Jungletronics

Raspberry Pi Camera Module

How To Connect the Rpicam, Take Pictures, Record Video, and Apply image effects #raspiSeries — Episode 1

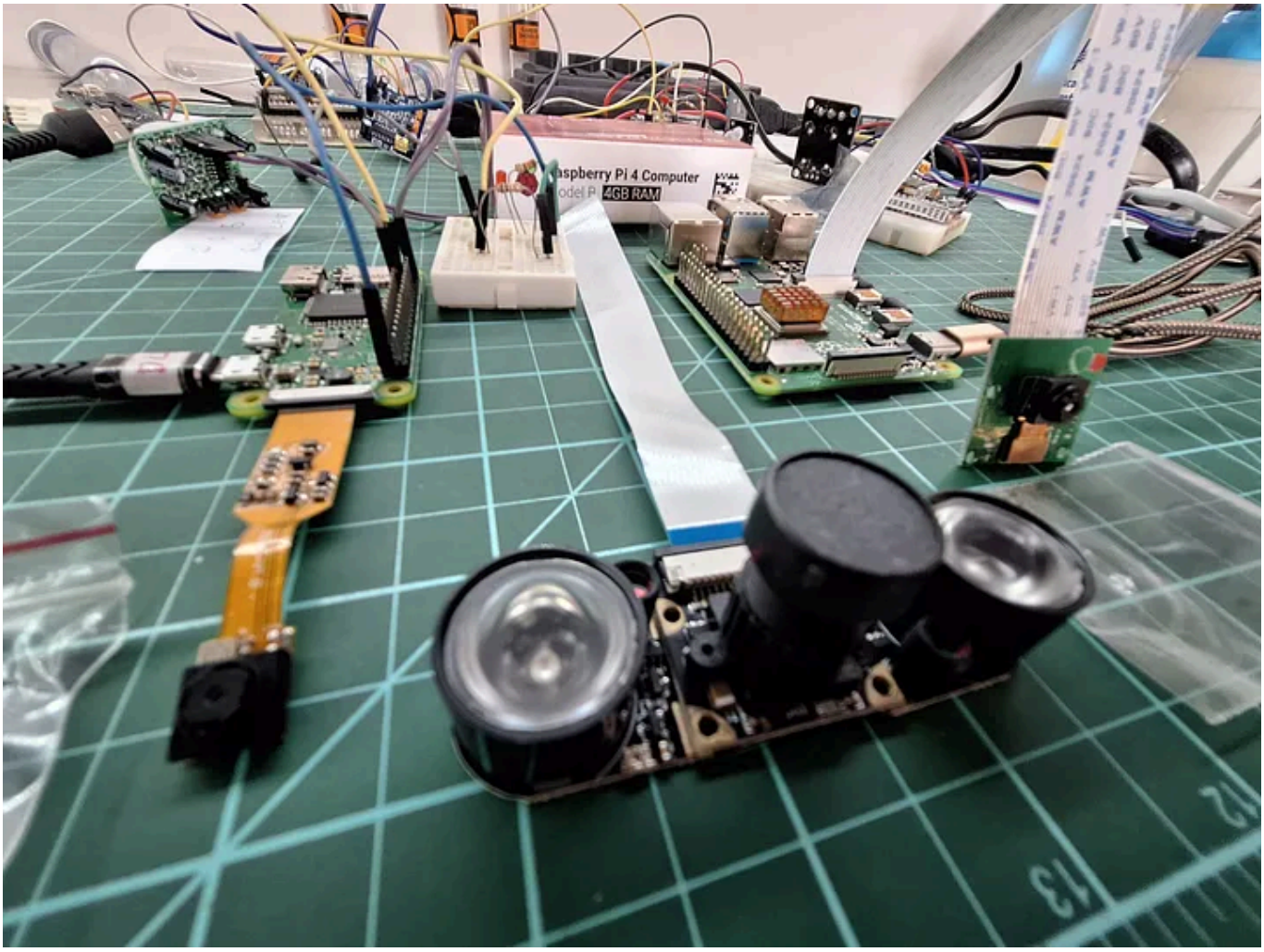
J3 Follow 11 min read · Aug 17, 2024

Hi there! Ready to have some fun with the Raspberry Pi Camera? Welcome!

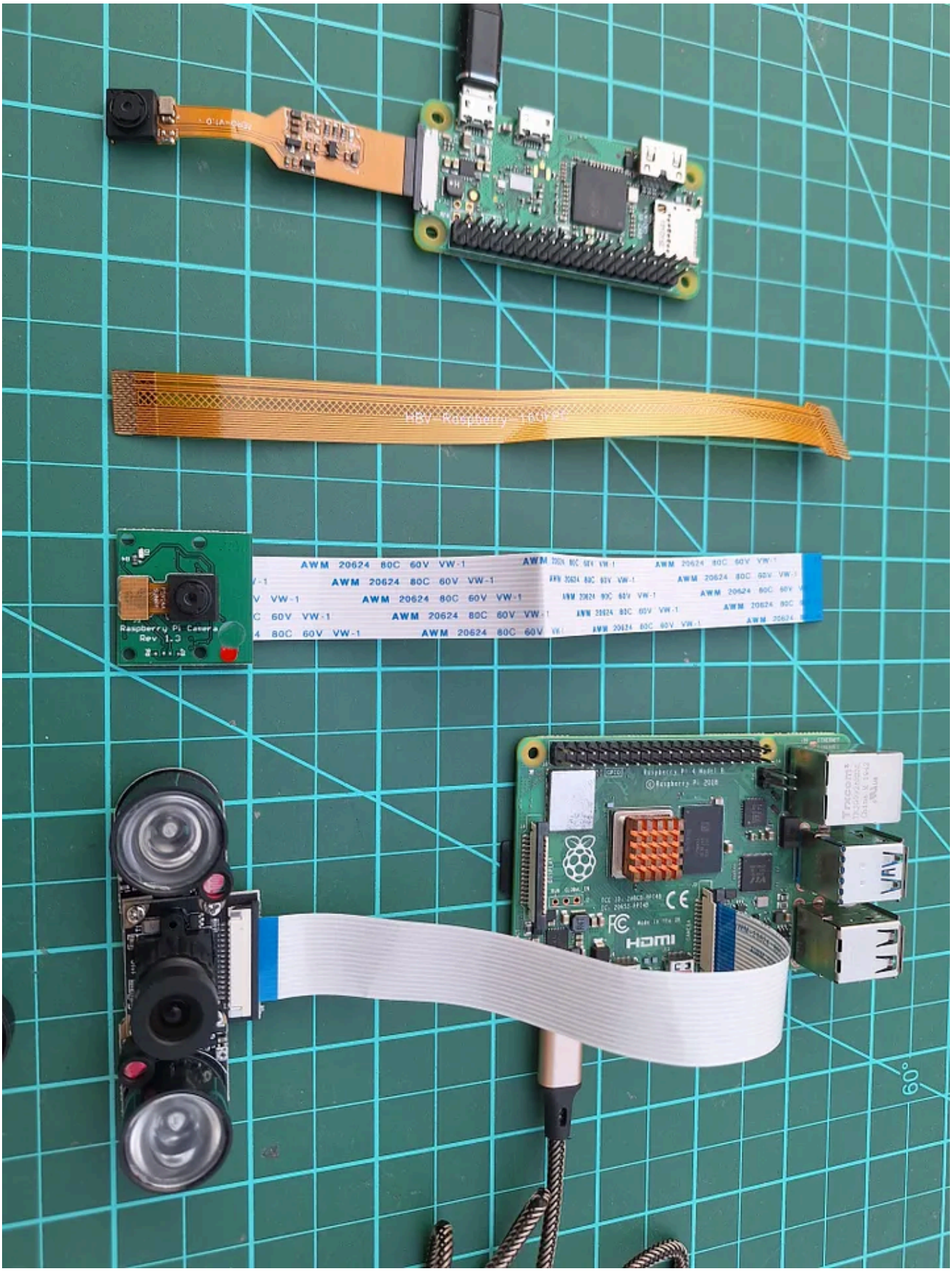
```
This episode guides you through
setting up your Raspberry Pi
with a camera and using terminal
commands to capture
photos and videos.

It then introduces using the Thonny IDE
to run scripts for photo previews
and video recording,
including flipping images
and setting resolutions.

It demonstrates how to capture an image
and convert it into a sketch-like image
using OpenCV.

Finally, how to record the timestamp in the
bottom-right corner of the photo,
```

```
which will play a key role in the
upcoming security system.
```

