COLUR DETECTION USING PYTHON

MINI PROJECT ON ANALYTICS

Submitted by

Bhavani. P (812620243002)

Vaishnavi. S (812620243022)

In partial fulfilment for the award of the degree

Of

BACHELOR OF TECHONOLOGY

IN

ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

M.A.M COLLEGE OF ENGINEERING

ANNA UNIVERSITY: CHENNAI - 600 025

NOVEMBER 2023

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this Mini Project Report "**COLOUR DETECTION USING PYTHON**" is the bonafide work of **VAISHNAVI S,BHAVANI P, w**ho carried out the project work under my supervision.

SIGNATURE

Mr. K. ILANGO

HEAD OF THE DEPARTMENT

Artificial Intelligence and Data Science

M.A.M. College of Engg.

Trichy, Chennai Trunk Road,

Siruganur, Tamil Nadu 621105.

SIGNATURE

Mr. K. ILANGO

SUPERVISOR

Artificial Intelligence and Data Science

M.A.M. College of Engg.

Trichy, Chennai Trunk Road.

Siruganur, Tamil Nadu 621105.

Submitted for the viva-voce examination held on _____

INTERNAL EXAMINER

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

We extend my sincere thanks to M.A.M. College of Engineering which provided me with the opportunity to fulfill our wish and achieve our goal. We would like to express deep debt to **Prof. K. Ilango** project guide for her vital suggestion, meticulous guidance and constant motivation. We extend our gratitude to **Prof. K. Ilango**, **Head of the department** for making this project successful up till this point. They spent a lot of time with us and gave all the related information and expertise about report writing.

ABSTRACT

The foremost stage in many of the image processing applications is Color detection. It is significantly used in applications such as self-driving cars, object detection, traffic signal detection, skin tone detection and object tracing. While tracing an object in motion, color is constant than any other attributes. This paper gives an approach to detect the label of the color by placing the cursor and double clicking at that position of the image and tracks the red, green and blue color objects using bounding box property. By examining the RGB values of every pixel in the image, the color of the pixels is recognized. All the objects of interest in the video are detected and tracked by a rectangular bounding box using HSV color model. The results of this implementation can be used in self driving cars to detect traffic signal, in some industrial robots to perform pick-and-place task in separating colored objects and as a tool in various drawing and image editing applications.

In this project, We build an application through which we can automatically get the name of the color by double clicking on it. We have a data file that contains the color name and its values. Then we calculate the distance from each color and find the shortest one. We extract the color RGB values and the color name of a pixel. This also tracks three different colors Red, Blue and Green from a video. If there's any color from Red, Blue and Yellow or all the three at the same time in the live stream, rectangular boxes of Red color for tracking of Red, blue rectangular box for Blue color and Green for Green color will bound the respective color objects and the name of the color is displayed on top of it. This helps in recognizing colors and in robotics. This type of system is used in driverless cars to detect traffic and vehicle back light and take decision to stop, start or continue driving. This also has many applications in industry to pick and place different colored objects by the robotic arm. This is done using OPENCV using python programming language.

CONTENTS

LIST OF FIGURES

LIST OF TABLES Table no

Title **Page no**

LIST OF ABBREVATIONS

OpenCV	Open source computer vision
RGB	Red Green Blue
HSV	Hue Saturation Value
CPU	Central processing unit
IEEE	Institute of Electrical and Electronics Engineering
AI	Artificial Intelligence

1.INTRODUCTION

1.1 Project Objective:

For a robot to visualize the environment, along with the object detection, detection of its color is also very crucial . Color detection is necessary to recognize objects, it is also used as a tool in various image editing and drawing apps. Some real world applications are: it is used in self driving cars to detect traffic signals and is also used in some industrial robots, to perform pick-and-place task in separating different colored objects. This design system can be implemented in various fields for various purposes such as Defence, industrial purposes, games, automation, security, monitoring etc. Even these systems can also play a vital role in field of radar and navigating such as detecting, tracking of a moving colored object etc. Detection of color plays an important role even in the field of medical, in detection of color of skin, identification of a face, recognizing license plate.

1.2 Project Outline:

Color is one of the salient features of a picture. The detection of color in a live stream or in a graphic image can be employed in numerous scientific and industrial applications. Color detection is the elemental step in many image processing applications. In today's graphical world, videos and images are omnipresent, it has been a complicated task to perform computer vision using robust and economical computer devices. Open-source computer vision library (OpenCV) paves a way to serve the rising demand of high-quality video and image processing. An image is always represented in the form of a matrix containing its pixel values. Images are often illustrated using several color models like CMYK, gray-scale, RGB, HSV etc. In this system, RGB model is employed to label the colors in a picture. Red, Green and Blue lights are mixed in numerous ways to generate ample amounts of colors. In RGB model image is represented in a matrix of PxQx3 pixels with P rows and Q columns of pixels in a picture. On which different operations can be applied to label the color in an image and while tracing an object in

motion, color is constant than any other attributes. It does not get effected to shifting, rotation and scaling operations. Here, the object is traced based on the HSV color model's color model is better version of RGB color model. Color detection can be applied to various applications. Recently it has gained the attention of scientists who work on skin tone detection which results in efficient human body detection and tracing. This color information can also be applied in license plate detection which aids in tracking the vehicle. Here, the name of the color is detected using RGB color model and the corresponding Red, Green and Blue values are displayed, the object detection and tracking is performed using HSV color model in OPENCV, python.

2.OPENCV

OpenCV (Open Source Computer Vision Library) is a library of programming functions mainly aimed at real-time computer vision. Originally developed by Intel, it was later supported by Willow Garage. It is a cross-platform library using which we can develop real-time **computer vision applications**. It mainly focuses on image processing, video capture and analysis including features like face detection and object detection.

2.1 History:

Officially launched in 1999 the OpenCV project was initially an Intel Research initiative to advance CPU -intensive applications, part of a series of projects including real-time ray tracing and 3D display walls. The main contributors to the project included a number of optimization experts in Intel Russia, as well as Intel's Performance Library Team. In the early days of OpenCV, the goals of the project were described as:

- Advance vision research by providing not only open but also optimized code for basic vision infrastructure. No more reinventing the wheel.
- Disseminate vision knowledge by providing a common infrastructure that developers could build on, so that code would be more readily readable and transferable.
- Advance vision-based commercial applications by making portable, performance-optimized code available for free – with a license that did not require code to be open or free itself.

The first alpha version of OpenCV was released to the public at the IEEE Conference on Computer Vision and Pattern Recognition in 2000, and five betas were released between 2001 and 2005. The first 1.0 version was released in 2006. A version 1.1 "pre-release" was released in October 2008.

The second major release of the OpenCV was in October 2009. OpenCV 2 includes major changes to the C++ interface, aiming at easier, more type-safe patterns, new

functions, and better implementations for existing ones in terms of performance (especially on multi-core systems). Official releases now occur every six months and development is now done by an independent Russian team supported by commercial corporations.

In August 2012, support for OpenCV was taken over by a non-profit foundation OpenCV.org, which maintains a developer and user site.

On May 2016, Intel signed an agreement to acquire Itseez, a leading developer of OpenCV.

In July 2020, OpenCV announced and began a Kickstarter campaign for the OpenCV AI Kit, a series of hardware modules and additions to OpenCV supporting Spatial AI.

2.2 Computer Vision:

Computer Vision can be defined as a discipline that explains how to reconstruct, interrupt, and understand a 3D scene from its 2D images, in terms of the properties of the structure present in the scene. It deals with modelling and replicating human vision using computer software and hardware.

Computer Vision overlaps significantly with the following fields −

- **Image Processing** − It focuses on image manipulation.
- **Pattern Recognition** − It explains various techniques to classify patterns.
- **Photogrammetry** − It is concerned with obtaining accurate measurements from images.

Image processing deals with image-to-image transformation. The input and output of image processing are both images.

Computer vision is the construction of explicit, meaningful descriptions of physical objects from their image. The output of computer vision is a description or an interpretation of structures in 3D scene.

2.3 Applications:

- 2D and 3D feature toolkits
- Egomotion estimation
- Facial recognition system
- Gesture recognition
- Human computer interaction (HCI)
- Mobile robotics
- Motion understanding
- Object detection
- Segmentation and recognition
- Stereopsis stereo vision: depth perception from 2 cameras
- Structure from motion (SFM)
- Motion tracking

2.4 Features of OpenCV Library:

Using OpenCV library, you can

- Read and write images
- Capture and save videos
- Process images (filter, transform)
- Perform feature detection
- Detect specific objects such as faces, eyes, cars, in the videos or images.
- Analyze the video, i.e., estimate the motion in it, subtract the background, and track objects in it.

OpenCV was originally developed in C++. In addition to it, Python and Java bindings were provided. OpenCV runs on various Operating Systems such as windows, Linux, OSx, FreeBSD, Net BSD, Open BSD, etc.

2.5 OpenCV Library Modules:

Core Functionality：

This module covers the basic data structures such as Scalar, Point, Range, etc., that are used to build OpenCV applications. In addition to these, it also includes the multidimensional array **Mat**, which is used to store the images. In the Java library of OpenCV, this module is included as a package with the name **org.opencv.core**.

Image Processing:

This module covers various image processing operations such as image filtering, geometrical image transformations, color space conversion, histograms, etc. In the Java library of OpenCV, this module is included as a package with the name **org.opencv.imgproc**.

Video and Video I/O:

This module covers the video analysis concepts such as motion estimation, background subtraction, and object tracking. In the Java library of OpenCV, this module is included as a package with the name **org.opencv.video**. This module explains the video capturing and video codecs using OpenCV library. In the Java library of OpenCV, this module is included as a package with the name **org.opencv.videoio**.

Objdetect:

This module includes the detection of objects and instances of the predefined classes such as faces, eyes, mugs, people, cars, etc. In the Java library of OpenCV, this module is included as a package with the name **org.opencv.objdetect**.

3.RGB COLOR MODEL

3.1 Introduction:

The RGB color model is an additive color model in which red, green, and blue light are added together in various ways to reproduce a broad array of colors. The name of the model comes from the initials of the three additive primary colors, red, green, and blue.

The main purpose of the RGB color model is for the sensing, representation, and display of images in electronic systems, such as televisions and computers, though it has also been used in conventional photography. Before the electronic age, the RGB color model already had a solid theory behind it, based in human perception of colors.

RGB is a device-dependent color model: different devices detect or reproduce a given RGB value differently, since the color elements (such as phosphors or dyes) and their response to the individual R, G, and B levels vary from manufacturer to manufacturer, or even in the same device over time. Thus an RGB value does not define the same color across devices without some kind of color management.

Typical RGB input devices are color TV and video cameras, image scanners, and digital cameras. Typical RGB output devices are TV sets of various technologies (CRT, LCD, plasma, OLED, quantum dots, etc.), computer and mobile phone displays, video projectors, multicolor LED displays and large screens such as Jumbotron. Color printers, on the other hand are not RGB devices, but subtractive color devices. This article discusses concepts common to all the different color spaces that use the RGB color model, which are used in one implementation or another in color image-producing technology.

3.2 Additive Colors:

To form a color with RGB, three light beams must be superimposed. Each of the three beams is called a component of that color, and each of them can have an arbitrary intensity, from fully off to fully on, in mixture.

The RGB color model is additive in sense that three light beams are added together, and their light spectra add, wavelength for wavelength, to make the final color's spectrum. This is essentially opposite to the model, particularly the CMY color model that applies to paints, inks, dyes, and other substances whose color depends on reflecting the light under which we see them. Because of properties, these three colors create white, this is in stark contrast to physical colors, such as dyes which create black when mixed.

Zero intensity for each component gives the darkest color, and full intensity of each gives a white; the quality of this white depends on the nature of the primary light sources, but if they are properly balanced, the result is a neutral white matching the system's white point. When the intensities for all the components are the same, the result is a shade of gray, darker or lighter depending on the intensity. When the intensities are different, the result is a colorized hue, more or less saturated depending on the difference of the strongest and weakest of the intensities of the primary colors employed.

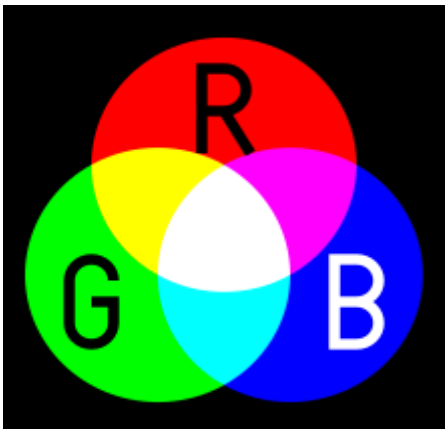


Figure 3.1: RGB color model

yields yellow; adding green to blue yields cyan; adding blue to red yields magenta; adding all three primary colors together yields white.

When one of the components has the strongest intensity, the color is a hue near this primary color (red-ish, green-ish, or blue-ish), and when two components have the same strongest intensity, then the color is a hue of a secondary color. A secondary color is formed by the sum of two primary colors of equal intensity: cyan is green+blue, magenta is blue+red, and yellow is red+green. Every secondary color is the complement of one primary color: cyan complements red, magenta complements green, and yellow complements blue. When all the primary colors are mixed in equal intensities, the result is white.

The RGB color model itself does not define what is meant by red, green, blue colorimetrically, and so the results of mixing them are not specified as absolute, but relative to the primary colors. When the exact chromaticities of the red, green, and blue primaries are defined, the color model then becomes an absolute color space, such as sRGB or Adobe RGB; see RGB color space for more details.

Figure 3.2 Color wheel cycle

The above colors in clockwise from the top represent:

red,orange,yellow,chartreuse,green,spring,cyan,azure,blue,violet,magneta and rose

3.3 PHYSICAL PRINCIPLES FOR THE CHOICE OF RED, GREEN AND BLUE:

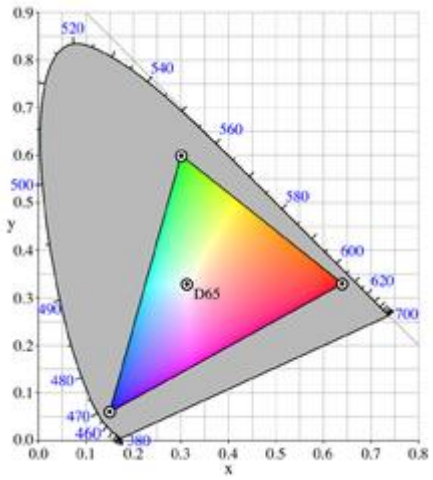


Figure3.3 RGB color gamut

A set of primary colors, such as the sRGB primaries, define a color triangle; only colors within this triangle can be reproduced by mixing the primary colors. Colors outside the color triangle are therefore shown here as gray. The primaries and the D65 white point of sRGB are shown. The background figure is the CIE xy chromaticity diagram.

The choice of primary colors is related to the physiology of the human eye; good primaries are stimuli that maximize the difference between the responses of the cone cells of the human retina to light of different wavelengths, and that thereby make a large color triangle.

The normal three kinds of light-sensitive photoreceptor cells in the human eye (cone cells) respond most to yellow (long wavelength or L), green (medium or M), and violet (short or S) light (peak wavelengths near 570 nm, 540 nm and 440 nm, respectively). The difference in the signals received from the three kinds allows the brain to differentiate a wide gamut of different colors, while being most sensitive (overall) to yellowish-green light and to differences between hues in the green-to-orange region.

As an example, suppose that light in the orange range of wavelengths (approximately 577 nm to 597 nm) enters the eye and strikes the retina. Light of these wavelengths would activate both the medium and long wavelength cones of the retina, but not equally—the long-wavelength cells will respond more. The difference in the response can be detected by the brain, and this difference is the basis of our perception of orange. Thus, the orange appearance of an object results from light from the object entering our eye and stimulating the different cones simultaneously but to different degrees.

Use of the three primary colors is not sufficient to reproduce all colors; only colors within the color triangle defined by the chromaticities of the primaries can be reproduced by additive mixing of non-negative amounts of those colors of light.

3.4 Numeric Representations:

A color in the RGB color model is described by indicating how much of each of the red, green, and blue is included. The color is expressed as an RGB triplet (*r*,*g*,*b*), each component of which can vary from zero to a defined maximum value. If all the components are at zero the result is black; if all are at maximum, the result is the brightest representable white.

These ranges may be quantified in several different ways:

1. From 0 to 1, with any fractional value in between. This representation is used in theoretical analyses, and in systems that use floating point representations.
2. Each color component value can also be written as a percentage, from 0% to 100%.
3. In computers, the component values are often stored as unsigned integer numbers in range 0 to 255, the range that a single 8-bit byte can offer. These are often represented as either decimal or hexadecimal numbers.

4. High-end digital image equipment are often able to deal with larger integer ranges for each primary color, such as 0..1023(10 bits), 0..65535(16 bits) or larger, by extending the 24-bits to 32-bit, 48-bit, 64-bit units.

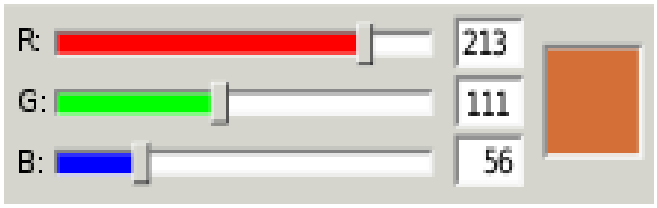


Figure 3.4 RGB color slider

A typical RGB color selector in graphics software. Each slider ranges from 0 to 255

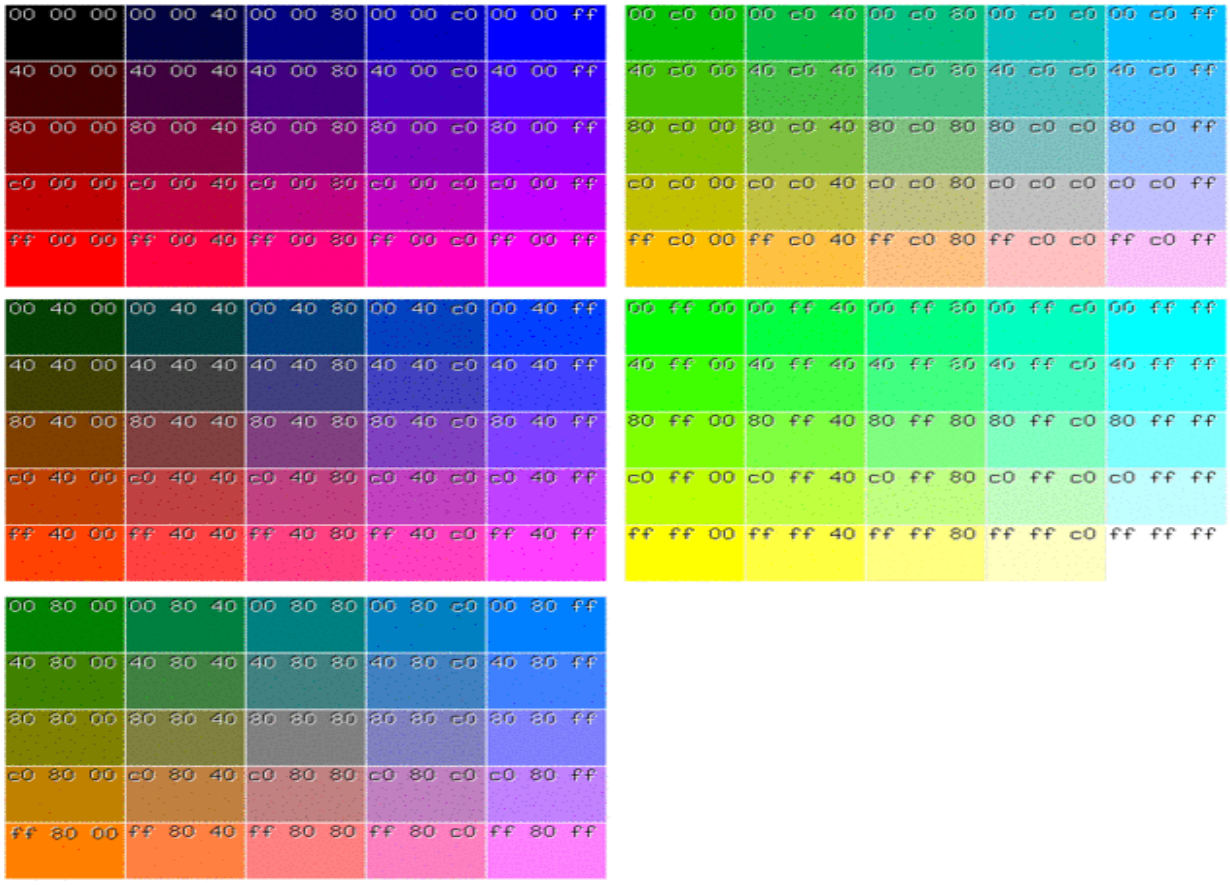


Figure 3.5 Hexadecimal 8-bit RGB representations of the main 125 colors

For example, brightest saturated red is written in the different RGB notations as:

Table 3.1Brightest saturated red in different RGB notations

Notation	**RGB triplet**
Arithmetic	(1.0, 0.0, 0.0)
Percentage	(100%, 0%, 0%)
Digital 8-bit per channel	(255, 0, 0) or sometimes #FF0000 (hexadecimal)
Digital 12-bit per channel	(4095, 0, 0)
Digital 16-bit per channel	(65535, 0, 0)
Digital 24-bit per channel	(16777215, 0, 0)
Digital 32-bit per channel	(4294967295, 0, 0)

In many environments, the component values within the ranges are not managed as linear, as in digital cameras and TV broadcasting and receiving due to gamma correction, for example. Linear and nonlinear transformations are often dealt with via digital image processing. Representations with only 8 bits per component are considered sufficient if gamma encoding is used.

4. HSV COLOR MODEL

4.1 Introduction:

A **color model** is an abstract mathematical model describing the way colors can be represented as tuples of numbers, typically as three or four values or color components. When this model is associated with a precise description of how the components are to be interpreted (viewing conditions, etc.), the resulting set of colors is called "color space." This section describes ways in which human color vision can be modeled. Unlike the RGB color model, which is hardware-oriented, the HSV model is user-oriented, based on the more intuitive appeal of combining hue, saturation, and value elements to create a color.

HSV is a cylindrical color model that remaps the RGB primary colors into dimensions that are easier for humans to understand. Like the Munsell Color System, these dimensions are hue, saturation, and value. Hue, saturation, and value are the main color properties that allow us to distinguish between different colors. Using color effectively is one of the most essential elements in photography, as color can draw the viewer's eye to your composition and affect the mood and emotional impact your photo.

It is important to note that the three dimensions of the HSV color model are interdependent. If the value dimension of a color is set to 0%, the amount of hue and saturation does not matter as the color will be black. Likewise, if the saturation of a color is set to 0%, the hue does not matter as there is no color used. Because the hue dimension is circular, the HSV color model is best depicted as a cylinder. This is illustrated in the interactive example below, where all possible color mixes are represented within the bounds of the cylinder.

4.2 Hue, Saturation, Value:

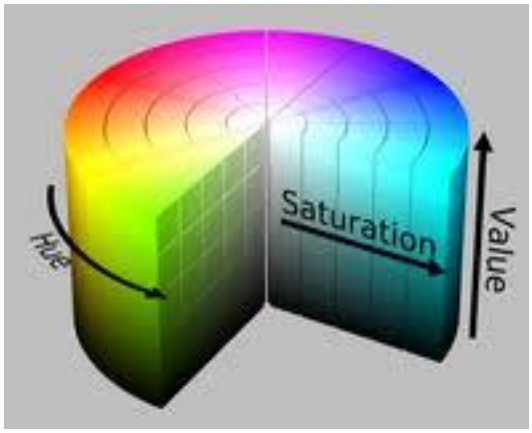


Figure 4.1

Hue specifies the angle of the color on the RGB color circle. A 0° hue results in red, 120° results in green, and 240° results in blue.Hue is the color portion of the model, expressed as a number from 0 to 360 degrees:

- **Red** falls between 0 and 60 degrees.
- **Yellow** falls between 61 and 120 degrees.
- **Green** falls between 121 and 180 degrees.
- **Cyan** falls between 181 and 240 degrees.
- **Blue** falls between 241 and 300 degrees.
- **Magenta** falls between 301 and 360 degrees.

Saturation controls the amount of color used. A color with 100% saturation will be the purest color possible, while 0% saturation yields grayscale. Saturation describes the amount of gray in a particular color, from 0 to 100 percent. Reducing this component toward zero introduces more gray and produces a faded effect. Sometimes, saturation appears as a range from 0 to 1, where 0 is gray, and 1 is a primary color.

Value controls the brightness of the color. A color with 0% brightness is pure black while a color with 100% brightness has no black mixed into the color. Because this

dimension is often referred to as brightness, the HSV color model is sometimes called HSB, including in P5.js. Value works in conjunction with saturation and describes the brightness or intensity of the color, from 0 to 100 percent, where 0 is completely black, and 100 is the brightest and reveals the most color.

4.3 HSV Representations:

The HSV color wheel is used to pick the desired color. Hue is represented by the circle in the wheel. A separate triangle is used to represent saturation and value. The horizontal axis of the triangle indicates value and the vertical axis represents saturation. When you need a particular color for your picture, first you need to pick a color from the hue (the circular region), and then from the vertical angle of the triangle you can select the desired saturation. For brightness, you can select the desired value from the horizontal angle of the triangle.

Sometimes the HSV model is illustrated as a cylindrical or conical object. When it is represented as a conical object, hue is represented by the circular part of the cone. The cone is usually represented in the three-dimensional form. The saturation is calculated using the radius of the cone and value is the height of the cone. A hexagonal cone can also be used to represent the HSV model. The advantage of the conical model is that it is able to represent the HSV color space in a single object. Due to the two-dimensional nature of computer interfaces, the conical model of HSV is best suited for selecting colors for computer graphics.

The application of the cylindrical model of HSV color space is similar to the conical model. Calculations are done in a similar way. Theoretically, the cylindrical model is the most accurate form of HSV color space calculation. In practical use, it is not possible to distinguish between saturation and hue when the value is lowered. The cylindrical model has lost its relevance due to this and the cone shape is preferred over it.

4.4 Advantages of HSV:

The HSV color space is quite similar to the way in which humans perceive color. The other models, except for HSL, define color in relation to the primary colors. The colors used in HSV can be clearly defined by human perception, which is not always the case with RGB or CMYK.

4.5 Uses and Applications of HSV:

Designers use the HSV color model when selecting colors for paint or ink because HSV better represents how people relate to colors than the RGB color model does.

The HSV color wheel also contributes to high-quality graphics. Although less well-known than its RGB and CMYK cousins, the HSV approach is available in many high-end image editing software programs.

Selecting an HSV color begins with picking one of the available hues and then adjusting the shade and brightness values.

HSV color space is widely used to generate high quality computer graphics or it is used to select various different colors needed for a particular picture.

An HSV color wheel is used to select the desired color. A user can select the particular color needed for the picture from the color wheel.

Used in color pickers, image editing software, image analysis, computer vision.

5.COLOR DETECTION USING RGB

5.1 Introduction:

Color detection is the process of detecting the name of any color. 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. In computer it will be done in a different way. With modern industrial production develops toward the high speed and automatic direction, color recognition has been widely used in various industrial detection and automatic control field. And the work of color identification which is led by the human eye in the long-term production has been replaced by more and more color sensors.

Color sensor detects color with comparison the object color with the reference color, and if they are consistent in a certain error range, then output the detection results. Color sensor can be applied in many fields, such as monitoring the production process and product quality in the industry; the realization of the true color copy without affected by environmental temperature, humidity, paper and toner influence in the electronic reproduction aspects; a disease indicator to study a sickness in the Medical; and automatic control in detection two adjacent label colors of a paper and automatically count the number of all sorts of color by auto-counter in the commodity packaging.

5.2 The Data set:

Computer screens and related devices also rely on mixing three colors, except they need a different set of primary colors because they are additive, starting with a black screen and adding color to it. For additive color on computers, the colors red, green and blue (RGB)

are used. Each pixel on a screen is typically made up of three tiny "lights"; one red, one green, and one blue. By increasing and decreasing the amount of light coming out of each

of these three, all the different colors can be made. The following interactive allows you to play around with RGB.

Colors are made up of 3 primary colors: red, green, and blue. In computers, we define each color value within a range of 0 to 255. So in many ways we can define a color 256*256*256 = 16,581,375. There are approximately 16.5 million different ways to represent a color. In our dataset, we need to map each color's values with their corresponding names. But don't worry, we don't need to map all the values. We will be using a dataset that contains RGB values with their corresponding names. Csv file includes 865 color names along with their RGB and hex values.

Steps for detecting color in an image:

Here are the steps to build an application in Python that can detect colors:

5.3 Algorithm:

Image Capture: The first step is to fetch a high-quality image with resolution. To loadan image from a file we use Cv2.imread(). The full path of the image has to be given as input or else the input has to be in our working directory.

Img=cv2.imread(img path)

Extraction of RGB Colors: In this process, the 3 layered colors are extracted from the input image. All the color images on screens such as televisions, computer, monitors, laptops and mobile screens are produced by the combination of Red, Green and Blue light. The intensity value of each color ranges between 0(lowest) to 255(highest). By combining any 3 primary colors at different intensity levels a large variety of colors are produced.

For Example: If the intensity value of the colors is 0, this combination corresponds to black. If the intensity value of the primary colors is 1, this combination corresponds to white.

Index= ["color", "color_name", "hex", "R", "G", "B"]. This function calculates the minimum distance from the coordinates. The minimum distance is calculated by considering moving towards the origin point is circulated among all colors to find the most matching color. The pandas library serves as an important utility to perform various operations on comma-separated values. We need to read the csv file and upload it into the pandas data frame.

D = abs(R-int(csv.loc[i ,"R"])) + abs (G-int (csv.loc[i ,"G"])) + abs (B- int (csv.loc [i,"B"]))

Image Display with Shades of Color: The rectangle window is used to display the image with shades of color. After the double-click is triggered, the RGB values and color name is updated.

To display an image Cv2.imshow() method is used. The color name and its intensity level will be displayed in a rectangular box.

text=getColorName(r,g,b) + 'R='+str(r) + 'G='+str(g) + 'B=' +str(b).

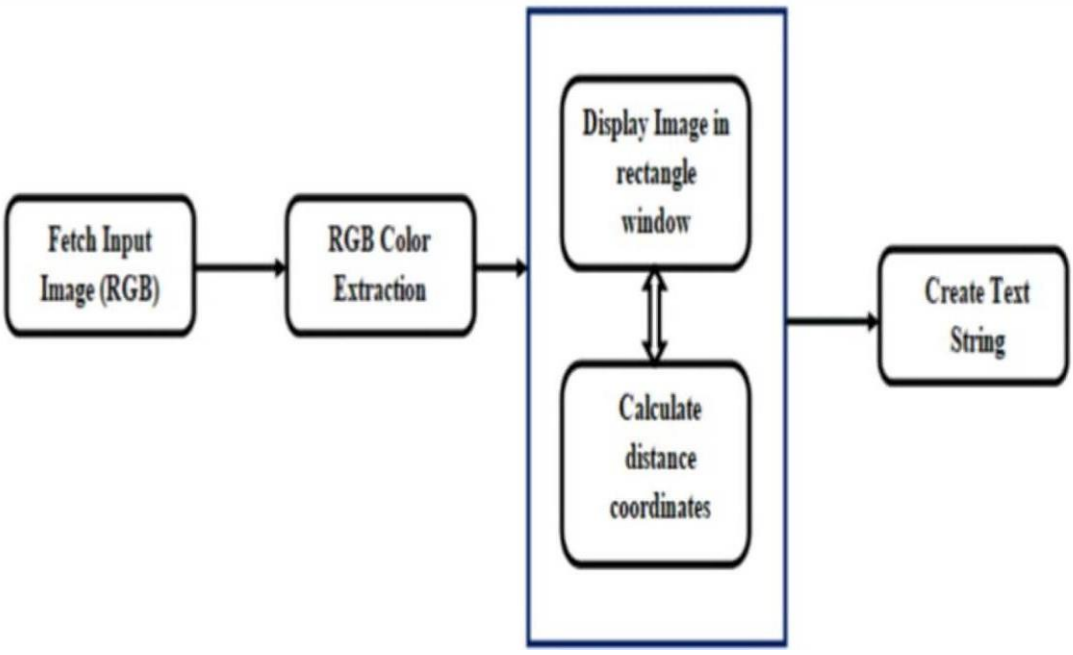


Figure 5.1 Architecture diagram for color detection

5.4 Import the required packages and load the image:

```
Import cv2
    import numpy as np
    import pandas as pdimg_path = "D://OpenCV//shape-detection//New folder//color palette.jpg"
    img = cv2.imread(img_path)
```

5.4.1 OpenCV-Python is a library of Python bindings designed to solve computer vision problems.

cv2.imread () method loads an image from the specified file. If the image cannot be read (because of missing file, improper permissions, unsupported or invalid format) then this method returns an empty matrix.

***Syntax*:** cv2.imread (path, flag)

Parameters:-

path: A string representing the path of the image to be read.

flag: It specifies the way in which image should be read. It's default value is cv2.IMREAD_COLOR

Return Value: This method returns an image that is loaded from the specified file.

cv2.IMREAD_COLOR: It specifies to load a color image. Any transparency of image will be neglected. It is the default flag. Alternatively, we can pass integer value 1 for this flag.

cv2.IMREAD_GRAYSCALE: It specifies to load an image in grayscale mode. Alternatively, we can pass integer value 0 for this flag.

cv2.IMREAD_UNCHANGED: It specifies to load an image as such including alpha channel. Alternatively, we can pass integer value -1 for this flag.

5.4.2 NumPy stands for Numerical Python. It is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, Fourier transform, and matrices. It is an open source project and you can use it freely. In Python we have lists that serve the purpose of arrays, but they are slow to process. It aims to provide an array object that is up to 50x faster than traditional Python lists. The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy. Arrays are very frequently used in data science, where speed and resources are very important.

5.4.3 Pandas is a Python package that provides fast, flexible, and expressive data structures designed to make working with structured (tabular, multidimensional, potentially heterogeneous) and time series data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python. Additionally, it has the broader goal of becoming the most powerful and flexible open source data analysis / manipulation tool available in any language**.**

Pandas can easily do:

- Easy handling of missing data in floating point as well as non-floating point data
- Size mutability: columns can be inserted and deleted from DataFrame and higher dimensional objects
- Automatic and explicit data alignment: objects can be explicitly aligned to a set of labels, or the user can simply ignore the labels and let Series, DataFrame, etc. automatically align the data for you in computations
- Powerful, flexible group by functionality to perform split-apply-combine operations on data sets, for both aggregating and transforming data
- Make it easy to convert ragged, differently-indexed data in other Python and NumPy data structures into DataFrame objects
- Intelligent label-based slicing, fancy indexing, and subsetting of large data sets
- Intuitive merging and joining data sets
- Flexible reshaping and pivoting of data sets

- Hierarchical labelling of axes (possible to have multiple labels per tick)
- Robust IO tools for loading data from flat files (CSV and delimited), Excel files, databases, and saving / loading data from the ultrafast HDF5 format
- Time series-specific functionality: date range generation and frequency conversion, moving window statistics, date shifting and lagging.

5.5 Read the CSV file with pandas:

The pandas library is very useful when we need to perform various operations on data files like CSV. pd.read_csv() reads the CSV file and loads it into the pandas DataFrame. We have assigned each column with a name for easy accessing.

index=[“color”,”color_name”,”hex”,”R”,”G”,”B”]

csv = pd.read_csv(‘colors.csv’, names=index, header=None)

5.5.1 Index () is an inbuilt function in Python, which searches for a given element from the start of the list and returns the lowest index where the element appears.

Syntax:

list_name.index(element,start,end)**:**

element - The element whose lowest index will be returned.

start *(Optional)* - The position from where the search begins.

end *(Optional)* - The position from where the search ends.

Returns:

Returns lowest index where the element appears.

a)List Index:

The index () method returns the position at the first occurrence of the specified value.

Syntax:

list. Index (*elmnt*)

Parameters:

Element – Any type (string,number,list, etc.). The element to search for

b**)String Index:**

The index () method finds the first occurrence of the specified value.

The index () method raises an exception if the value is not found.

The index () method is almost the same as the find () method, the only difference is that the find ()method returns -1 if the value is not found.

Syntax:

string.Index (value, start, end)

Value - The value to search for

start *(*Optional*)* - The position from where the search begins. Default is 0.

end *(*Optional*)* - The position from where the search ends. Default is to the end of the string.

5.5.2 A CSV (Comma Separated Values) format is one of the most simple and common ways to store tabular data. To represent a CSV file, it must be saved with the **.csv** file extension. To read a CSV file in Python, we can use the csv.reader () function. To write to a CSV file in Python, we can use the csv.writer () function. The csv.writer () function returns a writer object that converts the user's data into a delimited string. This string can later be used to write into CSV files using the writerow () function.

5.6 Set a mouse callback event on a window:

First, we created a window in which the input image will display. Then, we set a callback function which will be called when a mouse event happens.

cv2.namedWindow (‘color detection’)

cv2.setMouseCallback (‘color detection’, draw function)

First we create a mouse callback function which is executed when a mouse event take place. Mouse event can be anything related to mouse like left-button down, left-button up, left-button double-click etc. It gives us the coordinates (x, y) for every mouse event. With this event and location, we can do whatever we like. Creating mouse callback function has a specific format which is same everywhere. It differs only in what the function does. So our mouse callback function does one thing, it draws a circle where we double-click. For improvement we can also draw either rectangles or circles (depending on the mode we select) by dragging the mouse like we do in Paint application. So our mouse callback function has two parts, one to draw rectangle and other to draw the circles.

5.7 Create the draw_function:

It will calculate the rgb values of the pixel which we double click. The function parameters have the event name, (x, y) coordinates of the mouse position, etc. In the function, we check if the event is double-clicked then we calculate and set the r, g, b values along with x,y positions of the mouse.

```
def draw_function(event, x,y,flags,param):
    if event == cv2.EVENT_LBUTTONDBLCLK:
        global b,g,r,xpos,ypos, clicked
        clicked = True
        xpos = x
        ypos = y
        b,g,r = img[y,x]
        b = int(b)
        g = int(g)
        r = int(r)
```

To find the RGB value of a pixel we use PIL.Image.Image.getpixel ()
Call PIL.Image.open (fp) with fp as the filename of the image to return a PIL.Image.Image object.Call PIL.Image.Image.convert (mode) with mode as "RGB" to return a PIL.Image.Image object in the RGB colorspace. Call PIL.Image.Image.getpixel (xy) with xy as a tuple containing the x and y coordinates of the desired pixel to return its RGB value.

Input: red image

```
red_image = PIL.Image.open("red_image.png") {Create a PIL.Image object}
red_image_rgb = red_image.convert("RGB") {Convert to RGB colorspace}
rgb_pixel_value = red_image_rgb.getpixel((10,15)) {Get color from (x, y) coordinates}
print (rgb_pixel_value)
```

Output = (255, 0, 0)

In this way get the numbers for the given color pixel values

5.8 Calculate distance to get color name:

We have the r, g and b values. Now, we need another function which will return us the color name from RGB values. To get the color name, we calculate a distance (d) which tells us how close we are to color and choose the one having minimum distance.

Our distance is calculated by this formula:

d = abs (Red — ithRedColor) + (Green — ithGreenColor) + (Blue — ithBlueColor)

```
def getColorName(R,G,B):
    minimum = 10000
    for i in range(len(csv)):
        d = abs(R- int(csv.loc[i,"R"])) + abs(G-        int(csv.loc[i,"G"]))+ abs(B-
int(csv.loc[i,"B"]))
        if(d<=minimum):
            minimum = d
```

```
        cname = csv.loc[i,"color_name"]
    return cname
```

5.9 Display image on the window:

Whenever a double click event occurs, it will update the color name and RGB values on the window.

Using the cv2.imshow () function, we draw the image on the window. When the user double clicks the window, we draw a rectangle and get the color name to draw text on the window using cv2.rectangle and cv2.putText () functions.

```
while(1):cv2.imshow("color detection",img)
    if (clicked):
        cv2.rectangle(img,(20,20), (750,60), (b,g,r), -1)        text =
getColorName(r,g,b)+'R='+str(r)+'G='+ str(g)          +'B='+ str(b)
        cv2.putText(img, text,(50,50),2,0.8,
(255,255,255),2,cv2.LINE_AA)        if(r+g+b>=600):
            cv2.putText(img, text,(50,50),2,0.8,(0,0,0),2,cv2.LINE_AA)

        clicked=Falseif cv2.waitKey(20) & 0xFF ==27:
        break
cv2.destroyAllWindows()
```

cv2.imshow () method is used to display an image in a window. The window automatically fits to the image size. As Open CV is designed to solve the computer vision problems.

Syntax: cv2.imshow (window_name, image)

Parameters:

window_name: A string representing the name of the window in which image to be

displayed.

image: It is the image that is to be displayed.

Return Value: It doesn't returns anything.

destroyAllWindows() simply destroys all the windows we created. To destroy any specific window, use the function **cv2**. destroyWindow**()** where you pass the exact window name.

6.OBJECT DETECTION AND TRACKING USING HSV

6.1 Introduction:

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. For a robot to visualize the environment, along with the object detection, detection of its color in real-time is also very important. Detection and tracing of target is performed by color attribute using HSV color model. Here red, green and blue colors are bounded by their respective colored rectangular boxes and the name of the color is displayed on the top of the rectangular box.

Some real world applications:

- In self-driving car, to detect the traffic signals.
- Multiple color detection is used in some industrial robots, to performing pick-and-place task in separating different colored objects.

Steps for detection and tracking:

First step is to capture the video through webcam.This is taken as input .The image frames are accessed from the video stream.The HSV color model can handle changes caused by lighting.Hence the image stored in RGB format has to be transformed into HSV .This color model describes color in terms of the amount of gray and their brightness value.Hue value is extended from 0-179,Saturation value is extended from 0-255 and Value is extended from 0-255 respectively.The corresponding mask is made by defining the range of each color (red,green and blue).Noise causes internal imperfections,hence it has to be removed.Morpholoagical technique called dilation is used for this purpose.To specifically detect red,green and blue and discard others ,bitwise and is performed between the image frame and mask.All the points which are having same color or

intensity have to be boundedby a rectangular box .As the object with red, green or blue color advances in the liverecording, the bounding box also advances with it .Here the goal for detecting and tracing the red ,green and blue colored object is achieved.

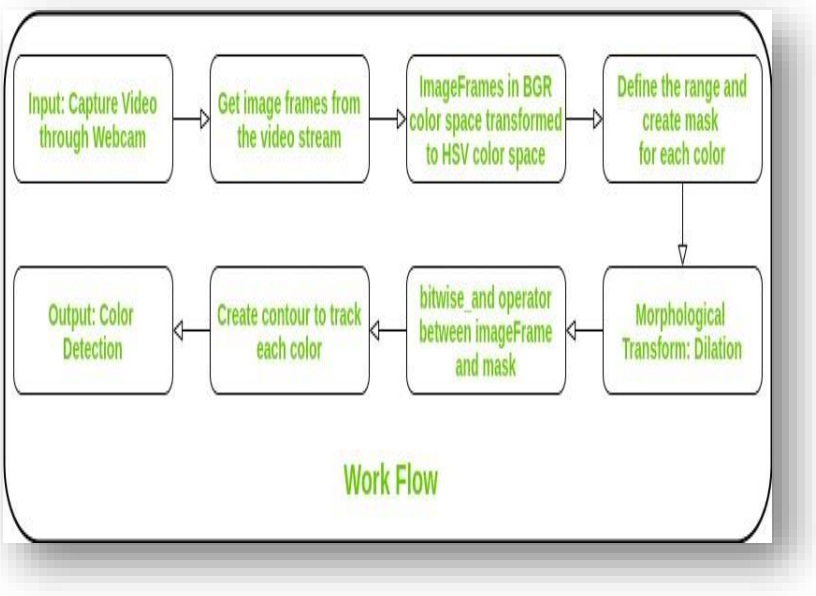


Figure 6.1 : Block diagram for object detection and tracking

6.2 Input: Capture video through webcam:

A video is a sequence of fast moving images. The measure of how fast the images are transitioning is given by a metric called **frames per second (FPS)**. If the video has an FPS of 40, it means that 40 images are being displayed every second. Alternatively, after every 25 milliseconds, a new frame is displayed. The other important attributes are the width and height of the frame.

6.2.1 Reading a Video:

In OpenCV, a video can be read either by using the feed from a camera connected to a computer or by reading a video file. The first step towards reading a video file is to create

a VideoCapture object. Its argument can be either the device index or the name of the video file to be read. In most cases, only one camera is connected to the system. So, all we do is pass '0' and OpenCV uses the only camera attached to the computer. When more than one camera is connected to the computer, we can select the second camera by passing '1', the third camera by passing '2' and so on.

We are going to apply the function cv2.VideoCapture and create a class instance. As an argument we can specify the input video file name. On the other hand, in order to access a video stream, we will put the camera parameters instead. We can provide an index of our camera from which we want to read data. In case that you only have one camera, by default, it will be indexed with *0*. If you have more than one camera the second will be named 1, third with 2 and so on. Let's show this in an example.

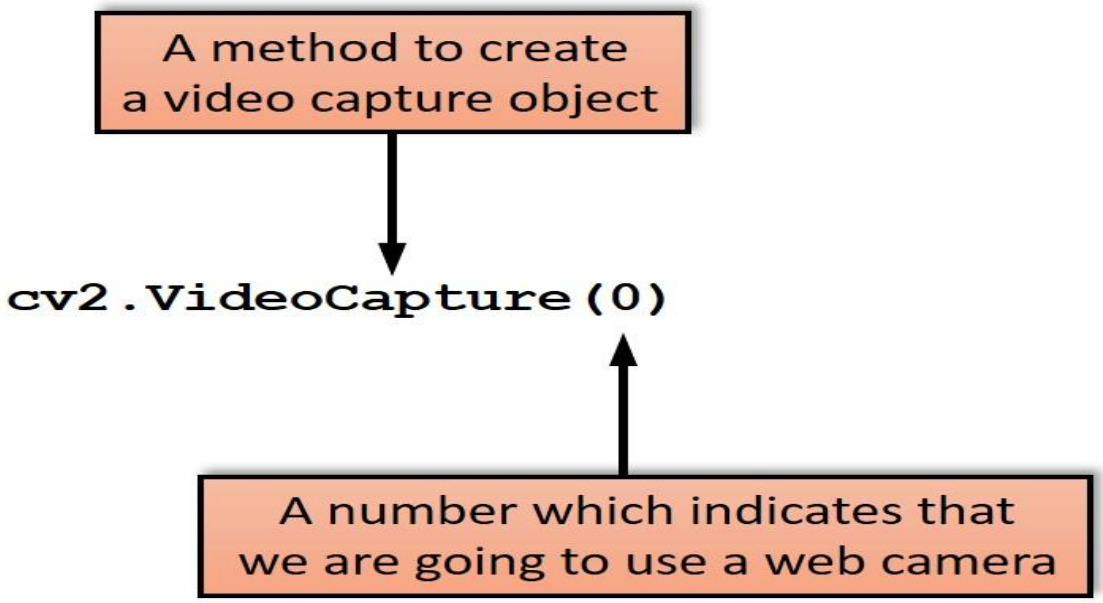


Figure 6.2 video capture

6.2.2 Displaying a video:

After reading a video file, we can display the video frame by frame. A frame of a video is simply an image and we display each frame the same way we display images, i.e., we use the function **imshow ()**.

As in the case of an image, we use the **waitKey ()** after imshow () function to pause each frame in the video. In the case of an image, we pass '0' to the waitKey () function, but for playing a video, we need to pass a number greater than '0' to the waitKey () function. This is because '0' would pause the frame in the video for an infinite amount of time and in a video we need each frame to be shown only for some finite interval of time. So, we need to pass a number greater than '0' to the waitKey () function. This number is equal to the time in milliseconds we want each frame to be displayed.

While reading the frames from a webcam, using waitKey (1) is appropriate because the display frame rate will be limited by the frame rate of the webcam even if we specify a delay of 1 ms in waitKey. While reading frames from a video that you are processing, it may still be appropriate to set the time delay to 1 ms so that the thread is freed up to do the processing we want to do. In rare cases, when the playback needs to be at a certain framerate, we may want the delay to be higher than 1 ms.

6.3. Read the video stream in image frames:

At the start of this process our indicator is on the first frame. When we apply command cap.read() the first frame from our video file will be loaded. It will be stored in a variable frame. If we call this command again, the second frame will be loaded and so on. Variable ret is a boolean data type that returns True if we are able to execute the read function successfully. Our frame can be loaded as a color image (it will have 3 channels) or grayscale image (it will have 1 channel). If we need more then one frame we will use "for" or "while" loops. We will explain that in more detail in the further text.

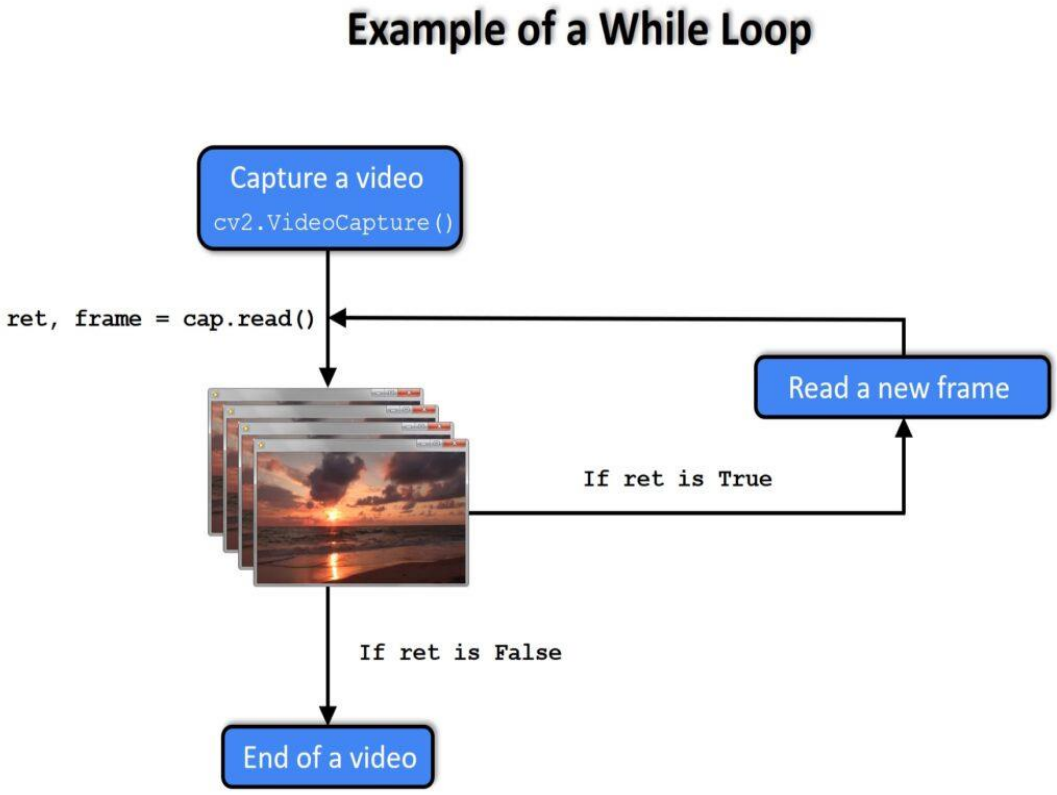


Figure 6.3 while loop

After we read a video file or capture a live stream, we want to process and display our video output. The following code creates a while loop that reads frames from our video continuously. We can do this with a command cap.read(). Our frame is stored in a frame variable and ret is boolean data type that returns *True* if Python is able to read the VideoCapture object. After we finish this process we can release our output with command cap.release().

6.4 Convert the imageFrame in BGR to HSV color space:

The HSV or Hue, Saturation and Value of a given object is the color space associated with the object in OpenCV where Hue represents the color, Saturation represents the greyness and Value represents the brightness and it is used to solve the problems related

to computer vision because of its better performance when compared to RGB or Red, Blue and Green color space and the Hue range in HSV is [0,179], the Saturation range in HSV is [0,255] and the Value range in HSV is [0,255] and to perform object detection, finding the range of HSV is necessary.

The syntax to define HSV range in OpenCV is as follows:

hsvcolorspace = cv.cvtColor(image, cv.COLOR_BGR2HSV)

lower_hsvcolorspace = np.array([Hue range, Saturation range, Value range])

upper_hsvcolorspace = np.array([Hue range, Saturation range, Value range])

Where hsvcolorspace is the conversion of the given image in RGB format to HSV format, lower_hsvcolorspace is the lower threshold for a range of some color, upper_hsvcolorspace is the upper threshold for a range of some color.

6.4.1 Working of HSV range in OpenCV:

- The HSV or Hue, Saturation, and value of a given object is the color space associated with the object in OpenCV.
- The Hue in HSV represents the color, Saturation in HSV represents the greyness, and Value in HSV represents the brightness.
- Whenever we want to solve problems related to object detection, it is necessary to use HSV and find the range of HSV.
- The Hue, Saturation, and Value in HSV have their own range of values.
- The Hue range in HSV is [0,179], the Saturation range in HSV is [0,255] and the Value range in HSV is [0,255].
- There is also an upper bound and lower bound range for a range of each color in HSV.
- The HSV or Hue, Saturation, and value of a given object provide better performance when compared to RGB or Red, Blue, and Green color space and hence it is used widely in the area of computer vision.

6.5. Masking:

Color segmentation or color filtering is widely used in OpenCV for identifying specific objects/regions having a specific color. The most widely used color space is RGB color space, it is called an **additive color space** as the three color shades add up to give color to the image. To identify a region of a specific color, put the threshold and create a mask to separate the different colors. HSV color space is much more useful for this purpose as the colors in HSV space are much more localized thus can be easily separated. Color Filtering has many applications and uses cases such as in Cryptography, infrared analysis, food preservation of perishable foods, etc. In such cases, the concepts of Image processing can be used to find out or extract out regions of a particular color. For color segmentation, all we need is the threshold values or the knowledge of the lower bound and upper bound range of colors in one of the color spaces. It works best in the Hue-Saturation-Value color space.

After specifying the range of color to be segmented, it is needed to create a mask accordingly and by using it, a particular region of interest can be separated out. Masking involves setting some of the pixel values in an image to zero, or some other "background" value. Masking can be done in one of two ways:

- Using an image as a mask. A mask image is simply an image where some of the pixel intensity values are zero, and others are non-zero. Wherever the pixel intensity value is zero in the mask image, then the pixel intensity of the resulting masked image will be set to the background value (normally zero). You might, for example, create a mask image using the Particle Analysis tool.
- Using a set of ROIs as the mask. The ROIs for each slice are used to define the mask.
- **Hard masking**. Pixels affected by masking have their intensity set to the background value.
- **Soft masking**. For pixels affected by masking, the resulting intensity change is depends on how much of the pixel is inside the ROI. For masking outside, if only

a small portion of the pixel is inside an ROI, the resulting will be close to the background value; if a large portion is inside the ROI, its intensity will be largely unaffected. Below you can see the result masking of an image using an Irregular ROI, with both hard and soft masking.

6.6 Morphological Transform:

Morphological transformations are some simple operations based on the image shape. It is normally performed on binary images. It is used to remove noise on the images. It needs two inputs, one is our original image, and second one is called **structuring element** or **kernel** which decides the nature of operation. Two basic morphological operators are Erosion and Dilation. Then its variant forms like Opening, Closing, Gradient etc. also comes into play.

6.6.1. Erosion:

The basic idea of erosion is just like soil erosion only, it erodes away the boundaries of foreground object (Always try to keep foreground in white). So what does it do? The kernel slides through the image (as in 2D convolution). A pixel in the original image (either 1 or 0) will be considered 1 only if all the pixels under the kernel is 1, otherwise it is eroded (made to zero).

So what happens is that, all the pixels near boundary will be discarded depending upon the size of kernel. So the thickness or size of the foreground object decreases or simply white region decreases in the image. It is useful for removing small white noises (as we have seen in colorspace chapter), detach two connected objects etc.

Syntax: erosion = cv2.erode (img,kernel,iterations = 1)

6.6.2. Dilation:

It is just opposite of erosion. Here, a pixel element is ‘1’ if at least one pixel under the kernel is ‘1’. So it increases the white region in the image or size of foreground object

increases. Normally, in cases like noise removal, erosion is followed by dilation. Because, erosion removes white noises, but it also shrinks our object. So we dilate it. Since noise is gone, they won't come back, but our object area increases. It is also useful in joining broken parts of an object.

Syntax: dilation = cv2.dilate (img,kernel,iterations = 1)

6.6.3 Opening:

Opening is just another name of **erosion followed by dilation**. It is useful in removing noise, as we explained above. Here we use the function, **cv2.morphologyEx ().**

Syntax: opening = cv2.morphologyEx (img, cv2.MORPH_OPEN, kernel)

6.6.4 Closing:

Closing is reverse of Opening, **Dilation followed by Erosion**. It is useful in closing small holes inside the foreground objects, or small black points on the object.

Syntax: closing = cv2.morphologyEx (img, cv2.MORPH_CLOSE, kernel)

6.6.5 Morphological Gradient:

It is the difference between dilation and erosion of an image. The result will look like the outline of the object.

Syntax: gradient = cv2.morphologyEx (img, cv2.MORPH_GRADIENT, kernel)

6.6.6 Top Hat:

It is the difference between input image and Opening of the image. Below example is done for a 9x9 kernel.

Syntax: tophat = cv2.morphologyEx (img, cv2.MORPH_TOPHAT, kernel)

6.6.7 Black Hat:

It is the difference between the closing of the input image and input image.

Syntax: blackhat = cv2.morphologyEx (img, cv2.MORPH_BLACKHAT, kernel)

6.6.8 Structuring Element:

We manually created a structuring elements in the previous examples with help of Numpy. It is rectangular shape. But in some cases, you may need elliptical/circular shaped kernels. So for this purpose, OpenCV has a function, **cv2.getStructuringElement ( )**. You just pass the shape and size of the kernel, you get the desired kernel.

6.7. Bitwise_and between the image frame and mask is performed to it and others are discarded:

Bitwise operations can be used in image manipulations. These bitwise techniques are used in many computer vision applications like for creating masks of the image, adding watermarks to the image and it is possible to create a new image using these bitwise operators. These operations work on the individual pixels in the image to give accurate results compared with other morphing techniques in OpenCV.

We can create a square, circle & rectangle with white pixels and a background with black pixels using these we will differentiate each bitwise function individually.

6.7.1.In this we use bitwise AND operator:

- **Bitwise AND:**

  This function calculates the conjunction of pixels in both images. This operation only considers pixels that are common with image 1 and image 2 and remaining pixels are removed from the output image.

```
bit-and = cv2.bitwise_and(img1,img2)
cv2_imshow(bit-and)
```

The AND function only the intersected regions in both images are displayed.

6.8. Create contour to track each color:

6.8.1 Contours:

Contours can be explained simply as a curve joining all the continuous points (along the boundary), having same color or intensity. The contours are a useful tool for shape analysis and object detection and recognition.

- For better accuracy, use binary images. So before finding contours, apply threshold or canny edge detection.
- Since OpenCV 3.2, **findContours()** no longer modifies the source image.
- In OpenCV, finding contours is like finding white object from black background. So remember, object to be found should be white and background should be black.

6.8.2 to draw the contours:

To draw the contours, **cv.drawContours** function is used. It can also be used to draw any shape provided you have its boundary points. Its first argument is source image, second argument is the contours which should be passed as a Python list, third argument is index of contours (useful when drawing individual contour. To draw all contours, pass -1) and remaining arguments are color, thickness etc.

- To draw all the contours in an image:
- cv.drawContours(img, contours, -1, (0,255,0), 3)
- To draw an individual contour, say 4th contour:
- cv.drawContours(img, contours, 3, (0,255,0), 3)
- But most of the time, below method will be useful:
  cnt = contours[4]
- cv.drawContours(img, [cnt], 0, (0,255,0), 3)

6.8.3 Contour approximation method:

If you pass **cv.CHAIN_APPROX_NONE**, all the boundary points are stored. But actually do we need all the points? For eg, you found the contour of a straight line. Do you need all the points on the line to represent that line? No, we need just two end points of that line. This is what **cv.CHAIN_APPROX_SIMPLE** does. It removes all redundant points and compresses the contour, thereby saving memory.

Below figure 6.4 demonstrate this technique. Just draw a circle on all the coordinates in the contour array (drawn in blue color). First image shows points I got with **cv.CHAIN_APPROX_NONE** (734 points) and second image shows the one with **cv.CHAIN_APPROX_SIMPLE** (only 4 points). See, how much memory it saves!!!

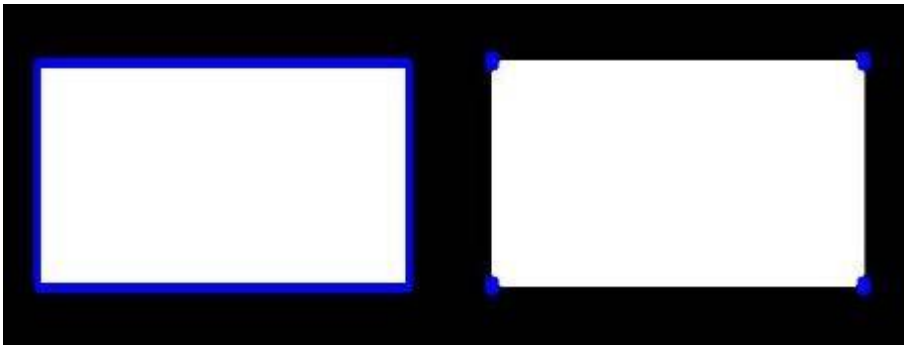


Figure 6.4

6.8.4 Bounding boxes:

Bounding boxes are one of the most popular—and recognized tools when it comes to image processing for image and video annotation projects. A bounding box is an imaginary rectangle that serves as a point of reference for object detection and creates a collision box for that object.

Image processing is one of the main reasons why computer vision continues to improve and drive innovative AI-based technologies. From self-driving cars to facial recognition technology—computer vision applications are the face of new tech. Data annotators draw these rectangles over images, outlining the object of interest within each image by defining its X and Y coordinates. This makes it easier for machine learning algorithms to find what they're looking for, determine collision paths, and conserves valuable computing resources. Bounding boxes are one of the most popular image annotation techniques in deep learning. Compared to other image processing methods, this method can reduce costs and increase annotation efficiency.

6.8.5 Using Bounding Boxes for Object Detection:

But how does object detection work in relation to bounding boxes for this we require looking at object detection as two components: object classification and object localization. In other words, to detect an object in an image, the computer needs to know what it is and where it is.

Take self-driving cars as an example. An annotator will draw bounding boxes around other vehicles and label them. This helps train an algorithm to understand what vehicles look like. Annotating objects such as vehicles, traffic signals, and pedestrians makes it possible for autonomous vehicles to maneuver busy streets safely. Self-driving car perception models rely heavily on bounding boxes to make this possible.

However, it's important to note that a single bounding box doesn't guarantee a perfect prediction rate. Enhanced object detection requires many bounding boxes in combination with data augmentation techniques.

6.9 CODINGS AND OUTPUT

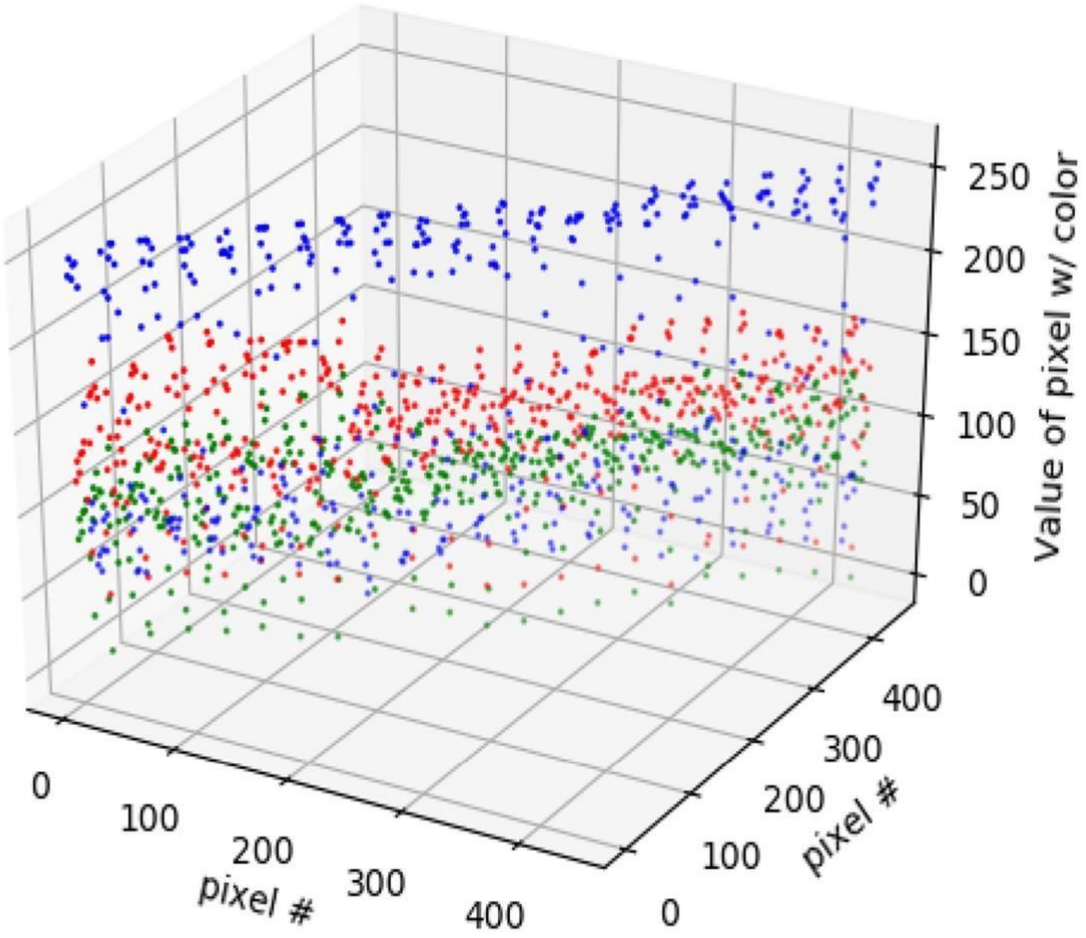
6.9.1 Colour Recognition Function

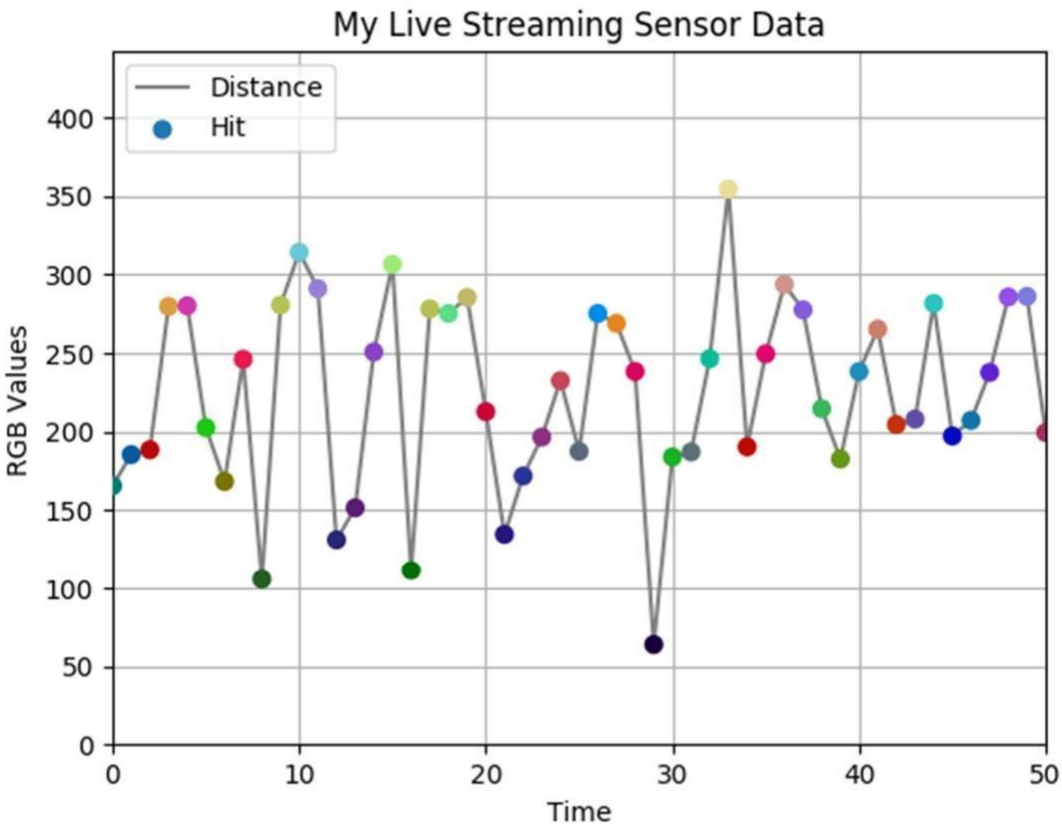
```
# Create function to calculate minimum distance from all colors and get the most matching color
def get_color_name(R,G,B):
    minimum = 1000
    for i in range(len(df)):
        d = abs(R - int(df.loc[i,'R'])) + abs(G - int(df.loc[i,'G'])) + abs(B - int(df.loc[i,'B']))
        if d <= minimum:
            minimum = d
            cname = df.loc[i, 'color_name']

    return cname
```

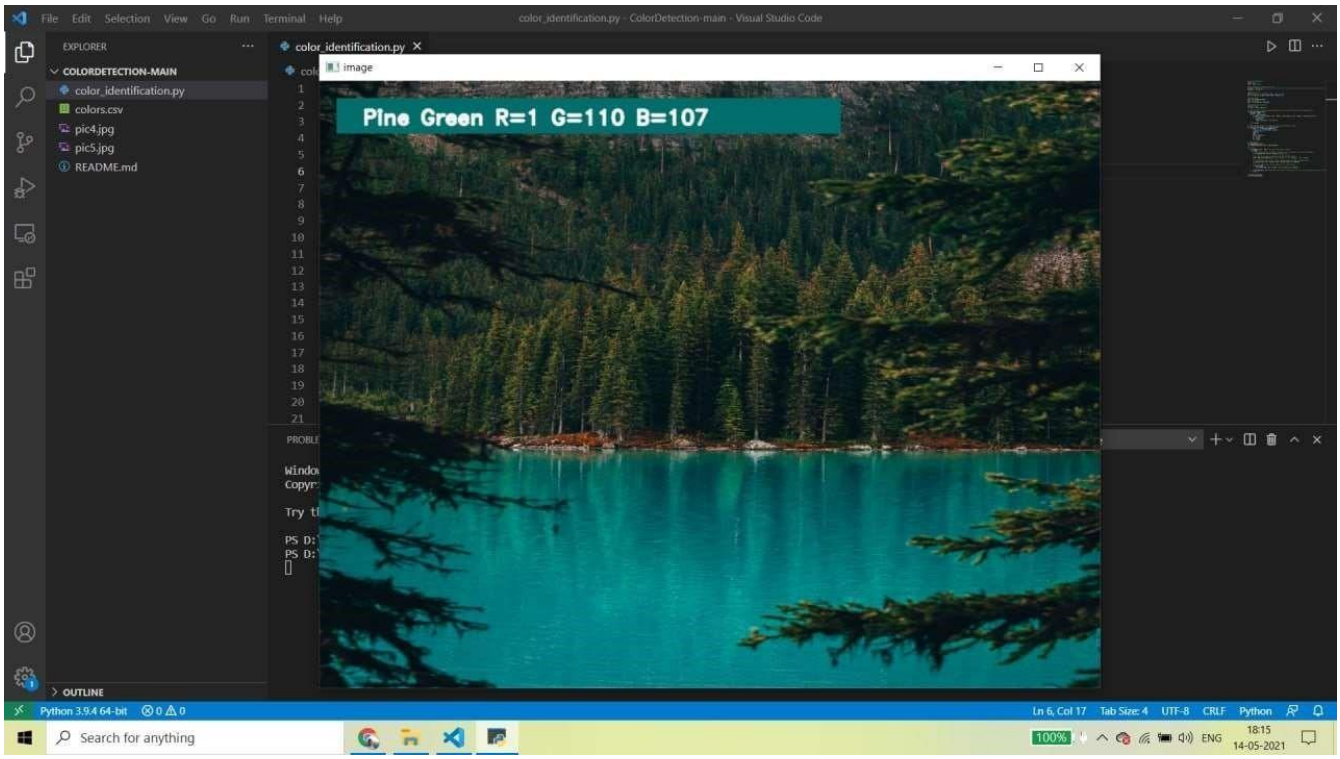
6.9.2 Mouse Click Function

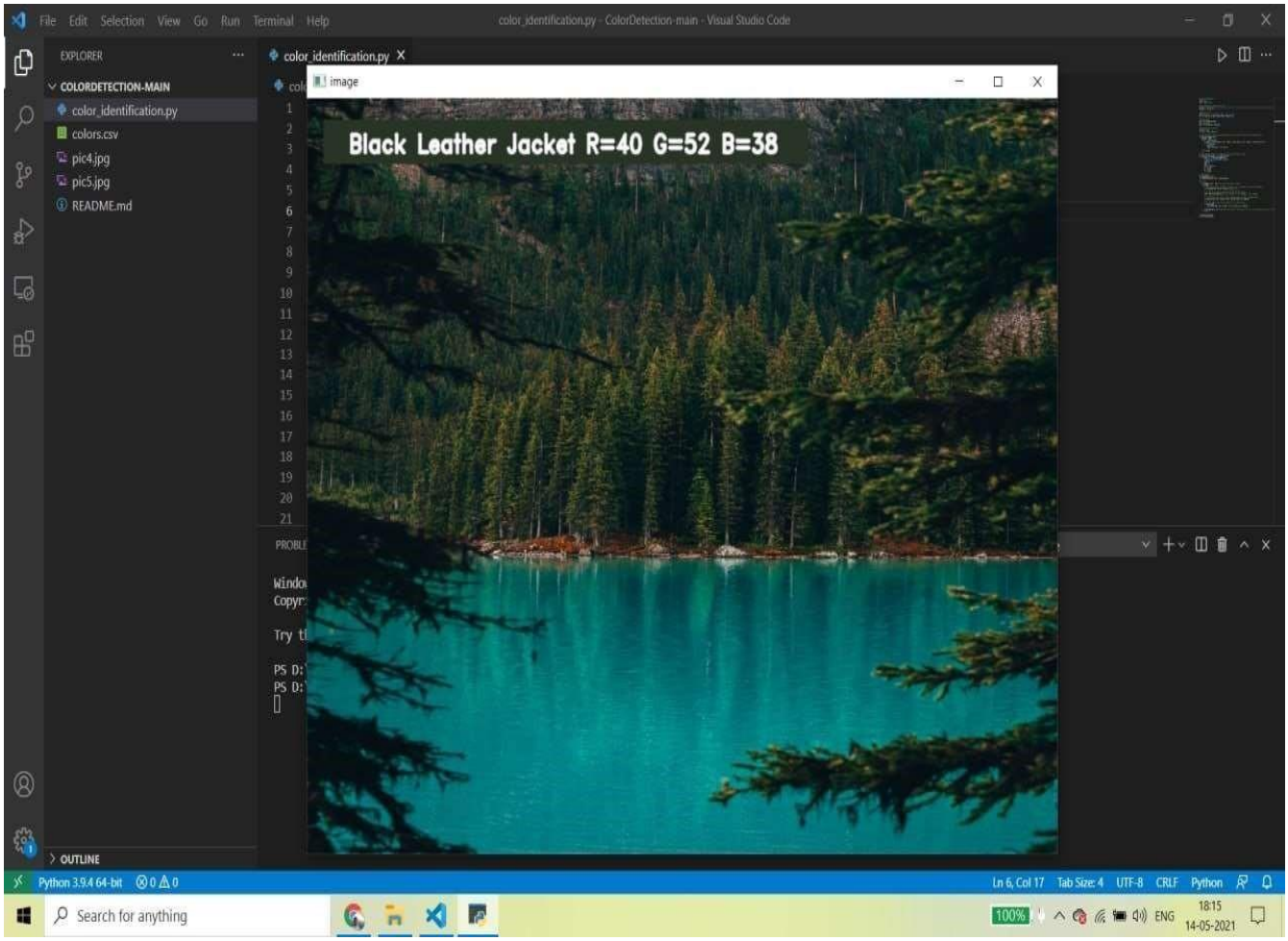
```
# Create function to get x,y coordinates of mouse when double clicked
def draw_function(event, x, y, flags, params):
    if event == cv2.EVENT_LBUTTONDBLCLK:
        global b, g, r, xpos, ypos, clicked
        clicked = True
        xpos = x
        ypos = y
        b,g,r = img[y,x]
        b = int(b)
        g = int(g)
        r = int(r)
```

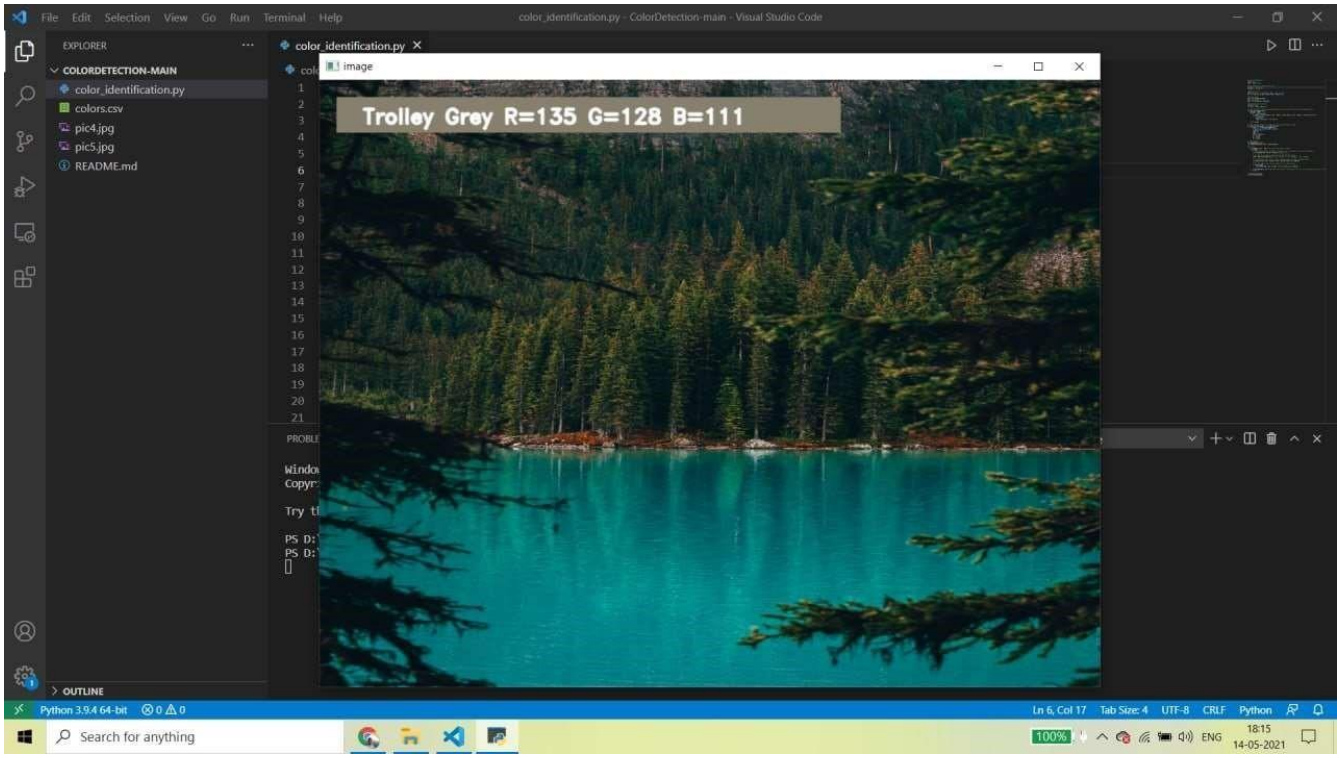

250
200
150
100
50
0
Value of pixel w/ color
0
100
200
300
400
pixel #
0
100
200
300
400
pixel #

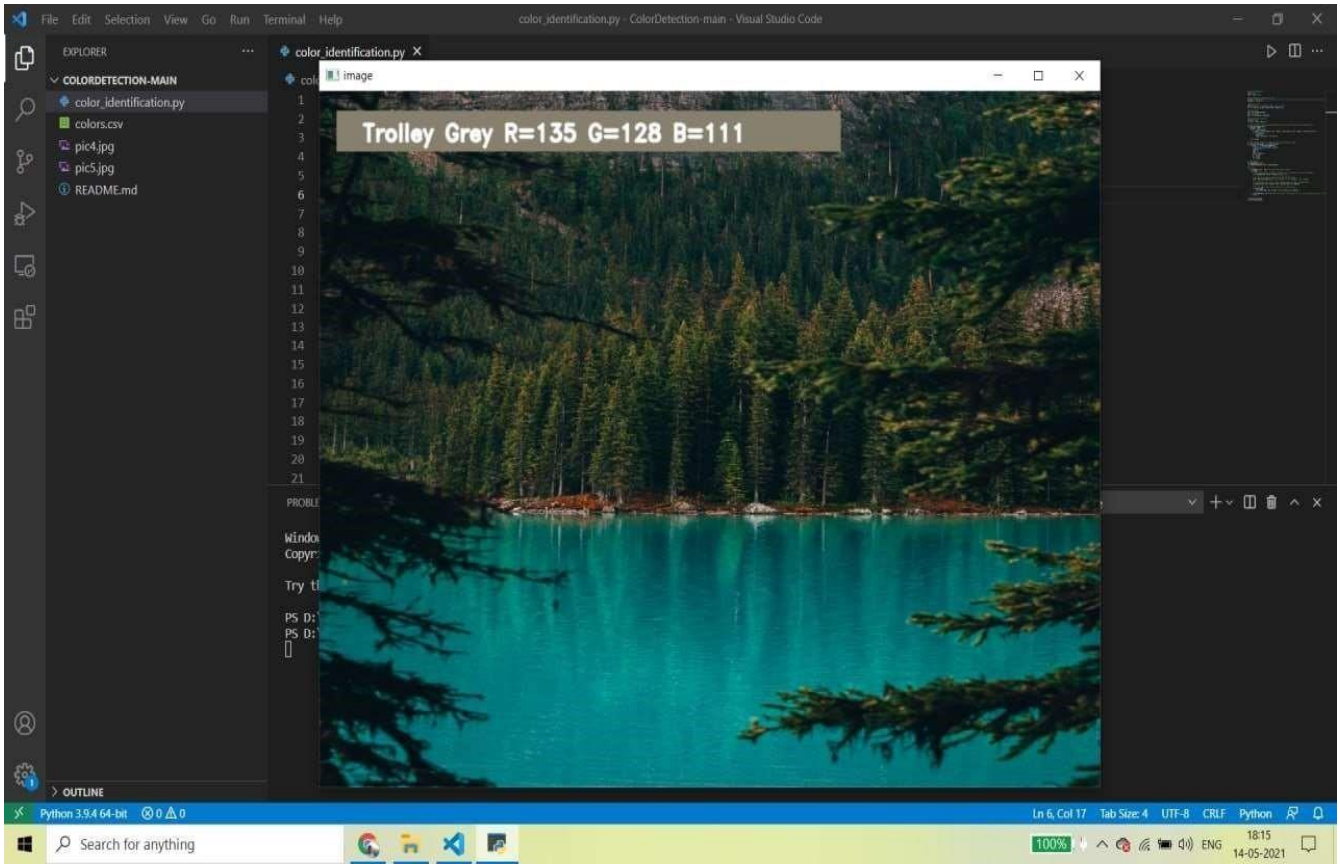

My Live Streaming Sensor Data
Distance
Hit
RGB Values
Time
0
50
100
150
200
250
300
350
400
0
10
20
30
40
50

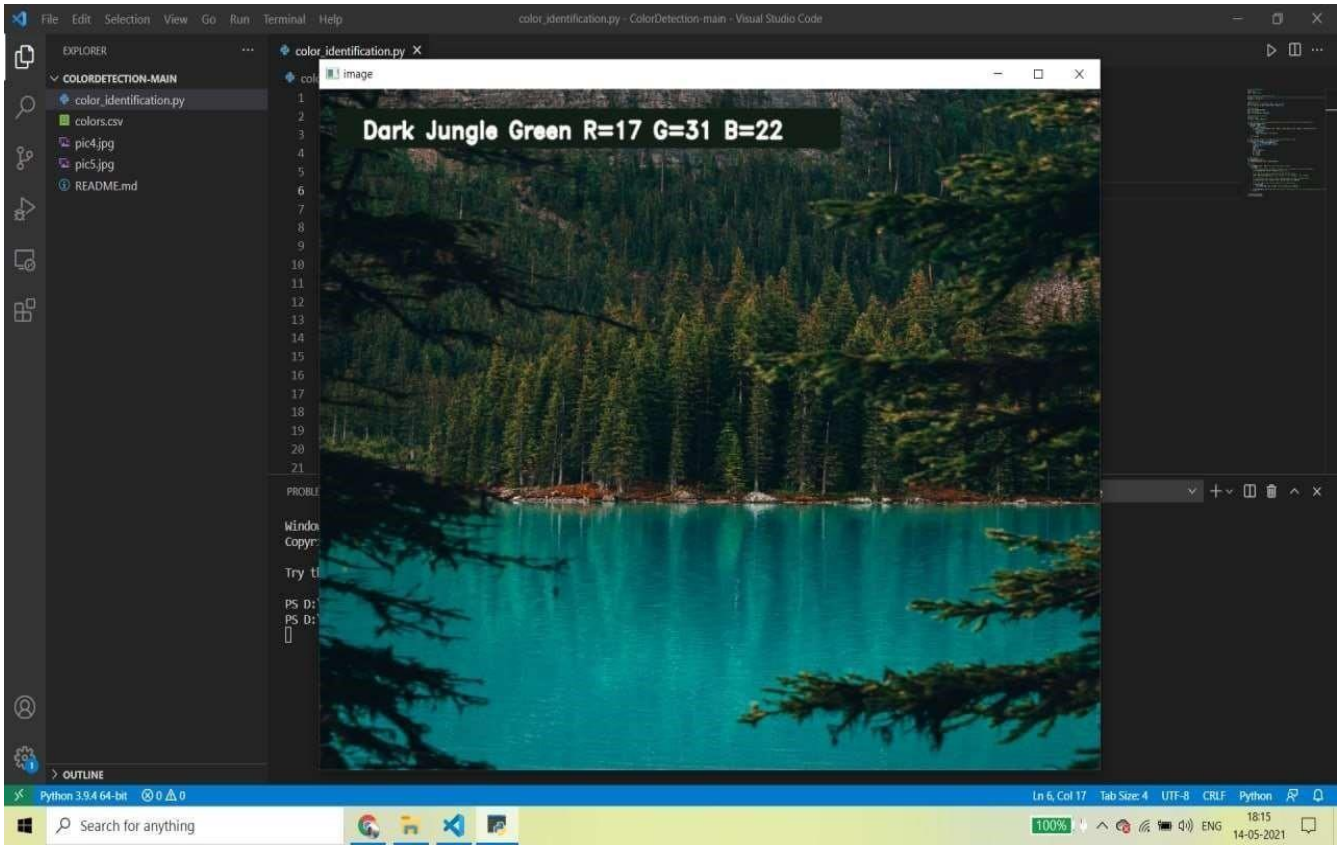
6.9.3 Outputs:

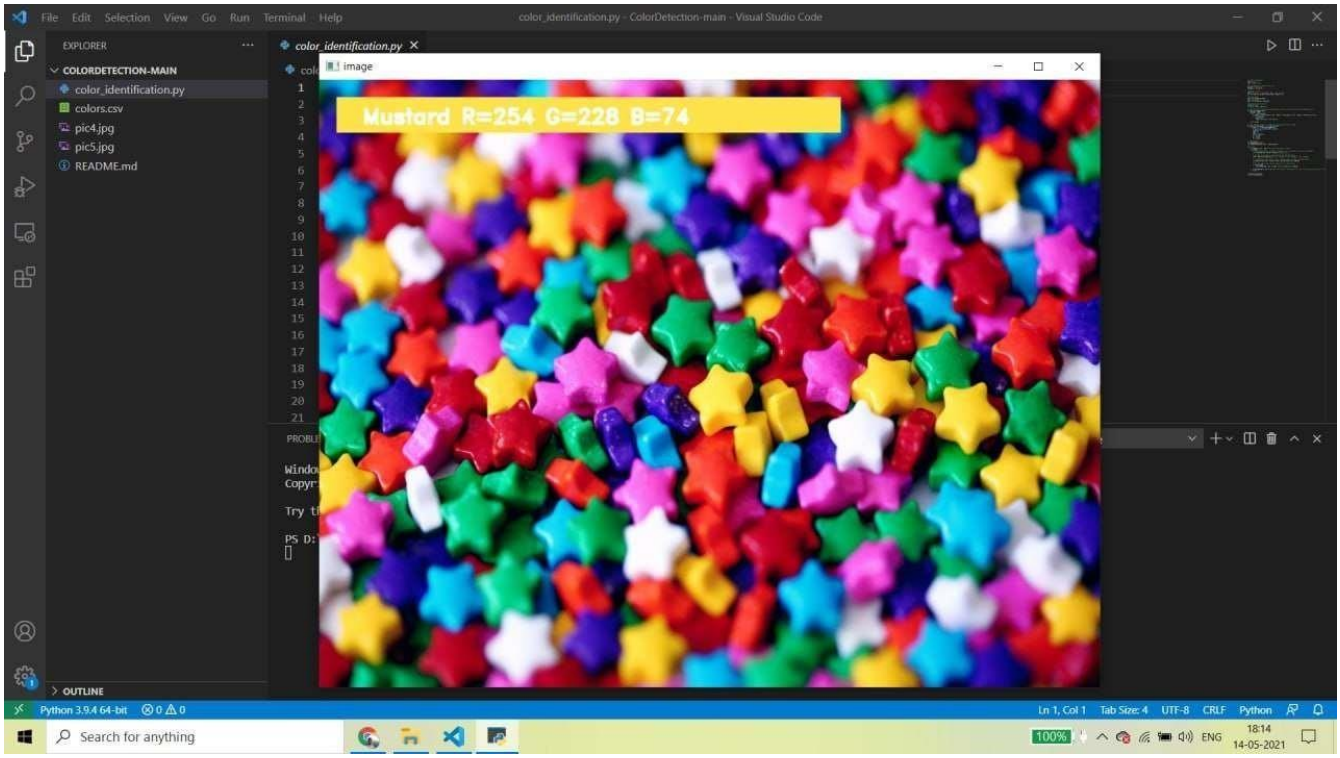

File Edit Selection View Go Run Terminal Help
color_identification.py - ColorDetection-main - Visual Studio Code
EXPLORER
COLORDETECTION-MAIN
color_identification.py
colors.csv
pic4.jpg
pic5.jpg
README.md
color_identification.py
image
Pine Green R=1 G=110 B=107
OUTLINE
Python 3.9.4 64-bit
Ln 6, Col 17
Tab Size: 4
UTF-8
CRLF
Python
Search for anything
100%
ENG
18:15
14-05-2021

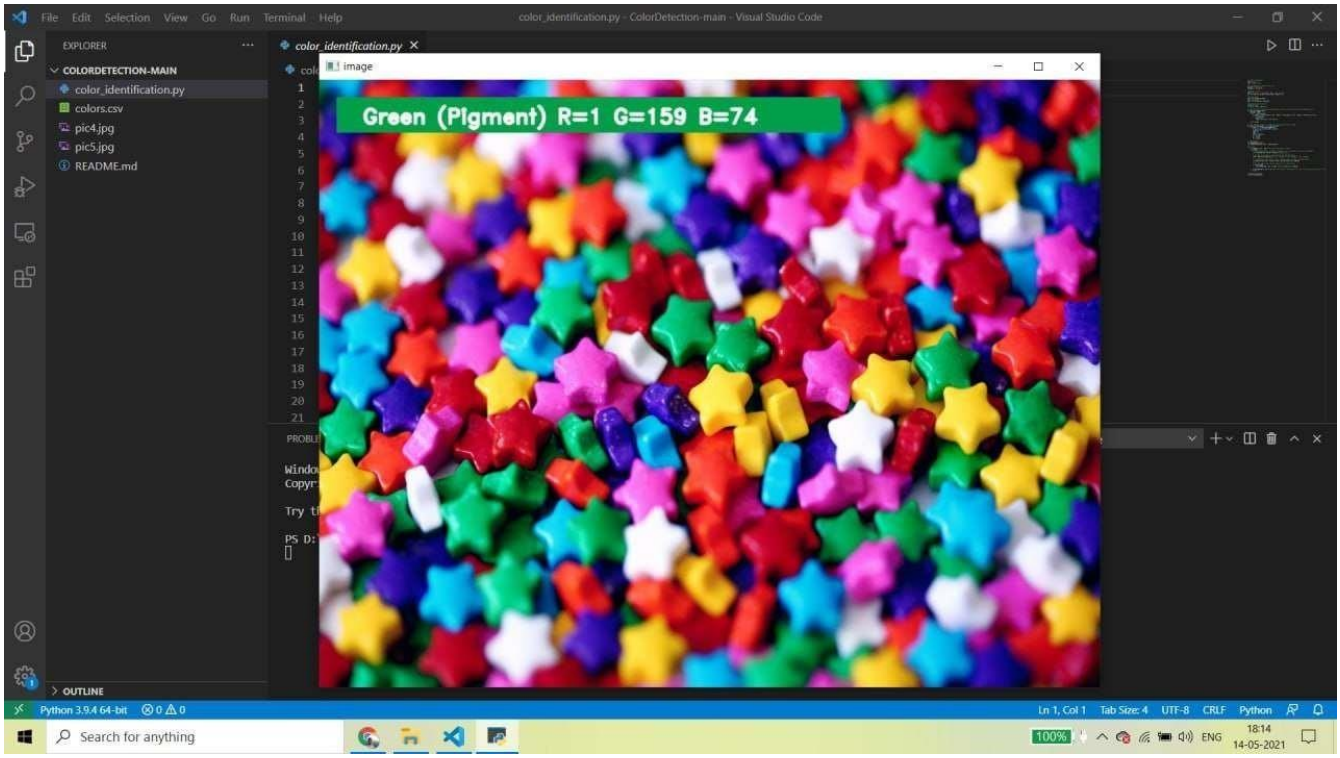

File Edit Selection View Go Run Terminal Help
color_identification.py - ColorDetection-main - Visual Studio Code
EXPLORER
COLORDETECTION-MAIN
color_identification.py
colors.csv
pic4.jpg
pic5.jpg
README.md
color_identification.py
image
Black Leather Jacket R=40 G=52 B=38
OUTLINE
Python 3.9.4 64-bit
Ln 6, Col 17
Tab Size: 4
UTF-8
CRLF
Python
Search for anything
100%
ENG
18:15
14-05-2021

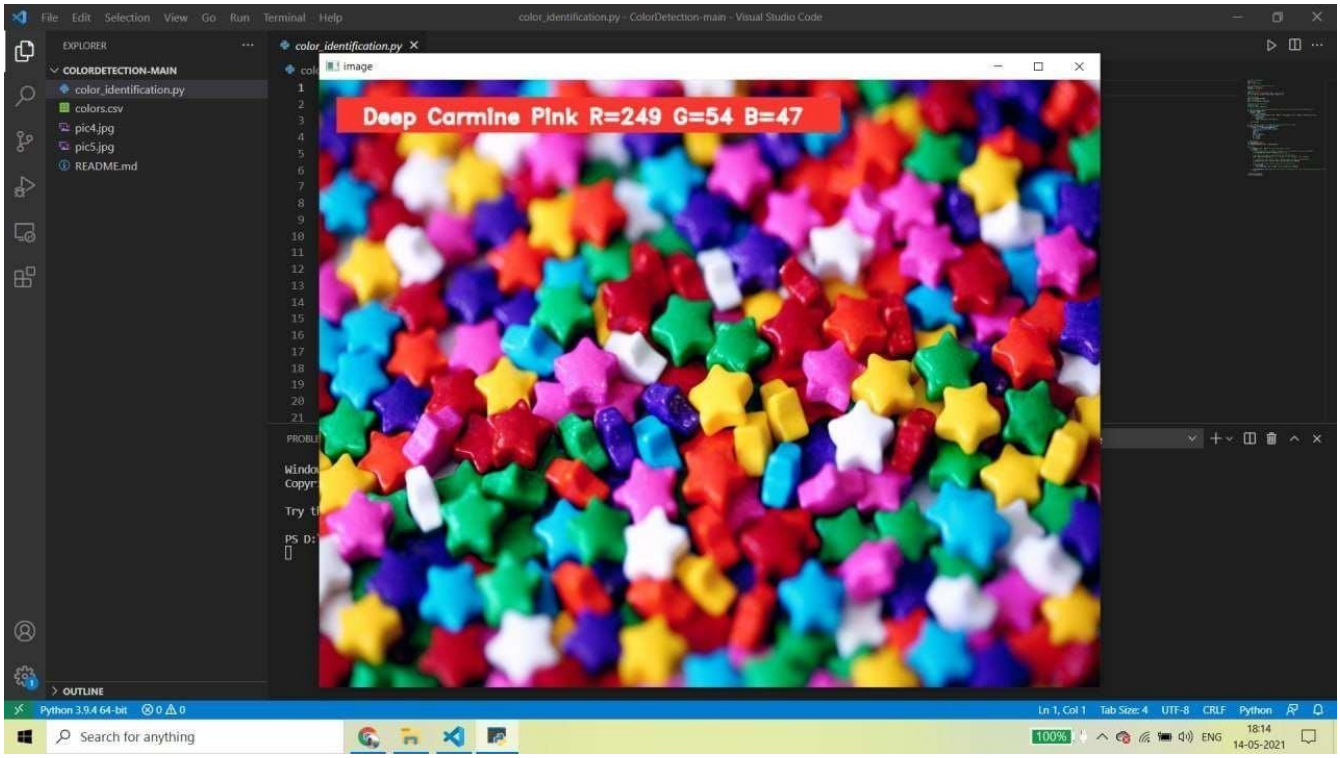

Trolley Grey R=135 G=128 B=111
color_identification.py - ColorDetection-main - Visual Studio Code
image
COLORDETECTION-MAIN
color_identification.py
colors.csv
pic4.jpg
pic5.jpg
README.md
OUTLINE
Search for anything

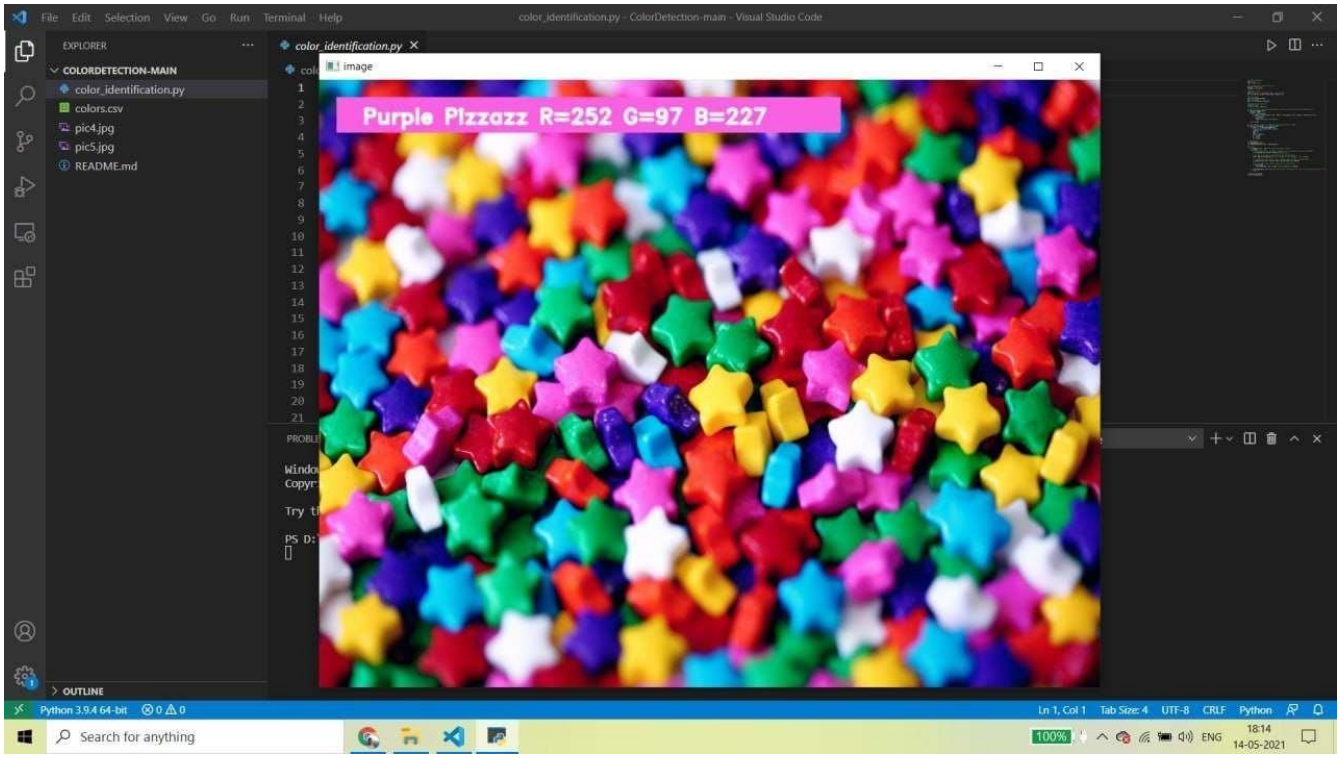

File Edit Selection View Go Run Terminal Help
color_identification.py - ColorDetection-main - Visual Studio Code
EXPLORER
COLORDETECTION-MAIN
color_identification.py
colors.csv
pic4.jpg
pic5.jpg
README.md
color_identification.py
image
Trolley Grey R=135 G=128 B=111
OUTLINE
Python 3.9.4 64-bit
Ln 6, Col 17
Tab Size: 4
UTF-8
CRLF
Python
Search for anything
100%
ENG
18:15
14-05-2021


File Edit Selection View Go Run Terminal Help
color_identification.py - ColorDetection-main - Visual Studio Code
EXPLORER
COLORDETECTION-MAIN
color_identification.py
colors.csv
pic4.jpg
pic5.jpg
README.md
color_identification.py
image
Dark Jungle Green R=17 G=31 B=22
OUTLINE
Python 3.9.4 64-bit
Ln 6, Col 17
Tab Size: 4
UTF-8
CRLF
Python
Search for anything
100%
ENG
18:15
14-05-2021


File Edit Selection View Go Run Terminal Help
color_identification.py - ColorDetection-main - Visual Studio Code
EXPLORER
COLORDETECTION-MAIN
color_identification.py
colors.csv
pic4.jpg
pic5.jpg
README.md
color_identification.py
image
Mustard R=254 G=228 B=74
OUTLINE
Python 3.9.4 64-bit
Ln 1, Col 1
Tab Size: 4
UTF-8
CRLF
Python
Search for anything
100%
ENG
18:14
14-05-2021


File Edit Selection View Go Run Terminal Help
color_identification.py - ColorDetection-main - Visual Studio Code
EXPLORER
COLORDETECTION-MAIN
color_identification.py
colors.csv
pic4.jpg
pic5.jpg
README.md
color_identification.py
image
Green (Pigment) R=1 G=159 B=74
OUTLINE
Python 3.9.4 64-bit
Ln 1, Col 1
Tab Size: 4
UTF-8
CRLF
Python
Search for anything
100%
ENG
18:14
14-05-2021


File Edit Selection View Go Run Terminal Help
color_identification.py - ColorDetection-main - Visual Studio Code
EXPLORER
COLORDETECTION-MAIN
color_identification.py
colors.csv
pic4.jpg
pic5.jpg
README.md
color_identification.py
image
Deep Carmine Pink R=249 G=54 B=47
OUTLINE
Python 3.9.4 64-bit
Ln 1, Col 1
Tab Size: 4
UTF-8
CRLF
Python
Search for anything
100%
ENG
18:14
14-05-2021


File Edit Selection View Go Run Terminal Help
color_identification.py - ColorDetection-main - Visual Studio Code
EXPLORER
COLORDETECTION-MAIN
color_identification.py
colors.csv
pic4.jpg
pic5.jpg
README.md
color_identification.py
image
Purple Pizzazz R=252 G=97 B=227
OUTLINE
Python 3.9.4 64-bit
Ln 1, Col 1
Tab Size: 4
UTF-8
CRLF
Python
Search for anything
100%
ENG
18:14
14-05-2021

7. Results and Discussions:

Figure 7.1 is the input image in which we have to detect the color. In this input image you can double click at any position to get the name of the color and their respective R, G, B values. Figure 7.2 is the output image with color intensity RGB values as R=238, G=206, B=84 for sandstorm color. Figure 7.3 is the output image with color intensity RGB values as R=199, G=67, B= 73 for Brick Red color.

Figure 7.1 Original input image of Buildings

Figure 7.2 Output image with Color intensity RGB values as R=238 G=206 B=84 for Sandstorm

Figure 7.3 Output image with Color intensity RGB values as R=199 G=67 B=73 for

Brick Red

Figure 7.4 shows the detection and tracing of red, blue and green colored target. Detection and tracing of target is performed by color attribute using HSV color model. Here red, green and blue colors are bounded by their respective colored rectangular boxes and the name of the color is displayed on the top of the rectangular box.

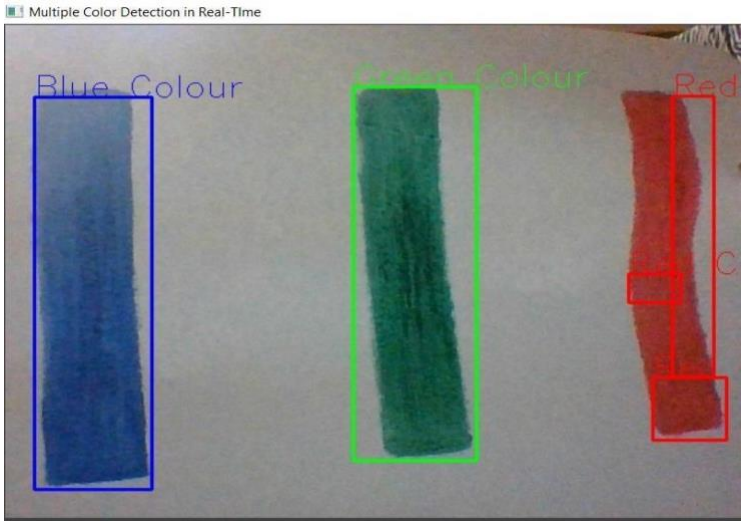


Figure 7.4 Object detection and tracing using red, green and blue color

8.Conclusion

By employing the proposed technique for object detection and tracing, we can directly acquire the label of the color and their R, G, B values just by clicking on the image which is provided as input. It also tracks three different colors (red, green and blue) using webcam. When there is any color from red, green, blue, or all the three at the same time, rectangular boxes of red color for tracking of red, blue rectangular box for tracking of blue, green rectangular box for tracking of green are formed and the name of the color is displayed on the top of the object.

9.References

[1] L. Feng, L. Xiaoyu and C. Yi, "An efficient detection method for rare colored capsule based on RGB and HSV color space," *2014 IEEE International Conference on Granular Computing (GrC)*, 2014, pp. 175-178, doi: 10.1109/GRC.2014.6982830.

[2] G. Pavithra, J. J. Jose and T. A. Chandrappa, "Real-time color classification of objects from video streams," *2017 2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*, 2017, pp. 1683-1686, doi: 10.1109/RTEICT.2017.8256886.

[3] S. Matuska, R. Hudec and M. Benco, "The comparison of CPU time consumption for image processing algorithm in Matlab and OpenCV," *2012 ELEKTRO*, 2012, pp. 75-78, doi: 10.1109/ELEKTRO.2012.6225575.

[4] Vishesh Goel, Sahil Singhal, Silica Kole “Specific Color Detection in Images using RGB Modelling in MATLAB” International Journal of Computer Applications (0975 – 8887) Volume 161 – No 8, March 2017.

[5] K. Cameron and M. S. Islam, "Multiple Objects Detection using HSV," *NAECON 2018 - IEEE National Aerospace and Electronics Conference*, 2018, pp. 270-273, doi: 10.1109/NAECON.2018.8556711.

[6] H. Altun, R. Sinekli, U. Tekbas, F. Karakaya and M. Peker, "An efficient color detection in RGB space using hierarchical neural network structure," *2011 International Symposium on Innovations in Intelligent Systems and Applications*, 2011, pp. 154-158, doi: 10.1109/INISTA.2011.5946088.

[7] M. M. Aznaveh, H. Mirzaei, E. Roshan and M. Saraee, "A new color-based method for skin detection using RGB vector space," *2008 Conference on Human System Interactions*, 2008, pp. 932-935, doi: 10.1109/HSI.2008.4581568.

[8] G. Li, R. Yuan, Z. Yang and X. Huang, "A Yellow License Plate Location Method Based on RGB Model of Color Image and Texture of Plate," *Second Workshop on Digital Media and its Application in Museum & Heritages (DMAMH 2007)*, 2007, pp. 42-46, doi: 10.1109/DMAMH.2007.35.