DIGITAL IMAGE PROCESSING
ELL715
ASSIGNMENT 2

26 August 2023

Group Members

1. Bhavik Shankla (2020MT60873)
2. Ritika Soni (2020MT10838)
3. Sai Kiran Gunnala (2020MT60889)
4. Sai Niketh Varanasi (2020MT60895)
5. Yash Pravin Shirke (2020MT60986)

Question

In this assignment, you'll get to play with two special tools: one that finds the edges of things and another that spots shapes. Grab a picture with lots of geometric shapes, like a painting with lines, circles, triangles, ellipse etc. and find these shapes. Actually I want you to implement canny edge algorithm and Hough transform once you find edges.

1. Demonstrate the detection of shapes in the image of your choice
2. Numerous built-in functions are at your disposal (check Google) for various geometrical shapes. Your task is to discover a new geometrical figure (you can design your own) and write the Hough transform for it.
3. Also extend your search in 3-dimension.

Solution

1.

Python Code:

```
import numpy as np
import matplotlib.pyplot as plt
import cv2
from google.colab.patches import cv2_imshow

%matplotlib inline
# Read in the image
image = cv2.imread('/content/combo.jpg')
cv2_imshow(image)


import cv2
from google.colab.patches import cv2_imshow
```

```
# Use the cvtColor() function to grayscale the resized image
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
cv2_imshow(gray_image)
cv2.waitKey(0)

# Window shown waits for any key pressing event
cv2.destroyAllWindows()


# Define our parameters for Canny
low_threshold = 50
high_threshold = 100
edges = cv2.Canny(gray_image, low_threshold, high_threshold)
cv2_imshow(edges)


# Apply Gaussian blur to reduce noise and improve circle detection
blurred_image = cv2.GaussianBlur(gray_image, (9, 9), 2)


# Define the Hough transform parameters
# Make a blank the same size as our image to draw on
rho = 1
theta = np.pi/180
threshold = 20
min_line_length = 11
max_line_gap = 11
line_image = np.copy(image) #creating an image copy to draw lines on
# Run Hough on the edge-detected image
lines = cv2.HoughLinesP(edges, rho, theta, threshold, np.array([]),
    min_line_length, max_line_gap)
# Iterate over the output "lines" and draw lines on the image copy
for line in lines:
    for x1,y1,x2,y2 in line:
        cv2.line(line_image,(x1,y1),(x2,y2),(255,0,0),5)

cv2_imshow(line_image)


# Apply Hough transform to greyscale image
circles = cv2.HoughCircles(blurred_image,cv2.HOUGH_GRADIENT,1,20,
    param1=60,param2=40,minRadius=0,maxRadius=0)


circles = np.uint16(np.around(circles))
# Draw the circles
for i in circles[0,:]:
    # draw the outer circle
    cv2.circle(image,(i[0],i[1]),i[2],(0,255,0),2)
    # draw the center of the circle
    cv2.circle(image,(i[0],i[1]),2,(0,0,255),3)
cv2_imshow(image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

2.

The detection of an arbitrary shape in an image using the Generalized Hough Transform involves a series of steps. First, an R-Table was created for the template shape, encoding gradient information for various angles and distances. Subsequently, an accumulator array, matching the dimensions of the input image, was generated and initialized. The primary process included "voting," during which edge pixels in the input image contributed votes to the accumulator cells based on calculated gradient attributes. Local maxima were then identified to detect potential positions of the shape. These positions were indicated in an output image, thereby illustrating instances of the arbitrary shape that had been detected using the bounding box technique.

Python Code:

```python
import numpy as np
from skimage import io
from skimage.io import imread, imshow
import matplotlib.pyplot as plt


def findRTheta(x1, y1,imgCenter):
    x2, y2 = imgCenter[0], imgCenter[1]
    r = [(x2-x1), (y2-y1)]
    if (x2-x1 != 0):
        return [int(np.rad2deg(np.arctan(int((y2-y1)/(x2-x1))))), r]
    else:
        return [0, 0]


def findGradient(x, y):
    if (x != 0):
        return int(np.rad2deg(np.arctan(int(y/x))))
    else:
        return 0


def buildRTable(img):
    rtable = [[0 for x in range(1)] for y in range(90)]  # Creating a empty
        list
    # r will be calculated corresponding to this point as reference point
    xCenter= int(img.shape[0]/2)
    yCenter= int(img.shape[1]/2)

    imgCenter= [xCenter,yCenter]

    filter = 3
    for x in range(img.shape[0]-(filter-1)):
        for y in range(img.shape[1]-(filter-1)):
            if (img[x, y] != 0):
                theta, r = findRTheta(x, y,imgCenter)
                if (r != 0):
                    rtable[np.absolute(theta)].append(r)

    for i in range(len(rtable)):
        rtable[i].pop(0)

    return rtable


def findMaxima(accumulator):
    ridx, cidx = np.unravel_index(accumulator.argmax(), accumulator.shape)
    return [accumulator[ridx, cidx], ridx, cidx]
```

```
def matchTable(testImage, table):
    # Matching the reference table with the given input Image for testing
        generalized Hough Transform
    n, m = testImage.shape
    acc = np.zeros((n+50, m+50))  # Accumulator array requires some extra space

    for x in range(1,n):
        for y in range(1,m):
            if testImage[x, y] != 0:
                theta = findGradient(x, y)
                vectors = table[theta]
                for vector in vectors:
                    acc[vector[0]+x, vector[1]+y] += 1
    return acc


img= 'kettle.png'

template = imread(img)
testImage = imread('shapes.png')

print("The Shape I am finding using General Hough Transform >>")
plt.figure(figsize=(4, 4)), imshow(template)
plt.title("Kettle")
plt.show()
print("The Image where I am fiding the custom shape >>")
plt.figure(figsize=(4, 4)), imshow(testImage)
plt.title("Image with a lot of shapes ")
plt.show()

table = buildRTable(template)
acc = matchTable(testImage, table)
val, ridx, cidx = findMaxima(acc)

# Drawing bounding-box in accumulator matrix
for i in range(ridx-5,ridx+6):
    acc[i,cidx-5]=val
    acc[i,cidx+5]=val

for i in range(cidx-5,cidx+6):
    acc[ridx+5,i]=val
    acc[ridx-5,i]=val

plt.figure(1)
plt.title("Accumulator")
imshow(acc)
plt.show()



# Drawing bounding-box in original image at the found location

# Calculating the half-width and height of custom shape
boxHeight = np.floor(template.shape[0] / 2) + 1
boxWidth = np.floor(template.shape[1] / 2) + 1

# Calculating coordinates of the box
top = int(max(ridx - boxHeight, 1))
bottom = int(min(ridx + boxHeight, testImage.shape[0] - 1))
left = int(max(cidx - boxWidth, 1))
right = int(min(cidx + boxWidth, testImage.shape[1] - 1))
```

```python
# Drawing the box
for i in range(top,bottom+1):
    testImage[i,left]=255
    testImage[i,right]=255

for i in range(left,right+1):
    testImage[top,i]=255
    testImage[bottom,i]=255

