FText

Description automatically generated

**B.Tech FIFTH SEMESTER**

**COMPUTER SCIENCE AND ENGINEERING**

**(CSE & CSE(AI))**

**COMPUTER VISION LABORATORY**

**CSE\_3181**



**LABORATORY MANUAL**

www.manipal.edu/mitblr

Text

Description automatically generated

**Vision**

**Excellence in Emerging Technical Education through Research, Innovation, and Teamwork**

**Mission**

**Educate students professionally to face social challenges by providing a healthy learning environment grounded well in emerging technologies, value based education , creativity, and nurturing teamwork.**

**Goal**

**Our goal is to be a world class technical institution fostering innovation, leadership and entrepreneurial spirit.**

**Program Outcomes:**

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

2. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

**CONTENTS**

|  |  |  |
| --- | --- | --- |
| **S.No** | **TITLE** | **Pg. No** |
|  | COURSE OBJECTIVES AND OUTCOMES | iii |
|  | EVALUATION PLAN | iii |
|  | INSTRUCTIONS TO THE STUDENTS | iv |
|  | INTRODUCTION | v |
| 1 | Basic operations | 1 |
| 2 | Intensity Transformations | 3 |
| 3 | Image Enhancement | 5 |
| 4 | Image Denoising Using Filters | 8 |
| 5 | Edge Detection | 9 |
| 6 | Feature Description Using SIFT | 11 |
| 7 | Geometric Transformations | 13 |
| 8 | Camera Calibration | 15 |
| 9 | Object Tracking | 17 |
| 10 | KLT Feature Tracker | 19 |
| 11 | Face Detection | 21 |
| 12 | Car Detection | 22 |

**Course Objectives**

This laboratory course enables students to

* Implement image enhancement techniques for feature extraction
* Implement different feature detection methods and understand mathematics of description methods.
* Implement methods for different camera models and their calibration.
* Implement object recognition through different learning algorithms.

# Course Outcomes

At the end of this course, students will be able to

1. Apply the concepts of image formation, colour models and linear filtering.

2. Understand the mathematics behind feature detection and description methods.

3. Demonstrate a thorough understanding of fundamental concepts in camera calibration.

4. Implement object tracking algorithms.

5. Design object recognition and categorization from images.

# Evaluation Plan

* Internal Assessment Marks : 60 marks
* Continuous evaluation: 40 Marks
  + The Continuous evaluation assessment will depend on punctuality, designing right algorithm, converting algorithm into an program, maintaining the observation note and answering the questions in viva voce.
  + Internal Exam :20 Marks

**Instructions to the Students**

# Pre- Lab Session Instructions

* Students should carry the Lab Manual Book and the required stationery to every lab session
* Be on time and follow the institution's guidelines
* Must Sign in the log register provided
* Make sure to occupy the allotted seat and answer the attendance
* Adhere to the rules and maintain the decorum.

# In- Lab Session Instructions

* Follow the instructions on the allotted exercises
* Show the program and results to the instructors on completion of experiments
* On receiving approval from the instructor, copy the program and results in the Lab record
* Prescribed textbooks and class notes can be kept ready for reference if required

# General Instructions for the Exercises in Lab

* Implement the given exercise individually and not in a group.
* The programs should meet the following criteria:
  + Programs should be interactive with appropriate prompt messages, error messages if any, and descriptive messages for outputs.
  + Programs should perform input validation (Data type, range error, etc.) and give appropriate error messages and suggest corrective actions.
  + Comments should be used to give the statement of the problem and every function should indicate the purpose of the function, inputs and outputs.
  + Statements within the program should be properly indented.
* Use meaningful names for variables and functions.
* Make use of constants and type definitions wherever needed.
* Plagiarism (copying from others) is strictly prohibited .
* In case a student misses a lab class, he/ she must ensure that the experiment is completed during the repetition lab with the permission of the faculty concerned.
* Questions for lab tests and examination are not necessarily limited to the questions in the manual, but may involve some variations and / or combinations of the questions.

**INTRODUCTION**

This course is designed to provide hands-on experience in computer vision, a field that focuses on developing algorithms and techniques for interpreting visual data from the world around us. The course covers a wide range of topics, starting with an introduction to the OpenCV library, one of the most popular computer vision libraries available today. We will also cover image and video manipulation, including operations at the pixel level, image enhancement techniques, and feature extraction methods. This course focuses also on more advanced topics, including camera models and calibration, stereo vision, tracking, and classification tasks such as face detection and recognition. Throughout the course, students will gain experience with implementing algorithms and techniques using OpenCV and other tools in a Linux environment. By the end of the course, students will have a solid understanding of the theory and practice of computer vision, as well as experience working on real-world applications.

* **About Python and Open-CV libraries**

**Python** is an interperted high-level programming language for general purpose programming. It supports multiple programming paradigms including object oriented and procedural, and has a large and comprehensive standard library.

**PyCharm** is a cross-platform IDE used in computer programming specifically for Python. The platform developed by JetBrains is mainly used in code analysis, graphical debugger etc… It supports web development with Django as well as Data Science with Anaconda.

**OpenCV**  is a library of Python bindings designed to solve computer vision problems. (Open Source Computer Vision Library) is an open-source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in commercial products.  By using it, one can process images and videos to identify objects, faces, or even the handwriting of a human. When it is integrated with various libraries, such as [NumPy](https://www.geeksforgeeks.org/python-numpy/), [Python](https://www.geeksforgeeks.org/python-programming-language/) is capable of processing the [OpenCV](https://www.geeksforgeeks.org/opencv-overview/) array structure for analysis.

* **Requirements:**

**Hardware:**

* A computer with at least 8GB of RAM and a modern processor.
* 32- or a 64-bit computer.
* A graphics card with CUDA support may be beneficial for certain tasks.

**Software:**

* Windows operating system
* OpenCV library installed and configured.
* Python 3.11
* PyCharm IDE
* **Python Installation Procedure**
* Download Python from <https://www.python.org/downloads/>
* Once the download is completed, run the .exe file to install Python.
* To check if python is successfully installed type the following in command prompt

Python --version

* Check if pip is installed successfully in the system

pip -V

### **PyCharm IDE Installation Procedure**

* To download PyCharm visit the website <https://www.jetbrains.com/pycharm/download/>  and click the “DOWNLOAD” link under the Community Section.
* Once the download is complete, run the .exe file for installing PyCharm.

### **Import OpenCV on PyCharm**

* Go to the *Python Packages* option at the bottom of the IDE window.
* Search for “opencv-python” and select the option from PyPI. Click on *Install Package.*
* The package is installed and is visible in the “Installed” Tab.
* To remove the package, click on the package name (here : opencv-python) from the *Installed*. Click three dots, select *“Delete Package”*.

**PROGRAMS**

1. **Basic Operations**
2. **Familiarize the students with the basic functions of OpenCV such as cv2.imread(), cv2.imshow(), and cv2.waitKey() and Write a basic Python program to load and display an image using OpenCV**

* **cv2.imread():** This function is used to read an image from a file into a numpy array, which can then be manipulated using OpenCV. The function takes one argument, which is the path to the image file. It returns a numpy array representing the image.
* **cv2.imshow():** This function is used to display an image in a window on the screen. The function takes two arguments: the name of the window (as a string), and the image data (as a numpy array).
* **cv2.waitKey():** This function waits for a key event from the user. It takes one argument, which is the delay in milliseconds. If the delay is set to 0, the function waits indefinitely for a key event.

**Procedure:**

**Step 1:** Load the image from the file

**Step 2:** Display the image in a window

**Step 3:** Wait for a key press and then close the window

**Program:**

import cv2

# Load the image from the file

img = cv2.imread('image.jpg')

# Display the image in a window

cv2.imshow('Image', img)

# Wait for a key press and then close the window

cv2.waitKey(0)

cv2.destroyAllWindows()

1. **Write a program to read a video file and display it on the screen**

**Procedure:**

**Step 1:** Import the necessary libraries: cv2 for OpenCV and numpy for numerical operations.

**Step 2:** Define the video file path using cv2.VideoCapture() function.

**Step 3:** Check if the video file is opened correctly by calling cap.isOpened() function.

**Step 4:** Loop through each frame of the video using while loop.

**Step 5:** Read the frame using cap.read() function.

**Step 6:** Check if the frame is empty using if statement.

**Step 7:** Display the frame on the screen using cv2.imshow() function.

**Step 8:** Wait for a key event to occur using cv2.waitKey() function.

**Step 9:** Check if the key pressed is 'q' to quit the video using if statement.

**Step 10:** Release the video capture object using cap.release() function.

**Step 11:** Close all windows using cv2.destroyAllWindows() function.

**Program**

import cv2

# open the video file

cap = cv2.VideoCapture('video.mp4')

# loop through frames in the video

while cap.isOpened():

# read a frame

ret, frame = cap.read()

# check if the frame was successfully read

if ret:

# display the frame on the screen

cv2.imshow('Video', frame)

# wait for a key press for 25 milliseconds

if cv2.waitKey(25) & 0xFF == ord('q'):

break

else:

break

# release the video capture object and close all windows

cap.release()

cv2.destroyAllWindows()

**2. Intensity Transformations**

**a) Write a program to perform Image Negation**

**Procedure:**

**Step 1:** Import the necessary OpenCV libraries.

**Step 2:** Load the input image using the cv2.imread() function.

**Step 3:** Convert the input image to grayscale using cv2.cvtColor() function.

**Step 4:** Compute the negative of the grayscale image using the cv2.bitwise\_not() function.

**Step 5:** Display the input image and the negative image using the cv2.imshow() function.

**Step 6:** Wait for a key press using the cv2.waitKey() function.

**Step 7:** If the key pressed is 's', save the negative image using the cv2.imwrite() function.

**Step 8:** Destroy all windows using the cv2.destroyAllWindows() function.

**Program:**

import cv2

# Load input image

img = cv2.imread('input\_image.jpg')

# Convert input image to grayscale

gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

# Compute negative of the grayscale image

neg = cv2.bitwise\_not(gray)

# Display the input image and negative image

cv2.imshow('Input Image', img)

cv2.imshow('Negative Image', neg)

# Wait for a key press

key = cv2.waitKey(0)

# If the key pressed is 's', save the negative image

if key == ord('s'):

cv2.imwrite('negative\_image.jpg', neg)

# Destroy all windows

cv2.destroyAllWindows()

**b) Write a program to perform Log transformation on the image**

**Procedure:**

**Step 1:** Import the necessary OpenCV modules and the input image.

**Step 2:** Define the gamma value and the constant C for log transformation.

**Step 3:** Convert the input image to grayscale if it is a color image.

**Step 4:** Perform the log transformation using the cv2.log() function and the given formula:

output\_image = C \* log(1 + input\_image)

**Step 6:** Display the input and output images side by side using cv2.hconcat() and cv2.imshow() functions.

**Step 7:** Save the output image using the cv2.imwrite() function.

**Program:**

import cv2

import numpy as np

# Read input image

img = cv2.imread('input\_image.jpg')

# Define gamma value for gamma correction

gamma = 1.5

# Define constant C for log transformation

C = 20

# Convert input image to grayscale

gray\_img = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

# Perform log transformation

log\_img = C \* np.log(1 + gray\_img)

# Display input and output images side by side

result = cv2.hconcat([gray\_img, log\_img])

cv2.imshow(Log transformation', result)

# Save the output image

cv2.imwrite('output\_image.jpg', result)

cv2.waitKey(0)

cv2.destroyAllWindows()

1. **Image Enhancement**
2. **Write a program to enhance the image using piece-wise linear transformation by gray-level slicing.**

**Procedure:**

**Step1:** Read input image

**Step2:** Find width and height of the image

**Step3:** Create an zeros array to store the sliced image

**Step4:** Specify the min and max range

**Step5:** Loop over the input image and if pixel value lies in desired range set it to 255

**Step6:** otherwise set it to desired value

**Step7:** Wait for a key press using the cv2.waitKey() function.

**Step8:** Write the original and sliced images using imwrite( ) function

**Step9:** Destroy all windows using the cv2.destroyAllWindows() function.

**Program:**

Import cv2

img = cv2.imread('background.jpg', 0)

# Find width and height of image

row, column= img.shape

# Create an zeros array to store the sliced image

img1 = np.zeros((row,column),dtype = 'uint8')

# Specify the min and max range

min\_range = 80

max\_range = 140

# Loop over the input image and if pixel value lies in desired range set it to 255

# otherwise set it to desired value

for i in range(row):

for j in range(column):

if img[i,j]>min\_range and img[i,j]<max\_range:

img1[i,j] = 255

else:

img1[i,j] = img[i-1,j-1]

cv2.imwrite('Original.jpg', img)

cv2.imwrite('slicedimage.jpg', img1)

cv2.waitKey(0)

cv2.destroyAllWindows()

**b) Write a program to enhance the contrast of the image using Histogram Equalization**

**Procedure:**

**Step1:** Import necessary libraries

**Step2:** Read the input image

**Step3:** Set the grid size and Plot the original image

**Step4:** Compute the histogram of the image

**Step5:** Plot and display the histogram of the image

**Step6:** Applying the Histogram equalization using the cv2.equalizeHist() function

**Step7:** Displaying the histogram equalized image

**Program:**

# Importing OpenCV

import cv2

# Importing numpy

import numpy as np

# Importing matplotlib.pyplot

import matplotlib.pyplot as plt

# Reading the image

img = cv2.imread(r"C:\Users\tushi\Downloads\PythonGeeks\deer.png")

img = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)

img\_1 = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

# Setting the grid size

plt.figure(figsize=(20,20))

# Plotting the original image

plt.subplot(221)

plt.title('Original')

plt.imshow(img)

# Plotting the histogram for the image

img\_hist = cv2.calcHist([img\_1],[0],None,[256],[0,256])

plt.subplot(222)

plt.title('Histogram 1')

plt.plot(img\_hist)

# Plotting the histogram using the ravel function

plt.subplot(223)

plt.hist(img\_1.ravel(), 256, [0,256])

plt.title('Histogram 2')

# Applying the Histogram equalization using the cv2.equalizeHist() function

final\_image = cv2.equalizeHist(img)

# Displaying the image

cv2.imshow('Histogram Equalization', final\_img)

cv2.waitKey(0)

cv2.destroyAllWindows()

**4. Image Denoising Using Filters**

**Implement the following low-pass and high-pass filters to improve the quality of the image: a) mean filter b) median filter c) Laplacian filter**

**Procedure:**

**Step 1:** Load the image using cv2.imread() function

**Step 2:** Apply spatial filtering using filter function and kernel matrix

**Step 3:** Display the filtered image using cv2.imshow() function

**Step 4:** Wait for user input using cv2.waitKey() function

**Step 5:** Destroy all windows using cv2.destroyAllWindows() function

**Program:**

import cv2

import numpy as np

# Load image

img = cv2.imread('image.jpg', cv2.IMREAD\_GRAYSCALE)

# Define kernel matrix

kernel = np.ones((5, 5), np.float32) / 25

# Apply mean filtering

mean\_filtered = cv2.filter2D(img, -1, kernel)

# Apply median filtering

median\_filtered = cv2.medianBlur(img, 5)

#Sharpen the image using the Laplacian operator

sharpened\_image2 **=** cv2.Laplacian(image, cv2.CV\_64F)

# Display filtered images

cv2.imshow('Original Image', img)

cv2.imshow('Mean Filtered Image', mean\_filtered)

cv2.imshow('Median Filtered Image', median\_filtered)

# Wait for user input

cv2.waitKey(0)

# Destroy all windows

cv2.destroyAllWindows()

**5. Edge Detection**

**Implement edge detection using Sobel and Canny edge detectors using OpenCV**

**Procedure:**

**Step 1:** Read the input image using OpenCV’s imread() function.

**Step 2:** Convert the image to grayscale using cvtColor() function.

**Step 3:** Apply Gaussian blur on the grayscale image using GaussianBlur() function to remove noise.

**Step 4:** Apply Sobel operator on the blurred image using Sobel() function to detect edges in x and y directions.

**Step 5:** Combine the edges detected in x and y directions to obtain the final edges using bitwise\_or() function.

**Step 6:** Apply non-maximum suppression on the final edges using Canny() function to thin out the edges.

**Step 7:** Display the original image and the final edge-detected image using imshow() function.

**Step 8:** Save the final edge-detected image using imwrite() function.

**Program:**

import cv2

# read input image

img = cv2.imread('input.jpg')

# convert image to grayscale

gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

# apply Gaussian blur to remove noise

blur = cv2.GaussianBlur(gray, (5,5), 0)

# apply Sobel operator to detect edges in x and y directions

sobelx = cv2.Sobel(blur, cv2.CV\_64F, 1, 0, ksize=3)

sobely = cv2.Sobel(blur, cv2.CV\_64F, 0, 1, ksize=3)

# combine edges detected in x and y directions

edges = cv2.bitwise\_or(sobelx, sobely)

# apply non-maximum suppression to thin out edges

edges = cv2.Canny(edges, 100, 200)

# display original image and edge-detected image

cv2.imshow('Original Image', img)

cv2.imshow('Edge-Detected Image', edges)

# save edge-detected image

cv2.imwrite('output.jpg', edges)

# wait for user to close the window

cv2.waitKey(0)

# close all windows

cv2.destroyAllWindows()

**6. Feature Description Using SIFT**

**Write a program to Match two images based on features extracted by SIFT algorithm**

**Procedure:**

Step 1: Import the necessary libraries and modules of OpenCV and numpy.

Step 2: Read two images using imread() function

Step 3: Convert the image to grayscale using cv2.cvtColor() method

Step 4: Create SIFT feature extractor object using cv2.xfeatures2d.SIFT\_create ( ).

Step 5: Pass the image to detectandCompute() to detect the keypoints and descriptors.

Step 6: match the descriptors

Step 7: sort the matches by distance and draw the first 50 matches

Step 8: Display the image using imshow()

Step 9: Wait for user input to close the window using cv2.waitKey() method.

Step 10: Release the resources using cv2.destroyAllWindows() method.

**Program:**

import cv2

# read the images

img1 = cv2.imread('book.jpg')

img2 = cv2.imread('table.jpg')

# convert images to grayscale

img1 = cv2.cvtColor(img1, cv2.COLOR\_BGR2GRAY)

img2 = cv2.cvtColor(img2, cv2.COLOR\_BGR2GRAY)

# create SIFT object

sift = cv2.xfeatures2d.SIFT\_create()

# detect SIFT features in both images

keypoints\_1, descriptors\_1 = sift.detectAndCompute(img1,None)

keypoints\_2, descriptors\_2 = sift.detectAndCompute(img2,None)

# create feature matcher

bf = cv2.BFMatcher(cv2.NORM\_L1, crossCheck=True)

# match descriptors of both images

matches = bf.match(descriptors\_1,descriptors\_2)

# sort matches by distance

matches = sorted(matches, key = lambda x:x.distance)

# draw first 50 matches

matched\_img = cv2.drawMatches(img1, keypoints\_1, img2, keypoints\_2, matches[:50], img2, flags=2)

# show the image

cv2.imshow('image', matched\_img)

# save the image

cv2.imwrite("matched\_images.jpg", matched\_img)

cv2.waitKey(0)

cv2.destroyAllWindows()

**7.** **Geometric Transformations**

**Implement the following geometric transformations on the image using OpenCV: a)Rotation b)Scaling c) Translation d)Shear**

**Procedure:**

**Step 1:** Read the input image using cv2.imread() function.

**Step 2:** Define the transformation matrix M to perform the desired transformation using cv2.getRotationMatrix2D(), cv2.getAffineTransform(), or cv2.getPerspectiveTransform() functions depending on the type of transformation required.

**Step 3:** Apply the transformation to the image using cv2.warpAffine() or cv2.warpPerspective() function depending on the type of transformation defined in step 2.

**Step 4:** Display the transformed image using cv2.imshow() function.

**Step 5:** Wait for a key event using cv2.waitKey() function.

**Step 6:** Release the memory and destroy all windows using cv2.destroyAllWindows() function.

**Program:**

**# program to rotate the image**

import cv2

# Read the input image

img = cv2.imread('input.jpg')

# Define the rotation angle

angle = 45

# Get the image dimensions

h, w = img.shape[:2]

# Calculate the rotation matrix M

M = cv2.getRotationMatrix2D((w/2, h/2), angle, 1)

# Apply the rotation to the image

rotated\_img = cv2.warpAffine(img, M, (w, h))

# Display the original and rotated images

cv2.imshow('Original Image', img)

cv2.imshow('Rotated Image', rotated\_img)

# Wait for a key event

cv2.waitKey(0)

# Release the memory and destroy all windows

cv2.destroyAllWindows()

#Program to scale an image:

**import** numpy as np

**import** cv2 as cv

img **=** cv.imread('girlImage.jpg', 0)

rows, cols **=** img.shape

img\_shrinked **=** cv.resize(img, (250, 200),

                         interpolation**=**cv.INTER\_AREA)

cv.imshow('img', img\_shrinked)

img\_enlarged **=** cv.resize(img\_shrinked, None,

                         fx**=**1.5, fy**=**1.5,

                         interpolation**=**cv.INTER\_CUBIC)

cv.imshow('img', img\_enlarged)

cv.waitKey(0)

cv.destroyAllWindows()

**#program to translate an image**

**import** numpy as np

**import** cv2 as cv

img **=** cv.imread('girlImage.jpg', 0)

rows, cols **=** img.shape

M **=** np.float32([[1, 0, 100], [0, 1, 50]])

dst **=** cv.warpAffine(img, M, (cols, rows))

cv.imshow('img', dst)

cv.waitKey(0)

cv.destroyAllWindows()

**#program to shear an image**

import numpy as np

import cv2 as cv

img = cv.imread('girlImage.jpg', 0)

rows, cols = img.shape

M = np.float32([[1, 0.5, 0], [0, 1, 0], [0, 0, 1]])

sheared\_img = cv.warpPerspective(img, M, (int(cols\*1.5), int(rows\*1.5)))

cv.imshow('img', sheared\_img)

cv.waitKey(0)

cv.destroyAllWindows()

**8. Camera Calibration**

**Write a program to implement camera calibration using OpenCV**

**Procedure:**

**Step 1:** Capture several images of a calibration pattern (e.g. a chessboard) from different angles with the camera you want to calibrate.

**Step 2:** Extract the corners of the calibration pattern in each image using OpenCV's findChessboardCorners() function.

**Step 3:** Create a list of object points, which are the 3D coordinates of the calibration pattern corners in a coordinate system relative to the camera. For example, you can define the first corner as (0,0,0), and then assume that the other corners are on a regular grid of known size.

**Step 4:** For each image, use OpenCV's cornerSubPix() function to refine the corner coordinates.

**Step 5:** Use OpenCV's calibrateCamera() function to obtain the camera matrix, distortion coefficients, and rotation and translation vectors for each image.

**Step 6:** Compute the reprojection error, which measures the average distance between the projected 3D calibration pattern corners and the corresponding 2D image points.

**Step 7:** Use the obtained calibration parameters to undistort images captured with the calibrated camera using OpenCV's undistort() function.

**Program:**

import numpy as np

import cv2

# Define the size of the calibration pattern

pattern\_size = (7, 6) # number of interior corners on each row and column

# Define the coordinates of the calibration pattern corners in the calibration pattern coordinate system

objp = np.zeros((pattern\_size[0]\*pattern\_size[1], 3), np.float32)

objp[:,:2] = np.mgrid[0:pattern\_size[0],0:pattern\_size[1]].T.reshape(-1,2) \* 30 # assume square size of 30mm

# Capture several images of the calibration pattern from different angles

images = [] # list of images

while True:

ret, img = cap.read() # capture an image from the camera

if not ret:

break

gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

ret, corners = cv2.findChessboardCorners(gray, pattern\_size, None)

if ret :

cv2.cornerSubPix(gray, corners, (11, 11), (-1, -1), (cv2.TERM\_CRITERIA\_EPS + cv2.TERM\_CRITERIA\_MAX\_ITER, 30, 0.001))

images.append((img, corners))

# Compute the camera calibration parameters

objpoints = [] # list of object points for each image

imgpoints = [] # list of image points for each image

for img, corners in images:

objpoints.append(objp)

imgpoints.append(corners)

ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints, gray.shape[::-1], None, None)

# Compute the reprojection error

mean\_error = 0

for i in range(len(objpoints)):

imgpoints2, \_ = cv2.projectPoints(objpoints[i], rvecs[i], tvecs[i], mtx, dist)

error = cv2.norm(imgpoints[i],imgpoints2, cv2.NORM\_L2)/len(imgpoints2)

mean\_error += error

print("Mean reprojection error: ", mean\_error/len(objpoints))

# Use the obtained calibration parameters to undistort an image

img = cv2.imread("test.jpg")

undistorted = cv2.undistort(img, mtx, dist)

cv2.imshow("Undistorted image", undistorted)

cv2.waitKey(0)

**9. Tracking of Moving Objects**

**Implement optical flow and track moving objects in a video using OpenCV**

**Procedure:**

**Step 1:** Load the input video and read the first frame.

**Step 2:** Convert the first frame to grayscale and use it as the reference image.

**Step 3:** Loop over the remaining frames in the video:

a. Convert the current frame to grayscale.

b. Compute the optical flow using a method such as Lucas-Kanade or Farneback.

c. Compute the magnitude and direction of the optical flow vectors.

d. Threshold the magnitude to remove noise and small motions.

e. Compute the contours of the thresholded motion map.

f. Filter the contours by area and aspect ratio to remove noise and false detections.

g. Draw bounding boxes around the remaining contours and track them using a simple Kalman filter or a more sophisticated tracker such as the SORT algorithm.

h. Update the reference image to the current frame.

i. Display the output video with the bounding boxes around the moving objects.

**Program:**

import cv2

import numpy as np

# Load the input video

cap = cv2.VideoCapture('input\_video.mp4')

# Read the first frame and convert it to grayscale

ret, frame = cap.read()

gray\_ref = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

# Define the motion threshold and contour area threshold

motion\_thresh = 50

min\_contour\_area = 1000

max\_contour\_aspect\_ratio = 5

# Initialize the tracker

tracker = cv2.MultiTracker\_create()

# Loop over the frames in the video

while True:

# Read the current frame and convert it to grayscale

ret, frame = cap.read()

if not ret:

break

gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

# Compute the optical flow using Farneback method

flow = cv2.calcOpticalFlowFarneback(gray\_ref, gray, None, 0.5, 3, 15, 3, 5, 1.2, 0)

# Compute the magnitude and direction of the optical flow vectors

mag, ang = cv2.cartToPolar(flow[..., 0], flow[..., 1])

# Threshold the magnitude to remove noise and small motions

motion\_mask = (mag > motion\_thresh).astype(np.uint8)

# Compute the contours of the thresholded motion map

contours, \_ = cv2.findContours(motion\_mask, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

# Filter the contours by area and aspect ratio to remove noise and false detections

detections = []

for contour in contours:

area = cv2.contourArea(contour)

bbox = cv2.boundingRect(contour)

aspect\_ratio = bbox[2] / bbox[3] if bbox[3] != 0 else 0

if area > min\_contour\_area and aspect\_ratio < max\_contour\_aspect\_ratio:

detections.append(bbox)

# Update the tracker with the current detections

tracker.update(frame, detections)

# Display the output video with the bounding boxes around the moving objects

for i, bbox in enumerate(tracker.getObjects()):

x, y, w, h = [int(v) for v in bbox]

cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)

cv2.imshow('Output', frame)

if cv2.waitKey(1) == ord('q'):

break

# Update the reference image to the current frame

**10. KLT Feature Tracker**

**Implement the Kanade-Lucas-Tomasi (KLT) feature tracker algorithm**

**Step 1:** Load the input video and read the first frame.

**Step 2:** Select a set of feature points to track using a feature detector such as Shi-Tomasi or Harris corner detector.

**Step 3:** Compute the optical flow for each feature point using the Lucas-Kanade algorithm.

**Step 4:** Update the feature points based on the computed optical flow vectors.

**Step 5:** Loop over the remaining frames in the video:

a. Track the feature points using the computed optical flow vectors.

b. Compute the optical flow for each feature point using the Lucas-Kanade algorithm.

c. Update the feature points based on the computed optical flow vectors.

d. Draw the tracked feature points on the output video.

**Program:**

import cv2

import numpy as np

# Kanade-Lucas-Tomasi (KLT) Feature Tracker

cap = cv2.VideoCapture('input\_video.mp4')

# Parameters for Shi-Tomasi corner detection

feature\_params = dict(maxCorners=100, qualityLevel=0.3, minDistance=7, blockSize=7)

# Parameters for Lucas-Kanade optical flow

lk\_params = dict(winSize=(15, 15), maxLevel=2, criteria=(cv2.TERM\_CRITERIA\_EPS | cv2.TERM\_CRITERIA\_COUNT, 10, 0.03))

# Take the first frame and detect corners in it

ret, old\_frame = cap.read()

old\_gray = cv2.cvtColor(old\_frame, cv2.COLOR\_BGR2GRAY)

p0 = cv2.goodFeaturesToTrack(old\_gray, mask=None, \*\*feature\_params)

# Create a mask image for drawing purposes

mask = np.zeros\_like(old\_frame)

while True:

ret, frame = cap.read()

if not ret:

break

frame\_gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

# Calculate optical flow using Lucas-Kanade algorithm

p1, st, err = cv2.calcOpticalFlowPyrLK(old\_gray, frame\_gray, p0, None, \*\*lk\_params)

# Select good points

good\_new = p1[st == 1]

good\_old = p0[st == 1]

# Draw the tracks

for i, (new, old) in enumerate(zip(good\_new, good\_old)):

a, b = new.ravel()

c, d = old.ravel()

mask = cv2.line(mask, (a, b), (c, d), (0, 255, 0), 2)

frame = cv2.circle(frame, (a, b), 5, (0, 255, 0), -1)

img = cv2.add(frame, mask)

cv2.imshow('frame', img)

if cv2.waitKey(1) == ord('q'):

break

# Update previous frame and points

old\_gray = frame\_gray.copy()

p0 = good\_new.reshape(-1, 1, 2)

cv2.destroyAllWindows()

cap.release()

**11. Face Detection in Image**

**Write a program to detect face of a person in a Image**

**Procedure:**

**Step 1:** Load the pre-trained face detection model using cv2.CascadeClassifier()

**Step 2:** Read the input image or stream from a camera

**Step 3:** Convert the input image to grayscale

**Step 4:** Detect the faces in the image using cv2.CascadeClassifier.detectMultiScale()

**Step 5:** Draw a rectangle around each detected face

**Step 6:** Display the output image with detected faces

**Step 7:** Wait for user input to exit or continue processing

**Program:**

import cv2

# Load the pre-trained face detection model

face\_cascade = cv2.CascadeClassifier('haarcascade\_frontalface\_default.xml')

# Read the input image or stream from a camera

cap = cv2.VideoCapture(0) # Use default camera

while True:

ret, frame = cap.read()

if not ret:

break

# Convert the input image to grayscale

gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

# Detect the faces in the image

faces = face\_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5)

# Draw a rectangle around each detected face

for (x, y, w, h) in faces:

cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)

# Display the output image with detected faces

cv2.imshow('Face Detection', frame)

# Wait for user input to exit or continue processing

if cv2.waitKey(1) == ord('q'):

break

# Release the resources and close the windows

cap.release()

cv2.destroyAllWindows()

**12. Car Detection in a Video**

**Write a program to detect a car in the given video using OpenCV**

**Procedure:**

Step1: Import required libraries

Step2: Read frames from input video

Step3: Convert frames to grayscale

Step4: Detect cars of different sizes in the input

Step5: Draw rectangle around detected car

Step6: Display video frames in the window

Step7: Loop through all frames for detection

Step8: Destroy all windows

**Program:**

# import libraries of python OpenCV

**import** cv2

haar\_cascade = 'cars.xml'

video = 'video.avi'

cap = cv2.VideoCapture(video)

car\_cascade = cv2.CascadeClassifier(haar\_cascade)

# reads frames from a video

ret, frames = cap.read()

# convert frames to gray scale

gray = cv2.cvtColor(frames, cv2.COLOR\_BGR2GRAY)

# Detects cars of different sizes in the input image

cars = car\_cascade.detectMultiScale(gray, 1.1, 1)

# To draw a rectangle in each cars

for (x,y,w,h) in cars:

cv2.rectangle(frames,(x,y),(x+w,y+h),(0,0,255),2)

# Display frames in a window

cv2.imshow('video', frames)

# loop runs if capturing has been initialized.

while True:

# reads frames from a video

ret, frames = cap.read()

# convert to gray scale of each frames

gray = cv2.cvtColor(frames, cv2.COLOR\_BGR2GRAY)

# Detects cars of different sizes in the input image

cars = car\_cascade.detectMultiScale(gray, 1.1, 1)

# To draw a rectangle in each cars

for (x,y,w,h) in cars:

cv2.rectangle(frames,(x,y),(x+w,y+h),(0,0,255),2)

# Display frames in a window

cv2.imshow('video', frames)

# Wait for Esc key to stop

if cv2.waitKey(33) == 27:

break

# De-allocate any associated memory usage

cv2.destroyAllWindows()