# Practical No.: 1

**Aim: Introduction to OpenCV New concept:**

1. **cv2:** it is the name of the module that belongs to OpenCV (Open Source Computer Vision Library). OpenCV is a popular library used for image and video processing, as well as machine learning applications related to computer vision. The cv2 module provides a range of functions for handling images, videos, and various computer vision tasks.
2. **imread:** it is used to read an image from a specified file path. It loads the image into a format that can be processed by OpenCV, typically as a NumPy array.
3. **cvtColor:** it is used to convert an image from one color space to another. It is used for converting between formats like RGB (Red, Green, Blue) and grayscale, or from BGR (Blue, Green, Red) to RGB, as OpenCV uses BGR by default.

**iv: imshow:** it is used to display an image in a window. It is used to show the result of some image processing operations or to visualize an image you’ve loaded.

1. **subplot:** is a function from the matplotlib library (not OpenCV) that allows you to create a grid of subplots within a single figure. We can display multiple images or plots in the same window using different subplots.

**Theory:**

Computer Vision is a field of artificial intelligence (AI) that enables computers to interpret and understand the visual world. It involves techniques for processing and analyzing images, videos, and visual data to make decisions or generate insights. Computer vision tasks often rely on machine learning and deep learning methods to automate the interpretation of visual data.

Applications of Computer Vision (CV):

* 1. **Healthcare & Medical Imaging:** Used to detect diseases like cancer, analyze X-rays and MRIs, and assist in surgeries.
  2. **Autonomous Vehicles:** Helps self-driving cars detect objects, recognize lanes, and read traffic signs.
  3. **Retail & E-Commerce:** CV helps with inventory tracking, product recognition, and checkout-free shopping (e.g., Amazon Go).
  4. **Agriculture:** Used for crop monitoring, automated harvesting, and weed detection.
  5. **Surveillance & Security:** Includes facial recognition, anomaly detection, and automated security monitoring.

**Program:**

**Program 1: To show the image:**

import cv2

img = cv2.imread("car.jpg") cv2.imshow("Output Image", img)

**Program 2: To convert rgb image to grayscale image: Method 1:**

import cv2

img = cv2.imread("car.jpg")

gray\_image = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY) cv2.imshow("Original Image", img)

cv2.imshow("Grayscale Image", gray\_image)

**Method 2:**

import cv2

img = cv2.imread("car.jpg")

gray\_image = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY) cv2.imshow("Original Image", img)

converted\_image = cv2.imread("test.jpg", cv2.IMREAD\_GRAYSCALE) cv2.imshow("Converted Image", converted\_image)

**Program 3: To display multiple images on same screen:**

import cv2

import matplotlib.pyplot as plt img0 = cv2.imread("car.jpg")

img = cv2.cvtColor(img0, cv2.COLOR\_BGR2RGB) gray\_image = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY) #Image1

plt.figure(figsize = (10,5)) plt.subplot(1,2,2) plt.imshow(img) #plt.imshow(img, cmap = 'gray') plt.title('Original Image') plt.axis('off')

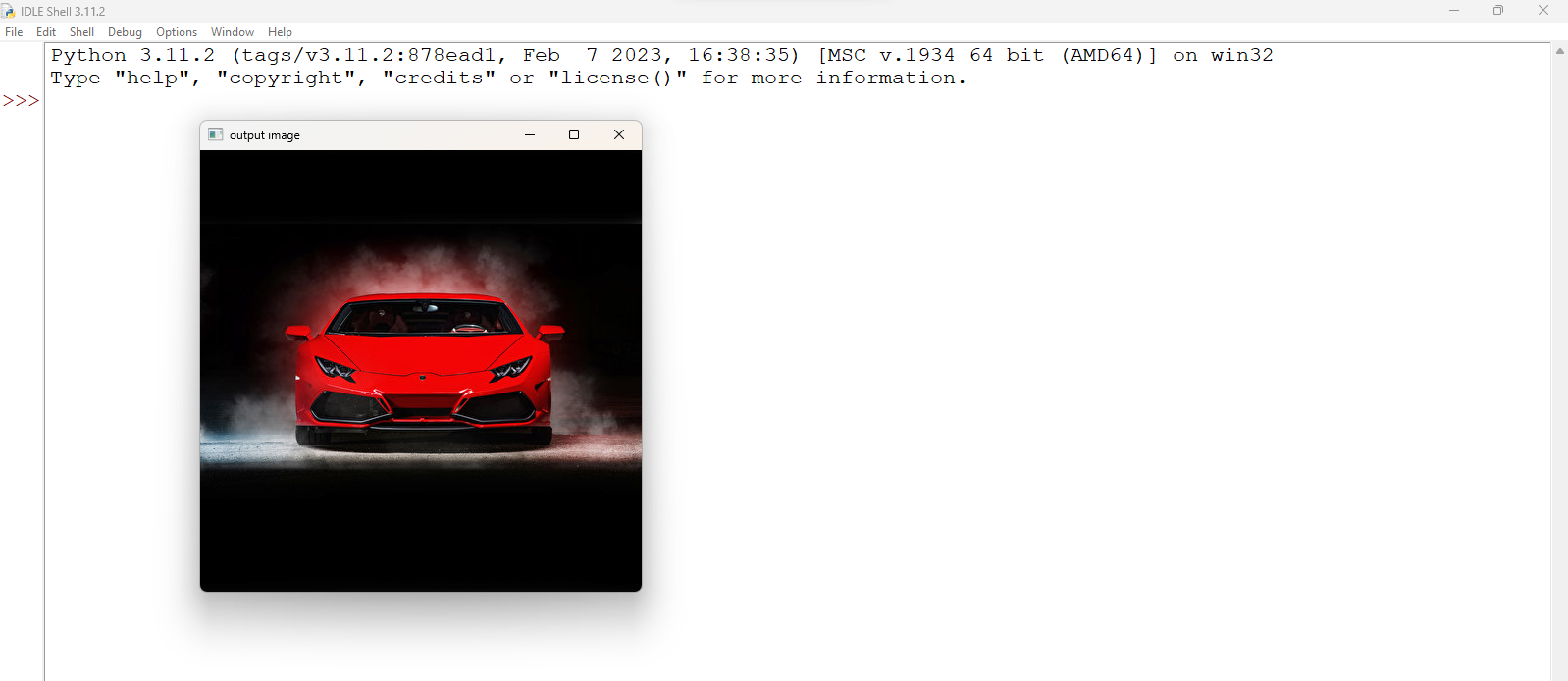
#Image2 plt.subplot(1,2,1) plt.imshow(gray\_image)

#plt.imshow(gray\_image, cmap = 'gray') plt.title('GrayScale Image')

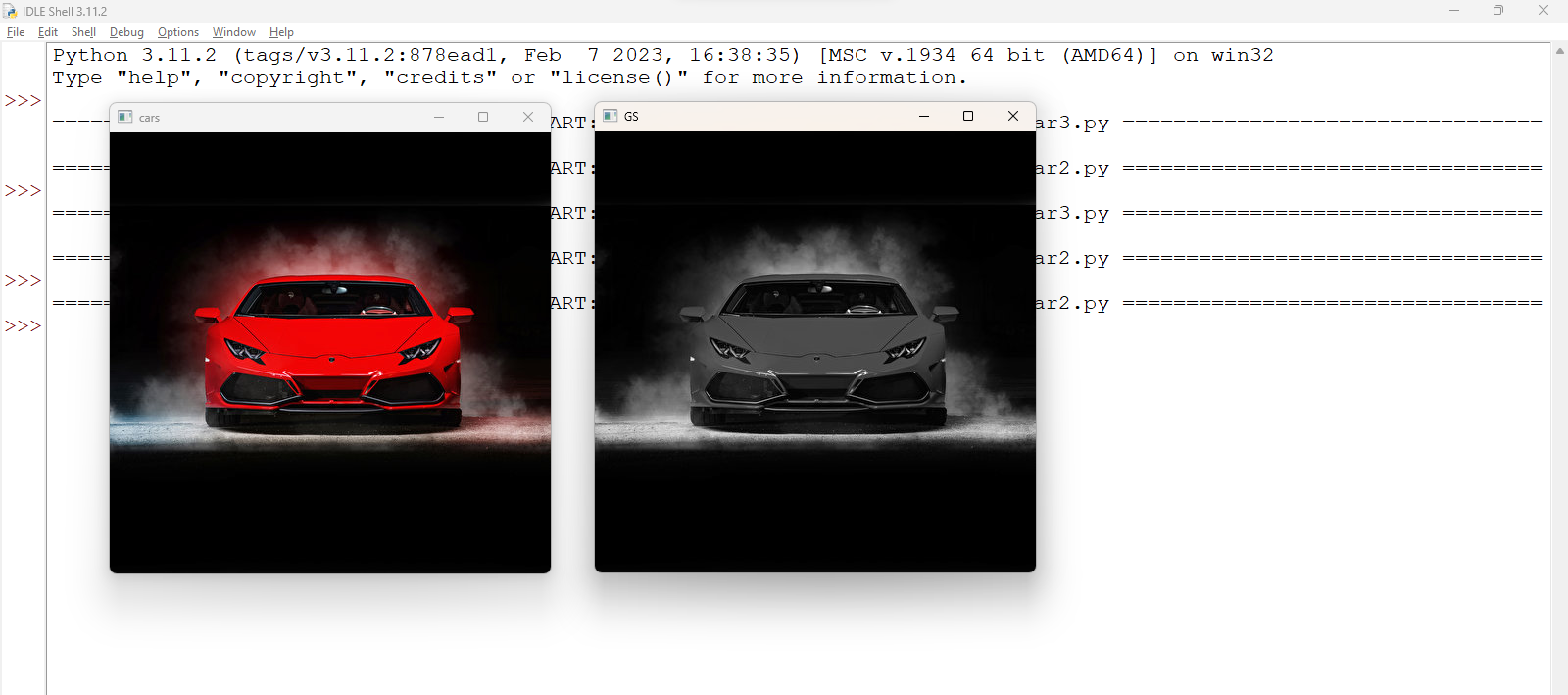
plt.axis('off') plt.show()

