EN2550 2021: Object Counting on a Convey Belt

180092F

K.L.G.J.Chandula

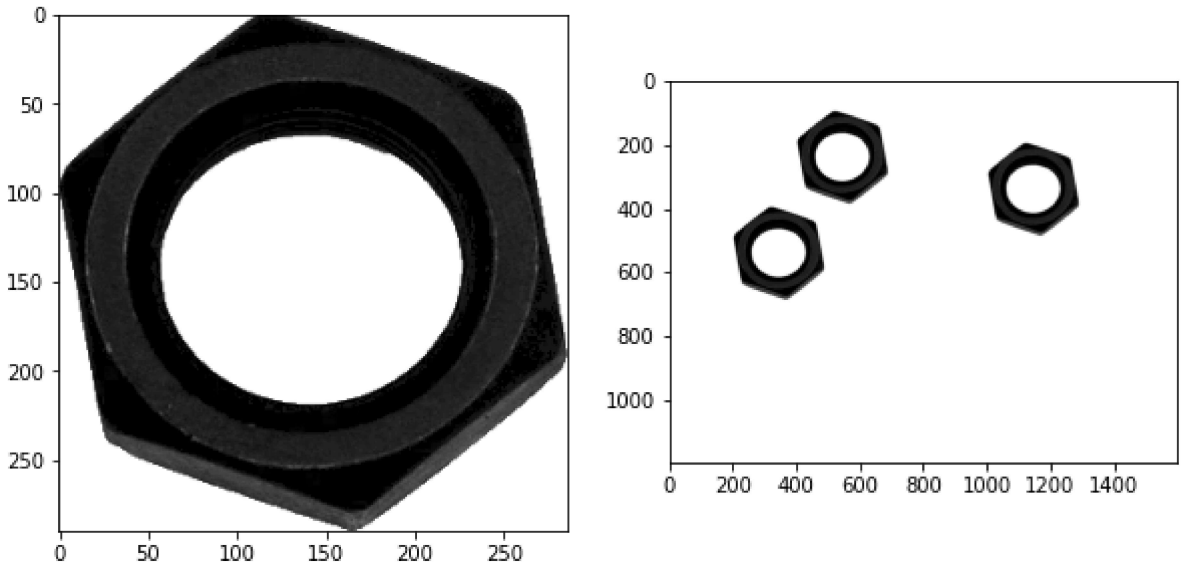
In this assignment, you will be counting and tracking the hexagonal nuts on a moving convey belt.

Let's first import required libraries

In [1]:
```
%matplotlib inline
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
```

Let's load and visualize the template image and the convey belt snapshot at a given time.

In [2]:
```
template_im = cv.imread(r'template.png', cv.IMREAD_GRAYSCALE)
belt_im =  cv.imread(r'belt.png', cv.IMREAD_GRAYSCALE)

fig, ax = plt. subplots(1,2,figsize=(10,10))
ax[0].imshow(template_im, cmap='gray')
ax[1].imshow(belt_im, cmap='gray')
plt.show()
```



