Programming for Artificial Intelligence Lab

Submitted to:

Rasikh Ali

Submitted by:

Shazra Zainab 003

Department:

Software Engineering

Section:

BSAI(4A)

Roll No:

SU92-BSAIM-F23-003

Task 05

>> 2.1 Getting Started

Reading an image in OpenCV using Python:

Input image:

Code explanation:

This Python code uses the OpenCV library to read and display an image. First, it imports OpenCV (cv2), which is a tool for working with images and videos. Then, it loads an image file named "istockphoto-2153573055612x612.jpg" using cv2.imread(), with cv2.IMREAD_COLOR ensuring that the image is read in full color. After that, the cv2.imshow() function creates a window titled "image" and displays the loaded image. The program then pauses with cv2.waitKey(0), meaning it waits for the user to press any key before closing. Finally, cv2.destroyAllWindows() ensures that all image windows are closed once a key is pressed. In short, this code opens an image, shows it on the screen, and waits for user interaction before closing it.

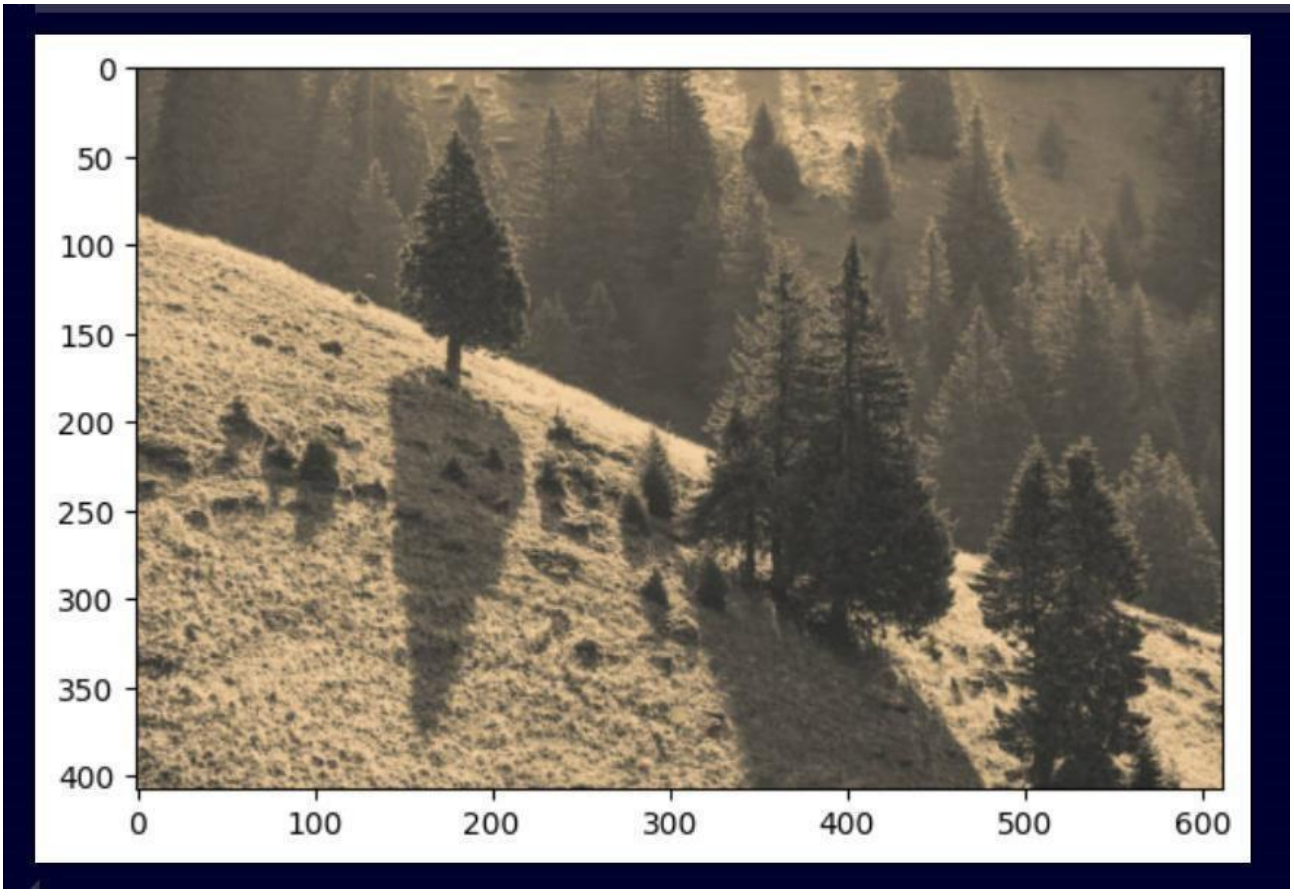
Output:

Input:

Code Explanation:

This Python code uses three important libraries—OpenCV (cv2), NumPy (np), and Matplotlib (plt)—to load and display an image. First, it reads an image file called "istockphoto-2153573055-612x612.jpg" using cv2.imread(). OpenCV loads images in BGR (Blue-Green-Red) format by default, but Matplotlib works with RGB (Red-Green-Blue) format, so the code converts the image from BGR to RGB using cv2.cvtColor(). After that, the image is displayed using Matplotlib's plt.imshow(). The plt.waitforbuttonpress() function makes the program wait until the user presses a button before closing the image window. Finally, plt.close('all') ensures that all Matplotlib windows are closed. This code helps display an image correctly using Matplotlib while handling color format conversion.

Output:



Input:

Code Explanation:

This Python program demonstrates how to use OpenCV's cv2.imread() method to read an image in grayscale mode. First, it imports the OpenCV library (cv2). Then, it defines the image file path as "Car.jpeg". Using cv2.imread(), the image is loaded with cv2.IMREAD_GRAYSCALE, which converts it into a black-and-white (grayscale) format instead of keeping its original colors. Next, the cv2.imshow() function is used to display the image in a window titled "image". The program then waits for the user to press any key with cv2.waitKey(0), ensuring the image remains visible until user input. Finally, cv2.destroyAllWindows() closes the image window. This code is useful when working with grayscale images, which can be helpful in tasks like edge detection and object recognition.

Output:

Display an image in OpenCV using Python:

Input:

Code Explanation:

This Python program shows how to display an image using OpenCV's cv2.imshow() method. First, it imports the cv2 library, which is used for image processing. Then, it sets the path of the image file (Car2.jpeg) stored on the computer. The image is read using cv2.imread(), which loads it into the program. Next, a window name "image" is set, and cv2.imshow() is used to open a window and display the loaded image. The program then waits for the user to press any key with cv2.waitKey(0), keeping the image visible until then. Finally, cv2.destroyAllWindows() closes the image window. This code is helpful for quickly opening and viewing images using Python.

Output:

Input:

Code Explanation:

This Python program uses OpenCV to load and display an image in grayscale mode. First, it imports the cv2 library, which is used for image processing. Then, it sets the file path of the image (Car2.jpeg) using a raw string (r''), which prevents errors caused by backslashes in Windows file paths. The image is read using cv2.imread(path, 0), where 0 tells OpenCV to convert the image to black and white (grayscale). Before displaying, the program checks if the image was loaded correctly—if not, it prints an error message. If the image is successfully loaded, cv2.imshow() opens a window to display it. The program waits for the user to press any key with cv2.waitKey(0), keeping the image visible until then. Finally, cv2.destroyAllWindows() closes the window. This code ensures that the image is loaded properly and displayed without crashing.

Output:

Writing an image in OpenCV using Python:

Input: 

Code Explanation:

This Python program reads an image file named "Nature.jpeg" from a specified location using the OpenCV (cv2) library and saves a copy of it in the same folder. First, it imports the necessary libraries, cv2 for handling images and os for working with files and directories. It then sets the image path and the working directory. The cv2.imread() function loads the image into Python, and os.chdir(directory) ensures we are in the correct folder. Before saving, the program prints the list of files in the folder. Using cv2.imwrite('savedImage.jpg', img), it saves the loaded image as "savedImage.jpg" in the same directory. After saving, it prints the updated list of files to confirm the new image is present. Finally, it prints a success message to indicate that the image was saved correctly.

Output:

```
Chat (CTRL + I) / Edit (CTRL + L)

[1]

Before saving image:
['Car.jpeg', 'Car2.jpeg', 'istockphoto-2153573055-612x612.jpg', 'Nature.jpeg', 'opencv.ipynb', 'venv']
After saving image:
['Car.jpeg', 'Car2.jpeg', 'istockphoto-2153573055-612x612.jpg', 'Nature.jpeg', 'opencv.ipynb', 'savedImage.jpg', 'venv']
Successfully saved
```

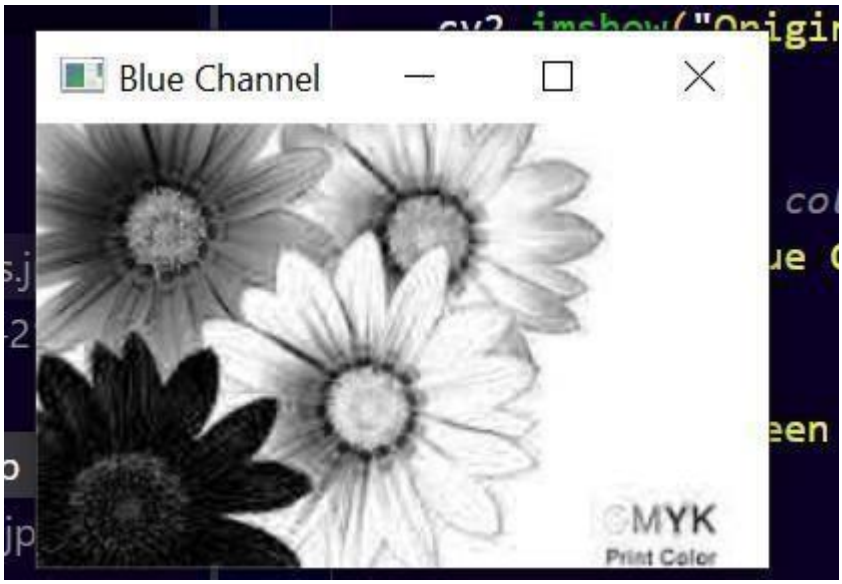
Color Spaces: Input:

Code Explanation:

This program loads an image from your computer, separates its colors (Blue, Green, and Red), and displays them one by one. First, it imports the cv2 library, which helps in handling images. The cv2.imread() function reads the image, and the raw string (r'') ensures the file path works correctly. Before proceeding, the program checks if the image is loaded properly; if not, it prints an error and stops. Then, it uses cv2.split(image) to break the image into three color channels: Blue (B), Green (G), and Red (R). The program first shows the original image and waits for a key press before moving to the next step. After that, it displays each color channel separately, waiting for a key press after each one. Finally, cv2.destroyAllWindows() closes all the image windows. This way, you can clearly see how an image is made up of three colors!

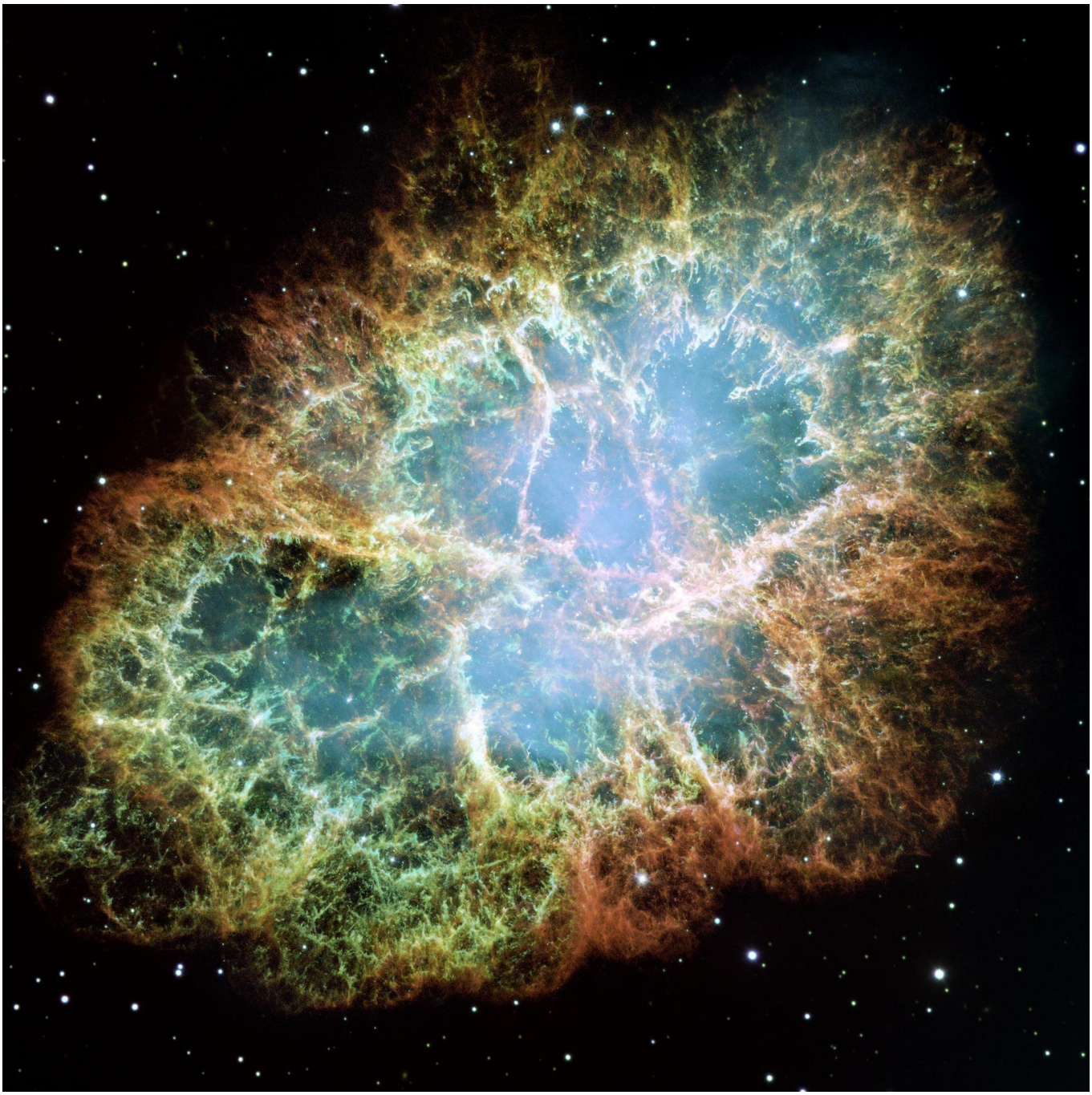
Output:


Original Ima...
CMYK
Print Color


Blue Channel
CMYK
Print Color

Arithmetic operations on images:

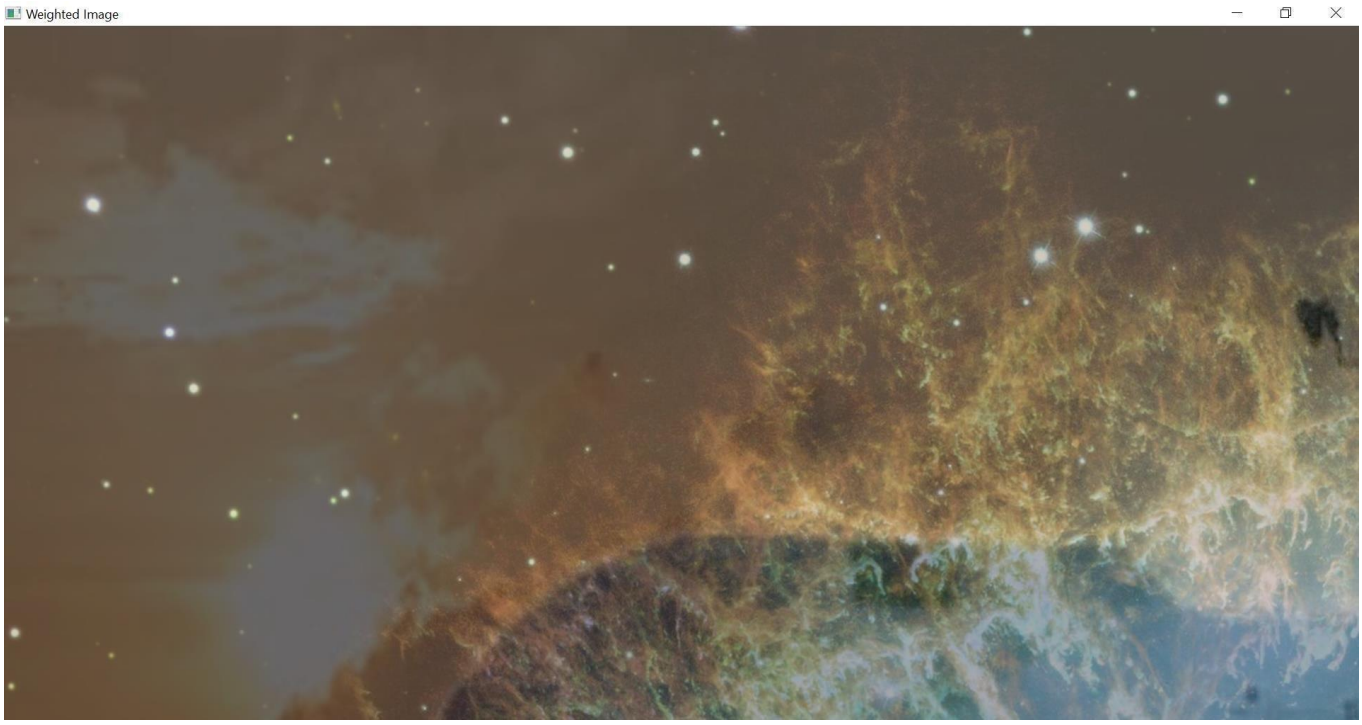
Input:



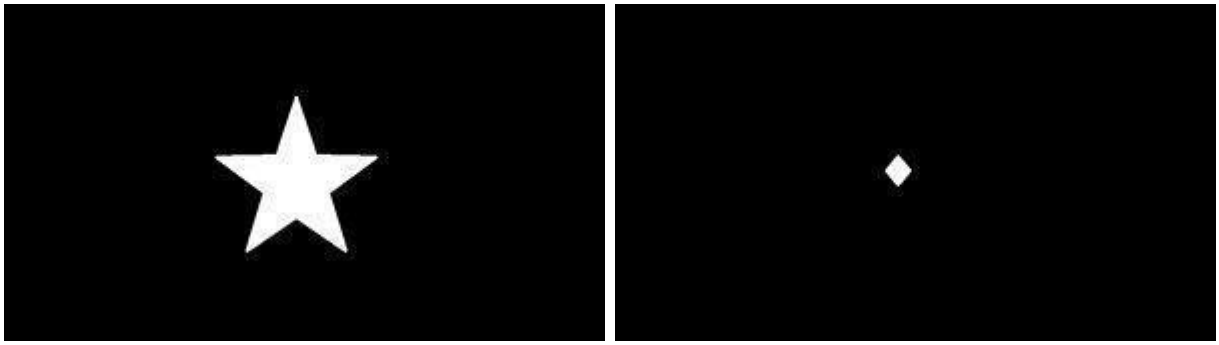
Code Explanation:

This program loads and processes images using OpenCV. It first reads images from the computer, checks if they exist, and then performs operations like displaying, converting colors, and saving. It also splits an image into blue, green, and red channels. Finally, it blends two images together using cv2.addWeighted(), where each image contributes a certain percentage to the final mix. The program ensures images are the same size before blending and properly closes windows after displaying them.

Output:



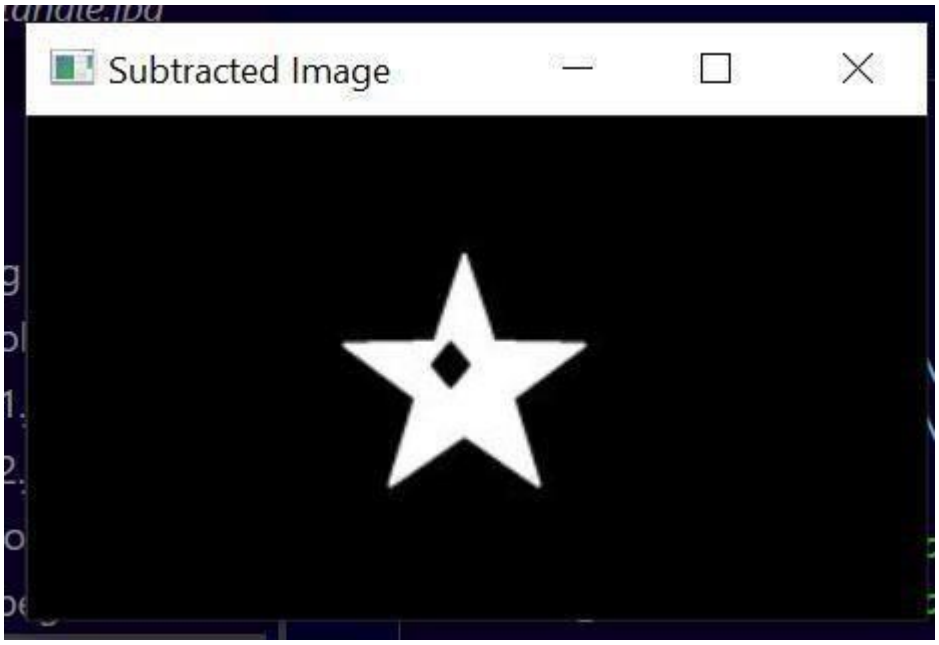
Subtraction: Input:



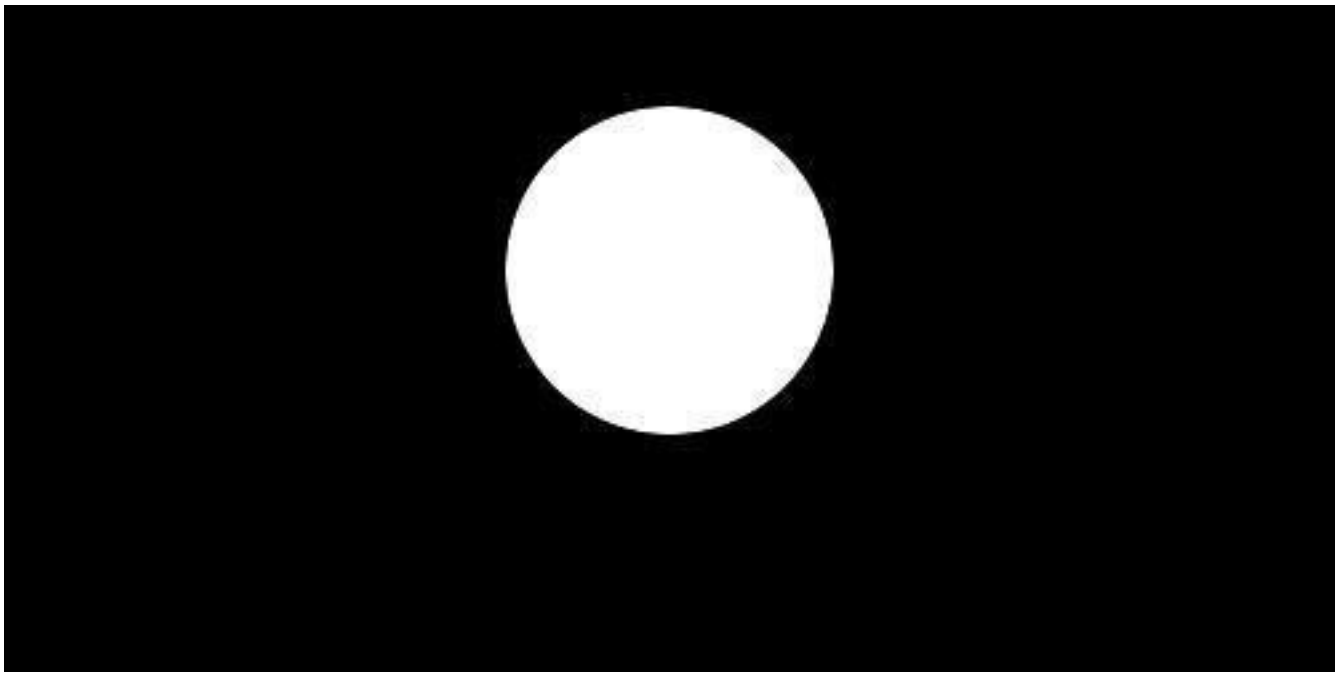
Code explanation:

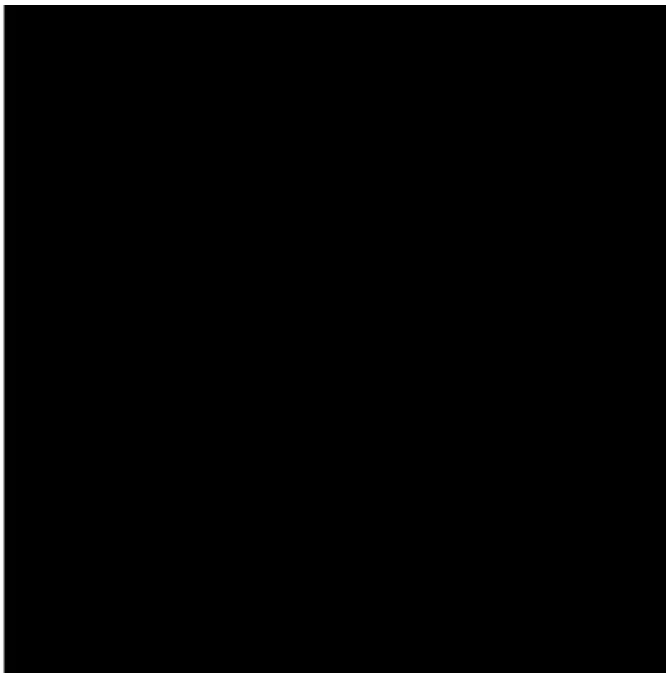
This program subtracts one image from another using OpenCV. It first loads two images from the specified file paths and checks if they exist. If either image is missing, the program prints an error and stops. Since both images need to be the same size for subtraction, it resizes the second image to match the first if necessary. Then, it performs pixel-wise subtraction using cv2.subtract(), where the pixel values of the second image are subtracted from the first. Finally, it displays the subtracted image in a window, and when the 'Esc' key is pressed, the window closes.

