EN2550_Assignment5

May 20, 2021

---

1 EN2550 2021: Object Counting on a Convey Belt

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

1.0.1 Let's first import required libraries

[2]:
```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
from google.colab.patches import cv2_imshow


%matplotlib inline
```

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

[ ]:
```
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 :

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

1.1.1 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**).

[ ]:
```
th_t, img_t = cv.threshold(template_im,0,255,cv.THRESH_BINARY_INV+cv.
↪THRESH_OTSU)
#th_b, img_b = "< Your code to apply thresholding to the belt_im>"
```

1.1.2 Morphological closing

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

[ ]:
```
kernel = #"< 3x3 matrix with all ones, with uint8 dtype>"
closing_t = cv.morphologyEx(img_t, cv.MORPH_CLOSE, kernel)
#closing_b = "< Your code to apply morphological closing for belt >"
```

1.1.3 Connected component analysis

Apply the `connectedComponentsWithStats` function (see this).

[ ]:
```
retval_t, labels_t, stats_t, centroids_t = cv.
↪connectedComponentsWithStats(closing_t)
retval_b, labels_b, stats_b, centroids_b = #"< Your code to perform connected␣
↪component analysis for closing_b>"
```

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

1.1.4 Contour analysis

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

Display these countours.

[ ]:
```
contours_t, hierarchy_t = cv.findContours(closing_t, cv.RETR_TREE, cv.
↪CHAIN_APPROX_SIMPLE)
contours_b, hierarchy_b = #"< Your code to perform counter analysis for␣
↪closing_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)
```

1.1.5 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.

[ ]:
```
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(cv.matchShapes(contours_t[0], c, cv.CONTOURS_MATCH_I1, 0.0))
```

1.2 Part - II

1.2.1 Frame tracking through image moments.

Use the `cv.contourArea()`, see this and calculate the the area of the `contours_b[1]`

[ ]:
```
ca = '<Your code goes here>'
```

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

[ ]:
```
M = '<Your code goes here>'
cx, cy = '<Your code goes here>'
```

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`

[ ]:
```
object_prev_frame = '< Your code here >'
```

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)

[ ]:
```
delta_x = '< Your code goes here >'
```

1.3 Part - III

1.3.1 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)

```
[ ]: def get_indexed_image(im):
         """ Thresholding, closing, and connected component analysis lumped
         """

         '< Your code goes here. Approximately 4 lines >'

         return retval, labels, stats, centroids
```

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

```
[ ]: 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 specified in delta.
         vector i specifies which elements in b to compare with those in a.
         """

         'Check whether the absolute different between all the elements of ith␣
      ↪column of each array is greater than the ith delta value (See thee example␣
      ↪in the next cell)'

         return None
```

```
[ ]: # 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 "
```

1.3.3 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)

```
[ ]: 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 >'
         return index
```

```
[ ]: # 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 "
```

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

```
[ ]: cap = cv.VideoCapture('conveyor_many_frame.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
         cv2_imshow(frame)
         if cv.waitKey(1) == ord('q'):
             break

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

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

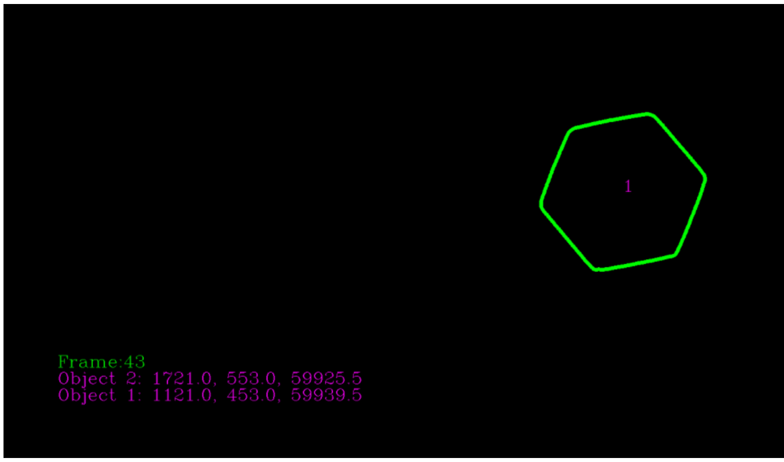
1.4 Steps:

1. Use the above code snippet to access each frame and remember to convert the frame into grey scale. Name the variable as `grey`

2. Call `get_indexed_image` and extract `retval, labels, stats, centroids`.
3. Find contours of all nuts present in a given frame of the belt.
4. Initiate a 3-D array with zeros to draw contours. Call this `im_contours_belt`
5. Draw each contour. Use `cv.drawContours`. See this

1.5 Object detection and tracking

For each contour of the belt frame,

1. Use `is_new` and `prev_index` functions to track each frame and get the indices of each nut.
2. Write a code to detect and track hexagonal nuts in each frame.
3. You may refer, annotation to understand how to add texts and labels to each frame.
4. Output for a random frame would be as follows 

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*

```
[ ]: '< Your code goes here >'
```