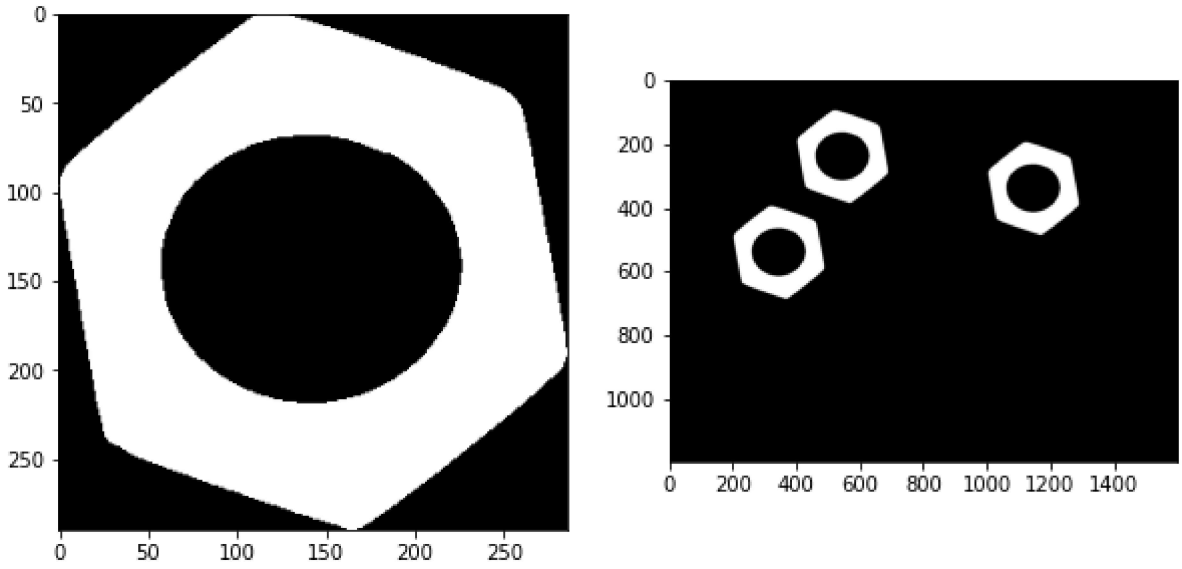
Part-I :

Before going into the implementation, let's play with some functions.

Otsu's thresholding Please read thresholding to get an idea about different types of thresholding and how to use them.(Please use cv.THRESH_BINARY_INV).

In [3]:
```
th_t, img_t = cv.threshold(template_im,0,255,cv.THRESH_BINARY_INV+cv.THRESH_OTSU)
th_b, img_b = cv.threshold(belt_im,0,255,cv.THRESH_BINARY_INV+cv.THRESH_OTSU) #"< Your
fig, ax = plt. subplots(1,2,figsize=(10,10))
ax[0].imshow(img_t, cmap='gray')
```

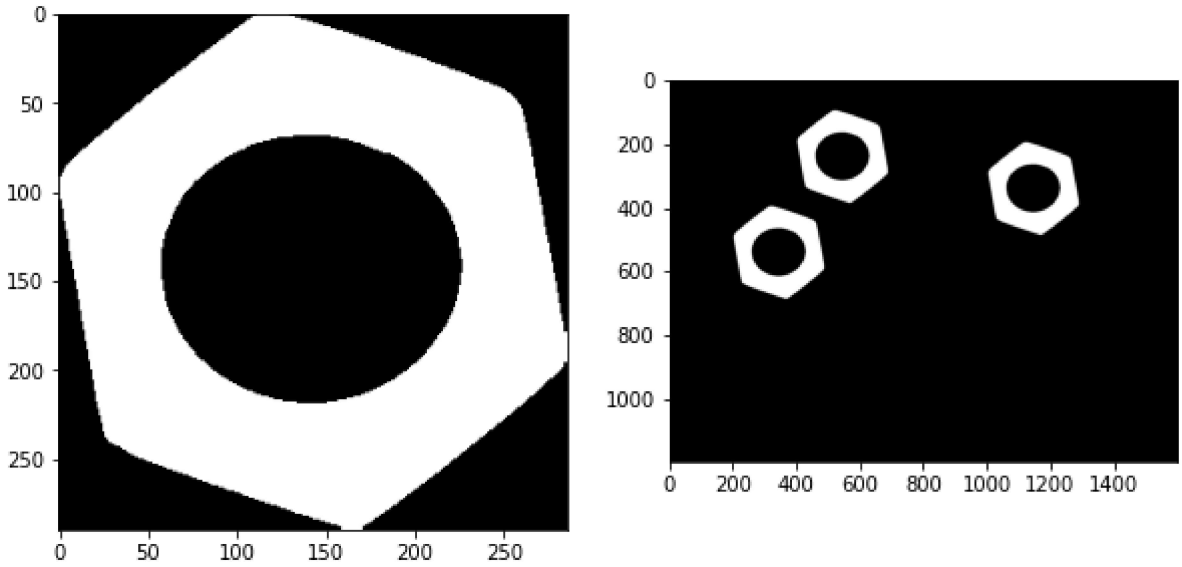
```
ax[1].imshow(img_b, cmap='gray')
plt.show()
```



