..........ASSIGNMENT 05 .... 180647M ....

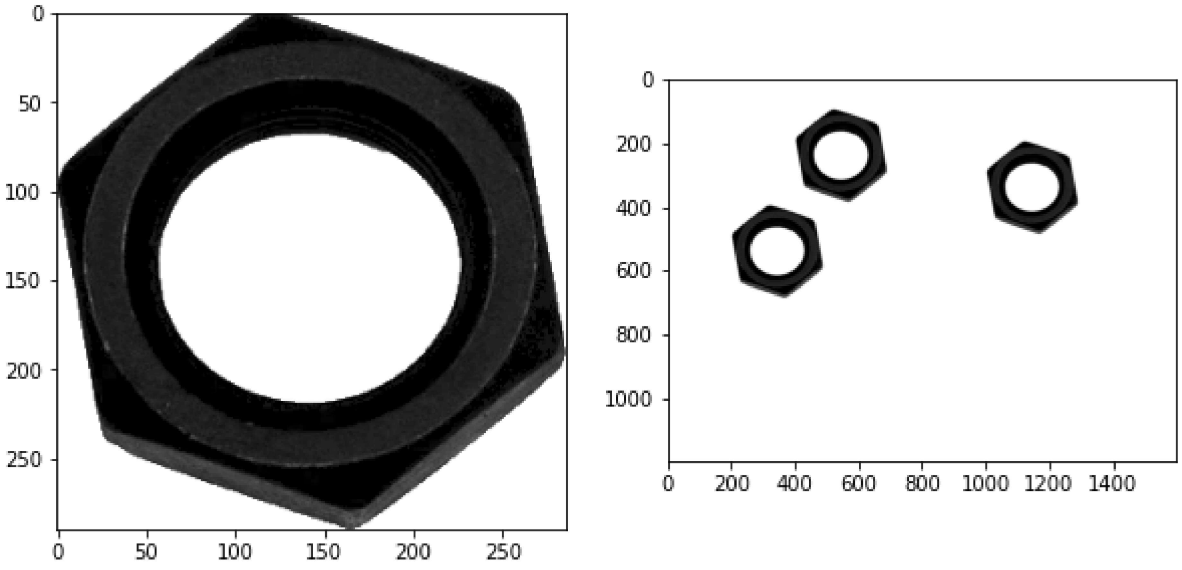
QUESTION NO 1

first import required libraries

```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
from google.colab.patches import cv2_imshow
%matplotlib inline
```

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

```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
from google.colab.patches import cv2_imshow
%matplotlib inline
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()
```



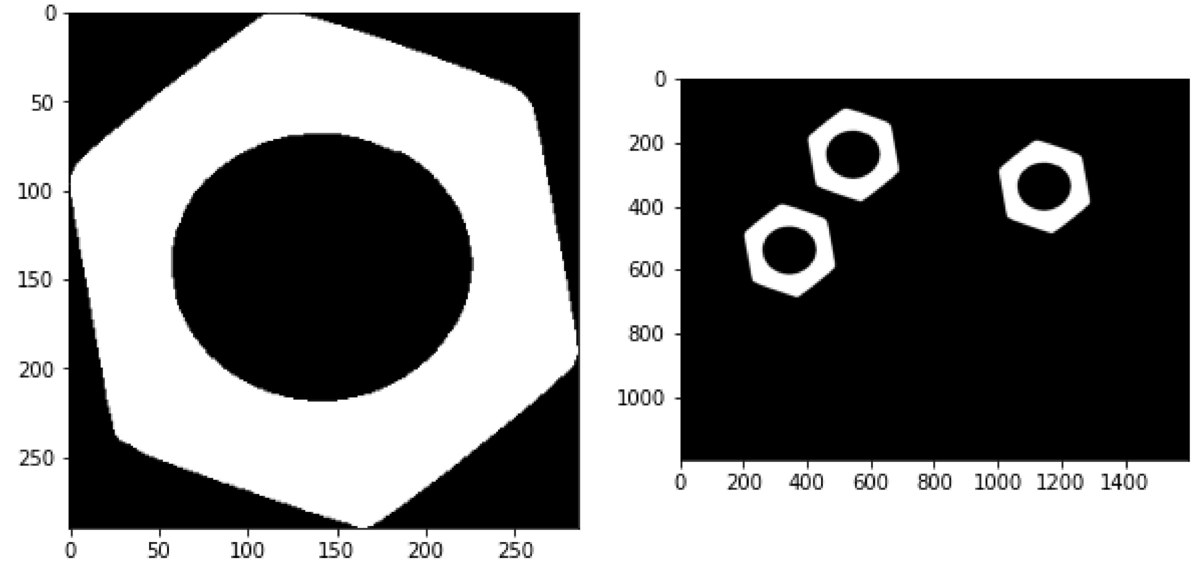
1.1 Part-I

1.1.1 Otsu’s thresholding

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

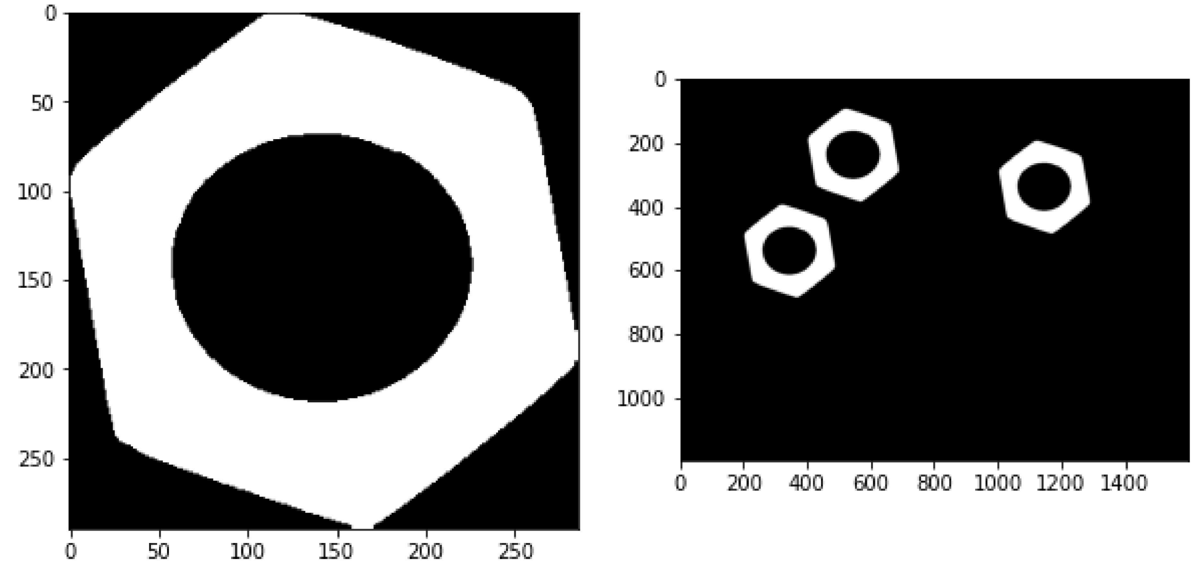
```
ax[1].imshow(img_b, cmap='gray')
print("threshold template,belt=",th_t,th_b)
plt.show()
```

```
threshold template,belt= 138.0 138.0
```



1.1.2 Morphological closing

```
kernel = np.ones((3,3))   #Use a 3 × 3 kernel (instruction)
closing_t = cv.morphologyEx(img_t, cv.MORPH_CLOSE, kernel)
closing_b = cv.morphologyEx(img_b, cv.MORPH_CLOSE, kernel)
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()
```

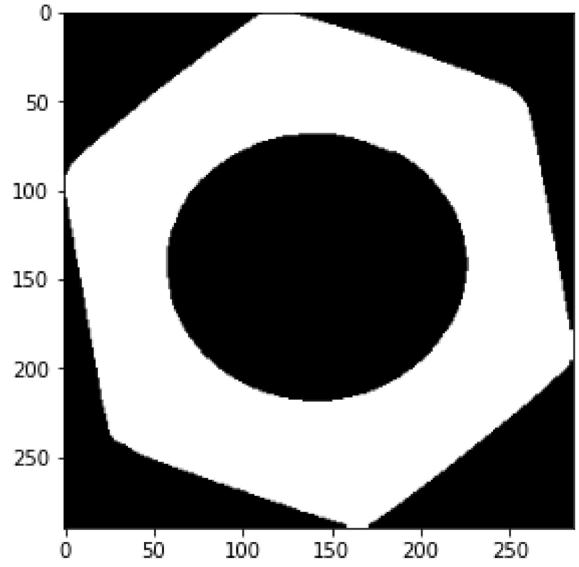


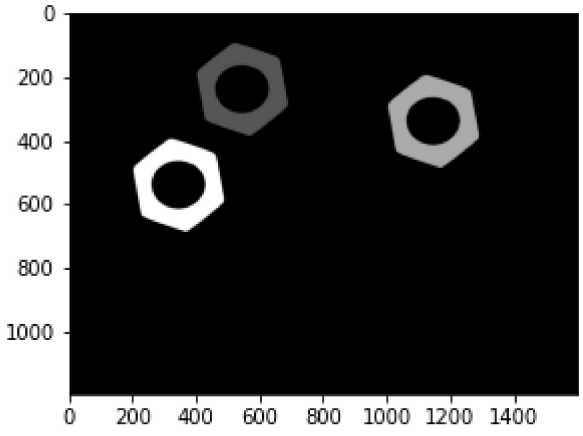
1.1.3 Connected component analysis

```
retval_t, labels_t, stats_t, centroids_t = cv.connectedComponentsWithStats(closing_t)
retval_b, labels_b, stats_b, centroids_b = cv.connectedComponentsWithStats(closing_b)
fig, ax = plt. subplots(1,2,figsize=(10,10))
print('retval_t=',retval_t,'\n','labels_t= \n',labels_t,'\n','stats_t= \n',stats_t,'\n','centroids_t
ax[0].imshow(labels_t, cmap='gray')
print("______________________________________________________________________________________________
print('retval_b=',retval_b,'\n','labels_b= \n',labels_b,'\n','stats_b= \n',stats_b,'\n','centroids_b
ax[1].imshow(labels_b, cmap='gray')
plt.show()
```

```
retval_t= 2
 labels_t=
 [[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]]
 stats_t=
 [[    0     0   286   290 42290]
 [    0     0   286   290 40650]]
 centroids_t=
 [[142.18770395 145.19172381]
 [142.82489545 143.780369  ]]
```

```
retval_b= 4
 labels_b=
 [[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]]
 stats_b=
 [[      0       0    1600    1200 1798161]
 [    400     100     286     290   40613]
 [   1000     200     286     290   40613]
 [    200     400     286     290   40613]]
 centroids_b=
 [[ 807.85728475  614.56805258]
 [ 542.82567158  243.78479797]
 [1142.82567158  343.78479797]
 [ 342.82567158  543.78479797]]
```





1. How many connected compoonets are detected in each image?
2. What are the statistics? Interpret these statistics.
3. What are the centroids?

1.Template image= 2 connected components................ Belt image= 4 connected components

2.Column 1:the leftmost (x) coordinate

Column 2:the topmost (y) coordinate

Column 3:the horizontal size of the bounding box.

Column 4:the vertical size of the bounding box.

Column 5:the total area (in pixels) of the connected component.

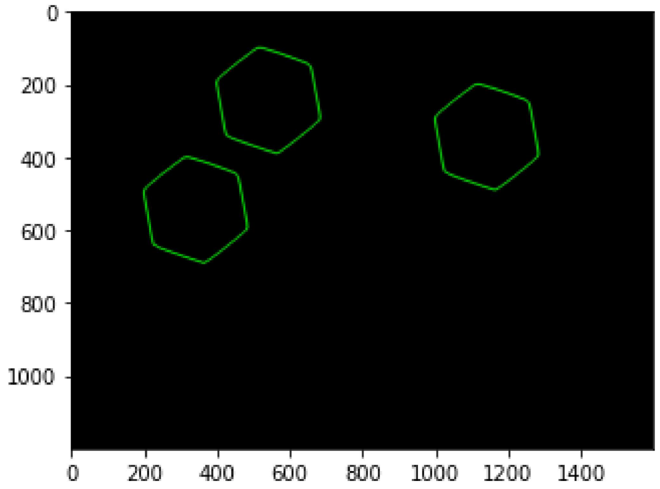
3.These are the centroids of the contours which were identified. First lement in the list represents the centroids of the background

---

1.1.4 Contour analysis

```
contours_t, hierarchy_t = cv.findContours(closing_t, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)
contours_b, hierarchy_b = cv.findContours(closing_b, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)
print('hierarchy_t=',hierarchy_t,' \n','hierarchy_b= \n',hierarchy_b)

# 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)
```

```
hierarchy_t= [[[-1 -1 -1 -1]]]
 hierarchy_b=
 [[[ 1 -1 -1 -1]
  [ 2  0 -1 -1]
  [-1  1 -1 -1]]]
<matplotlib.image.AxesImage at 0x7f8827b0c3d0>
```



1.1.5 Count the number of matching hexagonal nuts in belt.png

```
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_SIMPLE)
for j,c in enumerate(belt_cont):
  #print(contours_t[0])
  #print(c)
  print(cv.matchShapes(contours_t[0], c,cv.CONTOURS_MATCH_I1, 0.0))
```

```
0.00010071698397151607
0.00010071698397928763
0.00010071698397484674
```

1.2 Part - II

1.2.1 Frame tracking through image moments

calculate the the area of the contours_b[1]

```
ca = cv.contourArea(contours_b[1])
print(ca)
#ca2 = cv.contourArea(contours_t[0])
#print(ca2)
```

```
     60059.5
```

x and y coordinates of the centroid of contours_b[1]

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

```
     1142 343
```

Make an np array [cx, cy, ca, count]

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

```
     [1142, 343, 60059.5, 1]
```

define the threshold delta_x

```
delta_x=15
```

1.3 Part - III

1.3.1 1.Implement the function get_indexed_image *

```
def get_indexed_image(im): # an image as the input
 th_t, img_t = cv.threshold(im,0,255,cv.THRESH_BINARY_INV+cv.THRESH_OTSU) #thresholding
 kernel = np.ones((3,3)) #define kernal
 closing_t = cv.morphologyEx(img_t, cv.MORPH_CLOSE, kernel)  #closing
 retval, labels, stats, centroids = cv.connectedComponentsWithStats(closing_t) #connected component
 return retval, labels, stats, centroids  #return retval, labels, stats, centroids
```

1.3.2 2. Implement the function is_new

```
def is_new(a, b, delta, i):
 absolute_different = np.absolute(a - b)
 absolute_different[:,i] = (absolute_different[:,i] > delta[i]) # checks every row of the selected c
                                                                #absolute_different is a coloum vec
 return absolute_different[:,i].all() # All elements are true return true
```

```
 # check is_new expected answer False
a = np.array([[20.56100e+02, 40.53000e+02, 50.99385e+04, 200.00000e+00],
              [70.61000e+02, 40.53000e+02, 50.99385e+04, 100.00000e+00],
              [70.55200e+02, 40.43000e+02, 50.99385e+04, 3.00000e+00]])
b = np.array( [7.51000e+02, 4.53000e+02, 5.99385e+04, 3.00000e+00])
delta = np.array([delta_x])
print(delta)
i = np.array([0])
assert is_new(a, b, delta, i) ==True, " Check the function "
```

```
[15]
```

1.3.3 3 Implement the function prev_index

```
def prev_index(a, b, delta, i):
 absolute_different = np.absolute(a - b)
 absolute_different[:,i] = (absolute_different[:,i] <= delta[i])
 index = np.where(absolute_different[:,i]) # this returns the index and some more deatils
 print(index)
 index1=index[0] #filter out only the indexindex1=index[0] #filter out only the index
 return index1


# 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 "
```

load and access each frame of a video

```
cap = cv.VideoCapture('conveyor_with_rotation.mp4') # give the correct path here
while cap.isOpened():
  ret, frame = cap.read()
  if not ret:
    print("stream end?")
    break
cap.release()
cv.destroyAllWindows()
```

```
stream end?
```

.3.4 3. Implement a code to detect hexagonal nuts in a moving convey belt

convert the frame into grey scale.

```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
from google.colab.patches import cv2_imshow
%matplotlib inline
colour_frames = [] #save the colour frame
gray_frame = []  # save the grey frame
cap = cv.VideoCapture('conveyor_with_rotation.mp4')
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break
    colour_frames.append(frame)
    frame = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
    gray_frame.append(frame)

cap.release()
cv.destroyAllWindows()
print("Video capturing completed.")
```

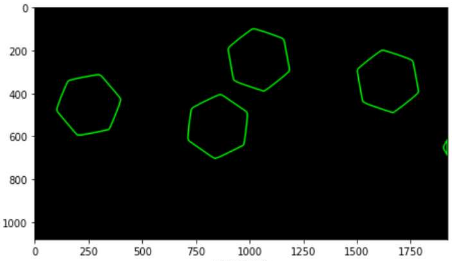
```
Video capturing completed.
```

Call get_indexed_image and extract retval, labels, stats, centroids.

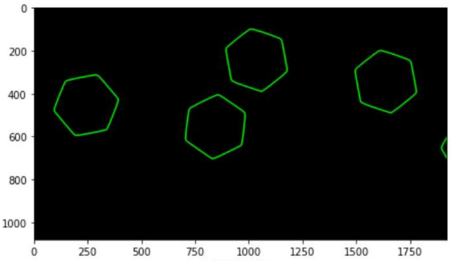
```
contour_plots = []
contours_list = []
for gray in gray_frames:

    retval, labels, stats, centroids = get_indexed_image(gray)
    belt = ((labels >= 1)*255).astype('uint8')
    contours,x  = cv.findContours(belt, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)
    contours_list.append(contours)

    im_contours_belt = np.zeros((belt.shape[0],belt.shape[1],3), np.uint8)
    c_plot = cv.drawContours(im_contours_belt, contours, -1, (0,255,0), 5).astype('uint8')
    contour_plots.append(c_plot)
```

Draw each contour

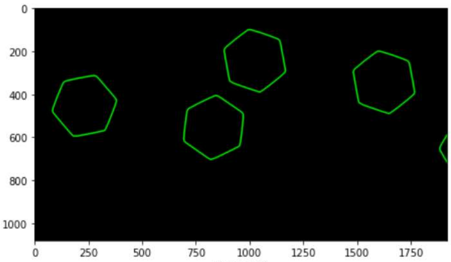
```
plt.figure(figsize=(25,20))
for i in range(12):
    plt.subplot(4,3,i+1)
    plt.imshow(contour_plots[130+i])
    plt.xlabel("Frame " + str(130+i))
plt.show()
```



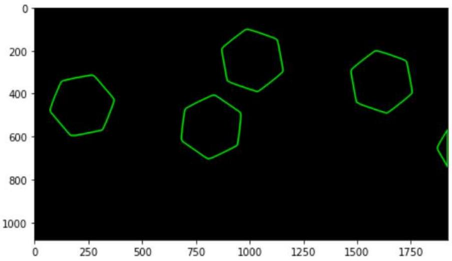
Frame 130



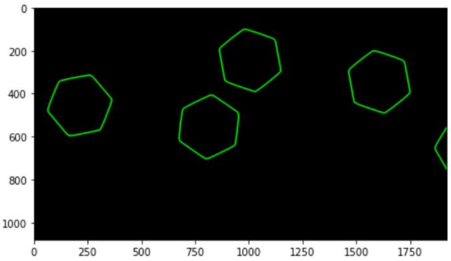
Frame 131



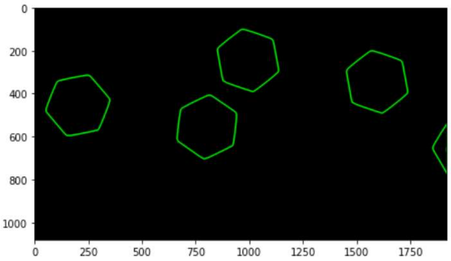
Frame 132



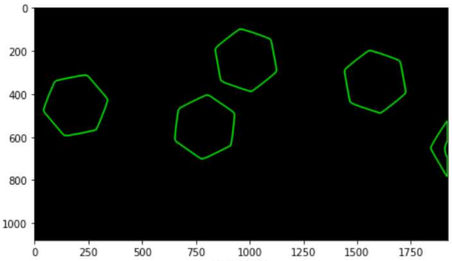
Frame 133



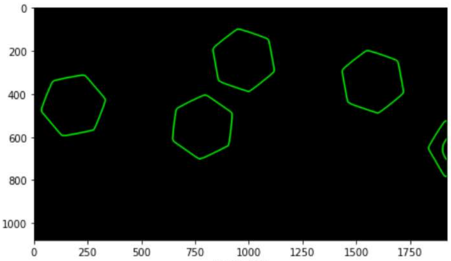
Frame 134



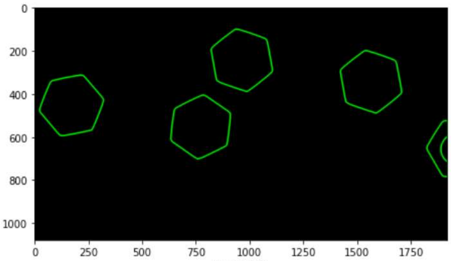
Frame 135



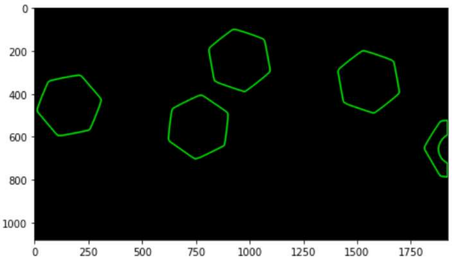
Frame 136



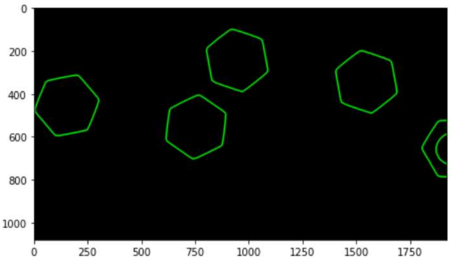
Frame 137



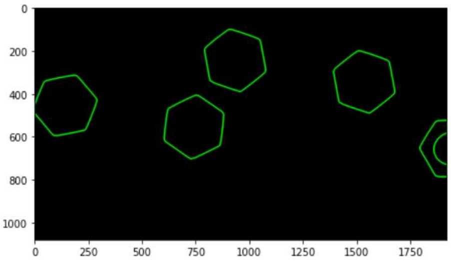
Frame 138



Frame 139



Frame 140



Frame 141

Detect the nuts ,details frame by frame

```
every_frame = []
for gray in gray_frames:

   retval, labels, stats, centroids = get_indexed_image(gray)
   belt = ((labels >= 1)*255).astype('uint8')
   contours,x  = cv.findContours(belt, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)

   count = 0
   frame = []

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

        if metric <= 0.5: # instruction
            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]))
    #the count of the last contour in the frame wil be the total nuts in that frame
   every_frame.append(frame)
print("Extraction completed.")
```

```
Extraction completed.
```

Finding the total nuts

```
total_nuts = int(every_frame[0][-1][-1])

delta_x = np.array([15])
i = np.array([0])

prev_frame = every_frame[0]

for frame_num in range(1, len(every_frame)):
    current = every_frame[frame_num] #The frame to be compared

    for nut in current:
        if is_new(prev_frame, nut, delta_x, i):
            total_nuts +=1  #count only if it's a new nut
            nut[-1] = total_nuts
        else:
            nut_index = prev_index(prev_frame, nut, delta_x, i)

            nut[-1]=prev_frame[int(nut_index)][-1]

    prev_frame = current

    #current frame is set to be the previous frame for the next frame
print("Total = ",total_nuts)
```

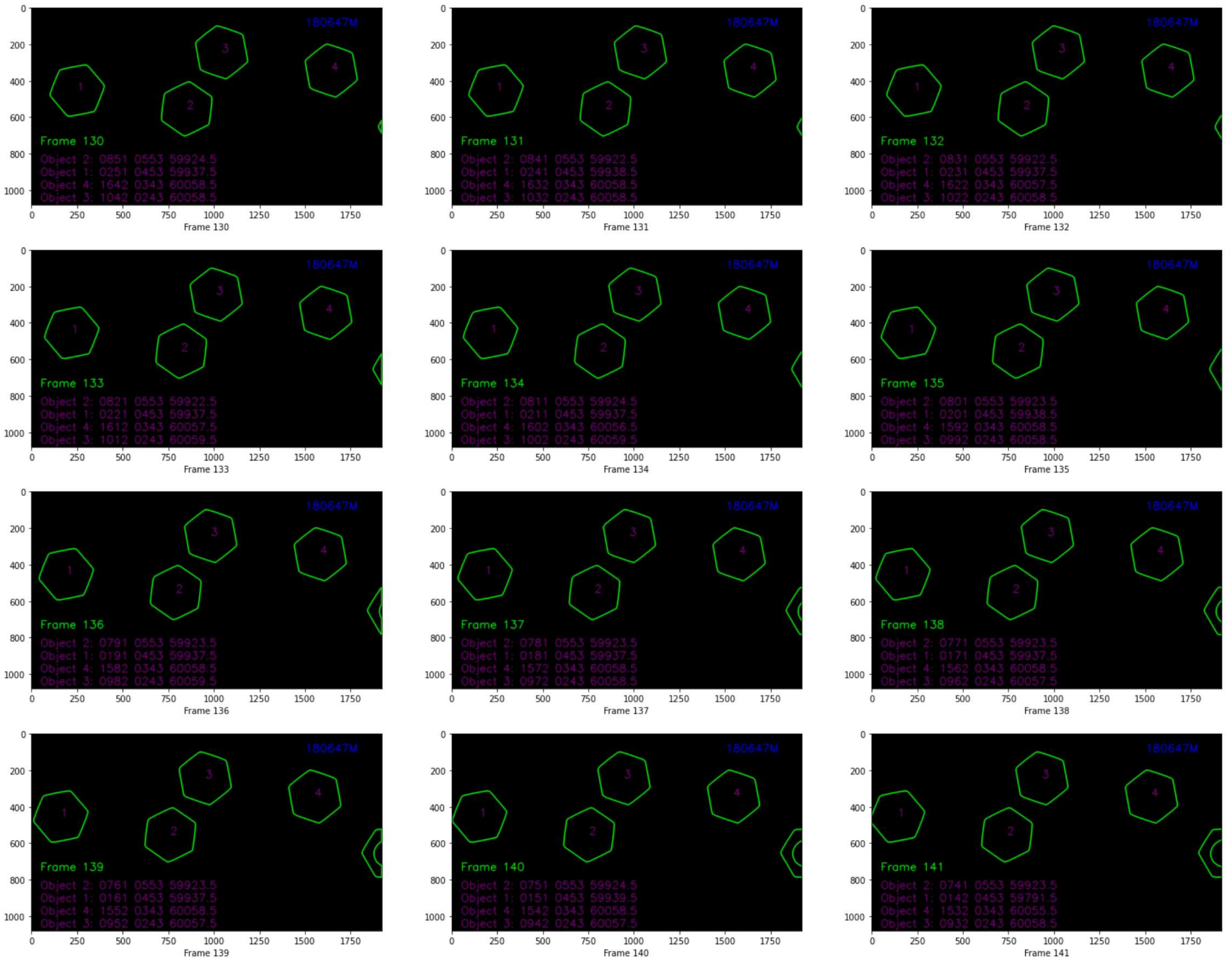
```
(array([1]), array([0]))
(array([0]), array([0]))
(array([1]), array([0]))
(array([0]), array([0]))
(array([1]), array([0]))
(array([0]), array([0]))
```

```
(array([1]), array([0]))
(array([0]), array([0]))
(array([1]), array([0]))
(array([0]), array([0]))
(array([1]), array([0]))
(array([0]), array([0]))
(array([1]), array([0]))
(array([0]), array([0]))
(array([1]), array([0]))
(array([0]), array([0]))
(array([1]), array([0]))
(array([0]), array([0]))
(array([1]), array([0]))
(array([0]), array([0]))
(array([1]), array([0]))
(array([0]), array([0]))
(array([1]), array([0]))
(array([0]), array([0]))
(array([1]), array([0]))
(array([0]), array([0]))
(array([1]), array([0]))
(array([0]), array([0]))
(array([1]), array([0]))
(array([0]), array([0]))
(array([1]), array([0]))
(array([0]), array([0]))
(array([1]), array([0]))
(array([0]), array([0]))
(array([1]), array([0]))
(array([0]), array([0]))
(array([1]), array([0]))
(array([0]), array([0]))
(array([1]), array([0]))
(array([0]), array([0]))
(array([1]), array([0]))
(array([0]), array([0]))
(array([1]), array([0]))
(array([0]), array([0]))
(array([1]), array([0]))
(array([0]), array([0]))
(array([1]), array([0]))
(array([0]), array([0]))
(array([1]), array([0]))
(array([0]), array([0]))
(array([1]), array([0]))
(array([0]), array([0]))
(array([1]), array([0]))
(array([0]), array([0]))
(array([1]), array([0]))
(array([0]), array([0]))
(array([1]), array([0]))
(array([0]), array([0]))
(array([1]), array([0]))
Total =  5
```

```
annotated_frames =[]
frame_num = 0
print("Frame annotation is in progress...")
for frame,contour_plot, contours in zip(every_frame, contour_plots, contours_list):

    img = contour_plot
    y = 0     # to change the position
    for nut in frame:

        # Annotate the index of the nut
        # Annotate the index of the nut
        img = cv.putText(img, str(int(nut[-1])),\
               (int(nut[0]),int(nut[1])),cv.FONT_HERSHEY_SIMPLEX, 2, (128,0,128), 4)

        # Annotate the Connected componets' details
        img = cv.putText(img, "Object {}: {:04} {:04} {:05}".format(int(nut[-1]), int(nut[0]), int(
                         (50,850 + 70*y), cv.FONT_HERSHEY_SIMPLEX, 2, (128,0,128), 4)

        y +=1     # to change the position

    # Annotate frame number
    img = cv.putText(img, "Frame "+str(frame_num) , (50,750) , cv.FONT_HERSHEY_SIMPLEX, 2, (0,255,0
    # Draw the contours
    img = cv.drawContours(img, contours, -1, (0,255,0), 5).astype('uint8')

    img = cv.putText(img, "180647M" , (1500,100) , cv.FONT_HERSHEY_SIMPLEX, 2, (255,0,0), 3) #Annota
    annotated_frames.append(img)
    frame_num +=1
print("Annotation completed.")
```

```
Frame annotation is in progress...
Annotation completed.
```

```
plt.figure(figsize=(25,20))
for i in range(12):
    plt.subplot(4,3,i+1)
    plt.imshow(annotated_frames[130 +i][:,:,::-1])
    plt.xlabel("Frame " + str(130 +i))
plt.show()
```



```
Frame 130
Object 2: 0851 0553 59924.5
Object 1: 0251 0453 59937.5
Object 4: 1642 0343 60058.5
Object 3: 1042 0243 60058.5

Frame 131
Object 2: 0841 0553 59922.5
Object 1: 0241 0453 59938.5
Object 4: 1632 0343 60058.5
Object 3: 1032 0243 60058.5

Frame 132
Object 2: 0831 0553 59922.5
Object 1: 0231 0453 59937.5
Object 4: 1622 0343 60057.5
Object 3: 1022 0243 60058.5

Frame 133
Object 2: 0821 0553 59922.5
Object 1: 0221 0453 59937.5
Object 4: 1612 0343 60057.5
Object 3: 1012 0243 60059.5

Frame 134
Object 2: 0811 0553 59924.5
Object 1: 0211 0453 59937.5
Object 4: 1602 0343 60056.5
Object 3: 1002 0243 60059.5

Frame 135
Object 2: 0801 0553 59923.5
Object 1: 0201 0453 59938.5
Object 4: 1592 0343 60058.5
Object 3: 0992 0243 60058.5

Frame 136
Object 2: 0791 0553 59923.5
Object 1: 0191 0453 59937.5
Object 4: 1582 0343 60058.5
Object 3: 0982 0243 60059.5

Frame 137
Object 2: 0781 0553 59923.5
Object 1: 0181 0453 59937.5
Object 4: 1572 0343 60058.5
Object 3: 0972 0243 60058.5

Frame 138
Object 2: 0771 0553 59923.5
Object 1: 0171 0453 59937.5
Object 4: 1562 0343 60058.5
Object 3: 0962 0243 60057.5

Frame 139
Object 2: 0761 0553 59923.5
Object 1: 0161 0453 59937.5
Object 4: 1552 0343 60058.5
Object 3: 0952 0243 60057.5

Frame 140
Object 2: 0751 0553 59924.5
Object 1: 0151 0453 59939.5
Object 4: 1542 0343 60058.5
Object 3: 0942 0243 60057.5

Frame 141
Object 2: 0741 0553 59923.5
Object 1: 0142 0453 59791.5
Object 4: 1532 0343 60055.5
Object 3: 0932 0243 60058.5
```

```
output = '180647M_en2550_a05.mp4'
fourcc = cv.VideoWriter_fourcc(*'MP4V')
duration = 9
fps = int(len(annotated_frames)/duration) # frame per second
height, width,_ = annotated_frames[0].shape
frame_size = (width, height)
isColor = True

out = cv.VideoWriter(output, fourcc, fps, frame_size, isColor)
for frame in annotated_frames:
    out.write(frame)
out.release()
print("Video writing completed.")
```

```
Video writing completed.
```

✓ 3s completed at 12:32 PM ● ×

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.