Coin-Detection-Assignment

August 20, 2019

1 Coin Detection

So far we have studied about various mophological operations and different thresholding techniques in some detail. Now it's time to apply these concepts for a practical application - **Coin Detection**.

1.1 Aim

In this assignment, you will work with 2 different images (so 2 different parts) and will use **only** morphological operations and thresholding techniques to detect the total number of coins present in the image. Your submission will be graded based on your use of the concepts covered in this module, experimentation performed to achieve at your final solution, documentation, and finally, the total number of coins successfully detected in the images. Each part will be of 15 marks. This assignment will be entirely **manually graded** so make sure that you do NOT remove any experimentation you have done as well as the observation you made after each step.

Proper documentation for each step should be provided with help of markdown

1.2 Outline

The main steps that you can follow to solve this assignment are:

1. Read the image.
2. Convert it to grayscale and split the image into the 3 (Red, Green and Blue) channels. Decide which of the above 4 images you want to use in further steps and provide reason for the same.
3. Use thresholding and/or morphological operations to arrive at a final binary image.
4. Use **simple blob detector** to count the number of coins present in the image.
5. Use **contour detection** to count the number of coins present in the image.
6. Use **CCA** to count the number of coins present in the image.

We have also provided the results we obtained at the intermediate steps for your reference.

2 Assignment Part - A

2.1 Step 1: Read Image

```
In [140]: import cv2
          import matplotlib.pyplot as plt
```

```
from dataPath import DATA_PATH
import numpy as np
%matplotlib inline
```

In [141]:
```
import matplotlib
matplotlib.rcParams['figure.figsize'] = (10.0, 10.0)
matplotlib.rcParams['image.cmap'] = 'gray'
```

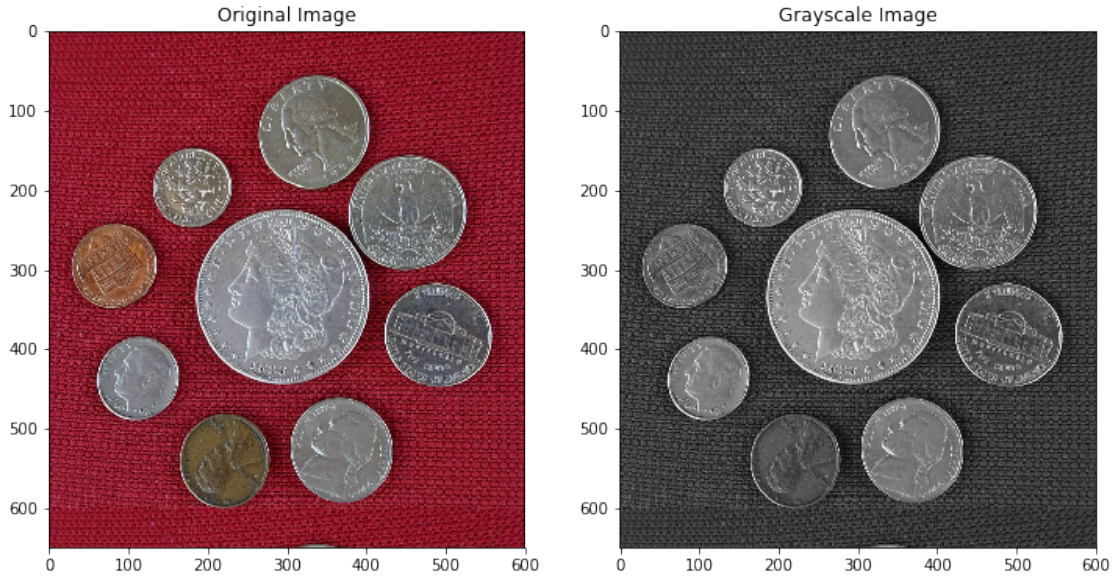
In [142]:
```
# Image path
imagePath = DATA_PATH + "images/CoinsA.png"
# Read image
# Store it in the variable image
###
### YOUR CODE HERE
###
imageCopy = image.copy()
plt.imshow(image[:,:,::-1]);
plt.title("Original Image")
```

Out[142]:
```
Text(0.5,1,'Original Image')
```

2.2 Step 2.1: Convert Image to Grayscale

```
In [143]: # Convert image to grayscale
          # Store it in the variable imageGray
          ###
          ### YOUR CODE HERE
          ###

In [144]: plt.figure(figsize=(12,12))
          plt.subplot(121)
          plt.imshow(image[:,:,::-1]);
```

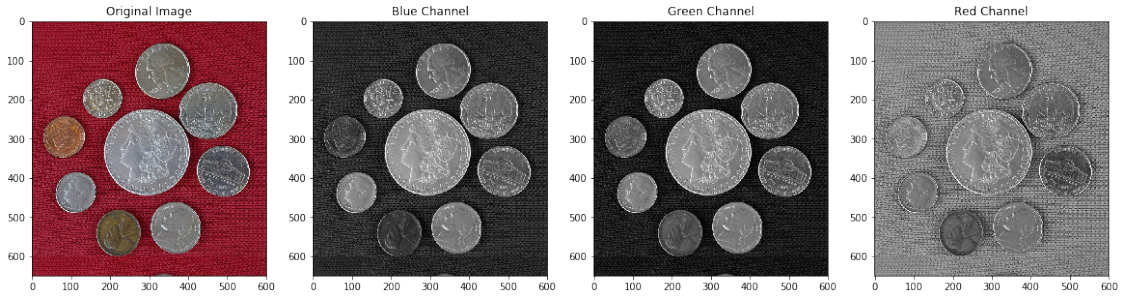
```
plt.title("Original Image")
plt.subplot(122)
plt.imshow(imageGray);
plt.title("Grayscale Image");
```



2.3 Step 2.2: Split Image into R,G,B Channels

In [145]:
```
# Split cell into channels
# Store them in variables imageB, imageG, imageR
###
### YOUR CODE HERE
###
```

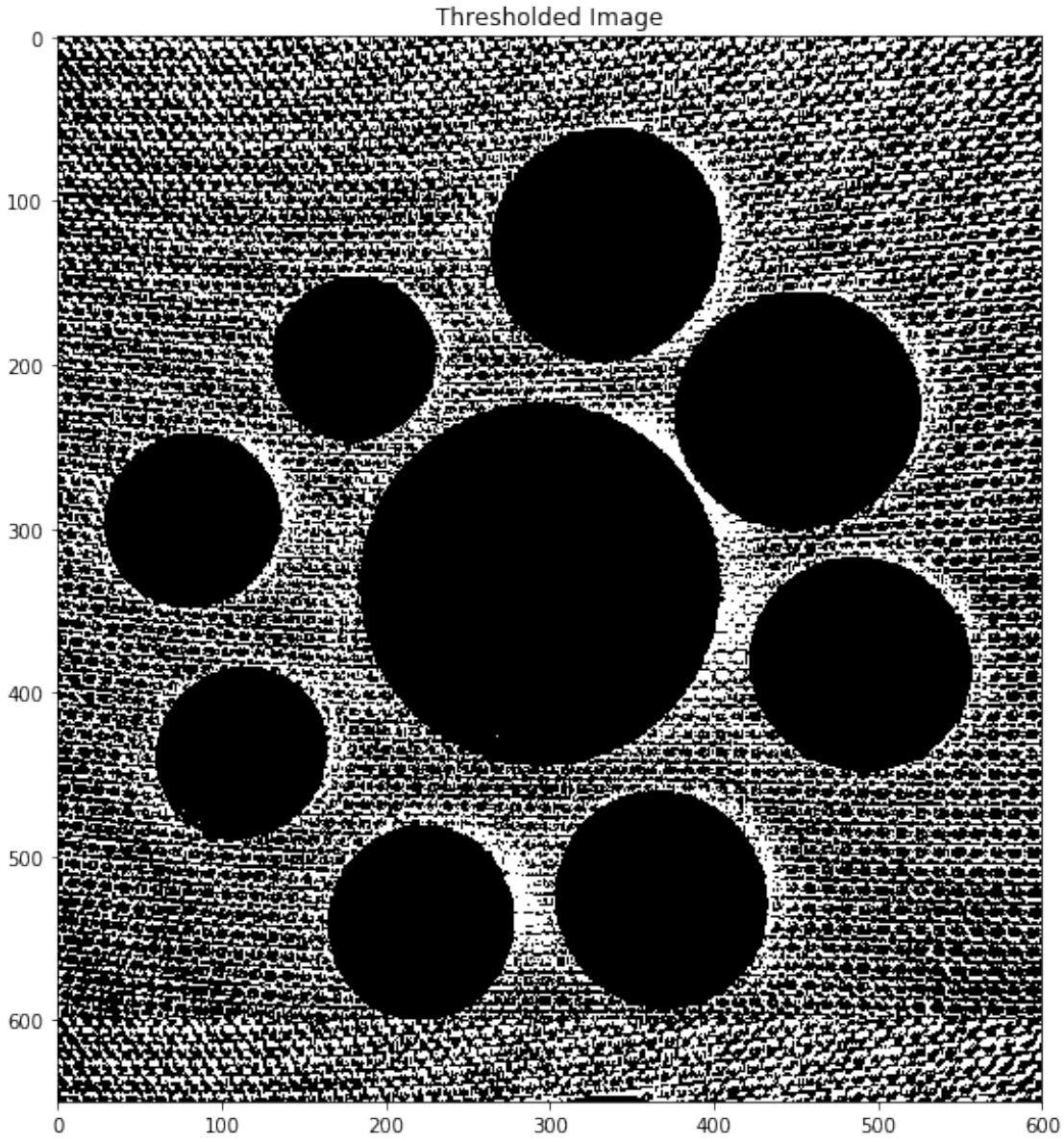
In [146]:
```
plt.figure(figsize=(20,12))
plt.subplot(141)
plt.imshow(image[:,:,::-1]);
plt.title("Original Image")
plt.subplot(142)
plt.imshow(imageB);
plt.title("Blue Channel")
plt.subplot(143)
plt.imshow(imageG);
plt.title("Green Channel")
plt.subplot(144)
plt.imshow(imageR);
plt.title("Red Channel");
```



2.4 Step 3.1: Perform Thresholding

You will have to carry out this step with different threshold values to see which one suits you the most. Do not remove those intermediate images and make sure to document your findings.

```
In [147]: ###
          ### YOUR CODE HERE
          ###

In [148]: # Display the thresholded image
          ###
          ### YOUR CODE HERE
          ###
```



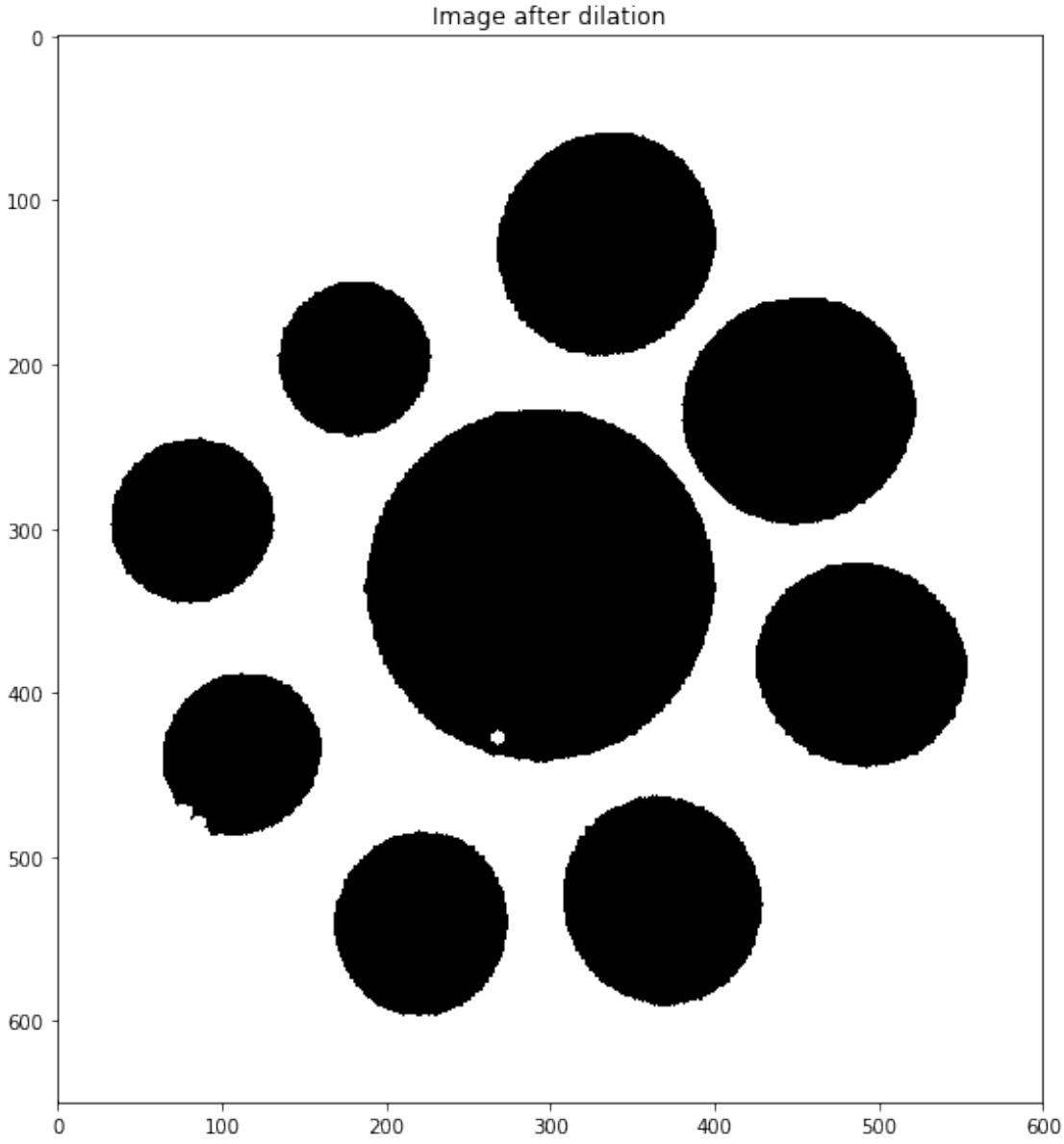
2.5 Step 3.2: Perform morphological operations

You will have to carry out this step with different kernel size, kernel shape and morphological operations to see which one (or more) suits you the most. Do not remove those intermediate images and make sure to document your findings.

```
In [149]: ###
          ### YOUR CODE HERE
          ###

In [150]: ###
```

```
    ### YOUR CODE HERE
    ###
```

In [151]:
```
# Display all the images
# you have obtained in the intermediate steps
###
### YOUR CODE HERE
###
```

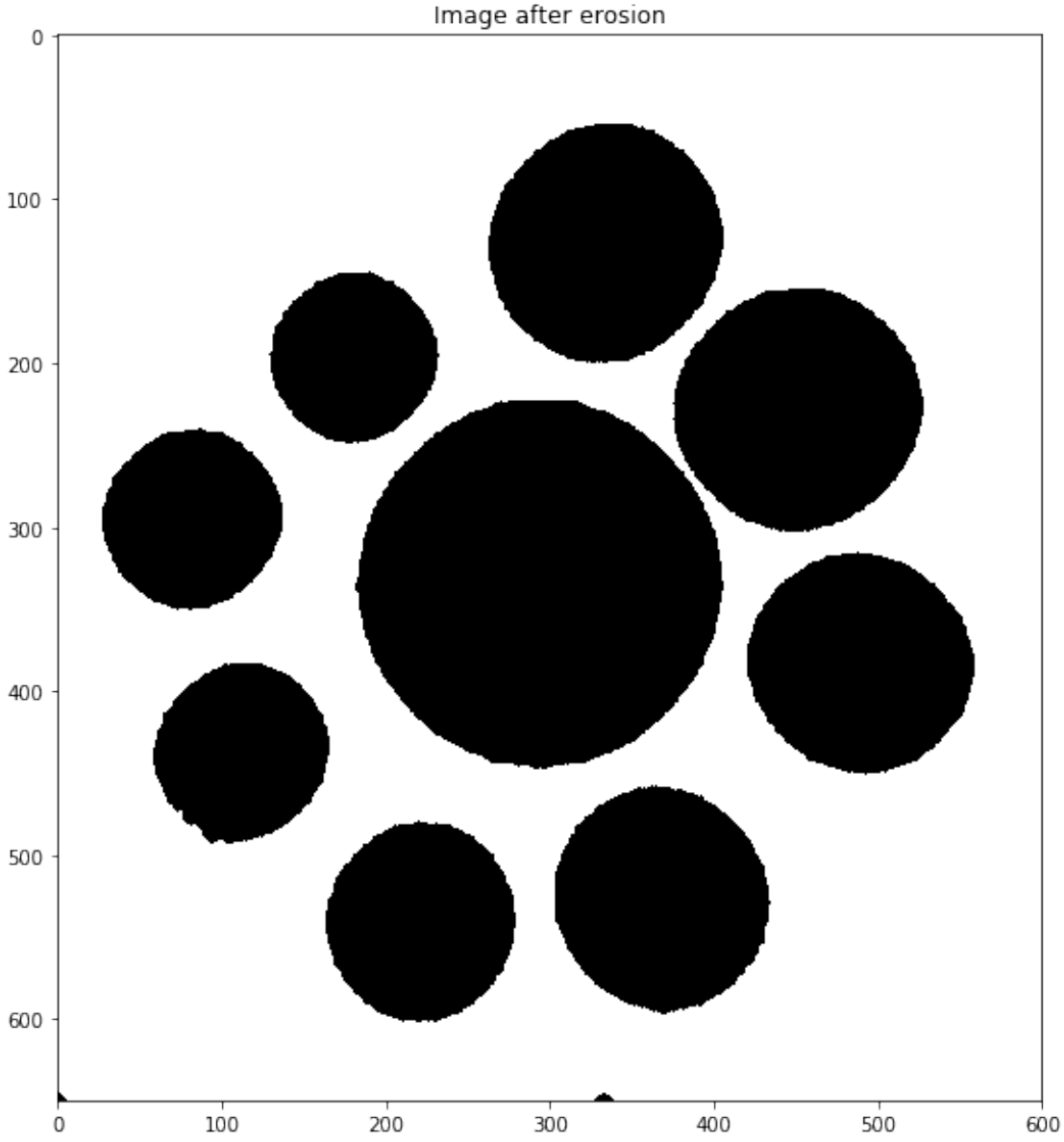


In [152]:
```
# Get structuring element/kernel which will be used for dilation
###
```

```
        ### YOUR CODE HERE
        ###
```

In [153]:
```
        ###
        ### YOUR CODE HERE
        ###
```

In [154]:
```
        ###
        ### YOUR CODE HERE
        ###
```



2.6 Step 4.1: Create SimpleBlobDetector

```
In [155]: # Set up the SimpleBlobdetector with default parameters.
          params = cv2.SimpleBlobDetector_Params()

          params.blobColor = 0

          params.minDistBetweenBlobs = 2

          # Filter by Area.
          params.filterByArea = False

          # Filter by Circularity
          params.filterByCircularity = True
          params.minCircularity = 0.8

          # Filter by Convexity
          params.filterByConvexity = True
          params.minConvexity = 0.8

          # Filter by Inertia
          params.filterByInertia =True
          params.minInertiaRatio = 0.8
```

```
In [156]: # Create SimpleBlobDetector
          detector = cv2.SimpleBlobDetector_create(params)
```

2.7 Step 4.2: Detect Coins

2.7.1 Hints

Use `detector.detect(image)` to detect the blobs (coins). The output of the function is a list of **keypoints** where each keypoint is unique for each blob.

Print the number of coins detected as well.

```
In [157]: # Detect blobs
          ###
          ### YOUR CODE HERE
          ###
```

```
In [158]: # Print number of coins detected
          ###
          ### YOUR CODE HERE
          ###
```

```
Number of coins detected = 9
```

Note that we were able to detect all 9 coins. So, that's your benchmark.

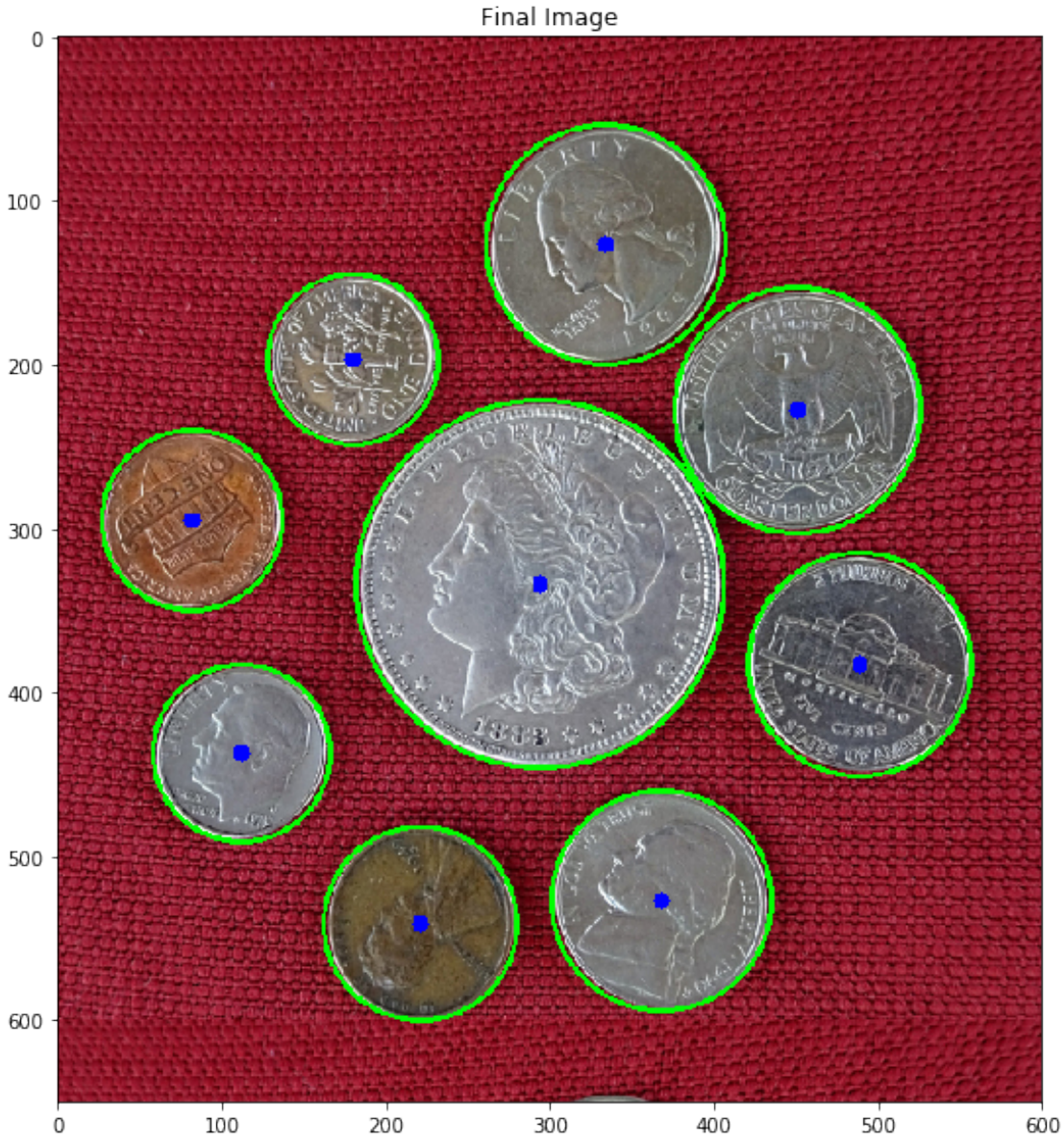
2.8 Step 4.3: Display the detected coins on original image

Make sure to mark the center of the blobs as well. **Use only the functions discussed in Image Annotation section in Week 1**

2.8.1 Hints

You can extract the coordinates of the center and the diameter of a blob using **`k.pt`** and **`k.size`** where `k` is a keypoint.

```
In [159]: # Mark coins using image annotation concepts we have studied so far
          ###
          ### YOUR CODE HERE
          ###

In [160]: # Display the final image
          ###
          ### YOUR CODE HERE
          ###
```



2.9 Step 4.4: Perform Connected Component Analysis

In the final step, perform Connected Component Analysis (CCA) on the binary image to find out the number of connected components. Do you think we can use CCA to calculate number of coins? Why/why not?

```
In [161]: def displayConnectedComponents(im):
              imLabels = im
              # The following line finds the min and max pixel values
              # and their locations in an image.
              (minVal, maxVal, minLoc, maxLoc) = cv2.minMaxLoc(imLabels)
```

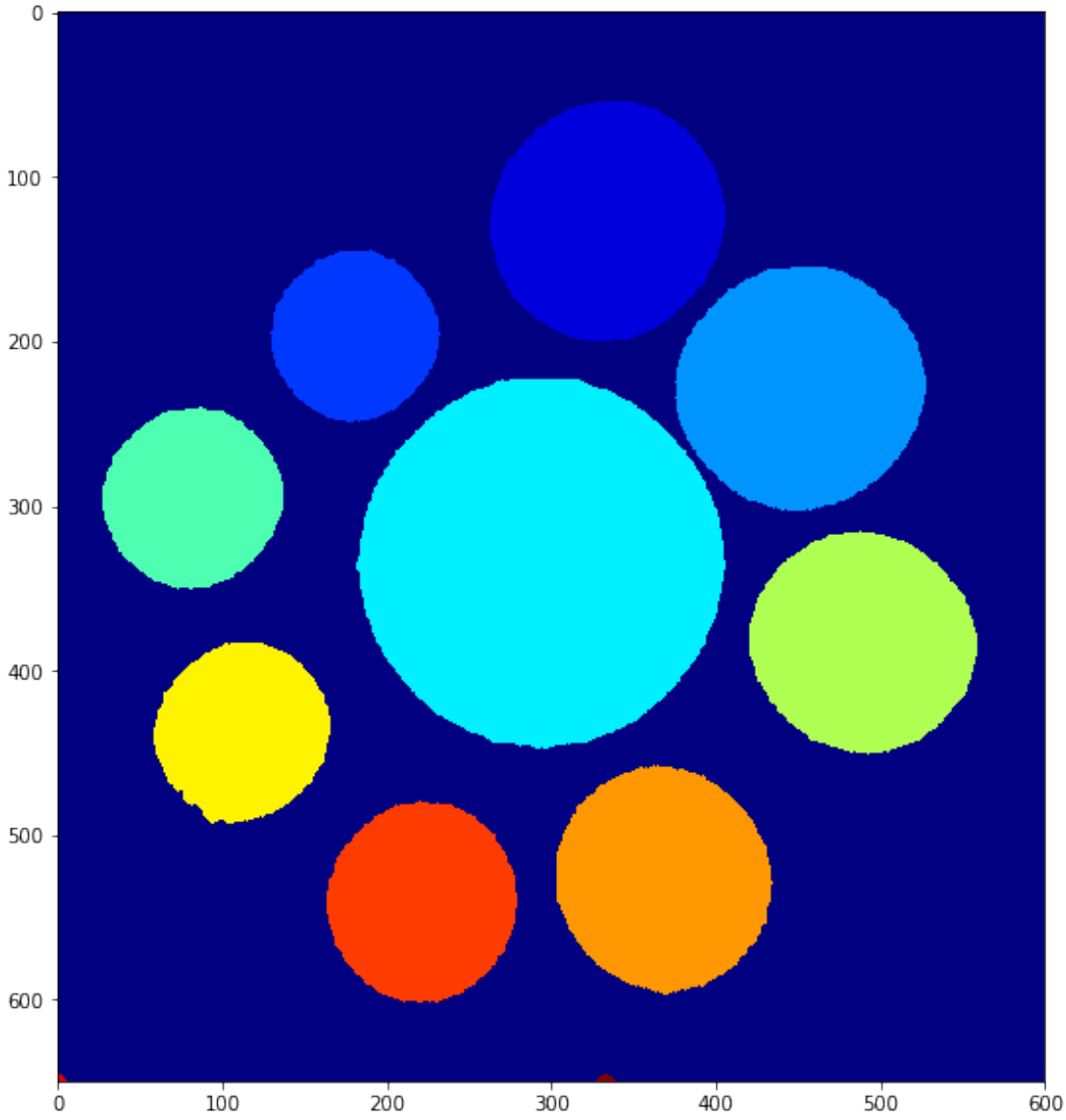
```
    # Normalize the image so the min value is 0 and max value is 255.
    imLabels = 255 * (imLabels - minVal)/(maxVal-minVal)
    # Convert image to 8-bits unsigned type
    imLabels = np.uint8(imLabels)
    # Apply a color map
    imColorMap = cv2.applyColorMap(imLabels, cv2.COLORMAP_JET)
    # Display colormapped labels
    plt.imshow(imColorMap[:,:,::-1])
```

In [162]:
```
# Find connected components
###
### YOUR CODE HERE
###
```

In [163]:
```
# Print number of connected components detected
###
### YOUR CODE HERE
###
```

```
Number of connected components detected = 12
```

In [164]:
```
# Display connected components using displayConnectedComponents
# function
###
### YOUR CODE HERE
###
```



2.10 Step 4.5: Detect coins using Contour Detection

In the final step, perform Contour Detection on the binary image to find out the number of coins.

```
In [165]: # Find all contours in the image
          ###
          ### YOUR CODE HERE
          ###

In [166]: # Print the number of contours found
          ###
```

```
        ### YOUR CODE HERE
        ###
```

```
Number of contours found = 10
```

```
In [167]: # Draw all contours
          ###
          ### YOUR CODE HERE
          ###
```

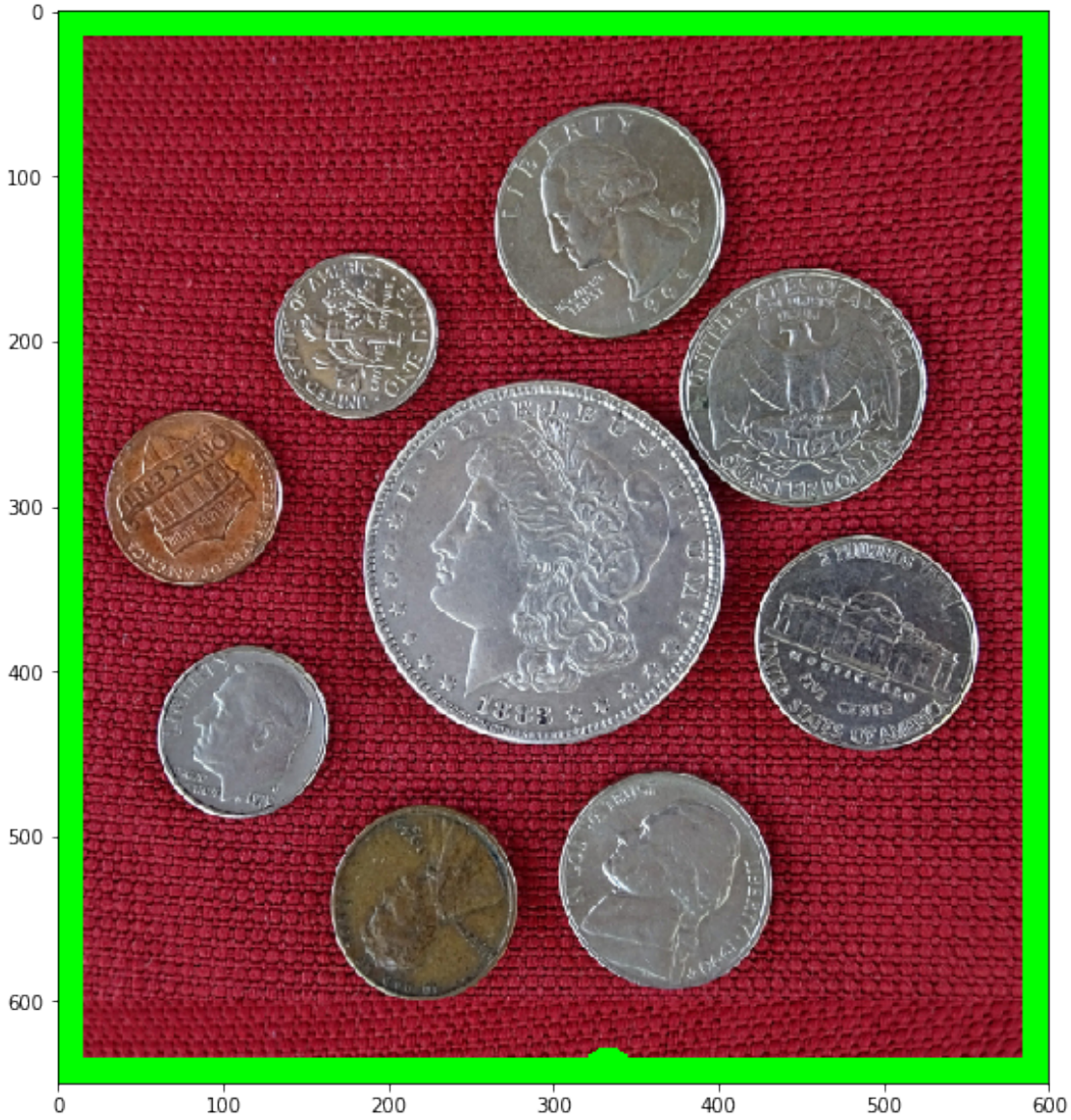
```
Out[167]: <matplotlib.image.AxesImage at 0x7fcf111c7748>
```

Let's only consider the outer contours.

```
In [168]: # Remove the inner contours
          # Display the result
          ###
          ### YOUR CODE HERE
          ###
```

```
Number of contours found = 1
```

```
Out[168]: <matplotlib.image.AxesImage at 0x7fcf1162d080>
```



So, we only need the inner contours. The easiest way to do that will be to remove the outer contour using area.

In [169]:
```
# Print area and perimeter of all contours
###
### YOUR CODE HERE
###
```

```
Contour #1 has area = 11118.0 and perimeter = 401.9310212135315
Contour #2 has area = 14133.5 and perimeter = 456.9432144165039
Contour #3 has area = 9075.5 and perimeter = 373.2030990123749
Contour #4 has area = 14482.5 and perimeter = 456.8010768890381
Contour #5 has area = 9404.5 and perimeter = 367.5462449789047
Contour #6 has area = 39121.5 and perimeter = 748.3645672798157
Contour #7 has area = 17612.0 and perimeter = 500.8427075147629
Contour #8 has area = 8298.5 and perimeter = 344.5756813287735
Contour #9 has area = 16370.5 and perimeter = 484.11478412151337
Contour #10 has area = 388697.0 and perimeter = 2497.4558429718018
```

In [170]:
```
# Print maximum area of contour
# This will be the box that we want to remove
###
### YOUR CODE HERE
###
```

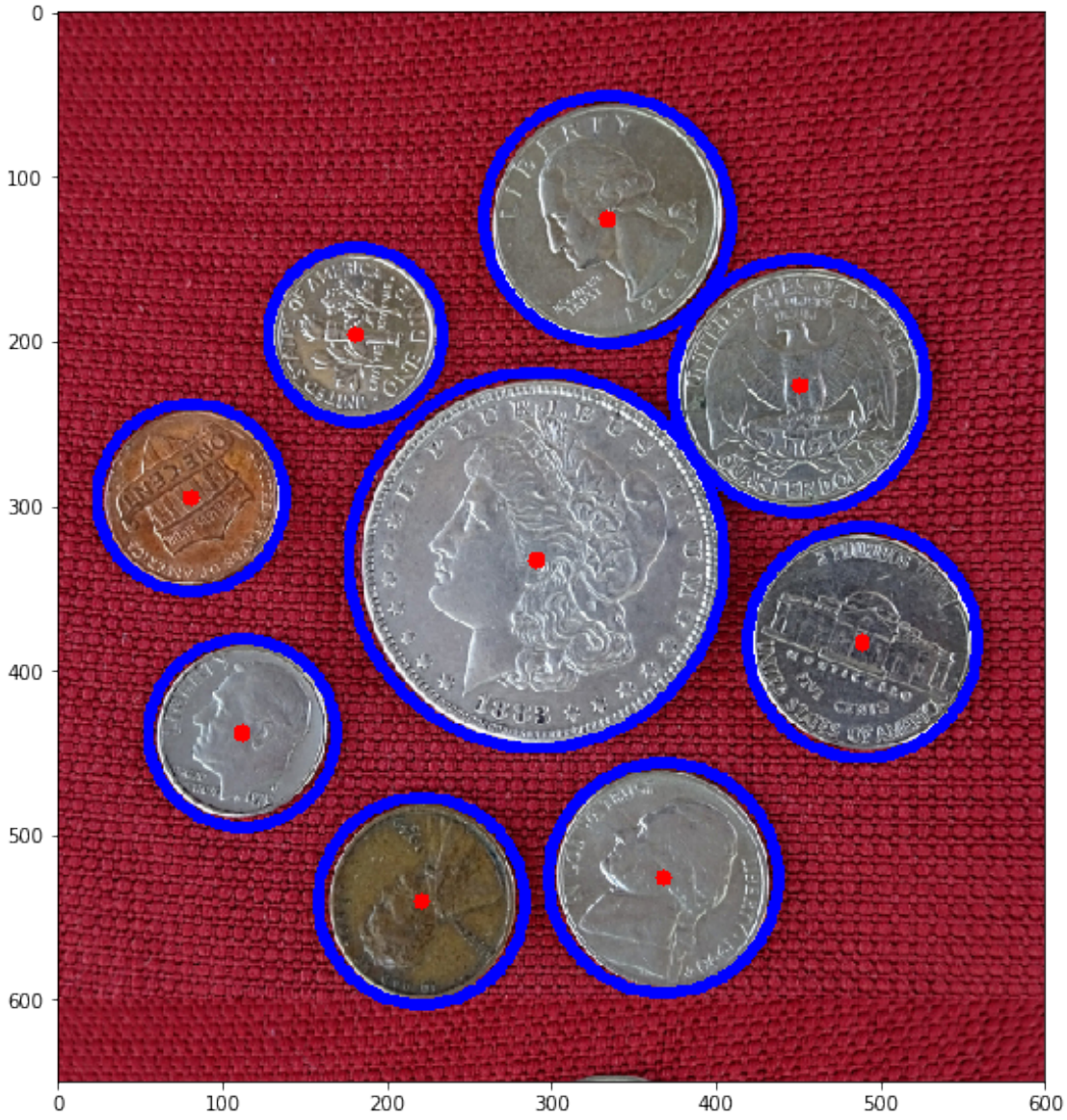
```
Maximum area of contour = 388697.0
```

In [171]:
```
# Remove this contour and plot others
###
### YOUR CODE HERE
###
```

Out[171]:
```
<matplotlib.image.AxesImage at 0x7fcf113b14e0>
```


```
In [173]: # Fit circles on coins
          ###
          ### YOUR CODE HERE
          ###
```

```
Number of coins detected = 9
```



3 Assignment Part - B

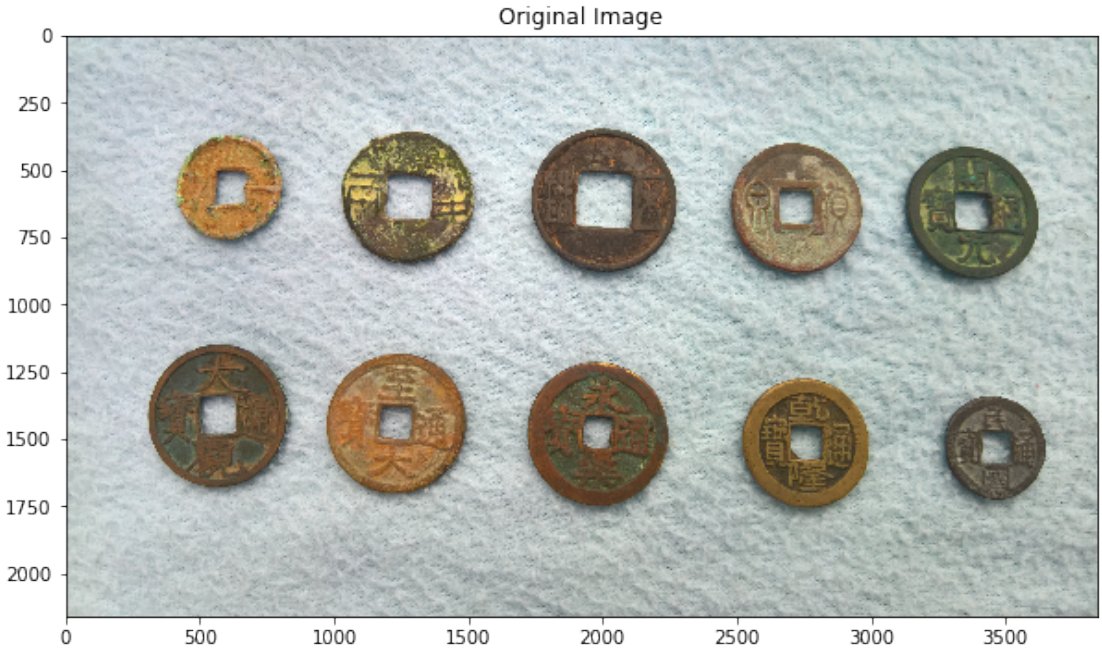
Follow the same steps as provided in Assignment Part - A

3.1 Step 1: Read Image

```
In [30]: # Image path
         imagePath = DATA_PATH + "images/CoinsB.png"
         # Read image
         # Store it in variable image
         ###
```

```
### YOUR CODE HERE
###
plt.imshow(image[:,:,::-1]);
plt.title("Original Image")
```

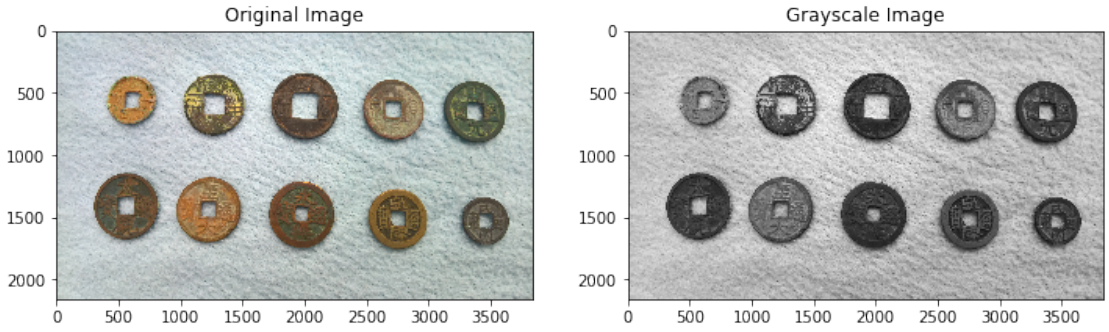
```
Out[30]: Text(0.5,1,'Original Image')
```



3.2 Step 2.1: Convert Image to Grayscale

```
In [31]: # Convert to grayscale
         # Store in variable imageGray
         ###
         ### YOUR CODE HERE
         ###
```

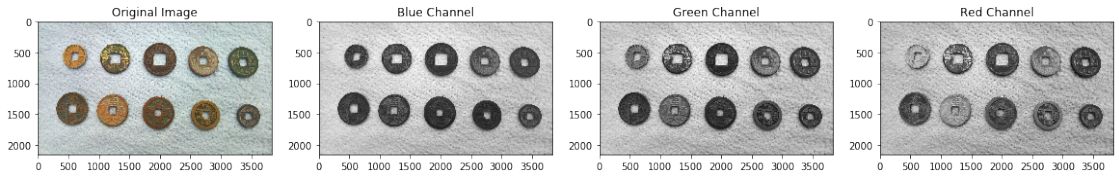
```
In [32]: plt.figure(figsize=(12,12))
         plt.subplot(121)
         plt.imshow(image[:,:,::-1]);
         plt.title("Original Image")
         plt.subplot(122)
         plt.imshow(imageGray);
         plt.title("Grayscale Image");
```



3.3 Step 2.2: Split Image into R,G,B Channels

```
In [33]: # Split cell into channels
         # Variables are: imageB, imageG, imageR
         ###
         ### YOUR CODE HERE
         ###
```

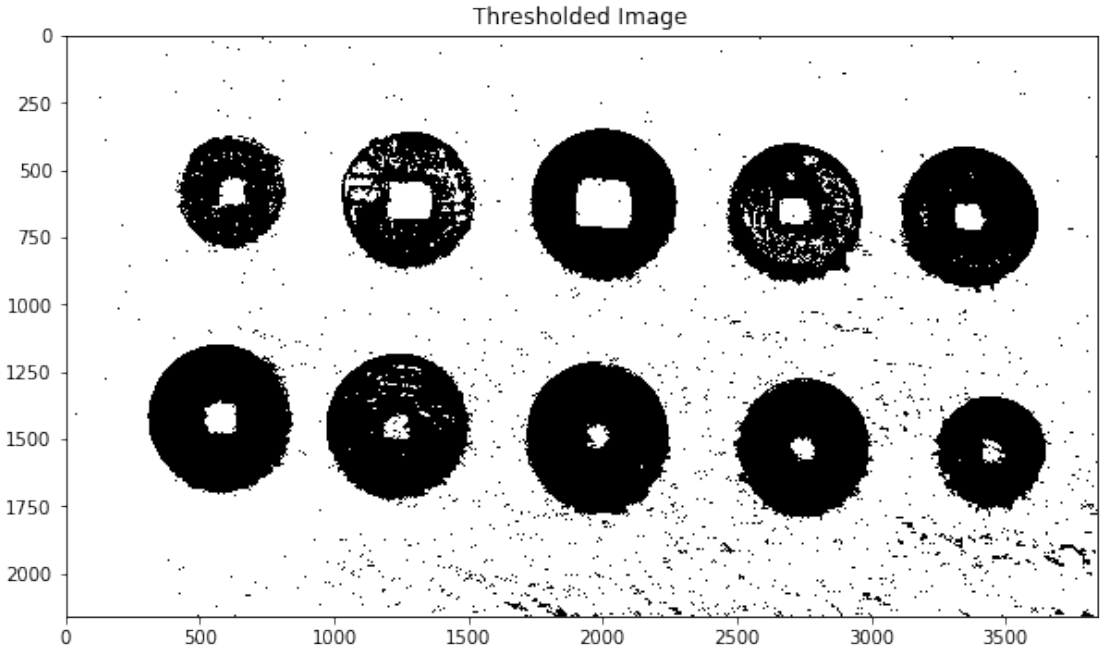
```
In [34]: plt.figure(figsize=(20,12))
         plt.subplot(141)
         plt.imshow(image[:,:,::-1]);
         plt.title("Original Image")
         plt.subplot(142)
         plt.imshow(imageB);
         plt.title("Blue Channel")
         plt.subplot(143)
         plt.imshow(imageG);
         plt.title("Green Channel")
         plt.subplot(144)
         plt.imshow(imageR);
         plt.title("Red Channel");
```



3.4 Step 3.1: Perform Thresholding

You will have to carry out this step with different threshold values to see which one suits you the most. Do not remove those intermediate images and make sure to document your findings.

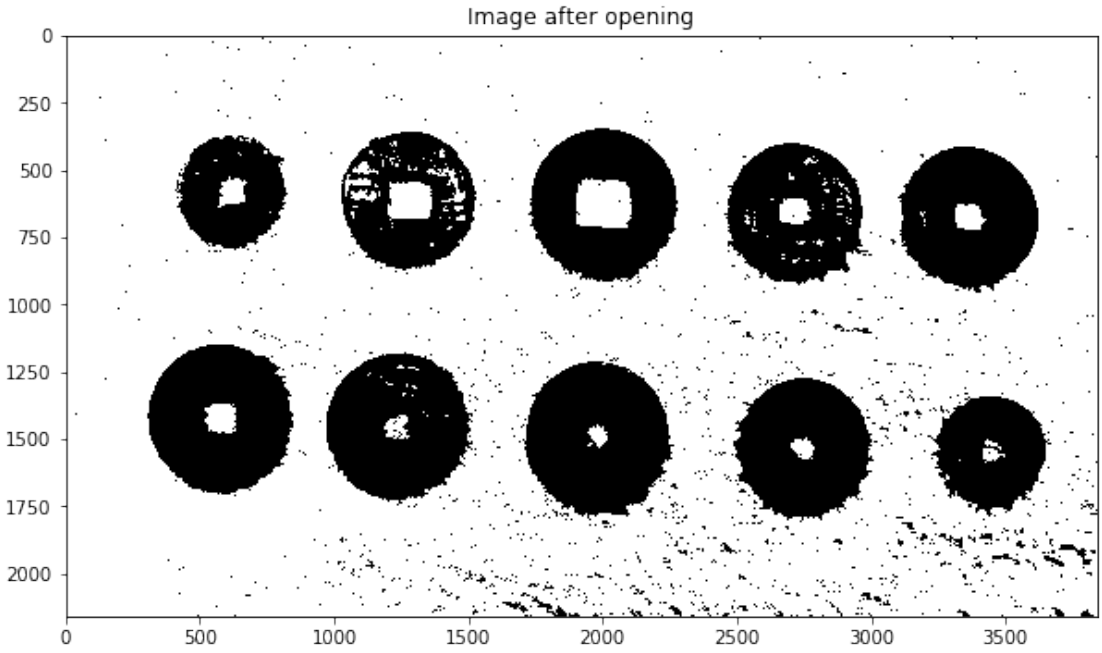
```
In [35]: ###
         ### YOUR CODE HERE
         ###

In [36]: # Display image using matplotlib
         ###
         ### YOUR CODE HERE
         ###
```



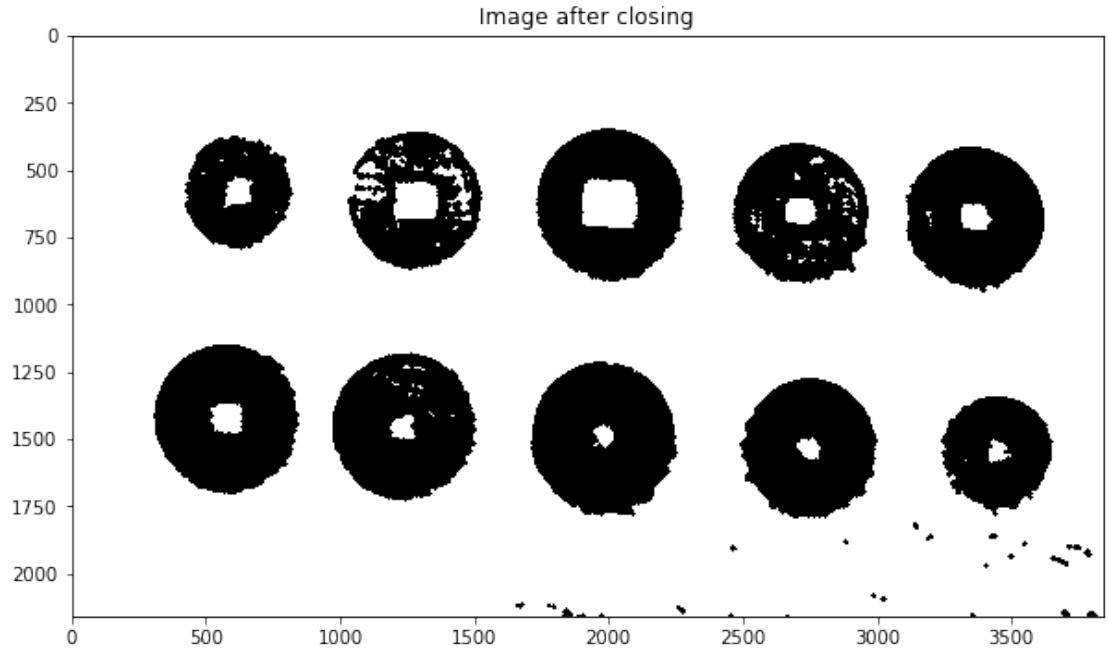
3.5 Step 3.2: Perform morphological operations

You will have to carry out this step with different kernel size, kernel shape and morphological operations to see which one (or more) suits you the most. Do not remove those intermediate images and make sure to document your findings.

```
In [37]: ###
         ### YOUR CODE HERE
         ###

In [38]: ###
         ### YOUR CODE HERE
         ###

In [39]: ###
         ### YOUR CODE HERE
         ###
```



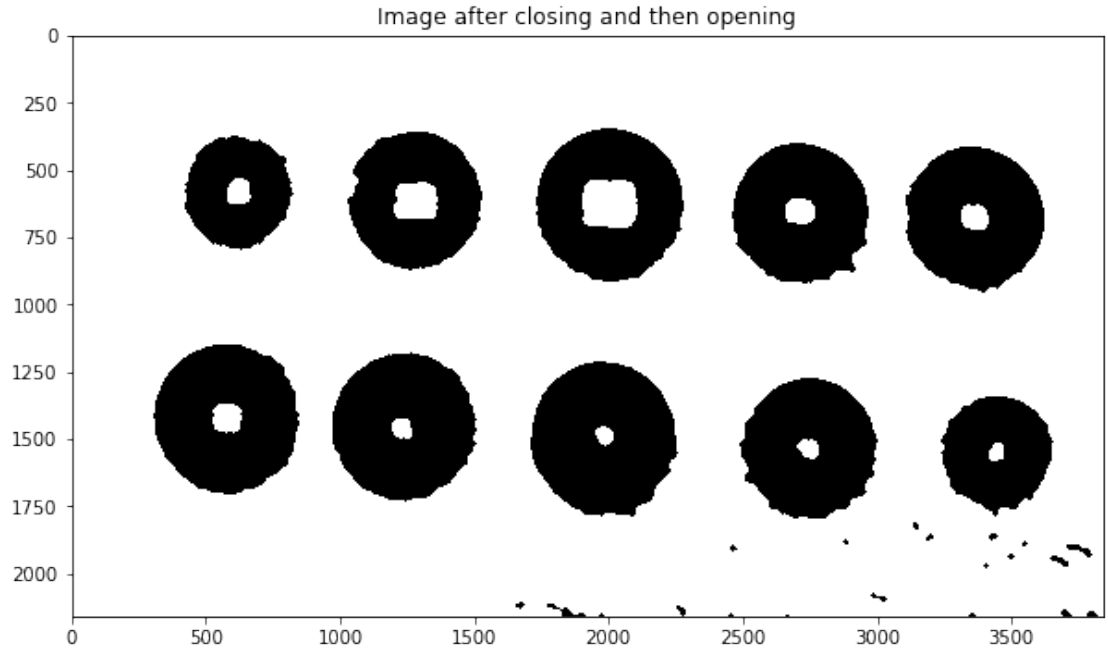

```
In [40]: ###
         ### YOUR CODE HERE
         ###

In [41]: ###
         ### YOUR CODE HERE
         ###

In [42]: ###
         ### YOUR CODE HERE
         ###
```



In [43]:
```
###
### YOUR CODE HERE
###
```

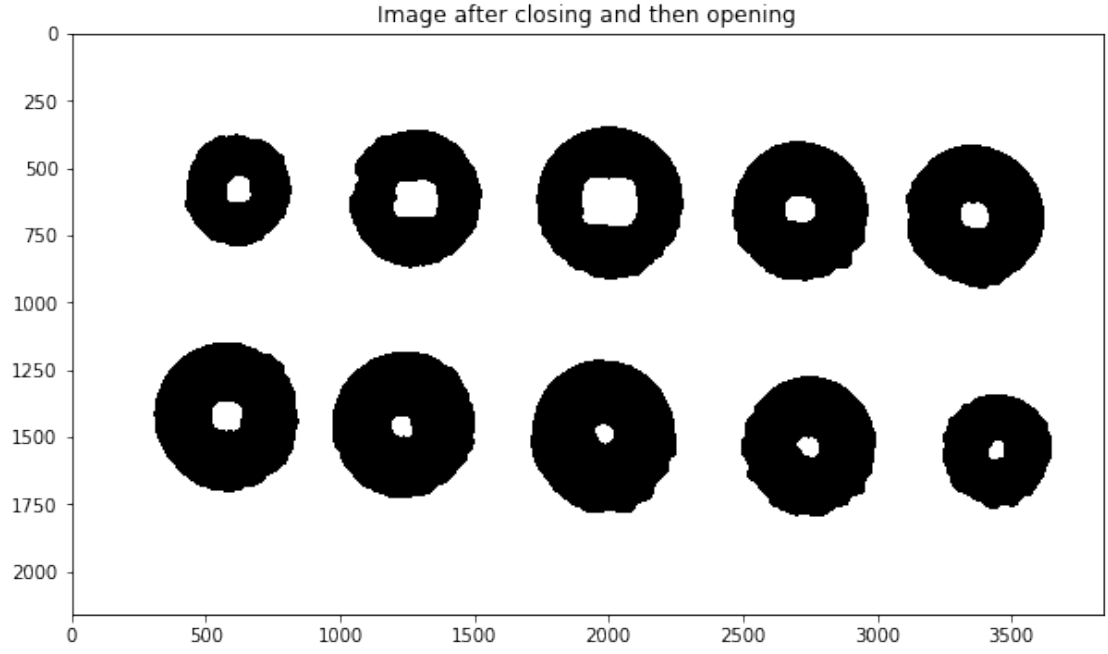
In [44]:
```
###
### YOUR CODE HERE
###
```

In [45]:
```
###
### YOUR CODE HERE
###
```



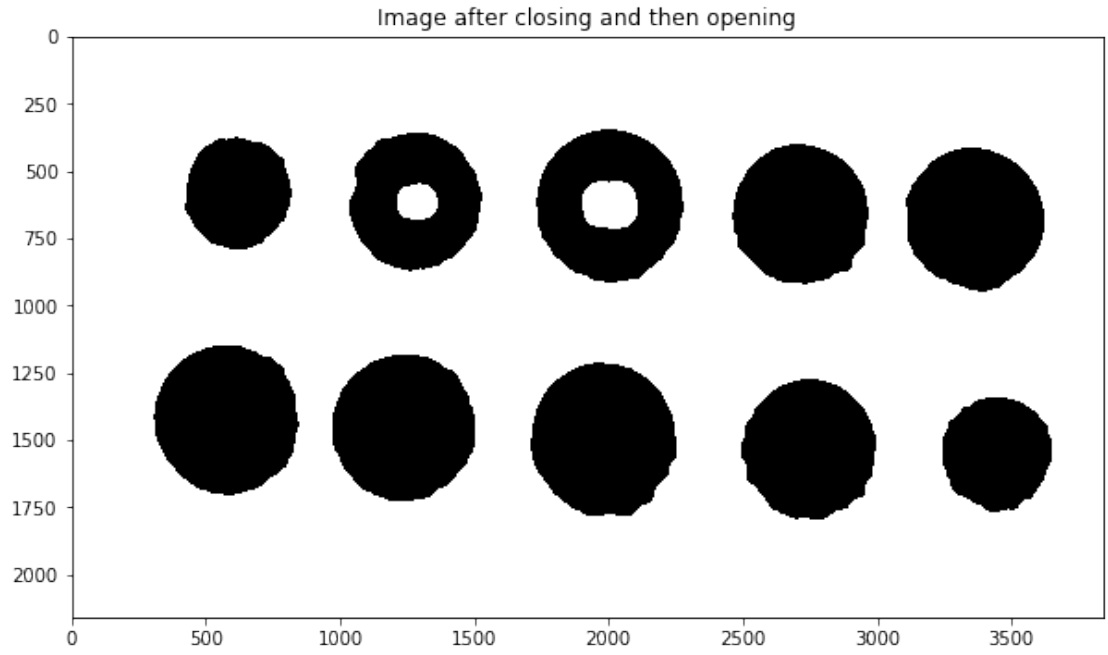
In [46]:
```
###
### YOUR CODE HERE
###
```

In [47]:
```
###
### YOUR CODE HERE
###
```

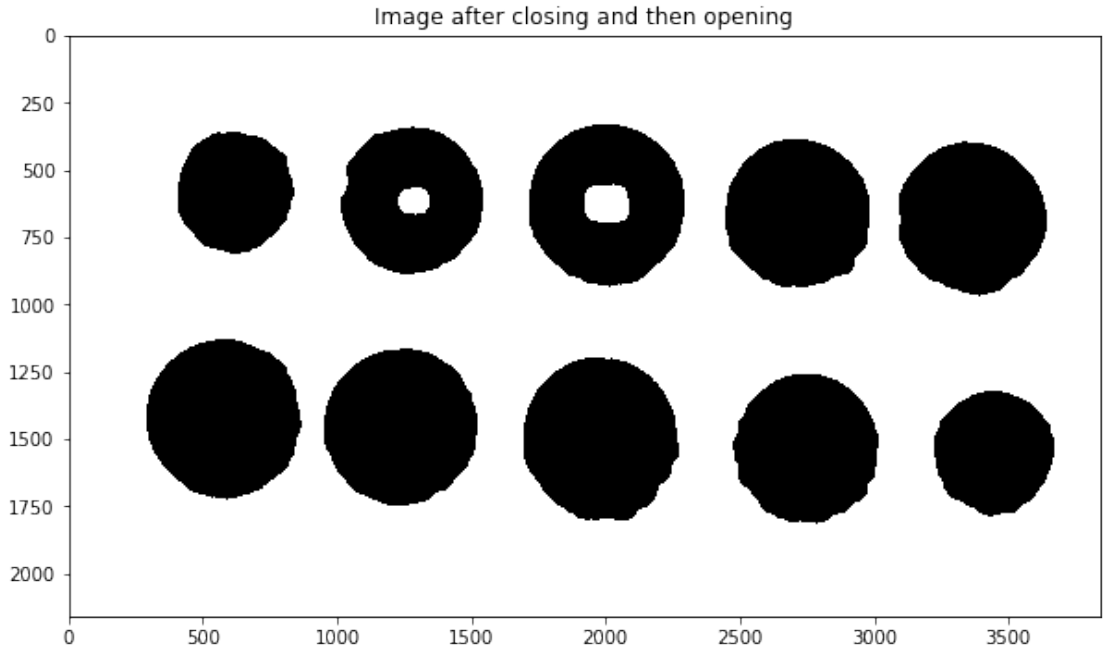


In [48]:
```
###
### YOUR CODE HERE
###
```

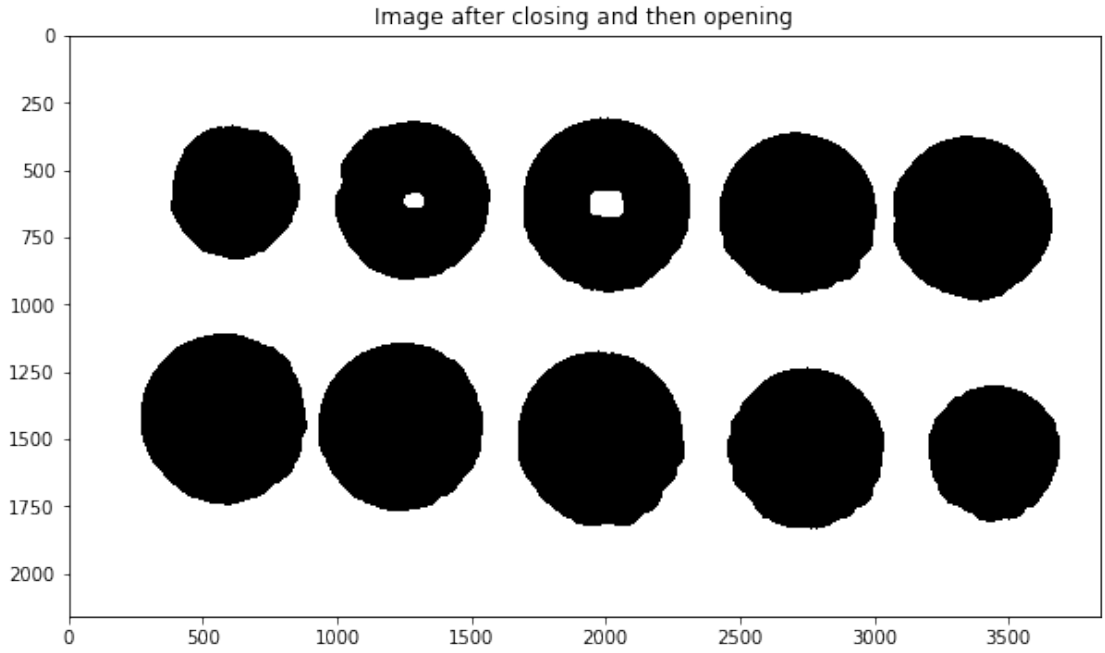
In [49]:
```
###
### YOUR CODE HERE
###
```



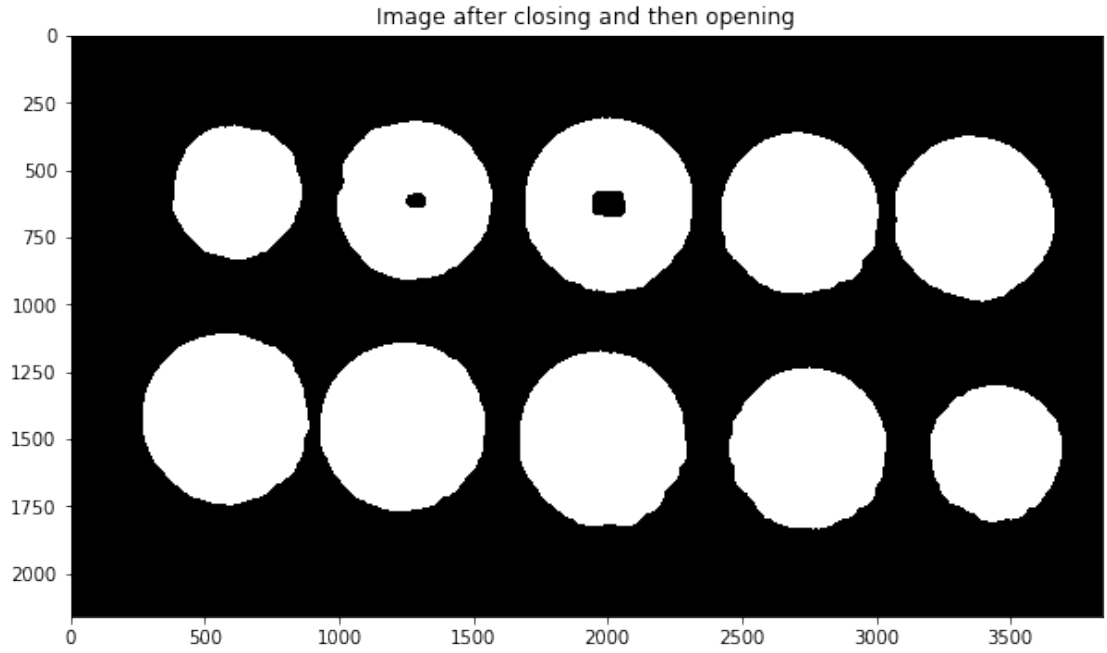
In [50]:
```
###
### YOUR CODE HERE
###
```



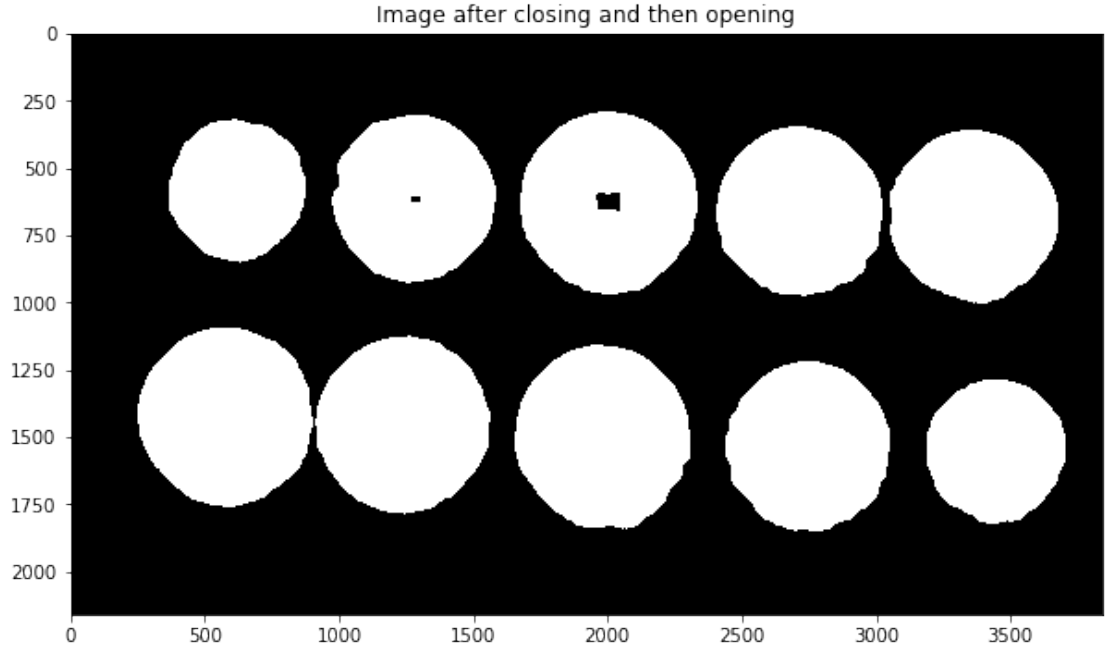
In [51]:
```
###
### YOUR CODE HERE
###
```



In [52]:
```
###
### YOUR CODE HERE
###
```



In [53]:
```
###
### YOUR CODE HERE
###
```



3.6 Step 4.1: Create SimpleBlobDetector

```
In [54]: # Set up the SimpleBlobdetector with default parameters.
         params = cv2.SimpleBlobDetector_Params()

         params.blobColor = 0

         params.minDistBetweenBlobs = 2

         # Filter by Area.
         params.filterByArea = False

         # Filter by Circularity
         params.filterByCircularity = True
         params.minCircularity = 0.8

         # Filter by Convexity
         params.filterByConvexity = True
         params.minConvexity = 0.8

         # Filter by Inertia
         params.filterByInertia =True
         params.minInertiaRatio = 0.8
```

```
In [55]: # Create SimpleBlobDetector
         detector = cv2.SimpleBlobDetector_create(params)
```

3.7 Step 4.2: Detect Coins

3.7.1 Hints

Use `detector.detect(image)` to detect the blobs (coins). The output of the function is a list of **keypoints** where each keypoint is unique for each blob.

Print the number of coins detected as well.

```
In [56]: # Detect blobs
         ###
         ### YOUR CODE HERE
         ###
```

```
Number of coins detected = 8
```

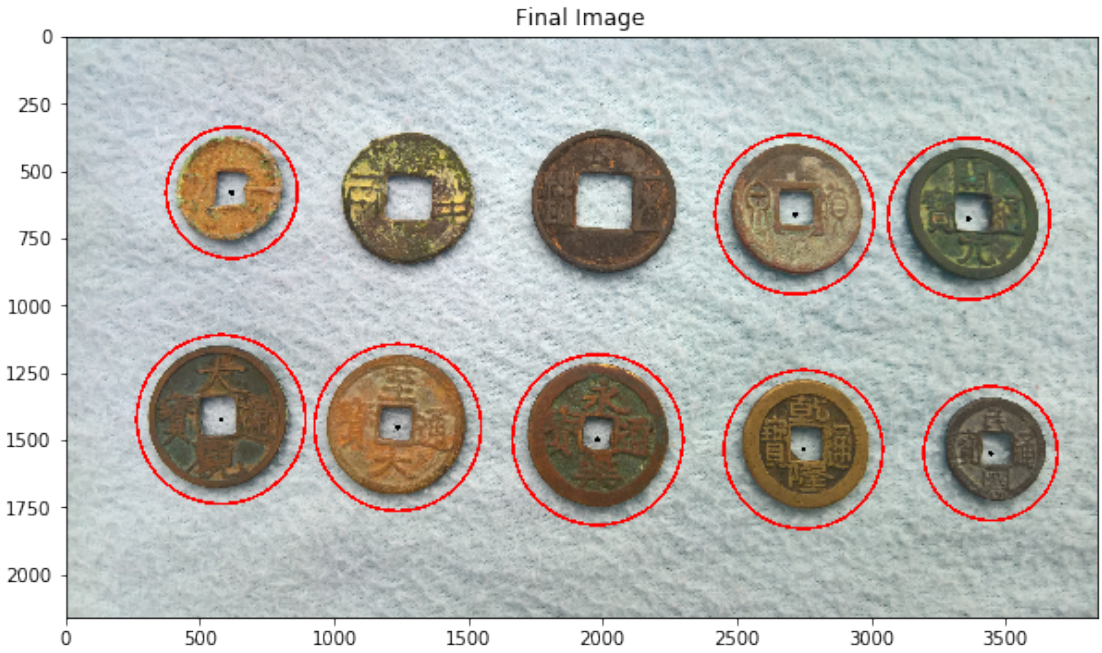
3.8 Step 4.3: Display the detected coins on original image

Make sure to mark the center of the blobs as well. **Use only the functions discussed in Image Annotation section in Week 1**

3.8.1 Hints

You can extract the coordinates of the center and the diameter of a blob using `k.pt` and `k.size` where `k` is a keypoint.

```
In [57]: ###
         ### YOUR CODE HERE
         ###

In [58]: ###
         ### YOUR CODE HERE
         ###

In [59]: ###
         ### YOUR CODE HERE
         ###
```



Note that we were able to detect 8 coins. So, that's your benchmark.

3.9 Step 4.4: Perform Connected Component Analysis

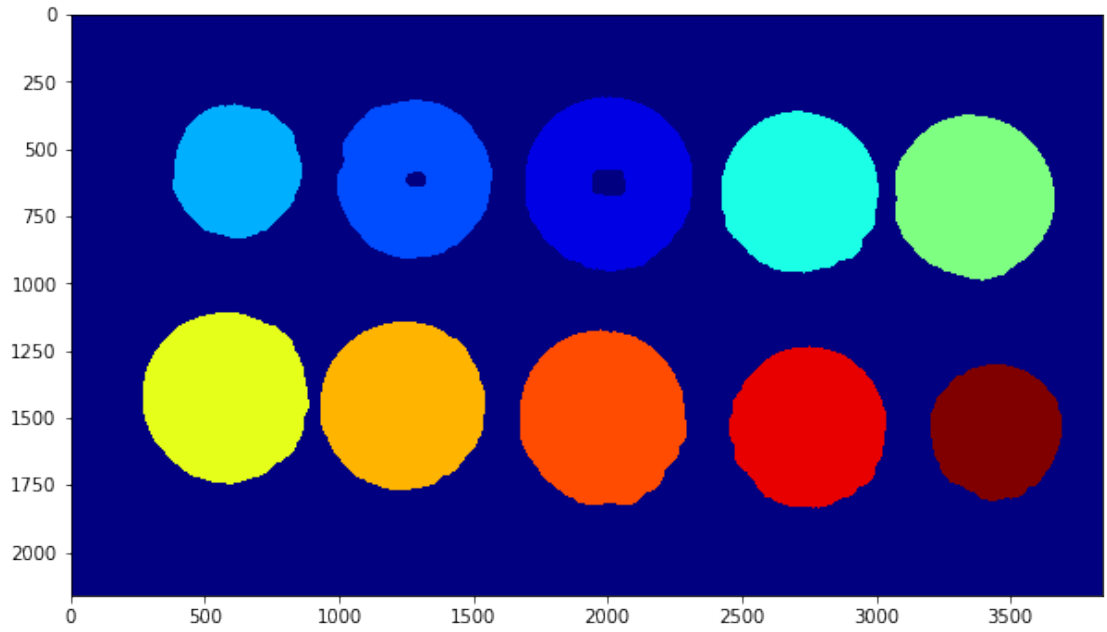
Now, let's perform Connected Component Analysis (CCA) on the binary image to find out the number of connected components. Do you think we can use CCA to calculate number of coins? Why/why not?

```
In [60]: ###
         ### YOUR CODE HERE
         ###
```

```
In [61]: ###
         ### YOUR CODE HERE
         ###
```

```
Number of connected components detected = 11
```

```
In [62]: ###
         ### YOUR CODE HERE
         ###
```



3.10 Step 4.5: Detect coins using Contour Detection

In the final step, perform Contour Detection on the binary image to find out the number of coins.

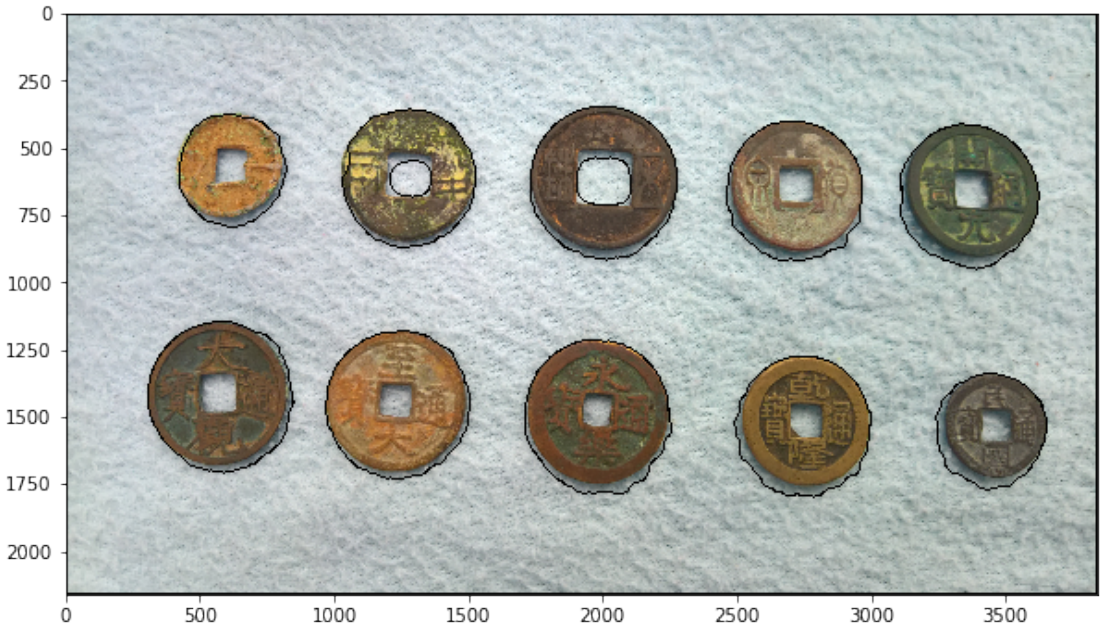
```
In [63]: # Find all contours in the image
         ###
         ### YOUR CODE HERE
         ###
```

```
In [64]: # Print the number of contours found
         ###
         ### YOUR CODE HERE
         ###
```

```
Number of contours found = 13
```

In [65]:
```
# Draw all contours
###
### YOUR CODE HERE
###
```

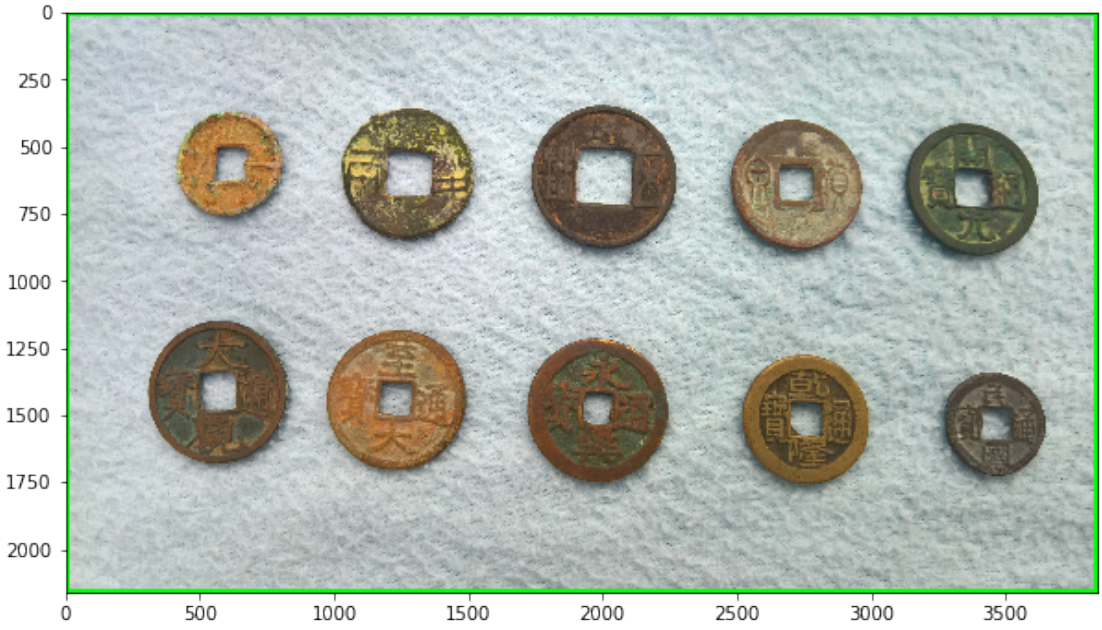
Out[65]: <matplotlib.image.AxesImage at 0x7fcf1191a9b0>



In [66]:
```
# Remove the inner contours
# Display the result
###
### YOUR CODE HERE
###
```

```
Number of contours found = 1
```

Out[66]: <matplotlib.image.AxesImage at 0x7fcf11921710>



What do you think went wrong? As we can see, the outer box was detected as a contour and with respect to it, all other contours are internal and that's why they were not detected. How do we remove that? Let's see if we can use area of contours here.

```
In [67]: # Print area and perimeter of all contours
         ###
         ### YOUR CODE HERE
         ###
```

```
Contour #1 has area = 135398.0 and perimeter = 1401.449918627739
Contour #2 has area = 203535.5 and perimeter = 1720.5697610378265
Contour #3 has area = 242100.5 and perimeter = 1871.0794281959534
Contour #4 has area = 229459.0 and perimeter = 1813.3809397220612
Contour #5 has area = 235121.5 and perimeter = 1837.1382957696915
Contour #6 has area = 16605.5 and perimeter = 485.2031031847
Contour #7 has area = 30872.0 and perimeter = 666.9015874862671
Contour #8 has area = 213446.5 and perimeter = 1743.339308977127
Contour #9 has area = 207230.0 and perimeter = 1723.2144236564636
Contour #10 has area = 130007.0 and perimeter = 1371.1067732572556
Contour #11 has area = 197785.0 and perimeter = 1696.1261086463928
Contour #12 has area = 243313.0 and perimeter = 1854.836783528328
Contour #13 has area = 8288401.0 and perimeter = 11996.0
```

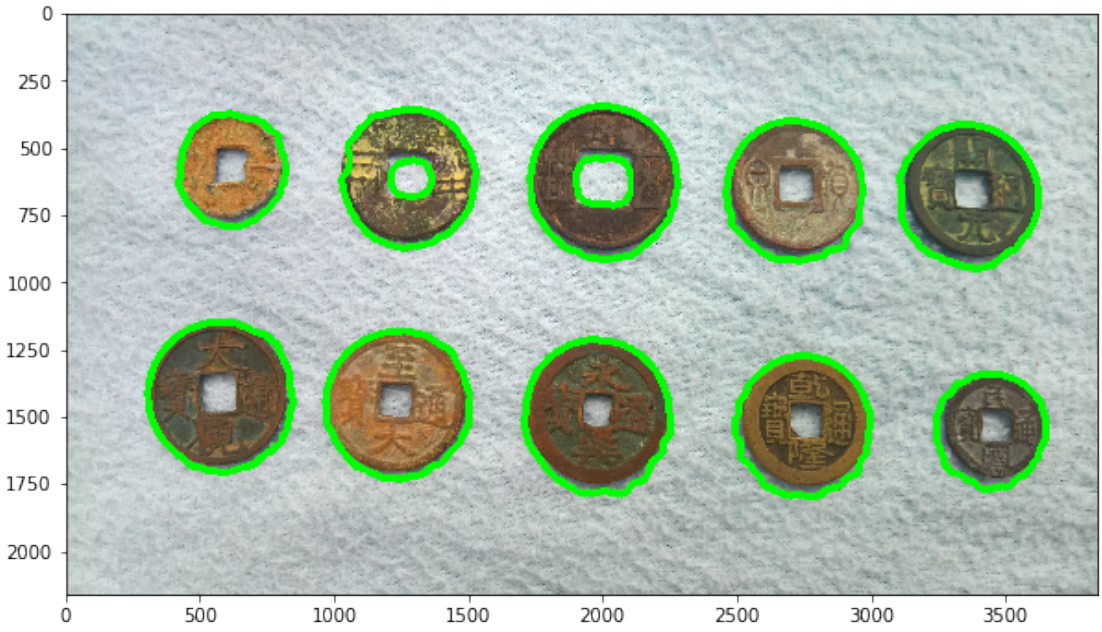
```
In [68]: # Print maximum area of contour
         # This will be the box that we want to remove
         ###
```

```
        ### YOUR CODE HERE
        ###
```

```
Maximum area of contour = 8288401.0
```

```
In [69]: # Remove this contour and plot others
         ###
         ### YOUR CODE HERE
         ###
```

```
Out[69]: <matplotlib.image.AxesImage at 0x7fcf194117b8>
```



Now, we have to remove the internal contours. Again here we can use area or perimeter.

```
In [70]: # Print sorted area of contours
         ###
         ### YOUR CODE HERE
         ###
```

```
16605.5
30872.0
130007.0
135398.0
197785.0
203535.5
207230.0
213446.5
```

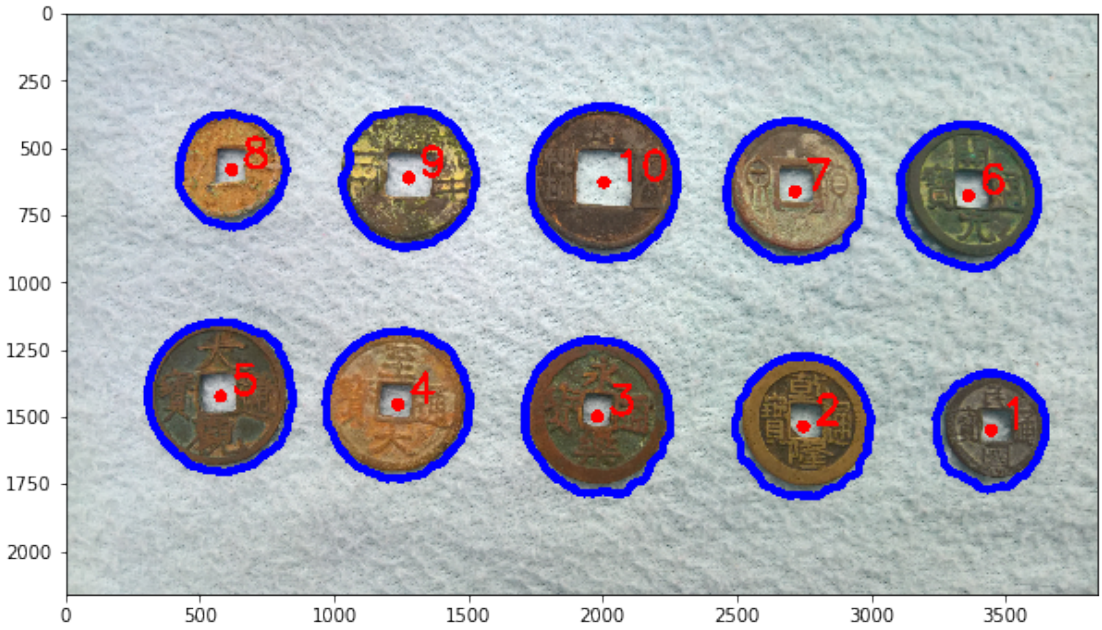
```
229459.0
235121.5
242100.5
243313.0
8288401.0
```

We can clearly see the jump from 2nd area to 3rd. These are the 2 inner contours.

```
In [71]: # Remove the 2 inner contours
         # Plot the rest of them
         ###
         ### YOUR CODE HERE
         ###
```

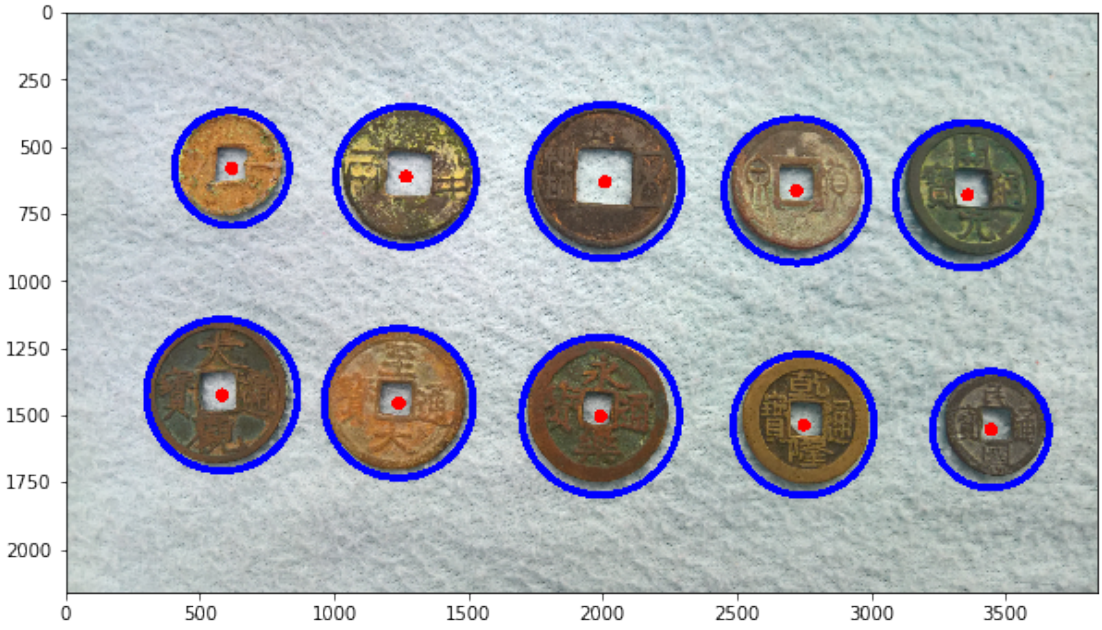
```
Number of contours = 10
```

```
Out[71]: <matplotlib.image.AxesImage at 0x7fcf1167c1d0>
```




```
In [72]: # Fit circles on coins
         ###
         ### YOUR CODE HERE
         ###
```

```
Number of coins detected = 10
```


0
250
500
750
1000
1250
1500
1750
2000
0
500
1000
1500
2000
2500
3000
3500