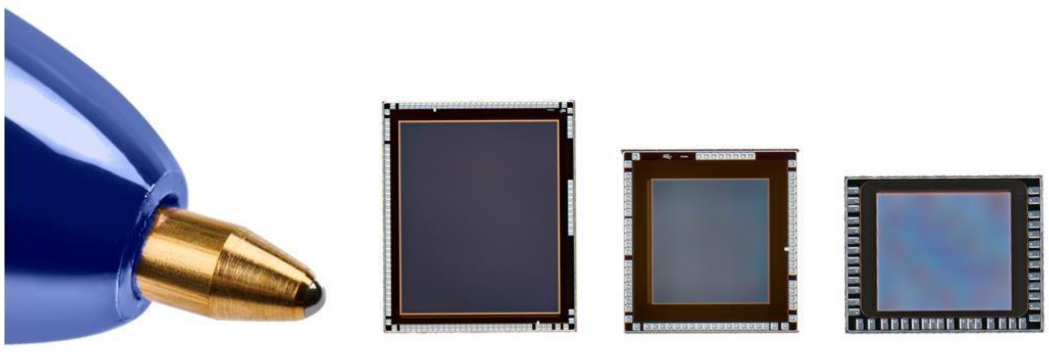
Linux Start Guide for ST BrightSense image sensors

Introduction

The purpose of this Linux Start Guide is to assist Linux users, developers, and enthusiasts in a seamless integration of the ST BrightSense product line from STMicroelectronics.

ST BrightSense is a range of smart high-performance CMOS image sensors developed by STMicroelectronics for professional and consumer vision applications. Relying on proprietary innovations and processes, the ST BrightSense portfolio leverage cutting-edge pixel technologies to provide superior image quality for smart, accurate and reactive vision-based systems. Combining advanced technology nodes and well-thought-out designs, ST BrightSense products highlight tiny form factor and ultra-low power. Their rich toolbox of on-chip features allows faster and lighter processing to support the next generation of smart devices.

Figure 1. **ST BrightSense product portfolio**



All ST BrightSense image sensors are provided with a range of hardware and software tools to enable simple and easy evaluation and development. Linux drivers for ST BrightSense products are available for free download on st.com for integration to Linux-based embedded processing platforms.

This Linux Start Guide aims to support the use of these Linux drivers by providing essential knowledge and practical guidelines for integrating these advanced image sensors into a Linux environment. It includes indications on hardware prerequisites, basic Linux knowledge sharing and comprehensive installation procedure along with practical tips.

Contents

1 Acronyms and abbreviations

Acronym/abbreviation	Definition
V4L2	Video4Linux2: collection of drivers and API for realtime video capture on Linux systems
I2C	inter-integrated circuit (serial bus)
v4l2-ctl	Application to control Video4Linux2 drivers
yavta	“Yet another V4L2 test application”: Application to control Video4Linux2 camera drivers
gstreamer	Multimedia framework

2 Before getting started

This section starts with an overview of the products supported by this Linux Start Guide. It provides also useful indications on where to find further product documentation and an overview of the compatible hardware tools that are needed to connect ST BrightSense image sensors to embedded processing platforms.

2.1 Supported devices

This Linux Start Guide covers all the products of the ST BrightSense portfolio, which includes at the time of writing the following image sensors.

Table 1. List of supported image sensors

Sensor reference	Resolution	Color pattern	Shutter	Output
VD55G0	0.38 MP	Monochrome	Global	MIPI CSI-2
VD55G1	0.56 MP	Monochrome	Global	MIPI CSI-2
VD56G3	1.53 MP	Monochrome	Global	MIPI CSI-2
VD66GY	1.53 MP	RGB	Global	MIPI CSI-2
VD16GZ	1.53 MP	RGB-IR	Global	MIPI CSI-2

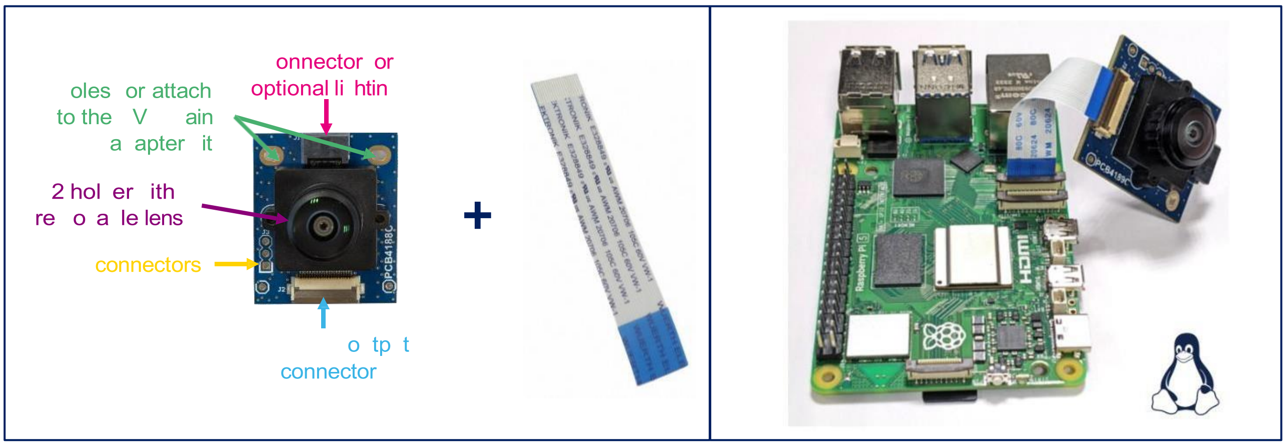
2.2 Compatible ST BrightSense hardware tools

To make integration onto Linux platform immediate, STMicroelectronics provides turnkey hardware tools enabling instant plug-and-play connection to embedded processing platforms. The Linux drivers supported by this Linux Start Guide have been developed to operate specifically on these hardware tools.

Every ST BrightSense image sensor is available with two main hardware kit options, which both contains the necessary board, optics and a 22-pin FFC/FPC cable.

The S-Board is a hardware kit including a sensor board for a given image sensor, that is equipped with M12 lens holder and removable lens, enabling user to change lens at any time.

Figure 2. Illustration of S-Board content and associated setup



The P-Board is a hardware kit including a board with connector to evaluation any camera module provided by STMicroelectronics and its authorized partners. This kit enables to evaluate image sensors built as turnkey camera module to save the effort associated to lens selection and focus.

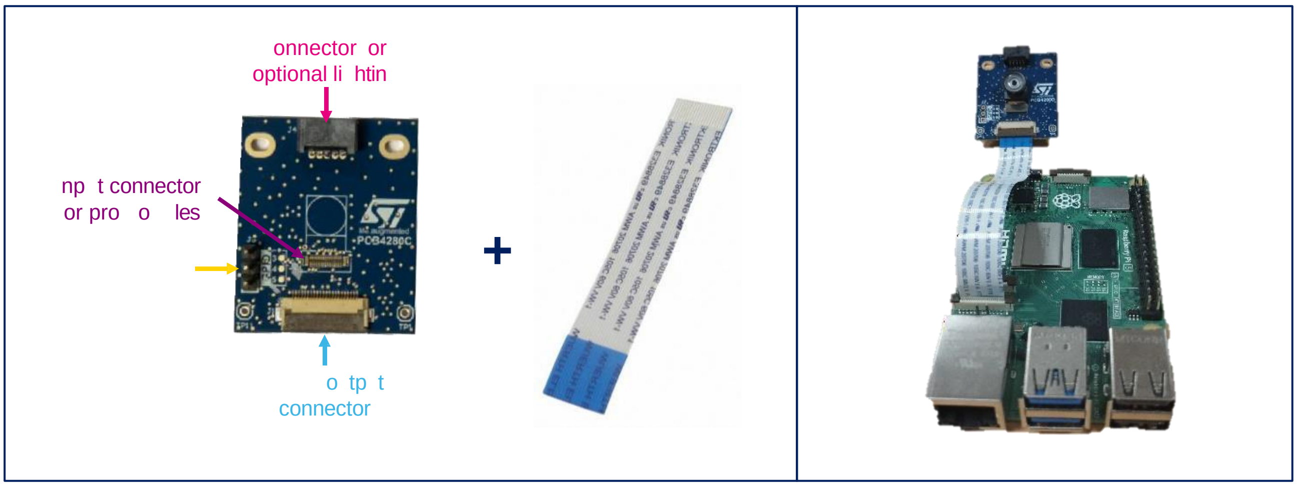


Figure 3. **Illustration of P-Board content and associated setup**

Table 2. **List of compatible ST BrightSense hardware tools**

Kit category	Kit reference	Sensor included?	Lens & holder?	Cable?	Output connector	Output
S-Boards	STEVAL-55G0MBI	VD55G0	Yes	Yes	FFC/FPC	MIPI CSI-2
	STEVAL-55G1MBI	VD55G1	Yes	Yes	FFC/FPC	MIPI CSI-2
	STEVAL-56G3MAI	VD56G3	Yes	Yes	FFC/FPC	MIPI CSI-2
	STEVAL-66GYMAI	VD66GY	Yes	Yes	FFC/FPC	MIPI CSI-2
	STEVAL-16GZMAI	VD16GZ	Yes	Yes	FFC/FPC	MIPI CSI-2
P-Board	STEVAL-CAM-M0I	None (promodule to order separately)	No	Yes	FFC/FPC	MIPI CSI-2

2.3 Finding further product information

To facilitate development and knowledge sharing, ST BrightSense documentation is made publicly available on st.com along with free software tools such as Linux drivers. While this Linux Start Guide focuses on the Linux driver installation and prerequisites, further information can be found regarding product specifications and features in the dedicated product documentation.

Each product webpage includes a “Doc entation” section giving access to all the necessary product documentation such as data brief, datasheet, user manual or application notes. The “Tools & o t are” section list all the compatible hardware and software tools available for this product and can guide you to the corresponding webpage, where further information and download are available for each hardware kit or software tool.

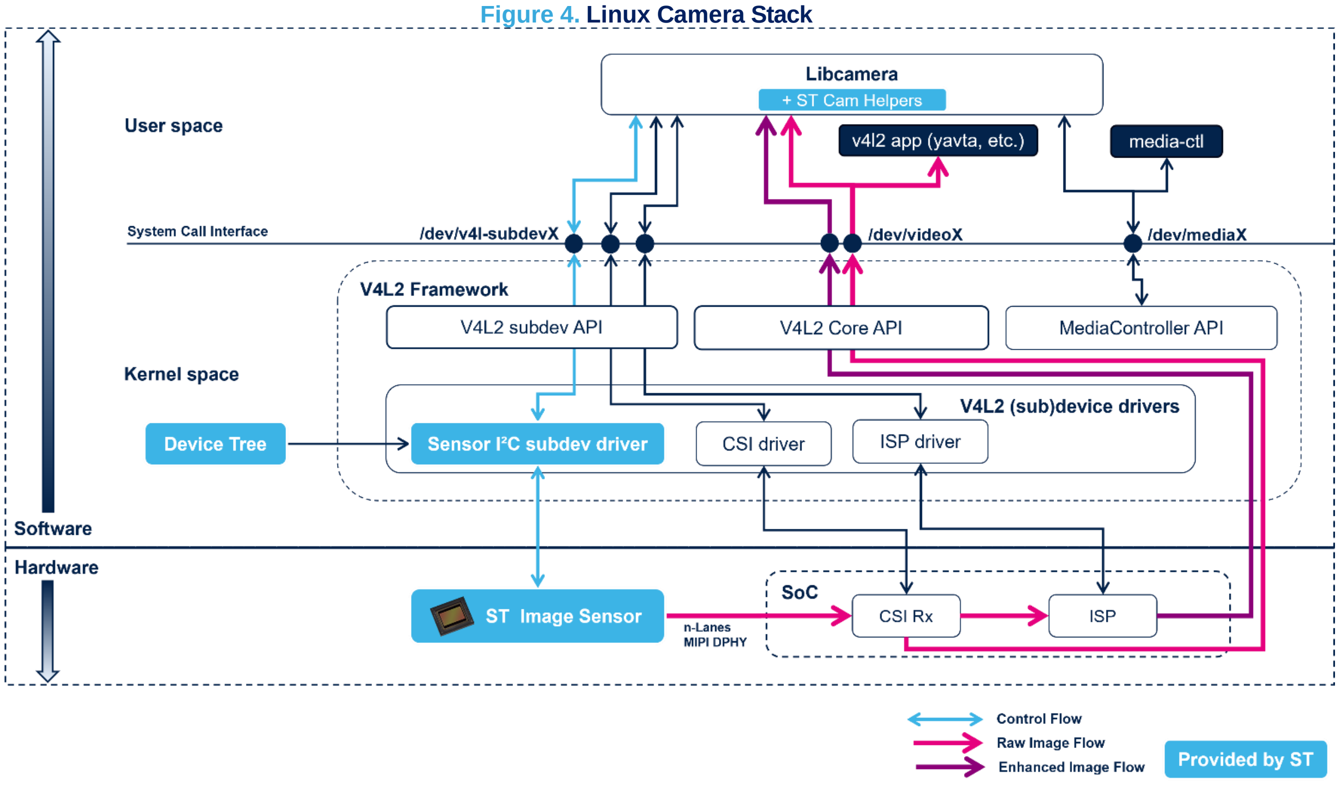
3 Introduction to Linux environment

3.1 About Linux camera stack

The Linux camera stack is a collection of open-source components that enable the interfacing and control of camera devices on Linux-based systems.

It encompasses both kernel-level and user-space components, providing a comprehensive framework for different camera applications (ranging from simple webcam utilities to complex computer vision systems).

Figure 4. Linux Camera Stack



Two key components of the camera stack are the Video for Linux 2 (V4L2) framework and the libcamera library.

3.2 Introduction to V4L2 framework

At the core of the Linux camera stack is Video for Linux 2 (V4L2), which serves as the kernel API for handling video devices.

V4L2 provides a collection of (sub)device drivers and a standardized API for camera drivers, allowing applications to access and control a wide range of video capture devices, including USB cameras, embedded camera modules, as well as the different hardware IPs that are involved in image processing.

V4L2 is responsible for exposing device nodes (/dev/videoX, /dev/v4l-subdevX, etc.) to user space. These nodes allow standard V4L2 applications to control and get frames from image sensors.

Many applications have V4L2 support, allowing for straightforward integration of image sensors (once they are V4L2-ready):

- v4l2-ctl: Swiss army knife for V4L2. Can be used to query or configure a V4L2 device.
- yavta: Yet Another V4L2 Test Application. A command-line tool with a very simple interface to configure and capture frames from a V4L2 device.
- VLC: A versatile media player with V4L2 support that allows capturing and streaming video from V4L2-compatible devices.
- GStreamer: A multimedia framework, providing V4L2 plugins for capturing, processing, and streaming video from V4L2-compatible devices.
- OpenCV: A popular computer vision library that includes V4L2 support for capturing and processing video streams from V4L2-compatible devices. It is widely used for image and video processing, object detection, and machine vision applications.

3.3 Introduction to Libcamera

Complementing V4L2 is libcamera. It operates as a user-space library and provides a unified interface for camera applications.

Libcamera abstracts the complexity of handling the media pipeline; it offers a consistent API for camera control and image processing and enables support of advanced ISP features available in modern SOCs. (add v4l2 constraint by libcamera)

As a newer addition to the Linux camera stack, libcamera is dynamic and evolving, with a focus on modernizing the camera stack and addressing the complexities of modern camera hardware and use cases.

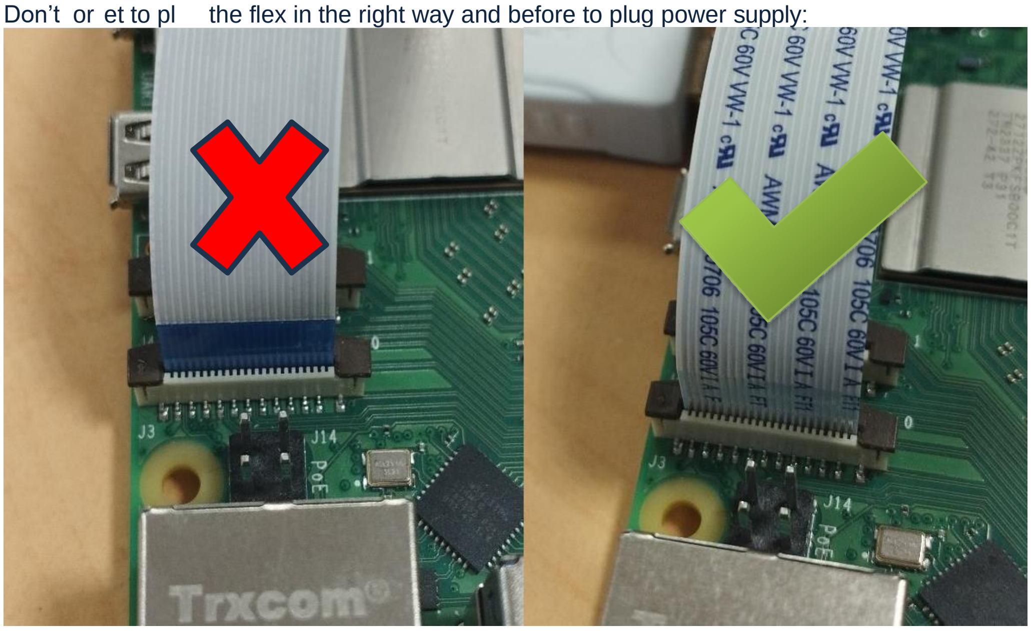
4 Getting started with Raspberry Pi

This section details the installation procedure to get the image sensor streaming onto a Raspberry Pi platform using the Linux driver package available for free download on st.com. This installation procedure can be reused for other embedded platforms relying on Linux environment that uses a V4L2 framework or Libcamera library.

4.1 Hardware setup

Please Follow the instructions from Raspberrypi.com: link

Don’t or et to pl the flex in the right way and before to plug power supply:



4.2 V4L2 driver

This section describes how to build and install the V4L2 driver on a Linux target platform.

For simplicity, the driver is built directly on the target, as it needs to be:

- Built for the corresponding target architecture,
- Built against the current running Linux kernel headers.

Note: Raspberry Pi is used as an example, however the process is similar on other Linux platforms.

1. V4L2 driver content

The V4L2 driver corresponding to the ST BrightSense image sensor must be downloaded from st.com.

Once unzipped on the target Linux platform, the following content should be available (example based on the vd55g1 driver):

```
vd55g1
├── dts                             Main entry point for device tree overlays
│   ├── rpi1_to_4                   Device tree overlays for RPi1 to RPi4 SBCs
│   └── rpi5                        Device tree overlays for RPi5 SBCs
├── Kbuild                          Configuration file for building kernel module
├── Makefile                        Build file containing recipe to compile the module
├── README.md                       Readme file
├── st-vd55g1.c                     Main v4l2 driver source file
└── st-vd55g1_patch.c               Additional source code containing sensor's firmware patch
```

2. Driver installation

The next steps assume that the Raspberry Pi is up and running with an up-to-date Operating System. As of writing, the latest Raspberry Pi OS (based on Kernel 6.6) was released on March 15th, 2024.