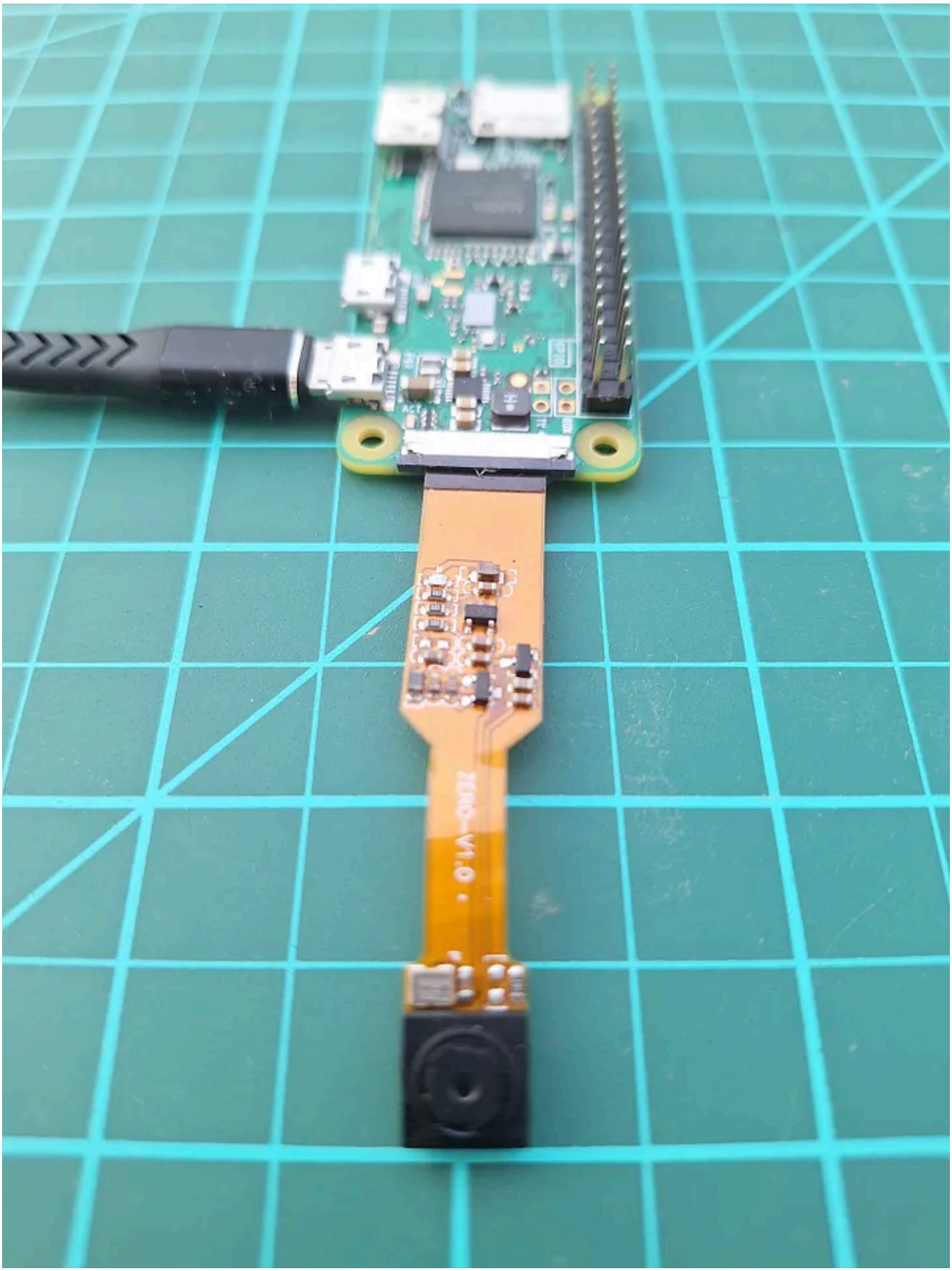
Here is my collections:



My workbench showcasing all my Raspberry Pi gear!

