Name : Chican Costin-Andrei

Group: 442Ca

Assignment 1
-FIPCV-

Subsection 1 - Assigned Image : im8.jpg

Perform color segmentation on the read image by choosing the correct color space, color component and threshold/thresholds. The output image should be a binary image with black for the background and white for all the objects in the foreground. Save the segmented image as *segmented_x.jpg*.

Explain your choice of the color component and the threshold used. If necessary, perform the appropriate morphological operations on the segmented image to remove potential noise and to fill the holes in objects. Save the resulting image as *segmented_improved_x.jpg*.

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# SECTION 1

img = cv2.imread('im_8.jpg')
img_gray = cv2.imread('im_8.jpg', 0)  # Flag = 0 => Read the image as
grayscale

B, G, R = cv2.split(img)  # Splitting the BGR image in 3 channels

imgHSV = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
H, S, V = cv2.split(imgHSV)  # Splitting the HSV image in 3 channels

imgYCrCb = cv2.cvtColor(img, cv2.COLOR_BGR2YCrCb)
Y, Cr, Cb = cv2.split(imgYCrCb)  # Splitting the YCrCb image in 3 channels

plt.figure(1)
plt.subplot(221), plt.imshow(img_gray, cmap='gray'), plt.title("Gray")
plt.subplot(222), plt.imshow(B, cmap='gray'), plt.title("B")
plt.subplot(223), plt.imshow(G, cmap='gray'), plt.title("G")
plt.subplot(224), plt.imshow(R, cmap='gray'), plt.title("R")

plt.figure(2)
plt.subplot(221), plt.imshow(img_gray, cmap='gray'), plt.title("Gray")
plt.subplot(222), plt.imshow(H, cmap='gray'), plt.title("H")
plt.subplot(223), plt.imshow(S, cmap='gray'), plt.title("S")
plt.subplot(224), plt.imshow(V, cmap='gray'), plt.title("V")
```

```
plt.figure(3)
plt.subplot(221), plt.imshow(img_gray, cmap='gray'), plt.title("Gray")
plt.subplot(222), plt.imshow(Y, cmap='gray'), plt.title("Y")
plt.subplot(223), plt.imshow(Cr, cmap='gray'), plt.title("Cr")
plt.subplot(224), plt.imshow(Cb, cmap='gray'), plt.title("Cb")
# plt.show()

trsh = 130
trsh2 = 199

th, dst1 = cv2.threshold(G, trsh, 255, cv2.THRESH_BINARY_INV)  # Values under
threshold become 255 (white)
th2, dst11 = cv2.threshold(G[0:2000, 0:1500], trsh2, 255, cv2.THRESH_BINARY)
# Values over threshold become 255(white)
dst2 = dst1.copy()
dst2[:2000, 0:1500] = dst11  # Selected region is replaced in original image

plt.figure(4)
plt.subplot(131), plt.imshow(dst1, cmap='gray'), plt.title("First threshold")
plt.subplot(132), plt.imshow(dst11, cmap='gray'), plt.title("Second
threshold")
plt.subplot(133), plt.imshow(dst2, cmap='gray'), plt.title("Combined image")
# plt.show()

cv2.imwrite('segmented_8.jpg', dst2, [cv2.IMWRITE_JPEG_QUALITY, 80])  #
Saving image

kernel = cv2.getStructuringElement(cv2.MORPH_DILATE, ksize=(4, 4))
dst2_dilate = cv2.dilate(dst2, kernel, iterations=7)  # Applying dilation
operation

kernel1 = cv2.getStructuringElement(cv2.MORPH_ERODE, ksize=(4, 4))
dst2_erode = cv2.erode(dst2_dilate, kernel1, iterations=6)  # Applying
erosion operation
cv2.imwrite('segmented_improved_8.jpg', dst2_erode,
[cv2.IMWRITE_JPEG_QUALITY, 80])

plt.figure(5)
plt.subplot(131), plt.imshow(dst2, cmap='gray'), plt.title("Original
Segmented Image")
plt.subplot(132), plt.imshow(dst2_dilate, cmap='gray'), plt.title("Dilated
Segmented Image")
plt.subplot(133), plt.imshow(dst2_erode, cmap='gray'), plt.title("Eroded
Segmented Image")
# plt.show()
```

In order to choose the right color space and color component, I have split the image in 3 channels for three different color spaces : RGB, HSV, YCbCr.

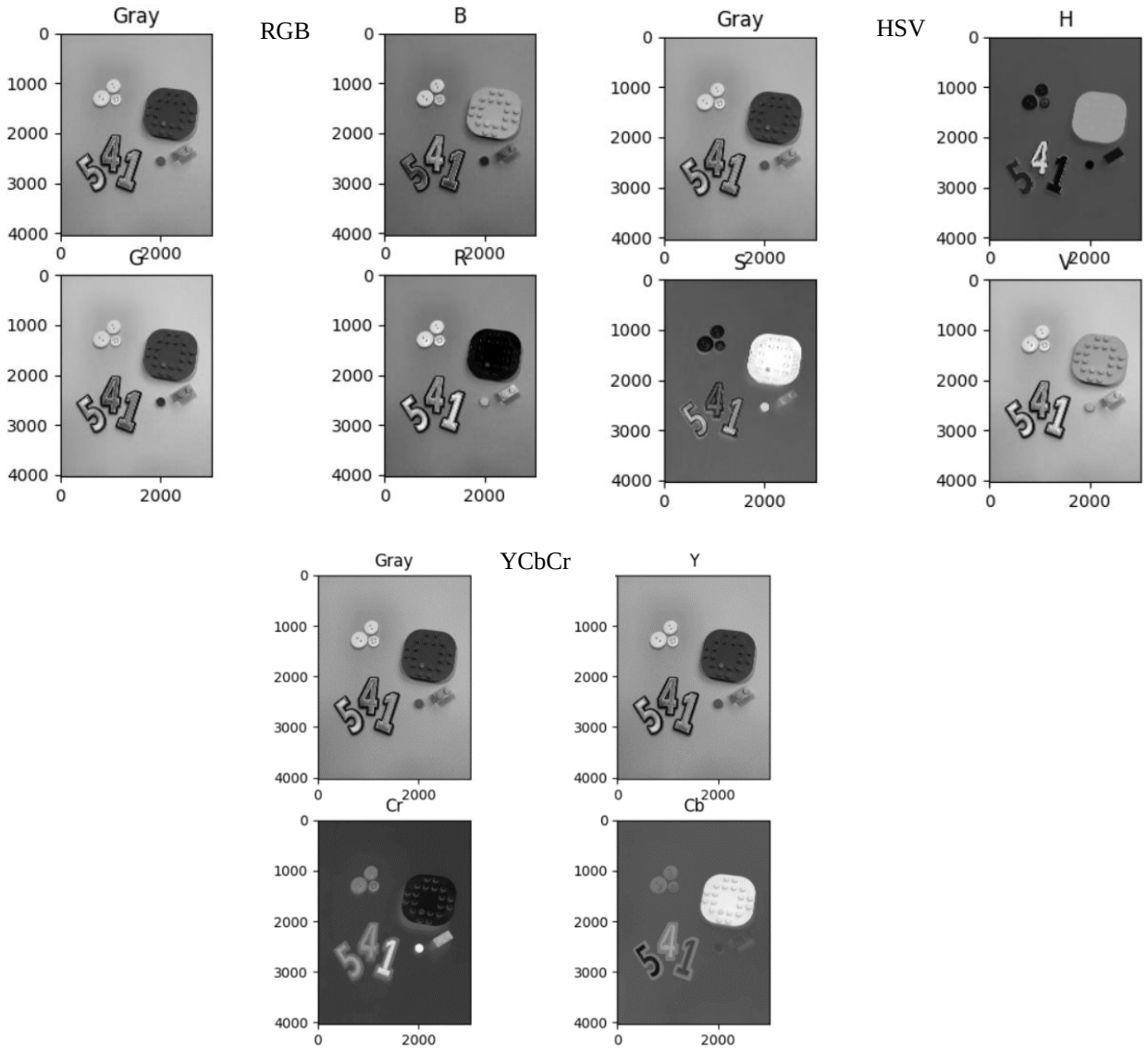

Figure 1 – Comparison between color spaces

When it comes to choosing the right color component for the segmentation operation, we need to choose the one with the best contrast between the background and the objects. By looking at the color spaces listed above, there is no clear winner, because, some objects are brighter than the background and some are darker, making the finding of a threshold for segmentation quite difficult. I've chosen to go with the blue channel of the RGB color space due to most objects being darker than the background.

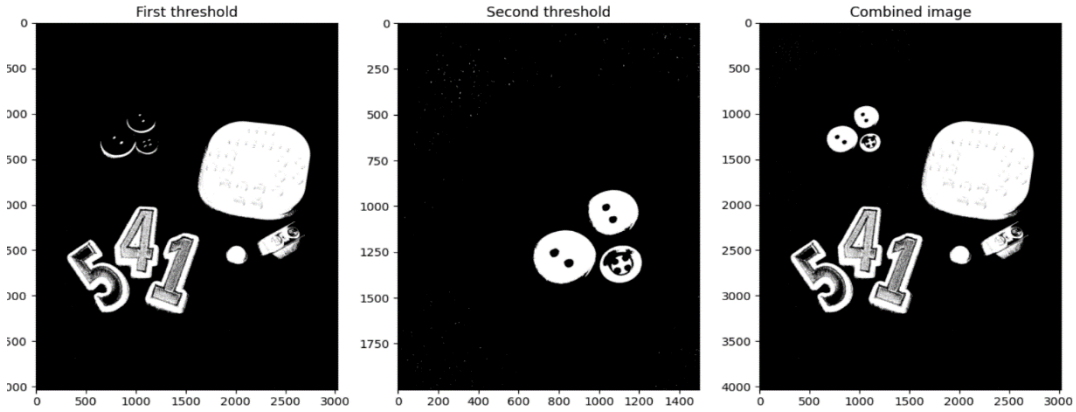

Figure 2 – Segmented Image

In the blue channel image, the buttons are lighter than the background and the other objects are darker than it. In order to segment the image, I have applied 2 thresholds, the first being for the entire image, which made all the objects white, except for the buttons. I've chosen a threshold of 130 and type BINARY_INV because the bigger values left lots of white dots on the background and the smaller values made the small LEGO piece untraceable. For the region of the buttons, I have applied a threshold of 199 and type BINARY, the smaller values leaving white dots on the image and the bigger ones degrading the buttons shape.

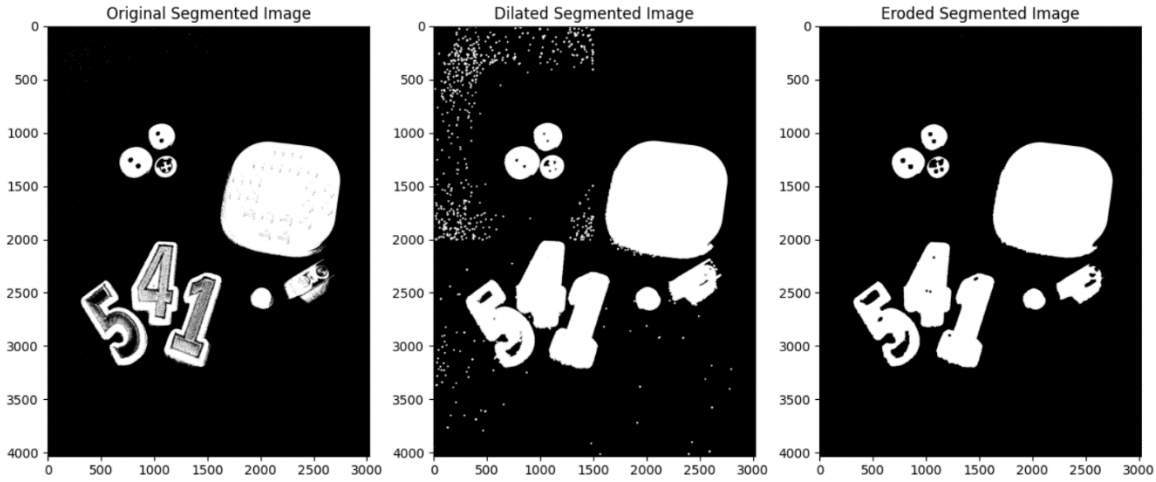

Figure 3 - Segmented Image Improved

As Figure 3 depicts, the segmented image has some defects, such as: the numbers and the LEGO bricks having too many black areas, the background having too many white dots. Firstly, I have applied the dilation operation, to fill the black areas on the objects. As a result, the objects remained with few black points on the surface, but the white points on the background got bigger. This downside was eliminated using the erosion operation with 6 iterations. Trying a number of iterations greater than 6 affected the shape of the buttons.

Subsection 2 - Assigned Image : im8.jpg ; Objects to Count : Numbers

Label the blobs in the previous image using connected components analysis. Save the image with labels as *blobs_x.jpg*.

Write the code to count the objects assigned to you in the table. To do this, you have to discard objects having incorrect numbers of connected components. Pay attention at connected objects. Display an image containing only the valid objects, where each valid object is depicted with a color of your choice. In the top right corner of this image, display a text like this one: “Counted objects: N”, where N is the number of objects that the algorithm has counted. Save the image as *valid_blobs_x.jpg*.

```
# SECTION 2

no_labels, imLabels, statistics, centre =
cv2.connectedComponentsWithStats(dst2_erode)  # Detecting components
print("Detected Objects", no_labels)

plt.figure(6)
plt.imshow(imLabels, cmap='gray'), plt.title('Labeled Image')
plt.show()
cv2.imwrite('blobs_8.jpg', imLabels, [cv2.IMWRITE_JPEG_QUALITY, 80])

selected = np.zeros(imLabels.shape)  # Build  full black image
no_shapes = 0
for i in range(1, no_labels):  # Loop starts from 1, label 0 is the
background
    height = statistics[i, cv2.CC_STAT_HEIGHT]  # Get the height statistics
    if 800 <= height < 900:  # Selecting only the heights that correspond to
numbers
        selected[imLabels == i] = 255  # Selected component becomes white
        no_shapes = no_shapes + 1

plt.figure(7)
plt.imshow(selected, cmap='gray'), plt.title('Detected Numbers')
plt.show()

img_selected = np.zeros_like(img)  # Convert the 1 channel image into 3
channels image
img_selected[:, :, 0] = selected
img_selected[:, :, 1] = selected
img_selected[:, :, 2] = selected

img_selected[np.where((img_selected == [255, 255, 255]).all(axis=2))] = [0,
0, 255]  # White pixels change color
img_selected = cv2.putText(img_selected, f'Counted Objects: {no_shapes}',
(1950, 150), cv2.FONT_HERSHEY_COMPLEX,
                           3, color=(255, 255, 0), thickness=2)
plt.figure(8)
plt.imshow(img_selected), plt.imshow(img_selected)
plt.show()
img_selected = cv2.cvtColor(img_selected, cv2.COLOR_BGR2RGB)
cv2.imwrite('valid_blobs_8.jpg', img_selected, [cv2.IMWRITE_JPEG_QUALITY,
80])
```

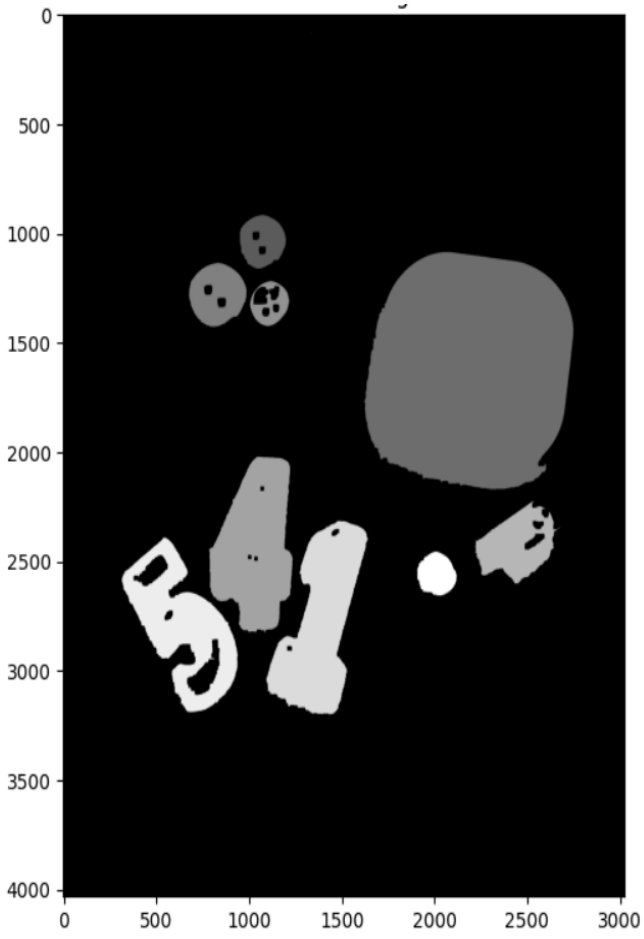

Figure 4 – Detected Objects

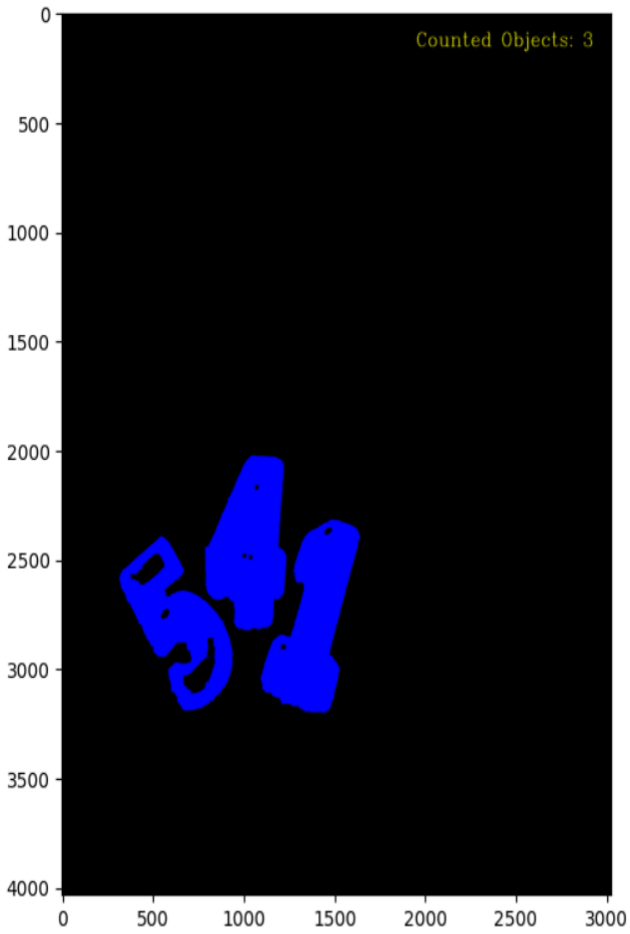

Figure 5 – Valid Objects

In order to detect the components, I have used cv2.connectedComponentsWithStats due to the fact that this method also provides the statistics for the detected objects. There was a number of 15 objects detected in the first picture instead of 9 due to the small imperfections in the binary mask.

The numbers in the second picture were selected based on their height. By looking at the object statistics, I have noticed that the numbers have heights between 800 and 900, so I have used a FOR loop to select only those objects. As the second picture depicts, all three numbers were detected.

Subsection 3 – Images to match : books.jpg, casa.jpg ; Features Method : ORB

Read the pair of *images to match*, assigned to you in the table. Use SIFT or ORB descriptors to match features, as indicated in the table. Use Brute Force Matcher to match the descriptors computed earlier. Create the Matcher object using cv2.BFMatcher(). Specify the correct distance measurement to be used: cv2.NORM_L2 or cv2.NORM_HAMMING. Once the BFMatcher object is created, use the method BFMatcher.match() to match the descriptors. Sort them in ascending order of their distances, so that best matches (with low distance) come in front. Draw only first 15-20 matches using the function cv2.drawMatches and save the final image as *mached.jpg*. Comment on the result of the matching process. Are the various features correctly identified?

```
# SECTION 3

img1 = cv2.imread('books.jpg', 0)  # Reading image as grayscale
img2 = cv2.imread('casa.jpg', 0)

orbObj = cv2.ORB_create()  # Create ORB Object
kp1, descriptors1 = orbObj.detectAndCompute(img1, None)  # Computing
descriptors and keypoints
kp2, descriptors2 = orbObj.detectAndCompute(img2, None)

brute_force = cv2.BFMatcher(cv2.NORM_HAMMING)  # ORB uses NORM_HAMMING
matched_descriptors = brute_force.match(descriptors1, descriptors2)  #
Matching descriptors
matched_descriptors = sorted(matched_descriptors, key=lambda j: j.distance)
# Sorting the matches based on distance

matches = cv2.drawMatches(img1, kp1, img2, kp2, matched_descriptors[:20],
None, flags=2)  # Drawing first 20 matches
# #Flag =2 => shows only matched keypoints
cv2.imwrite('matched.jpg', matches, [cv2.IMWRITE_JPEG_QUALITY, 80])

plt.figure(9)
plt.imshow(matches), plt.title('Matched Objects')
plt.show()
```

Figure 6 – Matched Features

Figure 6 shows the best 20 matches between the two images. By looking at the first 20 matches, we can tell that there were no wrong matches, all that had been found were coming from the ‘casa’ book, the algorithm not being tricked by the presence of two other books The ORB algorithm managed to detect various features such the curves of the letter ‘S’ or the corners of the letter A. All the 20 matches are correct, each corresponding to one or more features of a letter in the book title. In conclusion, the features were correctly identified.