Carson Stevens

CSCI 437 Homework 2

Question 1

(25 pts) A two dimensional correlation can be implemented more efficiently if the filter w(x,y) is separable, meaning that it can be written as a product of two functions, one that is a function only of x and the other only of y. In other words, w(x,y) = wx(x) wy(y).

The 3x3 averaging filter is an example of a separable filter:

1	**1**	**1**
1	**1**	**1**
1	**1**	**1**

This is separable because it can be written as

w(x,y) = wx(x)wy(y)

where

wx(x) =

1	**1**	**1**

wy(y) =

1
1
1

In the case of a separable filter, you can do a 1D correlation with w y (y) along the individual columns of the input image, followed by computing a 1D correlation with w x (x) along the rows of the previous result. Demonstrate the validity of this with an example.

Consider the small image matrix below (consider it to be padded outside the boundary with zeroes):

0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	**1**	**3**	**0**	**1**	**2**	0	0	0
0	0	0	**0**	**1**	**2**	**1**	**0**	0	0	0
0	0	0	**1**	**0**	**0**	**3**	**2**	0	0	0
0	0	0	**0**	**1**	**2**	**1**	**0**	0	0	0
0	0	0	**1**	**2**	**0**	**3**	**2**	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0

By hand, compute the 2D correlation of this matrix with the full 3x3 averaging filter. Then compute the two 1D correlations as described above, and show that they are the same.

(1x3) Result

0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	1/3	**4/3**	**4/3**	**4/3**	**3/3**	**3/3**	2/3	0	0
0	0	0	**1/3**	**3/3**	**4/3**	**3/3**	**1/3**	0	0	0
0	0	1/9	**1/3**	**1/3**	**3/3**	**5/3**	**5/3**	2/9	0	0
0	0	0	**1/3**	**3/3**	**4/3**	**3/3**	**1/3**	0	0	0
0	0	1/9	**3/3**	**3/3**	**5/3**	**5/3**	**5/3**	2/9	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0

(1x3) x (3x1) Result

0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	1/9	4/9	4/9	4/9	3/9	3/9	2/9	0	0
0	0	1/9	**5/9**	**7/9**	**8/9**	**6/9**	**4/9**	2/9	0	0
0	0	2/9	**6/9**	**8/9**	**11/9**	**11/9**	**9/9**	4/9	0	0
0	0	1/9	**3/9**	**7/9**	**11/9**	**11/9**	**7/9**	2/9	0	0
0	0	2/9	**5/9**	**7/9**	**12/9**	**13/9**	**11/9**	4/9	0	0
0	0	1/9	**4/9**	**6/9**	**9/9**	**8/9**	**6/9**	2/9	0	0
0	0	1/9	3/9	3/9	5/9	5/9	5/9	2/9	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0

(3x3) Result

0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	1/9	4/9	4/9	4/9	3/9	3/9	2/9	0	0
0	0	1/9	**5/9**	**7/9**	**8/9**	**6/9**	**4/9**	2/9	0	0
0	0	2/9	**6/9**	**8/9**	**11/9**	**11/9**	**9/9**	4/9	0	0
0	0	1/9	**3/9**	**7/9**	**11/9**	**11/9**	**7/9**	2/9	0	0
0	0	2/9	**5/9**	**7/9**	**12/9**	**13/9**	**11/9**	4/9	0	0
0	0	1/9	**4/9**	**6/9**	**9/9**	**8/9**	**6/9**	2/9	0	0
0	0	1/9	3/9	3/9	5/9	5/9	5/9	2/9	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0

The resulting matrices are the same

Question 2

Using the method of normalized cross-correlation, find all instances of the letter “a” in the image “textsample.tif”. To do this, first extract a template subimage w(x,y) of an example of the letter “a”. Then match this template to the image (you can use OpenCV’s “matchTemplate” function). Find local peaks in the correlation scores image. Then threshold these peaks (you will have to experiment with the threshold) so that you get 1’s where there are “a”s and nowhere else. You can use OpenCV’s “connectedComponentsWithStats” function to extract the centroids of the peaks. Take the locations found and draw a box (or some type of marker) overlay on the original image showing the locations of the “a”s. Your program should also count the number of detected “a”s.

Solution

I first read in the textsample.tif and converted it to a binary image. I then tried to separate some of the letters and normalize the shapes of the "a"s with morphological operations. I then let the user choose a template or read in a template from file. A scores image is then produced. After this, I counted the "a"s on the page and decreased my threshold until I had the all of the "a"s and nothing else selected. This part only worked so well because I started to get other letters than "a" while still missing some "a"s. This might have been easier to resolve if the picture resolution was better. The smallest kernels I could use to clean up the text were still too large. Some of the "a"s are also boxed multiple times. When I did the final drawing on the image, I made sure that the current matches weren't on top of each other and decreased my count to correct for the duplicates. I know the know the final count is missing a few of the "a"s located at the very bottom of the photo.

"a"s Count: 50

In [42]:

```
import cv2
import numpy as np

def getUserTemplate(img):
    try:
        r = cv2.selectROI("ROI Template Selection", img)
        x, y, w, h = r
        # Crop image
        imCrop = img[int(r[1]):int(r[1]+r[3]), int(r[0]):int(r[0]+r[2])]
        original_with_ROI=cv2.rectangle(img=img, pt1=(x, y), pt2=(x + w, y + h), color=(0
, 0, 255), thickness=2)
    finally:
        cv2.destroyAllWindows()
        return original_with_ROI, imCrop, w, h

def thinning(img1):
    kernel = cv2.getStructuringElement(cv2.MORPH_RECT,(3,3))
    # Create an empty output image to hold values
    thin = np.zeros(thresh_img.shape,dtype='uint8')
    img1 = thresh_img
    # Loop until erosion leads to an empty set
    while (cv2.countNonZero(img1)!=0):
        # Erosion
        erode = cv2.erode(img1,kernel)
        # Opening on eroded image
        opening = cv2.morphologyEx(erode,cv2.MORPH_OPEN,kernel)
        # Subtract these two
        subset = erode - opening
        # Union of all previous sets
        thin = cv2.bitwise_or(subset,thin)
        # Set the eroded image for next iteration
        img1 = erode.copy()

    thin = ~thin
    return thin



import PIL.Image
from io import BytesIO

#Use 'jpeg' instead of 'png' (~5 times faster)
def array_to_image(img, fmt='jpeg',width=500):
    #Create binary stream object
    f = BytesIO()
    #Convert array to binary stream object
    new_p = PIL.Image.fromarray(img)
    if new_p.mode != 'RGB':
        new_p = new_p.convert('RGB')
    new_p.save(f, fmt)
    return IPython.display.Image(data=f.getvalue(), width=width)



import IPython.display
import time
```

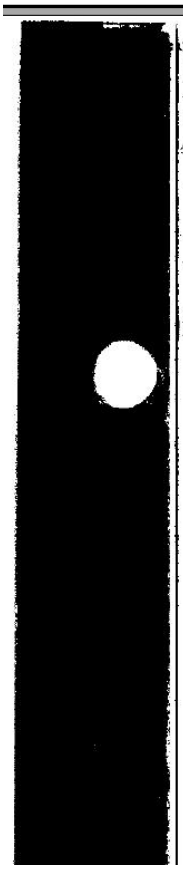
In [47]:

```
d1 = IPython.display.display("", display_id=1)
d4 = IPython.display.display("", display_id=4)
try:
    original_img = cv2.imread("textsample.tif")
    gray_img = cv2.cvtColor(original_img.astype(np.float32), cv2.COLOR_BGR2GRAY)
    _, thresh_img = cv2.threshold(gray_img, thresh=.7, maxval=255, type=cv2.THRESH_BINARY
)
#     thresh_img = cv2.adaptiveThreshold(src=original_img,maxValue=255, adaptiveMethod=cv
2.ADAPTIVE_THRESH_MEAN_C,thresholdType=cv2.THRESH_BINARY,blockSize=5,C=-10)

    kernel = cv2.getStructuringElement(cv2.MORPH_RECT,(2,1))
    kernel2 = cv2.getStructuringElement(cv2.MORPH_CROSS,(2,2))
    kernel3 = cv2.getStructuringElement(cv2.MORPH_RECT,(1,3))
    kernel4 = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(3,1))
    thin = cv2.dilate(thresh_img,kernel,3)
    thin = cv2.erode(thin,kernel2,5)
    thin = cv2.dilate(thresh_img,kernel4,1)
    thin = cv2.erode(thin,kernel3,2)
    d4.update(array_to_image(img=thin, fmt='jpeg', width=500))

    template = cv2.imread("template1.jpg",0)
    user_select = False

    if (template.shape[0] != 0 and not user_select):
        template_width = template.shape[0]
        template_height = template.shape[1]
    else:
        template_img, template, template_width, template_height = getUserTemplate(thin)
        cv2.imwrite("template.jpg", template)

finally:
    d1.update(array_to_image(img=template, fmt='jpeg', width=100))
    cv2.destroyAllWindows()
#     IPython.display.clear_output()
```

noticeable, was no longer constant (i.e., c) but increased (i.e., cS) with the base level, S. Brentano's function remains dependent on the Weber fraction, k, however. For Brentano, just as for Fechner (equation 3), the larger the value of k (the poorer the discriminability or resolving power on the particular modality), the slower the rise in subjective magnitude, S, with physical magnitude, I. Since it can readily be rewritten in logarithmic or relative units, as in equation 4, the power function can express ratio invariances even if it seems to do so less simply than does Fechner's logarithmic function: therefore, it, too, is consistent with the perceptual constancies (Yilmaz 1967). It may seem odd and pointless for the system to add expansivity and thus to undo part or all of the compressiveness inherent in Weber's law, but by doing so, it does not lose access to the ratio invariances, and possibly transforms them into a more usable form.

In [48]:

```
d2 = IPython.display.display("", display_id=2)
d3 = IPython.display.display("", display_id=3)
d5 = IPython.display.display("", display_id=5)

scores = cv2.matchTemplate(thin.astype(np.float32), template.astype(np.float32), method=c
v2.TM_CCOEFF_NORMED).astype(np.float32)
d2.update(array_to_image(img=scores, fmt='jpeg', width=800))

min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(scores)
threshold = 0.2
while (threshold == 0.2 or matches[0].shape[0] >= 105):
    matches = np.where(scores.astype(np.float32) >= max_val-threshold)
    threshold -= 0.0001
    d5.update(IPython.display.HTML(f"""<h1>Threshold:\t{np.round(1-threshold,5)}</h1>
                                       <h1>Current Match Count:\t{matches[0].shape[0]}</
h1>
                                   """))

last_bound_x = 0
last_bound_y = 0
matching = matches[0].shape[0]
for match in range(matches[0].shape[0]):
    x = matches[0][match]
    y = matches[1][match]
    if (x > last_bound_x+template_width/2 or y > last_bound_y+template_height/2):
        original_img = cv2.rectangle(img=original_img, pt1=(y,x), pt2=(y+template_height,
x+template_width), color=(0, 255, 255), thickness=3)
        last_bound_x = x
        last_bound_y = y
    else:
        matching -= 1
        d5.update(IPython.display.HTML(f"""<h1>Threshold:\t{np.round(1-threshold,5)}</h1>
                                       <h1>Current Match Count:\t{matching}</h1>
                                   """))
d3.update(array_to_image(img=original_img, fmt='jpeg', width=800))
```



noticeable, was no longer constant (i.e., c) but increased (i.e., cS) with the base level, S. Brentano's function remains dependent on the Weber fraction, k, however. For Brentano, just as for Fechner (equation 3), the larger the value of k (the poorer the discriminability or resolving power on the particular modality), the slower the rise in subjective magnitude, S, with physical magnitude, I. Since it can readily be rewritten in logarithmic or relative units, as in equation 4, the power function can express ratio invariances even if it seems to do so less simply than does Fechner's logarithmic function: therefore, it, too, is consistent with the perceptual constancies (Yilmaz 1967). It may seem odd and pointless for the system to add expansivity and thus to undo part or all of the compressiveness inherent in Weber's law, but by doing so, it does not lose access to the ratio invariances, and possibly transforms them into a more usable form.

Threshold: 0.8661

Current Match Count: 50

In [49]:
```
cv2.imwrite("as.jpg", original_img)
```

Out[49]: True

Question 3

You are given the binary image A and the structuring element B below (assume the origin of B is its center).

B

1	**1**	**1**

A

0	0	0	0	0	0	0	0	0	0
0	**1**	**1**	**1**	0	0	0	0	0	0
0	**1**	**1**	**1**	**1**	0	0	**1**	0	0
0	1	0	0	0	**1**	**1**	**1**	**1**	0
0	0	0	0	0	**1**	**1**	**1**	**1**	0
0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	0
0	**1**	**1**	**1**	0	**1**	**1**	**1**	**1**	0
0	0	0	0	0	**1**	0	0	0	0
0	0	0	**1**	**1**	**1**	**1**	**1**	**1**	0
0	0	**1**	**1**	**1**	**1**	**1**	**1**	0	0
0	0	0	0	0	0	0	0	0	0

4 Connected Components

0	0	0	0	0	0	0	0	0	0
0	**1**	**1**	**1**	0	0	0	0	0	0
0	**1**	**1**	**1**	**1**	0	0	**2**	0	0
0	**1**	0	0	0	**2**	**2**	**2**	**2**	0
0	0	0	0	0	**2**	**2**	**2**	**2**	0
0	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	0
0	**2**	**2**	**2**	0	**2**	**2**	**2**	**2**	0
0	0	0	0	0	**2**	0	0	0	0
0	0	0	**2**	**2**	**2**	**2**	**2**	**2**	0
0	0	**2**	**2**	**2**	**2**	**2**	**2**	0	0
0	0	0	0	0	0	0	0	0	0

For Opening (Erosion then Dilation)

Erosion

0	0	0	0	0	0	0	0	0	0
0	0	**1**	0	0	0	0	0	0	0
0	0	**1**	**1**	0	0	0	0	0	0
0	0	0	0	0	0	**1**	**1**	0	0
0	0	0	0	0	0	**1**	**1**	0	0
0	0	**1**	**1**	**1**	**1**	**1**	**1**	0	0
0	0	**1**	0	0	0	**1**	**1**	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	**1**	**1**	**1**	**1**	0	0
0	0	0	**1**	**1**	**1**	**1**	0	0	0
0	0	0	0	0	0	0	0	0	0

Final

Dilation

0	0	0	0	0	0	0	0	0	0
0	**1**	**1**	**1**	0	0	0	0	0	0
0	**1**	**1**	**1**	**1**	0	0	0	0	0
0	0	0	0	0	**1**	**1**	**1**	**1**	0
0	0	0	0	0	**1**	**1**	**1**	**1**	0
0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	0
0	**1**	**1**	**1**	0	**1**	**1**	**1**	**1**	0
0	0	0	0	0	0	0	0	0	0
0	0	0	1	**1**	**1**	**1**	**1**	1	0
0	0	1	**1**	**1**	**1**	**1**	1	0	0
0	0	0	0	0	0	0	0	0	0

Question 4

Read and display each image of the video, and print the frame number on the displayed image. Write a program that finds the five CCCs in each image, and marks each one with a cross hair or rectangle. Create an output video of your results and post it on YouTube.

Solution

Solution Video (https://youtu.be/8CRx5BAlzTg)

I started by taking the first frame of the video and thresholding the image using HSV thresholds. The image is then checked for the outer circle components and the inverse for the inner cicle components. If the inner was inside the outer and centers of the circles were within a threshold of 3px (only needs to be 1px though). For each match, I created a CCC class object. The object stores the previous position, and last position, area, last area, and the stats gathered from the components. Each is also initialized with a color (For EXTRA CREDIT) instead of a number to draw.

After the CCCs are created, the video repeats the image processing done to the first frame for all the rest. In the following loop for the rest of the video, after identifying two centroids that are close together, this is pasted to the CCC compare function that checks to see if it fits the thresholds of the previous size and position. If it matches, the attributes are updated and a box is drawn by call the drawBoundingBox method. The method gets the information of where to draw from the object stats that were just updated. The CCC's bounding box is drawn in the previous position if the position isn't updated (e.g. doesn't fit the thresholds). Finally, a video writer saves the frame.

In [39]:

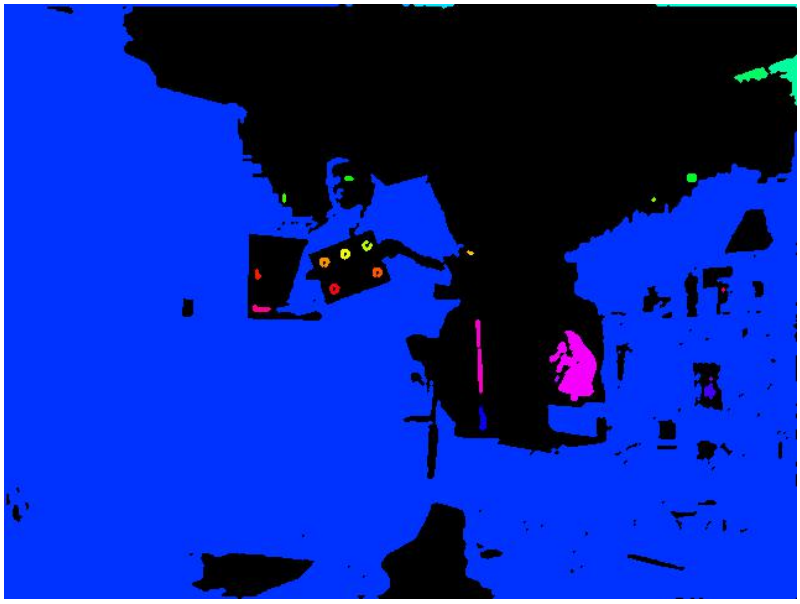
```
import cv2
import time
import PIL.Image
from io import BytesIO
import IPython.display
import ipywidgets as widgets
import numpy as np

#Use 'jpeg' instead of 'png' (~5 times faster)
def array_to_image(a, fmt='jpeg'):
    #Create binary stream object
    f = BytesIO()
    #Convert array to binary stream object
    PIL.Image.fromarray(a).save(f, fmt)
    return IPython.display.Image(data=f.getvalue(), width=400)

def color_components(labels):
    # Map component labels to hue val
    label_hue = np.uint8(179*labels/np.max(labels))
    blank_ch = 255*np.ones_like(label_hue)
    labeled_img = cv2.merge([label_hue, blank_ch, blank_ch])

    # cvt to BGR for display
    labeled_img = cv2.cvtColor(labeled_img, cv2.COLOR_HSV2BGR)

    # set bg label to black
    labeled_img[label_hue==0] = 0
    return labeled_img

def nothing(x):
    pass

def initializeCCCs():
    try:
        original_img = cv2.imread('frame1.jpg')
        thresh_img = processFrame(original_img)
        colors = [(0,0,255), (255,0,0), (0,255,0), (255, 255, 0), (0, 255,255)]
    finally:
        kernelClose = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (6, 6))
        kernelOpen = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (4, 4))
        kernelDilate = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (4, 4))
        kernelErode = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (3, 3))

        innerComponents = cv2.dilate(cv2.erode(~thresh_img, kernelErode),kernelDilate)

        filtered_img_close = cv2.morphologyEx(thresh_img, cv2.MORPH_CLOSE, kernelClose)
        filtered_img_open = cv2.morphologyEx(filtered_img_close, cv2.MORPH_OPEN, kernelOp
en,10)
        outerComponents = filtered_img_open

        num_labels_inner, labels_im_inner, innerStats, innerCentroids = cv2.connectedComp
onentsWithStats(innerComponents, connectivity=8)
        color_labeled_img_inner = color_components(labels_im_inner)
        d5.update(array_to_image(color_labeled_img_inner))

        num_labels_outer, labels_im_outer, outerStats, outerCentroids = cv2.connectedComp
onentsWithStats(outerComponents, connectivity=8)
        color_labeled_img_outer = color_components(labels_im_outer)
        d6.update(array_to_image(color_labeled_img_outer))

        CCCs = []
```

```
        ccc_counter = 0
        for i, i_centroid in enumerate(innerCentroids):
            if (i == 0):
                continue
            for j, o_centroid in enumerate(outerCentroids):
                if (j == 0):
                    continue
                xi = int(i_centroid[0])
                yi = int(i_centroid[1])
                xo = int(o_centroid[0])
                yo = int(o_centroid[1])
                if(np.abs(xi - xo) < 3):
                    if(np.abs(yi - yo) < 3):
                        ccc_counter += 1
                        ccc = CCC(str(ccc_counter), xi, yi, xo, yo, outerStats[j],colors[
ccc_counter-1])
                        original_img = ccc.drawBoundingBox(original_img)
                        CCCs.append(ccc)

    return CCCs

def find_nearest(array, value):
    array = np.asarray(array)
    idx = (np.abs(array - value)).argmin()
    return array[idx]

def processFrame(original_img):
    low_thresholds = [0, 0, 0]
    high_thresholds = [255, 255, 150]

    image_height = original_img.shape[0]
    image_width = original_img.shape[1]

    # Convert BGR to HSV.
    hsv_img = cv2.cvtColor(original_img, cv2.COLOR_BGR2HSV)# Split into the different ban
ds.
    planes = cv2.split(hsv_img)

    # Create output thresholded image.
    thresh_img = np.full((image_height, image_width), 255, dtype=np.uint8)

    for i in range(3):
        _,low_img = cv2.threshold(planes[i], low_thresholds[i], 255, cv2.THRESH_BINARY)
        _,high_img = cv2.threshold(planes[i], high_thresholds[i], 255, cv2.THRESH_BINARY_
INV)
        thresh_band_img = cv2.bitwise_and(low_img, high_img)
        thresh_img = cv2.bitwise_and(thresh_img, thresh_band_img)

    return thresh_img

class CCC:
    def __init__(self, name, x_inner, y_inner, x_outer, y_outer, stats, color):
        self.name = name
        self.x_inner = x_inner
        self.y_inner = y_inner
        self.previous_x_inner = x_inner
        self.previous_y_inner = y_inner
        self.x_outer = x_outer
        self.y_outer = y_outer
        self.previous_x_outer = x_outer
        self.previous_y_outer = y_outer
        self.stats = stats
```

```
        self.area = stats[4]
        self.previous_area = stats[4]
        self.color = color
        self.updated = False

    def compare(self, xi, yi, xo, yo, area):
        if (np.abs(self.x_inner-xi) > 10):
            return False
        if (np.abs(self.x_outer-xo) > 10):
            return False
        if (np.abs(self.y_inner - yi) > 10):
            return False
        if (np.abs(self.y_outer - yo) > 10):
            return False
        if (np.abs(self.area - area) > 150):
            return False
        return True

    def update(self, x_inner, y_inner, x_outer, y_outer, stats):
        self.previous_x_inner = self.x_inner
        self.previous_y_inner = self.y_inner
        self.previous_x_outer = self.x_outer
        self.previous_y_outer = self.y_outer
        self.previous_area = self.area
        self.x_inner = x_inner
        self.y_inner = y_inner

        self.x_outer = x_outer
        self.y_outer = y_outer


        try:
            self.stats = stats
            self.area = stats[4]
        except:
            pass
        self.updated = True

    def drawBoundingBox(self, img):
#     img = cv2.putText(img, text=str(self.name), org=(50, 50), fontFace=cv2.FONT_HERSHEY
_SIMPLEX,
#                       fontScale=1.5, color=(0, 255, 0))
        return cv2.rectangle(img=img,
                             pt1=(self.stats[0], self.stats[1]),
                             pt2=(self.stats[0]+self.stats[2], self.stats[1] + self.stats
[3]),
                             color=self.color, thickness=2)
```

In [40]:
```
d5 = IPython.display.display("", display_id=8)
d6 = IPython.display.display("", display_id=6)
d7 = IPython.display.display("", display_id=7)
try:
    video_name = "fiveCCC.avi"
    cam = cv2.VideoCapture(video_name)

    fps = 30
    frame = 0
    total_frames = int(cam.get(cv2.CAP_PROP_FRAME_COUNT))


    # Create output movie file.
    video_output_name = video_name[0:video_name.find(".")]+"_output"+".avi"
    fourcc = cv2.VideoWriter_fourcc('M', 'J', 'P', 'G')
    videoWriter = cv2.VideoWriter(apiPreference=cv2.CAP_FFMPEG, filename=video_output_nam
e, fourcc=fourcc, fps=30,
                                  frameSize=(int(cam.get(3)), int(cam.get(4))))

    CCCs = initializeCCCs()
    while True:
        try:
            t1 = time.time()
            # Try to read in video
            got_image, original_img = cam.read()
            if not got_image:
                break # break if no video
            frame += 1

            for ccc in CCCs:
                ccc.updated = False


            thresh_img = processFrame(original_img)
            kernelClose = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (2, 2))
            kernelOpen = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (3, 3))
            kernelDilate = cv2.getStructuringElement(cv2.MORPH_CROSS, (2, 2))
            kernelErode = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (2, 2))

            innerComponents = cv2.dilate(cv2.erode(~thresh_img, kernelErode),kernelDilate
)


            filtered_img_close = cv2.morphologyEx(thresh_img, cv2.MORPH_CLOSE, kernelClos
e)
            filtered_img_open = cv2.morphologyEx(filtered_img_close, cv2.MORPH_OPEN, kern
elOpen,10)
            outerComponents = filtered_img_open

            num_labels_inner, labels_im_inner, innerStats, innerCentroids = cv2.connected
ComponentsWithStats(innerComponents, connectivity=8)
            color_labeled_img_inner = color_components(labels_im_inner)
            d5.update(array_to_image(color_labeled_img_inner))

            num_labels_outer, labels_im_outer, outerStats, outerCentroids = cv2.connected
ComponentsWithStats(outerComponents, connectivity=8)
            color_labeled_img_outer = color_components(labels_im_outer)
            d6.update(array_to_image(color_labeled_img_outer))

            not_updated = []

            for i, i_centroid in enumerate(innerCentroids):
```

```
                if (i == 0):
                    continue
                for j, o_centroid in enumerate(outerCentroids):
                    if (j == 0):
                        continue
                    xi = int(i_centroid[0])
                    yi = int(i_centroid[1])
                    xo = int(o_centroid[0])
                    yo = int(o_centroid[1])
                    if(np.abs(xi - xo) < 3):
                        if(np.abs(yi - yo) < 3):
                            not_updated.append([[xi],[yi],[xo],[yo]])
                            for ccc in CCCs:
                                if (ccc.compare(xi, yi, xo, yo, outerStats[j][4])):
                                    ccc.update(xi, yi, xo, yo, outerStats[j])
                                    original_img = ccc.drawBoundingBox(original_img)
                                    break
                            else:
                                not_updated.pop()

            for ccc in CCCs:
                if (not ccc.updated):
                    ccc.drawBoundingBox(original_img)

            original_img = cv2.putText(original_img, text=str(frame), org=(50, 50), fontF
ace=cv2.FONT_HERSHEY_SIMPLEX,
                fontScale=1.5, color=(0, 255, 255), thickness=3)

            videoWriter.write(original_img)

            # Jupyter Video output display
#             thresh_img = cv2.cvtColor(thresh_img, cv2.COLOR_BGR2RGB)
            while time.time() < t1 + (1/(fps+ 1)):
                continue
            t2 = time.time()

            s = f"""<h1>{int(1/(t2-t1))} FPS</h1>"""
#                 d3.update( IPython.display.HTML(s) )
            d7.update(array_to_image(original_img))

        except KeyboardInterrupt:
            print ("Stream stopped by Keyboard")
            break

finally:
    cam.release()
    videoWriter.release()
    cv2.destroyAllWindows()
    print("Output Video Saved")
    print("End of Video")
```




```
Output Video Saved
End of Video
```

In [ ]: