TASK 5

SUBJECT:

Programming For AI

PROGRAM:

BS DATA SCIENCE

SUBMITTED TO:

Sir Rasikh Ali

SUBMITTED BY:

FIZZA FAROOQ

ROLL NUMBER:

SU92-BSDSM-F23-017

BSDS (4A)

Lab 5
Task 5. Open CV
Explanation of the Code

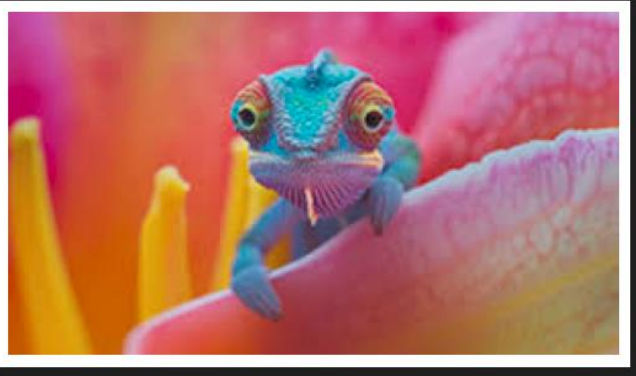
```
import cv2
import matplotlib.pyplot as plt
```

cv2 (OpenCV) is a powerful library for image processing and computer vision.**matplotlib.pyplot** is used for visualization, allowing images to be displayed in plots.

Reading an Image Using OpenCV

Reading image on opencv

```
import cv2
import matplotlib.pyplot as plt
img = cv2.imread("download.jpg", cv2.IMREAD_COLOR)
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.imshow(img_rgb)
plt.axis("off")
plt.show()
```

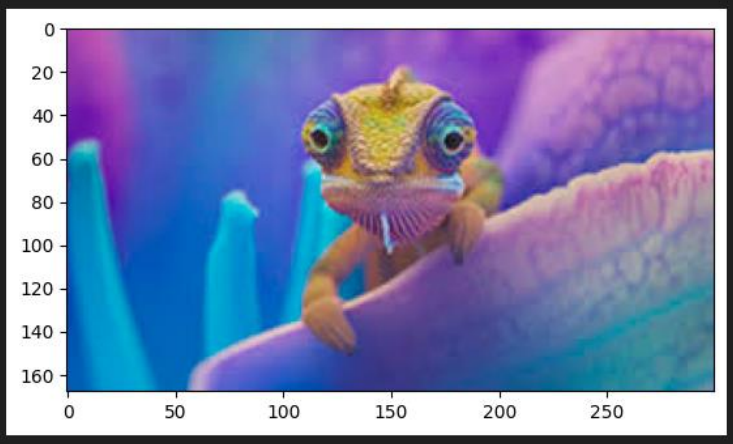


- `cv2.imread()` loads an image from a specified path.
- `"download.jpg"` is the image file being read.
- `cv2.IMREAD_COLOR` ensures that the image is read in color mode (ignoring any transparency information).

- OpenCV loads images in **BGR format** by default, whereas Matplotlib expects **RGB format**.

- `cv2.cvtColor(img, cv2.COLOR_BGR2RGB)` converts the image to RGB format so that it displays correctly.

```
img=cv2.imread("download.jpg")
#Displaying image using plt.imshow() method
plt.imshow(img)
plt.show()
```



- `plt.imshow(img_rgb)` displays the image.
- `plt.axis("off")` removes the axis labels to give a clean output.
- `plt.show()` renders the image on the screen.
- This again reads and displays the image, but since OpenCV loads images in BGR format, colors might appear incorrect.

Converting Image to Grayscale

convert img into grayscale

```
grayimg = cv2.cvtColor(RGB_img, cv2.COLOR_BGR2GRAY)
plt.imshow(grayimg, cmap='gray')
plt.axis("off")
plt.show()
```

- `cv2.COLOR_BGR2GRAY` converts the image into grayscale.
- `cmap='gray'` ensures that Matplotlib correctly displays the image in grayscale.
- This is useful for **reducing complexity** in image processing tasks.

Saving the Grayscale Image

save grayscale img

```
cv2.imwrite("gray_img.jpg", grayimg)
print("Gray scale img has been saved successfully ")
```

[6]

```
Gray scale img has been saved successfully
```

- `cv2.imwrite("gray_image.jpg", grayimg)` saves the processed grayscale image.
- The `print()` statement confirms that the image was successfully saved.
- Saves the grayscale image to disk.

Arthmetic operations

Arthmetic operations

```
import numpy as np
bright= cv2.add(RGB_img, np.ones(RGB_img.shape, dtype='uint8') * 10)
plt.imshow(bright)
plt.axis('off')
plt.show()
```

- Uses `cv2.add()` to increase brightness by adding 10 to all pixel values.

- Uses `np.ones()` to create an array of the same shape as the image.
- Displays the brightened image.

Applying Bitwise AND Operation

Bitwise operations in binary img

```
bitwise = cv2.bitwise_and(RGB_img, RGB_img)
plt.imshow(bitwise)
plt.axis('off')
plt.show()
```

- `cv2.bitwise_and()` performs an AND operation on the same image.
- The image remains unchanged, but this operation is useful in masking.

IMAGE PREPROCESSING

IMAGE RESIZE

IMAGE RESIZE

```
RGB_img= cv2.imread('download.jpg')
resize = cv2.resize(RGB_img, (300, 300))
plt.imshow(cv2.cvtColor(resize, cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.show()
```

[9]

Explanation:

- The function **`cv2.resize()`** resizes the image.
- `(300, 300)` specifies the new **width and height**.
- The aspect ratio **may change** if the new dimensions are not proportional.

Eroding the image

Eroding the img

```
kernel = np.ones((5,5), np.uint8)
Eroded = cv2.erode(RGB_img, kernel, iterations=1)

plt.imshow(cv2.cvtColor(Eroded, cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.show()
```

[10]

- **Erosion shrinks bright regions** (white areas in binary images).
- Used for **removing noise** or **isolating small objects**.
- A **5×5 kernel** slides over the image, removing pixels from edges.

Blur the image

Blur an img

```
blur = cv2.GaussianBlur(RGB_img, (15, 15), 0)
plt.imshow(cv2.cvtColor(blur, cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.show()
```

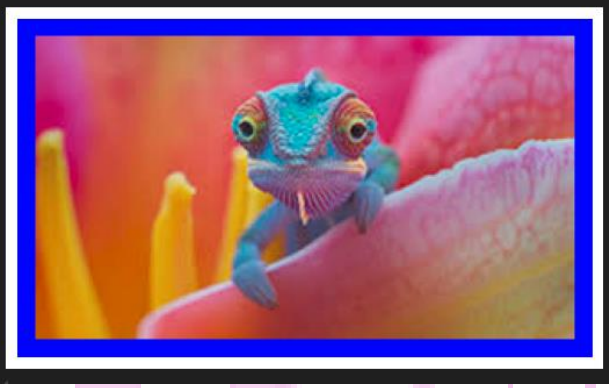
- Blurring removes **high-frequency noise**.
- The `(15,15)` **kernel size** controls how much the image is smoothed.
- Used in **edge detection and noise reduction**.

Create border around the image

- **Adds a 10-pixel border** around the image.
- The **color is blue** (255, 0, 0 in BGR).
- Useful for **padding** or **emphasizing regions**.

create Border around the img

```
border = cv2.copyMakeBorder(RGB_img, 10, 10, 10, 10, cv2.BORDER_CONSTANT, value=(255, 0, 0))
plt.imshow(cv2.cvtColor(border, cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.show()
```



GrayScalling The image

- Converts the image to **grayscale (black and white)**.
- Removes **color information**, leaving only brightness levels.
- Often used in **object detection and edge detection**.

Gray scaling the img

```
grayscaling = cv2.cvtColor(RGB_img, cv2.COLOR_BGR2GRAY)

plt.imshow(grayscaling, cmap='gray')
plt.axis('off')
plt.show()
```

1. Scalling

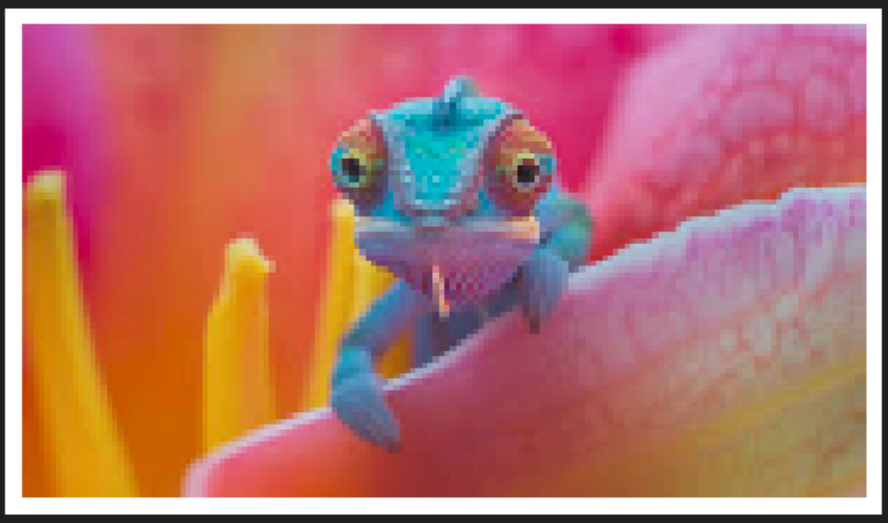
2. Resizes the image by **reducing width & height to 50%**.
3. `fx` and `fy` define the **scaling factors**.

1_Scalling

```
scaling = cv2.resize(RGB_img, None, fx=0.5, fy=0.5)
plt.imshow(cv2.cvtColor(scaling, cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.show()
```

[14]

...



4. Rotating

5. **Rotates** the image **45 degrees**.
6. The rotation is **centered**, maintaining the aspect ratio.

2.Rotating

```
(a,b) = RGB_img.shape[:2]
center = (b // 2, a // 2)
M = cv2.getRotationMatrix2D(center, 45, 1.0)
rotated = cv2.warpAffine(RGB_img, M, (b, a))

plt.imshow(cv2.cvtColor(rotated, cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.show()
```

15]

7. Shiftng

Moves the image **50 pixels right** and **30 pixels down**.

3_Shifting

```
f= np.float32([[1, 0, 50], [0, 1, 30]])
shifted = cv2.warpAffine(RGB_img, f, (b, a))
plt.imshow(cv2.cvtColor(shifted, cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.show()
```

[16]

8. Edge detection

9. Detects **edges** in the image.
10. **Threshold values (100, 200)** define which edges are detected.

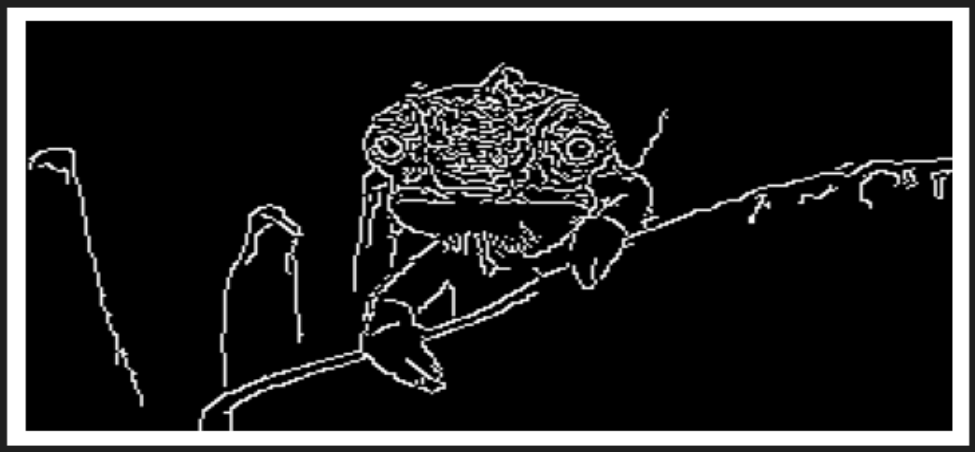
4_Edge detection

```
Edges = cv2.Canny(RGB_img, 100, 200)

plt.imshow(Edges, cmap='gray')
plt.axis('off')
plt.show()
```

[17]

...



Diliation

- **Expands bright regions**.
- Used to **thicken edges and objects**.

Dilation

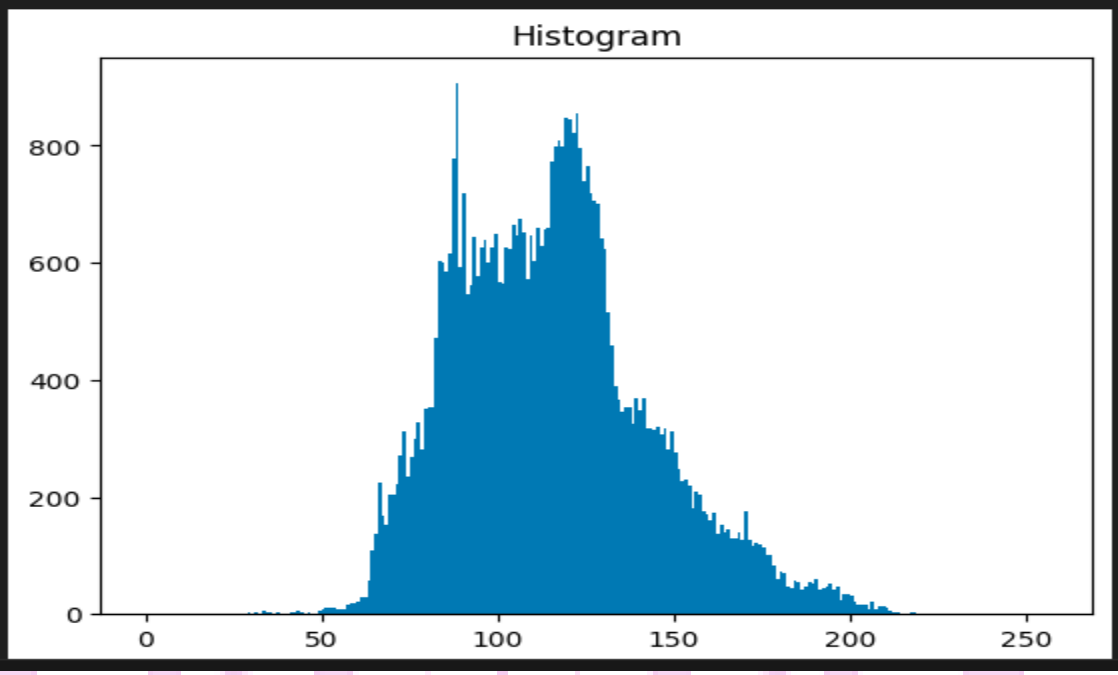
```
dilation= cv2.dilate(RGB_img, kernel, iterations=1)
plt.imshow(cv2.cvtColor(dilation, cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.show()
```

Analyze Using Histogram

- Plots the **pixel intensity distribution**.
- Used for **contrast enhancement**.

```
plt.hist(grayimg.ravel(), 256, [0, 256])
plt.title("Histogram")
plt.show()
```

```
C:\Users\M.A Computer\AppData\Local\Temp\ipykernel_5212\23
  plt.hist(grayimg.ravel(), 256, [0, 256])
```



SIMPLE THRESHOLDING
Image to binary

- **Converts grayscale to black & white**.
- **127** is the threshold value.

img --> binary

```
_, thresh = cv2.threshold(grayimg, 127, 255, cv2.THRESH_BINARY)
plt.imshow(thresh, cmap='gray')
plt.axis('off')
plt.show()
```

[20]

Adaptive thresholding

Adaptive thresholding adjusts the threshold dynamically.

Adaptiive thresholding

```
Adaptive = cv2.adaptiveThreshold(grayimg, 255, cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY, 11, 2)
plt.imshow(Adaptive, cmap='gray')
plt.axis('off')
plt.show()
```

[21]

Otsu Thresholding

- **Automatically finds the best threshold**.
- Finds an optimal cutoff to **maximize variance**.
- Ideal for **binarizing images without manual tuning**.

Otsu thresholding

```
_, otsu = cv2.threshold(grayimg, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
plt.imshow(otsu, cmap='gray')
plt.axis('off')
plt.show()
```

[22]

Segmentation Using thresholding

- Converts **black to white & white to black**.
- Used for **masking objects** in segmentation.
- Common in **document and medical image processing**.

segmentation using thresholding

```
_, segmented = cv2.threshold(grayimg, 150, 255, cv2.THRESH_BINARY_INV)
plt.imshow(segmented, cmap='gray')
plt.axis('off')
plt.show()
```

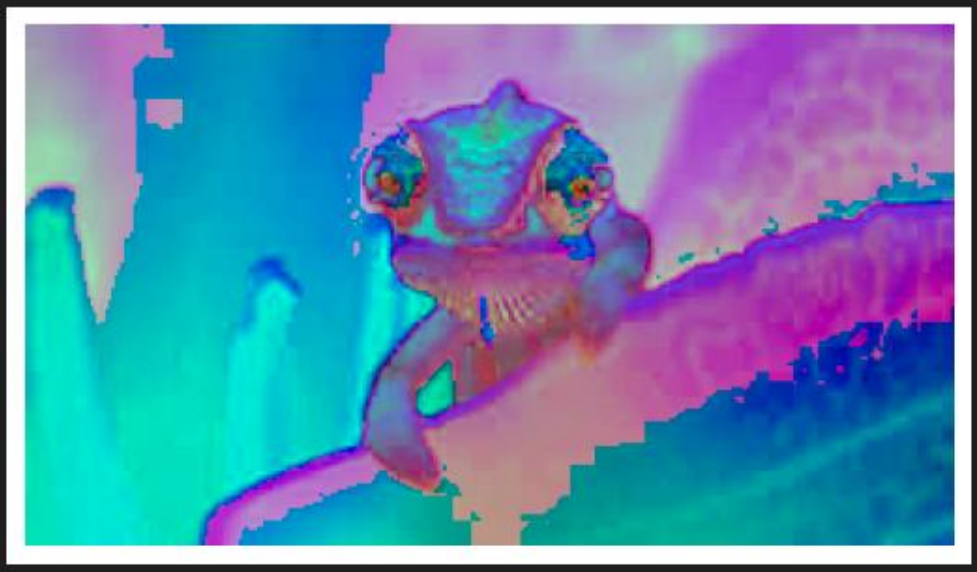
[23]

Image from one color to another

- Converts BGR to **HSV color space**.
- Separates color (`H`), saturation (`S`), and brightness (`V`).
- Useful for **color-based object detection**.

img from one color to anothor

```
hsv = cv2.cvtColor(RGB_img, cv2.COLOR_BGR2HSV)
plt.imshow(hsv)
plt.axis('off')
plt.show()
```



Denoising the colored image

- Removes **grainy noise while keeping edges sharp**.
- Uses `fastNlMeansDenoisingColored()` for colored images.
- Helps in **improving clarity** in noisy images.

Denoising the colored image

```
Denoise = cv2.fastNlMeansDenoisingColored(RGB_img, None, 10, 10, 7, 21)
plt.imshow(cv2.cvtColor(Denoise, cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.show()
```

[25]

View image in different colors

- Converts image to **LAB color space**.
- Useful for **color enhancement & correction**.
- Separates lightness (`L`) from color (`A` & `B`).

view image in different colors

```
img= cv2.cvtColor(RGB_img, cv2.COLOR_BGR2LAB)
plt.imshow(img)
plt.axis('off')
plt.show()
```

26]

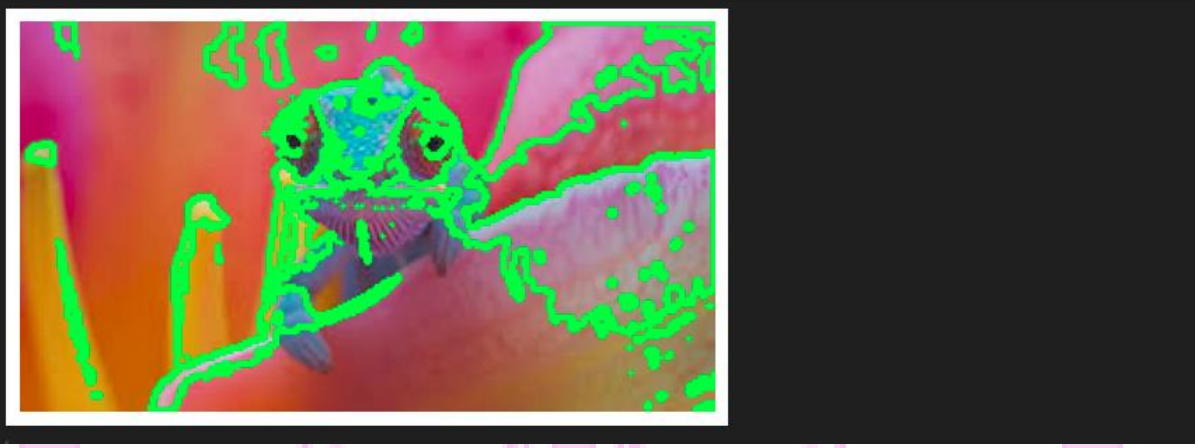
Coordinates of contours finding

- Finds **object boundaries** in an image.
- `cv2.findContours()` extracts shape outlines.
- Used in **shape detection & object tracking**.

Coordniates of contours finding

```
Contour, _ = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
cv2.drawContours(RGB_img, Contour, -1, (0, 255, 0), 2)
plt.imshow(cv2.cvtColor(RGB_img, cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.show()
```

[27]



Biletral filtering

- Reduces noise while **preserving edges**.
- Uses **spatial & intensity filtering** to smooth images.
- Helps in **cartoonization and skin smoothing**.

Bilateral filtering

```
RGB_img= cv2.imread('download.jpg')
bilateral = cv2.bilateralFilter(RGB_img, 9, 75, 75)
plt.imshow(cv2.cvtColor(bilateral, cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.show()
```

Image inpainting

- Restores **damaged/missing pixels** in an image.
- Uses `cv2.inpaint()` with `INPAINT_TELEA` algorithm.
- Useful for **removing scratches or unwanted objects**.

Image Inpainting

+ Code + Markdo

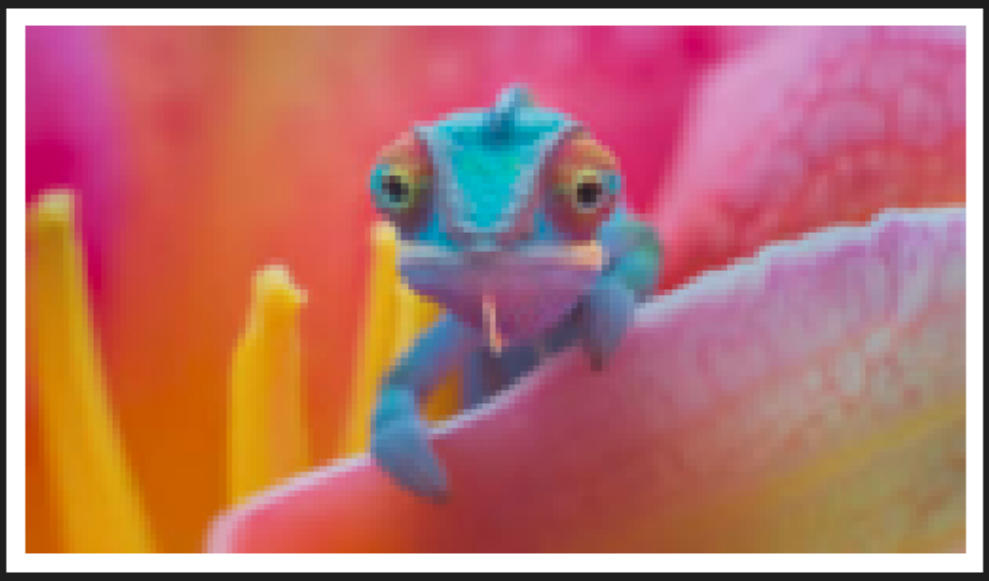
```
g = np.zeros(RGB_img.shape[:2], np.uint8)
Inpaint = cv2.inpaint(RGB_img, g, 3, cv2.INPAINT_TELEA)
plt.imshow(cv2.cvtColor(Inpaint, cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.show()
```

Image pyramids

- Reduces image resolution while **maintaining details**.
- Uses `cv2.pyrDown()` for smooth scaling.
- Helpful for **multi-scale analysis & object detection**.

Image pyramids

```
small= cv2.pyrDown(RGB_img)
plt.imshow(cv2.cvtColor(small, cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.show()
```



Morphological Operations

- Removes **small white noise** from images.
- `MORPH_OPEN = erosion + dilation`, cleaning tiny spots.
- Used for **background noise removal in binary images**.

Morphological Operations

```
X = cv2.morphologyEx(RGB_img, cv2.MORPH_OPEN, kernel)
plt.imshow(cv2.cvtColor(X, cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.show()
```

FEATURE DETECTION AND EXPLANATION

Used Houghline method for line detection

- Detects **edges** in an image using the **Canny edge detector**.
- Uses two thresholds (`100` and `200`) to filter out weak edges.
- Displays the detected edges in grayscale.

Used Houghline method for line detection

```
image = cv2.imread('download.jpg', cv2.IMREAD_GRAYSCALE)
edges = cv2.Canny(image, 50, 150)
lines = cv2.HoughLines(edges, 1, np.pi/180, 200)
image_clr = cv2.cvtColor(image, cv2.COLOR_GRAY2BGR)
if lines is not None:
    for rho, theta in lines[:, 0]:
        a, b = np.cos(theta), np.sin(theta)
        x0, y0 = a * rho, b * rho
        x1, y1 = int(x0 + 1000 * (-b)), int(y0 + 1000 * (a))
        x2, y2 = int(x0 - 1000 * (-b)), int(y0 - 1000 * (a))
        cv2.line(image_clr, (x1, y1), (x2, y2), (0, 0, 255), 2)
plt.imshow(cv2.cvtColor(image_clr, cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.show()
```

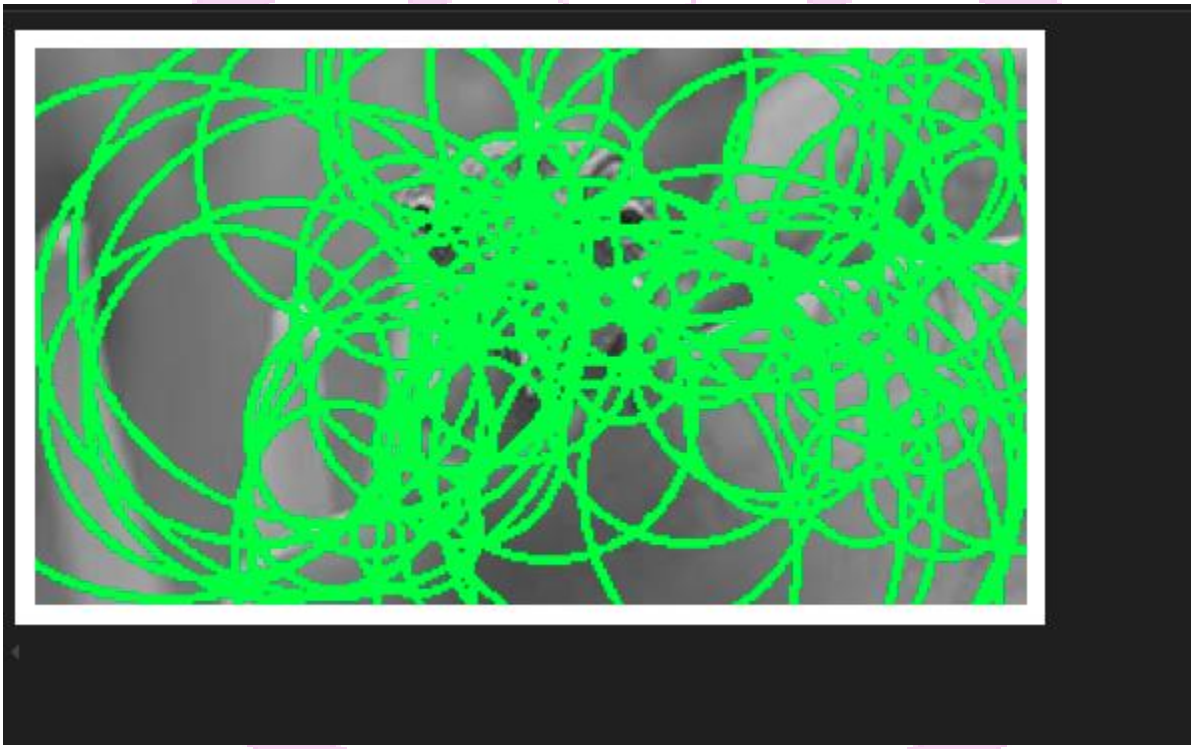
Use Houghcircles for circle detection

Explanation:

- Detects **circular shapes** in an image.
- Uses **gradient-based edge detection** for finding circles.
- Adjusts parameters like **min/max radius** and edge sensitivity.

Use Houghcircles for circle detection

```
image = cv2.imread('download.jpg', cv2.IMREAD_GRAYSCALE)
circle = cv2.HoughCircles(image, cv2.HOUGH_GRADIENT, dp=1.2, minDist=20, param1=50, param2=30, minRadius=10
image_clr= cv2.cvtColor(image, cv2.COLOR_GRAY2BGR)
if circle is not None:
    circles = np.uint16(np.around(circle))
    for i in circles[0, :]:
        cv2.circle(image_clr, (i[0], i[1]), i[2], (0, 255, 0), 2)
plt.imshow(cv2.cvtColor(image_clr, cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.show()
```

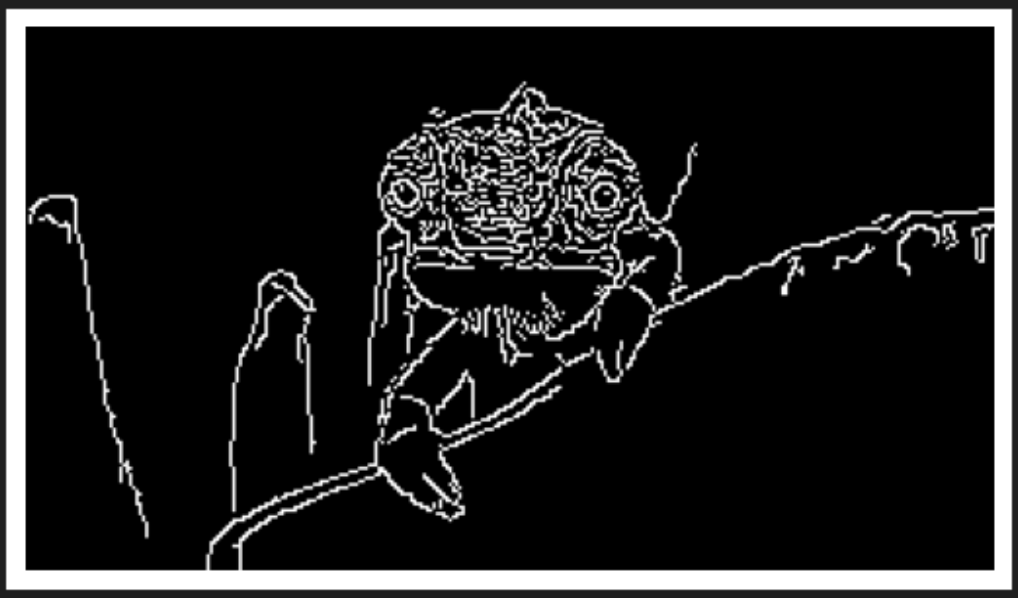


Detect Corners

Detect Corners

```
edge = cv2.Canny(RGB_img, 100, 200)
plt.imshow(edge, cmap='gray')
plt.axis('off')
plt.show()
```

34]



Corner Detection using harris corner

1. **Grayscale conversion aur Gaussian blur apply** kiya taake noise reduce ho.
2. **Canny edge detection se object boundaries detect** ki.
3. **Contours extract kiye aur ellipses fit** kiye using `cv2.fitEllipse()`.
4. **Ellipse green color me draw** kiya `cv2.ellipse()` se.

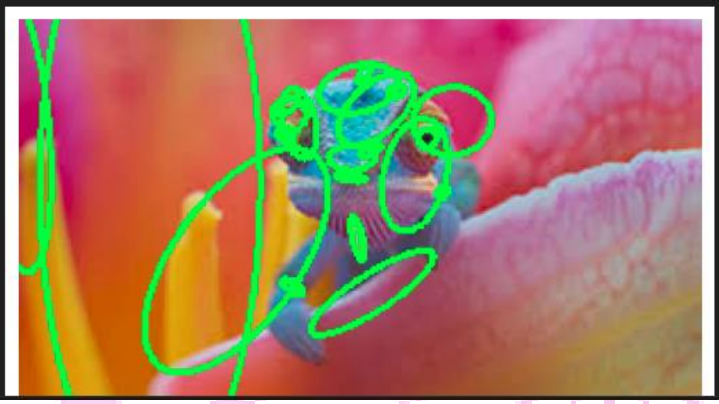
Corner Detection using harris corner

```
image = cv2.imread('download.jpg')
gray = cv2.cvtColor(RGB_img, cv2.COLOR_BGR2GRAY)
gray = np.float32(gray)
dst = cv2.cornerHarris(gray, 2, 3, 0.04)
dst = cv2.dilate(dst, None)
RGB_img[dst > 0.01 * dst.max()] = [0, 0, 255]
plt.imshow(cv2.cvtColor(RGB_img, cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.show()
```

Find ellips and corner from image

1. **Grayscale conversion aur Canny edge detection apply** kiya.
2. **Hough Transform se lines detect** ki using `cv2.HoughLines()`.
3. **Extracted lines ko red color me draw** kiya using `cv2.line()`.

```python
image = cv2.imread('download.jpg')
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
blurred = cv2.GaussianBlur(gray, (5, 5), 0)
edges = cv2.Canny(blurred, 50, 150)
contours, _ = cv2.findContours(edges, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
image_contour = image.copy()
for contour in contours:
    if len(contour) >= 5:
        ellipse = cv2.fitEllipse(contour)
        cv2.ellipse(image_contour, ellipse, (0, 255, 0), 2)
plt.imshow(cv2.cvtColor(image_contour, cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.show()
```



WORKING WITH VIDEOS

Play video with opencv

- **Video open kiya aur frame-by-frame read** kiya.
- **Frame ko RGB me convert** kiya taki colors accurate dikhain.
- **Frame ko display** karne ke liye `PIL.Image.open()` use kiya.
- **Previous frames ko remove karne ke liye `clear_output(wait=True)` use** kiya.

play video using opencv

```
import cv2
import IPython.display
from IPython.display import display, clear_output
import PIL.Image
import io
cap = cv2.VideoCapture('videoplayback.mp4')
if not cap.isOpened():
    print("Error opening video file")
while cap.isOpened():
    ret, frame = cap.read()

    if not ret:
        break
    frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    _, buffer = cv2.imencode('.jpg', frame_rgb)
    img_bytes = io.BytesIO(buffer)
    clear_output(wait=True)
    display(PIL.Image.open(img_bytes))
cap.release()
print("Video playback completed.")
```

Creating videos from multiple images

1. **Folder ke andar images ko sorted order me read** kiya.
2. **Video ka resolution aur framerate set** kiya.
3. **Images ko video me convert** kiya using `cv2.VideoWriter()`.

creating the video from multiple images

```
import os
image_folder = "images"
images = sorted([img for img in os.listdir(image_folder) if img.endswith(".jpg")])
if not images:
    print("No images found in the folder!")
else:
    first_image = cv2.imread(os.path.join(image_folder, images[0]))
    height, width, layers = first_image.shape
    video = cv2.VideoWriter('videoplayback.mp4', cv2.VideoWriter_fourcc(*'mp4v'), 5, (width, height))
    for image in images:
        img_path = os.path.join(image_folder, image)
        img = cv2.imread(img_path)
        video.write(img)
    video.release()
    print(" Video created successfully: videoplay.mp4")
```

✓ 0.0s

```
Video created successfully: videoplay.mp4
```

Extract images from files

1. **Video open kiya aur har frame extract** kiya.
2. **Frames ko `"frame_0000.jpg"`, `"frame_0001.jpg"` ke format me save** kiya.
3. **Total extracted frames count print** kar diya.

extract the images fom video

```python
import cv2
import os
video_path = "videoplayback.mp4"
cap = cv2.VideoCapture(video_path)
if not cap.isOpened():
    print("Error opening video file")
output_folder = "extracted_frames"
os.makedirs(output_folder, exist_ok=True)

frame_count = 0
while cap.isOpened():
    ret, frame = cap.read()

    if not ret:
        break
    frame_filename = os.path.join(output_folder, f"frame_{frame_count:04d}.jpg")
    cv2.imwrite(frame_filename, frame)

    frame_count += 1
cap.release()
print(f" {frame_count} frames extracted and saved in '{output_folder}'")
```

0s ✓ 0.2s

```
3 frames extracted and saved in 'extracted_frames'
```