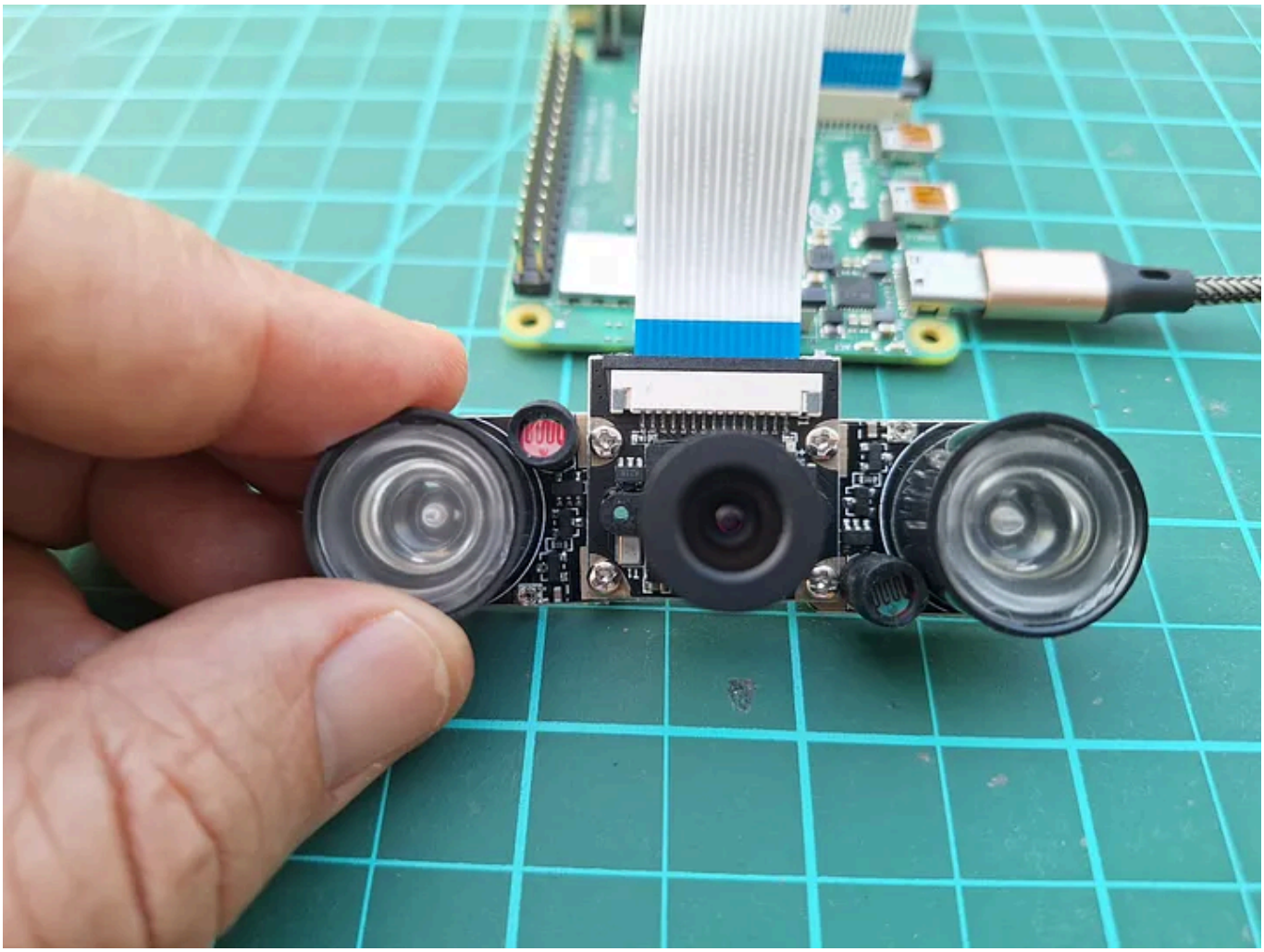


All three types of cameras mentioned above: Mini Size Camera 5MP OV5647 Sensor, Raspberry Pi Zero Camera Adapter, Raspberry Pi Mini Camera, Night Vision Camera for Raspberry Pi — IR-CUT 5MP (Version F)



Mini Size Camera 5MP OV5647 Sensor, 1080P HD Zero Camera Module for Raspberry Pi Modle Zero/RPi Zero W — 6 CM

In The center: Pi Zero Ribbon Cable Adapter 15CM 15 Pin to 22 Pin for Raspberry Pi 5/Zero/Zero W.



The IR LEDs, powered directly from the CSI port, can illuminate areas up to **8 meters** away. Optimal image quality is achieved at **distances** of **3 to 5 meters**. The camera is equipped with an a**djustable 3.6mm focal length lens** and offers a **75.7-degree viewing angle**. Getting started with the Camera Module. One issue with this type of camera is that it consumes too much power by constantly keeping the IR LED on. For battery-powered projects, consider using the official Raspberry Pi Noir Camera instead.

For production use, I recommend purchasing this camera!

Let's Get Started:

0 #Step — Please set your Raspberry Pi up and running in this **previous episode**.

To connect your camera with Raspberry Pi please follow official tutorial Getting started with the Camera Module.

Once your setup is ready let's code!

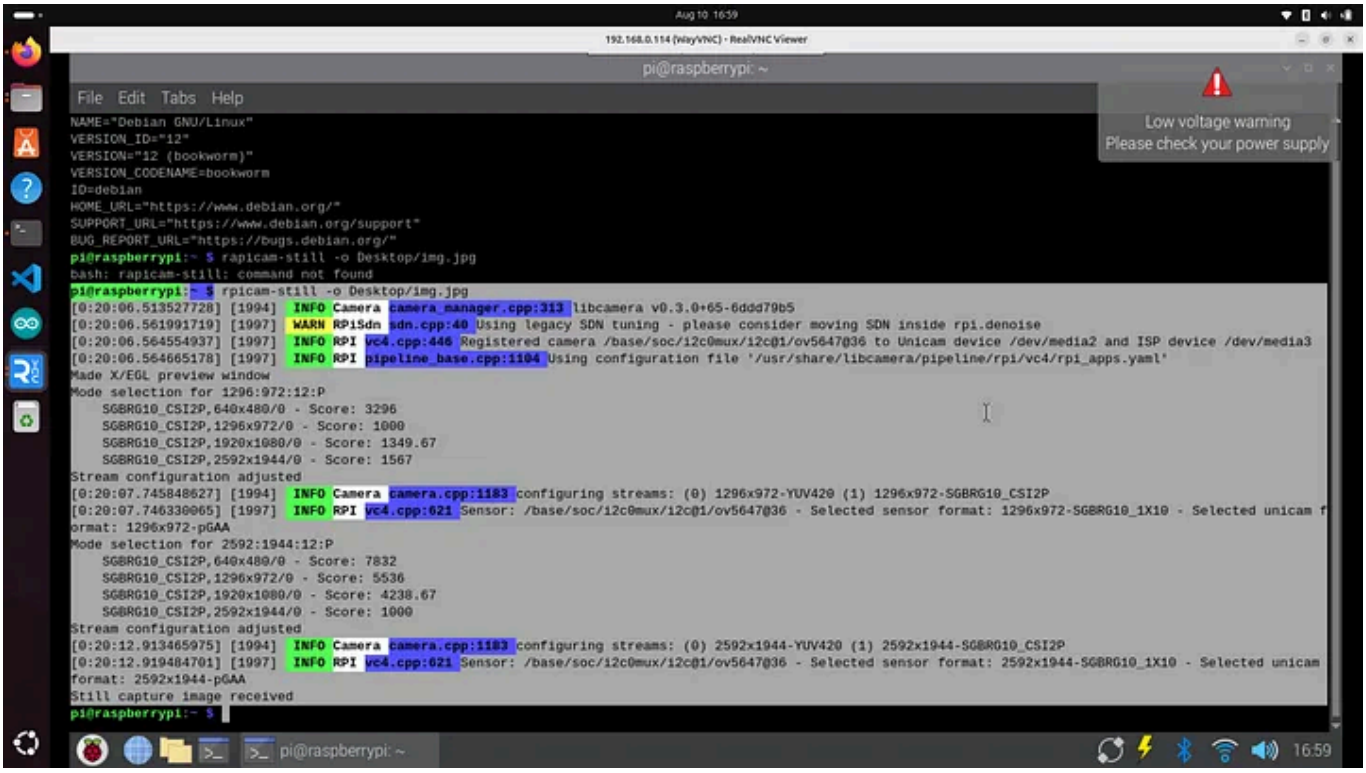
1 #Step — **Terminal** —Still —Take a Splash screen, Go To:

Terminal

```
pi@raspberrypi:~ $ rpicam-still -o Desktop/img.jpg
```

The command `rpicam-still -o Desktop/img.jpg` is used to capture an image with the Raspberry Pi Camera and save it to a specified location. Here's a breakdown:

- **`rpicam-still`:** This is a command-line tool for capturing still images using the Raspberry Pi Camera. It is commonly used for taking pictures in various formats with adjustable camera settings.
- **`-o Desktop/img.jpg`:** The `-o` option specifies the output file path where the captured image will be saved. In this case, the image will be saved as `img.jpg` in the `Desktop` directory.



Here is the output from my Raspberry Pi terminal. The image will be saved in the Desktop directory.

