

# Bayer capture and processing with the Raspberry Pi HQ camera in Python



Eric Bezzam · [Follow](#)

5 min read · Oct 20, 2021

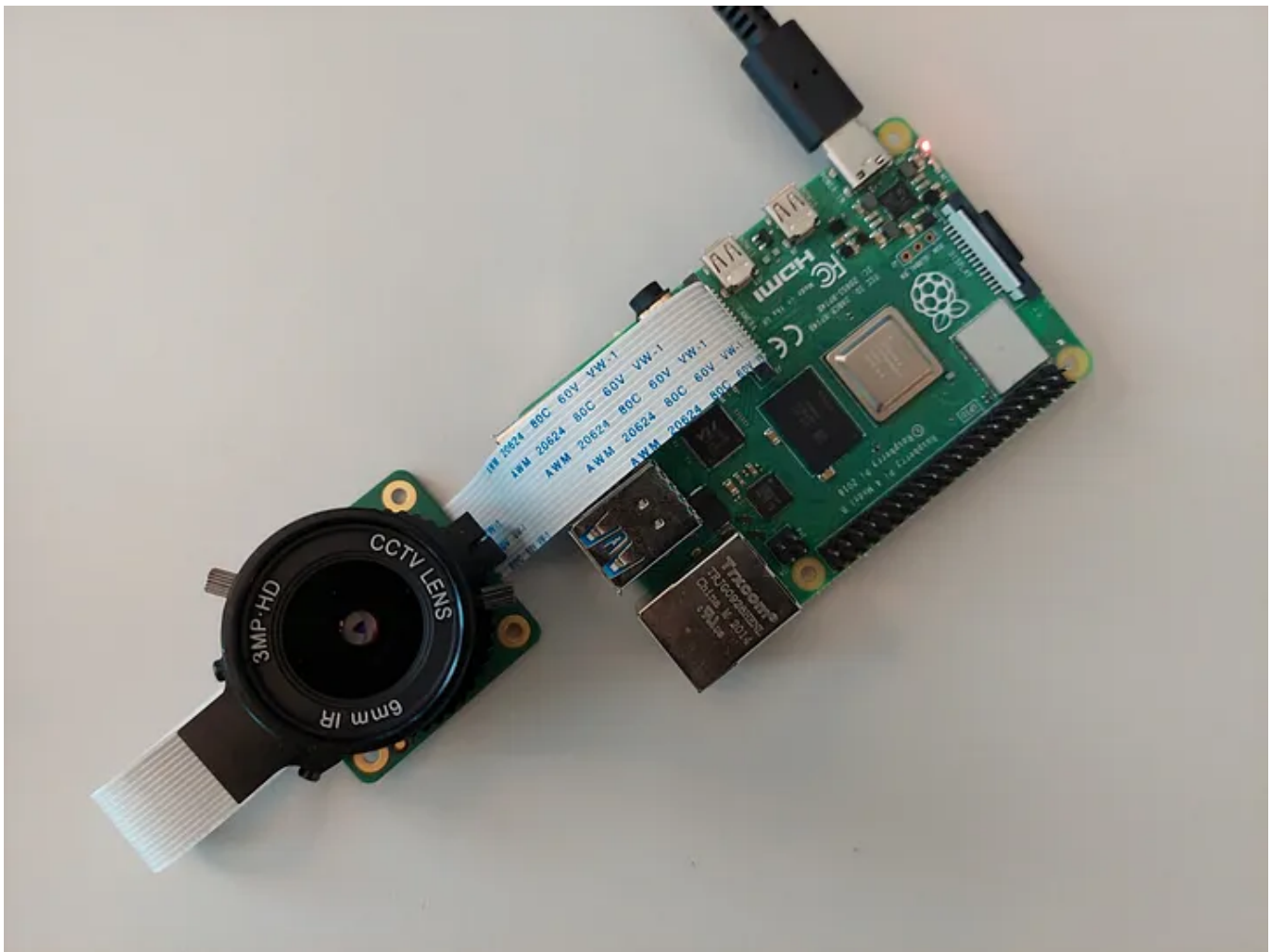


Listen



Share

*We've already seen how to capture images with the Raspberry Pi HQ camera via the command line. In this tutorial, we will see how to do it with Python. Moreover, we'll also see how to capture raw sensor (Bayer filter) data, and how to reconstruct an RGB image from it.*



Before you start:

- Your Raspberry Pi should be flashed with **Buster OS** as Bullseye does not support the Python package ( [picameras](#) ) used to interface with the camera!
- A lens should be fitted and focused; we'll use a 6mm CS-mount lens as shown [here](#).

## Image capture in Python

In a [previous tutorial](#), we saw how to capture images with the Raspberry Pi HQ camera via the command line.

```
raspistill -o test.jpg
```

For automation purposes, it may be of interest to trigger acquisitions from a scripting language such as Python. To this end, the library [PiCameraX](#) provides some useful functionality in order to configure camera settings (e.g. resolution, frame rate, exposure, ISO, etc).

PiCameraX can be installed via pip (note that the package requires Python 3.2 or above):

```
pip install picamerax
```

The following script captures an image to a file.

```
1  from time import sleep
2  from picamerax import PiCamera
3
4  camera = PiCamera()
5  camera.resolution = (1024, 768)
6  # Camera warm-up time
7  sleep(2)
8  camera.capture('test.jpg')
```

simple\_capture.py hosted with ❤ by GitHub

[view raw](#)

If you're working remotely, you can copy over the file as such:

```
scp pi@raspberrypi.local:/home/pi/test.jpg .
```

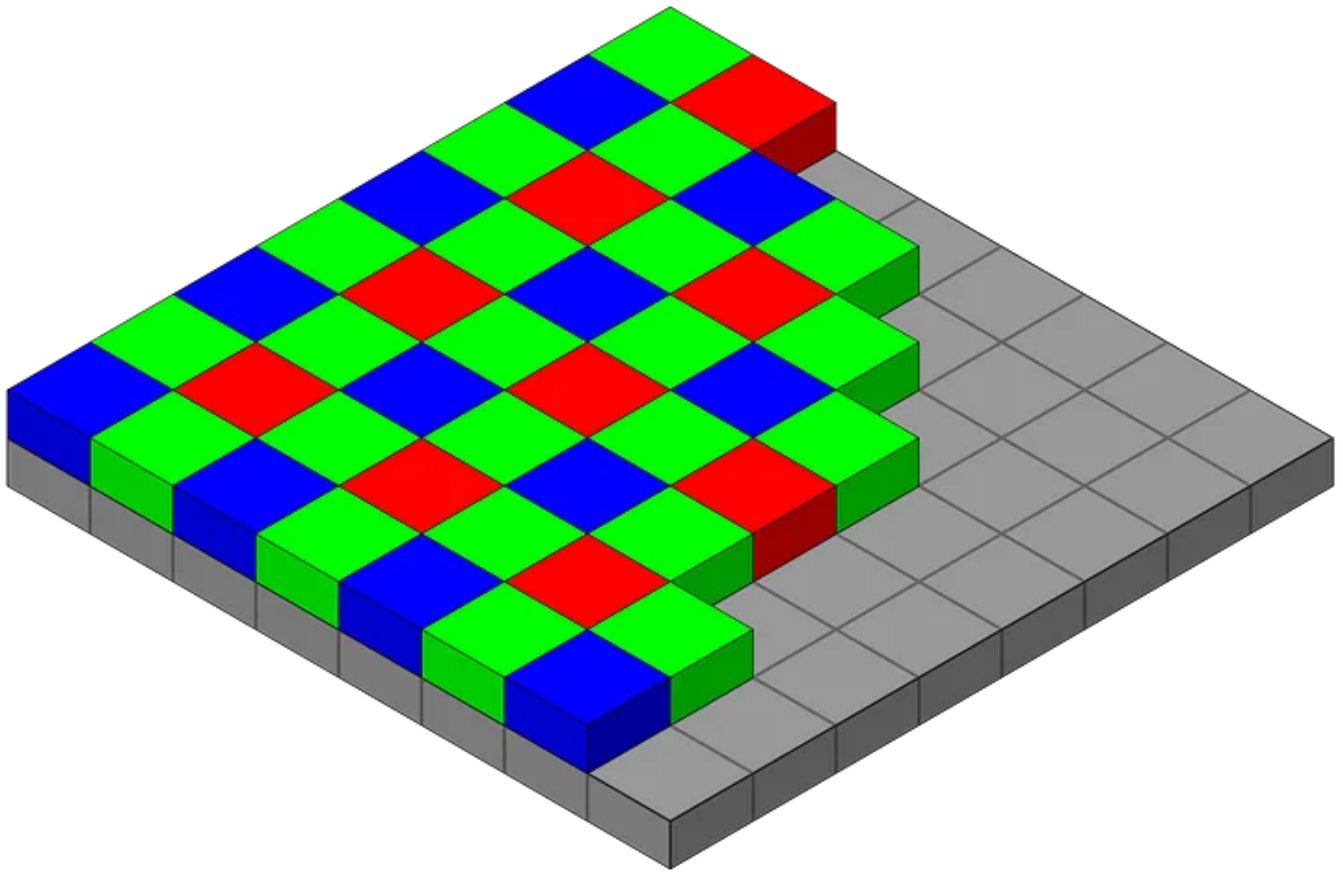


And that's it! Different recipes can be found on the PiCameraX documentation page ([basic](#) and [advanced](#)).

## **Capturing Bayer data via Python**

For the rest of this tutorial, we will focus on Bayer data capture, for which PiCameraX offers some [functionality](#). Bayer data is the raw data measured by the camera sensor before any GPU processing: demosaicing, color balancing, vignette compensation, smoothing, down-scaling, etc. [Wikipedia](#) provides a good overview about Bayer data / filters. *For certain applications, we may require the raw Bayer data, as it has minimal non-linear processing and we can have full control over any subsequent processing.*

A Bayer filter consists of a grid of red, green, and blue pixels with twice as many green pixels. On the HQ camera, the pixels are arranged in a BGGR pattern, as shown below.



