Универзитет „Св. Кирил и Методиј“ во Скопје
ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

-SEMINAR PAPER-

in course

DIGITAL IMAGE PROCESSING

Research topic

COLOUR SPACES DEMO APPLICATION
-REAL TIME INTERACTION WITH COLOUR SPACES-

Mentor

Ivica Dimitrovski Ph.D.

Student

Blerona Mulladauti, ID. 221541

Skopje, September 2024

Table of Contents

Introduction

Objective of the Demo Application

The Demo Application aims to provide an engaging, interactive visualization and manipulation of colour spaces in digital images. Instead of using a static image, the demo uses real-time conversion, enabling users to switch between colour spaces and adjust channel values dynamically, using the live feed from a webcam. This approach allows users to see the immediate effects of adjusting individual channels within various colour spaces.

The Demo App focuses on three widely used colour spaces: RGB (Red, Green, Blue), HSV (Hue, Saturation, Value), and LAB (Lightness, A, B). These colour spaces, among the many that exist, are fundamental in fields such as image editing, graphic design, and computer vision. To achieve this objective, the application leverages algorithms provided by OpenCV, a powerful open-source computer vision library. It captures a live video feed from the device's webcam and applies real-time adjustments by altering the channels of the selected colour space.

This hands-on experience enhances understanding of the differences between these three colour spaces. The demo serves as a bridge, connecting theoretical knowledge of colour spaces with practical visualization, thereby demonstrating their practical use in digital imaging.

Colour spaces are fundamental in digital imaging as they provide the necessary framework to represent, manipulate, and reproduce colours accurately across various devices and platforms. Choosing the right colour space can significantly impact the quality and efficiency of image processing tasks, ensuring colours are consistent and visually appealing to the human eye.

In the following pages we will look briefly at the theory basics in order to understand how the demo app code works and why it works that way.

Colour Space briefly in Theory

A colour space describes a specific, measurable, and fixed range of possible colours and luminance values. Its most basic practical function is to describe the capabilities of a capture or display device to reproduce colour information. When trying to reproduce colour on another device, colour spaces can show whether shadow/highlight detail and colour saturation can be retained, and by how much either will be compromised.

Colour models vs colour space

A "colour model" is an abstract mathematical model describing the way colours can be represented as tuples of numbers (e.g. triples in RGB or quadruples in CMYK); Unlike colour spaces, colour models aren't about expressing different ranges of colour and luminance, but about differently expressing the same range of colour and luminance. We're generally most familiar with the RGB colour model, wherein we describe a given colour in terms of its proportions of red, green, and blue. Other colour models simply plot and target colours in alternative ways — for example, HSV describes a colour in terms of its hue, saturation, and value (roughly equivalent to brightness). As with colour spaces, we can formulaically transform from one colour model to another — the difference is that this won't yield a visual change. Since "colour space" identifies a particular combination of the colour model and the mapping function, the word is often used informally to identify a colour model, this usage is incorrect in a strict sense. For example, although several specific colour spaces are based on the RGB colour model, there is no such thing as the singular RGB colour space.

Gamut

A colour gamut is the defining a range of chromaticity—essentially a set of possible hues and their respective maximum saturations.

Conversion

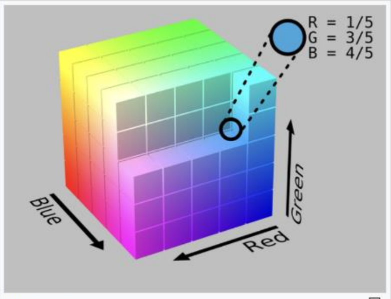
Colour space conversion is the translation of the representation of a colour from one basis to another. This typically occurs in the context of converting an image that is represented in one colour space to another colour space, the goal being to make the translated image look as similar as possible to the original.

RGB (Red, Green, Blue)

The RGB colour space represents images as an *m*-by-*n*-by-3 numeric array whose elements specify the intensity values of the red, green, and blue colour channels. The range of numeric values depends on the data type of the image.

- For single or double arrays, RGB values range from [0, 1].
- For uint8 arrays, RGB values range from [0, 255].
- For uint16 arrays, RGB values range from [0, 65535].