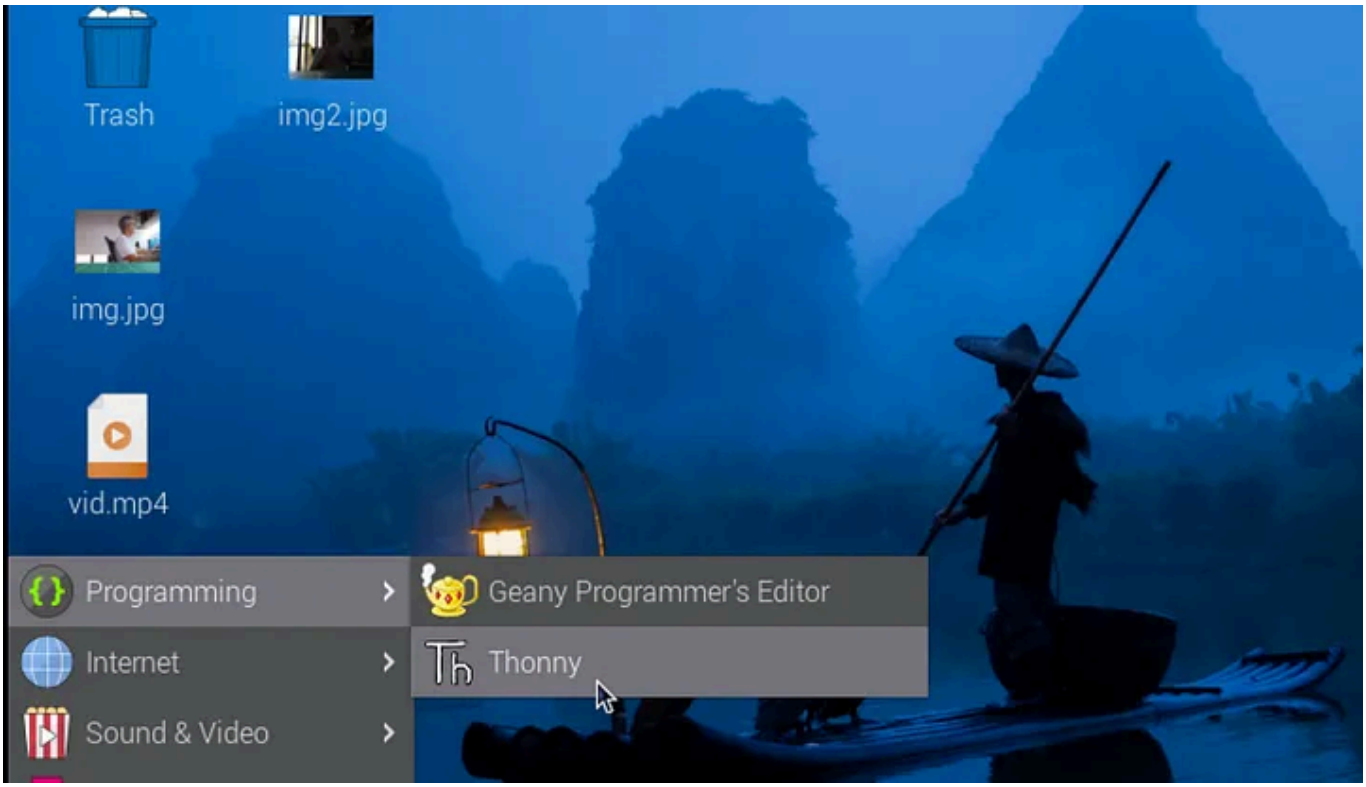
2 #Step — **Terminal** — Video — To record a video, On Terminal type:

```
pi@raspberrypi:~ $ rpicam-vid -o Desktop/vid.mp4
```

The command `rpicam-vid Desktop/vid.mp4`

- **`rpicam-vid`** : This is a command-line tool attempts to record a video with the Raspberry Pi Camera.
- **`-o Desktop/vid.mp4`** : The `-o` option specifies the output file, and `vid.mp4` is the video file being saved in the `Desktop` directory with the `.mp4` extension. To record a video, you would typically specify a video format such as `.h264`, `.mp4`, or `.mkv`.

Now let's use Thonny application:



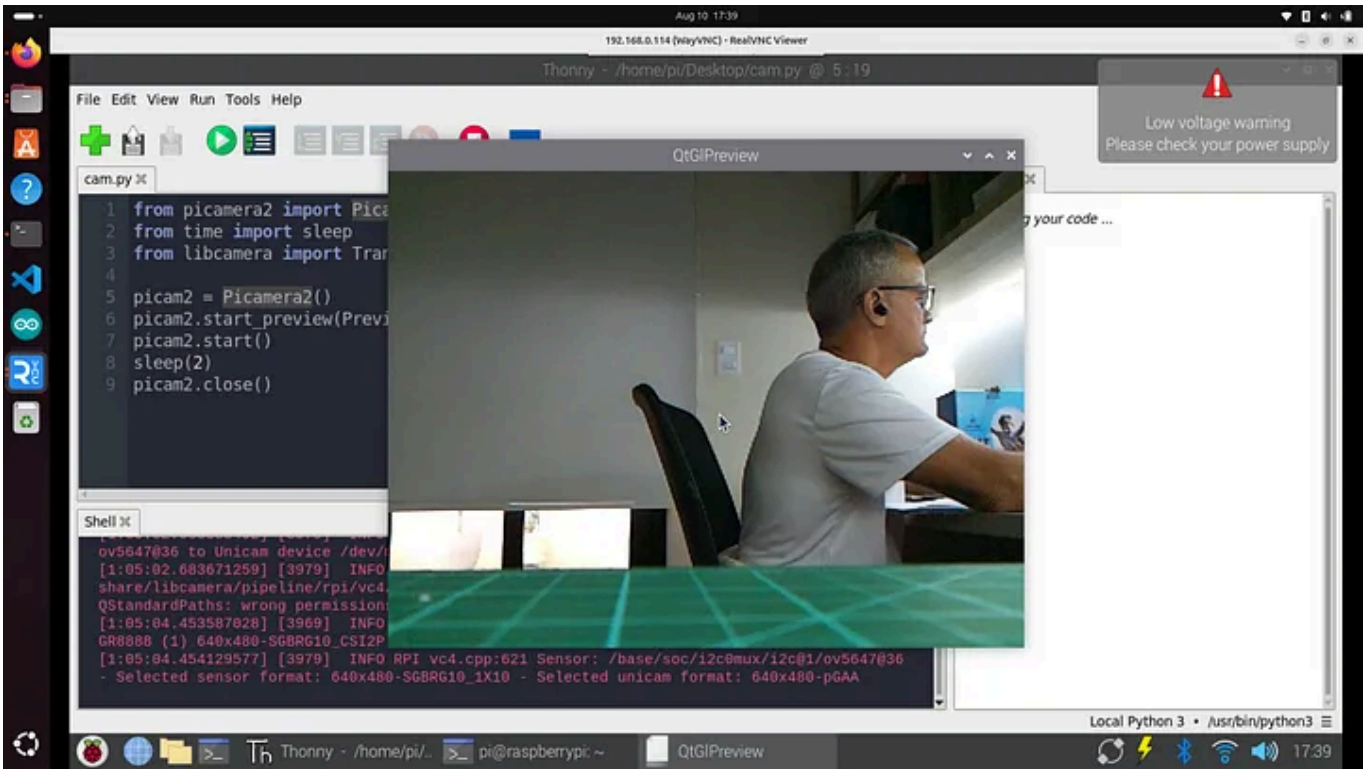
From now on, we'll be using the **Thonny** IDE. To open **Thonny**, navigate to **Programming** > **Thonny**, and then run the scripts below....

3 #Step — **Scripts For Photo Preview** —Thonny — Let's run one script and preview the image; All these code are on my GitHub Repo:

Desktop/**cam.py**

```
from picamera2 import Picamera2, Preview
from time import sleep
from libcamera import Transform

picam2 = Picamera2()
picam2.start_preview(Preview.QTGL, transform=Transform(hflip=True, vflip=True))
picam2.start()
sleep(2)
picam2.close()
```

This code initializes the Raspberry Pi camera using the `Picamera2` library. It starts a camera preview with horizontal and vertical flipping applied using the Transform class and displays it using the QTGL preview window. The preview runs for 2 seconds. After that, the camera is closed to stop the preview and release resources.



