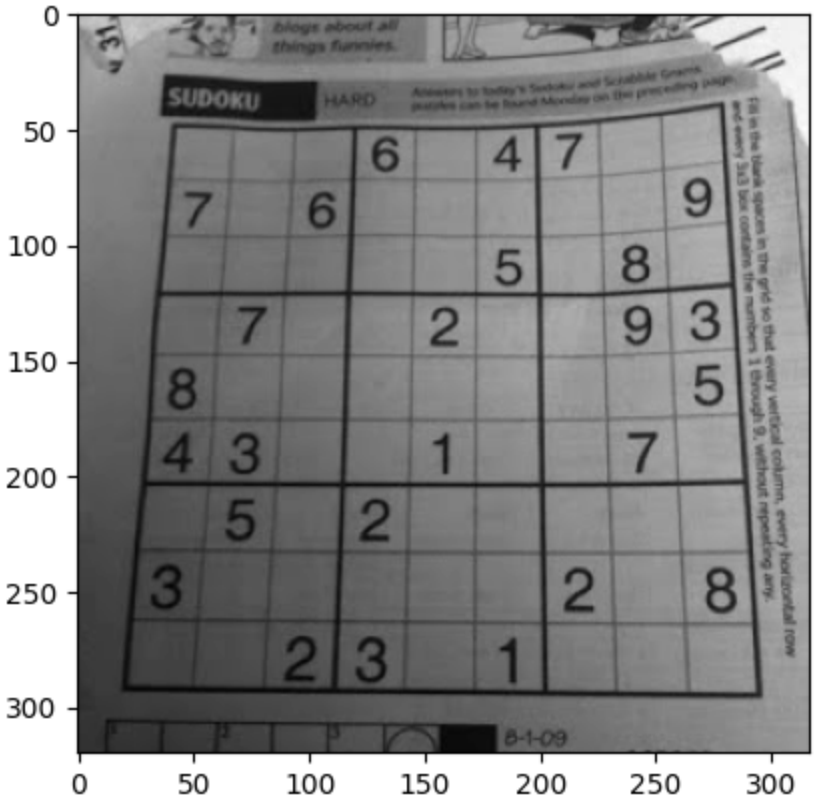
In [1]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
img = cv2.imread('img/sudoku.jpg')
img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
plt.imshow(img, cmap='gray')
```

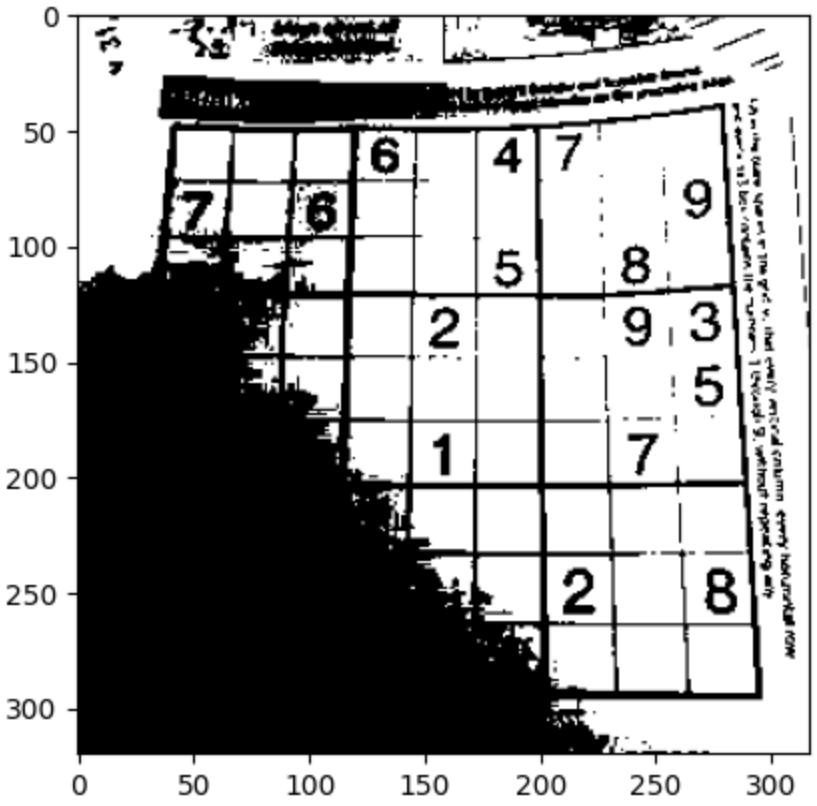
Out[2]:
```
<matplotlib.image.AxesImage at 0x16be4ccfe20>
```



In [3]:
```
# Global thresholding
def thresholding(img, threshold):
    new_img = np.zeros(img.shape)
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            if img[i, j] < threshold:
                new_img[i, j] = 0
            else:
                new_img[i, j] = 255
    return new_img
```

In [4]:
```
thresh_img = thresholding(img, 100)
plt.imshow(thresh_img, cmap='gray')
```

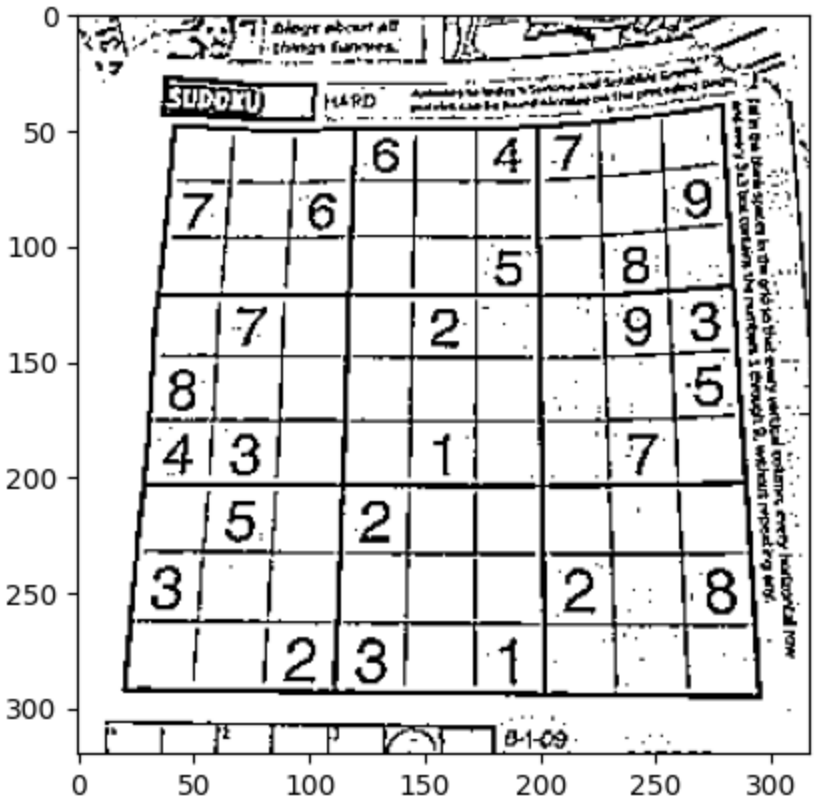
Out[4]:
```
<matplotlib.image.AxesImage at 0x16bd49c5eb0>
```



In [5]:
```python
# Adaptive thresholding
def adaptive_thresholding(img: np.ndarray, block_size: int, C: int) -> np.ndarray:
    warn = "block_size must be an odd positive integer"
    assert block_size % 2 == 1 and block_size > 0, warn
    height, width = img.shape
    new_img = np.zeros(img.shape)
    for i in range(height):
        for j in range(width):
            x_min = max(0, i - block_size // 2)
            y_min = max(0, j - block_size // 2)
            x_max = min(height - 1, i + block_size // 2)
            y_max = min(width - 1, j + block_size // 2)
            block = img[x_min:x_max+1, y_min:y_max+1]
            thresh = np.mean(block) - C
            if img[i, j] >= thresh:
                new_img[i, j] = 255
    return new_img
```

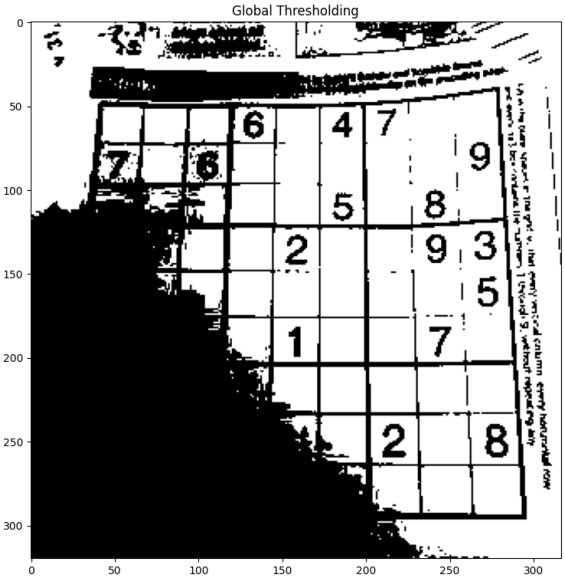
In [6]:
```python
adaptive_thresholding_img = adaptive_thresholding(img, 5, 5)
plt.imshow(adaptive_thresholding_img, cmap='gray')
```

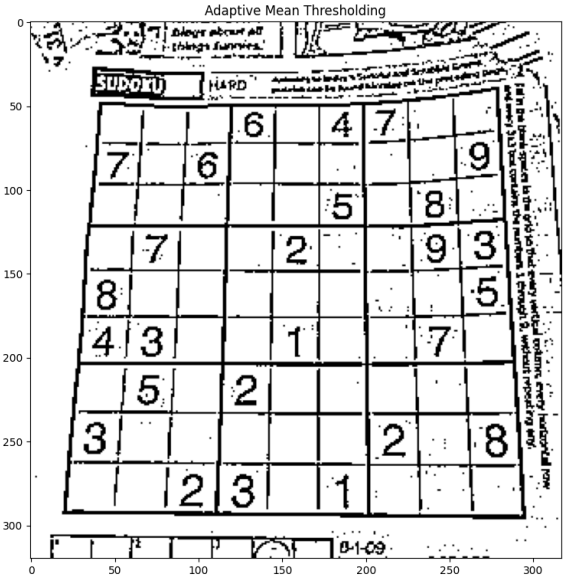
Out[6]:
```
<matplotlib.image.AxesImage at 0x16be4dcc040>
```

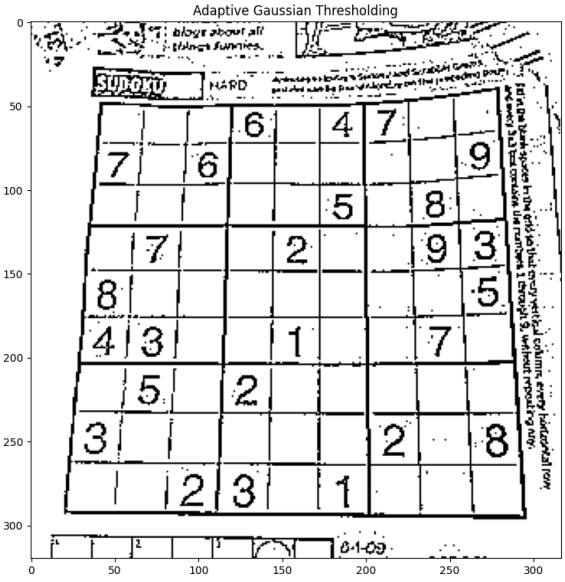


In [7]:
```
# Thresholding with opencv

global_thresh = cv2.threshold(img, 100, 255, cv2.THRESH_BINARY)[1]
adaptive_mean = cv2.adaptiveThreshold(img, 255,
                                      cv2.ADAPTIVE_THRESH_MEAN_C,
                                      cv2.THRESH_BINARY, 5, 5)
adaptive_gaussian = cv2.adaptiveThreshold(img, 255,
                                          cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
                                          cv2.THRESH_BINARY, 5, 5)

fig = plt.figure(figsize=(20, 20))
plt.subplot(2, 2, 1)
plt.title('Original Image')
plt.imshow(img, cmap='gray')

plt.subplot(2, 2, 2)
plt.title('Global Thresholding')
plt.imshow(global_thresh, cmap='gray')

plt.subplot(2, 2, 3)
plt.title('Adaptive Mean Thresholding')
plt.imshow(adaptive_mean, cmap='gray')

plt.subplot(2, 2, 4)
plt.title('Adaptive Gaussian Thresholding')
plt.imshow(adaptive_gaussian, cmap='gray')
```

Out[7]: <matplotlib.image.AxesImage at 0x16be6fb9a60>







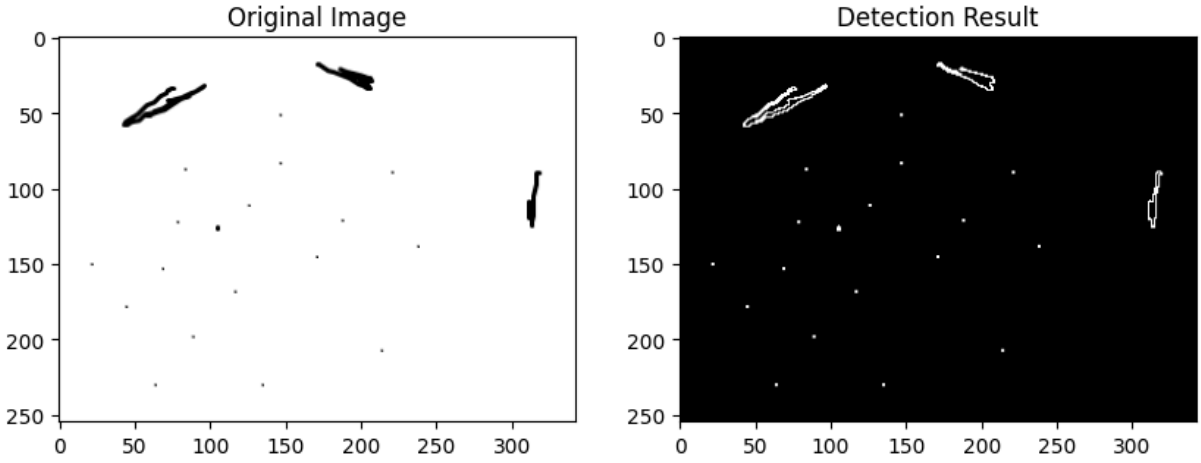
In [8]:
```python
# Isolated point detection
def point_detection(img: np.ndarray, threshold: int) -> np.ndarray:
    # Create a mask with the same size as the image
    mask = np.zeros_like(img)
    # Create Laplacian kernel
    # laplace_kernel = np.array([[0, 1, 0], [1, -4, 1], [0, 1, 0]])
    laplace_kernel = np.array([[1, 1, 1], [1, -8, 1], [1, 1, 1]])

    # apply the kernel to the image, and apply threshold
    laplacian = cv2.filter2D(img, -1, laplace_kernel)
    print(laplacian)

    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            if laplacian[i, j] < threshold:
                mask[i, j] = 0
            else:
                mask[i, j] = 255
```

```
    return mask
```

In [9]:
```
img_point = cv2.imread('img/isolated2.png')
img_point = cv2.cvtColor(img_point, cv2.COLOR_BGR2GRAY)

point_mask = point_detection(img_point, 100)

fig = plt.figure(figsize=(10, 9))
plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(img_point, cmap='gray')

plt.subplot(1, 2, 2)
plt.title('Detection Result')
plt.imshow(point_mask, cmap='gray')
```

```
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
```

Out[9]: <matplotlib.image.AxesImage at 0x16bea009370>



In [10]:
```
# Line detection
def line_detection(img: np.ndarray, threshold: int, mode: str) -> np.ndarray:
    # Create a mask with the same size as the image
    mask = np.zeros_like(img)

    # Create Horizontal and Vertical kernel
    horizontal_kernel = np.array([[-1, -1, -1], [2, 2, 2], [-1, -1, -1]])
    vertical_kernel = np.array([[-1, 2, -1], [-1, 2, -1], [-1, 2, -1]])
    kernel_45_neg = np.array([[-1, -1, 2], [-1, 2, -1], [2, -1, -1]])
    kernel_45_pos = np.array([[2, -1, -1], [-1, 2, -1], [-1, -1, 2]])


    # apply the kernel to the image

    if mode == 'horizontal':
```

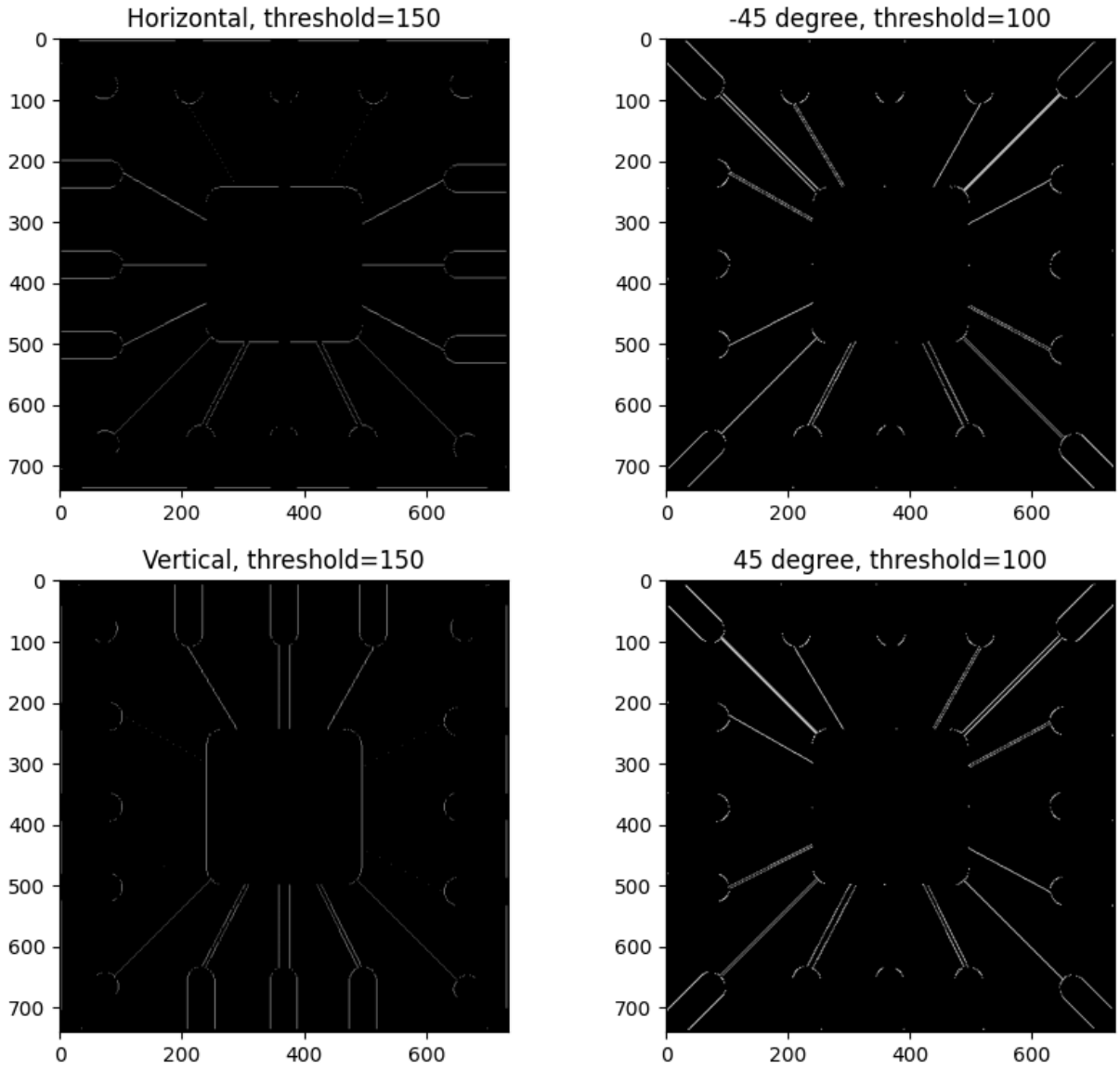
```
        kernel = np.array([[-1, -1, -1], [2, 2, 2], [-1, -1, -1]])
        kernel_apply = cv2.filter2D(img, -1, horizontal_kernel)
    elif mode == 'vertical':
        kernel = np.array([[-1, 2, -1], [-1, 2, -1], [-1, 2, -1]])
        kernel_apply = cv2.filter2D(img, -1, vertical_kernel)
    elif mode == '45_neg':
        kernel = np.array([[-1, -1, 2], [-1, 2, -1], [2, -1, -1]])
        kernel_apply = cv2.filter2D(img, -1, kernel_45_neg)
    elif mode == '45_pos':
        kernel = np.array([[2, -1, -1], [-1, 2, -1], [-1, -1, 2]])
        kernel_apply = cv2.filter2D(img, -1, kernel_45_pos)

    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            if kernel_apply[i, j] > threshold:
                mask[i, j] = 255
            else:
                mask[i, j] = 0

    return mask
```

In [11]:

```
line_detection_img = cv2.imread('img/line_detection.png', 0)
line_detected_img1 = line_detection(line_detection_img, 150, 'horizontal')
line_detected_img2 = line_detection(line_detection_img, 100, '45_neg')
line_detected_img3 = line_detection(line_detection_img, 150, 'vertical')
line_detected_img4 = line_detection(line_detection_img, 100, '45_pos')


fig = plt.figure(figsize=(10, 9))
plt.subplot(2, 2, 1)
plt.title('Horizontal, threshold=150')
plt.imshow(line_detected_img1, cmap='gray')

plt.subplot(2, 2, 2)
plt.title('-45 degree, threshold=100')
plt.imshow(line_detected_img2, cmap='gray')

plt.subplot(2, 2, 3)
plt.title('Vertical, threshold=150')
plt.imshow(line_detected_img3, cmap='gray')

plt.subplot(2, 2, 4)
plt.title('45 degree, threshold=100')
plt.imshow(line_detected_img4, cmap='gray')
```

Out[11]:

```
<matplotlib.image.AxesImage at 0x16be84422e0>
```

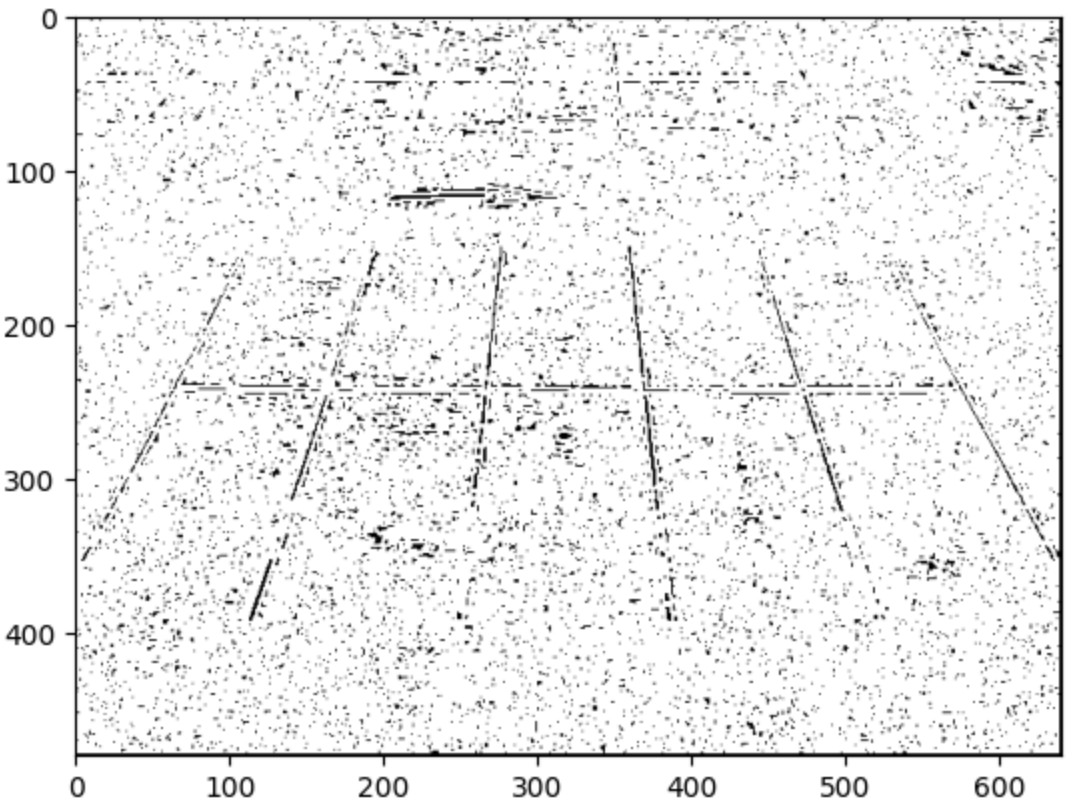


In [12]:
```python
# Edge detection: Marr-Hildreth Algorithm
def marr_hildreth_edge_detection(img: np.ndarray) -> np.ndarray:

    img = img.astype(np.float32)
    # Apply gaussian filter
    img = cv2.GaussianBlur(img, (3, 3), 0)

    # Apply laplace filter
    laplace_kernel = np.array([[0, 1, 0], [1, -4, 1], [0, 1, 0]])
    # laplace_kernel = np.array([[1, 1, 1], [1, -8, 1], [1, 1, 1]])
    laplacian = cv2.filter2D(img, -1, laplace_kernel)
    # laplacian = cv2.Laplacian(img, cv2.CV_64F, ksize=3)

    # zero crossing
    new_img = np.zeros(laplacian.shape)

    for i in range(1, laplacian.shape[0] - 1):
        for j in range(1, laplacian.shape[1] - 1):
            if laplacian[i, j] == 0:
                continue
```

```
            elif laplacian[i-1, j-1] * laplacian[i, j] < 0:
                new_img[i, j] = 255
            elif laplacian[i-1, j] * laplacian[i, j] < 0:
                new_img[i, j] = 255
            elif laplacian[i-1, j+1] * laplacian[i, j] < 0:
                new_img[i, j] = 255
            elif laplacian[i, j-1] * laplacian[i, j] < 0:
                new_img[i, j] = 255
            elif laplacian[i, j+1] * laplacian[i, j] < 0:
                new_img[i, j] = 255
            elif laplacian[i+1, j-1] * laplacian[i, j] < 0:
                new_img[i, j] = 255
            elif laplacian[i+1, j] * laplacian[i, j] < 0:
                new_img[i, j] = 255
            elif laplacian[i+1, j+1] * laplacian[i, j] < 0:
                new_img[i, j] = 255

    new_img = new_img.astype(np.uint8)
    return new_img
```

In [13]:
```
edge_detect = cv2.imread('img/line.png')
gray = cv2.cvtColor(edge_detect, cv2.COLOR_BGR2GRAY)
edge_detected_img = marr_hildreth_edge_detection(gray)
plt.imshow(edge_detected_img, cmap='gray')
```

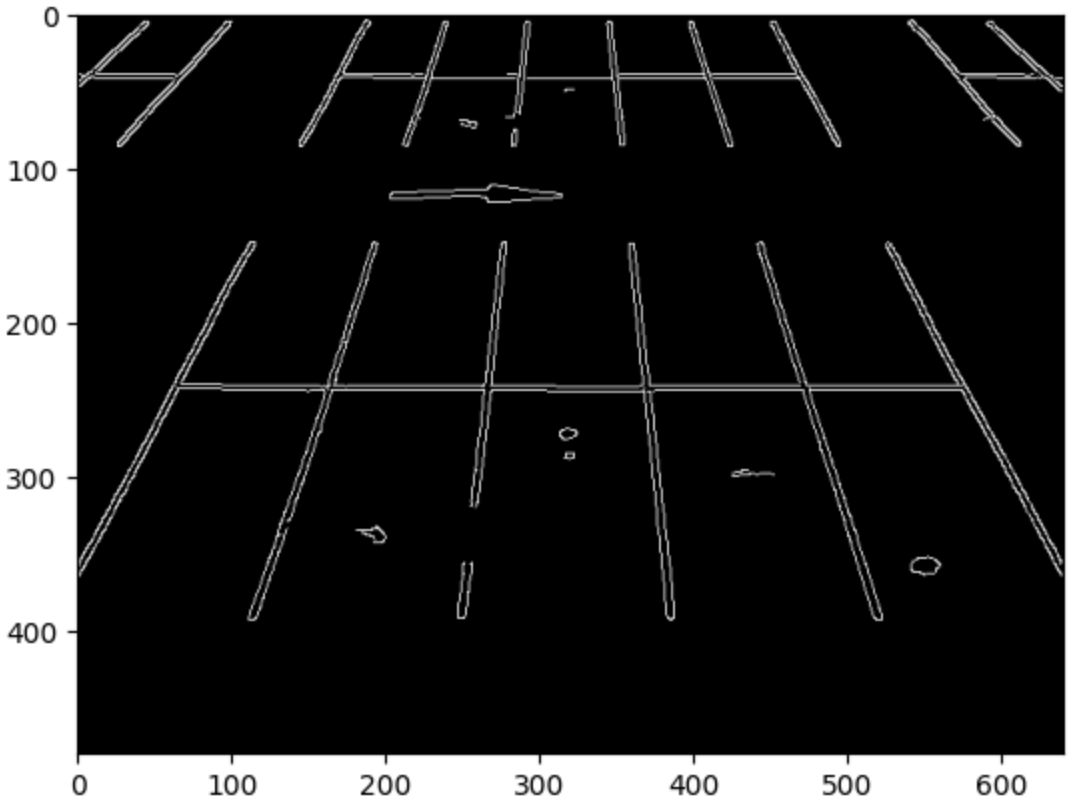
Out[13]: <matplotlib.image.AxesImage at 0x16be8255730>



In [14]:
```
edge_detect = cv2.imread('img/line.png')
gray = cv2.cvtColor(edge_detect, cv2.COLOR_BGR2GRAY)
blur = cv2.GaussianBlur(gray, (5, 5), 0)
```

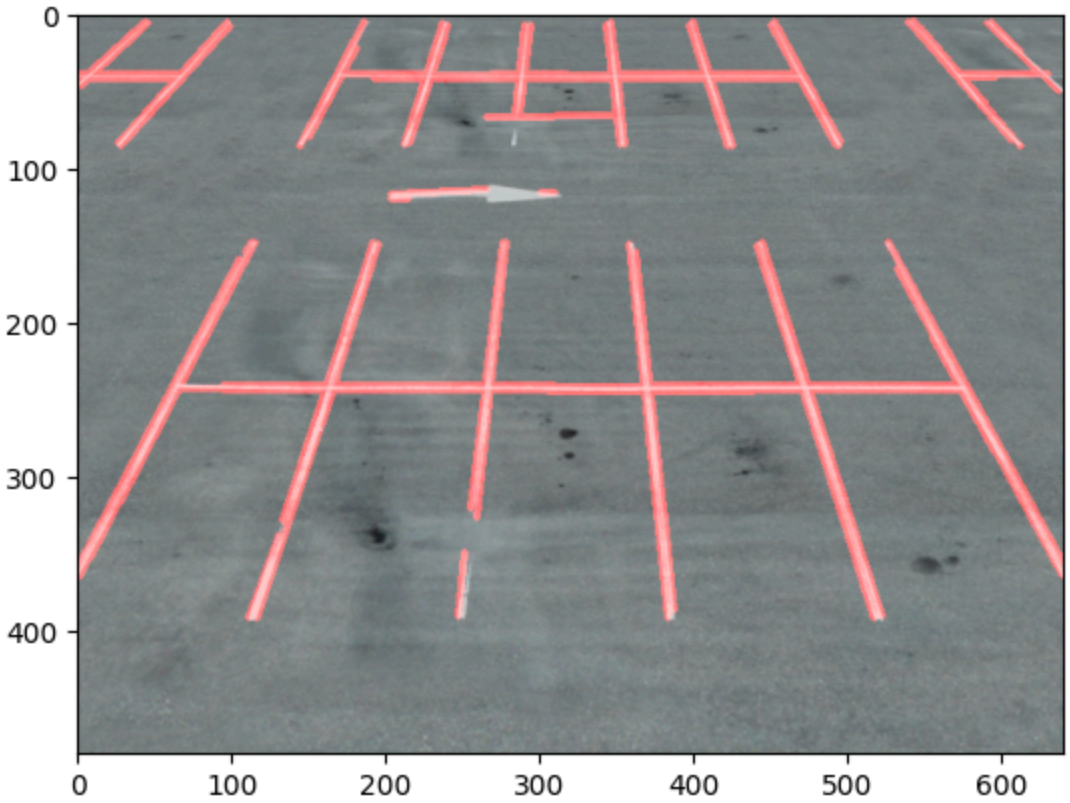
```
img_canny = cv2.Canny(blur, 50, 200)
plt.imshow(img_canny, cmap='gray')
```

Out[14]: <matplotlib.image.AxesImage at 0x16be8407d60>



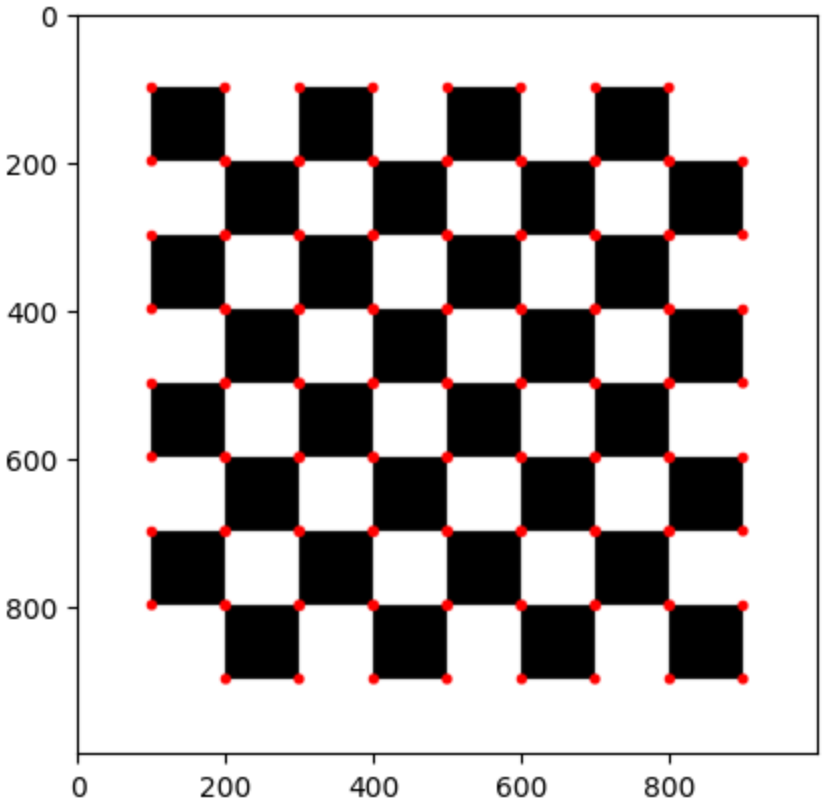
In [15]:
```
# Hough Line Transform
edge_detection = cv2.imread('img/line.png')
gray = cv2.cvtColor(edge_detection, cv2.COLOR_BGR2GRAY)
blur = cv2.GaussianBlur(gray, (5, 5), 0)
img_canny = cv2.Canny(blur, 5, 100)
lines = cv2.HoughLinesP(img_canny, 1, np.pi / 1440, 50, np.array([]),
                        minLineLength=5, maxLineGap=7)
line_img = np.zeros((img_canny.shape[0], img_canny.shape[1], 3),
                    dtype=np.uint8)

for line in lines:
        for x1,y1,x2,y2 in line:
            cv2.line(line_img, (x1, y1), (x2, y2), [255, 0, 0], 3)

img_copy  = cv2.addWeighted(edge_detection, 0.8, line_img, 1., 0.)

plt.imshow(img_copy, cmap='gray')
```

Out[15]: <matplotlib.image.AxesImage at 0x16be82b9ca0>



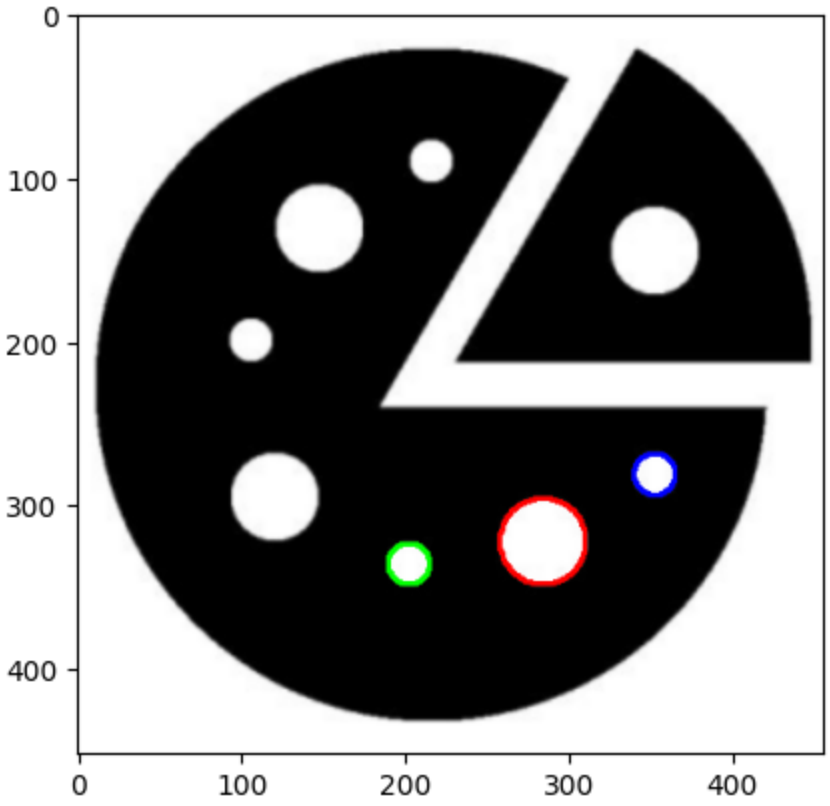
In [16]:
```
#corner detection: harris algorithm
img_chess = cv2.imread('img/chessboard.png')
img_chess = cv2.cvtColor(img_chess, cv2.COLOR_BGR2RGB)
gray = cv2.cvtColor(img_chess, cv2.COLOR_RGB2GRAY)
gray = np.float32(gray)
dst = cv2.cornerHarris(src=gray, blockSize=2, ksize=3, k=0.04)
dst = cv2.dilate(dst, None)

threshold = 0.01 * np.max(dst)

for y in range(dst.shape[0]):
    for x in range(dst.shape[1]):
        if dst[y, x] > threshold:
            cv2.circle(img_chess, (x, y), radius=4, color=(255, 0, 0), thickness=2)

plt.imshow(img_chess)
```

Out[16]: <matplotlib.image.AxesImage at 0x16be8646d00>



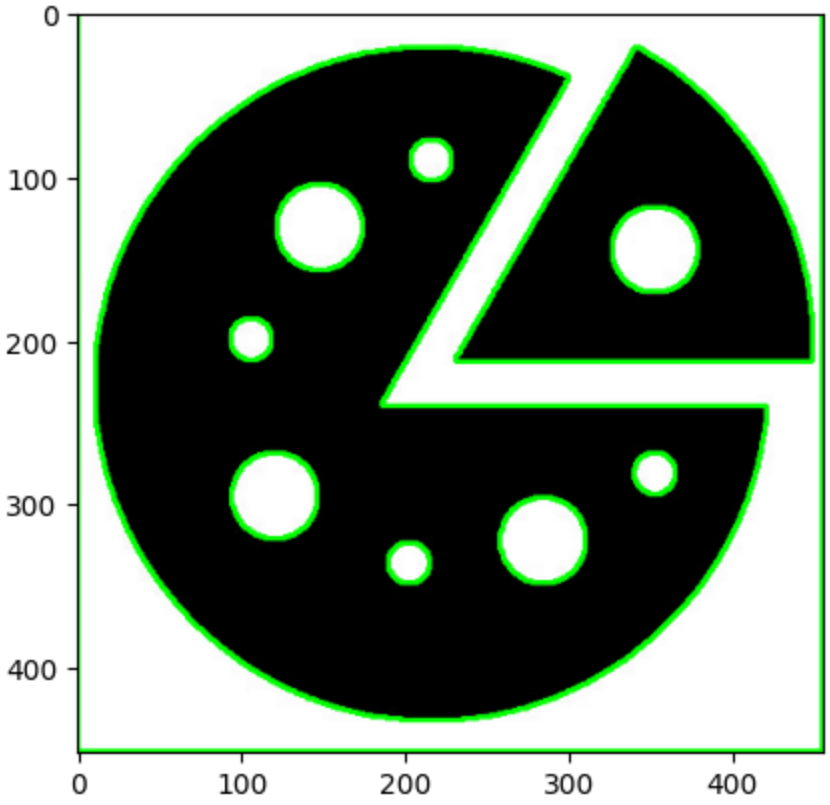
In [17]:
```
# Contour detection
img_con = cv2.imread('img/pie.png')
gray = cv2.cvtColor(img_con, cv2.COLOR_BGR2GRAY)
ret, thresh = cv2.threshold(gray, 127, 255, 0)
contours, _ = cv2.findContours(thresh, cv2.RETR_TREE,
        cv2.CHAIN_APPROX_SIMPLE)

img_copy = img_con.copy()
cv2.polylines(img_copy, [contours[4]], True, (0, 255, 0), 2)
cv2.polylines(img_copy, [contours[5]], True, (255, 0, 0), 2)
cv2.polylines(img_copy, [contours[6]], True, (0, 0, 255), 2)
plt.imshow(img_copy)
```

Out[17]:
```
<matplotlib.image.AxesImage at 0x16be9e9efa0>
```



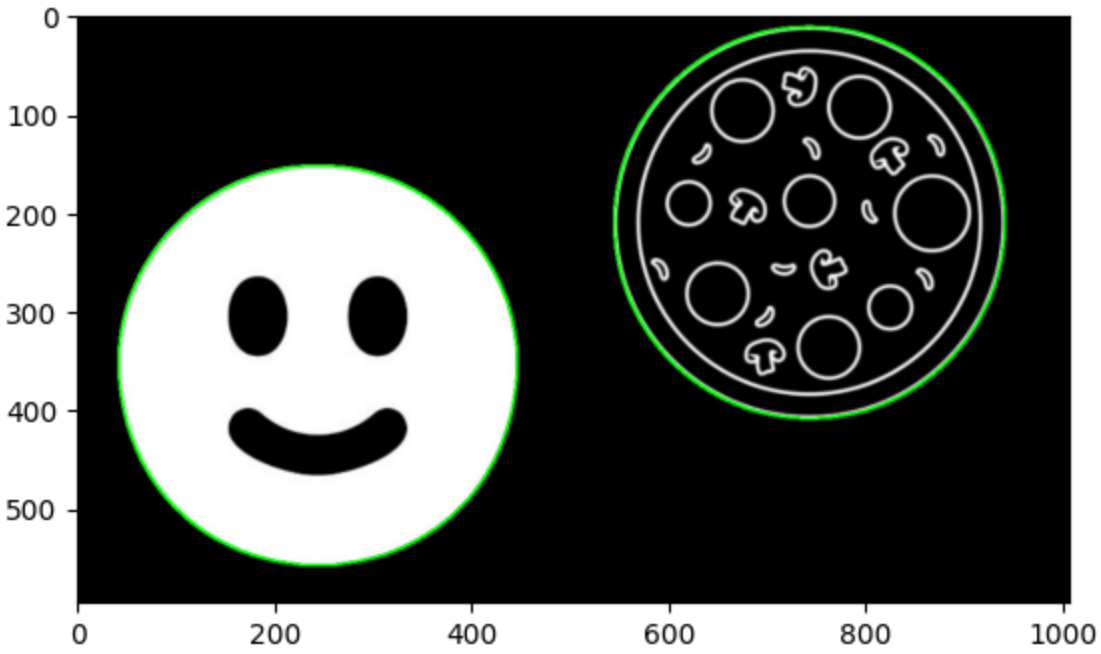
In [18]:
```
cv2.drawContours(img_copy, contours, -1, (0, 255, 0), 2)
plt.imshow(img_copy)
```

Out[18]: <matplotlib.image.AxesImage at 0x16be8d42f40>



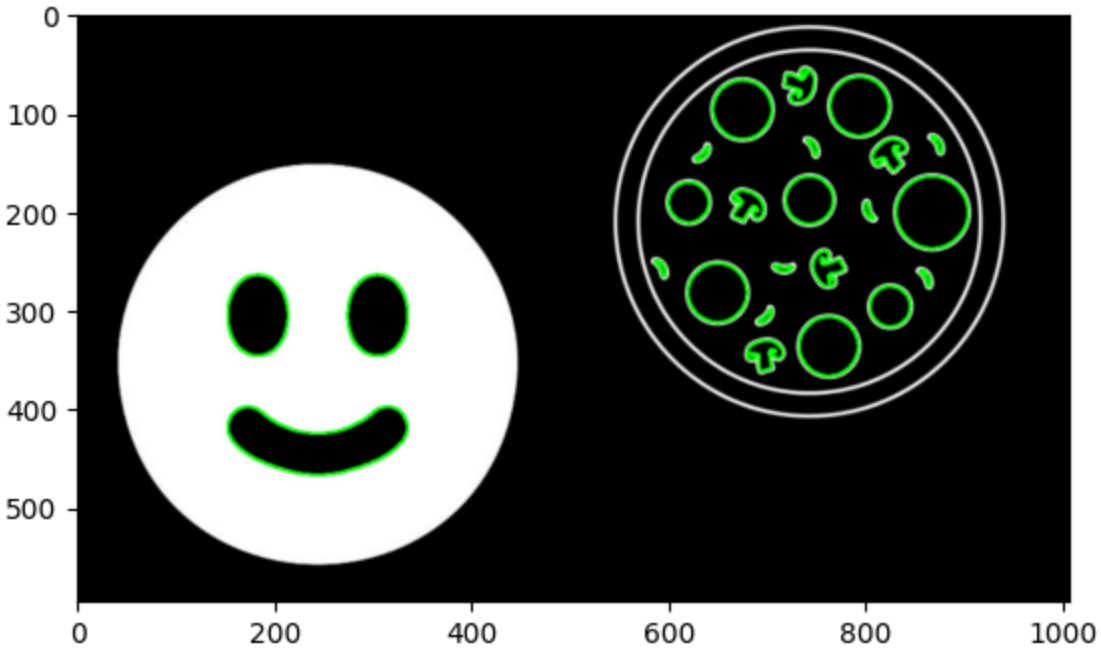
In [19]:
```
img_con = cv2.imread('img/face.png')
gray = cv2.cvtColor(img_con, cv2.COLOR_BGR2GRAY)
ret, thresh = cv2.threshold(gray, 127, 255, 0)
contours, hierarchy = cv2.findContours(thresh, cv2.RETR_TREE,
        cv2.CHAIN_APPROX_SIMPLE)
img_copy = img_con.copy()

for i in range(len(contours)):
    if hierarchy[0][i][3] == -1:
        cv2.drawContours(img_copy, contours, i, (0, 255, 0), 2)

plt.imshow(img_copy)
```

Out[19]: <matplotlib.image.AxesImage at 0x16be8c38760>



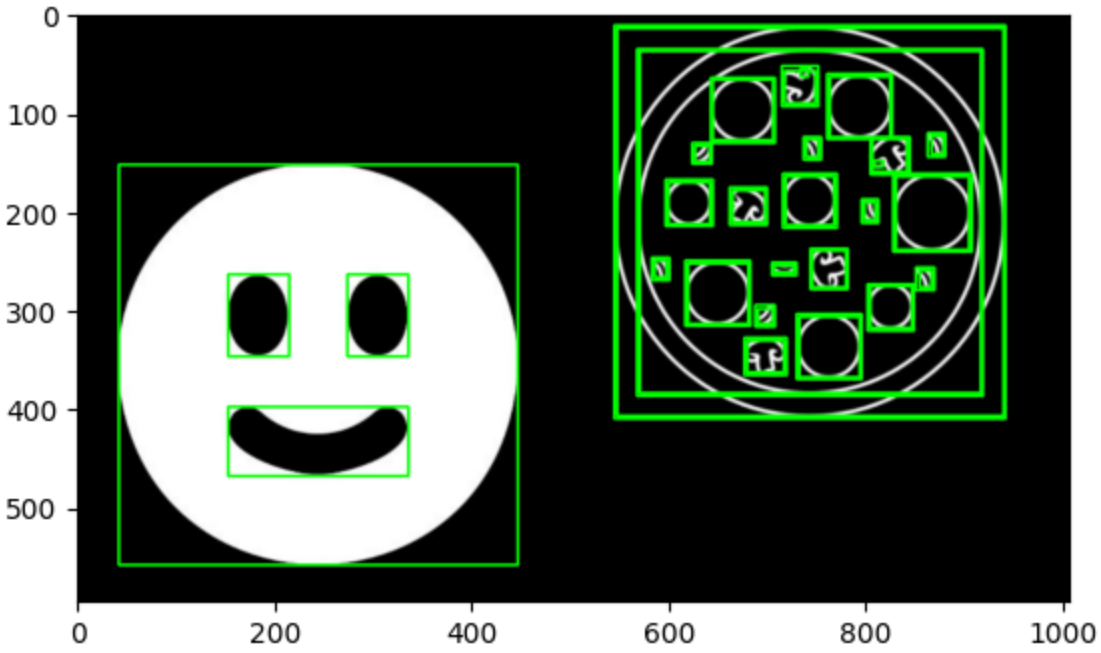
In [20]:
```
img_copy = img_con.copy()
for i in range(len(contours)):
    if hierarchy[0][i][2] == -1:
        cv2.drawContours(img_copy, contours, i, (0, 255, 0), 2)

plt.imshow(img_copy)
```

Out[20]: <matplotlib.image.AxesImage at 0x16be8c7ff10>



In [21]:
```
img_copy = img_con.copy()
for contour in contours:
    x, y, w, h = cv2.boundingRect(contour)
    cv2.rectangle(img_copy, (x, y), (x + w, y + h), (0, 255, 0), 2)

plt.imshow(img_copy)
```

Out[21]: <matplotlib.image.AxesImage at 0x16be8d268e0>



In [223…
```
image_path = 'img/plate2.jpg'
image = cv2.imread(image_path)
original_image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
plt.imshow(original_image)
```

Out[223… <matplotlib.image.AxesImage at 0x16bfd708760>

In [224…

```
gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
blurred = cv2.GaussianBlur(gray, (5, 5), 0)
edges = cv2.Canny(blurred, 30, 200)

contours, hierarchy = cv2.findContours(edges.copy(), cv2.RETR_TREE,
                cv2.CHAIN_APPROX_SIMPLE)


contours = sorted(contours, key = cv2.contourArea, reverse = True) [:30]
screenCnt = None


for contour in contours:
        perimeter = cv2.arcLength(contour, True)
        approx = cv2.approxPolyDP(contour, 0.018 * perimeter, True)
        if len(approx) == 4:
                screenCnt = approx

img_copy = original_image.copy()
cv2.drawContours(img_copy, [screenCnt], -1, (0, 255, 0), 2)
plt.imshow(img_copy)
```

Out[224… <matplotlib.image.AxesImage at 0x16bfd706fa0>

In [225…
```
x, y, w, h = cv2.boundingRect(screenCnt)
pts1 = np.float32([[x+10, y+20], [x+w, y+10], [x+w, y+h-10], [x, y+h]])
pts2 = np.float32([[0, 0], [w, 0], [w, h], [0, h]])
matrix = cv2.getPerspectiveTransform(pts1, pts2)
plate_img = cv2.warpPerspective(original_image, matrix, (w, h))
plt.imshow(plate_img)
```

Out[225… `<matplotlib.image.AxesImage at 0x16bfd4aa370>`

In [226…
```
gray_img = cv2.cvtColor(plate_img, cv2.COLOR_RGB2GRAY)
ret, thresh = cv2.threshold(gray_img, 150, 255, cv2.THRESH_BINARY)

contours2, hierarchy2 = cv2.findContours(thresh, cv2.RETR_TREE,
                cv2.CHAIN_APPROX_SIMPLE)
character_images = []
for i in range(len(contours2)):
        x2, y2, w2, h2 = cv2.boundingRect(contours2[i])
        if w2 > 20 and w2 < 100:

                character_img = plate_img[y2:y2+h2, x2:x2+w2]
```

```
                character_img_resized = cv2.resize(character_img, (w2, 50))
                character_images.append(character_img_resized)
                cv2.rectangle(plate_img, (x2, y2), (x2 + w2, y2 + h2),
                              (0, 255, 0), 2)

plt.imshow(plate_img)
```

Out[226… `<matplotlib.image.AxesImage at 0x16bfd47dd30>`

In [227…

```
for i in range(len(character_images)):

        plt.figsize=(50, 50)
        plt.subplot(1, len(character_images), i + 1)
        plt.imshow(character_images[i], cmap='gray')
```