STB600 Lab 5: Feature Extraction

Atieh Sahraeidolatkhaneh: atieh.sahraeidolatkhaneh@hv.se

Yongcui Mi: yongcui.mi@hv.se

1. Introduction

Image feature extraction is the process of constructing derived values (features) that are intended to be informative and non-redundant. Image features include colors, gray levels, shapes, textures and, more broadly, any piece of information which is relevant for solving a certain application.

When dealing with image classification, features should be most relevant for discrimination and have low intra-class variability and high between-class variability. There are two principal aspects of image feature extraction: *feature detection* and *feature description*. Numerical feature descriptions are usually "packaged" in the form of a feature vector (i.e., a $1 \times n$ or $n \times 1$ matrix) and can then be further processed.

The shape of an object is an important and basic visual feature for describing image content. Therefore, the first task of this lab is to learn how to extract different object shape features, such as area, perimeter, aspect ratio, etc. This is mainly done by extracting contours from images and then calculating the contour properties. The second task introduces the scale-invariant feature transform (SIFT) algorithm for local feature detection.

(PS: You can go through the functions using pictures from Lab 4. If you want to explore more, for instance to classify images by extracted features, you will find two subfolders containing two different classes of images in this lab folder.)

2. Learning Outcomes

After completing this lab, you should be able to:

- find contours in a binary image and compute basic shape features (area, perimeter, bounding boxes, etc.);
- interpret shape features (e.g. aspect ratio, equivalent diameter, extreme points) in terms of object geometry;

- detect SIFT keypoints in an image and visualize them;
- explain what information SIFT keypoints and descriptors capture.

3. Task 1: Calculate Different Contour Features

To find different features of contours, such as area, perimeter, centroid, bounding boxes etc., you should:

1. read an image and convert it to grayscale;
2. segment the object from the background (e.g. thresholding);
3. find the contours using `cv2.findContours()`;
4. pick one contour of interest (for example, the largest one);
5. compute and visualize its properties.

A minimal processing skeleton could look like this:

```
import cv2
import numpy as np

# Read image
# img = cv2.imread("path/to/image.png")

# Convert to grayscale
# gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Threshold to obtain a binary image
# _, th = cv2.threshold(gray, 0, 255,
#                       cv2.THRESH_BINARY + cv2.THRESH_OTSU)

# Find contours
# contours, hierarchy = cv2.findContours(th, cv2.RETR_EXTERNAL,
#                                        cv2.CHAIN_APPROX_SIMPLE)

# Choose one contour, for example the largest:
# cnt = max(contours, key=cv2.contourArea)

# Now compute features on 'cnt' as shown below
```

Listing 1: Example pipeline for contour extraction

Then get the properties of the contour by using different functions as shown below (assume `cnt` is a single contour):

```
# Moments:
M = cv2.moments(cnt)

# Area:
area = cv2.contourArea(cnt)

# Perimeter:
perimeter = cv2.arcLength(cnt, True)

# Convex Hull:
hull = cv2.convexHull(cnt)

# Checking Convexity:
k = cv2.isContourConvex(cnt)

# Straight Bounding Rectangle:
x, y, w, h = cv2.boundingRect(cnt)
cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 2)

# Rotated Rectangle:
rect = cv2.minAreaRect(cnt)          # (center, (w, h), angle)
box = cv2.boxPoints(rect)
box = np.int0(box)
cv2.drawContours(img, [box], 0, (0, 0, 255), 2)

# Fitting an Ellipse:
ellipse = cv2.fitEllipse(cnt)
cv2.ellipse(img, ellipse, (0, 255, 0), 2)

# Orientation (from fitted ellipse):
(xe, ye), (MA, ma), angle = cv2.fitEllipse(cnt)

# Aspect Ratio:
x, y, w, h = cv2.boundingRect(cnt)
aspectRatio = float(w) / h

```

```
# Equivalent Diameter:
area = cv2.contourArea(cnt)
equi_diameter = np.sqrt(4 * area / np.pi)

# Extreme Points:
leftmost   = tuple(cnt[cnt[:, :, 0].argmin()][0])
rightmost  = tuple(cnt[cnt[:, :, 0].argmax()][0])
topmost    = tuple(cnt[cnt[:, :, 1].argmin()][0])
bottommost = tuple(cnt[cnt[:, :, 1].argmax()][0])
```

Mini-task (recommended)

Pick **one image from each of the two class subfolders** and compute for each object:

- area, perimeter, aspect ratio, equivalent diameter;
- visualize the contour and bounding box on the image.

Compare the feature values between the two classes and write down which features seem most useful to distinguish them.

4. Task 2: An Introduction to the Scale-Invariant Feature Transform (SIFT)

The Scale-Invariant Feature Transform (SIFT) is a computer vision algorithm to detect, describe, and match local features in images. The objective is to find *keypoints* and associated *descriptors* that are distinctive and robust to changes in scale, rotation, and partially to illumination.

In this task, you will:

- construct a SIFT object,
- detect keypoints in an image,
- draw them on the image and inspect where they appear.

Try to understand, at a high level, why these keypoints are useful and what information they carry. Also, look up what the inputs to the `detect` or `detectAndCompute` functions are.

```
# In case of error regarding a missing module,
# try to install 'opencv-contrib-python' in your environment.

# pip install opencv-contrib-python
```

Basic usage:

```
# Create SIFT object:
sift = cv2.SIFT_create()

# Detect keypoints:
keypoints = sift.detect(gray_img, None)

# Or detect keypoints and compute descriptors in one step:
keypoints, descriptors = sift.detectAndCompute(gray_img, None)

# Draw keypoints:
img_kp = cv2.drawKeypoints(gray_img, keypoints, None)
# or with size and orientation:
# img_kp = cv2.drawKeypoints(gray_img, keypoints, None,
#                            flags=cv2.
   DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
```

Things to reflect on

- On which parts of the image do SIFT keypoints appear (corners, edges, flat regions)?
- If you run SIFT on two images from the same class, do you see similar patterns of keypoints?