# Setup Camera with V4L2, FFmpeg, and PiCamera

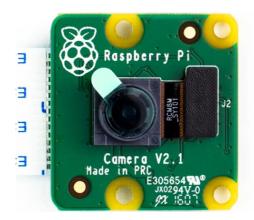
Enable Camera module on Raspberry Pi. Install drivers and applications to capture image or video from the camera. Compare the performance of some encoders used in ffmpeg and python picamera package.

#pi #camera #v4l2 #ffmpeg #python #picamera

Last update: April 23, 2021

# **Table of Content**

- 1. Enable Camera module
- 2. Increase GPU memory
- 3. Test Camera
- 4. Video for Linux 2 V4L2
  - 4.1. List devices
  - 4.2. Driver info
  - 4.3. Supported formats
  - 4.4. Capture JPEG Image
  - 4.5. Record H264 Video
- 5. FFmpeg
  - 5.1. Encoders
  - 5.2. Performance
    - 5.2.1. Raw to MJPEG (.avi)
      - 5.2.2. Raw to H264\_OMX @8Mbps (.mp4)
      - 5.2.3. Raw to H264\_V4L2M2M @8Mbps (.mp4)
      - 5.2.4. V4L2 MJPEG direct copy (.avi)
      - 5.2.5. V4L2 H264 direct copy (.mp4)
- 6. Install picamera
  - 6.1. Picamera H264 (.h264)
  - 6.2. Picamera H264 (.h264) with Text overlay



RaspberryPi Camera Module

This tutorial is for setting up the official Raspberry Pi Camera which is attached with a CSI cable. Other types of USB Camera should work on RPi out-of-the-box.

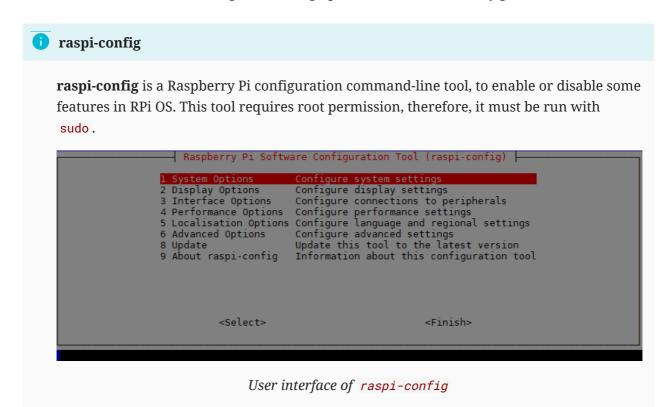
# 1. Enable Camera module

Run raspi-config configuration tool:

```
sudo raspi-config
```

Select Interfacing Options > Camera > Yes.

This method does the same thing with setting up start\_x=1 in /boot/config.txt.



# 2. Increase GPU memory

In the raspi-config configuration tool,

Go to **Performance Options** > **GPU Memory** then fill in **256** and select **OK**.

This method does the same thing with setting up <code>gpu\_mem=256</code> in /boot/config.txt.

### 3. Test Camera

Detect the camera connection by running the checking tool:

```
vcgencmd get_camera
```

which should print out supported=1 detected=1 telling that the camera is supported and connected.



vcgencmd

**vcgencmd** is a command line utility that can get various pieces of information from the VideoCore GPU on the Raspberry Pi. Check more detail in Raspberry Pi/vcgencmd

Raspicam commands has a set of tools to work with the camera module: raspistill, raspivid, and raspiyuv.

• Capture an image:

```
raspistill -o cam.jpg
```

• Record an video:

```
raspivid -o vid.h264
```

# 4. Video for Linux 2 - V4L2

Under Linux, the standard APIs for cameras (including webcams) is V4L (Video for Linux), and a number of applications have been written that support any camera with a V4L driver. An independent developer has now written an user space V4L driver for the Raspberry Pi camera but it is closed sourced, and can be a little slow because it runs as a user program rather than a kernel driver.

Recognizing that a V4L driver is needed, the Raspberry Pi Foundation reported that they were working with Broadcom to develop an official kernel V4L driver. As a kernel driver, it should be faster than the user space driver.

Finally, V4L2 was released under the name bcm2835-v412 which is included in to Raspbian OS by default. Use v412-ct1 utility tool to capture from the camera.

