

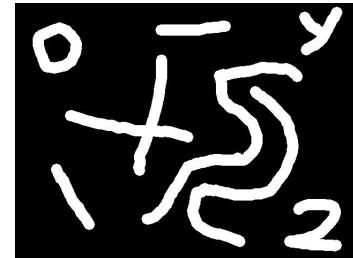


Lab Instructions - session 6

Connected Components, Thresholding, Morphology

Connected Components

We intend to find the connected components (i.e. groups of connected pixels) in the following image (**img1.bmp**). The image is binary-valued: the pixel intensity levels are either **0** or **255**.



File: **connected_components.py**

```
import numpy as np
import cv2

I = cv2.imread('img1.bmp', cv2.IMREAD_GRAYSCALE)
n,C = cv2.connectedComponents(I);

print("n=%d"%n)
print(np.unique(I))
print(np.unique(C))

cv2.imshow('I', I)
cv2.waitKey(0) # press any key to continue...

for k in range(n):

    # show the k-th connected component
    Ck = np.zeros(I.shape, dtype=I.dtype)
    Ck[C == k] = 255;

    cv2.imshow('C%d'%k, Ck)
    cv2.waitKey(0) # press any key to continue...

I = cv2.cvtColor(I, cv2.COLOR_GRAY2BGR)

font = cv2.FONT_HERSHEY_SIMPLEX

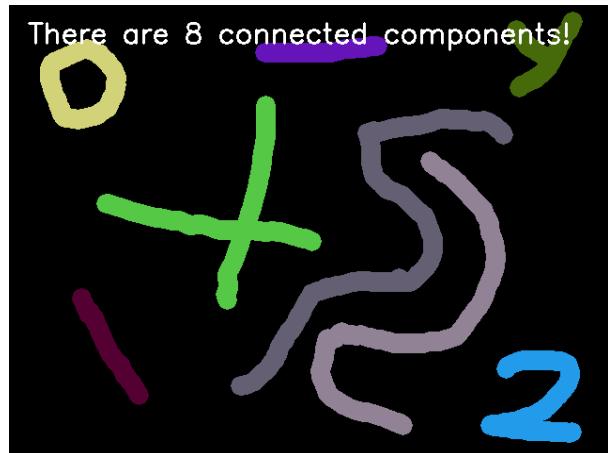
# note: background is also counted as a connected component by openCV
cv2.putText(I,'There are %d connected components!'%(n-1),(20,40), font,
1,(0,0,255),2)

cv2.imshow('Num', I)
cv2.waitKey(0)
```

- What are the values of `n`, `np.unique(I)` and `np.unique(C)`? Why?
- What does `Ck[C == k] = 255` do?
- Why `n=9` while there are only 8 connected components? Why the first connected components look like an inverted version of the original.

Task 0: Random Colors!

You have already detected connected components in a grayscale image using cv2.connectedComponents. Currently, each component is shown as a separate binary image. Let's now improve the visualization by assigning a random color to each connected component and showing them all together in a single RGB image.



Hints:

Use the following lines to guide your implementation:

```
# Create an output image with 3 channels (RGB)
output = np.zeros((C.shape[0], C.shape[1], 3), dtype=np.uint8)

# Assign a random color to each component (except background)
np.random.seed(42)
colors = [np.random.randint(0, 255, size=3).tolist() for _ in range(n)]

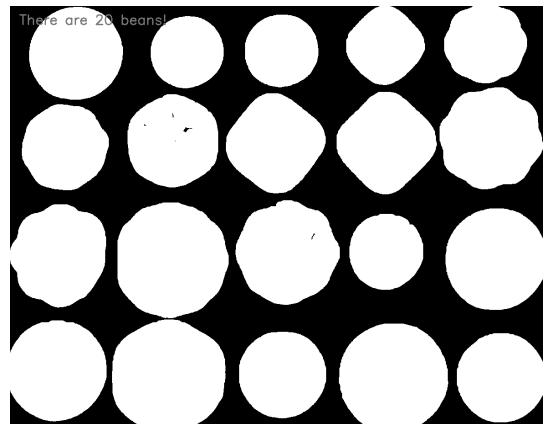
# Fill output image with color for each label
for i in range(n):
    output[C == i] = colors[i]
```

- What does `output[C == i] = colors[i]` do ?

Task 1: Coin Counting!

We want to count the number of coins in the following image (coins.jpg). The image contains several coins placed on a plain background.

You are provided with code that loads the image and converts it to grayscale. Your task is to threshold the image properly, so that the coins appear white (foreground) and the background is black. Then, using **cv2.connectedComponents**, count how many coins are present in the image.



File: `coin_counting.py`

```
import numpy as np
import cv2

I = cv2.imread('coins.jpg')
G = cv2.cvtColor(I, cv2.COLOR_BGR2GRAY)

#Step 0: Invert the grayscale image so that coins become white and
background becomes black
# This helps with better thresholding and connected component detection

cv2.imshow('Grayscale', G)
cv2.waitKey(0) # press any key to continue...
# Step 1: Try applying a threshold
# Replace 127 with a value you find appropriate
ret, T = cv2.threshold(G, 127, 255, cv2.THRESH_BINARY)

cv2.imshow('Thresholded', T)
cv2.waitKey(0)

# Step 2: Try improving segmentation using morphological operations
# You can try erosion to separate connected objects or remove noise
# You may also use dilation to fill holes inside coins
```



```
# Uncomment and try adjusting the kernel size
# kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5, 5))
# T = cv2.erode(T, kernel)
# cv2.imshow('After Erosion', T)
# cv2.waitKey(0)

# kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (15, 15))
# T = cv2.dilate(T, kernel)
# cv2.imshow('After Dilation', T)
# cv2.waitKey(0)

# Step 3: Count connected components
n, C = cv2.connectedComponents(T)

print("Number of connected components (including background):", n)
print("Estimated number of coins:", n - 1)

font = cv2.FONT_HERSHEY_SIMPLEX
cv2.putText(T, 'There are %d coins!' % (n - 1), (20, 40), font, 1, 255,
2)
cv2.imshow('Result', T)
cv2.waitKey(0)
```

- What does `cv2.threshold` do? What are its first, second and third arguments?
- What are the two major problems with the above approach?
- What does `cv2.erode` and `cv2.dilate` do?
- Why might the thresholded image include small white noise points that are not coins? how to fix it ?
- What happens if two coins are touching? How does erosion help? What kernel size works best ?

Task 2: Simple Background Subtraction

Consider the following pair of images. In the second image, few books have been placed in the scene.



The following code tries to count the number of books by subtracting the two images, thresholding the result and then counting the connected components. Your job is to fix this code to get the correct number of books and find the biggest book.

File: **task2.py**

```
import numpy as np
import cv2

I1 = cv2.imread('task2_1.jpg')
I2 = cv2.imread('task2_2.jpg')

cv2.imshow('Image 1', I1)
cv2.waitKey(0)

cv2.imshow('Image 2 (background)', I2)
cv2.waitKey(0)

K = np.abs(np.int16(I2)-np.int16(I1)) # take the (signed int) difference
K = K.max(axis=2) # choose the maximum value over color channels
K = np.uint8(K)
cv2.imshow('The difference image', K)
cv2.waitKey(0)

threshold = 37
ret, T = cv2.threshold(K, threshold, 255, cv2.THRESH_BINARY)
cv2.imshow('Thresholded', T)
cv2.waitKey(0)

## opening
# kernel = np.ones((5,5),np.uint8)
# T = cv2.morphologyEx(T, cv2.MORPH_OPEN, kernel)
# cv2.imshow('After Opening', T)
# cv2.waitKey(0)
```



```

## closing
# kernel = np.ones((10,10),np.uint8)
# T = cv2.morphologyEx(T, cv2.MORPH_CLOSE, kernel)
# cv2.imshow('After Closing', T)
# cv2.waitKey(0)
n,C = cv2.connectedComponents(T);

J = I2.copy()
J[T != 0] = [255,255,255]
font = cv2.FONT_HERSHEY_SIMPLEX
cv2.putText(J, 'There are %d books!' %(n-1), (20,40), font, 1,(0,0,255),2)
cv2.imshow('Number', J)
cv2.waitKey()

## connected components with statistics
# n,C,stats, centroids = cv2.connectedComponentsWithStats(T);

# for i in range(n):
#     print("-"*20)
#     print("Connected Component: ", i)
#     print("center= %.2f,%.2f"%(centroids[i][0], centroids[i][1]))
#     print("left= ", stats[i][0])
#     print("top= ", stats[i][1])
#     print("width= ", stats[i][2])
#     print("height= ", stats[i][3])
#     print("area= ", stats[i][4])

# j = n-1 # j: index of largest connected component (change this line)
# J[C == j] = [0,0,255] # Paint the largest connected component in RED
# cv2.imshow('Largest book in red', J)
# cv2.waitKey()

```

- Change the threshold variable and see the result. Find a reasonable threshold (it does not need to give the correct result.)
- Uncomment the four lines after the line `## opening`. Run the code. What does the opening operator do? Change the kernel size and see the results.
- Uncomment the four lines after the line `## closing`. Run the code. What does the closing operator do?
- Tune the threshold, opening kernel size and closing kernel size until you get the desired result, finding all the books and their number.
- Uncomment all the lines after `## connected components with statistics`. It gives statistics about each connected component including centroid, leftmost pixel location, top-most pixel location, width, height and area (number of pixels) of each connected component. We want to detect biggest book in the image and paint it in red. Currently, the code paints the last connected component (`j=n-1`). Use the statistics to find the connected component with **the largest area**, and paint the biggest book in red.

References

- [OpenCV-Python Tutorials - Image Thresholding](#)
- [OpenCV-Tutorials - Structural Analysis and Shape Descriptors](#)
- [OpenCV-Python Tutorials - Morphological Transformations](#)