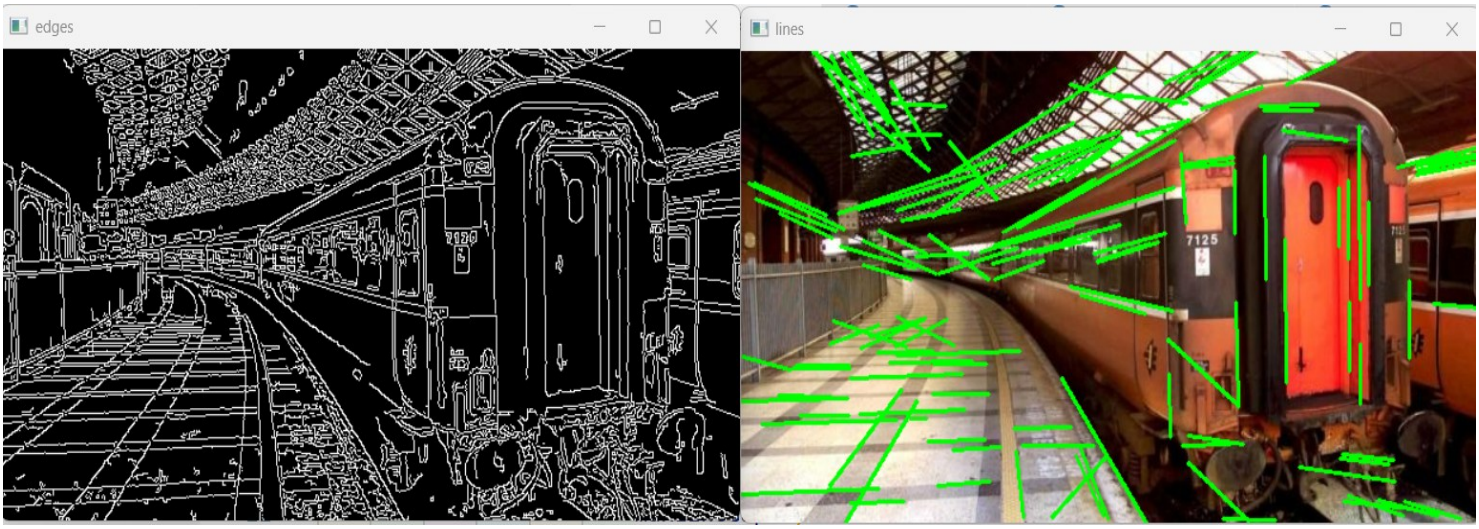
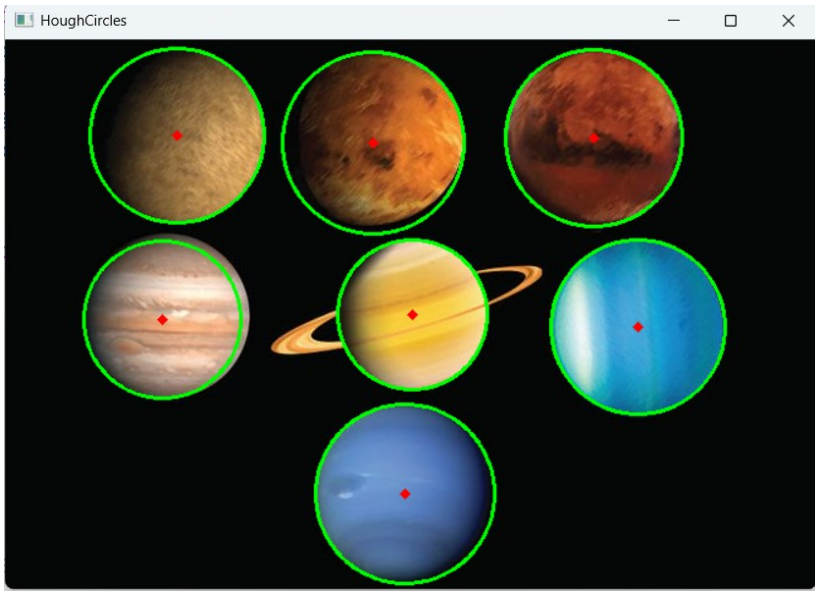
Line Detection

```python
# line detection using Hough Transform
import cv2
import numpy as np
img = cv2.imread('images/houghlines5.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
edges = cv2.Canny(gray, 50, 120)
lines = cv2.HoughLinesP(edges, rho=1,
                        theta=np.pi/180.0,
                        threshold=20,
                        minLineLength=40,
                        maxLineGap=5)
for line in lines:
    x1, y1, x2, y2 = line[0]
    cv2.line(img, (x1, y1), (x2, y2), (0, 255, 0), 2)
cv2.imshow("edges", edges)
cv2.imshow("lines", img)
cv2.waitKey()
cv2.destroyAllWindows()
```



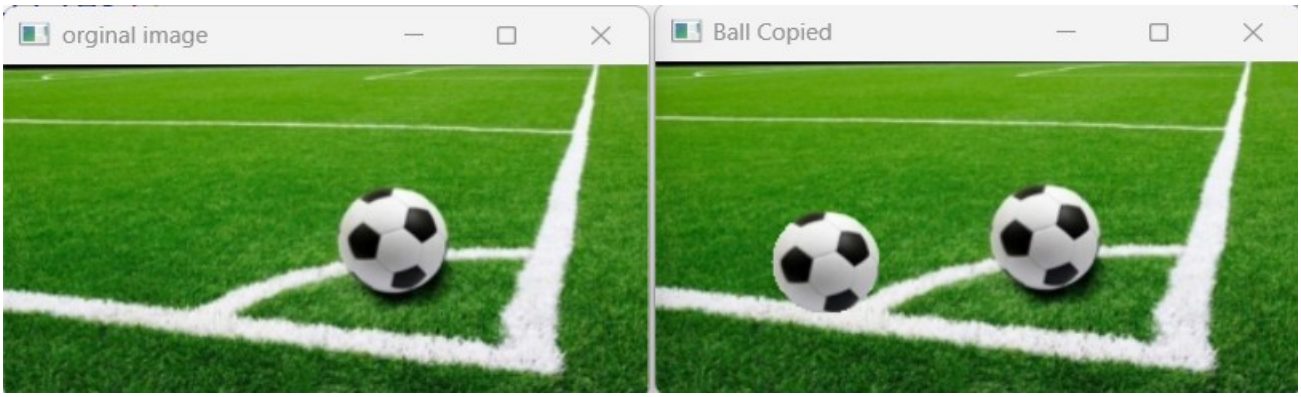
Circles Detection

```python
# Circles detection using Hough transform
import cv2
import numpy as np
planets = cv2.imread("images/planet_glow.jpg")
gray_img = cv2.cvtColor(planets, cv2.COLOR_BGR2GRAY)
gray_img = cv2.medianBlur(gray_img, 5)
circles = cv2.HoughCircles(gray_img, cv2.HOUGH_GRADIENT,
                           1, 120, param1=90, param2=40,
                           minRadius=0, maxRadius=0)
if circles is not None:
    circles = np.uint16(np.around(circles))
    for i in circles[0,:]:
        # draw the outer circle
        cv2.circle(planets, (i[0], i[1]), i[2],
                   (0, 255, 0), 2)
        # draw the center of the circle
        cv2.circle(planets, (i[0], i[1]), 2,
                   (0, 0, 255), 3)
cv2.imwrite("planets_hough_circles.jpg", planets)
cv2.imshow("HoughCircles", planets)
cv2.waitKey()
cv2.destroyAllWindows()
```



```python
import cv2 as cv
import numpy as np
img = cv.imread('images/foot_ball.png')
output = img.copy()
gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
gray = cv.medianBlur(gray, 5)
# اكتشاف الكرة باستخدام HoughCircles
circles = cv.HoughCircles(
    gray,cv.HOUGH_GRADIENT,
    dp=1.2,minDist=50,
    param1=100,param2=30,
    minRadius=20,maxRadius=80)
if circles is not None:
    circles = np.uint16(np.around(circles))
    for (x, y, r) in circles[0, :]:
        # إنشاء قناع للكرة
        mask = np.zeros_like(gray)
        cv.circle(mask, (x, y), r, 255, -1)

        # قص المنطقة المحيطة بالكرة من الصورة الأصلية
        y1, y2 = y - r, y + r
        x1, x2 = x - r, x + r
        ball_crop = img[y1:y2, x1:x2]
        mask_crop = mask[y1:y2, x1:x2]
        # المكان الجديد للصق الكرة
        ny, nx = 78, 58
        roi = output[ny:ny+ball_crop.shape[0], nx:nx+ball_crop.shape[1]]
        # دمج الكرة مع الخلفية بدون جعل الخلفية سوداء
        ball_area = cv.bitwise_and(ball_crop, ball_crop, mask=mask_crop)
        bg_area = cv.bitwise_and(roi, roi, mask=cv.bitwise_not(mask_crop))
        combined = cv.add(bg_area, ball_area)
        # وضع النتيجة على الصورة
        output[ny:ny+ball_crop.shape[0], nx:nx+ball_crop.shape[1]] = combined
cv.imshow('orginal image', img )
cv.imshow('Ball Copied', output)
cv.waitKey(0)
cv.destroyAllWindows()
```



Corners Detection

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
img = cv2.imread('images/corners.png')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# Shi-Tomasi Corner Detection
corners = cv2.goodFeaturesToTrack(gray, maxCorners=0,
                                  qualityLevel=0.01, minDistance=10)
corners = np.intp(corners)
for c in corners:
    x, y = c.ravel()
    cv2.circle(img, (x,y), 4, (255,0,0), -1)
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.title("Corner Detection")
plt.show()
```

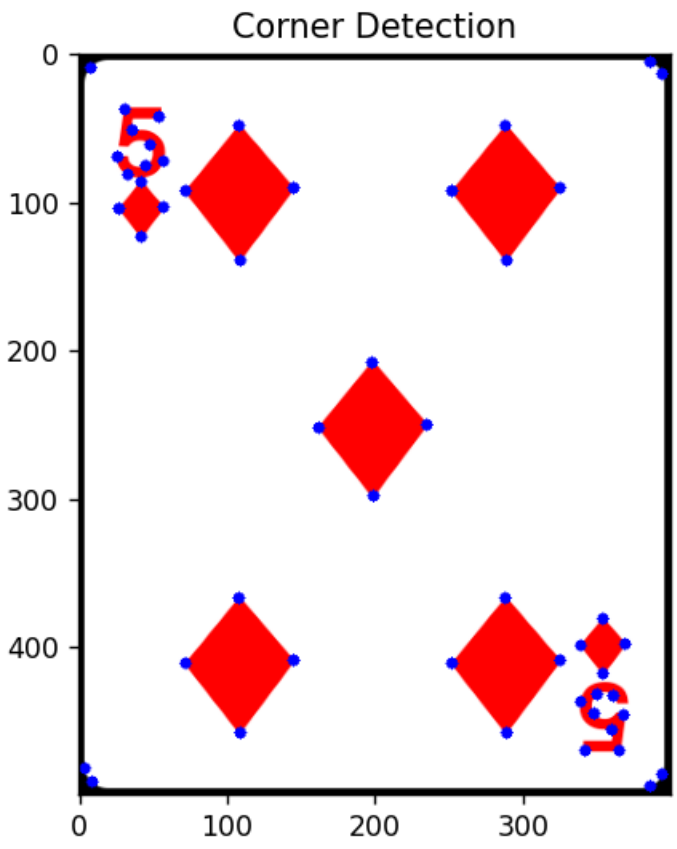
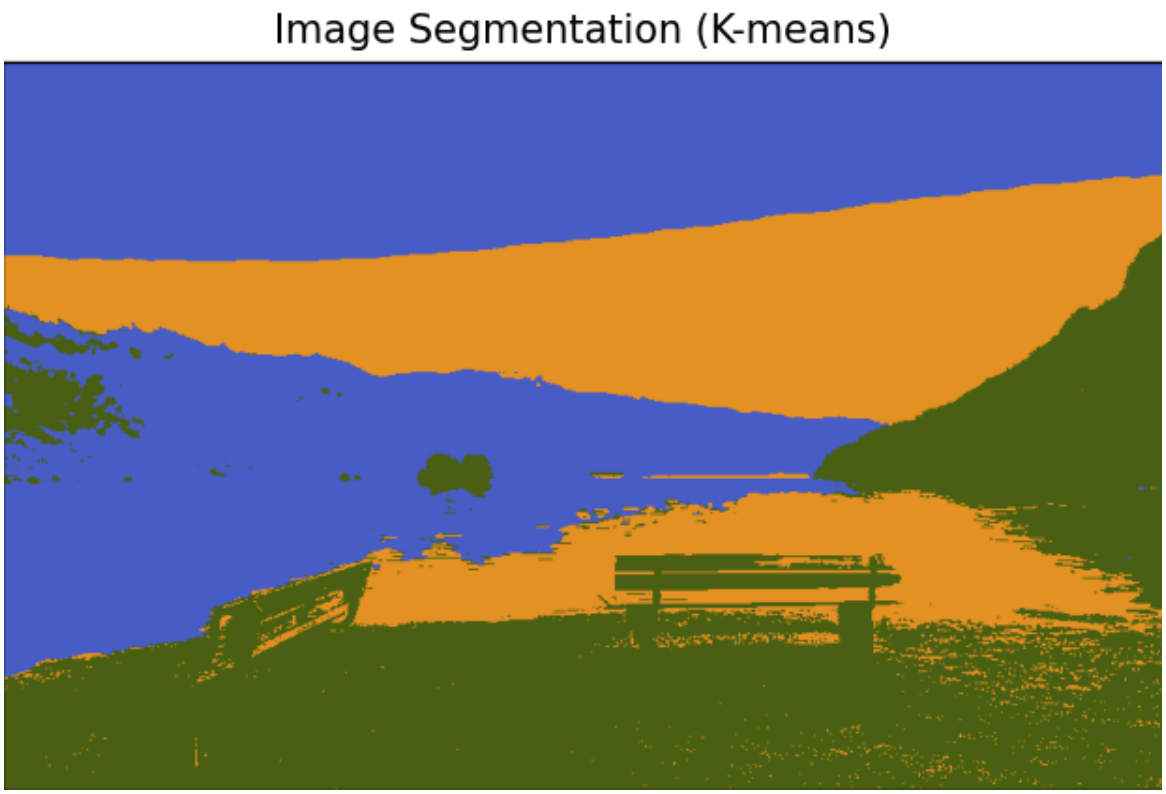


Image Segmentation

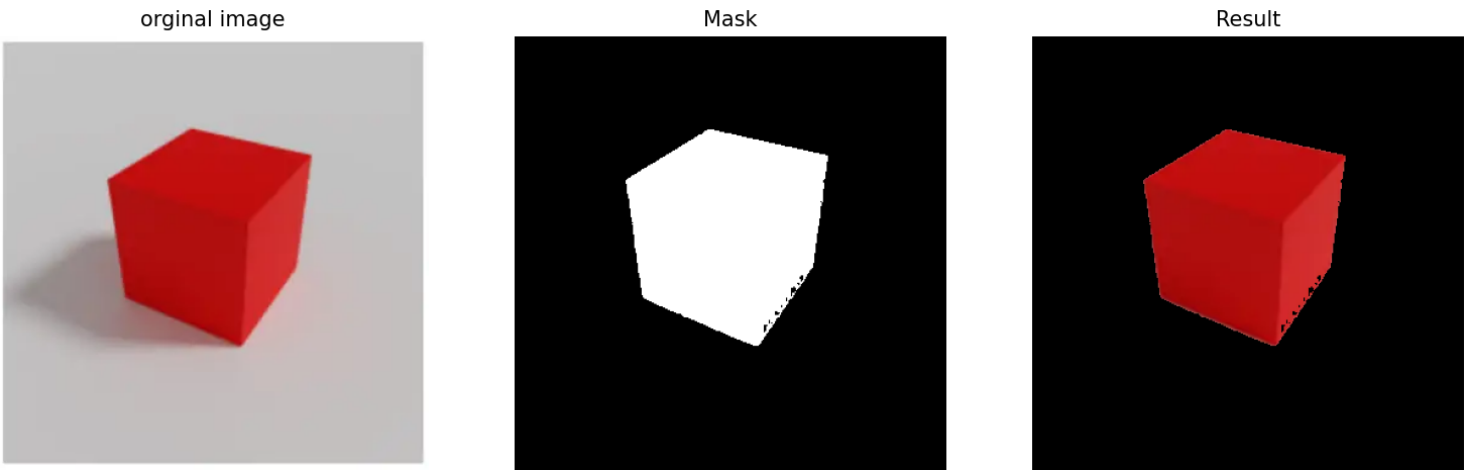
1. Image Segmentation using K-means

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
img = cv2.imread('images/images1.webp')
Z = img.reshape((-1,3))
Z = np.float32(Z)
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 10, 1.0)
K = 3
_, label, center = cv2.kmeans(Z, K, None, criteria, 10, cv2.KMEANS_RANDOM_CENTERS)
center = np.uint8(center)
res = center[label.flatten()]
segmented_img = res.reshape((img.shape))
plt.imshow(cv2.cvtColor(segmented_img, cv2.COLOR_BGR2RGB))
plt.title("Image Segmentation (K-means)")
plt.show()
```



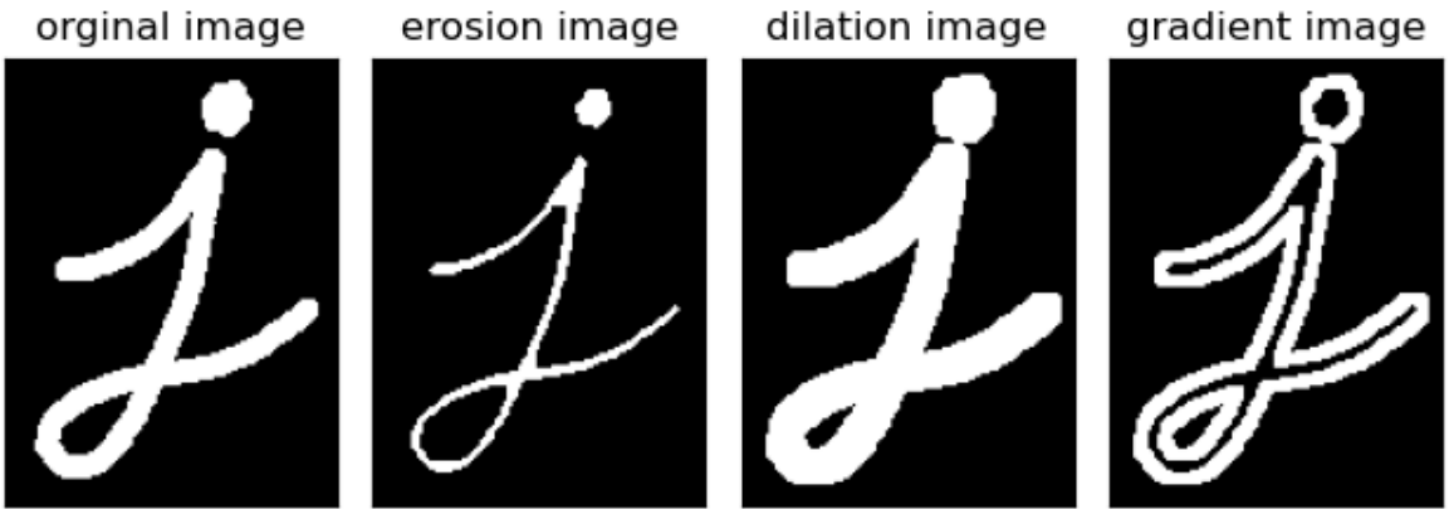
2. Image Segmentation using Color Masking

```python
import numpy as np
import matplotlib.pyplot as plt
import cv2
sample_image = cv2.imread('images/shapes.webp')
img = cv2.cvtColor(sample_image, cv2.COLOR_BGR2RGB)
# لاستخراج اللون المطلوب (بالـ RGB) تحديد مجال الألوان
low = np.array([0, 0, 0])       # الحد الأدنى للألوان
high = np.array([215, 80, 80])  # الحد الأعلى للألوان
# إنشاء قناع (mask) يحدد البكسلات داخل المجال
mask = cv2.inRange(img, low, high)
result = cv2.bitwise_and(img, img, mask=mask)
fig, axes = plt.subplots(1, 3, figsize=(15, 5))
axes[0].imshow(img)
axes[0].set_title("orginal image")
axes[0].axis("off")
axes[1].imshow(mask, cmap="gray")
axes[1].set_title("Mask")
axes[1].axis("off")
axes[2].imshow(result)
axes[2].set_title("Result")
axes[2].axis("off")
plt.show()
```

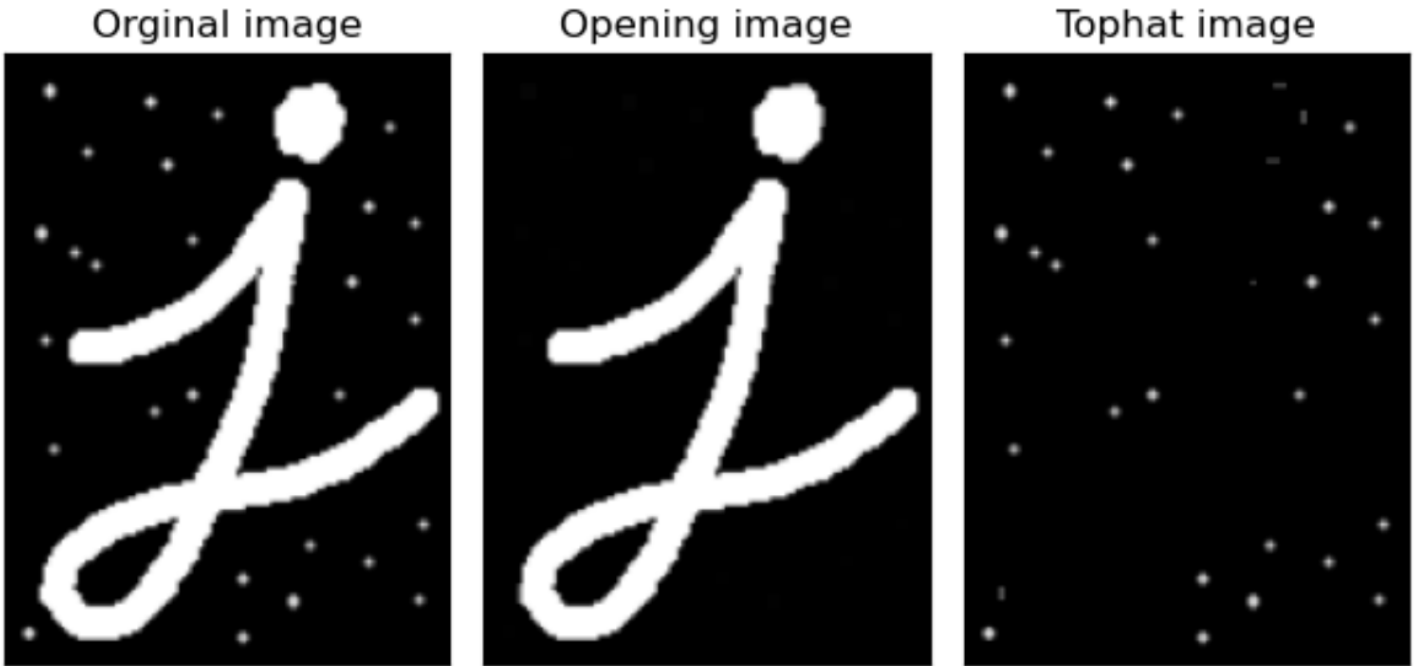


Morphological Transformation

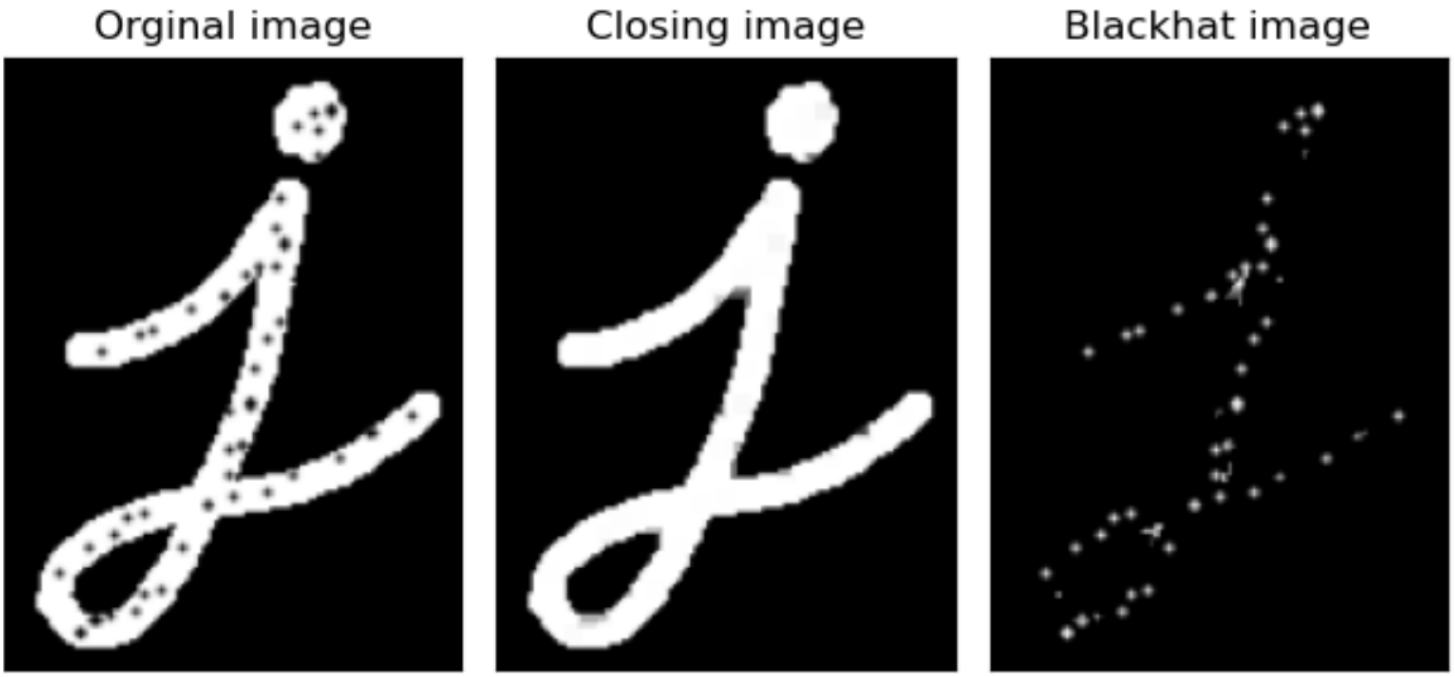
```python
#morphological Transformation
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
img = cv.imread('images/i.png', cv.IMREAD_GRAYSCALE)
kernel = np.ones((5,5),np.uint8)
erosion = cv.erode(img,kernel,iterations = 1)#shrinks objects=>Eliminating small object
dilation = cv.dilate(img,kernel,iterations = 1)#expands objects=>filling in small holse
gradient = cv.morphologyEx(img, cv.MORPH_GRADIENT, kernel)#the difference between dilation and erosion
plt.subplot(141),plt.imshow(cv.cvtColor(img, cv.COLOR_BGR2RGB)),plt.title("orginal image"),plt.xticks([]),plt.yticks([])
plt.subplot(142),plt.imshow(cv.cvtColor(erosion, cv.COLOR_BGR2RGB)),plt.title('erosion image'),plt.xticks([]),plt.yticks([])
plt.subplot(143),plt.imshow(cv.cvtColor(dilation, cv.COLOR_BGR2RGB)),plt.title('dilation image'),plt.xticks([]),plt.yticks([])
plt.subplot(144),plt.imshow(cv.cvtColor(gradient, cv.COLOR_BGR2RGB)),plt.title('gradient image'),plt.xticks([]),plt.yticks([])
plt.tight_layout()
plt.show()
```



```python
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
img = cv.imread('images/i_noise.png', cv.IMREAD_GRAYSCALE)
kernel = np.ones((5,5),np.uint8)
#opening consists of an erosion followed by a dilation
opening = cv.morphologyEx(img, cv.MORPH_OPEN, kernel)
#TOP HAT is the difference between input image and opening of the image
tophat = cv.morphologyEx(img, cv.MORPH_TOPHAT, kernel)
plt.subplot(131),plt.imshow(cv.cvtColor(img, cv.COLOR_BGR2RGB)),plt.title("Orginal image"),plt.xticks([]),plt.yticks([])
plt.subplot(132),plt.imshow(cv.cvtColor(opening, cv.COLOR_BGR2RGB)),plt.title('Opening image'),plt.xticks([]),plt.yticks([])
plt.subplot(133),plt.imshow(cv.cvtColor(tophat, cv.COLOR_BGR2RGB)),plt.title('Tophat image'),plt.xticks([]),plt.yticks([])
plt.tight_layout()
plt.show()
```



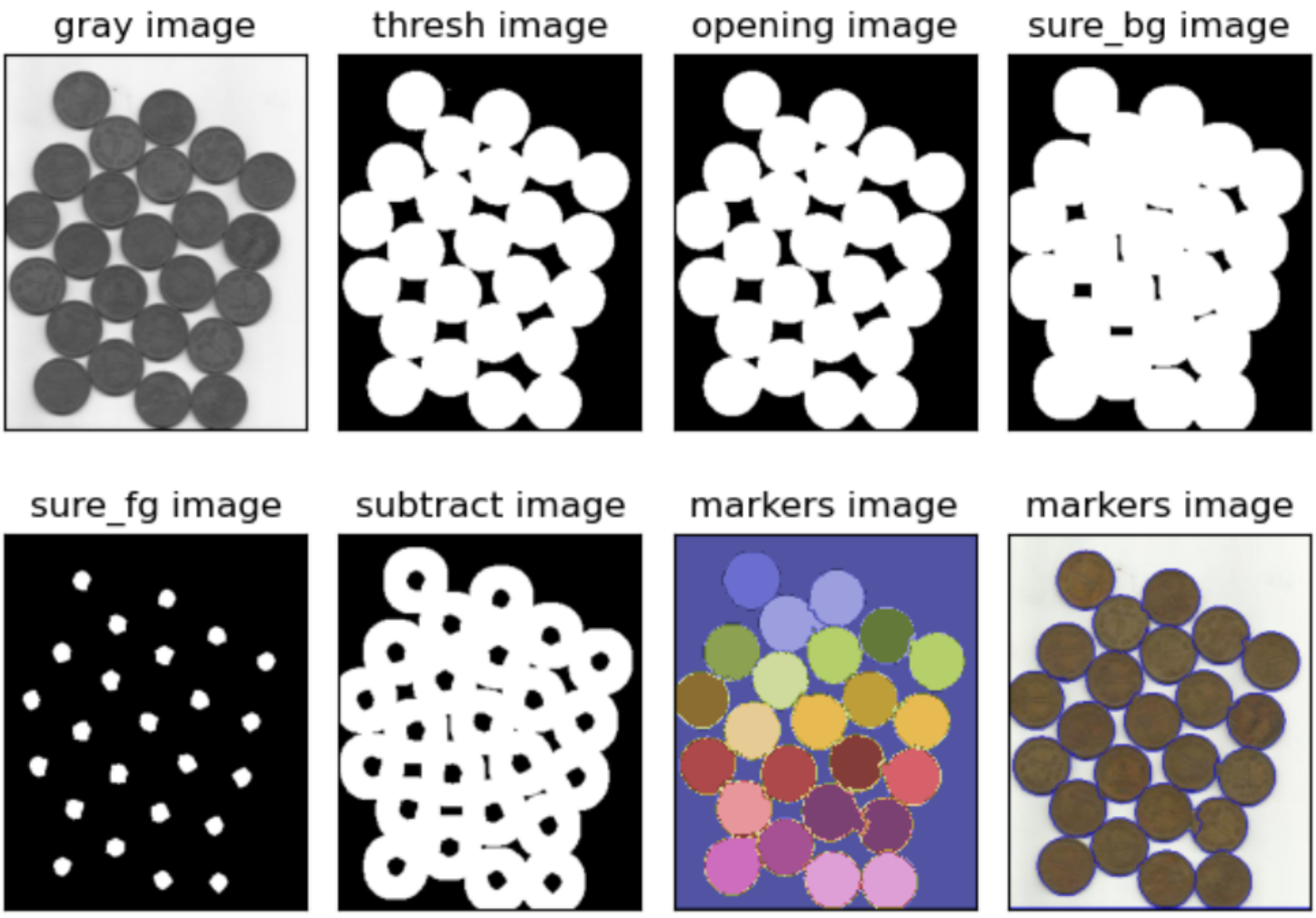
```python
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
img = cv.imread('images/i_noisee.png', cv.IMREAD_GRAYSCALE)
kernel = np.ones((5,5),np.uint8)
#closing consists of a dilation followed by an erosion
closing = cv.morphologyEx(img, cv.MORPH_CLOSE, kernel)
#BLACK HAT is the difference between input image and closing of the imag
blackhat = cv.morphologyEx(img, cv.MORPH_BLACKHAT, kernel)
plt.subplot(131),plt.imshow(cv.cvtColor(img, cv.COLOR_BGR2RGB)),plt.title("Orginal image"),plt.xticks([]),plt.yticks([])
plt.subplot(132),plt.imshow(cv.cvtColor(closing, cv.COLOR_BGR2RGB)),plt.title('Closing image'),plt.xticks([]),plt.yticks([])
plt.subplot(133),plt.imshow(cv.cvtColor(blackhat, cv.COLOR_BGR2RGB)),plt.title('Blackhat image'),plt.xticks([]),plt.yticks([])
plt.tight_layout()
plt.show()
```



3. Image segmentation using watershed algorithm

```python
#image segmentation->watershed algorithm
import cv2
import numpy as np
import matplotlib.pyplot as plt
img = cv2.imread('images/water_coins.jpg')
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
ret, thresh =cv2.threshold(gray,0,255,cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
# noise removal
kernel = np.ones((3,3),np.uint8)
opening = cv2.morphologyEx(thresh,cv2.MORPH_OPEN,kernel, iterations = 2)
# sure background area
sure_bg = cv2.dilate(opening,kernel,iterations=3)
# Finding sure foreground area
dist_transform = cv2.distanceTransform(opening,cv2.DIST_L2,5)
ret, sure_fg = cv2.threshold(dist_transform,0.7*dist_transform.max(),255,0)
# Finding unknown region
sure_fg = np.uint8(sure_fg)
unknown = cv2.subtract(sure_bg,sure_fg)
# Marker labelling
ret, markers = cv2.connectedComponents(sure_fg)
# Add one to all labels so that sure background is not 0, but 1
markers = markers+1

# Now, mark the ' of unknown with zero
markers[unknown==255] = 0
markers = cv2.watershed(img,markers)
seg=img.copy()
seg[markers == -1] = [255,0,0]
plt.subplot(241),plt.imshow(cv2.cvtColor(gray, cv2.COLOR_BGR2RGB)),plt.title("gray image"),plt.xticks([]),plt.yticks([])
plt.subplot(242),plt.imshow(cv2.cvtColor(thresh, cv2.COLOR_BGR2RGB)),plt.title('thresh image'),plt.xticks([]),plt.yticks([])
plt.subplot(243),plt.imshow(cv2.cvtColor(opening, cv2.COLOR_BGR2RGB)),plt.title('opening image'),plt.xticks([]),plt.yticks([])
plt.subplot(244),plt.imshow(cv2.cvtColor(sure_bg, cv2.COLOR_BGR2RGB)),plt.title('sure_bg image'),plt.xticks([]),plt.yticks([])
plt.subplot(245),plt.imshow(cv2.cvtColor(sure_fg, cv2.COLOR_BGR2RGB)),plt.title('sure_fg image'),plt.xticks([]),plt.yticks([])
plt.subplot(246),plt.imshow(cv2.cvtColor(unknown, cv2.COLOR_BGR2RGB)),plt.title('subtract image'),plt.xticks([]),plt.yticks([])
plt.subplot(247),plt.imshow(markers, cmap="tab20b"),plt.title('markers image'),plt.xticks([]),plt.yticks([])
plt.subplot(248),plt.imshow(cv2.cvtColor(seg, cv2.COLOR_BGR2RGB)),plt.title('segment image'),plt.xticks([]),plt.yticks([])
plt.tight_layout()
plt.show()
```


gray image
thresh image
opening image
sure_bg image
sure_fg image
subtract image
markers image
markers image