PURDUE UNIVERSITY

Elmore Family School of Electrical and Computer Engineering

Computer Vision

Homework 6

Adithya Sineesh

Email : asineesh@purdue.edu

Submitted: October 19, 2022

1 Theory Questions

Lecture 15 presented two very famous algorithms for image segmentation: The Otsu Algorithm and the Watershed Algorithm. These algorithms are as different as night and day. Present in your own words the strengths and the weaknesses of each. (Note that the Watershed algorithm uses the morphological operators that we discussed in Lecture 14.)

The strengths of Otsu's algorithm are:

1. Its fast
2. It is invariant to changes in brightness or contrast of the image

The weaknesses of Otsu's algorithm are:

1. It is susceptible to noise
2. It works well only if there is significant separation between the peaks of the histogram

The strengths of watershed algorithm are:

1. Its fast
2. It is intuitive

The weaknesses of watershed algorithm are:

1. It could lead to under-segmentation i.e. unwanted regions are present in the foreground segmentation
2. It can also lead to over segmentation i.e. the certain regions of interest are not present in the foreground segmentation.

2 Theory

In this homework, image segmentation and contour extraction algorithms will be implemented. Let us first look at the concepts behind them.

2.1 Otsu's Algorithm using RGB values

This algorithm divides all the pixels of the image into two classes, namely the foreground and the background. The following are its steps:

1. First a single channel of the image is selected. Then for pixel values ranging from 0 to 255, its histogram is calculated. The probability of a pixel being at a level i is given as:

$$p_i = \frac{n_i}{N}$$

   where N is the total number of pixels in the image and n_i is the number of pixels at the level i

2. Then we select a pixel level k such that all the pixels having a value less than k belong to the background (probability of which is given as w_0) and all the pixels having a value greater than k belong to the foreground (probability of which is given as w_1).

3. Then we calculate the mean of both the classes (μ_0 and μ_1) as follows

$$\mu_0 = \frac{\sum_{i=0}^{k} ip_i}{w_0}$$
$$\mu_1 = \frac{\sum_{i=k+1}^{N} ip_i}{w_1}$$

4. Now we calculate the between class scatter using the above two values.

$$\sigma_B^2 = w_0 w_1 (\mu_1 - \mu_0)^2$$

5. Finally, we select the threshold k which has the maximum σ_B^2 value. Then we threshold the image using that values such that all the pixels having a value less than k are set to 0 and all the pixels having a value greater than k are set to 1. This mask is then applied to the original image to return just its foreground.

6. The above steps can be repeated several times to obtain better segmentation. This number of iterations is a hyperparameter.

7. The final masks obtained for each of the RGB channels are combined using a bitwise and operation to get the overall segmentation mask.

2.2 Otsu's Algorithm using texture based segmentation

The following are the steps in the approach:

1. First the image is converted to grayscale.

2. Three windows of different sizes are constructed. We place these windows at each pixel and calculate the variance of all the pixels in the window.

3. Three masks are therefore obtained which correspond to the 3 channels that you would find in a RGB image. We then follow the steps in the previous section.

2.3 Contour extraction

Using the binary mask, we can obtain the contour for the image. If a pixel having a value of 1 has at least a pixel of value 0 in its 8 neighbourhood, then it is part of the contour. Else it isn't.

3 Experiment

3.1 Task 1

The 2 images given to us are:

Figure 1: First Image

Figure 2: Second Image

3.2 Task 2

The 2 images that are used for the second task:

Figure 3: First Image

Figure 4: Second Image

4 Results for Task 1

4.1 Using Otsu algorithm on RGB image

4.1.1 For Image 1

Figure 5: Mask from 1st color channel, 0 iterations

Figure 6: Mask from 2nd color channel, 1 iteration

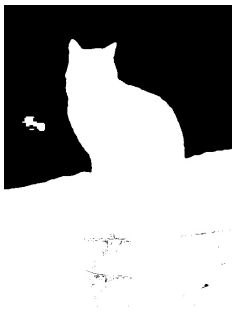


Figure 7: Mask from 3rd color channel, 4 iterations

Figure 8: Overall mask by combining the 3 masks using "AND" operator

Figure 9: Foreground of the image

4.1.2 For Image 2

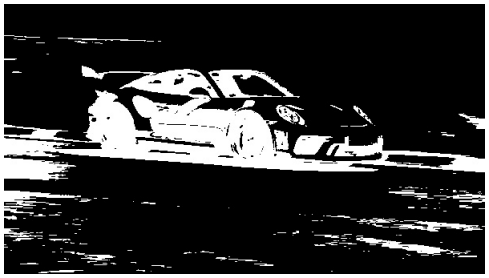


Figure 10: Mask from 1st color channel, 8 iterations

Figure 11: Mask from 2nd color channel, 8 iteration

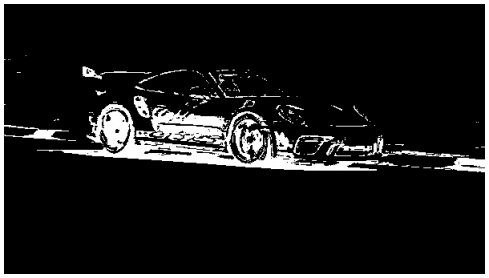


Figure 12: Mask from 3rd color channel, 1 iteration

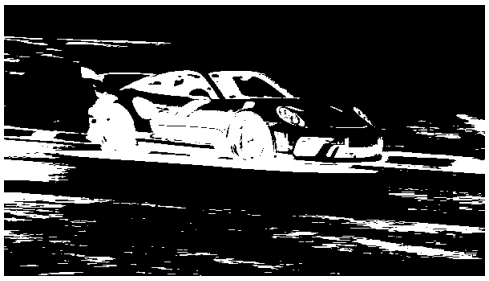


Figure 13: Overall mask by combining the 3 masks using "OR" operator

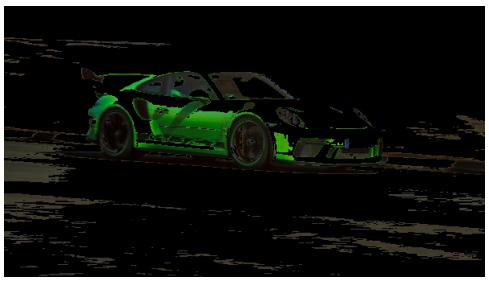


Figure 14: Foreground of the image

4.2 Using Otsu algorithm on texture

4.2.1 For Image 1

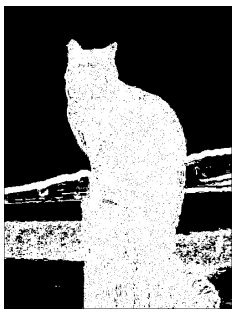


Figure 15: Mask from 1st channel where window size is 3, 1 iterations

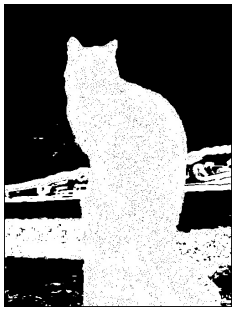


Figure 16: Mask from 2nd channel where window size is 5, 1 iteration

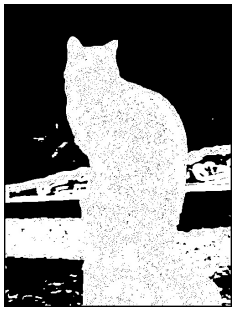


Figure 17: Mask from channel where window size is 7, 1 iteration

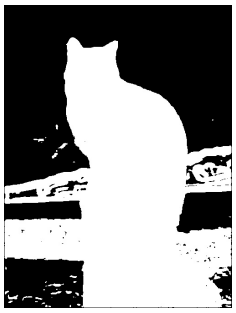


Figure 18: Overall mask by combining the 3 masks using "OR" operator

Figure 19: Foreground of the image

4.2.2 For Image 2

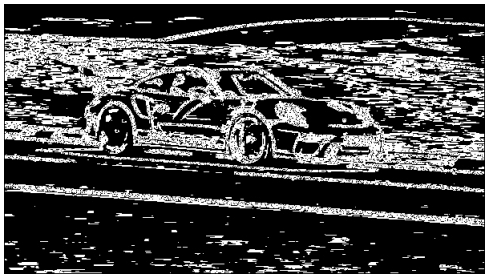


Figure 20: Mask from 1st channel where window size is 3, 2 iterations

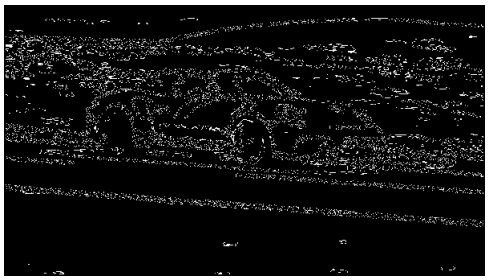


Figure 21: Mask from 2nd channel where window size is 5, 2 iterations

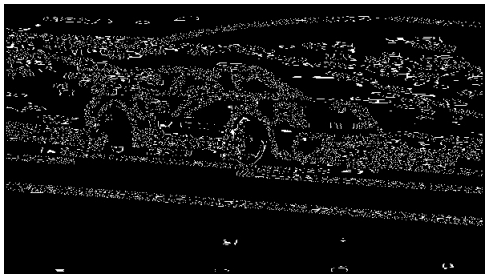


Figure 22: Mask from 3rd channel where window size is 7, 2 iterations

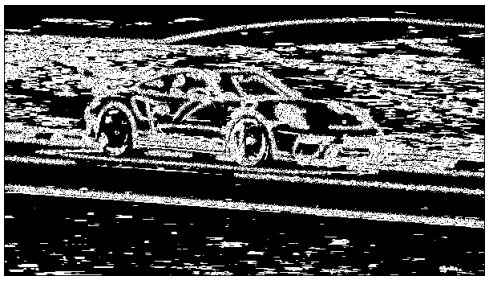


Figure 23: Overall mask by combining the 3 masks using "OR" operator

Figure 24: Foreground of the image

4.3 Obtaining Contour

4.3.1 For Image 1

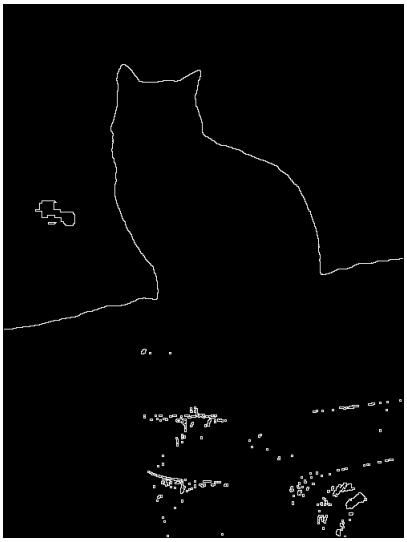


Figure 25: Using Otsu on RGB

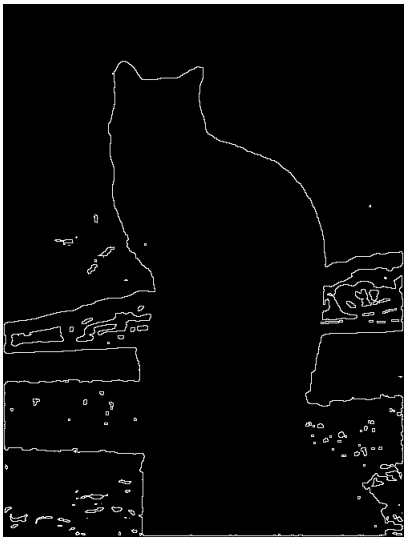


Figure 26: Using Otsu on texture

4.3.2 For Image 2

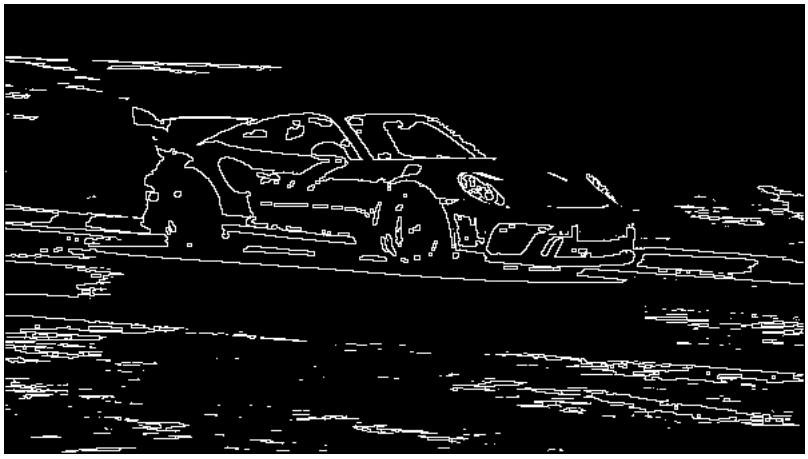


Figure 27: Using Otsu on RGB

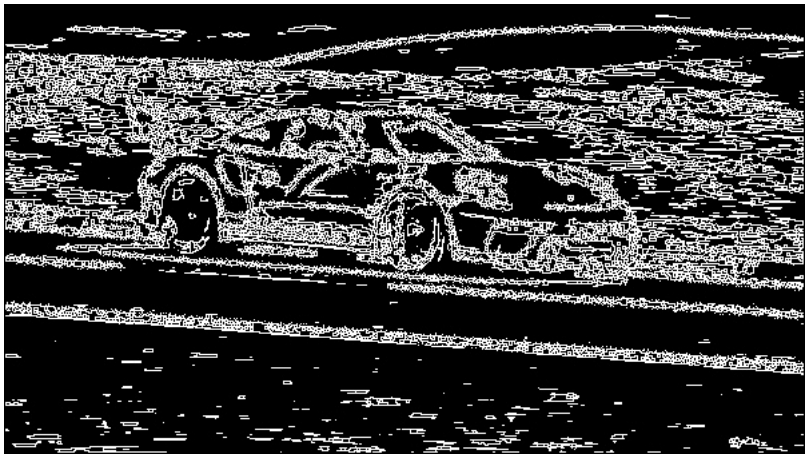


Figure 28: Using Otsu on texture

5 Results for Task 2

5.1 Using Otsu algorithm on RGB image

5.1.1 For Image 1

Figure 29: Mask from 1st color channel, 1 iteration

Figure 30: Mask from 2nd color channel, 1 iteration

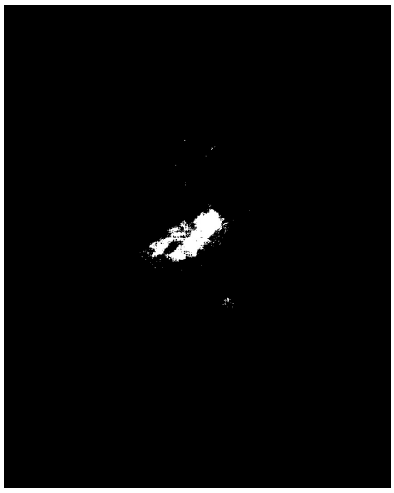


Figure 31: Mask from 3rd color channel, 2 iterations

Figure 32: Overall mask by combining the 3 masks using "OR" operator

Figure 33: Foreground of the image

5.1.2 For Image 2

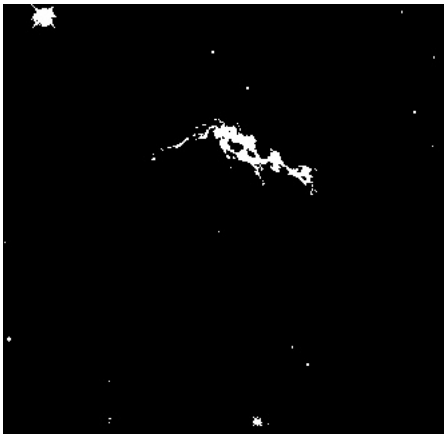


Figure 34: Mask from 1st color channel, 2 iterations

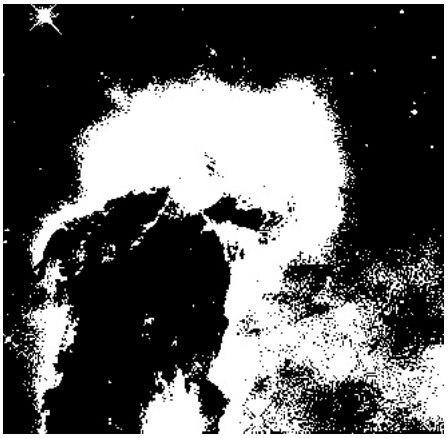


Figure 35: Mask from 2nd color channel, 1 iteration

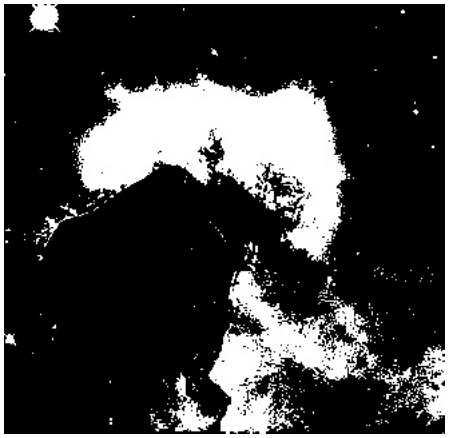


Figure 36: Mask from 3rd color channel, 2 iteration

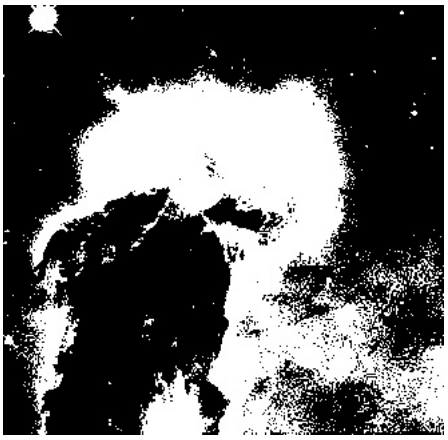


Figure 37: Overall mask by combining the 3 masks using "OR" operator

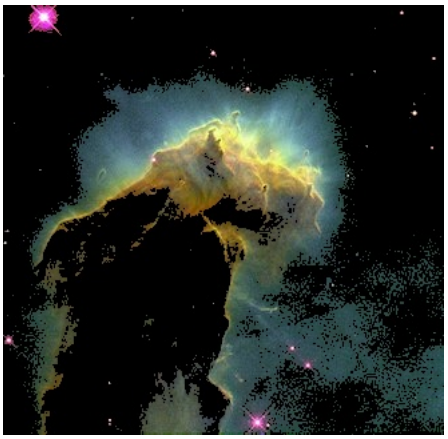


Figure 38: Foreground of the image

5.2 Using Otsu algorithm on texture

5.2.1 For Image 1

Figure 39: Mask from 1st channel where window size is 3, 1 iteration

Figure 40: Mask from 2nd channel where window size is 5, 1 iteration

Figure 41: Mask from channel where window size is 7, 1 iteration

Figure 42: Overall mask by combining the 3 masks using "OR" operator

Figure 43: Foreground of the image

5.2.2 For Image 2

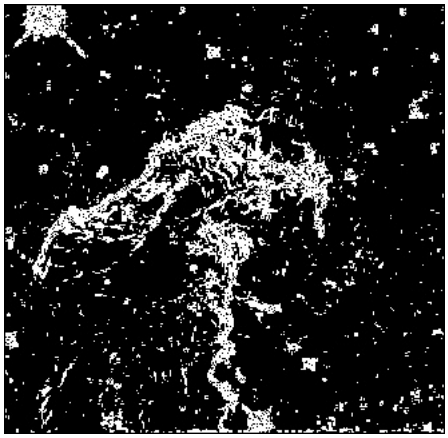


Figure 44: Mask from 1st channel where window size is 3, 1 iteration

Figure 45: Mask from 2nd channel where window size is 5, 1 iteration

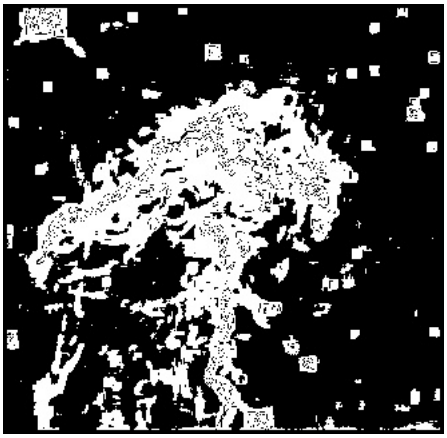


Figure 46: Mask from 3rd channel where window size is 7, 1 iteration

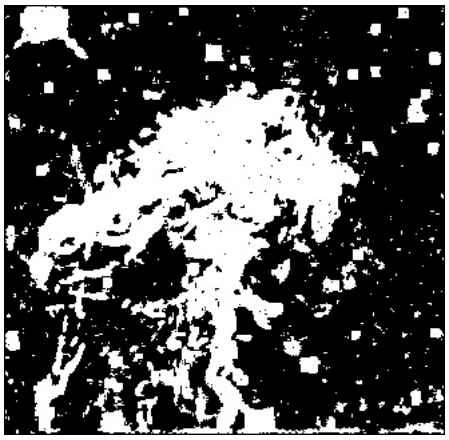


Figure 47: Overall mask by combining the 3 masks using "OR" operator

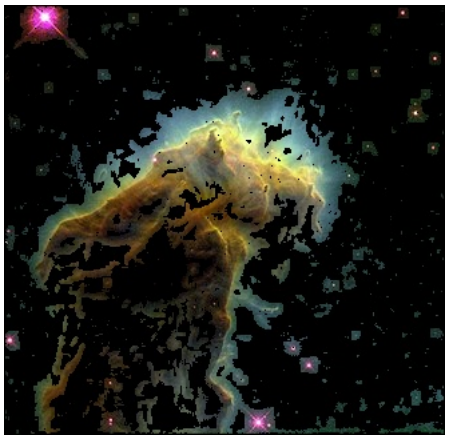


Figure 48: Foreground of the image

5.3 Obtaining Contour

5.3.1 For Image 1

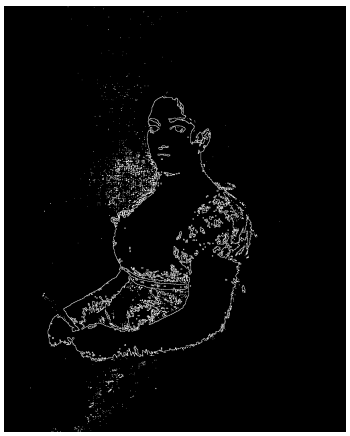


Figure 49: Using Otsu on RGB

Figure 50: Using Otsu on texture

5.3.2 For Image 2

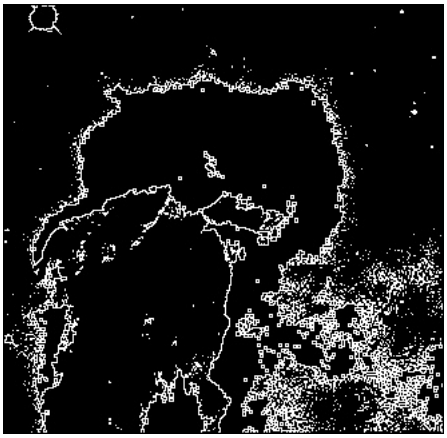


Figure 51: Using Otsu on RGB

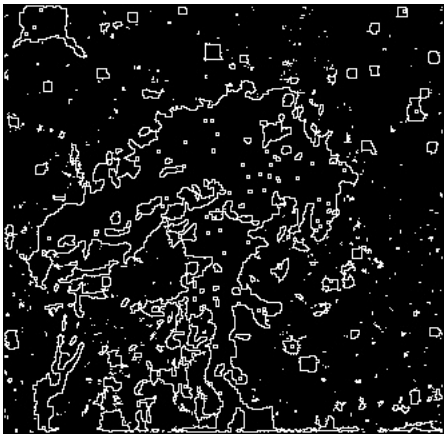


Figure 52: Using Otsu on texture

6 Notes

The following are the things that I have observed:

1. In the Fisher Discriminant, I only focused on maximizing the numerator σ_B^2 and ignored the denominator σ_W^2

2. To combine the masks obtained from the three channels, I also used the bitwise "OR" operator. In some cases, it gave better results than the bitwise "AND" operator.

3. Overall both the methods for Otsu did not give the best results. In my opinion the method using the RGB channels performed slightly better.

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
import cv2
```

In [2]:

```
#applying the segmentation mask on the image
def applying_mask(img,mask):
    masked = np.zeros(img.shape,dtype='uint8')
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            if mask[i,j]==1:
                masked[i,j,0] = img[i,j,0]
                masked[i,j,1] = img[i,j,1]
                masked[i,j,2] = img[i,j,2]
    return masked
```

In [3]:

```
#combining the masks obtained from the three channels
def combining_masks(masks,combine_type):
    maskr = masks[0]
    maskg = masks[1]
    maskb = masks[2]

    mask_f = np.zeros(maskr.shape)
    for i in range(mask_f.shape[0]):
        for j in range(mask_f.shape[1]):
            if maskr[i,j]==255.0 and maskg[i,j]==255.0 and maskb[i,j]==255.0 and combine_ty
                mask_f[i,j] = 1
            if combine_type == 1:
                if (maskr[i,j]==255.0 or maskg[i,j]==255.0 or maskb[i,j]==255.0) :
                    mask_f[i,j] = 1

    return mask_f
```

In [4]:

```
#Returns the texture of the image with 3 channels corresponding to 3 window sizes
def get_texture(image,windows):
    img = cv2.cvtColor(image,cv2.COLOR_RGB2GRAY)
    texture = np.zeros((img.shape[0],img.shape[1],3))

    for k in range(3):
        w = windows[k]
        for i in range(w,img.shape[0]-w):
            for j in range(w,img.shape[1]-w):
                texture[i,j,k] = np.var(img[i-w:i+w+1,j-w:j+w+1])

    texture = texture.astype('uint8')

    return texture
```

In [5]:

```
#Implementation of Otsu algorithm
def Otsu(image,channel,parameter,inv):

    img = image[:,:,channel]
    mask = 255 * np.ones(img.shape, 'uint8')

    for k in range (parameter[channel]):
        histogram, bin_edge = np.histogram(img[np.nonzero(mask)],np.arange(256))

        Ds = []

        for i in range(len(histogram)):
            h0,h1 = np.split(histogram,[i])
            w0 = np.sum(h0)/np.sum(histogram)
            w1 = np.sum(h1)/np.sum(histogram)

            mean0 = np.sum([a*b for a,b in enumerate(h0)])/w0
            mean1 = np.sum([a*b for a,b in enumerate(h1)])/w1

            var0 = np.sum([((a-mean0)**2)*b for a,b in enumerate(h0)])/w0
            var1 = np.sum([((a-mean1)**2)*b for a,b in enumerate(h1)])/w1

            between = w0*w1*(mean1-mean0)**2
            within = w0*var0 + w1*var1

            D = between/1
            D = np.nan_to_num(D)
            Ds.append(D)

        Dsn = np.array(Ds)
        threshold = Dsn.argmax()
        if (inv==0):
            _,mask = cv2.threshold(img,threshold,255,cv2.THRESH_BINARY_INV)

        if (inv==1):
            _,mask = cv2.threshold(img,threshold,255,cv2.THRESH_BINARY)

    return mask
```

In [6]:

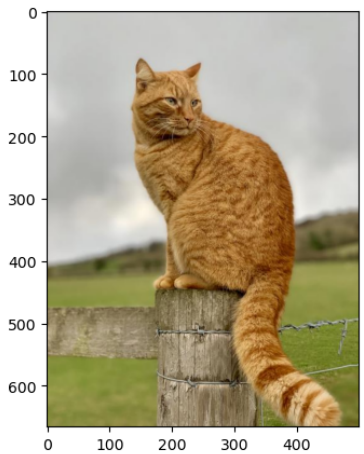
```
#Wrapper function for Otsu
def combine_Otsu(img,parameter,inv):
    mask_final = np.ones((img.shape[0],img.shape[1]))
    masks = []
    for i in range(3):
        maski = Otsu(img,i,parameter,inv)
        masks.append(maski)


    return masks
```

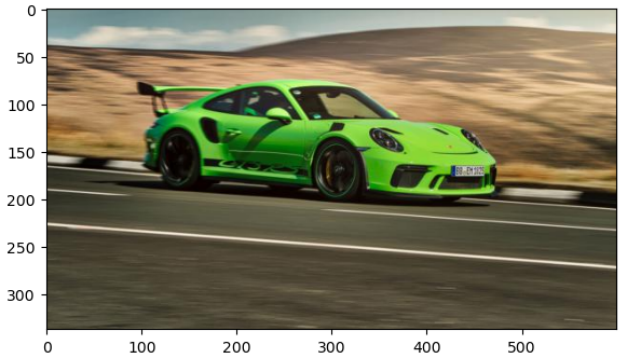
In [7]:

```
#Uses the mask to generate the contour
def get_contour(mask):
    contour = np.zeros(mask.shape)
    for i in range(1,mask.shape[0]-1):
        for j in range(1,mask.shape[1]-1):
            if mask[i,j] == 0:
                continue
            if np.min(mask[i-1:i+2,j-1:j+2])==0:
                contour[i,j] = 1.0


    return contour.astype('uint8')
```

In [8]:

```
#Displaying all the images
image1 = cv2.imread("cat.jpg")
image1 = cv2.cvtColor(image1,cv2.COLOR_BGR2RGB)

image2 = cv2.imread("car.jpg")
image2 = cv2.cvtColor(image2,cv2.COLOR_BGR2RGB)

fig = plt.figure(figsize=(15,5))
ax1 = fig.add_subplot(121)
ax2 = fig.add_subplot(122)

ax1.imshow(image1)
ax2.imshow(image2)

#Defining the number of iterations for 1st Otsu method for each of the RGB channels
image1_channel = [0,1,4]
image2_channel = [8,8,1]
```





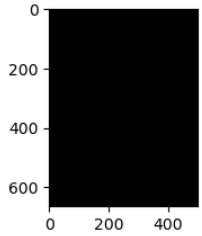
In [9]:

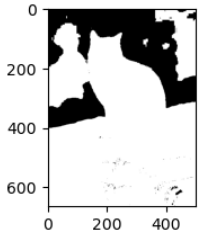
```
#For image 1, using the 1st Otsu method
mask_ind_1 = combine_Otsu(image1,image1_channel,0)

mask_1 = combining_masks(mask_ind_1,0) #Using and operator to combine the masks

iou1 = applying_mask(image1,mask_1)

fig = plt.figure(figsize=(15,5))
ax1 = fig.add_subplot(231)
ax2 = fig.add_subplot(232)
ax3 = fig.add_subplot(233)
ax4 = fig.add_subplot(235)
ax5 = fig.add_subplot(236)

ax1.imshow(mask_ind_1[0],cmap='gray')
ax2.imshow(mask_ind_1[1],cmap='gray')
ax3.imshow(mask_ind_1[2],cmap='gray')
ax4.imshow(mask_1,cmap='gray')
ax5.imshow(iou1,cmap='gray')

cv2.imwrite("mask11.jpg",mask_ind_1[0])
cv2.imwrite("mask12.jpg",mask_ind_1[1])
cv2.imwrite("mask13.jpg",mask_ind_1[2])
cv2.imwrite("mask14.jpg",mask_1*255.0)

iou1 = cv2.cvtColor(iou1, cv2.COLOR_RGB2BGR)
cv2.imwrite("mask15.jpg",iou1)
```

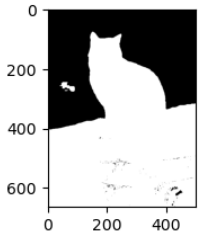
```
C:\Users\adisi\AppData\Local\Temp\ipykernel_3020\3543035357.py:16: RuntimeWa
rning: invalid value encountered in double_scalars
  mean0 = np.sum([a*b for a,b in enumerate(h0)])/w0
C:\Users\adisi\AppData\Local\Temp\ipykernel_3020\3543035357.py:19: RuntimeWa
rning: invalid value encountered in double_scalars
  var0 = np.sum([((a-mean0)**2)*b for a,b in enumerate(h0)])/w0
C:\Users\adisi\AppData\Local\Temp\ipykernel_3020\3543035357.py:17: RuntimeWa
rning: invalid value encountered in double_scalars
  mean1 = np.sum([a*b for a,b in enumerate(h1)])/w1
```

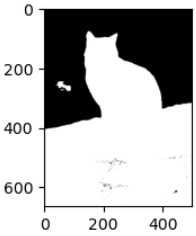
Out[9]:

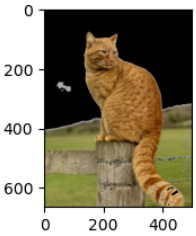
```
True
```


0
200
400
600
0
200
400


0
200
400
600
0
200
400


0
200
400
600
0
200
400


0
200
400
600
0
200
400


0
200
400
600
0
200
400

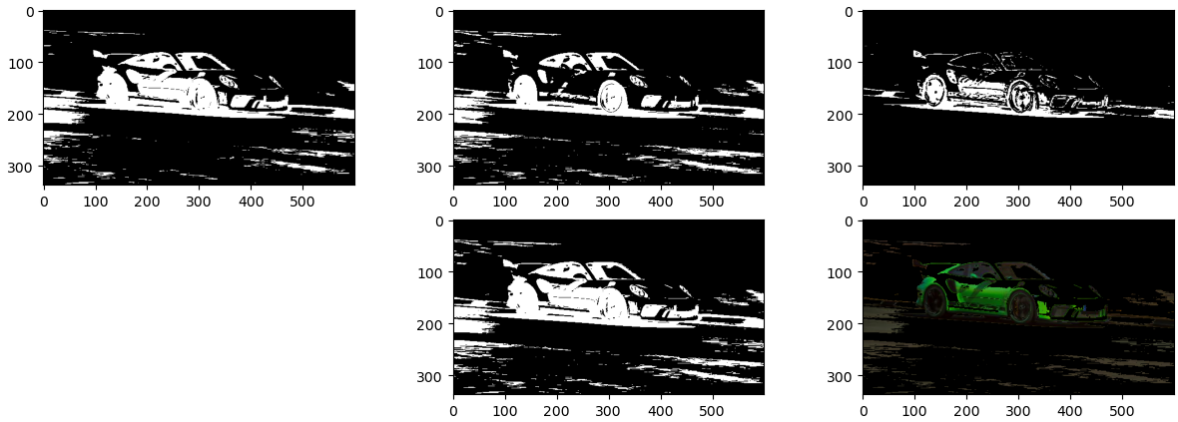
In [10]:

```
#For image 2, using the 1st Otsu method
mask_ind_2 = combine_Otsu(image2,image2_channel,0)
mask_2 = combining_masks(mask_ind_2,1) #Using or operator to combine the masks

iou2 = applying_mask(image2,mask_2)

fig = plt.figure(figsize=(15,5))
ax1 = fig.add_subplot(231)
ax2 = fig.add_subplot(232)
ax3 = fig.add_subplot(233)
ax4 = fig.add_subplot(235)
ax5 = fig.add_subplot(236)

ax1.imshow(mask_ind_2[0],cmap='gray')
ax2.imshow(mask_ind_2[1],cmap='gray')
ax3.imshow(mask_ind_2[2],cmap='gray')
ax4.imshow(mask_2,cmap='gray')
ax5.imshow(iou2,cmap='gray')

cv2.imwrite("mask21.jpg",mask_ind_2[0])
cv2.imwrite("mask22.jpg",mask_ind_2[1])
cv2.imwrite("mask23.jpg",mask_ind_2[2])
cv2.imwrite("mask24.jpg",mask_2*255.0)

iou2 = cv2.cvtColor(iou2, cv2.COLOR_RGB2BGR)
cv2.imwrite("mask25.jpg",iou2)
```

```
C:\Users\adisi\AppData\Local\Temp\ipykernel_3020\3543035357.py:16: RuntimeWa
rning: invalid value encountered in double_scalars
  mean0 = np.sum([a*b for a,b in enumerate(h0)])/w0
C:\Users\adisi\AppData\Local\Temp\ipykernel_3020\3543035357.py:19: RuntimeWa
rning: invalid value encountered in double_scalars
  var0 = np.sum([((a-mean0)**2)*b for a,b in enumerate(h0)])/w0
C:\Users\adisi\AppData\Local\Temp\ipykernel_3020\3543035357.py:17: RuntimeWa
rning: invalid value encountered in double_scalars
  mean1 = np.sum([a*b for a,b in enumerate(h1)])/w1
```

Out[10]:

```
True
```



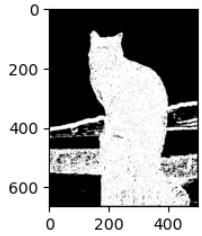
In [11]:

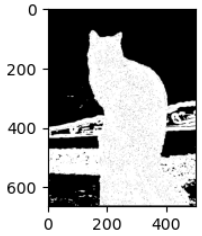
```
#For image 1 using the 2nd Otsu method
nimage1_channel = [1,1,1]

textures1 = get_texture(image1, [1,2,3])

inmask_ind_1 = combine_Otsu(textures1,nimage1_channel,1)

inmask_1 = combining_masks(inmask_ind_1,1) #Using or operator to combine the masks
niou1 = applying_mask(image1,inmask_1)

fig = plt.figure(figsize=(15,5))
ax1 = fig.add_subplot(231)
ax2 = fig.add_subplot(232)
ax3 = fig.add_subplot(233)
ax4 = fig.add_subplot(235)
ax5 = fig.add_subplot(236)

ax1.imshow(inmask_ind_1[0],cmap='gray')
ax2.imshow(inmask_ind_1[1],cmap='gray')
ax3.imshow(inmask_ind_1[2],cmap='gray')
ax4.imshow(inmask_1,cmap='gray')
ax5.imshow(niou1,cmap='gray')

cv2.imwrite("nmask11.jpg",inmask_ind_1[0])
cv2.imwrite("nmask12.jpg",inmask_ind_1[1])
cv2.imwrite("nmask13.jpg",inmask_ind_1[2])
cv2.imwrite("nmask14.jpg",inmask_1*255.0)

niou1 = cv2.cvtColor(niou1, cv2.COLOR_RGB2BGR)
cv2.imwrite("nmask15.jpg",niou1)
```

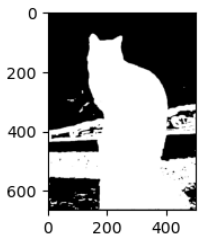
```
C:\Users\adisi\AppData\Local\Temp\ipykernel_3020\3543035357.py:16: RuntimeWa
rning: invalid value encountered in double_scalars
  mean0 = np.sum([a*b for a,b in enumerate(h0)])/w0
C:\Users\adisi\AppData\Local\Temp\ipykernel_3020\3543035357.py:19: RuntimeWa
rning: invalid value encountered in double_scalars
  var0 = np.sum([((a-mean0)**2)*b for a,b in enumerate(h0)])/w0
```

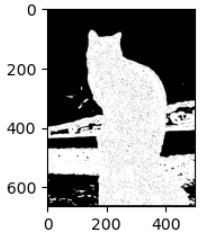
Out[11]:

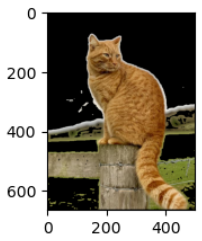
```
True
```


0
200
400
600
0
200
400


0
200
400
600
0
200
400


0
200
400
600
0
200
400


0
200
400
600
0
200
400


0
200
400
600
0
200
400

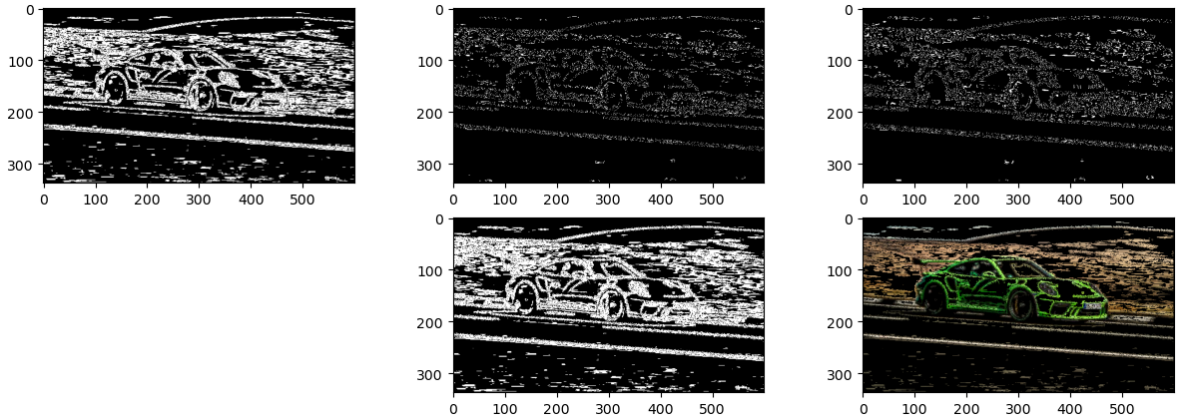
In [12]:

```
#For image 2 using the 2nd Otsu method
nimage2_channel = [2,2,2]
textures2 = get_texture(image2,[1,2,3])

inmask_ind_2 = combine_Otsu(textures2,nimage2_channel,1)

inmask_2 = combining_masks(inmask_ind_2,1) #Using and operator to combine the masks
niou2 = applying_mask(image2,inmask_2)

fig = plt.figure(figsize=(15,5))
ax1 = fig.add_subplot(231)
ax2 = fig.add_subplot(232)
ax3 = fig.add_subplot(233)
ax4 = fig.add_subplot(235)
ax5 = fig.add_subplot(236)

ax1.imshow(inmask_ind_2[0],cmap='gray')
ax2.imshow(inmask_ind_2[1],cmap='gray')
ax3.imshow(inmask_ind_2[2],cmap='gray')
ax4.imshow(inmask_2,cmap='gray')
ax5.imshow(niou2,cmap='gray')

cv2.imwrite("nmask21.jpg",inmask_ind_2[0])
cv2.imwrite("nmask22.jpg",inmask_ind_2[1])
cv2.imwrite("nmask23.jpg",inmask_ind_2[2])
cv2.imwrite("nmask24.jpg",inmask_2*255.0)

niou2 = cv2.cvtColor(niou2, cv2.COLOR_RGB2BGR)
cv2.imwrite("nmask25.jpg",niou2)
```

```
C:\Users\adisi\AppData\Local\Temp\ipykernel_3020\3543035357.py:16: RuntimeWa
rning: invalid value encountered in double_scalars
  mean0 = np.sum([a*b for a,b in enumerate(h0)])/w0
C:\Users\adisi\AppData\Local\Temp\ipykernel_3020\3543035357.py:19: RuntimeWa
rning: invalid value encountered in double_scalars
  var0 = np.sum([((a-mean0)**2)*b for a,b in enumerate(h0)])/w0
```

Out[12]:

```
True
```

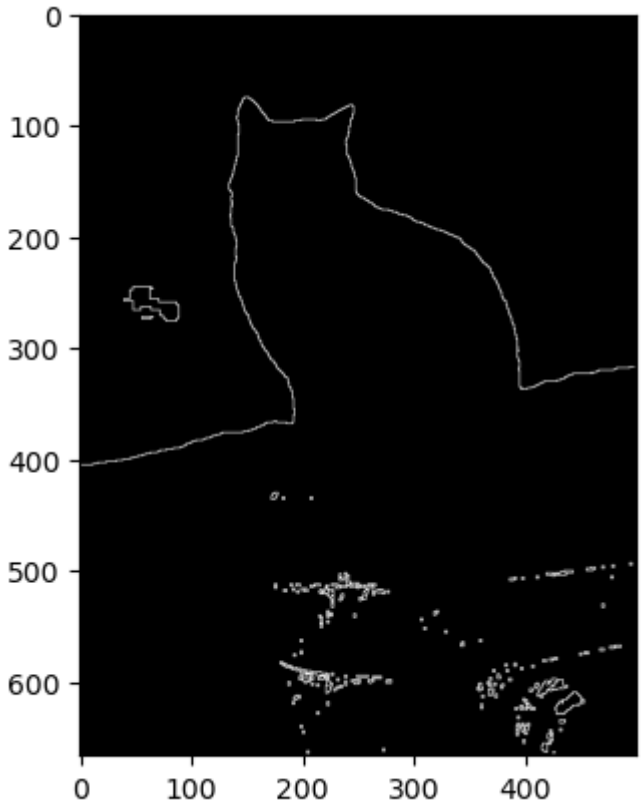


In [13]:

```
#Producing contours for the 1st image
c11 = get_contour(mask_1)
plt.imshow(c11,cmap='gray')
cv2.imwrite("contour11.jpg",c11*255.0)
```

Out[13]:

True

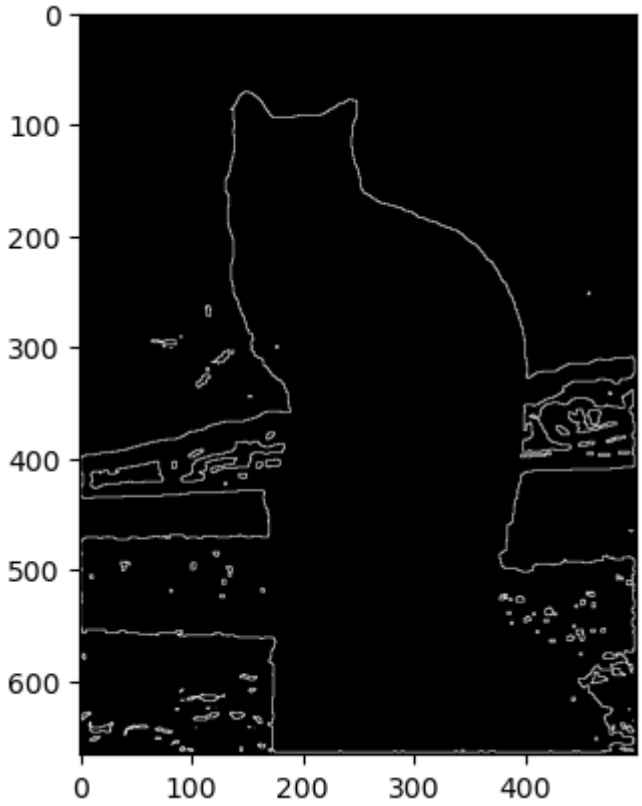


In [14]:

```
#Producing contours for the 1st image using texture
c12 = get_contour(inmask_1)

plt.imshow(c12,cmap='gray')
cv2.imwrite("contour12.jpg",c12*255.0)
```

Out[14]:

```
True
```

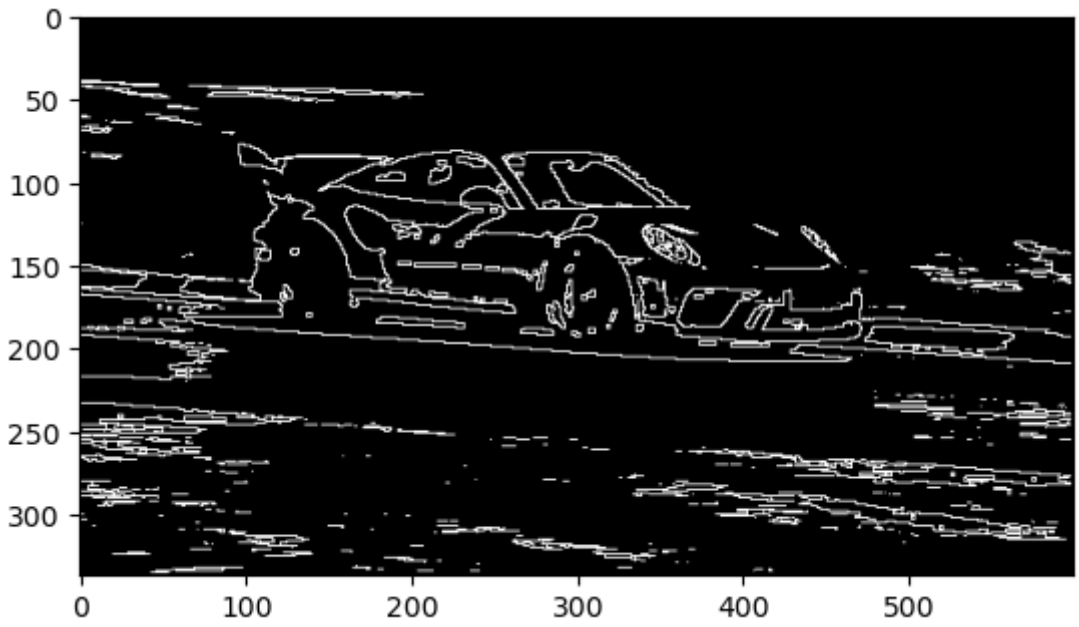


In [15]:

```
#Producing contours for the 2nd image
c21 = get_contour(mask_2)
plt.imshow(c21,cmap='gray')
cv2.imwrite("contour21.jpg",c21*255.0)
```

Out[15]:

True

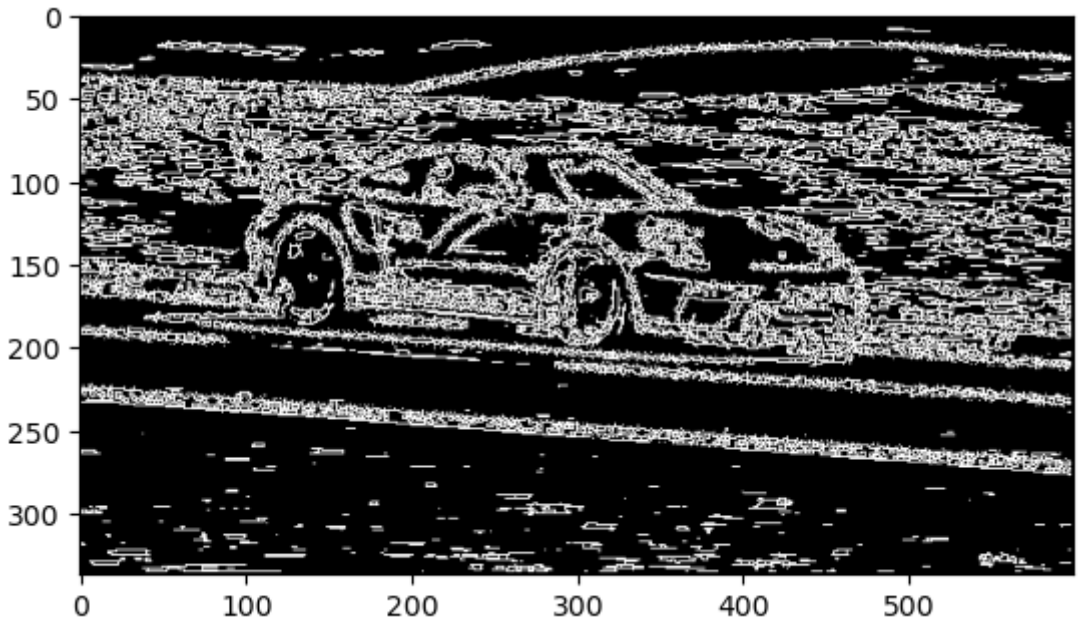


In [16]:

```
#Producing contours for the 2nd image using texture
c22 = get_contour(inmask_2)
plt.imshow(c22,cmap='gray')
cv2.imwrite("contour22.jpg",c22*255.0)
```

Out[16]:

```
True
```



In [ ]:

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
import cv2
```

In [2]:

```
#applying the segmentation mask on the image
def applying_mask(img,mask):
    masked = np.zeros(img.shape,dtype='uint8')
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            if mask[i,j]==1:
                masked[i,j,0] = img[i,j,0]
                masked[i,j,1] = img[i,j,1]
                masked[i,j,2] = img[i,j,2]
    return masked
```

In [3]:

```
#combining the masks obtained from the three channels
def combining_masks(masks,combine_type):
    maskr = masks[0]
    maskg = masks[1]
    maskb = masks[2]

    mask_f = np.zeros(maskr.shape)
    for i in range(mask_f.shape[0]):
        for j in range(mask_f.shape[1]):
            if maskr[i,j]==255.0 and maskg[i,j]==255.0 and maskb[i,j]==255.0 and combine_ty
                mask_f[i,j] = 1
            if combine_type == 1:
                if (maskr[i,j]==255.0 or maskg[i,j]==255.0 or maskb[i,j]==255.0) :
                    mask_f[i,j] = 1

    return mask_f
```

In [4]:

```
#Returns the texture of the image with 3 channels corresponding to 3 window sizes
def get_texture(image,windows):
    img = cv2.cvtColor(image,cv2.COLOR_RGB2GRAY)
    texture = np.zeros((img.shape[0],img.shape[1],3))

    for k in range(3):
        w = windows[k]
        for i in range(w,img.shape[0]-w):
            for j in range(w,img.shape[1]-w):
                texture[i,j,k] = np.var(img[i-w:i+w+1,j-w:j+w+1])

    texture = texture.astype('uint8')

    return texture
```

In [5]:

```
#Implementation of Otsu algorithm
def Otsu(image,channel,parameter,inv):

    img = image[:,:,channel]
    mask = 255 * np.ones(img.shape, 'uint8')

    for k in range (parameter[channel]):
        histogram, bin_edge = np.histogram(img[np.nonzero(mask)],np.arange(256))

        Ds = []

        for i in range(len(histogram)):
            h0,h1 = np.split(histogram,[i])
            w0 = np.sum(h0)/np.sum(histogram)
            w1 = np.sum(h1)/np.sum(histogram)

            mean0 = np.sum([a*b for a,b in enumerate(h0)])/w0
            mean1 = np.sum([a*b for a,b in enumerate(h1)])/w1

            var0 = np.sum([((a-mean0)**2)*b for a,b in enumerate(h0)])/w0
            var1 = np.sum([((a-mean1)**2)*b for a,b in enumerate(h1)])/w1

            between = w0*w1*(mean1-mean0)**2
            within = w0*var0 + w1*var1

            D = between/1
            D = np.nan_to_num(D)
            Ds.append(D)

        Dsn = np.array(Ds)
        threshold = Dsn.argmax()
        if (inv==0):
            _,mask = cv2.threshold(img,threshold,255,cv2.THRESH_BINARY_INV)

        if (inv==1):
            _,mask = cv2.threshold(img,threshold,255,cv2.THRESH_BINARY)

    return mask
```

In [6]:

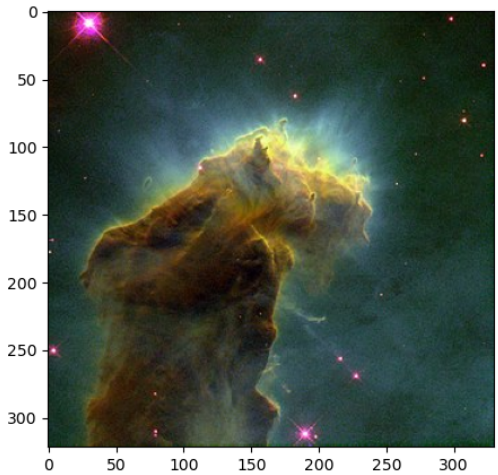
```
#Wrapper function for Otsu
def combine_Otsu(img,parameter,inv):
    mask_final = np.ones((img.shape[0],img.shape[1]))
    masks = []
    for i in range(3):
        maski = Otsu(img,i,parameter,inv)
        masks.append(maski)


    return masks
```

In [7]:

```
#Uses the mask to generate the contour
def get_contour(mask):
    contour = np.zeros(mask.shape)
    for i in range(1,mask.shape[0]-1):
        for j in range(1,mask.shape[1]-1):
            if mask[i,j] == 0:
                continue
            if np.min(mask[i-1:i+2,j-1:j+2])==0:
                contour[i,j] = 1.0


    return contour.astype('uint8')
```

In [33]:

```
#Displaying all the images
image1 = cv2.imread("goyaw.jpg")
image1 = cv2.cvtColor(image1,cv2.COLOR_BGR2RGB)

image2 = cv2.imread("pillar.jpg")
image2 = cv2.cvtColor(image2,cv2.COLOR_BGR2RGB)

fig = plt.figure(figsize=(15,5))
ax1 = fig.add_subplot(121)
ax2 = fig.add_subplot(122)

ax1.imshow(image1)
ax2.imshow(image2)

#Defining the number of iterations for 1st Otsu method for each of the RGB channels
image1_channel = [1,1,2]
image2_channel = [2,1,2]
```



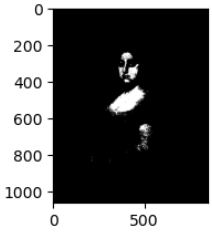
In [13]:

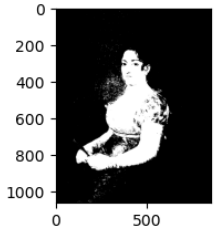
```
#For image 1, using the 1st Otsu method
mask_ind_1 = combine_Otsu(image1,image1_channel,1)

mask_1 = combining_masks(mask_ind_1,1) #Using or operator to combine the masks

iou1 = applying_mask(image1,mask_1)

fig = plt.figure(figsize=(15,5))
ax1 = fig.add_subplot(231)
ax2 = fig.add_subplot(232)
ax3 = fig.add_subplot(233)
ax4 = fig.add_subplot(235)
ax5 = fig.add_subplot(236)

ax1.imshow(mask_ind_1[0],cmap='gray')
ax2.imshow(mask_ind_1[1],cmap='gray')
ax3.imshow(mask_ind_1[2],cmap='gray')
ax4.imshow(mask_1,cmap='gray')
ax5.imshow(iou1,cmap='gray')

cv2.imwrite("mask11.jpg",mask_ind_1[0])
cv2.imwrite("mask12.jpg",mask_ind_1[1])
cv2.imwrite("mask13.jpg",mask_ind_1[2])
cv2.imwrite("mask14.jpg",mask_1*255.0)

iou1 = cv2.cvtColor(iou1, cv2.COLOR_RGB2BGR)
cv2.imwrite("mask15.jpg",iou1)
```

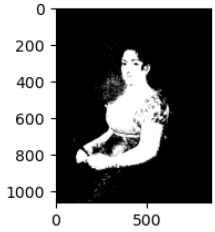
```
C:\Users\adisi\AppData\Local\Temp\ipykernel_8208\3543035357.py:16: RuntimeWa
rning: invalid value encountered in double_scalars
  mean0 = np.sum([a*b for a,b in enumerate(h0)])/w0
C:\Users\adisi\AppData\Local\Temp\ipykernel_8208\3543035357.py:19: RuntimeWa
rning: invalid value encountered in double_scalars
  var0 = np.sum([((a-mean0)**2)*b for a,b in enumerate(h0)])/w0
C:\Users\adisi\AppData\Local\Temp\ipykernel_8208\3543035357.py:17: RuntimeWa
rning: invalid value encountered in double_scalars
  mean1 = np.sum([a*b for a,b in enumerate(h1)])/w1
```

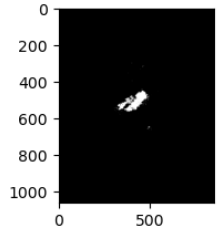
Out[13]:

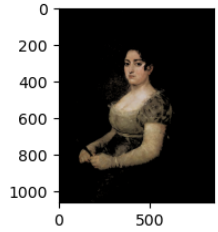
```
True
```


0
200
400
600
800
1000
0
500


0
200
400
600
800
1000
0
500


0
200
400
600
800
1000
0
500


0
200
400
600
800
1000
0
500


0
200
400
600
800
1000
0
500

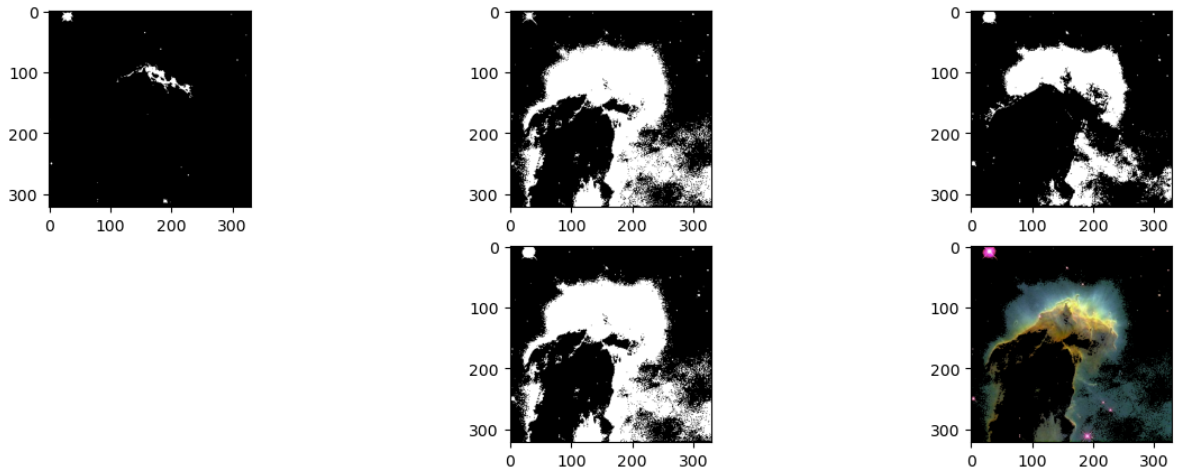
In [34]:

```
#For image 2, using the 1st Otsu method
mask_ind_2 = combine_Otsu(image2,image2_channel,1)
mask_2 = combining_masks(mask_ind_2,1) #Using or operator to combine the masks

iou2 = applying_mask(image2,mask_2)

fig = plt.figure(figsize=(15,5))
ax1 = fig.add_subplot(231)
ax2 = fig.add_subplot(232)
ax3 = fig.add_subplot(233)
ax4 = fig.add_subplot(235)
ax5 = fig.add_subplot(236)

ax1.imshow(mask_ind_2[0],cmap='gray')
ax2.imshow(mask_ind_2[1],cmap='gray')
ax3.imshow(mask_ind_2[2],cmap='gray')
ax4.imshow(mask_2,cmap='gray')
ax5.imshow(iou2,cmap='gray')

cv2.imwrite("mask21.jpg",mask_ind_2[0])
cv2.imwrite("mask22.jpg",mask_ind_2[1])
cv2.imwrite("mask23.jpg",mask_ind_2[2])
cv2.imwrite("mask24.jpg",mask_2*255.0)

iou2 = cv2.cvtColor(iou2, cv2.COLOR_RGB2BGR)
cv2.imwrite("mask25.jpg",iou2)
```

```
C:\Users\adisi\AppData\Local\Temp\ipykernel_8208\3543035357.py:16: RuntimeWa
rning: invalid value encountered in double_scalars
  mean0 = np.sum([a*b for a,b in enumerate(h0)])/w0
C:\Users\adisi\AppData\Local\Temp\ipykernel_8208\3543035357.py:19: RuntimeWa
rning: invalid value encountered in double_scalars
  var0 = np.sum([((a-mean0)**2)*b for a,b in enumerate(h0)])/w0
```

Out[34]:

```
True
```



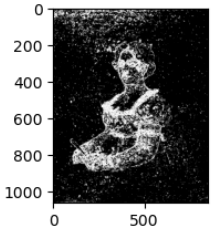
In [25]:

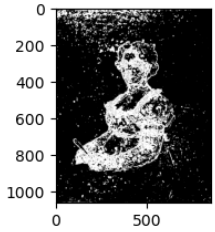
```
#For image 1 using the 2nd Otsu method
nimage1_channel = [1,1,1]

textures1 = get_texture(image1, [1,2,3])

inmask_ind_1 = combine_Otsu(textures1,nimage1_channel,1)

inmask_1 = combining_masks(inmask_ind_1,1) #Using or operator to combine the masks
niou1 = applying_mask(image1,inmask_1)

fig = plt.figure(figsize=(15,5))
ax1 = fig.add_subplot(231)
ax2 = fig.add_subplot(232)
ax3 = fig.add_subplot(233)
ax4 = fig.add_subplot(235)
ax5 = fig.add_subplot(236)

ax1.imshow(inmask_ind_1[0],cmap='gray')
ax2.imshow(inmask_ind_1[1],cmap='gray')
ax3.imshow(inmask_ind_1[2],cmap='gray')
ax4.imshow(inmask_1,cmap='gray')
ax5.imshow(niou1,cmap='gray')

cv2.imwrite("nmask11.jpg",inmask_ind_1[0])
cv2.imwrite("nmask12.jpg",inmask_ind_1[1])
cv2.imwrite("nmask13.jpg",inmask_ind_1[2])
cv2.imwrite("nmask14.jpg",inmask_1*255.0)

niou1 = cv2.cvtColor(niou1, cv2.COLOR_RGB2BGR)
cv2.imwrite("nmask15.jpg",niou1)
```

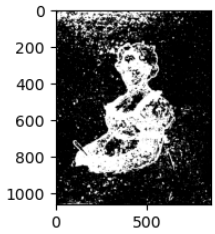
```
C:\Users\adisi\AppData\Local\Temp\ipykernel_8208\3543035357.py:16: RuntimeWa
rning: invalid value encountered in double_scalars
  mean0 = np.sum([a*b for a,b in enumerate(h0)])/w0
C:\Users\adisi\AppData\Local\Temp\ipykernel_8208\3543035357.py:19: RuntimeWa
rning: invalid value encountered in double_scalars
  var0 = np.sum([((a-mean0)**2)*b for a,b in enumerate(h0)])/w0
```

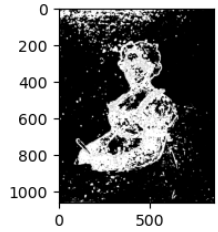
Out[25]:

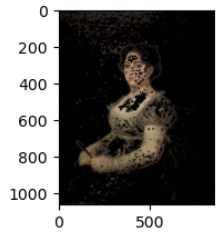
```
True
```


0
200
400
600
800
1000
0
500


0
200
400
600
800
1000
0
500


0
200
400
600
800
1000
0
500


0
200
400
600
800
1000
0
500


0
200
400
600
800
1000
0
500

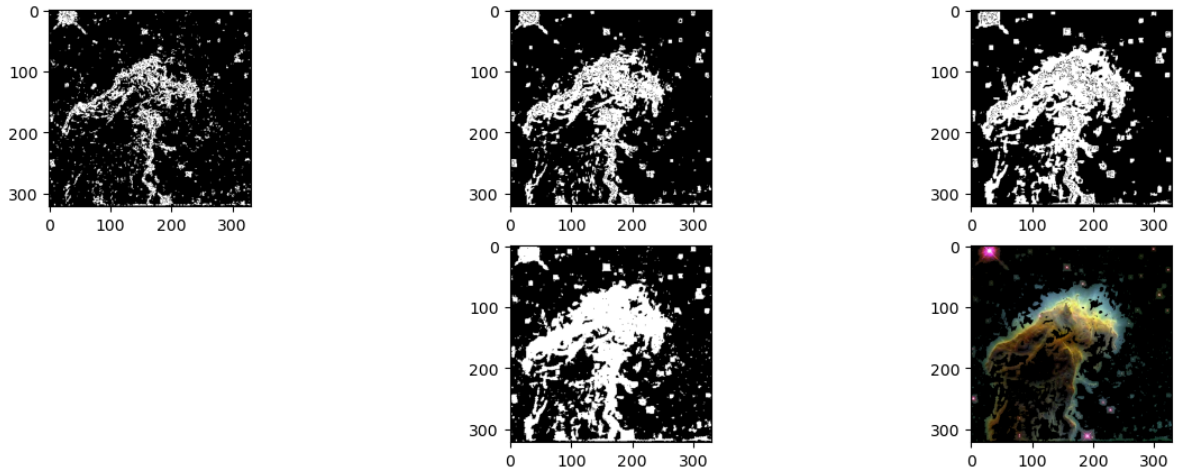
In [19]:

```
#For image 2 using the 2nd Otsu method
nimage2_channel = [1,1,1]
textures2 = get_texture(image2,[1,2,3])

inmask_ind_2 = combine_Otsu(textures2,nimage2_channel,1)

inmask_2 = combining_masks(inmask_ind_2,1) #Using and operator to combine the masks
niou2 = applying_mask(image2,inmask_2)

fig = plt.figure(figsize=(15,5))
ax1 = fig.add_subplot(231)
ax2 = fig.add_subplot(232)
ax3 = fig.add_subplot(233)
ax4 = fig.add_subplot(235)
ax5 = fig.add_subplot(236)

ax1.imshow(inmask_ind_2[0],cmap='gray')
ax2.imshow(inmask_ind_2[1],cmap='gray')
ax3.imshow(inmask_ind_2[2],cmap='gray')
ax4.imshow(inmask_2,cmap='gray')
ax5.imshow(niou2,cmap='gray')

cv2.imwrite("nmask21.jpg",inmask_ind_2[0])
cv2.imwrite("nmask22.jpg",inmask_ind_2[1])
cv2.imwrite("nmask23.jpg",inmask_ind_2[2])
cv2.imwrite("nmask24.jpg",inmask_2*255.0)

niou2 = cv2.cvtColor(niou2, cv2.COLOR_RGB2BGR)
cv2.imwrite("nmask25.jpg",niou2)
```

```
C:\Users\adisi\AppData\Local\Temp\ipykernel_8208\3543035357.py:16: RuntimeWa
rning: invalid value encountered in double_scalars
  mean0 = np.sum([a*b for a,b in enumerate(h0)])/w0
C:\Users\adisi\AppData\Local\Temp\ipykernel_8208\3543035357.py:19: RuntimeWa
rning: invalid value encountered in double_scalars
  var0 = np.sum([((a-mean0)**2)*b for a,b in enumerate(h0)])/w0
```

Out[19]:

```
True
```

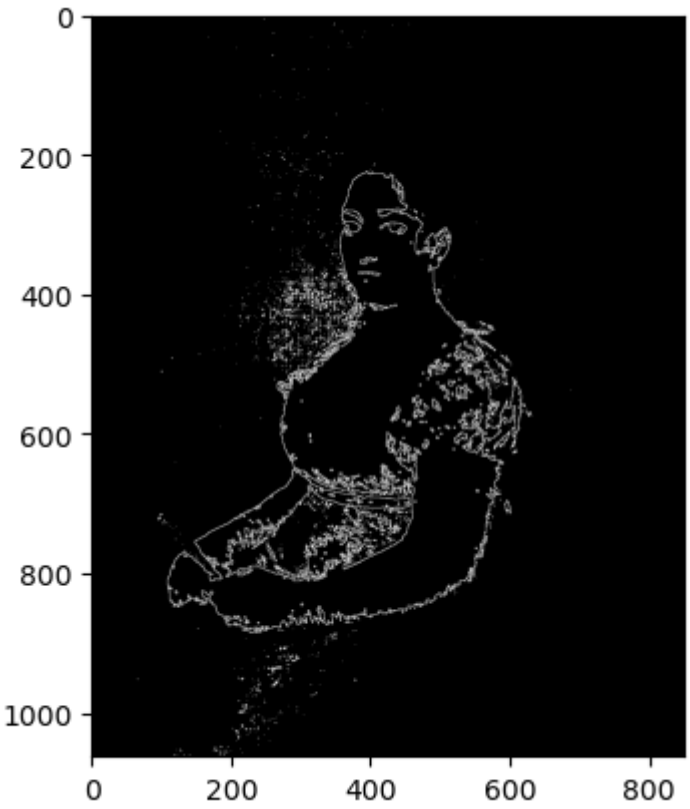


In [20]:

```
#Producing contours for the 1st image
c11 = get_contour(mask_1)
plt.imshow(c11,cmap='gray')
cv2.imwrite("contour11.jpg",c11*255.0)
```

Out[20]:

```
True
```

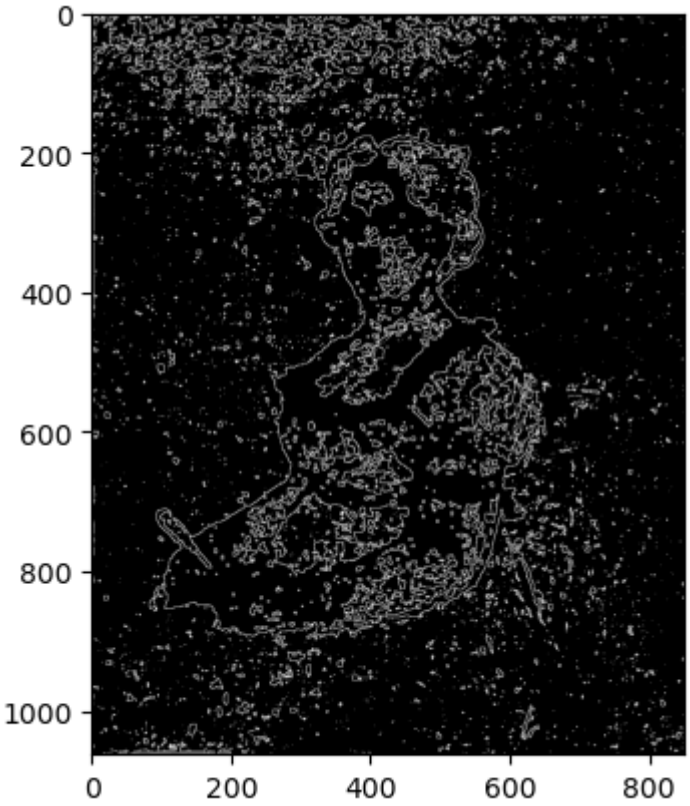


In [21]:

```
#Producing contours for the 1st image using texture
c12 = get_contour(inmask_1)

plt.imshow(c12,cmap='gray')
cv2.imwrite("contour12.jpg",c12*255.0)
```

Out[21]:

```
True
```

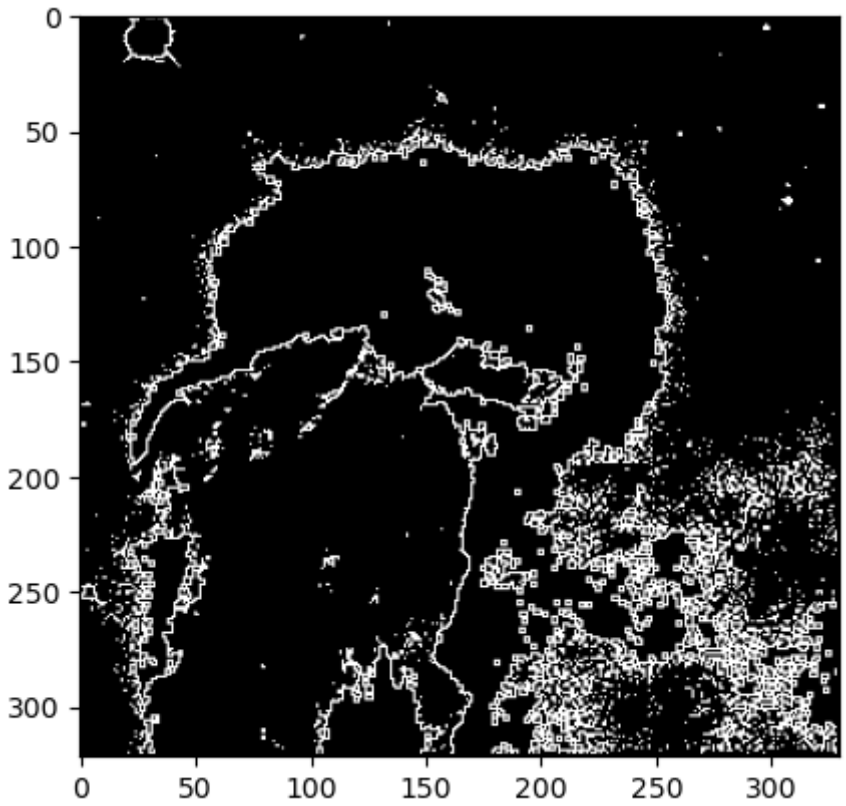


In [22]:

```
#Producing contours for the 2nd image
c21 = get_contour(mask_2)
plt.imshow(c21,cmap='gray')
cv2.imwrite("contour21.jpg",c21*255.0)
```

Out[22]:

True



In [23]:

```
#Producing contours for the 2nd image using texture
c22 = get_contour(inmask_2)
plt.imshow(c22,cmap='gray')
cv2.imwrite("contour22.jpg",c22*255.0)
```

Out[23]:

```
True
```