Name : Fernando I.A.M.D.

Index No.: 190172K

EN2550: Assignment 03 on Object Counting on a Conveyor Belt

Connected Component Analysis

In this part, we will generate an indexed image representing connected components in `conveyor_f101.png` image. Notice that, as there are three square nuts and one hexagonal nut in the image, there will be five connected components (backgound will be assigned the label 0).

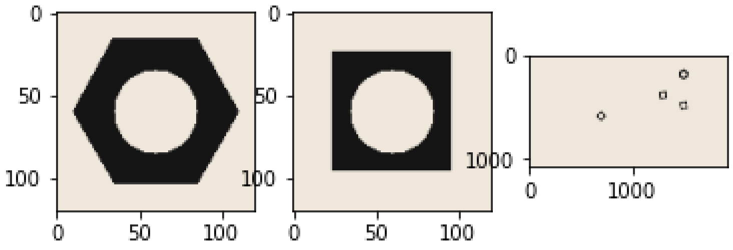
1. Open the `hexnut_template.png`, `squarenut_template.png` and `conveyor_f100.png` and display. This is done for you.

In [ ]:

```python
import cv2
import numpy as np
import sympy
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
from plyfile import PlyData,PlyElement
%matplotlib inline
```

In [ ]:

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

hexnut_template = cv.imread('hexnut_template.png', cv.IMREAD_COLOR)
squarenut_template =  cv.imread('squarenut_template.png', cv.IMREAD_COLOR)
conveyor_f100 =  cv.imread('conveyor_f100.png', cv.IMREAD_COLOR)

fig, ax = plt. subplots(1,3)
ax[0].imshow(cv.cvtColor(hexnut_template, cv.COLOR_RGB2BGR))
ax[1].imshow(cv.cvtColor(squarenut_template, cv.COLOR_RGB2BGR))
ax[2].imshow(cv.cvtColor(conveyor_f100, cv.COLOR_RGB2BGR))
plt.show()
```

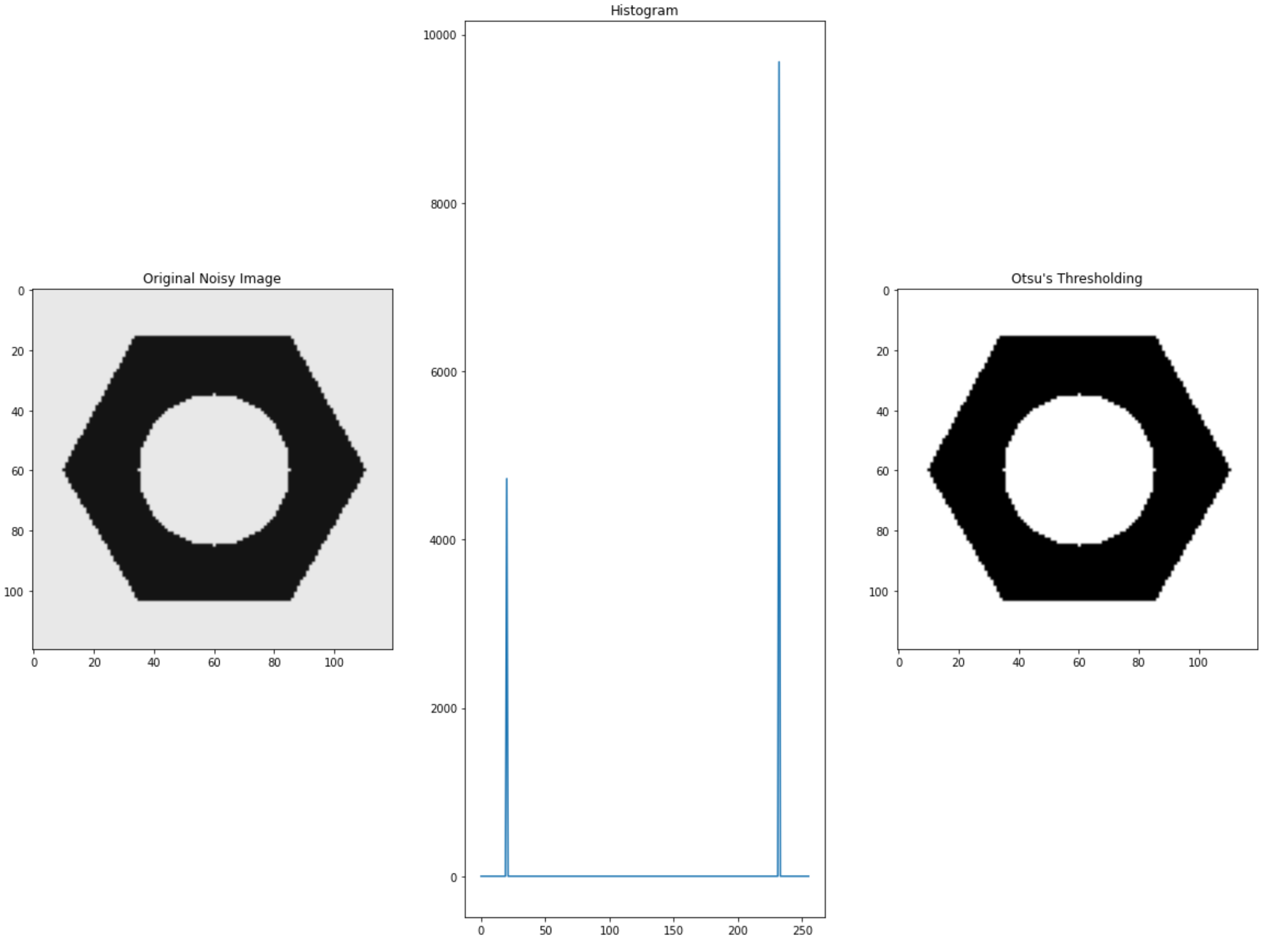


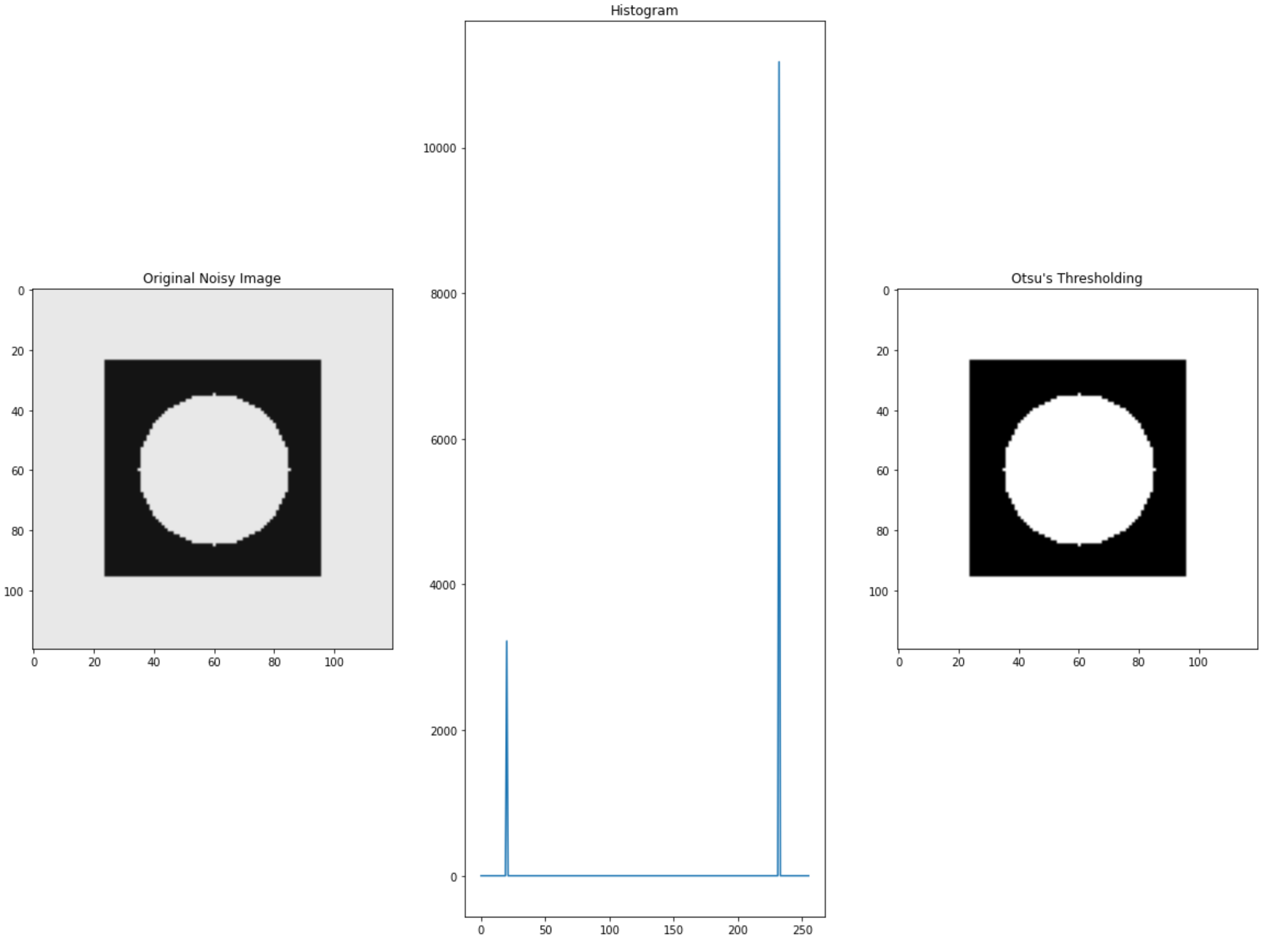
1. Convert the images to grayscale and apply Otsu's thresholding to obtain the binarized image. Do this for both the templates and belt images. See https://docs.opencv.org/master/d7/d4d/tutorial_py_thresholding.html for a guide. State the threshold value (automatically) selected in the operation. Display the output images.

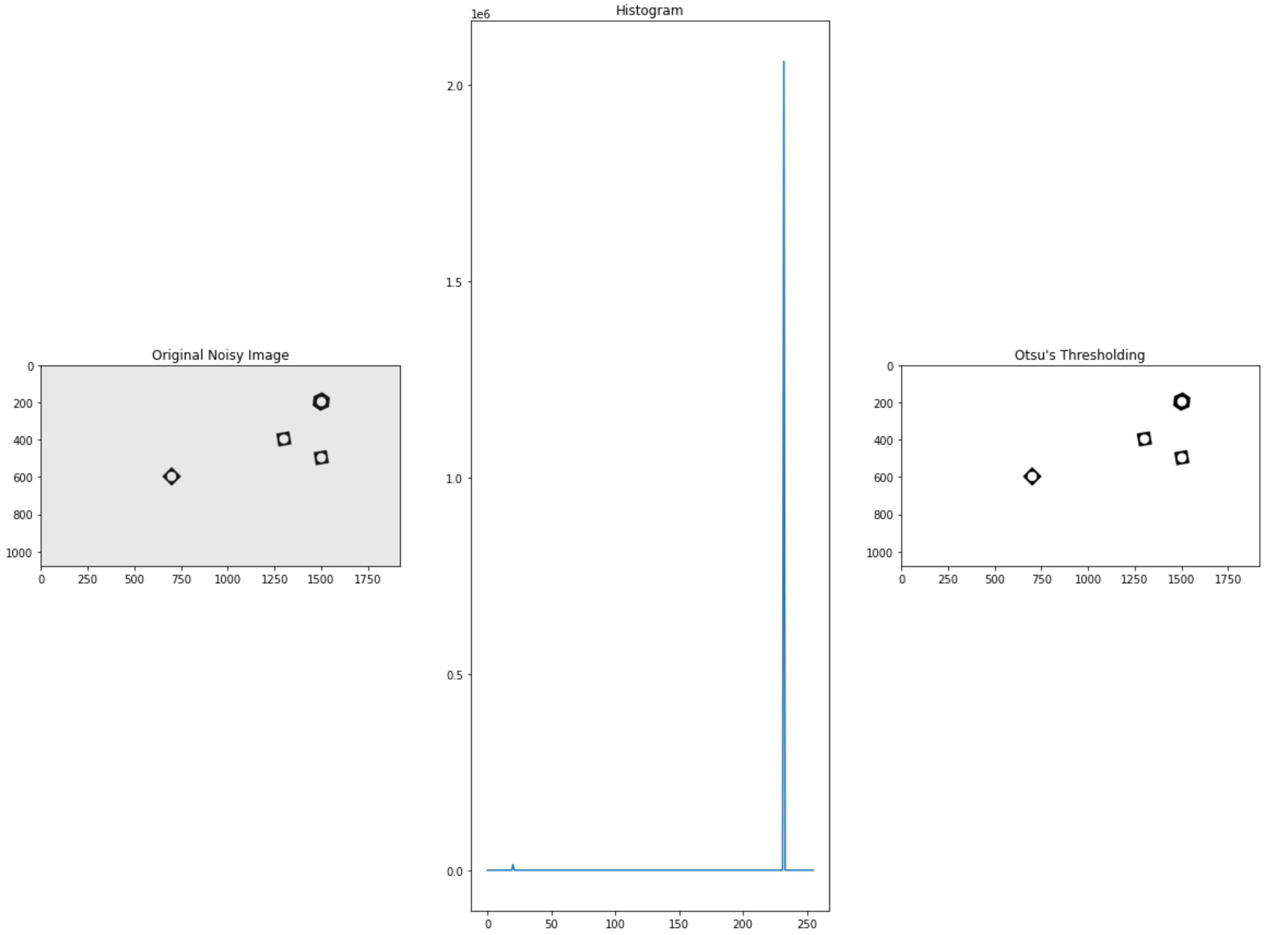
In [ ]:

```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt

img1 = cv.imread('hexnut_template.png',cv.IMREAD_GRAYSCALE)
hist_f1 = cv2.calcHist([img1],[0],None,[256],[0,256])

img2 = cv.imread('squarenut_template.png',cv.IMREAD_GRAYSCALE)
hist_f2 = cv2.calcHist([img2],[0],None,[256],[0,256])

img3 = cv.imread('conveyor_f100.png',cv.IMREAD_GRAYSCALE)
hist_f3 = cv2.calcHist([img3],[0],None,[256],[0,256])

# Otsu's thresholding
ret1,th1 = cv.threshold(img1,0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)
print("Automatically Selected Threshold value for hexnut_template =",ret1)

ret2,th2 = cv.threshold(img2,0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)
print("Automatically Selected Threshold value for squarenut_template =",ret2)

ret3,th3 = cv.threshold(img3,0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)
print("Automatically Selected Threshold value for conveyor_f100 =",ret3)

fig, ax = plt.subplots(1,3,figsize= (20,15))
ax[0].imshow(img1,cmap='gray',vmin=0,vmax=255)
ax[0].set_title('Original Noisy Image')
ax[1].plot(hist_f1)
ax[1].set_title('Histogram')
ax[2].imshow(th1,cmap='gray',vmin=0,vmax=255)
ax[2].set_title("Otsu's Thresholding")

fig, ax = plt.subplots(1,3,figsize= (20,15))
ax[0].imshow(img2,cmap='gray',vmin=0,vmax=255)
ax[0].set_title('Original Noisy Image')
ax[1].plot(hist_f2)
ax[1].set_title('Histogram')
ax[2].imshow(th2,cmap='gray',vmin=0,vmax=255)
ax[2].set_title("Otsu's Thresholding")

fig, ax = plt.subplots(1,3,figsize= (20,15))
ax[0].imshow(img3,cmap='gray',vmin=0,vmax=255)
ax[0].set_title('Original Noisy Image')
ax[1].plot(hist_f3)
ax[1].set_title('Histogram')
ax[2].imshow(th3,cmap='gray',vmin=0,vmax=255)
ax[2].set_title("Otsu's Thresholding")
plt.show()
```

```
Automatically Selected Threshold value for hexnut_template = 20.0
Automatically Selected Threshold value for squarenut_template = 20.0
Automatically Selected Threshold value for conveyor_f100 = 20.0
```


Original Noisy Image
0
20
40
60
80
100
0
20
40
60
80
100
Histogram
10000
8000
6000
4000
2000
0
0
50
100
150
200
250
Otsu's Thresholding
0
20
40
60
80
100
0
20
40
60
80
100


Original Noisy Image
0
20
40
60
80
100
0
20
40
60
80
100
Histogram
10000
8000
6000
4000
2000
0
0
50
100
150
200
250
Otsu's Thresholding
0
20
40
60
80
100
0
20
40
60
80
100

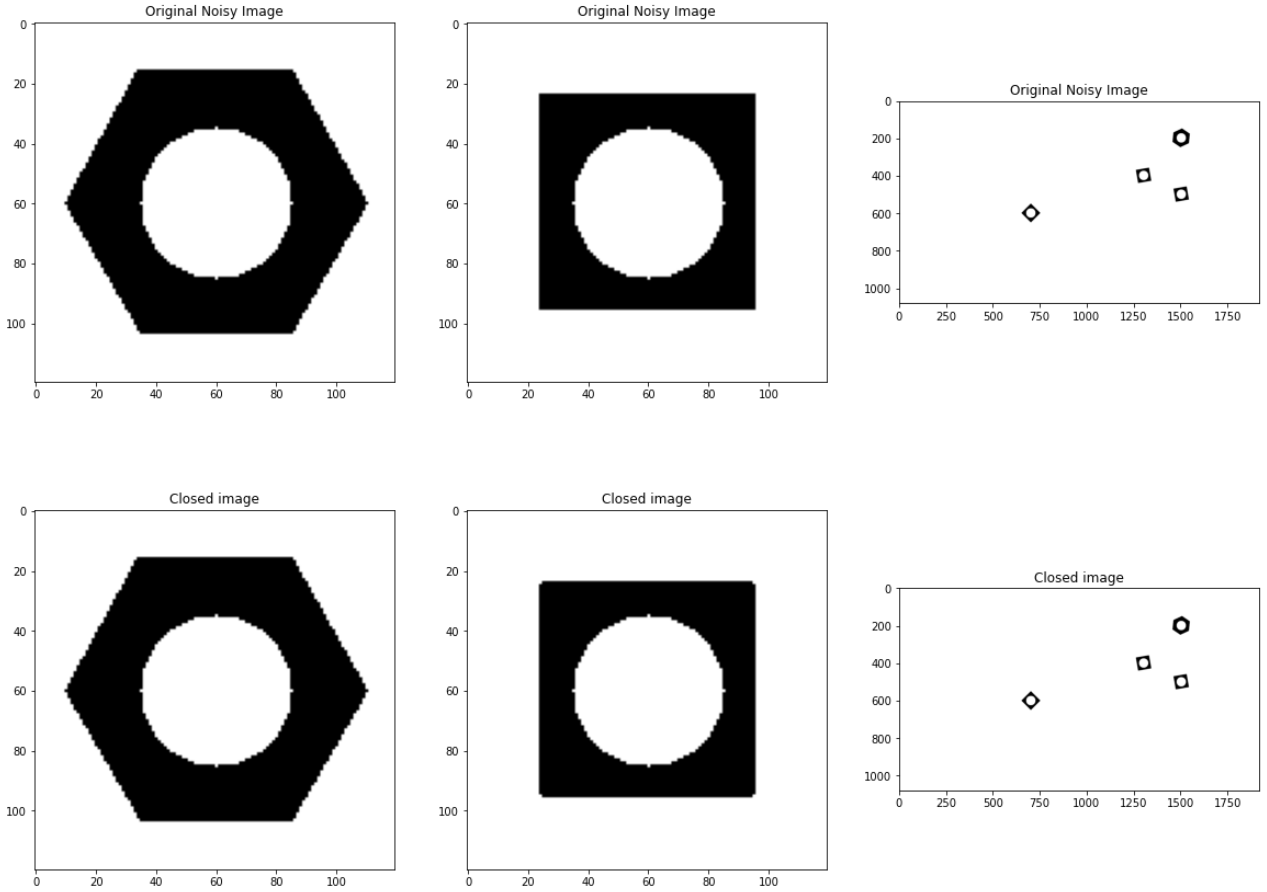


1. Carry out morphological closing to remove small holes inside the foreground. Use a $3 \\times 3$ kernel. See https://docs.opencv.org/master/d9/d61/tutorial_py_morphological_ops.html for a guide.

In [ ]:

```
kernel = np.array([[0,1,0],
                   [1,1,1],
                   [0,1,0]])
kernel = kernel.astype('uint8')

closing1 = cv.morphologyEx(th1, cv.MORPH_CLOSE, kernel)
closing2 = cv.morphologyEx(th2, cv.MORPH_CLOSE, kernel)
closing3 = cv.morphologyEx(th3, cv.MORPH_CLOSE, kernel)

fig, ax = plt.subplots(2,3,figsize= (20,15))
ax[0][0].imshow(th1,cmap='gray',vmin=0,vmax=255)
ax[0][0].set_title('Original Noisy Image')
ax[1][0].imshow(closing1,cmap='gray',vmin=0,vmax=255)
ax[1][0].set_title('Closed image')

ax[0][1].imshow(th2,cmap='gray',vmin=0,vmax=255)
ax[0][1].set_title('Original Noisy Image')
ax[1][1].imshow(closing2,cmap='gray',vmin=0,vmax=255)
ax[1][1].set_title('Closed image')

ax[0][2].imshow(th3,cmap='gray',vmin=0,vmax=255)
ax[0][2].set_title('Original Noisy Image')
ax[1][2].imshow(closing3,cmap='gray',vmin=0,vmax=255)
```

```python
ax[1][2].set_title('Closed image')
plt.show()
```



1. Connected components analysis: apply the `connectedComponentsWithStats` function (see https://docs.opencv.org/4.5.5/d3/dc0/group__imgproc__shape.html#ga107a78bf7cd25dec05fb4dfc and display the outputs as colormapped images. Answer the following questions

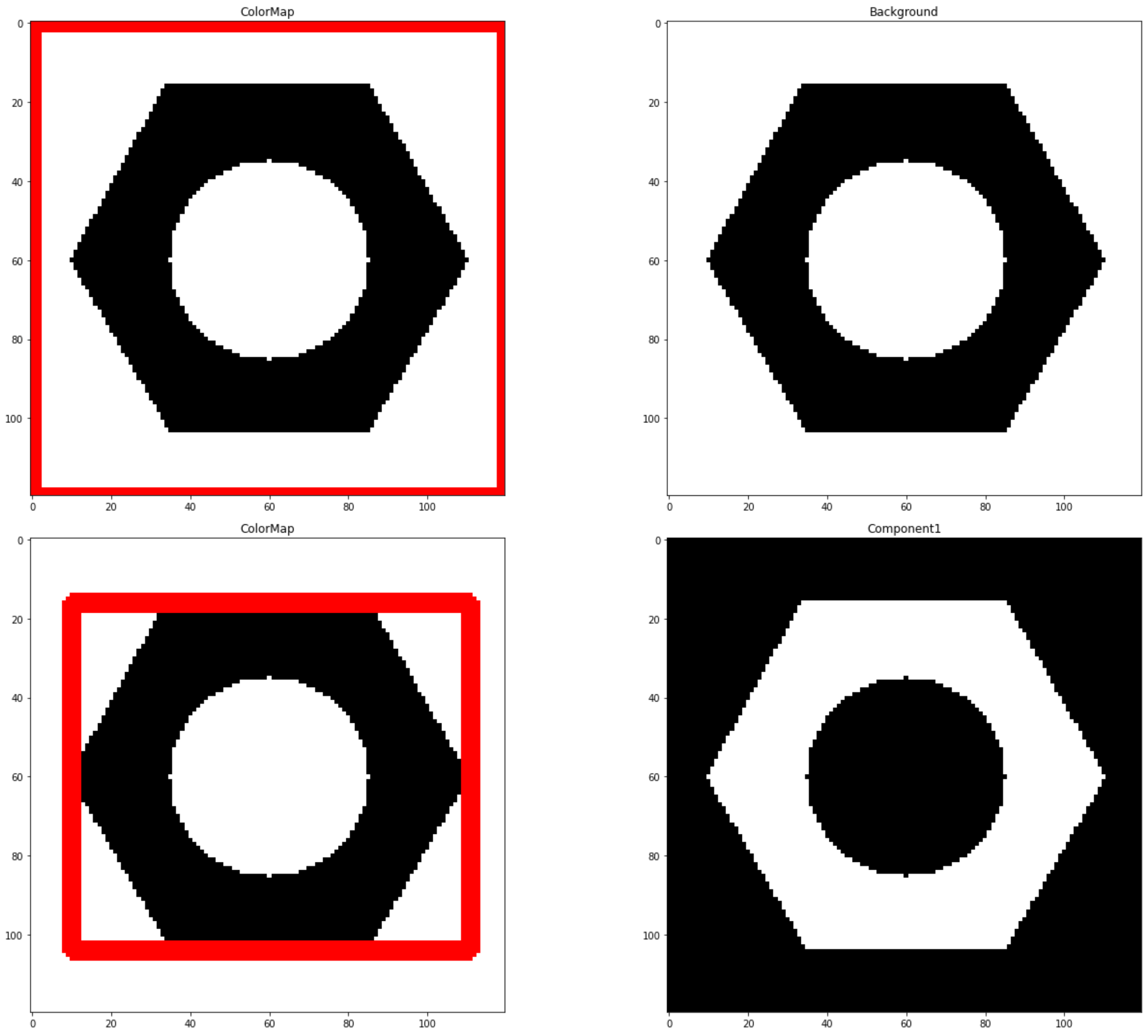
- How many connected components are detected in each image?
- What are the statistics? Interpret these statistics.
- What are the centroids?

For the hexnut template, you should get the object area in pixel as approximately `4728`.

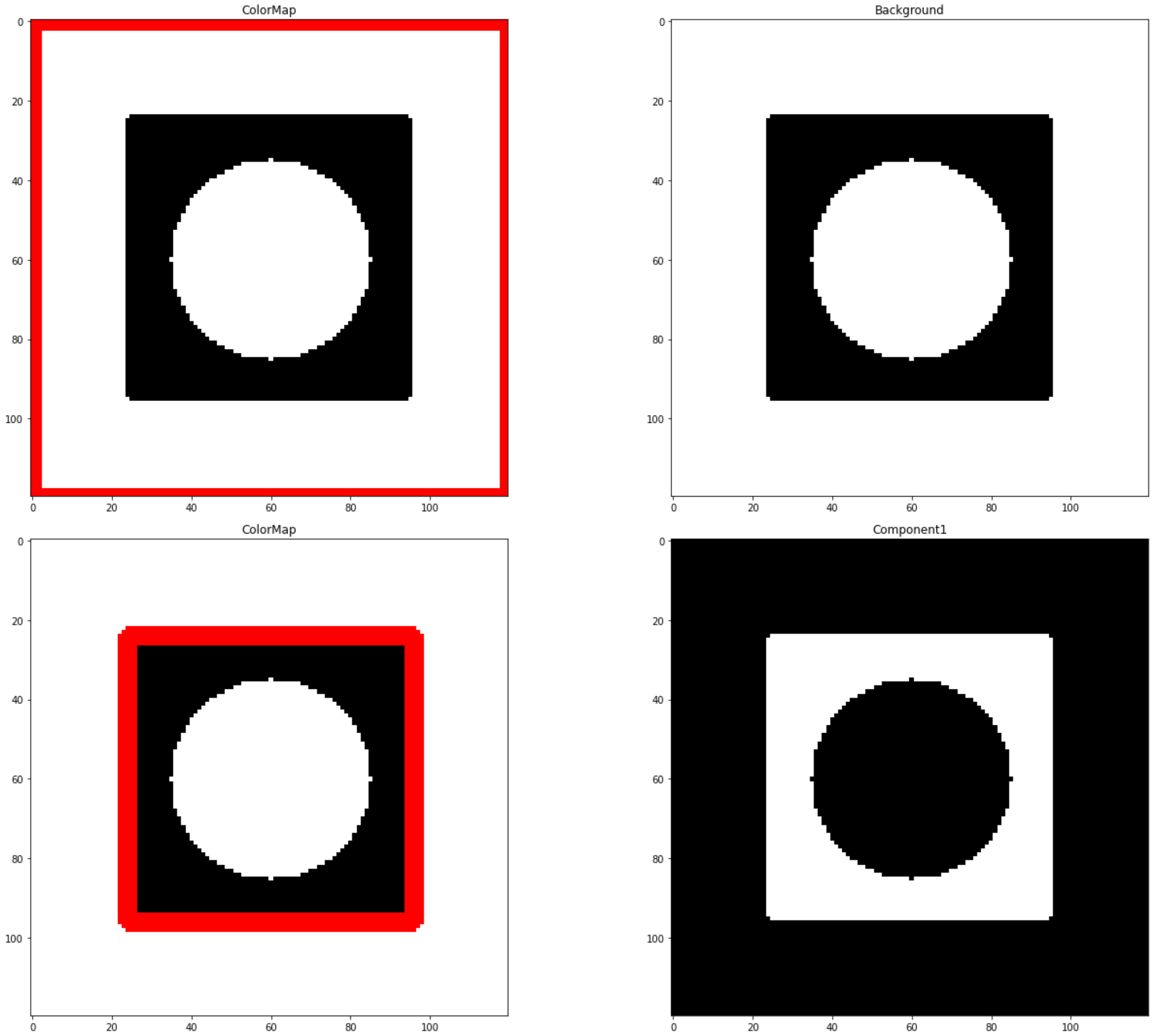
In [ ]:

```python
connectivity = 4
components = ["hexnut_template","squarenut_template","conveyor_f100"]
closed_images = [closing1,closing2,closing3]

for j in range(len(closed_images)):
    invert = cv.bitwise_not(closed_images[j])
    output = cv2.connectedComponentsWithStats(invert, connectivity, cv2.CV_32S)
    (numLabels, labels, stats, centroids) = output

    fig, ax = plt.subplots(numLabels,2,figsize= (20,15))
    print("----",components[j],"---")
    #Number of connected components
    print("Number of connected components =",numLabels)
    for i in range(numLabels):
        mask = np.zeros(closed_images[j].shape, dtype="uint8")
        #stats
        x = stats[i, cv2.CC_STAT_LEFT]
        y = stats[i, cv2.CC_STAT_TOP]
        w = stats[i, cv2.CC_STAT_WIDTH]
        h = stats[i, cv2.CC_STAT_HEIGHT]
        area = stats[i, cv2.CC_STAT_AREA]

        img = cv.cvtColor(closed_images[j] , cv.COLOR_GRAY2BGR)
        componentMask = (labels == i).astype("uint8") * 255
        mask = cv2.bitwise_or(mask, componentMask)
        cv.rectangle(img, (x, y), (x + w, y + h), (255, 0, 0), 3)

        ax[i][0].imshow(img)
        ax[i][0].set_title("ColorMap")
        ax[i][1].imshow(mask,cmap='gray')
        if(i==0):
            print("--Background--")
            ax[i][1].set_title("Background")
        else:
            print("--Component{num}--".format(num=i))
            ax[i][1].set_title('Component{label}'.format(label=i))
        print("     left most coordinate =",x,"|","top most coordinate =",y,"|","horizo
        print("     Area =",area)

        #centroid
        cx1,cy1 = centroids[i,0],centroids[i,1]
        print("     Centroid Coordinates are =",(cx1,cy1))
    fig.tight_layout()
    plt.show()
```

```
---- hexnut_template ---
Number of connected components = 2
--Background--
     left most coordinate = 0 | top most coordinate = 0 | horizontal size = 120 | vertic
al size = 120
     Area = 9676
     Centroid Coordinates are = (59.33712277800744, 59.63528317486565)
--Component1--
     left most coordinate = 10 | top most coordinate = 16 | horizontal size = 101 | vert
ical size = 88
     Area = 4724
     Centroid Coordinates are = (59.83361558001693, 59.22290431837426)
```

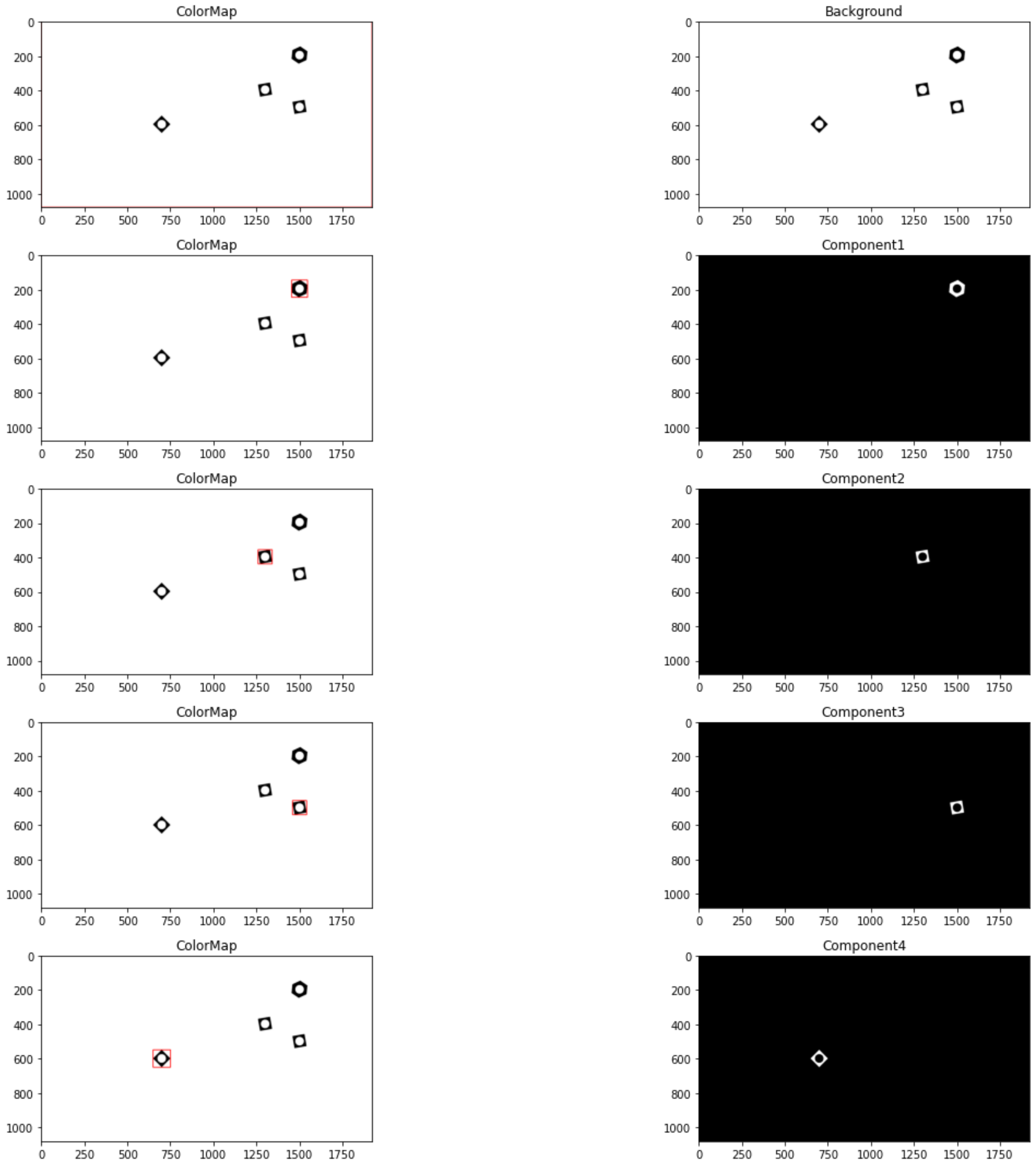


```
---- squarenut_template ---
Number of connected components = 2
--Background--
     left most coordinate = 0 | top most coordinate = 0 | horizontal size = 120 | vertic
al size = 120
     Area = 11181
     Centroid Coordinates are = (59.58769340846078, 59.58769340846078)
--Component1--
     left most coordinate = 24 | top most coordinate = 24 | horizontal size = 72 | verti
cal size = 72
     Area = 3219
     Centroid Coordinates are = (59.195402298850574, 59.195402298850574)
```



```
---- conveyor_f100 ---
Number of connected components = 5
--Background--
     left most coordinate = 0 | top most coordinate = 0 | horizontal size = 1920 | verti
cal size = 1080
     Area = 2059672
     Centroid Coordinates are = (957.3677522440466, 540.4425816343573)
--Component1--
     left most coordinate = 1454 | top most coordinate = 150 | horizontal size = 92 | ve
rtical size = 100
     Area = 4630
     Centroid Coordinates are = (1499.2412526997841, 199.28444924406048)
--Component2--
     left most coordinate = 1259 | top most coordinate = 359 | horizontal size = 82 | ve
rtical size = 82
     Area = 3079
     Centroid Coordinates are = (1299.1815524520948, 399.1815524520948)
--Component3--
     left most coordinate = 1459 | top most coordinate = 459 | horizontal size = 82 | ve
rtical size = 82
     Area = 3079
     Centroid Coordinates are = (1499.1815524520948, 499.1815524520948)
--Component4--
     left most coordinate = 650 | top most coordinate = 550 | horizontal size = 101 | ve
rtical size = 101
```

```
Area = 3140
Centroid Coordinates are = (700.0, 600.0)
```



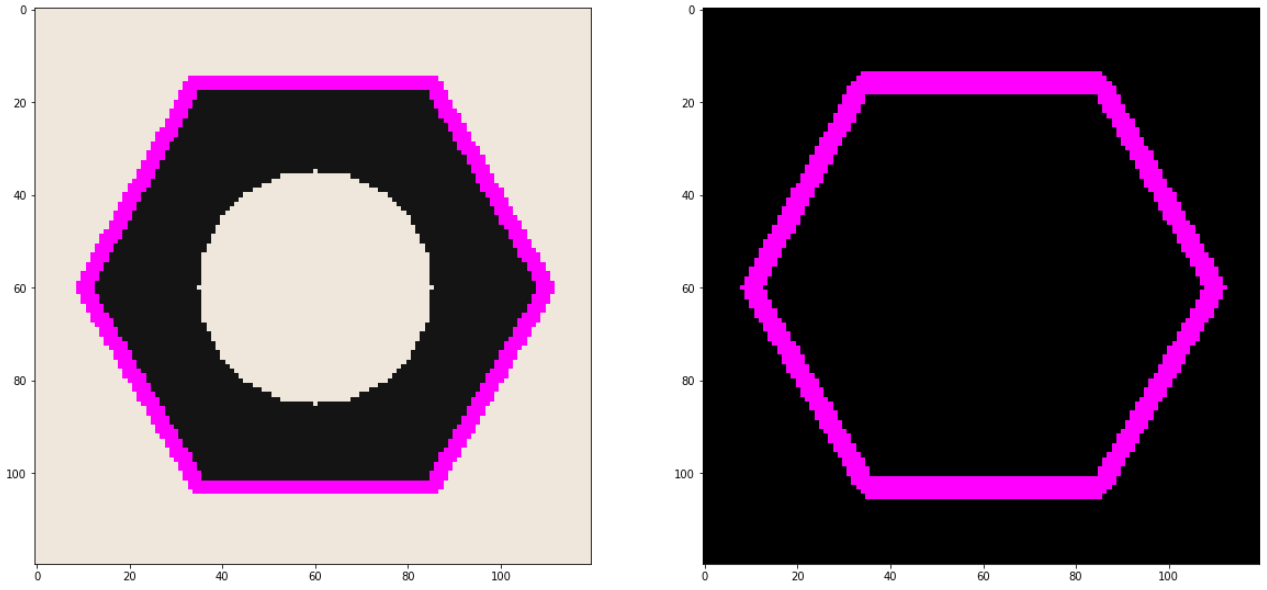
1. Contour analysis: Use `findContours` function to retrieve the *extreme outer* contours. (see https://docs.opencv.org/4.5.2/d4/d73/tutorial_py_contours_begin.html for help and https://docs.opencv.org/4.5.2/d3/dc0/group__imgproc__shape.html#gadf1ad6a0b82947fa1fe3c3d4 for information.

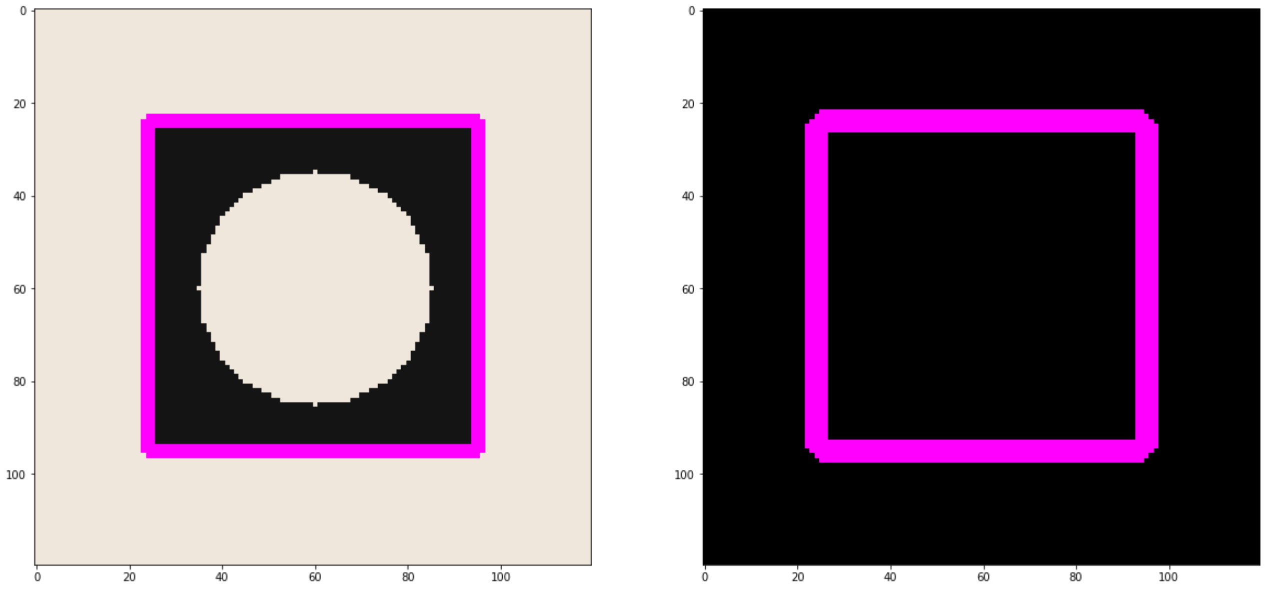
Display these contours. You should see something like the following:

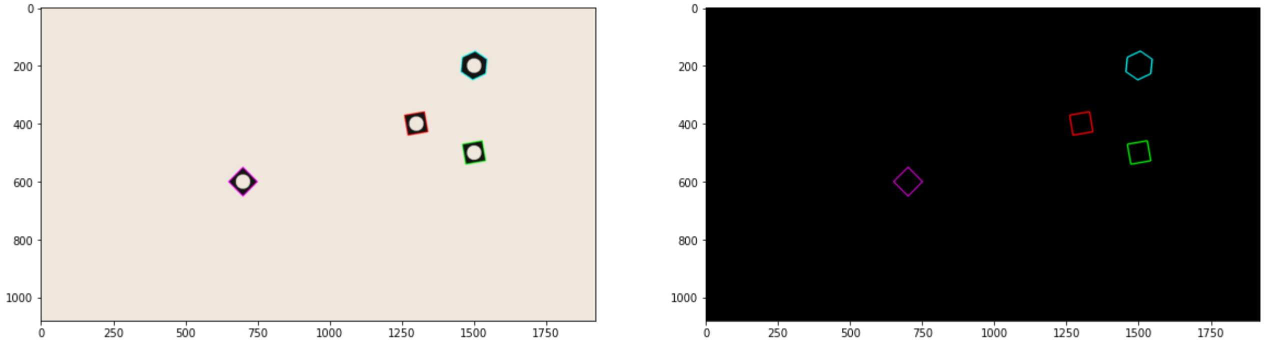
extreme_outer_contours

In [ ]:

```
hexnut_template = cv.imread('hexnut_template.png', cv.IMREAD_COLOR)
squarenut_template =  cv.imread('squarenut_template.png', cv.IMREAD_COLOR)
conveyor_f100 =  cv.imread('conveyor_f100.png', cv.IMREAD_COLOR)

images = [hexnut_template,squarenut_template,conveyor_f100]
colors = [(255,0,255), (0,255,0),(0,0,255),(255,255,0) ]

for j in range(len(images)):
    output = np.zeros(images[j].shape,dtype="uint8")
    contours, hierarchy = cv.findContours(cv.bitwise_not(closed_images[j]), cv.RETR_EXT
    for i in range(len(contours)):
        cnt = contours[i]
        cv.drawContours(images[j], [cnt], 0, colors[i%4], 2)
        cv.drawContours(output, [cnt], 0, colors[i%4], 3)

    fig, ax = plt.subplots(1,2,figsize= (20,15))
    ax[0].imshow(cv.cvtColor(images[j],cv.COLOR_BGR2RGB))
    ax[1].imshow(cv.cvtColor(output,cv.COLOR_BGR2RGB))
    plt.show()
```







Detecting Objects on a Synthetic Conveyor

In this section, we will use the synthetic `conveyor.mp4` sequence to count the two types of nuts.

1. Open the sequence and play it using the code below.

In [ ]:

```
cv.namedWindow('Conveyor', cv.WINDOW_NORMAL)
cap = cv.VideoCapture('conveyor.mp4')
f = 0
frame = []
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        print("Can't receive frame (stream end?). Exiting.")
        break

    f += 1
    text = 'Frame:' + str(f)
    cv.putText(frame,text , (100, 100), cv.FONT_HERSHEY_COMPLEX, 1, (0,250,0), 1, cv.LI
    cv.imshow('Conveyor', frame)

    if cv.waitKey(1) == ord('q'):
        break

cap.release()
cv.destroyAllWindows()
```

1. Count the number of matching hexagonal nuts in `conveyor_f100.png`. You can use `matchCountours` function as shown in https://docs.opencv.org/4.5.2/d5/d45/tutorial_py_contours_more_functions.html to match contours in each frame with that in th template.

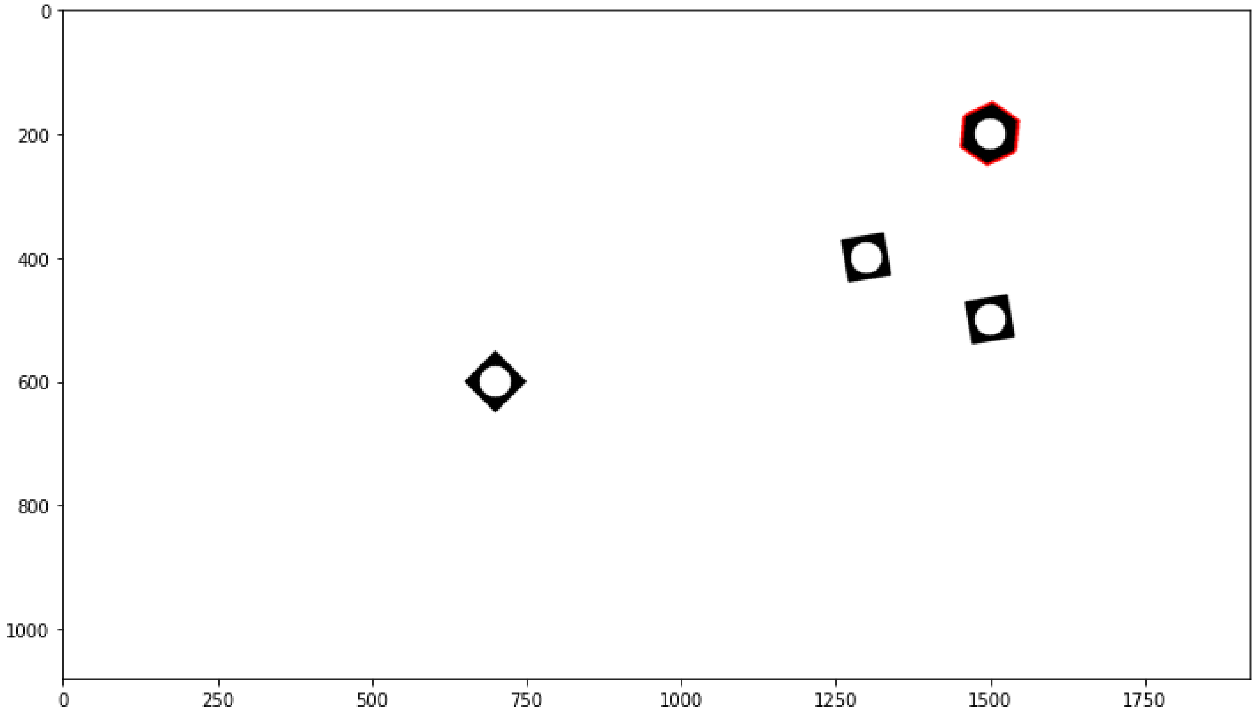
2. Count the number of objects that were conveyed along the conveyor belt: Display the count in the current frame and total count upto the current frame in the output video. Please compress your video (using Handbreak or otherwise) before uploading. It would be good to experiment first with the two adjacent frames `conveyor_f100.png` and `conveyor_f101.png`. In order to disregard partially appearing nuts, consider comparing the contour area in addition to using the `matchCountours` function.

In [ ]:

```
hex_nut = cv.bitwise_not(closing1)
```

```
ret, thresh = cv.threshold(hex_nut, 127, 255,0)
contours,hierarchy = cv.findContours(thresh,2,1)
cnt = contours[0]

conveyor = closing3
ret1, thresh1 = cv.threshold(conveyor, 127, 255,0)
contours1,hierarchy1 = cv.findContours(thresh1,2,1)
cnt1 = contours1[0]

matches = 0
match_cnt = []
for i in contours1:
    ret = cv.matchShapes(i,cnt,1,0.0)
    if (ret<0.001):
        matches+=1
        match_cnt.append(i)

conveyor = cv.cvtColor(conveyor,cv.COLOR_GRAY2RGB)
for j in match_cnt:
    cv.drawContours(conveyor, [j], 0, [255,0,0], 3)

print("Number of matches =",matches)
plt.figure(figsize = (12,8))
plt.imshow(conveyor)
plt.show()
```

```
Number of matches = 1
```



In [ ]:

```
conveyor_f100 =  cv.imread('conveyor_f100.png',0)
conveyor_f101 =  cv.imread('conveyor_f101.png',0)

frames = [conveyor_f100,conveyor_f101]
kernel = np.array([[0,1,0],
                   [1,1,1],
                   [0,1,0]])
kernel = kernel.astype('uint8')
```

```
for i in frames:

    ret,th = cv.threshold(i,0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)
    closing = cv.morphologyEx(th, cv.MORPH_CLOSE, kernel)

    invert = cv.bitwise_not(closing)
    output = cv2.connectedComponentsWithStats(invert, connectivity, cv2.CV_32S)
    (numLabels, labels, stats, centroids) = output

    for i in range(numLabels):
        #stats
        area = stats[i, cv2.CC_STAT_AREA]
        cx1,cy1 = centroids[i,0],centroids[i,1]
        print("Area =",area,"| Centroid =",(cx1,cy1))
    print("----------")
```

```
Area = 2059672 | Centroid = (957.3677522440466, 540.4425816343573)
Area = 4630 | Centroid = (1499.2412526997841, 199.28444924406048)
Area = 3079 | Centroid = (1299.1815524520948, 399.1815524520948)
Area = 3079 | Centroid = (1499.1815524520948, 499.1815524520948)
Area = 3140 | Centroid = (700.0, 600.0)
----------
Area = 2059672 | Centroid = (957.4353746615966, 540.4425816343573)
Area = 4630 | Centroid = (1489.2412526997841, 199.28444924406048)
Area = 3079 | Centroid = (1289.1815524520948, 399.1815524520948)
Area = 3079 | Centroid = (1489.1815524520948, 499.1815524520948)
Area = 3140 | Centroid = (690.0, 600.0)
----------
```

Here we can observe that the area of full nuts are in the range (3070,4650)

But to get the full hex nut we will have to use an area threshold 4600

Also after 1 frame the nuts have moved approximately 10 pixels in x direction

But this threshold doesn't work, Therefore after testing a threshold of 5 was chosen.

In [ ]:

```
# Yor code here.
hex_nut = cv.bitwise_not(closing1)
ret, thresh = cv.threshold(hex_nut, 127, 255,0)
contours,hierarchy = cv.findContours(thresh,2,1)
cnt = contours[0]
first = 0
initial_coor = []
total_matches = 0

kernel = np.array([[0,1,0],
                   [1,1,1],
                   [0,1,0]])
kernel = kernel.astype('uint8')

# Writing the video

frame_array = []
shape = (1080, 1920, 3)

# Your code here
cv.namedWindow('Conveyor', cv.WINDOW_NORMAL)
cap = cv.VideoCapture('conveyor.mp4')
f = 0
```

```
frame = []
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        print("Can't receive frame (stream end?). Exiting.")
        break

    gray = cv.cvtColor(frame,cv.COLOR_BGR2GRAY)
    ret,th = cv.threshold(gray,0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)
    closing = cv.morphologyEx(th, cv.MORPH_CLOSE, kernel)
    invert = cv.bitwise_not(closing)
    output = cv2.connectedComponentsWithStats(invert, connectivity, cv2.CV_32S)
    (numLabels, labels, stats, centroids) = output

    matches = 0
    for i in range(1,numLabels):
        #stats
        area = stats[i, cv2.CC_STAT_AREA]
        cx1,cy1 = centroids[i,0],centroids[i,1]

        componentMask = (labels == i).astype("uint8") * 255
        contours1,hierarchy1 = cv.findContours(componentMask , mode=cv.RETR_EXTERNAL, m

        for i in contours1:
            ret = cv.matchShapes(i,cnt,1,0.0)
            if (ret<0.001 and area>4600):
                matches+=1

                if(first == 0):
                    initial_coor.append([cx1,cy1])
                    first += 1

                if (cx1 < initial_coor[0][0]+5 and cx1 > initial_coor[0][0]-5):
                    total_matches += 1


    f += 1
    text = 'Frame:' + str(f)
    cv.putText(frame,text , (100, 100), cv.FONT_HERSHEY_COMPLEX, 1, (0,250,0), 1, cv.LI
    curr_tot_text = "Current Total in Frame: "+str(matches)
    cv.putText(frame,curr_tot_text , (100, 150), cv.FONT_HERSHEY_COMPLEX, 1, (0,250,0),
    total_text = "Total: "+str(total_matches)
    cv.putText(frame,total_text , (100, 200), cv.FONT_HERSHEY_COMPLEX, 1, (0,250,0), 1,

    cv.imshow('Conveyor', frame)
    frame_array.append(frame)

    if cv.waitKey(1) == ord('q'):
        break

cap.release()
cv.destroyAllWindows()


out = cv.VideoWriter('./conveyor_result_190172K.mp4',cv.VideoWriter_fourcc(*'h264'), 30

for i in range(len(frame_array)):
    cv.imshow('Frame', frame_array[i])
    if cv.waitKey(1) == ord('q'):
        break
    out.write(frame_array[i])

out.release()
cv.destroyAllWindows()
```

```
Can't receive frame (stream end?). Exiting.
```