Morphological closing

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

In [4]:
```
kernel1 = np.ones((3,3),dtype='uint8')#"< 3x3 matrix with all ones, with uint8 dtype>"
closing_t = cv.morphologyEx(img_t, cv.MORPH_CLOSE, kernel1)
#< Your code to apply morphological closing for belt >"
closing_b = cv.morphologyEx(img_b, cv.MORPH_CLOSE, kernel1)
fig, ax = plt. subplots(1,2,figsize=(10,10))
ax[0].imshow(closing_t, cmap='gray')
ax[1].imshow(closing_b, cmap='gray')
plt.show()
```

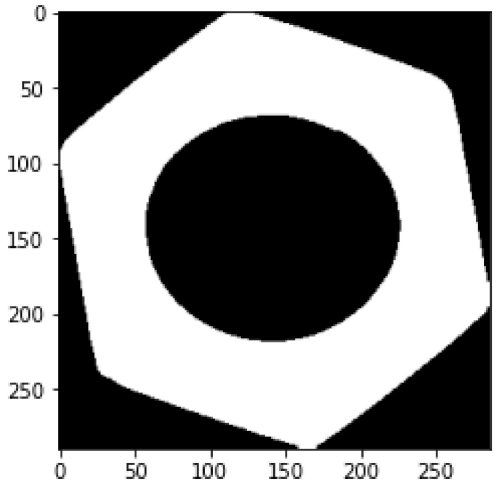


Connected component analysis

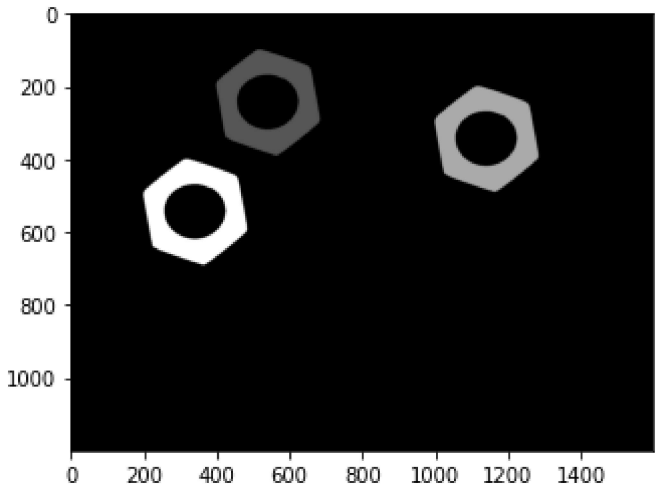
Apply the connectedComponentsWithStats function (see this).

In [5]:
```
retval_t, labels_t, stats_t, centroids_t = cv.connectedComponentsWithStats(closing_t)
retval_b, labels_b, stats_b, centroids_b =cv.connectedComponentsWithStats(closing_b) #"
print("No. of Labels",retval_t)
plt.imshow(labels_t,cmap='gray')

print("Stats\n",stats_t)

print("\nCentroids\n",centroids_t)
plt.show()
print("\n\nNo. of Labels",retval_b)
plt.imshow(labels_b,cmap='gray')
print("Stats\n",stats_b)
print("\nCentroids\n",centroids_b)
```

```
No. of Labels 2
Stats
 [[    0     0   286   290 42290]
 [    0     0   286   290 40650]]

Centroids
 [[142.18770395 145.19172381]
 [142.82489545 143.780369  ]]
```




```
No. of Labels 4
Stats
 [[      0       0    1600    1200 1798161]
 [    400     100     286     290   40613]
 [   1000     200     286     290   40613]
 [    200     400     286     290   40613]]

Centroids
 [[ 807.85728475  614.56805258]
 [ 542.82567158  243.78479797]
 [1142.82567158  343.78479797]
 [ 342.82567158  543.78479797]]
```



How many connected compoonets are detected in each image?