#### 4.1. List devices

```
v412-ctl --list-devices

bcm2835-codec-decode (platform:bcm2835-codec):
    /dev/video10
    /dev/video11
    /dev/video12

bcm2835-isp (platform:bcm2835-isp):
    /dev/video13
    /dev/video14
    /dev/video15
    /dev/video16

mmal service 16.1 (platform:bcm2835-v412):
    /dev/video0
```

#### 4.2. Driver info

```
v4l2-ctl -d /dev/video0 --all
```

# 4.3. Supported formats

```
v4l2-ctl --list-formats
```

```
[12]: 'NV21' (Y/CrCb 4:2:0)
[13]: 'RX24' (32-bit XBGR 8-8-8-8)
```

Please take a note for RGB3, JPEG, H264, and MJPEG, which can be used in OpenCV, or streaming directly.

## 4.4. Capture JPEG Image

```
v4l2-ctl --set-fmt-video=width=2592, height=1944, pixelformat=3
v4l2-ctl --stream-mmap=3 --stream-count=1 --stream-to=somefile.jpg
```

#### 4.5. Record H264 Video

A Note the value height=1088, not 1080.

```
v4l2-ctl --set-fmt-video=width=1920, height=1088, pixelformat=4
v412-ctl --stream-mmap=3 --stream-count=100 --stream-to=somefile.264
```

# 5. FFmpeg

The pre-built ffmpeg package of RPi already enables hardware accelerator support, with OpenMAX IL H.264 video encoder ( h264\_omx ).

```
sudo apt-get install ffmpeg -y
```

# Compile FFmpeg

An FFmpeg version with a specific library can be built by following this topic Compile FFmpeg with Hardware Accelerator.

#### 5.1. Encoders

To see all available encoders:

```
ffmpeg -encoders
```

If interested in h264 and mjpeg, use grep to search for the specific encoders:

```
ffmpeg -hide_banner -encoders | grep -E "h264|mjpeg"
```

# **6** Check encoder options

Before using an encoder, check its options by help command in ffmpeg. For example:

```
ffmpeg -h encoder=h264_omx
Encoder h264_omx [OpenMAX IL H.264 video encoder]:
   General capabilities: delay
   Threading capabilities: none
   Supported pixel formats: yuv420p
h264_omx AVOptions:
   frames if possible (from 0 to 1) (default 1)
   -profile
                 <int>
                           E..V..... Set the encoding profile (from
-99 to 100) (default -99)
      baseline 66
                           E..V.....
                  77
                           E..V....
      main
      high
                  100
                            E..V....
```

#### 5.2. Performance

Next, try to record some short video (60 seconds) with H264 format using different encoders. To measure the performance, use a small tool to check CPU and Memory Usage in monitor - Script to check performance.

#### Video settings

Video side: **1024x768** Framerate: **30** fps

Input Length: 60 seconds

Note that ffmpeg uses v412 driver!

### 5.2.1. Raw to MJPEG (.avi)

```
ffmpeg -y -hide_banner \
   -use_wallclock_as_timestamps 1 \
   -t 60 \
   -i /dev/video0 \
   -c:v mjpeg \
   raw_mjpeg.avi
```

#### • Performance:

• Total time: 63 seconds

• Average CPU: 93 (too high)

• Average MEM: 31

• Input FPS: 4.8 (dropped input)

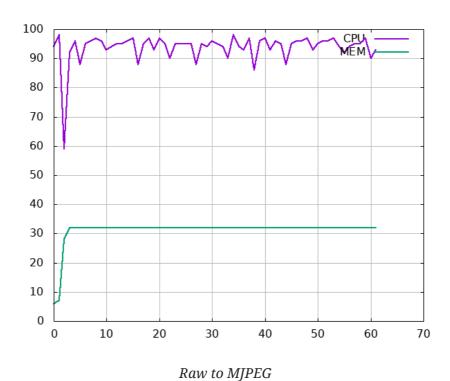
• Output FPS: 30

### • Quality:

• Format: JPEG

• Codec ID: MJPG

• Bit rate: 839 kb/s



### 5.2.2. Raw to H264\_OMX @8Mbps (.mp4)

```
ffmpeg -y -hide_banner \
   -use_wallclock_as_timestamps 1 \
   -t 60 \
   -i /dev/video0 \
   -c:v h264_omx \
   -b:v 8M \
   raw_h264omx.mp4
```

#### • Performance:

Total time: 63 secondsAverage CPU: 16 (OK)

• Average MEM: 27

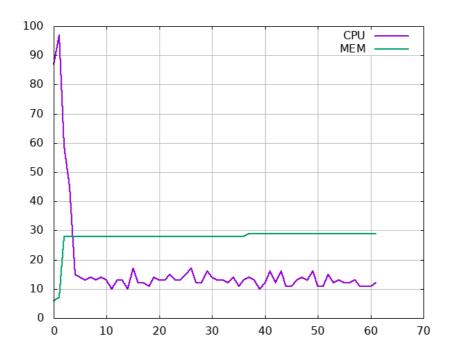
Input FPS: 30Output FPS: 30

### • Quality:

• Format: AVC (GOP: M=1, N=12)

• Codec ID: avc1

• Bit rate: 2 877 kb/s



Raw to H264\_OMX @8Mbps

### 5.2.3. Raw to H264\_V4L2M2M @8Mbps (.mp4)

```
ffmpeg -y -hide_banner \
   -use_wallclock_as_timestamps 1 \
   -t 60 \
   -i /dev/video0 \
   -c:v h264_v412m2m \
   -b:v 8M \
   raw_h264v412m2m.mp4
```

#### • Performance:

• Total time: 62 seconds

• Average CPU: 23

• Average MEM: 27

• Input FPS: 30

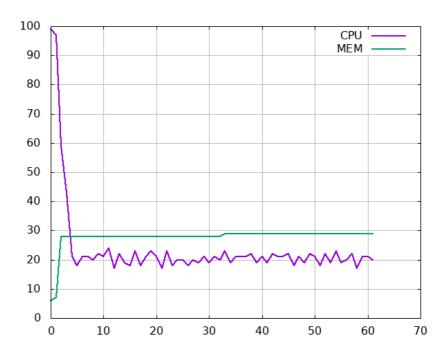
• Output FPS: 30

# • Quality:

• Format: AVC (GPO: M=1, N=60)

• Codec ID: avc1

• Bit rate: 1 783 kb/s



Raw to H264\_V4L2M2M @8Mbps

### 5.2.4. V4L2 MJPEG direct copy (.avi)

```
ffmpeg -y -hide_banner \
   -use_wallclock_as_timestamps 1 \
   -t 60 \
   -input_format mjpeg \
   -i /dev/video0 \
   -c:v copy \
   -t 60 \
   mjpeg_avi.avi
```

#### • Performance:

Total time: 67 secondsAverage CPU: 10 (Good)

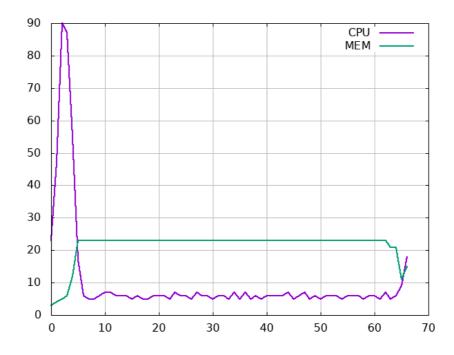
• Average MEM: 21

Input FPS: 30Output FPS: 30

### • Quality:

Format: JPEGCodec ID: MJPG

• Bit rate: 10.2 Mb/s (very high bandwidth)



Save V4L2 MJPEG strem

### 5.2.5. V4L2 H264 direct copy (.mp4)

```
ffmpeg -y -hide_banner \
   -use_wallclock_as_timestamps 1 \
   -t 60 \
   -input_format h264 \
   -i /dev/video0 \
   -c:v copy \
   -t 60 \
   h264_mp4.mp4
```

#### • Performance:

Total time: 67 secondsAverage CPU: 10 (Good)

• Average MEM: 24

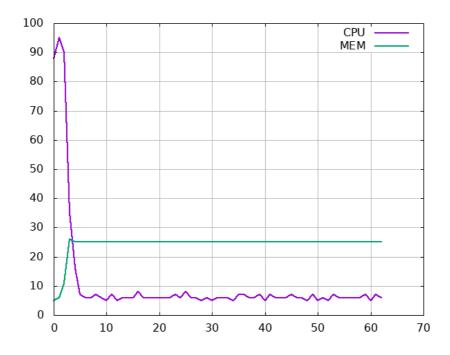
Input FPS: 30Output FPS: 30

• Quality:

• Format: AVC (GPO: M=1, N=60)

• Codec ID: avc1

• Bit rate: 5 506 kb/s (OK)



Save V4L2 H264 stream

After above tests, it says that using compressed input format from ∨412 is much more effective than compressing by an software encoder.

Let's add some timestamp to video by using drawtext filter with built-in expandable localtime variable in Text-expansion option.

```
ffmpeg -y -hide_banner \
    -use_wallclock_as_timestamps 1 \
    -t 10 \
    -i /dev/video0 \
    -vf "drawtext=text='%{localtime}':fontcolor=white:x=100:y=100" \
    -c:v h264_omx \
    -b:v 8M \
    raw_h264omx_text.mp4
```

# Filter and stream-copy cannot be used together

Text needs inserted and each frame needs re-encoded, therefore, stream-copy is unavailable.

```
# this will not work

ffmpeg -y -hide_banner \
    -use_wallclock_as_timestamps 1 \
    -t 10 \
    -input_format h264 \
    -i /dev/video0 \
    -vf "drawtext=text='%{localtime}':fontcolor=white:x=100:y=100" \
    -c:v copy \
    -t 10 \
    h264_mp4_text.mp4
```

# 6. Install picamera

The picamera package is a pure Python interface to the Raspberry Pi camera module for Python. Homepage is at https://picamera.readthedocs.io/en/latest/install.html.

If using the Raspbian distro, probably it has picamera installed by default. Run a test:

```
python -c "import picamera"
python3 -c "import picamera"
```

If no module found, install picamera from the system's package manager:

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

There are a lot of example in the official guide of Picamera at https://picamera.readthedocs.io/en/latest/recipes1.html.

Get maximum resolution of the camera:

```
import picamera
with picamera.PiCamera() as cam:
    print(cam.MAX_RESOLUTION)
```

Take a snapshot:

```
from time import sleep
from picamera import PiCamera

# setup a camera
camera = PiCamera()
camera.resolution = (1024, 768)

# camera warm-up time
sleep(2)

# capture an image
camera.capture('snapshot.jpg')
```

Now, for testing. record a 60-second video from camera and measure resource usage with monitor, then use ffmpeg to convert raw h264 to mp4:

#### 6.1. Picamera H264 (.h264)

```
from picamera import PiCamera

# setup a camera
camera = PiCamera()
camera.resolution = (1024, 768)
camera.framerate = 30

# record a video
camera.start_recording('picamera.h264')
camera.wait_recording(60)
camera.stop_recording()
```

```
ffmpeg -i picamera.h264 \
-c:v copy picamera.mp4
```

#### • Performance:

• Total time: 61 seconds

• Average CPU: 11 (Good)

• Average MEM: 5 (Good)

• Input FPS: 30

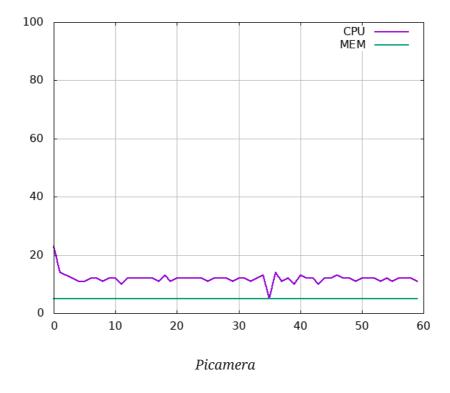
• Output FPS: 25

#### • Quality:

• Format: AVC (GPO: M=1, N=60)

• Codec ID: avc1

• Bit rate: 3 302 kb/s (Good)



# 6.2. Picamera H264 (.h264) with Text overlay

Now, try to detect how picamera can draw text on output video. Here is the test code:

```
from picamera import PiCamera
import datetime

# macro
TIMEFMT = '%Y-%m-%d %H:%M:%S.%f'

# setup a camera
camera = PiCamera()
```

```
camera.resolution = (1024, 768)
camera.annotate_text = datetime.datetime.now().strftime(TIMEFMT)

# record a video
camera.start_recording('picamera_text.h264')
start = datetime.datetime.now()

while (datetime.datetime.now() - start).seconds < 60:
    camera.annotate_text = datetime.datetime.now().strftime(TIMEFMT)
    camera.wait_recording(0.04) # 25fps

# stop it
camera.stop_recording()</pre>
```

Using picamera shows an impressive CPU and MEM usage, comparing to using FFmpeg. The result shows that the CPU uses twice as much as it does in non-overlay text, while the MEM keeps the same percentage.