EN2550 - Fundementals of Image Processing and Machine Vision [Assignment 3 - Object counting on a conveyer belt]

Name : R.G.S.M. RANATUNGA

Index No. : 190504H

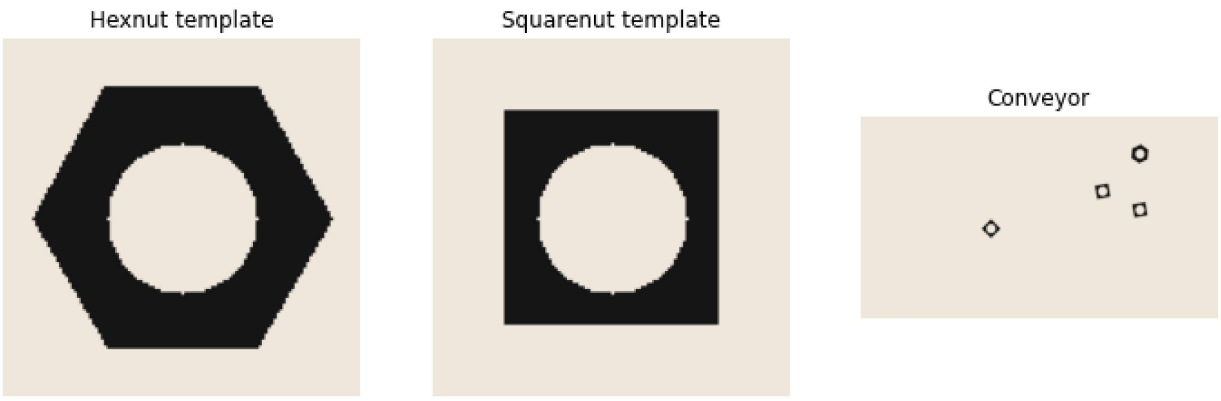
Connected Component Analysis

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 [ ]:
```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
import math
```

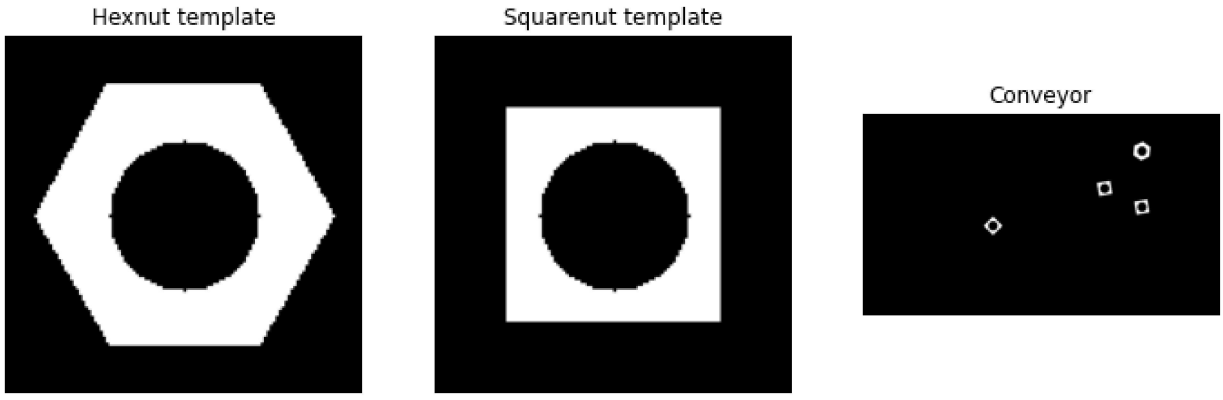
In [ ]:
```
hexnut_template = cv.imread('hexnut_template.png', cv.IMREAD_COLOR)
squarenut_template =  cv.imread('squarenut_template.png', cv.IMREAD_COLOR)
conveyor_f100 =  cv.imread('conveyor_f100.png', cv.IMREAD_COLOR)

fig, ax = plt. subplots(1,3,figsize = (12,5))
ax[0].set_title('Hexnut template')
ax[0].imshow(cv.cvtColor(hexnut_template, cv.COLOR_RGB2BGR))
ax[0].axis('off')
ax[1].set_title('Squarenut template')
ax[1].imshow(cv.cvtColor(squarenut_template, cv.COLOR_RGB2BGR))
ax[1].axis('off')
ax[2].set_title('Conveyor')
ax[2].imshow(cv.cvtColor(conveyor_f100, cv.COLOR_RGB2BGR))
ax[2].axis('off')
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. State the threshold value (automatically) selected in the operation. Display the output images.

In [ ]:
```
# Your code here
# Grayscale image
img1 = cv.cvtColor(hexnut_template, cv.COLOR_BGR2GRAY)
img2 = cv.cvtColor(squarenut_template, cv.COLOR_BGR2GRAY)
img3 = cv.cvtColor(conveyor_f100, cv.COLOR_BGR2GRAY)

# Otsu thresholding
ret1,th1 = cv.threshold(img1,0,255,cv.THRESH_BINARY_INV+cv.THRESH_OTSU)
print("Threshold value of hexnut template -",ret1)
ret2,th2 = cv.threshold(img2,0,255,cv.THRESH_BINARY_INV+cv.THRESH_OTSU)
print("Threshold value of squarenut template -",ret2)
ret3,th3 = cv.threshold(img3,0,255,cv.THRESH_BINARY_INV+cv.THRESH_OTSU)
print("Threshold value of coveyor -",ret3)

fig,ax = plt.subplots(1,3,figsize = (12, 5))

ax[0].set_title('Hexnut template')
ax[0].imshow(th1,'gray')
ax[0].axis('off')
ax[1].set_title('Squarenut template')
ax[1].imshow(th2,'gray')
ax[1].axis('off')
ax[2].set_title('Conveyor')
ax[2].imshow(th3,'gray')
ax[2].axis('off')
```

Out[ ]:
```
Threshold value of hexnut template - 20.0
Threshold value of squarenut template - 20.0
Threshold value of coveyor - 20.0
(-0.5, 1919.5, 1079.5, -0.5)
```

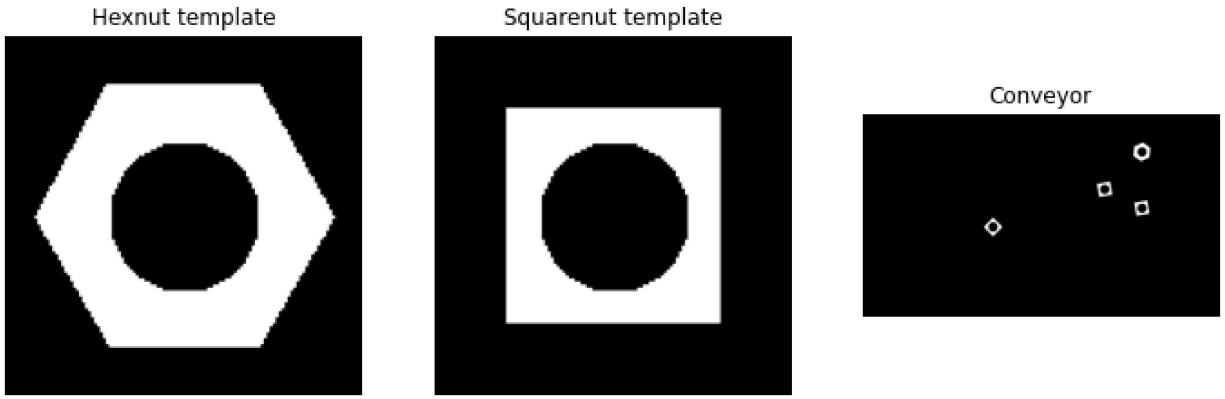


1. Carry out morphological closing to remove small holes inside the foreground. Use a 3×3 kernel.

In [ ]:
```
# Your code here.
kernel = np.ones((3,3),np.uint8) # Defining the kernel

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(1,3,figsize = (12, 5))

ax[0].set_title('Hexnut template')
ax[0].imshow(closing1,'gray')
ax[0].axis('off')
ax[1].set_title('Squarenut template')
ax[1].imshow(closing2,'gray')
ax[1].axis('off')
ax[2].set_title('Conveyor')
ax[2].imshow(closing3,'gray')
ax[2].axis('off')
```

Out[ ]:
```
(-0.5, 1919.5, 1079.5, -0.5)
```



1. Connected components analysis: apply the connectedComponentsWithStats function 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.

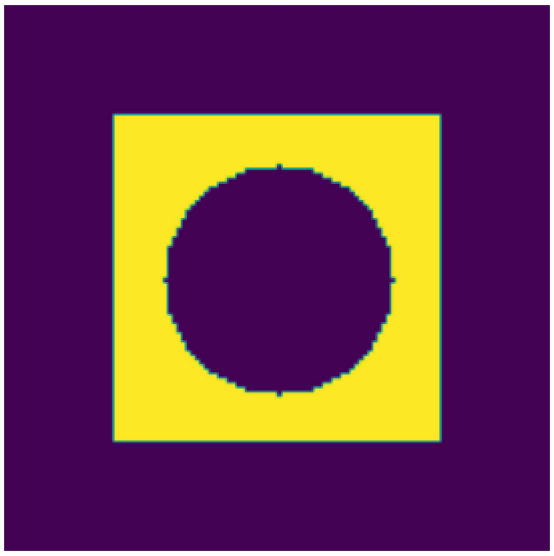
- column 1: The leftmost (x) coordinate which is the inclusive start of the bounding box in the horizontal direction.
- column 2: The topmost (y) coordinate which is the inclusive start of the bounding box in the vertical direction.
- column 3: The horizontal size of the bounding box (width).
- column 4: The vertical size of the bounding box (height).
- column 5: The total area (in pixels) of the connected component.

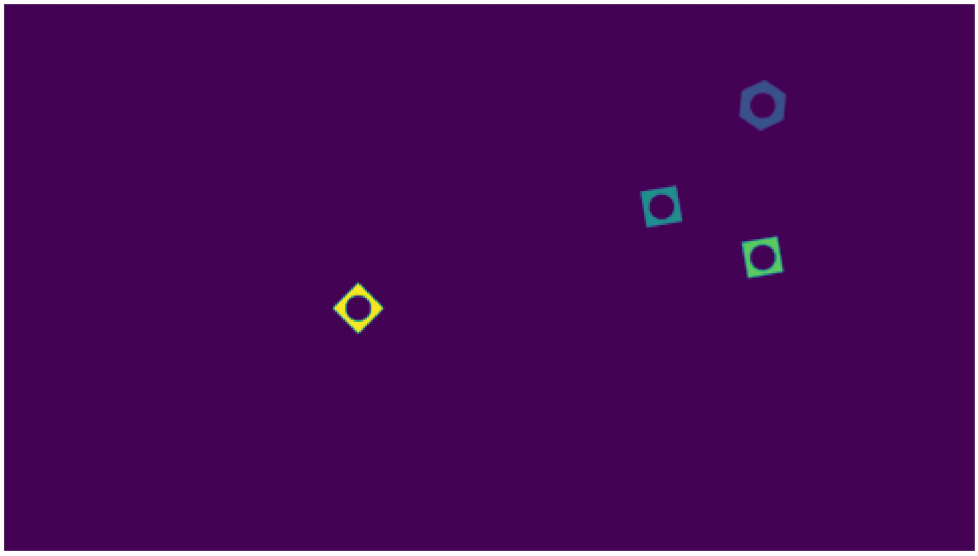
First row in each corresponds to the background.

In [ ]:
```
# Your code here.
connectivity = 4

output1 = cv.connectedComponentsWithStats(th1, connectivity, cv.CV_32S)
```

```
(minValue, maxValue, minPosition, maxPosition) = cv.minMaxLoc(output1[1])
binaryIm1 = np.uint8(255 * (output1[1] - minValue) / (maxValue - minValue))

cv.applyColorMap(binaryIm1,cv.COLORMAP_JET)

fig,ax = plt.subplots(figsize = (12, 5))
ax.imshow(output1[1])
ax.axis('off')

print('---Hexnut template--->')
n1 = output1[0]
print('Connected components','\n',n1)
stats1 = output1[2]
print('Statistics','\n',stats1)
centroids1 = output1[3]
print('Centroids','\n',centroids1)

output2 = cv.connectedComponentsWithStats(th2, connectivity, cv.CV_32S)

(minValue, maxValue, minPosition, maxPosition) = cv.minMaxLoc(output2[1])
binaryIm2 = np.uint8(255 * (output2[1] - minValue) / (maxValue - minValue))

cv.applyColorMap(binaryIm2,cv.COLORMAP_JET)

fig,ax = plt.subplots(figsize = (12, 5))
ax.imshow(output2[1])
ax.axis('off')

print('---Squarenut template---')
n2 = output2[0]
print('Connected components','\n',n2)
stats2 = output2[2]
print('Statistics','\n',stats2)
centroids2 = output2[3]
print('Centroids','\n',centroids2)

output3 = cv.connectedComponentsWithStats(th3, connectivity, cv.CV_32S)

(minValue, maxValue, minPosition, maxPosition) = cv.minMaxLoc(output3[1])
binaryIm3 = np.uint8(255 * (output3[1] - minValue) / (maxValue - minValue))

cv.applyColorMap(binaryIm3,cv.COLORMAP_JET)

fig,ax = plt.subplots(figsize = (10, 5))
ax.imshow(output3[1])
ax.axis('off')

print('---Conveyor---')
n3 = output3[0]
print('Connected components','\n',n3)
stats3 = output3[2]
print('Statistics','\n',stats3)
centroids3 = output3[3]
print('Centroids','\n',centroids3)
```

```
---Hexnut template--->
Connected components
 2
Statistics
 [[   0    0  120  120 9676]
 [  10   16  101   88 4724]]
Centroids
 [[59.33712278 59.63528317]
 [59.83361558 59.22290432]]
---Squarenut template---
Connected components
 2
Statistics
 [[    0     0   120   120 11177]
 [   24    24    72    72  3223]]
Centroids
 [[59.58772479 59.58772479]
 [59.19578033 59.19578033]]
---Conveyor---
Connected components
 5
Statistics
 [[      0       0    1920    1080 2059662]
 [   1454     150      92     100    4632]
 [   1259     359      82      82    3083]
 [   1459     459      82      82    3083]
 [    650     550     101     101    3140]]
Centroids
 [[ 957.36550852  540.44326593]
 [1499.24136442  199.28454231]
 [1299.18196562  399.18196562]
 [1499.18196562  499.18196562]
 [ 700.          600.        ]]
```





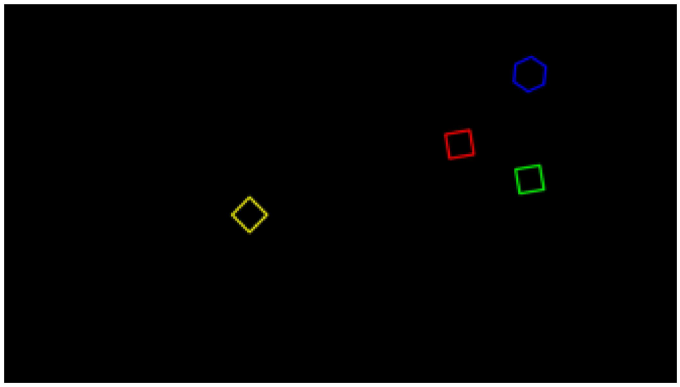
1. Contour analysis: Use findContours function to retrieve the extreme outer contours.

Display these contours.

In [ ]:
```
# Your code here.
img = np.zeros((conveyor_f100.shape[0],conveyor_f100.shape[1],conveyor_f100.shape[2]))

imgn3 = cv.bitwise_not(img3)

ret3,th3 = cv.threshold(img3,100,255,cv.THRESH_BINARY_INV)
contours3, hierarchy3 = cv.findContours(th3, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)

cv.drawContours(img, [contours3[0]], 0, (127,127,0), 5)
cv.drawContours(img, [contours3[1]], 0, (0,255,0), 5)
cv.drawContours(img, [contours3[2]], 0, (255,0,0), 5)
cv.drawContours(img, [contours3[3]], 0, (0,0,255), 5)

fig,ax = plt.subplots()

ax.imshow(img)
ax.axis('off')
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

Out[ ]:
```
(-0.5, 1919.5, 1079.5, -0.5)
```



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.l
    cv.imshow('Conveyor', frame)

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

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

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

1. Count the number of matching hexagonal nuts in conveyor_f100.png. You can use matchCountours function to match contours in each frame with that in the template.

In [ ]:
```
# Your code here.
imgn1 = cv.bitwise_not(img1)
ret1,th1 = cv.threshold(imgn1,100,255,cv.THRESH_BINARY_INV)
imgn2 = cv.bitwise_not(img2)
ret2,th2 = cv.threshold(imgn2,100,255,cv.THRESH_BINARY_INV)
```

```
imgn3 = cv.bitwise_not(img3)
ret3,th3 = cv.threshold(imgn3,100,255,cv.THRESH_BINARY_INV)

contours1 ,hierarchy = cv.findContours(th1,cv.RETR_TREE, cv.CHAIN_APPROX_SIMPLE) #refe

contours2 ,hierarchy = cv.findContours(th2,cv.RETR_TREE, cv.CHAIN_APPROX_SIMPLE) #refe

contours3 ,hierarchy = cv.findContours(th3,cv.RETR_TREE, cv.CHAIN_APPROX_SIMPLE) #targ

count = 0
for c in contours3[1:]:
    match = cv.matchShapes(contours1[1], c, 1, 0.0)
    if match < 0.001:
        count+=1
print('Count of Hexnuts -',count)
count = 0
for c in contours3[1:]:
    match = cv.matchShapes(contours2[1], c, 1, 0.0)
    if match < 0.001:
        count+=1
print('Count of Squarenuts -',count)
```

```
Count of Hexnuts - 1
Count of Squarenuts - 3
```

1. 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 [ ]:
```
def frame_process(frame,kernel):
    frame_gray = cv.cvtColor(frame, cv.COLOR_RGB2GRAY)
    ret,th = cv.threshold(frame_gray,0,255,cv.THRESH_BINARY_INV+cv.THRESH_OTSU)
    closing = cv.morphologyEx(th, cv.MORPH_CLOSE, kernel)
    return closing
```

In [ ]:
```
# Your code here.
cap = cv.VideoCapture('conveyor.mp4')
f = 0
frames = []
total_count = 0
previous_f = []
track_objects = {}
track_id = 0

# Writing the video

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

# Your code here.

while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
```

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

    f += 1
    frame_modified = frame_process(frame,kernel)

    contours ,hierarchy = cv.findContours(frame_modified,cv.RETR_EXTERNAL, cv.CHAIN_AP

    count = 0
    current_f = []
    l = 0

    for c in contours:

        match1 = cv.matchShapes(contours1[1], c, 1, 0.0)
        match2 = cv.matchShapes(contours2[1], c, 1, 0.0)
        area = cv.contourArea(c)

        if match1 < 0.001 and area > 6400:

            cv.drawContours(frame, contours, l,(0,0,255), 7, cv.LINE_8, hierarchy, 0)
            moment = cv.moments(c)
            cx, cy = int(moment['m10']/moment['m00']),int(moment['m01']/moment['m00'])
            current_f.append((cx, cy))
            count +=1

        elif match2 < 0.001 and area > 4900:

            cv.drawContours(frame, contours, l,(0,0,255), 7, cv.LINE_8, hierarchy, 0)
            moment = cv.moments(c)
            cx, cy = int(moment['m10']/moment['m00']),int(moment['m01']/moment['m00'])
            current_f.append((cx, cy))
            count += 1
        l+=1

    # Tracking and annotating objects
    if f <= 2:
        for pt in current_f:
            for pt2 in previous_f:
                d = math.hypot(pt2[0] - pt[0], pt2[1] - pt[1])
                if d < 100:
                    track_objects[track_id] = pt
                    track_id += 1
    else:
        track_obj_copy = track_objects.copy()
        current_f_copy = current_f.copy()
        for id, pt2 in track_obj_copy.items():
            obj_exists = False
            for pt in current_f_copy:
                d = math.hypot(pt2[0] - pt[0], pt2[1] - pt[1])

                if d < 100:
                    track_objects[id] = pt
                    obj_exists = True
                    if pt in current_f:
                        current_f.remove(pt)
                    continue

            if not obj_exists:
                track_objects.pop(id)
```

```
        for pt in current_f:
            track_objects[track_id] = pt
            track_id += 1

    for id, pt in track_objects.items():
        cv.circle(frame, pt, 5, (0, 0, 255), -1)
        cv.putText(frame, str(id), (pt[0], pt[1] - 7), 0, 1, (0, 0, 255), 2)

    prev_frame = current_f.copy()

    text1 = 'Frame: ' + str(f)
    text2 = "Objects in Frame: "+ str(count)
    text3 = "Total no. of Objects: "+str(track_id)
    cv.putText(frame,text1 , (100, 80), cv.FONT_HERSHEY_COMPLEX, 0.8, (255,10,0), 1, c
    cv.putText(frame,text2 , (100, 110), cv.FONT_HERSHEY_COMPLEX, 0.8, (255,10,0), 1,
    cv.putText(frame,text3 , (100, 140), cv.FONT_HERSHEY_COMPLEX, 0.8, (255,10,0), 1,

    frame_array.append(frame)

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

out = cv.VideoWriter('./conveyor_result_190504H.mp4',cv.VideoWriter_fourcc(*'h264'), 3

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])

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

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