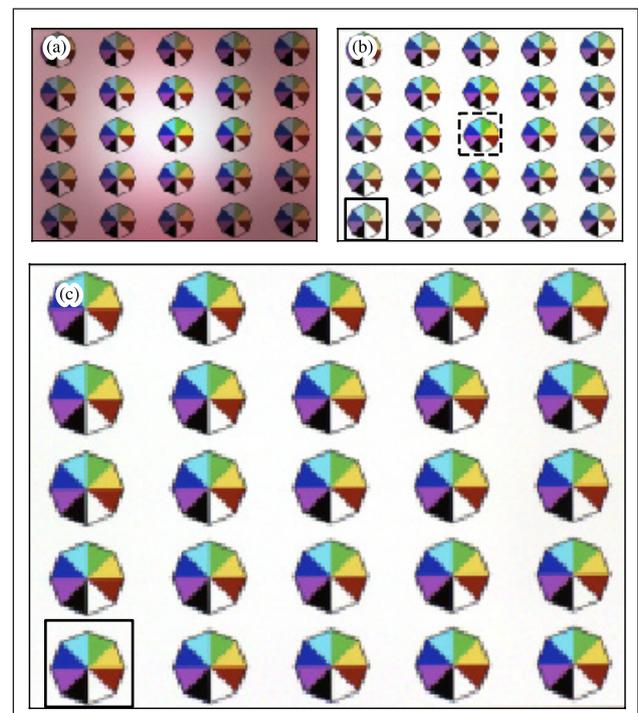


**Figure 5:** Images acquired under different uniform illuminations in the calibration jig shown in Figure 3(a). Light was either normally incident on the sensor (top row) or incident at an angle (bottom row), achieved by rotating the camera module in its cylindrical holder. The vignetting pattern clearly shifts, demonstrating that it is largely due to the angle of incidence. All images are normalised relative to the centre of the normally-incident white image. Note the crosstalk where green illumination is detected by red pixels in the tilted green image.

normalise the sensor, we will correct this fall-off in other images that are similarly illuminated. For many applications, this is sufficient – if the images collected usually have a uniform white background, for example in bright-field microscopy, correcting the vignetting restores the background to make it uniform again. However, it is interesting to note the vignetting-corrected images under red, green, and blue illumination; there is a clear decrease in saturation towards the edges of the image as can be seen in **Figure 6**. Vignetting correction as applied to a test image of colour wheels can be seen in **Figure 7**.



**Figure 6:** The same set of images presented in Figure 4, normalised to the white image. This is equivalent to the built-in lens shading correction, and will make the white image uniform and white by definition. RGB components of a line section through the image centre is overlaid on each plot, as in Figure 4.



**Figure 7:** Images of the colour wheel test target, with (a) only colour-balance correction  $Cb_{qq}$ , (b) vignetting correction  $Cb_{qq} LST_{ijq}$  and (c) colour-unmixing  $U_{ijpq} Cb_{qq} LST_{ijq}$ . The colour wheels extracted in Figure 9 are outlined with black boxes.

### Colour crosstalk

This decrease in saturation indicates that there is crosstalk between the colour channels – this is expected due to the lenslets not being centred on the pixels underneath at the edges of the sensor. However, it is reasonable to suppose that this crosstalk is a linear effect and can thus be modeled and compensated for [15]. While vignetting requires only one scalar parameter per channel, crosstalk requires a matrix capturing the influence of each colour channel on each of the others. This matrix, as with the efficiency parameters for vignetting, will vary as a function of position on the sensor.

We can split the correction of the image into a few different steps, each represented by a different matrix. This is helpful because it means we need not acquire enough information for the entire correction each time we recalibrate. The components we consider are:

#### Lens shading table/white image normalisation

$LST_{ijk}$  has  $w \times h \times 3$  degrees of freedom, i.e. it is an image. In this analysis, we correct for vignetting before correcting for colour crosstalk. Note that the Raspberry Pi's pipeline is usually set up with each channel normalised to have unity gain for the brightest pixel, so that the lens shading table does not affect colour balance. Simply dividing through by a white image applies both the lens shading table and the colour balance.

**Colour balance matrix**  $Cb_{pq}$  is a diagonal  $3 \times 3$  matrix with three degrees of freedom, responsible for equalising the gains of the different components of the image. Normally the gain for the green channel is fixed at unity (in order to avoid changing the overall brightness too much) and thus there are only two meaningful degrees of freedom.

**Colour response matrix**  $Cx_{pq}$  is  $3 \times 3$  i.e. it's a matrix that maps colour to colour, and is not spatially varying. It should depend on the overlap of the spectra of the three illumination colours used and the camera's colour filters. This matrix may also apply colour balance, so if  $Cb_{pq}$  is to be meaningful, we must define this matrix such that it does not change the overall balance of the colours. This gives it six meaningful degrees of freedom.

**Spatially varying colour-unmixing matrix**  $U_{ijpq}$  is a  $w \times h \times 3 \times 3$  array, describing leakage of one channel to another. Depending on how this is calculated, it may include both  $Cb_{pq}$  and  $Cx_{pq}$ , but it is possible to split these out. We define  $U_{ijpq}$  such that it does not affect the colour balance or colour crosstalk at the brightest point of the image (generally the centre).

The normalisation we've done to images in the previous section, dividing through by a white image, combines  $LST$  and  $Cb$ , i.e. if we have a white image  $W_{ijp}$  (where the white level is 255),

$$Cb_{pp}LST_{ijp} = \frac{255}{W_{ijp}}$$

If we start with an unnormalised image  $Image_{ijp}$ , we can colour-correct and unmix by doing:

$$Pure_{ijp} = \sum_{qr} U_{ijpq} Cx_{qr} Cb_{rr} LST_{ijr} Image_{ijr}$$

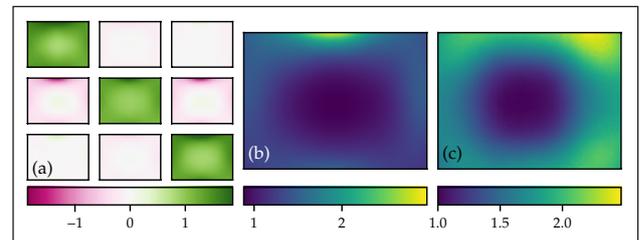
This will unmix the colours such that the images taken under red, green, and blue illumination become pure red, green, or blue. However, if we just want to normalise colour response across the sensor (which has lower noise and more normal-looking colours) we simply remove  $Cx$ :

$$Unmixed_{ijp} = \sum_q U_{ijpq} Cb_{qq} LST_{ijq} Image_{ijq}$$

An example matrix is  $U_{ijpq}$  shown in **Figure 8(a)**.

Both of the spatially varying matrices (i.e. the vignetting correction and the colour-unmixing matrix) should be smooth, as the effects they are correcting for ought to vary smoothly across the sensor. Any high-frequency components in these matrices probably indicate dirt in the optical system, or some other problem. For this reason, and to keep computational cost low, we have reduced the images in size by a factor of 16 by taking the mean of a square bin of pixels.

In order to calculate  $U_{ijpq}$ , we must use the images acquired under red, green, and blue illumination. In order to separate out vignetting ( $Cb_{kk}LST_{ijk}$ ), we first normalise each of the single-colour images by dividing through by a white reference image. If we stack these three images together, the result has four dimensions, with size  $w \times h \times 3 \times 3$ . This matrix describes the response of each pixel to red, green, and blue light, in a  $3 \times 3$  matrix. Inverting this matrix for each pixel obtains  $Cx_{qr}U_{ijpq}$ , i.e. it will convert the RGB values such that the image is uniform and fully-saturated red, green, or blue for each of the illumination values. We use `numpy.linalg.inv` to invert the crosstalk matrix for each pixel in a Python `for` loop; this is tolerably fast for our downsampled images, despite the

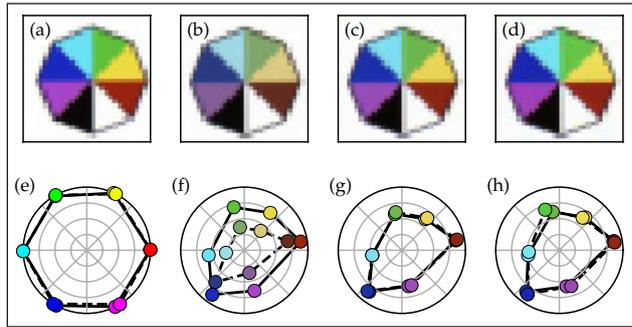


**Figure 8:** (a) A representation of the  $3 \times 3$  spatially-varying matrix that unmixes the colour channels at the edge of the image. Crosstalk is strongest between green and the other two channels due to the spatial arrangement of the Bayer pattern, which means red and blue are never adjacent pixels. (b) The increase in noise in the image, due to the unmixing of the colour channels by the matrix in (a). This is when the colours are normalised to the centre of the image; if the colours are fully unmixed, the oversaturated image has 2.3 times more noise again. (c) The additional noise penalty of normalising the image to correct for vignetting.

computational inefficiency of the interpreted `for` loop. Using this matrix to correct the images produces fully-saturated colours, which may or may not be desirable. We would expect the recorded image to be less than fully saturated, due to overlap between the spectra of the colour channels of the screen and the colour filters in the camera; this is characterised by  $Cx_{pq}$ . Most consumer cameras will enhance saturation to some extent to correct for this and produce attractive images, but in scientific imaging it is more usual to record the image as measured by the camera.

Our approach is to normalise to the centre of the sensor, i.e. we normalise  $U_{ijpq}$  to be the identity in the centre of the image. This is a simple change to make – we simply measure the colour response to the red, green, and blue LEDs, and construct a matrix that converts between the “pure” colours one obtains from the inverted matrix, and the actual colours measured in the centre of the sensor. This  $3 \times 3$  matrix is the inverse of  $Cx$ . This will partially undo the correction we have applied, and results in a less extreme change to the image. Reducing the change results in a predictable decrease in high-frequency noise in the image. This change in noise can be simply quantified; each colour value reported is a linear combination of a number of pixel readings. This can be represented for one pixel as  $c = \sum a_i p_i$  where  $c$  is the output value,  $a_i$  are the coefficients (which ultimately come from the matrices we have defined previously) and  $p_i$  are the pixel values in the raw image. On the assumption that the camera read-out noise is independent for each pixel, the noise in the output value will be proportional to  $\sqrt{\sum a_i^2}$  while the magnitude of the output value will be roughly constant, across a well-corrected image. As the colour unmixing matrix has larger elements towards the edges, this results in the signal-to-noise ratio (SNR) getting worse towards the edges of the image. A plot of the increase in noise due to colour unmixing ( $U_{ijpq}$ ) and vignetting correction ( $LST_{ijp}$ ) is shown in **Figure 8(b, c)**. Note that this does not include the increased noise due to colour balance (adjusting the overall brightness of the three colour channels), and it refers to the matrix that unmixes the colours to achieve uniform response over the camera rather than fully unmixing to pure red, green and blue. The latter would increase the noise by a further factor of 2.3 for the optical system described here.

A comparison of the different corrections is given in **Figure 9**. The matrix correction is able to recover consistent colours out to the edges of the image. While we found that the white reference image is quite specific to each optical system, it was possible to use a colour-unmixing matrix calculated from a set of calibration images acquired in a jig to unmix the colours in an image acquired using a lens imaging a computer screen. This suggests that the correction generated using the calibration jig is applicable to most optical systems where light is normally incident across the sensor. Clearly if the sensor is significantly tilted, or if very different optics are used, it is likely that the correction will be less good. However, the majority of scientific applications that require removing the stock lens do have close to normal incidence across the image, and thus would benefit from this calibration. It would be



**Figure 9:** Images of colour wheels extracted from the centre of a test image **(a)**, and the corner of the same image **(b–d)**. Below are polar plots of the six coloured segments for the central (solid lines) and corner (dashed lines) wheels of the image **(e–h)**. **(b, f)** only have vignetting correction, so the saturation is significantly lower in **(b)**, and the two sets of colours are separated in the polar plot **(f)**. **(c, g)** are unmixed using in-situ calibration, so the colours in **(c)** match those in **(a)**, and the two sets of colours lie on top of each other in **(g)**. **(d, h)** are unmixed using a combination of an in-situ white image and single-colour images from the calibration jig; the colours do not match as closely as in **(c, g)** but are much improved from **(b, f)**. **(e)** is the same as **(g)** but unmixed to fully-saturated colours rather than matching the central response, giving oversaturated images and increased noise. Uncropped images from which **(a–c)** were taken are shown in **Figure 7**.

convenient to have an analytic function that generates a correction for any given lens – however we have not found a function that fits well without significant optimisation. We have detailed a few candidate functions, such as the convolution of a Gaussian and a square (based on the assumption of square pixels, and Gaussian spots having passed through the lenslet array), in an iPython notebook in the associated data archive [8] in “analysis/modelling.ipynb”.

### Correcting images in experiments

The `userland` library on the Raspberry Pi now includes bindings to the necessary MMAL<sup>8</sup> functions to manipulate the “lens shading table” used to correct images for vignetting. Together with the colour balance settings, this allows us to correct for vignetting ( $Cb_{LST_{ijp}}$ ). Unfortunately, the table of gains cannot include  $U_{ijpq}$  as it has only three rather than four dimensions – colour unmixing must be done in post-processing.

The example C code provided in the `userland` library is able to take a white reference image and generate the required lens shading table to correct images. Our modified fork of the `picamera` library [6] is written in Python rather than C for ease of use, and makes it simple to include a number of improvements in the generation of the correction table. This also greatly simplifies the procedure of dynamically adjusting the lens shading table when using the camera from a Python application.

The lens shading table has a resolution 64 times lower than the image sensor, and is interpolated – this not only saves memory, it also enforces a certain level of

smoothness in the correction (the low-resolution grid is interpolated smoothly up to the full resolution of the sensor). We average over patches of  $3 \times 3$  pixels in order to reduce noise in the correction table.

In some applications, it is relatively simple to obtain images that should be very close to uniform. For example, in bright-field microscopy the sample can be removed, to leave what should be a uniform white image. In these situations, an in-situ calibration can be run which may have the welcome side-effect of correcting for non-uniformities elsewhere in the imaging system. A Python routine that performs this calibration is included in the OpenFlexure Microscope software, and is archived along with this manuscript [8].

Often obtaining a white image is rather simpler than obtaining the pure red, green, and blue images required for fully unmixing the colour channels. As we have shown above, it is possible to use images acquired in a calibration jig to correct for the crosstalk between colour channels in an experiment. This combined approach may represent a good compromise between ease of use and calibration accuracy.

#### **Reuse potential and adaptability**

Our extension of the `picamera` library can be used by any hardware project that currently uses Python to acquire images from the Raspberry Pi camera. We have also included the key routines necessary to calculate the lens shading table from a reference white image, allowing custom imaging systems to eliminate the non-uniform shading common to so many Raspberry Pi camera based projects. Finally, the methods presented in this paper, and the accompanying Python code, allow fully colour-corrected images to be produced, if the raw images are saved and post-processed. The same processing methods should work on JPEG images, if suitable modifications are made to the loading and saving routines. However, we would advise caution when working with compressed images, in case the pixel values have been transformed in a nonlinear way – this would invalidate the assumption that a simple matrix inversion can compensate for colour crosstalk.

The methods we have used to correct for vignetting and crosstalk should hold true for any reasonably linear CMOS camera, and there are indeed many consumer-grade sensors currently available with CRA compensation in their lenslet arrays. While the details of the code to acquire and load the raw images will vary from camera to camera, the core algorithm should work in its present form.

#### **Discussion**

The Raspberry Pi Camera Module v2 is a popular, convenient, and high quality image sensor. When used with alternative optics in place of the stock wide-angle lens, it suffers from both vignetting and colour crosstalk at the edges of the image. Both of these effects can be corrected using the method and code presented in the repository accompanying this manuscript [5, 8].

While the correction is clearly similar between different systems that project light onto the sensor at close-to-normal incidence, our experience is that the flat-field correction does still vary from system to system, and so acquiring a

uniform white image is still important. However, the more involved calibration of colour response does seem to translate from one system to another. If it is not possible to build a full calibration jig that allows uniform colours to be imaged, it may be an acceptable substitute to use the calibration images included with our data archive [8] in combination with a white image acquired on that specific optical system.