The preview runs for just 2 seconds and then disappears...

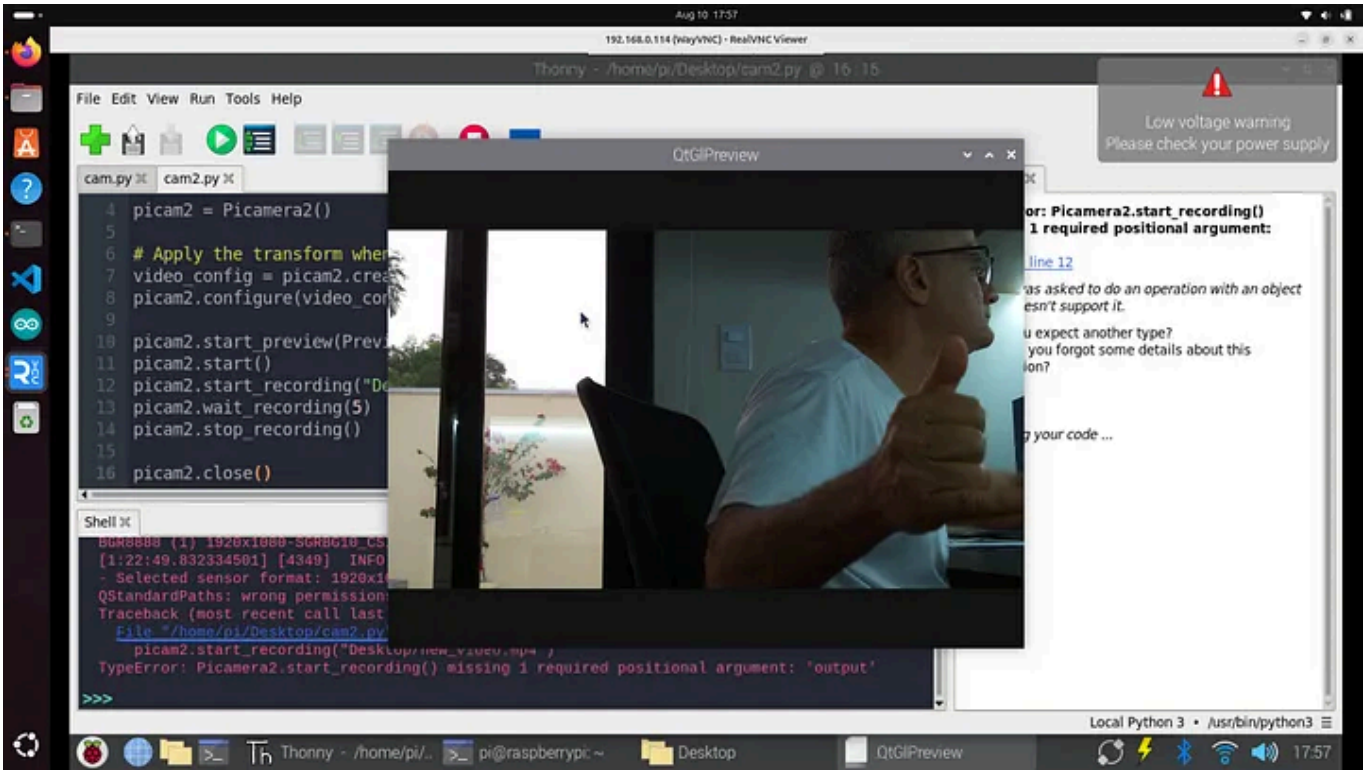
4 #Step — **Scripts For Video** — Thonny:

```
from picamera2 import Picamera2, Preview
from libcamera import Transform
picam2 = Picamera2()
# Apply the transform when configuring the camera
video_config = picam2.create_video_configuration(transform=Transform(hflip=True,
picam2.configure(video_config)
picam2.start_preview(Preview.QTGL)  # Optional: Show preview with flipping
picam2.start()
picam2.start_recording("Desktop/new_video.mp4")
picam2.wait_recording(5)  # Record for 5 seconds
```

```
picam2.stop_recording()
picam2.close()
```

Here's a brief explanation of the code:

1. **Import Libraries:** It imports `Picamera2` for camera operations and `Transform` for image manipulation.
2. **Initialize Camera:** Creates an instance of `Picamera2`.
3. **Configure Camera:** Sets up the camera configuration to flip the image horizontally and vertically.
4. **Start Preview and Recording:** Begins showing a preview (optional) and starts recording a video to `Desktop/`**`new_video.mp4`** for 5 seconds.
5. **Stop and Close:** Stops the recording and closes the camera.

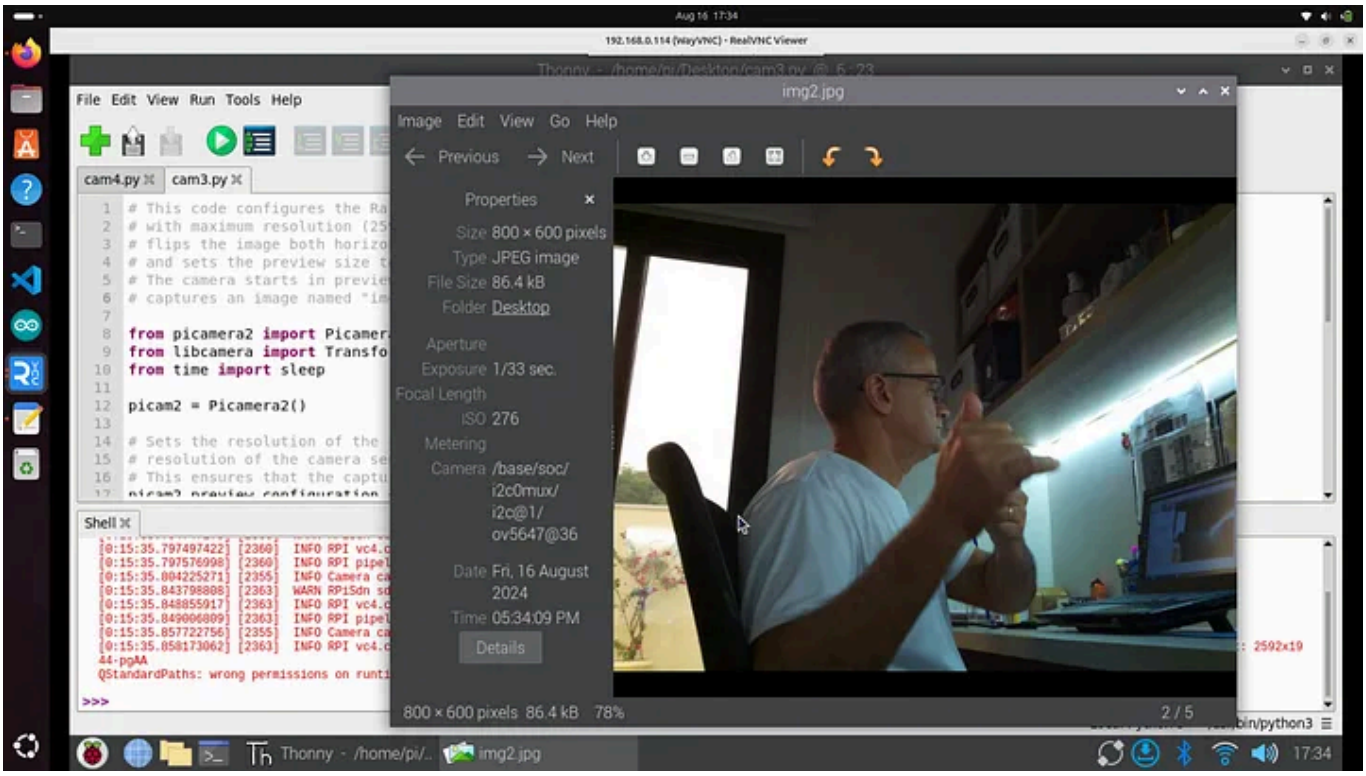


Here it is the **cam2.py** script running...

5 #Step — **Scripts for Image** — Set Resolutions — Thonny:

This code configures the Raspberry Pi camera with maximum resolution (2592x1944), flips the image both horizontally and vertically, and sets the preview size to 800x600. The camera starts in preview mode, waits 2 seconds, captures an image named `img2.jpg`, and then closes the camera.

```
from picamera2 import Picamera2
from libcamera import Transform
from time import sleep

picam2 = Picamera2()

# Sets the resolution of the sensor output to the maximum
# resolution of the camera sensor, which is 2592x1944 pixels.
# This ensures that the captured image will be of high quality.
picam2.preview_configuration.sensor.output_size = (2592, 1944)

# Sets the size of the preview window to 800x600 pixels.
# This controls the size of the image displayed on the screen
# while the camera is running in preview mode.
picam2.preview_configuration.main.size = (800, 600)

# Apply the transform to flip the image horizontally and vertically
picam2.preview_configuration.transform = Transform(hflip=True, vflip=True)

# Configure the camera for preview
picam2.configure("preview")

# Start the camera with the preview
picam2.start(show_preview=True)

# Pause for 2 seconds to allow the camera to stabilize
sleep(2)

# Capture the image and save it as 'max.jpg'
picam2.capture_file("img2.jpg")

# Close the camera
picam2.close()
```



Here it is the result: img2.jpg.

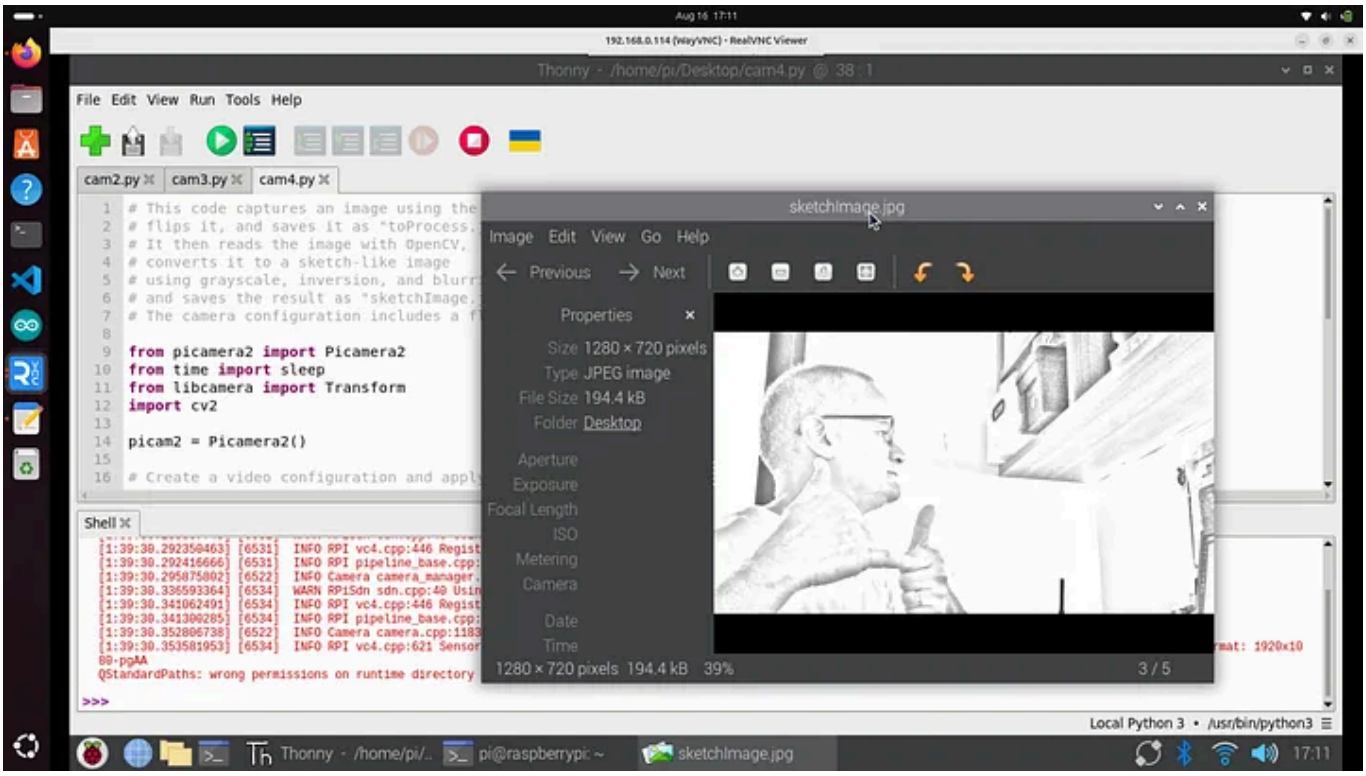
6 #Step — **Sketch-like** image — Thonny:

This code captures an image using the Raspberry Pi camera, flips it, and saves it as “toProcess.jpg.” It then reads the image with OpenCV, converts it to a sketch-like image using grayscale, inversion, and blurring techniques, and saves the result as “sketchImage.jpg.” The camera configuration includes a flipped preview with specified dimensions.

Note: Install OpenCV first on Raspi Terminal; See here all about OpenCV Tutorials:

```
sudo apt update
sudo apt install python3-opencv
```

```
from picamera2 import Picamera2
from time import sleep
from libcamera import Transform
import cv2

picam2 = Picamera2()

# Create a video configuration and apply the transform for flipping
video_config = picam2.create_video_configuration(transform=Transform(hflip=True,
picam2.configure(video_config)

# Set the resolution for 800 x 600
picam2.preview_configuration.size = (800, 600)
picam2.start(show_preview=True)

sleep(2)

picam2.capture_file("toProcess.jpg")
picam2.close()

img = cv2.imread("toProcess.jpg")

# Convert to sketch
greyscale = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
inverted = 255 - greyscale
blur_inverted = cv2.GaussianBlur(inverted, (125, 125), 0)
inverted_blur = 255 - blur_inverted
sketch = cv2.divide(greyscale, inverted_blur, scale=256)
cv2.imwrite("sketchImage.jpg", sketch)
```



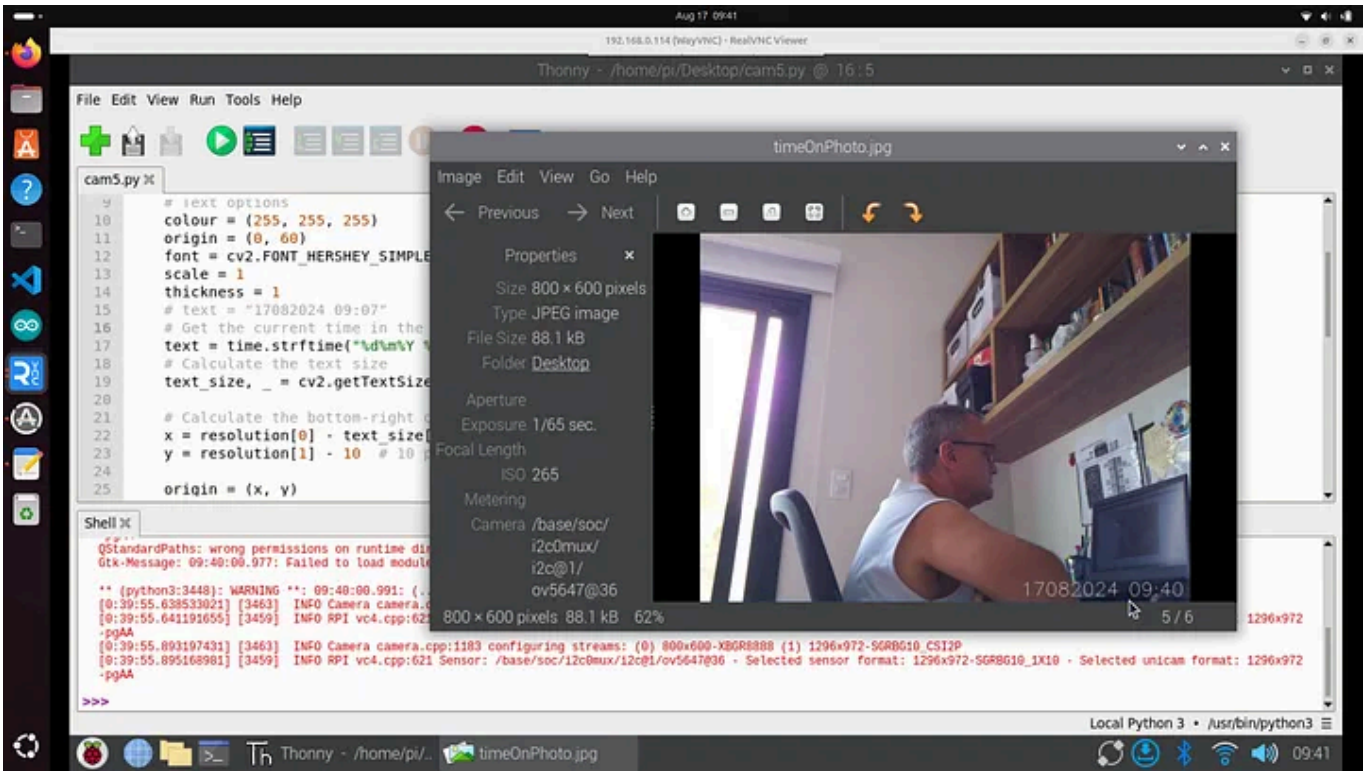
Here is the result: **sketchImage.jpg**

7 #Step — **TimeStamp** image— Thonny:

This script captures an image with a Raspberry Pi camera, flips it both horizontally and vertically, and overlays the current date and time in the bottom-right corner. The `apply_text` function dynamically calculates the text's position using OpenCV. The camera is configured for preview and capture modes with the same resolution to maintain overlay consistency. The `picam2.pre_callback` inserts the timestamp before capturing the image, which is then saved as `timeOnPhoto.jpg`.

```
from picamera2 import Picamera2, MappedArray
from libcamera import Transform
import cv2, time

resolution = (800, 600)


def apply_text(request):
```

```
        # Text options
        colour = (255, 255, 255)
        origin = (0, 60)
        font = cv2.FONT_HERSHEY_SIMPLEX
        scale = 1
        thickness = 1
        # text = "17082024 09:07"
        # Get the current time in the format "DDMMYYYY HH:MM"
        text = time.strftime("%d%m%Y %H:%M")
        # Calculate the text size
        text_size, _ = cv2.getTextSize(text, font, scale, thickness)

        # Calculate the bottom-right origin
        x = resolution[0] - text_size[0] - 10  # 10 pixels padding from the right
        y = resolution[1] - 10  # 10 pixels padding from the bottom

        origin = (x, y)
        with MappedArray(request, "main") as m:
            cv2.putText(m.array, text, origin, font, scale, colour, thickness)


    # Create camera object
    picam2 = Picamera2()

    # Create two separate configs - one for preview and one for capture.
    # Make sure the preview is the same resolution as the capture, to make
    # sure the overlay stays the same size
    capture_config = picam2.create_still_configuration({"size": resolution}, transfo
    preview_config = picam2.create_preview_configuration({"size": resolution}, trans

    # Set the current config as the preview config
    picam2.configure(preview_config)

    # Add the timestamp
    picam2.pre_callback = apply_text
    # Start the camera
    picam2.start(show_preview=True)
    time.sleep(2)

    # Switch to the capture config and then take a picture
    image = picam2.switch_mode_and_capture_file(capture_config, "timeOnPhoto.jpg")

    # Close the camera
    picam2.close()
```



Here is the result. Notice the timestamp in the bottom-right corner of the photo, which will play a key role in the upcoming security system.

That's it for now folks!

In the next episode, we'll Build a Residential Intrusion Detection System.

See you soon!

Bye!

👉GitHub Repo Episode 1

👌 Review all additions.

Credits & References

Raspberry Pi OS by raspberrypi.com

Related Posts

Raspberry PI

0#Episode — **#raspiSeries** — Raspberry Pi Intro — Quick And Easy Way To Install Raspberry Pi OS

1#Episode — **#raspiSeries** — Raspberry Pi Camera Module — How To Connect the Rpicam, Take Pictures, Record Video, and Apply image effects

2#Episode — **#raspiSeries** — Raspberry Pi Camera Project — How to Build a Residential Intrusion Detection System

3#Episode — **#raspiSeries** — Raspberry Pi Camera Project — How to Build a Residential Intrusion Detection System one)

OpenCV

00 Episode#Hi Python Computer Vision — PIL! — **An Intro To Python Imaging Library** #PyVisionSeries

01 Episode# Jupyter-lab — Python — **OpenCV** — Image Processing Exercises #PyVisionSeries

02 Episode# OpenCV — **Image Basics** — Create Image From Scratch #PyVisionSeries

03 Episode# OpenCV — **Morphological Operations** — How To Erode, Dilate, Edge Detect w/ Gradient #PyVisionSeries

04 Episode# OpenCV — **Histogram Equalization** — HOW TO Equalize Histograms Of Images — #PyVisionSeries

05 Episode# OpenCV — OpenCV — **Resize an Image** — How To Resize Without Distortion

07 Episode# **YOLO — Object Detection** — The state of the art in object detection Framework!

08 Episode# OpenCV — **HaashCascate — Object Detection** — Viola–Jones object detection framework — #PyVisionSeries

Notes:

When the scripts are executed, the following logs appear:

```
INFO Camera camera_manager.cpp:313 libcamera v0.3.0+65-6ddd79b5
WARN RPiSdn sdn.cpp:40 Using legacy SDN tuning - please consider moving SDN inside r
INFO RPI vc4.cpp:446 Registered camera /base/soc/i2c0mux/i2c@1/ov5647@36 to Unicam d
INFO RPI pipeline_base.cpp:1104 Using configuration file '/usr/share/libcamera/pipel
INFO Camera camera_manager.cpp:313 libcamera v0.3.0+65-6ddd79b5
WARN RPiSdn sdn.cpp:40 Using legacy SDN tuning - please consider moving SDN inside r
INFO RPI vc4.cpp:446 Registered camera /base/soc/i2c0mux/i2c@1/ov5647@36 to Unicam d
INFO RPI pipeline_base.cpp:1104 Using configuration file '/usr/share/libcamera/pipel
sions on runtime directory /run/user/1000, 0770 instead of 0700
INFO Camera camera.cpp:1183 configuring streams: (0) 640x480-XBGR8888 (1) 640x480-SG
INFO RPI vc4.cpp:621 Sensor: /base/soc/i2c0mux/i2c@1/ov5647@36 - Selected sensor for
```

The log messages we're seeing are generated when we run our `cam.py` script. Here's an explanation of what's happening:

libcamera Initialization:

The libcamera version is identified as v0.3.0+65–6ddd79b5. This is the library managing the camera hardware on your Raspberry Pi.

Camera Registration:

The camera sensor model **ov5647** (a common Raspberry Pi camera module) is registered to the **Unicam device** (/dev/media3) and the ISP (Image Signal Processor) device (/dev/media1).

Legacy SDN Tuning Warning:

The warning indicates that the camera is using a legacy SDN (Sensor Device Node) tuning configuration. It suggests updating the configuration to place the SDN inside rpi.denoise, which is part of the Raspberry Pi's image processing pipeline.

Configuration File:

The camera is using a configuration file located at **/usr/share/libcamera/pipeline/rpi/vc4/rpi_apps.yaml**. This file defines how the camera pipeline is configured for Raspberry Pi.

Stream Configuration:

The camera is configured to stream in two formats:
640x480-XBGR8888: This is the format for the camera preview (display).
640x480-SGBRG10_CSI2P: This is the raw format from the camera sensor, which uses 10-bit Bayer pattern (SGBRG).

QStandardPaths Warning:

The warning about "wrong permissions on runtime directory" suggests that

the directory /run/user/1000 has incorrect permissions (set to 0770 instead of the recommended 0700). This could cause permission-related issues, though it may not directly impact camera functionality.

Sensor Format:

The selected sensor format is 640x480-SGBRG10_1X10, which corresponds to the 10-bit Bayer format. The unicam format is set to 640x480-p, indicating progressive scan at this resolution.

Summary:

The camera has been successfully detected and configured, but you might need to address the legacy SDN tuning and directory permission warnings for optimal performance. The camera is ready to capture images or video in 640x480 resolution.

The warning message you encountered suggests that your Raspberry Pi camera setup is using a legacy `SDN` (Spatial Denoise) tuning, and the system recommends moving the SDN configuration inside `rpi.denoise`.

This typically involves modifying the camera tuning file, but since `libcamera` handles most configurations automatically, manual changes might be necessary in the configuration files used by `libcamera` for your camera pipeline. Below is a step-by-step guide to updating the configuration:

Steps to Move SDN Inside `rpi.denoise`:

1 . Locate the Tuning File:

- The configuration files are usually located in `/usr/share/libcamera/pipeline/rpi/`.
- The exact file for tuning depends on your camera model, which in my case appears to be related to the `ov5647` sensor.

2 . Backup the Current Configuration:

- Before making any changes, it's always a good idea to back up the existing tuning file:

```
sudo cp /usr/share/libcamera/pipeline/rpi/vc4/rpi_apps.yaml /usr/share/libcamera
```

3 . Edit the Configuration File:

- Open the tuning file for your camera in a text editor:

```
sudo nano /usr/share/libcamera/pipeline/rpi/vc4/rpi_apps.yaml
```

- Look for the section that contains the SDN configuration. It will likely be under the `denoise` settings or as a standalone section.

4 . Modify the SDN Section:

- If the SDN tuning is found outside of `rpi.denoise`, move the relevant configuration under the `rpi.denoise` block. The configuration might look something like this:

```
rpi.denoise:
  sdn:
    # Your SDN configuration here
```

- Ensure the SDN parameters are correctly nested within the `rpi.denoise` block.

5 . Save and Reboot:

- Save the changes by pressing `CTRL + O`, then press `Enter`. Exit the editor with `CTRL + X`.
- Reboot the Raspberry Pi to apply the changes.

Here is the procedure:

```
cd /usr/share/libcamera/pipeline/rpi/vc4
ls
sudo nano rpi_apps.yaml
```

For your reference, here is the complete file with the `rpi.denoise` settings applied:

/usr/share/libcamera/pipeline/rpi/vc4/**rpi_apps.yaml**

```
{
        "version": 1.0,
        "target": "bcm2835",

        "pipeline_handler":
        {       # The minimum number of internal buffers to be allocated for
                # Unicam. This value must be greater than 0, but less than or
                # equal to min_total_unicam_buffers.
                #
                # A larger number of internal buffers can reduce the occurrence
                # of frame drops during high CPU loads, but might also cause
                # additional latency in the system.
                #
                # Note that the pipeline handler might override this value and
                # not allocate any internal buffers if it knows they will never
                # be used. For example if the RAW stream is marked as mandatory
                # and there are no dropped frames signalled for algorithm
                # convergence.
                #
                "min_unicam_buffers": 2,

                # The minimum total (internal + external) buffer count used for
                # Unicam. The number of internal buffers allocated for Unicam is
                # given by:
                #
                # internal buffer count = max(min_unicam_buffers,
                #         min_total_unicam_buffers - external buffer count)
                #
                "min_total_unicam_buffers": 4,

                # Override any request from the IPA to drop a number of startup
                # frames.
                #
                # "disable_startup_frame_drops": false,

                # The application will always provide a request buffer for the
                # RAW stream, if it has been configured.
                "raw_mandatory_stream": true,

                # The application will always provide a request buffer for the
                # Output 0 stream, if it has been configured.
                "output0_mandatory_stream": true
        },
```

```
        # Adding the denoise section with SDN configuration
        "rpi.denoise":
        {
                "mode": "auto",  # Options: "auto", "off", "cdn_off", "cdn_hq"
                "sdn":
                {
                        "strength": 0.5  # Adjust the strength of the spatial de
                }
        }
}
```

```
Linux Terminal Commands:

[Access the REPO]
https://github.com/giljr/Raspberry_new_repo

[Copy Raspberry Pi Desktop *py to Ubuntu - Change Port for Your Connection]
sudo scp pi@192.168.0.114:/home/pi/Desktop/*.py /home/j3/Documents/raspi_project

[For instance:]
j3@j3-LAPTOP:~$ sudo scp pi@192.168.0.114:/home/pi/Desktop/cam5.py /home/j3/Docu
[sudo] password for j3:
pi@192.168.0.114's password:
cam5.py

[Change the Owner for your User Name]
sudo chown -R j3:j3 /home/j3/Documents/raspi_projects/Episode_1
```

Raspberry Pi 4 | Raspicam | Cameras | Night Vision Devices | Timestamp Camera

Published in Jungletronics

472 followers · Last published 3 days ago

Follow

Explore our insights on Django, Python, Rails, Ruby, and more. We share code, hacks, and academic notes for developers and tech enthusiasts. Happy reading!

Written by J3

1.2K followers · 68 following

Follow

😎 Gilberto Oliveira Jr | 🖥️ Computer Engineer | Python | 🧩 C | 💎 Rails | 🤖 AI & IoT | ✍️

No responses yet

Write a response

What are your thoughts?

More from J3 and Jungletronics

In Jungletronics by J3

In Jungletronics by J3

From Zero to Hotwire—Rails 8

Create a Rails 8 Blog with Turbo Streams #HotwirePoweredSeries—Episode 0

Mar 8 14 3

How To Securely Save Credentials in Python

Like API tokens, passwords, or other sensitive data #PurePythonSeries—Episode #19

Aug 25, 2024 11

In Jungletronics by J3

Mastering Import Maps in Rails 8

A Step-by-Step Guide to Efficient JavaScript Management # Rails8Series—Episode 2

Jan 10 13

In Jungletronics by J3

Mastering Enums in Rails 8

A Practical Guide to Simplify and Enhance Your ActiveRecord Models # Rails8Series—…

Jan 23 10 2

See all from J3 See all from Jungletronics