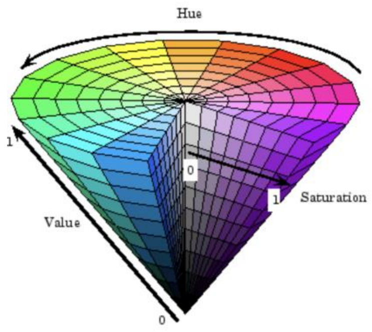
The normal human eye contains three types of colour-sensitive cone cells. Each cell is responsive to light of either long, medium, or short wavelengths, which we generally categorize as red, green, and blue. Taken together, the responses of these cone cells are called the Tristimulus values, and the combination of their responses is processed into the psychological effect of colour vision. RGB use in colour space definitions employ primaries (and often a white point) based on the RGB colour model, to map to real world colour. Applying Grassmann's law of light additivity, the range of colours that can be produced are those enclosed within the triangle on the chromaticity diagram defined using the primaries as vertices. The primary colours are usually mapped to xyY chromaticity coordinates, though the **u′,v′** coordinates from the UCS chromaticity diagram may be used. Both xyY and u′,v′ are derived from the CIE 1931 colour space, a device independent space also known as **XYZ** which covers the full gamut of human-perceptible colours visible to the CIE 2° standard observer.



RGB cube

HSV (Hue, Saturation, Value)

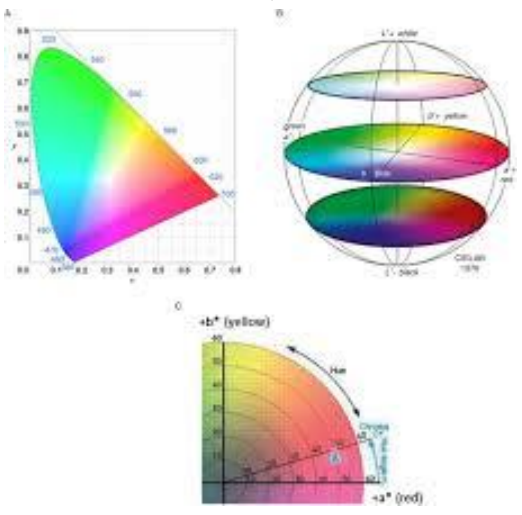
HSV corresponds better to how people experience colour than the RGB colour space does. For example, this colour space is often used by people who are selecting colours, such as paint or ink colour, from a colour wheel or palette.



- **Hue**, which corresponds to the colour’s position on a colour wheel. *H* is in the range [0, 1]. As *H* increases, colours transition from red to orange, yellow, green, cyan, blue, magenta, and finally back to red. Both 0 and 1 indicate red.
- **Saturation**, which is the amount of hue or departure from neutral. *S* is in the range [0, 1]. As *S* increases, colours vary from unsaturated (shades of gray) to fully saturated (no white component).
- **Value**, which is the maximum value among the red, green, and blue components of a specific colour. *V* is in the range [0, 1]. As *V* increases, the corresponding colours become increasingly brighter.

LAB (Lightness, A, B)

CIE 1976 L*a*b* is a device-independent colour spaces developed by the International Commission on Illumination, known by the acronym CIE. These colour spaces model colour according to the typical sensitivity of the three types of cone cells in the human eye.The L*a*b* colour space provides a more perceptually uniform colour space than the XYZ model. Colours in the L*a*b* colour space can exist outside the RGB *gamut* (the valid set of RGB colours). For example, when you convert the L*a*b* value [100, 100, 100] to the RGB colour space, the returned value is [1.7682, 0.5746, 0.1940], which is not a valid RGB colour



- **L- Luminance** or brightness of the image. Values are in the range [0, 100], where 0 specifies black and 100 specifies white. As L* increases, colours become brighter.
- **A -** Amount of red or green tones in the image. A large positive *a** value corresponds to red/magenta. A large negative *a** value corresponds to green. Although there is no single range for *a**, values commonly fall in the range [-100, 100] or [-128, 127).
- **B -** Amount of yellow or blue tones in the image. A large positive *b** value corresponds to yellow. A large negative *b** value corresponds to blue. Although there is no single range for *b**, values commonly fall in the range [-100, 100] or [-128, 127).

Errors in color space conversion

- Rounding Errors and Precision Loss. When converting between color spaces (e.g., RGB to HSV or LAB), the algorithms often involve mathematical operations that can introduce rounding errors due to the finite precision of digital representations. Small rounding errors can lead to slight variations in the converted color values, especially when repeatedly converting back and forth between spaces. This can be particularly noticeable in edge cases, such as very bright or very dark colors, where precision is already limited.

Now that we know the theory, we can continue with the analysis of the code behind the Demo Application.