With the corrections we describe, it should be possible to use the Raspberry Pi Camera Module in a variety of scientific applications. Correction for the vignetting effect can be done in real time using a modified version of the `picamera` library [6]. Alternatively, it is possible to compile the correction into the underlying C libraries so that all applications in the system that use the camera will be corrected, without modifying their code. Correction for colour crosstalk cannot currently be done in real time through the GPU-accelerated processing pipeline, but is easily done after the fact using our associated Python package [5].

#### **Future Work**

Comparing calibrations on a variety of systems and camera modules would be interesting, and we invite the community to try out this method and share their results through the associated repository [5]. This may make it possible to simplify the calibration process in the future, for example by finding a suitable parametric function that generates a reasonable, if not perfect, calibration. It would also be possible to integrate full correction (i.e. saturation loss as well as vignetting) into a real time display of the camera images, perhaps accelerated using a GPU.

#### **Conclusion**

The lenslet array on the Raspberry Pi camera module v2 causes both vignetting and pixel crosstalk when used with optics that have close to normal incidence across the whole sensor. Both of these effects can be calibrated out, resulting in a threefold increase in noise at the edges of the image. Vignetting can be corrected in real time using the camera's image processing pipeline, while colour crosstalk may only be corrected after images have been transferred off the GPU. We hope that the calibration methods and code described in this manuscript will be of use to many other projects seeking to use this extremely convenient low cost, high performance camera sensor.

#### **Build Details**

##### **Availability of materials and methods**

The calibration jig requires a 3D printer, basic tools including hex keys, M2 and M3 screws, and elastic bands. These are all widely available. The electronic parts required are a Raspberry Pi computer, Raspberry Pi camera module v2, Arduino Mega and a NeoPixel LED. The first three are widely used, and available from global suppliers such as RS Components and Farnell. NeoPixels are available from many hobbyist electronics shops, and the generic LED module (WS2812) can be sourced more widely still. The imaging system, used to calibrate the camera in conjunction with a display screen, uses the same camera module together with a 50 mm focal length achromatic lens (ThorLabs AC127-050-A) which can be obtained from ThorLabs internationally.

### Ease of build

The calibration requires four printed components, held together with a small number of screws. Removing the lens from the Raspberry Pi camera is the trickiest part, but it is not a difficult build. We would estimate the calibration jig can be constructed (assuming parts are printed) in 15–30 minutes by someone who is prepared to be reasonably careful, and it does not require much previous experience.

The imaging set-up is an infinity-corrected optics module from the OpenFlexure Microscope, and is very similar to the calibration module in terms of ease of assembly.

### Dependencies, operating software, and peripherals

The hardware components depend on a Raspberry Pi and camera module (v2) as well as an Arduino (Mega, though any model would substitute) and a single NeoPixel LED.

Calibration software is written in Python, and should be compatible with Python 2 or Python 3. It depends on a number of relatively standard libraries (numpy [22], scipy [28], opencv-python [9]) as well as a forked version of the picamera library (<https://github.com/rwb27/picamera>). Analysis software additionally requires matplotlib [14], pillow, and Jupyter notebook [24, 18]. Analysis does not require the picamera library and does not require to be run on a Raspberry Pi.

### Hardware documentation and files location

**Name:** GitLab

**Persistent identifier:** [https://gitlab.com/bath\\_open\\_instrumentation\\_group/picamera\\_cra\\_compensation/](https://gitlab.com/bath_open_instrumentation_group/picamera_cra_compensation/) (live repository) <https://doi.org/10.15125/BATH-00764> (archival version)

**Licence:** CERN Open Hardware License

**Publisher:** Dr Richard Bowman

**Date published:** 19/7/2019

Software is stored together with the hardware files, in the same repository.

**Licence:** GNU GPL v3

**Date published:** 19/7/2019

### Data Accessibility Statement

Data, code, and hardware designs referenced in this manuscript are available either from our live repository [5] or from the University of Bath Research Data Archive [8].

### Notes

<sup>1</sup> Graphics Processing Unit.

<sup>2</sup> Complementary Metal–Oxide–Semiconductor.

<sup>3</sup> Central Processing Unit.

<sup>4</sup> Single Lens Reflex.

<sup>5</sup> Light Emitting Diode.

<sup>6</sup> Pulse-Width Modulation, a method of dimming by rapidly switching the LED on and off.

