Master OpenCV 2

Created By	FREDDIE MAR MAÑEGOS
Last Edited	@Nov 4, 2019 3:26 PM
Tags	Artificial Intelligence Computer Vision

4 Image Segmentation

Image Segmentation is partitioning images into different regions.

1. Understanding contours
2. Sorting contours by size or left to right
3. Approximating contours & finding their convex hull
4. Matching Contour Shapes
5. Mini Project # 2 – Identifying Shapes
6. Line Detection
7. Circle Detection
8. Blob Detection
9. Mini Project # 3 – Counting Circles and Ellipses

4.1 Contours

Contours are continuous lines or curves that bound or cover the full boundary of an object in an image.

Contours are very important in:

- Object Detection

- Shape Analysis

OpenCV stores Contours in a list of lists.

cv2.findContours(image, Retrieval Mode, Approximation Method)

- Returns → contours, hierarchy
- Contours are stored as a numpy array of (x,y) points that form the contour.
- Hierarchy describes the child-parent relationships between contours (i.e. contours within contours)

Retrieval Mode:

- cv2.CHAIN_APPROX_NONE – Stores all the points along the line (inefficient!)
- cv2.CHAIN_APPROX_SIMPLE – Stores the end points of each line

Drawing Contours:

cv2.drawContours(image, contours, specific contour, color,thickness)

- The 'contours' parameter is the output from findContours
- Specific contour relates to which contour you wish to draw e.g. Contour[0], Contour[1].
- If you wish to draw all contours, use '-1'

```
import cv2
import numpy as np

# Let's load a simple image with 3 black squares
image = cv2.imread('images/shapes.jpg')
cv2.imshow('Input Image', image)
cv2.waitKey(0)

# Grayscale
gray = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)

# Find Canny edges
edged = cv2.Canny(gray, 30, 200)
cv2.imshow('Canny Edges', edged)
cv2.waitKey(0)

# Finding Contours
# Use a copy of your image e.g. edged.copy(), since findContours alters the image
_, contours, hierarchy = cv2.findContours(edged, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
cv2.imshow('Canny Edges After Contouring', edged)
```

```
cv2.waitKey(0)

print("Number of Contours found = " + str(len(contours)))

# Draw all contours
# Use '-1' as the 3rd parameter to draw all
cv2.drawContours(image, contours, -1, (0,255,0), 3)

cv2.imshow('Contours', image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Hierarchy in Contours

Hierarchy Types (the first two are the most useful)

- cv2.RETR_LIST – Retrieves all contours
- cv2.RETR_EXTERNAL - Retrieves external or outer contours only
- cv2.RETR_COMP - Retrieves all in a 2-level hierarchy
- cv2.RETR_TREE - Retrieves all in full hierarchy

Hierarchy is stored in the following format: [Next, Previous, First Child, Parent]

cv2.findContours(image, Retrieval Mode, Approximation Method)

Returns → ret, contours, hierarchy

NOTE

In OpenCV 2.X, findContours returns only 2 arguments which is contours, hierarchy

If you're using OpenCV 2.X replace line 12 with:

contours, hierarchy = cv2.findContours(image, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)

The variable 'contours' are stored as a numpy array of (x,y) points that form the contour

While, 'hierarchy' describes the child-parent relationships between contours (i.e. contours within contours)

Approximation Methods

Using cv2.CHAIN_APPROX_NONE stores all the boundary points. But we don't necessarily need all bounding points. If the points form a straight line, we only need the start and ending points of that line.

Using cv2.CHAIN_APPROX_SIMPLE instead only provides these start and end points of bounding contours, thus resulting in much more efficient storage of contour information..

Sorting Contours

Sorting contours is quite useful when doing image processing.

Sorting by Area can assist in Object Recognition (using contour area)

- Eliminate small contours that may be noise
- Extract the largest contour

Sorting by spatial position (using the contour centroid)]

- Sort characters left to right
- Process images in specific order

```
import cv2
import numpy as np

# Load our image
image = cv2.imread('images/bunchofshapes.jpg')
cv2.imshow('0 - Original Image', image)
cv2.waitKey(0)

# Create a black image with same dimensions as our loaded image
blank_image = np.zeros((image.shape[0], image.shape[1], 3))

# Create a copy of our original image
orginal_image = image

# Grayscale our image
gray = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)

# Find Canny edges
edged = cv2.Canny(gray, 50, 200)
cv2.imshow('1 - Canny Edges', edged)
cv2.waitKey(0)

# Find contours and print how many were found
contours, hierarchy = cv2.findContours(edged.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
print ("Number of contours found = ", len(contours))
```

```
#Draw all contours
cv2.drawContours(blank_image, contours, -1, (0,255,0), 3)
cv2.imshow('2 - All Contours over blank image', blank_image)
cv2.waitKey(0)

# Draw all contours over blank image
cv2.drawContours(image, contours, -1, (0,255,0), 3)
cv2.imshow('3 - All Contours', image)
cv2.waitKey(0)

cv2.destroyAllWindows()
```

Contour Moments

Moments are a set of scalers that are an aggregate of set of vectors. Used in Mechanics and Statistics quite frequently and has now been adopted in Computer Vision.

Without getting to heavy into the mathematics behind it think of it as a measure of image intensities

An image with pixel intensities I(x,y), moments are given by:

$$M_{ij} = \sum_x \sum_y I(x,y)$$

Approximating Contours

Approximating contours is useful when correcting slight distortions your contour

We can use approxPolyDP to achieve this:

cv2.approxPolyDP(contour, Approximation Accuracy, Closed)

- Contour – is the individual contour we wish to approximate
- Approximation Accuracy – Important parameter is determining the accuracy of the approximation. Small values give precise approximations, large values give more generic approximation. A good rule of thumb is less than 5% of the contour perimeter
- Closed – a Boolean value that states whether the approximate contour should be open or closed

```python
import cv2
import numpy as np

# Load the shape template or reference image
template = cv2.imread('images/4star.jpg',0)
cv2.imshow('Template', template)
cv2.waitKey()

# Load the target image with the shapes we're trying to match
target = cv2.imread('images/shapestomatch.jpg')
target_gray = cv2.cvtColor(target,cv2.COLOR_BGR2GRAY)

# Threshold both images first before using cv2.findContours
ret, thresh1 = cv2.threshold(template, 127, 255, 0)
ret, thresh2 = cv2.threshold(target_gray, 127, 255, 0)

# Find contours in template
_, contours, hierarchy = cv2.findContours(thresh1, cv2.RETR_CCOMP, cv2.CHAIN_APPROX_SIMPLE)

# We need to sort the contours by area so that we can remove the largest
# contour which is the image outline
sorted_contours = sorted(contours, key=cv2.contourArea, reverse=True)

# We extract the second largest contour which will be our template contour
template_contour = contours[1]

# Extract contours from second target image
_, contours, hierarchy = cv2.findContours(thresh2, cv2.RETR_CCOMP, cv2.CHAIN_APPROX_SIMPLE)

for c in contours:
    # Iterate through each contour in the target image and
    # use cv2.matchShapes to compare contour shapes
    match = cv2.matchShapes(template_contour, c, 3, 0.0)
    print(match)
    # If the match value is less than 0.15 we
    if match < 0.15:
        closest_contour = c
    else:
        closest_contour = []

cv2.drawContours(target, [closest_contour], -1, (0,255,0), 3)
cv2.imshow('Output', target)
cv2.waitKey()
cv2.destroyAllWindows()
```