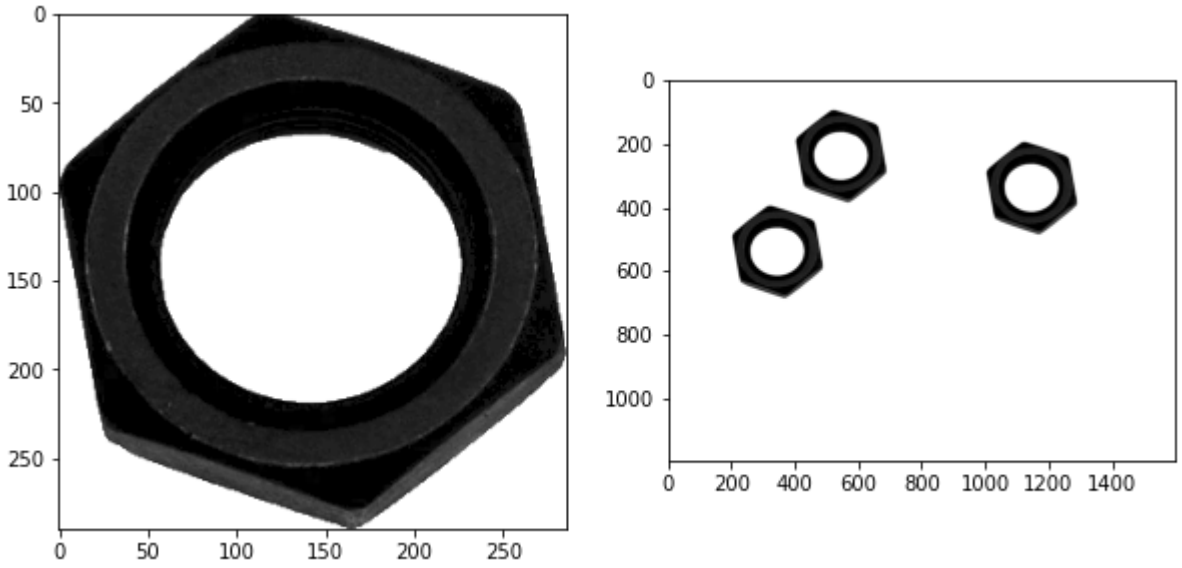
Index Number = 180411K

EN2550 2021: Object Counting on a Convey Belt

Submitted on: July 10, 2021

In [267…
```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

In [268…
```
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 1

Otsu's thresholding

In [269…
```
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)
```

Morphological closing

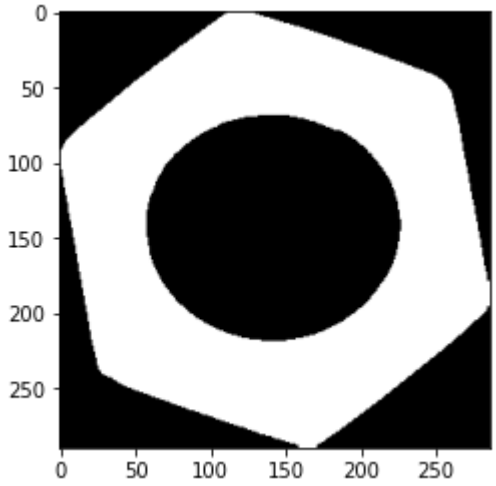
In [270…
```
kernel=np.ones((3,3),np.uint8)
closing_t = cv.morphologyEx(img_t, cv.MORPH_CLOSE, kernel) # Dilation followed by Erosi
closing_b = cv.morphologyEx(img_b, cv.MORPH_CLOSE, kernel)
```

Connected component analysis

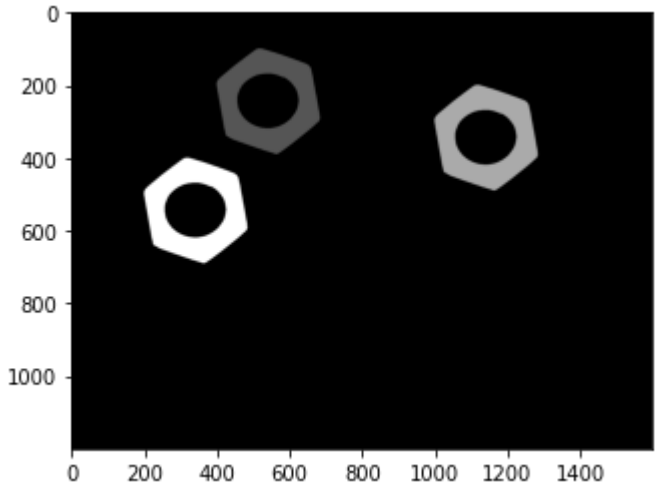
In [271…

```
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('........Template image.......')
print("No.of connected components = ", retval_t)
plt.imshow(labels_t.astype('uint8'), cmap ='gray')
plt.show()
print("statistics = ",'\n', stats_t)
print("Centroids = ",'\n', centroids_t)

print('.........Belt image.........')
print("No.of connected components = ", retval_b)
plt.imshow(labels_b.astype('uint8'), cmap ='gray')
plt.show()
print("statistics = ",'\n', stats_b)
print("Centroids = ",'\n', centroids_b)
```

```
........Template image.......
No.of connected components =  2
```




```
statistics =
 [[    0     0   286   290 42290]
 [    0     0   286   290 40650]]
Centroids =
 [[142.18770395 145.19172381]
 [142.82489545 143.780369  ]]
.........Belt image.........
No.of connected components =  4
```




```
statistics =
 [[      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 components are detected in each image?

Template Image = 2 (including background)

Belt Image = 4 (including background)

What are the statistics? Interpret these statistics.

Statistics give the properties of the connected component such as the horizontal and verical size of bounding box, total area of connected component and many more. Statistics are accessed via stats(label, COLUMN). Where COLUMN is one of ConnectedComponentsTypes, selecting the statistic.

Col 1: cv.CC_STAT_LEFT: the leftmost (x) coordinate which is the inclusive start of the bounding box in the horizontal direction.

Col 2: cv.CC_STAT_TOP: the topmost (y) coordinate which is the inclusive start of the bounding box in the vertical direction.

Col 3: cv.CC_STAT_WIDTH: the horizontal size of the bounding box.

Col 4: cv.CC_STAT_HEIGHT: the vertical size of the bounding box.

Col 5: cv.CC_STAT_AREA: the total area (in pixels) of the connected component.

What are the centroids?

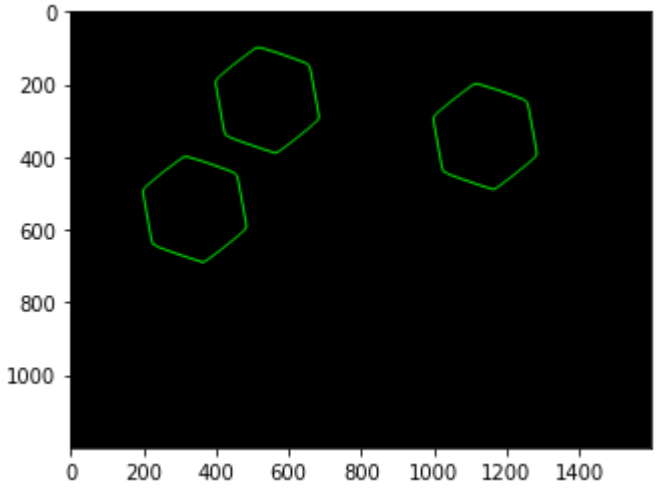
Each row gives the (x,y) coordinates of centroid of the corresponding connected component.

Find contours

In [272…

```
#cv.CHAIN_APPROX_SIMPLE removes all redundant points and compresses the contour, thereb
# here we only recieve outer contours
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
#Contours is a Python list of all the contours in the image. Each individual contour is
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)
plt.show()

label = 1 # remember that the label of the background is 0
belt = ((labels_b >= label)*255).astype('uint8')
NoOfNuts=0
belt_cont, template_hierarchy = cv.findContours(belt, cv.RETR_EXTERNAL, cv.CHAIN_APPROX
```

```
for j,c in enumerate(belt_cont):
    print('Smilarity with contour ',j+1,' = ',cv.matchShapes(contours_t[0], c, cv.CONTO
    NoOfNuts+=1
    #the lower the result, the better match it is. It is calculated based on the hu-mom
print ('Number of Nuts =',NoOfNuts)
```




```
Smilarity with contour  1  =  0.00010071698397173812
Smilarity with contour  2  =  0.00010071698397950968
Smilarity with contour  3  =  0.00010071698397506879
Number of Nuts = 3
```

Part 2

Frame tracking through image moments.

In [273…
```
cnt=contours_b[1]
ca = cv.contourArea(cnt)
print('area = ', ca)
```

```
area =  60059.5
```

In [274…
```
M = cv.moments(cnt)
cx = int(M['m10']/M['m00'])# extract x coordinates of the centroid of contours_b[1]
cy = int(M['m01']/M['m00'])#extract y coordinates of the centroid of contours_b[1]
print('cx = ',cx,'cy = ', cy)
```

```
cx =  1142 cy =  343
```

In [275…
```
count=1
object_prev_frame = np.array([cx,cy,ca,count])
```

In [276…
```
delta_x = 15
```

Part 3

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 [277…
```
def get_indexed_image(im):
    th, img = cv.threshold(im,0,255,cv.THRESH_BINARY_INV+cv.THRESH_OTSU)
    kernel=np.ones((3,3),np.uint8)
    closing= cv.morphologyEx(img, cv.MORPH_CLOSE, kernel) # Dilation followed by Erosio
    retval, labels, stats, centroids = cv.connectedComponentsWithStats(closing)
    return retval, labels, stats, centroids
```

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

In [278…
```
def is_new(a, b, delta, i):
    for row in a: # iterating for each row in a
        dif_is_smaller=True
        for ind in range(i.shape[0]):
            index=i[ind] #colomn to be compared
            b_val= b[index] #corresponding value in b array
            delta_val=delta[index] #corresponding value in delta array
            if (np.abs(row[index]-b_val)) > delta_val:
                dif_is_smaller=False
        if dif_is_smaller: #We return false if there is atleast one value in any row si
            return False
    return True
```

In [279…
```
 # 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 "
```

3) . 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 [280…
```
def prev_index(a, b, delta, i):
    index = -1
    i_val=i[0]
    b_val=b[i_val]
    delta_val=delta[i_val]
    for row_index in range(a.shape[0]):
        if abs(a[row_index][i_val]-b_val)<delta_val:#check wether the differnce is belo
            return row_index
    return index
```

In [281…
```
# 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])
assert prev_index(a,b,delta,i) == 1, " Check the function "
```

load and access each frame of a video

In [282…
```
cap = cv.VideoCapture('conveyor_with_rotation.mp4') # give the correct path here
setOfFrames=[]
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        print("Can't receive frame (stream end?). Exiting ...")
        break
    setOfFrames.append(frame)
    if cv.waitKey(1) == ord('q'):
        break
cap.release()
cv.destroyAllWindows()
print('Finished getting frames')
```

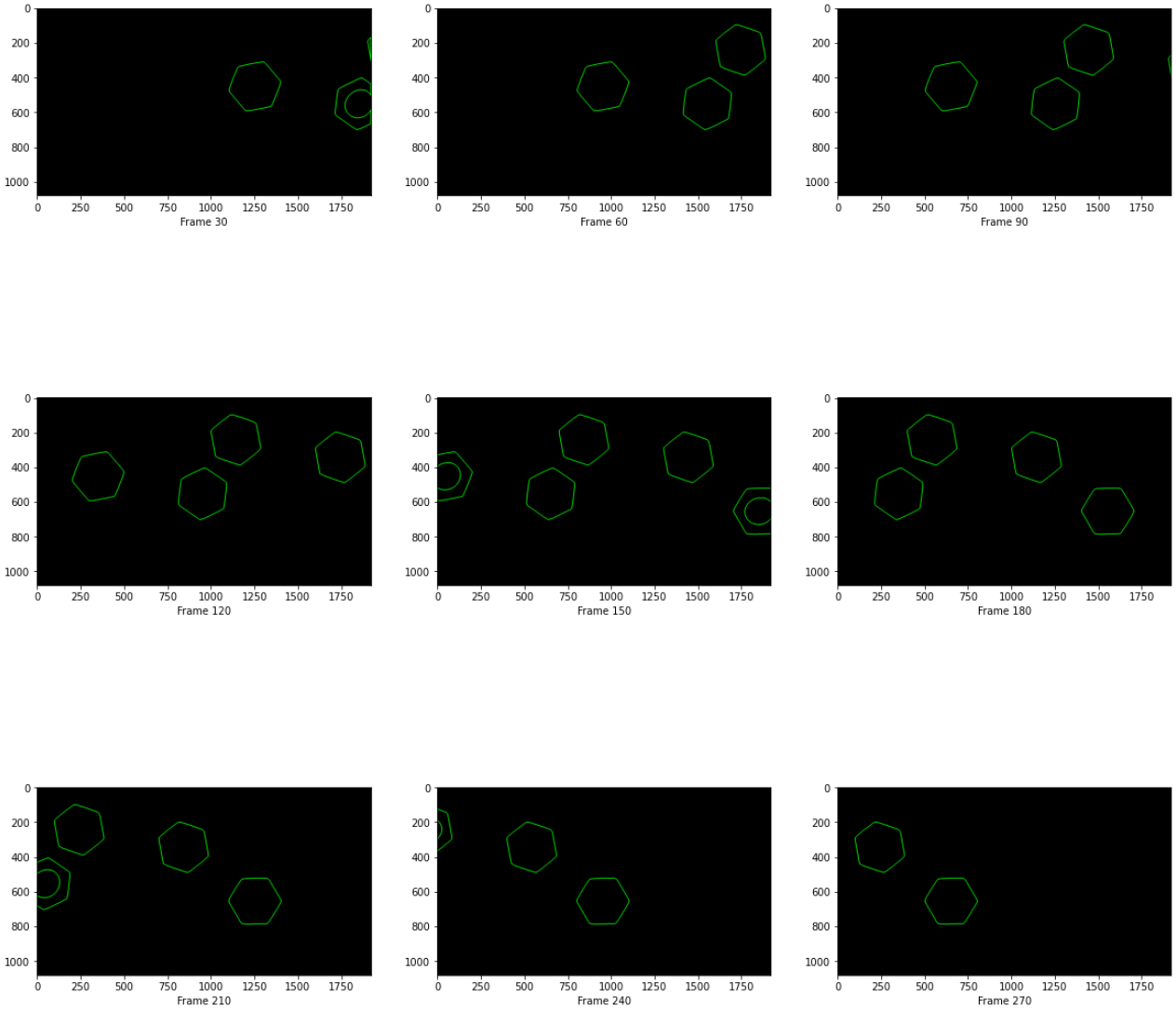
```
Can't receive frame (stream end?). Exiting ...
Finished getting frames
```

Part 4

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

In [283…
```
contour_Plots=[]
grayFrames=[]
#excecuting for each frame
for frame in setOfFrames:
    grey=cv.cvtColor(frame,cv.COLOR_BGR2GRAY) #convering to gray scale
    grayFrames.append(grey)
    retval, labels, stats, centroids= get_indexed_image(grey) # get label image
    belt = ((labels >= 1)*255).astype('uint8')
    belt_cont, template_hierarchy = cv.findContours(belt, cv.RETR_EXTERNAL, cv.CHAIN_AP
    im_contours_belt = np.zeros((belt.shape[0],belt.shape[1],3), np.uint8) #Initiate a
    conts = cv.drawContours(im_contours_belt, belt_cont, -1, (0,255,0), 3).astype('uint
    contour_Plots.append(conts)
```

In [284…
```
# visualize contour plots
plt.figure(figsize=(20,20))
increment=0
for i in range(1,len(setOfFrames)):
    if i%30 ==0: # As there are 280 frames I plot 9 out of them at 30 frame gap
        plt.subplot(3,3,increment+1)
        plt.imshow(contour_Plots[i])
        plt.xlabel("Frame " + str(i))
        increment+=1
plt.show()
```



Part 5

Object detection and tracking

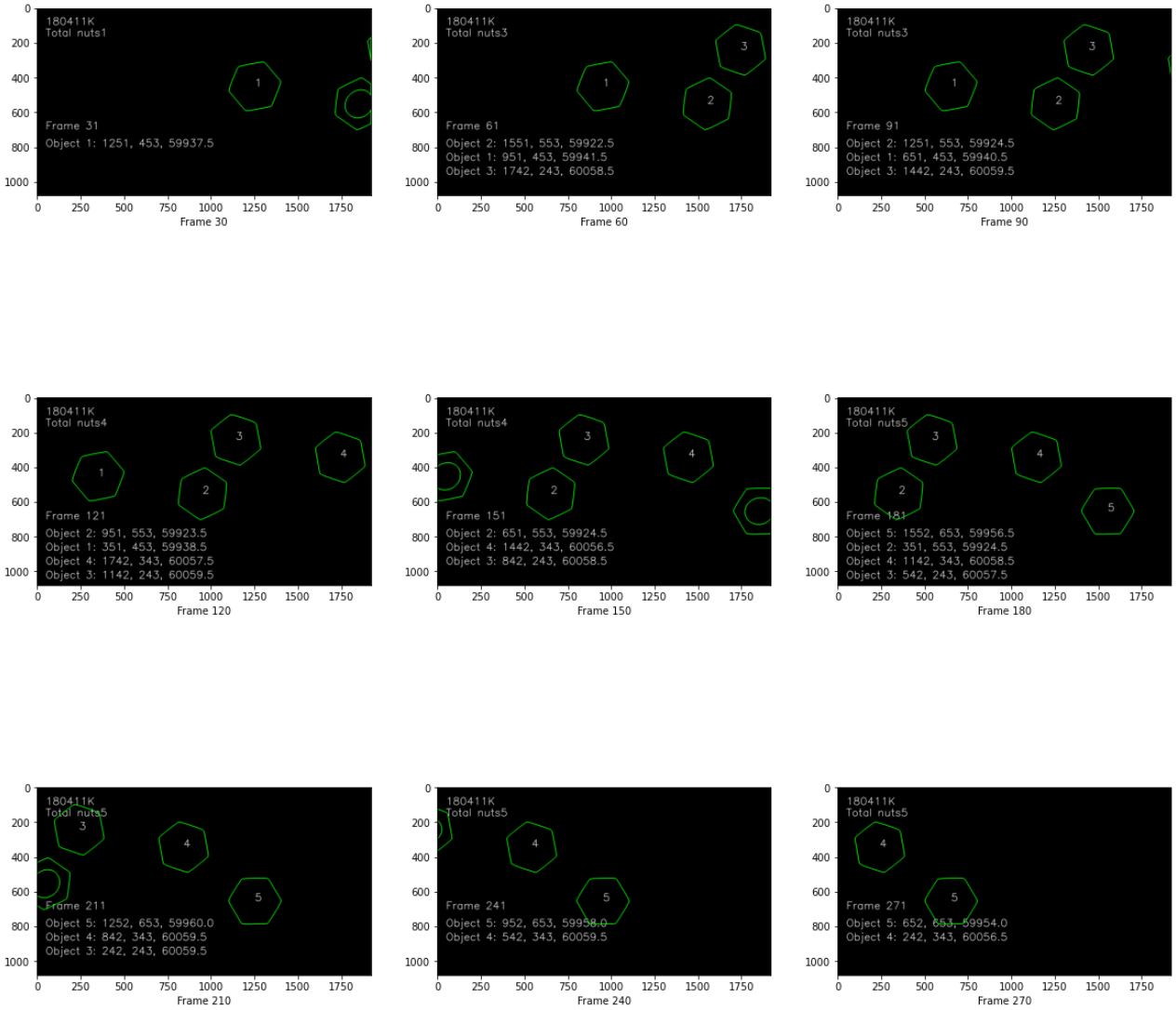
In [285…

```
#Initialize variables
a=np.array([])
i=np.array([0])
delta=np.array([15])
count=0# nut count
frame_num =0
font = cv.FONT_HERSHEY_SIMPLEX
annotated_set_of_frames=[]
#runing loop for every frame
for frame in grayFrames:
    retval, labels, stats, centroids= get_indexed_image(frame) # get label image
    belt = ((labels >= 1)*255).astype('uint8')
    belt_cont, template_hierarchy = cv.findContours(belt, cv.RETR_EXTERNAL, cv.CHAIN_AP
    im_contours_belt = np.zeros((belt.shape[0],belt.shape[1],3), np.uint8) #Initiate a
    place=0
    frame_num+=1
    for contour in belt_cont:
        val=cv.matchShapes(contours_t[0], contour, cv.CONTOURS_MATCH_I1, 0.0) # match c
        if val>0.5:  # Set matching threshold as 0.5
```

```
            continue

        M  = cv.moments(contour)
        ca = M['m00'] # area of the contour
        cx = int(M['m10']/M['m00']) #x and y coordinates of the centroid of contour
        cy = int(M['m01']/M['m00'])
        object_curr_frame = np.array([cx,cy,ca,count])

        if a.shape[0]==0: #If a is empty add the data of the first nut(contour)
            a=np.append(a,object_curr_frame).reshape(1,4)
            index=a.shape[0]-1
            count+=1
        elif is_new(a, object_curr_frame, delta, i): # if a new nut(contour) is detecte
            a=np.concatenate((a,np.array([object_curr_frame])),axis=0)
            index=a.shape[0]-1
            count+=1
        else:
            index=prev_index(a, object_curr_frame, delta, i) # If the detected nut (con
            a[index] = object_curr_frame

        # Annotating the frames with details
        cv.putText(im_contours_belt, str(index+1),(cx,cy),font, 2, (255,255,255),2,cv.L
        cv.putText(im_contours_belt, str('Object '+ str(index+1)+ ': '+ str(cx)+', '+ s
        place+=1
    cv.putText(im_contours_belt, "Frame "+str(frame_num) , (50,700) ,font, 2, (255,255,
    # Index number = 180411K
    cv.putText(im_contours_belt, "180411K" , (50,100) , font, 2, (255,255,255), 2,cv.LI
    cv.putText(im_contours_belt, str("Total nuts"+str(count)) , (50,165) , font, 2, (25
    annotated_frame = cv.drawContours(im_contours_belt, belt_cont, -1, (0,255,0), 3).as
    annotated_set_of_frames.append(annotated_frame)
```

In [286…

```
#Visualising annotated frames
plt.figure(figsize=(20,20))
increment=0
for i in range(1,len(annotated_set_of_frames)):
    if i%30 ==0: # As there are 280 frames i plot 9 out of them at 30 frame gap
        plt.subplot(3,3,increment+1)
        plt.imshow(annotated_set_of_frames[i])
        plt.xlabel("Frame " + str(i))
        increment+=1
plt.show()
```



In [292…

```
# Making output Video
height, width, depth= annotated_set_of_frames[0].shape
fps = len(annotated_set_of_frames)//9 # source video duration is 9 seconds. So, frames
FourCC = cv.VideoWriter_fourcc(*'MP4V')#FourCC is a 4-byte code used to specify the vid
Output = cv.VideoWriter('180411K_en2550_a05.mp4',FourCC,fps,(width,height))
for frame in annotated_set_of_frames:
    Output.write(frame)
Output.release()
cv.destroyAllWindows()
print('finished making video')
```

```
finished making video
```