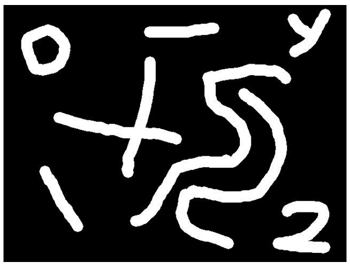
Lab Instructions - session 6

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**.

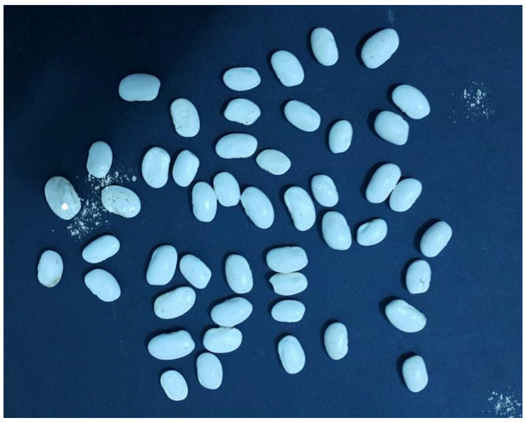


File: **connected_components.py**

```
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 1: Bean Counting!



We want to count the number of beans in the following image. I have written a code which transforms the image to grayscale and then thresholds the image, setting the pixel intensities above 127 to 255 and the others to 0. It then counts the connected components of the thresholded image. But the code does not work as intended. Your job as a *bean counter* is to fix the bugs and count the number of beans in the image.

By the way, there are **48** beans! :)

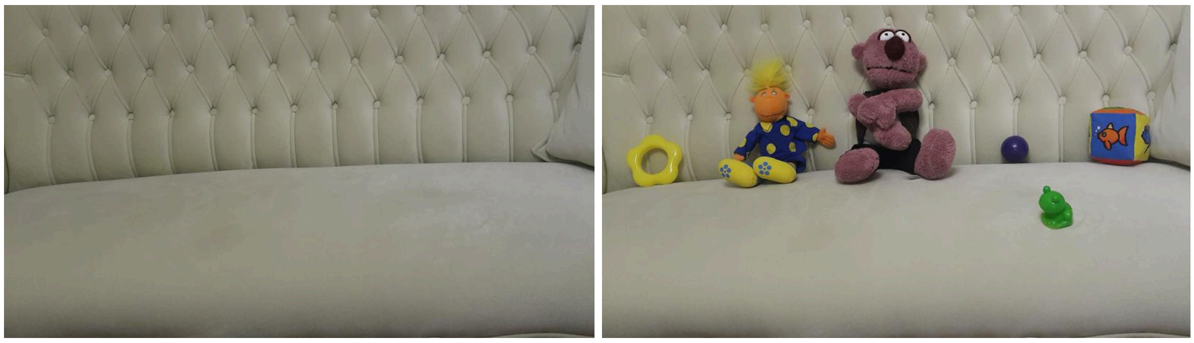
File: **bean_counting.py**

```
import numpy as np
import cv2

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

ret, T = cv2.threshold(G,127,255,cv2.THRESH_BINARY)

cv2.imshow('Thresholded', T)
cv2.waitKey(0) # press any key to continue...

## erosion
# kernel = np.ones((5,5),np.uint8)
# T = cv2.erode(T,kernel)
# cv2.imshow('After Erosion', T)
# cv2.waitKey(0) # press any key to continue...

n,C = cv2.connectedComponents(T);

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

- What does `cv2.threshold` do? What are its first, second and third arguments? THRESH_BINARY -> I[x,y] > thresh ? maxval : 0
- What are the two major problems with the above approach? Overlapping beans
  Lighting and Fixed Thresholding
- Uncomment the 4 lines after the line (not including) `## erosion`
- Run the code. What does `cv2.erode` do?
- The erosion kernel size is `(5,5)`. Change it until you get the desired result (48 beans).

Task 2: Simple Background Subtraction

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



The following code tries to count the number of toys 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 toys and find the biggest toy.
File: **task2.py**

```
import numpy as np
import cv2

I1 = cv2.imread('scene1.jpg')
I2 = cv2.imread('scene2.jpg')

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


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

K = np.abs(np.int16(I2)-np.int16(I1)) # take the (signed int) differnce
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 = 80
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 toys!'%(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 Toy 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 toys 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 **Jenab Khan** (the biggest toy) 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 toy in red.

References

- OpenCV-Python Tutorials - Image Thresholding
- OpenCV-Tutorials - Structural Analysis and Shape Descriptors
- OpenCV-Python Tutorials - Morphological Transformations