# Result Image
print("Shape Found >>")
plt.figure(figsize=(5, 5)), imshow(testImage)
plt.title("Shape Found ")
plt.show()
```

3.

As the question stated we need to extend our search to 3D parameter space, so our group has chosen to proceed with the detection of the circle with 3 parameters. The 3 parameters are x, y, and r where (x, y) is the coordinate of the center and r is the radius of the circle.

Now, after loading the image, we convert it into a gray-scale (to apply the Hough transform), and for the detection of cycles, it's best practice to blur the image to remove noise and improve edge detection. We have used Gaussian blur to remove noise. Later we apply a canny edge detection algorithm to identify edges and then proceed to apply circle Hough transform algorithm to detect circles. In circle Hough transform we will have a 3D parameter space and hence we get a cone and then we vote along this cone.

Python Code:

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


image = cv2.imread('coins.jpg',1)  # Loading the image
plt.imshow(image)
plt.show()

# Loading the same image as a grayscale image
gray_image=cv2.imread('coins.jpg',0)
plt.imshow(gray_image, cmap='gray')
plt.show()


# blurring the image to identify circles to remove noise using gaussian filter
blurred_image = cv2.GaussianBlur(gray_image, (7, 7), 13.5)
plt.imshow(blurred_image, cmap='gray')
plt.show()


# Apply Canny edge detection to the image to detect the edges
edges = cv2.Canny(blurred_image, 50, 150)

plt.imshow(edges, cmap='gray')
plt.show()

# as we were to extend our search to 3D parameter space our group has chosen
    circle
# where the 3 parameters are x,y,r - (x,y) - coordinates of centre and r being
    the radius of circle

circles = cv2.HoughCircles(image=blurred_image, method=cv2.HOUGH_GRADIENT,
    dp=1, minDist=80,minRadius=80, maxRadius=300)


circles = np.uint16(np.around(circles))
# drawing the circles found on the original image
for i in circles[0,:]:
    # draw the outline of circle
    cv2.circle(image,(i[0],i[1]),i[2],(255,0,0),5)
    # draw the center of the circle
    cv2.circle(image,(i[0],i[1]),2,(0,0,255),5)
plt.imshow(image)
plt.show()
```

```
# *******TRYING THIS FOR ANOTHER IMAGE***********
image2 = cv2.imread('sp.jpg',1)  # Loading the image
plt.imshow(image2)
plt.show()

# Loading the same image as a grayscale image
gray_image2=cv2.imread('sp.jpg',0)
plt.imshow(gray_image2, cmap='gray')
plt.show()


# blurring the image to identify circles to remove noise using gaussian filter
blurred_image2 = cv2.GaussianBlur(gray_image2, (7, 7), 13.5)
plt.imshow(blurred_image2, cmap='gray')
plt.show()


# Apply Canny edge detection to the image to detect the edges
edges = cv2.Canny(blurred_image2, 50, 150)

plt.imshow(edges, cmap='gray')
plt.show()


# as we were to extend our search to 3D parameter space our group has chosen
    circle
# where the 3 parameters are x,y,r - (x,y) - coordinates of centre and r being
    the radius of circle
circles = cv2.HoughCircles(image=blurred_image2, method=cv2.HOUGH_GRADIENT,
    dp=2, minDist=80,minRadius=10, maxRadius=100)


circles = np.uint16(np.around(circles))
# drawing the circles found on the original image
for i in circles[0,:]:
    # draw the outline of circle
    cv2.circle(image2,(i[0],i[1]),i[2],(255,0,0),5)
    # draw the center of the circle
    cv2.circle(image2,(i[0],i[1]),2,(0,0,255),5)
plt.imshow(image2)
plt.show()
```

We have taken 2 images to perform and depict the results. When we use the function we can observe the min and max radius of circles in the image and accordingly, we can set the parameters of minRadius and maxRadius for our search.

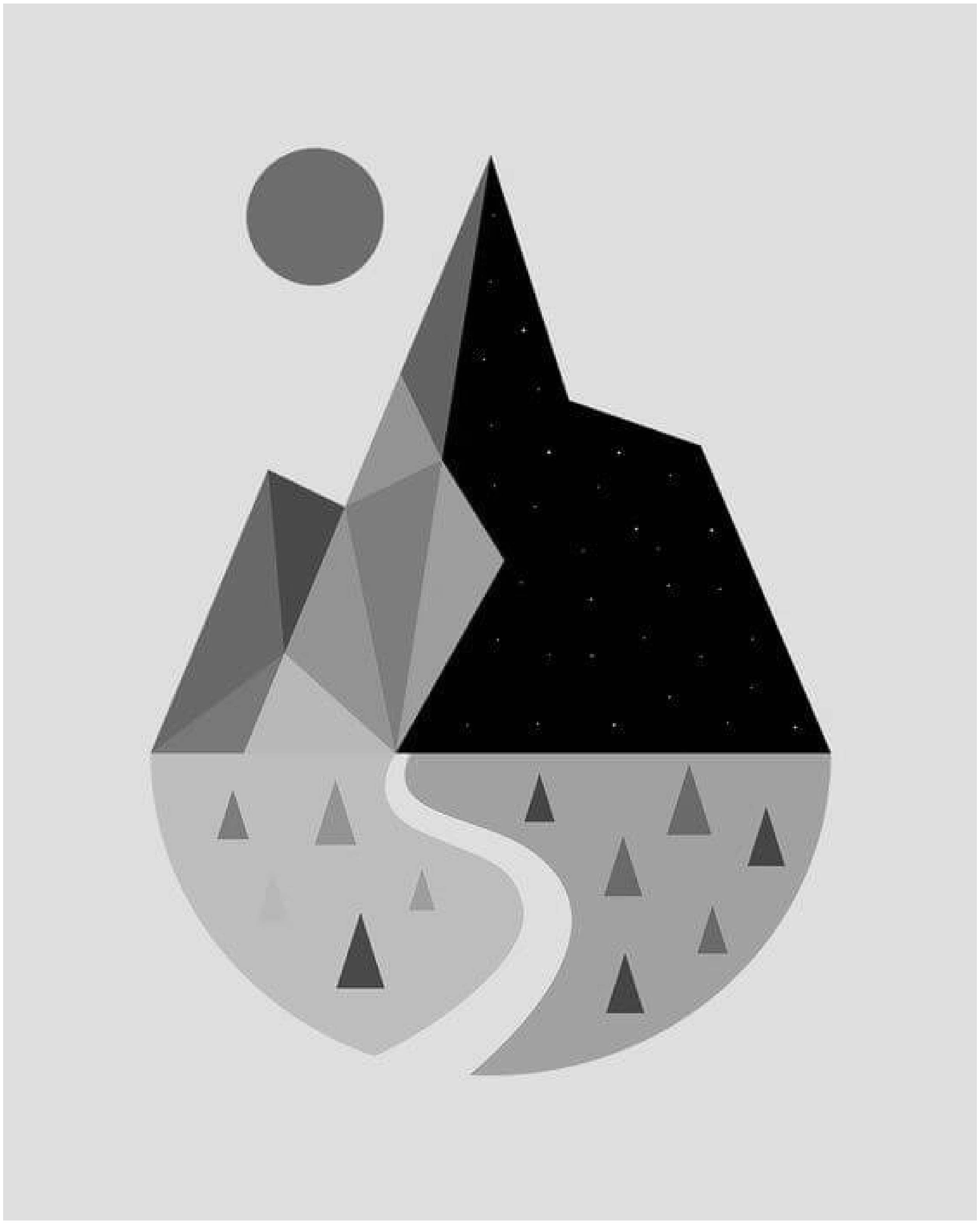
We have kept resolution (dp factor) dp=1 for the first image as the edges of the coins were clearly detected from the canny edge algorithm, whereas dp=2 for the second as the color was dim and some edges weren't recognized by the canny edge algorithm. So, increasing resolution will enable the circle hough transformation algo to detect finer edges.

This way we can extend our search to 3D parameterized space to detect circles of different radii.

```python
import numpy as np
import matplotlib.pyplot as plt
import cv2
from google.colab.patches import cv2_imshow

%matplotlib inline
# loading in the image
image = cv2.imread('/content/combo.jpg')
cv2_imshow(image)
```

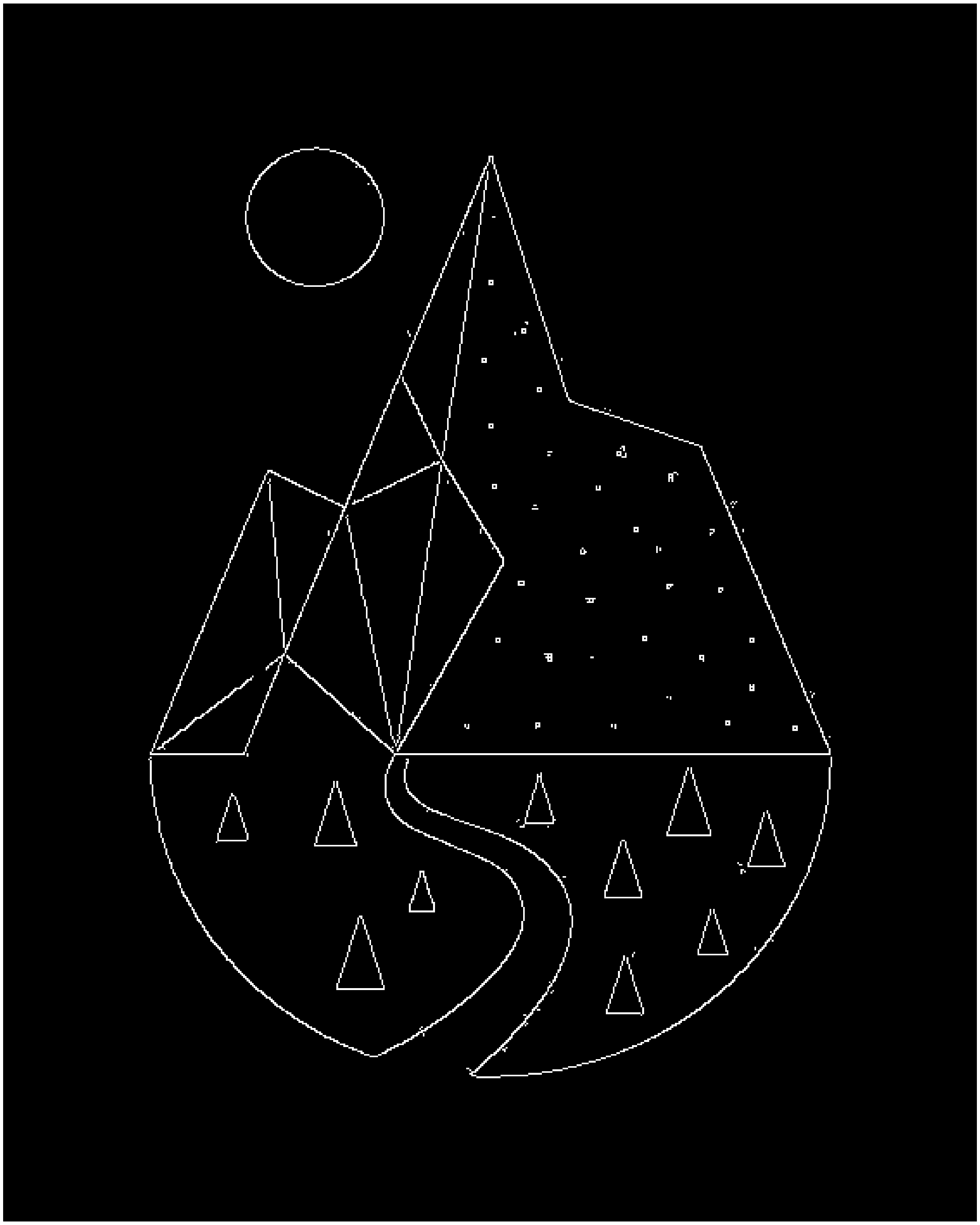
```
import cv2
from google.colab.patches import cv2_imshow
```

```python
# Use the cvtColor() function to grayscale the resized image
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
cv2_imshow(gray_image)
cv2.waitKey(0)

# Window shown waits for any key pressing event
cv2.destroyAllWindows()
```

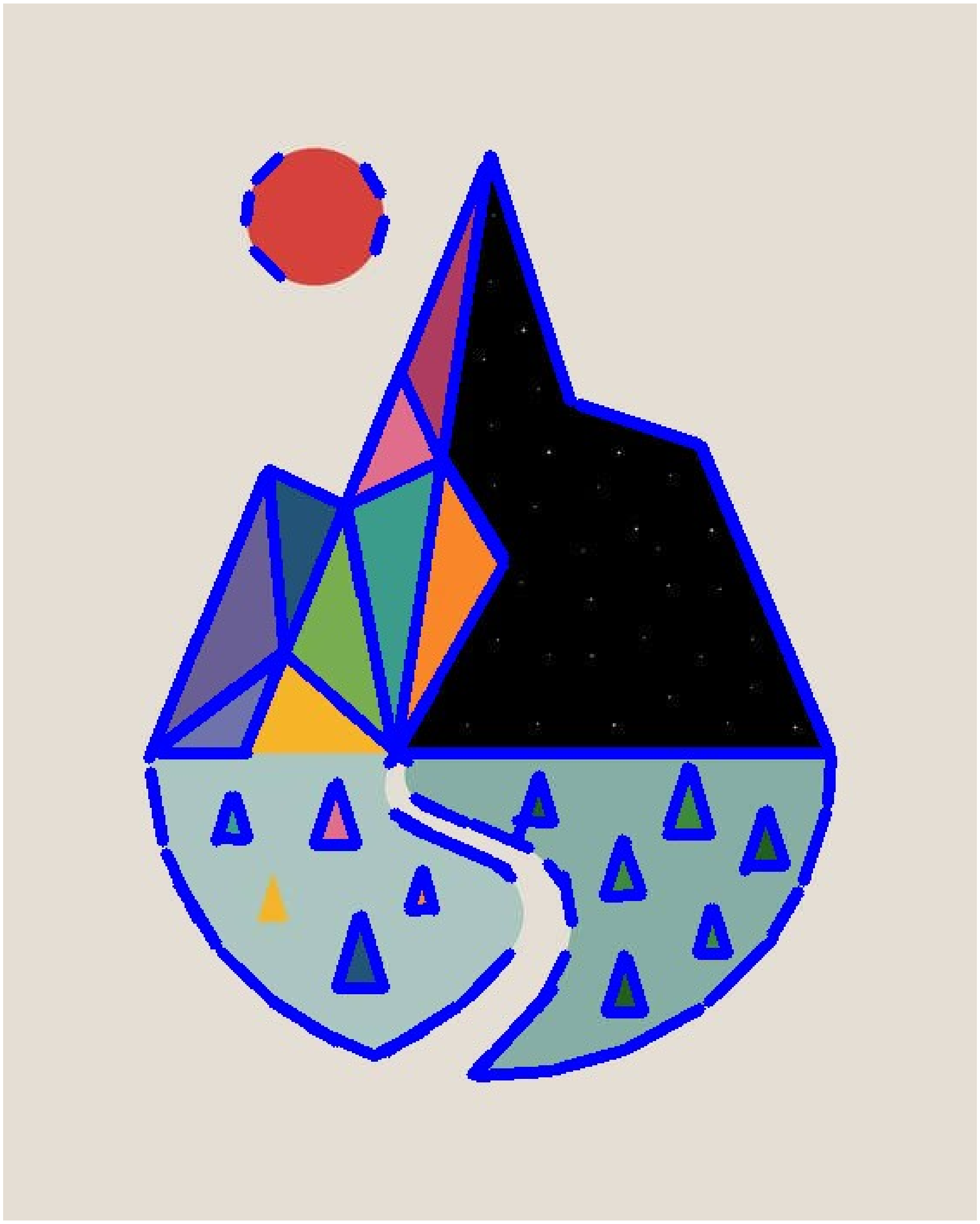


```
# Define our parameters for Canny
low_threshold = 50
high_threshold = 100
```

```
edges = cv2.Canny(gray_image, low_threshold, high_threshold)
cv2_imshow(edges)
```



```python
# Apply Gaussian blur to reduce noise and improve circle detection
blurred_image = cv2.GaussianBlur(gray_image, (9, 9), 2)

# Define the Hough transform parameters
# Make a blank the same size as our image to draw on
rho = 1
theta = np.pi/180
threshold = 20
min_line_length = 11
max_line_gap = 11
line_image = np.copy(image) #creating an image copy to draw lines on
# Run Hough on the edge-detected image
lines = cv2.HoughLinesP(edges, rho, theta, threshold, np.array([]),
                        min_line_length, max_line_gap)
# Iterate over the output "lines" and draw lines on the image copy
for line in lines:
    for x1,y1,x2,y2 in line:
        cv2.line(line_image,(x1,y1),(x2,y2),(255,0,0),5)

cv2_imshow(line_image)
```



```
# Apply Hough transform to greyscale image
circles = cv2.HoughCircles(blurred_image,cv2.HOUGH_GRADIENT,1,20,
                           param1=60,param2=40,minRadius=0,maxRadius=0)
```

```python
circles = np.uint16(np.around(circles))
# Draw the circles
for i in circles[0,:]:
    # draw the outer circle
    cv2.circle(image,(i[0],i[1]),i[2],(0,255,0),2)
    # draw the center of the circle
    cv2.circle(image,(i[0],i[1]),2,(0,0,255),3)
cv2_imshow(image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Ass2Q2

August 26, 2023

[10]:
```python
import numpy as np
from skimage import io
from skimage.io import imread, imshow
import matplotlib.pyplot as plt
```

[11]:
```python
def findRTheta(x1, y1,imgCenter):
    x2, y2 = imgCenter[0], imgCenter[1]
    r = [(x2-x1), (y2-y1)]
    if (x2-x1 != 0):
        return [int(np.rad2deg(np.arctan(int((y2-y1)/(x2-x1))))), r]
    else:
        return [0, 0]
```

[12]:
```python
def findGradient(x, y):
    if (x != 0):
        return int(np.rad2deg(np.arctan(int(y/x))))
    else:
        return 0
```

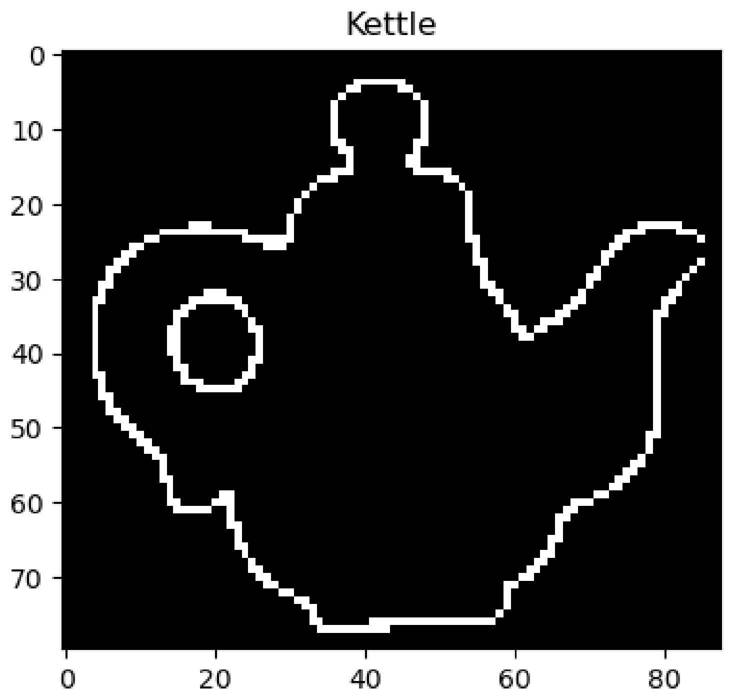
[13]:
```python
def buildRTable(img):
    rtable = [[0 for x in range(1)] for y in range(90)]  # Creating a empty list
    # r will be calculated corresponding to this point as reference point
    xCenter= int(img.shape[0]/2)
    yCenter= int(img.shape[1]/2)

    imgCenter= [xCenter,yCenter]

    filter = 3
    for x in range(img.shape[0]-(filter-1)):
        for y in range(img.shape[1]-(filter-1)):
            if (img[x, y] != 0):
                theta, r = findRTheta(x, y,imgCenter)
                if (r != 0):
                    rtable[np.absolute(theta)].append(r)

    for i in range(len(rtable)):
        rtable[i].pop(0)
    return rtable
```

[14]:
```
def findMaxima(accumulator):
    ridx, cidx = np.unravel_index(accumulator.argmax(), accumulator.shape)
    return [accumulator[ridx, cidx], ridx, cidx]
```

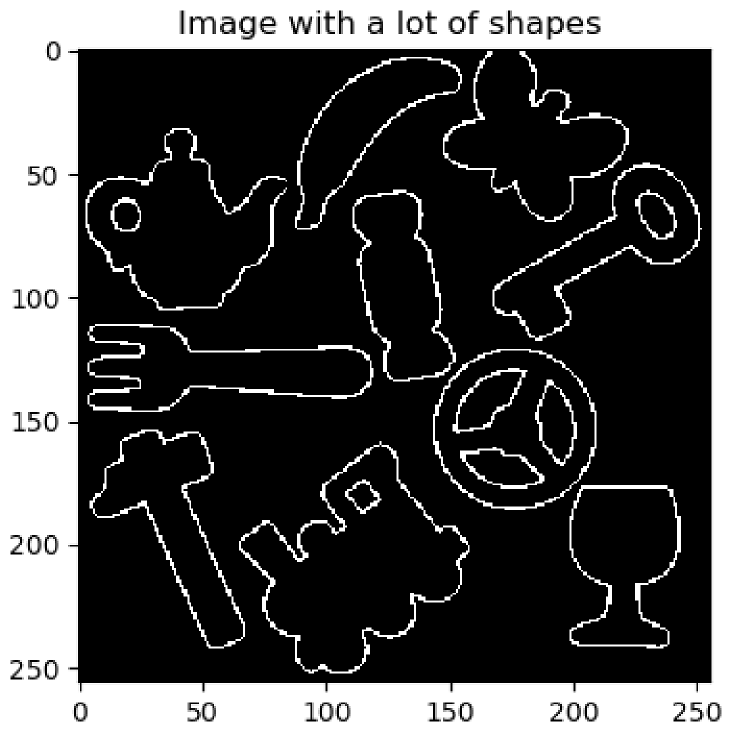
[15]:
```
def matchTable(testImage, table):
    # Matching the reference table with the given input Image for testing␣
 ↪generalized Hough Transform
    n, m = testImage.shape
    acc = np.zeros((n+50, m+50))  # Accumulator array requires some extra space

    for x in range(1,n):
        for y in range(1,m):
            if testImage[x, y] != 0:
                theta = findGradient(x, y)
                vectors = table[theta]
                for vector in vectors:
                    acc[vector[0]+x, vector[1]+y] += 1
    return acc
```

[16]:
```
img= 'kettle.png'

template = imread(img)
testImage = imread('shapes.png')

print("The Shape I am finding using General Hough Transform >>")
plt.figure(figsize=(4, 4)), imshow(template)
plt.title("Kettle")
plt.show()
print("The Image where I am fiding the custom shape >>")
plt.figure(figsize=(4, 4)), imshow(testImage)
plt.title("Image with a lot of shapes ")
plt.show()
```

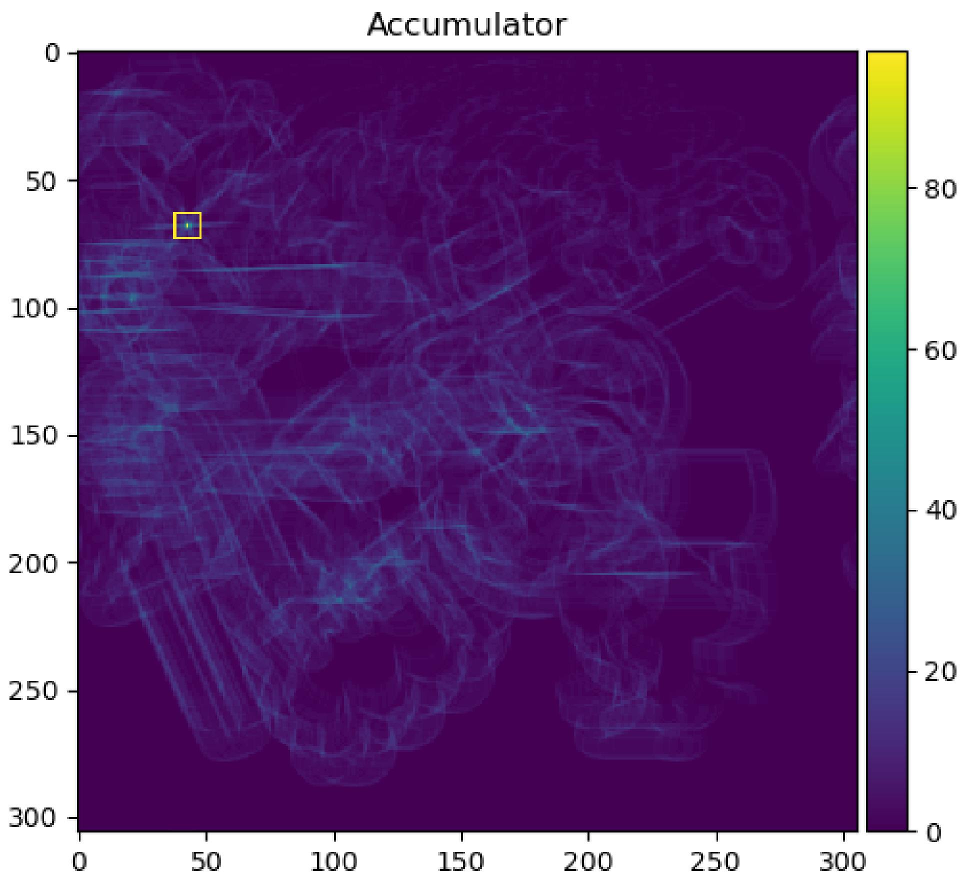
```
The Shape I am finding using General Hough Transform >>
```



The Image where I am fiding the custom shape >>

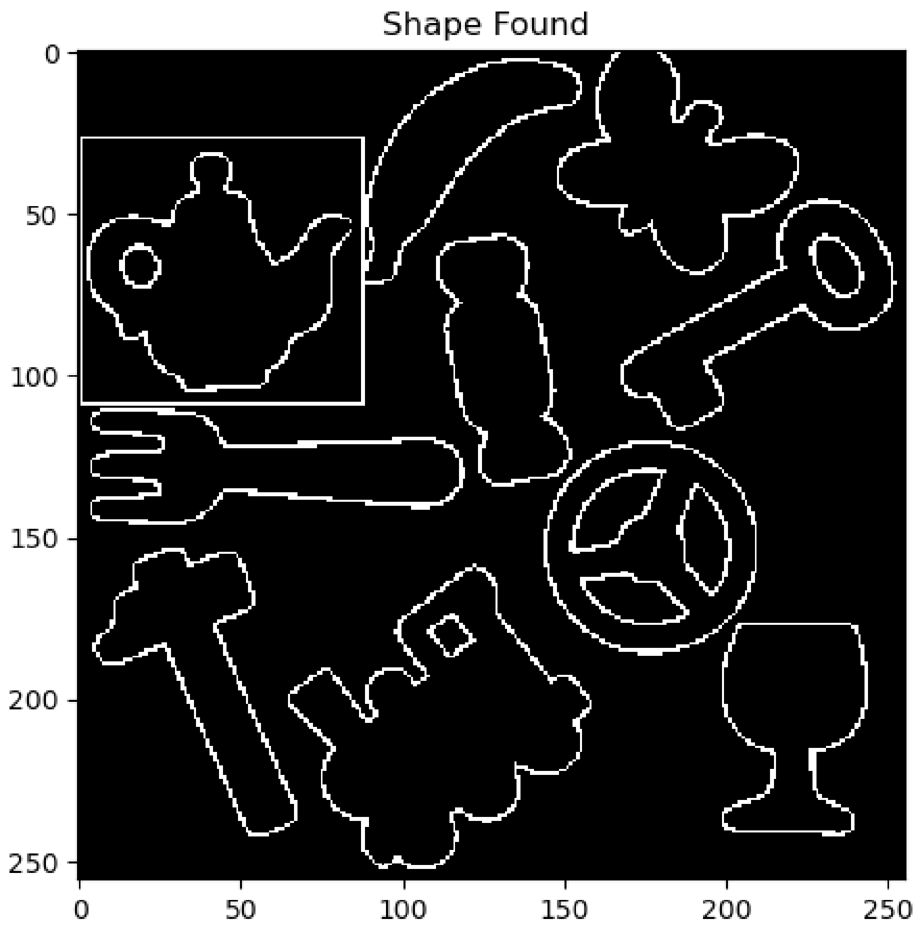


[17]:
```
table = buildRTable(template)
acc = matchTable(testImage, table)
val, ridx, cidx = findMaxima(acc)

# Drawing bounding-box in accumulator matrix
for i in range(ridx-5,ridx+6):
    acc[i,cidx-5]=val
    acc[i,cidx+5]=val

for i in range(cidx-5,cidx+6):
    acc[ridx+5,i]=val
    acc[ridx-5,i]=val

plt.figure(1)
plt.title("Accumulator")
imshow(acc)
plt.show()
```

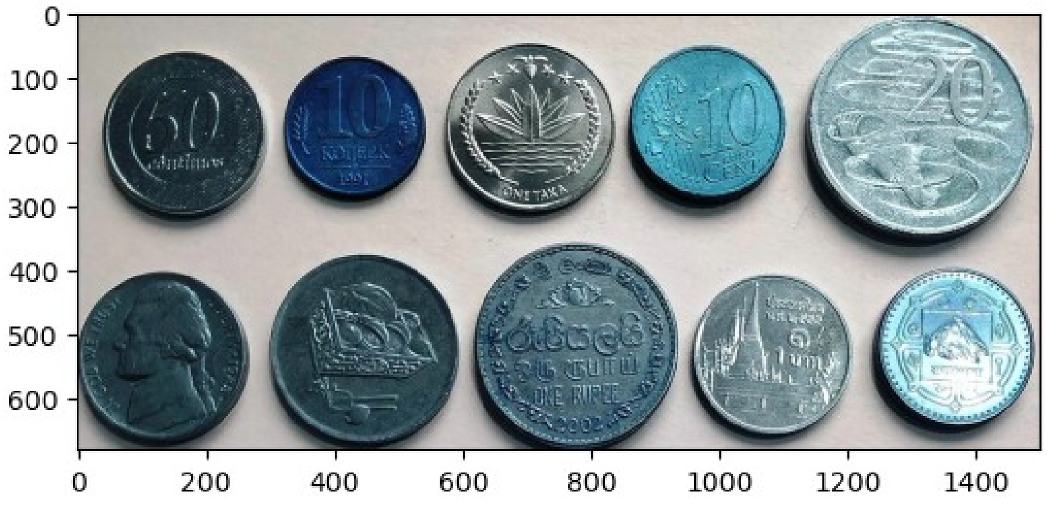


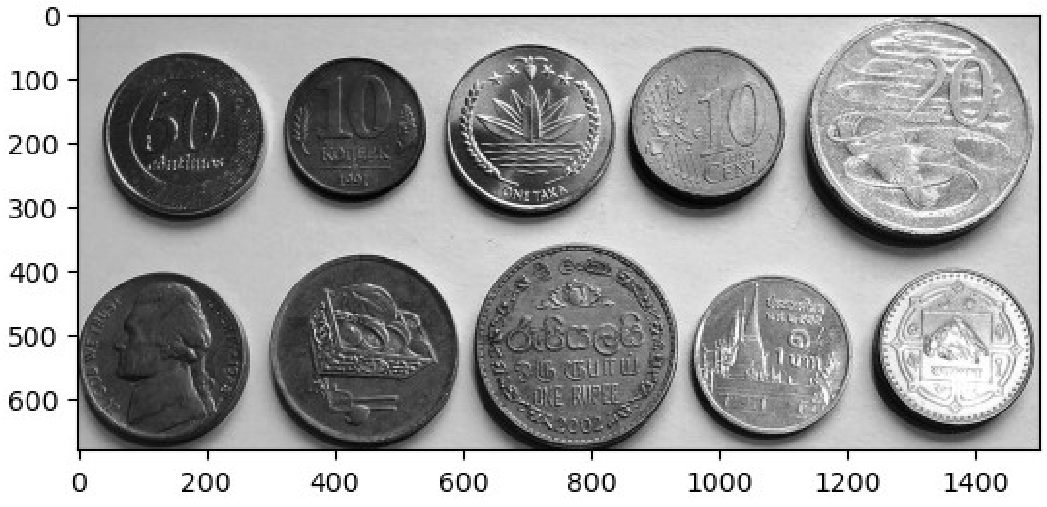
[18]:
```python
# Drawing bounding-box in original image at the found location

# Calculating the half-width and height of custom shape
boxHeight = np.floor(template.shape[0] / 2) + 1
boxWidth = np.floor(template.shape[1] / 2) + 1

# Calculating coordinates of the box
top = int(max(ridx - boxHeight, 1))
bottom = int(min(ridx + boxHeight, testImage.shape[0] - 1))
left = int(max(cidx - boxWidth, 1))
right = int(min(cidx + boxWidth, testImage.shape[1] - 1))

# Drawing the box
for i in range(top,bottom+1):
    testImage[i,left]=255
    testImage[i,right]=255

for i in range(left,right+1):
    testImage[top,i]=255
    testImage[bottom,i]=255

# Result Image
print("Shape Found >>")
plt.figure(figsize=(5, 5)), imshow(testImage)
plt.title("Shape Found ")
plt.show()
```

```
Shape Found >>
```


Shape Found
0
50
100
150
200
250
0
50
100
150
200
250

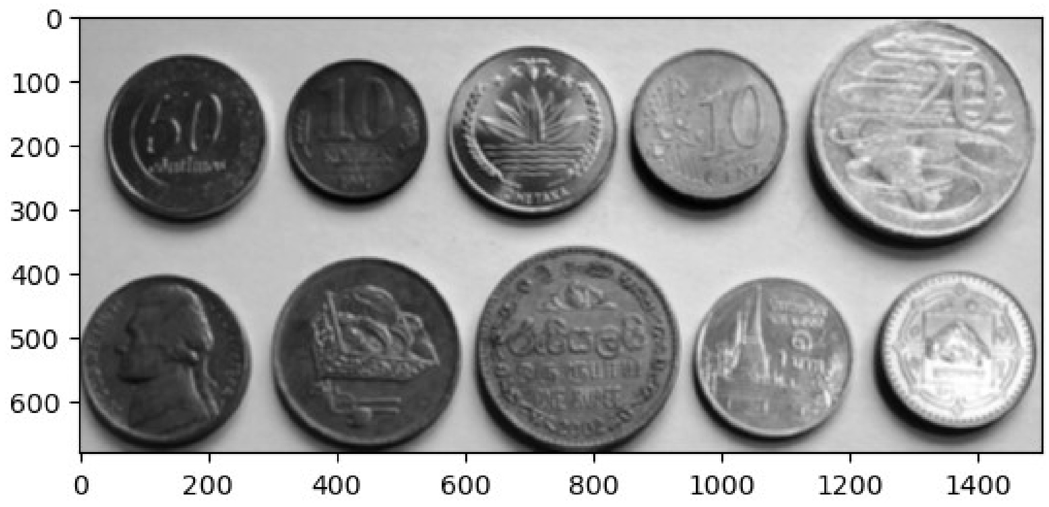
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import math

image = cv2.imread('coins.jpg',1)  # Loading the image
plt.imshow(image)
plt.show()
gray_image=cv2.imread('coins.jpg',0)  # Loading the same image as a
grayscale image
plt.imshow(gray_image, cmap='gray')
plt.show()
```



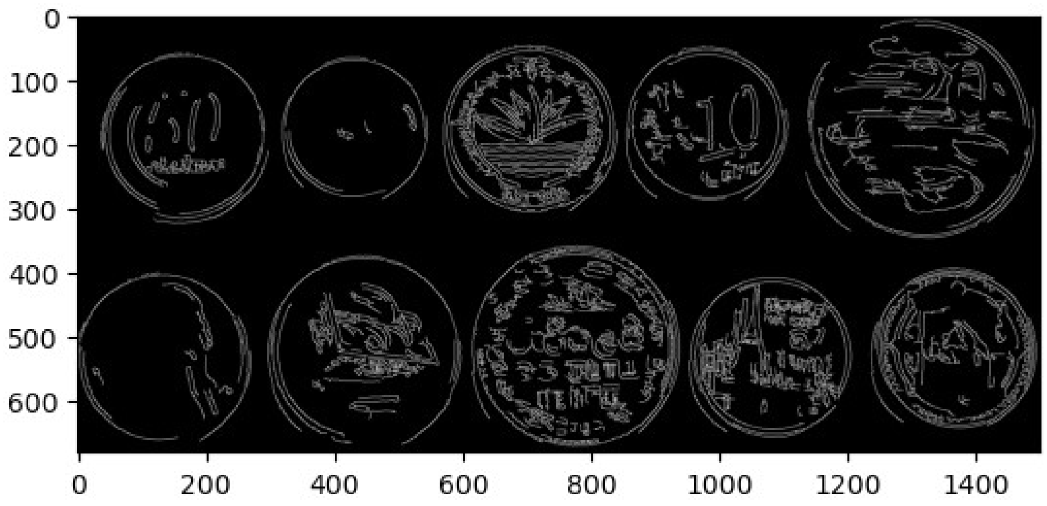


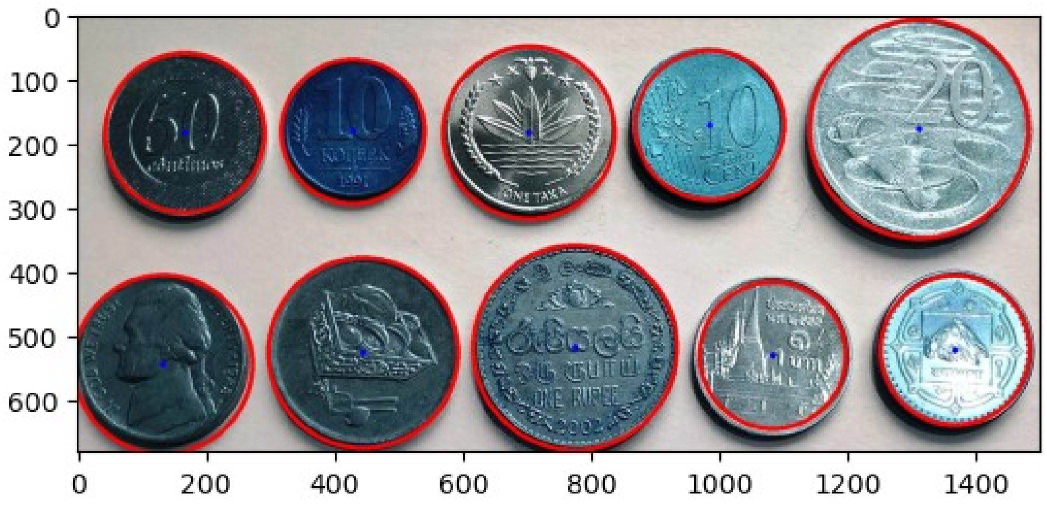
```
# blurring the image to identify circles to remove noise using
gaussian filter
blurred_image = cv2.GaussianBlur(gray_image, (7, 7), 13.5)
```

```
plt.imshow(blurred_image, cmap='gray')
plt.show()
```

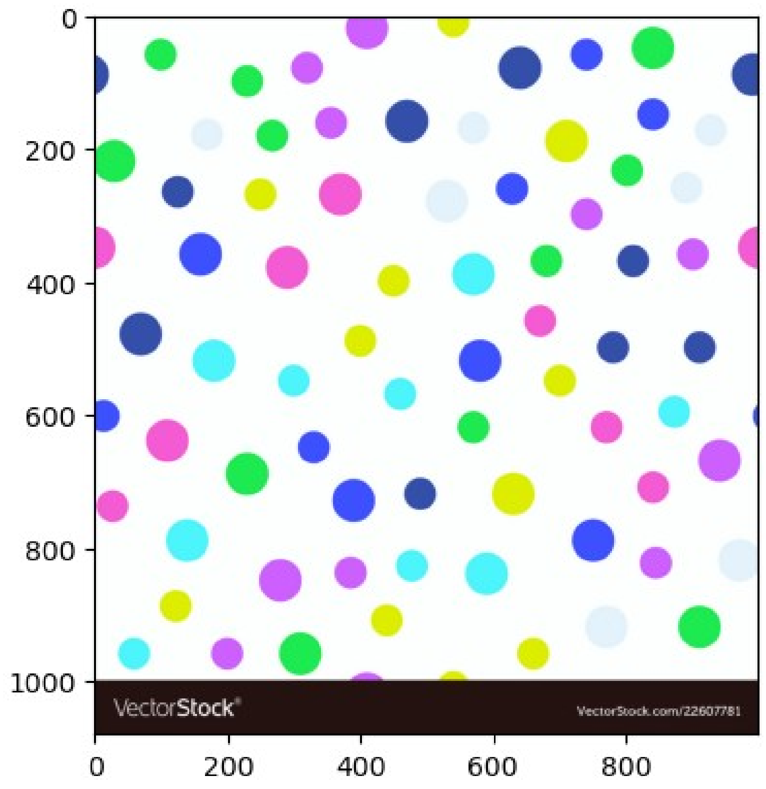


```
# Apply Canny edge detection to the image to detect the edges
edges = cv2.Canny(blurred_image, 50, 150)

plt.imshow(edges, cmap='gray')
plt.show()

# as we were to extend our search to 3D parameter space our group has
chosen circle
# where the 3 parameters are x,y,r - (x,y) - coordinates of centre and
r being the radius of circle

circles = cv2.HoughCircles(image=blurred_image,
method=cv2.HOUGH_GRADIENT, dp=1,
                            minDist=80,minRadius=80, maxRadius=300)

circles = np.uint16(np.around(circles))
# drawing the circles found on the original image
for i in circles[0,:]:
    # draw the outline of circle
    cv2.circle(image,(i[0],i[1]),i[2],(255,0,0),5)
    # draw the center of the circle
    cv2.circle(image,(i[0],i[1]),2,(0,0,255),5)
plt.imshow(image)
plt.show()
```

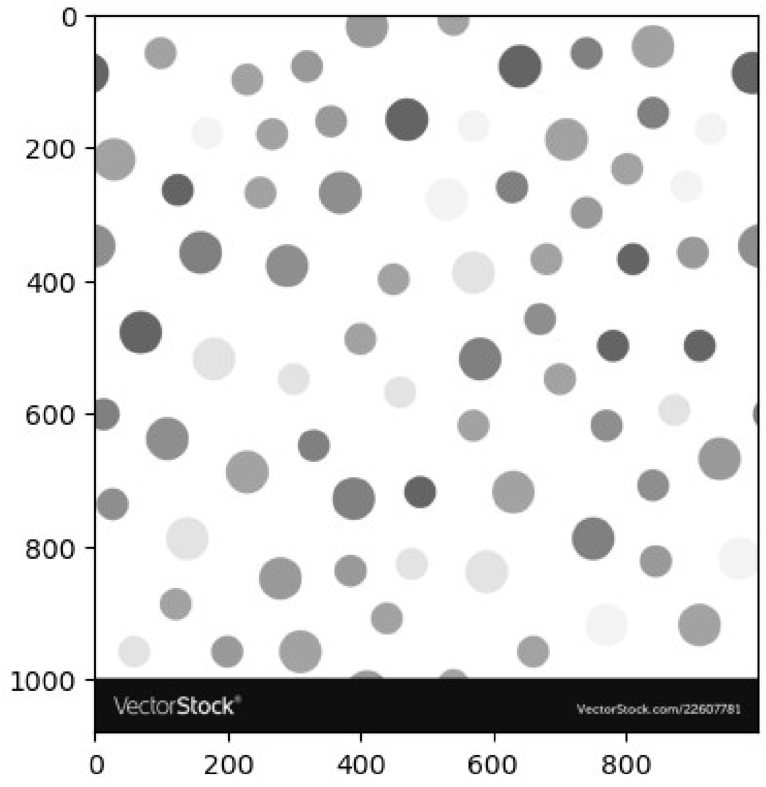




```
# trying this for another image
image2 = cv2.imread('sp.jpg',1)  # Loading the image
plt.imshow(image2)
plt.show()
gray_image2=cv2.imread('sp.jpg',0)  # Loading the same image as a
grayscale image
plt.imshow(gray_image2, cmap='gray')
plt.show()
```

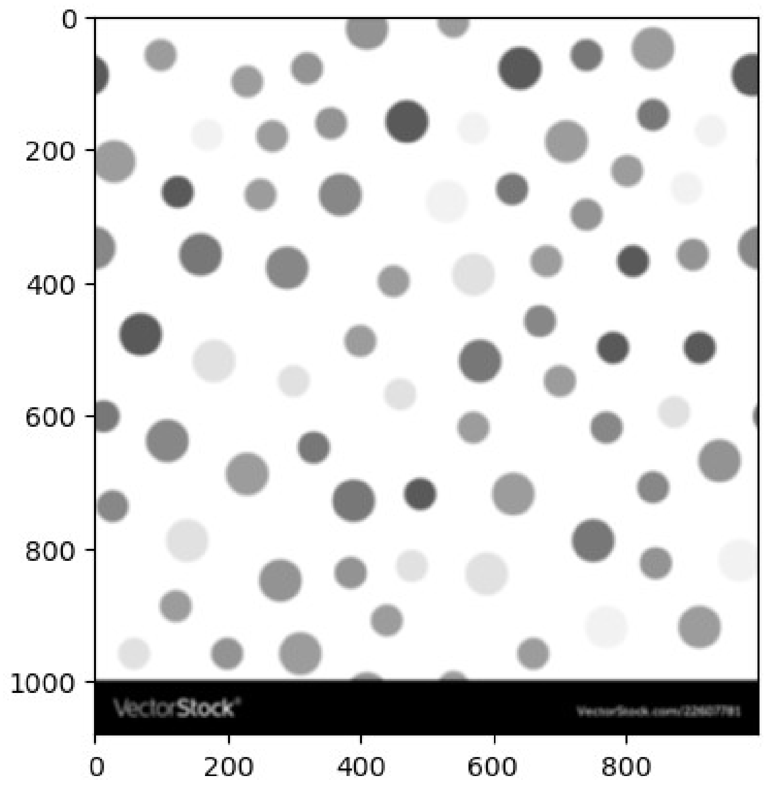


0
200
400
600
800
1000
VectorStock®
VectorStock.com/22607781

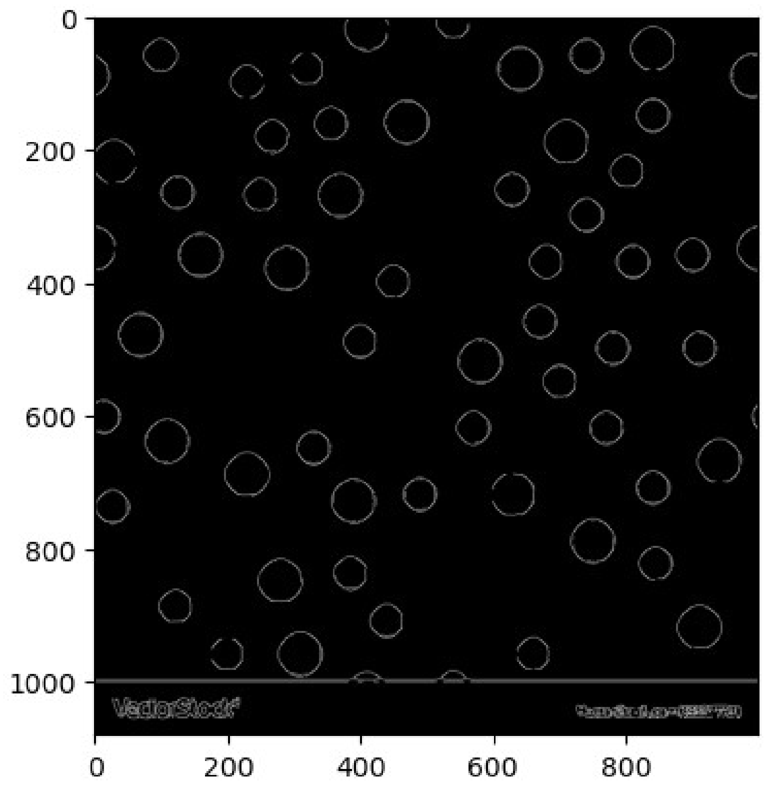


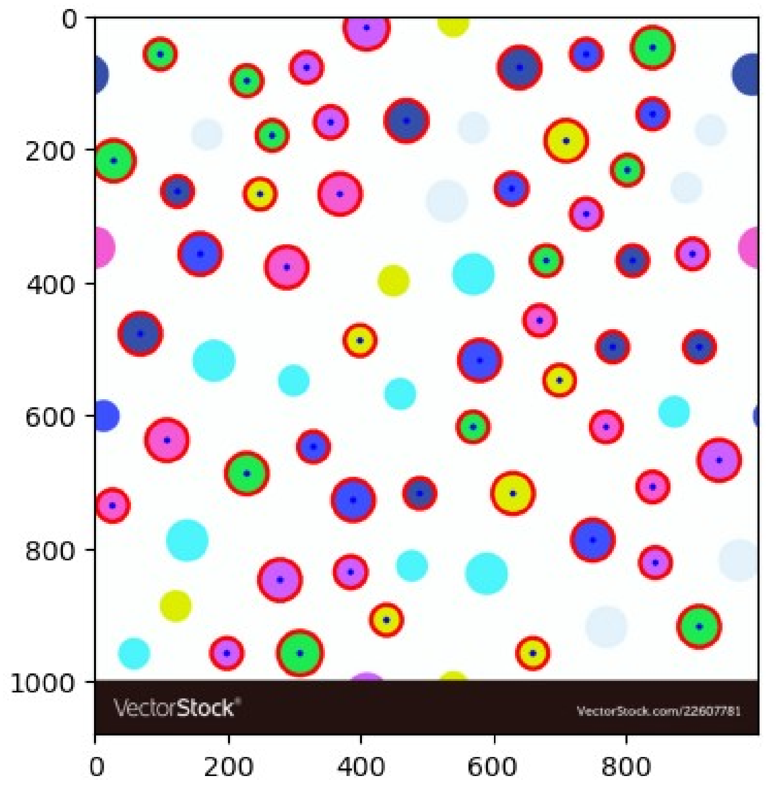
0
200
400
600
800
1000
VectorStock®
VectorStock.com/22607781

```
# blurring the image to identify circles to remove noise using
gaussian filter
blurred_image2 = cv2.GaussianBlur(gray_image2, (7, 7), 13.5)
plt.imshow(blurred_image2, cmap='gray')
plt.show()
```



```
# Apply Canny edge detection to the image to detect the edges
edges = cv2.Canny(blurred_image2, 50, 150)

plt.imshow(edges, cmap='gray')
plt.show()

# as we were to extend our search to 3D parameter space our group has
chosen circle
# where the 3 parameters are x,y,r - (x,y) - coordinates of centre and
r being the radius of circle
circles = cv2.HoughCircles(image=blurred_image2,
method=cv2.HOUGH_GRADIENT, dp=2,
                            minDist=80,minRadius=10, maxRadius=100)

circles = np.uint16(np.around(circles))
# drawing the circles found on the original image
for i in circles[0,:]:
    # draw the outline of circle
    cv2.circle(image2,(i[0],i[1]),i[2],(255,0,0),5)
    # draw the center of the circle
    cv2.circle(image2,(i[0],i[1]),2,(0,0,255),5)
plt.imshow(image2)
plt.show()
```




0
200
400
600
800
1000
0
200
400
600
800
VectorStock®
VectorStock.com/22607781