<sup>7</sup> Joint Photographic Experts Group.

<sup>8</sup> Multi-Media Abstraction Layer.

### Acknowledgements

This work would not have been possible without the underlying picamera [17] and userland [25] libraries, which are developed by a dedicated community of volunteers. We would also like all of the forum contributors [7]

who have tested and commented on both the calibration code and the forked picamera library [6]. Particular thanks is due to Dave Jones ([waveform80](https://github.com/waveform80)) and Dave Stevenson ([6by9](https://github.com/6by9)), both of whom have worked with us directly as well as developing the libraries we rely on.

### Funding Information

We would like to acknowledge financial support from EPSRC (EP/P029426/1, EP/R013969/1, EP/R011443/1), the Royal Commission for the Exhibition of 1851 (Research Fellowship for RWB), and the Royal Society (URF\R1\180153, RGF\EA\181034).

### Competing Interests

RWB and JS have shares in OpenFlexure Industries, which may in the future sell products based around the Raspberry Pi camera module.

### Author Contributions

RWB performed experiments, analysed the data, and wrote the manuscript. BIV acquired the initial data and highlighted the issue in the OpenFlexure Microscope project. JC maintains the software and performed experiments. JS assisted with data analysis. All authors contributed to discussions and writing of the manuscript.

### References

1. Custom lens shading discussion. *Raspberry Pi forum*.
2. Raspberry pi foundation. <https://perma.cc/ZYH4-2G5W>.
3. Raspberry pi camera module documentation. <https://perma.cc/TEE3-GWC9>, 2016.
4. Community microscope. <https://perma.cc/39EB-DMKS>, 2019.
5. **Bowman, R.** CRA compensation code and hardware. [https://gitlab.com/bath\\_open\\_instrumentation\\_group/picamera\\_cra\\_compensation](https://gitlab.com/bath_open_instrumentation_group/picamera_cra_compensation), snapshot at <https://doi.org/10.5281/zenodo.3699731>. This is a live repository, for an archived version that corresponds to the work done in this paper, see [8].
6. **Bowman, R.** Forked picamera library. <https://github.com/rwb27/picamera/tree/lens-shading>, with a static archive at <https://doi.org/10.5281/zenodo.3699687>.
7. **Bowman, R.** Pull request thread on picamera. <https://perma.cc/246YEKJN>, Feb. 2018.
8. **Bowman, R., Vodenicharski, B., Collins, J., and Stirling, J.** Dataset for “Flat-field and colour correction for the Raspberry Pi camera module”. *University of Bath Research Data Archive*, <https://doi.org/10.15125/BATH-00734>. This is an archived version of [5].
9. **Bradski, G.** The OpenCV Library. *Dr. Dobb's Journal of Software Tools* (2000).
10. **Catrysse, P. B., Liu, X., and El Gamal, A.** Qe reduction due to pixel vignetting in cmos image sensors. In *Sensors and Camera Systems for Scientific, Industrial, and Digital Photography Applications* (2000), vol. 3965, International Society for Optics and Photonics, pp. 420–430. DOI: <https://doi.org/10.1117/12.385460>