Source: [https://en.wikipedia.org/wiki/Bayer\\_filter#/media/File:Bayer\\_pattern\\_on\\_sensor.svg](https://en.wikipedia.org/wiki/Bayer_filter#/media/File:Bayer_pattern_on_sensor.svg)

That means if you were to simply extract the red, green, and blue channels from the Bayer data in order to form an RGB image, the resulting image would be too green. Interpolation methods can be used to combine the different pixel values in order to create a more natural-looking image. This process of building a full color image from a color filter array (CFA) such as Bayer is known as demosaicing.

PiCameraX has a recipe for capturing the Bayer data and demosaicing, which we've modified below to have the image as a PNG file.

```
1  import picamerax
2  import picamerax.array
3  import numpy as np
4  from PIL import Image
5
6  with picamerax.PiCamera() as camera:
7      with picamerax.array.PiBayerArray(camera) as stream:
8          camera.capture(stream, 'jpeg', bayer=True)
9          output = (stream.demosaic() >> 2).astype(np.uint8)
10         img = Image.fromarray(output)
11         img.save("test_bayer.png")
```



Not so great 😞

The purple traces seem to be indicative of clipping, and we still need color balancing in order to adjust red, green, and blue levels appropriately. PiCameraX doesn't offer such functionality so we will show how to do this in order to get a natural-looking image, as well as deal with clipping.

## From Bayer to RGB

We'll have quite a few differences and additions compared to the PiCameraX recipe. Specifically we will:

- Collect `uint16` data as the HQ camera supports 12 bit raw data. Moreover, we need to use OpenCV in order to save 16 bit RGB data. Note that OpenCV expects channels in the BGR format.
- Use OpenCV's demosaicing algorithm, which uses bilinear interpolation (bottom of these docs). Note that OpenCV also has a particular way to define Bayer filter patterns, using the second row, second and third column sub-pixels colors to define the layout. So the conversion for our case is COLOR BayerRG2RGB.

- Apply color balancing using red and blue gains estimated by the Raspberry Pi's AWB (automatic white balance) algorithm. More info on this can be found in Section 5.7 of [their documentation](#).
- Apply a color correction matrix (CCM), Section 5.11 of the [Raspberry Pi camera documentation](#).

This recipe, largely inspired by [this blog](#) (which also provides the black level to subtract and the CCM values), can be found below.

```

1  from time import sleep
2  import picamera
3  import picamera.array
4  import numpy as np
5  import cv2
6
7
8  camera = picamera.PiCamera()
9
10 # get gains for white balancing
11 sleep(2)
12 awb_gains = camera.awb_gains
13 camera.awb_mode = "off"
14 camera.awb_gains = awb_gains
15
16 # perform capture
17 stream = picamera.array.PiBayerArray(camera)
18 camera.capture(stream, "jpeg", bayer=True)
19
20 # get raw Bayer data
21 n_bits = 12
22 output = np.sum(stream.array, axis=2).astype(np.uint16)
23
24 # demosaicing
25 rgb = cv2.cvtColor(output, cv2.COLOR_BayerRG2RGB)
26
27 # subtract black level
28 black_level = 256.3
29 rgb = rgb - black_level
30
31 # white balance
32 rgb[:, :, 0] *= float(awb_gains[0])
33 rgb[:, :, 2] *= float(awb_gains[1])
34
35 # color correction
36 M = np.array(
37     [
38         [2.0659, -0.93119, -0.13421],
39         [-0.11615, 1.5593, -0.44314],
40         [0.073694, -0.4368, 1.3636],
41     ]
42 )
43 rgb_flat = rgb.reshape(-1, 3, order="F")
44 rgb = (rgb_flat @ M.T).reshape(rgb.shape, order="F")
45
46 # clip
47 rgb = rgb / (2 ** n_bits - 1 - black_level)
48 rgb[rgb < 0] = 0

```



```
48  rgb[rgb > 1] = 1
49  rgb[rgb < 0] = 0
50  output = (rgb * (2 ** n_bits - 1)).astype(np.uint16)
51
52  # save
53  cv2.imwrite("test_bayer_corrected.png", cv2.cvtColor(output, cv2.COLOR_RGB2BGR))
```

bayer\_output\_processed\_and\_hosted\_with ❤️ by GitHub

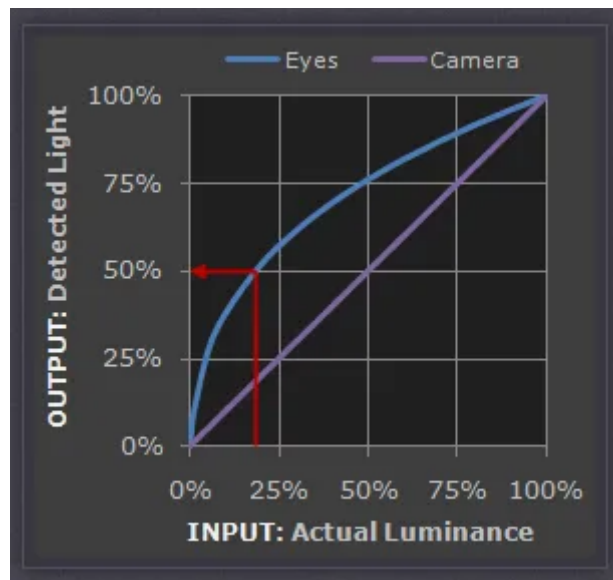
Running the above recipe, we would...still not get an natural-looking image.



