Practical 1 Downsample and up sample

import cv2

import matplotlib.pyplot as plt

# Read image in grayscale

img = cv2.imread('11.jpg', cv2.IMREAD\_GRAYSCALE)

# Downsample and upsample the image

down\_img = cv2.resize(img, None, fx=0.5, fy=0.5, interpolation=cv2.INTER\_AREA)

up\_img = cv2.resize(down\_img, (img.shape[1], img.shape[0]), interpolation=cv2.INTER\_LINEAR)

# Display images

titles = ["Original", "Downsampled", "Upsampled"]

images = [img, down\_img, up\_img]

plt.figure(figsize=(12, 4))

for i in range(3):

plt.subplot(1, 3, i + 1)

plt.title(titles[i])

plt.imshow(images[i], cmap='gray')

plt.axis('off')

plt.tight\_layout()

plt.show()

Practical 2 Triangle signal

import numpy as np

import matplotlib.pyplot as plt

from scipy.signal import correlate, sawtooth

# Generate triangle signal

t = np.linspace(0, 3, 300, endpoint=False)

triangle\_signal = sawtooth(2 \* np.pi \* t, width=0.5)

# Plot the full triangle signal

plt.figure(figsize=(10, 4))

plt.plot(t, triangle\_signal, label="Triangle Signal")

plt.title("Triangle Signal")

plt.xlabel("Time")

plt.ylabel("Amplitude")

plt.grid()

plt.legend()

plt.show()

# Take a segment of the signal

segment\_length = len(triangle\_signal) // 3

signal\_segment = triangle\_signal[:segment\_length]

# Plot signal segment

plt.figure(figsize=(10, 4))

plt.plot(t[:segment\_length], signal\_segment, label="Signal Segment", color='orange')

plt.title("Segment of Triangle Signal")

plt.xlabel("Time")

plt.ylabel("Amplitude")

plt.grid()

plt.legend()

plt.show()

# Compute and plot cross-correlation

correlation = correlate(triangle\_signal, signal\_segment, mode='full')

lags = np.arange(-segment\_length + 1, len(triangle\_signal))

plt.figure(figsize=(10, 4))

plt.plot(lags, correlation, label="Cross-Correlation")

plt.title("Cross-Correlation")

plt.xlabel("Lags")

plt.ylabel("Correlation")

plt.grid()

plt.legend()

plt.show()

Practical 3 sound wave

import cv2

import numpy as np

import matplotlib.pyplot as plt

from scipy.io import wavfile

from scipy.signal import convolve

# Template Matching Function

def template\_matching(image\_path, template\_path):

image = cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE)

template = cv2.imread(template\_path, cv2.IMREAD\_GRAYSCALE)

if image is None or template is None:

print("Error: Image or template not found.")

return

result = cv2.matchTemplate(image, template, cv2.TM\_CCOEFF\_NORMED)

\_, \_, \_, top\_left = cv2.minMaxLoc(result)

h, w = template.shape

cv2.rectangle(image, top\_left, (top\_left[0] + w, top\_left[1] + h), 255, 2)

plt.subplot(1, 2, 1), plt.imshow(template, cmap='gray'), plt.title("Template")

plt.subplot(1, 2, 2), plt.imshow(image, cmap='gray'), plt.title("Matched Image")

plt.show()

# Sound Convolution Function

def sound\_convolution(input\_wav, kernel):

sample\_rate, sound\_data = wavfile.read(input\_wav)

if sound\_data.ndim > 1:

sound\_data = sound\_data[:, 0]

sound\_data = sound\_data / np.max(np.abs(sound\_data))

convolved\_data = convolve(sound\_data, kernel, mode='same')

plt.plot(sound\_data, label="Original")

plt.plot(convolved\_data, label="Convolved", color='orange')

plt.legend()

plt.show()

# Example usage

template\_matching('11.jpg', 'template.jpg')

sound\_convolution('sample.wav', np.ones(5) / 5)

Practical 4 image transformation

import cv2

import numpy as np

import matplotlib.pyplot as plt

# Load grayscale image

def load\_image(path):

image = cv2.imread(path, cv2.IMREAD\_GRAYSCALE)

if image is None:

print("Error: Image not found.")

return None

return image

# Image transformations

def log\_transform(image):

return np.uint8(255 / np.log(1 + np.max(image)) \* np.log(1 + image))

def power\_law\_transform(image, gamma=1.0):

return np.uint8(255 \* ((image / 255) \*\* gamma))

def contrast\_adjustment(image, alpha=1.5, beta=0):

return cv2.convertScaleAbs(image, alpha=alpha, beta=beta)

def histogram\_equalization(image):

return cv2.equalizeHist(image)

def thresholding(image, threshold=128):

return cv2.threshold(image, threshold, 255, cv2.THRESH\_BINARY)[1]

def halftoning(image):

return np.where(image > 127, 255, 0).astype(np.uint8)

# Display images

def display\_results(images, titles):

plt.figure(figsize=(15, 8))

for i, (img, title) in enumerate(zip(images, titles)):

plt.subplot(2, 4, i + 1)

plt.imshow(img, cmap='gray')

plt.title(title)

plt.axis('off')

plt.show()

# Main Execution

image = load\_image('11.jpg')

if image is not None:

results = [

image,

log\_transform(image),

power\_law\_transform(image, gamma=0.5),

contrast\_adjustment(image, alpha=2.0, beta=50),

histogram\_equalization(image),

thresholding(image, threshold=128),

halftoning(image)

]

titles = ["Original", "Log Transform", "Power-law Transform",

"Contrast Adjustment", "Histogram Equalization",

"Thresholding", "Halftoning"]

display\_results(results, titles)

Practical 5 Image Sobel and enchanment

import cv2

import numpy as np

import matplotlib.pyplot as plt

def image\_enhancement(image\_path):

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

if img is None:

print("Error: Unable to load image!")

return

# Apply Sobel and Laplacian filters

sobel\_x = cv2.convertScaleAbs(cv2.Sobel(img, cv2.CV\_64F, 1, 0, ksize=3))

sobel\_y = cv2.convertScaleAbs(cv2.Sobel(img, cv2.CV\_64F, 0, 1, ksize=3))

gradient\_mag = cv2.convertScaleAbs(cv2.magnitude(sobel\_x, sobel\_y))

laplacian = cv2.convertScaleAbs(cv2.Laplacian(img, cv2.CV\_64F))

# Display images

titles = ["Original", "Sobel X", "Sobel Y", "Gradient Magnitude", "Laplacian"]

images = [img, sobel\_x, sobel\_y, gradient\_mag, laplacian]

plt.figure(figsize=(10, 6))

for i in range(5):

plt.subplot(2, 3, i+1)

plt.imshow(images[i], cmap='gray')

plt.title(titles[i])

plt.axis('off')

plt.show()

# Example usage

image\_enhancement('11.jpg') # Replace with your image path

Practical 6 Noise

import cv2

import numpy as np

import matplotlib.pyplot as plt

def noise\_smoothing(image\_path):

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

if img is None:

print("Error: Unable to load image!")

return

# Add noise

noise = np.random.normal(0, 25, img.shape).astype(np.uint8)

noisy\_img = cv2.add(img, noise)

# Apply smoothing filters

gaussian\_blur = cv2.GaussianBlur(noisy\_img, (5, 5), 1)

median\_blur = cv2.medianBlur(noisy\_img, 5)

# Display images

titles = ["Original", "Noisy", "Gaussian Blur", "Median Blur"]

images = [img, noisy\_img, gaussian\_blur, median\_blur]

plt.figure(figsize=(10, 6))

for i in range(4):

plt.subplot(2, 2, i + 1)

plt.imshow(images[i], cmap='gray')

plt.title(titles[i])

plt.axis('off')

plt.show()

# Example usage

noise\_smoothing('11.jpg') # Replace with your image path

Practical 7 Smooth sharped

import cv2

import numpy as np

import matplotlib.pyplot as plt

def apply\_image\_enhancements(image\_path):

# Load the image in grayscale

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

if img is None:

print("Error: Unable to load image!")

return

# 1. Smoothing using Gaussian Blur (removes noise)

smoothed = cv2.GaussianBlur(img, (5, 5), 1)

# 2. Sharpening using Laplacian (enhances edges)

laplacian = cv2.Laplacian(img, cv2.CV\_64F, ksize=3)

sharpened = cv2.convertScaleAbs(img - laplacian)

# 3. Unsharp Masking (sharpens by enhancing high-frequency details)

gaussian\_blur = cv2.GaussianBlur(img, (5, 5), 1)

unsharp\_mask = cv2.addWeighted(img, 1.5, gaussian\_blur, -0.5, 0)

# Plot the results

plt.figure(figsize=(12, 8))

plt.subplot(2, 2, 1)

plt.title("Original Image")

plt.axis('off')

plt.imshow(img, cmap='gray')

plt.subplot(2, 2, 2)

plt.title("Smoothing (Gaussian Blur)")

plt.axis('off')

plt.imshow(smoothed, cmap='gray')

plt.subplot(2, 2, 3)

plt.title("Sharpening (Laplacian)")

plt.axis('off')

plt.imshow(sharpened, cmap='gray')

plt.subplot(2, 2, 4)

plt.title("Unsharp Masking")

plt.axis('off')

plt.imshow(unsharp\_mask, cmap='gray')

plt.tight\_layout()

plt.show()

# Example usage

image\_path = '11.jpg' # Replace with your image path

apply\_image\_enhancements(image\_path)

Practical 8 X Y gradiant

import cv2

import matplotlib.pyplot as plt

def apply\_edge\_detection(image\_path):

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

if img is None:

print("Error: Unable to load image!")

return

sobel\_x = cv2.convertScaleAbs(cv2.Sobel(img, cv2.CV\_64F, 1, 0, ksize=3))

sobel\_y = cv2.convertScaleAbs(cv2.Sobel(img, cv2.CV\_64F, 0, 1, ksize=3))

sobel\_combined = cv2.convertScaleAbs(cv2.magnitude(sobel\_x, sobel\_y))

canny\_edges = cv2.Canny(img, 100, 200)

plt.figure(figsize=(12, 8))

titles = ["Original Image", "Sobel - X", "Sobel - Y", "Sobel - Combined"]

images = [img, sobel\_x, sobel\_y, sobel\_combined]

for i, (title, image) in enumerate(zip(titles, images)):

plt.subplot(2, 2, i + 1)

plt.imshow(image, cmap='gray')

plt.title(title)

plt.axis('off')

plt.figure()

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

plt.title("Canny Edge Detection")

plt.axis('off')

plt.show()

# Path to the image

image\_path = '11.jpg' # Replace with your image path

apply\_edge\_detection(image\_path)

Practical 9Erosion and diluation

import cv2

import numpy as np

import matplotlib.pyplot as plt

def morphological\_processing(image\_path):

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

if img is None:

print("Error: Unable to load image!")

return

kernel = np.ones((5, 5), np.uint8)

operations = {

"Erosion": cv2.erode(img, kernel),

"Dilation": cv2.dilate(img, kernel),

"Opening": cv2.morphologyEx(img, cv2.MORPH\_OPEN, kernel),

"Closing": cv2.morphologyEx(img, cv2.MORPH\_CLOSE, kernel),

"Gradient": cv2.morphologyEx(img, cv2.MORPH\_GRADIENT, kernel),

"Top Hat": cv2.morphologyEx(img, cv2.MORPH\_TOPHAT, kernel),

"Black Hat": cv2.morphologyEx(img, cv2.MORPH\_BLACKHAT, kernel)

}

plt.figure(figsize=(15, 10))

plt.subplot(3, 3, 1)

plt.title("Original Image")

plt.axis('off')

plt.imshow(img, cmap='gray')

for i, (title, result) in enumerate(operations.items(), 2):

plt.subplot(3, 3, i)

plt.title(title)

plt.axis('off')

plt.imshow(result, cmap='gray')

plt.tight\_layout()

plt.show()

# Path to the image

image\_path = '11.jpg' # Replace with the path to your image

morphological\_processing(image\_path)

Practical 10 edge and corner detection

import cv2

import numpy as np

import matplotlib.pyplot as plt

from skimage.feature import hog

def feature\_extraction(image\_path):

img = cv2.imread(image\_path)

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

# 1. Harris Corner Detection

harris\_corners = cv2.cornerHarris(gray, 2, 3, 0.04)

harris\_corners = cv2.dilate(harris\_corners, None)

img\_harris = img.copy()

img\_harris[harris\_corners > 0.01 \* harris\_corners.max()] = [0, 0, 255]

# 2. Shi-Tomasi Corner Detection

corners = cv2.goodFeaturesToTrack(gray, 100, 0.01, 10)

img\_shi\_tomasi = img.copy()

for corner in np.int0(corners):

cv2.circle(img\_shi\_tomasi, tuple(corner.ravel()), 3, (0, 255, 0), -1)

# 3. Blob Detection

detector = cv2.SimpleBlobDetector\_create()

keypoints = detector.detect(gray)

img\_blob = cv2.drawKeypoints(img, keypoints, None, (0, 0, 255))

# 4. HoG Features

hog\_features, hog\_image = hog(gray, pixels\_per\_cell=(8, 8), cells\_per\_block=(2, 2), visualize=True)

# 5. Haar Features (Face Detection)

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

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

img\_haar = img.copy()

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

cv2.rectangle(img\_haar, (x, y), (x + w, y + h), (255, 0, 0), 2)

# Plot results

plt.figure(figsize=(15, 10))

titles = ["Original Image", "Harris Corners", "Shi-Tomasi Corners", "Blob Detection", "HoG Features", "Haar Features (Face Detection)"]

images = [img, img\_harris, img\_shi\_tomasi, img\_blob, hog\_image, img\_haar]

for i, (title, image) in enumerate(zip(titles, images), 1):

plt.subplot(2, 3, i)

plt.title(title)

plt.axis('off')

plt.imshow(cv2.cvtColor(image, cv2.COLOR\_BGR2RGB) if i != 5 else image, cmap='gray' if i == 5 else None)

plt.tight\_layout()

plt.show()

# Path to the image

image\_path = '7.jpg' # Replace with the path to your image

feature\_extraction(image\_path)

Practical 11 Canny

import cv2

import numpy as np

import matplotlib.pyplot as plt

def shape\_segmentation(image\_path):

img = cv2.imread(image\_path)

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

blurred = cv2.GaussianBlur(gray, (5, 5), 1)

# 1. Edge-based Segmentation: Canny Edge Detection

edges = cv2.Canny(blurred, 50, 150)

# 2. Line Detection using Hough Transform

lines = cv2.HoughLines(edges, 1, np.pi / 180, 100)

img\_lines = img.copy()

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(img\_lines, (x1, y1), (x2, y2), (0, 255, 0), 2)

# 3. Circle Detection using Hough Transform

circles = cv2.HoughCircles(blurred, cv2.HOUGH\_GRADIENT, 1.2, 30, param1=50, param2=30, minRadius=10, maxRadius=100)

img\_circles = img.copy()

if circles is not None:

circles = np.uint16(np.around(circles))

for (x, y, r) in circles[0, :]:

cv2.circle(img\_circles, (x, y), r, (0, 255, 255), 3)

cv2.circle(img\_circles, (x, y), 2, (255, 0, 0), 3)

# 4. Region-based Segmentation: Watershed Algorithm

\_, binary = cv2.threshold(blurred, 0, 255, cv2.THRESH\_BINARY + cv2.THRESH\_OTSU)

kernel = np.ones((3, 3), np.uint8)

sure\_bg = cv2.dilate(binary, kernel, iterations=2)

dist\_transform = cv2.distanceTransform(binary, cv2.DIST\_L2, 5)

\_, sure\_fg = cv2.threshold(dist\_transform, 0.7 \* dist\_transform.max(), 255, 0)

unknown = cv2.subtract(sure\_bg, sure\_fg)

markers = cv2.connectedComponents(np.uint8(sure\_fg))[1] + 1

markers[unknown == 255] = 0

watershed\_img = img.copy()

markers = cv2.watershed(watershed\_img, markers)

watershed\_img[markers == -1] = [255, 0, 0]

# Plot the results

plt.figure(figsize=(15, 10))

titles = ["Original Image", "Edge Detection (Canny)", "Line Detection (Hough)", "Circle Detection (Hough)", "Region-based Segmentation (Watershed)"]

images = [img, edges, img\_lines, img\_circles, watershed\_img]

for i, (title, image) in enumerate(zip(titles, images), 1):

plt.subplot(2, 3, i)

plt.title(title)

plt.axis('off')

plt.imshow(cv2.cvtColor(image, cv2.COLOR\_BGR2RGB) if i != 2 else image, cmap='gray' if i == 2 else None)

plt.tight\_layout()

plt.show()

# Path to the image

image\_path = '10-11.jpg' # Replace with the path to your image

shape\_segmentation(image\_path)