11. **Catrysse, P. B., and Wandell, B. A.** Roadmap for cmos image sensors: Moore meets planck and sommerfeld. In *Proc. of SPIE* (2005), vol. 5678. DOI: <https://doi.org/10.1117/12.592483>
12. **Coward, C.** Picoscope: A digital raspberry-pi based microscope that you can build for less than 60 dollars. <https://perma.cc/V2K2-2RVM>, 2017.
13. **Grant, S. D., Richford, K., Burdett, H., McKee, D., and Patton, B.** Low-cost, open-access refractive index mapping of algal cells using the transport of intensity equation. *bioRxiv* (2019), 640755. DOI: <https://doi.org/10.1101/640755>
14. **Hunter, J. D.** Matplotlib: A 2d graphics environment. *Computing in Science Engineering* 9, 3 (May 2007), 90–95. DOI: <https://doi.org/10.1109/MCSE.2007.55>
15. **Jian, Z., Suying, Y., and Jiangtao, X.** Pixel cross-talk compensation for cmos image sensors. In *2009 Third International Symposium on Intelligent Information Technology Application* (2009), vol. 2, pp. 165–168. DOI: <https://doi.org/10.1109/IITA.2009.161>
16. **Jones, D.** Picamera documentation: Background processing. <https://perma.cc/8LC6-GSHP>.
17. **Jones, D.** Picamera library. <https://github.com/waveform80/picamera>, library is included in the static archive at <https://doi.org/10.5281/zenodo.3699687>.
18. **Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J., Grout, J., Corlay, S., Ivanov, P., Avila, D., Abdalla, S., and Willing, C.** Jupyter notebooks – a publishing format for reproducible computational workflows. In *Positioning and Power in Academic Publishing: Players, Agents and Agendas* (2016), F. Loizides and B. Schmidt, Eds., IOS Press, pp. 87–90.
19. **Li, H., Soto-Montoya, H., Voisin, M., Valenzuela, L. F., and Prakash, M.** Octopi: Open configurable high-throughput imaging platform for infectious disease diagnosis in the field. *bioRxiv* (2019). DOI: <https://doi.org/10.1101/684423>
20. **Maia Chagas, A., Prieto-Godino, L. L., Arrenberg, A. B., and Baden, T.** The €100 lab: A 3d-printable open-source platform for fluorescence microscopy, optogenetics, and accurate temperature control during behaviour of zebrafish, drosophila, and caenorhabditis elegans. *PLoS biology* 15 (July 2017), e2002702. DOI: <https://doi.org/10.1371/journal.pbio.2002702>
21. **Nuñez, I., Matute, T., Herrera, R., Keymer, J., Marzullo, T., Rudge, T., and Federici, F.** Low cost and open source multifuorescence imaging system for teaching and research in biology and bioengineering. *PLoS one* 12 (2017), e0187163. DOI: <https://doi.org/10.1371/journal.pone.0187163>
22. **Oliphant, T. E.** *A guide to NumPy*, vol. 1. Trelgol Publishing USA, 2006.
23. **Pagnutti, M. A., Ryan, R. E., V, G. J. C., Gold, M. J., Harlan, R., Leggett, E., and Pagnutti, J. F.** Laying the foundation to use raspberry pi 3 v2 camera module imagery for scientific and engineering purposes. *Journal of Electronic Imaging* 26, 1 (2017), 1–13. DOI: <https://doi.org/10.1117/1.JEI.26.1.013014>
24. **Perez, F., and Granger, B. E.** Ipython: A system for interactive scientific computing. *Computing in Science Engineering* 9, 3 (May 2007), 21–29. DOI: <https://doi.org/10.1109/MCSE.2007.53>
25. **Raspberry Pi, and Broadcom Europe.** Userland library. <https://github.com/raspberrypi/userland>, library is included in the Raspbian operating system.
26. **Sharkey, J. P., Foo, D. C. W., Kabla, A., Baumberg, J. J., and Bowman, R. W.** A one-piece 3d printed flexure translation stage for opensource microscopy. *Review of Scientific Instruments* 87, 2 (2016), 025104. DOI: <https://doi.org/10.1063/1.4941068>
27. **Sony Semiconductor Solutions.** Imx219 data-sheet, 2014.
28. **Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Jarrod Millman, K., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C., Polat, I., Feng, Y., Moore, E. W., van der Plas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., and Contributors, S.** SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* (2020). DOI: <https://doi.org/10.1038/s41592-019-0686-2>

### Peer Review

The Journal of Open Hardware thanks Ted Corcovilos and Tim Rudge for their contribution to the peer review of this work.

**How to cite this article:** Bowman, RW, Vodenicharski, B, Collins, JT and Stirling, J. 2020. Flat-Field and Colour Correction for the Raspberry Pi Camera Module. *Journal of Open Hardware*, 4(1): 1, pp.1–9. DOI: <https://doi.org/10.5334/joh.20>

**Published:** 13 April 2020

**Copyright:** © 2020 The Author(s). This is an open-access article distributed under the terms of the Creative Commons Attribution 4.0 International License (CC-BY 4.0), which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited. See <http://creativecommons.org/licenses/by/4.0/>.

 *Journal of Open Hardware* is a peer-reviewed open access journal published by Ubiquity Press.

**OPEN ACCESS** 