Yes! You can **modify the "Duplicated RGB" window** to only open **when needed** by a YOLO model while keeping the **camera processing running in the background**.

**How to Make YOLO Trigger the Duplicate RGB Window**

1. **Modify camera\_processing.py** to keep the **Duplicate RGB window closed by default**.
2. **Use a shared flag (multiprocessing.Value or a shared variable)** to **open the window only when requested**.
3. **Trigger the window from YOLO** when detection is required.

**Step 1: Modify camera\_processing.py**

Update the script to **only show "Duplicate RGB" when requested**:

python

CopyEdit

import cv2

import numpy as np

import multiprocessing

# Shared flag to control "Duplicate RGB" window visibility

show\_rgb\_window = multiprocessing.Value('b', False) # Boolean flag (False by default)

# Create a function that YOLO can call to open the window

def toggle\_rgb\_window(state):

"""Function to toggle the visibility of the 'Duplicate RGB' window."""

show\_rgb\_window.value = state

# Create display windows

cv2.namedWindow('All Cameras', cv2.WINDOW\_NORMAL)

cv2.resizeWindow('All Cameras', 1920, 1080)

def preprocess\_image(img):

"""Ensure all images have 3 channels (convert to RGB if needed)."""

if img.shape[2] == 4: # If image has 4 channels (RGBA)

img = img[:, :, :3] # Drop the alpha channel

return img

def add\_label(image, label, position=(10, 50), font\_scale=0.8, color=(255, 255, 255)):

"""Adds a text label to an image."""

labeled\_image = image.copy() # Work on a copy to avoid modifying original

cv2.putText(labeled\_image, label, position, cv2.FONT\_HERSHEY\_SIMPLEX,

font\_scale, color, 2, cv2.LINE\_AA)

return labeled\_image # Return the labeled image

while True:

try:

# Convert all sensor images to 3-channel RGB format

sensor\_data\_processed = {key: preprocess\_image(img) for key, img in sensor\_data.items()}

# Check if all images are available before displaying

if any(img.shape[0] == 0 for img in sensor\_data\_processed.values()):

print("Waiting for camera feeds...")

continue

# Get the RGB image \*\*without\*\* label for the Duplicate RGB window

rgb\_clean = sensor\_data\_processed['rgb\_image'].copy()

# Create a \*\*labeled\*\* RGB image for the "All Cameras" window

rgb\_labeled = add\_label(sensor\_data\_processed['rgb\_image'], "RGB Camera")

# Concatenate images in a 2-row format for display

top\_row = np.concatenate([

rgb\_labeled, # \*\*Labeled RGB Image for "All Cameras"\*\*

add\_label(sensor\_data\_processed['sem\_image'], "Semantic Segmentation"),

add\_label(sensor\_data\_processed['inst\_image'], "Instance Segmentation")

], axis=1)

lower\_row = np.concatenate([

add\_label(sensor\_data\_processed['depth\_image'], "Depth Camera"),

add\_label(sensor\_data\_processed['dvs\_image'], "DVS Camera"),

add\_label(sensor\_data\_processed['opt\_image'], "Optical Flow")

], axis=1)

# Combine both rows into a single tiled view

tiled = np.concatenate((top\_row, lower\_row), axis=0)

# Display the combined camera feeds in the main window

cv2.imshow('All Cameras', tiled)

# Show "Duplicate RGB" window \*\*only if enabled\*\*

if show\_rgb\_window.value:

cv2.imshow("Duplicate RGB", rgb\_clean)

else:

cv2.destroyWindow("Duplicate RGB") # Close the window when not needed

# Exit on pressing 'q'

if cv2.waitKey(1) == ord('q'):

break

except Exception as e:

print(f"Error displaying images: {e}")

# Cleanup: Close all OpenCV windows

cv2.destroyAllWindows()

**Step 2: Modify Your Main Script to Start camera\_processing.py in Background**

Now, your **main script** will:

* Start camera\_processing.py in the background.
* Allow **YOLO** to enable/disable the "Duplicate RGB" window when needed.

**Updated Main Script (main.py):**

python

CopyEdit

import subprocess

import time

import multiprocessing

# Start the camera processing script in the background

camera\_process = subprocess.Popen(['python', 'camera\_processing.py'])

print("Camera processing is running in the background!")

# Shared flag to control "Duplicate RGB" window visibility

show\_rgb\_window = multiprocessing.Value('b', False) # Boolean flag (False by default)

# Function that YOLO can call to enable/disable the "Duplicate RGB" window

def trigger\_rgb\_window(state):

"""Enable or disable the 'Duplicate RGB' window."""

show\_rgb\_window.value = state

print(f"🚀 Duplicate RGB Window {'Enabled' if state else 'Disabled'}!")

# Simulate YOLO detecting an object

time.sleep(5) # Wait for camera to initialize

print("Simulating YOLO object detection...")

# \*\*Trigger the 'Duplicate RGB' window for 10 seconds\*\*

trigger\_rgb\_window(True)

time.sleep(10)

trigger\_rgb\_window(False)

# Keep the script running

try:

while True:

time.sleep(1)

except KeyboardInterrupt:

print("\nStopping camera processing...")

camera\_process.terminate()

print("Camera process stopped.")

**Step 3: Call the trigger\_rgb\_window() Function in YOLO**

If you are running **YOLO object detection**, modify your detection callback to **enable "Duplicate RGB" only when an object is detected**.

**Example YOLO Detection Callback:**

python

CopyEdit

def yolo\_detection\_callback(detected\_objects):

"""Trigger the RGB window only when an object is detected."""

if detected\_objects:

print("🔴 Object detected! Showing 'Duplicate RGB' window...")

trigger\_rgb\_window(True) # Enable window

else:

print("🟢 No object detected. Hiding 'Duplicate RGB' window.")

trigger\_rgb\_window(False) # Disable window

**Final Setup**

| **Action** | **Effect** |
| --- | --- |
| **YOLO detects an object** | Opens "Duplicate RGB" window |
| **YOLO does not detect anything** | Closes "Duplicate RGB" window |
| **Runs in background?** | ✅ Yes |
| **User presses 'Q'** | Exits cleanly |

🚀 **Now, the "Duplicate RGB" window will only appear when YOLO requests it while the camera runs in the background!** 🎯