## Perceptual adjustments

There are two reasons behind this very dark image:

- Our image data is 12 bit but we are storing it as 16 bits so we are not using the full dynamic range.
- Our eyes do not perceive light the way cameras do, i.e. we perceive light in a non-linear fashion as shown below.



Source: <https://www.cambridgeincolour.com/tutorials/gamma-correction.htm>

In order to account for this perception mismatch, we need to apply gamma correction and normalize before visualizing the image.

Applying solely normalization produces a much more pleasing image.

```
1  import cv2
2  from PIL import Image
3  import numpy as np
4
5  # need OpenCV to open 16 bit RGB
6  fp = "test_bayer_corrected.png"
7  img = cv2.imread(fp, cv2.IMREAD_UNCHANGED)
8  img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
9
10 # normalize and save
11 img_float = img / img.max()
12 data = 255 * img_float
13 img_uint8 = data.astype(np.uint8)
14 Image.fromarray(img_uint8).save("normalized.png")
```

save\_normalized\_image.py hosted with ❤ by GitHub

[view raw](#)




But the image is still rather dark, and the colors don't match our real world perception.

Gamma correction will remedy this discrepancy. To this end, there are some simple approaches, but we will use the Rec. 709 standard:

```
1  import cv2
2  from PIL import Image
3  import numpy as np
4
5
6  def gamma_correction(vals, gamma=2.2):
7      cc = 0.018
8      inv_gam = 1 / gamma
9      clip_val = (1.099 * np.power(cc, inv_gam) - 0.099) / cc
10     return np.where(vals < cc, vals * clip_val, 1.099 * np.power(vals, inv_gam) - 0.099)
11
12
13 # need OpenCV to open 16 bit RGB
14 fp = "test_bayer_corrected.png"
15 img = cv2.imread(fp, cv2.IMREAD_UNCHANGED)
16 img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
17 img = gamma_correction(img, gamma=2.2)
18
19 # normalize and save
20 img_float = img / img.max()
21 data = 255 * img_float
22 img_uint8 = data.astype(np.uint8)
23 Image.fromarray(img_uint8).save("gamma_corrected.png")
```

save\_gamma\_corrected\_image.py hosted with ❤ by GitHub

[view raw](#)

Open in app 

[Sign up](#)

[Sign In](#)





That looks much better 😎

### Extra links

Official post by Raspberry Pi: <https://www.raspberrypi.com/news/processing-raw-image-files-from-a-raspberry-pi-high-quality-camera/>

Stolls with my Dog has also done a very thorough analysis of the HQ camera performance: <https://www.strollswithmydog.com/pi-hq-cam-sensor-performance/>

Getting raw Bayer data without PiCameraX on the Bullseye OS:  
<https://www.raspberrypi.com/documentation/accessories/camera.html#raw-image-capture>

Bayer

Python

Raspberry Pi

Color Correction

Gamma Correction



Follow

## Written by Eric Bezzam

29 Followers

PhD student at EPFL. Previously at Snips/Sonos, DSP Concepts, Fraunhofer IDMT, and Jacobs University. Most of past work in audio and now breaking into optics!

### More from Eric Bezzam



Eric Bezzam

## Headless Raspberry Pi HQ camera setup and focusing

Raspberry Pi officially offers two cameras: the Camera Module v2 (until January 2024) and the HQ camera. On their website, you can find a...

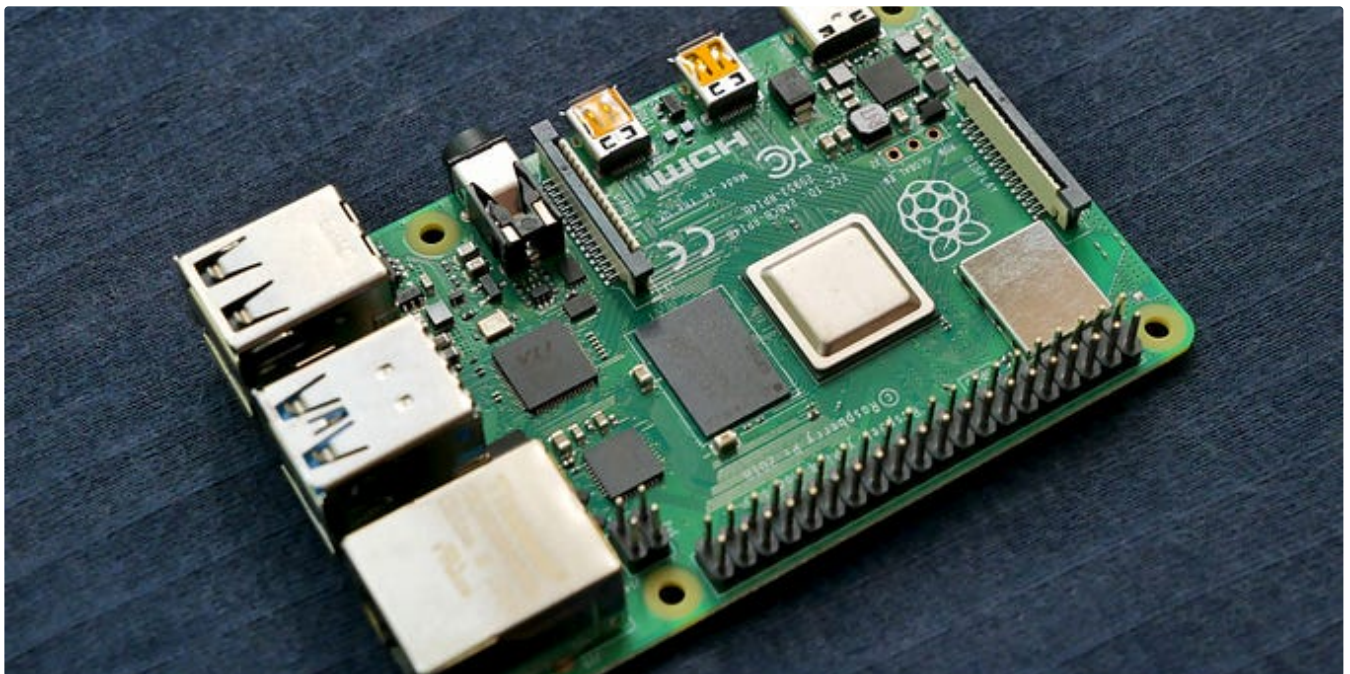
7 min read · Oct 12, 2021




11







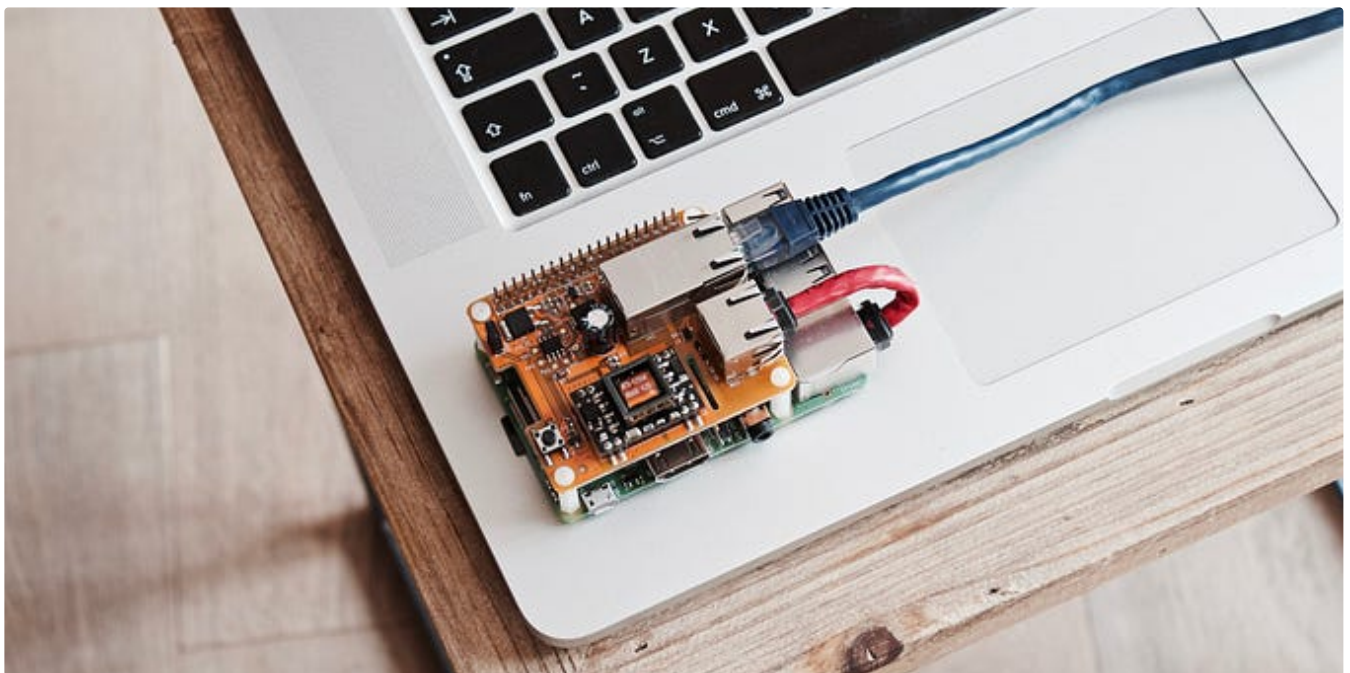
 Eric Bezzam

## Setting up a Raspberry Pi without a monitor (headless)

With the Raspberry Pi 4 opting for micro HDMI ports, not everyone may have an adapter handy. Or you may not have an external screen to hook...

6 min read · Oct 8, 2021

 22 

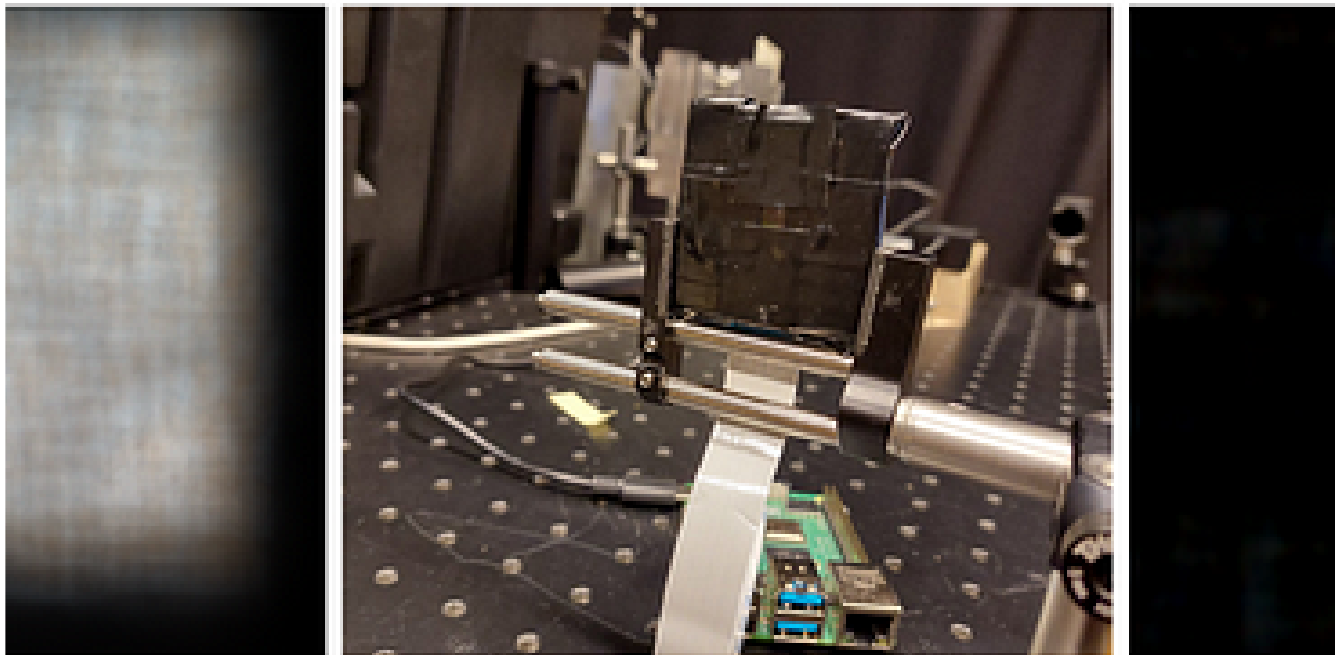


 Eric Bezzam

## Connecting a Raspberry Pi to university / WPA enterprise WiFi

Raspberry Pi (RPi) boards are a great, cost-effective platform to learn about computing and realize an idea into a working prototype. In a...

4 min read · Oct 3, 2022



 Eric Bezzam

## A complete lensless imaging tutorial - hardware, software and algorithms

In this post, we go through all the steps to build and use a lensless camera, specifically a modification of the DiffuserCam [1] proposed...

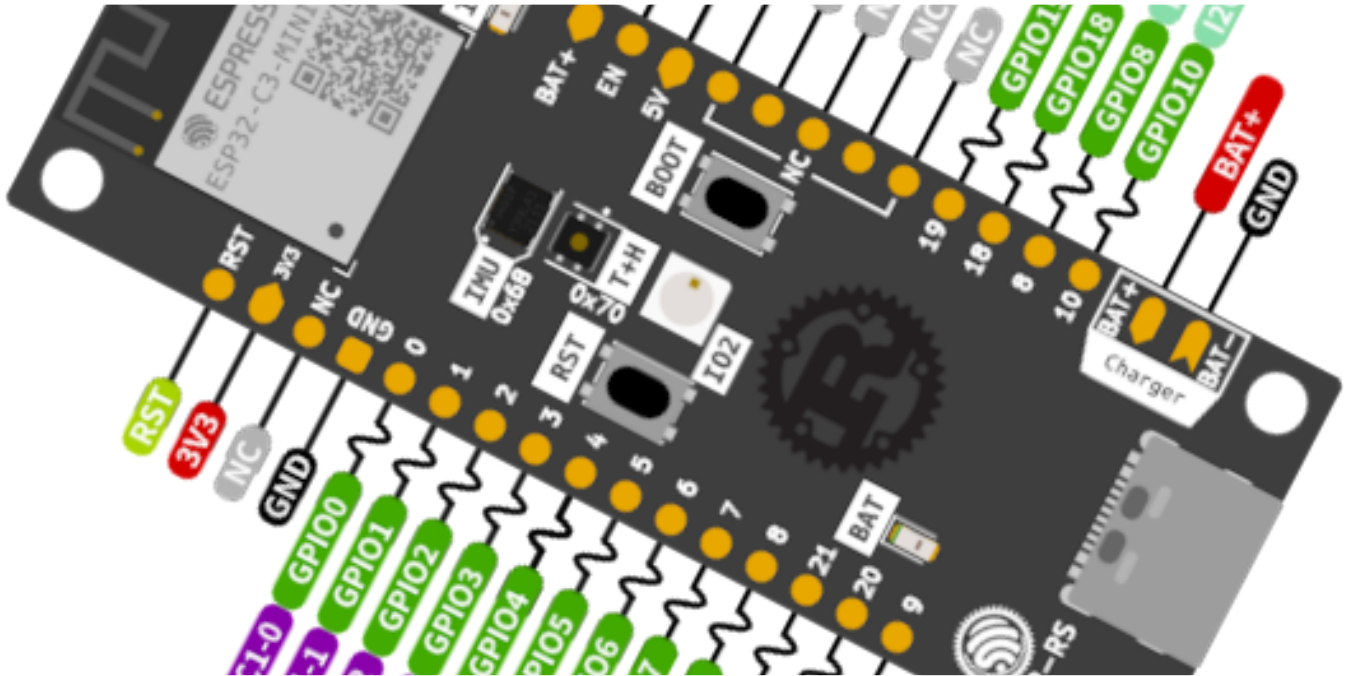
5 min read · Feb 11, 2022



See all from Eric Bezzam



## Recommended from Medium



 Cyril Marpaud

### Embedded Rust on ESP32C3 Board, a Hands-on Quickstart Guide

Today, I'll be showing you how to use the Rust programming language on a Rust ESP board, a recent embedded platform packed with Wi-Fi and...

6 min read · Feb 23

 15 





Kevin Chisholm in Flutter

## What's new in Flutter 3.13

2D scrolling, faster graphics, Material 3 updates and more

12 min read · Aug 17



3.6K



20

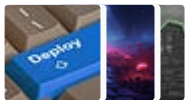


### Lists



#### Coding & Development

11 stories · 118 saves



#### Predictive Modeling w/ Python

20 stories · 296 saves



#### Practical Guides to Machine Learning

10 stories · 312 saves



#### ChatGPT

21 stories · 116 saves

```
id Prompt x + v
MainUser\Documents\Proto Bioengineering\Tutorial-Code\Connect-to-BLE-Device>python3 scanner
AA:B4:6C: None
59:2C:1E: FB4145
07:AD:7D: Xsens DOT
0D:B0:9A: None
F3:0A:F3: None
CB:80:A5: None
46:BD:C4: None
CC:72:84: None
55:C3:80: CS200-A
85:CA:CC: None
C5:01:2F: [TV] Samsung AU8000 50 TV
F5:79:A9: None
F0:07:9B: TV

MainUser\Documents\Proto Bioengineering\Tutorial-Code\Connect-to-BLE-Device>
```

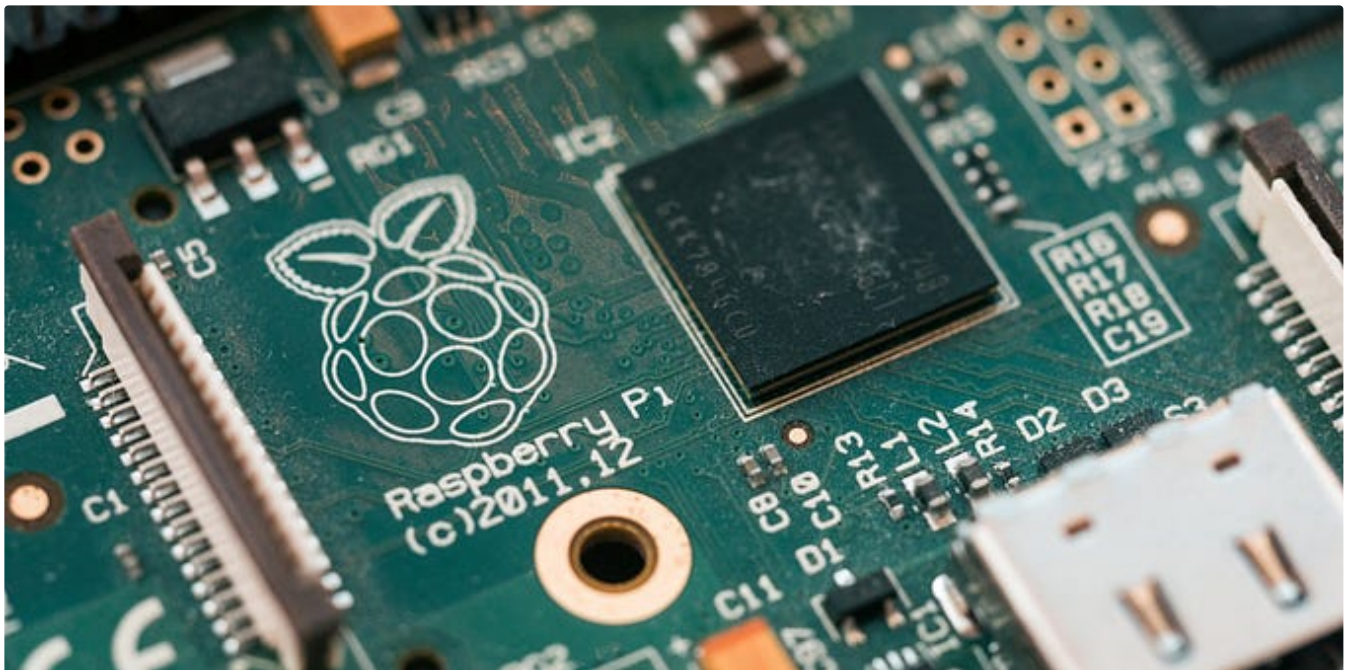
 Proto Bioengineering

## TLDR: How to Make a Simple Bluetooth Scanner with Python

A condensed version of our Bluetooth scanner articles for Mac, Windows, and Linux.

3 min read · Mar 16

 1 




 Chris Grime

## Turn a Raspberry Pi Into a Web Server

A few years ago this site was running off a Raspberry Pi in my room and it worked well! Over time I had to move my site over to a web...

5 min read · Jul 1



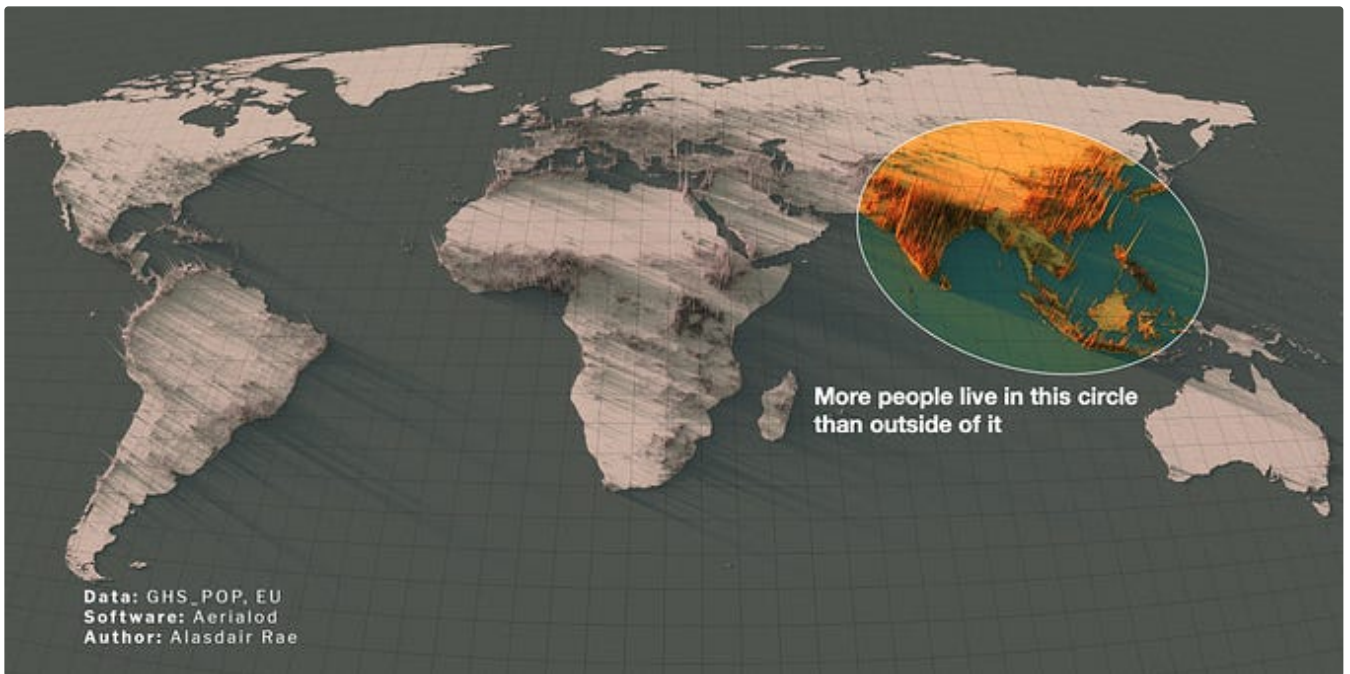
 Hemant Kshirsagar


## Revolutionize Your 3D Computer Vision with Open3D.

An Open-Source Library for High-Performance Processing and Machine Learning.

3 min read · Mar 27





 Tomas Pueyo

## Why Half of Humanity Lives in This Circle

One single accident

★ · 11 min read · Jul 31

 8.5K    78



See more recommendations