Homework 2
BSysE 530
Binary Image Processing and Morphological Operators

Abhijay Ghildyal

EECS Washington State University
abhijay.ghildyal@wsu.edu

30$^{\text{th}}$ January 2019

1 Introduction

In this homework binary image processing will be performed using morphological operations. Morphological operations analyze/transform geometrical structures in the image using set theory, lattice theory, topology etc. To practically implement and learn the usability of different operations, apple images from previous homework and some images on circles and holes have been used.

Objectives:

1. Find area of each apple
2. Filter using area to remove smaller objects
3. Find perimeter
4. Find compactness of apples
5. Write about different morphological operations
6. Define a structuring element to find the boundary of circles
7. Find the number of holes and their diameter

These exercises are interesting as they can be used in the pre or post processing operations in the pipeline of various vision based tasks. Morphological operations extract image components that are useful in the representation and description of region shape, such as boundaries extraction, skeletons, convex hull, morphological filtering, thinning, pruning.

2 Materials and Methods

For the various implementation tasks in this assignment OpenCV and Python were used.

Functions/Operations used for various tasks:

1. Structuring Element: cv2.getStructuringElement
2. Erosion, Dilation, Opening, Closing: cv2.morphologyEx
3. Smoothening: cv2.blur
4. Outline/contour: cv2.findContours
5. Area: cv2.contourArea
6. Perimeter: cv2.arcLength
7. Compactness: $(Perimeter^2)/(4 * pi * Area)$
8. Gradient for finding boundary: cv2.morphologyEx, cv2.MORPH_GRADIENT
9. Circle boundary: cv2.HoughCircles

3 Results and Discussions

3.1 Ans 1 and 2

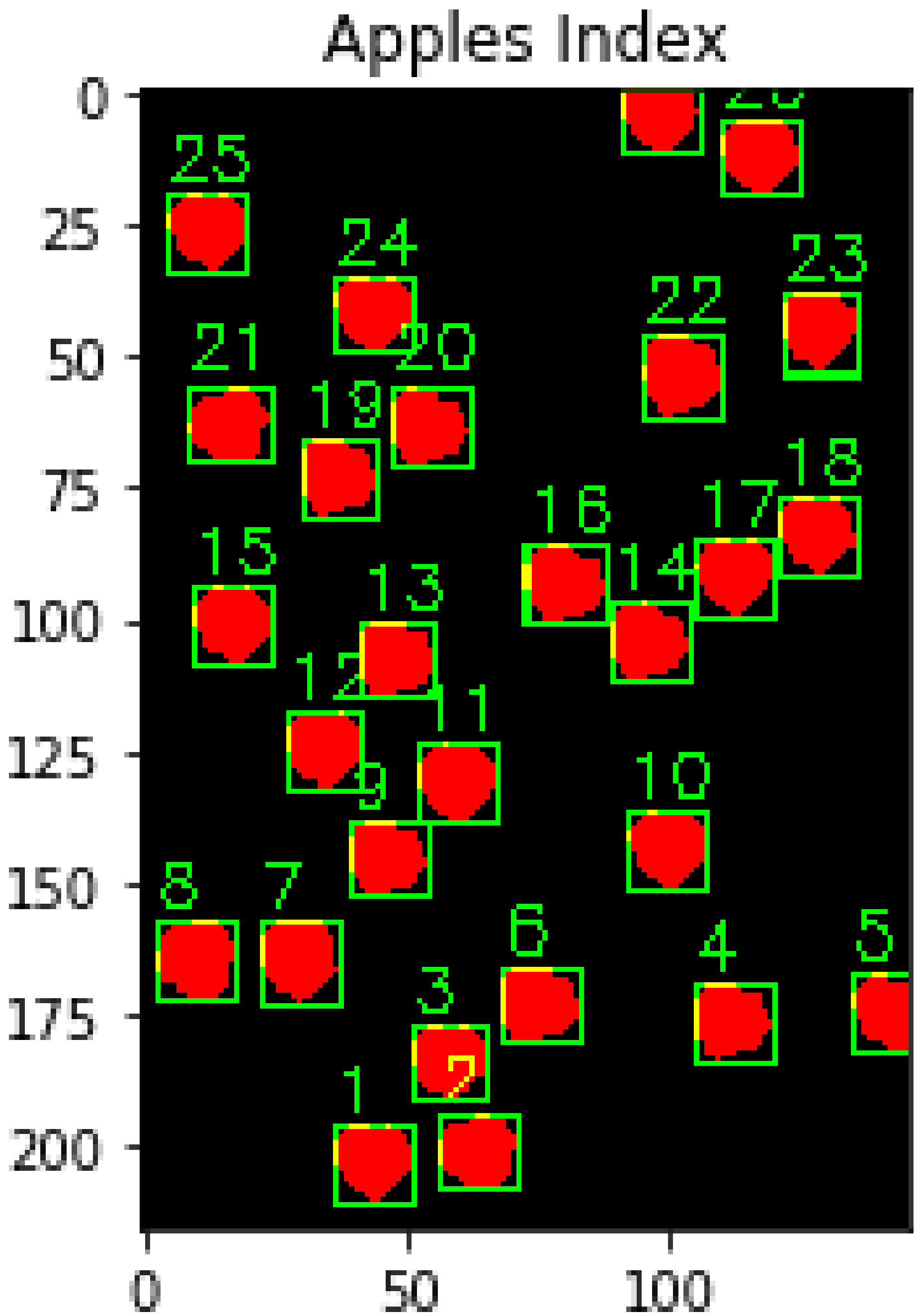


Figure 1: Fake Apples

See Fig. 1

Index	Area	Perimeter	Compactness
1	142.0	46.28	1.2
2	140.0	45.8	1.19
3	133.5	45.21	1.22
4	141.5	48.38	1.32
5	103.0	42.14	1.37
6	138.5	45.56	1.19
7	141.5	46.04	1.19
8	141.5	46.38	1.21
9	141.0	46.97	1.25
10	140.0	45.46	1.17
11	145.0	45.46	1.13
12	135.5	45.21	1.2
13	140.5	46.73	1.24
14	144.5	47.21	1.23
15	143.0	46.28	1.19
16	146.0	47.8	1.25
17	137.0	46.28	1.24
18	139.0	47.46	1.29
19	142.5	46.38	1.2
20	141.0	47.8	1.29
21	141.0	46.63	1.23
22	143.0	46.63	1.21
23	139.0	45.8	1.2
24	140.0	45.46	1.17
25	142.0	46.28	1.2
26	143.0	44.63	1.11
27	120.5	43.56	1.25

Figure 2: Apples

See Fig. 2

Index	Area	Perimeter	Compactness
1	586.0	100.08	1.36
2	1169.5	140.47	1.34
3	639.0	99.25	1.23
4	513.0	88.77	1.22
5	378.5	74.18	1.16
6	219.0	55.46	1.12
7	352.0	71.94	1.17
8	1602.5	163.3	1.32
9	889.0	126.23	1.43
10	304.5	65.7	1.13
11	597.5	96.33	1.24
12	689.5	101.01	1.18
13	1544.5	160.61	1.33
14	322.0	69.11	1.18
15	1199.0	135.05	1.21
16	1621.0	160.37	1.26
17	766.0	111.74	1.3
18	2247.0	225.68	1.8
19	373.5	74.18	1.17
20	970.5	138.81	1.58
21	1058.0	127.05	1.21
22	182.0	50.63	1.12
23	776.0	111.74	1.28
24	806.5	116.81	1.35
25	771.5	135.64	1.9
26	306.0	66.28	1.14
27	1704.5	162.95	1.24
28	888.0	142.57	1.82
29	631.5	99.84	1.26
30	1486.5	151.3	1.23
31	550.0	98.91	1.42
32	1059.5	127.5	1.22
33	181.5	56.38	1.39

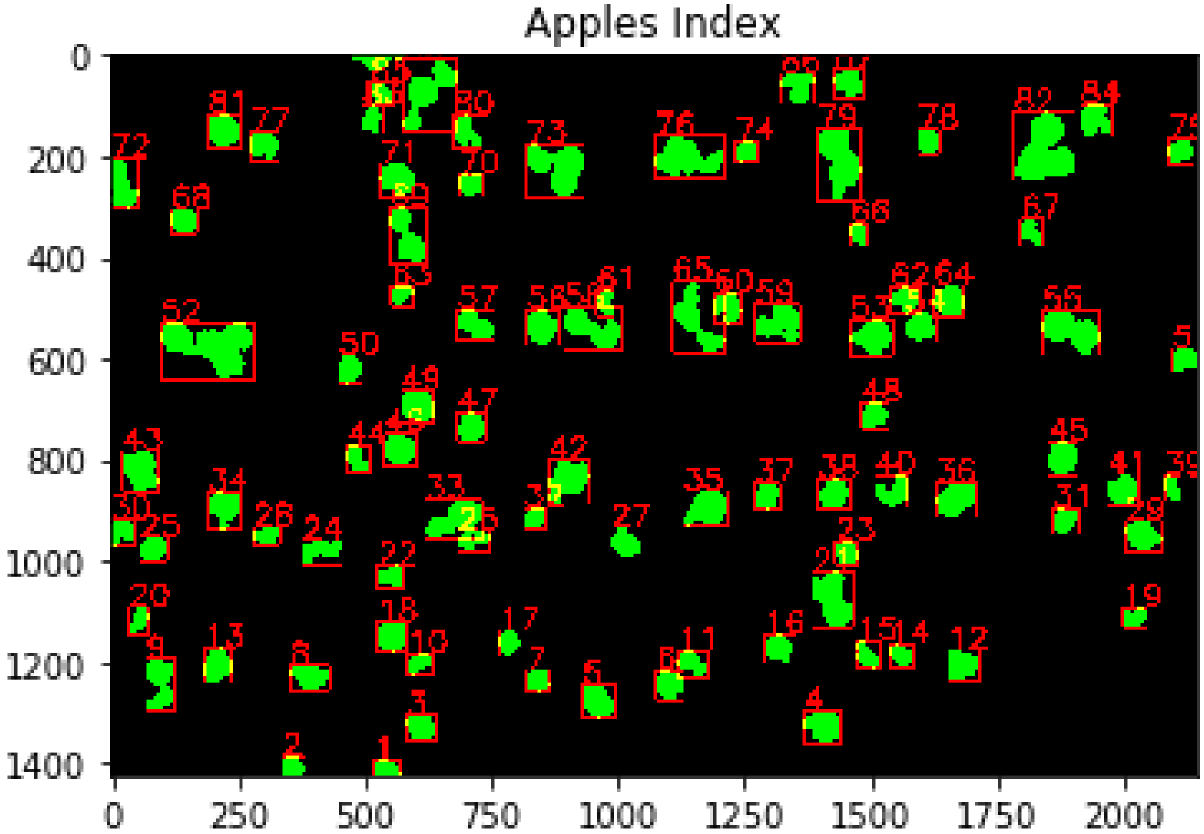


Figure 3: Aurora Wall

See Fig. 3

Index	Area	Perimeter	Compactness
1	1241.0	150.77	1.46
2	1370.5	145.84	1.23
3	2165.0	184.37	1.25
4	3337.0	231.34	1.28
5	3023.0	218.85	1.26
6	2390.0	188.85	1.19
7	1551.5	155.98	1.25
8	2608.5	217.44	1.44
9	3773.5	317.81	2.13
10	1284.5	146.61	1.33
11	1718.5	190.61	1.68
12	2775.0	215.54	1.33
13	2708.0	208.85	1.28
14	1377.5	151.78	1.33
15	1483.5	170.47	1.56
16	2322.5	186.95	1.2
17	1516.0	153.54	1.24
18	2638.5	208.75	1.31
19	1305.0	154.23	1.45
20	1300.0	160.71	1.58
21	5589.0	342.11	1.67
22	1547.0	171.05	1.51
23	1602.5	153.15	1.16
24	2405.5	245.24	1.99
25	1973.5	170.81	1.18
26	1509.5	215.1	2.44
27	2273.0	193.34	1.31
28	1283.0	146.57	1.33
29	2982.5	223.24	1.33
30	1826.5	191.98	1.61
31	1736.5	166.95	1.28
32	1257.0	136.57	1.18
33	4900.5	346.84	1.95
34	3027.0	243.34	1.56
35	4190.5	274.55	1.43
36	3883.0	252.65	1.31
37	1968.0	173.54	1.22
38	2627.5	197.92	1.19