**Output:**

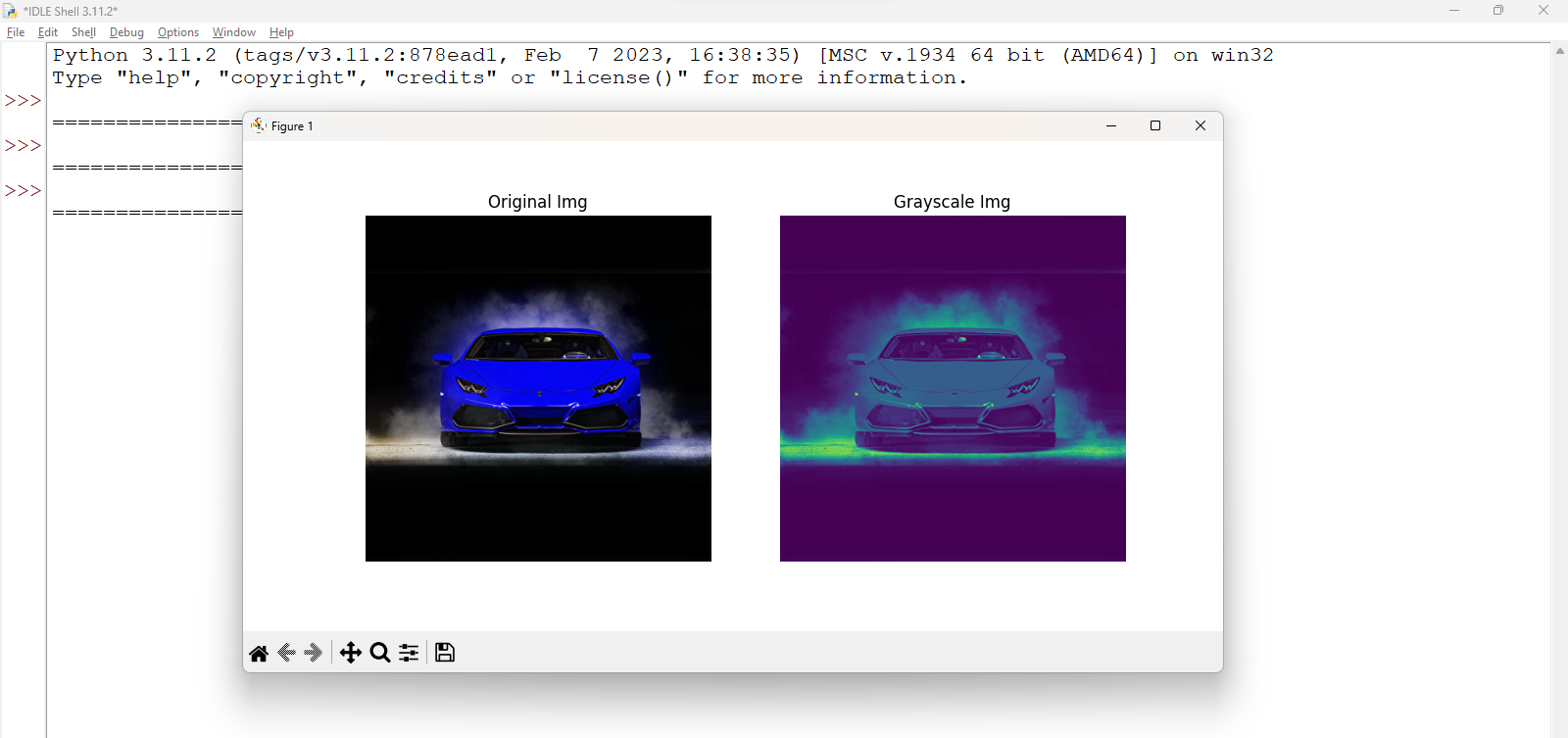
**Program 1:**



**Program 2:**

****

**Program 3:**

****

## Practical No.: 2

### Aim: Understanding geometric translation and histogram New Concept:

1. **np.clip:** it is a NumPy function that limits the values in an array to a specified range. Any values lower than the minimum are set to the minimum, and values higher than the maximum are set to the maximum.
2. **image.astype(float):** it is used to convert an image or array to a specific data type, in this case, a float. This is useful when you need more precision in calculations, such as for image transformations, processing, or normalization.
3. **np.uint8:** it is a NumPy data type that stands for unsigned 8-bit integer**.** It can store integer values from 0 to 255. This is the most common data type for images, where pixel values typically range from 0 to 255.
4. **warpAffine:** it is a function in OpenCV that applies an affine transformation to an image. An affine transformation preserves points, straight lines, and planes, but it can perform operations like rotation, scaling, translation, and shearing.
5. **calcHist**: it is a function in OpenCV that calculates the histogram of an image or a region of an image. A histogram is a graphical representation of the distribution of pixel intensities in an image.
6. **equalizeHist:** it is a function in OpenCV that performs histogram equalization on an image. This technique enhances the contrast of an image by spreading out the most frequent intensity values. It is used to improve the quality of images with poor contrast.

### Theory:

**Translation:**

Translation refers to the process of shifting an image or object in space along the horizontal (x-axis) and/or vertical (y-axis) direction without changing its content. The object or image is moved by a certain number of pixels or units in a given direction.

### Histogram:

A histogram in computer vision is a graphical representation of the distribution of pixel intensities in an image. It shows how many pixels in the image have each possible intensity value (from 0 to 255 for grayscale images).

* + **For grayscale images:** The x-axis of the histogram represents pixel intensity values (ranging from 0 for black to 255 for white), while the y-axis represents the frequency or count of pixels with those intensities.
  + **For color images:** The histogram is typically calculated separately for each color channel (Red, Green, and Blue).

### Program:

**Program1: To convert original image to brightened, dark, rotate and translate image:**

import cv2

from scipy import ndimage import matplotlib.pyplot as plt import numpy as np

#Load the image

image = cv2.imread("car.jpg")

#Increase Brightness

image\_float = image.astype(float)

# Increase the brightness (values greater than 1 between 2 to 5) brightness\_factor = 2.5 # Increase or decrease this value as needed brightened\_image = image\_float \* brightness\_factor

#Decrease the Brightness (values between 0 and 1) brightness\_factor1 = 0.4 # Increase or decrease this value as needed dark\_image = image\_float \* brightness\_factor1

# Clip the pixel values to the valid range [0, 255] brightened\_image = np.clip(brightened\_image, 0, 255)

dark\_image = np.clip(dark\_image, 0, 255)

# Convert the image back to unsigned 8-bit integers brightened\_image = brightened\_image.astype(np.uint8) dark\_image = dark\_image.astype(np.uint8)

# Image Rotation

image\_rotate = ndimage.rotate(image,45)

#Image translation

height, width = image.shape[:2] #Store height and Weight T = np.float32([[1, 0, 100], [0, 1, 200]])

img\_trans = cv2.warpAffine(image, T, (width, height))

#Show images

#cv2.imshow("Original Image",image) #cv2.imshow("Brightened image",brightened\_image) #cv2.imshow("Dark image",dark\_image) #cv2.imshow("Rotated image",image\_rotate) #cv2.imshow("Translated image",img\_trans)

plt.figure(figsize = (10,5))

#Image1 plt.subplot(2,3,1) plt.imshow(image) plt.title('Original Image')

plt.axis('off')

#Image2 plt.subplot(2,3,2)

plt.imshow(brightened\_image) plt.title('Brightened Image') plt.axis('off')

#Image3 plt.subplot(2,3,3) plt.imshow(dark\_image) plt.title('Dark Image') plt.axis('off')

#Image4 plt.subplot(2,3,4) plt.imshow(image\_rotate) plt.title('Rotated Image') plt.axis('off')

#Image5 plt.subplot(2,3,5) plt.imshow(img\_trans)

plt.title('Translated Image') plt.axis('off')

plt.show()

#cv2.waitKey(0) #cv2.destroyAllWindows()

### Program2: Histogram plotting:

import cv2

import matplotlib.pyplot as plt # Step 2: Read the image

image = cv2.imread("car.jpg")

# Step 3: Convert to grayscale (optional)

gray\_img = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY) # Step 4: Calculate the histogram

hist = cv2.calcHist([gray\_img], [0], None, [256], [0, 256]) # Step 5: Plot the histogram (optional)

plt.plot(hist)

plt.title('Histogram of Grayscale Image') plt.xlabel('Pixel Intensity') plt.ylabel('Frequency')

plt.show()

### Program3: Histogram equalization:

#import required Libraries import cv2

from scipy import ndimage import numpy as np

import matplotlib.pyplot as plt

#Load the image

image = cv2.imread("car.jpg ")

#Convert to Grayscale

gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

#Perform Histogram Equalization equalized\_img = cv2.equalizeHist(gray)

#cv2.imshow("Original Gray Image",gray) #cv2.imshow("Histogram Equalized Image",equalized\_img)

plt.figure(figsize = (10,5))

#Image1 plt.subplot(2,2,1)

plt.imshow(gray, cmap = 'gray') plt.title('Original Gray Image') plt.axis('off')

#Image2 plt.subplot(2,2,2)

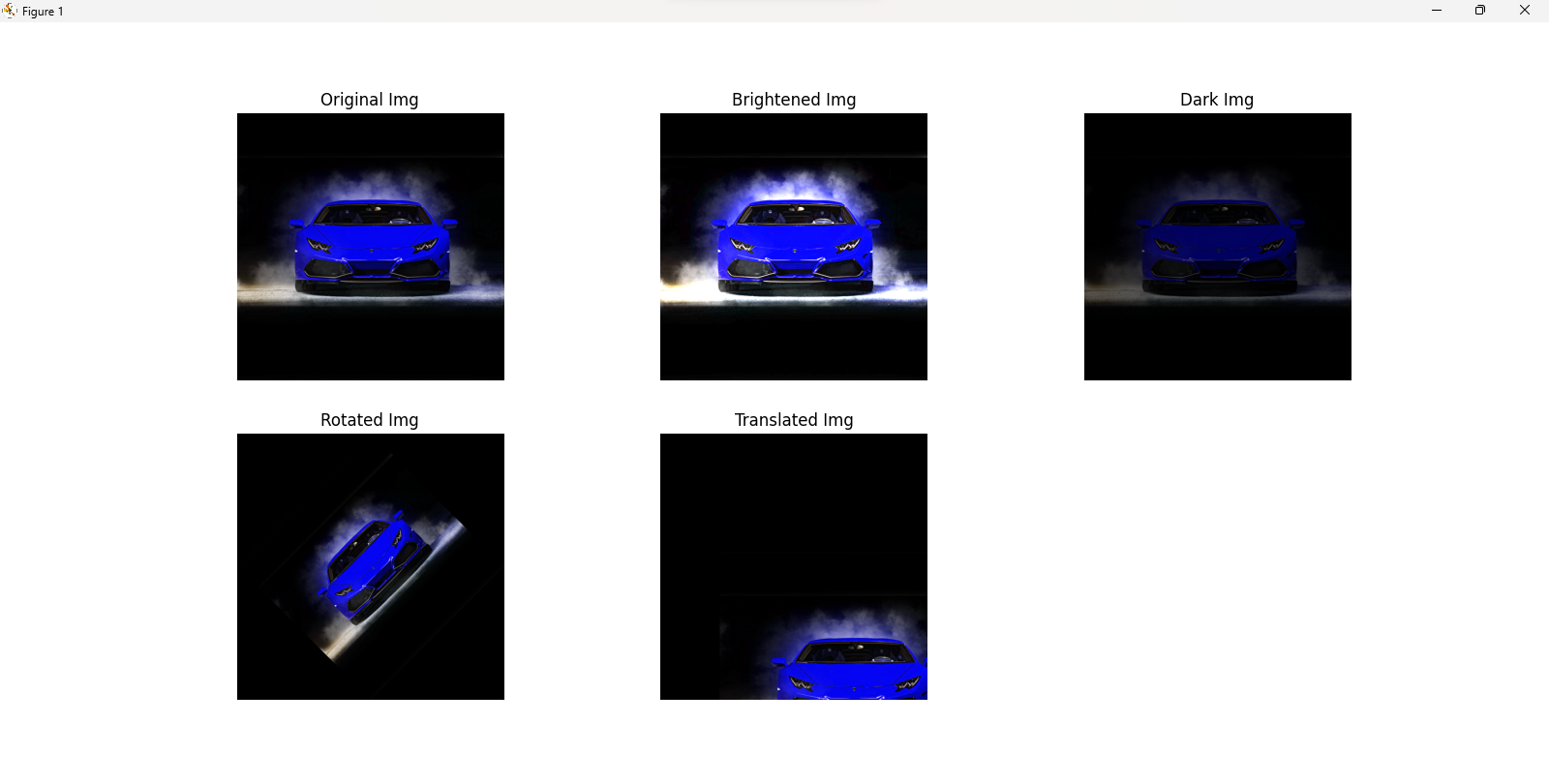
plt.imshow(equalized\_img, cmap = 'gray') plt.title('Histogram Equalized Image') plt.axis('off')

plt.show()

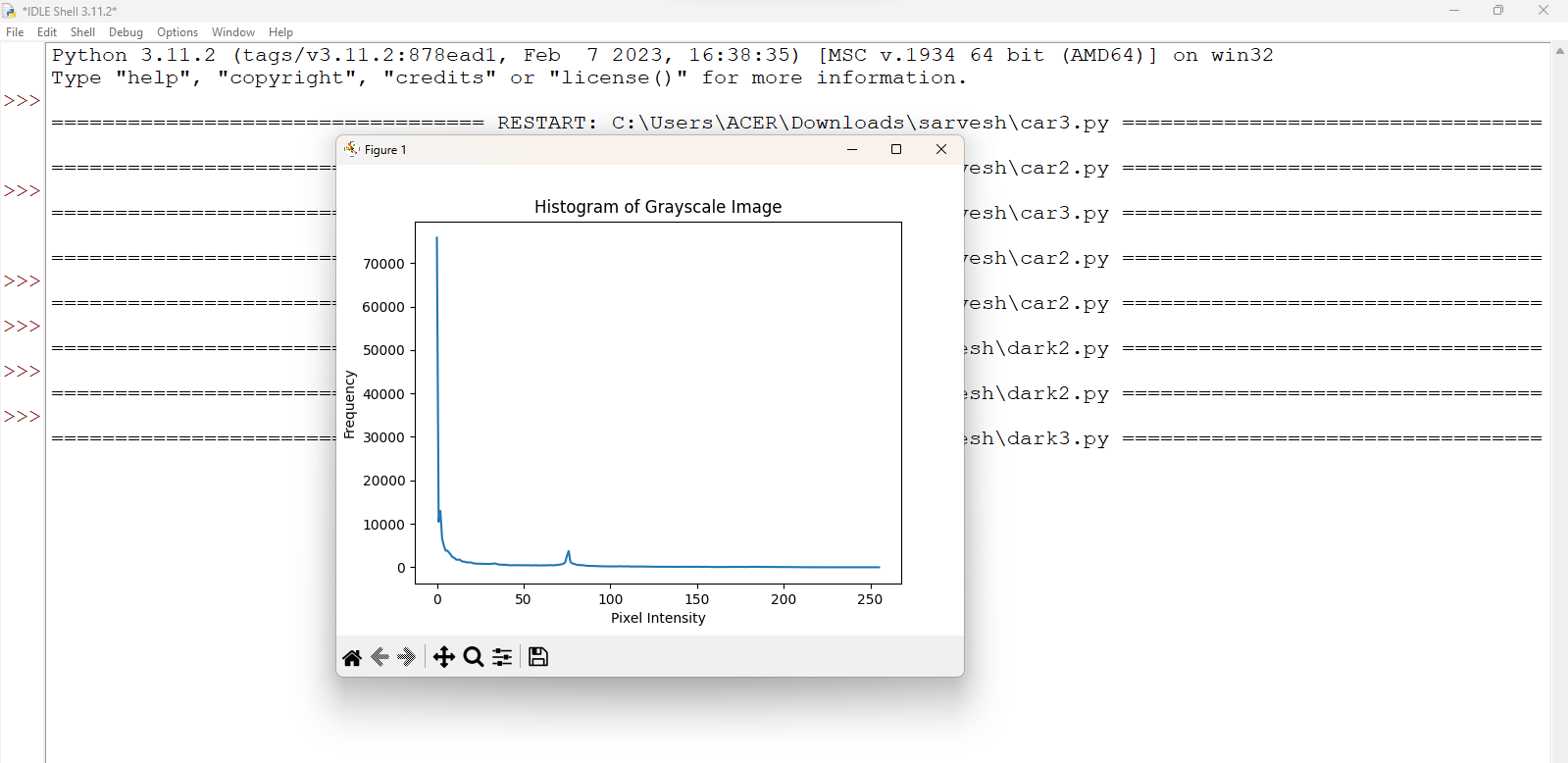
#cv2.waitKey(0) #cv2.destroyAllWindows()

**Output:**

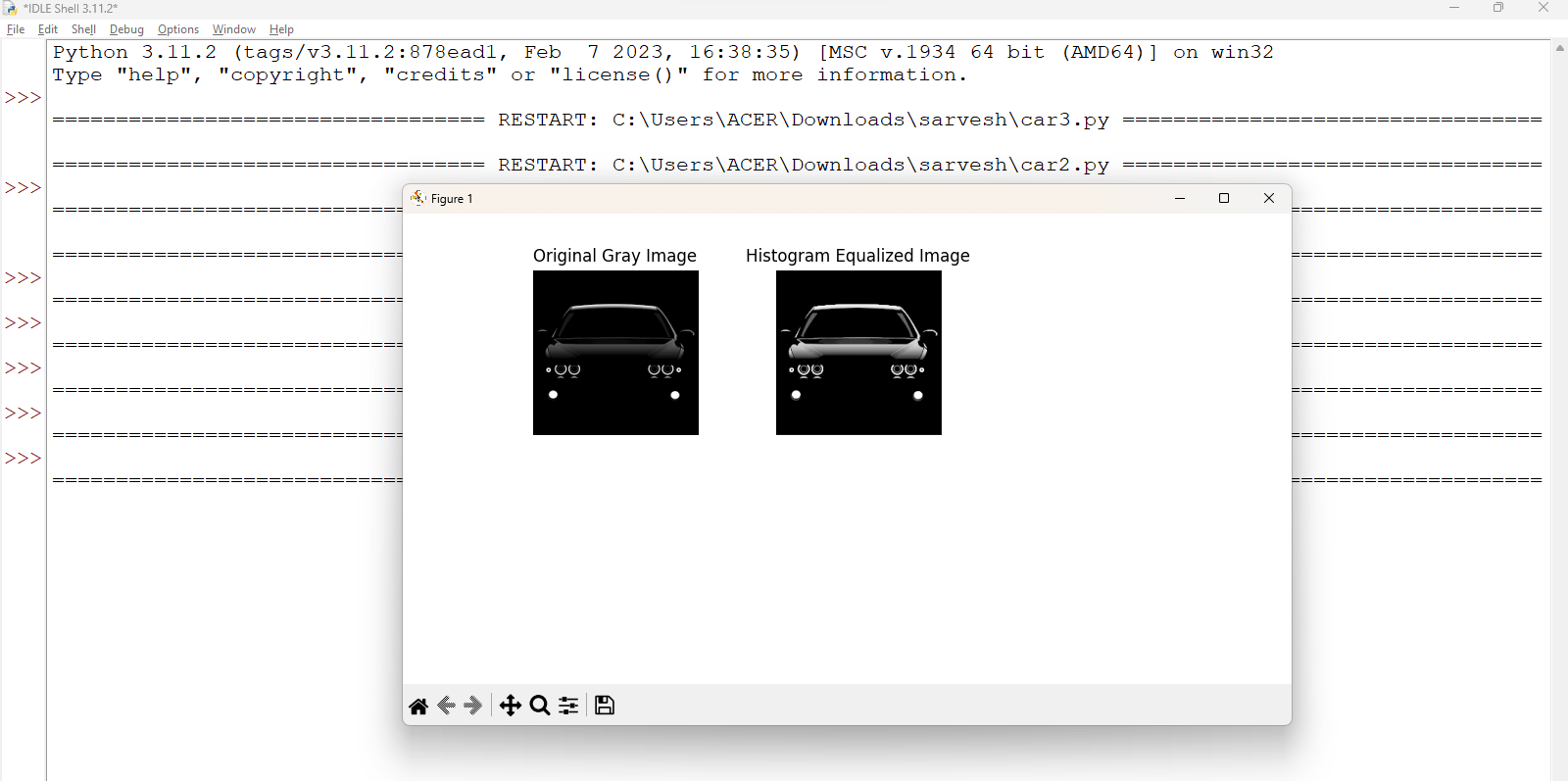
**Program 1:**

****

**Program 2:**

****

**Program 3:**

****

## Practical No.: 4

### Aim: Converting rgb image to binary image using thresholding method New Concept:

1. **cv2.IMREAD\_GRAYSCALE:** This is a flag used when reading an image with OpenCV (cv2). It specifies that the image should be loaded in grayscale (single-channel) mode, rather than the default color mode (BGR). When cv2.IMREAD\_GRAYSCALE is passed as an argument to the cv2.imread() function, the image is converted into a 2D array, where each pixel represents the intensity of light (from black to white), rather than having three channels (BGR).
2. **cv2.threshold:** This function is used for image binarization in OpenCV. It converts an image into a binary image based on a specified threshold value. The cv2.threshold function applies a threshold to an image and converts pixel values to either a high value (maxval) or a low value (usually 0), depending on whether they exceed a given threshold value (thresh). This is useful for segmenting objects from the background.
3. **cv2.THRESH\_BINARY:** This is a thresholding type used with cv2.threshold function. It applies a binary threshold to the image, where pixel values greater than the threshold are set to the maximum value, and those less than the threshold are set to 0. When using cv2.THRESH\_BINARY, the pixel intensity is either set to the maximum value (for example, 255) if it exceeds the threshold or to 0 if it doesn't. This creates a stark black-and-white image, useful for binary image analysis.
4. **image.shape:** The shape attribute of an image represents the dimensions of the image in the form of a tuple (height, width, channels). The image.shape provides the dimensions of the image:
   * height is the number of rows (the number of pixels along the vertical axis),
   * width is the number of columns (the number of pixels along the horizontal axis),
   * channels refers to the number of color channels (3 for RGB/BGR, 1 for grayscale).
5. **threshold\_value:** A value used in thresholding to define the cut-off point. Pixel values above the threshold are assigned one value (e.g., 255), and pixel values below the threshold are assigned another value (e.g., 0). The threshold\_value is the numerical value that is used to distinguish between foreground and background pixels during thresholding. For example, in binary thresholding, any pixel with a value greater than threshold\_value will be converted to 255 (white), and those less than the threshold will be converted to 0 (black).

### Theory:

1. **RGB to Binary:**
   * **RGB to Binary** is the process of converting an image from the RGB (Red, Green, Blue) color space to a binary (black and white) format. This conversion is crucial for many computer vision tasks, such as object detection, segmentation, and feature extraction.
   * **RGB Color Model**: In the RGB model, images are represented by three color channels: Red, Green, and Blue. Each pixel has a combination of these three channels, and each channel can have intensity values ranging from 0 to 255. This allows for millions of different colors to be represented.
   * **Binary Image**: A binary image is an image that consists of only two colors (usually black and white), represented by pixel values of 0 (black) and 255 (white). In a binary image, pixel values are either on (white) or off (black), with no intermediate shades.

### Applications:

* + 1. **Object detection**: Binarization is useful for detecting objects against a contrasting background.
    2. **Image segmentation**: Dividing an image into regions (foreground and background) based on pixel intensities.
    3. **Edge detection**: Helps to highlight features by emphasizing high-intensity regions.

### Threshold method:

* + The Threshold Method is a fundamental image segmentation technique used to convert an image into a binary form by applying a specific threshold value. The method classifies pixel values based on their intensity (or grayscale value) into two distinct categories: foreground (object) and background.

### Steps Involved:

* **Gray Level Intensity**: An image typically has pixel values that represent different shades of gray (for grayscale images) or color intensities (for RGB images). In the thresholding process, each pixel intensity is compared to a predefined threshold value.
* **Thresholding**: The core idea of the threshold method is:
  + If the pixel value is greater than or equal to the threshold value, it is assigned the maximum value (often 255 for white in binary images).
  + If the pixel value is less than the threshold value, it is assigned the minimum value (usually 0 for black in binary images).
* This effectively segments the image into two regions:
  + Foreground (objects of interest, often white),
  + Background (everything else, often black).
* The threshold value can be manually selected or determined using algorithms (e.g., Otsu's method) that compute the optimal threshold based on the image's histogram.

### Applications:

* **Image Binarization**: Essential for creating binary images that are easier to analyze and manipulate.
* **Object Detection and Recognition**: By segmenting the foreground from the background, thresholding helps identify objects within an image.
* **Edge Detection**: A form of thresholding is often applied after edge detection algorithms to enhance or simplify the results.

### Program:

import cv2

import numpy as np

import matplotlib.pyplot as plt

# Load the grayscale image

image = cv2.imread('car.jpg', cv2.IMREAD\_GRAYSCALE)

total=np.sum(image) w,h=image.shape avg=int(total/(w\*h))

# Apply thresholding

threshold\_value = avg # You can adjust this value

\_, thresholded\_image = cv2.threshold(image, threshold\_value, 255, cv2.THRESH\_BINARY)

# Display the thresholded image #cv2.imshow('Thresholded Image', thresholded\_image)

#Image1

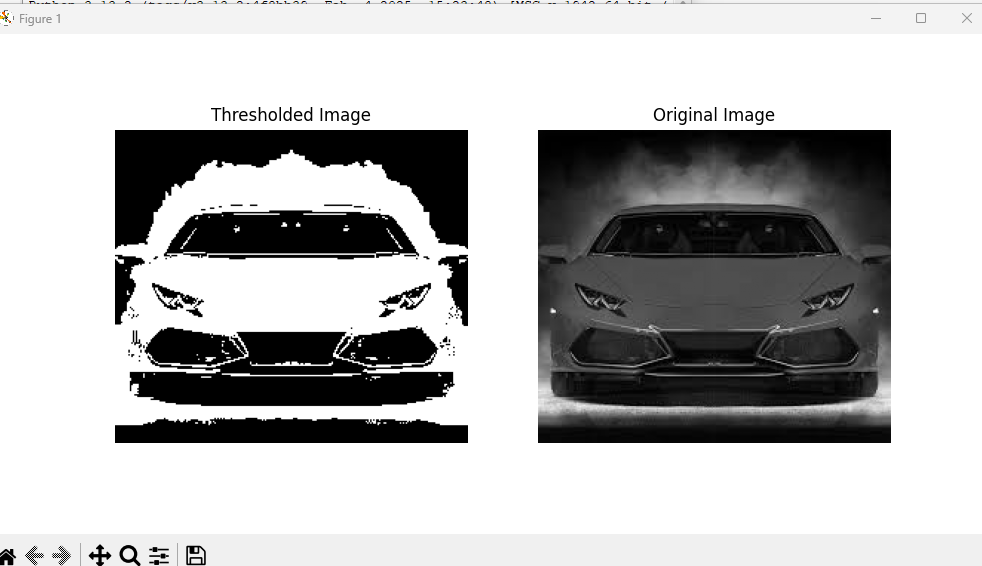
plt.figure(figsize = (10,5)) plt.subplot(1,2,2) #plt.imshow(image) plt.imshow(image, cmap = 'gray') plt.title('Original Image') plt.axis('off')

#Image2 plt.subplot(1,2,1)

#plt.imshow(thresholded\_image) plt.imshow(thresholded\_image, cmap = 'gray') plt.title('Thresholded Image')

plt.axis('off') plt.show()

cv2.waitKey(0) cv2.destroyAllWindows() **Output:**

****

## Practical No.: 5

### Aim: Color model conversion New Concept:

1. **cv2.COLOR\_BGR2HSV:** This is a flag used in OpenCV (cv2), which denotes a color space conversion from BGR (Blue, Green, Red) to HSV (Hue, Saturation, Value). The BGR color space is used by default in OpenCV, but the HSV color space is often preferred for certain image processing tasks because it separates the chromatic content (hue) from intensity (value), which can be more intuitive for color manipulation.
2. **cv2.COLOR\_BGR2LAB:** This flag is used for color conversion from BGR (Blue, Green, Red) to LAB (Lightness, A, B). The LAB color space is based on human vision and is device- independent, making it useful in various applications like image segmentation and recognition.
3. **cv2.COLOR\_BGR2YCrCb:** This is used to convert an image from BGR to the YCrCb color space. YCrCb separates the luminance (Y) from the chrominance (Cr and Cb), making it particularly useful for compression (e.g., JPEG) and video applications.
4. **cv2.COLOR\_BGR2CMYK:** This conversion flag indicates converting an image from BGR to CMYK (Cyan, Magenta, Yellow, Key/Black). The CMYK color space is commonly used in printing. However, OpenCV does not have a direct function for this conversion because CMYK is not typically used in image processing tasks for display. Often, conversion from RGB to CMYK is done manually.
5. **np.zeros\_like:** This is a NumPy function that creates a new array of the same shape and type as the provided input array, but with all elements initialized to zero. It is useful when you want to create an empty (black) image or array with the same dimensions and data type as an existing one.

### Theory:

1. **HSV Image:**

The HSV color space is a cylindrical representation of colors that is often used in computer vision, especially in tasks involving color segmentation, filtering, and image enhancement. It stands for Hue, Saturation, and Value, which are the three components of a color in this space.

### Components:

* + **Hue (H):** Represents the type of color (e.g., red, blue, green, etc.). It is measured in degrees on the color wheel, typically ranging from 0° to 360°:
    - 0° or 360° is red,
    - 120° is green,
    - 240° is blue, and so on.
  + **Saturation (S):** Represents the intensity or purity of the color. A saturation of 0 means the color is a shade of gray, and a saturation of 100 means the color is fully saturated, without any gray.
  + **Value (V):** Represents the brightness of the color. A value of 0 means black, and a value of 100 means full brightness.

### LAB Image:

The LAB color space is a perceptually uniform color model that was developed to approximate human vision. It is device-independent, meaning it can represent colors consistently across different devices (e.g., monitors, printers, etc.). The LAB color space consists of three components: L (Lightness), A (Green to Red), and B (Blue to Yellow).

### Components:

* + **L (Lightness):** Represents the brightness of the color, where 0 is black and 100 is white. This is similar to the intensity component in grayscale images.
  + **A (Green to Red):** Represents the color spectrum from green to red. Negative values indicate green, and positive values indicate red.
  + **B (Blue to Yellow):** Represents the color spectrum from blue to yellow. Negative values indicate blue, and positive values indicate yellow.

### YCbCr Image:

The YCbCr color space is primarily used in video compression and broadcast standards (e.g., JPEG, MPEG, and digital television). It separates the image into one luminance component and two chrominance components.

### Components:

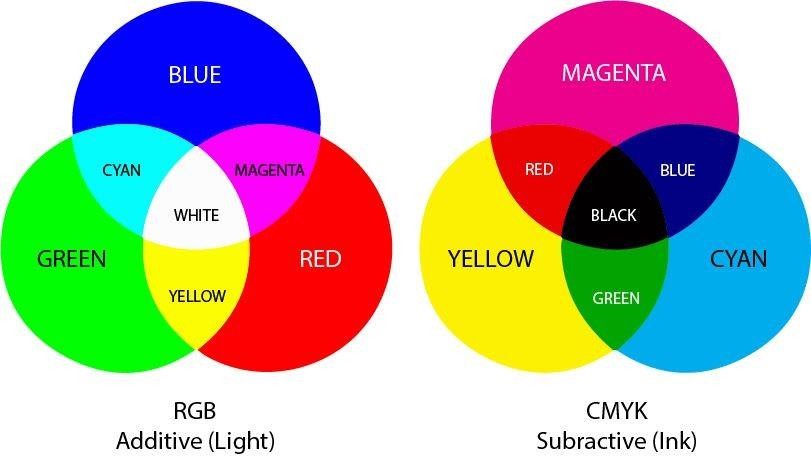
* + **Y (Luma):** Represents the brightness or lightness of the color. It is a grayscale image that contains all the intensity information of the image.
  + **Cb (Chroma Blue):** Represents the difference between the blue component and the luminance. It encodes the chrominance (color) information, but specifically how much blue is present in the image.
  + **Cr (Chroma Red):** Represents the difference between the red component and the luminance. Similar to Cb, it encodes the chrominance, but specifically how much red is present.

### CMYK Image:

The CMYK color space is primarily used in color printing. It is a subtractive color model, meaning that it represents colors based on the subtraction of various percentages of the four color components from a white background.

### Components:

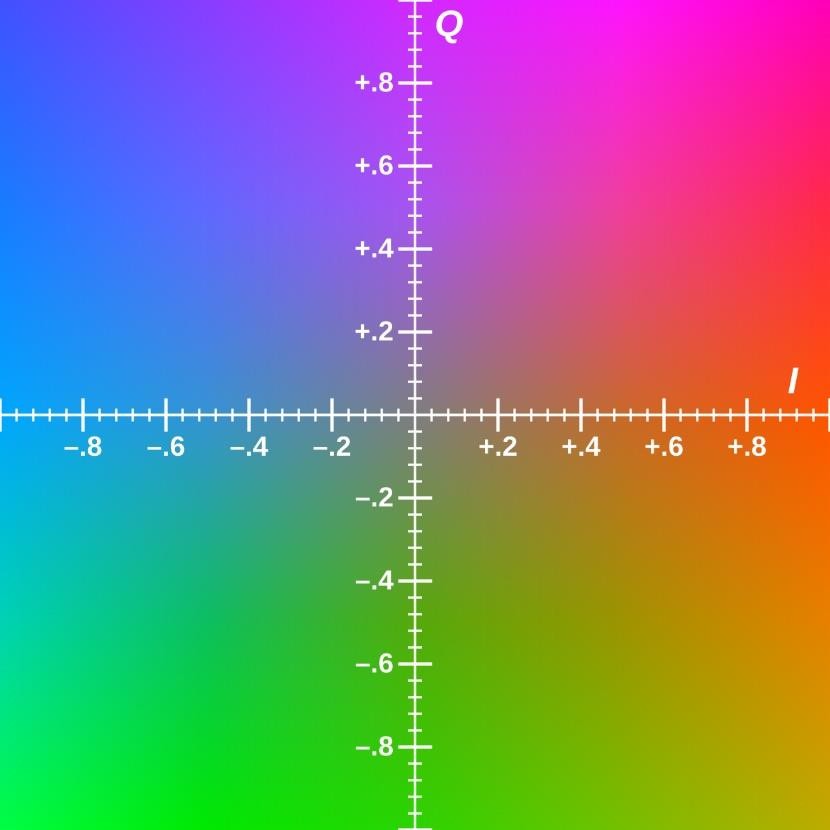
* + **C (Cyan):** A greenish-blue color.
  + **M (Magenta):** A purplish-red color.
  + **Y (Yellow):** A bright yellow color.
  + **K (Key/Black):** Used to add depth and detail to the image, and to prevent the colors from being washed out due to the limits of cyan, magenta, and yellow inks.



1. **YIQ Image:** The YIQ color space is used primarily in the NTSC television broadcast standard. It is similar to YCbCr but was designed to be more compatible with color TV systems in the United States. Like YCbCr, it separates luminance from chrominance.

### Components:

* + **Y (Luminance):** Represents the brightness of the image, similar to the Y component in YCbCr.
  + **I (In-phase):** Represents the red-to-green color component. It indicates how much red is present in relation to green.
  + **Q (Quadrature):** Represents the blue-to-yellow color component, similar to the Cb and Cr components in YCbCr.



### Program:

import cv2

import numpy as np

import matplotlib.pyplot as plt

# Load the image

image = cv2.imread('car.jpg')

rgb\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

# Convert to HSV (Hue, Saturation, Value)

hsv\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2HSV)

# Convert to LAB (CIELAB) color space

lab\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2LAB)

# Convert to YCbCr

ycbcr\_image = cv2.cvtColor(rgb\_image, cv2.COLOR\_BGR2YCrCb)

# Convert to CMYK

#cmyk\_image = cv2.cvtColor(rgb\_image, cv2.COLOR\_BGR2CMYK)

# Convert to YIQ

yiq\_image = np.zeros\_like(rgb\_image, dtype=np.float32)

yiq\_image[:,:,0] = 0.299 \* rgb\_image[:,:,2] + 0.587 \* rgb\_image[:,:,1] + 0.114 \* rgb\_image[:,:,0]

yiq\_image[:,:,1] = 0.596 \* rgb\_image[:,:,2] - 0.274 \* rgb\_image[:,:,1] - 0.322 \* rgb\_image[:,:,0]

yiq\_image[:,:,2] = 0.211 \* rgb\_image[:,:,2] - 0.523 \* rgb\_image[:,:,1] + 0.312 \* rgb\_image[:,:,0] yiq\_image = np.clip(yiq\_image, 0, 255).astype(np.uint8)

# Display the original and converted images #cv2.imshow('Original Image', image) #cv2.imshow('HSV Image', hsv\_image) #cv2.imshow('LAB Image', lab\_image) #cv2.imshow('YCbCr Image', ycbcr\_image) #cv2.imshow('CMYK Image', cmyk\_image) #cv2.imshow('YIQ Image', yiq\_image)

#Image1

plt.figure(figsize = (10,5)) plt.subplot(4,2,1) plt.imshow(rgb\_image) #plt.imshow(image, cmap = 'gray') plt.title('Original Image') plt.axis('off')

#Image2 plt.subplot(4,2,2) plt.imshow(hsv\_image)

#plt.imshow(hsv\_image, cmap = 'gray') plt.title('HSV Image')

plt.axis('off')

#Image3 plt.subplot(4,2,3) plt.imshow(rgb\_image)

#plt.imshow(image, cmap = 'gray') plt.title('Original Image') plt.axis('off')

#Image4 plt.subplot(4,2,4) plt.imshow(lab\_image)

#plt.imshow(lab\_image, cmap = 'gray') plt.title('LAB Image')

plt.axis('off')

#Image5 plt.subplot(4,2,5) plt.imshow(rgb\_image)

#plt.imshow(image, cmap = 'gray') plt.title('Original Image')

plt.axis('off')

#Image6 plt.subplot(4,2,6) plt.imshow(ycbcr\_image)

#plt.imshow(ycbcr\_image, cmap = 'gray') plt.title('YCbCr Image')

plt.axis('off')

#Image7 plt.subplot(4,2,7) plt.imshow(rgb\_image)

#plt.imshow(image, cmap = 'gray') plt.title('Original Image') plt.axis('off')

#Image8 plt.subplot(4,2,8) plt.imshow(yiq\_image)

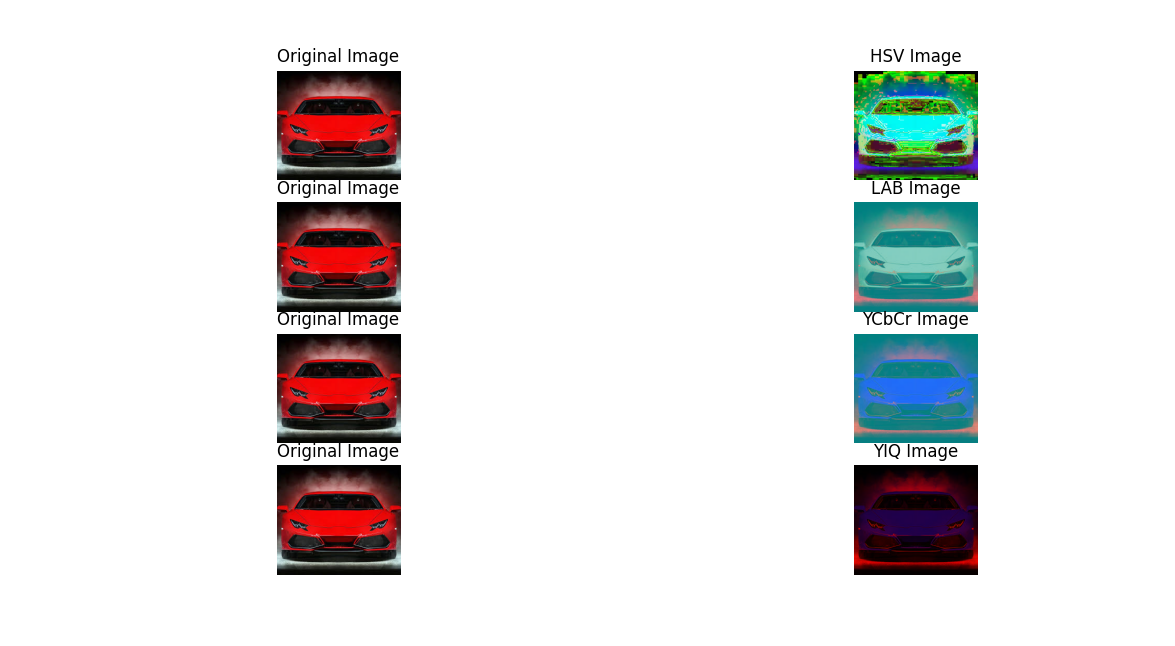
#plt.imshow(yiq\_image, cmap = 'gray') plt.title('YIQ Image')

plt.axis('off') plt.show()

# Wait for a key press and then close the windows cv2.waitKey(0)

cv2.destroyAllWindows()

**Output:**

****

## Practical No.: 7

### Aim: Shift Invariant Fourier Transform New Concept:

1. **cv2.SIFT\_create():** it is a function in OpenCV (a popular computer vision library) that creates a SIFT detector object. This object can be used to detect key points (distinct features) in an image and compute descriptors (representations) for those key points.
2. **sift.detectAndCompute:** it is a method used to detect key points and compute descriptors for an image. Points in the image that have distinctive patterns and can be used to identify parts of the image (e.g., corners, edges). A vector representation for each key point, capturing the local image patterns around the key point.
3. **cv2.BFMatcher():** it is a Brute Force Matcher in OpenCV. It is used to match feature descriptors between two images based on their distance. A brute force approach means it compares every descriptor from one set to every descriptor from another set to find the best match.
4. **bf.knnMatch:** it is a method of the Brute Force Matcher (BFMatcher) used for finding the k- nearest neighbors between two sets of descriptors. K-Nearest Neighbors (KNN) means that for each descriptor from the first image, it will find k closest matches from the second image.
5. **cv2.drawMatches:** it is a function that visually draws the matches between key points in two images. It helps to visualize the feature matching process, showing how key points from one image are aligned with key points in another image.
6. **cv2.DrawMatchesFlags\_NOT\_DRAW\_SINGLE\_POINTS:** this is a flag used with cv2.drawMatches to control how key points are drawn. cv2.DrawMatchesFlags\_NOT\_DRAW\_SINGLE\_POINTS means that key points that do not have matches will not be drawn. If you want to show only the matches (and not the single, unmatched key points), you can use this flag. If you want to draw all key points (even those without matches), you can use a different flag (e.g., cv2.DrawMatchesFlags\_DEFAULT).

### Theory:

SIFT (Scale-Invariant Feature Transform) is an algorithm used in computer vision to detect and describe important features in an image. These features, known as keypoints, are points in the image that stand out—such as corners, edges, or blobs. What makes SIFT special is that it can find these keypoints regardless of the image size, rotation, or lighting changes, making it scale- and rotation- invariant. Once the keypoints are detected, SIFT creates descriptors, which are numerical values that describe the local appearance around each keypoint. These descriptors can then be used to match features between different images, which is helpful in tasks like object recognition, image stitching, and motion tracking.

### Program:

import cv2

# Load two images

image1 = cv2.imread("car1.jpg") image2 = cv2.imread("car2.jpg")

# Convert images to grayscale

gray1 = cv2.cvtColor(image1, cv2.COLOR\_BGR2GRAY) gray2 = cv2.cvtColor(image2, cv2.COLOR\_BGR2GRAY)

# Initialize SIFT detector sift = cv2.SIFT\_create()

# Detect keypoints and compute descriptors for both images keypoints1, descriptors1 = sift.detectAndCompute(gray1, None) keypoints2, descriptors2 = sift.detectAndCompute(gray2, None)

# Initialize a Brute Force Matcher bf = cv2.BFMatcher()

# Match descriptors between the two images

matches = bf.knnMatch(descriptors1, descriptors2, k=2)

# Apply ratio test to filter good matches good\_matches = []

for m, n in matches:

if m.distance < 0.5 \* n.distance: good\_matches.append(m)

# Draw matches

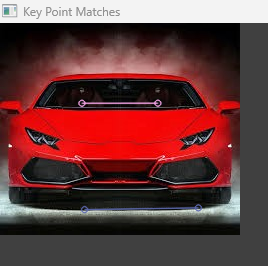
matched\_image = cv2.drawMatches(image1, keypoints1, image2, keypoints2, good\_matches, None, flags=cv2.DrawMatchesFlags\_NOT\_DRAW\_SINGLE\_POINTS)

# Display the matched image

cv2.imshow('Key Point Matches', matched\_image) cv2.waitKey(0)

cv2.destroyAllWindows()

**Output:**

****

## Practical No.: 8

### Aim: Image Stitching New Concept:

1. **imgs.append:** this is a basic Python list operation, not specific to OpenCV, but often used in computer vision scripts. It adds an image (often read via cv2.imread) to a list called imgs.
2. **cv2.resize:** resizing images (scaling them up/down) is crucial for preprocessing in computer vision. Changes the size (width and height) of an image.
3. **cv2.Stitcher.create():** used for image stitching — combining multiple images into a panorama. Creates an instance of the Stitcher class.
4. **stitchy.stitch:** performs the actual stitching of the images provided. Calls the stitch() method on the Stitcher object to combine images.
5. **cv2.STITCHER\_OK:** a status code constant used to check the result of the stitching process. It represents that stitching was successful.

### Theory:

Image stitching, also known as panoramic image stitching, is the process of combining multiple images into one cohesive and seamless larger image. This technique is commonly used in applications such as creating panoramas, VR environments, and even in medical imaging or aerial reconnaissance. The process involves detecting common features in the overlapping areas of adjacent images, aligning them properly, and blending them to create a continuous view without visible seams.

### Image Acquisition

The first step is to capture multiple images of the scene. These images must have significant overlap between consecutive frames, which allows for the identification of common features between them. These images can be taken with the same camera, or they can come from different sources, but they should have consistent lighting and focus to minimize complications.

### Feature Detection and Matching

Feature detection is one of the most critical aspects of image stitching. The goal is to identify distinctive and repeatable patterns in the images, which can then be used to match corresponding points across the images.

### Image Registration (Geometric Transformation)

Once the matching keypoints are identified, the next step is to align the images. This process is known as image registration. The idea is to find a transformation that maps the coordinates of the keypoints in one image to the corresponding coordinates in another image.

### Image Blending

Once the images are aligned, the next challenge is to blend them seamlessly. This step involves addressing issues like color mismatches, visible seams, and ghosting effects (where parts of the images do not align perfectly).

### Post-processing

After the images are blended, post-processing steps like exposure adjustment, color correction, and cropping might be necessary to further improve the visual appeal of the stitched image. Some systems

also perform lens distortion correction if the original images have noticeable lens distortion, which can affect alignment accuracy.

### Program:

import cv2 image\_paths=["img1.jpg","img2.jpg","img3.jpg","img4.jpg"]

# initialized a list of images imgs = []

for i in range(len(image\_paths)): imgs.append(cv2.imread(image\_paths[i])) imgs[i]=cv2.resize(imgs[i],(0,0),fx=0.4,fy=0.4)

# showing the original pictures cv2.imshow('1',imgs[0])

cv2.imshow('2',imgs[1])

cv2.imshow('3',imgs[2])

cv2.imshow('4',imgs[3]) stitchy=cv2.Stitcher.create() (dummy,output)=stitchy.stitch(imgs)

if dummy != cv2.STITCHER\_OK:

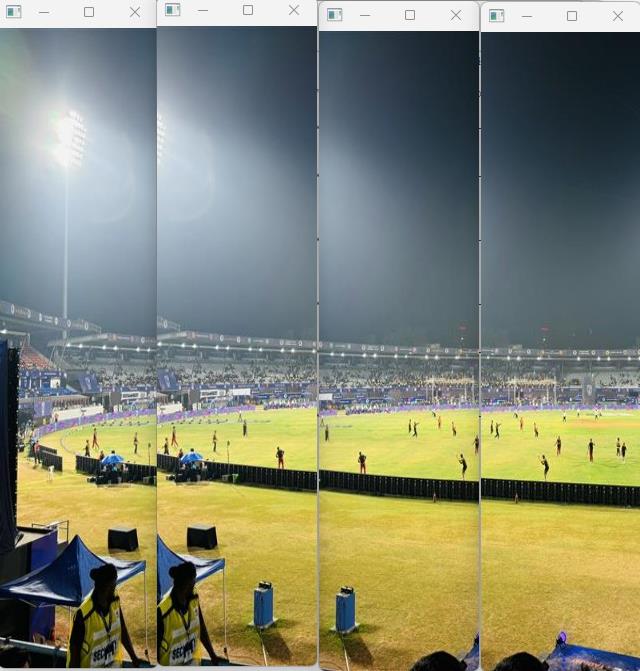
# checking if the stitching procedure is successful print("stitching ain't successful")

else:

print('Your Panorama is ready!!!') # final output

cv2.imshow('final result',output) cv2.waitKey(0)

Single Images:



**Output:**



## Practical No.: 9

### Aim: 2D to 3D conversion New Concept:

**PIL:** PIL is a library in Python that adds image processing capabilities. It lets you open, manipulate, and save many different image file formats. PIL is usually accessed through its fork: Pillow.

**img.convert:** This method is used to convert an image to a different color mode, such as:

* 'L' for grayscale,
* 'RGB' for true color,
* 'RGBA' for color with transparency,
* '1' for black and white (binary).

**shift\_image:** Not a built-in function but usually a custom function in Computer Vision used to translate (shift) an image along the x and/or y axes.

**depth\_data:** Refers to data representing depth (distance from the camera to each point in the scene). Common in applications like 3D reconstruction, stereo vision, or using depth sensors (e.g., Kinect, LiDAR). It is a 2D NumPy array where each value represents depth at a pixel.

**shift\_amount:** Usually refers to the number of pixels by which an image or data is to be shifted in shift\_image or similar operations. It can be a tuple like (dx, dy) for horizontal and vertical shifts.

**np.zeros\_like(data):** Creates a NumPy array of zeros with the same shape and type as data. Useful when initializing an empty image or depth map.

**data.shape:** This gives the dimensions of the data, typically a NumPy array. In Computer Vision:

* For grayscale image: (height, width)
* For RGB image: (height, width, 3)

**Image.fromarray:** A method from PIL that creates an Image object from a NumPy array.

**Image.open:** This is used to open an image file and load it as a PIL Image object.

### Theory:

2D images contain intensity/color information at each pixel, but no information about depth (distance from the camera). Converting a 2D image into a 3D representation means estimating the depth for each pixel — essentially creating a 3D model or depth map from flat image data.

2D to 3D conversion is key in:

* Autonomous vehicles (for obstacle detection)
* AR/VR (placing virtual objects in 3D space)
* Robotics (navigation and interaction)
* 3D modeling (from photos)
* Medical imaging (3D reconstruction of organs)

### Program:

from PIL import Image import numpy as np

def shift\_image(img, depth\_img, shift\_amount=10): # Ensure base image has alpha

img = img.convert("RGBA") data = np.array(img)

# Ensure depth image is grayscale (for single value) depth\_img = depth\_img.convert("L")

depth\_data = np.array(depth\_img)

deltas = ((depth\_data / 255.0) \* float(shift\_amount)).astype(int) # This creates the transparent resulting image.

# For now, we're dealing with pixel data. shifted\_data = np.zeros\_like(data)

height, width, \_ = data.shape for y, row in enumerate(deltas):

for x, dx in enumerate(row):

if x + dx < width and x + dx >= 0: shifted\_data[y, x + dx] = data[y, x]

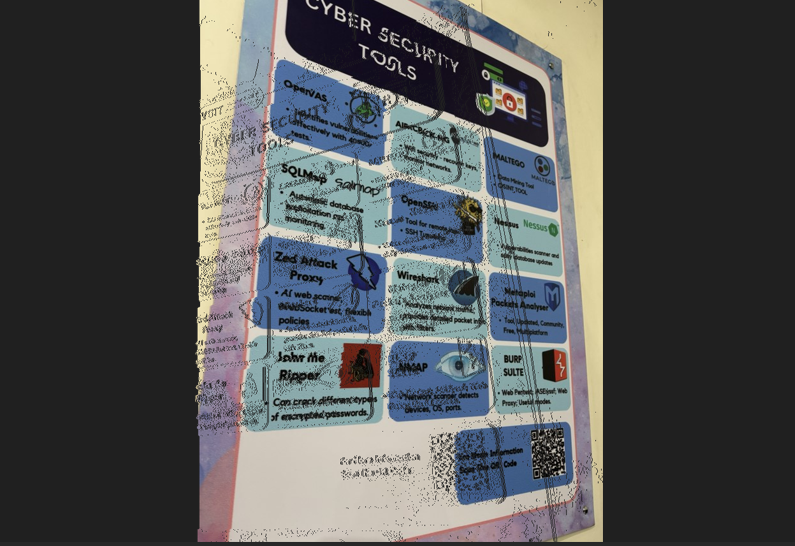
# Convert the pixel data to an image.

shifted\_image = Image.fromarray(shifted\_data.astype(np.uint8)) return shifted\_image

img = Image.open("2D\_1.jpeg") depth\_img = Image.open("2D\_2.jpeg")

shifted\_img = shift\_image(img, depth\_img, shift\_amount=10) shifted\_img.show()

**Output:**

****

# Practical No.: 10

**Aim:** Object Detection.

**New Concept:**

* **Ultralytics** – A Python library that provides easy access to YOLO (You Only Look Once) models for object detection tasks.
* **YOLO** – A fast, real-time object detection model that identifies and classifies objects in images or video streams.
* **cv2.VideoCapture(0)** – Opens the default webcam (device 0) for capturing live video.
* **cap.set(x, y)** – Sets a video capture property like frame width or height (x is the property ID, y is the value).
* **cap.read()** – Reads a frame from the webcam and returns the image data.
* **bounding box** – A rectangle that surrounds a detected object in an image, showing its location.
* **cv2.FONT\_HERSHEY\_SIMPLEX** – A simple font style used in OpenCV for displaying text on images.
* **cv2.rectangle** – Draws a rectangle (e.g., a bounding box) on an image at specified coordinates.

**Theory:** In this Practical we are using the YOLO (You Only Look Once) object detection model from the Ultralytics library to detect and label objects in a live webcam feed or a static image. It starts by capturing video using OpenCV's cv2.VideoCapture and sets the frame size with cap.set. The program then reads each frame using cap.read and processes it through the YOLO model to identify objects. For every detected object, the program draws a rectangular box (called a bounding box) around it using cv2.rectangle and displays the object’s name using a simple font (cv2.FONT\_HERSHEY\_SIMPLEX). This helps in visually recognizing and labeling various objects like people, cars, and animals directly on the video or image in real time.

**Program:**

from ultralytics import YOLO import cv2

import math

# start webcam

#cap = cv2.VideoCapture(0) #cap.set(3, 640)

#cap.set(4, 480)

# model

model = YOLO("yolov8n.pt") # object classes

classNames = ["person", "bicycle", "car", "motorbike", "aeroplane", "bus", "train", "truck", "boat", "traffic light", "fire hydrant", "stop sign", "parking meter", "bench", "bird", "cat", "dog", "horse", "sheep", "cow", "elephant", "bear", "zebra", "giraffe", "backpack",

"umbrella", "handbag", "tie", "suitcase", "frisbee", "skis", "snowboard", "sports ball", "kite", "baseball bat", "baseball glove", "skateboard", "surfboard", "tennis racket", "bottle", "wine glass", "cup", "fork", "knife", "spoon", "bowl", "banana", "apple", "sandwich", "orange",

"broccoli", "carrot", "hot dog", "pizza", "donut", "cake", "chair", "sofa", "pottedplant", "bed", "diningtable", "toilet", "tvmonitor", "laptop", "mouse", "remote", "keyboard", "cell phone", "microwave", "oven", "toaster", "sink", "refrigerator", "book", "clock", "vase", "scissors", "teddy bear", "hair drier", "toothbrush"]

#while True:

img = cv2.imread("getimage.jpeg") results = model(img, stream=True)

# coordinates for r in results:

boxes = r.boxes

for box in boxes: # bounding box

x1, y1, x2, y2 = box.xyxy[0]

x1, y1, x2, y2 = int(x1), int(y1), int(x2), int(y2) # convert to int values

# put box in cam

cv2.rectangle(img, (x1, y1), (x2, y2), (255, 0, 255), 3)

# confidence

confidence = math.ceil((box.conf[0]\*100))/100 print("Confidence --->",confidence)

# class name

cls = int(box.cls[0])

print("Class name -->", classNames[cls])

# object details org = [x1, y1]

font = cv2.FONT\_HERSHEY\_SIMPLEX

fontScale = 1

color = (255, 0, 0)

thickness = 2

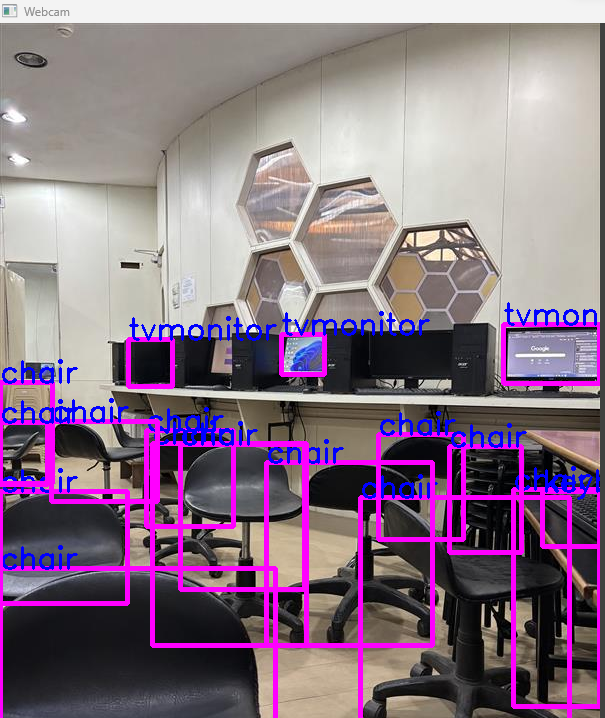
cv2.putText(img, classNames[cls], org, font, fontScale, color, thickness) cv2.imshow('Webcam', img)

#if cv2.waitKey(1) == ord('q'):

#break

#cap.release() cv2.waitKey(0) cv2.destroyAllWindows()

**Output:**

****