Assignment 3

Karunanayaka Y.S.

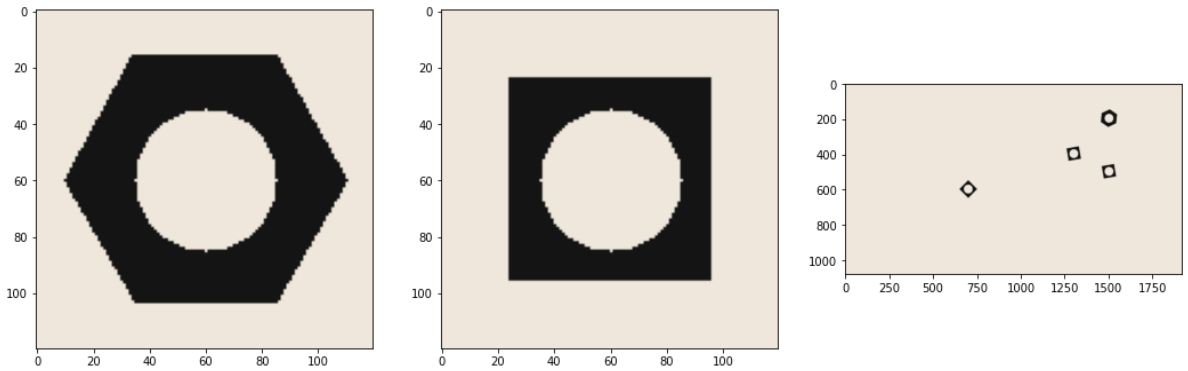
190301H

Github Link: https://github.com/YSK-Machine-Vision-EN2550/Assignment_3

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

Question 1

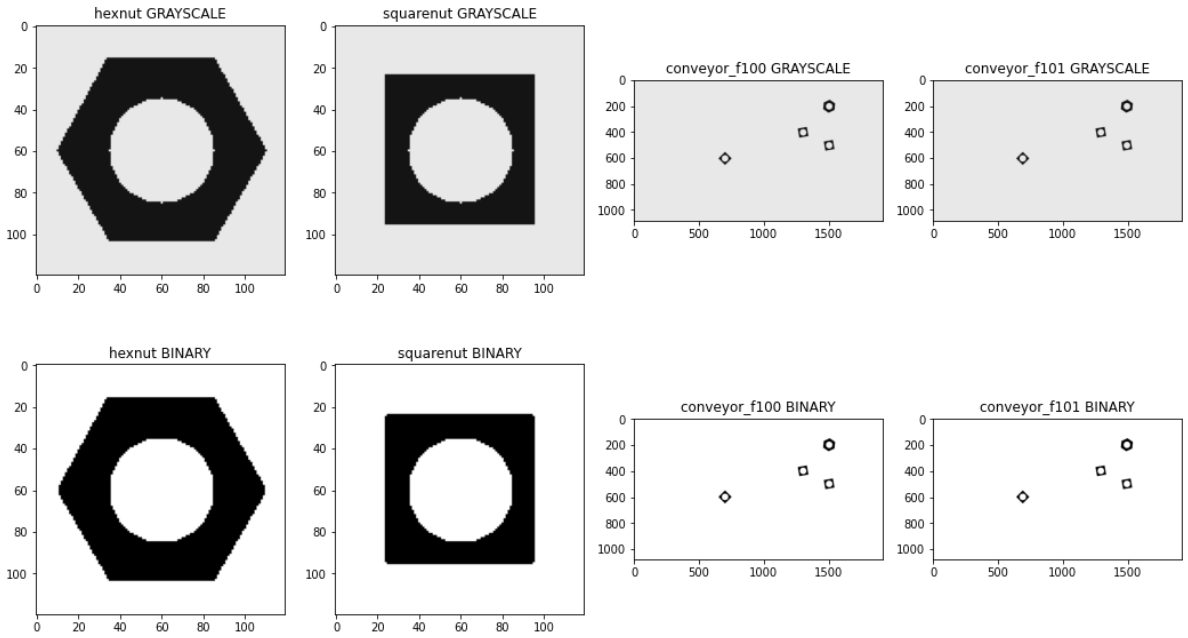
In [2]:
```
fig, ax = plt. subplots(1,3, figsize = (18, 9))

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)

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()
```



part A

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 [3]:
```
# add images and their display name to a list to use in matplotlib plotting
conveyor_f101 =  cv.imread('conveyor_f101.png', cv.IMREAD_COLOR)

images = [("hexnut", hexnut_template),
          ("squarenut", squarenut_template),
          ("conveyor_f100", conveyor_f100),
          ("conveyor_f101", conveyor_f101)]

binarized = [] # binarized images will be stroed here
thresholds = []

_, ax = plt.subplots(2,4, figsize = (18,10))

for index, (title, image) in enumerate(images):

    gray = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
    blur = cv.GaussianBlur(gray, (5,5), 0)
    threshold, binary_img = cv.threshold(blur,0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)
    binarized.append(binary_img)
    thresholds.append(threshold)

    ax[0, index].imshow(cv.cvtColor(gray, cv.COLOR_BGR2RGB))
    ax[0, index].set_title(title + " GRAYSCALE")

    ax[1, index].imshow(cv.cvtColor(binary_img, cv.COLOR_BGR2RGB))
    ax[1, index].set_title(title + " BINARY")

plt.show()

print("\n\nPrinting Threshold values for the images")
print("-----------------------------------------")

for index, threshold in enumerate(thresholds):
    print("{:13} threshold-> {}".format(images[index][0], threshold))
```

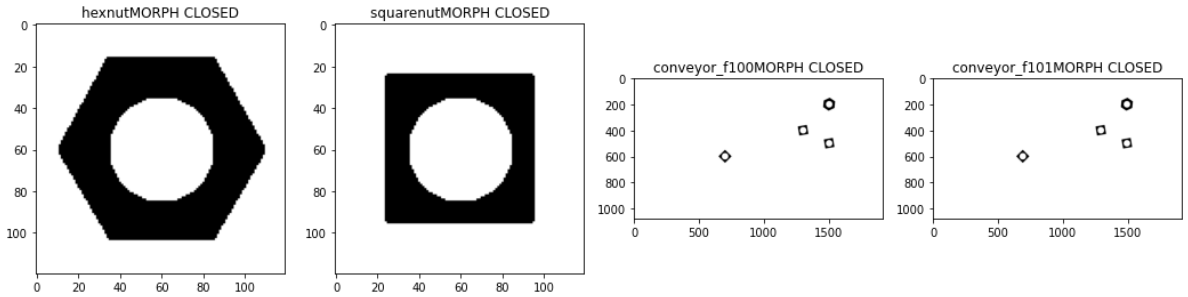



```
Printing Threshold values for the images
----------------------------------------
hexnut        threshold-> 116.0
squarenut     threshold-> 116.0
conveyor_f100 threshold-> 128.0
conveyor_f101 threshold-> 128.0
```

Part B

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

In [4]:
```
filter_kernel = np.ones((3,3),np.uint8)
new_binarized = []

_, ax = plt.subplots(1,4, figsize = (18,9))

for index, (title, image) in enumerate(images):

    filtered_image = cv.morphologyEx(binarized[index], cv.MORPH_CLOSE, filter_kernel)
    new_binarized.append(filtered_image)

    ax[index].imshow(cv.cvtColor(filtered_image, cv.COLOR_BGR2RGB))
    ax[index].set_title(title + "MORPH CLOSED")
plt.show()
```



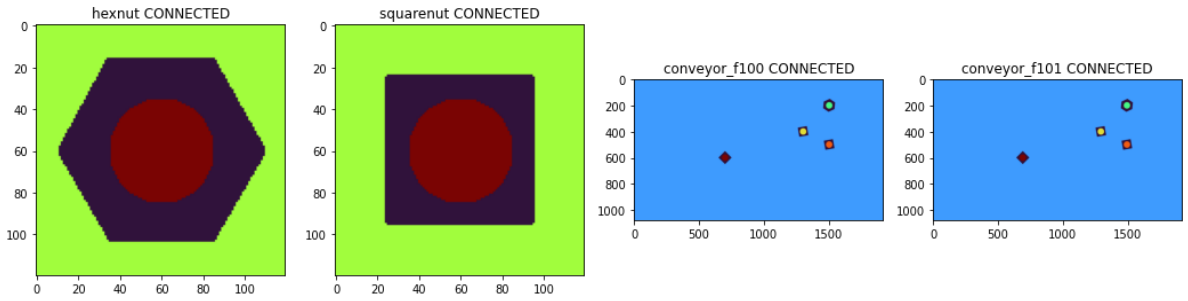
Part C

Connected components analysis: apply the connectedComponentsWithStats function (see https://docs.opencv.org/4.5.5/d3/dc0/group__imgproc__shape.html#ga107a78bf7cd25dec05fb4dfc5c9e765f) 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 [5]:
```
connected = []
extracted_data = {"titles" : [], "connected" : [], "stats" : [], "centroids" : []}

_, ax = plt.subplots(1,4,figsize = (18,9))

for index, (title, image) in enumerate(images):

    _, labels, stats, centroids = cv.connectedComponentsWithStats(new_binarized[index])
    colormapped = cv.applyColorMap((labels/np.amax(labels)*255).astype('uint8'), cv.COLORMAP_TURBO)
    connected.append(colormapped)

    ax[index].imshow(cv.cvtColor(colormapped,cv.COLOR_BGR2RGB))
    ax[index].set_title(title + " CONNECTED")

    ## add data to the dictionary so we can get those data later
    extracted_data['titles'].append(title)
    extracted_data['connected'].append(np.shape(stats)[0])
    extracted_data['stats'].append(stats)
    extracted_data['centroids'].append(centroids)

plt.show()
```



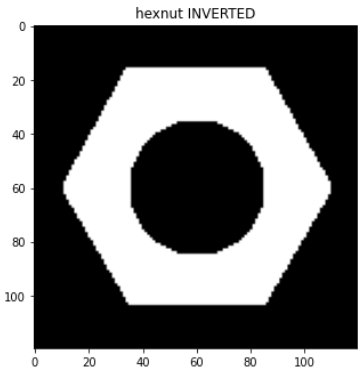
In [6]:
```
for index, title in enumerate(extracted_data['titles']):
    print((title + " Connected Componenet Details").center(120).upper())
    print("====================================".center(120))

    print("No of Components -> {}".format(extracted_data['connected'][index]))
    print("Stats            -> {}".format(extracted_data['stats'][index].ravel().tolist()     ))
    print("Centroids        -> {}\n\n".format(extracted_data['centroids'][index].ravel().tolist()))
```

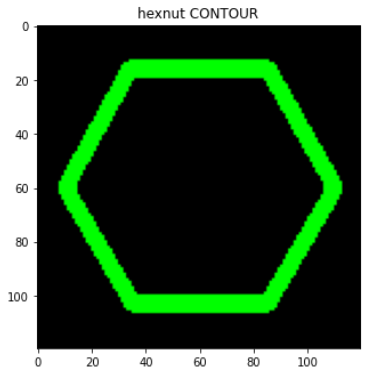
```
                                    HEXNUT CONNECTED COMPONENET DETAILS
                                    ====================================
No of Components -> 3
Stats            -> [11, 16, 99, 88, 4726, 0, 0, 120, 120, 7717, 36, 36, 49, 49, 1957]
Centroids        -> [59.83368599238256, 59.223233178163355, 59.168847997926655, 59.54269793961384, 60.0, 60.0]


                                    SQUARENUT CONNECTED COMPONENET DETAILS
                                    ====================================
No of Components -> 3
Stats            -> [24, 24, 72, 72, 3223, 0, 0, 120, 120, 9220, 36, 36, 49, 49, 1957]
Centroids        -> [59.196400868755816, 59.196400868755816, 59.5, 59.5, 60.0, 60.0]


                                  CONVEYOR_F100 CONNECTED COMPONENET DETAILS
                                    ====================================
No of Components -> 6
Stats            -> [651, 151, 895, 499, 13922, 0, 0, 1920, 1080, 2051850, 1476, 176, 49, 49, 1957, 1276, 376, 49, 49, 1957, 1476, 476, 49, 49, 1957, 676, 576, 49, 49, 1957]
Centroids        -> [1274.7777618158311, 400.0543025427381, 956.2525252820625, 540.8829807247118, 1500.0, 200.0, 1300.0, 400.0, 1500.0, 500.0, 700.0, 600.0]


                                  CONVEYOR_F101 CONNECTED COMPONENET DETAILS
                                    ====================================
No of Components -> 6
Stats            -> [641, 151, 895, 499, 13922, 0, 0, 1920, 1080, 2051850, 1466, 176, 49, 49, 1957, 1266, 376, 49, 49, 1957, 1466, 476, 49, 49, 1957, 666, 576, 49, 49, 1957]
Centroids        -> [1264.7777618158311, 400.0543025427381, 956.3585271827862, 540.8829807247118, 1490.0, 200.0, 1290.0, 400.0, 1490.0, 500.0, 690.0, 600.0]
```