Index	Area	Perimeter	Compactness
39	1136.5	138.33	1.34
40	2592.5	249.24	1.91
41	2732.5	208.61	1.27
42	4564.0	306.94	1.64
43	4063.5	270.55	1.43
44	1548.0	167.54	1.44
45	2887.5	205.1	1.16
46	3002.5	209.24	1.16
47	2294.5	187.78	1.22
48	1955.0	175.54	1.25
49	3184.0	218.85	1.2
50	1904.5	169.3	1.2
51	1747.0	170.43	1.32
52	11407.5	626.94	2.74
53	3763.0	252.45	1.35
54	2535.0	195.68	1.2
55	5997.0	353.22	1.66
56	3113.5	216.75	1.2
57	2795.0	223.82	1.43
58	4859.5	368.78	2.23
59	4275.5	324.21	1.96
60	2461.0	216.85	1.52
61	1207.5	149.78	1.48
62	2028.5	209.58	1.72
63	1212.5	143.3	1.35
64	2935.0	210.51	1.2
65	5210.0	445.99	3.04
66	1107.0	130.91	1.23
67	1489.5	167.64	1.5
68	1975.5	169.64	1.16
69	4411.5	340.49	2.09
70	1629.0	163.88	1.31
71	3463.0	239.82	1.32
72	3542.0	282.85	1.8
73	6834.5	454.72	2.41
74	1321.0	142.91	1.23
75	1798.0	165.2	1.21
76	6494.5	437.75	2.35
77	2390.0	196.51	1.29

Index	Area	Perimeter	Compactness
78	1609.5	153.15	1.16
79	7000.5	420.29	2.01
80	2237.0	224.17	1.79
81	3165.0	218.17	1.2
82	9874.5	520.17	2.18
83	1514.0	165.88	1.45
84	2785.0	234.17	1.57
85	1339.5	146.33	1.27
86	2750.0	240.51	1.67
87	2332.5	195.44	1.3
88	6062.0	483.3	3.07
89	2010.0	257.88	2.63

3.2 Ans 3

Erosion
In this operation the areas of foreground pixels shrink in size and the holes within those areas become larger.
The structuring element is superimposed on top of the input image so that the origin of the structuring element coincides with the input pixel coordinates. Foreground pixel in a structuring element, is the corresponding pixel in the image underneath where the input is left as it is. If any of the corresponding pixels in the image are background, however, the input pixel is also set to background value. In a 3x3 structuring element, the effect of this operation is to remove any foreground pixel that is not completely surrounded by other white pixels.

Dilation
The effect of this operator on a binary image is to enlarge the boundaries of regions of foreground pixels (white pixels). Hence, areas of foreground pixels grow in size while holes within those regions become smaller.
For the background pixel i.e. the input pixel, the structuring element is superimposed on top of the input image. The origin of the structuring element coincides with the input pixel position. If at least one pixel in the structuring element coincides with a foreground pixel in the image underneath, then the input pixel is set to the foreground value.
For a 3x3 structuring element, the effect of this operation is to set to the foreground color any background pixels that have a neighboring foreground pixel. Such pixels must lie at the edges of white regions, and so the foreground regions grow and holes inside a region shrink.

Opening
Opening operation is defined as an erosion followed by a dilation using the same structuring element. It is used for eliminating salt noise and using the property of idempotence it reverses the effect of erosion with dilation, making it a single operation for this purpose.

Closing
Closing is the reverse of Opening operation and is used to close the holes or gaps in background color (eg. pepper noise).

Binary hit-miss operation
This operation is a kind of image pattern matching and marking. Erosion, dilation, opening, closing, thinning and thickening can all be derived from the hit-and-miss transform in conjunction with simple set operations.

Thinning
This operation is useful when used with edge detectors as it reduces all the lines to the thickness of a single pixel. Thinning of an image I by a structuring element J = I - hit_miss(I,J).

Thickening
This operation is useful when it is required to grow certain regions. Thickening of an image I by a structuring element J = I $\cup$ hit_miss(I,J).

Skeletonization/Medial Axis Transform
Skeleton can be imagined as the ridges on the 3-D surface. The skeleton is obtained in two ways. The first is to use morphological thinning, eroding away pixels from boundary and preserving end points of line segments. This is done until more thinning is not possible. The other method is to first calculate the distance transform of the image. The skeleton would then lie along the singularities in the distance transform i.e. creases or curvature discontinuities. The second approach is more widely used.

Gradient
This operation in Opencv gets the difference between the dilation and erosion of image to get the outline of the object.

BlackHat
It is the difference between the closing of the input image and input image.

TopHat
It is the difference between input image and Opening of the image.

3.3 Ans 4

By using a kernel of appropriate size, gradient method in opencv can be applied to find the edges or boundaries of a certainly shaped object.

```
kernel = np.ones((4,4),np.uint8)
gradient = cv2.morphologyEx(img, cv2.MORPH_GRADIENT, kernel)
```

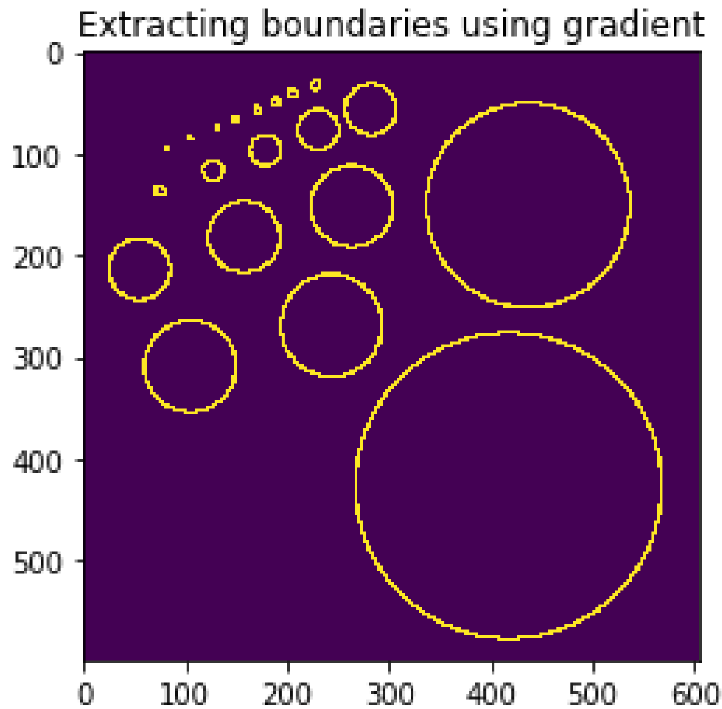


Figure 4: Extracting boundaries using morphological operation called gradient

These boundaries correspond to the actual boundaries of the original circles.

