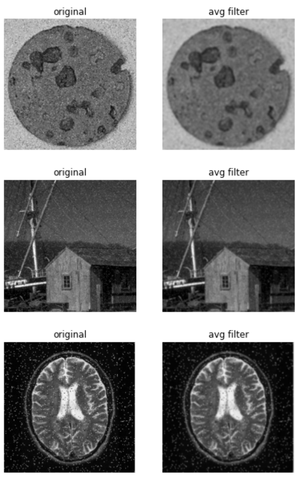
Part 1:

In [1]:
```
import cv2
import numpy as np
from image import my_image,Show,segmentation
from skimage import color
import matplotlib.pyplot as plt
```

Average filtering

The main idea of the average filter is to run through the signal entry by entry, replacing each entry with the average of neighboring entries. it preserves edges while removing noise,Such noise reduction is a typical pre-processing step to improve the results of later processing (for example, edge detection)

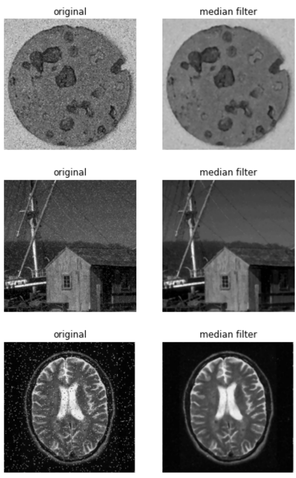
In [2]:
```
order=6
kernel = np.ones((order, order), np.float32)/order**2

img1=my_image.readimage(r'images/test1.png')
avg1 = cv2.filter2D(img1, -1, kernel)
Show.compareim(img1,avg1,'original','avg filter',size=0.7)

img3=my_image.readimage(r'images/test3.png')
avg3 = cv2.filter2D(img3, -1, kernel)
Show.compareim(img3,avg3,'original','avg filter',size=0.7)

img4=my_image.readimage(r'images/test4.png')
avg4 = cv2.filter2D(img4, -1, kernel)
Show.compareim(img4,avg4,'original','avg filter',size=0.7)
```



Median filtering

The main idea of the median filter is to run through the signal entry by entry, replacing each entry with the average of neighboring entries. it preserves edges while removing noise,Such noise reduction is a typical pre-processing step to improve the results of later processing (for example, edge detection)

In [3]:
```
img1=my_image.readimage(r'images/test1.png')
med1 = cv2.medianBlur(img1, ksize=5)
Show.compareim(img1,med1,'original','median filter',size=0.7)

img3=my_image.readimage(r'images/test3.png')
med3 = cv2.medianBlur(img3, ksize=7)
Show.compareim(img3,med3,'original','median filter',size=0.7)

img4=my_image.readimage(r'images/test4.png')
med4 = cv2.medianBlur(img4, ksize=7)
Show.compareim(img4,med4,'original','median filter',size=0.7)
```

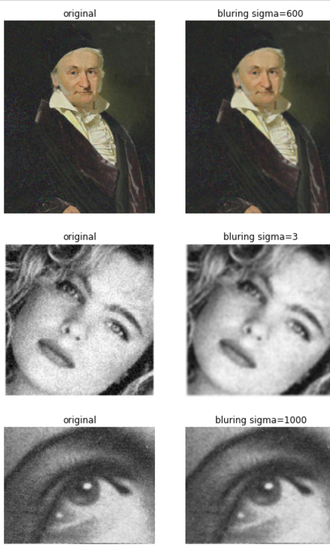


The result is much better compare with avg filter!

Gaussian filtering

it is best to take advantage of the Gaussian blur's separable property by dividing the process into two passes. In the first pass, a one-dimensional kernel is used to blur the image in only the horizontal or vertical direction. In the second pass, the same one-dimensional kernel is used to blur in the remaining direction. The resulting effect is the same as convolving with a two-dimensional kernel in a single pass, but requires fewer calculations.

In [5]:
```
img2=my_image.readimage('images/gauss.png')
img2=my_image.make_gauss_noise(img2,var=300,mean=0)
gsn2=my_image.gauss_filter(img2.copy(),ksize=7,sigma=600)
Show.compareim(img2,gsn2,'original','bluring sigma=600',size=0.8)

img3=my_image.readimage('images/gausstest2.png')
gsn3=my_image.gauss_filter(img3.copy(),ksize=7,sigma=3)
Show.compareim(img3,gsn3,'original','bluring sigma=3',size=0.8)

img4=my_image.readimage('images/gausstest.png')
gsn4=my_image.gauss_filter(img4.copy(),ksize=7,sigma=1000)
Show.compareim(img4,gsn4,'original','bluring sigma=1000',size=0.8)
```



هر چه سیگما بزرگتر شود میزان بلورشدگی و کاهش نویزی که به صورت گوسی بوده کم میشود.

X. Checkpoint

Part 2:

In [1]:
```
from Image import my_image,Show,segmentation
import numpy as np
import cv2
from skimage import color
import matplotlib.pyplot as plt
```

1.Edge detection:

a) Sobel :

-Detects edges where the gradient magnitude is high.This makes the Sobel edge detector more sensitive to diagonal edge than horizontal and vertical edges.

-The operator uses two kernels which are convolved with the original image to calculate approximations of the derivatives (one for horizontal changes, and one for vertical). If we define A as the source image, and Gx and Gy are two images which at each point contain the vertical and horizontal derivative approximations.

- At each point in the image, the resulting gradient approximations can be combined to give the gradient magnitude.

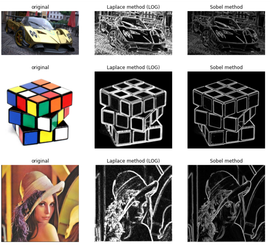
-above process is in image.py

b) Laplace :

-Unlike the Sobel edge detector, the Laplacian edge detector uses only one kernel. It calculates second order derivatives in a single pass so it is computationally faster to calculate (only one kernel vs two kernels)

-because we're working with second order derivatives, the laplacian edge detector is extremely sensitive to noise.so we reduce noise first by median filter and second with Gauss filter

In [2]:
```
img1=my_image.readimage('images/PGN2.jpg')
_,edge1=my_image.edge_detection(img1,mod='sobel',blur_sigma=0.2)
_,edge1=my_image.edge_detection(img1,mod='laplace',blur_sigma=0.2,floor=
25)
Show.comparein(img1,Edge1,'original','Laplace method (LOG)',1.2,1,edge1,
'Sobel method')

img2=my_image.readimage('images/edgedetection/test1.png')
_,edge2=my_image.edge_detection(img2,mod='sobel',blur_sigma=0.2)
_,Edge2=my_image.edge_detection(img2,mod='laplace',blur_sigma=5,floor=25
)
Show.comparein(img2,Edge2,'original','Laplace method (LOG)',1.2,1,edge2,
'Sobel method')

img5=my_image.readimage('images/edgedetection/test3.png')
_,edge5=my_image.edge_detection(img5,mod='sobel',blur_sigma=0.2)
_,Edge5=my_image.edge_detection(img5,mod='laplace',blur_sigma=1,floor=25
)
Show.comparein(img5,Edge5,'original','Laplace method (LOG)',1.2,1,edge5,
'Sobel method')
```



after edge detection with sobel; i use thresholding. So it takes a little more time to compute.

- as we can saw in the last picture laplace method is more sensitive to noise than sobel method:

2.Hough transforms:

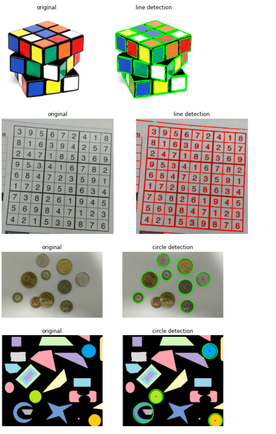
Line & circle detection :

-first i do edge detection after importing the image

-second i threshold it

-third i do line & circle detection with HoughLinesP & Houghcircle

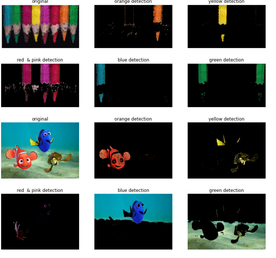
In [4]:
```
# LINE detection
img1=my_image.readimage('images/edgedetection/test1.png')
line1=my_image.line_detection2(img1,r=0,g=255,thickness=2)
Show.comparein(img1,line1,'original','line detection',size=0.8)

img4=my_image.readimage('images/mm.jpg')
line4=my_image.line_detection2(img4)
Show.comparein(img4,line4,'original','line detection',size=1)


# CIRCLE Detection
origin1=my_image.readimage('images/t2.jpg')
cimg1=my_image.circle_detector(origin1)
Show.comparein(origin1,cimg1,'original','circle detection',size=0.9)

origin2=my_image.readimage('images/t3.png')
cimg2=my_image.circle_detector(origin2,5,7,14,thickness=7)
Show.comparein(origin2,cimg2,'original','circle detection',size=0.9)
```



3. Segmentation

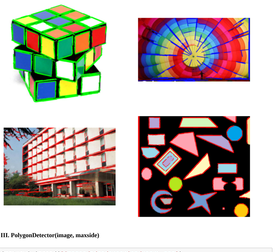
i. ColorSegmentator(image, min_color, max_color)

In [6]:
```
################################    EXAMPLE 1   ##################
############
pen = my_image.readimage('images/color2.jpg')
pen = cv2.medianBlur(pen, ksize=11)

result1_redpink=segmentation.ColorSegmentator(pen,120,179)
result1_blue=segmentation.ColorSegmentator(pen,85,120)
result1_yellow=segmentation.ColorSegmentator(pen,22,35)
result1_orange=segmentation.ColorSegmentator(pen,8,17)
result1_green=segmentation.ColorSegmentator(pen,36,80)

Show.comparein(pen,result1_orange,'original','orange detection'
               ,1.3,1,result1_yellow,'yellow detection')
Show.comparein(result1_redpink,result1_blue,'red  & pink detection',
               'blue detection',1.3,1,result1_green,'green detection')

################################    EXAMPLE 2   ##################
############
nemo = my_image.readimage('images/color1.jpg')
result1_redpink=segmentation.ColorSegmentator(nemo,119,179)
result1_blue=segmentation.ColorSegmentator(nemo,85,120)
result1_yellow=segmentation.ColorSegmentator(nemo,22,35)
result1_orange=segmentation.ColorSegmentator(nemo,8,13)
result1_green=segmentation.ColorSegmentator(nemo,36,80)

Show.comparein(nemo,result1_orange,'original','orange detection'
               ,1.3,1,result1_yellow,'yellow detection')
Show.comparein(result1_redpink,result1_blue,'red  & pink detection',
               'blue detection',1.3,1,result1_green,'green detection')
```



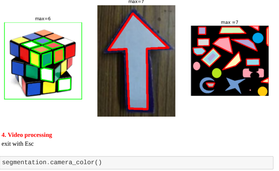
II. LinesDetector(image, minlenght)

In [7]:
```
img1=my_image.readimage('images/edgedetection/test1.png')
img2=my_image.readimage('images/test5.png')
img3=my_image.readimage('images/test8.jpg')
img4=my_image.readimage('images/t3.png')

line1=segmentation.LinesDetector(img1,minlenght=30,r=0,g=255,b=0)
line2=segmentation.LinesDetector(img2,minlenght=30,r=0,g=0,b=255)
line3=segmentation.LinesDetector(img3,minlenght=30)
line4=segmentation.LinesDetector(img4,minlenght=10)

Show.comparein(line1,line2)
Show.comparein(line3,line4)
```



III. PolygonDetector(image, maxside)

In [8]:
```
img1=cv2.imread('images/edgedetection/test1.png')
img2=cv2.imread('images/t4.jpg')
img4=cv2.imread('images/t3.png')

s1=segmentation.PolygonDetector(img1,0,r=0,g=255,b=0,thickness=8)
s2=segmentation.PolygonDetector(img2,7)
s4=segmentation.PolygonDetector(img4,7,thickness=11)

Show.comparein(cv2.cvtColor(s1, cv2.COLOR_BGR2RGB),cv2.cvtColor(s2, cv2.
COLOR_BGR2RGB)
               ,'max=6','max=7',1.3,1,cv2.cvtColor(s4, cv2.COLOR_BGR2RGB
),'max =7')
```



4. Video processing

exit with Esc

In [12]:
```
segmentation.camera_color()
```

In [13]:
```
segmentation.camera_Line(minlenght=30)
```

In [14]:
```
segmentation.camera_Polygon(maxside=30)
```

X. Checkpoint

Part 3: Segmentation

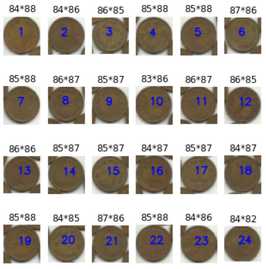
In [1]:
```
import matplotlib.pyplot as plt
import cv2
from Image import Show
from ImageSignal import ImageSignal,pyzbar
```

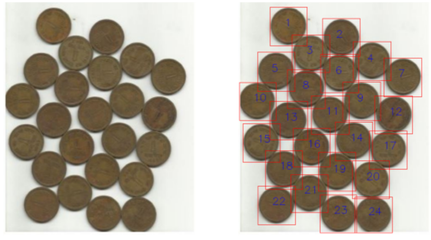
A : find Layers:

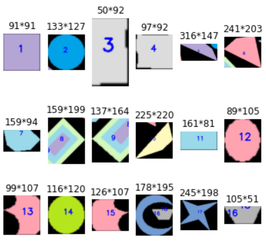
In [2]:
```
img1 = cv2.imread('images/qr2.png')
s1= ImageSignal.segment_watershed(img1,inv=True,size=1)

img2 = cv2.imread('images/tq.jpg')
s2= ImageSignal.segment_watershed(img2,inv=True,size=1)

img3 = cv2.imread('images/t2.jpg')
s3= ImageSignal.segment_watershed(img3,inv=True,size=2,num=True)

img4 = cv2.imread('images/t3.png')
s4= ImageSignal.segment_watershed(img4,inv=False,size=3)

img5 = cv2.imread('images/t4.jpg')
s5= ImageSignal.segment_watershed(img5,inv=False,size=2,num=False)

img6 = cv2.imread('images/qr4.png')
s6= ImageSignal.segment_watershed(img6,inv=True,size=2,num=False)

Show.compareim(cv2.cvtColor(s1,cv2.COLOR_BGR2RGB),cv2.cvtColor(s2,cv2.CO
LOR_BGR2RGB),'','',1.4,1,
               cv2.cvtColor(s3,cv2.COLOR_BGR2RGB))
Show.compareim(cv2.cvtColor(s5,cv2.COLOR_BGR2RGB),cv2.cvtColor(s6,cv2.CO
LOR_BGR2RGB),'','',1.4,1,
              cv2.cvtColor(s4,cv2.COLOR_BGR2RGB))
```

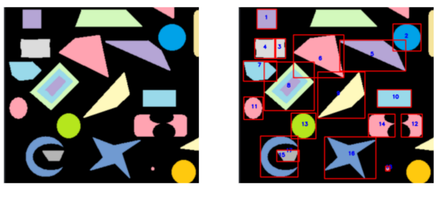
B : find pixel and seperate Layers:

In [3]:
```
#####################################     EXAMPLE 1   ###################
#############
img2 = cv2.imread('images/tq.jpg')
warped2,pixel2=[],[]
pts2,image2= ImageSignal.QR_point(img2.copy(),inv=True,ratio=40,num=True
)
for row in pts2:
    [warp,width,height]=ImageSignal.four_point_transform(image2,row)
    warped2.append(warp)
    pixel2.append(str(width)+'*'+str(height))
Show.show_segments(warped2,pixel2,Qr=0)
Show.compareim(cv2.cvtColor(img2,cv2.COLOR_BGR2RGB),
               cv2.cvtColor(s2,cv2.COLOR_BGR2RGB),size=1);plt.show()
#####################################     EXAMPLE 2   ###################
#############
img4 = cv2.imread('images/t3.png')
warped4,pixel4=[],[]
pts4,image4= ImageSignal.QR_point(img4.copy(),inv=False,ratio=16,num=Tru
e)
for row in pts4:
    [warp,width,height]=ImageSignal.four_point_transform(image4,row)
    warped4.append(warp)
    pixel4.append(str(width)+'*'+str(height))
Show.show_segments(warped4,pixel4,Qr=0)
Show.compareim(cv2.cvtColor(img4,cv2.COLOR_BGR2RGB),
               cv2.cvtColor(s4,cv2.COLOR_BGR2RGB),size=1);plt.show()
```









X. Checkpoint