CONTENTS

Introduction :

• Color is that portion of the visible spectrum of light that is reflected back from a surface. The amount of light that a surface reflects or absorbs determines its color. In particular, black surfaces absorb all light, while white surfaces reflect all light.

• We can also say that Color is a 3-D array in which the 3 dimensional are Red, Green and Blue.

• RGB (red, green, and blue) refers to a system for representing the colors to be used on a computer display. Red, green, and blue can be combined in various proportions to obtain any color in the visible spectrum. Levels of R, G, and B can each range from 0 to 100 percent of full intensity.

• Color detection is the process of detecting the name of any color. For humans this is an extremely easy task but for computers, it is not straightforward. Human eyes and brains work together to translate light into color. Light receptors that are present in our eyes transmit the signal to the brain. Our brain then recognizes the color. Since childhood, we have mapped certain lights with their color names. We will be using the somewhat same strategy to detect color names.

• For a color to be represented it is done by three values, which corresponds to the value of Red, Green and Blue.

• These RGB values can be represented in a number which is in between the values of 0 and 255.

• Pure red color refers to having R.G.B values of 255,0,0 where the green and blue values are zero.

• If we try to use all the R.G.B values of each and every color then we will have approximately 1.6 million combinations to code, which is next to impossible.

• So we use a CSV file to read a certain number of numbers and their hexadecimal values along with the R.G.B value.

For a robot to visualize the environment, along with the object detection, detection of its color in real-time is also very important. This is an

implementation of detecting multiple colors (here, only red, green and blue colors have been considered) in real-time using Python programming language.

Aim of the project:

In this Python project, we will learn about colors and how we can extract color RGB values and the color name of a pixel.

We will be implementing the detection of multiple colors (here, only red, green and blue colors have been considered) in real-time using Python programming language.

In this python project we will be giving a real time video as input using webcam and, in a particular photo, it will be able to detect and generate multiple color name for a particular image frame for that particular video.

Modules and Implementation:

Modules used:

There are couple of modules and libraries used in this program which are explained below with what task they performed to help execute the program.

OpenCV library:

• OpenCV library is designed for computer vision.

• OpenCV is a huge open-source library for computer vision, machine learning, and image processing.

• It can process images and videos to identify objects, faces, or even the handwriting of a human.

• In this program OpenCV is used to read the image, which we will specify with the path.

• Also OpenCV is will be used to find all the R.G.B values of the particular pixel that we double click on.

Numpy:

• NumPy is a Python library used for working with large, multi-dimensional arrays and matrices of numerical data. It provides tools for performing mathematical operations on these arrays and matrices, such as linear algebra operations, statistical functions, and more.

• NumPy is particularly useful for working with large arrays of data, as it allows you to perform operations on the entire array, rather than having to loop through each element individually. This makes it much more efficient than using Python's built-in lists or dictionaries for large data sets. NumPy also includes a number of functions for reading and writing data to and from files, as well as for performing operations such as sorting and searching.

• In addition to its utility for scientific computing, NumPy is also widely used as a foundation for other libraries in the Python ecosystem, such as Pandas (for data analysis) and scikit-learn (for machine learning). It is a powerful and essential tool for anyone working with numerical data in Python.

• In this project, numpy is being used to manipulate the image data and to perform calculations on the pixel values. The numpy library provides support for large, multi-dimensional arrays and matrices of numerical data, and it has a large collection of mathematical functions to operate on these arrays. It is a key library for scientific computing with Python and is widely used in machine learning and data analysis.

• In this project, numpy is being used to: Load the image with the cv2.imread() function, which returns the image data as a numpy array.Extract the pixel values at a specific location in the image using indexing, like this: b,g,r = img[y,x]. This returns the blue, green, and red channel values for the pixel at position (x,y) in the image as a numpy array.Perform calculations on the pixel values, such as converting them to integers with int() and subtracting them from each other. Overall, numpy is being used to manipulate and analyze the image data and to perform mathematical operations on it.

Steps to implement the color detection program:

Step 1: Capture video through webcam as input.

Step 2: Read the video stream in image frames.

Step 3: Convert the imageFrame in BGR(RGB color space represented as three matrices of red, green and blue with integer values from 0 to 255) to HSV(hue-saturation-value) color space.

Hue: describes a color portion of model, expressed as a number from 0 to 360

saturation: represents the amount of gray color in that color and

value: describes the brightness or intensity of the color. This can be represented as three matrices in the range of 0-179, 0-255 and 0-255 respectively.

Step 4: Define the range of each color and create the corresponding mask.

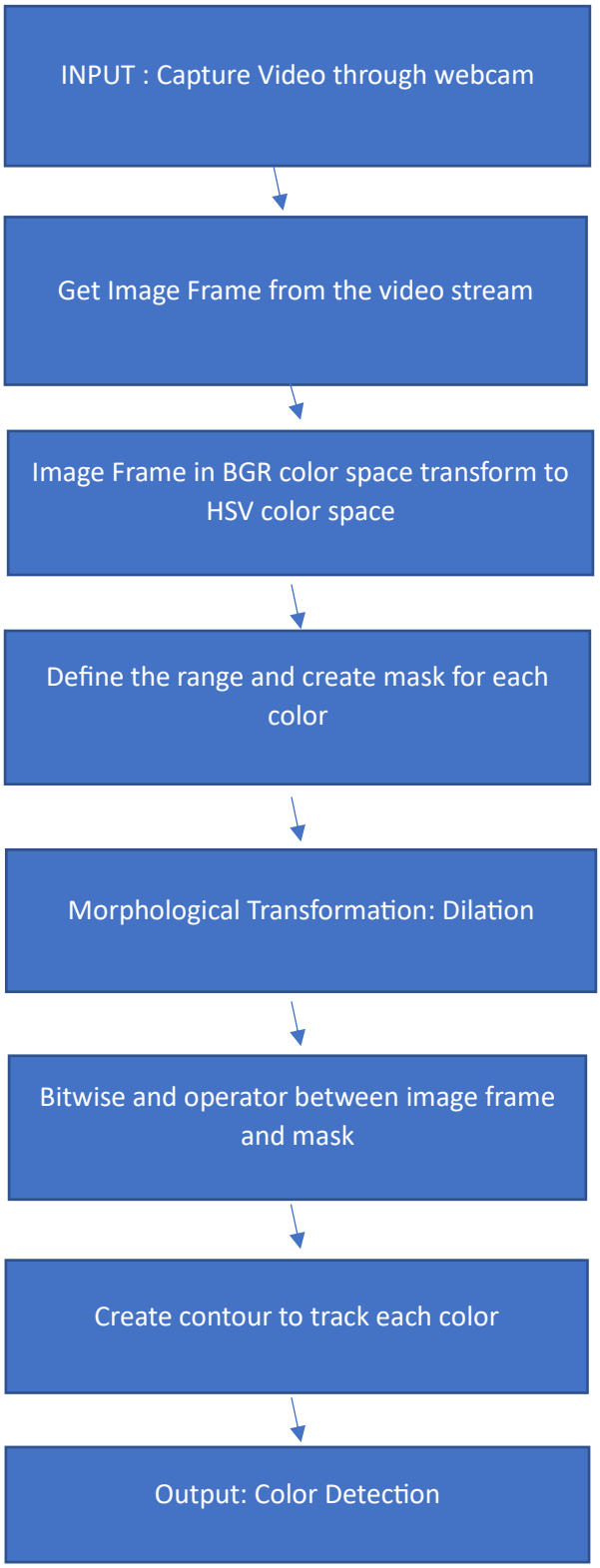
Step 5: Morphological Transform: Dilation, to remove noises from the images.

Step 6: bitwise_and between the image frame and mask is performed to specifically detect that particular color and discrad others.

Step 7: Create contour for the individual colors to display the detected colored region distinguishly.

Step 8: Detection of the colors in real-time as output.

Flowchart:


INPUT : Capture Video through webcam
Get Image Frame from the video stream
Image Frame in BGR color space transform to HSV color space
Define the range and create mask for each color
Morphological Transformation: Dilation
Bitwise and operator between image frame and mask
Create contour to track each color
Output: Color Detection

Tools used:

PyCharm is an integrated development environment (IDE) for Python. It is a tool that helps you write, run, and debug your Python code.

With PyCharm, you can write high-quality, efficient code faster, and you can also get help with any errors or problems you might have.

PyCharm has many useful features, such as syntax highlighting, code completion, and a powerful debugger, that can make it easier to develop Python projects. It also has a user-friendly interface, which makes it easy for beginners to learn and use.

Implementation of this project:-

1.Importing libraries in python:

```
import numpy as np

import cv2
```

Import cv2 imports the OpenCV (Open Source Computer Vision) library, which is a collection of tools for performing tasks related to computer vision such as image processing and video analysis. With this import statement, you can use the functions, methods, and objects provided by OpenCV in your code.

Import numpy as np imports the NumPy library, which is a collection of functions for working with arrays and performing numerical operations. NumPy is a fundamental library for scientific computing with Python. The as np part of the import statement gives the NumPy library the alias np, which allows you to use np as a shorthand to refer to the library when you want to use its functions and objects.

2.Capturing Video through webcam :

```
webcam=cv2.videocapture(0)
```

cv2.videocapture is a function of openCV library(used for computer vision, machine learning, and image processing) which allows working with video either by capturing via live webcam or by a video file

This will return video from the first webcam on your computer.

3. Loop:

We are using video (image frame after image frame) as input.

We are using while loop on the later codes so that we are able to move to the next frame after getting previous frame.

Without this loop we will only get one frame from the input and won't be able to go to the next frame.

4.Reading the video from the webcam in image frame:

```
imageFrame = webcam.read()
```

.read() returns 2 things, boolean and data. If there are not 2 variables, a tuple will be assigned to one variable. The boolean is mostly used for error catching. Assigning a tuple to 1 var with 2 data items can sometimes be useful, but in this instance, we should create 2 vars and split the data into 2 variables to make it easier.

5.Convert BGR Image Frame into HSV color space:

```
hsvFrame = cv2.cvtColor(imageFrame, cv2.COLOR_BGR2HSV)
```

The cvtColor() function in OpenCV takes two parameters namely image and code where the image is the image whose color space is to be converted to different color space and the code represents the color conversion code.

6. 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)
```

We **use the np. array() function** to make a numpy array. All you need to do is pass a list to it, and optionally, we can also specify the data type of the data.

The **inRange()** function returns an array consisting of elements equal to 0 if the elements of the source array lie between the elements of the two arrays .

The same is done for green and blue color.

7. 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")

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

The process of performing convolution with kernel having anchor point of a particular shape in a given input image is called dilate() function in OpenCV. The dilate() function starts with computing the minimum pixel value by overlapping the kernel over the input image.

In order to be able to perform bit wise conjunction of the two arrays corresponding to the two images in OpenCV, we make use of bitwise_and operator. To be able to make use of bitwise_and operator in our program, we must import the module cv2.

The same is done for green and blue color.

8. 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,cv2.FONT_HERSHEY_SIMPLEX,
1.0,(0, 0, 255))
```

cv.findContours() function, first one is source image, second is contour retrieval mode, third is contour approximation method. And it outputs the contours and hierarchy.

The cv2. boundingRect() function of OpenCV is **used to draw an approximate rectangle around the binary image**.

The same is done for green and blue color.

9. Program Termination:

```
cv2.imshow("Multiple Color Detection in Real-TIme", imageFrame)
if cv2.waitKey(10) & 0xFF == ord('q'):
```

```
cap.release()

cv2.destroyAllWindows()

break  cv2.imshow("Multiple Color Detection in Real-TIme", imageFrame)

if cv2.waitKey(10) & 0xFF == ord('q'):

cap.release()

cv2.destroyAllWindows()

break
```

cv2. imshow() method is **used to display an image in a window**. The window automatically fits the image size.

waitkey() function of Python **OpenCV** allows users to display a window for given milliseconds or until any key is pressed.

cap.release() will **Closes video file or capturing device**.

cv2.destroyAllWindows() is a function from the OpenCV library that closes all windows created by OpenCV. When this function is called, any windows that are currently open and were created using OpenCV will be closed and their resources will be released. This function is typically used to clean up after a program that displays images or videos using OpenCV. It is a good idea to call this function at the end of your program to ensure that all windows are properly closed and resources are released

SOURCE CODE:

main.py | hsv_color_picker.py

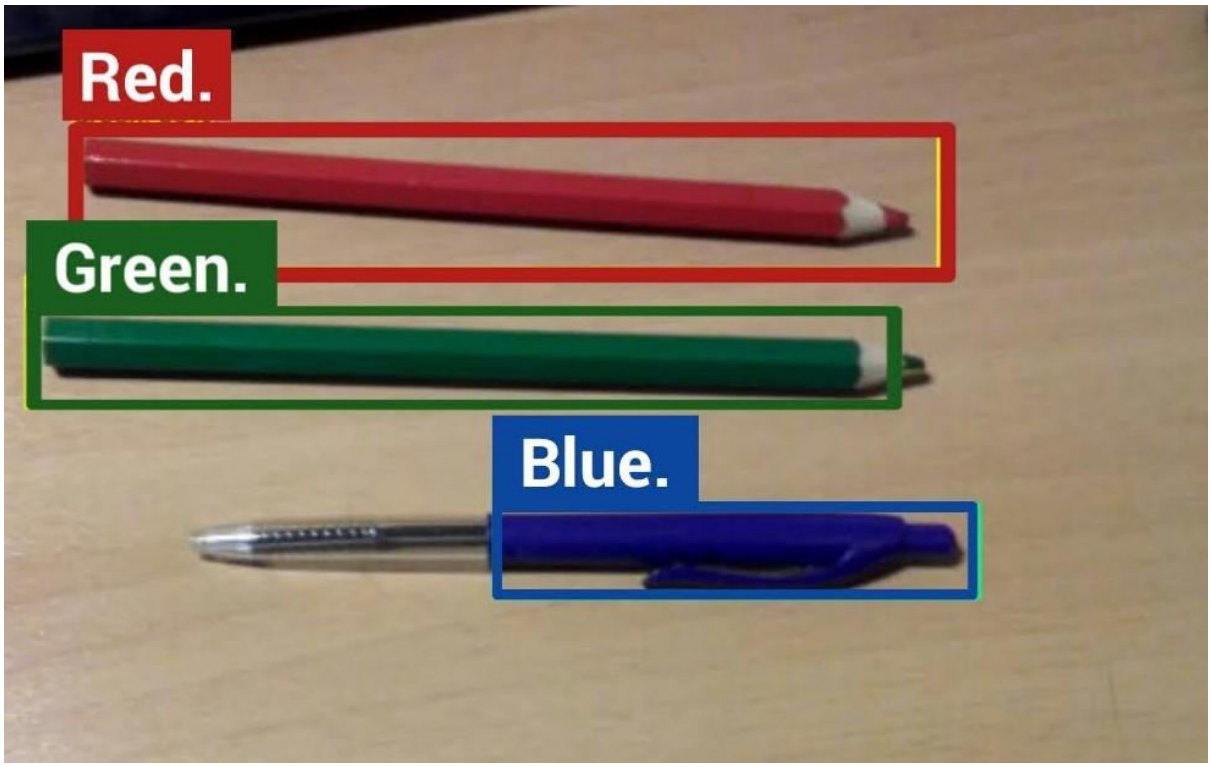
```
import numpy as np
import cv2

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

# Start a while loop
while (1):

    # 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([0, 50, 120], np.uint8)
    red_upper = np.array([10, 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([40, 70, 80], np.uint8)
    green_upper = np.array([70, 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([90, 60, 0], np.uint8)
    blue_upper = np.array([121, 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")

    # 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)
```

```
61
62      # Creating contour to track red color
63      contours, hierarchy = cv2.findContours(red_mask,
64                                             cv2.RETR_TREE,
65                                             cv2.CHAIN_APPROX_SIMPLE)
66
67      for pic, contour in enumerate(contours):
68          area = cv2.contourArea(contour)
69          if (area > 300):
70              x, y, w, h = cv2.boundingRect(contour)
71              imageFrame = cv2.rectangle(imageFrame, (x, y),
72                                         (x + w, y + h),
73                                         (0, 0, 255), 2)
74
75              cv2.putText(imageFrame, "Red Colour", (x, y),
76                          cv2.FONT_HERSHEY_SIMPLEX, 1.0,
77                          (0, 0, 255))
78
79      # Creating contour to track green color
80      contours, hierarchy = cv2.findContours(green_mask,
81                                             cv2.RETR_TREE,
82                                             cv2.CHAIN_APPROX_SIMPLE)
83
84      for pic, contour in enumerate(contours):
85          area = cv2.contourArea(contour)
86          if (area > 300):
87              x, y, w, h = cv2.boundingRect(contour)
88              imageFrame = cv2.rectangle(imageFrame, (x, y),
89                                         (x + w, y + h),
90                                         (0, 255, 0), 2)
91
```

```
91
92                  cv2.putText(imageFrame, "Green Colour", (x, y),
93                              cv2.FONT_HERSHEY_SIMPLEX,
94                              1.0, (0, 255, 0))
95
96          # Creating contour to track blue color
97          contours, hierarchy = cv2.findContours(blue_mask,
98                                                 cv2.RETR_TREE,
99                                                 cv2.CHAIN_APPROX_SIMPLE)
100         for pic, contour in enumerate(contours):
101             area = cv2.contourArea(contour)
102             if (area > 300):
103                 x, y, w, h = cv2.boundingRect(contour)
104                 imageFrame = cv2.rectangle(imageFrame, (x, y),
105                                            (x + w, y + h),
106                                            (255, 0, 0), 2)
107
108                 cv2.putText(imageFrame, "Blue Colour", (x, y),
109                             cv2.FONT_HERSHEY_SIMPLEX,
110                             1.0, (255, 0, 0))
111
112         # Program Termination
113         cv2.imshow("Multiple Color Detection in Real-TIme", imageFrame)
114         if cv2.waitKey(10) & 0xFF == ord('q'):
115             cap.release()
116             cv2.destroyAllWindows()
117             break
```

Output:



Conclusion:

Color Detection project enable us to detect multiple colors in real-time. It uses the cv2 and numpy libraries to handle image manipulation and data manipulation tasks .

It can also be used as a learning tool to understand how colors are represented in a computer system and how to manipulate them using Python.

In the future, this project can be extended to support more advanced features, such as detection of traffic signals in self driving car or to performing pick-and-place task in separating different colored objects in some industrial robots.

References:

https://en.wikipedia.org/wiki/Color https://data-flair.training/blogs/project-in-python-colour-detection/ https://github.com/codebrainz/color-names/blob/master/output/colors.csv
https://www.geeksforgeeks.org/introduction-to-pandas-in-python/
https://www.javatpoint.com/numpy-tutorial
https://www.youtube.com/watch?v=VU07jbfe9dU&t=281