**King Faisal University**

**College of Computer Sciences and Information Technology**



**Application that Detect glass bottle of juice and determine juice level**

**Student Name:** Abdulaziz Abdulhamid Alnaim

**Student ID:** 211513441

**Course Name:** Computer Vision

**Instructor:** Dr. Mohammed Bilal

**Report**

**Project name:** Detect glass bottle of juice, and determine juice level, full, half full, or empty.

**System requirements:**

* Anaconda framework which includes many web programming languages like HTML, javascript, python programming language, Jupyter notebook, anaconda prompts and others tools.
* Opencv library.
* Any internet browser like Microsoft edge, Google chrome.

**The program processor**

1. Import the required libraries or packages: opencv, matplotlib.pyplot, and numpy
2. **import** cv2 as cv
3. **import** matplotlib.pyplot as plt
4. **import** numpy as np
5. Read the original image, applying initial step of mean shift segmentation of an image.
6. img = cv.imread('full.jpg')
7. blurred=cv.pyrMeanShiftFiltering(img,31,91)
8. Convert the result image to gray level image and enhance image edges with canny edge detection algorithm
9. gray = cv.cvtColor(blurred,cv.COLOR\_BGR2GRAY)
10. edges = cv.Canny(gray,50,250)
11. gray=edges
12. Convert the result image to binary equivalent with thresholding depends on Otsu’s algorithm
13. ret, thresh = cv.threshold(gray,150,255,cv.THRESH\_BINARY\_INV+cv.THRESH\_OTSU)



Figure :Original image

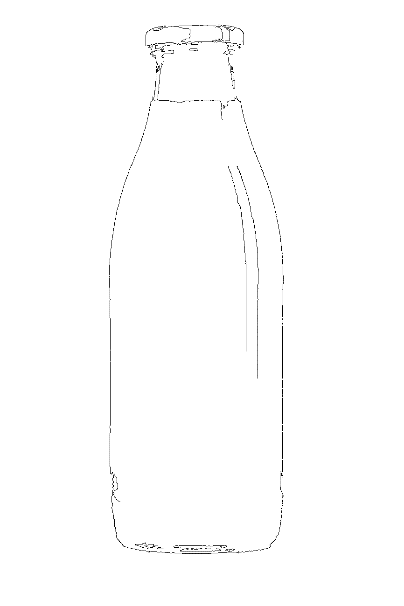


Figure :Binary equivalent

Note that we used THRESH\_BINARY\_INV

1. Noise removal with opening morphological operation
2. kernelOpen=np.ones((20,20))
3. maskopen=cv.morphologyEx(thresh,cv.MORPH\_OPEN,kernelClose)
4. More enhancing using another morphological operation to easy separate bottle object from the background with dilation operation
5. kernel = np.ones((5,5),np.uint8)
6. sure\_bg = cv.dilate(maskopen,kernel,iterations=1)
7. Extract the area were we sure that the contains the bottle object
8. dist\_transform = cv.distanceTransform(maskopen,cv.DIST\_L2,5)
9. ret, sure\_fg = cv.threshold(dist\_transform,0.7\*dist\_transform.max(),255,0)
10. Finding unknown region by subtracting foreground area from background area
11. sure\_fg = np.uint8(sure\_fg)
12. unknown = cv.subtract(sure\_bg,sure\_fg)
13. labeling each region with markers using cv,connectedComponents, but it marks background with zero to prepare result to watershed algorithm we need to add 1 to all markers
14. ret, markers = cv.connectedComponents(sure\_fg)
15. # Add one to all labels so that sure background is not 0, but 1
16. markers = markers+1
17. # Now, mark the region of unknown with zero
18. markers[unknown==255] = 0
19. Segment image with watershed using previous markers all negative region will be considered noise, our region of interest will be marked with value 1, the boundaries get -1 value. Finally, we get the desired mask for bottle object.
20. markers = cv.watershed(img,markers)
21. **for** i **in** range(0, ret+1):
22. # color = np.random.randint(255,size=3)
23. img[markers == i]=[0, 0 ,0]
24. img[markers==-1]=[255,0,0]
25. x[markers == 1 ]=[255,255,255]

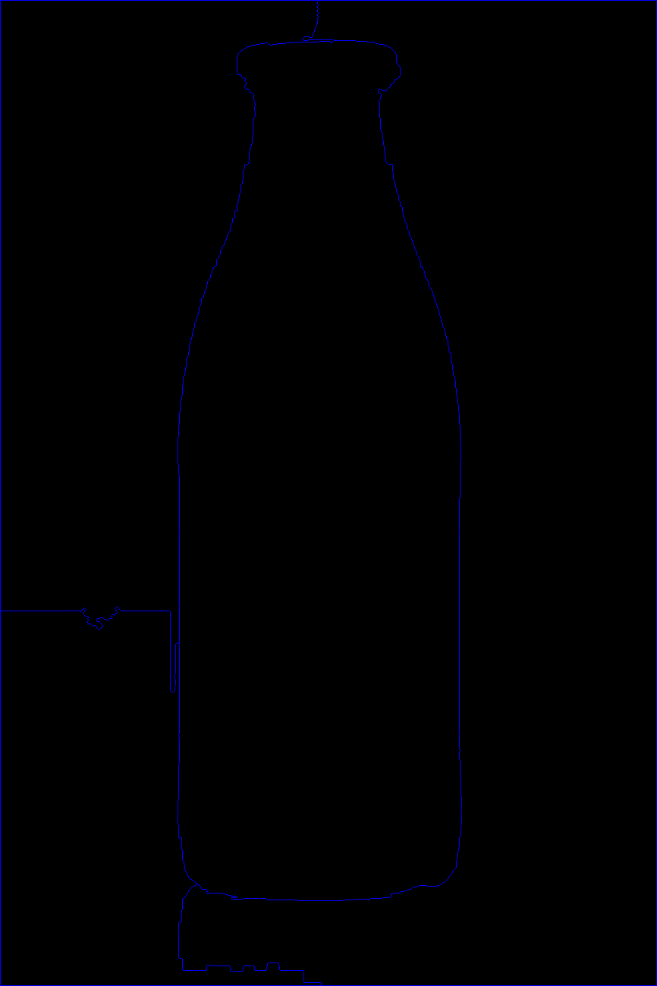


Figure : Segment image

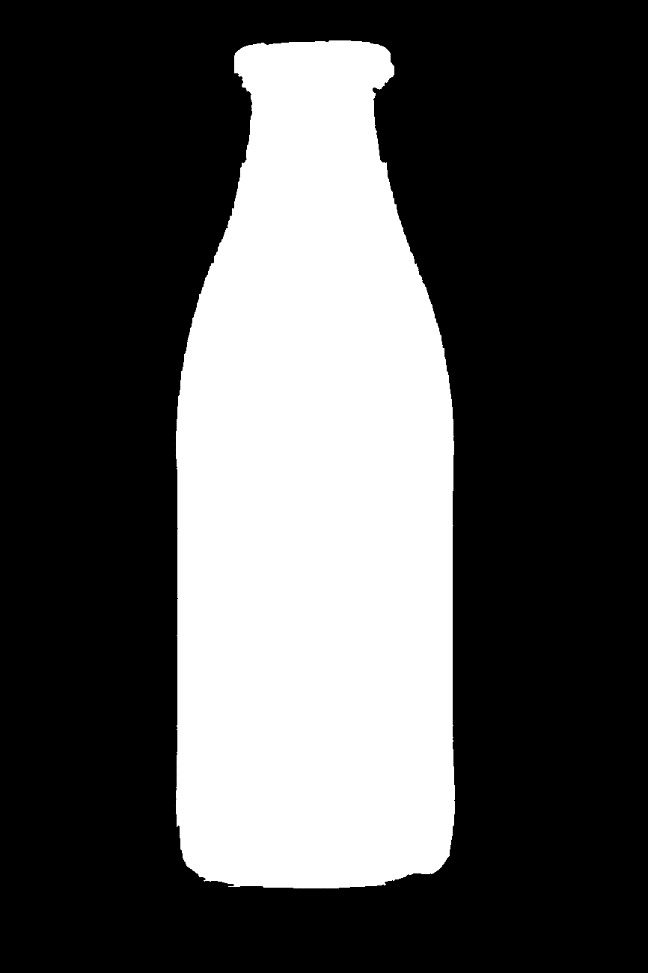


Figure :Our desired mask

1. Using the mask we get bottle object with black background
2. img1 = imgtest.copy()
3. img2=x.copy()
4. img2[np.where(x == 255)] = img1[np.where(x == 255)]
5. forprint = cv.cvtColor(img2, cv.COLOR\_BGR2RGBA)



Figure :Bottle with black background

1. Remove black background with gives the background 0 value transparent, so we get only the bottle object (we choose this noised image to show that algorithm can work in hard situations)
2. image = cv.cvtColor(img2, cv.COLOR\_BGR2RGBA)
3. image[np.all(image == [0, 0, 0, 255], axis=2)] = [0, 0, 0, 0]



Figure : Pure bottle object

1. At previous step we detect successfully the bottle, now we need to recognize the juice level: full, half full, or empty, we accomplished this by calculating the pixels values to determine the pure glass (without juice), then calculating the ration between the pure glass pixels number and the all pixels number, by putting appropriate thresholds we can determine the ranges (also we can add more thresholds to get more ranges like semi full or semi empty, but according to project requirements we have only 3 ranges).
2. glass\_MIN = np.array([180, 180, 180,100], np.uint8)
3. glass\_MAX = np.array([255, 255, 255,255], np.uint8)
4. dst = cv.inRange(image, glass\_MIN, glass\_MAX)
5. no\_glass = cv.countNonZero(dst)
6. **print**('The number of Glass pixels is: ' + str(no\_glass))
7. //////////////////
8. imagee = cv.cvtColor(imgtest, cv.COLOR\_BGR2RGBA)
9. all\_MIN = np.array([0, 0, 0, 0], np.uint8)
10. all\_MAX = np.array([255, 255, 255, 255] , np.uint8)
11. dst = cv.inRange(imagee, all\_MIN, all\_MAX)
12. no\_all = cv.countNonZero(dst)
13. **print**('The number of all pixels is: ' + str(no\_all))
14. /////////////////
15. z=no\_glass/no\_all
16. **print**(z)
17. **if**(z<0.05):
18. **print**('full')
20. **if**(z>0.1):
22. **print**('empty')
24. **if**(z>0.05 **and** z<0.1):
25. **print**('half full')

The result for this image

The number of Glass pixels is: 20341

The number of all pixels is: 1125800

0.018068040504530113 full

1. Others cases



Figure :Half full glass bottle of juice

The result:

The number of Glass pixels is: 14056

The number of all pixels is: 151650

0.09268710847345862 half full



Figure :Empty bottle (with shadows)

The result:

The number of Glass pixels is: 66646

The number of all pixels is: 250000

0.266584 empty



Figure :Half full bottle with noise



Figure :Pure bottle

The result:

The number of Glass pixels is: 3274

The number of all pixels is: 50325

0.06505712866368604 half full