SIFT

SIFT is a feature detection technique. Speaking of feature detection, we must know are features of an image first. Features can be anything such as corners, blobs, shapes, etc., that define the image. As the name suggests, SIFT is a scale-invariant method, and hence it involves scaling the picture. That is, the features are detected at any scale of the given image. For example, a corner in a zoomedout version of one image might appear like a circle in a zoomed-in version of the same image. And hence, this corner may be missed out in the zoomed-out image. But as SIFT is scale-invariant, it will capture this corner also

COMPLETE SOURCE CODE A:

```
import numpy as np
import cv2
path = r'C:\Users\WAINAINA\Desktop\VISION\WAINAINA.jpg'
image = cv2.imread(path)
sift = cv2.xfeatures2d.SIFT_create()
key_points = sift.detect(image)
new_image = cv2.drawKeypoints(image, key_points, None)
cv2.imshow("Detected Features", new_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

COMPLETE SOURCE CODE B:

```
import cv2
# reading the image
path = r'C:\Users\WAINAINA\Desktop\VISION\WAINAINA.jpg'
img = cv2.imread(path)
# convert to greyscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# create SIFT feature extractor
sift = cv2.xfeatures2d.SIFT_create()
```

```
# detect features from the image
keypoints, descriptors = sift.detectAndCompute(img, None)
# draw the detected key points
sift_image = cv2.drawKeypoints(gray, keypoints, img)
# show the image
cv2.imshow('image', sift_image)
```

Feature matching in OpenCV to match 2 images:

```
import cv2
# read the images
path1 = r'C:\Users\WAINAINA\Desktop\VISION\ZETECH.jpg'
img1 = cv2.imread(path1)
path2 = r'C:\Users\WAINAINA\Desktop\VISION\GEO.jpg'
img2 = cv2.imread(path2)
# 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()
```

MATCHING SIMILAR IMAGES:

```
import cv2
import numpy as np
path = r'C:\Users\WAINAINA\Desktop\VISION\ZETECH.jpg'
img_ = cv2.imread(path)
#img_ = cv2.resize(img_, (0,0), fx=1, fy=1)
img1 = cv2.cvtColor(img_,cv2.COLOR_BGR2GRAY)
path = r'C:\Users\WAINAINA\Desktop\VISION\ZETECH.jpg'
img= cv2.imread(path)
#img = cv2.resize(img, (0,0), fx=1, fy=1)
img2 = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
sift = cv2.xfeatures2d.SIFT_create()
# find the key points and descriptors with SIFT
kp1, des1 = sift.detectAndCompute(img1,None)
kp2, des2 = sift.detectAndCompute(img2,None)
#cv2.imshow('original_image_left_keypoints',cv2.drawKeypoints(img_,kp1,None))
#FLANN_INDEX_KDTREE = 0
#index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
#search_params = dict(checks = 50)
#match = cv2.FlannBasedMatcher(index_params, search_params)
```

```
match = cv2.BFMatcher()
matches = match.knnMatch(des1,des2,k=2)
good = []
for m,n in matches:
if m.distance < 0.03*n.distance:
good.append(m)
draw_params = dict(matchColor = (0,255,0), # draw matches in green color
singlePointColor = None,
flags = 2)
img3 = cv2.drawMatches(img_,kp1,img,kp2,good,None,**draw_params)
cv2.imshow("original_image_drawMatches.jpg", img3)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

<u>Panorama stitching algorithm</u>:

The panorama stitching algorithm can be divided into four basic fundamental steps. These steps are as follows:

1. Detection of keypoints (points on the image) and extraction of local invariant descriptors (SIFT feature) from input images.
2. Finding matched descriptors between the input images.
3. Calculating the homography matrix using the RANSAC algorithm.
4. The homography matrix is then applied to the image to wrap and fit those images and merge them into one.

```
import cv2
import numpy as np
path = r'C:\Users\WAINAINA\Desktop\VISION\ZETECH.jpg'
img_ = cv2.imread(path)
#img_ = cv2.imread('original_image_left.jpg')
```

```
#img_ = cv2.resize(img_, (0,0), fx=1, fy=1)
img1 = cv2.cvtColor(img_,cv2.COLOR_BGR2GRAY)
path = r'C:\Users\WAINAINA\Desktop\VISION\ZETECH.jpg'
img= cv2.imread(path)
img = cv2.imread('original_image_right.jpg')
#img = cv2.resize(img, (0,0), fx=1, fy=1)
img2 = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
sift = cv2.xfeatures2d.SIFT_create()
# find key points
kp1, des1 = sift.detectAndCompute(img1,None)
kp2, des2 = sift.detectAndCompute(img2,None)
#cv2.imshow('original_image_left_keypoints',cv2.drawKeypoints(img_,kp1,None))
#FLANN_INDEX_KDTREE = 0
#index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
#search_params = dict(checks = 50)
#match = cv2.FlannBasedMatcher(index_params, search_params)
match = cv2.BFMatcher()
matches = match.knnMatch(des1,des2,k=2)
good = []
for m,n in matches:
if m.distance < 0.03*n.distance:
good.append(m)
draw_params = dict(matchColor=(0,255,0),
singlePointColor=None,
flags=2)
img3 = cv2.drawMatches(img_,kp1,img,kp2,good,None,**draw_params)
#cv2.imshow("original_image_drawMatches.jpg", img3)
MIN_MATCH_COUNT = 10
if len(good) > MIN_MATCH_COUNT:
```

```
src_pts = np.float32([ kp1[m.queryIdx].pt for m in good ]).reshape(-1,1,2)
dst_pts = np.float32([ kp2[m.trainIdx].pt for m in good ]).reshape(-1,1,2)
M, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC, 5.0)
h,w = img1.shape
pts = np.float32([ [0,0],[0,h-1],[w-1,h-1],[w-1,0] ]).reshape(-1,1,2)
dst = cv2.perspectiveTransform(pts, M)
img2 = cv2.polylines(img2,[np.int32(dst)],True,255,3, cv2.LINE_AA)
#cv2.imshow("original_image_overlapping.jpg", img2)
else:
print("Not enought matches are found - %d/%d", (len(good)/MIN_MATCH_COUNT))
dst = cv2.warpPerspective(img_,M,(img.shape[1] + img_.shape[1], img.shape[0]))
dst[0:img.shape[0],0:img.shape[1]] = img
cv2.imshow("original_image_stitched.jpg", dst)
def trim(frame):
#crop top
if not np.sum(frame[0]):
return trim(frame[1:])
#crop top
if not np.sum(frame[-1]):
return trim(frame[:-2])
#crop top
if not np.sum(frame[:,0]):
return trim(frame[:,1:])
#crop top
if not np.sum(frame[:,-1]):
return trim(frame[:,:-2])
return frame
cv2.imshow("original_image_stitched_crop.jpg", trim(dst))
#cv2.imsave("original_image_stitched_crop.jpg", trim(dst))
```

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

A **complete source code to snitch the two images Vertically:**

```
# import required library
import cv2
import matplotlib.pyplot as plt
path = r'C:\Users\WAINAINA\Desktop\VISION\MANGU1.jpg'
img1 = cv2.imread(path)
path = r'C:\Users\WAINAINA\Desktop\VISION\MANGU2.jpg'
img2= cv2.imread(path)
# both image height and width should be same
img1 = cv2.resize(img1, (1200, 300))
img2 = cv2.resize(img2, (1200, 300))
# join the two images vertically
img = cv2.vconcat([img1, img2])
# Convert the BGR image to RGB
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.imshow(img) # or use cv2.imshow("MY OUTPUT",img)
plt.show()
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Texture Analysis

```
import cv2
import numpy as np
import skimage
path = r'C:\Users\WAINAINA\Desktop\VISION\WAINAINA.jpg'
image_spot = cv2.imread(path)
```

```
gray = cv2.cvtColor(image_spot, cv2.COLOR_BGR2GRAY)
# Find the GLCM
import skimage.feature as feature
# Param:
# source image
# List of pixel pair distance offsets - here 1 in each direction
# List of pixel pair angles in radians
graycom = skimage.feature.graycomatrix(gray, [1], [0, np.pi/4, np.pi/2,
3*np.pi/4], levels=256)
# Find the GLCM properties
contrast = skimage.feature.graycoprops(graycom, 'contrast')
dissimilarity = skimage.feature.graycoprops(graycom, 'dissimilarity')
homogeneity = skimage.feature.graycoprops(graycom, 'homogeneity')
energy = skimage.feature.graycoprops(graycom, 'energy')
correlation = skimage.feature.graycoprops(graycom, 'correlation')
ASM = skimage.feature.graycoprops(graycom, 'ASM')
print("Contrast: {}".format(contrast))
print("Dissimilarity: {}".format(dissimilarity))
print("Homogeneity: {}".format(homogeneity))
print("Energy: {}".format(energy))
print("Correlation: {}".format(correlation))
print("ASM: {}".format(ASM))
```

LBP

Local Binary Pattern (LBP) is a simple yet very efficient texture operator which labels the pixels of an image by thresholding the neighborhood of each pixel and considers the result as a binary number. LBP is a texture operator that labels the pixels of an image by thresholding the surrounding pixels and expressing them in binary numbers. What amaze me about LBP is that the operation returns a grayscale image that clearly displays the texture within the image.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
def get_pixel(img, center, x, y):
new_value = 0
try:
# If local neighbourhood pixel
# value is greater than or equal
# to center pixel values then
# set it to 1
if img[x][y] >= center:
new_value = 1
except:
# Exception is required when
# neighbourhood value of a center
# pixel value is null i.e. values
# present at boundaries.
pass
return new_value
# Function for calculating LBP
def lbp_calculated_pixel(img, x, y):
center = img[x][y]
val_ar = []
# top_left
val_ar.append(get_pixel(img, center, x-1, y-1))
# top
val_ar.append(get_pixel(img, center, x-1, y))
# top_right
val_ar.append(get_pixel(img, center, x-1, y + 1))
```

```
# right
val_ar.append(get_pixel(img, center, x, y + 1))
# bottom_right
val_ar.append(get_pixel(img, center, x + 1, y + 1))
# bottom
val_ar.append(get_pixel(img, center, x + 1, y))
# bottom_left
val_ar.append(get_pixel(img, center, x + 1, y-1))
# left
val_ar.append(get_pixel(img, center, x, y-1))
# Now, we need to convert binary
# values to decimal
power_val = [1, 2, 4, 8, 16, 32, 64, 128]
val = 0
for i in range(len(val_ar)):
val += val_ar[i] * power_val[i]
return val
path = r'C:\Users\WAINAINA\Desktop\VISION\WAINAINA.jpg'
img_bgr = cv2.imread(path, 1)
height, width, _ = img_bgr.shape
# We need to convert RGB image
# into gray one because gray
# image has one channel only.
img_gray = cv2.cvtColor(img_bgr,
cv2.COLOR_BGR2GRAY)
# Create a numpy array as
# the same height and width
# of RGB image
img_lbp = np.zeros((height, width),
```

```
np.uint8)
for i in range(0, height):
for j in range(0, width):
img_lbp[i, j] = lbp_calculated_pixel(img_gray, i, j)
plt.imshow(img_bgr)
plt.show()
plt.imshow(img_lbp, cmap ="gray")
plt.show()
print("LBP Program is finished")
```