Artificial Intelligence

Group Project

Submitted to: Chandani Kumari

Submitted by:

Hardik Verma (102283044)

Aalok Kumar Yadav (102103846)

Aarushi Juneja (102103781)

Daksh Sharma (102153043)

Problem Statement:

To detect multiple colours on real time (RGB).

Problem Description:

In this project, the fundamentals of computer vision are used to track three different colors: Red, Blue, and Green. When the code is run, a window will open using the webcam, and if there is any of these three colors present, rectangular boxes of respective colors (Red for Red, Blue for Blue, and Green for Green) will be displayed around the objects with those colors, along with text indicating the name of the color on top of the object. This system can be helpful in color recognition and robotics applications. Similar color detection systems are also used in driverless cars to detect traffic lights and vehicle taillights, and make decisions about stopping, starting, or continuing driving. It can also have many applications in industries for tasks such as picking and placing differently colored objects using robotic arms. OpenCV is a computer vision library, and in this project, OpenCV is used with Python.

Real-time multiple color detection refers to the process of identifying and extracting multiple colors from a live or continuous video stream or image feed. This technology can have various uses across different industries and applications. Some of the common uses of real-time multiple color detection include:

1. **Computer Vision and Image Processing**: Real-time multiple color detection can be used in computer vision and image processing applications to detect and track objects based on their color. This can be useful in areas such as robotics, autonomous vehicles, surveillance, and object recognition systems.
2. **Industrial Automation**: Real-time multiple color detection can be employed in industrial automation for tasks such as quality control, sorting, and assembly line operations. For example, in a manufacturing setting, it can be used to detect and sort products based on their color or to identify defects in products by analyzing color variations.
3. **Medical Imaging**: In medical imaging, real-time multiple color detection can be used for tasks such as identifying and tracking specific anatomical structures or markers,

detecting tumors or lesions, or monitoring physiological changes in real-time. This can be valuable in areas such as radiology, surgery, and telemedicine.

4. **Augmented Reality and Virtual Reality**: Real-time multiple color detection can be used in augmented reality (AR) and virtual reality (VR) applications to detect and track real-world objects or markers, and overlay virtual content or effects on top of them. This can enhance the immersive experience in AR/VR applications, such as gaming, training simulations, and virtual tours.
5. **Art and Design**: Real-time multiple color detection can be utilized in art and design applications for tasks such as color sampling, color matching, and palette generation. It can be used by artists, designers, and architects to capture colors from the real world and use them in their creative projects.
6. **Human-Computer Interaction**: Real-time multiple color detection can be used in human-computer interaction (HCI) applications, such as gesture recognition or touchless interfaces, where users can interact with devices or systems using color-based cues or gestures.
7. **Education and Research**: Real-time multiple color detection can be used in educational and research settings for tasks such as data collection, analysis, and visualization. It can be used in fields such as psychology, sociology, and environmental science to study and analyze color-related phenomena in real-time.

These are just a few examples of the many potential uses of real-time multiple color detection. The technology has broad applications across various industries and fields, and its versatility makes it a valuable tool in a wide range of applications where color information is critical.

Code and Explanation:

1. The code uses the OpenCV library for image processing and computer vision tasks.

2. It captures video frames from the default webcam (index 0) using the cv2.VideoCapture() function and stores it in the webcam object.

3. Inside a while loop that runs indefinitely, it reads the current video frame from the webcam using the webcam.read() function, and stores it in the imageFrame variable.

4. The imageFrame is then converted from BGR (RGB color space) to HSV (hue-saturation-value) color space using the cv2.cvtColor() function with the flag cv2.COLOR_BGR2HSV.

5. Three different color ranges (red, green, and blue) are defined using lower and upper limits in the HSV color space. These ranges are used to create masks for each color using the cv2.inRange() function, which filters out pixels that fall within the specified color range.

6. Morphological transformations such as dilation are applied to each mask using the cv2.dilate() function to reduce noise and fill gaps in the detected regions.

7. The cv2.bitwise_and() function is then used to perform a bitwise AND operation between the original imageFrame and each mask, which results in the extracted regions of each color.

8. Contours (boundary curves) are then detected in each color mask using the cv2.findContours() function.

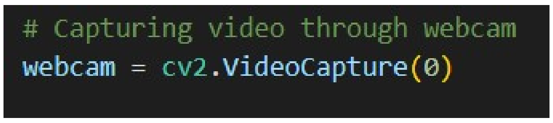
9. For each contour with an area greater than 300 (to filter out small noise), a bounding rectangle is drawn around the contour using the cv2.rectangle() function with the corresponding color (red, green, or blue) and a label indicating the color name is added using the cv2.putText() function.

10. The processed frame with color detections and labels is displayed in a window using the cv2.imshow() function.

11. The code waits for a key event and if the 'q' key is pressed, it releases the webcam object, closes all OpenCV windows, and terminates the program using cv2.waitKey() and cv2.destroyAllWindows() functions within the while loop.

Brief description on the used modules:

1. **numpy (imported as np)** - Used for numerical operations on arrays, specifically for creating and manipulating arrays of lower and upper bounds for color range.
2. **cv2 - OpenCV (Open Source Computer Vision Library)** module for Python, used for various computer vision tasks such as reading images and videos, image processing, and object detection.
3. **cv2.COLOR_BGR2HSV** - A constant used in the cv2.cvtColor() function to specify the color conversion from BGR to HSV color space.
4. **cv2.inRange()** - A function that creates a binary mask by thresholding the input image with lower and upper bounds of color range.
5. **cv2.dilate()** - A function that performs dilation operation on a binary image, used for morphological transformation to improve the detection of color regions.
6. **cv2.bitwise_and()** - A function that performs bitwise AND operation on two images, used to apply the binary mask to the original image to extract the regions of interest.
7. **cv2.findContours()** - A function that finds contours (i.e., the boundaries) of objects in a binary image.
8. **cv2.rectangle()** - A function that draws rectangles on an image, used to draw bounding boxes around the detected color regions.
9. **cv2.putText()** - A function that puts text on an image, used to label the detected color regions.
10. **cv2.FONT_HERSHEY_SIMPLEX** - A constant used in the cv2.putText() function to specify the font type for the text.
11. **cv2.imshow()** - A function that displays an image in a window, used to show the processed video frames in real-time.
12. **cv2.waitKey()** - A function that waits for a key event in OpenCV window, used to terminate the program when the 'esc' key is pressed.
13. **cv2.destroyAllWindows()** - A function that destroys all the OpenCV windows created by the program, used to clean up the resources before program termination.



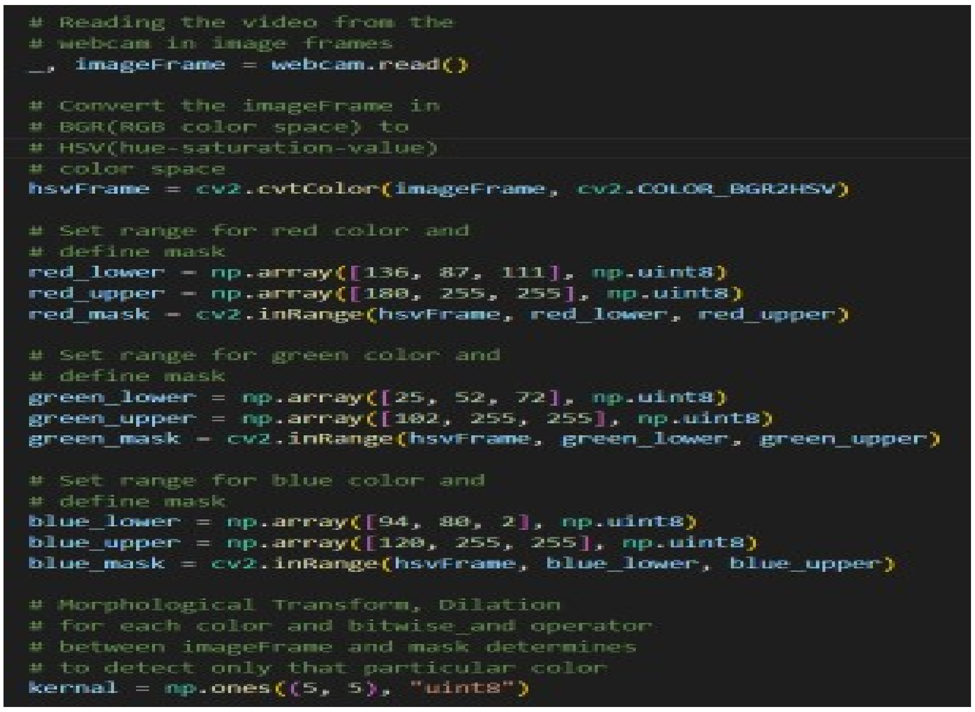
```python
# Capturing video through webcam
webcam = cv2.VideoCapture(0)
```

cv2: This is the Python module of OpenCV, a popular computer vision library used for image and video processing.

cv2.VideoCapture(0): This creates a VideoCapture object, which represents a connection to a video capturing device, such as a webcam. The parameter 0 specifies the index of the camera to be used. In this case, 0 refers to the default camera, which is typically the built-in camera of the computer.

webcam: This is a variable that holds the VideoCapture object, representing the connection to the webcam. It can be used to retrieve video frames from the camera using other functions provided by OpenCV.

So, the code is setting up a connection to the default camera (camera index 0) using OpenCV, and the webcam variable can be used to capture video frames from the camera for further processing, such as displaying the video stream, performing image processing tasks, or analyzing the video frames in real-time.



```python
# Reading the video from the
# webcam in image frames
_, imageFrame = webcam.read()

# Convert the imageFrame in
# BGR(RGB color space) to
# HSV(hue-saturation-value)
# color space
hsvFrame = cv2.cvtColor(imageFrame, cv2.COLOR_BGR2HSV)

# Set range for red color and
# define mask
red_lower = np.array([136, 87, 111], np.uint8)
red_upper = np.array([180, 255, 255], np.uint8)
red_mask = cv2.inRange(hsvFrame, red_lower, red_upper)

# Set range for green color and
# define mask
green_lower = np.array([25, 52, 72], np.uint8)
green_upper = np.array([102, 255, 255], np.uint8)
green_mask = cv2.inRange(hsvFrame, green_lower, green_upper)

# Set range for blue color and
# define mask
blue_lower = np.array([94, 80, 2], np.uint8)
blue_upper = np.array([120, 255, 255], np.uint8)
blue_mask = cv2.inRange(hsvFrame, blue_lower, blue_upper)

# Morphological Transform, Dilation
# for each color and bitwise_and operator
# between imageFrame and mask determines
# to detect only that particular color
kernal = np.ones((5, 5), "uint8")
```

This code is a Python implementation using OpenCV library for detecting red, green, and blue colors in a video stream from a webcam. Here's how it works:

The code reads frames from the webcam using webcam.read(), which returns a status flag (‘_’) and an image frame (imageFrame).

The cv2.cvtColor() function is used to convert the imageFrame from BGR (RGB color space) to HSV (hue-saturation-value) color space. HSV color space is often used for color-based object detection as it separates the color information from the brightness information, making it more robust to lighting changes.

Three color ranges (lower and upper bounds) are defined for red, green, and blue colors in HSV space using np.array(). These ranges define the color thresholds for detecting the respective colors.

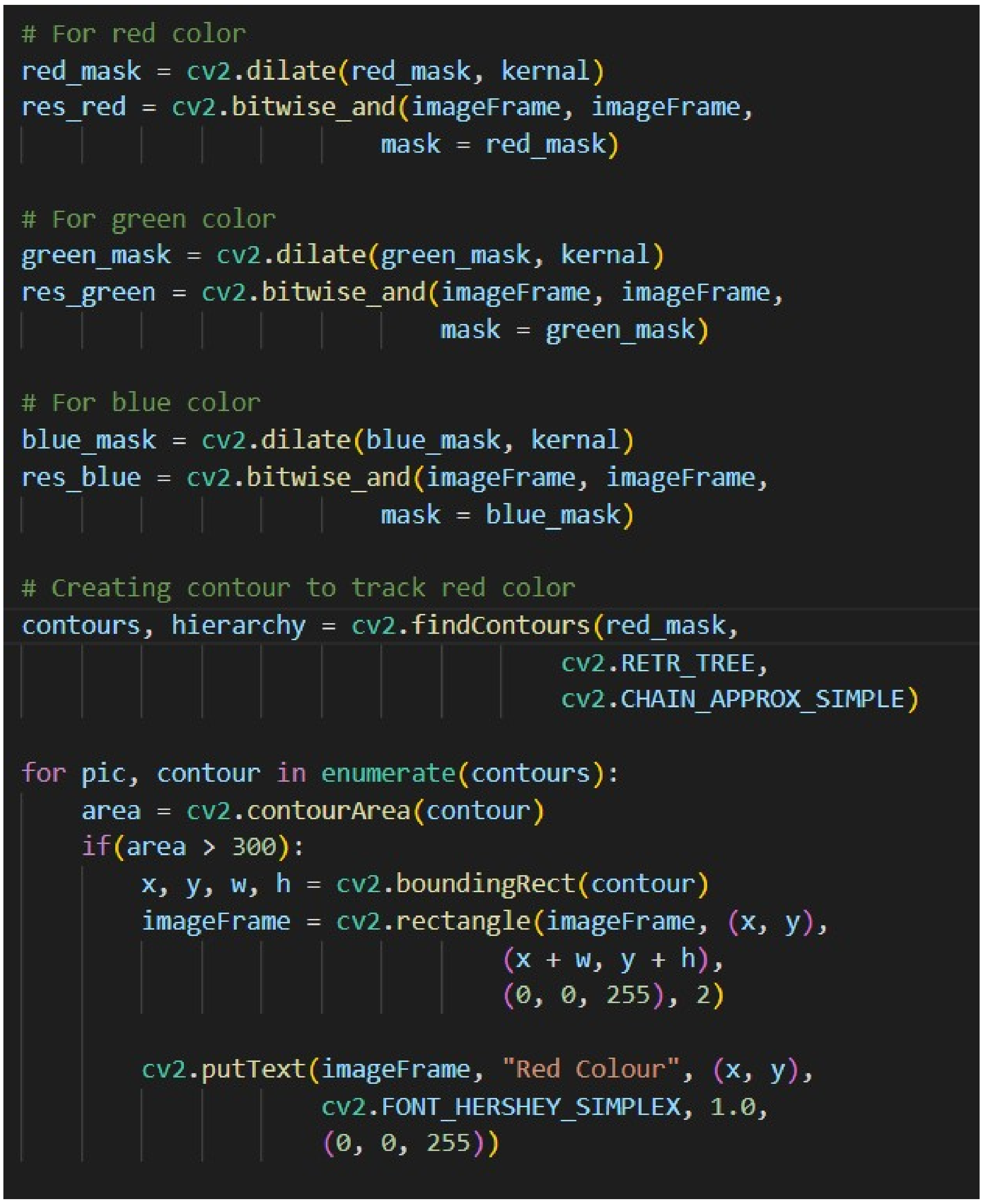
cv2.inRange() function is used to create binary masks for each color by thresholding the HSV frame using the defined color ranges. The resulting masks (red_mask, green_mask, and blue_mask) will have white pixels where the colors fall within the specified ranges, and black pixels elsewhere.

np.ones() function is used to create a 5x5 kernel (kernal) of type "uint8", which is used for morphological transformations in the next step.

Morphological transformations like dilation and erosion are commonly used for noise reduction and filling gaps in binary images. These operations are applied on each color mask using cv2.dilate() function with the kernal as the argument. This helps in improving the accuracy of color detection.

Finally, cv2.bitwise_and() function is used to perform a bitwise AND operation between the imageFrame and each color mask (red_mask, green_mask, and blue_mask) to obtain the regions in the frame where the respective colors are detected. These regions will appear in color, while the rest of the frame will be black.

Overall, this code segment captures video frames from a webcam, converts them to HSV color space, creates binary masks for red, green, and blue colors, performs morphological transformations, and then applies bitwise AND operation to detect and highlight the regions of the respective colors in the video frames.



```python
# For red color
red_mask = cv2.dilate(red_mask, kernal)
res_red = cv2.bitwise_and(imageFrame, imageFrame,
            mask = red_mask)

# For green color
green_mask = cv2.dilate(green_mask, kernal)
res_green = cv2.bitwise_and(imageFrame, imageFrame,
              mask = green_mask)

# For blue color
blue_mask = cv2.dilate(blue_mask, kernal)
res_blue = cv2.bitwise_and(imageFrame, imageFrame,
            mask = blue_mask)

# Creating contour to track red color
contours, hierarchy = cv2.findContours(red_mask,
                  cv2.RETR_TREE,
                  cv2.CHAIN_APPROX_SIMPLE)

for pic, contour in enumerate(contours):
  area = cv2.contourArea(contour)
  if(area > 300):
    x, y, w, h = cv2.boundingRect(contour)
    imageFrame = cv2.rectangle(imageFrame, (x, y),
                (x + w, y + h),
                (0, 0, 255), 2)

    cv2.putText(imageFrame, "Red Colour", (x, y),
          cv2.FONT_HERSHEY_SIMPLEX, 1.0,
          (0, 0, 255))
```

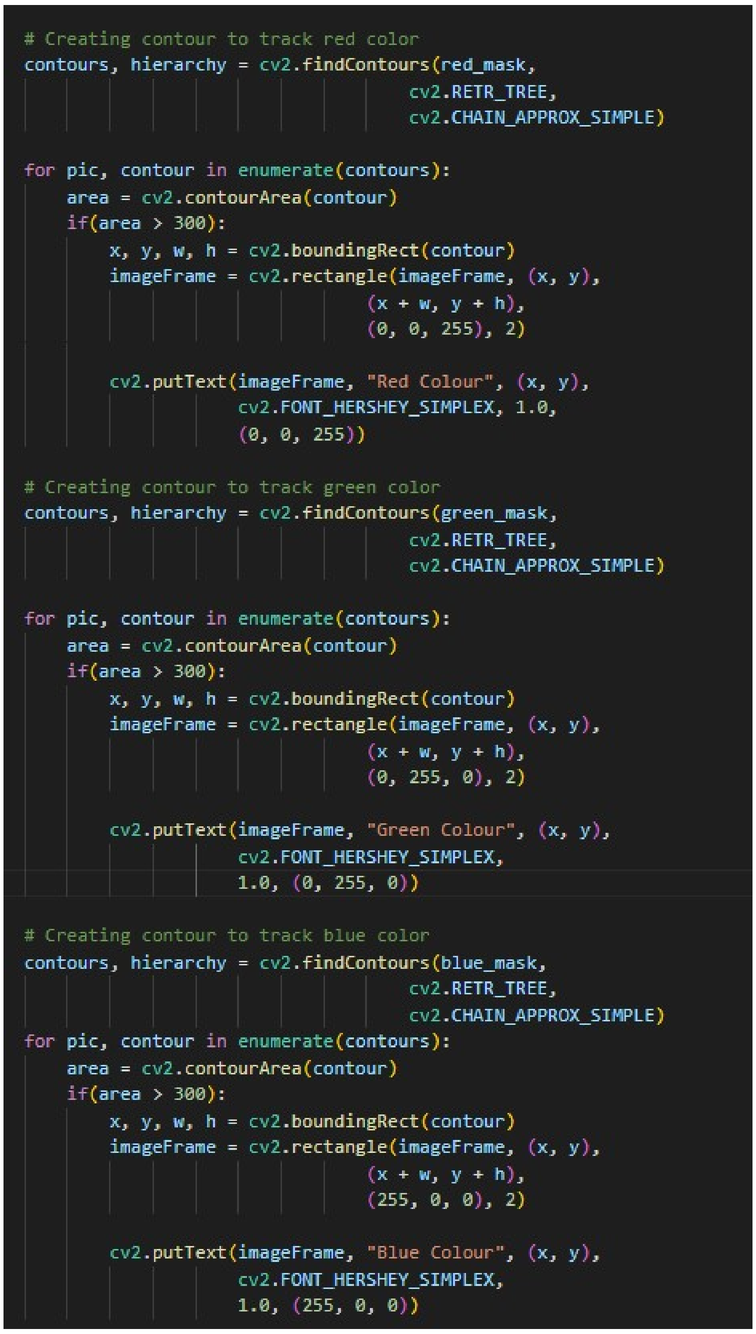
This part of the code is responsible for processing the color masks obtained from the previous step (dilating the masks with a kernel) and using them to detect and draw bounding boxes around regions of red, green, and blue colons in the imageFrame video frame.

1. For the red color: cv2.dilate() function is used to perform dilation on the red_mask using the kernal created earlier. Dilation is a morphological operation that helps to fill gaps and expand the regions of the binary mask, which can improve the accuracy of color detection.
   cv2.bitwise_and() function is used to perform a bitwise AND operation between the imageFrame and the dilated red_mask, resulting in the res_red image. This operation effectively masks out all regions in the imageFrame that do not correspond to the red color, leaving only the regions where the red color is detected.
2. Similar operations are performed for the green color:
   cv2.dilate() function is used to perform dilation on the green_mask using the kernal.

cv2.bitwise_and() function is used to perform a bitwise AND operation between the imageFrame and the dilated green_mask, resulting in the res_green image.

3. Similar operations are performed for the blue color:
   cv2.dilate() function is used to perform dilation on the blue_mask using the kernal.
   cv2.bitwise_and() function is used to perform a bitwise AND operation between the imageFrame and the dilated blue_mask, resulting in the res_blue image.
4. cv2.findContours() function is used to find contours in the red_mask. Contours are the boundaries of connected regions in a binary image.
5. A loop is used to iterate over each contour found in the red_mask:
   cv2.contourArea() function is used to calculate the area of the contour.
   If the area of the contour is greater than 300 (an arbitrary threshold), it is considered as a valid contour representing a region of red color.
   cv2.boundingRect() function is used to calculate the bounding rectangle around the contour, which represents the position and size of the detected region of red color.
   cv2.rectangle() and cv2.putText() functions are used to draw a bounding box and put a text label ("Red Colour") on the imageFrame around the detected region of red color, respectively.

   Similar operations can be performed for green and blue colors using res_green, green_mask, res_blue, and blue_mask to detect and draw bounding boxes around regions of green and blue colors in the imageFrame video frame, respectively.



```python
# Creating contour to track red color
contours, hierarchy = cv2.findContours(red_mask,
                                       cv2.RETR_TREE,
                                       cv2.CHAIN_APPROX_SIMPLE)

for pic, contour in enumerate(contours):
    area = cv2.contourArea(contour)
    if(area > 300):
        x, y, w, h = cv2.boundingRect(contour)
        imageFrame = cv2.rectangle(imageFrame, (x, y),
                                   (x + w, y + h),
                                   (0, 0, 255), 2)

        cv2.putText(imageFrame, "Red Colour", (x, y),
                    cv2.FONT_HERSHEY_SIMPLEX, 1.0,
                    (0, 0, 255))

# Creating contour to track green color
contours, hierarchy = cv2.findContours(green_mask,
                                       cv2.RETR_TREE,
                                       cv2.CHAIN_APPROX_SIMPLE)

for pic, contour in enumerate(contours):
    area = cv2.contourArea(contour)
    if(area > 300):
        x, y, w, h = cv2.boundingRect(contour)
        imageFrame = cv2.rectangle(imageFrame, (x, y),
                                   (x + w, y + h),
                                   (0, 255, 0), 2)

        cv2.putText(imageFrame, "Green Colour", (x, y),
                    cv2.FONT_HERSHEY_SIMPLEX,
                    1.0, (0, 255, 0))

# Creating contour to track blue color
contours, hierarchy = cv2.findContours(blue_mask,
                                       cv2.RETR_TREE,
                                       cv2.CHAIN_APPROX_SIMPLE)
for pic, contour in enumerate(contours):
    area = cv2.contourArea(contour)
    if(area > 300):
        x, y, w, h = cv2.boundingRect(contour)
        imageFrame = cv2.rectangle(imageFrame, (x, y),
                                   (x + w, y + h),
                                   (255, 0, 0), 2)

        cv2.putText(imageFrame, "Blue Colour", (x, y),
                    cv2.FONT_HERSHEY_SIMPLEX,
                    1.0, (255, 0, 0))
```

This code segment is used to create contours around objects of specific colors (red, green, and blue) in an image, and draw rectangles around those objects with corresponding color labels using OpenCV library in Python.

For each color (red, green, blue), a mask is created using the respective color filter, such as red_mask, green_mask, and blue_mask, which are assumed to be binary images with white pixels representing the presence of the corresponding color and black pixels representing the absence of that color in the original image.

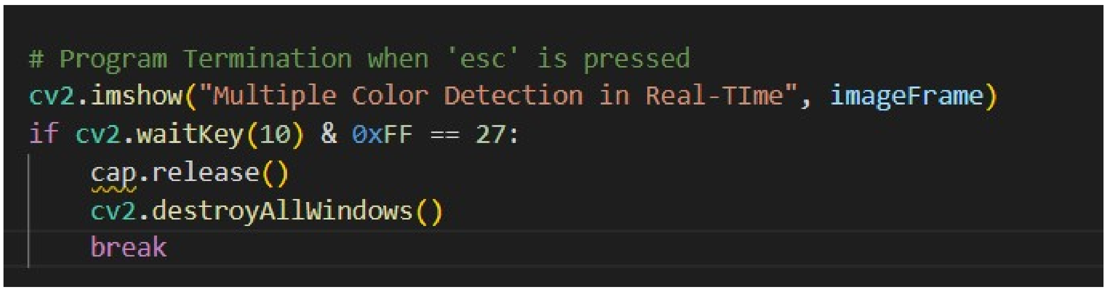
The cv2.findContours() function is then called with the mask image as input, along with the retrieval mode (cv2.RETR_TREE) and contour approximation method (cv2.CHAIN_APPROX_SIMPLE). This function finds contours (i.e., boundaries) of connected regions in the binary mask image.

The contours are stored in the "contours" variable, and the hierarchy (relationship between contours, if any) is stored in the "hierarchy" variable.

A loop iterates over each contour found in the mask image using the "contours" variable. For each contour, the area is calculated using cv2.contourArea() function, which gives the area of the contour region.

If the area of the contour is greater than 300 pixels (indicating a sufficiently large object), a bounding rectangle is drawn around the contour using cv2.rectangle() function, and a text label with the corresponding color name (e.g., "Red Colour", "Green Colour", "Blue Colour") is put on the image using cv2.putText() function.

The process is repeated for each color, so that contours are drawn and labels are put for objects of red, green, and blue colors separately, using their respective masks and corresponding color codes for rectangles and labels.



```
# Program Termination when 'esc' is pressed
cv2.imshow("Multiple Color Detection in Real-TIme", imageFrame)
if cv2.waitKey(10) & 0xFF == 27:
    cap.release()
    cv2.destroyAllWindows()
    break
```

This code segment is used to terminate the program when the 'esc' key is pressed by the user while an image frame is being displayed using OpenCV library in Python.

The cv2.imshow() function is called to display the image frame with a window title "Multiple Color Detection in Real-Time". The image frame is stored in the variable "imageFrame".

The cv2.waitKey(10) function is called to wait for a key event for a maximum of 10 milliseconds. The returned value is then bitwise ANDed with 0xFF (which is 11111111 in binary), which is a mask to extract the least significant 8 bits of the returned value.

The extracted value is then compared with 27, which represents the ASCII value of the 'esc' key.

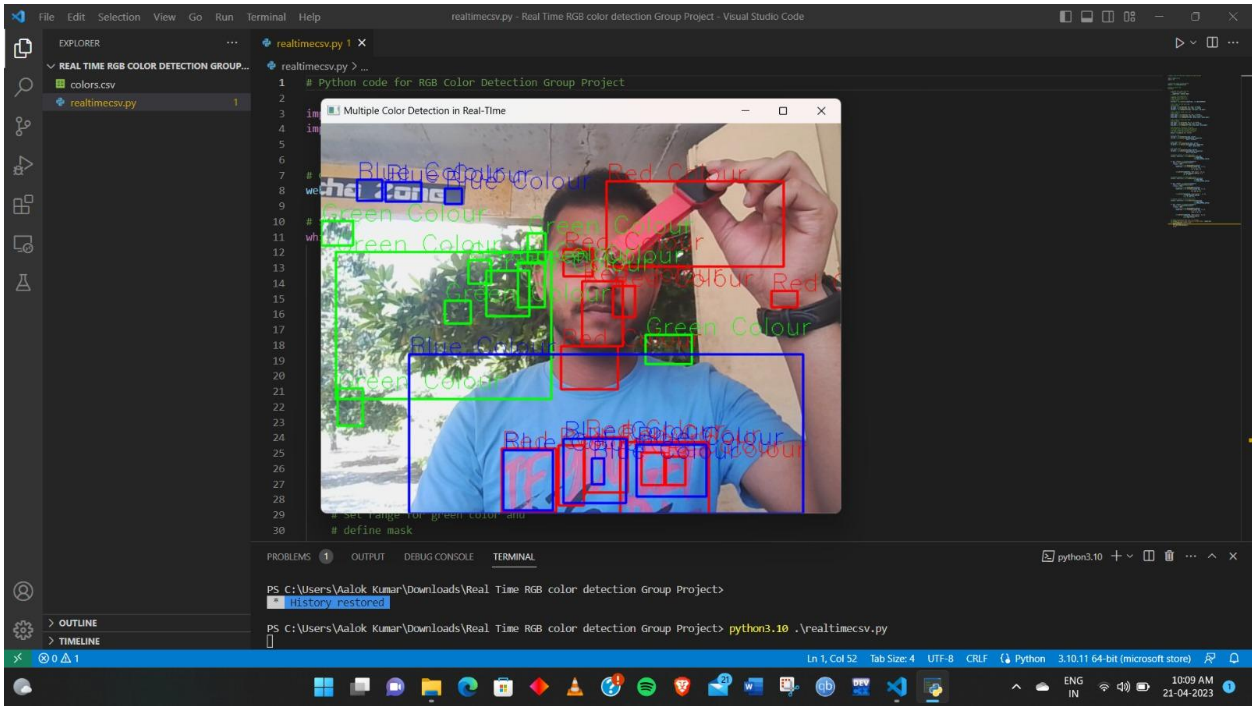
If the extracted value is equal to 27 (indicating that the 'esc' key has been pressed), the code inside the if statement is executed.

The cap.release() function is called to release any resources (such as camera) that were acquired by the program using the cv2.VideoCapture() function.

The cv2.destroyAllWindows() function is called to close all the OpenCV windows that were opened during the program execution.

The "break" statement is used to exit the loop in which this code segment is being executed, effectively terminating the program.

So, when the 'esc' key is pressed, the program releases any acquired resources, closes OpenCV windows, and terminates the program execution.



```
# Python code for RGB Color Detection Group Project
```

```
# define mask
```

```
PS C:\Users\Aalok Kumar\Downloads\Real Time RGB color detection Group Project>
 *  History restored

PS C:\Users\Aalok Kumar\Downloads\Real Time RGB color detection Group Project> python3.10 .\realtimecsv.py
```

In conclusion, the RGB Color Detection Group Project implemented in Python allows the real-time detection and tracking of red, green, and blue colors using the webcam of the device. The program first converts the frames captured from the webcam from the BGR color space to the HSV color space, and then applies a range of color thresholds to create masks for each of the three colors. It then uses morphological transformations and bitwise operators to filter out the pixels in the image frame that do not correspond to the desired color range. Finally, it identifies the contours of each color and draws a bounding box around them. The program terminates when the 'esc' key is pressed. This code can be used for various computer vision applications such as object detection, recognition, and tracking.