Output:



Bitwise Operations on Binary Images: Input:

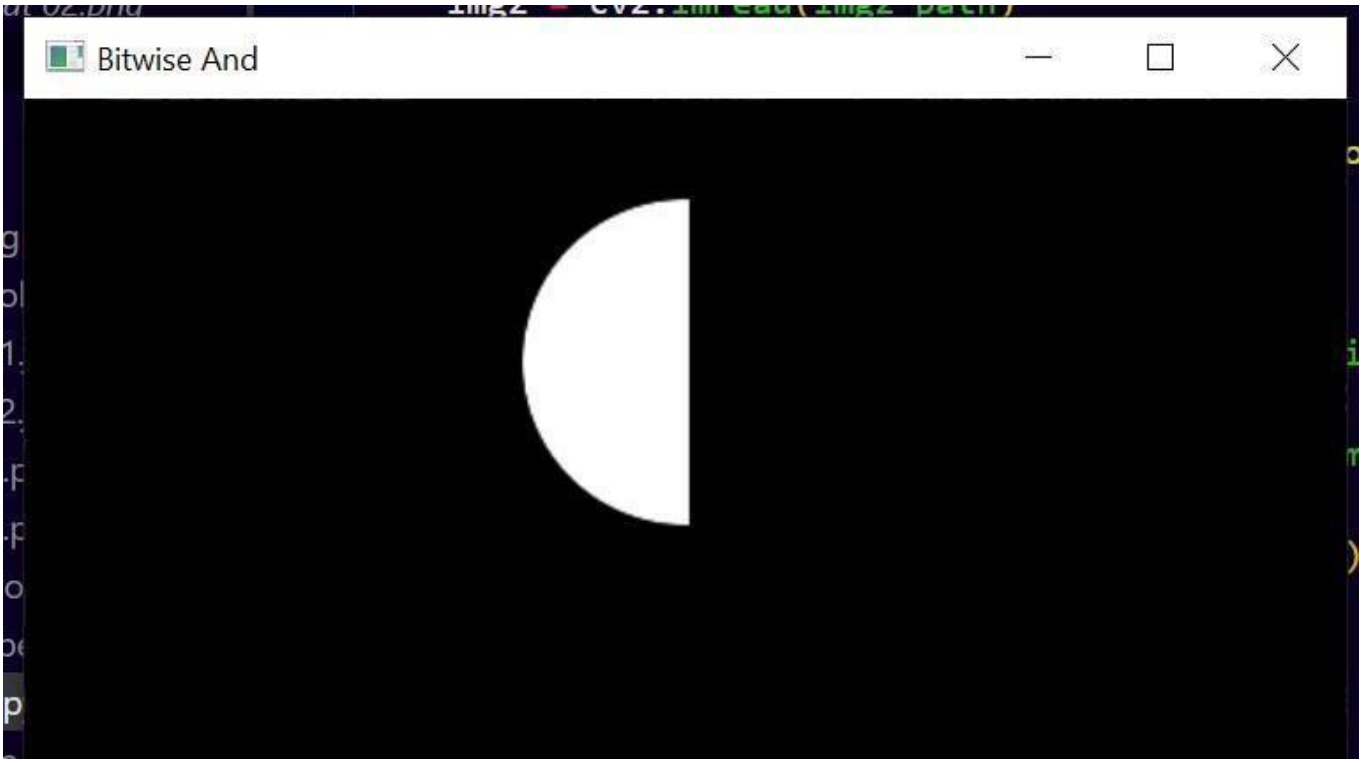




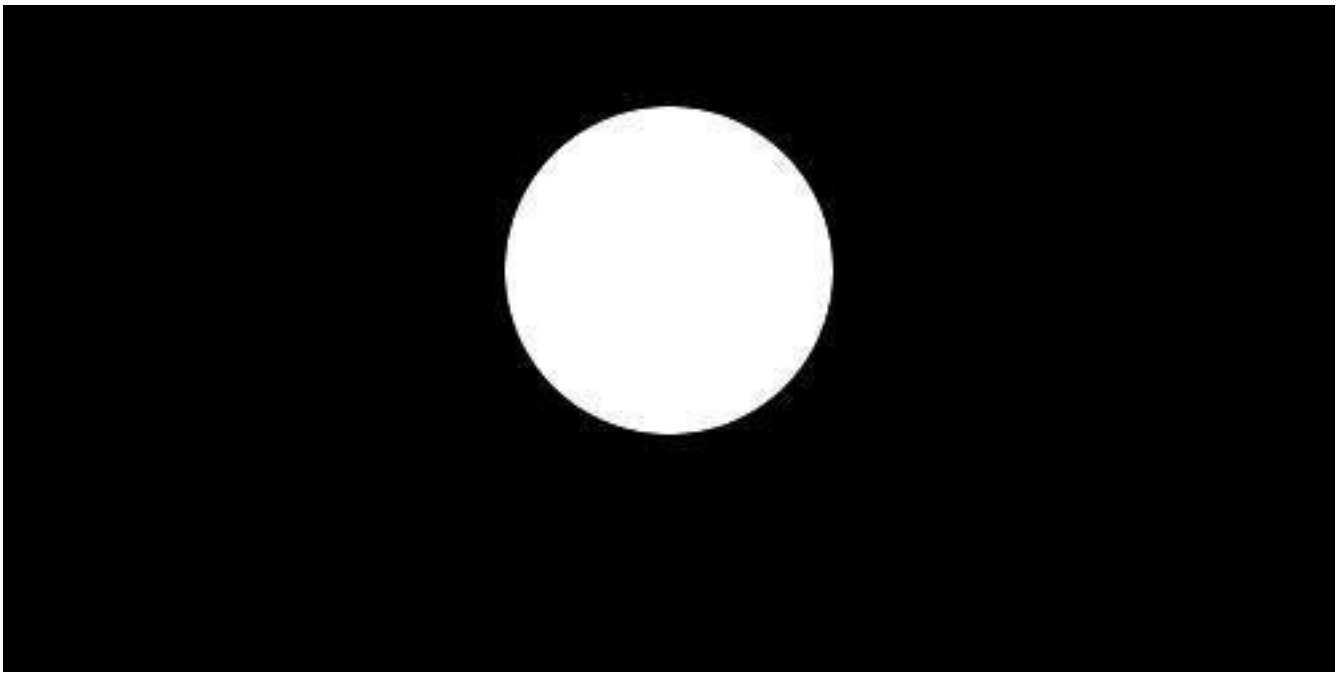
Code Explanation:

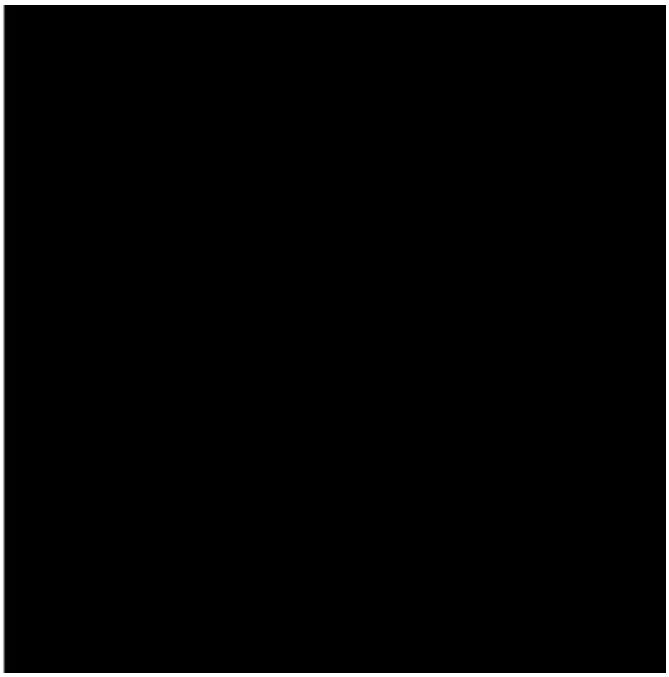
This program loads two images from the computer and applies a bitwise AND operation on them. It first checks if the images exist; if not, it stops the program. If the images are not the same size, the second image is resized to match the first one. Then, it performs a bitwise AND operation, meaning it compares each pixel from both images and keeps only the parts where both images overlap. The final processed image is displayed in a window, and when the 'Esc' key is pressed, the window closes.

Output:



Input:

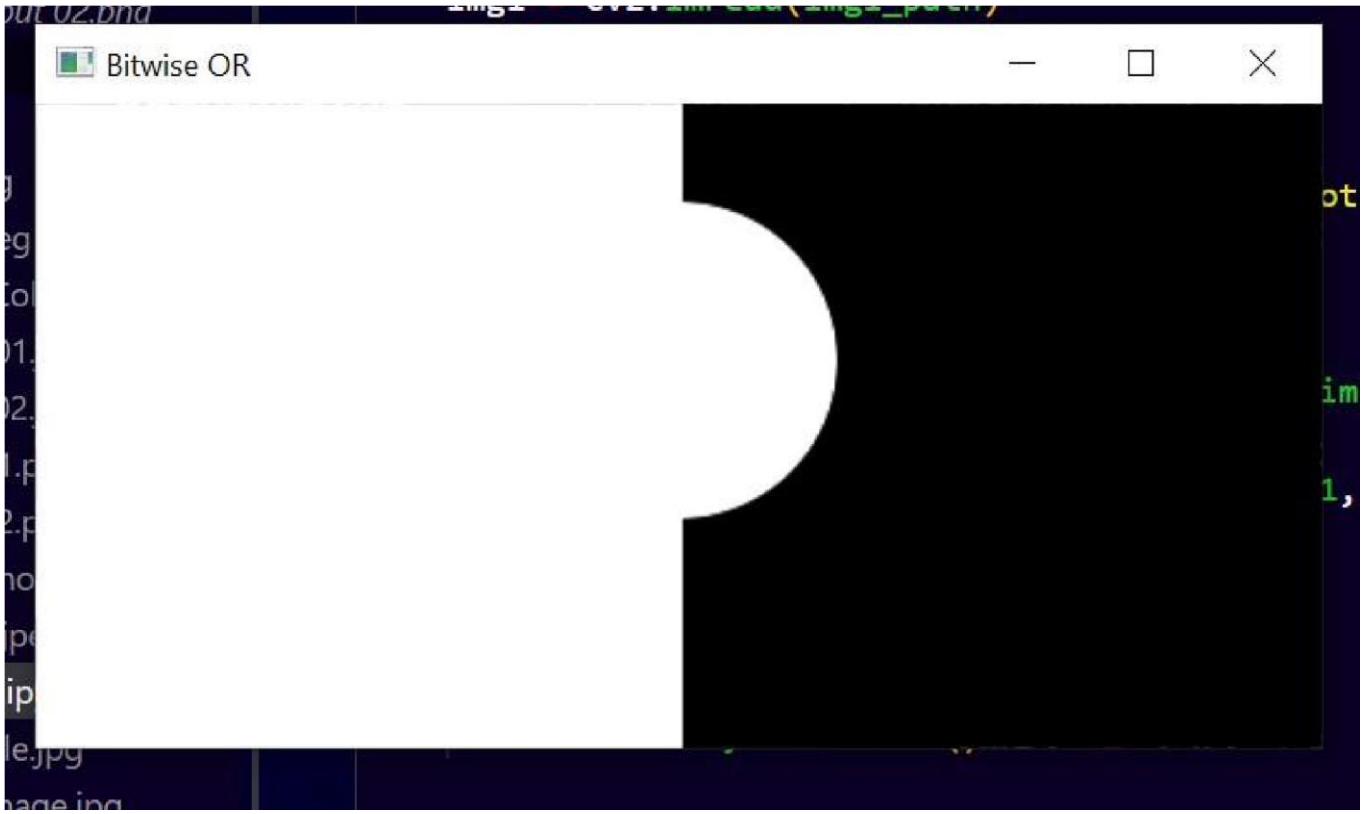




Code explanation:

This program takes two images and combines them using a bitwise OR operation. It first loads the images and checks if they exist. If one or both are missing, the program stops. If the images are not the same size, the second image is resized to match the first one. Then, the bitwise OR operation is applied, which keeps all the bright parts from both images, making them merge in a cool way. The final image is displayed, and when the 'Esc' key is pressed, the window closes.

Output:



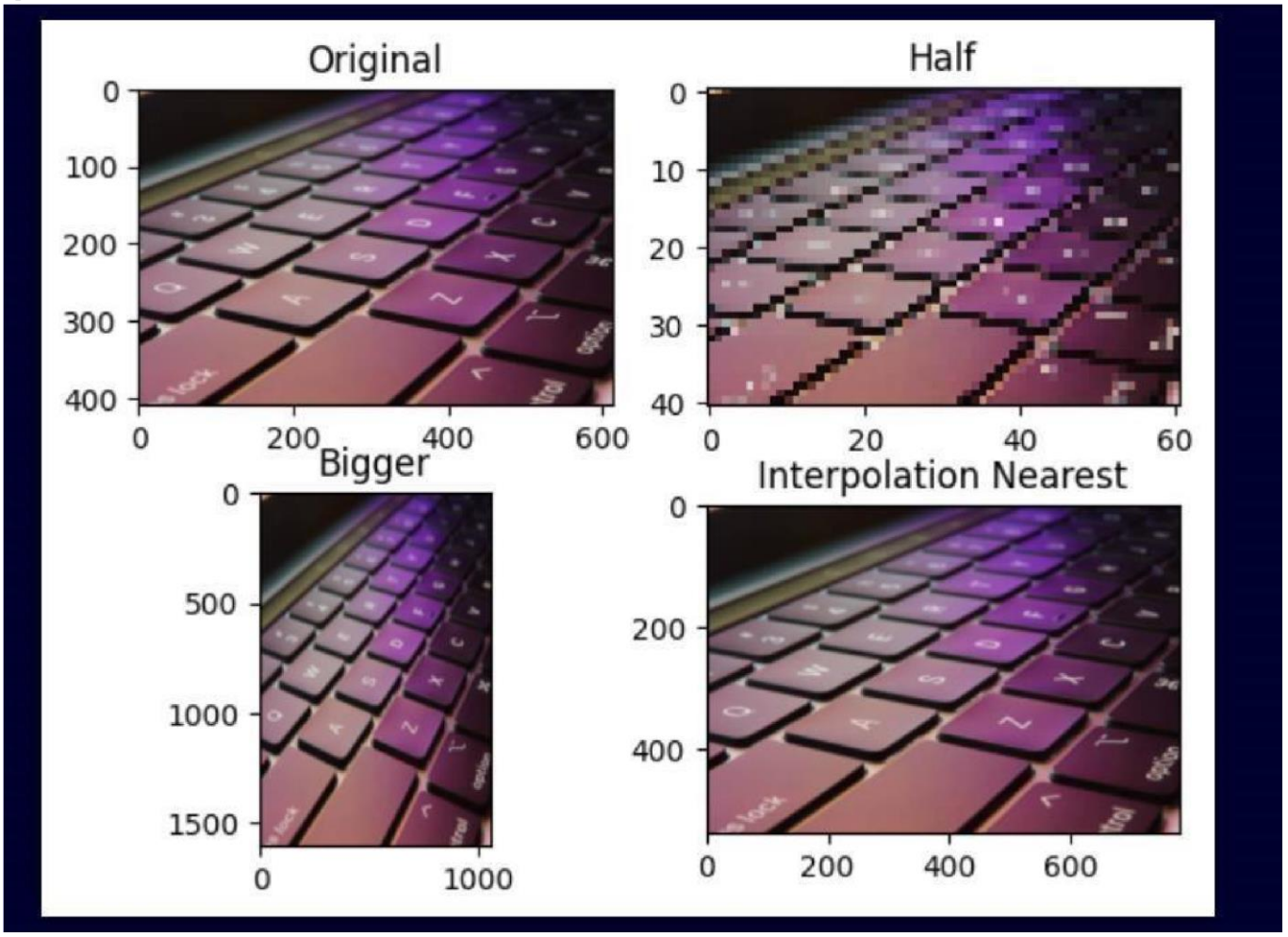
>> 2.2 Image Processing Image Resizing:

Input:

Code Explanation:

This code loads an image from the computer using OpenCV and then resizes it in different ways to see how it looks at different sizes. First, it makes the image 10% smaller than the original. Then, it creates a bigger version with specific dimensions. Another version is resized using a special method called interpolation, which helps keep the quality smooth. After resizing, the code arranges these four images (original, small, big, and smooth-resized) in a 2x2 grid using Matplotlib and displays them all together in one window. This helps compare how different resizing methods affect an image's appearance.

Output:



Eroding an Image:

Input:

Code explanation:

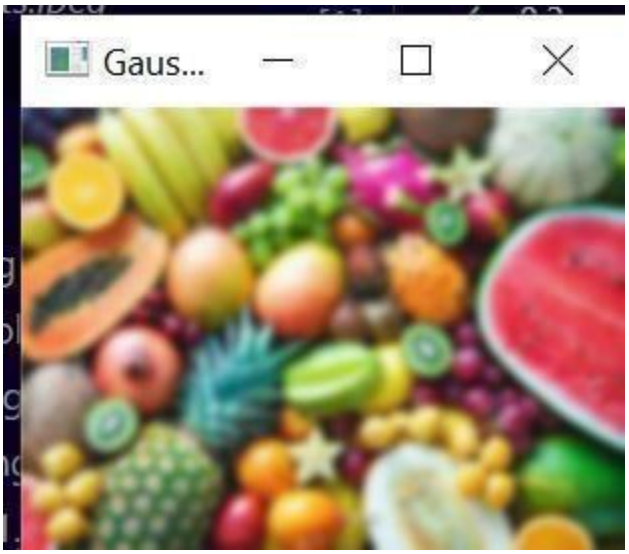
This code loads an image and applies erosion, which means shrinking or thinning bright areas (like white parts) in the image. First, it loads the image using OpenCV. Then, it creates a small 6x6 square (kernel) made of ones, which acts like a stamp that presses down on the image. When erosion is applied, it removes small white spots and makes shapes thinner. This is useful for cleaning up noise or making text in an image clearer. Finally, the code displays the eroded image in a window so you can see the changes.

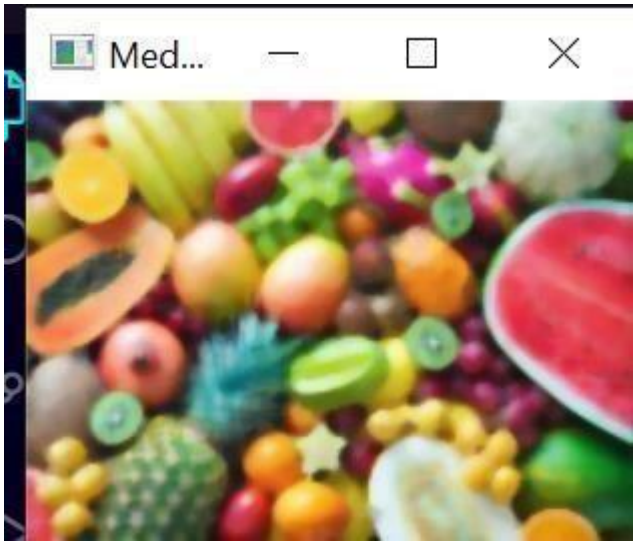
Output:

Blurring an Image Input:

Code Explanation:

This code loads an image from your computer and applies three different types of blurring to it. First, it shows the original image. Then, it applies Gaussian Blur, which makes the image smooth by averaging nearby pixels. Next, it applies Median Blur, which helps remove noise while keeping edges sharp. Finally, it applies Bilateral Blur, which smooths the image but keeps edges clear. Each blurred image is displayed one by one, and the program waits for a key press before moving to the next step. In the end, all windows are closed. **Output:**





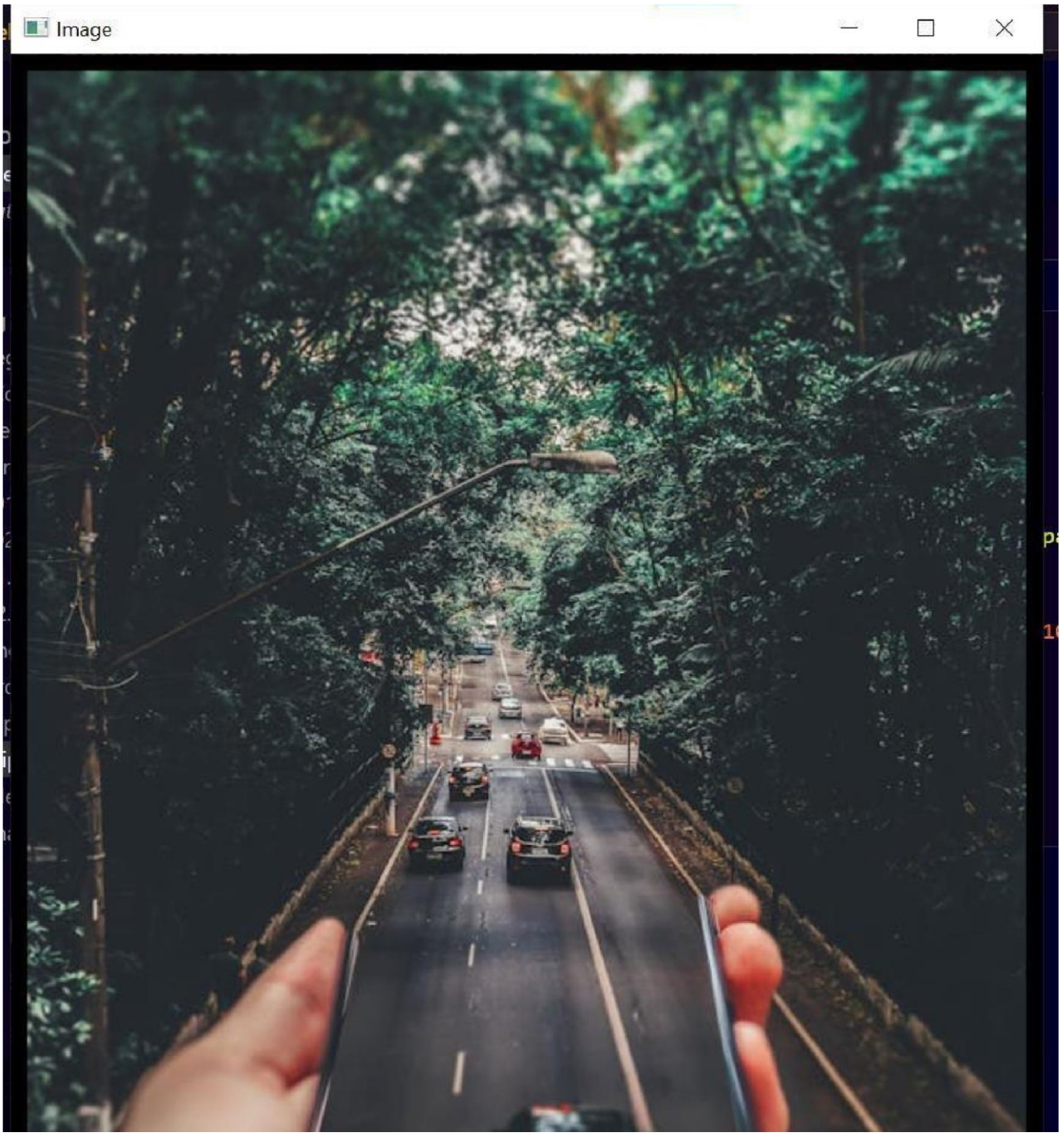
Create Border around Images

Input:

Code Explanation:

This code loads a picture from your computer and adds a black border around it. First, it checks if the image is loaded properly; if not, it shows an error message. Then, it creates a new image with a 10-pixel-wide border on all sides. The border is black because we set the value to 0. Finally, the new image with the border is displayed in a window, and the program waits for you to press a key before closing everything.

Output:

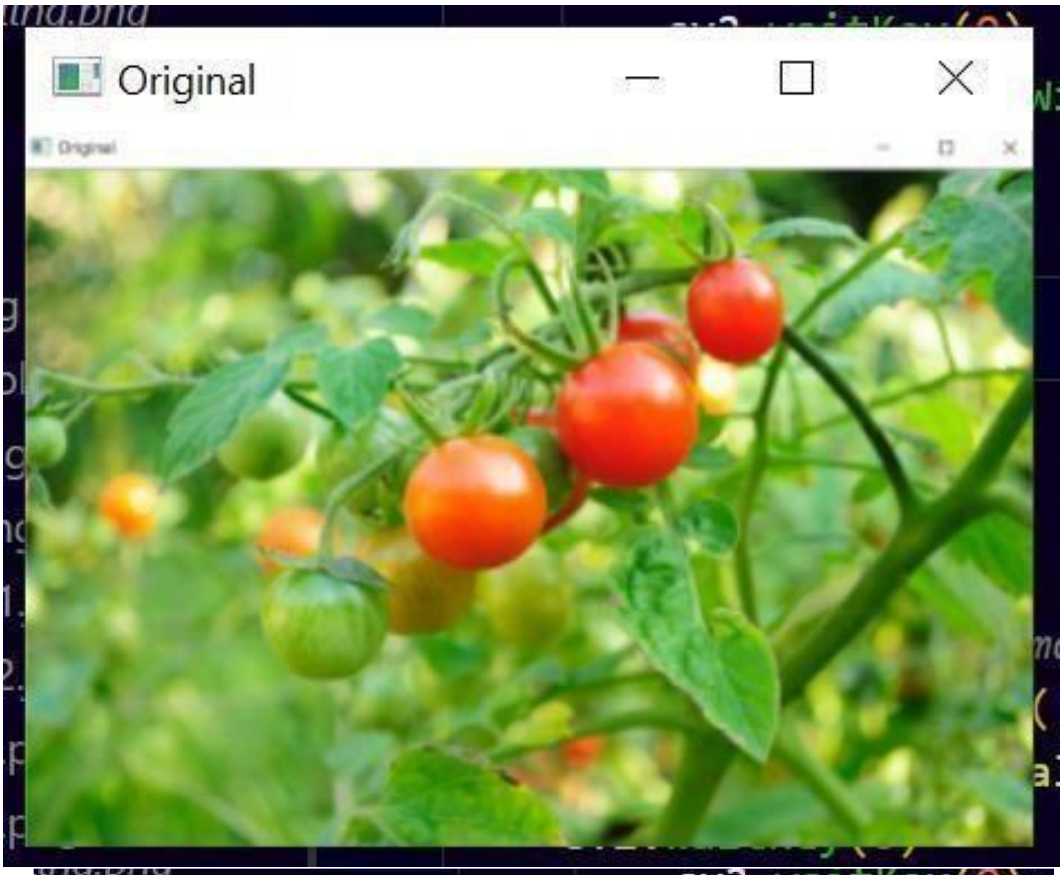


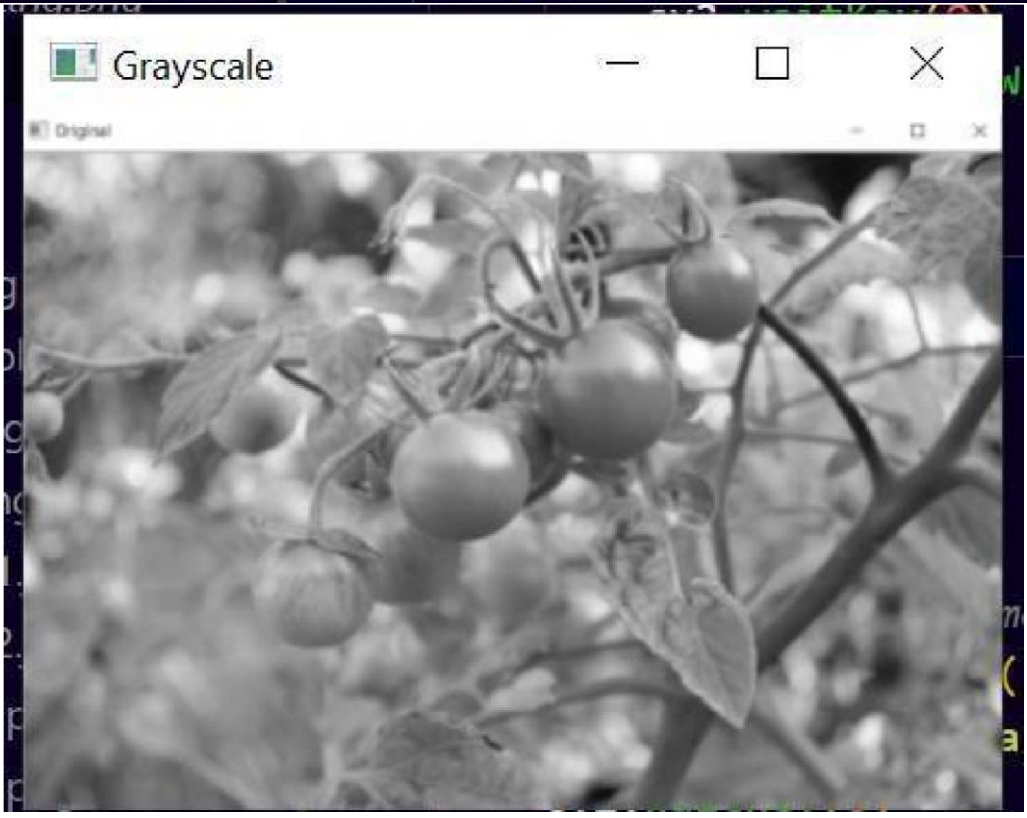
Grayscaling of Images Input:

Code Explanation:

This code opens a picture from your computer and shows it in color first. Then, it changes the picture to black and white (grayscale), which removes all colors and keeps only shades of gray. The new grayscale image is displayed in a window. The program waits for you to press a key before closing everything.

Output:


Original
Original


Grayscale
Original

Scaling, Rotating, Shifting and Edge Detection Image Resizing Input:

Code Explanation:

This code takes a picture of a butterfly from your computer, makes it bigger and smaller, and then shows all three versions side by side. First, it reads the image and changes its colors so that it looks right when displayed. Then, it makes one version of the image three times bigger and another version three times smaller. After that, it arranges the original, bigger, and smaller images next to each other in a row. Finally, it removes any extra lines (ticks) around the pictures and displays them using a graphing tool so you can easily see the differences. If the picture isn't found, it shows an error message.

Output:

Image Rotation Code

Explanation:

This code takes a picture from your computer, spins it by 30 degrees, and shows both the original and rotated images side by side. First, it loads the picture and fixes the colors so they look right. Then, it finds the center of the image and sets up a rotation rule to turn it without changing its size. After that, it applies the rotation and creates two sections to display both images. It removes extra markings around them and finally shows the images neatly arranged for easy comparison. If the picture isn't found, it gives an error message.

Output:

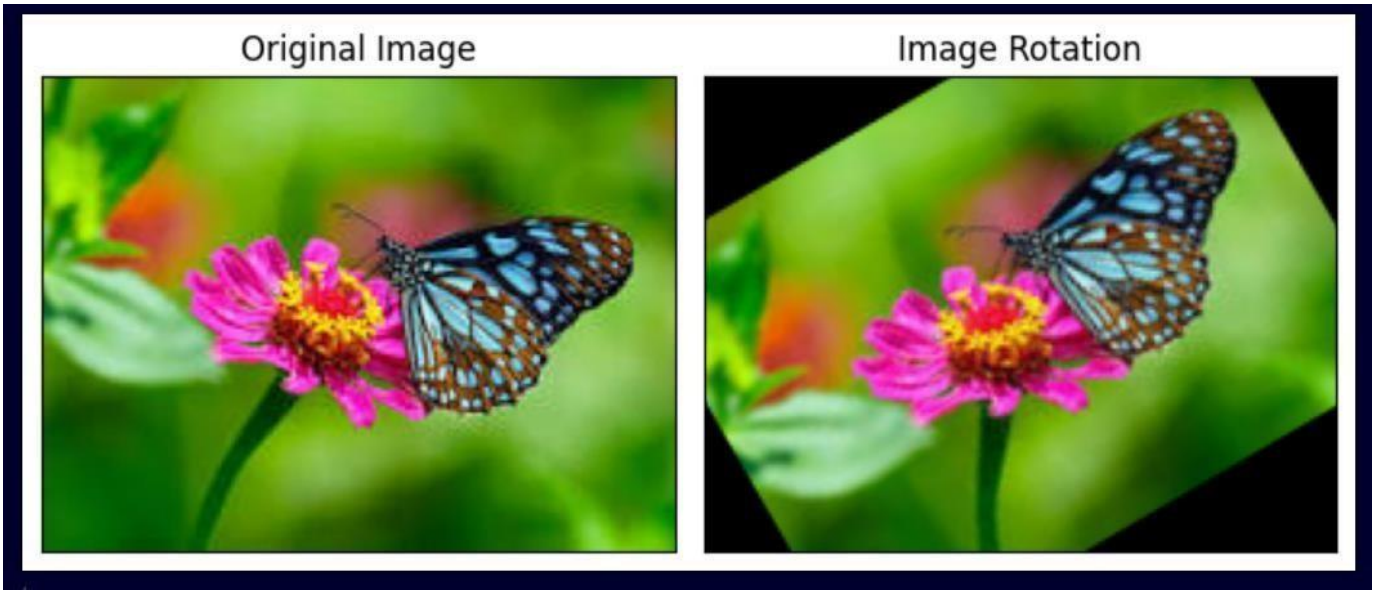


Image Translation Code

Explanation:

This code takes a picture from your computer and moves it to the right by 100 pixels and down by 70 pixels. First, it loads the picture and fixes the colors. Then, it figures out the width and height of the image.

After that, it creates a special rule (a translation matrix) that tells the image how much to shift. The image is then moved to a new position using this rule. Finally, the code shows the original image and the shifted image side by side, removes extra marks around them, and displays them neatly. If the picture isn't found, it shows an error message.

Output:

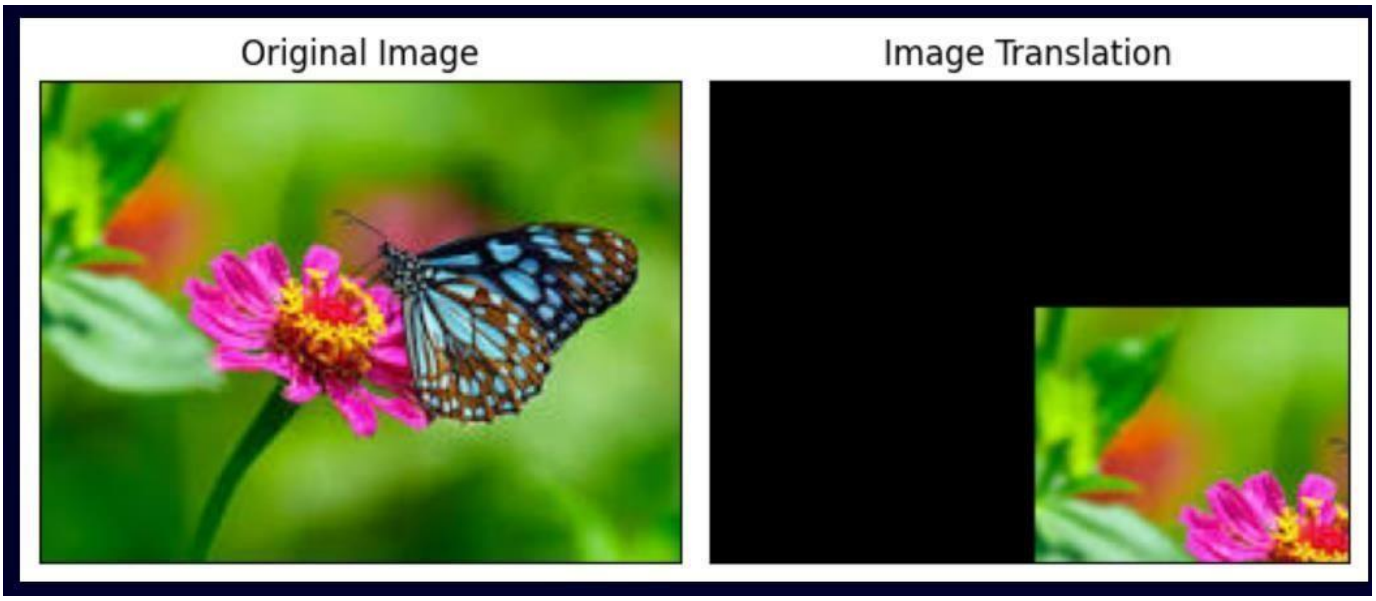


Image Normalization Code Explanation:

This code takes a picture from your computer and makes its colors more balanced. First, it loads the image and fixes the colors so they look right. Then, it splits the image into three color parts: blue, green, and red. Each color part is adjusted to keep the brightness levels between 0 and 1, making the colors look better. After that, the adjusted colors are put back together to form the final image. The code also prints out the blue part of the new image and then shows the improved picture without any extra marks around it. If the image isn't found, it gives an error message.

Output:

Edge detection of Image Code

Explanation:

This code takes a picture from your computer and finds its edges to show only the outlines. First, it loads the image and fixes the colors so they look normal. Then, it uses a special tool called Canny edge detection, which finds the sharp parts in the image where colors change a lot. After that, it creates two sections: one to show the original picture and one to show only the edges in black and white. The code removes extra markings around the images and finally displays both side by side. If the picture isn't found, it gives an error message.

Output:

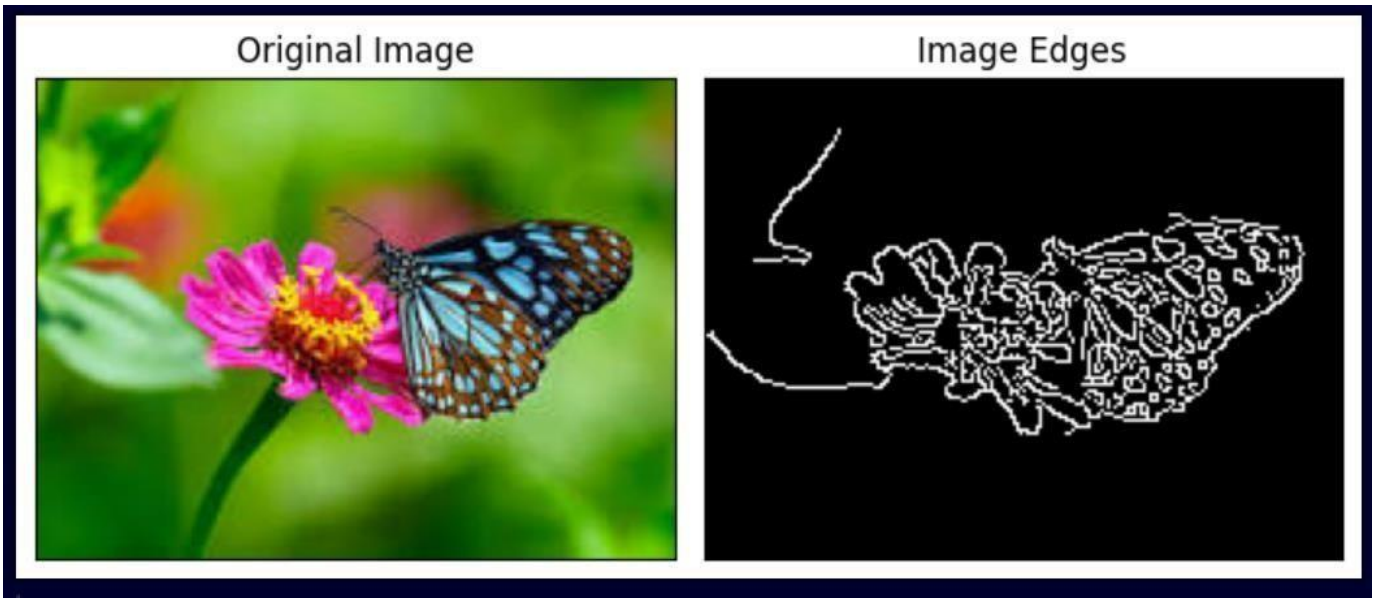


Image Blurring Code

explanation:

This code takes a picture from your computer and makes it look smoother by adding a blur effect. First, it loads the image and fixes the colors to look normal. Then, it applies a soft blur using a special method called Gaussian blur, which helps remove sharp edges and noise. After that, it creates two sections: one to show the original picture and one to show the blurred version. The code removes extra markings around the images and finally displays them side by side. If the picture isn't found, it shows an error message.

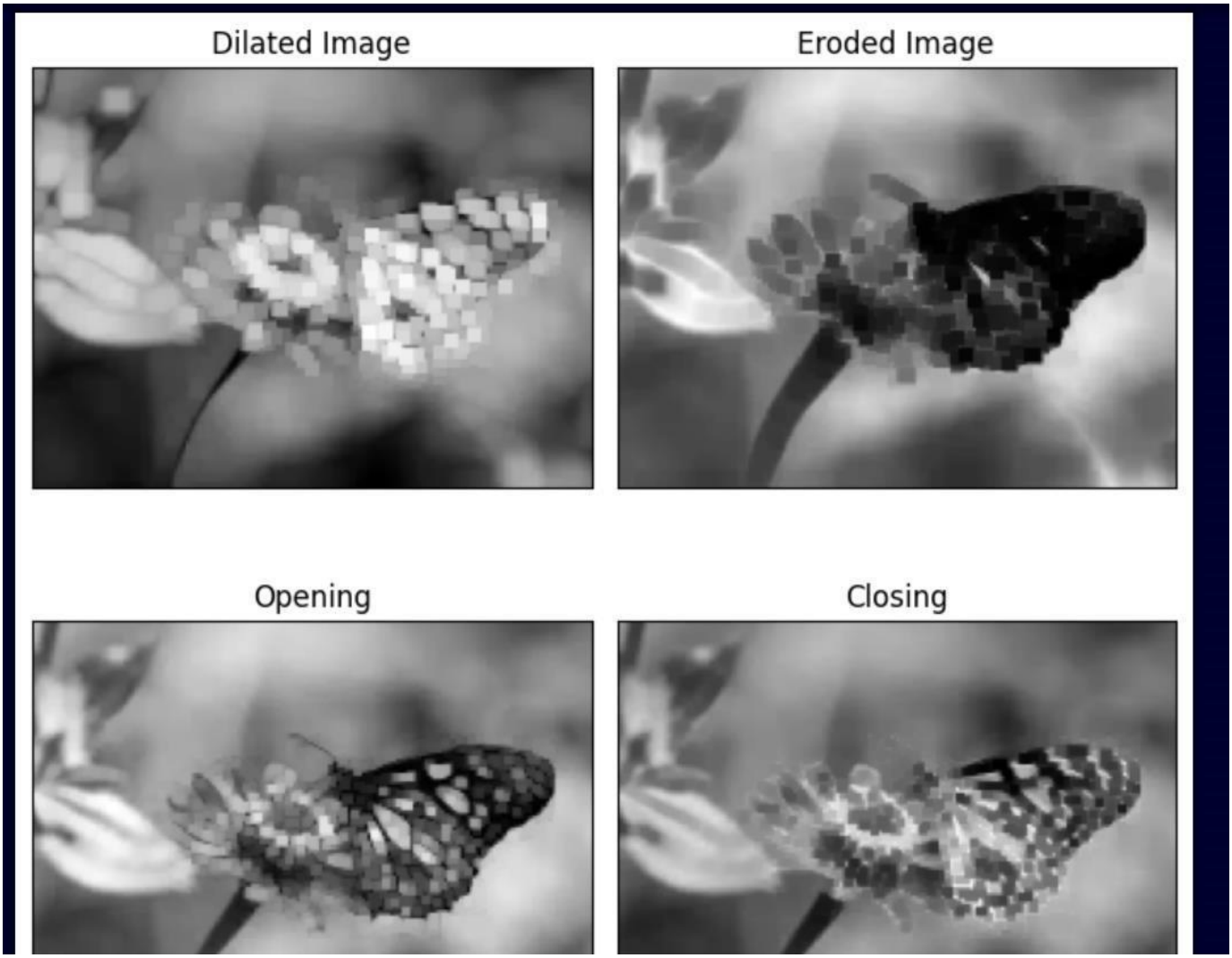
Output:

Morphological Image Processing

Code Explanation:

This code takes a picture and changes it in different ways to show how shapes can be modified. First, it loads the image and converts it to black and white (grayscale). Then, it creates a small box (kernel) that helps in changing the image. It performs four changes: **dilation** (making bright areas thicker), **erosion** (making bright areas thinner), **opening** (removing small noise by first eroding and then dilating), and **closing** (filling small gaps by first dilating and then eroding). Finally, it shows all four versions in a grid so you can compare them. If the image is missing, it shows an error message instead.

Output:



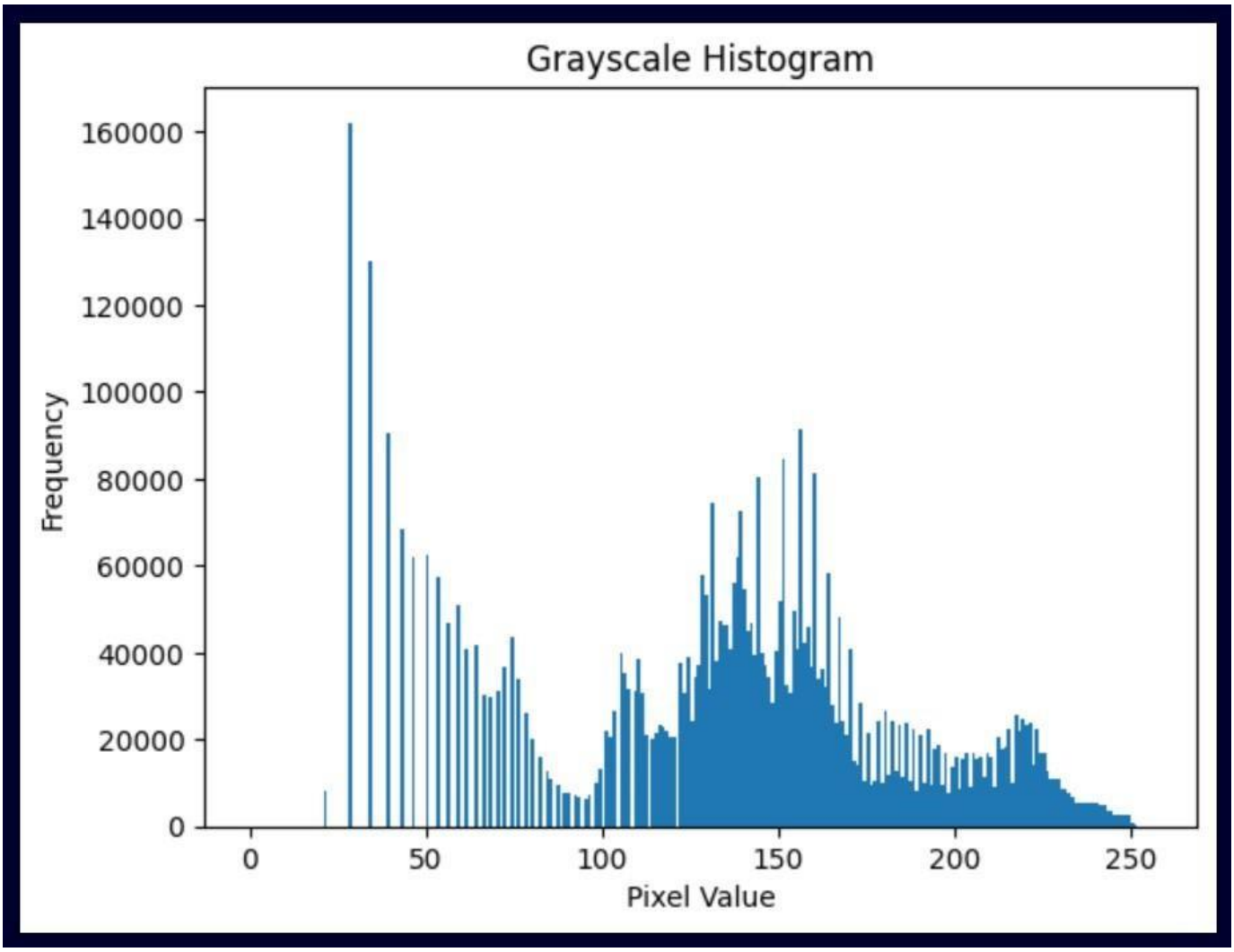
Analyze an image using Histogram

Input:

Code Explanation:

This code takes a black-and-white (grayscale) picture and counts how many times each shade of gray appears in it. First, it loads the image from the computer. If the image is missing, it shows an error message. Then, it looks at all the tiny dots (pixels) in the image and counts how many are dark, light, or in between. It makes a graph (histogram) to show this count, where the left side is for dark pixels and the right side is for bright ones. Finally, it displays the graph so you can see the brightness levels in the image!

Output:



Simple Thresholding: Input:

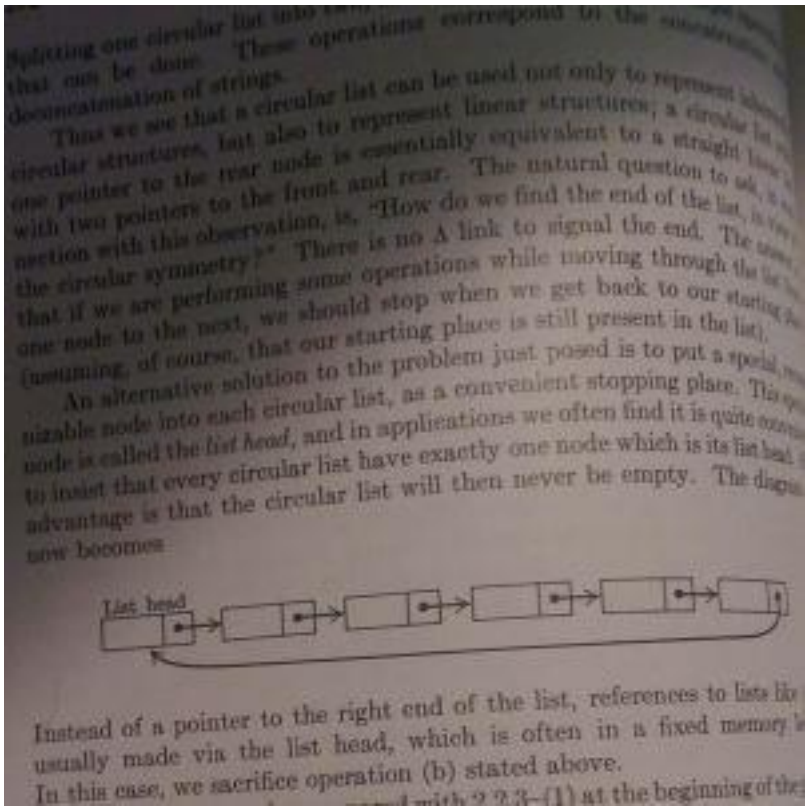
Code Explanation: This code loads a picture and changes it into black and white (grayscale). Then, it applies different types of filters (thresholding) to the image. Each filter checks the brightness of each pixel: if it's brighter than 120, it changes to white (255), and if it's darker, it changes to black (0) or something else based on the method. The program shows five different versions of the image with these

filters, like making parts of the image disappear or keeping only bright areas. Finally, it waits until a key is pressed before closing all the image windows.

Output:

Adaptive Thresholding

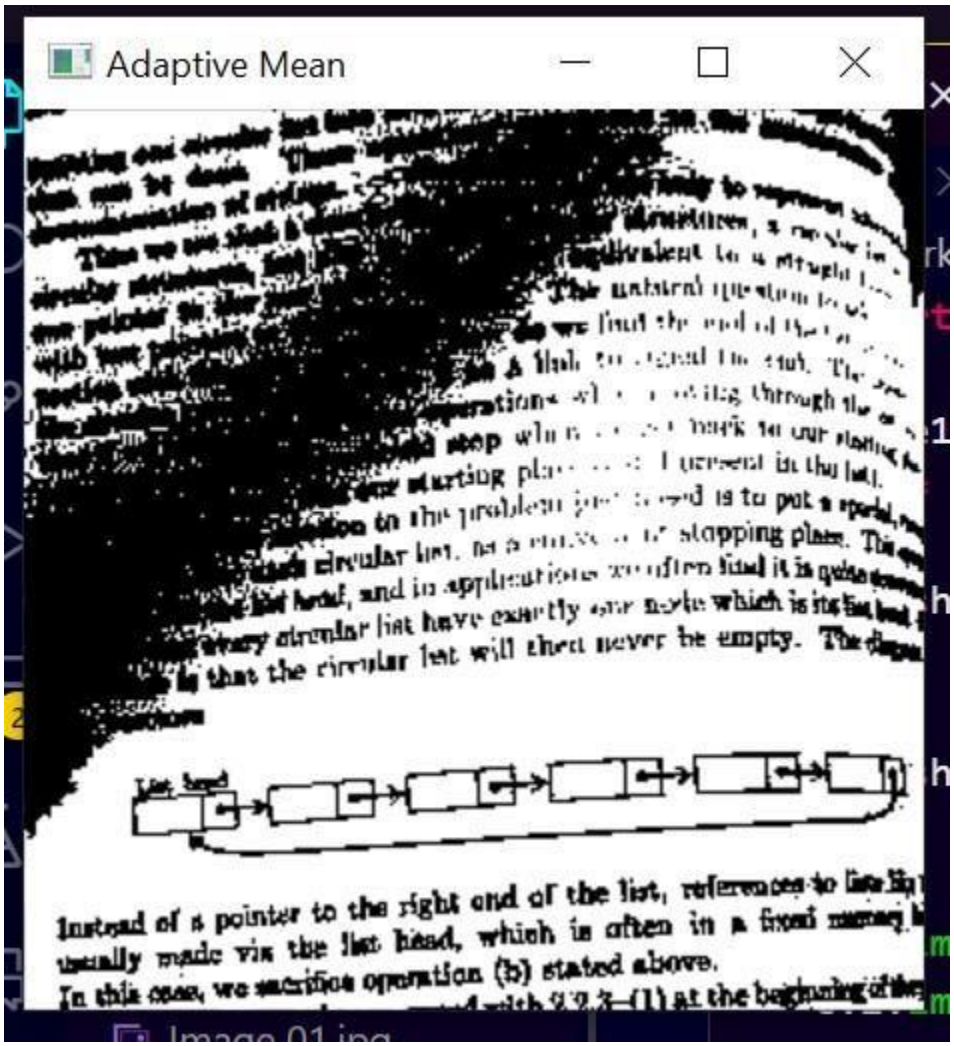
Input:

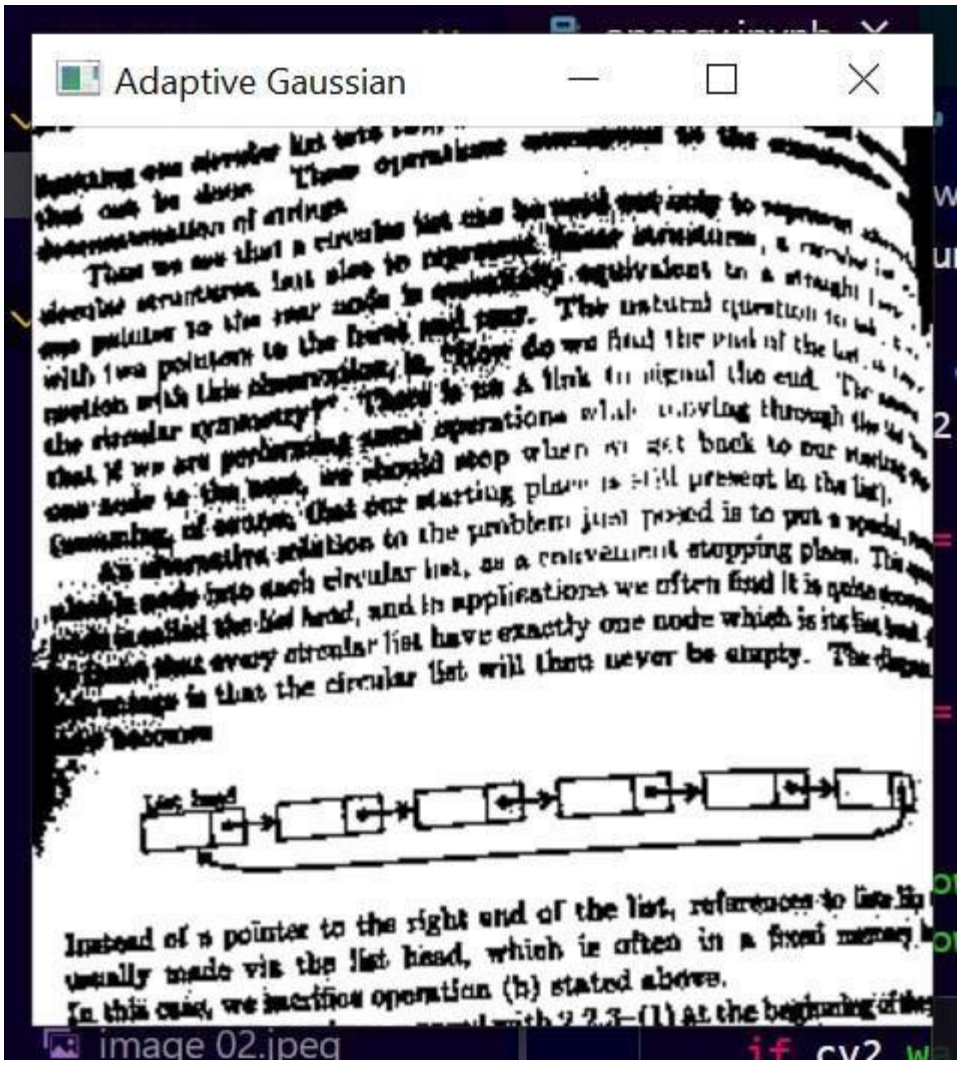


Code Explanation:

This code takes a picture and turns it into a black-and-white version using something called "adaptive thresholding." First, it loads the image and converts it into grayscale (which means no colors, just shades of gray). Then, it applies two different methods to decide which parts of the image should be white and which should be black. The first method (Adaptive Mean) looks at the average brightness of nearby pixels, while the second method (Adaptive Gaussian) gives more importance to pixels closer to the center. Finally, it shows both results in separate windows, and the program waits until you press a key to close them.

Output:


Adaptive Mean
List head
Instead of a pointer to the right end of the list, references to list
usually made via the list head, which is often in a fixed memory
In this case, we sacrifice operation (b) stated above.

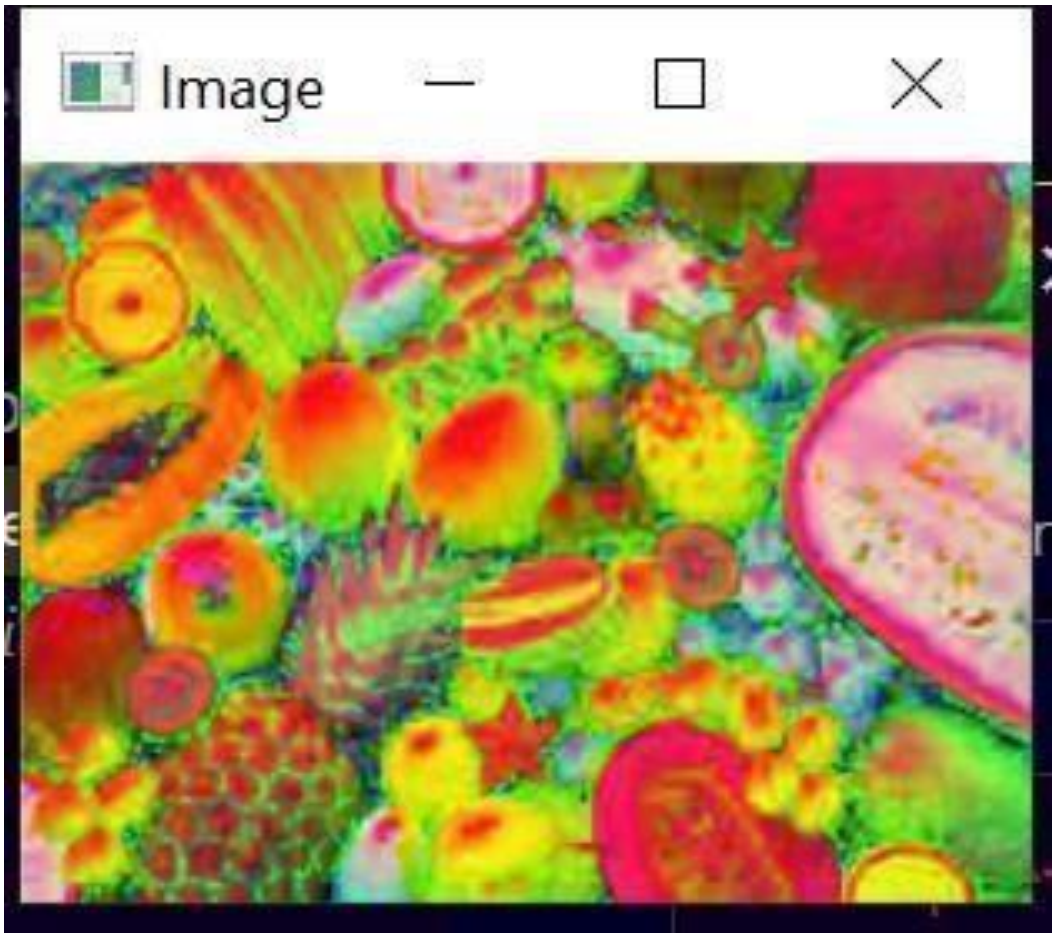


Convert an image from one color space to another Input:

Code Explanation:

This Python program uses the `cv2` library to read and display an image in a different color format. First, it sets the image's file path and tries to load it using `cv2.imread()`. If the image can't be loaded (maybe the file is missing or the path is wrong), it prints an error message. If the image loads successfully, it converts the colors from BGR (Blue-Green-Red) to HSV (Hue-Saturation-Value) using `cv2.cvtColor()`. Then, it shows the modified image in a window. The program waits until the user presses a key before closing the image window properly.

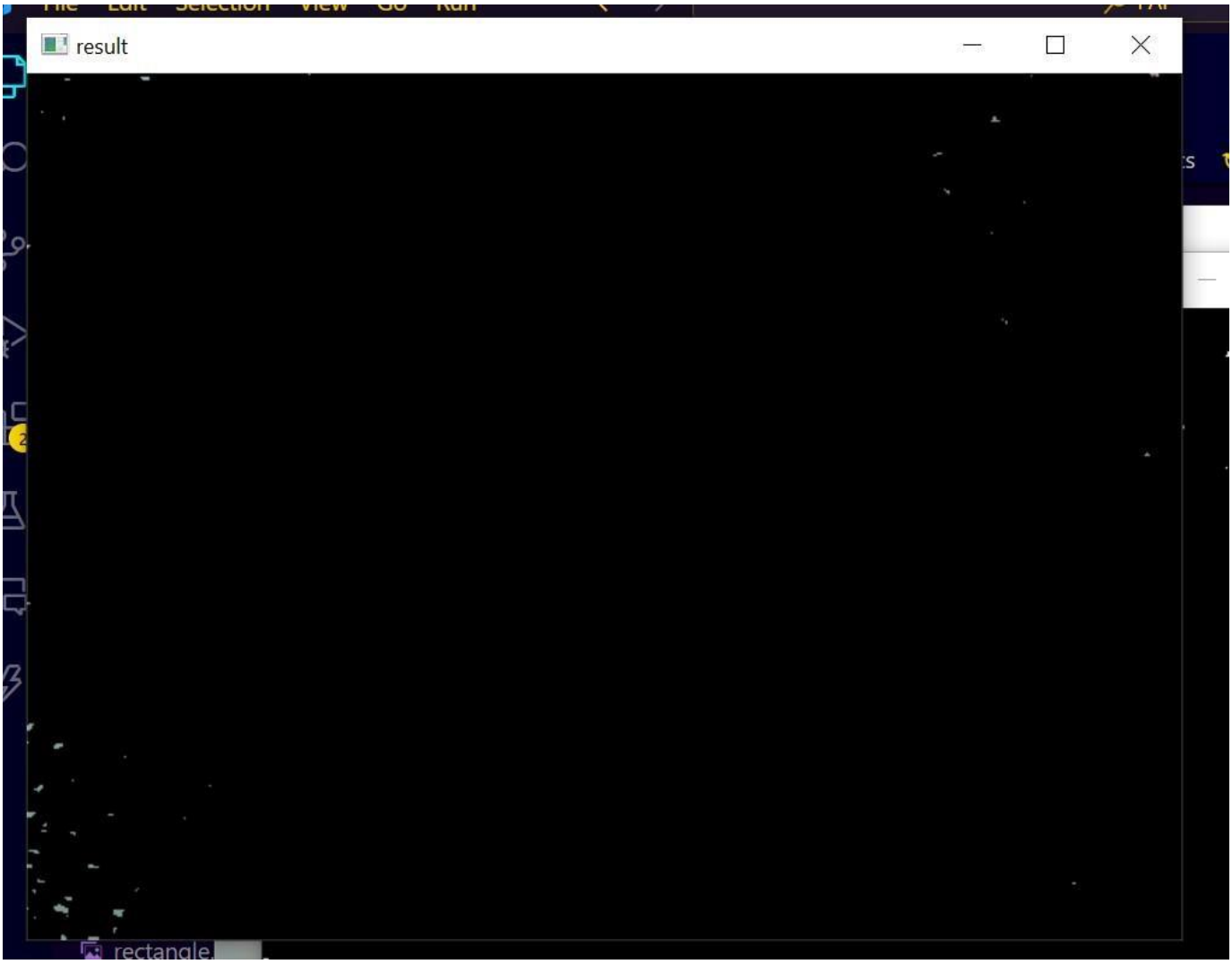
Output:



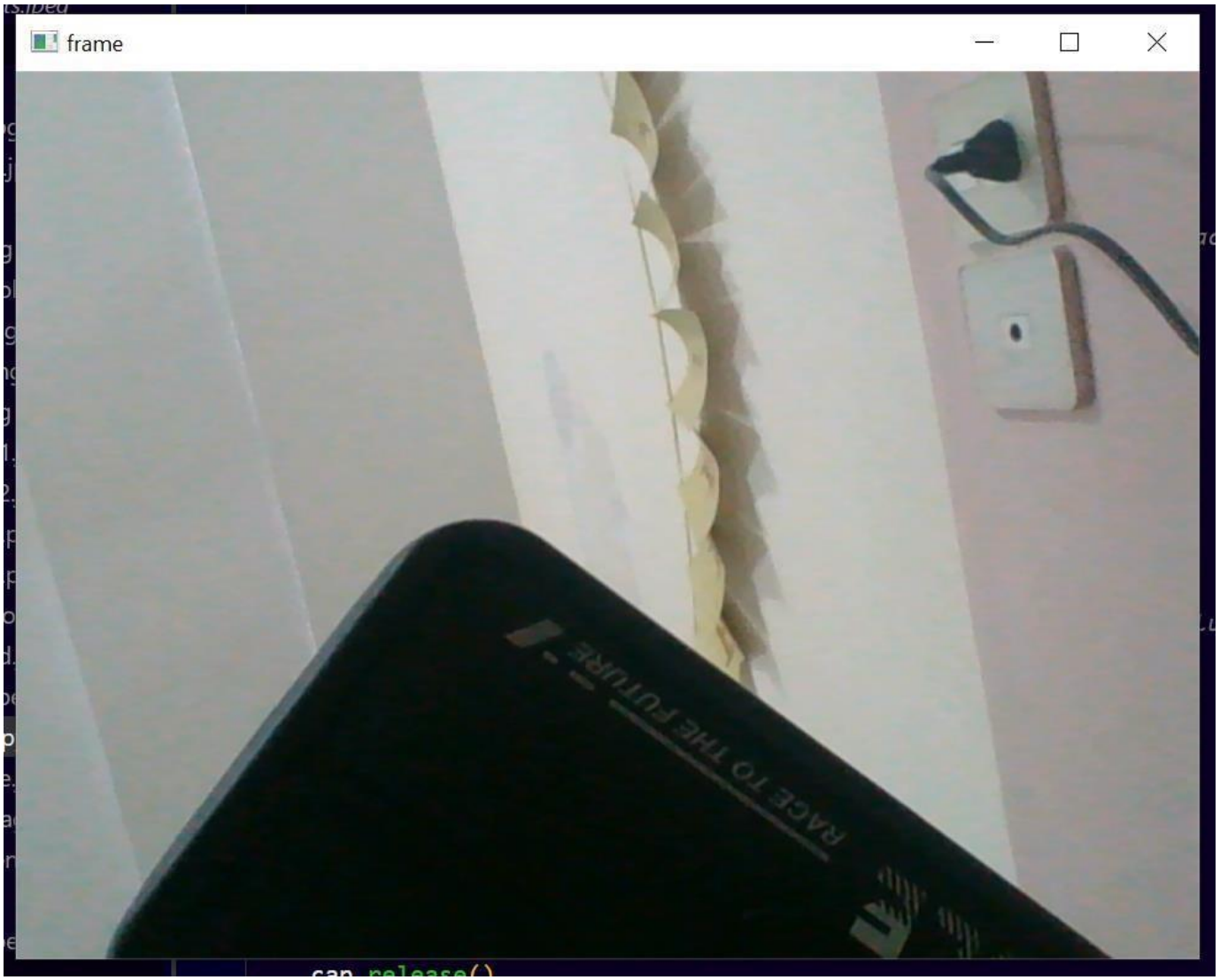
Filter Color with OpenCV

Code Explanation:

Output:


result
rectangle


mask



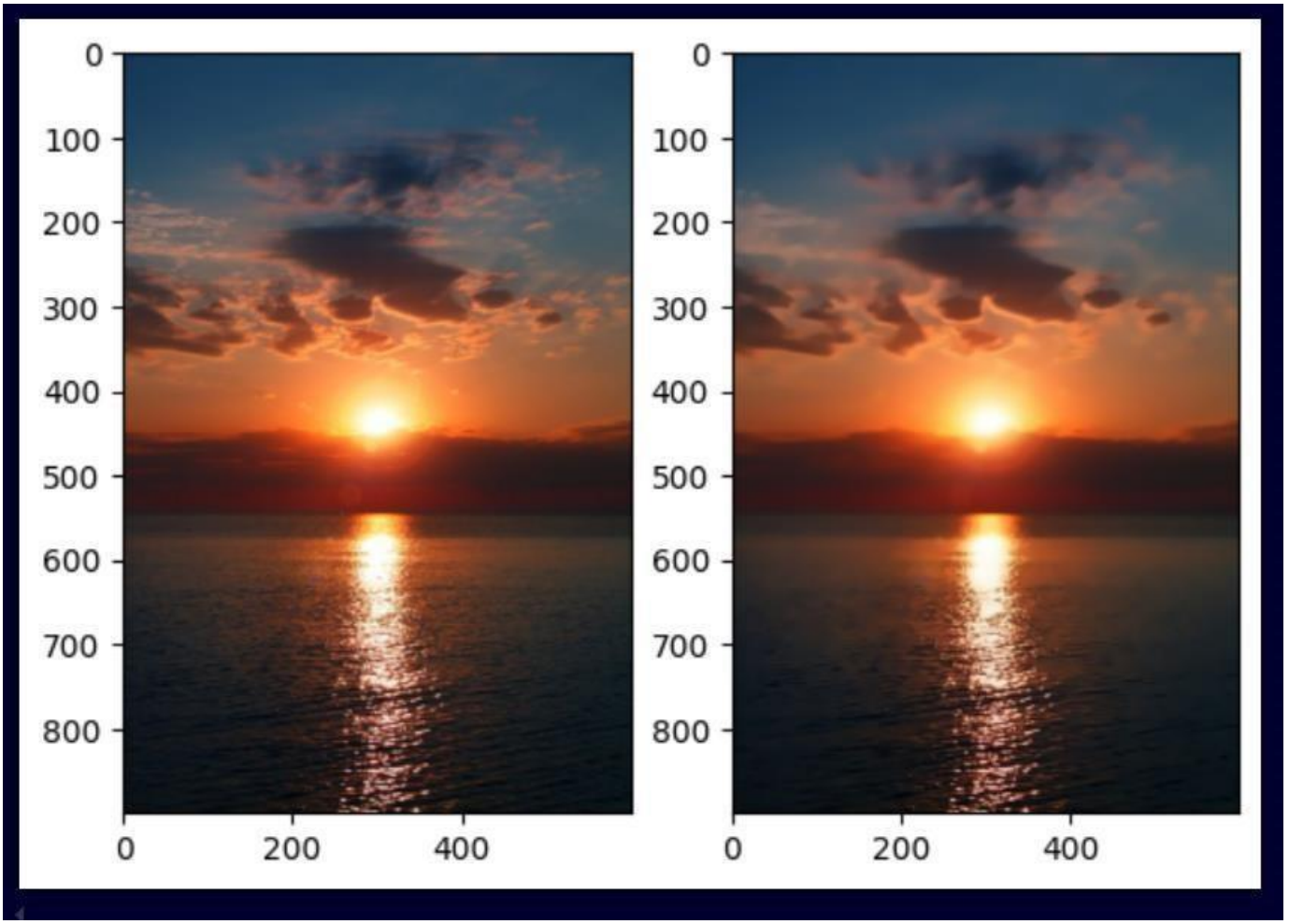
Denoising of colored images

Input:

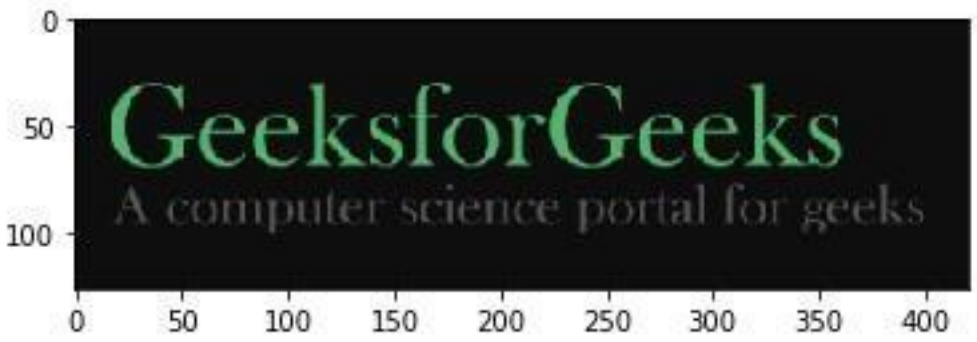
Code Explanation:

This Python program removes noise from an image using the `cv2` and `matplotlib` libraries. First, it loads the image from the given path using `cv2.imread()`. If the image isn't found, it prints an error message. If the image loads successfully, it applies a noise-reduction technique called `fastNlMeansDenoisingColored()`, which smooths out unwanted grainy spots while keeping details clear. The program then displays both the original and the denoised images side by side using `matplotlib`. Since OpenCV reads images in BGR format, the colors are converted to RGB before displaying them correctly.

Output:



Visualizing image in different color spaces Input:



Code Explanation:

This Python program loads and displays an image in grayscale using the `cv2` library. First, it tries to read the image from the given path in grayscale mode using `cv2.imread()`. If the image is not found, it prints an error message. If the image loads successfully, it opens a window and shows the grayscale image using `cv2.imshow()`. The program then waits for the user to press a key before closing the window with `cv2.destroyAllWindows()`. This makes sure the image stays open until the user decides to close it.

Output:

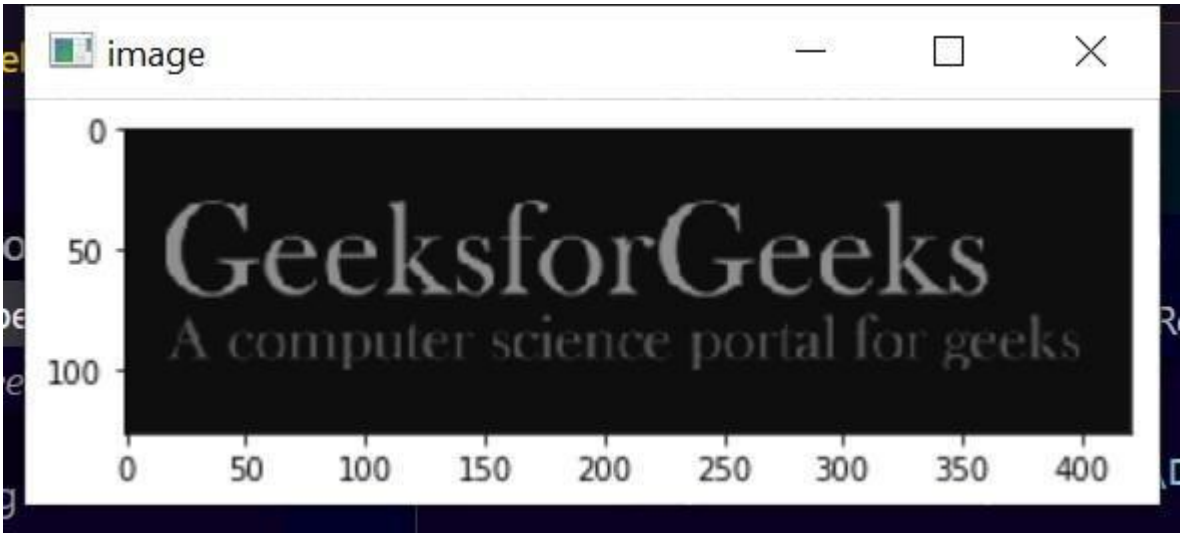
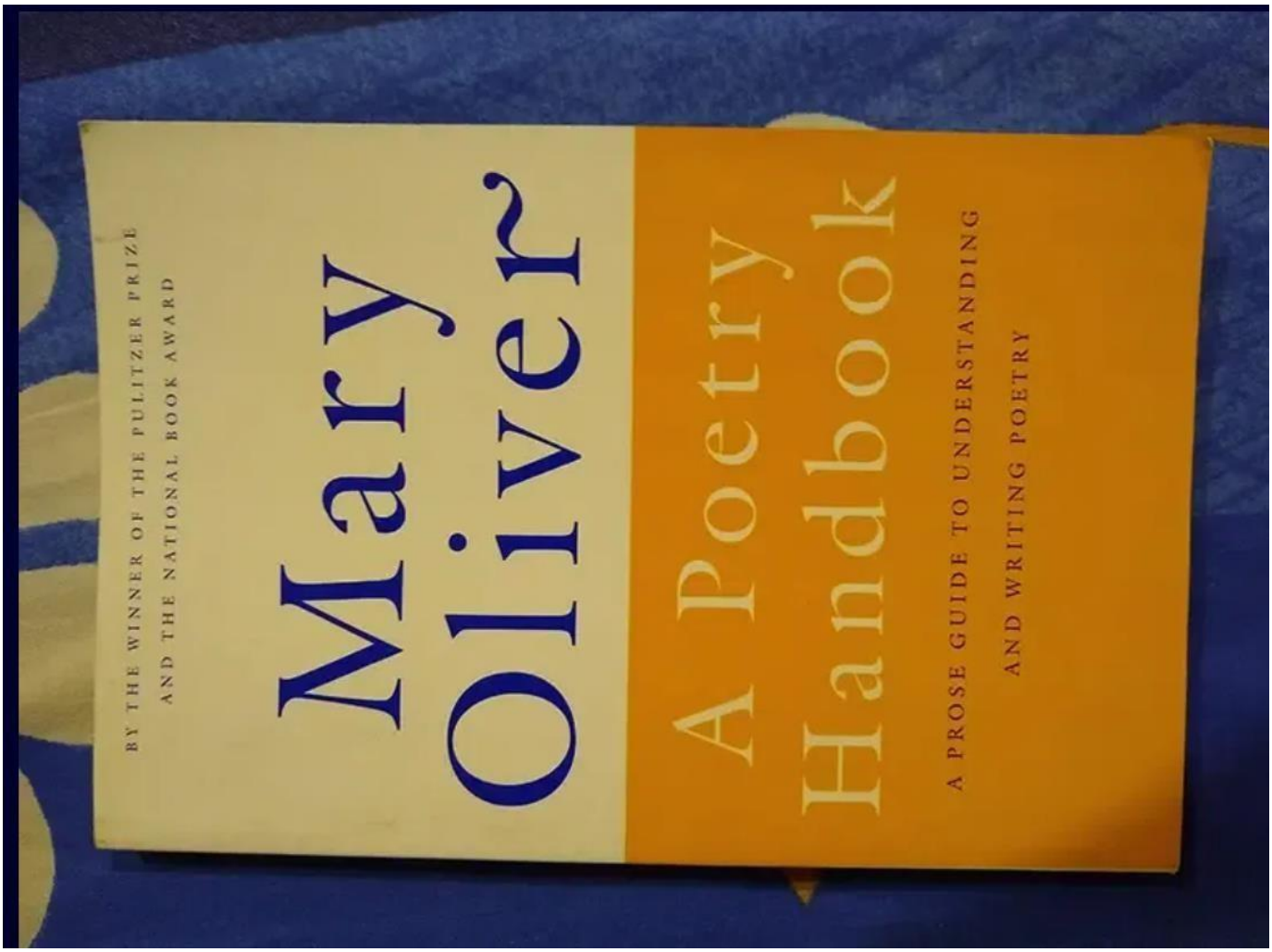


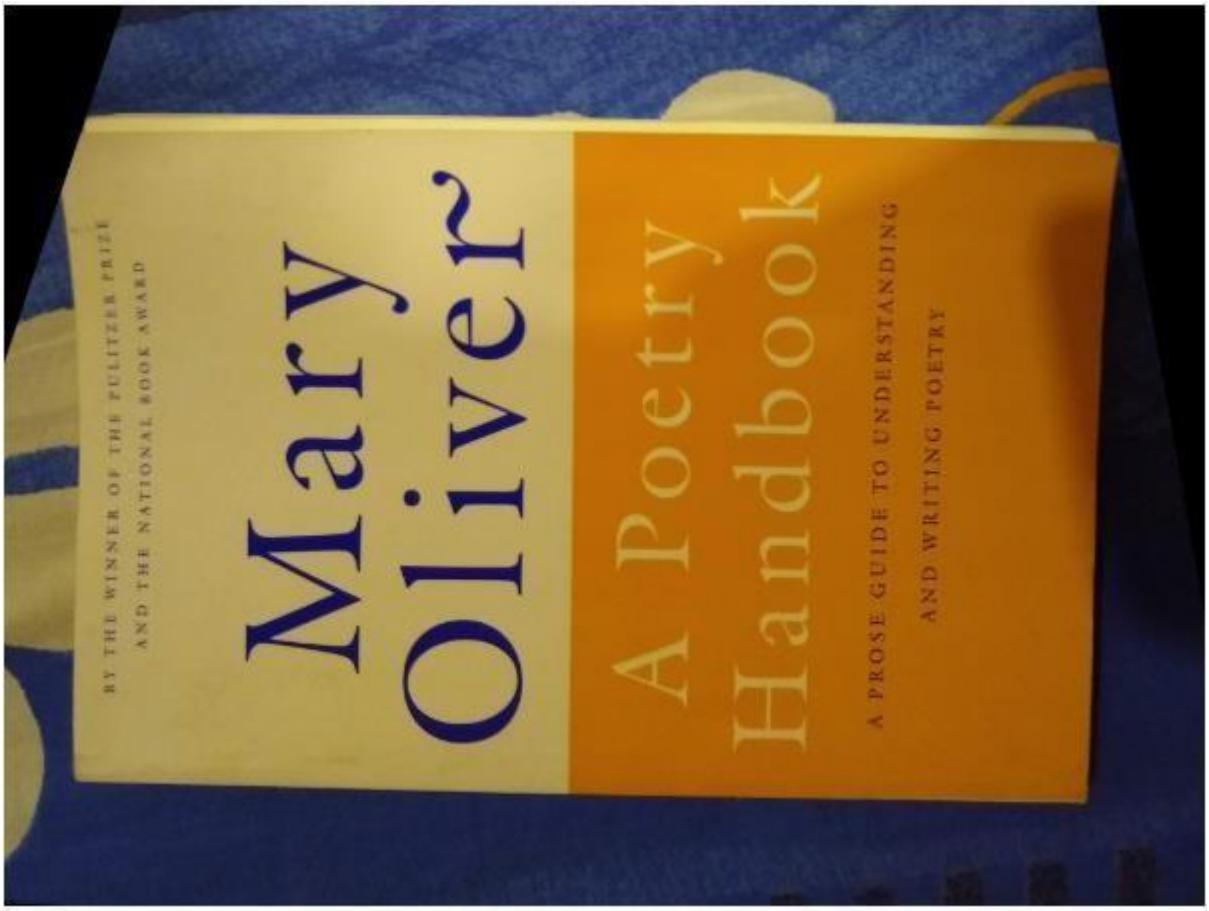
Image Registration Input:



Code Explanation:

This Python program aligns two images using feature matching and homography transformation. It first loads the images from the specified file paths and checks if they are successfully loaded. If the images are valid, it converts them to grayscale for processing. Then, it uses an ORB (Oriented FAST and Rotated BRIEF) detector to detect key points and extract feature descriptors from both images. The program matches these features using a Brute Force matcher with the Hamming distance as the comparison method. It sorts the matches based on their accuracy and selects the top 90% for better alignment. Using these matched points, it calculates a homography matrix with the RANSAC algorithm to find the best transformation between the two images. Finally, it applies this transformation to align the first image with the second and saves the aligned image as `output.jpg`.

Output:

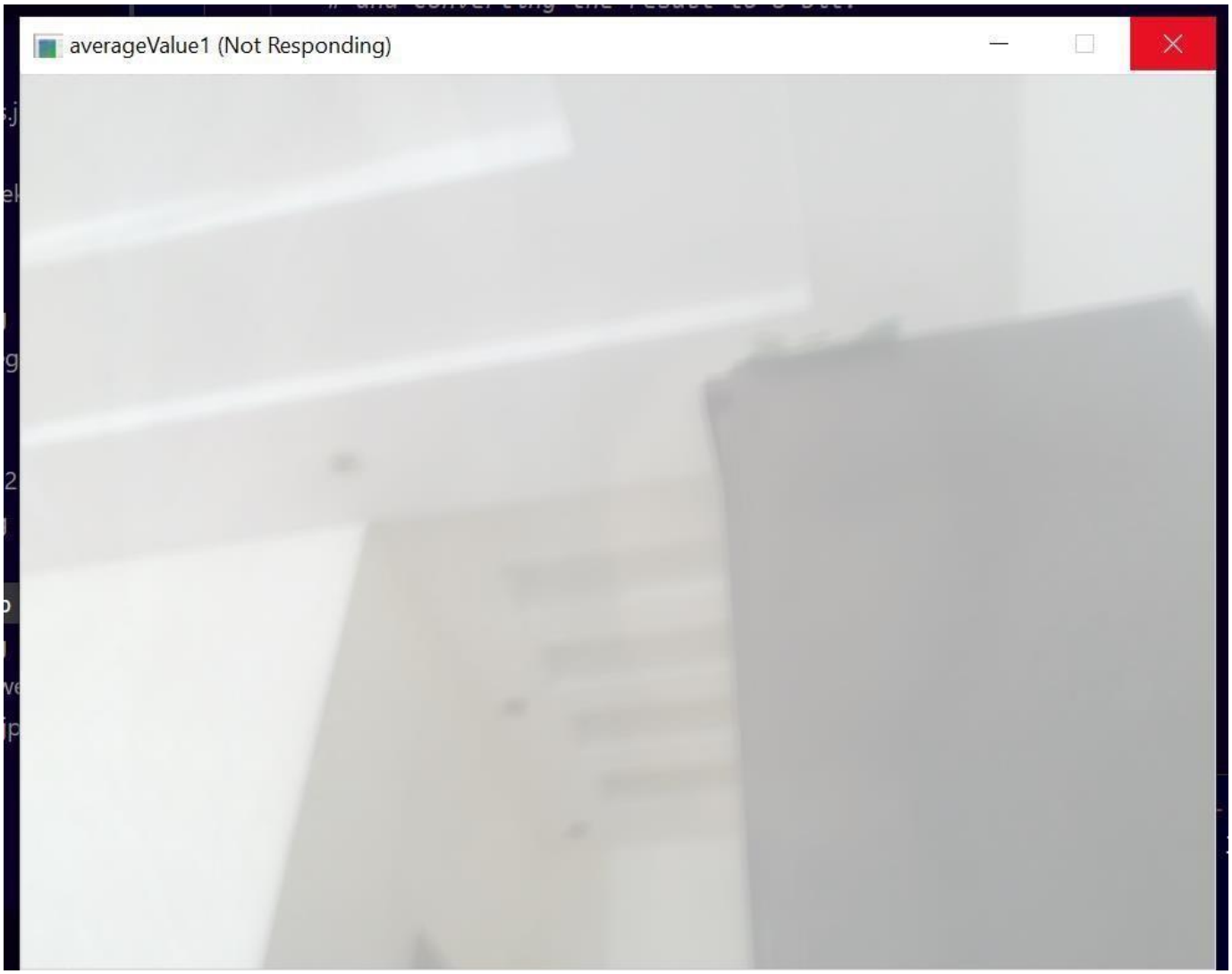


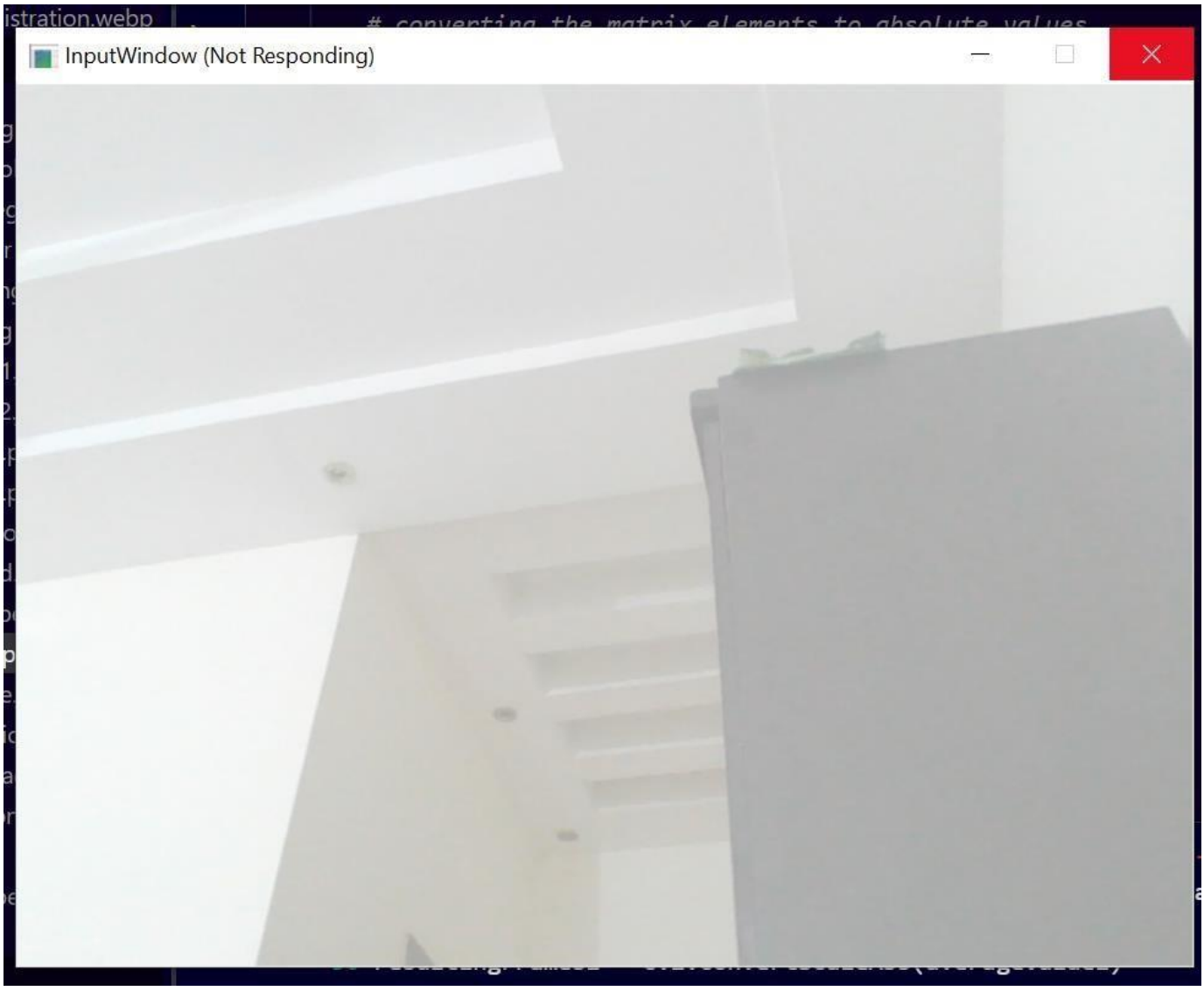
Background Subtraction in an Image using Concept of Running Average

Code Explanation:

This Python program uses OpenCV to perform background subtraction using the concept of running averages. It starts by capturing video frames from the camera and initializes a floating-point matrix to store the running average of the frames. In a loop, it continuously reads frames from the camera and updates the running average using `cv2.accumulateWeighted()`, which gradually blends the new frames into the background model. The accumulated result is then converted into an 8-bit image using `cv2.convertScaleAbs()`. The program displays both the original live camera feed and the processed background model in separate windows. It runs until the user presses the 'Esc' key, after which it releases the camera and closes all OpenCV windows.

Output:


averageValue1 (Not Responding)

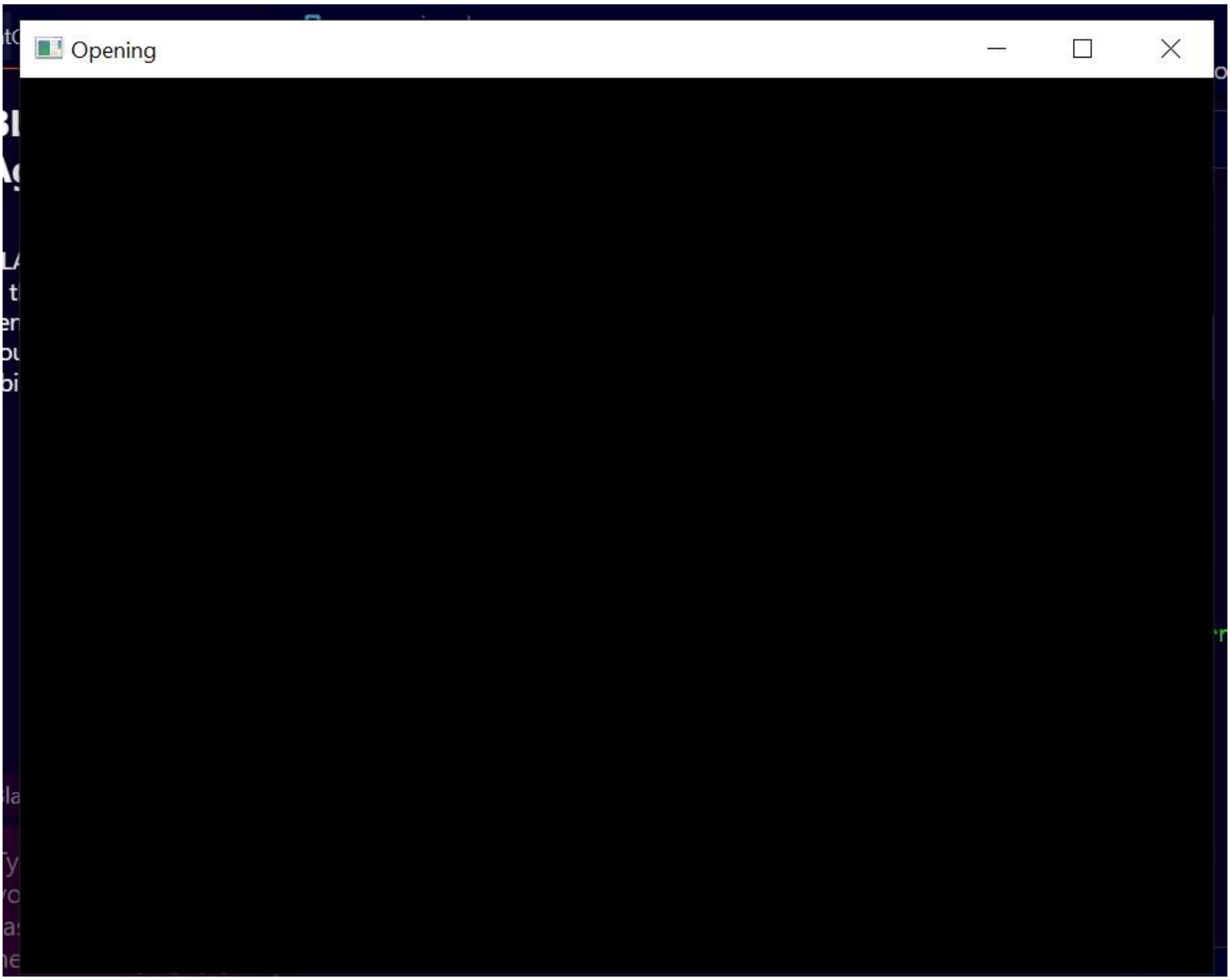


Morphological Operations in Image Processing (Opening) Code

Explanation:

This code is like a little robot that looks through a camera and tries to find the color blue. First, it turns on the camera and keeps looking at what it sees. Then, it changes the colors in the image to a special way that makes it easier to find blue. After that, it checks every part of the image and picks out the parts that are blue. It makes a mask, which is like a magic filter that only keeps the blue parts and hides the rest. To make sure the blue parts are clean and not messy, it smooths them out a bit. Finally, it shows two pictures—one with just the blue parts and another where the blue is cleaned up. The program keeps running until you press the 'a' key, and then it stops, turns off the camera, and closes everything.

Output:


Opening

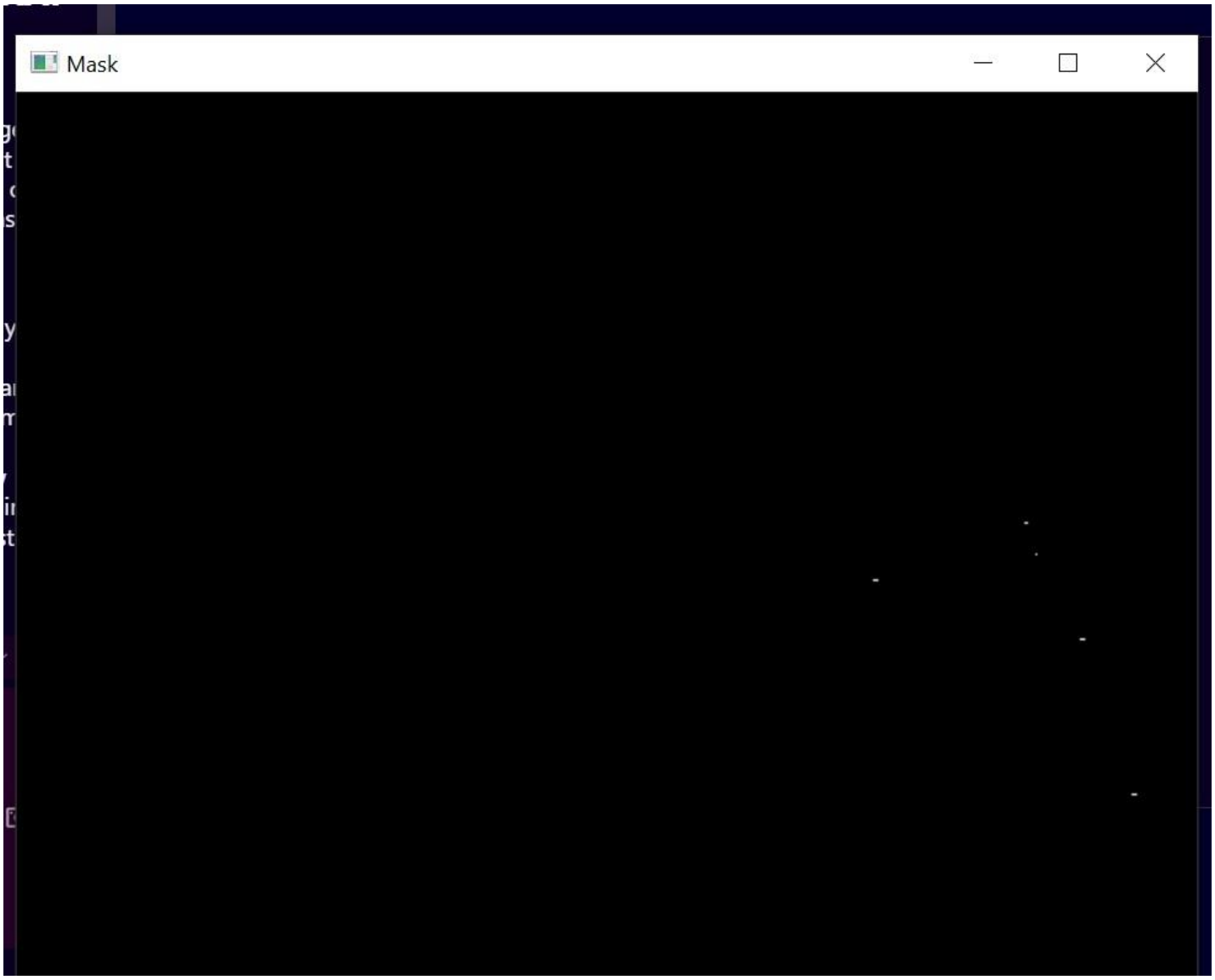


Image segmentation using Morphological operations Input:


भारत INDIA
2002
INDIA
सत्यमेव जयते
2002
रुपया

Code Explanation:

This code is like a magic trick for pictures! First, it takes a picture from your computer and reads it. Then, it turns the picture into black and white (grayscale) so it's easier to work with. After that, it looks at all the colors and decides which parts should be completely white and which should be completely black, like a coloring book! This helps highlight important parts of the picture, like shapes or objects. Finally, it shows the new black-and-white picture on the screen and waits for you to press a key before closing everything.

Output:

Image Translation Input:

Code Explanation:

This code is like moving a picture on the screen! First, it picks a picture from your computer and reads it. Then, it checks how big the picture is by finding its height and width. After that, it decides to move the picture a little bit to the right and a little bit down—exactly one-fourth of its size. It creates a special "move" instruction and uses it to shift the whole picture. Finally, it shows both the original picture and the moved picture in two windows. It waits until you press a key before closing everything.

Output:

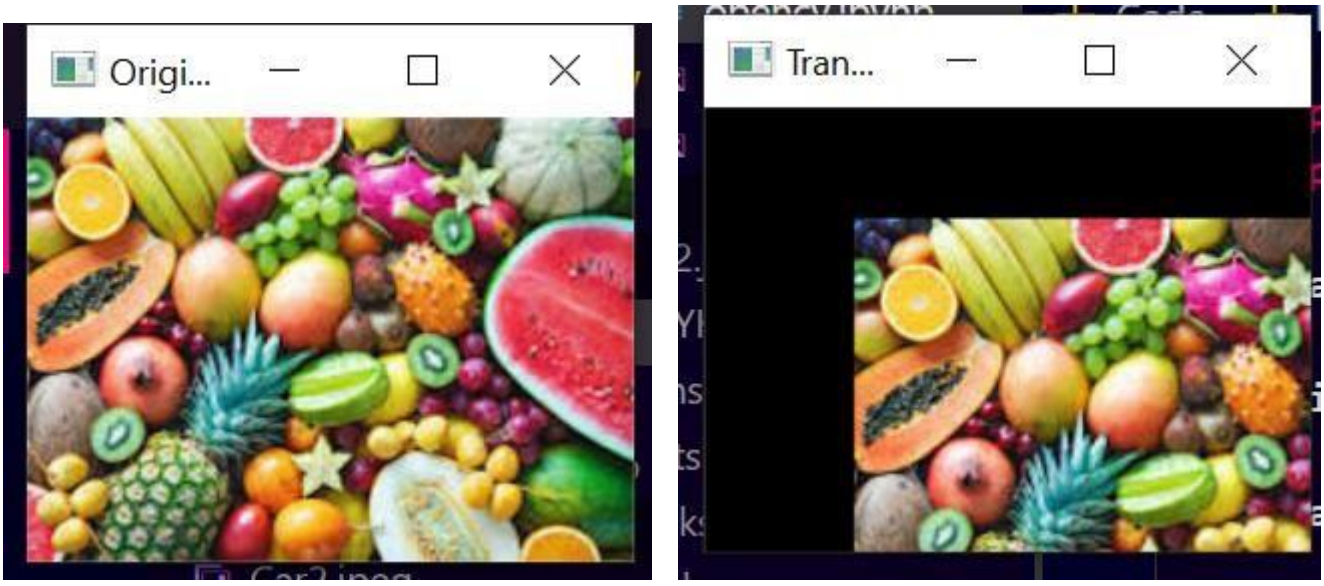
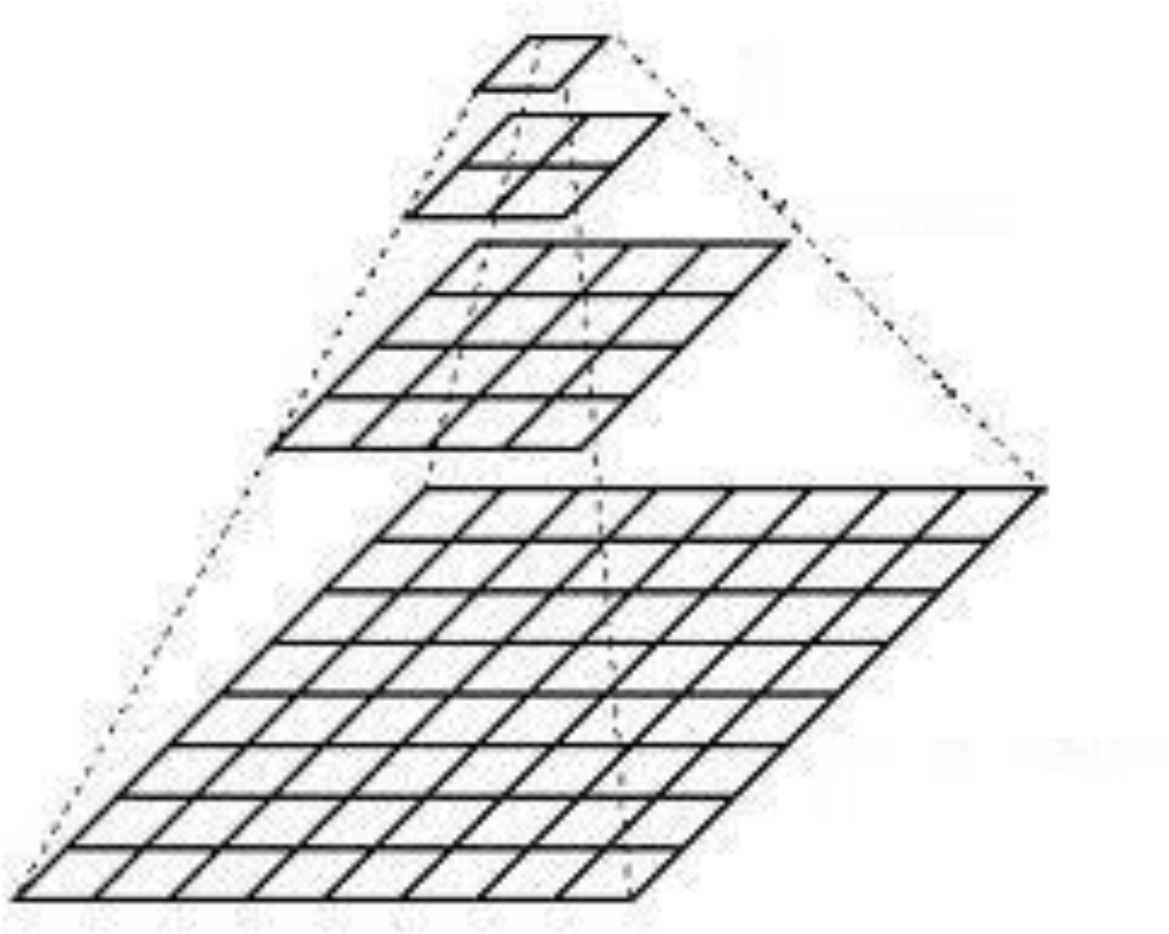


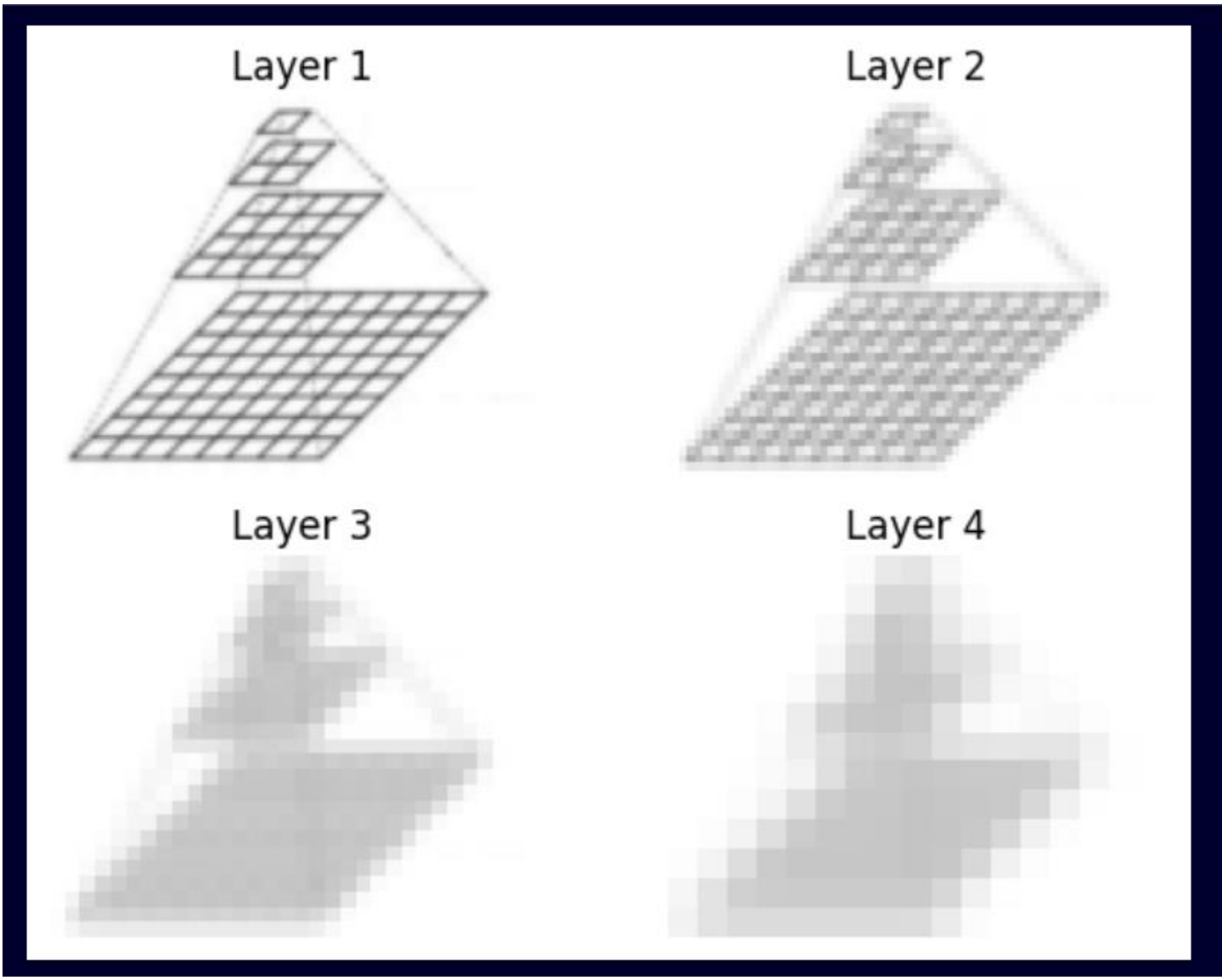
Image Pyramid Input:



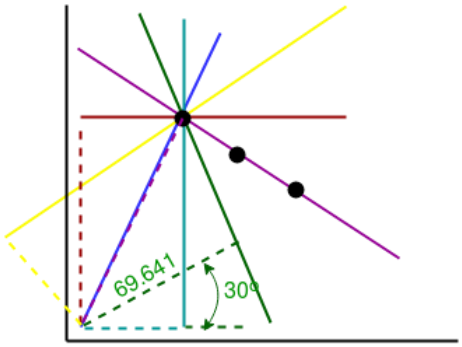
Code Explanation:

This code is like making a magic trick with a picture! First, it takes a picture from your computer and makes a copy of it. Then, it starts a loop to make the picture smaller step by step, like zooming out. Each time, it shrinks the picture a little using a special tool called "pyramid down." It shows the smaller version in two ways: one in a pop-up window and another inside a grid using Matplotlib. It also gives each small picture a name like "Layer 1," "Layer 2," and so on. The program waits for you to press a key after showing each small version before moving to the next. In the end, it closes all windows and properly shows the images in Matplotlib.

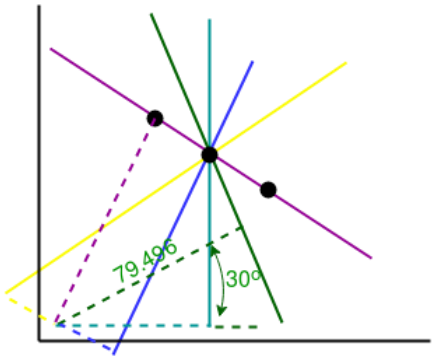
Output:



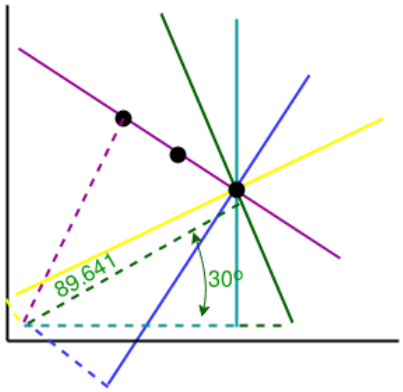
>> 2.3 Feature Detection and Description Input:



Angle	Dist.
0	40
30	69.6
60	81.2
120	40.6
150	0.4



Angle	Dist.
0	57.1
30	79.5
60	80.5
120	23.4
150	-19.5

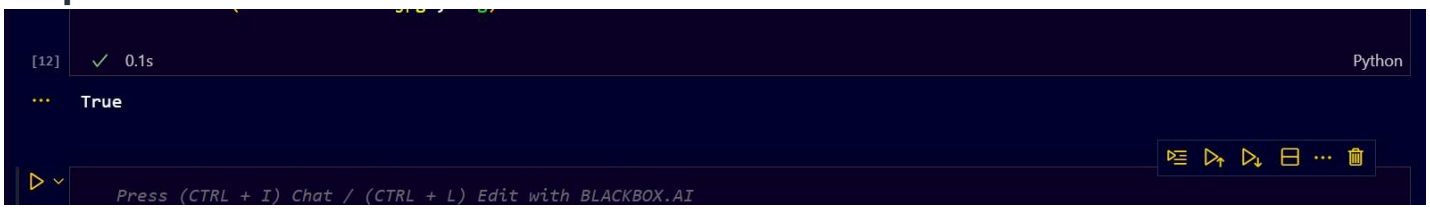


Angle	Dist.
0	74.6
30	89.6
60	80.6
120	6.0
150	-39.6

Code Explanation:

This program is all about finding straight lines in a picture using a cool trick called the Hough Transform. First, it loads the image and converts it into grayscale to make things easier to process. Then, it applies edge detection using the Canny method, which highlights the edges in the image. After that, the Hough Transform looks at these edges and figures out which ones form straight lines. It does this by finding values for r (distance from the center) and theta (angle) for each possible line. Once it identifies the lines, it calculates their start and end points based on some math with sine and cosine. Finally, it draws those lines in red on the original image and saves the result as a new image called "linesDetected.jpg".

Output:

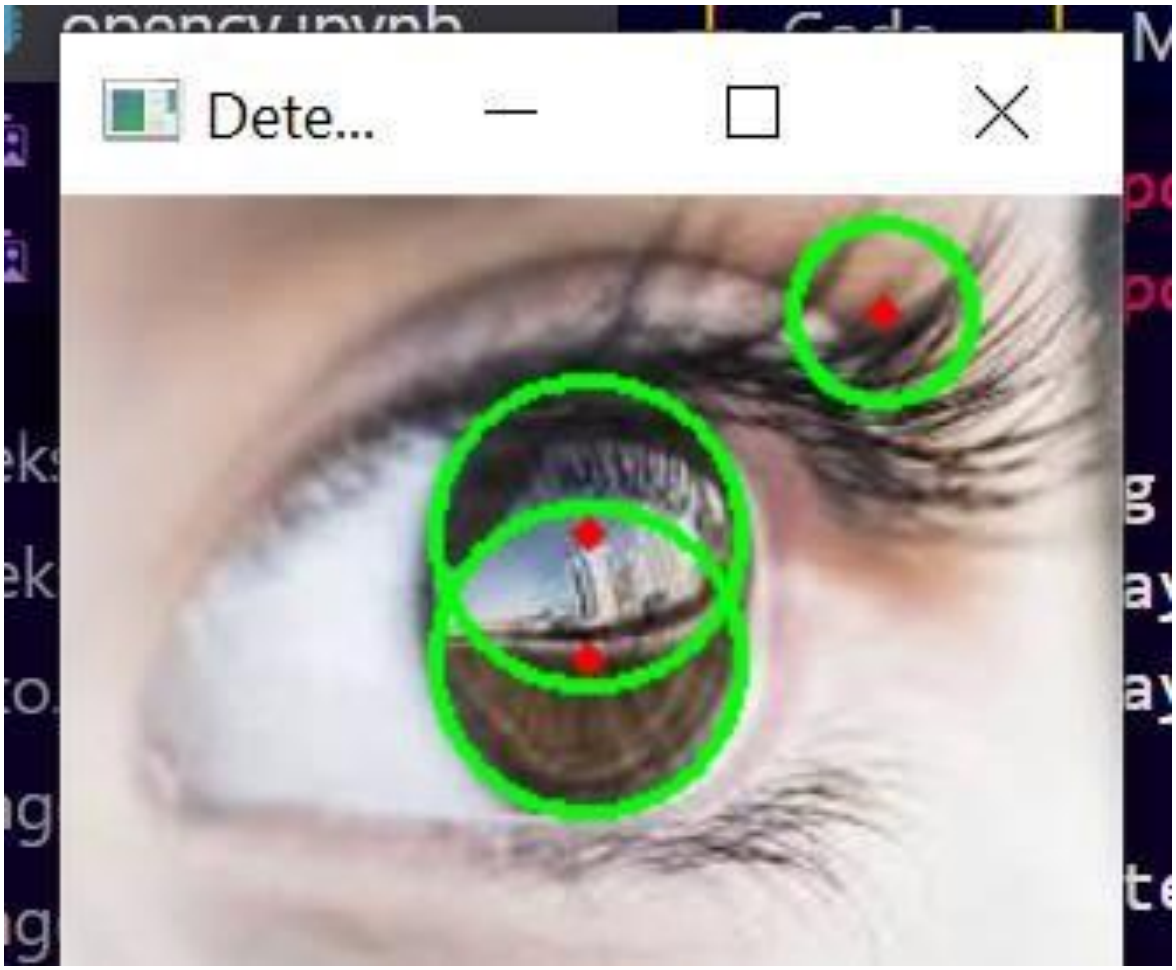


Circle Detection Input:

Code Explanation:

This program detects circles in an image using the Hough Circle Transform in OpenCV. First, it reads the image from the specified path and converts it to grayscale to simplify processing. Then, it applies a slight blur to reduce noise. The cv2.HoughCircles function is used to find circles in the image by detecting edges and circular patterns. If any circles are found, their coordinates and radius are extracted, and the program draws a green circle around each detected one, along with a small red dot at its center. Finally, the image with the detected circles is displayed, and the program waits for a key press before closing the window.

Output:

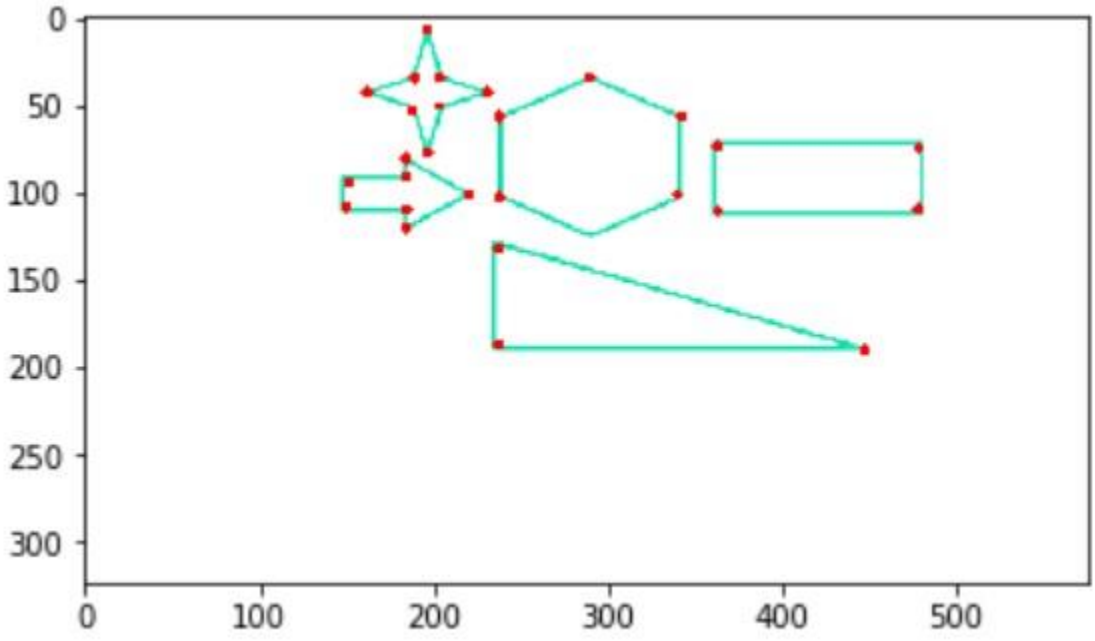


Detect corner of an image Input:

Code Explanation:

This program detects corners in an image using OpenCV. First, it loads the image and converts it to grayscale since detecting corners works better on a single-color channel. Then, it uses the cv2.goodFeaturesToTrack function to find the most important corners in the image. The function takes parameters like the number of corners to detect, the minimum quality of corners, and the minimum distance between them. Once the corners are found, the program loops through each one and draws a small white circle at its location. Finally, the modified image is displayed using Matplotlib, ensuring the colors appear correctly by converting it to RGB format before showing it.

Output:



Corner Detection with Shi-Tomasi method Input:

Code Explanation:

This program detects corners in an image using the Shi-Tomasi method in OpenCV. First, it loads the image from the specified path and checks if the file exists to prevent errors. Then, it converts the image to grayscale, as corner detection works better in a single-color channel. The cv2.goodFeaturesToTrack function is used to find the top 100 strongest corners based on quality and minimum distance constraints. If corners are detected, they are converted to integer values for proper processing. The program then loops through each detected corner and marks it with a small red circle. Finally, the modified image is displayed using Matplotlib after converting it to RGB format to ensure correct color representation.

Output:

Corner detection with Harris Corner Detection
Input:

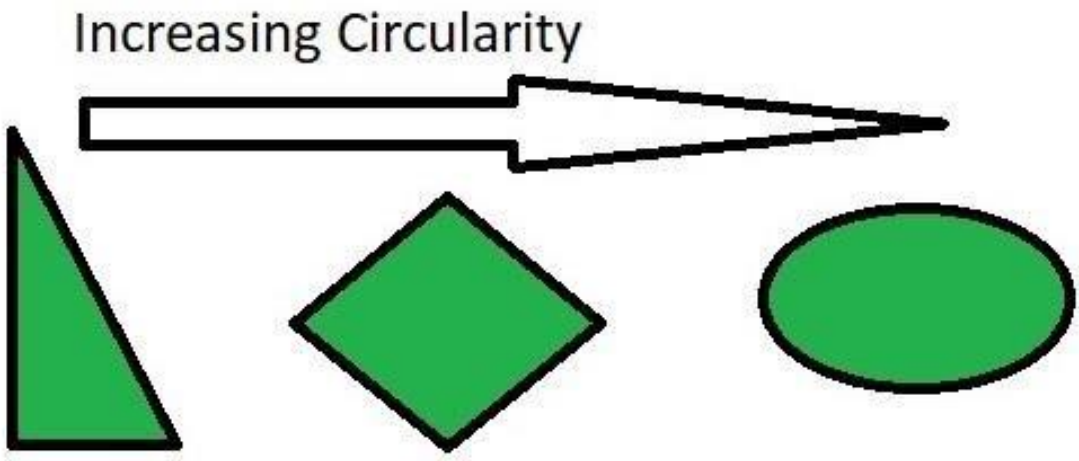
Code Explanation:

This program detects corners in an image using the Harris Corner Detection method in OpenCV. It starts by loading the image and verifying that the file exists to prevent errors. The image is then converted to grayscale since corner detection works best with a single-color channel. The grayscale image is converted to a 32-bit floating point format, which is required for accurate computation. The

cv2.cornerHarris function is applied to identify corner points, and the result is enhanced using dilation to make the detected corners more visible. A threshold is set to mark the strongest corners in red on the original image. Finally, the processed image is displayed using OpenCV, and memory is released when a key is pressed.

Output:

Find Circles and Ellipses in an Image Input:

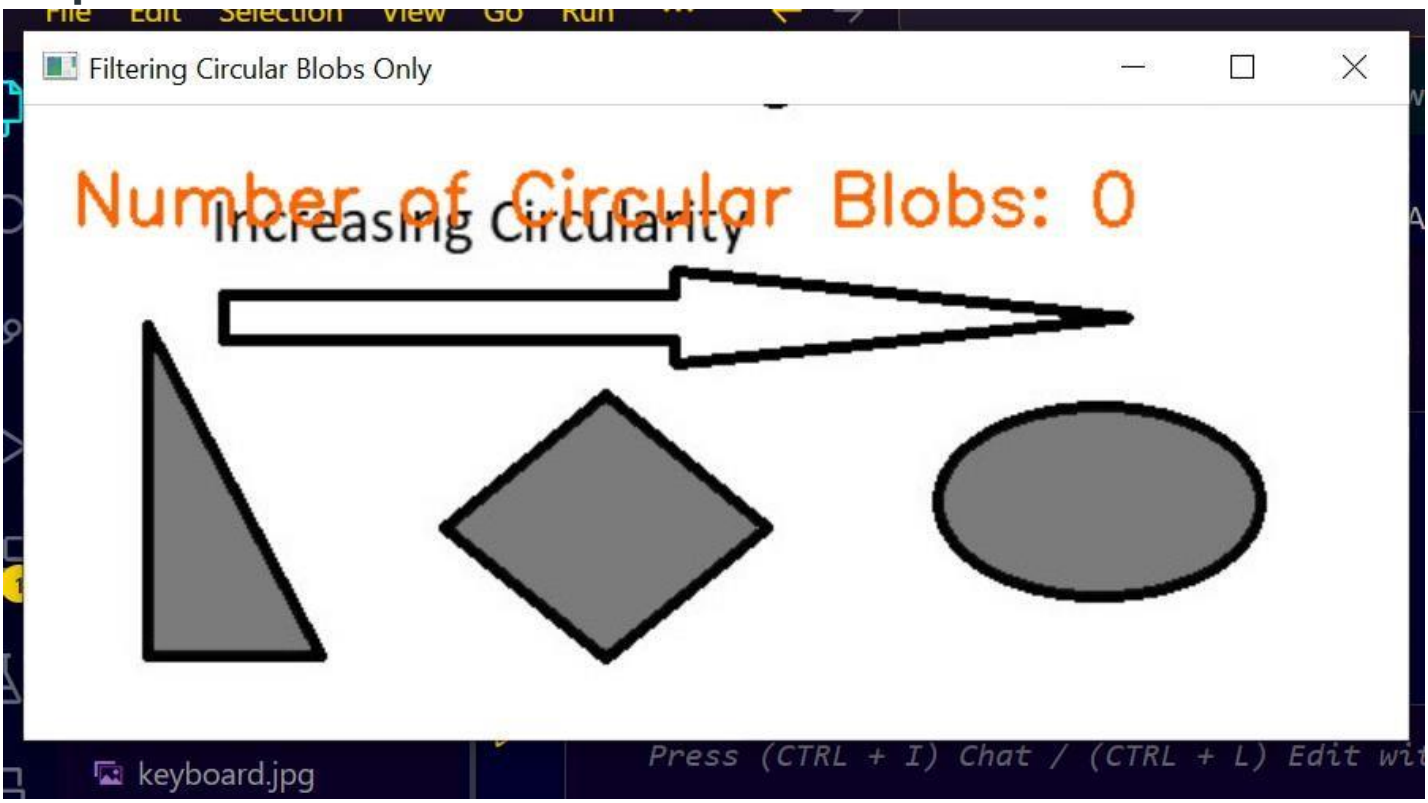


Code Explanation:

This program detects circular blobs in an image using OpenCV's SimpleBlobDetector. It first loads the image in grayscale mode and checks if the file exists. Then, it sets filtering parameters to detect blobs based on size, roundness (circularity), convexity, and shape consistency (inertia). A blob detector is created with these settings, and it scans the image to find keypoints representing the

detected blobs. The program then marks the detected blobs with red circles and displays the total count as text on the image. Finally, the processed image is shown in a window, which closes when a key is pressed.

Output:



>> 2.4 Drawing Functions

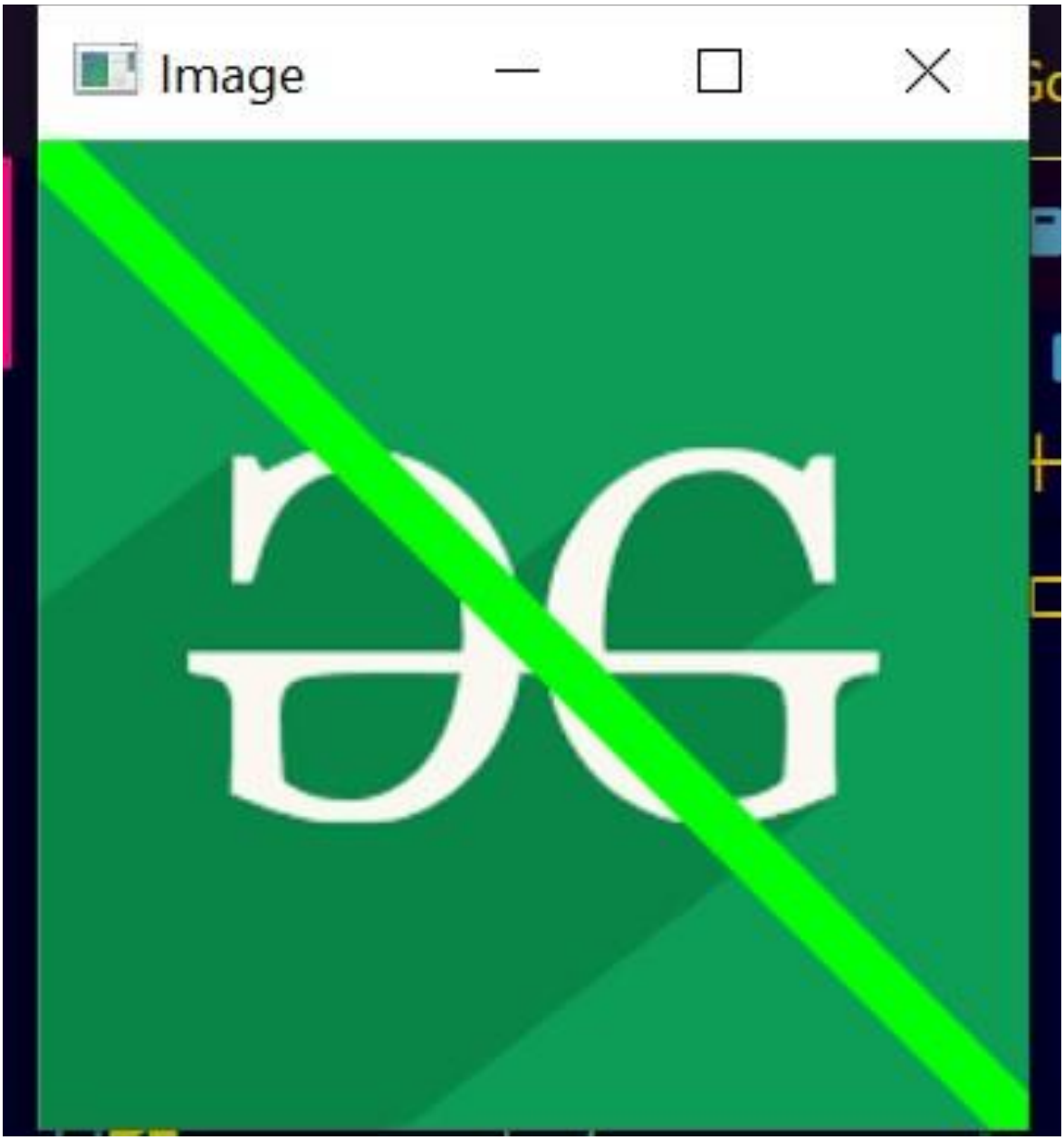
Draw a line:

Input:

Code Explanation:

This program draws a green diagonal line on an image using OpenCV. It first loads the image from the specified file path and checks if it exists to prevent errors. The program then defines the start and end points of the line, with (0, 0) representing the top-left corner and (250, 250) as the bottomright corner. The line color is set to green in BGR format (0, 255, 0), and its thickness is set to 9 pixels. The cv2.line() function is used to draw the line on the image. Finally, the modified image is displayed in a window until a key is pressed, after which the program releases any allocated resources and closes the window.

Output:

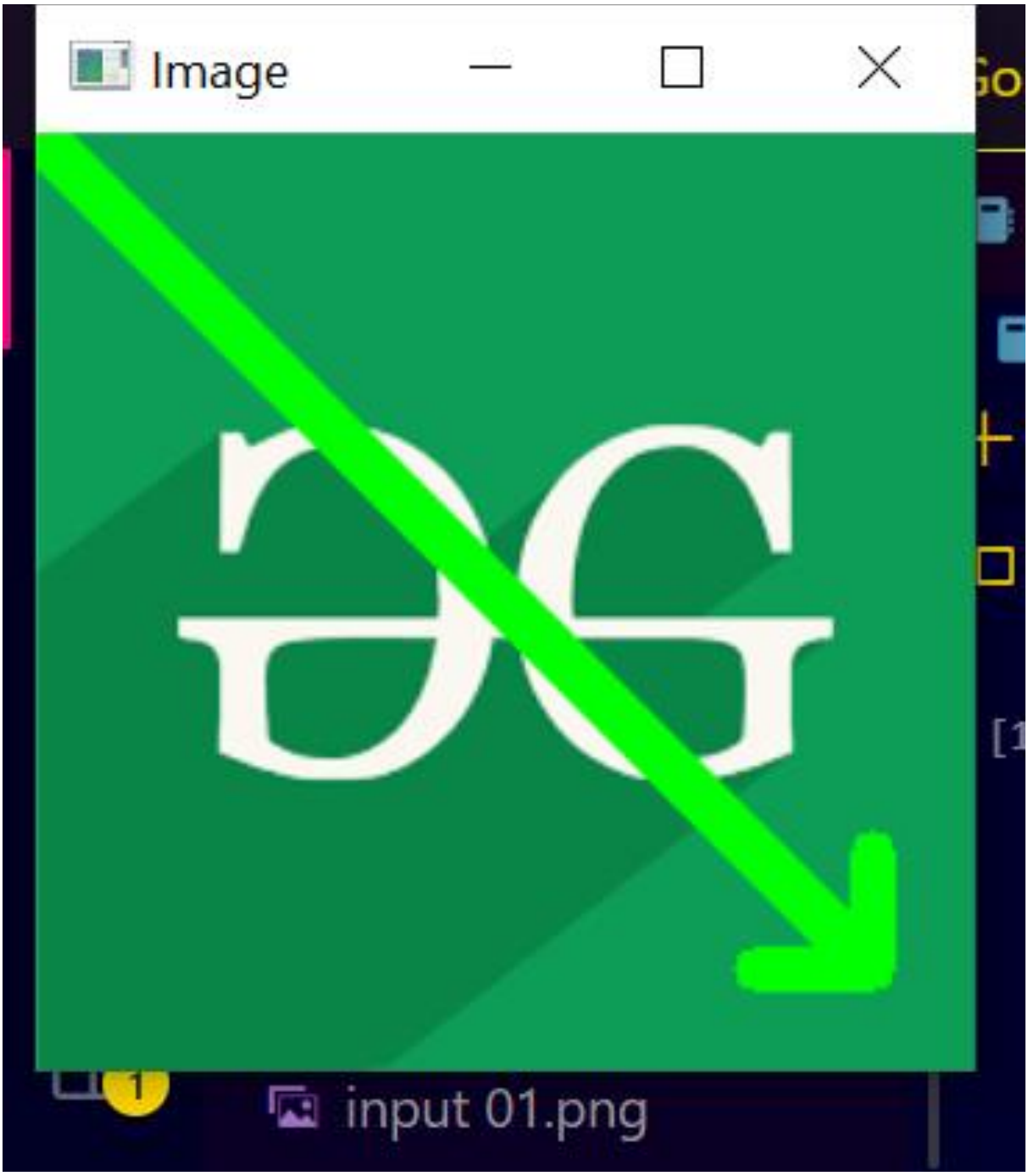


Draw arrow segment
Input:

Code Explanation:

This Python program uses the OpenCV library to draw a green arrowed line on an image. It first loads an image from the specified file path using cv2.imread(), then defines the start and end points of the arrow as (0, 0) (top-left corner) and (200, 200), respectively. The arrow is drawn using cv2.arrowedLine() with a green color (0, 255, 0) in BGR format and a thickness of 9 pixels. The modified image is then displayed in a window named "Image" using cv2.imshow(), and it remains visible until a key is pressed, after which all OpenCV windows are closed using cv2.destroyAllWindows().

Output:

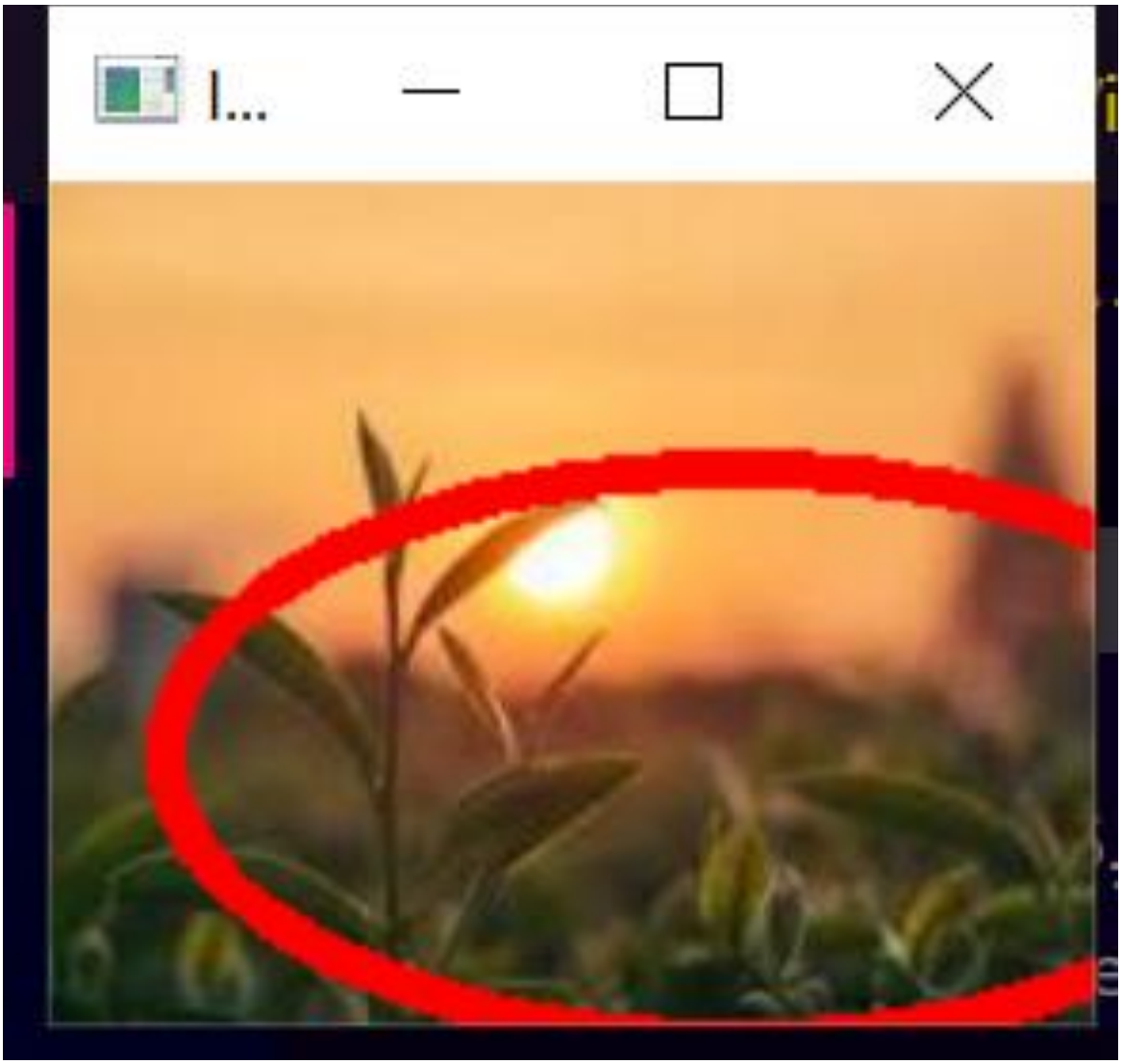


Draw an ellipse
Input:

Code Explanation:

This program uses OpenCV to draw a red ellipse on an image. First, it loads an image from the computer. Then, it sets the position of the ellipse’s center, its size, and how stretched it should be. The program also decides the color (red), thickness of the outline, and the angle of rotation. After that, it draws the ellipse on the image and displays it in a window. The program waits for a key press before closing the image window.

Output:

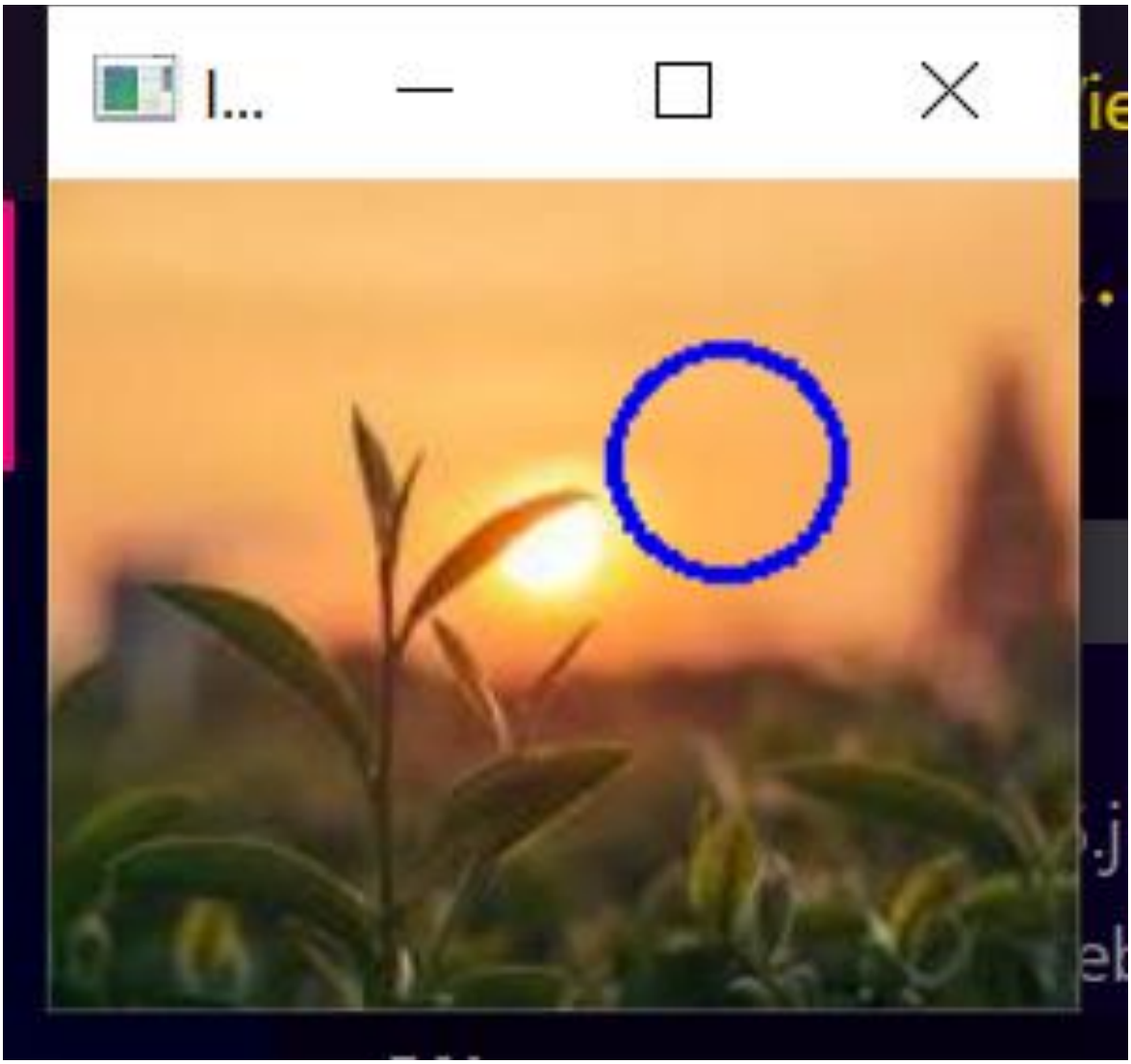


Draw a circle
Input:

Code Explanation:

This program uses OpenCV to draw a blue circle on an image. First, it loads an image from a given file path using cv2.imread(). Then, it defines the circle's center position, radius, color (blue in BGR format), and thickness of the border. The cv2.circle() function is used to draw the circle on the image. Finally, the modified image is displayed in a window using cv2.imshow(), and the program waits for a key press before closing the window.

Output:

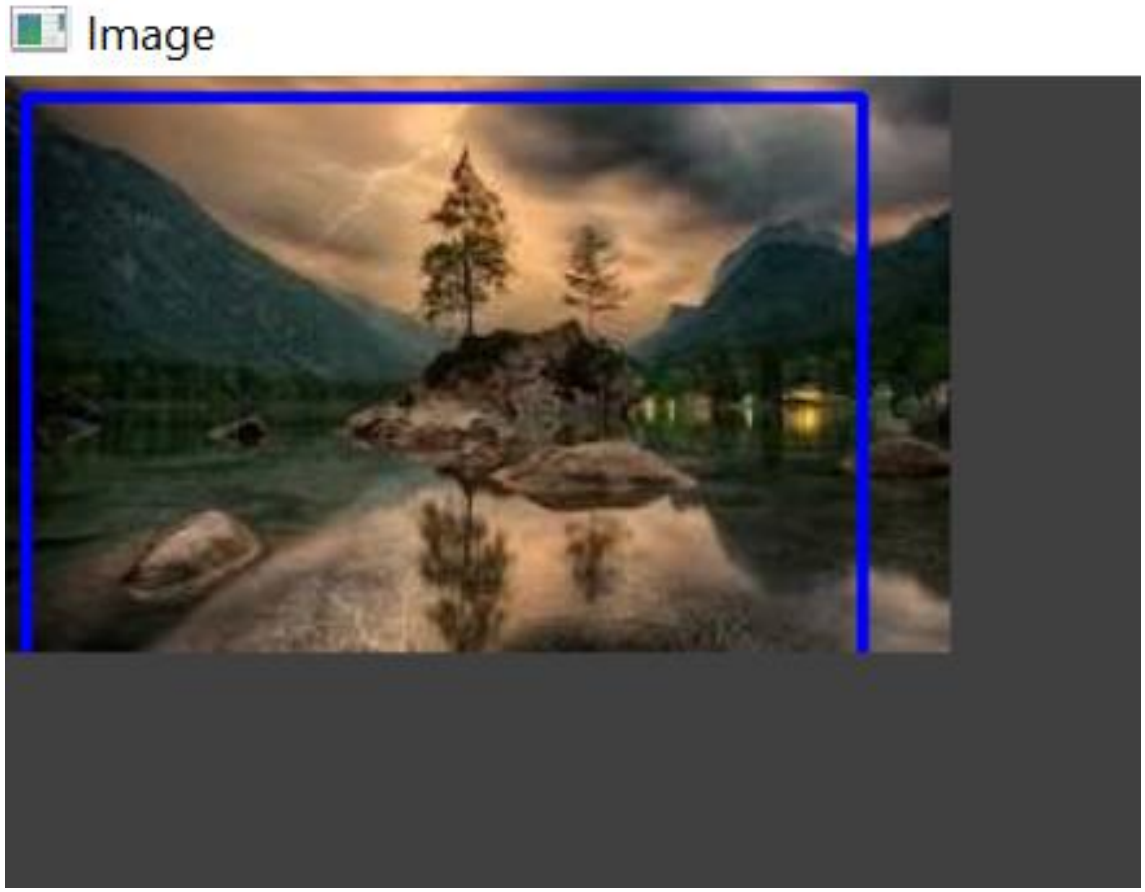


Draw a rectangle
Input:

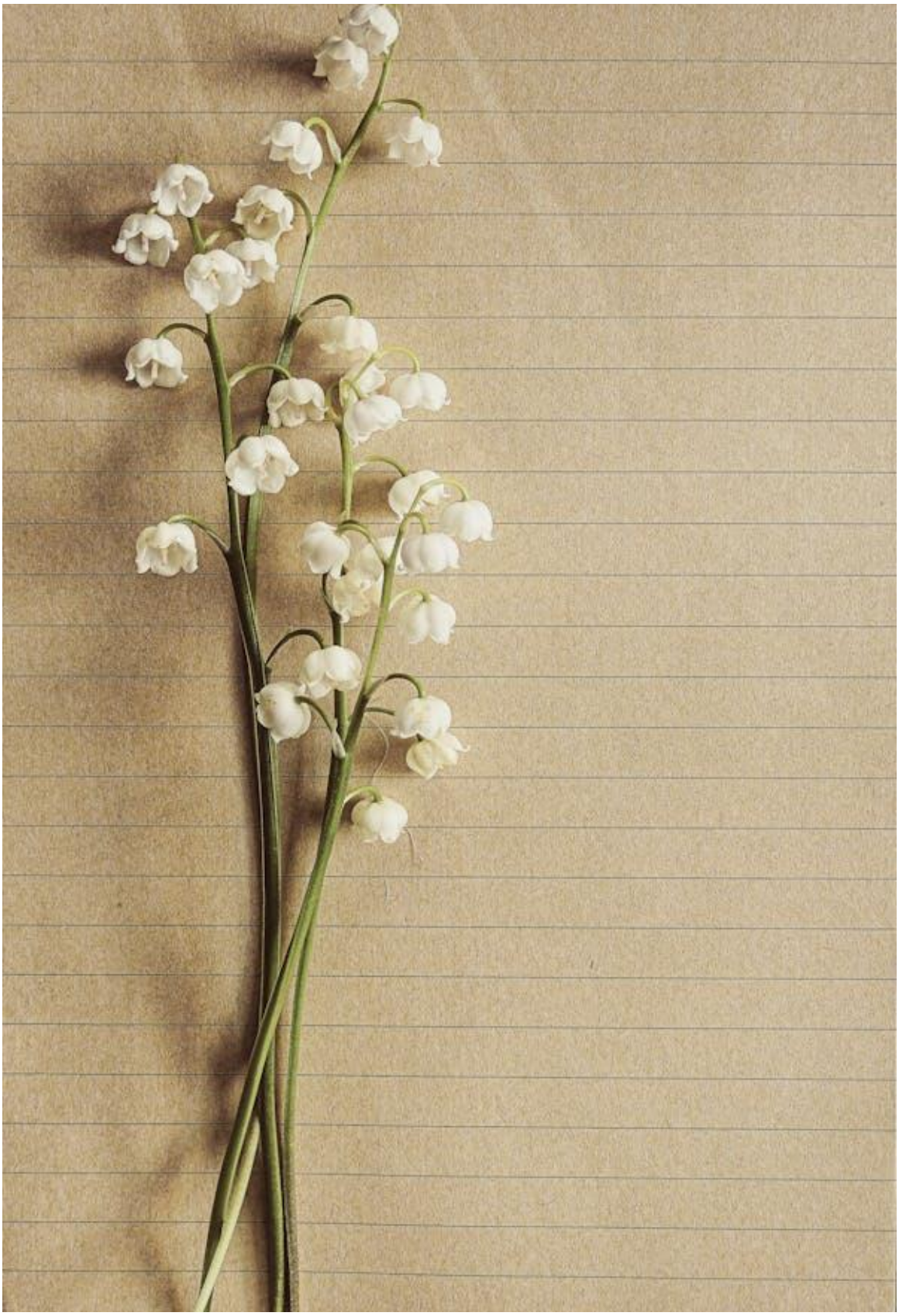
Code Explanation:

This program uses OpenCV to draw a blue rectangle on an image. It first loads an image from the specified file path using cv2.imread(). Then, it defines the starting and ending points of the rectangle, which represent the top-left and bottom-right corners. The color of the rectangle is set to blue using the BGR format, and the thickness of the border is set to 2 pixels. The cv2.rectangle() function is used to draw the rectangle on the image. Finally, the image is displayed in a window using cv2.imshow(), and the program waits for a key press before closing the window.

Output:



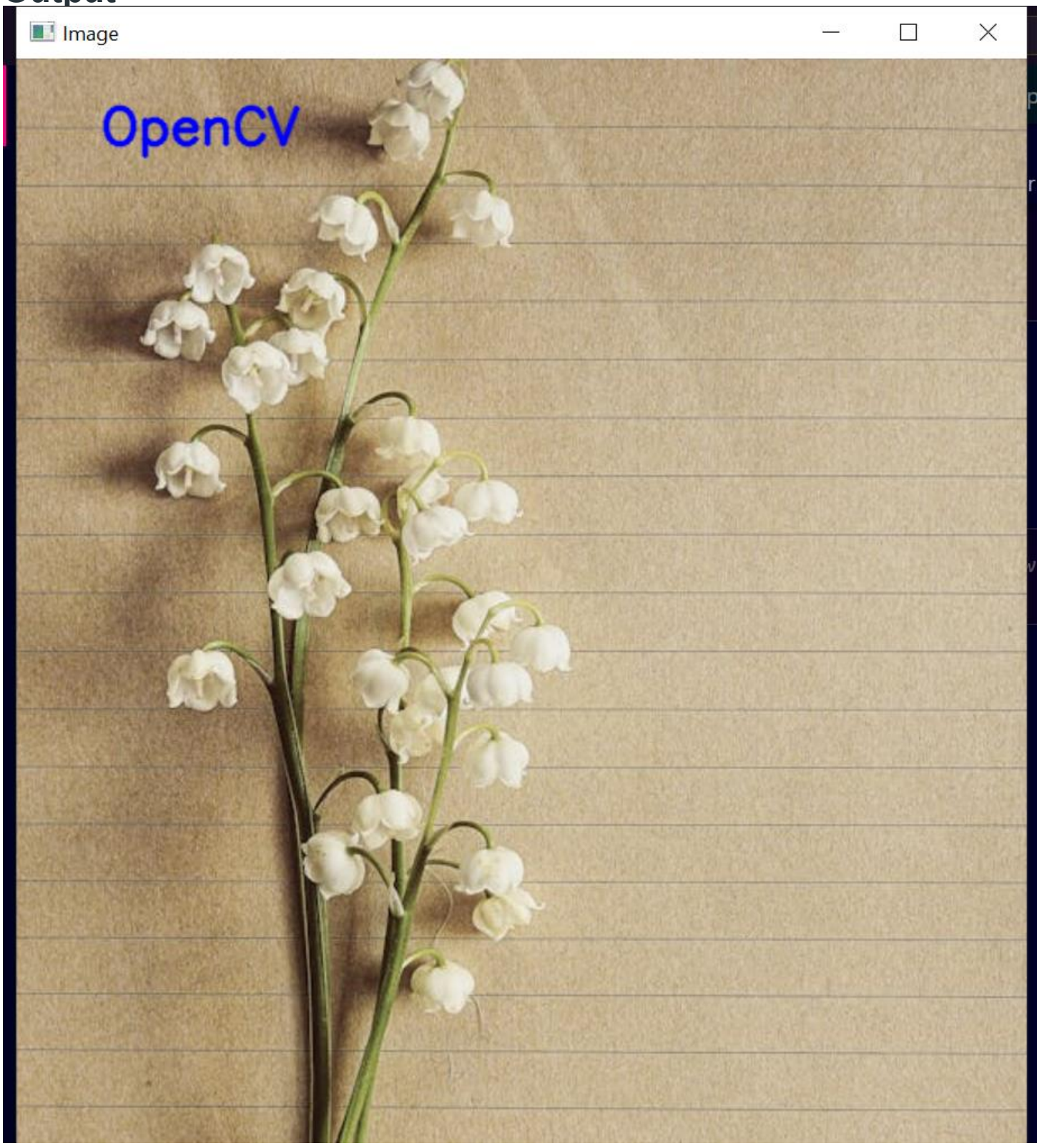
Draw a text string
Input:



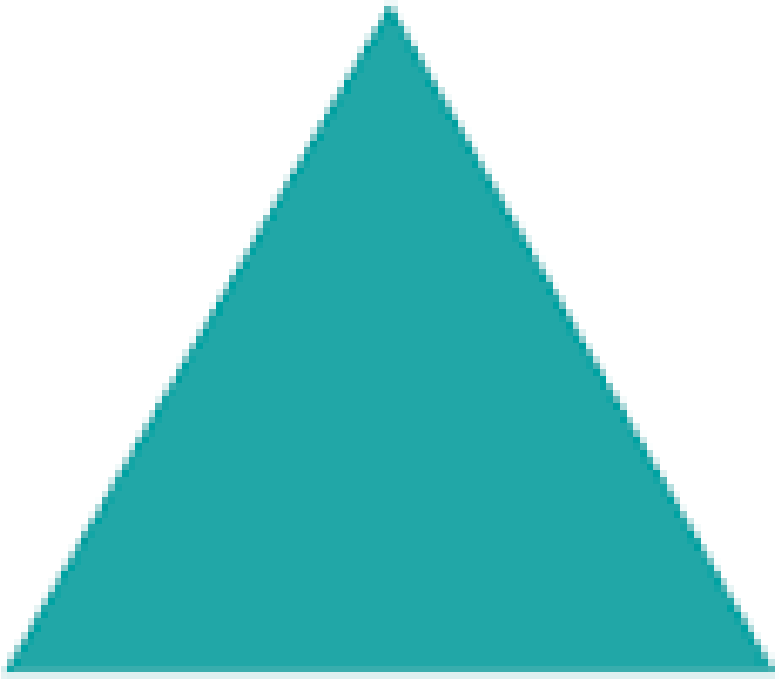
Code Explanation:

This program uses OpenCV to add text to an image. It first loads an image from the specified file path using cv2.imread(). The font type is set to cv2.FONT_HERSHEY_SIMPLEX, and the position where the text will be placed is defined as (50, 50). The text size, color (blue in BGR format), and thickness are also specified. The cv2.putText() function is then used to write the word "OpenCV" on the image with anti-aliased text for smooth edges. Finally, the modified image is displayed in a window using cv2.imshow(), and the program waits for a key press before closing the window.

Output



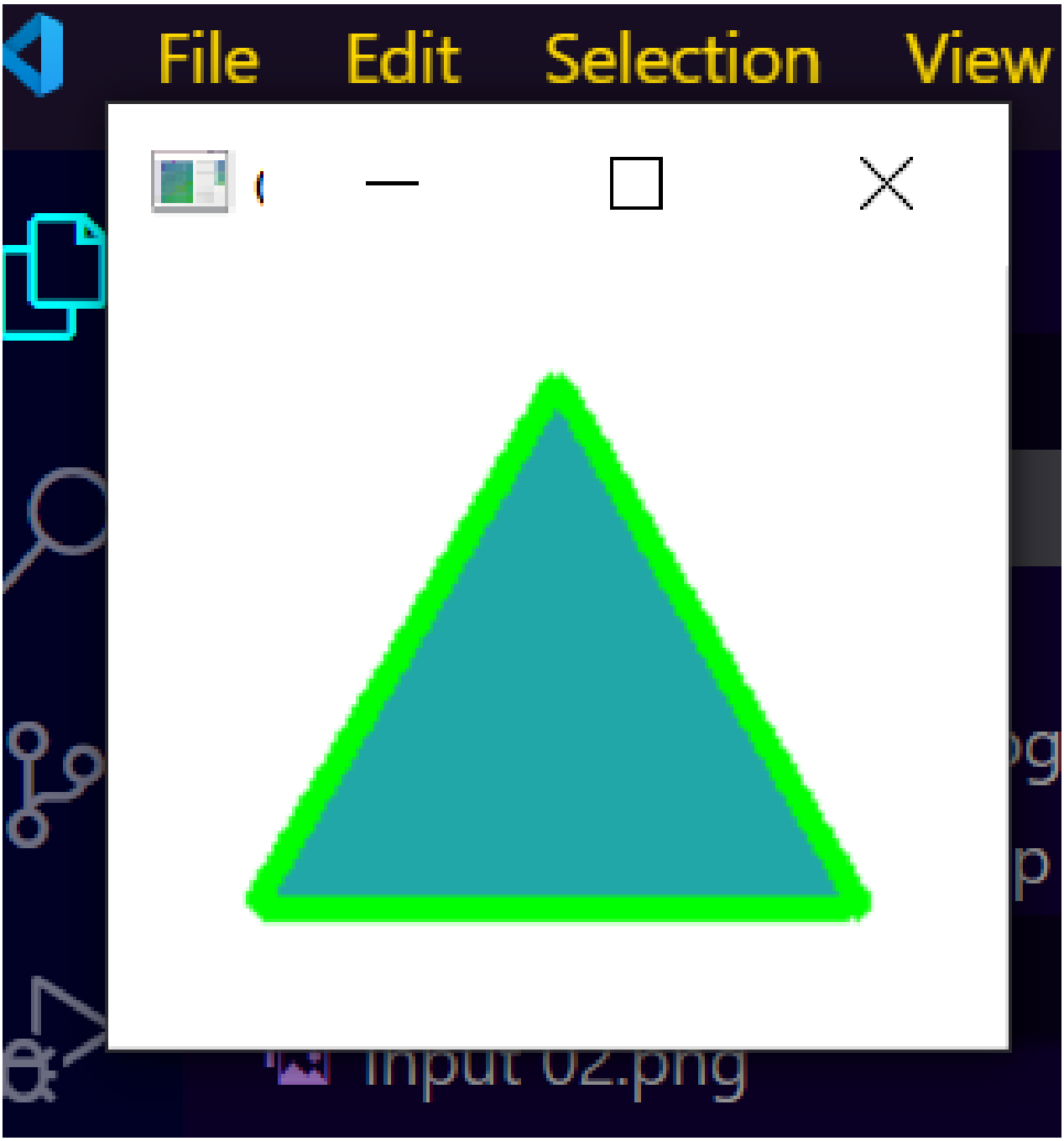
Find and Draw Contours
Input:

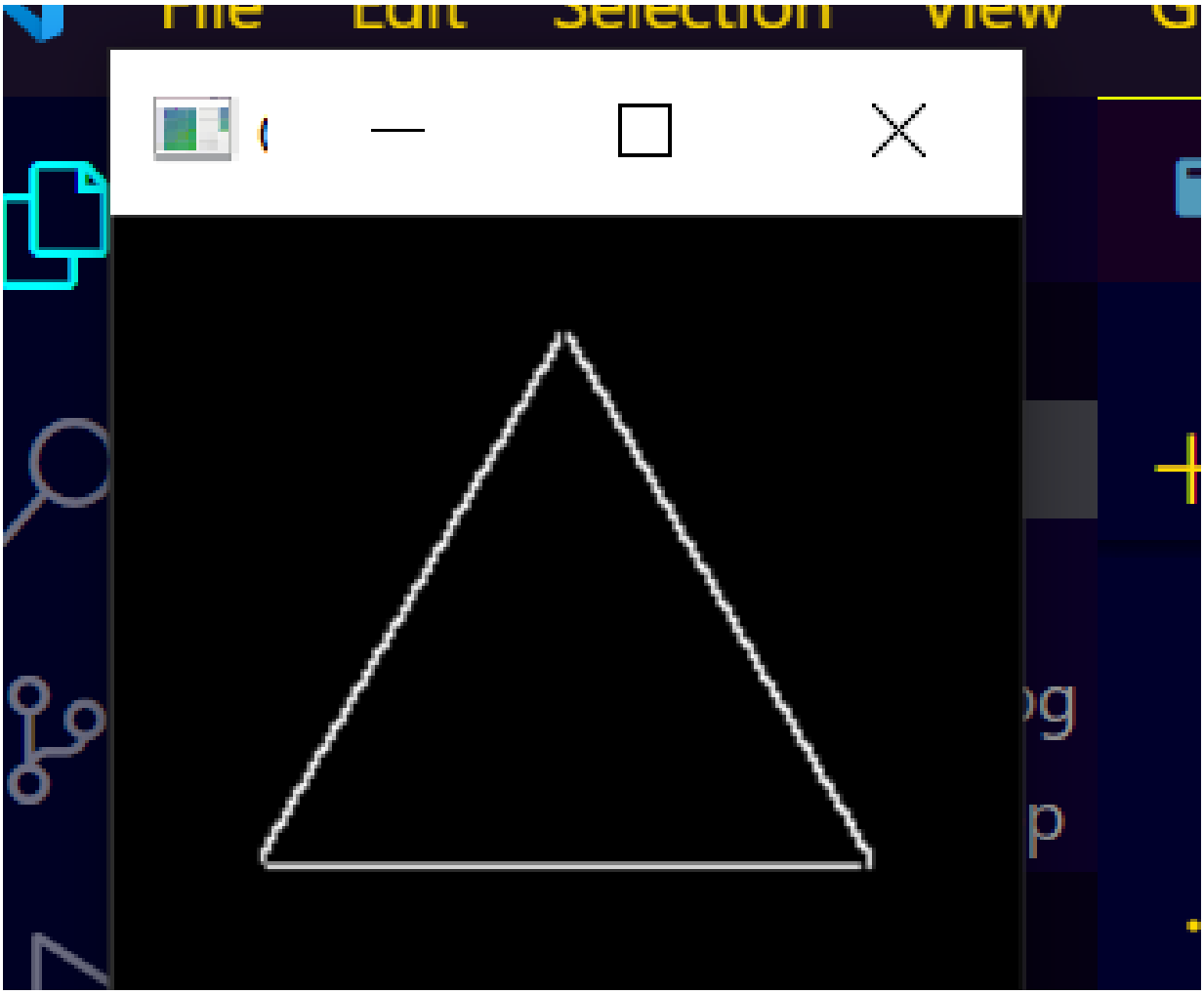


Code Explanation:

This program uses OpenCV to detect and draw contours in an image. It first loads an image and converts it to grayscale using cv2.cvtColor(). Then, it detects edges in the image with the Canny edge detection method. The cv2.findContours() function is used to find the contours of objects in the image, and the total number of contours found is printed. Finally, cv2.drawContours() is used to outline the detected contours in green, and the processed images are displayed using cv2.imshow(). The program waits for a key press before closing the image windows.

Output:


File
Edit
Selection
View
Input 02.png



Draw a triangle with centroid

Code Explanation:

This program uses OpenCV to draw a triangle and find its centroid. It first creates a black window of size 400x300 pixels using NumPy. Three points representing the triangle's vertices are defined, and the triangle is drawn by connecting these points with blue lines using cv2.line(). The centroid of the triangle is then calculated by averaging the x and y coordinates of the three points. A small green circle is drawn at the centroid using cv2.circle(). Finally, the image is displayed using cv2.imshow(), and the program waits for a key press before closing the window.

Output:

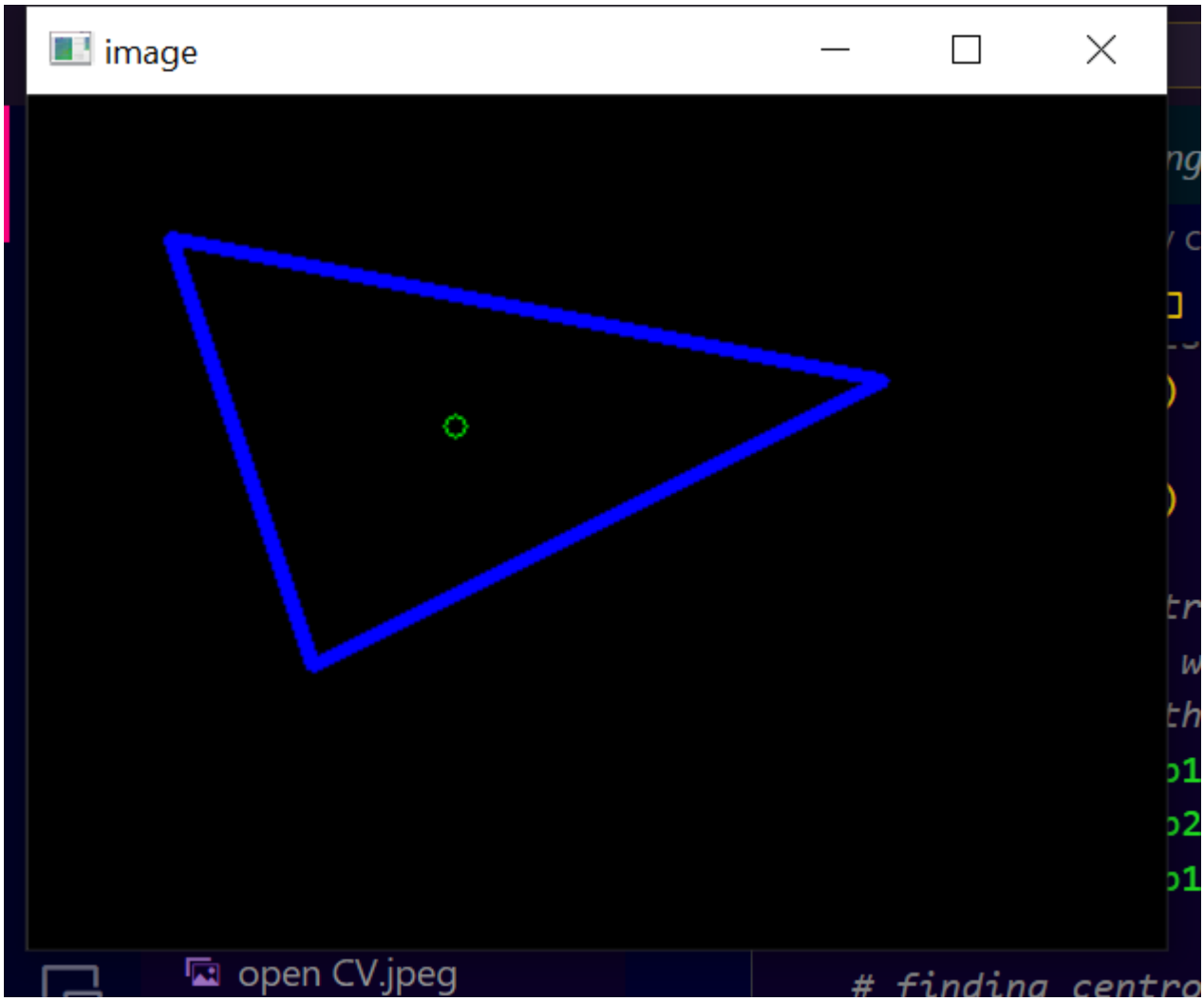


image
open CV.jpeg