Full Code with Comments

Python

```python
from picamera2 import Picamera2
import cv2 as cv
import numpy as np

# Function that does nothing, used as a placeholder for trackbar callback
def nothing(x):
    pass

# Create a window for trackbars
cv.namedWindow('trackbars')
cv.resizeWindow('trackbars', (480, 400))  # Resize the trackbar window

# Create trackbars for adjusting the HSV color range
cv.createTrackbar('huelower', 'trackbars', 50, 179, nothing)  # Lower bound for Hu
cv.createTrackbar('huehigher', 'trackbars', 100, 179, nothing) # Upper bound for H
cv.createTrackbar('satlower', 'trackbars', 50, 255, nothing)    # Lower bound for
cv.createTrackbar('sathigher', 'trackbars', 255, 255, nothing)  # Upper bound for
cv.createTrackbar('vallower', 'trackbars', 50, 255, nothing)    # Lower bound for
cv.createTrackbar('valhigher', 'trackbars', 255, 255, nothing)  # Upper bound for

# Initialize Picamera2
picam2 = Picamera2()
config = picam2.create_preview_configuration(main={"format": "RGB888", "size": (64
picam2.configure(config)
picam2.start()

while True:
    # Capture frame from PiCamera2
    frame = picam2.capture_array()
    rframe = cv.resize(frame, (640, 480))  # Ensure consistent size
    cv.imshow("original", rframe)

    # Convert frame to HSV color space
    hsv = cv.cvtColor(rframe, cv.COLOR_BGR2HSV)

    # Get current positions of trackbars
    HL = cv.getTrackbarPos('huelower', 'trackbars')
    HU = cv.getTrackbarPos('huehigher', 'trackbars')
    SL = cv.getTrackbarPos('satlower', 'trackbars')
    SH = cv.getTrackbarPos('sathigher', 'trackbars')
    VL = cv.getTrackbarPos('vallower', 'trackbars')
    VH = cv.getTrackbarPos('valhigher', 'trackbars')
```

```
        # Define lower and upper bounds for color filtering
        l_b = np.array([HL, SL, VL])
        u_b = np.array([HU, SH, VH])

        # Create mask for the HSV range
        FGmask = cv.inRange(hsv, l_b, u_b)
        cv.imshow("FGmask", FGmask)

        # Extract the foreground using the mask
        FG = cv.bitwise_and(rframe, rframe, mask=FGmask)
        cv.imshow("FG", FG)

        # Create the background mask (inverse of foreground mask)
        BGmask = cv.bitwise_not(FGmask)
        cv.imshow('BG', BGmask)

        # Convert the grayscale background mask to a 3-channel image
        BG = cv.cvtColor(BGmask, cv.COLOR_GRAY2BGR)

        # Combine the extracted foreground and background
        final = cv.add(FG, BG)
        cv.imshow('final', final)

        # Exit the loop when 'q' key is pressed
        if cv.waitKey(1) & 0xff == ord('q'):
            break

    # Stop the camera and close OpenCV windows
    picam2.stop()
    cv.destroyAllWindows()
```

Explanation of the Code's Functionality

The Python script uses the **Picamera2** library to capture a live video feed from a Raspberry Pi camera and the **OpenCV** library to perform real-time image processing. Its primary purpose is to demonstrate **color-based image segmentation**, where it isolates an object of a specific color from its background.

Here's a breakdown of how the code works:

1. Setup and Initialization

- The script imports necessary libraries: `Picamera2` for camera access, `cv2` (OpenCV) for image processing, and `numpy` for handling numerical data.
- It creates a graphical user interface (GUI) with **trackbars**. These interactive sliders allow the user to adjust the **Hue**, **Saturation**, and **Value** ranges in real-time.
- The `Picamera2` object is initialized and configured to capture a 640x480 pixel video stream in `RGB888` format.

2. Main Processing Loop

- The script enters an infinite `while True` loop to continuously capture and process video frames.
- **Image Capture**: A single frame is captured from the camera.
- **Color Space Conversion**: The captured frame is converted from its default **BGR** (Blue-Green-Red) color space to **HSV** (Hue-Saturation-Value). This is a crucial step because HSV separates color information (Hue) from brightness (Value) and intensity (Saturation), making it much easier to isolate a specific color.
- **Applying the Filter**:
  - The values from the trackbars are read to define the lower and upper bounds of the desired color range. These are stored in NumPy arrays, for example, `[h_min, s_min, v_min]` and `[h_max, s_max, v_max]` .
  - The `cv.inRange()` function creates a **binary mask**. This is a black-and-white image where any pixel within the defined HSV range becomes white (255), and all other pixels become black (0).
- **Separating Foreground and Background**:
  - `cv.bitwise_and()` is used with the binary mask to create a **foreground** image. This image contains only the pixels from the original frame that matched the color filter, with the rest of the image being black.
  - The `cv.bitwise_not()` function creates an inverse mask, which is used to isolate the **background**.
- **Combining the Images**:
  - The background mask is converted to a 3-channel image to match the foreground image.
  - Finally, the `cv.add()` function combines the foreground image with the background image. The result is a new image where the object of the target color is placed on a clean, solid white background.

3. Exit and Cleanup

- The loop continues until the user presses the 'q' key.
- Once the loop is broken, the `picam2.stop()` function turns off the camera, and `cv.destroyAllWindows()` closes all the display windows created by OpenCV.