**Group Members :** Ibrahim Asad & Tien Dat Johny Do
**Project Title :** Cell Segmentation with the use of Binary Mask

**Course Code:** BMEN 509 - Winter 2024
**Lab Section:** B01

Ibrahim Asad, UCID: 30088413
Tien Dat Johny Do, UCID : 30087967

**Link to GitHub:**
https://github.com/TienDatJohnyDo/Biomedical-Project

**Instructor:** Jalil Addeh
**Submission Day:** April 08, 2024

## Introduction:

- **Project motivation :**

University researchers have spent numerous hours studying cells through staining experiments, radiation exposure experiments, and many other cell studies. At University of Calgary, the biomedical department's most time consuming task is analysing cells through microscopes. Our group wants to tackle one of the major problems in cell imaging being segmentation. Segmentation is the initial stage of cell image processing, which helps identify individual cells within an image. Using segmentation masks, it helps indicate if the pixel in the image is a cell or pertains to the background. This helps researchers understand cell characteristics. This project relates to biomedical imaging because it allows researchers to understand and better visualize cells within a biomedical image and start the image processing pipeline more efficiently.

- **Clear statement of objective and specific aims:**

Our group wants to develop a method in which cell segmentation becomes more efficient within the biomedical imaging process for our researchers. By addressing this problem, our researchers will have more time analyzing the cells and our method will accelerate research and streamline the initial stage of image processing. Our specific aim is to develop an advanced image processing method that can accurately segment cells individually using segmentation masks and reduce the time and effort of this for our researchers.

- **Theory:**

The theory and methodology behind our project was to establish cell segmentation more effectively. We did this by altering the images contrast to by converting it to grayscale. Changing the greyscale is important as it is often used in various other image segmentation techniques as it allows for the colour of the image to be various shades of grey allowing for easier visibility of parts within the image. We then use a gaussian filter to remove noise within the image and as well as thresholding to help smoothen the image, reduce the total noise, and change the image to black and white. Lastly cv2 and np.zeros_like tools are used to create the binary masks by detecting edges and then finding areas where the edges are then located. Once done we take the altered image and set the transparency to 0.5 and overlay it on the original image.

## Materials and Methods:

- **Data for the Project**

Data source : Online data source from the Cell Image Library Dataset
http://www.cellimagelibrary.org/pages/datasets
http://www.cellimagelibrary.org/images/CCDB_3632
http://www.cellimagelibrary.org/project/P1700

**Modality of data including acquisition parameters :** The file loaded below for Figures 1 through 8 is an MRC file of the immunological synapse between a human cytotoxix T lymphocyte and a target cell. Some acquisition parameters is that this image is from the blood system from the CTL immunological synapse that has a thickness of 0.15 micrometers. The experiment date was capture on the 11-08-2005.

The Second File is a stack of several IMOD st formatted images of serial section electron micrographs. The electron micrographs of the cell body are 0.25 micrometers thick through a cortical pyramidal neuron from the brain and an Alzheimer's Patient over several decades ago and therefore experimental date is not indicated.

**Anatomy/physiology/object :** First Image is the structure of the immunological synapse between a human cytotoxic T lymphocyte CTL and a target cell. Second Image is the reconstruction of cortical neurons from biopsy material obtained from an Alzheimer's Patient.

**Relevancy of data :** These data have many elements such as centrioles, microtubles, golgi apparatus and lytic granules within the CTL and synaptic cleft and our group wants to segment each part. The Images from the dataset are also useful when doing the project as the images replicate what the researcher does however with less human error as the images seen below have large shapes indicating where to segment them.

**Sample Screenshot of image data loaded:**

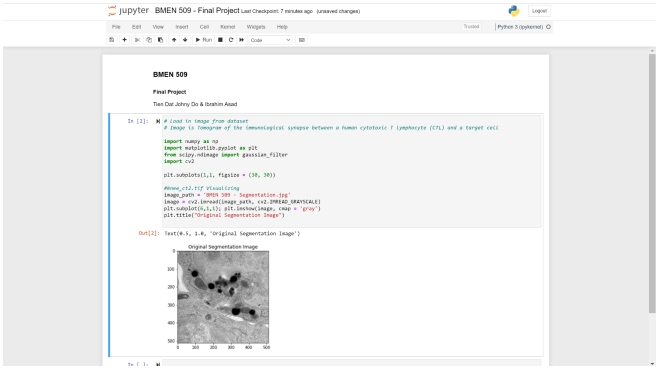



**Figure 1**

- **Restatement of Tools & Data**

  Below is a consolidated list of tools/libraries our group plans on using and the rationale behind it. More information of how our group plans on using these tools and data is provided within our Data & Tools report, but this is condensed.

**Matlab** - Our group plans to use this tool in order to visualize our results from our processes.
It allows for image processing, and allows for image visualization and analysis. Our group as well has experience with this tool which makes it easier to use.

**Jupyter Notebook -** The use of this notebook would be our coding environment, library integrations, and where our findings will be outputted. Jupyter Notebook is a great development environment that allows many libraries and can be integrated with visualization and our group can take advantage of these capabilities to do cell segmentation.

**OpenCV -** This tool is used for our image pre-processing, processing, and analysis.This library that can be used for image processing with edge detection, and segmentation. Our group wants to use this library because it is a well known documented preprocessing library for image analysis

**Scikit-Image-** This tool is used for our image pre-processing, processing, and analysis. Scikit-image can help with segmentation, and feature extraction.

**NumPy-** This tool is used for computations for image processing. This library can handle image data and do mathematical functions on the arrays.

**ITKSnap-** Great for segmentation, could be a good alternative for cell segmentation, if the python source code does not produce favorable results.

**SciPy-** This method would then allow us to brighten the image so that we can see dark spots where the cells are located in the sample data from the dataset found.

- **Pipeline of Data Processing:**

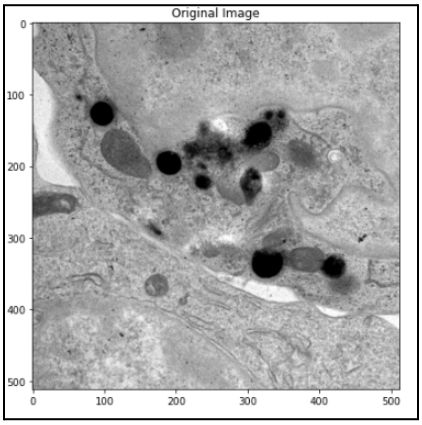



**Figure 2**

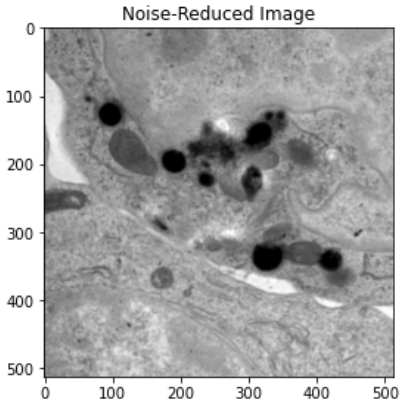



**Figure 3**

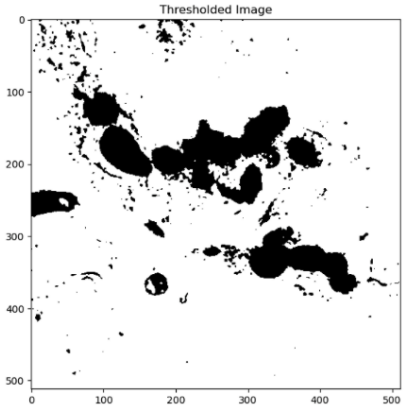



**Figure 4**

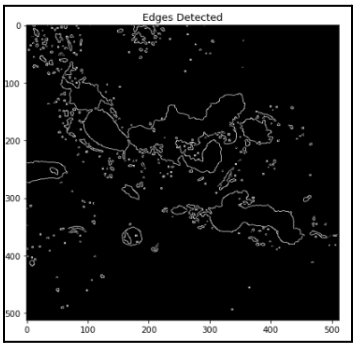



**Figure 5**

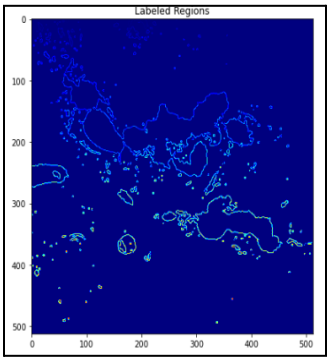



**Figure 6**

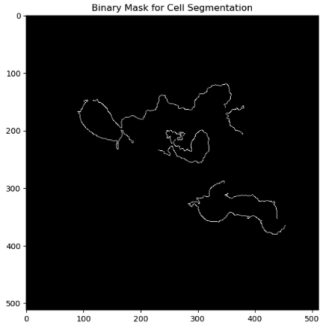



**Figure 7**

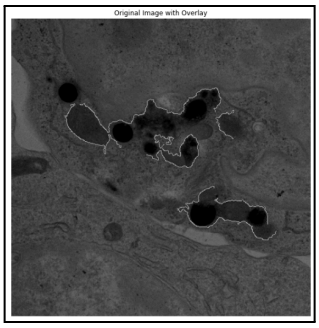



**Figure 8**

1. The first step is to find an Image and load it into the program. The image should have Cells located within the image as seen in **Figure 1.**
2. Convert the image to grayscale using cv2.cvtColor to get rid of any potential colours. This allows for the image to be only shades of grey and gets rid of any complexities within the image potentially.
3. Reduce the Noise from the Image using a Gaussian Filter, cv2.GaussianBlur this helps to reduce the noise of the image as indicated by **Figure 3**.
4. Increase contrast by applying cv2.threshold to change the image contrast as seen in **Figure 4**, this will later help for edge detection.
5. Using cv2.Canny take the updated image and any colour within the range 30-100 to detect and find edges as seen in **Figure 5.**
6. Obtain Labels and Stats using cv2.connectedComponentsWithStats. (**Figure 6)**
7. Based on the Area given from the edges and Labels use a threshold to determine areas for cell segmentation and then apply the Binary Mask. (**Figure 7)**
8. Once Complete overlay with a transparency value of 0.5 over the original image as seen in **Figure 8.**

**Results:**

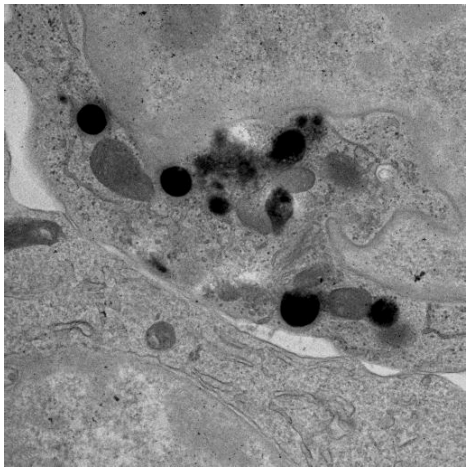


**Figure 9**

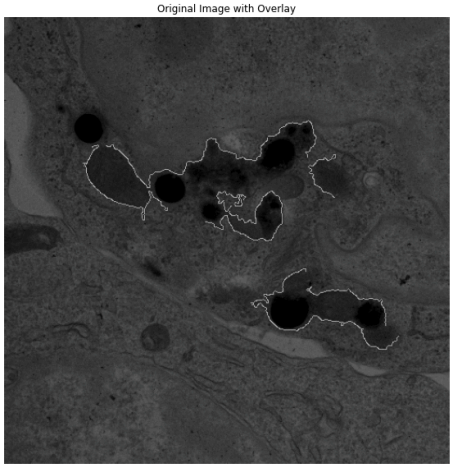



**Figure 10**

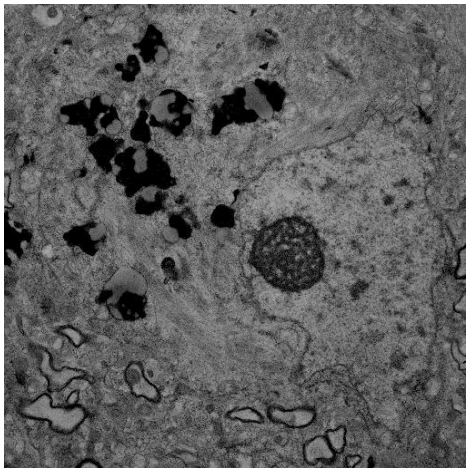


**Figure 11**

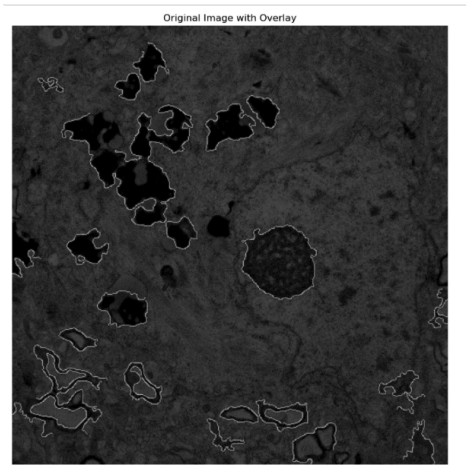



**Figure 12**

When Looking through the results and methods of the program there are some limitation of the overall design process. The first part of the methods after loading in the image are key as it allows for there to be a consistency and have the image be turned to grey scale. The reasoning behind this is because most image processing tools in the medical field often do so, so that it allows for clearer visibility of organs, cells, and tissues therefore reducing complexities that colours may bring. However this step may not be needed if the image is already seen as a grey scale but for consistency it is required. The next issue can be seen through Figure 4 to Figure 8, the figures show that noise is also picked up from the image when applying the threshold and edge detection as possible cells and in Figure 7 the image processing application is not able to detect some cells. Figure 8 also shows that it sometimes has trouble detecting whole cells and is instead doing a generalized area when creating the binary masks.

The data used was from the immunological synapse image and after trying the same process on the second image for the Alzheimer's patient we can see from Figure 11, the original image, and Figure 12, the image processed image, that the results are better. This new image has less noise and has clearer cells and therefore effectively is able to encapsulate a higher ratio of cells to noise when performing the cell segmentations and the mask. Showing that more noise reduction and altering the threshold may be needed so that the model can more effectively perform Cell Segmentations.

## Discussions:

The original motivation for the project was to perform segmentations and we have been able to do so using various tools to create a mask for cell segmentation where the program can help indicate if the pixel in the image is a cell or pertains to the background. However as seen with Figure 12 and Figure 10 different images may not have the most accurate segmentation and this is due to our thresholds, noise, and contrasting of the image. Therefore, if an image is too dark around cells the model will have a harder time to detect the cells. There is also an issue where the edge detection does not function properly if there is too much noise or other outliers in the image that do not get smoothed out with the gaussian filter. Overall our Image Processing can produce masks around cells however it sometimes has difficulties but the

overall region of cells are accurate for the most part and can still greatly help with the main objective in mind with the results it produces, as it effectively takes the image and creates a mask over it.

If a similar project were to take inspiration we would advise them to use the cv2 methods that we had used to detect edges and alter the threshold based on their needs as it greatly helped us find the cells and create the masks. Lastly we would recommend them to alter the image by smoothing it out even more to reduce the total noise of the image. Noise and Contrast from the original image can greatly affect a programs ability to detect edges when creating masks and may produce unwanted results.

## Conclusion:

The Overall objective for this project was to tackle a major problem in cell imaging being segmentation. We created segmentation masks to help indicate if a pixel in a image was a cell or pertained to the background. This would allow for greater efficiency within the biomedical imaging process for our researchers, as it allows for researchers to spend more time analyzing these cells.
We did this by changing the contrast of the image, and reducing the noise using a gaussian filter. We then used a threshold to change the image's appearance by changing the values of pixel that fall within a certain range. Lastly we created a mask by detecting edges and labeling then drawing regions around the cells that were detected and later applied a transparent level and overlaid it over the original image. We were able to produce results where the cells have been detected correctly however some cells as mentioned before were not able to be detected due to our thresholds in other images. A solution however would be to either alter the threshold manually every time dependent on the image as some cells appear darker in images compared to others. However, another possible way of doing this is by detecting the overall “darkness” of the image using tools such as a histogram and then calculating for the threshold to be a variable rate so each image has its own threshold.

### References:


a. Isahit. (2024, March 1). *Why to use grayscale conversion during image processing?* https://www.isahit.com/blog/why-to-use-grayscale-conversion-during-image-processing#:~:text=It%20helps%20in%20simplifying%20algorithms,It%20enhances%20easy%20visualisation.
b. Wong, W. W. (2010). *Datasets. Cellimagelibrary.org.* http://www.cellimagelibrary.org/pages/datasets
c. Wong, W. W. (n.d.). *The Cell Image Library.* Www.cellimagelibrary.org. Retrieved April 7, 2024, from http://www.cellimagelibrary.org/images/CCDB_3632
d. *The Cell Image Library.* (n.d.). Www.cellimagelibrary.org. Retrieved April 7, 2024, from http://www.cellimagelibrary.org/project/P1700

## Appendices:

BMEN 509

Final Project

Tien Dat Johny Do & Ibrahim Asad

### Objective 1 - Find Cell image from Cell image library

In [...]

```
# Load in image from dataset
# Image is Tomogram of the immunological synapse between a human cytotoxic T lymphocyte (CTL) and a target cell

import numpy as np
import matplotlib.pyplot as plt
from scipy.ndimage import gaussian_filter
import cv2
from skimage import exposure

plt.figure(figsize=(50, 50))

#knee_ct2.tif Visualizing
image_path = 'BMEN 509 - Segmentation.jpg'
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
plt.subplot(6,1,1); plt.imshow(image, cmap = 'gray')
plt.title("Original Segmentation Image")
```

Out[...]

```
Text(0.5, 1.0, 'Original Segmentation Image')
```

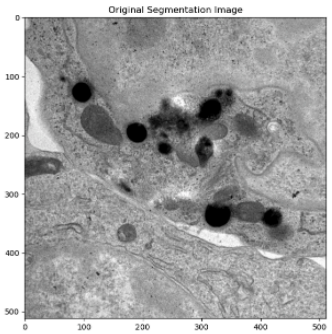



**Figure 13**

### Objective 2 - Noise reduction, Contrast, and Normalization

In [...]

```
# Load image
image_path = 'BMEN 509 - Segmentation.jpg'
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Noise Reduction
image_smoothed = gaussian_filter(image, sigma=1)

# Contrast Enhancement
# Method 1: Histogram Equalization
image_equalized = cv2.equalizeHist(image_smoothed)

# Method 2: Contrast Stretching
p2, p98 = np.percentile(image_smoothed, (2, 98))
image_stretched = exposure.rescale_intensity(image_smoothed, in_range=(p2, p98))

# Normalization
image_normalized = cv2.normalize(image_stretched, None, alpha=0, beta=1, norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_32F)

# Displaying images
plt.figure(figsize=(20, 20))

plt.subplot(4, 1, 1)
plt.imshow(image, cmap='gray')
plt.title("Original Image")

plt.subplot(4, 1, 2)
plt.imshow(image_smoothed, cmap='gray')
plt.title("Noise-Reduced Image")

plt.subplot(4, 1, 3)
plt.imshow(image_equalized, cmap='gray')
plt.title("Histogram Equalized Image")

plt.subplot(4, 1, 4)
plt.imshow(image_normalized, cmap='gray')
plt.title("Normalized Image")

plt.show()
```

**Figure 14**

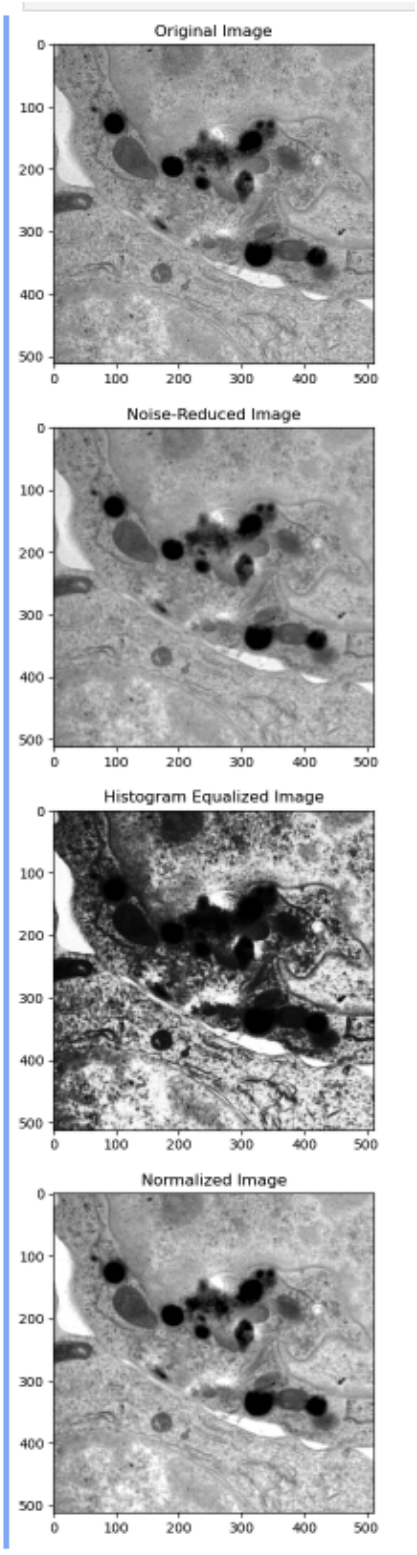

Original Image
Noise-Reduced Image
Histogram Equalized Image
Normalized Image
0
100
200
300
400
500

**Figure 15**

Objective 3 - Thresholding, edge detection, and final refinements

```
In [...
# Noise Reduction
image_smoothed = cv2.GaussianBlur(image, (5, 5), 0)

# Thresholding
_, thresh = cv2.threshold(image_smoothed, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)

# Edge Detection
edges = cv2.Canny(thresh, 30, 100)

# Refinement (Optional)
# You can apply morphological operations like dilation or erosion to refine the edges if needed.
# Example: edges = cv2.dilate(edges, None, iterations=3)

# Displaying images
plt.figure(figsize=(15, 15))

plt.subplot(2, 2, 1)
plt.imshow(image, cmap='gray')
plt.title("Original Image")

plt.subplot(2, 2, 2)
plt.imshow(thresh, cmap='gray')
plt.title("Thresholded Image")

plt.subplot(2, 2, 3)
plt.imshow(edges, cmap='gray')
plt.title("Edges Detected")

plt.show()
```

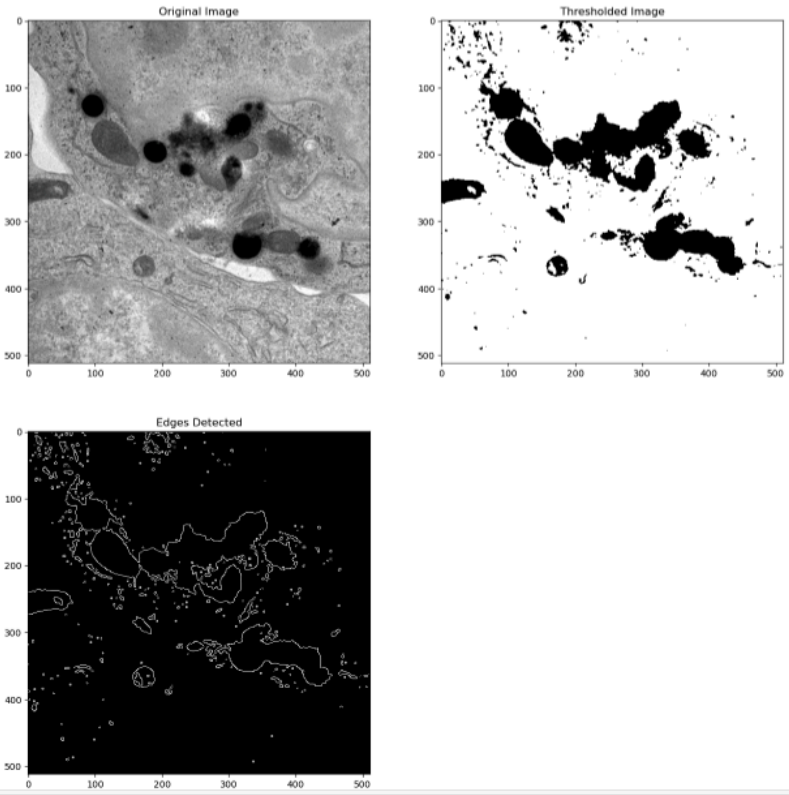



**Figure 16**

## Objective 4 - Label area/regions

In [ ]:

```
# Noise Reduction
image_smoothed = cv2.GaussianBlur(image, (5, 5), 0)

# Thresholding
_, thresh = cv2.threshold(image_smoothed, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)

# Edge Detection
edges = cv2.Canny(thresh, 30, 100)

# Labeling
num_labels, labels, stats, centroids = cv2.connectedComponentsWithStats(edges)

# Display labeled regions
label_image = np.zeros_like(image)
for label in range(1, num_labels):
    label_image[labels == label] = label * 255 / num_labels

# Displaying images
plt.figure(figsize=(15, 15))

plt.subplot(2, 2, 1)
plt.imshow(image, cmap='gray')
plt.title("Original Image")

plt.subplot(2, 2, 2)
plt.imshow(thresh, cmap='gray')
plt.title("Thresholded Image")

plt.subplot(2, 2, 3)
plt.imshow(edges, cmap='gray')
plt.title("Edges Detected")

plt.subplot(2, 2, 4)
plt.imshow(label_image, cmap='jet')
plt.title("Labeled Regions")

plt.show()
```

**Figure 17**

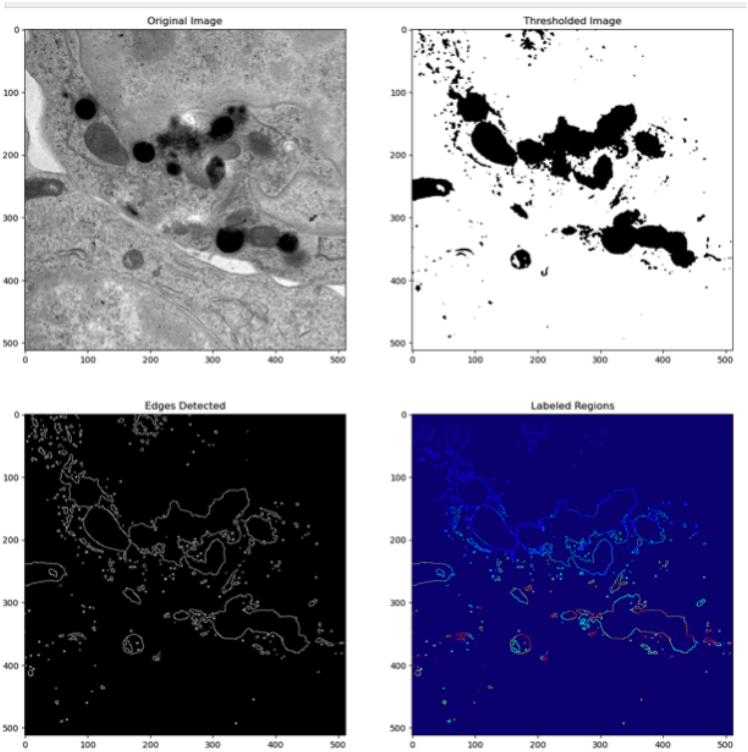



**Figure 18**

## Objective 5 - Create Mask

In [...

```
# Noise Reduction
image_smoothed = cv2.GaussianBlur(image, (5, 5), 0)

# Thresholding
_, thresh = cv2.threshold(image_smoothed, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)

# Edge Detection
edges = cv2.Canny(thresh, 30, 100)

# Labeling
num_labels, labels, stats, centroids = cv2.connectedComponentsWithStats(edges)

# Creating a binary mask for cell segmentation
binary_mask = np.zeros_like(image, dtype=np.uint8)
for label in range(1, num_labels):
    # Filter regions based on area (adjust the min and max area thresholds as needed)
    area = stats[label, cv2.CC_STAT_AREA]
    if 100 < area < 10000:  # Example area thresholds for cells
        # Draw the region on the binary mask
        binary_mask[labels == label] = 255

# Displaying images
plt.figure(figsize=(15, 15))

plt.subplot(2, 2, 1)
plt.imshow(image, cmap='gray')
plt.title("Original Image")

plt.subplot(2, 2, 2)
plt.imshow(thresh, cmap='gray')
plt.title("Thresholded Image")

plt.subplot(2, 2, 3)
plt.imshow(edges, cmap='gray')
plt.title("Edges Detected")

plt.subplot(2, 2, 4)
plt.imshow(binary_mask, cmap='gray')
plt.title("Binary Mask for Cell Segmentation")

plt.show()
```

**Figure 19**

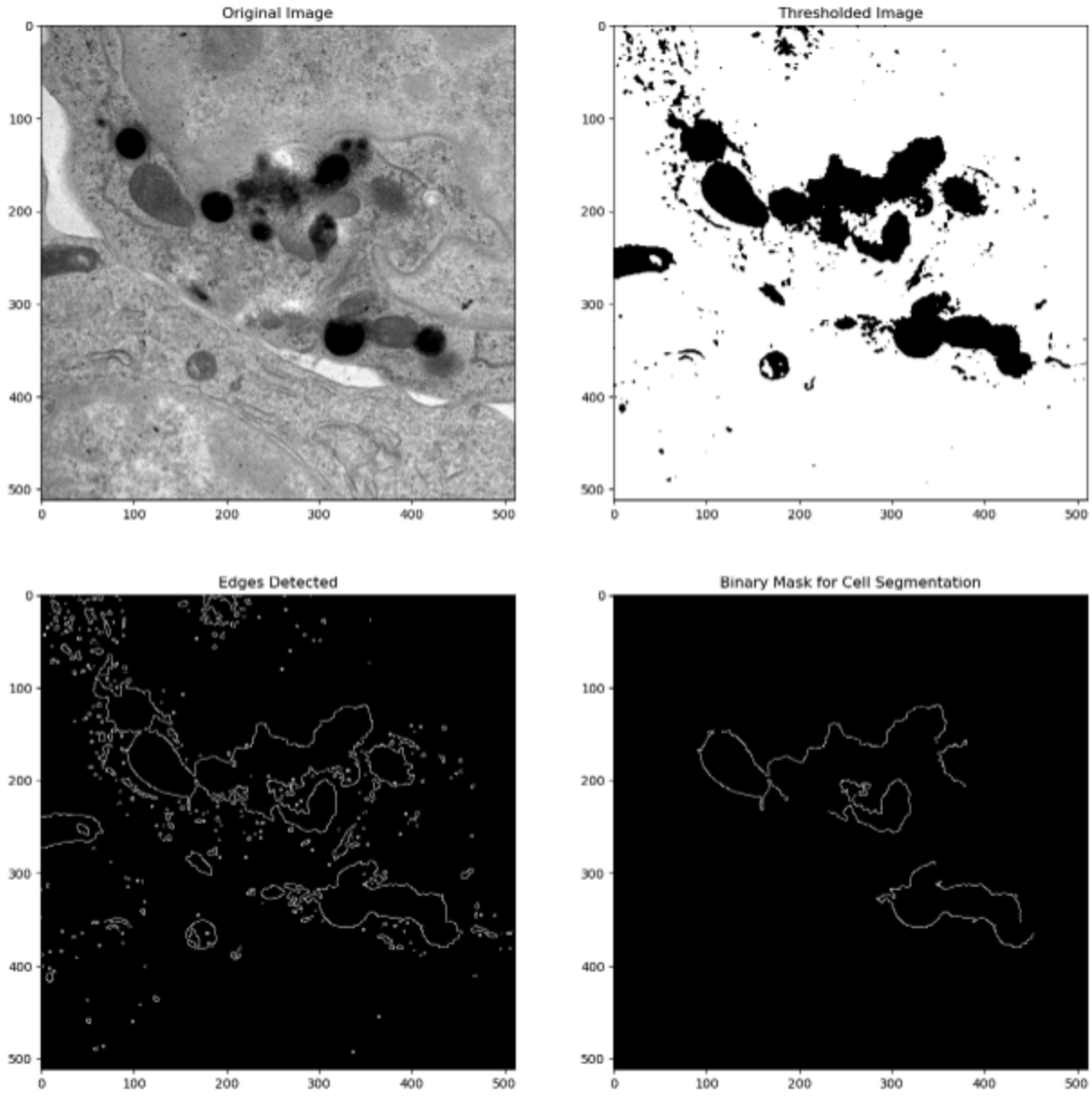

Original Image
Thresholded Image
Edges Detected
Binary Mask for Cell Segmentation
0
100
200
300
400
500

**Figure 20**

## Objective 6 - Visulization of the Mask and Original Image

In [ ]:

```
# Convert image to grayscale

image = cv2.imread(image_path)

# Convert image to grayscale
image_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
# Noise Reduction
image_smoothed = cv2.GaussianBlur(image_gray, (5, 5), 0)

# Thresholding
_, thresh = cv2.threshold(image_smoothed, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)

# Edge Detection
edges = cv2.Canny(thresh, 30, 100)

# Labeling
num_labels, labels, stats, centroids = cv2.connectedComponentsWithStats(edges)

# Creating a binary mask for cell segmentation
binary_mask = np.zeros_like(image_gray, dtype=np.uint8)
for label in range(1, num_labels):
    # Filter regions based on area (adjust the min and max area thresholds as needed)
    area = stats[label, cv2.CC_STAT_AREA]
    if 100 < area < 10000:  # Example area thresholds for cells
        # Draw the region on the binary mask
        binary_mask[labels == label] = 255

# Apply transparency to the binary mask
mask_with_alpha = cv2.merge([binary_mask, binary_mask, binary_mask])

# Set transparency level (adjust as needed)
alpha = 0.5

# Blend the original image and the mask
overlay = cv2.addWeighted(image, 1-alpha, mask_with_alpha, alpha, 0)

# Display the overlay
plt.figure(figsize=(10, 10))
plt.imshow(cv2.cvtColor(overlay, cv2.COLOR_BGR2RGB))
plt.title("Original Image with Overlay")
plt.axis('off')
plt.show()
```

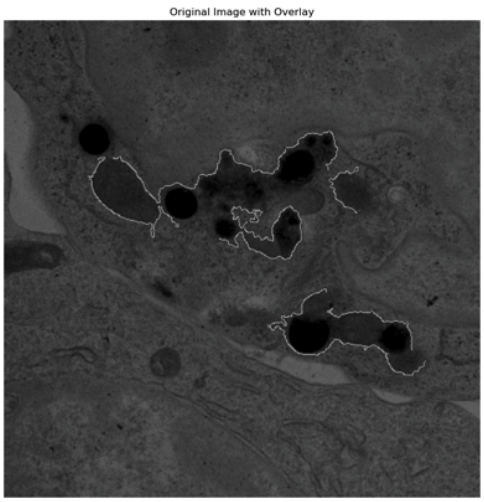



**Figure 21**

In [...

```
# Convert image to grayscale
image_path = 'Alz.jpg'
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

image = cv2.imread(image_path)

# Convert image to grayscale
image_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
# Noise Reduction
image_smoothed = cv2.GaussianBlur(image_gray, (5, 5), 0)

# Thresholding
_, thresh = cv2.threshold(image_smoothed, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)

# Edge Detection
edges = cv2.Canny(thresh, 30, 100)

# Labeling
num_labels, labels, stats, centroids = cv2.connectedComponentsWithStats(edges)

# Creating a binary mask for cell segmentation
binary_mask = np.zeros_like(image_gray, dtype=np.uint8)
for label in range(1, num_labels):
    # Filter regions based on area (adjust the min and max area thresholds as needed)
    area = stats[label, cv2.CC_STAT_AREA]
    if 100 < area < 10000:  # Example area thresholds for cells
        # Draw the region on the binary mask
        binary_mask[labels == label] = 255

# Apply transparency to the binary mask
mask_with_alpha = cv2.merge([binary_mask, binary_mask, binary_mask])

# Set transparency level (adjust as needed)
alpha = 0.5

# Blend the original image and the mask
overlay = cv2.addWeighted(image, 1-alpha, mask_with_alpha, alpha, 0)

# Display the overlay
plt.figure(figsize=(10, 10))
plt.imshow(cv2.cvtColor(overlay, cv2.COLOR_BGR2RGB))
plt.title("Original Image with Overlay")
plt.axis('off')
plt.show()
```

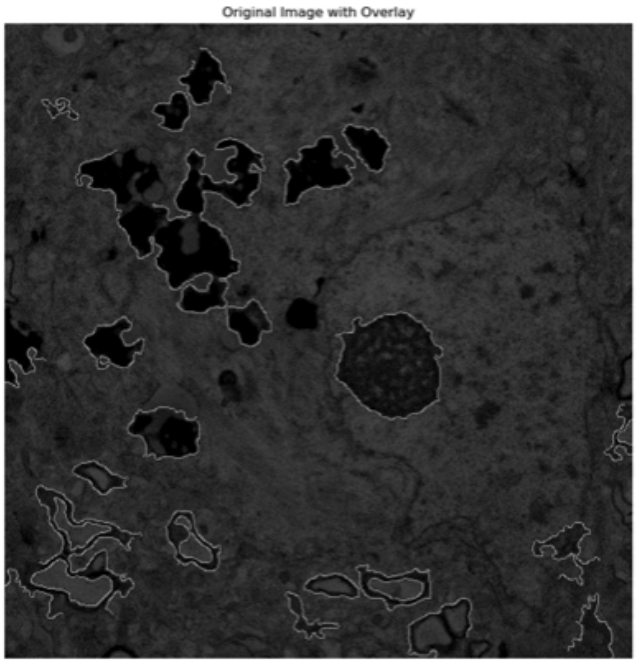



**Figure 22**

# BMEN 509

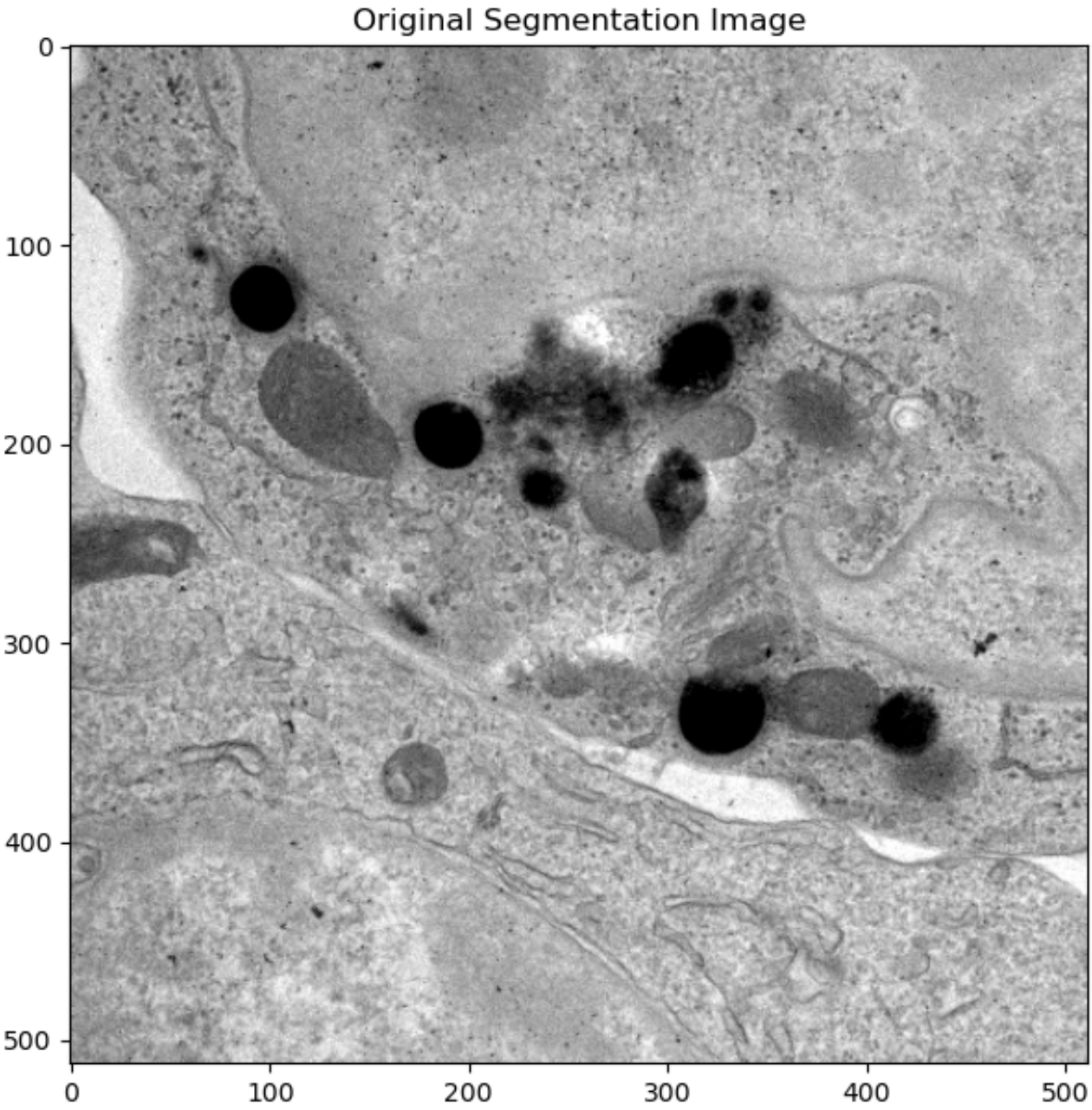
## Final Project

Tien Dat Johny Do & Ibrahim Asad

# Objective 1 - Find Cell image from Cell image library

In [1]:
```
# Load in image from dataset
# Image is Tomogram of the immunological synapse between a human cytotoxic T

import numpy as np
import matplotlib.pyplot as plt
from scipy.ndimage import gaussian_filter
import cv2
from skimage import exposure

plt.figure(figsize=(50, 50))

#knee_ct2.tif Visualizing
image_path = 'BMEN 509 - Segmentation.jpg'
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
plt.subplot(6,1,1); plt.imshow(image, cmap = 'gray')
plt.title("Original Segmentation Image")
```

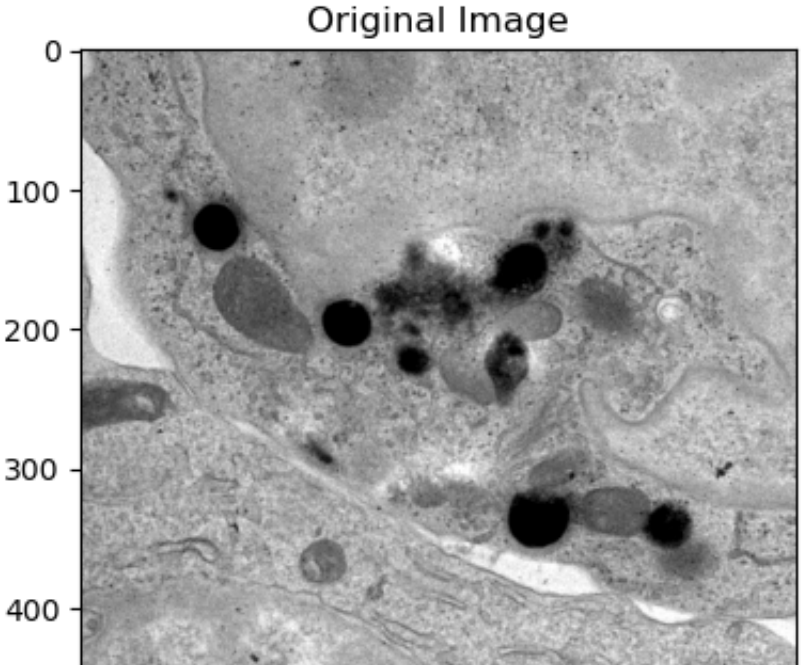
Out[1]:
```
Text(0.5, 1.0, 'Original Segmentation Image')
```

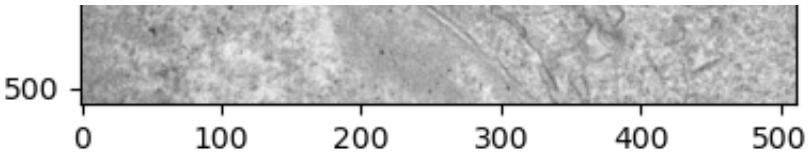


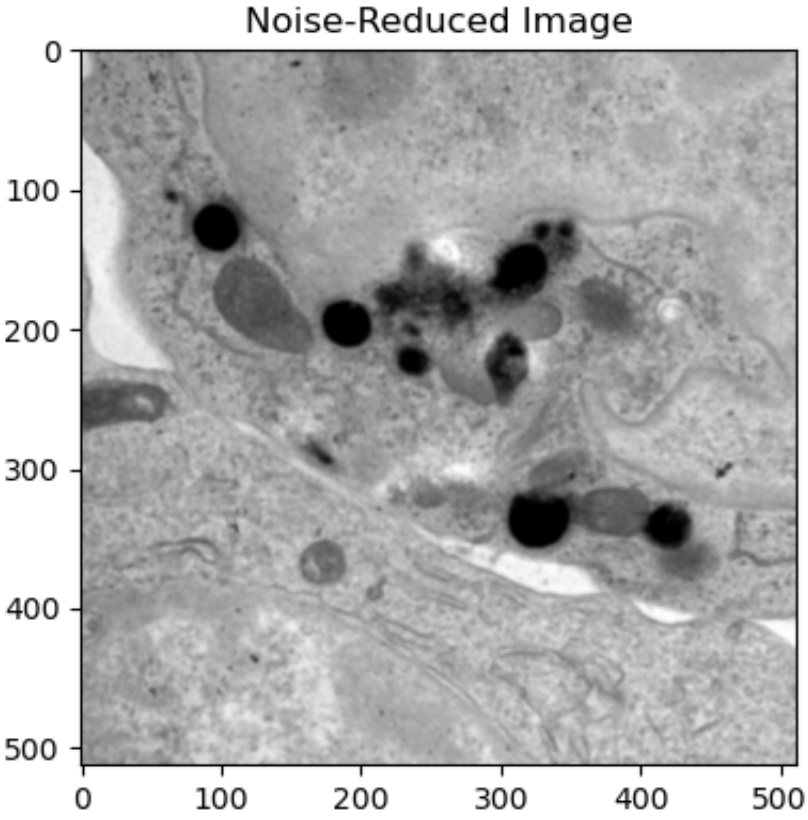

# Objective 2 - Noise reduction, Contrast, and Normalization

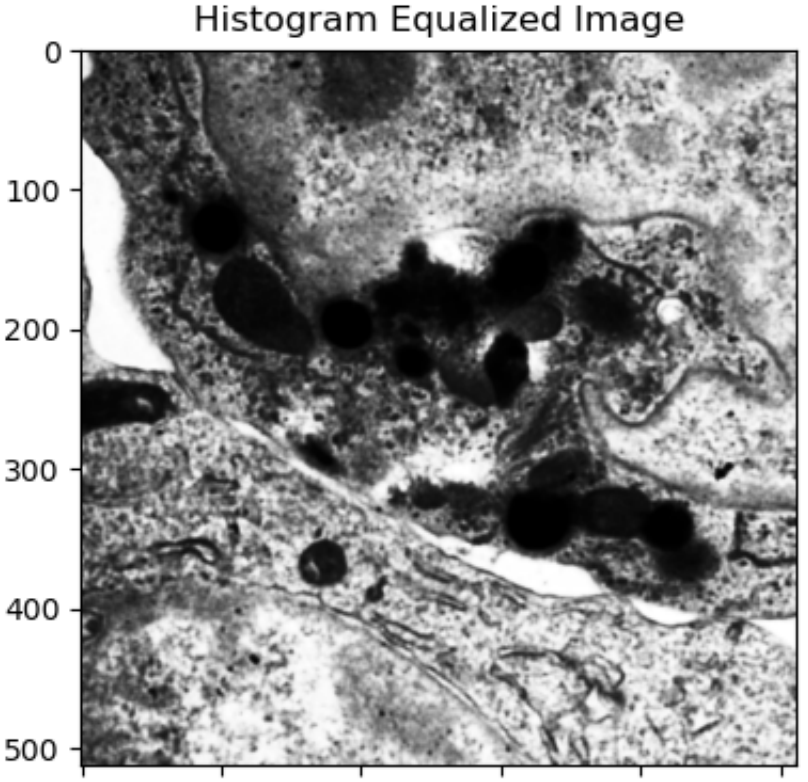
In [2]:
```
# Load image
image_path = 'BMEN 509 - Segmentation.jpg'
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Noise Reduction
image_smoothed = gaussian_filter(image, sigma=1)
```

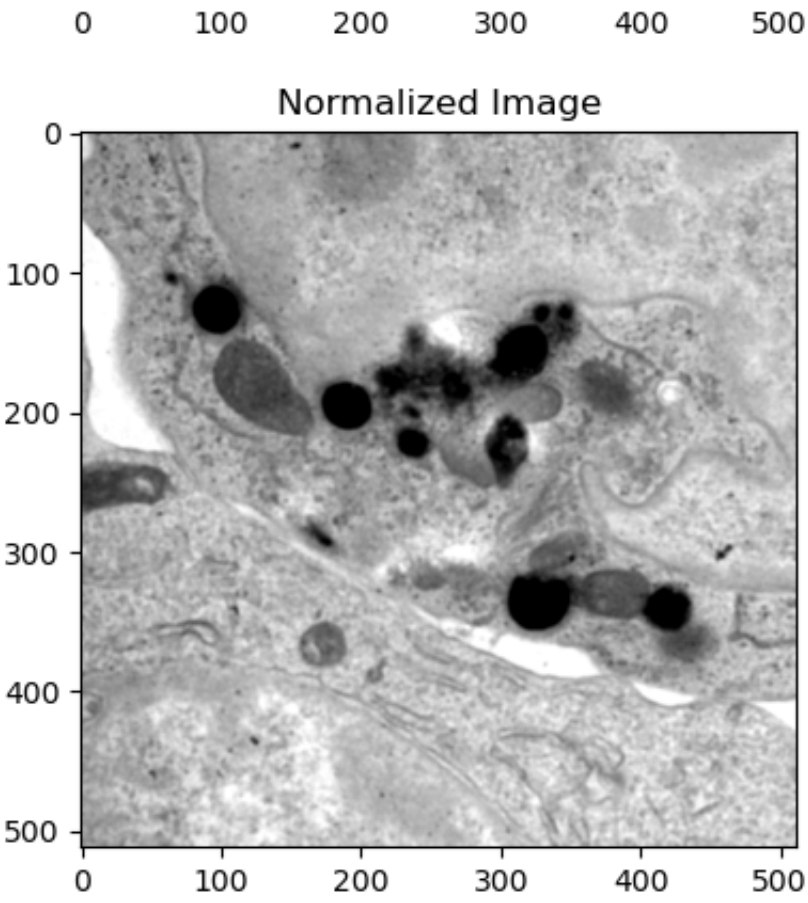
```
# Contrast Enhancement
# Method 1: Histogram Equalization
image_equalized = cv2.equalizeHist(image_smoothed)

# Method 2: Contrast Stretching
p2, p98 = np.percentile(image_smoothed, (2, 98))
image_stretched = exposure.rescale_intensity(image_smoothed, in_range=(p2, p

# Normalization
image_normalized = cv2.normalize(image_stretched, None, alpha=0, beta=1, nor

# Displaying images
plt.figure(figsize=(20, 20))

plt.subplot(4, 1, 1)
plt.imshow(image, cmap='gray')
plt.title("Original Image")

plt.subplot(4, 1, 2)
plt.imshow(image_smoothed, cmap='gray')
plt.title("Noise-Reduced Image")

plt.subplot(4, 1, 3)
plt.imshow(image_equalized, cmap='gray')
plt.title("Histogram Equalized Image")

plt.subplot(4, 1, 4)
plt.imshow(image_normalized, cmap='gray')
plt.title("Normalized Image")

plt.show()
```




500
0
100
200
300
400
500


Noise-Reduced Image
0
100
200
300
400
500
0
100
200
300
400
500

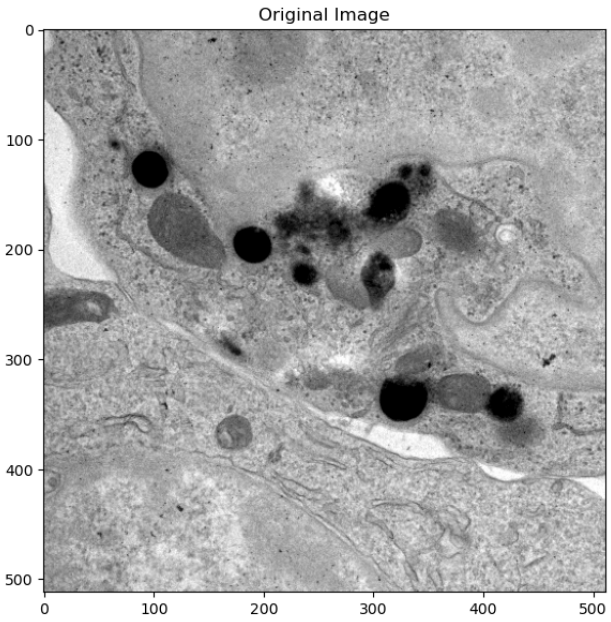

Histogram Equalized Image
0
100
200
300
400
500

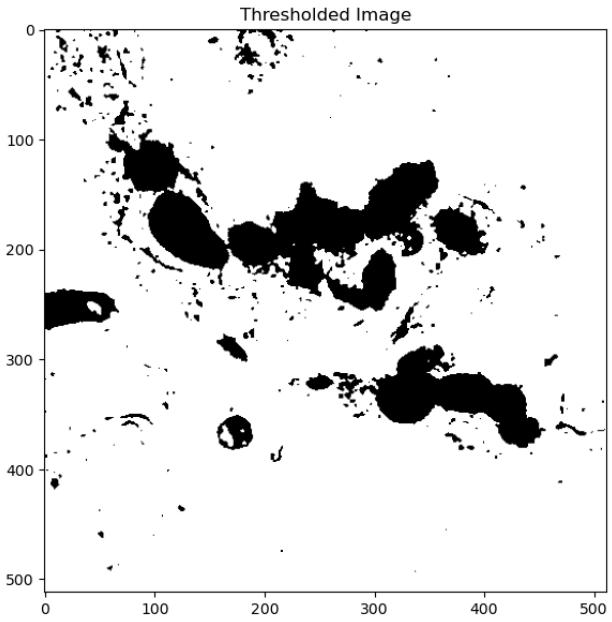


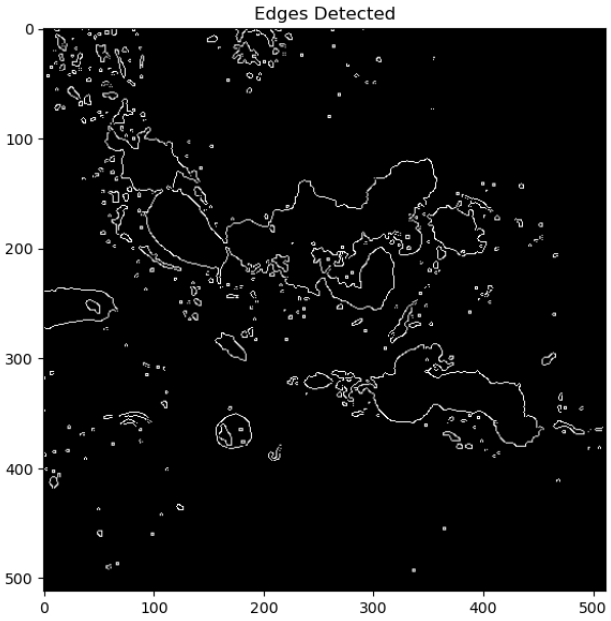

# Objective 3 - Thresholding, edge detection, and final refinements

In [3]:
```
# Noise Reduction
image_smoothed = cv2.GaussianBlur(image, (5, 5), 0)

# Thresholding
_, thresh = cv2.threshold(image_smoothed, 0, 255, cv2.THRESH_BINARY + cv2.TH

# Edge Detection
edges = cv2.Canny(thresh, 30, 100)

# Refinement (Optional)
# You can apply morphological operations like dilation or erosion to refine
# Example: edges = cv2.dilate(edges, None, iterations=3)

# Displaying images
plt.figure(figsize=(15, 15))

plt.subplot(2, 2, 1)
plt.imshow(image, cmap='gray')
```

```
plt.title("Original Image")

plt.subplot(2, 2, 2)
plt.imshow(thresh, cmap='gray')
plt.title("Thresholded Image")

plt.subplot(2, 2, 3)
plt.imshow(edges, cmap='gray')
plt.title("Edges Detected")

plt.show()
```

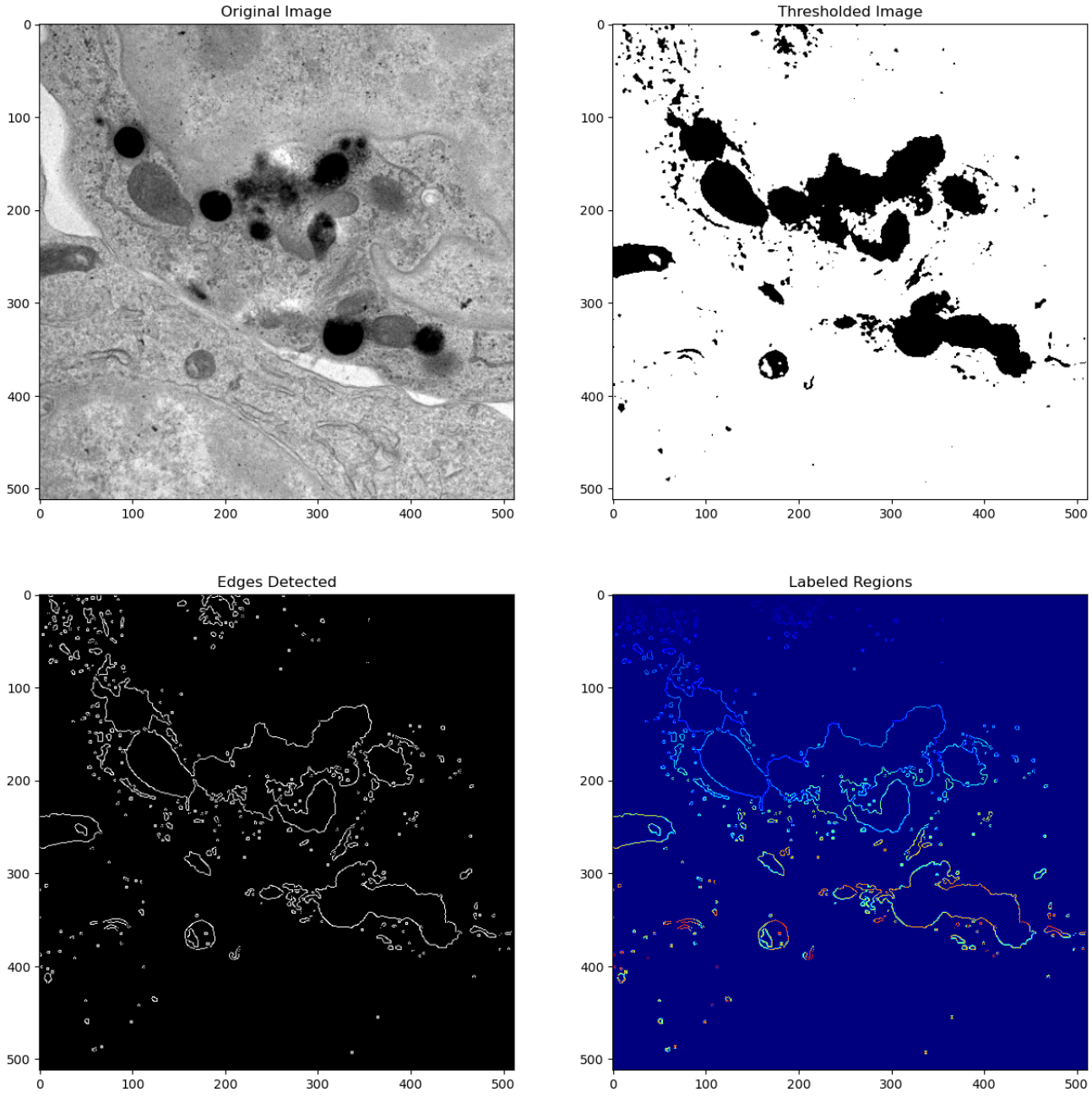


# Objective 4 - Label area/regions

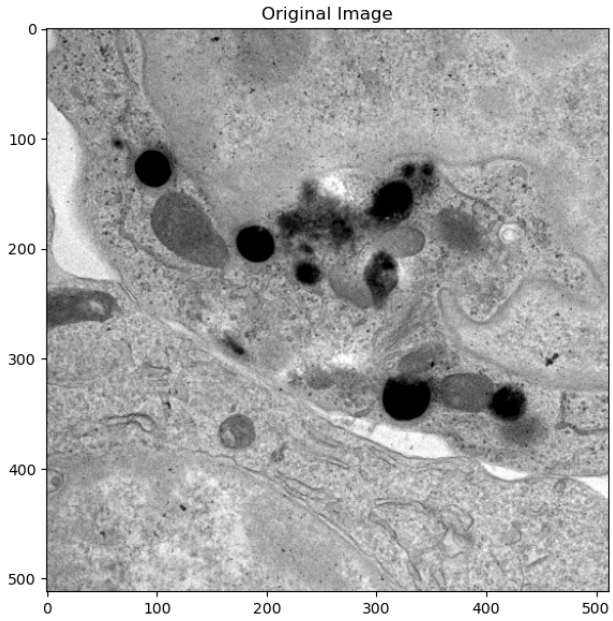
In [4]:
```
# Noise Reduction
image_smoothed = cv2.GaussianBlur(image, (5, 5), 0)

# Thresholding
_, thresh = cv2.threshold(image_smoothed, 0, 255, cv2.THRESH_BINARY + cv2.TH

# Edge Detection
edges = cv2.Canny(thresh, 30, 100)

# Labeling
num_labels, labels, stats, centroids = cv2.connectedComponentsWithStats(edge

# Display labeled regions
label_image = np.zeros_like(image)
for label in range(1, num_labels):
    label_image[labels == label] = label * 255 / num_labels

# Displaying images
plt.figure(figsize=(15, 15))

plt.subplot(2, 2, 1)
plt.imshow(image, cmap='gray')
plt.title("Original Image")

plt.subplot(2, 2, 2)
plt.imshow(thresh, cmap='gray')
plt.title("Thresholded Image")

plt.subplot(2, 2, 3)
plt.imshow(edges, cmap='gray')
plt.title("Edges Detected")

plt.subplot(2, 2, 4)
plt.imshow(label_image, cmap='jet')
plt.title("Labeled Regions")

plt.show()
```

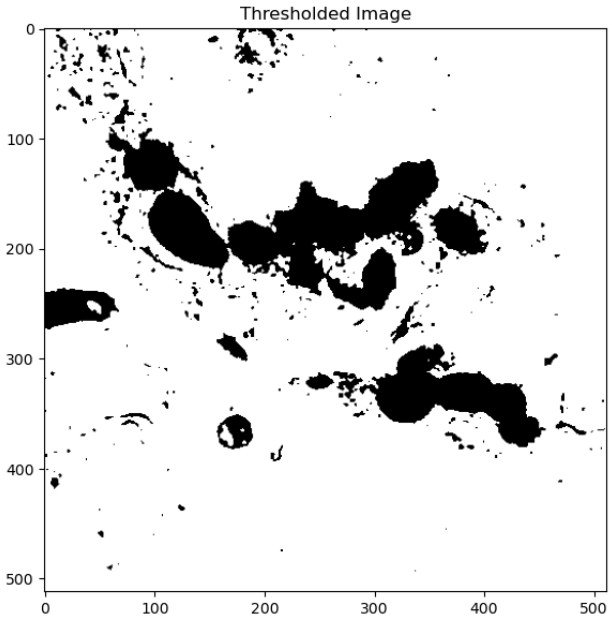


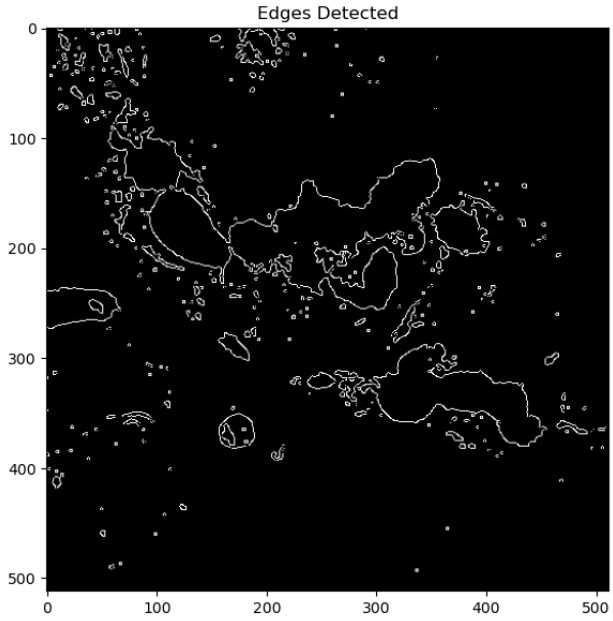

# Objective 5 - Create Mask

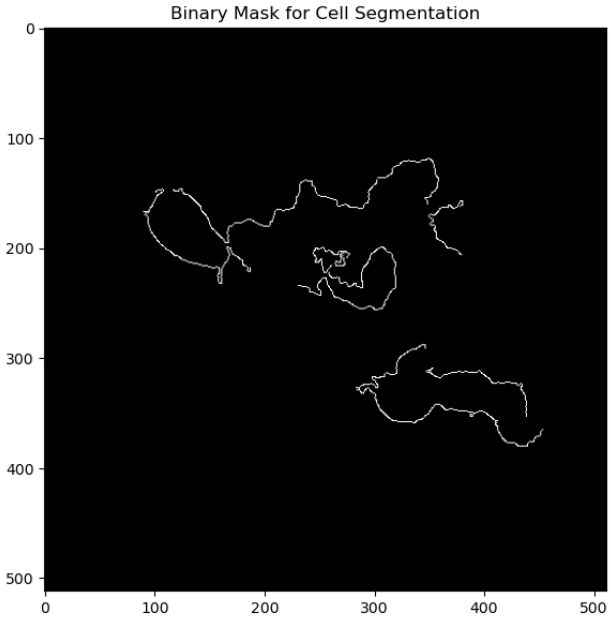
In [5]:
```
# Noise Reduction
image_smoothed = cv2.GaussianBlur(image, (5, 5), 0)

# Thresholding
_, thresh = cv2.threshold(image_smoothed, 0, 255, cv2.THRESH_BINARY + cv2.TH

# Edge Detection
edges = cv2.Canny(thresh, 30, 100)
```

```
# Labeling
num_labels, labels, stats, centroids = cv2.connectedComponentsWithStats(edge

# Creating a binary mask for cell segmentation
binary_mask = np.zeros_like(image, dtype=np.uint8)
for label in range(1, num_labels):
    # Filter regions based on area (adjust the min and max area thresholds a
    area = stats[label, cv2.CC_STAT_AREA]
    if 100 < area < 10000:  # Example area thresholds for cells
        # Draw the region on the binary mask
        binary_mask[labels == label] = 255

# Displaying images
plt.figure(figsize=(15, 15))

plt.subplot(2, 2, 1)
plt.imshow(image, cmap='gray')
plt.title("Original Image")

plt.subplot(2, 2, 2)
plt.imshow(thresh, cmap='gray')
plt.title("Thresholded Image")

plt.subplot(2, 2, 3)
plt.imshow(edges, cmap='gray')
plt.title("Edges Detected")

plt.subplot(2, 2, 4)
plt.imshow(binary_mask, cmap='gray')
plt.title("Binary Mask for Cell Segmentation")

plt.show()
```



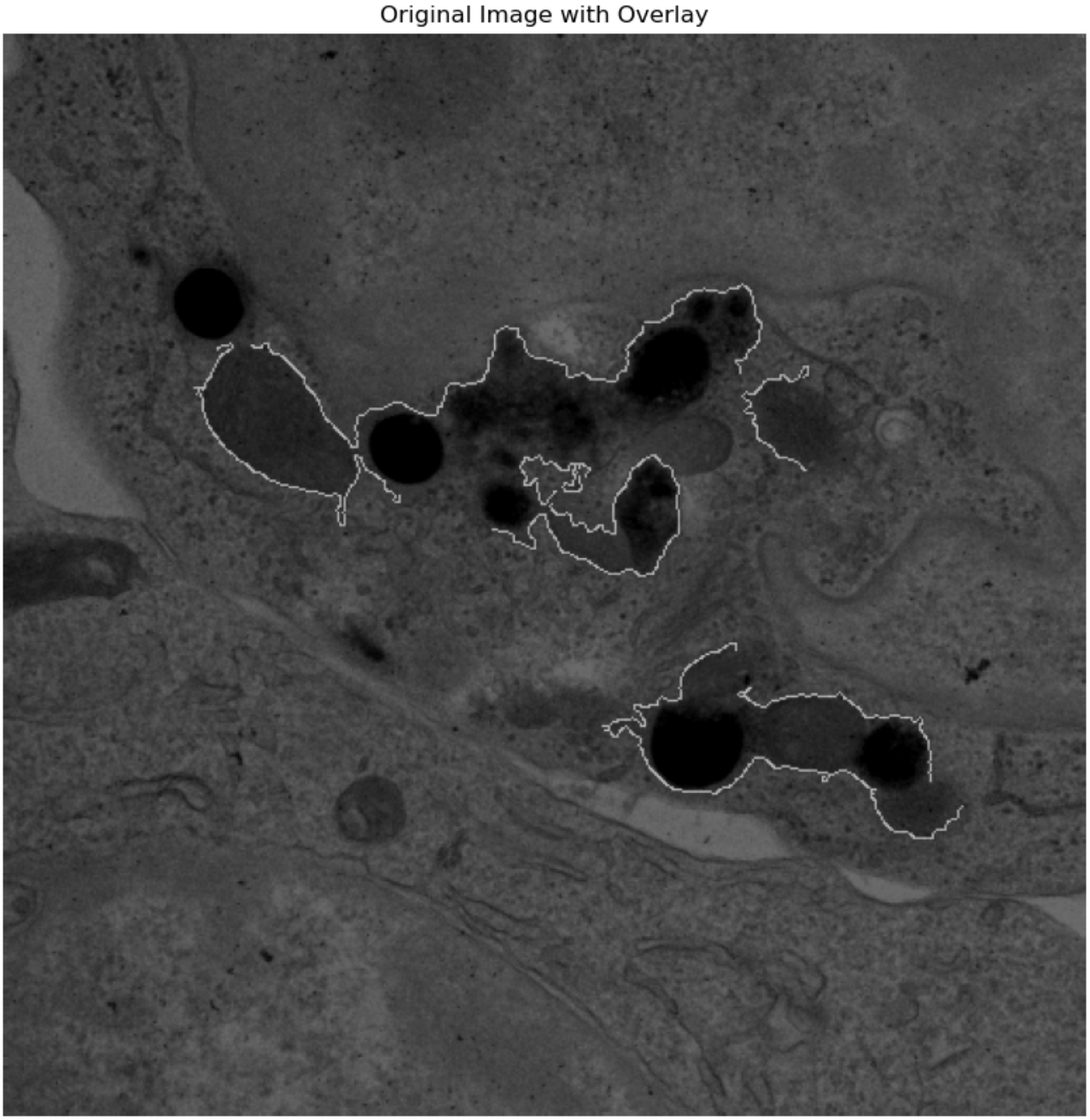




# Objective 6 - Visulization of the Mask and Original Image

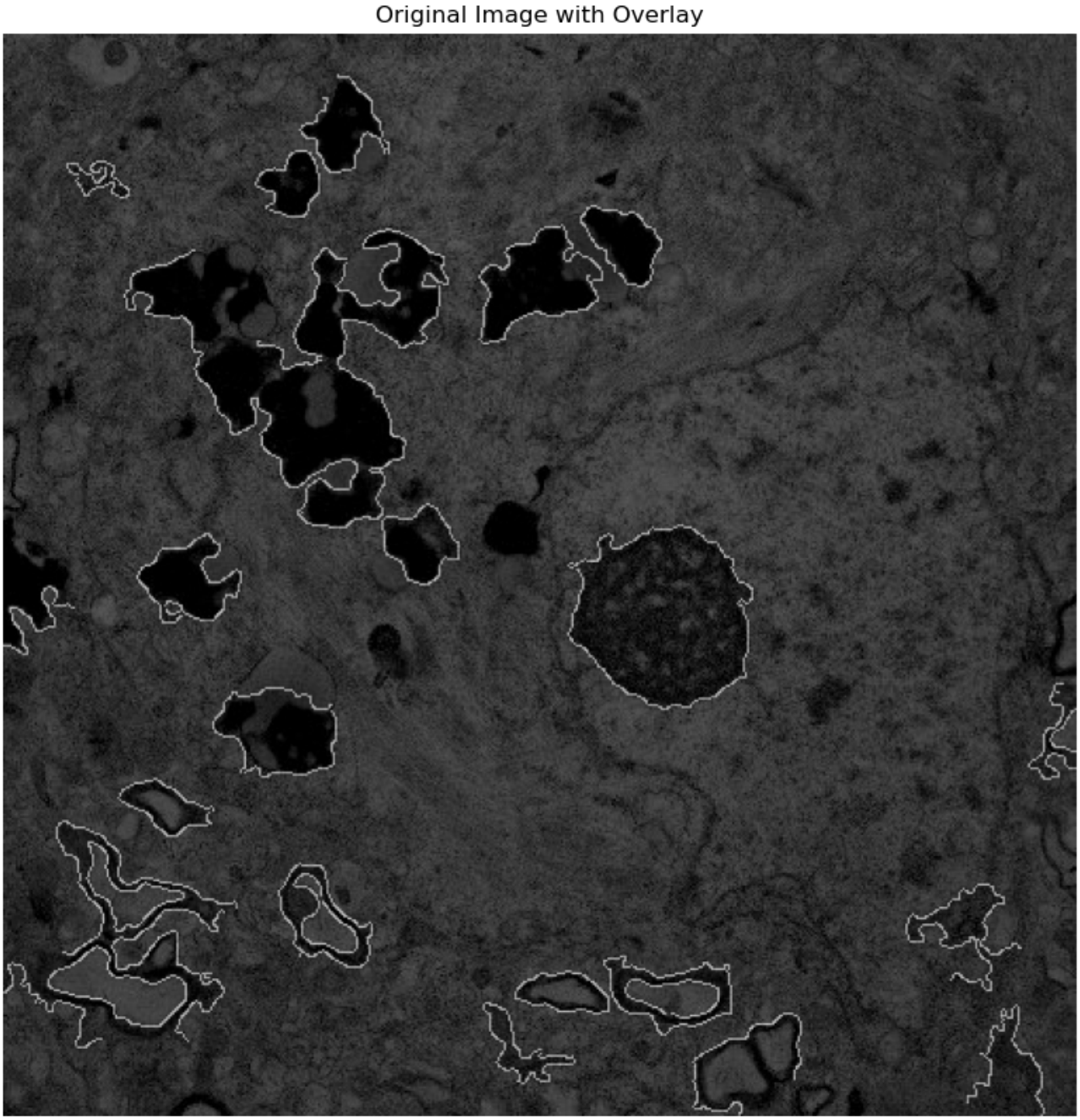
In [6]:
```
# Convert image to grayscale

image = cv2.imread(image_path)

# Convert image to grayscale
image_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
# Noise Reduction
```

```
image_smoothed = cv2.GaussianBlur(image_gray, (5, 5), 0)

# Thresholding
_, thresh = cv2.threshold(image_smoothed, 0, 255, cv2.THRESH_BINARY + cv2.TH

# Edge Detection
edges = cv2.Canny(thresh, 30, 100)

# Labeling
num_labels, labels, stats, centroids = cv2.connectedComponentsWithStats(edge

# Creating a binary mask for cell segmentation
binary_mask = np.zeros_like(image_gray, dtype=np.uint8)
for label in range(1, num_labels):
    # Filter regions based on area (adjust the min and max area thresholds a
    area = stats[label, cv2.CC_STAT_AREA]
    if 100 < area < 10000:  # Example area thresholds for cells
        # Draw the region on the binary mask
        binary_mask[labels == label] = 255

# Apply transparency to the binary mask
mask_with_alpha = cv2.merge([binary_mask, binary_mask, binary_mask])

# Set transparency level (adjust as needed)
alpha = 0.5

# Blend the original image and the mask
overlay = cv2.addWeighted(image, 1-alpha, mask_with_alpha, alpha, 0)

# Display the overlay
plt.figure(figsize=(10, 10))
plt.imshow(cv2.cvtColor(overlay, cv2.COLOR_BGR2RGB))
plt.title("Original Image with Overlay")
plt.axis('off')
plt.show()
```




In [7]:
```
# Convert image to grayscale
image_path = 'Alz.jpg'
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

image = cv2.imread(image_path)

# Convert image to grayscale
image_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
# Noise Reduction
image_smoothed = cv2.GaussianBlur(image_gray, (5, 5), 0)

# Thresholding
```

```
_, thresh = cv2.threshold(image_smoothed, 0, 255, cv2.THRESH_BINARY + cv2.TH

# Edge Detection
edges = cv2.Canny(thresh, 30, 100)

# Labeling
num_labels, labels, stats, centroids = cv2.connectedComponentsWithStats(edge

# Creating a binary mask for cell segmentation
binary_mask = np.zeros_like(image_gray, dtype=np.uint8)
for label in range(1, num_labels):
    # Filter regions based on area (adjust the min and max area thresholds a
    area = stats[label, cv2.CC_STAT_AREA]
    if 100 < area < 10000:  # Example area thresholds for cells
        # Draw the region on the binary mask
        binary_mask[labels == label] = 255

# Apply transparency to the binary mask
mask_with_alpha = cv2.merge([binary_mask, binary_mask, binary_mask])

# Set transparency level (adjust as needed)
alpha = 0.5

# Blend the original image and the mask
overlay = cv2.addWeighted(image, 1-alpha, mask_with_alpha, alpha, 0)

# Display the overlay
plt.figure(figsize=(10, 10))
plt.imshow(cv2.cvtColor(overlay, cv2.COLOR_BGR2RGB))
plt.title("Original Image with Overlay")
plt.axis('off')
plt.show()
```




In [ ]:

In [ ]: