SRM Institute of Science and Technology
Delhi – Meerut Road, Sikri Kalan, Ghaziabad, Uttar Pradesh – 201204
Department of Computer Applications
Circular – 2023-24
BCA DS 6th Sem

Introduction to Computer Vision (UDS23601J)

Lab Manual

Lab 1: Read, Displaying, Write Images using OpenCV

Title: Reading, Displaying, and Writing Images with OpenCV

Aim: To familiarize students with basic image handling operations using the OpenCV library.

Procedure:

1. Import the OpenCV library.
2. Read an image from a specified file path using cv2.imread().
3. Display the image using cv2.imshow().
4. Write the image to a new file using cv2.imwrite().
5. Use cv2.waitKey() to manage window display.

Source Code:

```
import cv2

# Read the image
img = cv2.imread('image.jpg')  # Replace 'image.jpg' with your image path

# Display the image
cv2.imshow('Original Image', img)
cv2.waitKey(0)  # Wait until a key is pressed

# Write the image to a new file
cv2.imwrite('output_image.jpg', img)
print("Image saved as output_image.jpg")

cv2.destroyAllWindows() # Close all windows
```

Input: An image file (e.g., image.jpg).

Expected Output:

The original image will be displayed in a window.

A new image file named output_image.jpg will be created in the same directory as the script.

Lab 2: Working Basic Image Operating using OpenCV

Title: Basic Image Operations using OpenCV

Aim: To perform fundamental image manipulations such as accessing pixel values, image slicing, and basic arithmetic operations.

Procedure:

1. Read an image using cv2.imread().
2. Access and modify pixel values.
3. Extract regions of interest (ROIs) using slicing.
4. Perform image arithmetic (addition, subtraction) using cv2.add() and cv2.subtract().

Source Code: (Provide Python code with OpenCV)

Input: One or two image files.

Expected Output: Display of modified images, ROIs, or results of arithmetic operations.

Lab 3: Working with Image Annotation using OpenCV

Title: Image Annotation with OpenCV

Aim: To learn how to annotate images by drawing shapes and text.

Procedure:

1. Read an image using cv2.imread().
2. Draw rectangles, circles, and lines using functions like cv2.rectangle(), cv2.circle(), and cv2.line().
3. Add text to images using cv2.putText().

Source Code: (Provide Python code with OpenCV)

Input: An image file.

Expected Output: Display of the image with added annotations.

Lab 4: Implement different Morphological Operations

Title: Morphological Operations

Aim: To implement and understand morphological operations like erosion, dilation, opening, and closing.

Procedure:

1. Read an image using cv2.imread().
2. Define a kernel (structuring element) using cv2.getStructuringElement().
3. Apply erosion, dilation, opening, and closing using cv2.erode(), cv2.dilate(), cv2.morphologyEx().

Source Code: (Provide Python code with OpenCV)

Input: A grayscale image.

Expected Output: Display of images resulting from each morphological operation.

Lab 5: Working with Contour Analysis

Title: Contour Analysis

Aim: To detect and analyze contours in images.

Procedure:

1. Read an image and convert it to grayscale.
2. Apply edge detection (e.g., Canny) or thresholding.
3. Find contours using cv2.findContours().
4. Draw contours using cv2.drawContours().
5. Calculate contour properties (area, perimeter) using cv2.contourArea() and cv2.arcLength().

Source Code: (Provide Python code with OpenCV)

Input: A grayscale image.

Expected Output: Display of the original image with detected contours highlighted.

Lab 6: Convert the images into different color spaces

Title: Color Space Conversion

Aim: To convert images between different color spaces (e.g., BGR, RGB, HSV, Grayscale).

Procedure:

1. Read an image using cv2.imread().
2. Convert between color spaces using cv2.cvtColor().

Source Code: (Provide Python code with OpenCV)

Input: An image file.

Expected Output: Display of the image in different color spaces.

Lab 7: Implement Canny Edge Detection

Title: Canny Edge Detection

Aim: To implement the Canny edge detection algorithm.

Procedure:

1. Read an image and convert it to grayscale.
2. Apply the Canny edge detection algorithm using cv2.Canny().

Source Code: (Provide Python code with OpenCV)

Input: An image file.

Expected Output: Display of the edges detected in the image.

Lab 8: Face Blending

Title: Face Blending

Aim: To blend two face images.

Procedure:

1. Read two face images.
2. Detect facial keypoints.
3. Perform image alignment.
4. Create a mask.
5. Blend the images.

Source Code: (Provide Python code with OpenCV)

Input: Two face images

Expected Output: Display the blended image

Lab 9: Implement Geometric Transforms in OpenCV

Title: Geometric Transforms

Aim: To apply geometric transformations to images (e.g., translation, rotation, scaling).

Procedure:

1. Read an image using cv2.imread().
2. Define transformation matrices using cv2.getRotationMatrix2D() or manually.
3. Apply transformations using cv2.warpAffine() or cv2.resize().

Source Code: (Provide Python code with OpenCV)

Input: An image file.

Expected Output: Display of the transformed images.

Lab 10: Image segmentation using GrabCut in openCV

Title: Image segmentation using GrabCut

Aim: To segment an image using the GrabCut algorithm.

Procedure:

1. Read an image using cv2.imread().
2. Define a rectangular region of interest (ROI) containing the object to be segmented.
3. Create masks for background and foreground.
4. Apply the GrabCut algorithm using cv2.grabCut().
5. Extract the foreground.

Source Code: (Provide Python code with OpenCV)

Input: An image file.

Expected Output: Display of the segmented foreground.

Lab 11: Motion Detection in OpenCV

Title: Motion Detection

Aim: To detect motion in a video stream.

Procedure:

1. Capture video from a camera or read from a video file.
2. Calculate the difference between successive frames.
3. Apply thresholding to the difference image.
4. Detect contours in the thresholded image.
5. Draw bounding boxes around moving objects.

Source Code: (Provide Python code with OpenCV)

Input: A video stream (camera or video file).

Expected Output: Display of the video with bounding boxes around detected moving objects.

Lab 12: Tracking using MeanShift and CamShift

Title: Object Tracking using MeanShift and CamShift

Aim: To track objects in a video stream using the MeanShift and CamShift algorithms.

Procedure:

1. Capture video from a camera or read from a video file.
2. Select an object to track by defining a region of interest (ROI).
3. Calculate the histogram of the ROI.
4. Initialize the tracking window.
5. Track the object using cv2.meanShift() or cv2.CamShift().

Source Code: (Provide Python code with OpenCV)

Input: A video stream.

Expected Output: Display of the video with a tracking window around the selected object.

Lab 13: Face Detection using OpenCV

Title: Face Detection

Aim: To detect faces in images and video streams.

Procedure:

1. Load a pre-trained Haar cascade classifier using cv2.CascadeClassifier().
2. Read an image or capture video.
3. Detect faces using cascade.detectMultiScale().
4. Draw rectangles around the detected faces.

Source Code: (Provide Python code with OpenCV)

Input: An image or video stream.

Expected Output: Display of the image or video with rectangles around detected faces.

Lab 14: Work with a YOLO/single shot object detection system.

Title: Object Detection with YOLO

Aim: To perform object detection using a pre-trained YOLO model.

Procedure:

1. Download the YOLO model weights and configuration files.
2. Load the YOLO network using cv2.dnn.readNet().
3. Read an image or video frame.
4. Perform forward pass through the network to get detections.
5. Process the output to get bounding boxes, class labels, and confidence scores.
6. Draw bounding boxes and labels on the image.

Source Code: (Provide Python code with OpenCV and optionally torch for loading and running YOLO)

Input: An image or video stream.

Expected Output: Display of the image or video with bounding boxes and labels around detected objects.

Lab 15: Image Classification using OpenCV

Title: Image Classification

Aim: To classify images using pre-trained models with OpenCV

Procedure:

1. Load pre-trained model.
2. Load and preprocess the image.
3. Perform the classification.
4. Display the results

Source Code: (Provide Python code)

Input: An image file

Output: The class of the image