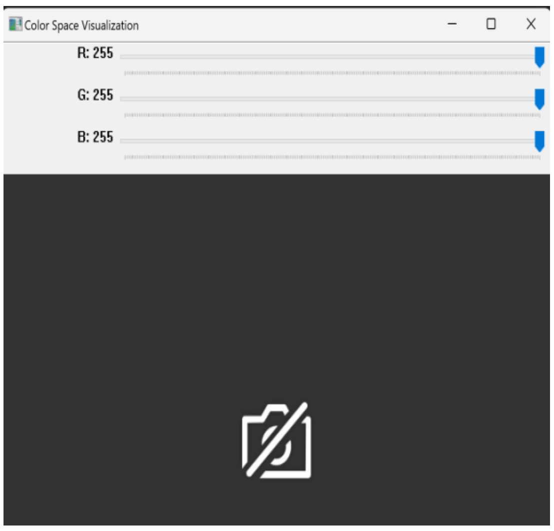
Demo Application

As previously mentioned, the Demo Application converts between 3 colour spaces (RGB, HSL, LAB) of the real time video feed captured through device’s web camera. Apart from switching between colour spaces, user can also adjust the channel values accordingly and keep switching to see the differences between the colours in each space.

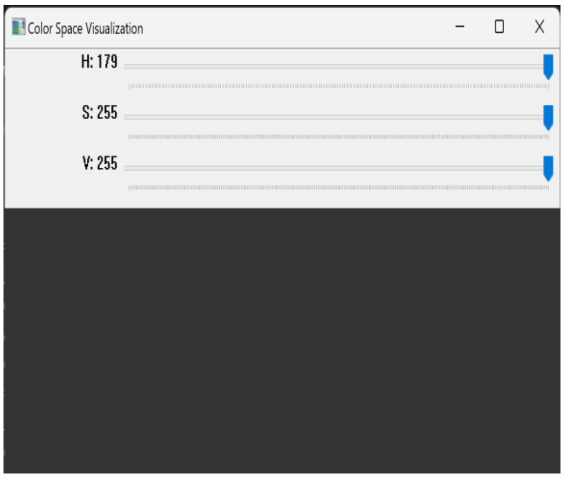
User Guide

The first window that appears, is the live video feed by default in RGB colour space, on top of the stream, there are three bars, in front of which you can see the initials of the channels for the colour space and the values they can obtain.

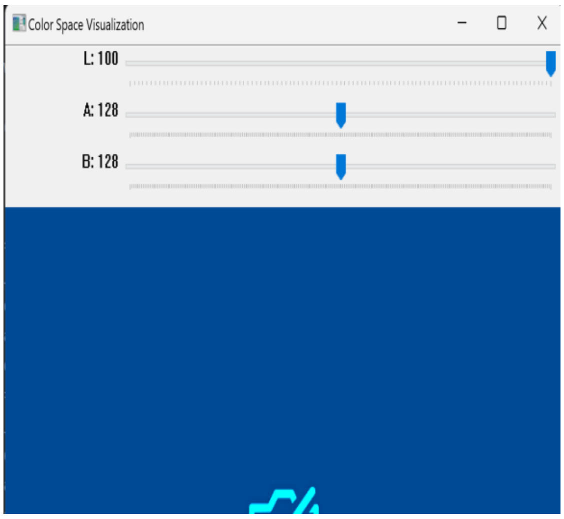
The following images show how the app interface looks on all 3 colour spaces (the webcam being turned off).



1.1 The RGB colour space



1.2 The HSV colour space



1.3 The LAB colour space

To switch to HSL colour space, user should click the ‘h’ key on the keyboard. To switch to LAB, click the ‘l’ key on the keyboard and to go back to RGB, the key ‘r’. Additionally, the app allows user to capture a screenshot of the feed by clicking the ‘c’ key, after which the image will be saved. Lastly, to exit the application, the user simply must click the ‘q’ key.

Code Analysis

Structure and organization

Code is written in python, and mainly organized in 3 functions:

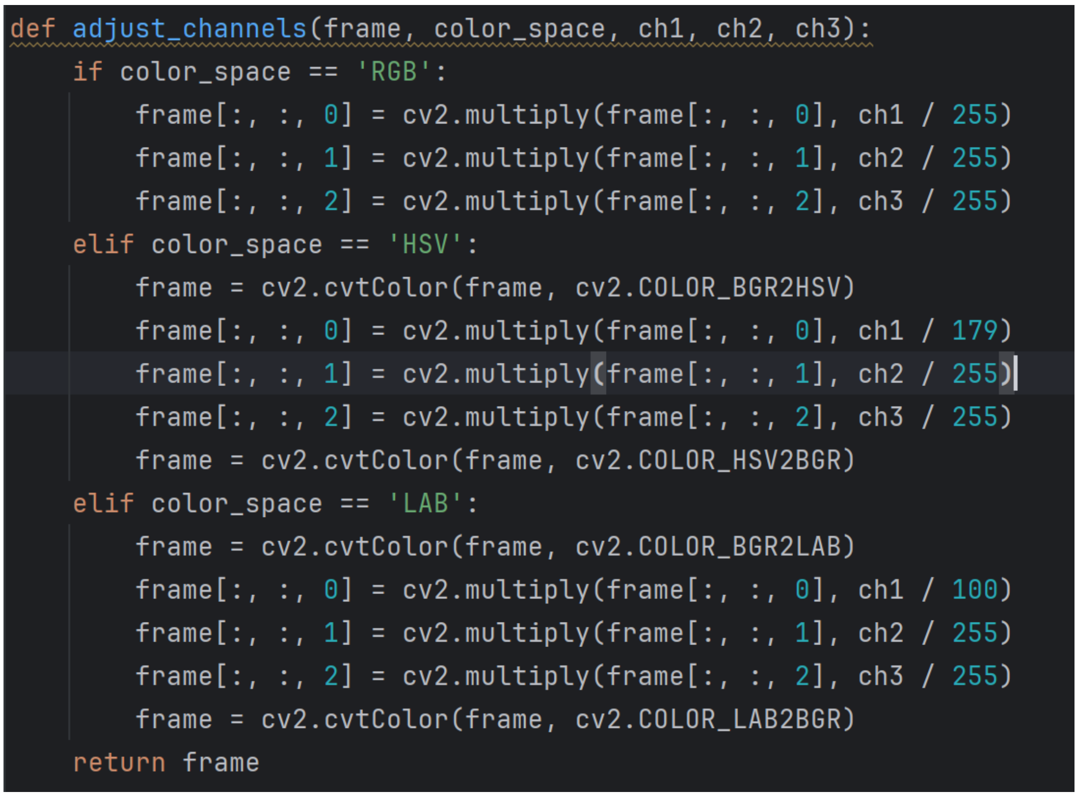
- **adjust_channels(frame,color_space,ch1,ch2,ch3)**
  adjusts the colour channel values based on the selected colour space, differentiating them with a simple if else.
- **Create_trackbars(colour_space)**
  Uses OpenCV library’s code to create trackbars for each channel (explained in detail further in the document).
- **Main**
  The main is the part that handles the webcam input, space selection and visualization.

Library Usage and Algorithm analysis

OpenCv library is open source and contains over 2500 algorithms. The Demo App code uses ready functions from the OpenCv library for achieving the result.

1. **Adjust_channels function**

The purpose of this function is to adjust the color channes values of the selected colour space (RGB, HSV or LAB) using OpenCv’s



```python
def adjust_channels(frame, color_space, ch1, ch2, ch3):
    if color_space == 'RGB':
        frame[:, :, 0] = cv2.multiply(frame[:, :, 0], ch1 / 255)
        frame[:, :, 1] = cv2.multiply(frame[:, :, 1], ch2 / 255)
        frame[:, :, 2] = cv2.multiply(frame[:, :, 2], ch3 / 255)
    elif color_space == 'HSV':
        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
        frame[:, :, 0] = cv2.multiply(frame[:, :, 0], ch1 / 179)
        frame[:, :, 1] = cv2.multiply(frame[:, :, 1], ch2 / 255)
        frame[:, :, 2] = cv2.multiply(frame[:, :, 2], ch3 / 255)
        frame = cv2.cvtColor(frame, cv2.COLOR_HSV2BGR)
    elif color_space == 'LAB':
        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2LAB)
        frame[:, :, 0] = cv2.multiply(frame[:, :, 0], ch1 / 100)
        frame[:, :, 1] = cv2.multiply(frame[:, :, 1], ch2 / 255)
        frame[:, :, 2] = cv2.multiply(frame[:, :, 2], ch3 / 255)
        frame = cv2.cvtColor(frame, cv2.COLOR_LAB2BGR)
    return frame
```

- **cv2.cvtColor()** method.

  frame = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV) //*from RGB to HSV*
  frame = cv2.cvtColor(frame, cv2.COLOR_HSV2BGR) //*from HSV to RGB*
  frame = cv2.cvtColor(frame, cv2.COLOR_BGR2LAB) //*from RGB to LAB*
  frame = cv2.cvtColor(frame, cv2.COLOR_LAB2BGR) //*from LAB to RGB*

  the method takes 2 parameters, one is the frame whose colour space will the converted and the second is what tells from which-to-which colour space it will change.

Conversion from LAB to RGB

The first step in converting LAB to RGB is to convert LAB back to the XYZ color space.

Given an LAB color with components L∗L^*L∗, a∗a^*a∗, and b∗b^*b∗:

$$Y = \frac{L^* + 16}{116} \qquad X = X_n \times f^{-1}(X)$$

$$X = \frac{a^*}{500} + Y \qquad Y = Y_n \times f^{-1}(Y)$$

$$Z = Y - \frac{b^*}{200} \qquad Z = Z_n \times f^{-1}(Z)$$

$$\begin{bmatrix} R' \\ G' \\ B' \end{bmatrix} = \begin{bmatrix} 3.2404542 & -1.5371385 & -0.4985314 \\ -0.9692660 & 1.8760108 & 0.0415560 \\ 0.0556434 & -0.2040259 & 1.0572252 \end{bmatrix} \times \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

Once you have the XYZ values, the next step is to convert them to linear RGB.

The conversion from XYZ to linear RGB is done using the inverse of the matrix transformation that was used to go from RGB to XYZ:

Finally, convert the linear RGB values to the standard RGB values (sRGB) by applying gamma correction:

For each channel (R, G, B):

- If the linear value is less than or equal to 0.0031308: RGB=12.92×Linear RGB
- Otherwise: RGB=1.055×(Linear RGB)$^{1/2.4}$−0.055

<u>Conversion from HSV to RGB</u>

What cv2.color_HSV2BGR does is firstly it normalized the HSV values. The hue HHH is usually given in degrees from 0 to 360 but scaled in OpenCV from 0 to 179. Normalize it to a range of 0 to 1. Saturation SSS is scaled from 0 to 255 in OpenCV, so normalize it to a range of 0 to 1. Value VVV is also scaled from 0 to 255 in OpenCV, so normalize it to a range of 0 to 1.

$$S' = \frac{S}{255} \qquad H' = \frac{H}{60} \qquad V' = \frac{V}{255}$$

Calculate Chroma, represents the intensity of the color: C=V′×S′C = V' \times S'C=V′×S′.
X is a secondary value used to determine the exact RGB components:

X=C×(1−|(H′mod 2)−1|)

m is used to adjust the resulting RGB values:

m=V′−Cm = V' - Cm=V′−C

$$R = (R' + m) \times 255$$
$$G = (G' + m) \times 255$$
$$B = (B' + m) \times 255$$

Depending on the value of the hue (H'), assign preliminary RGB values:

<u>Conversion from RGB to HSV</u>

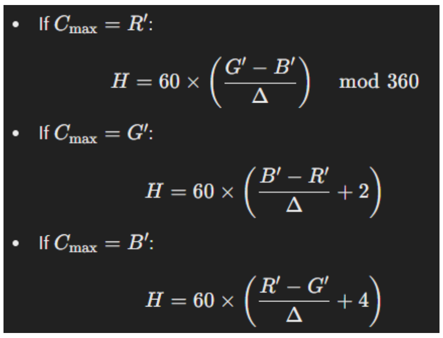
Normalize the RGB values by dividing each by 255 (if they are 8-bit values) to bring them to a range between 0 and 1:

$$R' = \frac{R}{255}, \quad G' = \frac{G}{255}, \quad B' = \frac{B}{255}$$

Determine the maximum and minimum values among the normalized R′, G′, and B′:

Cmax=max(R′,G′,B′) Cmin=min(R′,G′,B′) Compute the difference, also known as the chroma: Δ=Cmax−Cmin.

The hue depends on which of the RGB components is the maximum.



- If $C_{\max} = R'$:

$$H = 60 \times \left(\frac{G' - B'}{\Delta}\right) \mod 360$$

- If $C_{\max} = G'$:

$$H = 60 \times \left(\frac{B' - R'}{\Delta} + 2\right)$$

- If $C_{\max} = B'$:

$$H = 60 \times \left(\frac{R' - G'}{\Delta} + 4\right)$$

If Δ=0 (when all RGB values are the same), the color is grayscale, and the hue is set to 0 by convention.

Since OpenCV uses an 8-bit representation for HSV images, the hue is scaled to a range of 0 to 179: H=H/2.

The vice versa conversions are simply the inverse of what we just did until now. As you may notice there is no LAB to HSV conversion function. This is because OpenCv only does conversions through the RGB colour space, it cannot convert directly between the other colour spaces but rather through RGB.

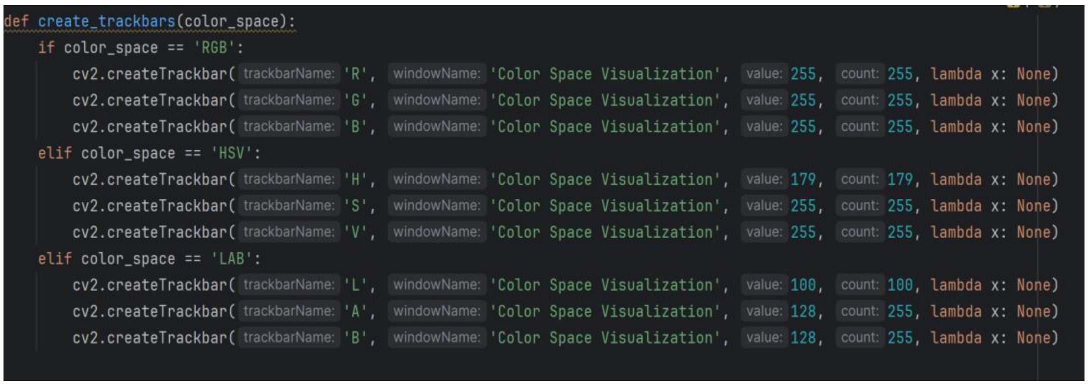
- **Cv2.multiply()**

frame[:, :, 0] = cv2.multiply(frame[:, :, 0], ch1 / 255)

frame[:,:,0] selects the first color channel of the image, in our case the video feed. The notation uses numpy slicing, : selects all rows, : all columns 0 selects first channel (Blue in OpenCv default BGR format).

cv2.multiply is an OpenCV function that performs per-element multiplication between two arrays, with additional features such as handling different data types and preventing overflow by scaling the output. Ch1 is an array that holds values used for normalization, by dividing with 255 in this case they normalized to range [0,1], effectively scaling the pixel intensities for that channel. The use of cv2.multiply ensures that each color channel or component is adjusted according to the specified scaling factors. This multiplication is done to adjust the contribution of each color channel or component in its respective color space.

2. **Create_trackbars(colour_space)**
   Purpose of this function in the code is to create the trackbars for adjusting the channels. A trackbar is a graphical UI element that allows users to adjust a value within a specified range by moving a slider.

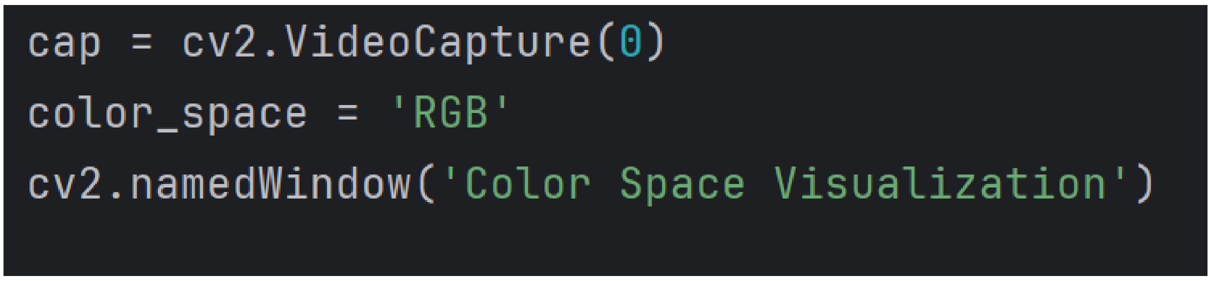


```python
def create_trackbars(color_space):
    if color_space == 'RGB':
        cv2.createTrackbar( trackbarName: 'R',  windowName: 'Color Space Visualization',  value: 255,  count: 255, lambda x: None)
        cv2.createTrackbar( trackbarName: 'G',  windowName: 'Color Space Visualization',  value: 255,  count: 255, lambda x: None)
        cv2.createTrackbar( trackbarName: 'B',  windowName: 'Color Space Visualization',  value: 255,  count: 255, lambda x: None)
    elif color_space == 'HSV':
        cv2.createTrackbar( trackbarName: 'H',  windowName: 'Color Space Visualization',  value: 179,  count: 179, lambda x: None)
        cv2.createTrackbar( trackbarName: 'S',  windowName: 'Color Space Visualization',  value: 255,  count: 255, lambda x: None)
        cv2.createTrackbar( trackbarName: 'V',  windowName: 'Color Space Visualization',  value: 255,  count: 255, lambda x: None)
    elif color_space == 'LAB':
        cv2.createTrackbar( trackbarName: 'L',  windowName: 'Color Space Visualization',  value: 100,  count: 100, lambda x: None)
        cv2.createTrackbar( trackbarName: 'A',  windowName: 'Color Space Visualization',  value: 128,  count: 255, lambda x: None)
        cv2.createTrackbar( trackbarName: 'B',  windowName: 'Color Space Visualization',  value: 128,  count: 255, lambda x: None)
```

Syntax of cv2.createTrackbar
The openCv cv2.createTrackbar() is used, for each space accordingly.
cv2.createTrackbar('R', 'Color Space Visualization', 255, 255, lambda x: None)
the method takes the following parameters, the trackbar name, in this case R for the R channel in RGB, window name, the initial position of the slider, the maximum value (0 to value). The last one is a callback function called when the trackbar value changes, it receives the new position as its argument. In this case because no action is performed when trackbar changes, lambda x:None is a dummy function used.

3. Main

The main function continuously captures frames from the webcam, processes them according to the selected color space, and displays the adjusted frames in real time.

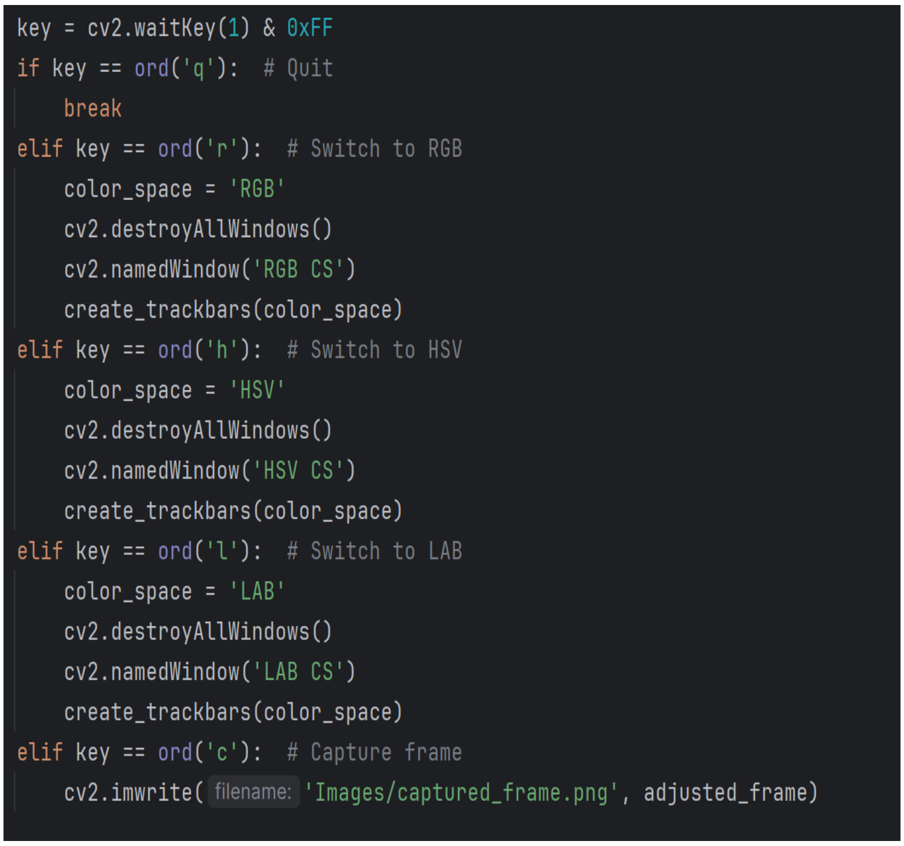


```python
cap = cv2.VideoCapture(0)
color_space = 'RGB'
cv2.namedWindow('Color Space Visualization')
```

- **cv2.VideoCapture(0)** is used to open the webcam. The cv2.VideoCapture() method returns a VideoCapture object, which represents the input video stream. This object can be used to read frames from the video input source, set properties of the video stream, and release the resources used by the stream.

After initializing the cap, a loop Is created to read frames, adjust color channels, handle user input to switch or capture frames and thereby displaying the result. All the previous are done using the following OpenCv methods:

- cv2.getTrackbarPost(): Retrieves the current position of a trackbar.
- cv2.imshow(): Displays an image in a window.
- cv2.waitKey(): Waits for a key press.
- cv2.destroyAllWindows(): Closes all OpenCV windows.
- cv2.imwrite(): Saves the current frame as an image file.



```
key = cv2.waitKey(1) & 0xFF
if key == ord('q'):  # Quit
    break
elif key == ord('r'):  # Switch to RGB
    color_space = 'RGB'
    cv2.destroyAllWindows()
    cv2.namedWindow('RGB CS')
    create_trackbars(color_space)
elif key == ord('h'):  # Switch to HSV
    color_space = 'HSV'
    cv2.destroyAllWindows()
    cv2.namedWindow('HSV CS')
    create_trackbars(color_space)
elif key == ord('l'):  # Switch to LAB
    color_space = 'LAB'
    cv2.destroyAllWindows()
    cv2.namedWindow('LAB CS')
    create_trackbars(color_space)
elif key == ord('c'):  # Capture frame
    cv2.imwrite( filename: 'Images/captured_frame.png', adjusted_frame)
```

This part of the code handles the user inputs and the actions the app will take based on that input. After each switch of the colour space it destroys the current window and creates a new for the selected colour space, altogether with the according trackbars.

Performance and Error handling

Consider using lower-resolution frames for faster processing or implementing parallel processing for multiple inputs or higher frame rates to further optimize performance. Conversion: The use of cv2.cvtColor() is optimized for fast color space conversions, making it suitable for real-time applications.

Multiplication: cv2.multiply() is used to adjust the channel values efficiently, leveraging OpenCV's optimizations for matrix operations.

Trackbars are created dynamically each time the color space changes, which allows for efficient user interaction.

Real-time processing is achieved by leveraging OpenCV's optimized functions.

Efficient use of loops and conditional statements for handling user inputs and dynamically changing the color space and visualization.

OpenCV functions are highly optimized for performance, particularly for operations involving image processing and real-time applications.

Error Handling:

- **Webcam Input:** Checks if the webcam frame is successfully captured with if not ret.
- **Color Space Switching:** Properly destroys and recreates OpenCV windows when switching color spaces to ensure the trackbars are updated correctly.

Demo testing and results

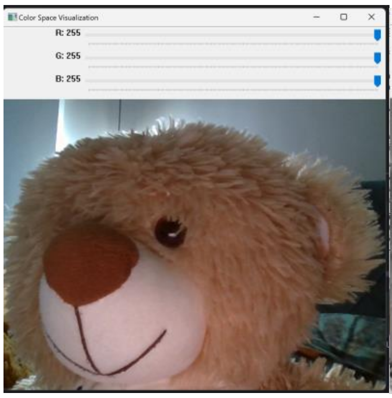
The application was tested multiple times by interactively switching between different colour spaces and adjusting channel values to observe the changes in real time. Because of how powerful OpenCv and its algorithms are there were no errors in the results, only minor UI changes were made to the code to make the app appearance look more comprehensive and eye pleasing.

As for the visual results, screenshots or captured frames are provided to demonstrate how space switching, or channel adjustments affect the frame appearance.

Strengths and limitations of demo app

This colour spaces switching Demo Application offers real time interaction and visualization of color space adjustments. It also displays an efficient use of OpenCV's capabilities for fast image processing.

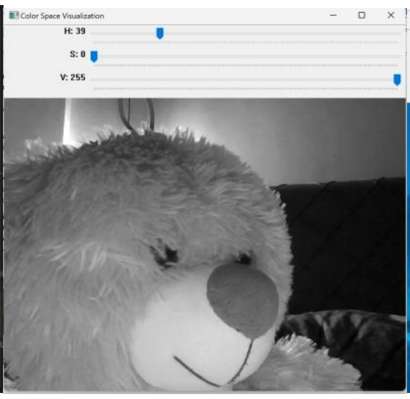
The application relies on the availability of a webcam and a graphical user interface, which may limit its use in non-interactive environments. Future improvements could include support for more color spaces and additional user-friendly features.


Color Space Visualization

R: 255

G: 255

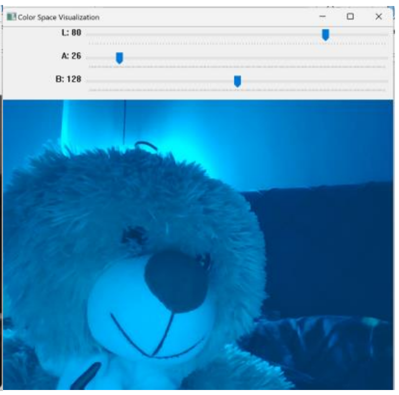
B: 255


Color Space Visualization

H: 39

S: 0

V: 255


Color Space Visualization

L: 80

A: 26

B: 128

References

https://www.mathworks.com/help/images/understanding-color-spaces-and-color-space-conversion.html

https://sourcegraph.com/blog/strange-loop/strange-loop-2019-rgb-to-xyz-the-science-and-history-of-color

https://blog.frame.io/2020/02/03/color-spaces-101/

https://courses.finki.ukim.mk/course/view.php?id=2469

https://en.wikipedia.org/wiki/CIELAB_color_space

https://en.wikipedia.org/wiki/RGB_color_spaces

https://www.geeksforgeeks.org/python-play-a-video-using-opencv/

https://roboflow.com/use-opencv/convert-color-space-with-cv2-cvtcolor#:~:text=The%20cv2.,conversion%20you%20want%20to%20apply.

https://roboflow.com/use-opencv/convert-color-space-with-cv2-cvtcolor#:~:text=The%20cv2.,conversion%20you%20want%20to%20apply.

https://opencv.org/

https://github.com/davidapr4/HSV-Color-Detection-and-Tracking-Python-OpenCV/blob/main/