3.4 Ans 5

Using the hough transform technique a certain shape can be detected by using a set of accumulator arrays. An accumulator array tries to gather maximum points onto itself using an optimization technique. A voted approach is taken on the accumulator at the end to select the final points. Gradient based technique was later used to augment the previous technique which is faster and reduces the amount of voting (compute intensive).

```
circles = cv2.HoughCircles( gray, cv2.HOUGH_GRADIENT, 1, gray.shape
    [0]/4, param1=1, param2=20, minRadius=0, maxRadius=40)
```

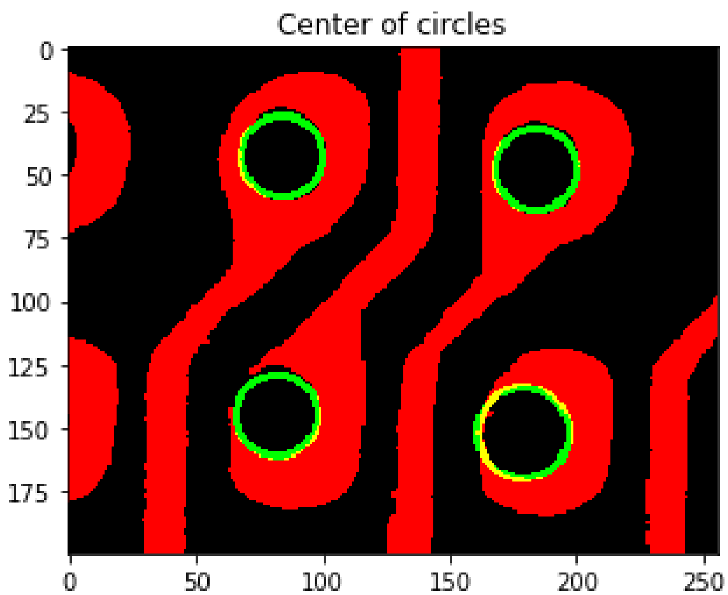


Figure 5: Finding holes (circles) using Hough Gradient

x_{center}, y_{center}, radius (diameter is 2*radius) are as follows:

```
array([[[ 84.5,  43.5,  16.8],
        [184.5,  48.5,  16.8],
        [179.5, 152.5,  18.1],
        [ 82.5, 145.5,  16. ]]], dtype=float32)
```

4 Conclusion

Analyzing the size, shape, area and compactness of apples will be highly useful in crop load estimation which may assist farmers to understand the statistics regarding their yield and make plans accordingly for an efficient harvesting strategy.

The operations performed in this assignment are useful in image processing tasks and are computationally efficient if being used real time. Even though being computationally less intensive, not a lot can be said about their ubiquitous use because the thresholds used in analysis would differ with changing background conditions and hence should only be used for augmenting the results of some of the latest algorithms for a generalized application.

Part 2: Binary Image Processing and Morphological Operators

In [2]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import NoNorm
from skimage import measure
```

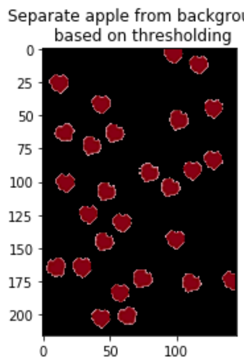
In [3]:
```
def process_apples(img):

    # get hsv values for thresholding; with rgb it was not coming to be be less accurate
    img_hsv=cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

    # Threshold based on both light and dark red color ranges
    # lower mask (0-10)
    lower_red = np.array([0,50,50])
    upper_red = np.array([10,255,255])
    mask0 = cv2.inRange(img_hsv, lower_red, upper_red)

    # upper mask (170-180)
    lower_red = np.array([170,50,50])
    upper_red = np.array([180,255,255])
    mask1 = cv2.inRange(img_hsv, lower_red, upper_red)

    # join masks
    mask = mask0+mask1

    # set my output img to zero everywhere except my mask
    output_img = img.copy()
    output_img[np.where(mask==0)] = 0

    output_img = cv2.cvtColor( output_img, cv2.COLOR_BGR2RGB)
    imgplot = plt.imshow( output_img)
    plt.title("Separate apple from background\n based on thresholding")
    plt.show()

    return output_img

font = cv2.FONT_HERSHEY_SIMPLEX
def calcArea_filterApples_calcPerimeterCompactness(img, area_threshold, font_size, font_weight, rec_Width=1, appleCo
lor='R'):

    mask = np.ones(img.shape, dtype="uint8") * 255
    img_ = np.zeros(( img.shape[0], img.shape[1], 3))

    # Finding contours of the image
    cnts = cv2.findContours(img, 1, cv2.CHAIN_APPROX_SIMPLE)

    contourArea = []
    contourPerimeter = []
    appleCount = 0

    for i, contour in enumerate(cnts[1]):
        area = cv2.contourArea(contour)
        if area>area_threshold:
            cv2.drawContours(mask, [contour], -1, 0, -1)
            appleCount+=1

            contourArea.append(area)
            contourPerimeter.append(round(cv2.arcLength(contour,True),2))

            x,y,w,h = cv2.boundingRect(contour)

            if appleColor == 'R':
                cv2.rectangle(img_,(x,y),(x+w,y+h),(0,255,0),rec_Width)
                cv2.putText(img_, str(appleCount),(x,y-3), font, font_size,(0,255,0), font_weight, cv2.LINE_AA)
            else:
                cv2.rectangle(img_,(x,y),(x+w,y+h),(255,0,0),rec_Width)
                cv2.putText(img_, str(appleCount),(x,y-3), font, font_size,(255,0,0), font_weight, cv2.LINE_AA)

    if appleColor == 'R':
        img_[:,:,0] = cv2.bitwise_not(mask.copy())
    else:
        img_[:,:,1] = cv2.bitwise_not(mask.copy())
    imgplot = plt.imshow( img_)
    plt.title("Apples Index")
    plt.show()

    return contourArea, contourPerimeter, img_[:,:,0]
```

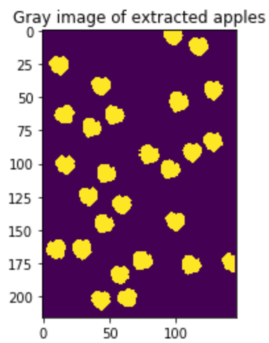
1

FakeApples.bmp

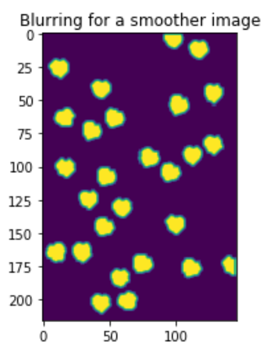
a. FakeApples.bmp

In [21]:
```
img=cv2.imread("FakeApples.bmp")
img = process_apples(img)
gray = cv2.cvtColor( img, cv2.COLOR_BGR2GRAY)
gray[gray>0] = 255
imgplot = plt.imshow( gray)
plt.title("Gray image of extracted apples")
plt.show()
```

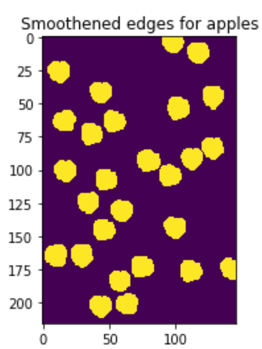




In [22]:
```
blur = cv2.blur(gray,(3,3))
imgplot = plt.imshow( blur)
plt.title("Blurring for a smoother image")
plt.show()
```



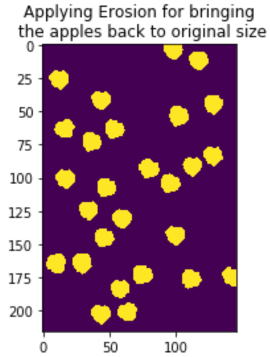
In [23]:
```
gray= blur.copy()
gray[gray>0] = 255
imgplot = plt.imshow( gray)
plt.title("Smoothened edges for apples")
plt.show()
```



In [24]:
```python
# Taking a square kernel
kernel = np.ones((3,3),np.uint8)
print (kernel)
```

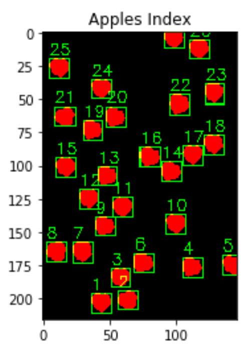
```
[[1 1 1]
 [1 1 1]
 [1 1 1]]
```

In [25]:
```python
erode = cv2.morphologyEx(gray, cv2.MORPH_ERODE, kernel)
imgplot = plt.imshow( erode)
plt.title("Applying Erosion for bringing\n the apples back to original size")
plt.show()
```



In [26]:
```python
area_threshold = 100
font_weight = 1
font_size = 0.4
contourArea, contourPerimeter, img_ = calcArea_filterApples_calcPerimeterCompactness( erode, area_threshold, font_si
ze, font_weight)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```



In [27]:
```python
print ("Index, Area, Perimeter, Compactness")
for i, area in enumerate(contourArea):
    print (str(i+1)+", "+str(contourArea[i])+", "+str(contourPerimeter[i])+", "+str(round( (contourPerimeter[i]**2)/
(4*np.pi*contourArea[i]),2)))
```

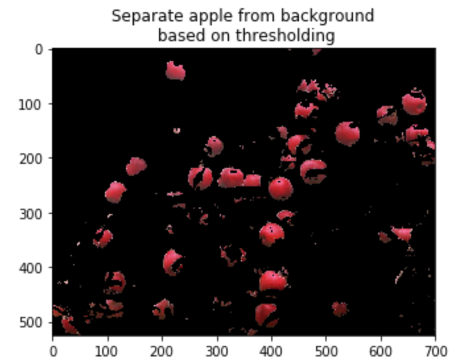
```
Index, Area, Perimeter, Compactness
1, 142.0, 46.28, 1.2
2, 140.0, 45.8, 1.19
3, 133.5, 45.21, 1.22
4, 141.5, 48.38, 1.32
5, 103.0, 42.14, 1.37
6, 138.5, 45.56, 1.19
7, 141.5, 46.04, 1.19
8, 141.5, 46.38, 1.21
9, 141.0, 46.97, 1.25
10, 140.0, 45.46, 1.17
11, 145.0, 45.46, 1.13
12, 135.5, 45.21, 1.2
13, 140.5, 46.73, 1.24
14, 144.5, 47.21, 1.23
15, 143.0, 46.28, 1.19
16, 146.0, 47.8, 1.25
17, 137.0, 46.28, 1.24
18, 139.0, 47.46, 1.29
19, 142.5, 46.38, 1.2
20, 141.0, 47.8, 1.29
21, 141.0, 46.63, 1.23
22, 143.0, 46.63, 1.21
23, 139.0, 45.8, 1.2
24, 140.0, 45.46, 1.17
25, 142.0, 46.28, 1.2
26, 143.0, 44.63, 1.11
27, 120.5, 43.56, 1.25
```

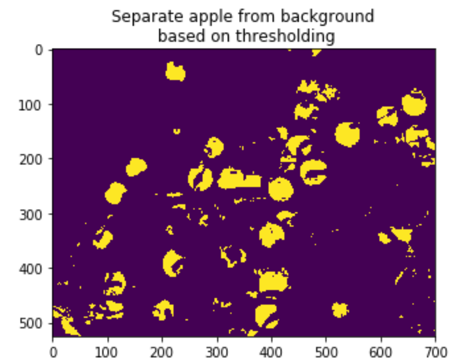
In [28]:
```python
print ("No. of apples: "+str(len(contourArea)))
```

```
No. of apples: 27
```

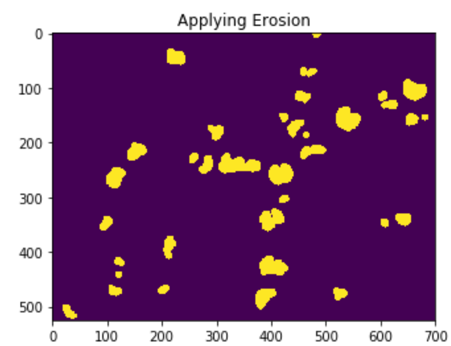
b. Apples.JPG

In [29]:
```
img=cv2.imread("Apples.JPG")
img = process_apples(img)
gray = cv2.cvtColor( img, cv2.COLOR_BGR2GRAY)
gray[gray>0] = 255
imgplot = plt.imshow( gray)
plt.title("Separate apple from background\n based on thresholding")
plt.show()
```

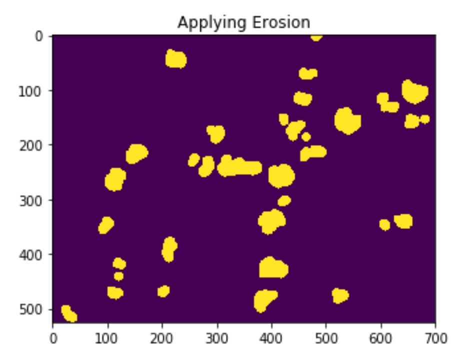




In [30]:
```
kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(12,12))
opening = cv2.morphologyEx( gray, cv2.MORPH_OPEN, kernel)
imgplot = plt.imshow( opening)
plt.title("Applying Opening")
plt.show()
```

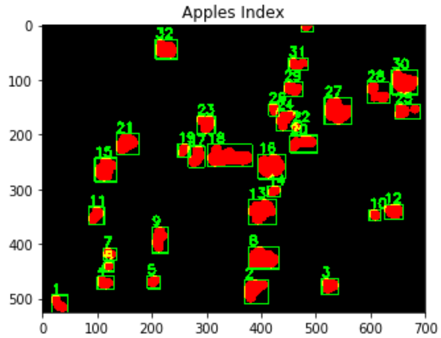


In [31]:
```
kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(5,5))
dilating = cv2.morphologyEx( opening, cv2.MORPH_DILATE, kernel)
imgplot = plt.imshow( dilating)
plt.title("Applying Dilation")
plt.show()
```



In [32]:
```
area_threshold = 120
font_weight = 3
font_size = 0.8
rec_Width = 2
contourArea, contourPerimeter, img_ = calcArea_filterApples_calcPerimeterCompactness( dilating, area_threshold, font
_size, font_weight, rec_Width)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```



In [33]:
```
# Compactness can be defined for example as the perimeter squared, divided by 4piarea, so that a circle has a compac
tness of 1.
# https://en.wikipedia.org/wiki/Compactness_measure_of_a_shape
# http://ceur-ws.org/Vol-1814/paper-04.pdf
# http://answers.opencv.org/question/51602/has-opencv-built-in-functions-to-calculate-circularity-compactness-etc-fo
r-contoursblobs/
# https://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html?highlight=moments#s
tructural-analysis-and-shape-descriptors
```

In [34]:
```
for i, area in enumerate(contourArea):
    print (str(i+1)+", "+str(contourArea[i])+", "+str(contourPerimeter[i])+", "+str(round( (contourPerimeter[i]**2)/
(4*np.pi*contourArea[i]),2)))
```

```
1, 586.0, 100.08, 1.36
2, 1169.5, 140.47, 1.34
3, 639.0, 99.25, 1.23
4, 513.0, 88.77, 1.22
5, 378.5, 74.18, 1.16
6, 219.0, 55.46, 1.12
7, 352.0, 71.94, 1.17
8, 1602.5, 163.3, 1.32
9, 889.0, 126.23, 1.43
10, 304.5, 65.7, 1.13
11, 597.5, 96.33, 1.24
12, 689.5, 101.01, 1.18
13, 1544.5, 160.61, 1.33
14, 322.0, 69.11, 1.18
15, 1199.0, 135.05, 1.21
16, 1621.0, 160.37, 1.26
17, 766.0, 111.74, 1.3
18, 2247.0, 225.68, 1.8
19, 373.5, 74.18, 1.17
20, 970.5, 138.81, 1.58
21, 1058.0, 127.05, 1.21
22, 182.0, 50.63, 1.12
23, 776.0, 111.74, 1.28
24, 806.5, 116.81, 1.35
25, 771.5, 135.64, 1.9
26, 306.0, 66.28, 1.14
27, 1704.5, 162.95, 1.24
28, 888.0, 142.57, 1.82
29, 631.5, 99.84, 1.26
30, 1486.5, 151.3, 1.23
31, 550.0, 98.91, 1.42
32, 1059.5, 127.5, 1.22
33, 181.5, 56.38, 1.39
```

In [35]:
```
print ("No. of apples: "+str(len(contourArea)))
```

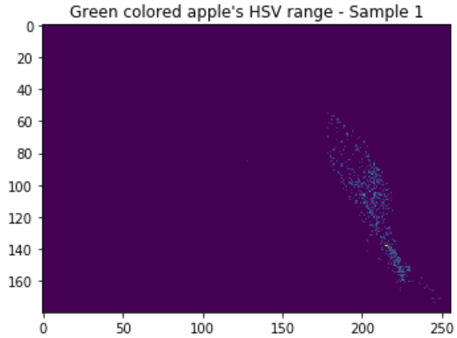
```
No. of apples: 33
```

c. AuroraWall.JPG

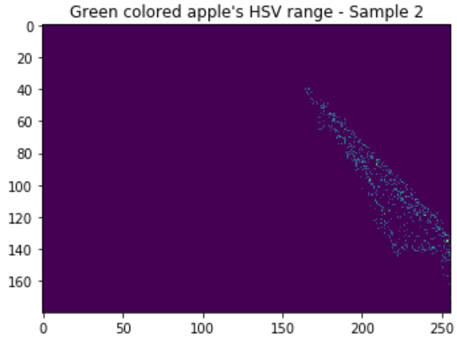
In [84]:

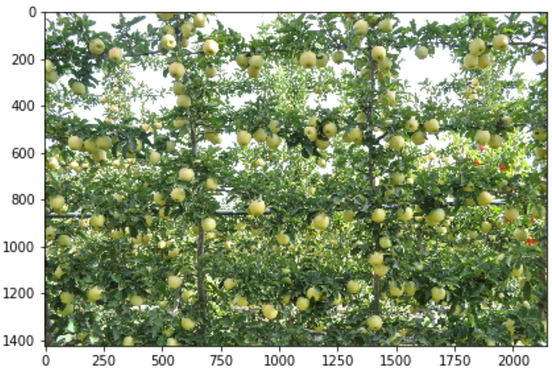
```
# Histogram of green colored apple's HSV
img=cv2.imread("1.png")
img_hsv=cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
hist = cv2.calcHist( [img], [0, 1], None, [180, 256], [0, 180, 0, 256])
plt.imshow(hist,interpolation = 'nearest')
plt.title('Green colored apple\'s HSV range - Sample 1')
plt.show()

img=cv2.imread("2.png")
img_hsv=cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
hist = cv2.calcHist( [img], [0, 1], None, [180, 256], [0, 180, 0, 256])
plt.imshow(hist,interpolation = 'nearest')
plt.title('Green colored apple\'s HSV range - Sample 2')
plt.show()

img=cv2.imread("AuroraWall.JPG")

# get hsv values for thresholding; with rgb it was not coming to be be less accurate
img_hsv=cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

# Threshold based on both light green color ranges

lower_green = np.array([10,20,60])
upper_green = np.array([41,255,255])
mask = cv2.inRange(img_hsv, lower_green, upper_green)

# set my output img to zero everywhere except my mask
output_img = img.copy()
output_img[np.where(mask==0)] = 0

gray = cv2.cvtColor( output_img, cv2.COLOR_BGR2GRAY)
gray[gray>0] = 255
img_=cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
f = plt.figure(figsize=(15,12))
ax= f.subplots(1,2)

ax[0].imshow(img_)
ax[1].imshow(gray)
plt.title("Extracted apples based on thresholding")
plt.show()

# Select a kernel for applying morphological operation
kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(11,12))

# Applying Erosion to find round (based on kernel) objects or apples
erosion = cv2.erode( output_img, kernel, iterations = 1)
gray = cv2.cvtColor( erosion, cv2.COLOR_BGR2GRAY)
gray[gray>0] = 255
imgplot = plt.imshow( gray)
plt.title("Applying Erosion")
plt.show()

# Applying Opening to remove noise
kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(11,12))
opening = cv2.morphologyEx(erosion, cv2.MORPH_OPEN, kernel)
gray = cv2.cvtColor( opening, cv2.COLOR_BGR2GRAY)
gray[gray>0] = 255
imgplot = plt.imshow( gray)
plt.title("Applying Opening")
plt.show()

# Applying Dilation to increase to original apple size
kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(8,9))
dilation = cv2.dilate(opening,kernel,iterations = 1)
gray = cv2.cvtColor( dilation, cv2.COLOR_BGR2GRAY)
gray[gray>0] = 255
imgplot = plt.imshow( gray)
plt.title("Applying Dilation")
plt.show()
```

Green colored apple's HSV range - Sample 1

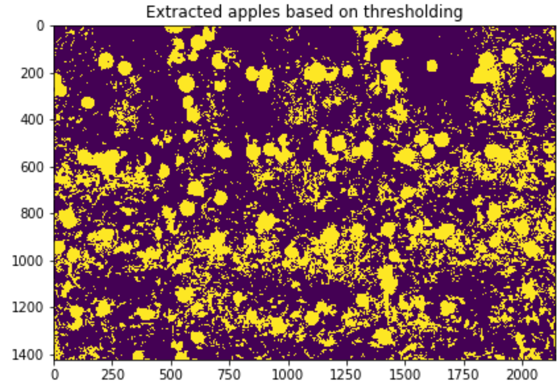


Green colored apple's HSV range - Sample 2

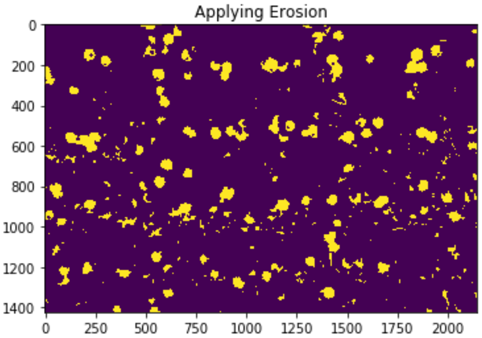




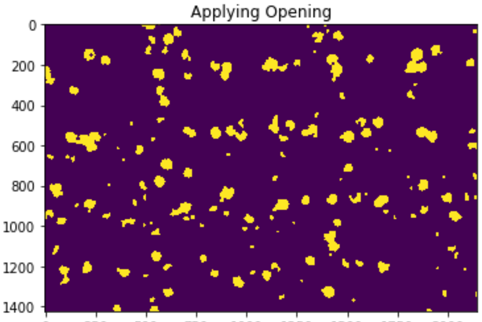
Extracted apples based on thresholding

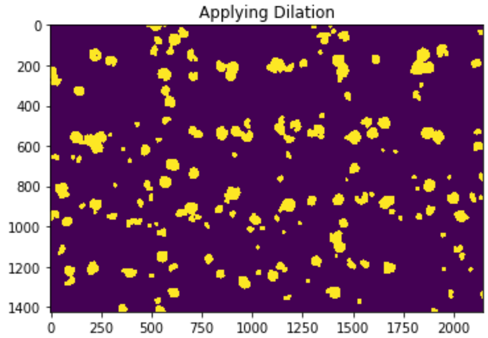


Applying Erosion



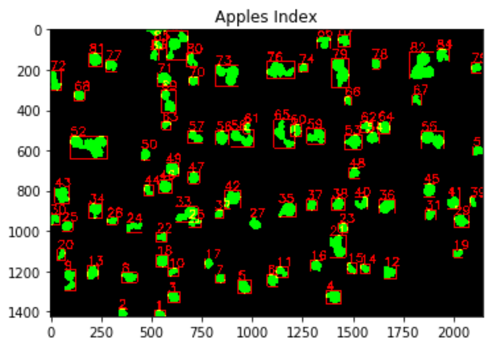
Applying Opening





In [87]:
```
area_threshold = 1000
font_weight = 5
font_size = 2
contourArea, contourPerimeter, img_ = calcArea_filterApples_calcPerimeterCompactness( gray, area_threshold, font_siz
e, font_weight, 3, 'G')
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```



In [39]:
```
for i, area in enumerate(contourArea):
    print (str(i+1)+", "+str(contourArea[i])+", "+str(contourPerimeter[i])+", "+str(round( (contourPerimeter[i]**2)/
(4*np.pi*contourArea[i]),2)))
```

```
1, 1241.0, 150.77, 1.46
2, 1370.5, 145.84, 1.23
3, 2165.0, 184.37, 1.25
4, 3337.0, 231.34, 1.28
5, 3023.0, 218.85, 1.26
6, 2390.0, 188.85, 1.19
7, 1551.5, 155.98, 1.25
8, 2608.5, 217.44, 1.44
9, 3773.5, 317.81, 2.13
10, 1284.5, 146.61, 1.33
11, 1718.5, 190.61, 1.68
12, 2775.0, 215.54, 1.33
13, 2708.0, 208.85, 1.28
14, 1377.5, 151.78, 1.33
15, 1483.5, 170.47, 1.56
16, 2322.5, 186.95, 1.2
17, 1516.0, 153.54, 1.24
18, 2638.5, 208.75, 1.31
19, 1305.0, 154.23, 1.45
20, 1300.0, 160.71, 1.58
21, 5589.0, 342.11, 1.67
22, 1547.0, 171.05, 1.51
23, 1602.5, 153.15, 1.16
24, 2405.5, 245.24, 1.99
25, 1973.5, 170.81, 1.18
26, 1509.5, 215.1, 2.44
27, 2273.0, 193.34, 1.31
28, 1283.0, 146.57, 1.33
29, 2982.5, 223.24, 1.33
30, 1826.5, 191.98, 1.61
31, 1736.5, 166.95, 1.28
32, 1257.0, 136.57, 1.18
33, 4900.5, 346.84, 1.95
34, 3027.0, 243.34, 1.56
35, 4190.5, 274.55, 1.43
36, 3883.0, 252.65, 1.31
37, 1968.0, 173.54, 1.22
38, 2627.5, 197.92, 1.19
39, 1136.5, 138.33, 1.34
40, 2592.5, 249.24, 1.91
41, 2732.5, 208.61, 1.27
42, 4564.0, 306.94, 1.64
43, 4063.5, 270.55, 1.43
44, 1548.0, 167.54, 1.44
45, 2887.5, 205.1, 1.16
46, 3002.5, 209.24, 1.16
47, 2294.5, 187.78, 1.22
48, 1955.0, 175.54, 1.25
49, 3184.0, 218.85, 1.2
50, 1904.5, 169.3, 1.2
51, 1747.0, 170.43, 1.32
52, 11407.5, 626.94, 2.74
53, 3763.0, 252.45, 1.35
54, 2535.0, 195.68, 1.2
55, 5997.0, 353.22, 1.66
56, 3113.5, 216.75, 1.2
57, 2795.0, 223.82, 1.43
58, 4859.5, 368.78, 2.23
59, 4275.5, 324.21, 1.96
60, 2461.0, 216.85, 1.52
61, 1207.5, 149.78, 1.48
62, 2028.5, 209.58, 1.72
63, 1212.5, 143.3, 1.35
64, 2935.0, 210.51, 1.2
65, 5210.0, 445.99, 3.04
66, 1107.0, 130.91, 1.23
67, 1489.5, 167.64, 1.5
68, 1975.5, 169.64, 1.16
69, 4411.5, 340.49, 2.09
70, 1629.0, 163.88, 1.31
71, 3463.0, 239.82, 1.32
72, 3542.0, 282.85, 1.8
73, 6834.5, 454.72, 2.41
74, 1321.0, 142.91, 1.23
75, 1798.0, 165.2, 1.21
76, 6494.5, 437.75, 2.35
77, 2390.0, 196.51, 1.29
78, 1609.5, 153.15, 1.16
79, 7000.5, 420.29, 2.01
80, 2237.0, 224.17, 1.79
81, 3165.0, 218.17, 1.2
82, 9874.5, 520.17, 2.18
83, 1514.0, 165.88, 1.45
84, 2785.0, 234.17, 1.57
85, 1339.5, 146.33, 1.27
86, 2750.0, 240.51, 1.67
87, 2332.5, 195.44, 1.3
88, 6062.0, 483.3, 3.07
89, 2010.0, 257.88, 2.63
```

Number of apples detected

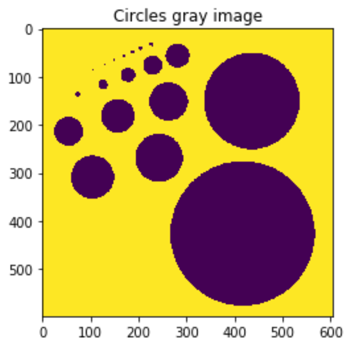
In [377]:
```
print ("No. of apples: "+str(len(contourArea)))
```

```
No. of apples: 89
```

4

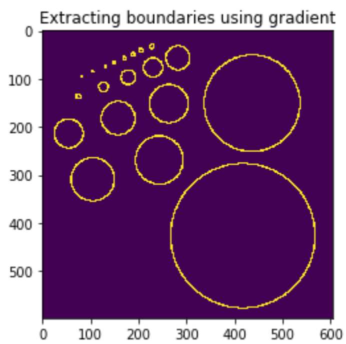
In [379]:
```
img=cv2.imread("Circles.png")
```

In [380]:
```
gray = cv2.cvtColor( img, cv2.COLOR_BGR2GRAY)
gray[gray>0] = 255
imgplot = plt.imshow( gray)
plt.title("Circles gray image")
plt.show()
```



In [381]:
```
kernel = np.ones((4,4),np.uint8)
gradient = cv2.morphologyEx(img, cv2.MORPH_GRADIENT, kernel)
```

In [382]:
```
gray = cv2.cvtColor( gradient, cv2.COLOR_BGR2GRAY)
gray[gray>0] = 255
imgplot = plt.imshow( gray)
plt.title("Extracting boundaries using gradient")
plt.show()
```

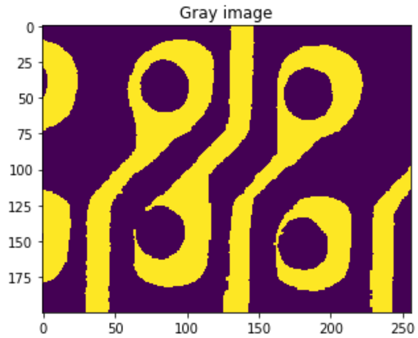


5

In [4]:
```
img=cv2.imread("pcb.jpg")
```

In [5]:
```
gray = img[:,:,0].copy()
gray[gray<200] = 0
gray[gray>200] = 255
```

In [6]:
```
imgplot = plt.imshow( gray)
plt.title("Gray image")
plt.show()
```



In [29]:
```
kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(5,5))
```

In [30]:
```
kernel
```

Out[30]:
```
array([[0, 0, 1, 0, 0],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [0, 0, 1, 0, 0]], dtype=uint8)
```

In [59]:
```
circles = cv2.HoughCircles( gray, cv2.HOUGH_GRADIENT, 1, gray.shape[0]/4, param1=1, param2=20, minRadius=0, maxRadiu
s=40)
```

In [60]:
```
circles
```

Out[60]:
```
array([[[ 84.5,  43.5,  16.8],
        [184.5,  48.5,  16.8],
        [179.5, 152.5,  18.1],
        [ 82.5, 145.5,  16. ]]], dtype=float32)
```

In [79]:
```
cimg_ = np.zeros(gray.shape)
cimg = np.zeros(list(gray.shape)+[3])
cimg[:,:,0] = gray.copy()
```

In [80]:
```
for i in circles[0,:]:
    cv2.circle(cimg_,(i[0],i[1]),i[2],255,2)
```

In [81]:
```
cimg[:,:,1] = cimg_
```

In [82]:
```
imgplot = plt.imshow( cimg)
plt.title("Center of circles")
plt.show()
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```