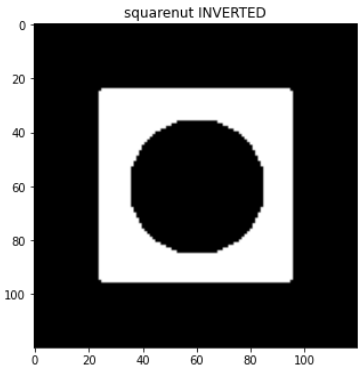
Part D

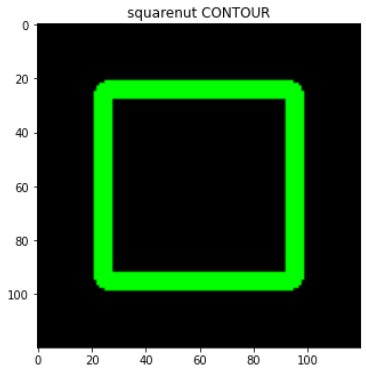
Contour analysis: Use findContours function to retrieve the extreme outer contours. (see https://docs.opencv.org/4.5.2/d4/d73/tutorial_py_contours_begin.html for help and https://docs.opencv.org/4.5.2/d3/dc0/group__imgproc__shape.html#gadf1ad6a0b82947fa1fe3c3d497f260e0 for information.

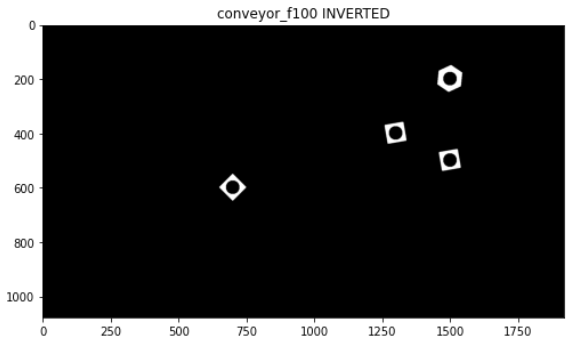
In [7]:
```
counturs_list = []

_, ax = plt.subplots(4,2, figsize = (18,24))

for index, (title, image) in enumerate(images):

    # cannot directly convert to grayscale. Do it in steps
    inverted = cv.cvtColor(255 - new_binarized[index], cv.COLOR_RGB2BGR) # inverting Binary Image by 255 - IM
    inverted = cv.cvtColor(inverted, cv.COLOR_BGR2GRAY)

    temp_contours = cv.findContours(inverted, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)

    if len(temp_contours) == 2:
        contours = temp_contours[0]
    else:
        contours = temp_contours[1]

    counturs_list.append(contours)

    background = np.zeros(image.shape, dtype='uint8')
    for contour_point in contours:
        cv.drawContours(background, [contour_point], -1, (0,255,0), 5)


    ax[index][0].imshow(cv.cvtColor(inverted, cv.COLOR_RGB2BGR))
    ax[index][0].set_title(title + " INVERTED")

    ax[index][1].imshow(cv.cvtColor(background, cv.COLOR_RGB2BGR))
    ax[index][1].set_title(title + " CONTOUR")


plt.show()
```

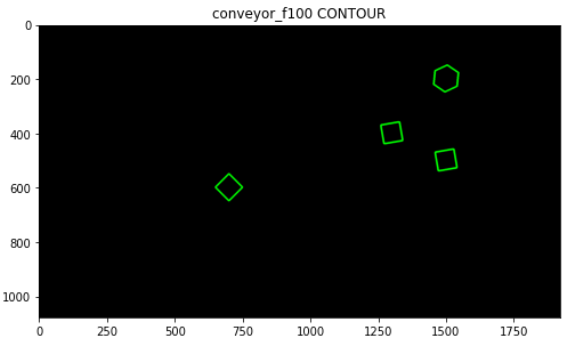


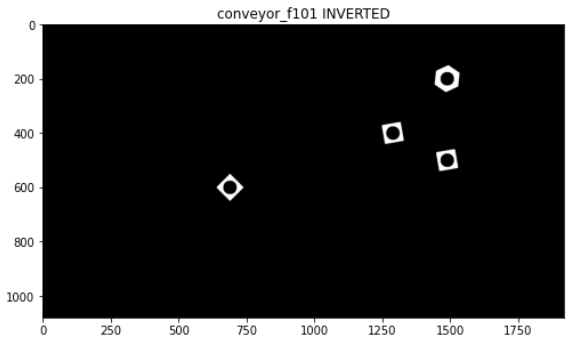


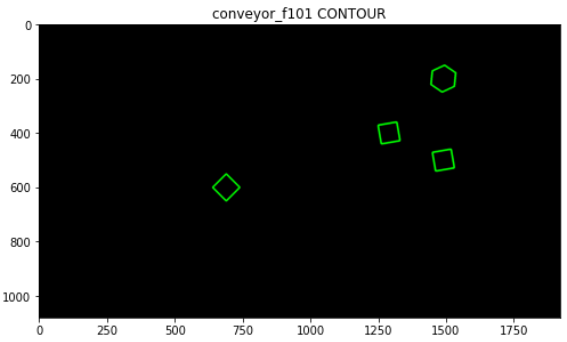












Question 2

Part A

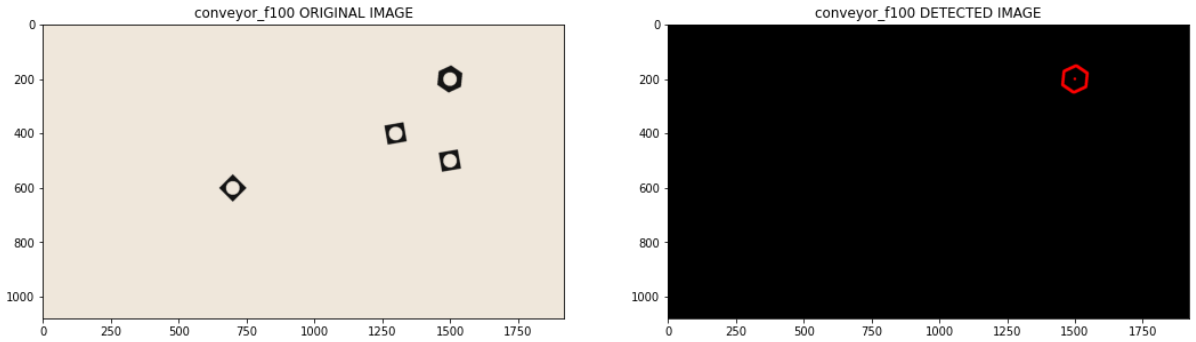
In this section, we will use the synthetic conveyor.mp4 sequence to count the two types of nuts.

In [8]:
```
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.LINE_AA)
    cv.imshow('Conveyor', frame)

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

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

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

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.

In [9]:
```
# extract the reference contours
hex_contour = counturs_list[0][0]
sqr_contour = counturs_list[1][0]
counter = 0

background = np.zeros(images[2][1].shape, dtype='uint8')

for temp_contour in counturs_list[2]:
    # get the error of matching contours
    error = cv.matchShapes(hex_contour, temp_contour, 1, 0.0)

    if error < 1e-3:
        counter += 1 # increment the counter
        cv.drawContours(background, [temp_contour], -1, (0,0,255), 10) # draw the contour

        # Extract centroid Information
        M = cv.moments(temp_contour)
        centroid = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"]))
        background = cv.circle(background, centroid, radius=0, color=(0, 0, 255), thickness=10)


_, ax = plt.subplots(1,2,figsize = (18,9))
ax[0].imshow(cv.cvtColor(images[2][1] , cv.COLOR_BGR2RGB))
ax[0].set_title(images[2][0] + " ORIGINAL IMAGE")
ax[1].imshow(cv.cvtColor(background , cv.COLOR_BGR2RGB))
ax[1].set_title(images[2][0] + " DETECTED IMAGE")
plt.show()

print("{} number of hexogonal contours detected".format(counter))
```




```
1 number of hexogonal contours detected
```

Part B

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 [14]:
```
object_list = []
MIN_DELTA_DISTANCE = 15

def object_lock(object_point):
    global object_list
    found_status = False

    for index, point in enumerate(object_list):
        distance = np.sqrt((point[0] - object_point[0]) ** 2 + (point[1] - object_point[1]) ** 2)

        if distance < MIN_DELTA_DISTANCE:
            # means found a frame before with the same nut
            object_list[index] = object_point # update with new point
            found_status = True
            break

    if not found_status:
        object_list.append(object_point)
        found_status = False

    return int(not found_status)
```

In [27]:
```
frame_array = []
shape = (1080, 1920, 3)

cv.namedWindow('Conveyor', cv.WINDOW_NORMAL)
cap = cv.VideoCapture('conveyor.mp4')
f = 0
Hex_nuts = 0
Sqr_nuts = 0

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

    # lets do the processing to identify the contours
    gray_frame = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
    blur_frame = cv.GaussianBlur(gray_frame, (5,5), 0)
    threshold, binary_frame = cv.threshold(blur_frame,0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)
    filtered_frame = cv.morphologyEx(binary_frame, cv.MORPH_CLOSE, filter_kernel)

    inverted_frame = cv.cvtColor(255 - filtered_frame, cv.COLOR_RGB2BGR) # inverting Binary Image by 255 - IM
    inverted_frame = cv.cvtColor(inverted_frame, cv.COLOR_BGR2GRAY)

    temp_contours = cv.findContours(inverted_frame, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)

    if len(temp_contours) == 2:
        contours = temp_contours[0]
    else:
        contours = temp_contours[1]

    for temp_contour in contours:
        # get the error of matching contours
        Hex_error = cv.matchShapes(hex_contour, temp_contour, 1, 0.0)
        Sqr_error = cv.matchShapes(sqr_contour, temp_contour, 1, 0.0)
        # Extract centroid Information
        M = cv.moments(temp_contour)
        centroid = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"]))

        if Hex_error < 1e-3:
            cv.drawContours(frame, [temp_contour], -1, (0,0,255), 10) # draw the contour
            frame = cv.circle(frame, centroid, radius=0, color=(0, 0, 255), thickness=10)
            Hex_nuts += object_lock(centroid)

        if Sqr_error < 1e-3:
            cv.drawContours(frame, [temp_contour], -1, (0,255,0), 10) # draw the contour
            frame = cv.circle(frame, centroid, radius=0, color=(0, 255, 0), thickness=10)
            Sqr_nuts += object_lock(centroid)


    f += 1
    cv.putText(frame, f"Frame      : {f}", (50, 50), cv.FONT_HERSHEY_COMPLEX, 1, (0,46,245), 1, cv.LINE_AA)
    cv.putText(frame, f"Hex Nuts   : {Hex_nuts}" , (50, 80), cv.FONT_HERSHEY_COMPLEX, 1, (0,46,245), 1, cv.LINE_AA)
    cv.putText(frame, f"Squar Nuts : {Sqr_nuts}" , (50, 110), cv.FONT_HERSHEY_COMPLEX, 1, (0,46,245), 1, cv.LINE_AA)

    cv.putText(frame, f"Assignment 3 (190301H)" , (600, 80), cv.FONT_HERSHEY_COMPLEX, 2, (0,46,245), 2, cv.LINE_AA)

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

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

cap.release()
cv.destroyAllWindows()

out = cv.VideoWriter('./conveyor_result_190301H.mp4',cv.VideoWriter_fourcc(*'h264'), 30, (shape[1], shape[0]))
for i in range(len(frame_array)):
    out.write(frame_array[i])
out.release()
```

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

In [ ]:

In [ ]: