



DIGITAL IMAGE PROCESSING DEMONSTRATION

CPE 0322.1 – Elective 3 (Lab)

ABSTRACT

This laboratory activity covers an introduction to image segmentation through the watershed algorithm. It tackles the concept and fundamentals through a thorough discussion of the procedure and simulation of a program that counts the number of objects in a picture.

Calungsod, Kaye A.

2018-01175

LABORATORY ACTIVITY #1

DIGITAL IMAGE PROCESSING DEMONSTRATION

INSTRUCTIONS

- Demonstrate image analysis using segmentation using any applicable image processing analysis algorithm.
- Follow the template provided for final written report.
- Provide your own introduction and discussion.
- Provide the procedure and instruction.
- Show the algorithm – flowchart and command program.
- Show the output simulation.
- Present your observation and conclusion.
- Include references.
- Make a video demonstration on how you performed the activity.

INTRODUCTION

Digital image processing consists of the manipulation of images using digital computers (Da Silva et. al., 2005, chap. 4). It refers to the use of computer algorithms to perform image processing on digital or digitized images, leading to the extraction of attributes from the processed images and to the recognition and mapping of individual objects, features or patterns (IGI Global, n.d.). Its use has been increasing exponentially in the last decades. Its applications range from medicine to entertainment, passing by geological processing and remote sensing. Multimedia systems, one of the pillars of the modern information society, rely heavily on digital image processing (Da Silva et. al., 2005, chap. 4).

Image Segmentation is the digital image processing technique by which a digital image is partitioned into various subgroups (of pixels) called Image Objects, which can reduce the complexity of the image, and thus analysing the image becomes simpler (Prasad, n.d.). Through the use of various image segmentation algorithms to split and group a certain set of pixels together from the image, assignment of labels to pixels is made possible with pixels of the same label falling under a category set by the programmer (Prasad, n.d.).

Using these labels, programmers can specify boundaries, draw lines, and separate the most required objects in an image from the rest of the not-so-important ones (Prasad, n.d.).

In this activity, I will demonstrate and tackle in detail about Image Segmentation through the use of a modified watershed algorithm available in Open CV.

DISCUSSION

According to Mathworks (n.d.), **image analysis** involves processing an image into fundamental components to extract meaningful information. Image analysis can include tasks such as finding shapes,

detecting edges, removing noise, counting objects, and calculating statistics for texture analysis or image quality. It can be as simple as reading bar coded tags or as sophisticated as identifying a person from their face (Freebase, n.d.).

In this activity, the image analysis to be implemented is simple – counting the number of objects in Figure 1.



Fig. 1 water_coins.jpg

As mentioned in the introduction, the algorithm to be used in the implementation of this image analysis is through **watershed** or specifically **marker-based watershed segmentation**.

According to scikit-image (n.d.), the watershed is a classical algorithm used for segmentation or separation of different objects in an image. Starting from user-defined markers, the watershed algorithm treats pixels values as a local topography (elevation). The algorithm floods basins from the markers until basins attributed to different markers meet on watershed lines. In many cases, markers are chosen as local minima of the image, from which basins are flooded (sckit-image, n.d.).

This approach often leads to an over-segmented result due to noise or any other irregularities in the image therefore markers are put into place in the topographic surface in order to prevent this from happening.

Examples of watershed segmentation application can be seen in road segmentation – automatic detection of markers is accomplished in the PROMETHEUS project, coffee-beans separation – markers done through distance function, and finally traffic monitoring – markers are implemented through the use of both distance function and ground extraction.

PROCEDURE AND INSTRUCTION

Steps in installing the IDE and relevant packages to implement the demonstration:

1. Download and install the latest python version in this site: <https://www.python.org/downloads/>
2. Open the command line of your PC by pressing Win+R.
3. Type: pip install opencv-python to install the open CV packages compatible to the version of python that you have.
4. Type: pip install numpy to install the numpy packages compatible to the version of python that you have.
5. Type: pip install matplotlib to install the matplotlib packages compatible to the version of python that you have.
6. Open Python IDLE by clicking the windows batch file entitled "idle". It can be found in the following folder path:
C:\Users\<username>\AppData\Local\Programs\Python\Python39\Lib\idlelib
7. Check if the packages are successfully installed by typing in Python IDLE:
 - import cv2; print(cv2.__version__)
 - import numpy; print(numpy.__version__)
 - import matplotlib; print(matplotlib.__version__)

According to Boucher (2010), segmenting an image using watershed segmentation is a two-step process:

- Finding the markers and the segmentation criterion (the criterion or function which will be used to split the regions – it is most often the contrast or gradient, but not necessarily).
- Performing a marker-controlled watershed with these two elements.

For the cv.watershed() function to work properly, preprocessing and preparation is needed. Especially since Figure 1 – the image to be analyzed – has its objects mutually touching each other. Taking that into account, distance transform function will be utilized in the segmentation of objects.

The first step is to find the approximate estimate of the coins. This can be done by performing Otsu's binarization on the grey tone version of the original image.

```
gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
ret, thresh = cv.threshold(gray, 0, 255, cv.THRESH_BINARY_INV+cv.THRESH_OTSU)
```

According to HBY coding academic (2019), Otsu's method is an adaptive thresholding way for binarization in image processing. It can find the optimal threshold value of the input image by going through all possible threshold values (from 0 to 255).

After knowing the threshold, we need remove any small white noises in the image. These white noises are small objects from the foreground (bright pixels) of an image that we need to place in the background.

In order to implement this, we can use morphological opening based on the threshold gotten beforehand.

```
# noise removal
kernel = np.ones((3,3),np.uint8)
opening = cv.morphologyEx(thresh,cv.MORPH_OPEN,kernel, iterations = 2)
```

At this point, the image should now appear like this:

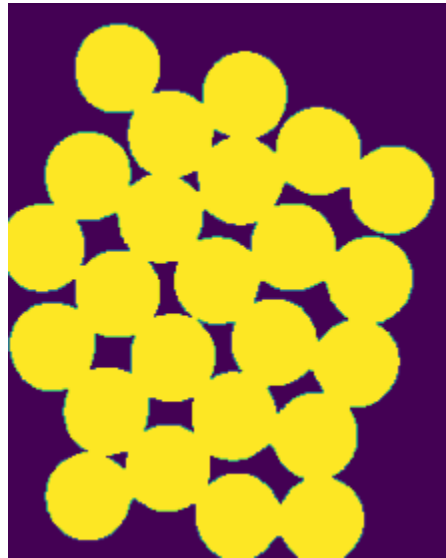


Fig. 2 Processed Image –after noise removal

Based on Figure 2, we can say for sure that the region near to center of objects is considered foreground and region much away from the object are background. The only region we are not sure is the boundary region of the coins.

Taking this into account, the next step, then, is to extract the area which we are sure are coins. This can be done through erosion. Erosion removes the boundary pixels. So whatever's remaining, we can be sure that it is a coin. That would work if objects were not touching each other; however, since the objects are touching each other for the image chosen, another good option would be to find the distance transform and applying a proper threshold.

Additionally, we also need to find the area in which we are sure that are not coins to help us find the boundary for each image. For that, we use dilation. Dilation increases object boundary to background. This way, we can make sure whatever region in background in result is really a background, since boundary region is removed.

```
# sure background area
sure_bg = cv.dilate(opening,kernel,iterations=3)
# Finding sure foreground area
dist_transform = cv.distanceTransform(opening,cv.DIST_L2,5)
ret, sure_fg = cv.threshold(dist_transform,0.7*dist_transform.max(),255,0)
```

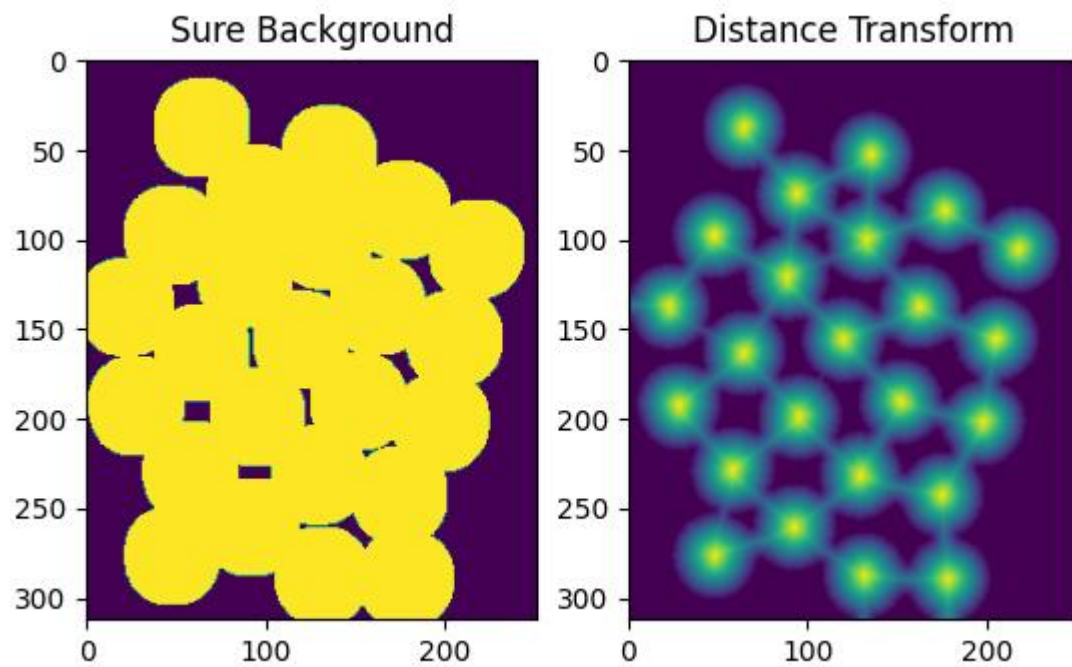


Fig. 3 Processed Image – dilation and distance transform

For the remaining regions that we don't know if they are coins or background, the built in Open CV watershed algorithm should be able to take care of it. These areas are normally around the boundaries of coins where foreground and background meet. We call it border. It can be obtained from subtracting `sure_fg` area from `sure_bg` area.

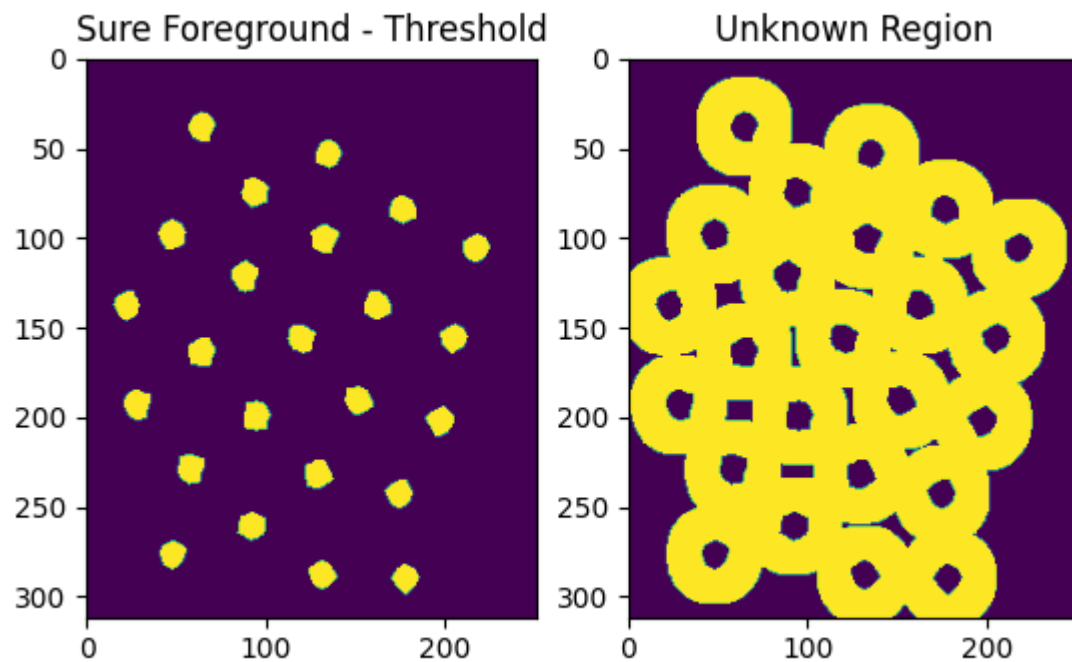


Fig. 4 Processed Image – threshold and border

Now we know for sure the region where the coins are and backgrounds are, we can now create a marker and label the regions inside it. A marker is an array of same size as that of original image, but with int32 data type.

The regions we know for sure (whether foreground or background) are labeled with any positive integers, but different integers and the area we don't know for sure are just left as zero. For this we use `cv.connectedComponents()`. It labels background of the image with 0, while other objects are labeled with integers starting from 1.

```
# Marker labelling
ret, markers = cv.connectedComponents(sure_fg)
```

Since we know that if background is marked with 0, watershed will consider it as unknown area. So we want to mark it with different integer. Instead, we will mark unknown region, defined by `unknown`, with 0.

```
# Add one to all labels so that sure background is not 0, but 1
markers = markers+1
# Now, mark the region of unknown with zero
markers[unknown==255] = 0
```

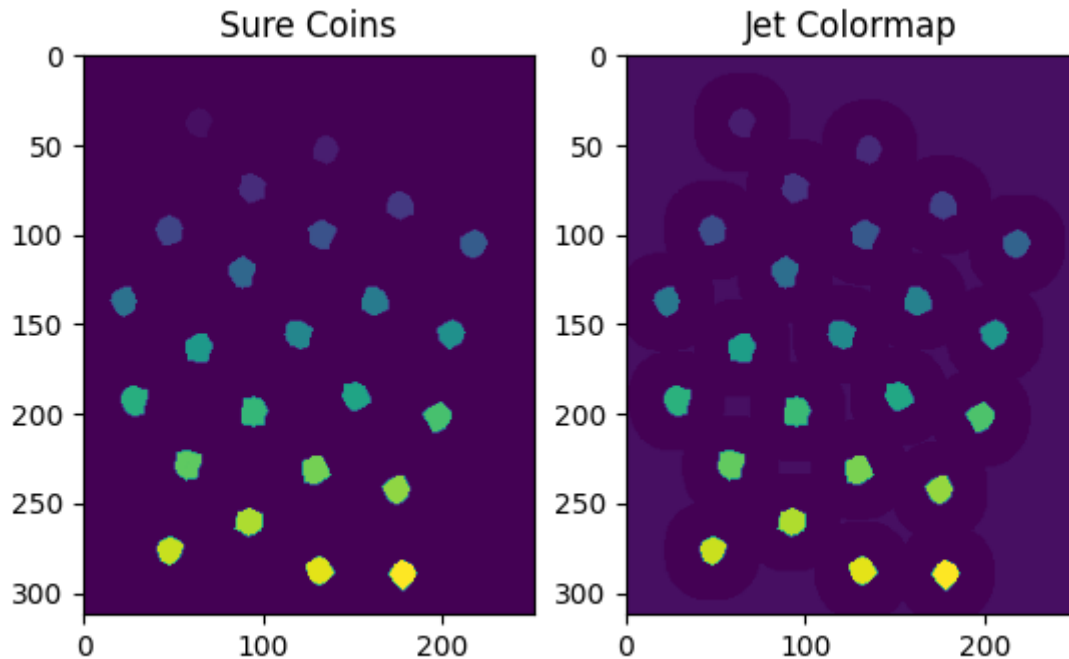


Fig. 5 Processed Image – putting on markers

See the result shown in Figure 5 for JET color map. The dark purple region – a tad darker than the background shows the unknown region. Sure coins are colored with different values. The remaining area – the background, are shown in a lighter hue compared to the unknown region.

Now that the marker is ready and the image is processed completely, it is time for final step, apply watershed or `cv.watershed()`. The image for the marker will be modified and the boundary region will be marked with -1.

```
markers = cv.watershed(img, markers)
img[markers == -1] = [255, 0, 0]
```

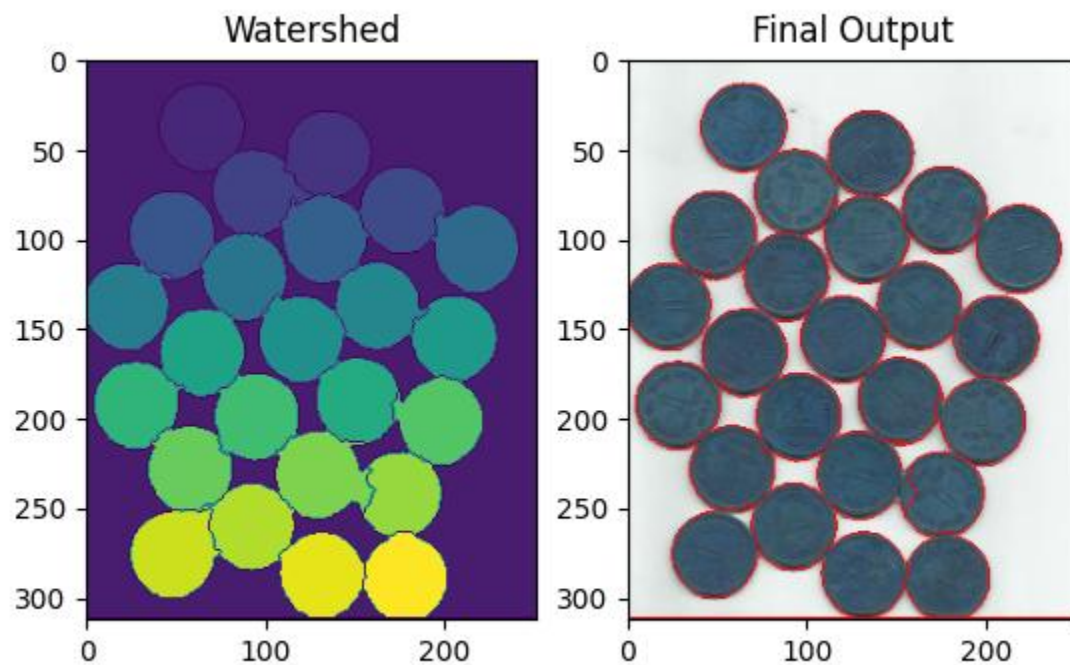



Fig. 6 Processed Image – watershed and final output

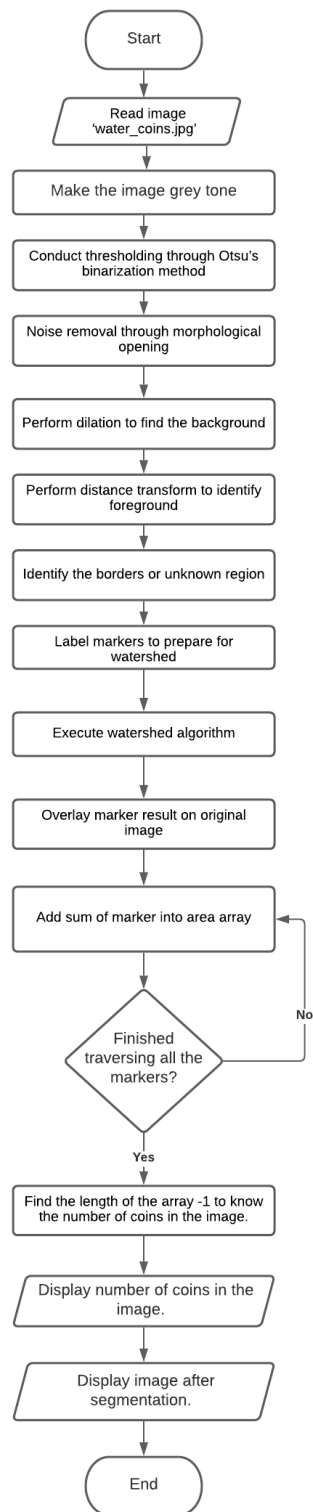
Now that the image is finally processed and segmented, image analysis can now commence. It can vary depending on the image and algorithm to be used but in this one, I decided to create an array called `area` wherein it consists of the sum of the array elements of each marker. The elements are fetched through a `for range()` loop with the variable `ret` – referring to the number of components in the image, being the amount of times the program would traverse.

```
area = [np.sum(markers==val) for val in range(ret)]

print("-Number of Pixels: " + str(img.size))
print("-Shape/Dimensions: " + str(img.shape))
print("-Number of Coins: " + str(len(area)-1))
print("-Array of the Processed Image: " + str(img))
```

The number of coins in Figure 1 is then counted through the use of the function `len()`. Finding the length of the `area` -1 (since we're discounting the background as a component), would yield 24. This value is correct and double-checked after manually counting the number of coins in Figure 1.

FLOWCHART



- Thresholding
 - An image processing method that creates a bitonal (aka binary) image based on setting a threshold value on the pixel intensity of the original image. The thresholding process is sometimes described as separating an image into foreground values (black) and background values (white).
- Otsu's Binarization
 - Otsu's method is an adaptive thresholding way for binarization in image processing. It can find the optimal threshold value of the input image by going through all possible threshold values (from 0 to 255).
 - In computer vision and image processing, Otsu's method, named after Nobuyuki Otsu, is used to perform automatic image thresholding. In the simplest form, the algorithm returns a single intensity threshold that separate pixels into two classes, foreground and background.
- Morphological Opening
 - Morphological opening removes small objects from the foreground (usually taken as the bright pixels) of an image.
 - Erosion followed by dilation.
- Dilation
 - Dilation adds pixels to the boundaries of objects in an image, while erosion removes pixels on object boundaries. The number of pixels added or removed from the objects in an image depends on the size and shape of the structuring element used to process the image.
- Distance Transform
 - The distance transform is an operator normally only applied to binary images. The result of the transform is a graylevel image that looks similar to the input image, except that the graylevel intensities of points inside foreground regions are changed to show the distance to the closest boundary from each point.

COMMAND PROGRAM

```
opencv_coins.py - C:\Users\william sy\Desktop\Digital Image Processing\opencv_...
File Edit Format Run Options Window Help
#libraries/packages to be used
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt

# declaring figure 1 - shows original and processed image.
f1, (orig,processed) = plt.subplots(1,2)

img = cv.imread('water_coins.jpg')

orig.imshow(img)
orig.set_title("Original Image")

# image properties: array of original image
print("Image Properties")
print("-Array of the Original Image: " + str(img))
print("\n")

# gray-scale and otsu binarization thresholding
gray = cv.cvtColor(img,cv.COLOR_BGR2GRAY)
ret, thresh = cv.threshold(gray,0,255,cv.THRESH_BINARY_INV+cv.THRESH_OTSU)

# noise removal
kernel = np.ones((3,3),np.uint8)
opening = cv.morphologyEx(thresh,cv.MORPH_OPEN,kernel, iterations = 2)

# declaring figure 2 - shows gray-scale and image with noise removal.
f2, (gr,noise) = plt.subplots(1,2)

gr.imshow(gray)
gr.set_title("Gray tone")

noise.imshow(opening)
noise.set_title("Noise Removal")

# sure background area
sure_bg = cv.dilate(opening,kernel,iterations=3)

# finding sure foreground area using distance transform
dist_transform = cv.distanceTransform(opening,cv.DIST_L2,5)

Ln: 27 Col: 35

opencv_coins.py - C:\Users\william sy\Desktop\Digital Image Processing\opencv_...
File Edit Format Run Options Window Help
# finding sure foreground area using distance transform
dist_transform = cv.distanceTransform(opening,cv.DIST_L2,5)
ret, sure_fg = cv.threshold(dist_transform,0.7*dist_transform.max(),255,0)

# declaring figure 3 - shows background and distance transform.
f3, (back,dist) = plt.subplots(1,2)
back.imshow(sure_bg)
back.set_title("Sure Background")

dist.imshow(dist_transform)
dist.set_title("Distance Transform")

# finding unknown region
sure_fg = np.uint8(sure_fg)
unknown = cv.subtract(sure_bg,sure_fg)

# declaring figure 4 - shows foreground and unknown region.
f4, (fore,un) = plt.subplots(1,2)

fore.imshow(sure_fg)
fore.set_title("Sure Foreground - Threshold")

un.imshow(unknown)
un.set_title("Unknown Region")

# marker labelling
ret, markers = cv.connectedComponents(sure_fg)

# declaring figure 5 - shows markers and colormap.
f5, (mark,final) = plt.subplots(1,2)
mark.imshow(markers)
mark.set_title("Sure Coins")

# add one to all labels so that sure background is not 0, but 1
markers = markers+1
# now, mark the region of unknown with zero
markers[unknown==255] = 0

final.imshow(markers)
final.set_title("Jet Colormap")

# perform watershed and overlay markers to final processed result
markers = cv.watershed(img,markers)
img[markers == -1] = [255,0,0]

# declaring figure 6 - shows watershed and final result.
f6, (ws,result) = plt.subplots(1,2)
ws.imshow(markers)
ws.set_title("Watershed")

result.imshow(img)
result.set_title("Final Output")

processed.imshow(img)
processed.set_title("After Processing")

# stores the sum of each marker in an array
area = [np.sum(markers==val) for val in range(ret)]

# image properties: size, dimensions, number of coins, array of processed image
print("-Number of Pixels: " + str(img.size))
print("-Shape/Dimensions: " + str(img.shape))
print("-Number of Coins: " + str(len(area)-1))
print("-Array of the Processed Image: " + str(img))

plt.show()

Ln: 27 Col: 35
```

```
#libraries/packages to be used
```

```
import numpy as np
```

```
import cv2 as cv
```

```
from matplotlib import pyplot as plt
```

```
# declaring figure 1 - shows original and processed image.
```

```
f1, (orig,processed) = plt.subplots(1,2)
```

```
img = cv.imread('water_coins.jpg')

orig.imshow(img)
orig.set_title("Original Image")

# image properties: array of original image
print("Image Properties")
print("-Array of the Original Image: " + str(img))
print('\n')

# gray-scale and otsu binarization thresholding
gray = cv.cvtColor(img,cv.COLOR_BGR2GRAY)

ret, thresh =
cv.threshold(gray,0,255,cv.THRESH_BINARY_INV+cv.THRESH_OTSU)

# noise removal
kernel = np.ones((3,3),np.uint8)
opening = cv.morphologyEx(thresh,cv.MORPH_OPEN,kernel, iterations = 2)

# declaring figure 2 - shows grayscale and image with noise removal.
f2, (gr,noise) = plt.subplots(1,2)

gr.imshow(gray)
gr.set_title("Grey tone")

noise.imshow(opening)
noise.set_title("Noise Removal")

# sure background area
sure_bg = cv.dilate(opening,kernel,iterations=3)
```

```
# finding sure foreground area using distance transform
dist_transform = cv.distanceTransform(opening,cv.DIST_L2,5)
ret, sure_fg =
cv.threshold(dist_transform,0.7*dist_transform.max(),255,0)

# declaring figure 3 - shows background and distance transform.
f3, (back,dist) = plt.subplots(1,2)
back.imshow(sure_bg)
back.set_title("Sure Background")

dist.imshow(dist_transform)
dist.set_title("Distance Transform")

# finding unknown region
sure_fg = np.uint8(sure_fg)
unknown = cv.subtract(sure_bg,sure_fg)

# declaring figure 4 - shows foreground and unknown region.
f4, (fore,un) = plt.subplots(1,2)

fore.imshow(sure_fg)
fore.set_title("Sure Foreground - Threshold")

un.imshow(unknown)
un.set_title("Unknown Region")

# marker labelling
ret, markers = cv.connectedComponents(sure_fg)

# declaring figure 5 - shows markers and colormap.
```

```
f5, (mark,final) = plt.subplots(1,2)
mark.imshow(markers)
mark.set_title("Sure Coins")

# add one to all labels so that sure background is not 0, but 1
markers = markers+1

# now, mark the region of unknown with zero
markers[unknown==255] = 0

final.imshow(markers)
final.set_title("Jet Colormap")

# perform watershed and overlay markers to final processed result
markers = cv.watershed(img,markers)
img[markers == -1] = [255,0,0]

# declaring figure 6 - shows watershed and final result.
f6, (ws,result) = plt.subplots(1,2)
ws.imshow(markers)
ws.set_title("Watershed")

result.imshow(img)
result.set_title("Final Output")

processed.imshow(img)
processed.set_title("After Processing")

# stores the sum of each marker in an array
area =[np.sum(markers==val) for val in range(ret)]
```

```

# image properties: size, dimensions, number of coins, array of
processed image

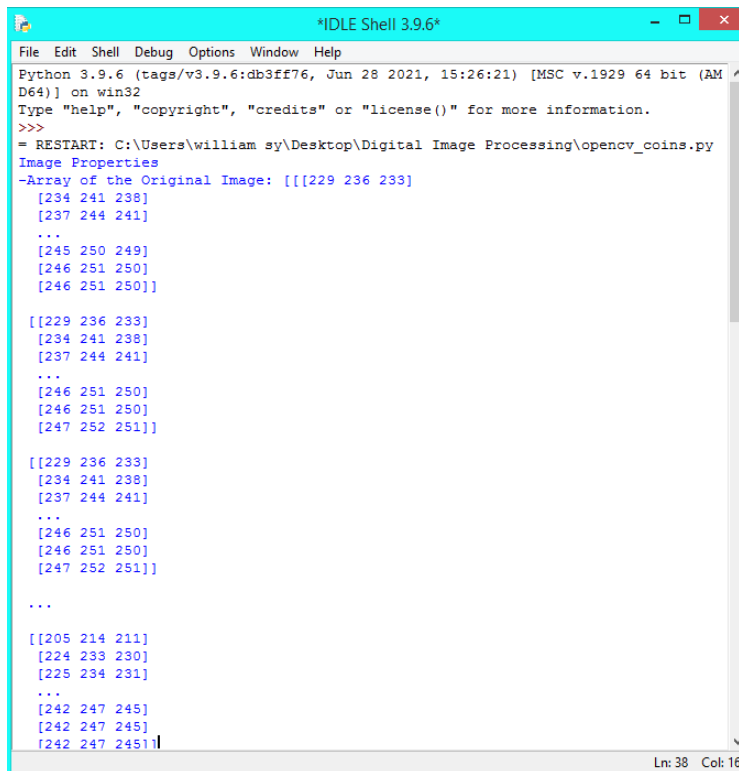
print("-Number of Pixels: " + str(img.size))
print("-Shape/Dimensions: " + str(img.shape))
print("-Number of Coins: " + str(len(area)-1))
print("-Array of the Processed Image: " + str(img))

plt.show()

```

OUTPUT SIMULATION

The first four screenshots below shows the array of the original image and final processed image, its size (in pixels), its dimensions (with the format: height, width, RGB color) and finally the number of coins of the original image.



```

Python 3.9.6 (tags/v3.9.6:db3ff76, Jun 28 2021, 15:26:21) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\william sy\Desktop\Digital Image Processing\opencv_coins.py
Image Properties
-Array of the Original Image: [[[229 236 233]
[234 241 238]
[237 244 241]
...
[245 250 249]
[246 251 250]
[246 251 250]]

[[229 236 233]
[234 241 238]
[237 244 241]
...
[246 251 250]
[246 251 250]
[247 252 251]]

[[229 236 233]
[234 241 238]
[237 244 241]
...
[246 251 250]
[246 251 250]
[247 252 251]]

...

[[205 214 211]
[224 233 230]
[225 234 231]
...
[242 247 245]
[242 247 245]
[242 247 245]]

```



```
*IDLE Shell 3.9.6*
File Edit Shell Debug Options Window Help

[[199 208 205]
[218 227 224]
[220 229 226]
...
[241 246 244]
[241 246 244]
[241 246 244]]

[[193 202 199]
[212 221 218]
[215 224 221]
...
[240 245 243]
[240 245 243]
[240 245 243]]]

-Number of Pixels: 235872
-Shape/Dimensions: (312, 252, 3)
-Number of Coins: 24
-Array of the Processed Image: [[[255 0 0]
[255 0 0]
[255 0 0]
...
[255 0 0]
[255 0 0]
[255 0 0]]

[[255 0 0]
[234 241 238]
[237 244 241]
...
[246 251 250]
[246 251 250]
[255 0 0]]

[[255 0 0]
[234 241 238]
[237 244 241]
Ln: 38 Col: 16
```

```
*IDLE Shell 3.9.6*
File Edit Shell Debug Options Window Help

...
[246 251 250]
[246 251 250]
[255 0 0]]

[[255 0 0]
[234 241 238]
[237 244 241]
...
[246 251 250]
[246 251 250]
[255 0 0]]

...

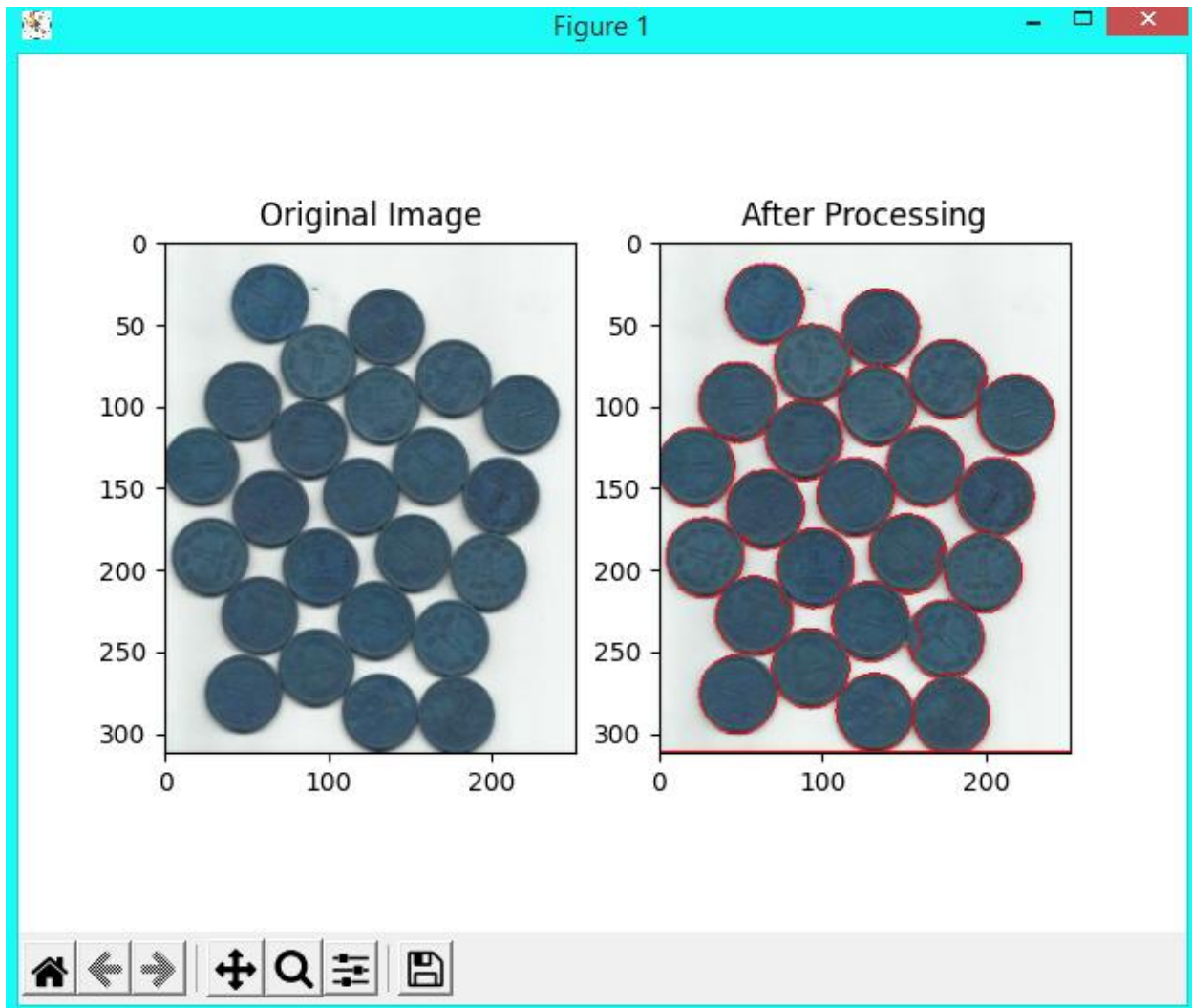
[[255 0 0]
[224 233 230]
[225 234 231]
...
[242 247 245]
[242 247 245]
[255 0 0]]

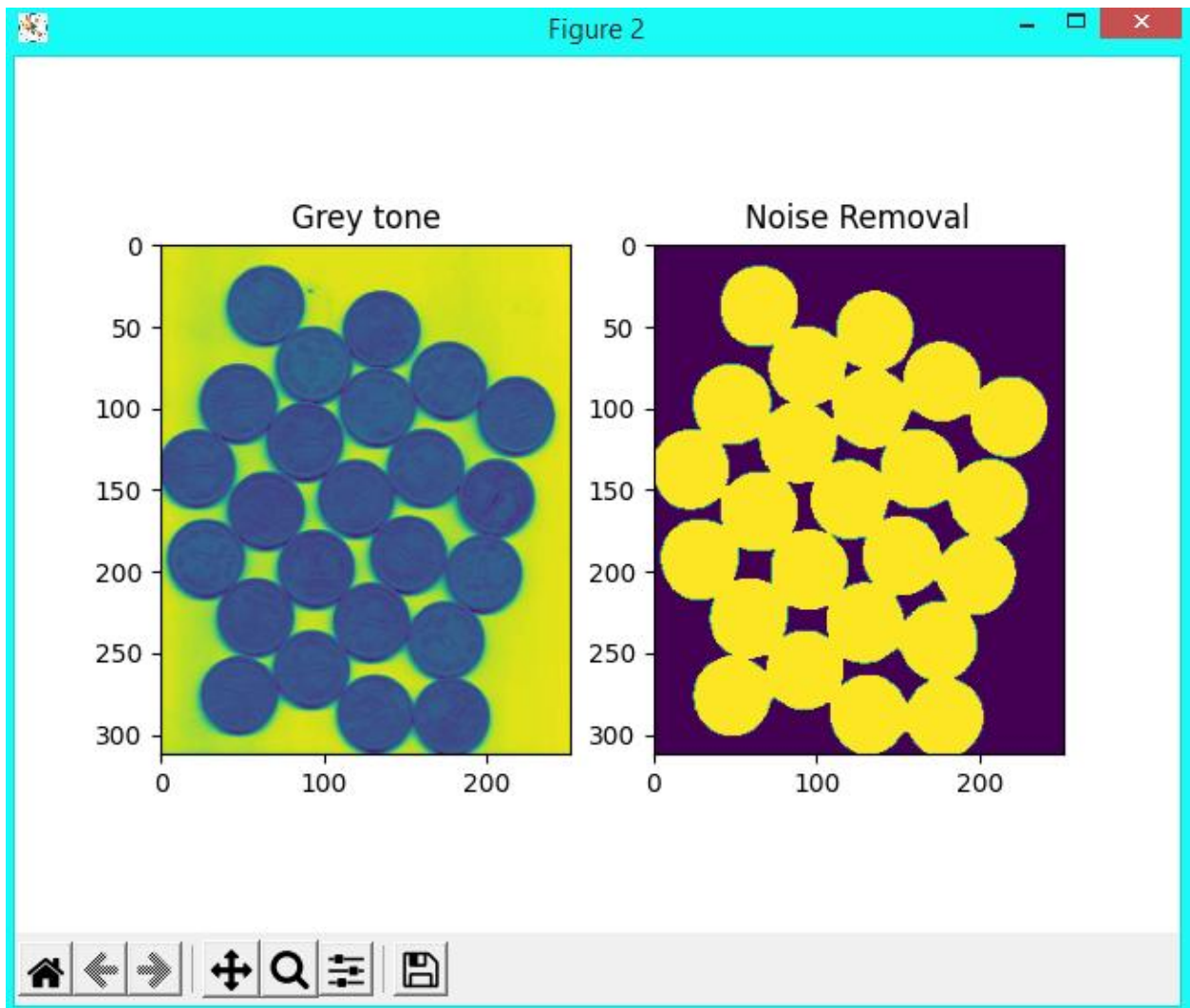
[[255 0 0]
[218 227 224]
[220 229 226]
...
[241 246 244]
[241 246 244]
[255 0 0]]

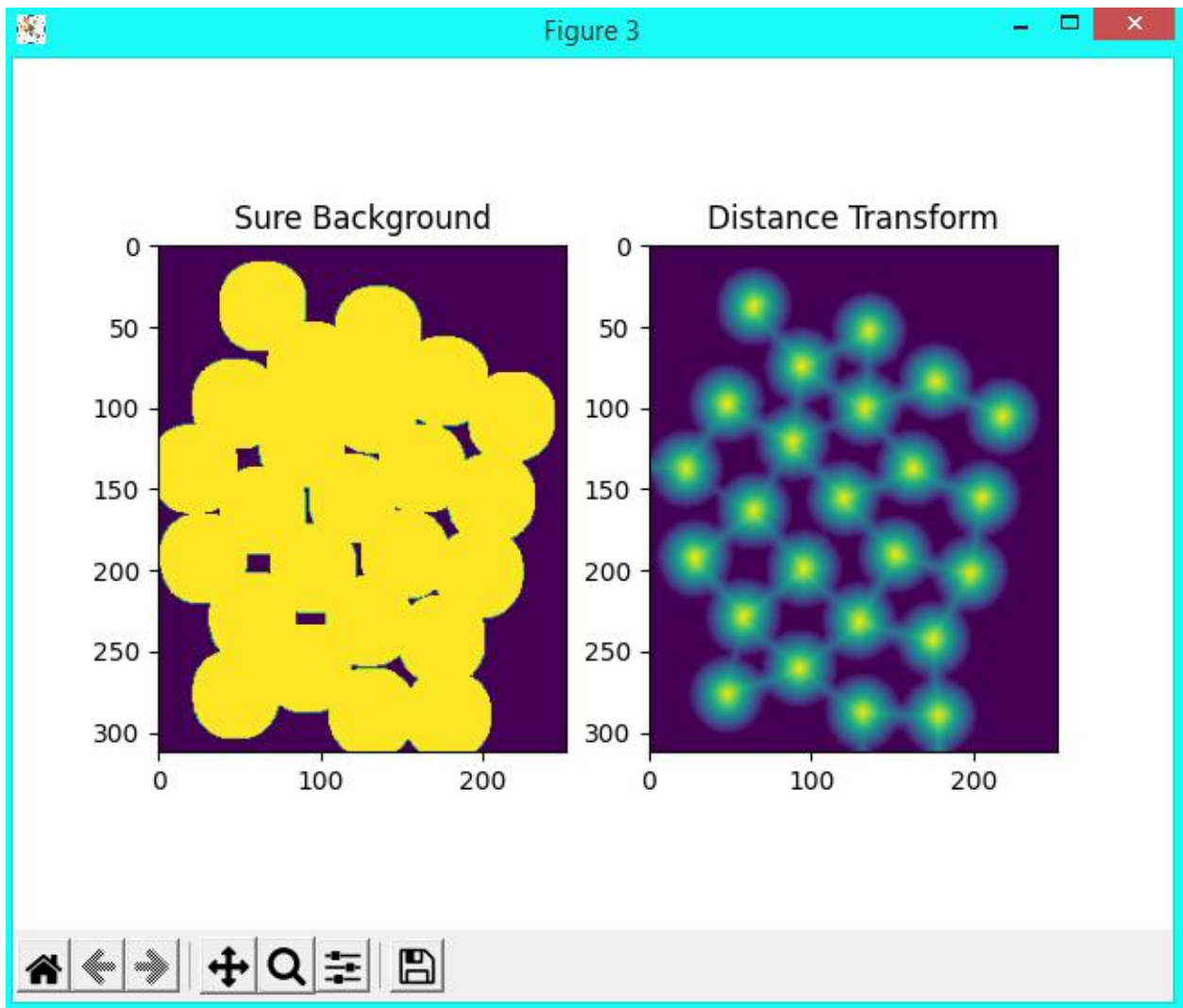
[[255 0 0]
[255 0 0]
[255 0 0]
...
[255 0 0]
[255 0 0]
[255 0 0]]]

Ln: 38 Col: 16
```

The following six (6) figures show the changes of the image from the original to the final processed result.







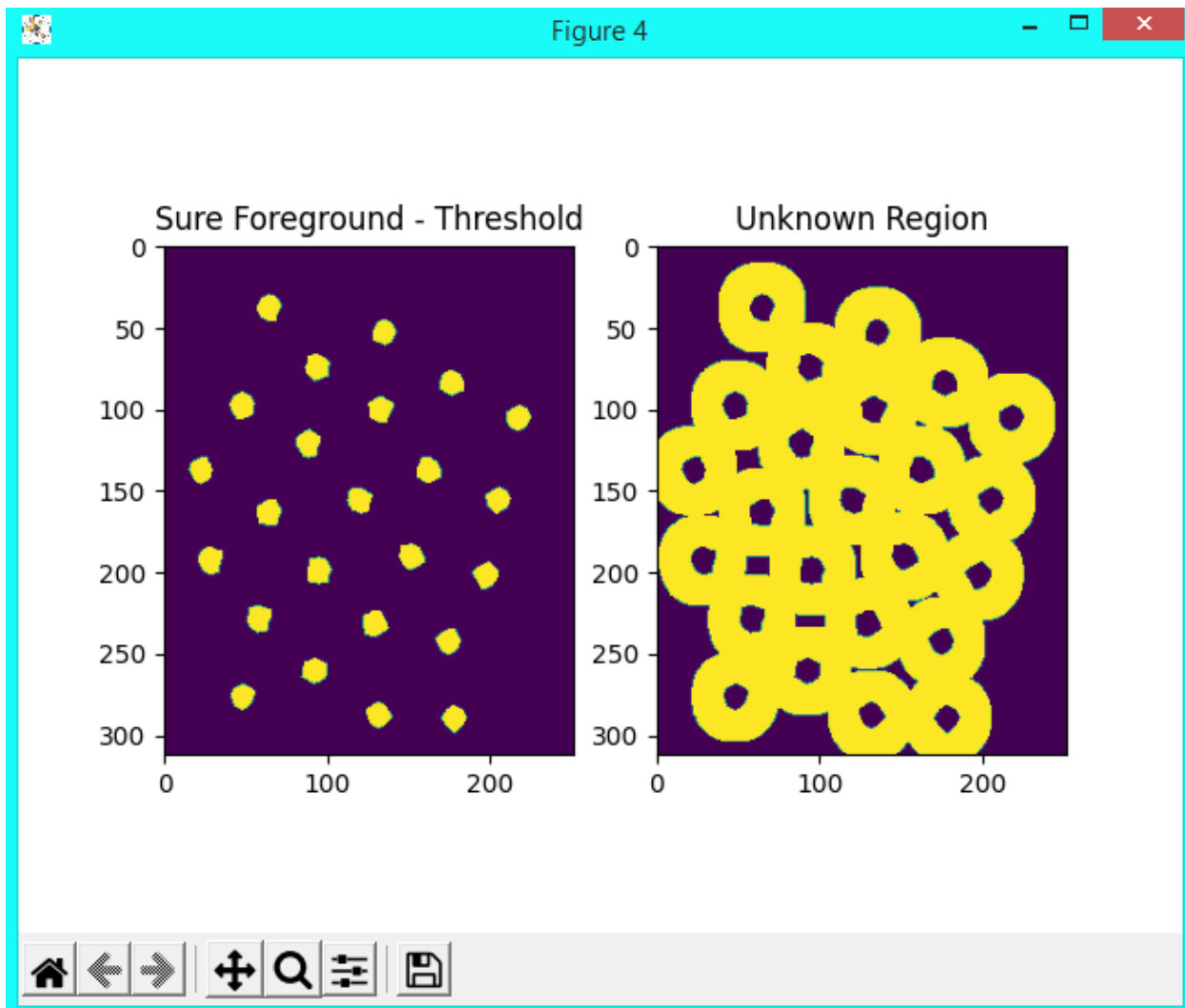




Figure 5

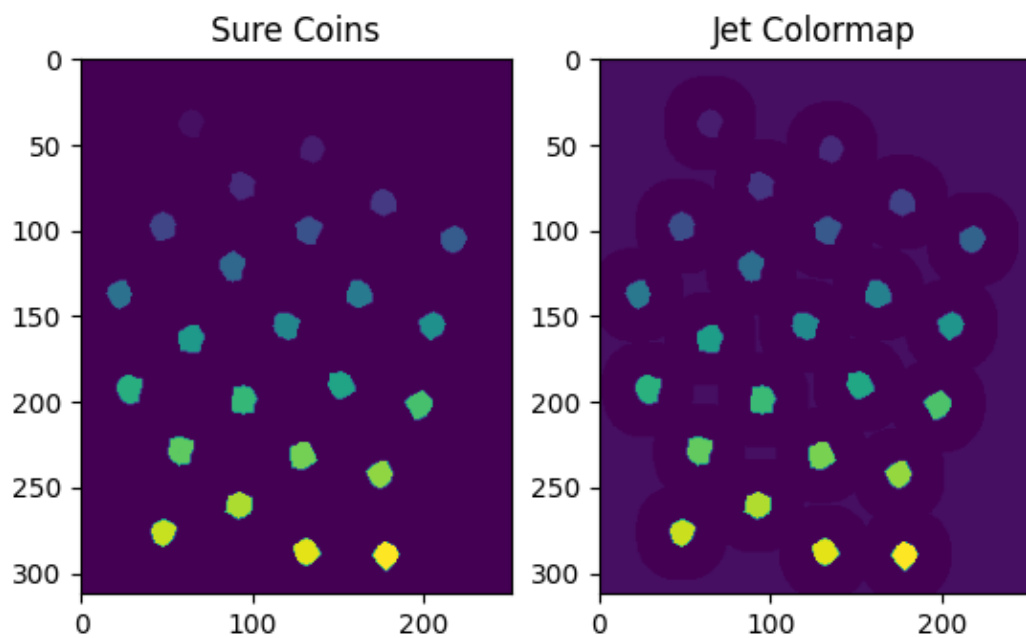
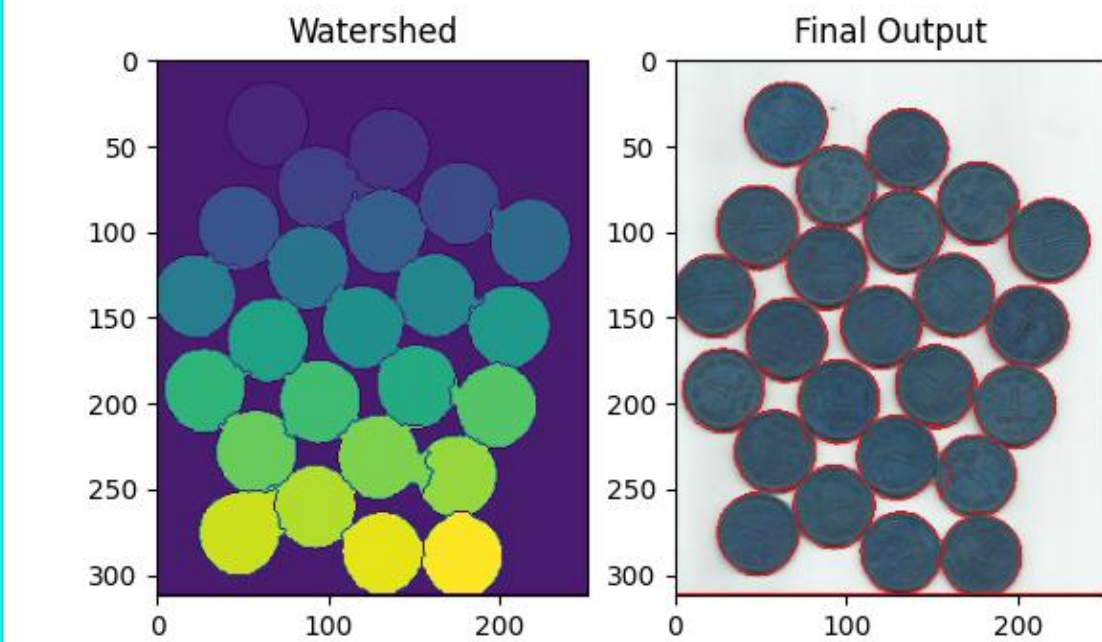


Figure 6



OBSERVATIONS AND CONCLUSION

In the activity conducted, I have observed and learned how many image processing techniques work hand in hand together in order to use the watershed algorithm properly. From Otsu's binarization method to erosion to distance transform and dilation – it's a lot before something as simple as counting the number of objects in an image can even commence. I have tried the same command program on other pictures before in my study of the watershed algorithm and it didn't really work as well in the image analysis part even if the pre-processing and segmentation was done properly. The counting is very faulty when I used the area array in images that are not overlapping each other. This only means that the code algorithm I used isn't robust or flexible enough for other images. Further improvement is necessary along with a deeper knowledge on how each function in Open CV – Python work before I can tackle more complex image analysis topics like facial recognition and etc.

Based on the activity conducted, I therefore conclude that digital image processing is a worthy and challenging endeavour that could have a lot of potential. This activity made me realize why digital image processing is a tool used in artificial intelligence and machine learning. It has a lot of potential. It's also very versatile and can be modified to suit the programmer's own purposes.

Watershed segmentation takes a lot of preparation and pre-processing. Noise can really drastically affect the final output and over-segmentation is still a problem even with markers. Simplifying an already seemingly simple image takes a lot of work. It needs to be modified based on the content of the image itself. It might take a while before I can fully create my own program in computer vision due to how large the subject is in general, but this subject is definitely worth delving into in the next few weeks.

REFERENCES

- Beucher, S. (2010, May 18). *IMAGE SEGMENTATION AND MATHEMATICAL MORPHOLOGY*. CMM. <https://people.cmm.minesparis.psl.eu/users/beucher/wtshed.html>
- Da Silva, E, Mendonca, G. (2005). *The Electronic Engineering Handbook*. Academic Press. <https://www.sciencedirect.com/science/article/pii/B9780121709600500645>
- HBY coding academic. (2019, January 19). *Otsu thresholding — image binarization*. Medium. <https://hbyacademic.medium.com/otsu-thresholding-4337710dc519>
- IGI Global. (n.d.). Digital Image Processing. In *IGI-Global.com Dictionary*. <https://www.igi-global.com/dictionary/digital-image-processing-dip/48620>
- Mathworks. (n.d.). *Image Analysis*. <https://www.mathworks.com/discovery/image-analysis.html>
- Open CV. (n.d.). *Image Segmentation with Watershed Algorithm*. https://docs.opencv.org/4.5.2/d3/db4/tutorial_py_watershed.html
- Prasad, S. (n.d.). *What is Image Segmentation?*. AnalytixLabs. <https://www.analytixlabs.co.in/blog/what-is-image-segmentation/>
- Scikit-image. (n.d.). *Watershed segmentation*. https://scikit-image.org/docs/dev/auto_examples/segmentation/plot_watershed.html