Introduction to Image Processing and Computer Vision

Lab 2: Segmentation and Labeling

Contents

1 Introduction

In computer vision, image segmentation is the process of partitioning a digital image into multiple segments. More precisely, image segmentation is the process of assigning a label to every pixel in an image such that pixels with the same label share certain characteristics. [1]

In this case, we are trying to separate leaves on given sample photos from the background. There are three types of leaves in different shapes and colors. Backgrounds are even more varied, with different items, patterns, shapes and colors.

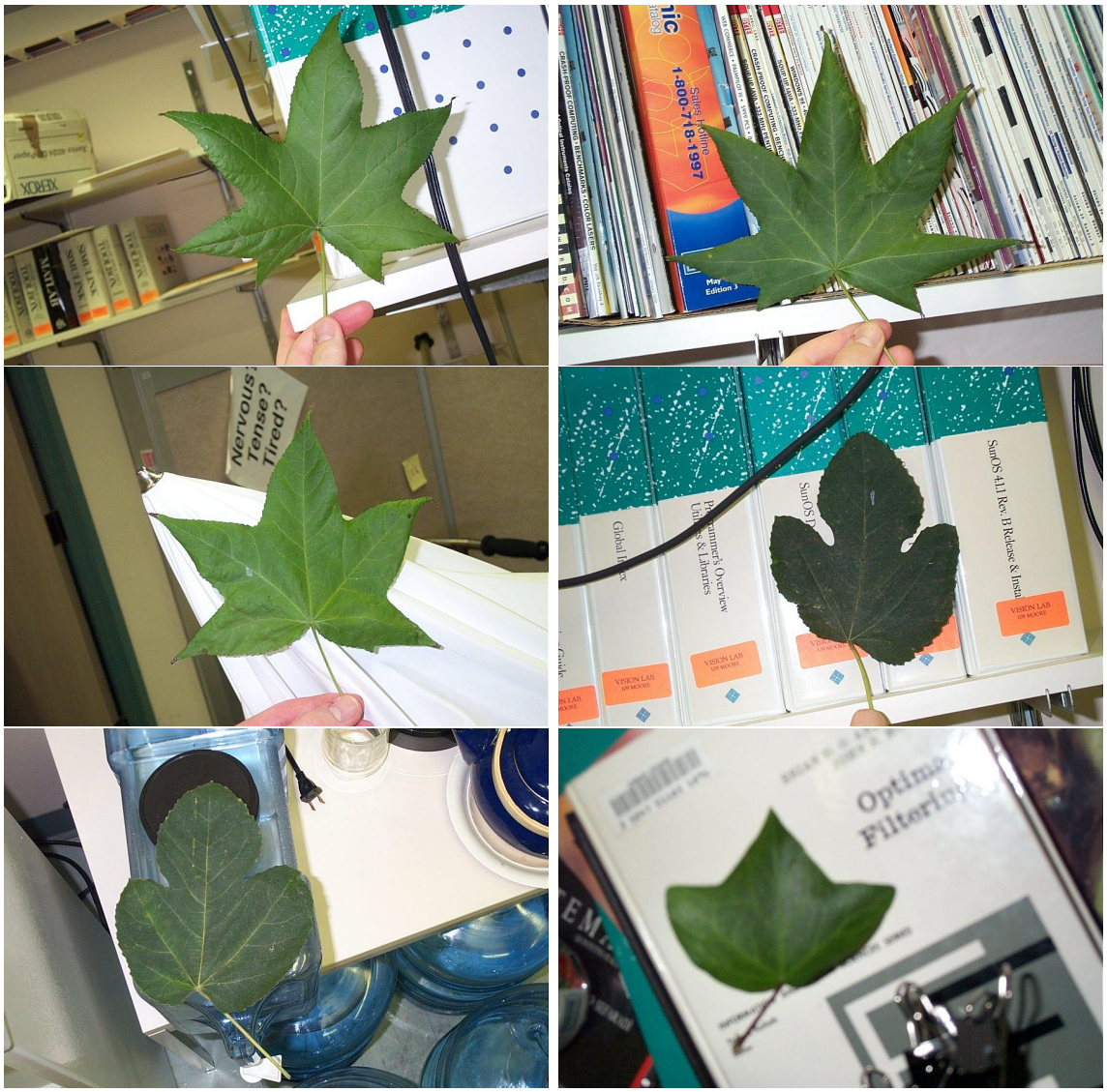


Figure 1: Sample of input images.

Having these images as an input, we have to output leaves separated from the background, masks which are used to "cut" leaves from the picture and finally, original photos with leaves

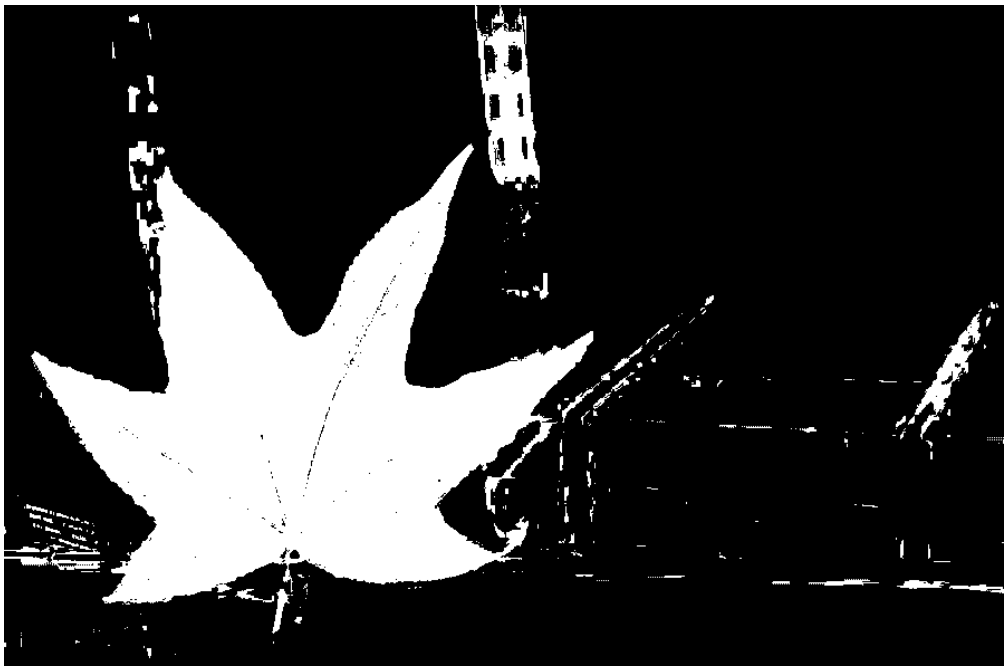
marked with the rectangle (bounding box).

1.1 Algorithm description

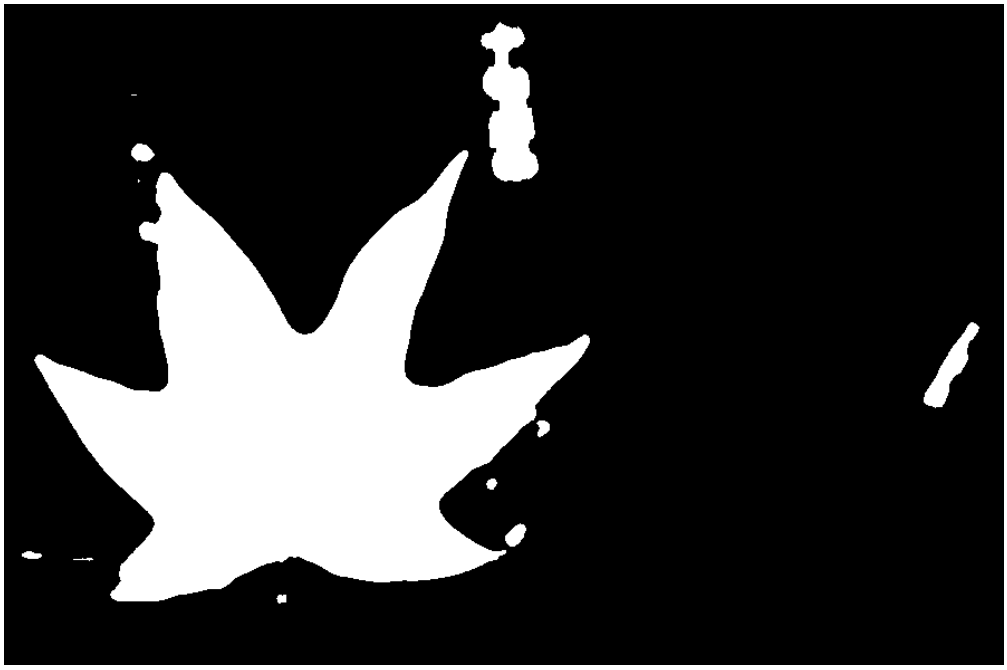
Our script is built around an observation that all leaves in provided images are in some shade of green, to add more, they are usually the only things of that color on the pictures.

First, a mask is created for each image. The first step of creating the mask is removal of all elements which are not in particular range of shades of green (HSV color palette is used, since it is more convenient for such applications, i.e. it is easier to get shades of particular color). The shades were obtained by manual testing, since it was the quickest way in this case, but software such as *GIMP* might have also been helpful in reading colors and choosing ranges.

```
# remove all colors except of particular shades of green
hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
lower_green = np.array([30, 22, 22])
upper_green = np.array([85, 235, 195])
mask = cv2.inRange(hsv, lower_green, upper_green)
```



Then, the mask is "cleaned up" a bit from the background noise which may appear, since the color range on which the mask was created includes gray and white (some leaves have very light parts due to lighting). After that, a *median blur* [3] is applied, both to "fill" holes in masks and to clean the background from noise.



Opening [2] could also be used, but as it turns out, although even more noise and false

positives are removed, masks are hugely degraded. This topic is elaborated more in *Section 2.2*.

```
# blur, optionally use opening
if USEOPENING:
    mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel)
mask = cv2.medianBlur(mask, 15)
```

Next, the contour detection [4] is used on the mask, which usually contains a leaf and some minor background noise at this point. The largest contour is choosen – the sought leaf.

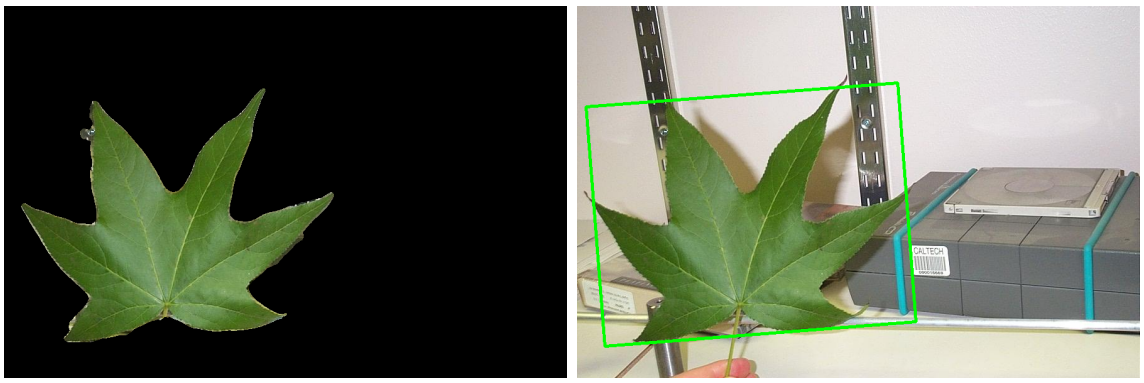
```
# find the contours in the thresholded image, then sort the
    contours
# by their area, keeping only the largest one
conts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,cv2
   .CHAIN_APPROX_SIMPLE)
conts = conts[1]
c = sorted(conts, key = cv2.contourArea, reverse = True)[0]
mask = cv2.drawContours(np.zeros((height,width,3), np.uint8
   ), [c], 0, (255,255,255), cv2.FILLED)
mask = cv2.cvtColor(mask,cv2.COLOR_BGR2GRAY)
```

Finally, one copy of the image is segmented, and a bounding box is drawn on another.

```
segmented = cv2.bitwise_and(image, image, mask=mask)
```

...

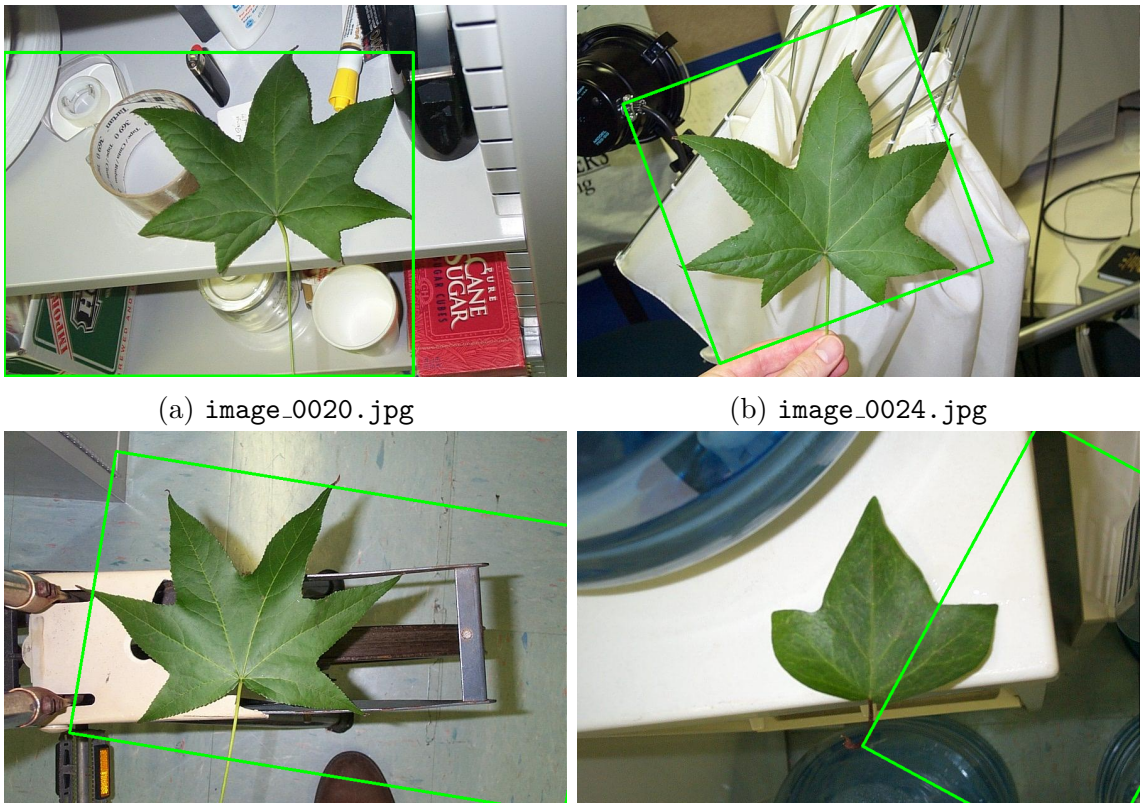
```
    # compute the rotated bounding box of the largest contour
    rect = cv2.minAreaRect(c)
    box = cv2.boxPoints(rect)
    box = np.int0(box)
    # draw a bounding box
    cv2.drawContours(image, [box], -1, (0, 255, 0), 3)
```



2 Results

2.1 Results assessment

A high accuracy was obtained, i.e. leaves are choosen correctly on all pictures except four – accuracy 97.85%, where on only one a leaf is completely missed.



(a) `image_0020.jpg` (b) `image_0024.jpg`

(c) `image_0032.jpg` (d) `image_0146.jpg`

Figure 2: Failed bounding boxes.

By looking at masks of these images, one may find that color separation wasn't good enough. To improve that, division of the color range into many, but more specific might be helpful.

(a) `image_0020.jpg` (b) `image_0024.jpg` (c) `image_0032.jpg` (d) `image_0146.jpg`

Figure 3: Masks of failed attempts (without largest contour selection applied).

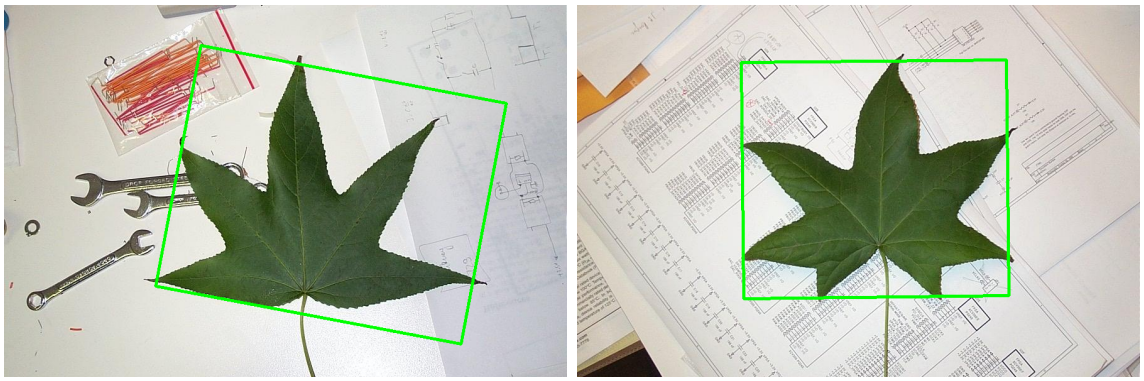
To be fair, the algorithm is sometimes too aggresive, other times – not enough. In few cases much noise is left on the mask.

Very often, leaves ends are cut off, therefore we do not obtain the whole leaf. There are two reasons for that – the first one is cleanup. Median blur causes thin ends to disappear.

(a) Before blurring (b) After blurring

Figure 4: Example of median blur destroing sharp leave ends on the mask.

The other reason is different leaf color on edges. We could include it in the thresholding range but that would cause too many different objects appearing. On the other hand, the issue appears only on particular types of leaves and it is not too drastic.



(a) `image_0005.jpg` (b) `image_0015.jpg`

(c) `image_0028.jpg` (d) `image_0053.jpg`

Figure 5: Examples of cut ends on some types of leaves.

2.2 Another approach

There is a way to make the method even more accurate in bounding boxes, accuracy 99.5%* and make masks backgrounds even more clean. It can be achieved by using *opening* [2], although not without some kind of sacrificion. Even more severe edge cutting would appear and many masks would be much more eroded, and "squared", so much less helpful in segmentation. Also many bounding boxes would cover even less of the leaves.

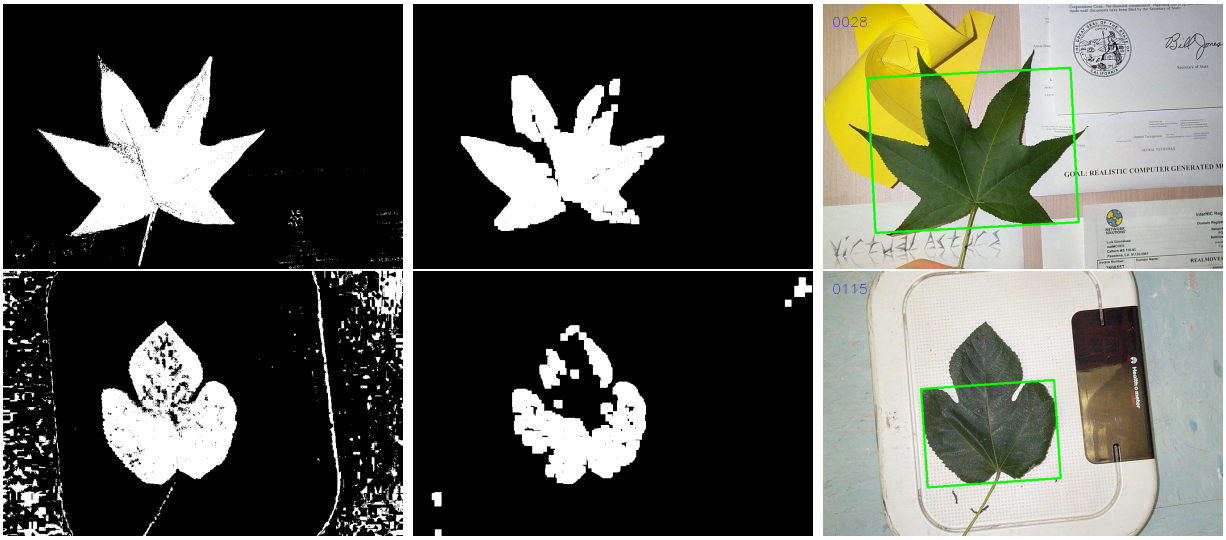


Figure 6: Examples of failures when using opening. Files `image_0028.jpg` (top) and `image_0115.jpg` (bottom). In the left-to-right order: mask, the mask after opening, the original image with bounding box drawn on.

3 Source code

```
import numpy as np
import cv2
import os

DEBUG_MODE = False # Shows all steps of processing without
   writing anything to disk.
SOURCEPATH = "leaves/"
OUTPUTPATH = "output/"
USEOPENING = False # Use morphological opening. See report for
   more information.

if not os.path.exists(SOURCEPATH):
```

*Only one image failed out of 186.

```
    print("Path␣" + SOURCEPATH + "␣does␣not␣exist!")
    exit(1)
if not os.path.exists(OUTPUTPATH):
    print("Path␣" + OUTPUTPATH + "␣does␣not␣exist!")
    exit(1)

# kernel for opening
kernel = np.ones((15, 15), np.uint8)

# for all given pictures...
for imgnum in range(1,186+1):
    # create filenamestring for particular image, read it
    imgnumstring ='{0:04d}'.format(imgnum)
    imageName = "image_" + imgnumstring
    image = cv2.imread(SOURCEPATH + imageName + '.jpg', cv2.
       IMREAD_COLOR)
    height, width = image.shape[:2]
    if DEBUG_MODE:
        # draw image name on the picture
        cv2.putText(image, imageName, (20,50), cv2.
           FONT_HERSHEY_SIMPLEX, 1, 255)
    # remove all colors except of particular shades of green
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    lower_green = np.array([30, 22, 22])
    upper_green = np.array([85, 235, 195])
    mask = cv2.inRange(hsv, lower_green, upper_green)
    if DEBUG_MODE:
        cv2.imshow("mask1", mask)
    # blur, optionally use opening
    if USEOPENING:
        mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel)
    mask = cv2.medianBlur(mask, 15)
    if DEBUG_MODE:
        cv2.imshow("mask2", mask)
    # find the contours in the thresholded image, then sort the
       contours
    # by their area, keeping only the largest one
    conts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,cv2
       .CHAIN_APPROX_SIMPLE)
    conts = conts[1]
    c = sorted(conts, key = cv2.contourArea, reverse = True)[0]
```

```
    mask = cv2.drawContours(np.zeros((height,width,3), np.uint8
       ), [c], 0, (255,255,255), cv2.FILLED)
    mask = cv2.cvtColor(mask,cv2.COLOR_BGR2GRAY)
    segmented = cv2.bitwise_and(image, image, mask=mask)
    if DEBUG_MODE:
        cv2.imshow("mask3", mask)
        cv2.imshow("segmented", segmented)
    else:
        # save final mask and the segmented image
        cv2.imwrite(OUTPUTPATH + imageName + "-mask" + ".png",
           mask)
        cv2.imwrite(OUTPUTPATH + imageName + "-segmented" + ".
           png", segmented)
    # compute the rotated bounding box of the largest contour
    rect = cv2.minAreaRect(c)
    box = cv2.boxPoints(rect)
    box = np.int0(box)
    # draw a bounding box
    cv2.drawContours(image, [box], -1, (0, 255, 0), 3)
    if DEBUG_MODE:
        cv2.imshow("Image", image)
        cv2.waitKey(0)
    else:
        cv2.imwrite(OUTPUTPATH + imageName + "-box" + ".png",
           image)
        print(imageName + ".jpg" + "␣saved␣in␣" + OUTPUTPATH)
exit(0)
```

References

[1] https://en.wikipedia.org/wiki/Image_segmentation

[2] https://docs.opencv.org/2.4/doc/tutorials/imgproc/opening_closing_hats/opening_closing_hats.html

[3] https://docs.opencv.org/2.4/doc/tutorials/imgproc/gausian_median_blur_bilateral_filter/gausian_median_blur_bilateral_filter.html

[4] https://docs.opencv.org/3.1.0/d4/d73/tutorial_py_contours_begin.html