Computer Vision Lab Experiments - Code Compilation

# Experiment 1: Basic Image Processing Operations

import cv2  
import numpy as np  
import matplotlib.pyplot as plt  
  
# Read image  
image = cv2.imread('input.jpg')  
cv2.imshow('Original Image', image)  
cv2.waitKey(0)  
cv2.destroyAllWindows()  
  
# Convert and save to another format  
cv2.imwrite('converted.png', image)  
  
# Convert to Grayscale  
gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)  
cv2.imshow('Grayscale', gray)  
cv2.waitKey(0)  
cv2.destroyAllWindows()  
  
# Compute Histogram  
hist = cv2.calcHist([gray], [0], None, [256], [0, 256])  
plt.plot(hist)  
plt.title('Histogram')  
plt.xlabel('Pixel value')  
plt.ylabel('Frequency')  
plt.show()  
  
# Histogram Equalization  
cdf = hist.cumsum()  
cdf\_normalized = cdf \* hist.max() / cdf.max()  
cdf\_m = np.ma.masked\_equal(cdf, 0)  
cdf\_m = (cdf\_m - cdf\_m.min()) \* 255 / (cdf\_m.max() - cdf\_m.min())  
cdf = np.ma.filled(cdf\_m, 0).astype('uint8')  
equalized = cdf[gray]  
cv2.imshow('Equalized Image', equalized)  
cv2.waitKey(0)  
cv2.destroyAllWindows()

# Experiment 2: Geometric Transformations (Translation, Rotation, Scaling)

import cv2  
import numpy as np  
  
# Load image  
image = cv2.imread('input.jpg')  
  
# Translation  
rows, cols = image.shape[:2]  
M\_translate = np.float32([[1, 0, 50], [0, 1, 100]])  
translated = cv2.warpAffine(image, M\_translate, (cols, rows))  
  
# Rotation  
M\_rotate = cv2.getRotationMatrix2D((cols/2, rows/2), 45, 1)  
rotated = cv2.warpAffine(image, M\_rotate, (cols, rows))  
  
# Scaling  
resized = cv2.resize(image, None, fx=0.5, fy=0.5, interpolation=cv2.INTER\_LINEAR)  
  
cv2.imshow('Translated', translated)  
cv2.imshow('Rotated', rotated)  
cv2.imshow('Scaled', resized)  
cv2.waitKey(0)  
cv2.destroyAllWindows()

# Experiment 3: Edge Detection using Canny

# import cv2

# import numpy as np

# import matplotlib.pyplot as plt

# # Step 1: Read the image

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

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

# # Step 2: Apply Gaussian Blur (common preprocessing for both methods)

# blurred = cv2.GaussianBlur(gray, (5, 5), 1.4)

# # Step 3: Laplacian Edge Detection

# laplacian = cv2.Laplacian(blurred, cv2.CV\_64F)

# laplacian = cv2.convertScaleAbs(laplacian)

# # Step 4: Canny Edge Detection

# canny\_edges = cv2.Canny(blurred, threshold1=100, threshold2=200)

# # Step 5: Display all results

# cv2.imshow('Original Grayscale', gray)

# cv2.imshow('Gaussian Blurred', blurred)

# cv2.imshow('Laplacian Edge Detection', laplacian)

# cv2.imshow('Canny Edge Detection', canny\_edges)

# cv2.waitKey(0)

# cv2.destroyAllWindows()

# # Step 6: Plot using matplotlib (optional)

# plt.figure(figsize=(12, 6))

# plt.subplot(1, 3, 1)

# plt.title("Laplacian")

# plt.imshow(laplacian, cmap='gray')

# plt.axis('off')

# plt.subplot(1, 3, 2)

# plt.title("Canny")

# plt.imshow(canny\_edges, cmap='gray')

# plt.axis('off')

# plt.subplot(1, 3, 3)

# plt.title("Blurred Input")

# plt.imshow(blurred, cmap='gray')

# plt.axis('off')

# plt.tight\_layout()

# plt.show()

# Experiment 4: Image Segmentation using K-Means

import cv2  
import numpy as np  
  
# Read image  
image = cv2.imread('input.jpg')  
Z = image.reshape((-1, 3))  
Z = np.float32(Z)  
  
# Define criteria and apply KMeans  
criteria = (cv2.TERM\_CRITERIA\_EPS + cv2.TERM\_CRITERIA\_MAX\_ITER, 10, 1.0)  
K = 5 # Try 20 also  
\_, labels, centers = cv2.kmeans(Z, K, None, criteria, 10, cv2.KMEANS\_RANDOM\_CENTERS)  
  
# Convert back and reshape  
centers = np.uint8(centers)  
res = centers[labels.flatten()]  
result\_image = res.reshape((image.shape))  
  
cv2.imshow(f'Segmented with K={K}', result\_image)  
cv2.waitKey(0)  
cv2.destroyAllWindows()

# Experiment 5: Harris Corner Detection

import cv2  
import numpy as np  
  
# Read image and convert to grayscale  
image = cv2.imread('input.jpg')  
gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)  
gray = np.float32(gray)  
  
# Harris corner detection  
dst = cv2.cornerHarris(gray, 2, 3, 0.04)  
dst = cv2.dilate(dst, None)  
image[dst > 0.01 \* dst.max()] = [0, 0, 255]  
  
cv2.imshow('Harris Corners', image)  
cv2.waitKey(0)  
cv2.destroyAllWindows()

# Experiment 6: Object Recognition using HOG + ML

import cv2  
from skimage.feature import hog  
from sklearn.svm import LinearSVC  
import joblib  
  
# Load and preprocess image  
image = cv2.imread('input.jpg', cv2.IMREAD\_GRAYSCALE)  
features, hog\_image = hog(image, pixels\_per\_cell=(8, 8), cells\_per\_block=(2, 2), visualize=True)  
  
# Dummy classifier for demonstration  
clf = LinearSVC()  
# clf.fit(X\_train, y\_train) # Train with actual dataset  
# prediction = clf.predict([features])

# Experiment 7: Camera Calibration Using OpenCV

import cv2  
import numpy as np  
import glob  
import yaml  
  
# Prepare object points  
objp = np.zeros((7\*7, 3), np.float32)  
objp[:, :2] = np.mgrid[0:7, 0:7].T.reshape(-1, 2)  
objpoints = []  
imgpoints = []  
  
images = glob.glob('chessboard\*.jpg')  
  
for fname in images:  
 img = cv2.imread(fname)  
 gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)  
 ret, corners = cv2.findChessboardCorners(gray, (7, 7), None)  
 if ret:  
 objpoints.append(objp)  
 imgpoints.append(corners)  
 cv2.drawChessboardCorners(img, (7, 7), corners, ret)  
 cv2.imshow('Corners', img)  
 cv2.waitKey(500)  
  
cv2.destroyAllWindows()  
  
# Calibration  
ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints, gray.shape[::-1], None, None)  
  
# Save calibration  
data = {'camera\_matrix': mtx.tolist(), 'dist\_coeff': dist.tolist()}  
with open("calibration.yaml", "w") as f:  
 yaml.dump(data, f)