1. By default, Linux kernel headers are installed on latest RPi OS releases, if not, run the following commands:

```
$ sudo apt update
# If you are using the 32-bit version of Raspberry Pi OS
$ sudo apt install linux-headers-rpi-{v6,v7,v7l}
# If you are using the 64-bit version of Raspberry Pi OS
$ sudo apt install linux-headers-rpi-v8
```

2. Compile the driver source using the Makefile (this will generate a .ko module file) :

```
$ make
```

3. Install the freshly built module in the kernel modules folder:

```
$ sudo cp *.ko /lib/modules/$(uname -r)
$ sudo depmod -a
```

4. Finally, reboot to apply the modifications:

```
$ sudo reboot
```

3. Device tree configuration

The devicetree is a tree data structure with nodes that describe the hardware components of a system.

When a new hardware is connected to a system, its device tree must be updated accordingly.

1. Ensure the device tree compiler tool is present on your system:

```
$ sudo apt update
$ sudo apt install device-tree-compiler
```

2. From the dts folder, compile the device tree overlay matching your platform and plugin board (for the example ``pcb4189.dts`` is chosen):

```
$ cd dts
$ sudo dtc pcb4189.dts -o /boot/firmware/overlays/pcb4189.dtbo
```

3. Set the device tree overlay for your Raspberry Pi (this step may differ from platform to platform):

```
$ sudo sh -c "echo 'dtoverlay=pcb4189' >> /boot/firmware/config.txt"
```

4. Finally, reboot to apply the modifications:

```
$ sudo reboot
```

4.3 Libcamera

1. RPi OS Libcamera update

By default, Raspberry Pi OS comes with Libcamera library. Using Libcamera allows to benefit from Broadcom HW ISP available on the RPi SoC for image processing (debayering, 3A, etc.).

Each Image Sensor must be described in Libcamera. For that reason, the RPi OS Libcamera package should be upgraded with a new version where all ST BrightSense Image Sensors are described.

Note: The below steps only target Raspberry Pi platform as the libcamera package provided is RPi OS specific.

Once the libcamera package has been downloaded from st.com it can be installed with the below command line:

```
$ sudo dpkg -i libcamera-ipa_x.x.x.deb libcamera-cam_x.x.x.deb
```

2. Libcamera usage

With libcamera upgraded, it’s possi le to se all the rpicam-apps (please refer to RPi-OS documentation to get all details on those libcamera-apps)

For a quickstart, please find below a few one-liners:

```
# List the cameras attached to the board and the sensor modes supported
$ rpicam-hello --list-cameras

# Display camera preview (with default mode)
$ rpicam-hello --timeout 0

# Display camera preview (with chosen 640x480 mode)
$ rpicam-hello --timeout 0 --viewfinder-width 640 --viewfinder-height 480

# After 5 secs preview, capture raw frame (before any ISP processing) as well as jpeg
$ rpicam-still --raw -o frame.jpg

#Change Camera tunning
$rpicam-hello –tuning-file path_of_file.json
```

To continue to access to the Specifique setting (not controlled by libcamera) you can use subdev
For a quickstart, please find below a few one-liners:

```
#Identify your sensor’s V4L2 subdev file. Replace sensor with your sensor name:
ls /sys/bus/i2c/drivers/<sensor>/*/video4linux

#With the previous command result as subdev_file, query the sensor V4L2 controls:
v4l2-ctl -d /dev/<subdev_file> --all

#Set the chosen control to the wanted value:
v4l2-ctl -d /dev/<subdev_file> -c <control>=<value>

#For example, to set the HDR mode on the v4l2-subdev2 sensor’s file (if your sensor
supports HDR):
v4l2-ctl -d /dev/v4l2-subdev2 -c hdr_sensor_mode=0
```

3. Code example using Libcamera

To use the capacity of libcamera you can developpe your own application, here you will find an example to stream with python.

First of all, install the package for python 3 :

```
$ sudo apt install -y python3-pyqt5 python3-opengl
$ sudo apt install -y python3-picamera2
```

Example code to run at 60 fps in 1120x1360 for VD56G3/VD66GY and capture image through button.
To find more control please refer to picamera manual

```
#!/usr/bin/python3

# This example is essentially the same as app_capture.py, however here
# we use the Qt signal/slot mechanism to get a callback (capture_done)
# when the capture, that is running asynchronously, is finished.
from PyQt5 import QtCore
from PyQt5.QtWidgets import (QApplication, QHBoxLayout, QLabel, QPushButton,
                             QVBoxLayout, QWidget)
from picamera2 import Picamera2,Preview
from picamera2.previews.qt import QGlPicamera2
import io
import time
picam2 = Picamera2()

picam2.configure(picam2.create_preview_configuration(main={"size": (1120,
1360)},raw={"format": "R8"}))
picam2.set_controls({"ExposureTime":1000,"AnalogueGain":1.0,"AeEnable":False,"FrameRate
":60})
print(picam2.sensor_format)

app = QApplication([])

def on_button_clicked():
    button.setEnabled(False)
    cfg = picam2.create_still_configuration()
    picam2.switch_mode_and_capture_file(cfg, "test.jpg",
signal_function=qpicamera2.signal_done)


def capture_done(job):
    picam2.wait(job)
    button.setEnabled(True)


qpicamera2 = QGlPicamera2(picam2, width=1120, height=1360, keep_ar=False)
button = QPushButton("Click to capture JPEG")
label = QLabel()
window = QWidget()
qpicamera2.done_signal.connect(capture_done)
button.clicked.connect(on_button_clicked)

label.setFixedWidth(400)
label.setAlignment(QtCore.Qt.AlignTop)
layout_h = QHBoxLayout()
layout_v = QVBoxLayout()
layout_v.addWidget(label)
layout_v.addWidget(button)
layout_h.addWidget(qpicamera2, 80)
layout_h.addLayout(layout_v, 20)
window.setWindowTitle("Qt Picamera2 App")
window.resize(1120, 1360)
window.setLayout(layout_h)

picam2.start()
window.show()
app.exec()
```

1 Getting started with other V4L2 tools

While some embedded platforms such as the Raspberry Pi series rely on Libcamera as a camera software library, other Linux-based plat or s on’t ene it ro it and only support V4L2. In such situation, standard V4L2 tools must be used to communicate with the image sensor.

The following paragraphs present a few command-line tools along with a compilation of useful one-liners.

1.1 v4l2-ctl

Swiss army knife for v4l2: this command-line tool can be used to query, configure, or stream from a v4l2 device.

Available in all distributions, on Debian-like distros it can be installed with the following:

```
$ sudo apt install v4l-utils
```

Useful one-liners:

```
# List formats and controls (with their menus when available)
$ v4l2-ctl --list-formats-ext --list-ctrls --list-ctrls-menu

# Select Format (resolution: 640x480 and pixel format: GRBG Bayer 10Bit)
$ v4l2-ctl --set-fmt-video width=640,height=480,pixelformat=BA10

# Configure V4L2 controls (switch to manual mode and configure exposure (1800 lines))
$ v4l2-ctl --set-ctrl auto_exposure=1
$ v4l2-ctl --set-ctrl exposure=1800

# Configure Vertical Blanking to change framerate (vblank is resolution dependant)
$ v4l2-ctl --set-ctrl vertical_blanking=3056

# Capture 20 frames (concatenated in one 'capture_20f.raw10' file)
$ v4l2-ctl --stream-mmap --stream-count=20 --stream-to=capture_20f.raw10
```

1.2 yavta

Yet Another V4L2 Test Applications. A simple command-line tool supporting many of the latest V4L2 capabilities and used to interact with a V4L2 device.

On Debian-like distributions it can be installed with the following:

```
$ sudo apt install yavta
```

Useful one-liners:

```
# List formats and controls
$ yavta --list-controls --enum-formats /dev/video0

# Select Format (resolution: 640x480 and pixel format: GRBG Bayer 8Bit)
$ yavta --size 640x480 --format SGRBG8 /dev/video0

# Configure V4L2 controls (switch to manual mode and configure exposure (1800 lines))
$ yavta --set-control '0x009a090 1' /dev/video0
$ yavta --set-control '0x00980911 1800' /dev/video0

# Configure Vertical Blanking to change framerate (vblank is resolution dependant)
$ yavta --set-control '0x009e0901 3056' /dev/video0

# Capture (30 frames saved in 30 files)
$ yavta --capture=30 --file=frame-#.raw8 /dev/video0
```

1.3 V4L2 code example

To use the capacity of V4L2 api you can developpe your own application, here you will find an example to stream with python.

First of all, install the package for python 3 :

```
$ pip3 install v4l2-python3
$ pip3 install opencv-python
$ pip3 install python3-nmap
```

```
from v4l2 import *
import cv2
import fcntl
import  mmap
import numpy as np

vd = open('/dev/video0', 'rb+', buffering=0)
print(">> get device capabilities")
cp = v4l2_capability()
fcntl.ioctl(vd, VIDIOC_QUERYCAP, cp)
print(">> device setup")
fmt = v4l2_format()
fmt.type = V4L2_BUF_TYPE_VIDEO_CAPTURE
fmt.fmt.pix.pixelformat=V4L2_PIX_FMT_SGRBG8
fmt.fmt.pix.width=300
fmt.fmt.pix.height=300
fcntl.ioctl(vd, VIDIOC_S_FMT, fmt)  # set whatever default settings we got before
fcntl.ioctl(vd, VIDIOC_G_FMT, fmt)  # get current settings
print("width:", fmt.fmt.pix.width, "height", fmt.fmt.pix.height)

print(">> init mmap capture")
req = v4l2_requestbuffers()
req.type = V4L2_BUF_TYPE_VIDEO_CAPTURE
req.memory = V4L2_MEMORY_MMAP
req.count =2
fcntl.ioctl(vd, VIDIOC_REQBUFS, req)  # tell the driver that we want some buffers
buffers = []
req.count =2

for ind in range (req.count):
    # setup a buffer
    buf = v4l2_buffer()
    buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE
    buf.memory = V4L2_MEMORY_MMAP
    buf.index = ind
    fcntl.ioctl(vd, VIDIOC_QUERYBUF, buf)
    mm = mmap.mmap(vd.fileno(), buf.length, mmap.MAP_SHARED, mmap.PROT_READ |
mmap.PROT_WRITE, offset=buf.m.offset)
    buffers.append(mm)
    # queue the buffer for capture
    fcntl.ioctl(vd, VIDIOC_QBUF, buf)
print(">> Start streaming")
buf_type = v4l2_buf_type(V4L2_BUF_TYPE_VIDEO_CAPTURE)
fcntl.ioctl(vd, VIDIOC_STREAMON, buf_type)

while True:  # capture 50 frames
    buf = v4l2_buffer()
    buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE
    buf.memory = V4L2_MEMORY_MMAP
    fcntl.ioctl(vd, VIDIOC_DQBUF, buf)  # get image from the driver queue
    mm = buffers[buf.index]
image_data=np.frombuffer(mm,dtype=np.uint8,count=fmt.fmt.pix.bytesperline*fmt.fmt.pix.h
eight).reshape(fmt.fmt.pix.height,fmt.fmt.pix.bytesperline)
    cv2.imshow("test",image_data)
    k= cv2.waitKeyEx(1)
    if k==27:
        exit(1)
    fcntl.ioctl(vd, VIDIOC_QBUF, buf)  # requeue the buffer
print(">> Stop streaming")
fcntl.ioctl(vd, VIDIOC_STREAMOFF, buf_type)

vd.close()
```

Revision history

Table 6. Document revision history

Date	Version	Changes
31-May-2024	1	Initial release
11-March-2025	1.1	Add fix for libcamera