**Lab 05 Task**

****

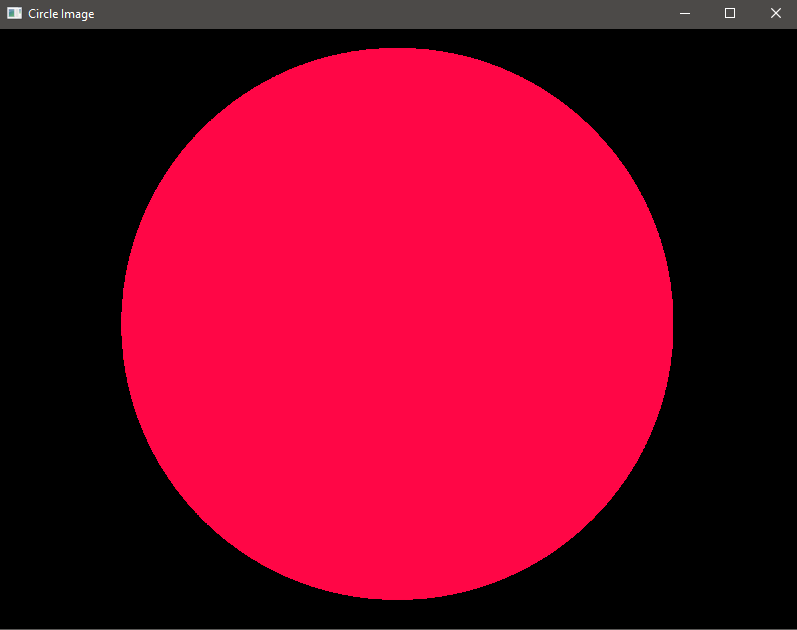
**Name Hammad Arshad**

**Roll no SU92-BSAIM-F23-030**

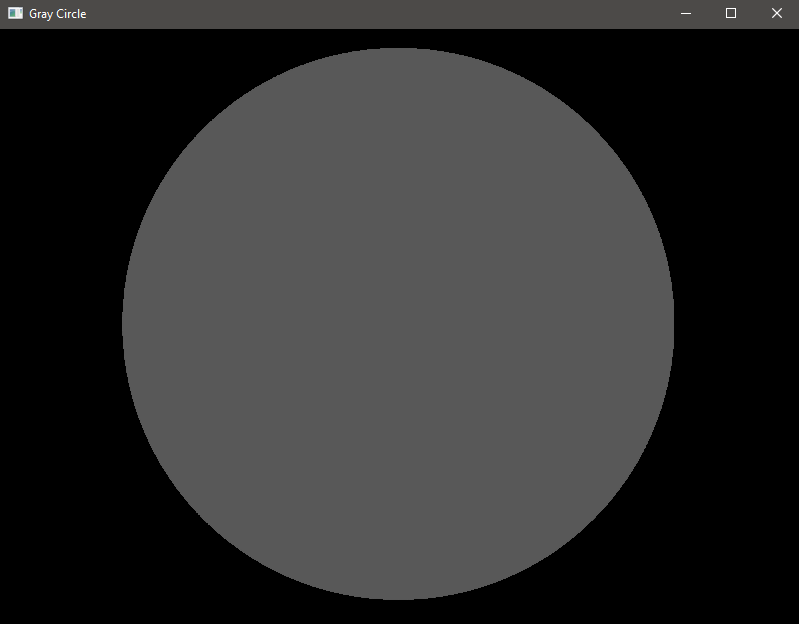
**Section BSAI-4A**

**Subject PAI (LAB)**

**Reading and Displaying an Image**



**Convert to Grayscale**



**How it works:**

The code loads an image from a file and stores it as a variable.

The image is displayed using OpenCV’s built-in function.

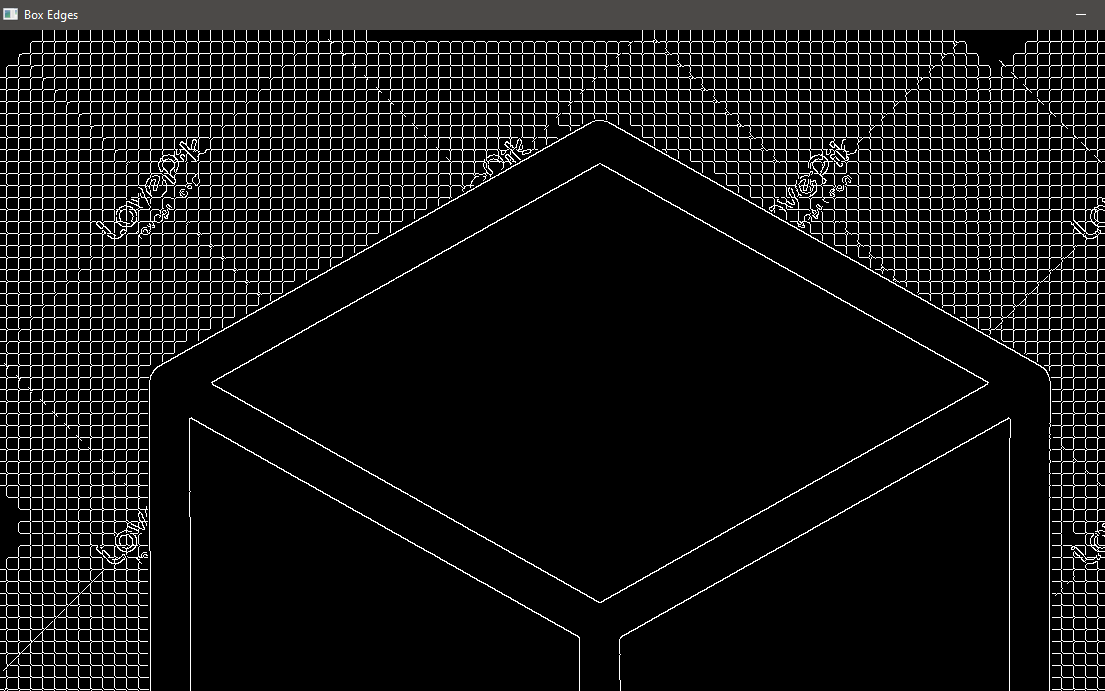
Optionally, we can convert the image from BGR (default OpenCV format) to RGB so that colors appear correctly when using matplotlib.

**Why we do this:**

To work with an image, we must first load it into memory.

Converting colors to RGB is important when using libraries like Matplotlib, which expect images in RGB format.

**Edge Detection**



**How it works?**

The images are converted to grayscale first.

cv2.Canny() is used to detect edges using two threshold values (50 and 150).

The edges are then displayed using cv2.imshow().

**Why?**

Edges highlight the boundaries of objects, making it easier to detect shapes like circles and rectangles.

The Canny Edge Detection algorithm is widely used in computer vision because it removes noise and detects clear edges.

**Saving an Image**

**How it works:**

After making modifications, the processed image is saved using OpenCV’s save function.

The quality of the image can be controlled using parameters.

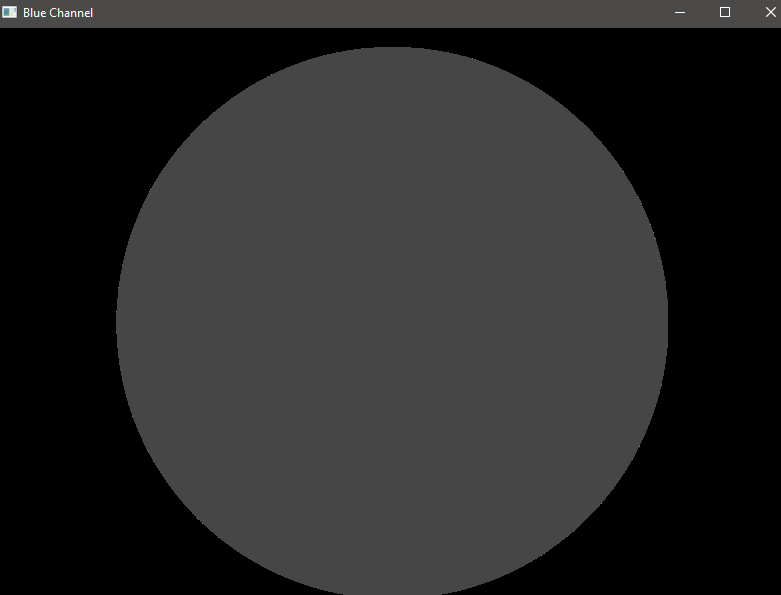
Why we do this:

We need to store the processed images for later use.

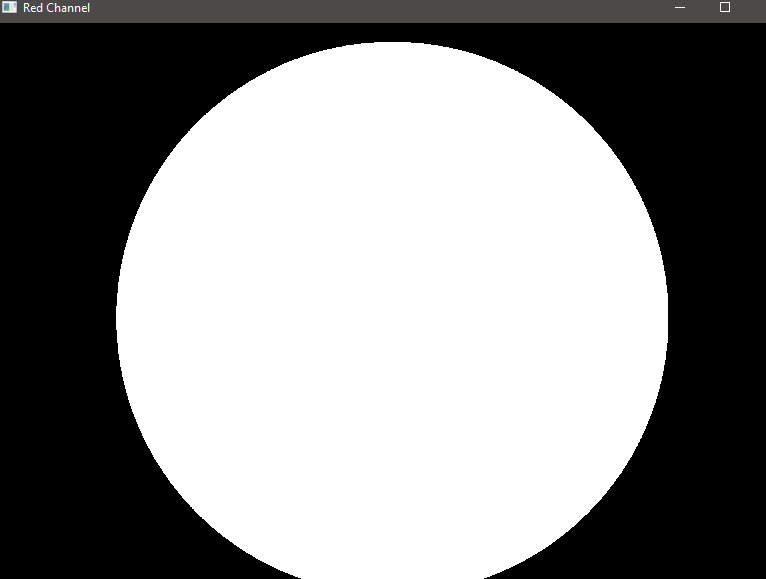
Adjusting quality helps save storage space while keeping acceptable visual details.

**Working with Colors**

**Green channel**  


**Blue**

**Red**



**How it works:**

The image is split into Red, Green, and Blue (RGB) channels.

Each channel is extracted separately to understand how the image is formed.

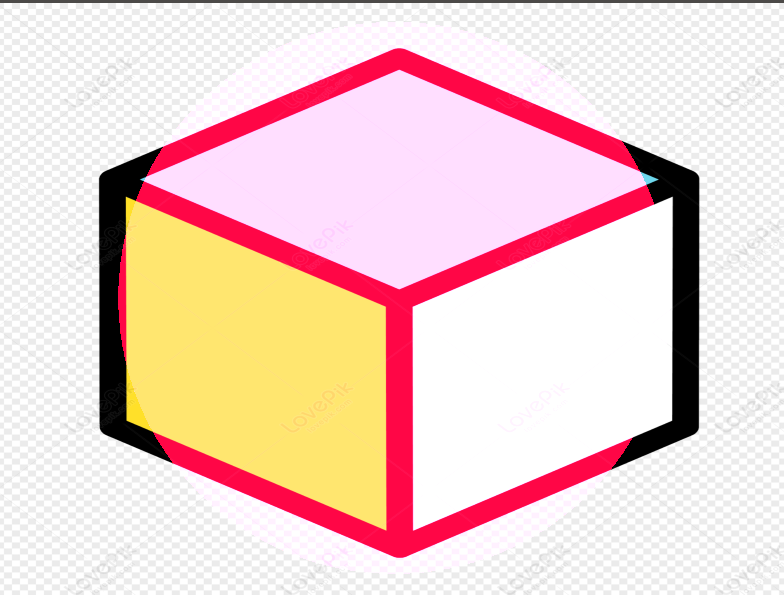
**Why we do this:**

Splitting color channels helps in image filtering, enhancement, and object detection.

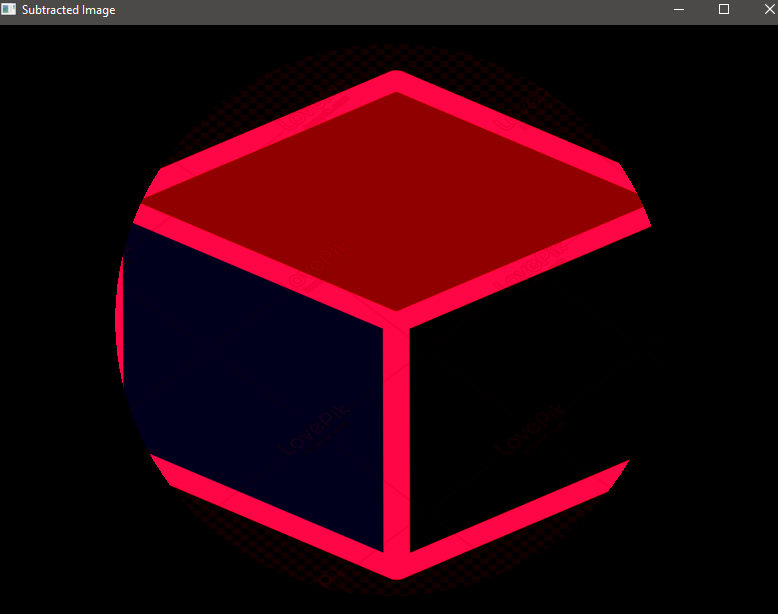
Many computer vision tasks process images in grayscale or individual channels rather than full color.

**Image Arithmetic (Math with Images)**

**Addition**



**Subtraction**



**How it works:**

The pixel values of two images are added together to brighten the image.

Subtraction is used to find differences between two images.

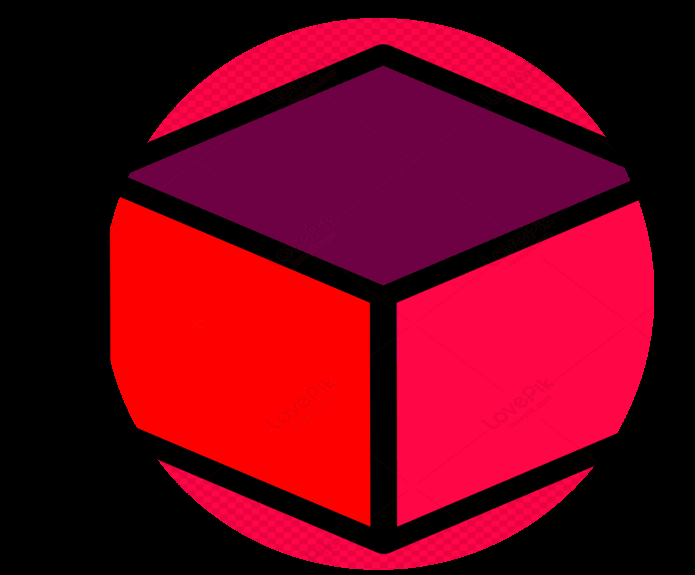
**Why we do this:**

Addition can help in image blending, where we mix two images smoothly.

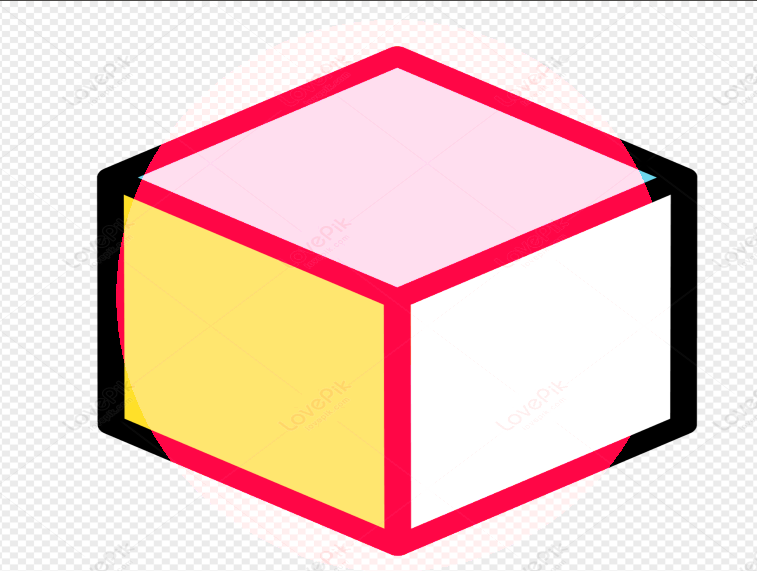
Subtraction is useful in detecting changes in an image, such as movement or differences between two frames.

**Bitwise Operations (AND, OR, XOR, NOT)**

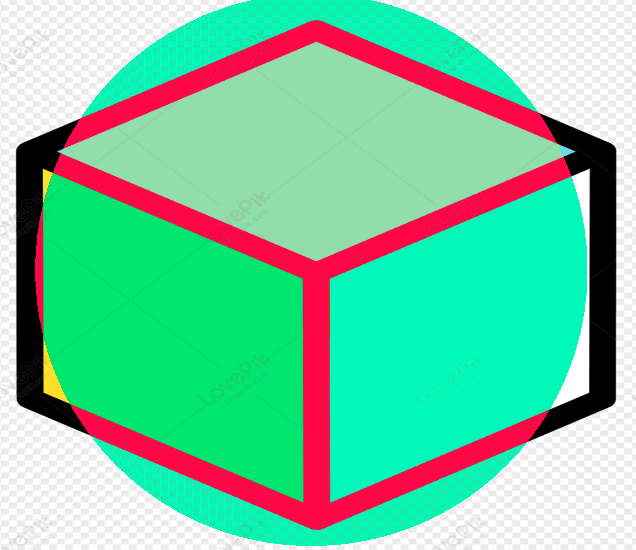
**AND:**

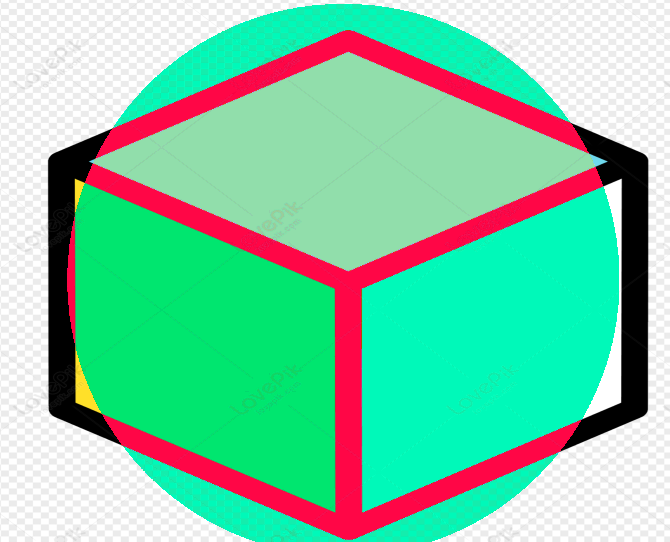


OR:



XOR



NOT:  


**How it works:**

These operations work pixel by pixel to compare two images.

AND keeps only the overlapping parts of two images.

OR combines both images.

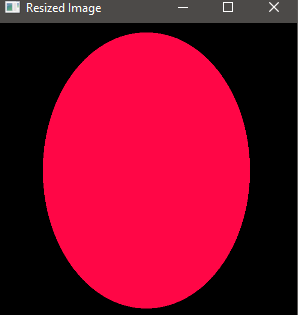
XOR removes common areas and keeps only the differences.

NOT inverts an image, turning black to white and vice versa.

**Why we do this:**

These operations are used in masking, object detection, and creating transparent images.

**Resizing an Image**



**How it works:**

The image is resized to a new width and height using OpenCV’s resize function.

A scaling factor can also be used instead of fixed dimensions.

**Why we do this:**

Resizing is necessary when working with large images to speed up processing.

It helps maintain consistency when processing multiple images.

**Blurring an Image**



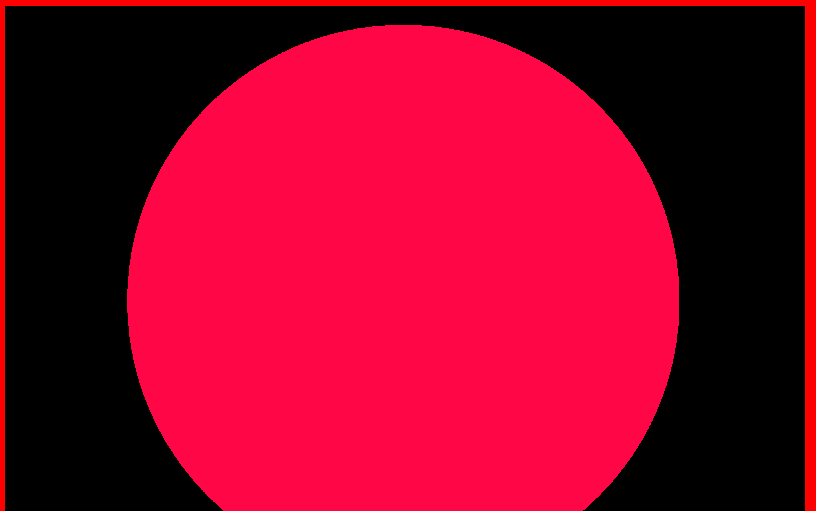
**How it works:**

A filter is applied over the image to smooth out details and reduce noise.

**Why we do this:**

Blurring is useful in face detection and background removal by reducing unnecessary details.

**Adding Borders**



**How it works:**

A border (frame) is added around the image using different styles (constant color, reflection, etc.).

**Why we do this:**

Borders are often used in photo editing applications or to highlight an image.

**Removing Noise (Denoising)**

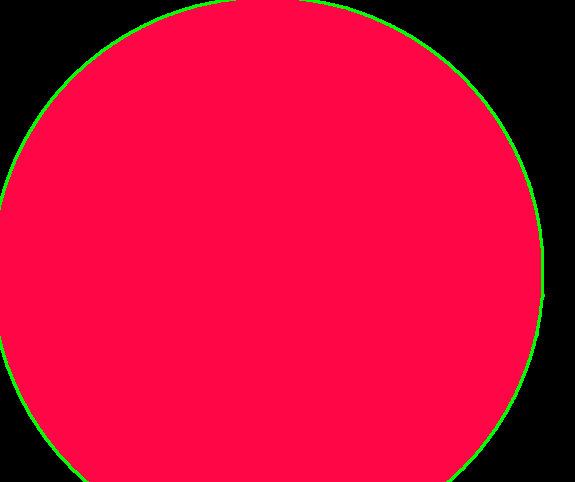
**How it works:**

Special filters are applied to remove unwanted spots or graininess from an image.

**Why we do this:**

Many images, especially scanned or low-light images, contain unwanted noise that affects clarity.

**Contour Detection (Finding Object Outlines)**



**How it works:**

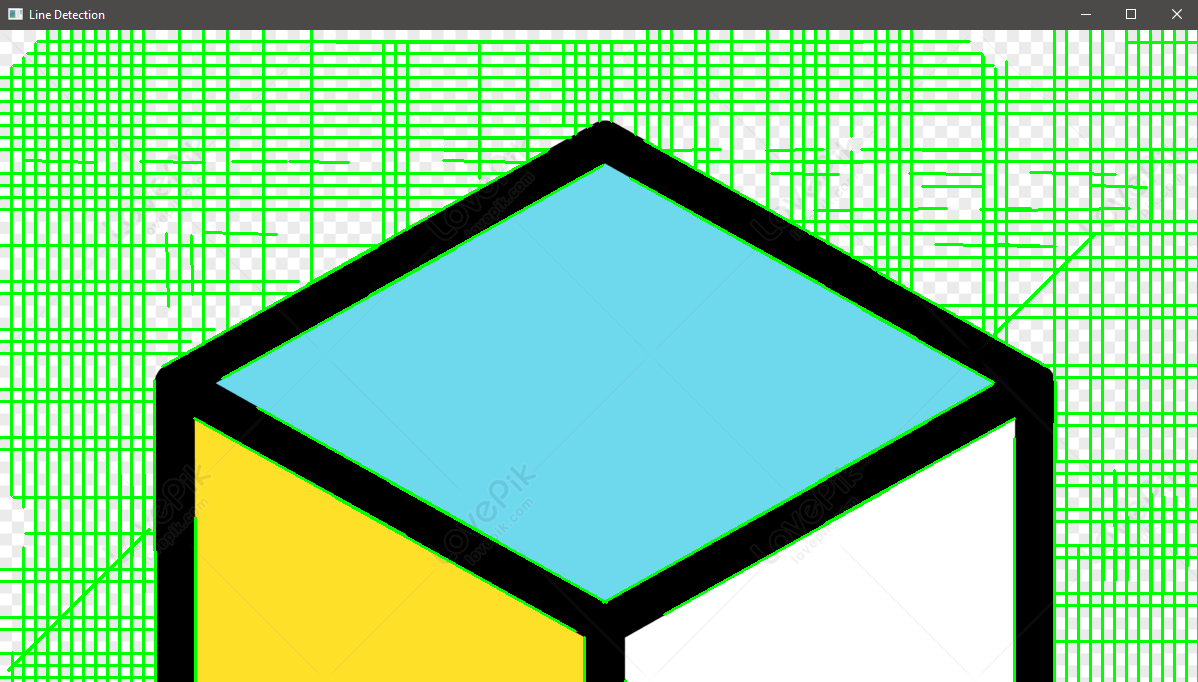
The image is converted to grayscale, and edges are detected.

The contours (outlines of objects) are found and drawn on the image.

**Why we do this:**

Contour detection is used in object recognition, handwriting analysis, and shape analysis.

**Line Detection**



**How it works:**

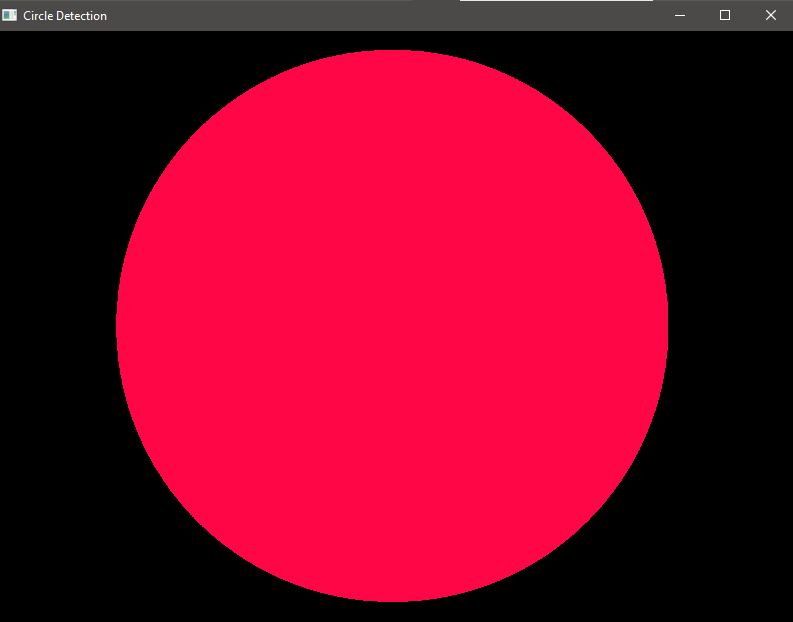
The image is processed to detect straight lines using an algorithm.

Detected lines are drawn on the image.

**Why we do this:**

Line detection is useful in road lane detection (self-driving cars) and barcode scanning.

**Circle Detection**



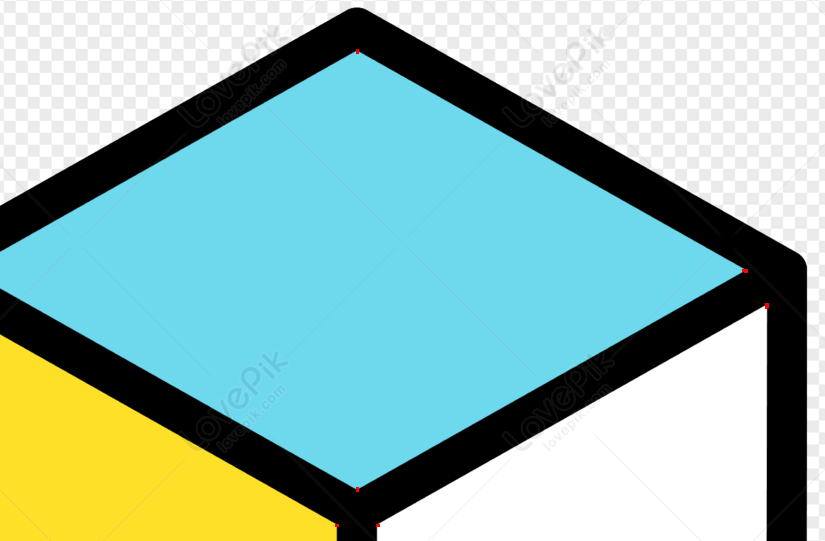
**How it works:**

The program looks for circular shapes in the image and marks them.

**Why we do this:**

This is useful in coin counting, ball detection in sports, and robotic vision.

**Corner Detection**



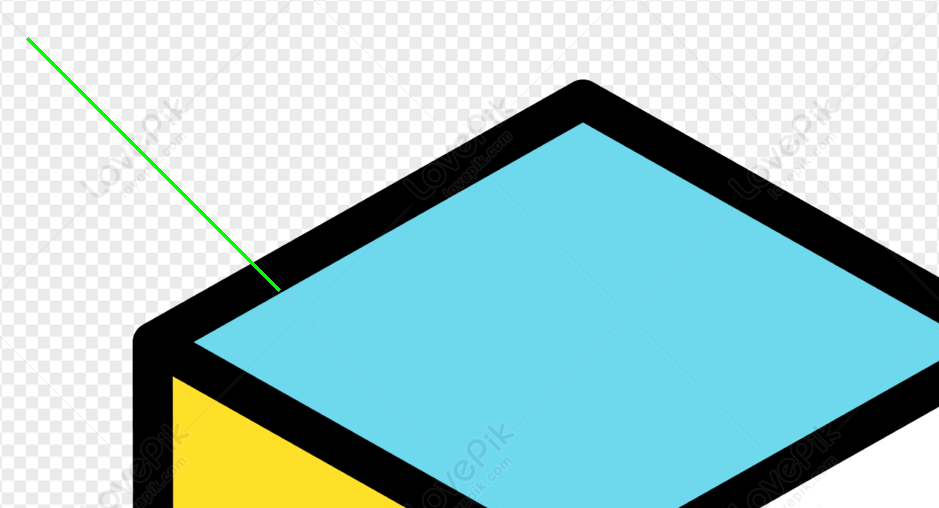
**How it works:**

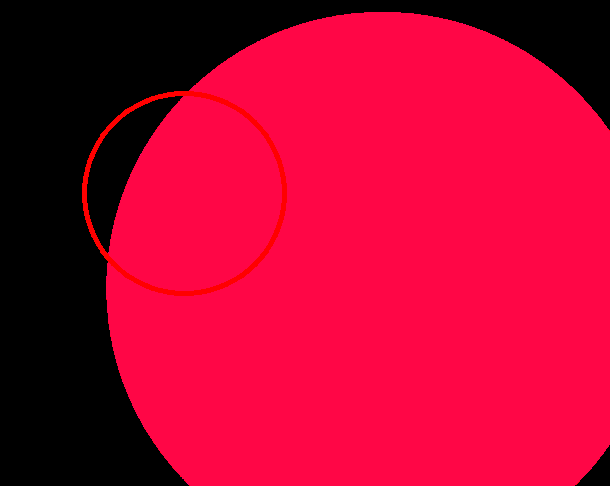
The program detects sharp corners in an image, which are points where two edges meet.

**Why we do this:**

Corner detection is used in motion tracking and image alignment.

**Drawing Lines and Circles**





**How it works:**

The program allows us to draw lines, circles, or other shapes on an image.

**Why we do this:**

Useful for highlighting objects, creating annotations, or marking detected objects in an image.

**Working with Videos**

**How it works:**

The program opens a video file and plays it frame by frame.

**Why we do this:**

Video processing is used in CCTV surveillance, motion detection, and real-time monitoring.