Pixels
EVERY PIXEL IS KNOWLEDGE

فيما أفتاه

Intro To Computer Vision

Represented by : Basma Ahmed .

The content for today:

1. DL vs OpenCV – What's the Difference?
2. Introduction to OpenCV.
3. Basic Operations with OpenCV.
4. Color Spaces.
5. Filters in Image Processing.
6. Edge Detection.
7. Drawing Board LAB Using Webcam.

1. What is OpenCV?

- Open-source computer vision library.
- Built around **classical computer vision algorithms** (not AI-based).
- Great for basic tasks like filtering, edge detection, object tracking, etc.

1.2 What is Deep Learning?

- A subset of Machine Learning using **neural networks**, especially **Convolutional Neural Networks (CNNs)** for images.
- Excellent for **complex vision tasks**: image classification, face recognition, object detection.

1.3 Difference Between OpenCV and Deep Learning (DL):

Point	OpenCV	Deep Learning (DL)
Type	Based on rules and math	Learns from data
Flexibility	Fixed functions	Adapts and improves
Complexity	Good for simple tasks	Handles complex tasks
Learning	Doesn’t learn	Learns from examples
Speed	Fast, runs on normal computers	Slower, needs GPU for speed
Setup	Easy to set up	Requires models and datasets
Used For	Filters, edge detection, color tracking	Object detection, face recognition, classification

2. Introduction to OpenCV:

- OpenCV (Open-Source Computer Vision Library) is an open-source library designed for real-time computer vision tasks.

- Written in C++ but also provides Python bindings.

Features of OpenCV:

➢ Image processing (reading, writing, editing images).

➢ Real-time video processing.

➢ Machine learning tools for computer vision.

Pixels
EVERY PIXEL IS KNOWLEDGE

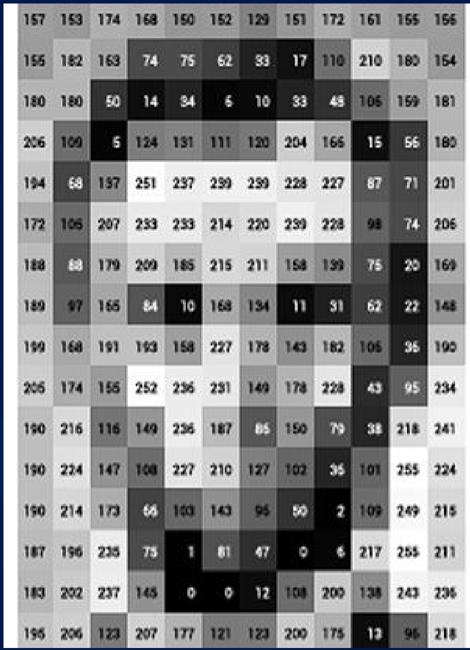
What is an image ?

2.1 What is an image ?

An image is represented by its dimensions (height and width) based on the number of pixels.

Example :

if the dimensions of an image are 100 x 100 (width x height), the total number of pixels in the image is 100000.



157	153	174	168	150	152	129	151	172	161	156	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	106	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

2.2 Setting Up OpenCV:

- Install OpenCV via Python's **"pip install OpenCV-python"** command.

```
C:\Users\Basma Ahmed>pip install opencv-python
Collecting opencv-python
  Downloading opencv-python-4.10.0.84.tar.gz (95.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6.6/95.1 MB 295.2 kB/s eta 0:05:00
```

- Import it in your Python code as **import cv2.**

```
import cv2
print(cv2.__version__)
```

The Output :

```
Run:  test
"C:\Users\Basma Ahmed\Pycharm
4.11.0
```

3. Basic Operations in OpenCV :

1. Reading Images/Videos:

1. cv2.imread(*image_name , *flags): Reads an image from a file.
   flag : 1 => colored img , 0 => grayscale img.

2. cv2.VideoCapture(arg)

   arg : 0 => open camera of computer OR ‘vid_name’

2. Save Image:

cv2.imwrite(filename, image): Saves an image to a file.

Pixels
EVERY PIXEL IS KNOWLEDGE

3. Display Image:

1. cv2.imshow(*window_title , *image).
2. cv2.waitKey(*milliseconds).
3. cv2.destroyAllWindows().

4. Edit Image:

1. cropped = image[y1:y2, x1:x2]
2. resized = cv2.resize(image, (width, height))
3. rotated = cv2.rotate(image, rotation_flag)
4. merged = cv2.addWeighted(img1, alpha, img2, beta, gamma)

Example 1 :

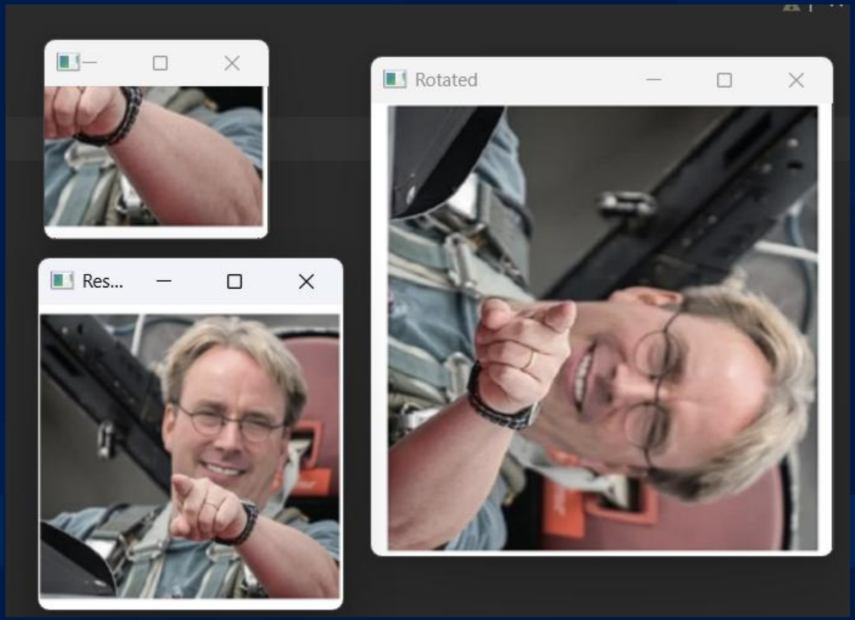
```python
import cv2
# Read an image in color mode (1) or grayscale mode (0)
image_color = cv2.imread("C:\\Users\\Basma Ahmed\\Pictures\\Screenshots\\Screenshot 2024-12-07 053802.png", 1)
image_gray = cv2.imread("C:\\Users\\Basma Ahmed\\Pictures\\Screenshots\\Screenshot 2024-12-07 053802.png", 0)

cv2.imshow(winname: "Color Image", image_color)
cv2.imshow(winname: "Grayscale Image", image_gray)

cv2.waitKey(0)
cv2.destroyAllWindows()
cv2.imwrite(filename: "new.png", image_color)
print("Image saved successfully!")
```

The Output :

Example 2 :

```
import cv2

img = cv2.imread('new.png')
cropped = img[100:300, 150:350]
# Resize
resized = cv2.resize(img, dsize: (200, 200))

# Rotate
rotated = cv2.rotate(img, cv2.ROTATE_90_CLOCKWISE)
# Display all
cv2.imshow( winname: "Cropped", cropped)
cv2.imshow( winname: "Resized", resized)
cv2.imshow( winname: "Rotated", rotated)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

The Output :



Example 3 :

Merging two photos :

..py | merge.py | new.png | test.py

```python
import cv2

image1 = cv2.imread('new.png')
image2 = cv2.imread("C:\\Users\\Basma Ahmed\\Pictures\\Screenshots\\Screenshot 2024-12-07 065433.png")

image2_resized = cv2.resize(image2, (image1.shape[1], image1.shape[0]))
merged_image = cv2.addWeighted(image1, 0.7, image2_resized, 0.3, 0)

cv2.imshow('Merged Image', merged_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

The Output :

Task 1:

1. Load 2 images.
2. Crop a specific part from one.
3. Resize and rotate it.
4. Then merge it onto another image (like placing a sticker).

4. Color Spaces :

A **color space** is a way to **represent colors** in an image using numbers. Each pixel has values that define its color in a specific color space.

Common Color Spaces:

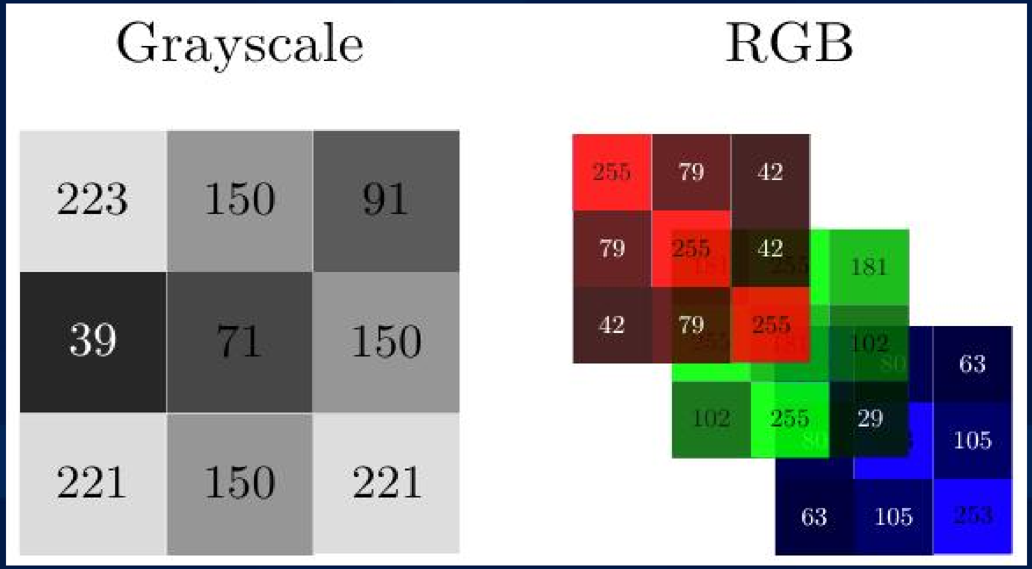
- BGR (default in OpenCV).
- Grayscale (for simplification).
- HSV (Hue, Saturation, Value).
- RGB (used mostly for visualization).

4.2 Images & Pixels :

Grayscale images contain only shades of gray, represented by a single intensity value per pixel, while RGB images consist of three – color channels (Red, Green, Blue) per pixel, allowing for a wider range of colors.

Grayscale :

each pixel can take
a value from (0 -> 255)
0 -> very dark , 255 -> very bright.



RGB :

each pixel can take
three values for each color from
(Red , Green , Blue)

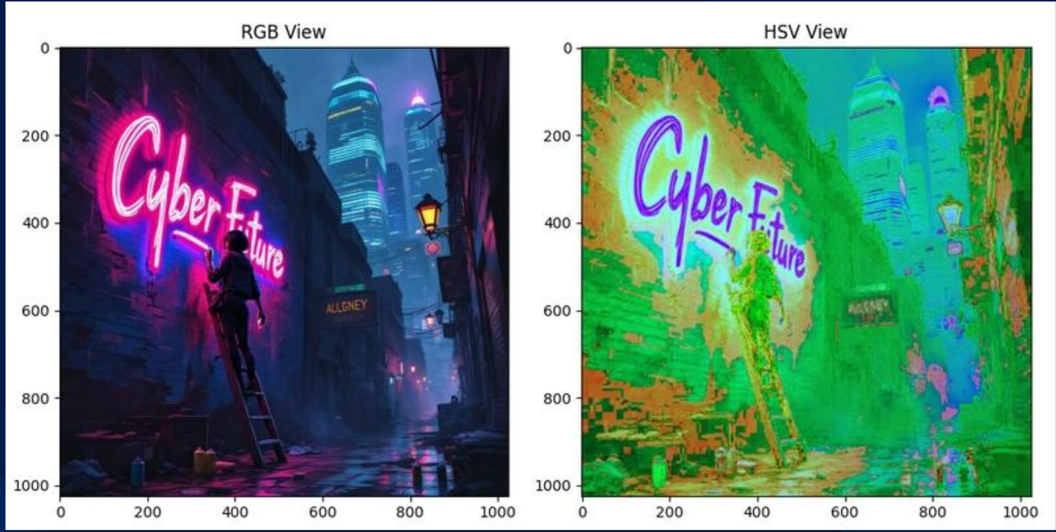
HSV Color Space:

Each pixel = **[H, S, V]**

H (Hue):
Color type → from **0 to 180.**

S (Saturation):
Color strength → **0 to 255.**

V (Value):
Brightness → **0 to 250 .**



Why HSV is Useful?

- It separates **color from lighting**.
- Makes it easier to **detect colors** regardless of lighting.
- Often used in object tracking, color masking, etc.

Example :

```python
import cv2

img = cv2.imread("new.png")

# Convert to Grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Convert to HSV
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)



# Show all
cv2.imshow( winname: "Original", img)
cv2.imshow( winname: "Grayscale", gray)
cv2.imshow( winname: "HSV", hsv)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

Task 2:

- **Read an image** from a file using cv2.imread().

- **Convert the image to grayscale** and display both original & grayscale versions.

- **Save the grayscale image** using cv2.imwrite().

- **Wait for a key press** before closing the windows.

Pixels
EVERY PIXEL IS KNOWLEDGE

5. Filters in Image Processing :

Filters are **image processing techniques** used to **enhance**, **blur**, **denoise**, or **sharpen** an image by modifying pixel values using a kernel (matrix).
filters are applied using **convolution** .

Types of Filters in OpenCV :

1. **Averaging (cv2.blur) :** takes the **average** of all pixels.

2. **Gaussian Blur (cv2.GaussianBlur) :** Applies **weighted blur** using a **Gaussian distribution.**

3. **Median Blur (cv2.medianBlur) :**Takes the **median** of all pixels .

4. **Bilateral Filter (cv2.bilateralFilter):** Blurs the image **while keeping edges sharp**

Example :

```python
import cv2

img = cv2.imread("new.png")

# Apply filters
blur = cv2.blur(img, (5, 5))
gaussian = cv2.GaussianBlur(img, (5, 5), 0)
median = cv2.medianBlur(img, 5)
bilateral = cv2.bilateralFilter(img, 9, 75, 75)

# Show results
cv2.imshow("Original", img)
cv2.imshow("Blur", blur)
cv2.imshow("Gaussian Blur", gaussian)
cv2.imshow("Median Blur", median)
cv2.imshow("Bilateral Filter", bilateral)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

Task 3:

1. Load an image of your choice

2. Apply:

   - Averaging Filter → cv2.blur()
   - Gaussian Blur → cv2.GaussianBlur()
   - Median Filter → cv2.medianBlur()

3. Display all images

4. Compare them and answer: Which filter smooths best?

6. Edge Detection :

Edge detection is the process of identifying points in an image where the **brightness changes sharply**, which usually indicates **boundaries of objects, shapes, or textures .**

Why Use Edge Detection?

- To **simplify** the image by reducing it to lines or contours.
- To help the system **understand the structure** of objects in an image.
- Preprocessing step before finding **contours** or **object shapes**

6.1 Common Edge Detection Methods :

1. Sobel Operator (cv2.Sobel)

Sobel is used to **detect edges** by finding the **gradient (change in intensity)** of the image.

Syntax:

```
sobelX = cv2.Sobel(image, cv2.CV_64F, 1, 0, ksize=5)  # X direction
sobelY = cv2.Sobel(image, cv2.CV_64F, 0, 1, ksize=5)  # Y direction
```

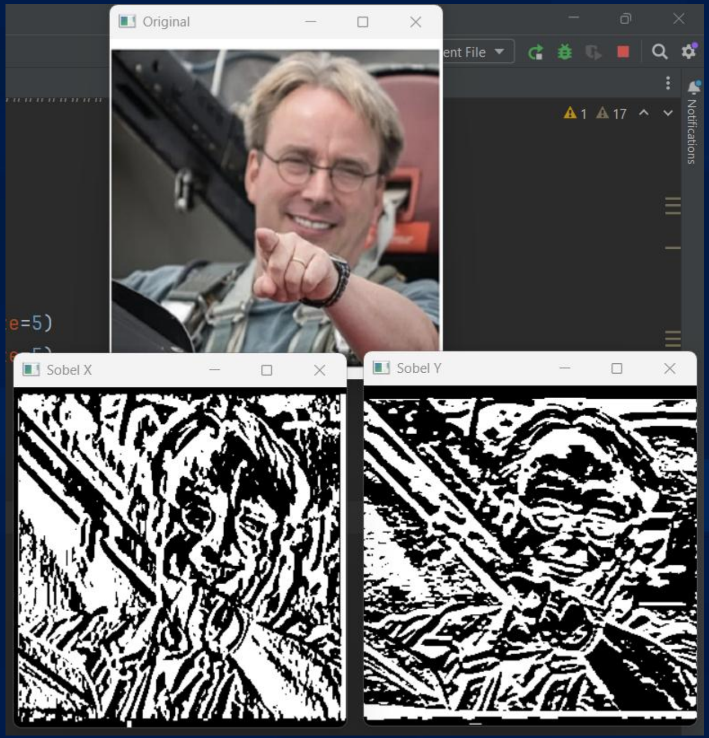
- **cv2.CV_64F**: Output data type (to hold negative gradients).
- **1, 0**: Apply derivative in X only.
- **0, 1**: Apply derivative in Y only.
- **ksize**: Kernel size (odd number, usually 3 or 5).

- **Use case: Emphasizing horizontal or vertical edges separately.**

Example :

```python
import cv2

img = cv2.imread("new.png")
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Sobel
sobelX = cv2.Sobel(gray, cv2.CV_64F, 1, 0, ksize=5)
sobelY = cv2.Sobel(gray, cv2.CV_64F, 0, 1, ksize=5)


# Show Results
cv2.imshow("Original", img)
cv2.imshow("Sobel X", sobelX)
cv2.imshow("Sobel Y", sobelY)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

The Output :



6.1 Common Edge Detection Methods :

2. Laplacian Operator (cv2.Laplacian)

The **Laplacian Operator** detects **edges in all directions at once** (no need to split X and Y like Sobel).

Syntax:

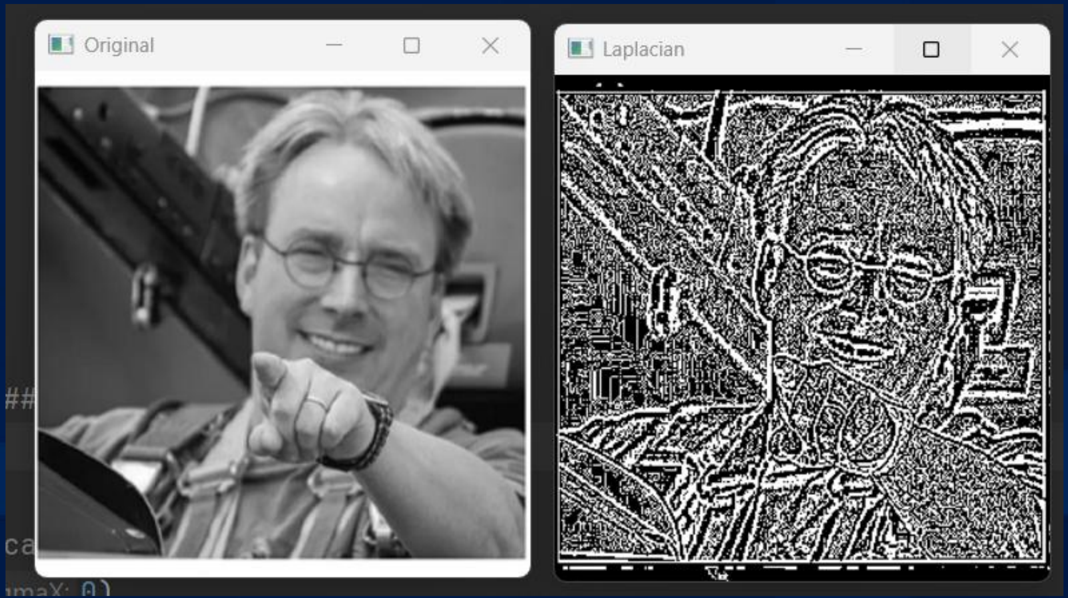
```
laplacian = cv2.Laplacian(image, cv2.CV_64F)
```

- **Use case: Quick edge detection in multiple directions**

Example :

```python
import cv2

img = cv2.imread("new.png", 0)
laplacian = cv2.Laplacian(img, cv2.CV_64F)

cv2.imshow( winname: "Original", img)
cv2.imshow( winname: "Laplacian", laplacian)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

The Output :



3. Canny Edge Detection (cv2.Canny):

Most popular and accurate method. It uses:

- Noise reduction (Gaussian Blur).
- Gradient calculation.
- Non-maximum suppression.
- Hysteresis thresholding.

Fig10. Input Image

Fig13. Canny Edge Detection

Syntax:

```
edges = cv2.Canny(image, threshold1, threshold2)
```

- **Use case: Object detection, face detection, lane detection, etc.**

6.2 How Canny Edge Detection Works in OpenCV ?

1. Noise Reduction (Gaussian Blur) :

- Images often have **noise** (random dots/pixels).
- This noise can be **mistaken as edges**.
- So, we apply a **Gaussian Blur** to smooth the image before edge detection.

2. Gradient Calculation :

- Calculates how fast the brightness is changing in **X and Y directions**.
- This detects **where edges might exist**.
- Uses Sobel filters under the hood.

6.2 How Canny Edge Detection Works in OpenCV ?

3. Non-Maximum Suppression :

- After finding edges, the result is often **thick**.

This step **thins the edges** by keeping only the **strongest pixel** along each edge line.
Makes the edges sharper and cleaner.

4. Hysteresis Thresholding :

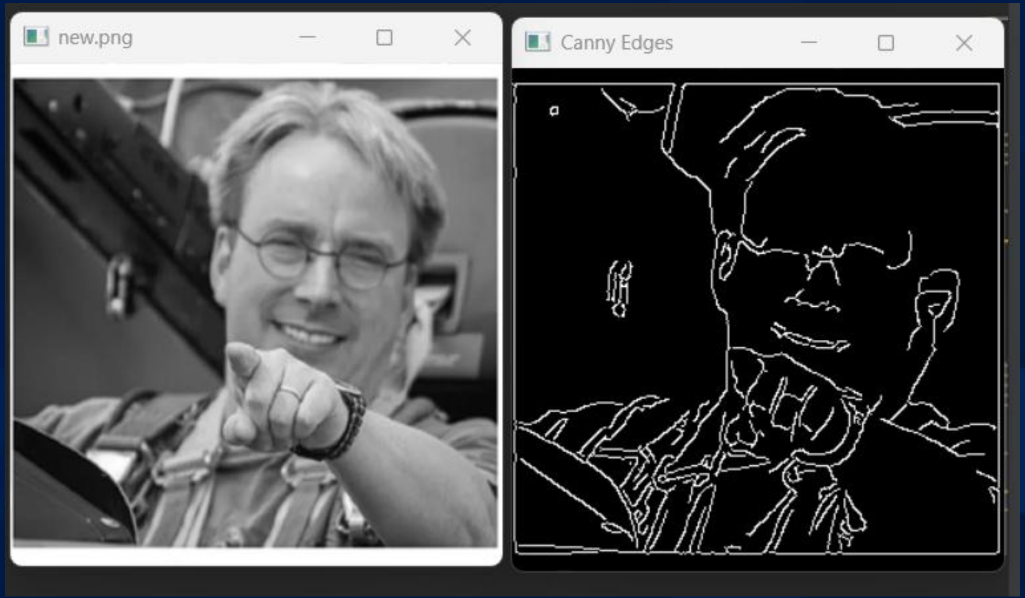
This step **decides which edges to keep** using two thresholds

- **Above threshold2** → Strong edge (definitely an edge).
- **Below threshold1** → Ignored (not an edge).
- **In between** → Weak edge (kept only if connected to strong edge).

Example :

```python
import cv2

img = cv2.imread("new.png", 0)  # Read in grayscale
blurred = cv2.GaussianBlur(img, ksize: (5, 5), sigmaX: 0)
edges = cv2.Canny(blurred, 100, 200)

cv2.imshow( winname: "new.png",img)
cv2.imshow( winname: "Canny Edges", edges)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

The Output :



Task 4:

1. Load an image.
2. Convert it to grayscale.
3. Apply **Canny edge detection.**
4. Show both the original and edge-detected images.

◇ ***(Bonus: Add Gaussian blur before edge detection)***

Pixels
EVERY PIXEL IS KNOWLEDGE

Break Time

7. Steps for Drawing Board LAB Using Webcam :

1. Access webcam using cv2.VideoCapture.
2. Convert image to **HSV** color space for color detection.
3. Create a **color mask** to track a specific color.
4. Use **contours** to find the object position.
5. Draw on a **blank canvas.**
6. Combine **live feed + canvas** for the final output .

7. Drawing Board LAB Using Webcam :

1. Accessing the Webcam :

```python
cap = cv2.VideoCapture(0)  # 0 = default camera

ret, frame = cap.read()
cv2.imshow( winname: "Webcam", frame)
cv2.waitKey(0)
cv2.destroyAllWindows()

```

7. Drawing Board LAB Using Webcam :

2. Color Spaces – BGR vs HSV :

Recap:

- OpenCV uses **BGR** by default.
- HSV is better for detecting colors .

3. Binary Masks :

A **mask** is a black & white image:

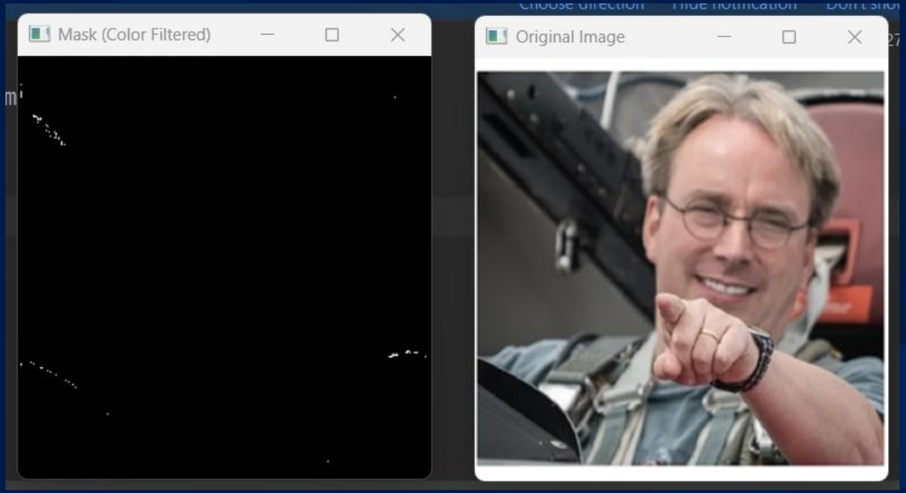
- White (255): where the color is detected.
- Black (0): where it's not

Pixels
EVERY PIXEL IS KNOWLEDGE

Example :

```python
import cv2
import numpy as np

frame = cv2.imread("new.png")  # Load an image (or use webcam frame)

# 1. Convert image from BGR to HSV color space
hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

# 2. Define HSV range for the target color (e.g., blue)
lower_bound = np.array([100, 150, 0])
upper_bound = np.array([140, 255, 255])

# 3. Create a mask - only pixels within range will be white (255), rest will be black (0)
mask = cv2.inRange(hsv, lower_bound, upper_bound)

# 4. Show the result
cv2.imshow("Original Image", frame)
cv2.imshow("Mask (Color Filtered)", mask)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

The Output :



7. Drawing Board LAB Using Webcam :

4. What Are Contours in OpenCV?

Contours are simply the **boundaries or outlines** of shapes in an image.
Think of contours as the "line" that wraps around any visible object.

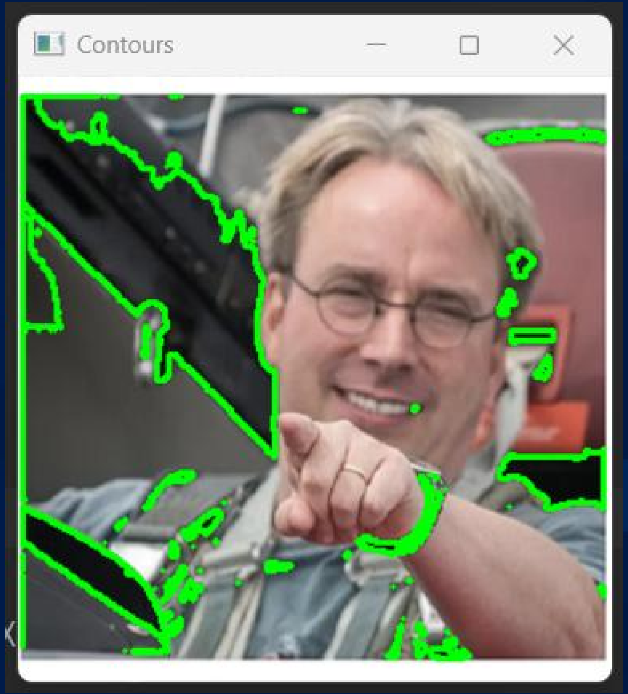
Why Do We Use Contours ?

- To **Detect Shapes.**
- To **Track Objects** :Once we detect the contour of an object (e.g., a ball), we can follow its **movement** frame by frame (in a video or webcam feed)

Example :

```python
import cv2
import numpy as np

# 1. Load and mask image
img = cv2.imread("new.png")
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
lower_black = np.array([0, 0, 0])
upper_black = np.array([180, 255, 50])
mask = cv2.inRange(hsv, lower_black, upper_black)

# 2. Find contours
contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

# 3. Draw contours on original image
cv2.drawContours(img, contours, -1, color: (0,255,0), thickness: 2)

# 4. Show result
cv2.imshow( winname: "Contours", img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

The Output :



7. Drawing Board LAB Using Webcam :

5. Drawing Functions in OpenCV :

- cv2.line(img, pt1, pt2, color, thickness): Draws a line .
- cv2.rectangle(img, pt1, pt2, color, thickness): Draws a rectangle .
- cv2.circle(img, center, radius, color, thickness): Draws a circle .
- cv2.putText(img, text, position, font, size, color, thickness): Adds text .

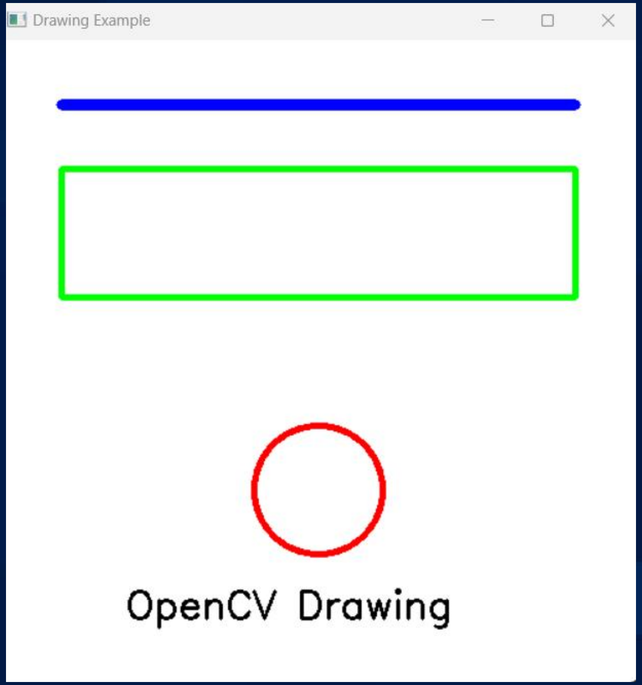
Pixels
EVERY PIXEL IS KNOWLEDGE

Example :

test.py ..py new.png

```python
import cv2
import numpy as np

image = np.ones((500, 500, 3), dtype=np.uint8) * 255

cv2.line(image, pt1: (50, 50), pt2: (450, 50), color: (255, 0, 0), thickness: 7)

cv2.rectangle(image, (50, 100), (450, 200), (0, 255, 0), 3)

cv2.circle(image, center: (250, 350), radius: 50, color: (0, 0, 255), thickness: 3)

cv2.putText(image, text: "OpenCV Drawing", org: (100, 450), cv2.FONT_HERSHEY_SIMPLEX, fontScale: 1, color: (0, 0, 0), thickness: 2)

cv2.imshow( winname: "Drawing Example", image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

The Output :



فيما أفناه

Task 5:

Draw different geometric shapes
(a rectangle, a circle, and a line) on an image .

Solution

test.py | ..py | new.png

```
import cv2
import numpy as np

# Step 1: Create a blank white image
image = np.ones((500, 500, 3), dtype=np.uint8) * 255

# Step 2: Draw a rectangle (top-left: (100, 100), bottom-right: (400, 400), color: blue)
cv2.rectangle(image, (100, 100), (400, 400), (255, 0, 0), 3)

# Step 3: Draw a circle (center: (250, 250), radius: 50, color: red)
cv2.circle(image, center: (250, 250), radius: 50, color: (0, 0, 255), thickness: 3)

# Step 4: Draw a line (from (50, 50) to (450, 450), color: green)
cv2.line(image, pt1: (50, 50), pt2: (450, 450), color: (0, 255, 0), thickness: 3)

# Step 5: Display the image
cv2.imshow( winname: "Geometric Shapes", image)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

Hands-On Practice:

Notebook:

https://colab.research.google.com/drive/17NP-H5QN9K4LBJ0F5MOZJ1bX0-Z-cKlB?usp=sharing

Notebook tasks:

https://colab.research.google.com/drive/1pZXG4cXhSUR5yNXULpp2Otvav-2gC5pU?usp=sharing

Any questions

THANK YOU .