Template image=2 (including background)

Belt image=4 (including background)

What are the statistics? Interpret these statistics.

Statistics are properties of each connected component. Statics consists of following columns

Column 1: cv.CC_STAT_LEFT: The leftmost (x) coordinate which is the inclusive start of the bounding box in the horizontal direction. Column 2: cv.CC_STAT_TOP: the topmost (y) coordinate which is the inclusive start of the bounding box in the vertical direction. Column 3: cv.CC_STAT_WIDTH: the horizontal size of the bounding box. Column 4: cv.CC_STAT_HEIGHT: the vertical size of the bounding box. Column 5: cv.CC_STAT_AREA: the total area (in pixels) of the connected component.

What are the centroids?
X,Y coordintes of centroid of each component

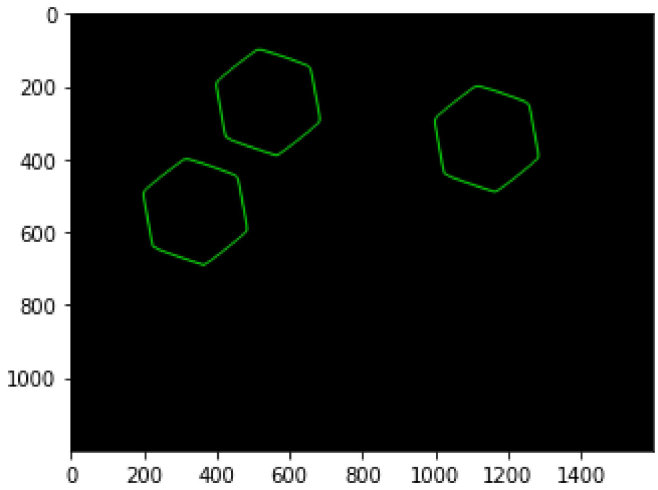
Contour analysis

Use findContours function to retrieve the extreme outer contours. (see for help and see for information.)

Display these countours.

In [6]:
```
contours_t, hierarchy_t = cv.findContours(closing_t, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_
contours_b, hierarchy_b =cv.findContours(closing_b, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_S
# Visualizing contours

im_contours_belt = np.zeros((belt_im.shape[0],belt_im.shape[1],3), np.uint8)
conts = cv.drawContours(im_contours_belt, contours_b, -1, (0,255,0), 3).astype('uint8')
plt.imshow(conts)
```

Out[6]:
```
<matplotlib.image.AxesImage at 0x1dec7ac9f48>
```



Count the number of matching hexagonal nuts in belt.png. Use the matchShapes function as shown in examples to match contours in the belt image with that in the template.

Get an idea about the value output by the cv.matchShapes when both the template and the reference image have the same shape. Understand the given code snippet.

In [7]:
```
label = 1 # remember that the label of the background is 0
belt = ((labels_b >= label)*255).astype('uint8')
belt_cont, template_hierarchy = cv.findContours(belt, cv.RETR_EXTERNAL, cv.CHAIN_APPROX
for j,c in enumerate(belt_cont):
        print(cv.matchShapes(contours_t[0], c, cv.CONTOURS_MATCH_I1, 0.0))
```

```
0.00010071698397173812
0.00010071698397950968
0.00010071698397506879
```

Part - II

Frame tracking through image moments. Use the cv.contourArea(), see this and calculate the the area of the contours_b[1]

In [ ]:
```

```

In [8]:
```
ca = area = cv.contourArea(contours_b[1])#'<Your code goes here>'\
print(ca)
```

```
60059.5
```

Use the cv.moments to extract the x and y coordinates of the centroid of contours_b[1].

In [9]:
```
M = cv.moments(contours_b[1])
#print(M)
cx = int(M['m10']/M['m00'])
cy = int(M['m01']/M['m00'])
```

Make a variable called count to represent the number of contours and set it to the value 1. Make an np array [cx, cy, ca, count] and name this as object_prev_frame

In [10]:
```

```

```
count=1
object_prev_frame =np.array([cx,cy,ca,count])
print(object_prev_frame)
```

```
[1.14200e+03 3.43000e+02 6.00595e+04 1.00000e+00]
```

Similarly, you can create the object_curr_frame(to describe the current values) and define the threshold delta_x to check whether the corresponding element of both the object_curr_frame and object_prev_frame are less than the delta_x. You can set delta_x as 15 or so. (Here the delta_x can be thought of as the movement of the cx from frame to frame)

In [11]:
```
delta_x = 15
```

Part - III

1. Implement the function get_indexed_image, which takes an image as the input, performs thresholding, closing, and connected component analysis and return retval, labels, stats, centroids. (Grading)

In [12]:
```
def get_indexed_image(im):
    """ Thresholding, closing, and connected component analysis lumped
    """
    th, img = cv.threshold(im,0,255,cv.THRESH_BINARY_INV+cv.THRESH_OTSU)
    kernel= cv.getStructuringElement(cv.MORPH_RECT,(3,3))
    closing= cv.morphologyEx(img, cv.MORPH_CLOSE, kernel1)
    retval, labels, stats, centroids = cv.connectedComponentsWithStats(closing)
    return retval, labels, stats, centroids
```

1. Implement the function is_new, which checks the dissimilarity between 2 vectors. (Grading)

In [13]:
```
def is_new(a, b, delta, i):
    """ Vector Dissimilarity with an Array of Vectors
    Checks if vector b is similar to a one or more vectors in a outside the tolerances
    vector i specifies which elements in b to compare with those in a.
    """
    check = []
    for j in i:
        abs_Dff=np.absolute(b-a)# get the absolute differences
        abs_Dff[:,j]=abs_Dff[:,j]>delta[j]

        #check= abs_Dff[:,i].all()
        check.append(abs_Dff[:,i].all())
    check = np.array(check).all()
    return check
```

In [14]:
```
a = np.array([[1.36100e+03, 5.53000e+02, 5.99245e+04, 2.00000e+00],
 [7.61000e+02, 4.53000e+02, 5.99385e+04, 1.00000e+00],
 [1.55200e+03, 2.43000e+02, 6.00585e+04, 3.00000e+00]])
b = np.array([7.51000e+02, 4.53000e+02, 5.99385e+04, 3.00000e+00])
delta = np.array([delta_x])
i = np.array([0])
abs_Dff=np.absolute(b-a)

delta = np.array([delta_x])
```

```
i = np.array([0])

print(abs_Dff)
print(b-a)
print()
print(a)
print()
print(b)


a = np.array([[1.36100e+03, 5.53000e+02, 5.99245e+04, 2.00000e+00],
 [7.61000e+02, 4.53000e+02, 5.99385e+04, 1.00000e+00],
 [1.55200e+03, 2.43000e+02, 6.00585e+04, 3.00000e+00]])
b = np.array([7.51000e+02, 4.53000e+02, 5.99385e+04, 3.00000e+00])
delta = np.array([delta_x])
i = np.array([0])

print( is_new(a, b, delta, i) )
```

```
[[610. 100.  14.   1.]
 [ 10.   0.   0.   2.]
 [801. 210. 120.   0.]]
[[-610. -100.   14.    1.]
 [ -10.    0.    0.    2.]
 [-801.  210. -120.    0.]]

[[1.36100e+03 5.53000e+02 5.99245e+04 2.00000e+00]
 [7.61000e+02 4.53000e+02 5.99385e+04 1.00000e+00]
 [1.55200e+03 2.43000e+02 6.00585e+04 3.00000e+00]]

[7.51000e+02 4.53000e+02 5.99385e+04 3.00000e+00]
False
```

In [15]:

```
# check is_new  expected answer False

a = np.array([[1.36100e+03, 5.53000e+02, 5.99245e+04, 2.00000e+00],
 [7.61000e+02, 4.53000e+02, 5.99385e+04, 1.00000e+00],
 [1.55200e+03, 2.43000e+02, 6.00585e+04, 3.00000e+00]])
b = np.array([7.51000e+02, 4.53000e+02, 5.99385e+04, 3.00000e+00])
delta = np.array([delta_x])
i = np.array([0])

assert is_new(a, b, delta, i) == False, " Check the function "
```

1. If the array a is in the shape of (number of nuts , len(object_prev_frame)) (i.e. array a is made by stacking all the object_prev_frame for each frame. If b is in the form of [cx, cy, ca, count], write the function prev_index to find the index of a particular nut in the previous frame. (Grading)

In [16]:

```
def prev_index(a, b, delta, i):
    """ Returns Previous Index
    Returns the index of the apppearance of the object in the previous frame.
    (See thee example in the next cell)
    """
    '''index = -1
    '< Your code goes here >''''

    index = -1
    abs_Dff = np.absolute(a - b)
```

```
    matching_rows = []
    for j in i:
        abs_Dff[:,j] = (abs_Dff[:,j] <= delta[j])

        matching_row = np.where(abs_Dff[:,j])[0] # finding the row where  condition is
        matching_rows.append(matching_row)

    values, counts = np.unique(matching_rows, return_counts=True) # best match
    best_match = values[np.argmax(counts)]

    matched = a[best_match]
    index = matched[-1] # since count keeps the index of a nut.

    return index
```

In [17]:

```
# check prev_index expected answer 1
a = np.array([[1.36100e+03, 5.53000e+02, 5.99245e+04, 2.00000e+00],
[7.61000e+02, 4.53000e+02, 5.99385e+04, 1.00000e+00],
[1.55200e+03, 2.43000e+02, 6.00585e+04, 3.00000e+00]])
b = np.array([7.51000e+02, 4.53000e+02, 5.99385e+04, 3.00000e+00])
delta = np.array([delta_x])
i = np.array([0])
print(prev_index(a,b,delta,i))
```

```
1.0
```

You can use following code snippet load and access each frame of a video

In [18]:

```
cap = cv.VideoCapture('conveyor_with_rotation.mp4') # give the correct path here
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        print("Can't receive frame (stream end?). Exiting ...")
        break
    cv.imshow('frame',frame)
    if cv.waitKey(1) == ord('q'):
        break

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

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

1. Implement a code to detect hexagonal nuts in a moving convey belt. (Grading)

Steps:

Use the above code snippet to access each frame and remember to convert the frame into grey scale. Name the variable as grey Call get_indexed_image and extract retval, labels, stats, centroids. Find contours of all nuts present in a given frame of the belt. Initiate a 3-D array with zeros to draw contours. Call this im_contours_belt Draw each contour. Use cv.drawContours. See this

In [ ]:

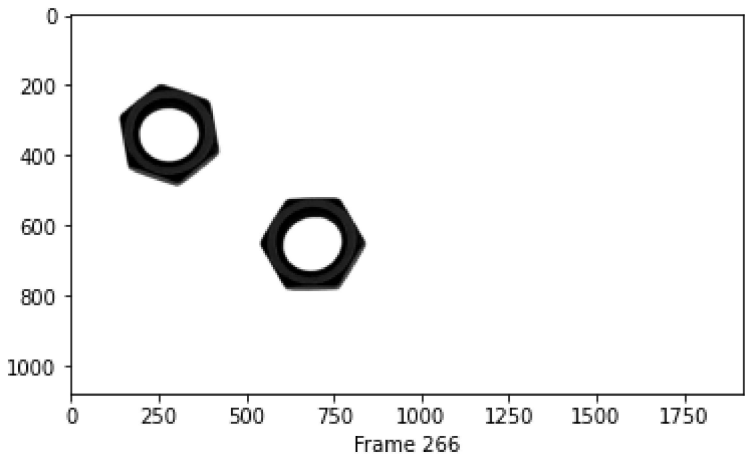
In [ ]:

1. Implement a code to detect hexagonal nuts in a moving convey belt. (Grading) Steps: Use the above code snippet to access each frame and remember to convert the frame into grey scale. Name the variable as grey Call get_indexed_image and extract retval, labels, stats, centroids. Find contours of all nuts present in a given frame of the belt. Initiate a 3-D array with zeros to draw contours. Call this im_contours_belt Draw each contour. Use cv.drawContours. See this

In [19]:
```
frames = []  # storing  frames
cap = cv.VideoCapture('conveyor_with_rotation.mp4') # give the correct path here
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        print("Can't receive frame (stream end?). Exiting ...")
        break
    frame = cv.cvtColor(frame, cv.COLOR_BGR2GRAY) # converting  to grayscale
    frames.append(frame) # storing frames
    if cv.waitKey(1) == ord('q'):
        break
tot_Frames=len(frames)
cap.release()
cv.destroyAllWindows()
print("process completed.")
print("Total Frames: ",tot_Frames)
```

```
Can't receive frame (stream end?). Exiting ...
process completed.
Total Frames:  280
```

In [33]:
```
# ploting random frame
x = np.random.randint(0,280)
plt.subplot()
plt.imshow(frames[x], cmap ='gray')
plt.xlabel("Frame " + str(x))
plt.show()
```

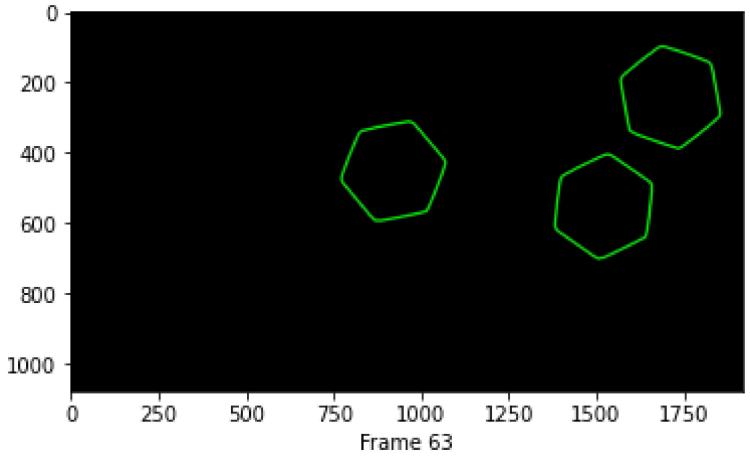


In [21]:
```
# Find contours
contour_plots = []
contours_list = []
for gray in frames:
    # finding contours
```

```
    retval, labels, stats, centroids = get_indexed_image(gray) # Connected Components A
    belt = ((labels >= 1)*255).astype('uint8')
    contours,hierarchy  = cv.findContours(belt, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPL
    contours_list.append(contours)
    # draw contours
    im_contours_belt = np.zeros((belt.shape[0],belt.shape[1],3), np.uint8)
    cont_plot = cv.drawContours(im_contours_belt, contours, -1, (0,255,0), 5).astype('u
    contour_plots.append(cont_plot)
```

In [22]:

```
# ploting  contour plots on  random frame

x = np.random.randint(0,280)
plt.subplot()
plt.imshow(contour_plots[x])
plt.xlabel("Frame " + str(x))
plt.show()
```



Object detection and tracking For each contour of the belt frame,

Use is_new and prev_index functions to track each frame and get the indices of each nut. Write a code to detect and track hexagonal nuts in each frame. Hint: If you are thresholding on areas (template and contour) you can use 500 as the threshold. You can set the matching threshold to be 0.5 and experiment

In [23]:

```
video = []
for i in frames:
    # Thresholding, closing, and connected component analysis

    retval, labels, stats, centroids  = get_indexed_image(i)
    belt = ((labels >= 1)*255).astype('uint8')
    contours,hierarchy  = cv.findContours(belt, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPL

    count = 0 # No. of nuts
    frame = []

    for contour in contours:
        metric = cv.matchShapes(contours_t[0], contour, cv.CONTOURS_MATCH_I1, 0.0) #mat

        if metric <= 0.5: # Set threshold to  0.5 smaller the value more matching
            count +=1
            M  = cv.moments(contour)
```

```
            ca = M['m00']
            cx, cy = int(M['m10']/M['m00']), int(M['m01']/M['m00'])
            frame.append(np.array([cx, cy, ca, count]))

    video.append(frame)
print(video[100])
```

```
[array([1.15100e+03, 5.53000e+02, 5.99235e+04, 1.00000e+00]), array([5.51000e+02, 4.5300
0e+02, 5.99395e+04, 2.00000e+00]), array([1.34200e+03, 2.43000e+02, 6.00585e+04, 3.00000
e+00])]
```

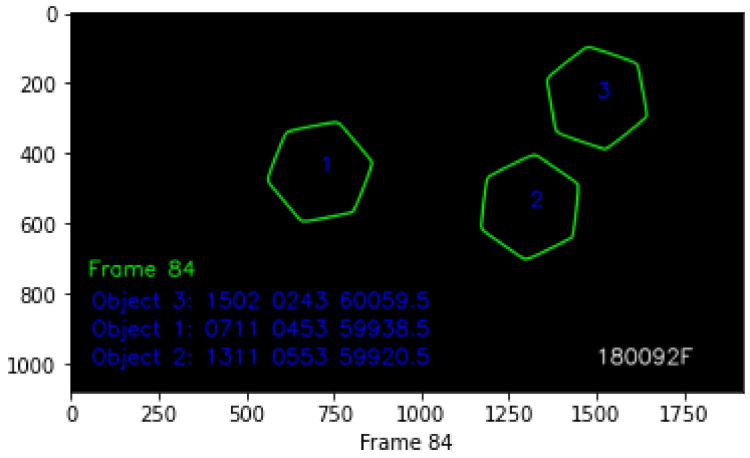
In [ ]:

In [25]:
```
# counting nuts
total_nuts = int(video[0][-1][-1]) #initial number of nuts in frame 0

prev_frame = video[0] # frame 0
delta_x = np.array([15])
i = np.array([0])
for n in range(1, len(video)):
    current_frame = video[n]
    #check whether nut is new one
    for nut in current_frame:

        if is_new(prev_frame, nut, delta_x, i):

            total_nuts +=1
            nut[-1] = total_nuts # Assigning new count

        else:
            # if not a new one
            nut_index = prev_index(prev_frame, nut, delta_x, i)
            nut[-1] = nut_index # assigning old count

    prev_frame = current_frame
```

In [26]:
```
annotated_frames =[] # adding annotations
frame_num = 0

for frame, cont_frame, contours in zip(video, contour_plots, contours_list):

    img = cont_frame
    y = 0 # variable to adjust the position of annotations
    for nut in frame:
        img = cv.putText(img, str(int(nut[-1])),(int(nut[0]),int(nut[1])),cv.FONT_HERSH
        img = cv.putText(img, "Object {}: {:04} {:04} {:05}".format(int(nut[-1]), int(n

        y +=1  # variable to adjust the position of annotations
    img = cv.putText(img, "Frame "+str(frame_num) , (50,750) , cv.FONT_HERSHEY_SIMPLEX,
    img = cv.drawContours(img, contours, -1, (0,255,0), 5).astype('uint8')
    img = cv.putText(img, "180092F" , (1500,1000) , cv.FONT_HERSHEY_SIMPLEX, 2, (255,25
    annotated_frames.append(img)
    frame_num +=1
```

In [32]:
```
# ploting  annotated frame
```

```
x = np.random.randint(0,280)
plt.subplot()
plt.imshow(annotated_frames[x])
plt.xlabel("Frame " + str(x))
plt.show()
```



In [28]:

```
# saving video
output = '180092F_en2550_a05.mp4'
duration = 10
fourcc = cv.VideoWriter_fourcc(*'MP4V')
fps = int(len(annotated_frames)/duration) # Framerate
height, width,_ = annotated_frames[0].shape
frame_size = (width, height)
out = cv.VideoWriter(output, fourcc, fps, frame_size, 1)
for frame in annotated_frames:
    out.write(frame)
out.release()
print('Video saved')
```

```
Video saved
```

In [ ]: