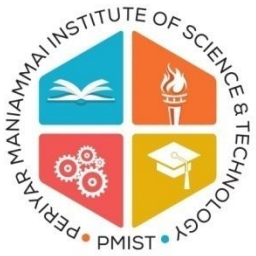
# IMAGE PROCESSING AND VISION TECHNIQUES
# LAB MANUAL




PERIYAR
MANIAMMAI
INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University)
Established Under Sec. 3 of UGC Act, 1956 • NAAC Accredited
think • innovate • transform

DEPARTMENT OF COMPUTER SCIENCE
VALLAM, THANJAVUR

# Index

**LAB MANUAL**
**Computer Vision Using OpenCV**

**Experiment 1: Fundamental Image Operations**
**Aim:**
Perform fundamental operations on an image using OpenCV

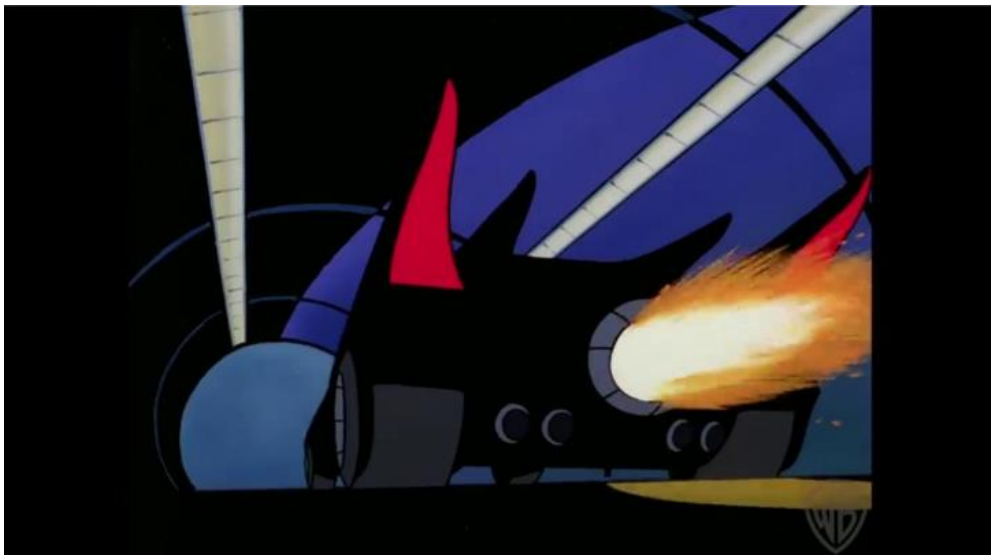
**Operations:**

1. Import library
2. Reading an Image
3. Displaying an Image
4. Displaying a video
5. Converting image to gray scale
6. Wait for a key press to close the windows
7. Close all OpenCV windows

**Pseudo Code:**

```
# Import necessary libraries
import cv2
# Load an image from file
image = cv2.imread("image_path.jpg")

# Display the image
cv2.imshow("Original Image", image)

# Convert the image to grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Display the grayscale image
cv2.imshow("Grayscale Image", gray_image)

# Wait for a key press to close the windows
cv2.waitKey(0)
# Close all OpenCV windows
cv2.destroyAllWindows()

# Display a video
video_path = "video_file_path.mp4"
video_capture = cv2.VideoCapture(video_path)
if not video_capture.isOpened():
  print("Error: Could not open video file.")
  exit()
    while True:
      ret, frame = video_capture.read()
      if not ret:
         print("Error: Failed to read frame.")
         break
       cv2.imshow("Video", frame)
       if cv2.waitKey(1) & 0xFF == ord('q'):
          break
video_capture.release()
cv2.destroyAllWindows()
```

**Result:**



**Experiment 2: Image Manipulation**

**Aim:**

Manipulate an Image using OpenCV

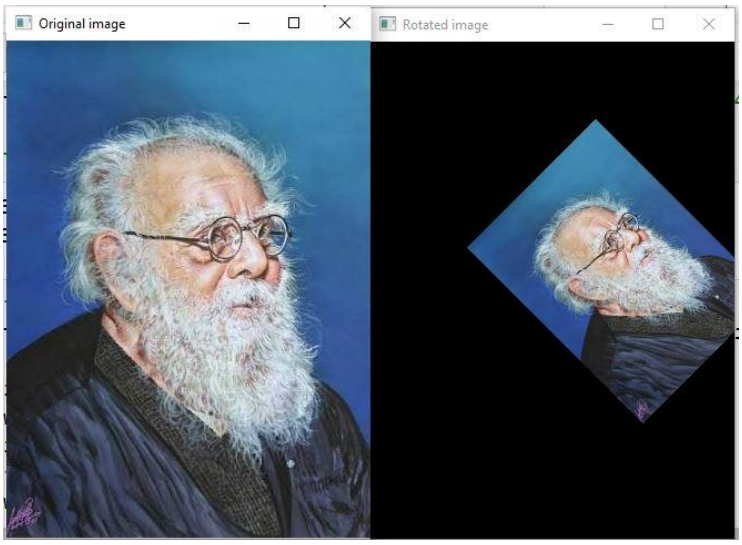
**Operations:**

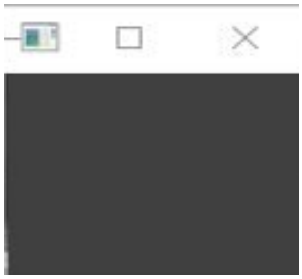
1. Import library
2. Loading an Image
3. Displaying an Image
4. Resizing an Image
5. Rotate an Image
6. Crop an Image
7. Blurring an Image

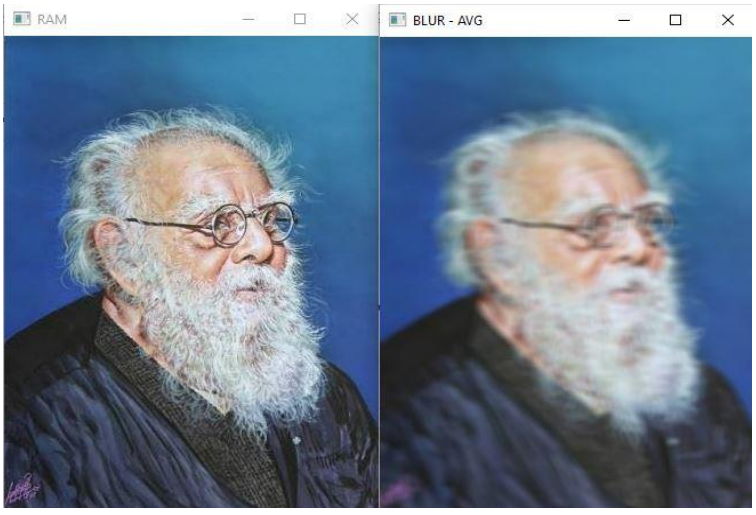
**Pseudo Code:**

```
# Import necessary libraries
import cv2

# Load an image from file
image = cv2.imread("image_path.jpg")

# Display the image
cv2.imshow("Original Image", image)

# Resize the image
resized_image = cv2.resize(image, (new_width, new_height))

# Display the resized image
cv2.imshow("Resized Image", resized_image)

# Rotate the image
rotation_matrix = cv2.getRotationMatrix2D(center, angle, scale)
rotated_image = cv2.warpAffine(image, rotation_matrix, (width, height))
cv2.imshow("Rotated Image", rotated_image)

# Crop a region of interest (ROI) from the image
roi = image[y1:y2, x1:x2]
cv2.imshow("ROI", roi)

# Apply Gaussian blur to the image
blurred_image = cv2.GaussianBlur(image, (kernel_width, kernel_height),
sigma)
cv2.imshow("Blurred Image", blurred_image)

# Wait for a key press to close the windows
cv2.waitKey(0)
# Close all OpenCV windows
cv2.destroyAllWindows()
```

**RESULT:**





**Experiment 3: Image thresholding**

**Aim:**

Perform Image thresholding using OpenCV

**Operations:**

1. Import library
2. Load an Image
3. Convert Image to grayscale image
4. Apply Binary thresholding
5. Apply Binary inverse thresholding
6. Apply Adaptive thresholding (Gaussian)
7. Apply Adaptive thresholding (Mean)
8. Display the images

**Pseudocode:**

***# Import necessary libraries***

```
import cv2
```

***# Load an image from file***

```
image = cv2.imread("image_path.jpg")
```

***# Convert the image to grayscale***

```
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

***# Apply Binary thresholding method***

```
ret, binary_thresholded_image = cv2.threshold(gray_image, threshold_value,
max_value, cv2.THRESH_BINARY)
```

***# Apply Binary inverse threshold method***

```
ret, binary_inverse_thresholded_image = cv2.threshold(gray_image,
threshold_value, max_value, cv2.THRESH_BINARY_INV)
```

***# Apply Adaptive thresholding (Gaussian)***

```
adaptive_thresholded_image_gaussian = cv2.adaptiveThreshold(gray_image,
max_value, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, block_size,
C)
```

***# Apply Adaptive thresholding (Mean)***

```
adaptive_thresholded_image_mean = cv2.adaptiveThreshold(gray_image,
max_value, cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY, block_size, C)
```

***# Display the original and thresholded images for each method***

```
cv2.imshow("Original Image", image)
cv2.imshow("Binary Thresholded Image", binary_thresholded_image)
cv2.imshow("Binary Inverse Thresholded Image",
binary_inverse_thresholded_image)
cv2.imshow("Adaptive Thresholded Image (Gaussian)",
adaptive_thresholded_image_gaussian)
cv2.imshow("Adaptive Thresholded Image (Mean)",
adaptive_thresholded_image_mean)
```

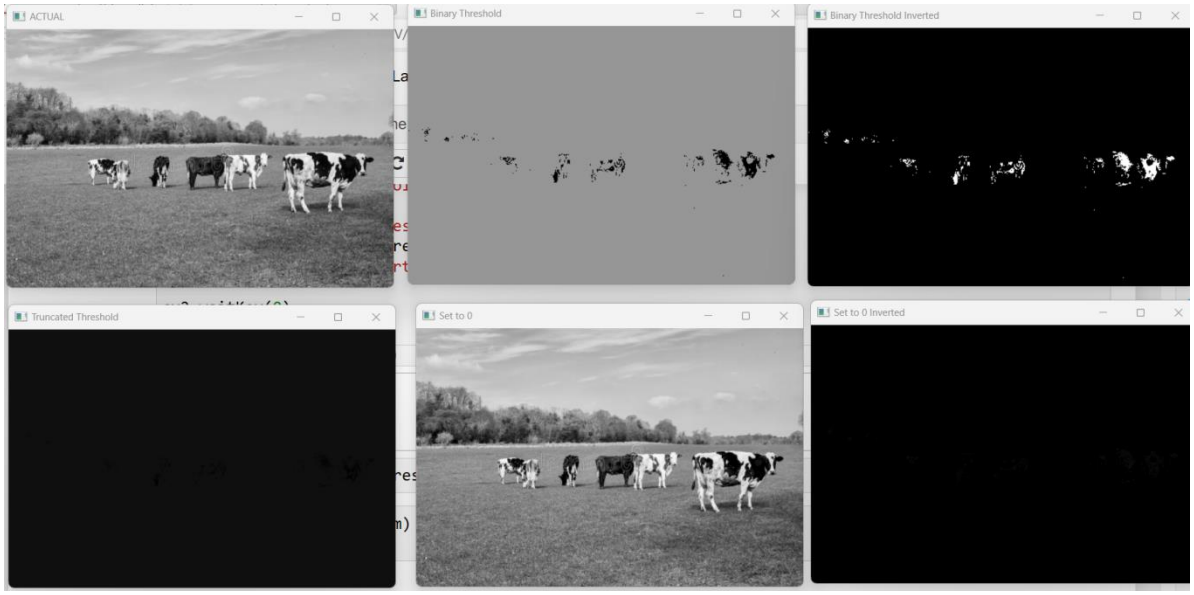
***# Wait for a key press to close the windows***

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

***# Close all OpenCV windows***

```
cv2.destroyAllWindows()
```

**Result:**



**Experiment 4: Image Filtering**
**Aim:**
Perform Image filtering using OpenCV

**Operations:**

1. Import library
2. Load an Image
3. Convert Image to grayscale image
4. Apply Gaussian Blur
5. Apply Median blur
6. Apply Bilateral blur
7. Apply Box blur
8. Display the images

**Pseudocode:**

***# Import necessary libraries***
```
import cv2
```

***# Load an image from file***
```
image = cv2.imread("image_path.jpg")
```

***#Convert the image to grayscale***
```
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

***#Apply different image filters***
***#Gaussian blur***
```
gaussian_blurred_image = cv2.GaussianBlur(image, (kernel_size_x,
kernel_size_y), sigma_x)
```

***#Median blur***
```
median_blurred_image = cv2.medianBlur(image, kernel_size)
```

***#Bilateral filter***
```
bilateral_filtered_image = cv2.bilateralFilter(image, diameter,
sigma_color, sigma_space)
```

***#Box filter (simple averaging)***
```
box_filtered_image = cv2.boxFilter(image, -1, (kernel_size_x,
kernel_size_y))
```

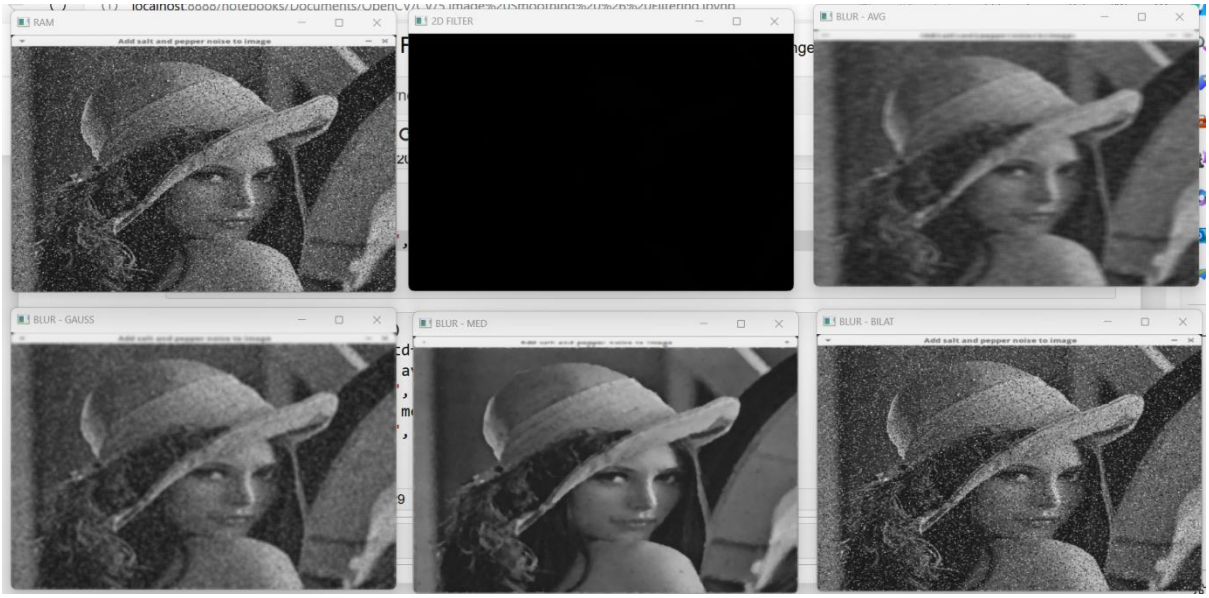
***#Display the original and filtered images for each method***
```
cv2.imshow("Original Image", image)
cv2.imshow("Gaussian Blurred Image", gaussian_blurred_image)
cv2.imshow("Median Blurred Image", median_blurred_image)
cv2.imshow("Bilateral Filtered Image", bilateral_filtered_image)
cv2.imshow("Box Filtered Image", box_filtered_image)
```

***#Wait for a key press to close the windows***
```
cv2.waitKey(0)
```

***#Close all OpenCV windows***

```
cv2.destroyAllWindows()
```

**Result :**



**Experiment 5: Morphological Operations**

**Aim:**

Perform Morphological Operations on an Image using OpenCV

**Operations:**

1. Import library
2. Load an Image
3. Convert Image to grayscale image
4. Define a kernel for the operations
5. Erosion
6. Dilation
7. Opening
8. Closing
9. Top Hat
10. Bottom Hat
11. Display the outcome

**Pseudocode:**

```
# Import necessary libraries
import cv2

# Load an image from file
image = cv2.imread("image_path.jpg")

# Convert the image to grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Define a kernel for the operations
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (kernel_width,
kernel_height))

# Erosion
eroded_image = cv2.erode(gray_image, kernel, iterations=iterations)

# Dilation
dilated_image = cv2.dilate(gray_image, kernel, iterations=iterations)

# Opening (Erosion followed by dilation)
opened_image = cv2.morphologyEx(gray_image, cv2.MORPH_OPEN, kernel,
iterations=iterations)

# Closing (Dilation followed by erosion)
closed_image = cv2.morphologyEx(gray_image, cv2.MORPH_CLOSE, kernel,
iterations=iterations)

# Morphological gradient (Difference between dilation and erosion)
gradient_image = cv2.morphologyEx(gray_image, cv2.MORPH_GRADIENT, kernel,
iterations=iterations)
```

***# Top hat (Original image - Opening)***

```
tophat_image = cv2.morphologyEx(gray_image, cv2.MORPH_TOPHAT, kernel,
iterations=iterations)
```

***# Black hat (Closing - Original image)***

```
blackhat_image = cv2.morphologyEx(gray_image, cv2.MORPH_BLACKHAT, kernel,
iterations=iterations)
```

***# Display the original and processed images for each operation***

```
cv2.imshow("Original Image", gray_image)
cv2.imshow("Eroded Image", eroded_image)
cv2.imshow("Dilated Image", dilated_image)
cv2.imshow("Opened Image", opened_image)
cv2.imshow("Closed Image", closed_image)
cv2.imshow("Gradient Image", gradient_image)
cv2.imshow("Top Hat Image", tophat_image)
cv2.imshow("Black Hat Image", blackhat_image)
```

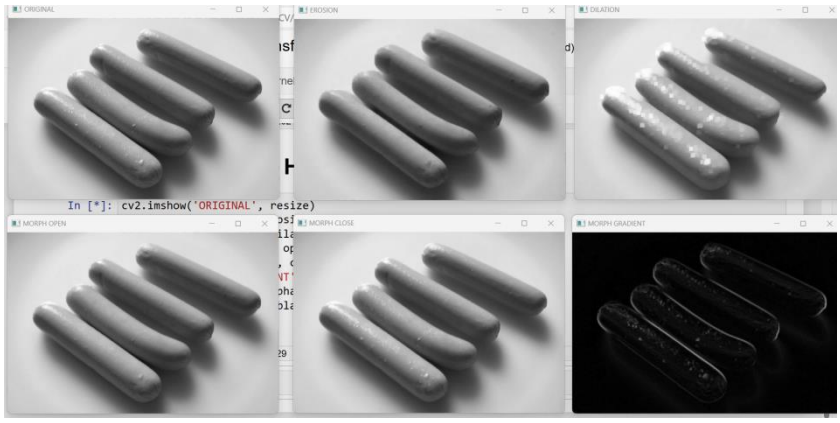
***# Wait for a key press to close the windows***

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

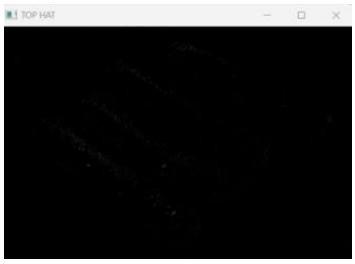
***# Close all OpenCV windows***

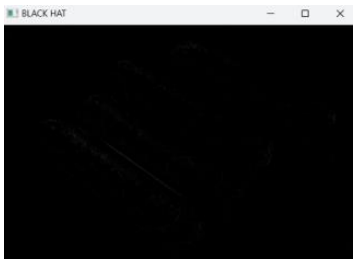
```
cv2.destroyAllWindows()
```

**Result :**







**Experiment 6: Image segmentation using Watershed Algorithm**

**Aim:**
Perform image segmentation using watershed algorithm in OpenCV

**Pseudocode:**

```
# Import necessary libraries
import cv2
import numpy as np
from matplotlib import pyplot as plt

# Load an image from file
image = cv2.imread("image_path.jpg")
plt.imshow(image)

# Convert the image to grayscale
img_RGB = cv2.cvtColor(image,cv2.COLOR_BGR2RGB)
img_gray = cv2.cvtColor(img_RGB,cv2.COLOR_RGB2GRAY)
plt.imshow(img_gray)



# Image thresholding

Image_median = cv2.medianBlur(img_gray,1)
ret,threshold_image = cv2.threshold(Image_median, threshold_value, 255,
cv2.THRESH_BINARY_INV)
adaptive_threshold_image =
cv2.adaptiveThreshold(Image_median,threshold_value,cv2.ADAPTIVE_THRESH_GAUS
SIAN_C,\cv2.THRESH_BINARY,11,2)
plt.imshow(ath2)

# Noise Removal
kernal = np.ones((3,3),np.uint8)
opening_image =
cv2.morphologyEx(adaptive_threshold_image,cv2.MORPH_OPEN,kernal, iterations
= 5)
plt.imshow(opening_image)

# Sure Background Area
sure_bg = cv2.dilate(opening_image,kernal,iterations = 4)
plt.imshow(sure_bg)

# Finding Sure Foreground Area
dist_transform = cv2.distanceTransform(opening_image,cv2.DIST_L2,5)
plt.subplot(1,2,1),plt.imshow(dist_transform)
plt.title('Dist_transform'),
ret, sure_fg = cv2.threshold(dist_transform,0.7*dist_transform.max(),255,0)
plt.subplot(1,2,2),plt.imshow(sure_fg)
```

```
plt.title('Sobel X')
```

***# Finding Unknown Region***

```
sure_fg = np.uint8(sure_fg)
unknown = cv2.subtract(sure_bg,sure_fg)
plt.imshow(unknown)
```

***# Marker Labelling***

```
ret, markers = cv2.connectedComponents(sure_fg)
markers = markers+1
plt.imshow(markers)
```

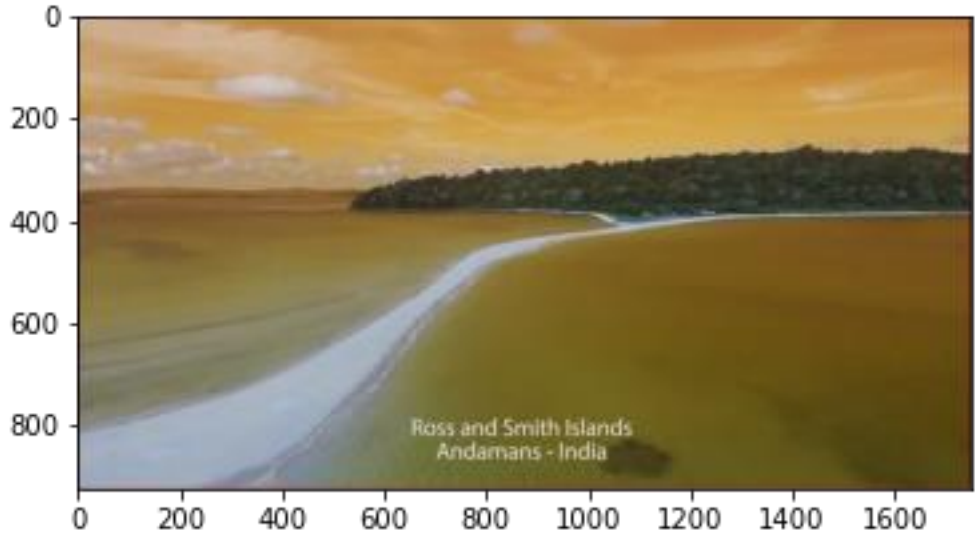
***# Marking Region of Unknown***

```
markers[unknown==255] = 5
plt.imshow(markers)
```

***# Applying Watershed***

```
markers = cv2.watershed(image,markers)
image[markers == -1] = [255,0,0]
plt.imshow(image)
```

**Result :**



**Experiment 7: Image segmentation using Mean shift algorithm**

**Aim:**
Perform image segmentation using mean shift algorithm in OpenCV

**Pseudocode:**

```
# Import necessary libraries
import cv2

# Load an image from file
image = cv2.imread("image_path.jpg")

# Convert the image to the required format for Mean Shift
# Define parameters for Mean Shift
spatial_radius = 10
color_radius = 20
max_pyramid_level = 2

# Apply Mean Shift algorithm for segmentation
shifted_image = cv2.pyrMeanShiftFiltering(image, sp=spatial_radius,
sr=color_radius, maxLevel=max_pyramid_level)

# Display the original and segmented images
cv2.imshow("Original Image", image)
cv2.imshow("Mean Shift Segmented Image", shifted_image)

# Wait for a key press to close the window
cv2.waitKey(0)

# Close OpenCV window
cv2.destroyAllWindows()
```

**Result :**

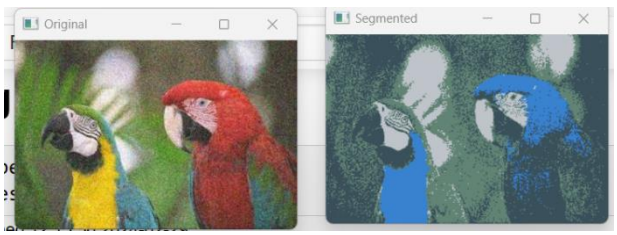
**Experiment 8: Image segmentation using Clustering**

**Aim:**
Perform image segmentation using mean shift algorithm in OpenCV

**Pseudocode:**

```
# Import necessary libraries
import cv2
import numpy as np

# Load an image from file
image = cv2.imread("image_path.jpg")

# RGB conversion
img_rgb = cv2.cvtColor(image,cv2.COLOR_BGR2RGB)

# Reshaping image
img_res = img_rgb.reshape((-1,3))

# Datatype Conversion
vec = np.float32(img_res)

# Iteration Termination Criteria
crit = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER,1000,0.2)
# Cluster Initiation
k = 3
attempts = 10
# K-Means clustering
ret,labels,center = cv2.kmeans (vec,k,None,crit,attempts
,cv2.KMEANS_RANDOM_CENTERS)
# Unit Clustering
center = np.uint8(center)
# Flattening and Reshaping
img_seg = center[labels.flatten()]
img_seg = img_seg.reshape((img.shape))

# Display Original and Segmented Image
cv2.imshow('Original Image',img)
cv2.imshow('Segmented Image',img_seg)
if cv2.waitKey(0) & 0xFF == 27:
    cv2.destroyAllWindows()
```

**Result:**



**Experiment 9: Implementation of Edge detection and Feature Extraction using Histogram of Oriented Gradients (HOG)**

**Aim:**

To perform

1. Edge detection using Canny algorithm
2. Feature extraction using Histogram of oriented gradients

**Operations:**

1. Edge detection using Canny algorithm
2. Feature extraction using Histogram of oriented gradients

**Pseudocode:**

**1. Edge detection using Canny algorithm**

```
# Import necessary libraries
import cv2
import numpy as np
from matplotlib import pyplot as plt
% matplotlib inline

# Load an image from file
image = cv2.imread("image_path.jpg")

# Convert the image to grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Apply Gaussian blur to reduce noise
blurred_image = cv2.GaussianBlur(gray_image, (kernel_size_x,
kernel_size_y), sigma_x)

# Perform edge detection using the Canny algorithm
edges = cv2.Canny(blurred_image, threshold1, threshold2)

# Display the original and edge-detected images
cv2.imshow("Original Image", image)
cv2.imshow("Edge-detected Image", edges)

# Wait for a key press to close the window
cv2.waitKey(0)

# Close OpenCV window
cv2.destroyAllWindows()
```

**Result :**

**Feature Extraction using Histogram of Oriented Gradients**

***# Import necessary libraries***

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
% matplotlib inline
from skimage.feature import hog
from skimage import data,exposure
```

***# Load an image from file***

```
image = cv2.imread("image_path.jpg")
```

***# Applying HOG***

```
fc,img_hog = hog(img,orientations = 8, pixels_per_cell = (16,6),
cells_per_block = (1,1), visualize = True,
                 multichannel = True)
```

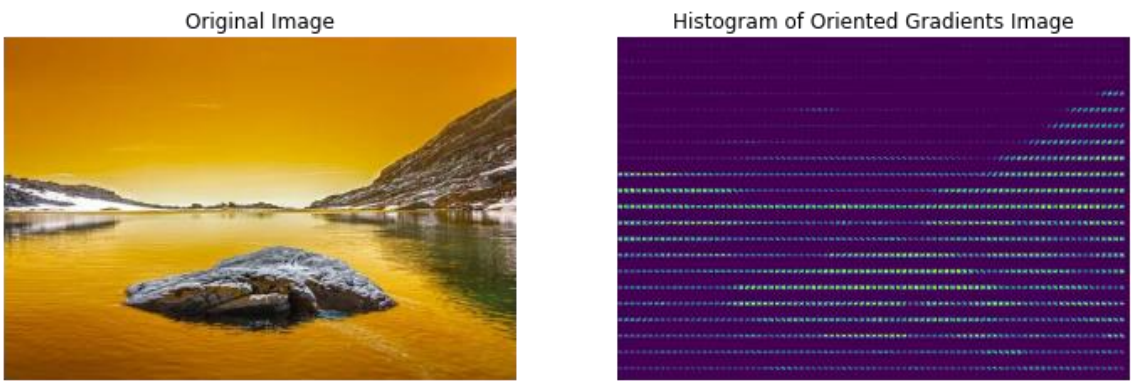
***# Rescaling an Image***

```
rescale_inten = exposure.rescale_intensity(img_hog, in_range = (0,10))
```

***# Displaying an Original Image and HOG features***

```
figure,(a1,a2) = plt.subplots(1,2,figsize = (12,6), sharex = True, sharey =
True)

a1.axis('off')
a1.imshow(img)
a1.set_title('Original Image')

a2.axis('off')
a2.imshow(rescale_inten)
a2.set_title('Histogram of Oriented Gradients Image')

plt.show()
```

**Result :**



**Experiment 10: Implementation of Scale invariant Fourier Transform**

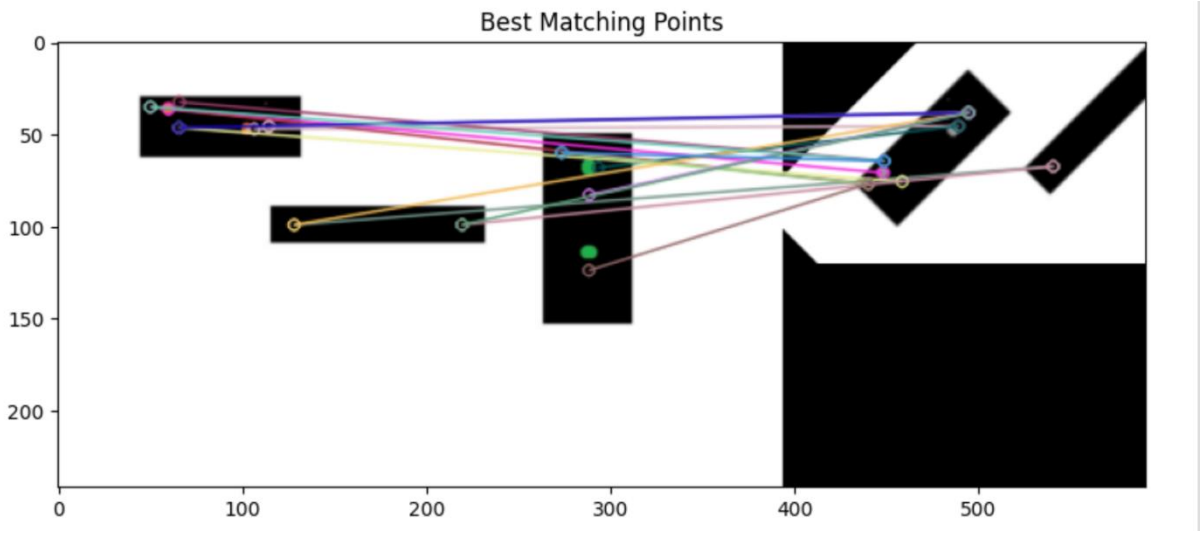
**Aim:**

Perform Feature description using SIFT algorithm

**Pseudocode:**

```
# Import necessary libraries
import cv2

# Load an image from file
image = cv2.imread("image_path.jpg")

# Convert the image to grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Create a SIFT object
sift = cv2.SIFT_create()

# Detect keypoints and compute descriptors
keypoints, descriptors = sift.detectAndCompute(gray_image, None)

# Draw keypoints on the image (optional)
image_with_keypoints = cv2.drawKeypoints(image, keypoints, None)

# Display the image with keypoints (optional)
cv2.imshow("Image with Keypoints", image_with_keypoints)

# Wait for a key press to close the window
cv2.waitKey(0)

# Close OpenCV window
cv2.destroyAllWindows()
```

**Result :**



**Experiment 11: Implementation of Dense Optical Flow model**

**Aim:**

Build a Motion detection using Dense Optical Flow model

**Pseudocode:**

***# Import necessary libraries***

```
import cv2
```

***# Video Capture***

```
cap = cv2.VideoCapture(0)
```

***# Reading & Color Conversion***

```
ret, frame = cap.read()
prv = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
```

***# Zero Matrix***

```
hsv = np.zeros_like(frame)
hsv[...,1] = 255
```

***Dense Optical Flow***

```
while(1):
    ret, fra = cap.read()
    next = cv2.cvtColor(fra,cv2.COLOR_BGR2GRAY)
```

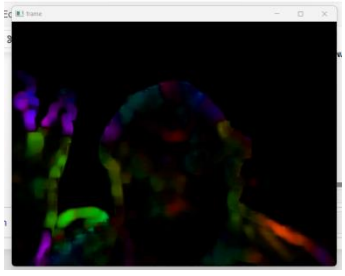
***#cv2.calcOpticalFlowFarneback(prev, next, pyr_scale, levels, winsize, iterations, poly_n, poly_sigma, flags[, flow])***

```
    flow = cv2.calcOpticalFlowFarneback(prv,next, None, 0.5, 3, 15, 3, 5,
1.2, 0)

    mag, ang = cv2.cartToPolar(flow[...,0], flow[...,1])
    hsv[...,0] = ang*180/np.pi/2
    hsv[...,2] = cv2.normalize(mag,None,0,255,cv2.NORM_MINMAX)
    rgb = cv2.cvtColor(hsv,cv2.COLOR_HSV2BGR)

    cv2.imshow('frame',rgb)
    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break

cap.release()
```

**# Close OpenCV window**

```
cv2.destroyAllWindows()
```

**Result :**



**Experiment 12: Face Recognition using Haar Cascade method**

**Aim:**

Build a Face Detection Model using Haar Cascade Method

**Pseudocode:**

```
# Import necessary libraries
import OpenCV

# Load the pre-trained Haar cascade classifier for face detection
face_cascade = cv2.CascadeClassifier("haarcascade_frontalface_default.xml")

# Load an image from file
image = cv2.imread("image_path.jpg")

# Convert the image to grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Perform face detection using the Haar cascade classifier
faces = face_cascade.detectMultiScale(gray_image, scaleFactor=1.1,
minNeighbors=5, minSize=(30, 30))

# Draw rectangles around the detected faces
for (x, y, w, h) in faces:
    cv2.rectangle(image, (x, y), (x+w, y+h), (0, 255, 0), 2)

# Display the image with detected faces
cv2.imshow("Detected Faces", image)

# Wait for a key press to close the window
cv2.waitKey(0)

# Close OpenCV window
cv2.destroyAllWindows()
```

**Result:**

**Experiment 13: Implementation of Object tracking**

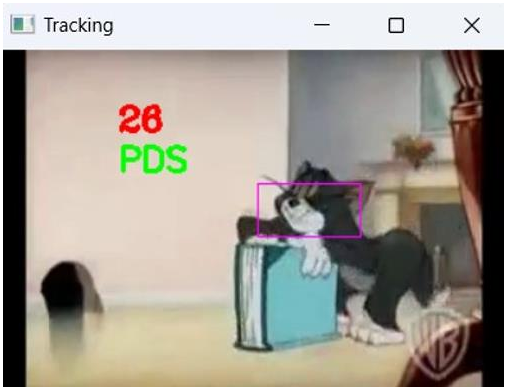
**Aim:**

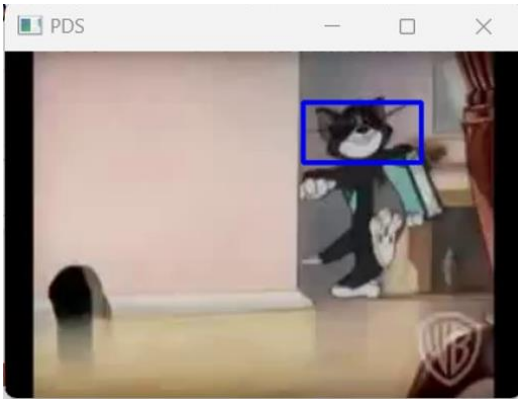
Perform Object tracking using OpenCV

**Pseudocode:**

```
# Import necessary libraries
import cx2

# Video Capture
cap = cv2.VideoCapture(0)

# Define Bounding Box
def drawBox(img,bbox):
    x, y, w, h  = int(bbox[0]),int(bbox[1]),int(bbox[2]),int(bbox[3])
    cv2.rectangle(img,(x,y),((x+w),(y+h)),(255,0,255),3,1)

cv2.putText(img,"Tracking",(75,75),cv2.FONT_HERSHEY_SIMPLEX,0.7,(0,255,0),2
)

#Object tracking

tracker = cv2.TrackerMOSSE_create()
#tracker = cv2.TrackerCSRT_create()
sucess, img = cap.read()
bbox = cv2.selectROI("Tracking",img,False)
tracker.init(img,bbox)

while True:
    timer = cv2.getTickCount()
    sucess, img = cap.read()

    sucess, bbox = tracker.update(img)

    if sucess:
       drawBox(img,bbox)
    else:

cv2.putText(img,"Lost",(75,75),cv2.FONT_HERSHEY_SIMPLEX,0.7,(0,0,255),2)

    fps = cv2.getTickFrequency()/(cv2.getTickCount()-timer)

cv2.putText(img,str(int(fps)),(75,50),cv2.FONT_HERSHEY_COMPLEX,0.7,(0,0,255
),2)
# Display the tracked image
    cv2.imshow("Tracking",img)

    if cv2.waitKey(1) & 0xff == ord('q'):
        break
cap.release()

# Close OpenCV window
cv2.destroyAllWindows()
```

**Result:**



**Experiment 14: Implementation of Template Matching**

**Aim:**

Perform Template Matching using OpenCV

**Pseudocode:**

```
# Import necessary libraries
import cv2

import numpy as np

# Reading in Video
cap = cv2.VideoCapture("Video.mp4")

# Reading in template image
Temp = cv2.imread("Template_Image_Path.png",0)

# Setting template Height
w,h = temp.shape[::-1]

# Template Matching Loop
cv2.waitKey(4)
while (cap.isOpened()):

# Reading in frame by frame
    ret,frame = cap.read()

# Converting to Gray Scale
    img_gray = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)

# Template Matching
    mat = cv2.matchTemplate(img_gray,temp,cv2.TM_CCOEFF_NORMED)

# Setting Threshold
    threshold = 0.5

# Matched Location
    loc = np.where(mat >= threshold)

# Matched Region
    for pt in zip(*loc[::-1]):
        cv2.rectangle(frame,pt,(pt[0] + w, pt[1] + h),(0,255,255),2)

# Displaying Output
    cv2.imshow("Template Matching",frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```

**Result :**

Template Image