4B-BAUTISTA-MP4

Machine Problem No. 4: Feature Extraction and Image Matching in Computer Vision

Objectives:

1. To apply different feature extraction methods (SIFT, SURF, ORB, HOG, Harris Corner Detection).

2. To perform feature matching using Brute-Force and FLANN matchers.

3. To implement the Watershed algorithm for image segmentation.

4. To visualize and analyze keypoints and matches between images.

5. To evaluate the performance of different feature extraction methods on different images

Task 1: Harris Corner Detection

• Load any grayscale image.

• Apply the Harris Corner Detection algorithm to find corners.

• Display the original image and the image with detected corners marked in red.

# Harris Corner Detection

def harris\_corner\_detection(image\_path):

  # Load the image

    img = cv2.imread('img1.png')

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

    # Harris Corner Detection

    gray = np.float32(gray\_img)

    corners = cv2.cornerHarris(gray, blockSize=2, ksize=3, k=0.04)

    # Mark corners on the original image

    img[corners > 0.01 \* corners.max()] = [0, 0, 255]

    # Display the result

    plt.imshow(cv2.cvtColor(img, cv2.COLOR\_BGR2RGB))

    plt.title('Harris Corner Detection')

    plt.show()

harris\_corner\_detection('img1.png')

Task 2: HOG Feature Extraction

• Load an image of a person (or another object).

• Convert the image to grayscale.

• Extract HOG (Histogram of Oriented Gradients) features from the image.

• Display the original image and the visualized HOG features.

# HOG Feature Extraction

def hog\_feature\_extraction(image\_path):

   # Extract HOG feature

  img = cv2.imread('img1.png')

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

  hog\_feature, hog\_image = hog(gray\_image, pixels\_per\_cell=(8, 8), cells\_per\_block=(2, 2),

                                  visualize=True, feature\_vector=True)

  hog\_image\_rescale = exposure.rescale\_intensity(hog\_image, in\_range=(0, 10))

  # Display the original the HOG images

  plt.subplot(1, 2, 1)

  plt.imshow(gray\_image, cmap='gray')

  plt.title('Original Image')

  plt.subplot(1, 2, 2)

  plt.imshow(hog\_image\_rescale, cmap='gray')

  plt.title('HOG Image')

  plt.show()

hog\_feature\_extraction('img1.png')

Task 3: ORB Feature Extraction and Matching

• Load two different images.

• Apply ORB (Oriented FAST and Rotated BRIEF) to detect and compute keypoints and descriptors

for both images.

• Use the FLANN-based matcher to match the ORB descriptors of the two images.

• Visualize the matching keypoints between the two images

# ORB Feature Matching

def orb\_feature\_matching(image\_path1, image\_path2):

    # Load the images

  img1 = cv2.imread(image\_path1, cv2.IMREAD\_GRAYSCALE)

  img2 = cv2.imread(image\_path2, cv2.IMREAD\_GRAYSCALE)

  # ORB detector

  orb = cv2.ORB\_create()

  # Find keypoints and description with ORB

  kp1, des1 = orb.detectAndCompute(img1, None)

  kp2, des2 = orb.detectAndCompute(img2, None)

  # FLANN-based matcher

  FLANN\_INDEX\_LSH = 6

  index\_params = dict(algorithm=FLANN\_INDEX\_LSH, table\_number=6, key\_size=12, multi\_probe\_level=1)

  search\_params = dict(checks=50)

  flann = cv2.FlannBasedMatcher(index\_params, search\_params)

  # Match discription

  matches = flann.knnMatch(des1, des2, k=2)

  good\_matches = []

  for m, n in matches:

    if m.distance < 0.7 \* n.distance:

      good\_matches.append(m)

  # Draw matching keypoints

  matched\_image = cv2.drawMatches(img1, kp1, img2, kp2, good\_matches, None, flags=cv2.DrawMatchesFlags\_NOT\_DRAW\_SINGLE\_POINTS)

  # Display the matched Image

  plt.imshow(matched\_image)

  plt.title('Matched Image')

  plt.show()

orb\_feature\_extraction\_and\_matching('img1.png', 'img2.png')

Task 4: SIFT and SURF Feature Extraction

• Load two images of your choice.

• Apply both SIFT and SURF algorithms to detect keypoints and compute descriptors.

• Visualize the keypoints detected by both methods in two separate images.

# SIFT and SURF Feature Extraction

def sift\_and\_surf\_feature\_extraction(image\_path1, image\_path2):

    # Load the images in grayscale

    img1 = cv2.imread(image\_path1, cv2.IMREAD\_GRAYSCALE)

    img2 = cv2.imread(image\_path2, cv2.IMREAD\_GRAYSCALE)

     # SIFT detector

    sift = cv2.SIFT\_create()

    kp1, des1 = sift.detectAndCompute(img1, None)

    kp2, des2 = sift.detectAndCompute(img2, None)

    # SURF detector

    surf = cv2.xfeatures2d.SURF\_create()

    kp1\_surf, des1\_surf = surf.detectAndCompute(img1, None)

    kp2\_surf, des2\_surf = surf.detectAndCompute(img2, None)

    # Draw keypoints for both SIFT and SURF

    img1\_sift\_kp = cv2.drawKeypoints(img1, kp1, None, flags=cv2.DrawMatchesFlags\_DRAW\_RICH\_KEYPOINTS)

    img1\_surf\_kp = cv2.drawKeypoints(img1, kp1\_surf, None, flags=cv2.DrawMatchesFlags\_DRAW\_RICH\_KEYPOINTS)

    # Display SIFT and SURF keypoints

    plt.subplot(1, 2, 1)

    plt.imshow(img1\_sift\_kp, cmap='gray')

    plt.title('SIFT Keypoints')

    plt.subplot(1, 2, 2)

    plt.imshow(img1\_surf\_kp, cmap='gray')

    plt.title('SURF Keypoints')

    plt.show()

sift\_and\_surf\_feature\_extraction('img1.png', 'img2.png')

Task 5: Feature Matching using Brute-Force Matcher

• Load two images and extract ORB descriptors.

• Match the descriptors using the Brute-Force Matcher.

• Display the matched keypoints between the two images.

# Brute Force Feature Matching

def brute\_force\_feature\_matching(image\_path1, image\_path2):

    # Initialize ORB detector

    orb = cv2.ORB\_create()

    img1 = cv2.imread(image\_path1, cv2.IMREAD\_GRAYSCALE)

    img2 = cv2.imread(image\_path2, cv2.IMREAD\_GRAYSCALE)

    # Find the keypoints and descriptors with ORB

    kp1, des1 = orb.detectAndCompute(img1, None)

    kp2, des2 = orb.detectAndCompute(img2, None)

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

    # Match descriptors

    matches = bf.match(des1, des2)

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

    matched\_image = cv2.drawMatches(img1, kp1, img2, kp2, matches[:30], None, flags=cv2.DrawMatchesFlags\_NOT\_DRAW\_SINGLE\_POINTS)

    # Display the matched keypoints using cv2\_imshow

    cv2\_imshow(matched\_image)

brute\_force\_feature\_matching('img1.png', 'img2.png')

Task 6: Image Segmentation using Watershed Algorithm

• Load any image of your choice.

• Convert the image to grayscale and apply a threshold to separate foreground from the

background.

• Apply the Watershed algorithm to segment the image into distinct regions.

• Display the segmented image.

def watershed\_segmentation(image\_path):

    # Load the image

    img = cv2.imread(image\_path)

    # Convert the image to grayscale

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

    \_, thresh = cv2.threshold(gray, 127, 255, cv2.THRESH\_BINARY\_INV)

    # Noise removal using morphological operations

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

    opening = cv2.morphologyEx(thresh, cv2.MORPH\_OPEN, kernel, iterations=2)

    sure\_bg = cv2.dilate(opening, kernel, iterations=3)

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

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

    unknown = cv2.subtract(sure\_bg, np.uint8(sure\_fg))

    \_, markers = cv2.connectedComponents(np.uint8(sure\_fg))

    markers = markers + 1

    markers[unknown == 255] = 0

    # Apply the Watershed algorithm

    img[markers == -1] = [255, 0, 0]

    markers = cv2.watershed(img, markers)

    img[markers == -1] = [255, 0, 0]

    img[markers == 1] = [0, 255, 0]

    # Display the segmented image

    cv2\_imshow(img)

# Example usage:

watershed\_segmentation('img1.png')