Yizhen Jia ID: 111520996

Final for SBU Medical Imaging: Please show all work. Final is due in by 12 noon, May 5. Please send me results (and code) via email. Please put your name and Stony brook id number on any files sent to me. Please make one easily readable pdf file with all your results in addition to the original code.

1. Testing the functionality of image segmentation techniques and comparing their accuracy requires us to work with images for which the correct segmentation is known. Develop some test images by:

(a) Create an image SEG1 containing artificial objects on a background of constant gray-level. Generate simple geometric objects such as squares, rectangles, diamonds, stars, circles, etc., each having a constant gray-level different from that of the background, some of them darker and some brighter than the background. Determine the area of each object and store it in an appropriate form.
(b) Superimpose additive Gaussian noise with a given standard deviation, thus creating an image SEG2.
(c) Superimpose random impulse noise of a given severity over the image SEG2, thus creating an image SEG3.

By varying the shapes of the objects, standard deviation of the Gaussian additive noise, and severity of the impulse noise, sets of controlled properties can be generated. To create a simple set of images for segmentation experiments, make a single image SEG1, apply three levels of Gaussian additive noise, and three levels of impulse noise. You will obtain a set of ten images that will be used in the segmentation ~~problems~~ below. (20 points)

2. To assess the correctness of a segmentation, a set of measures must be developed to allow quantitative comparison among methods. Develop a program for calculating the following two segmentation accuracy indices:

(a) "Relative signed area error" is expressed in percent and computed as:

$$A_{error} = \frac{\sum_{i=1}^{N} T_i - \sum_{j=1}^{M} A_j}{\sum_{i=1}^{N} T_i} \times 100$$

where T_i is the true area of the i-th object and A_j is the measured area of the j-th object, N is the number of objects in the image, M is the number of objects after segmentation. Areas may be expressed in pixels.

(b) "Labelling error" (denoted as L_{error}) is defined as the ratio of the number of incorrectly labeled pixels (object pixels labeled as background as vice versa) and the number of pixels of true objects $\sum_{i=1}^{N} T_i$ according to prior knowledge, and is expressed as percent. (20 points)

3. Implement the following methods for segmentation and apply to the test images created in Problem 1. For each method and each image, quantitatively assess the segmentation accuracy using the indices developed in Problem 2. Compare the segmentation accuracy for individual methods.

(a) Basic thresholding.
(b) Chan-Vese.

(20 points)

4. Take the image heart.jpg and rotate it 15 degrees. Call the new image heart.15.jpg. Align (register the images) using a rigid registration method. Explicitly explain your method. You may any package that you like, just give all the details. (20 points)

5. Take the image heart.jpg and apply the linear heat equation to smooth it. Specifically, write a program which will do the following:

(a) Load the image. Call this $I_0(x, y)$.

(b) Discretize the heat equation

$$\frac{\partial I}{\partial t} = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$$
$$I(x, y, 0) = I_0(x, y).$$

You can use centered differences for the spatial derivatives.

Take $\Delta x = \Delta y = 1, \quad \Delta t = 0.1$.

Please submit all your formulas and your code.

(c) Output the smoothed image with the following number of iterations: n=10, n=100, n=1000.

(20 points)

yizjia_577fq1-3

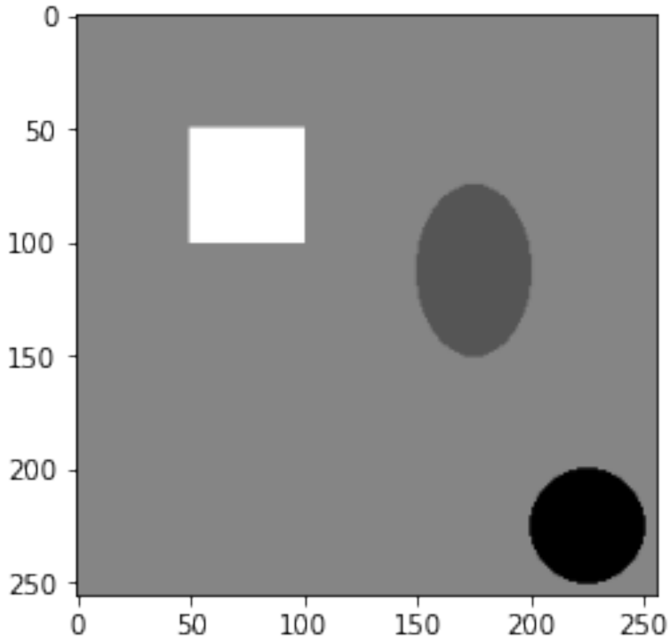
May 4, 2023

1 Problem 1

[92]:
```
from PIL import Image, ImageDraw
import numpy as np
import matplotlib.pyplot as plt
```

[93]:
```
# Open original image
original_image = Image.open('heart.jpg')

# Get the size of the original image
width, height = original_image.size

# Create a new image with a gray background
SEG1 = Image.new('L', (width, height), 128)

# Create a draw object
draw = ImageDraw.Draw(SEG1)

# Define the geometric shapes and their gray-levels
shapes = [
    {'shape': 'rectangle', 'coords': [(50, 50), (100, 100)], 'color': 200},
    {'shape': 'circle', 'coords': [(200, 200), (250, 250)], 'color': 50},
    {'shape': 'ellipse', 'coords': [(150, 75), (200, 150)], 'color': 100},
]

# A dictionary to store area of each shape
area_dict = {}

# Draw each shape on the image
for i, shape in enumerate(shapes):
    if shape['shape'] == 'rectangle':
        draw.rectangle(shape['coords'], fill=shape['color'])
        area_dict[f"object_{i+1}"] = abs((shape['coords'][1][0] -␣
↪shape['coords'][0][0]) * (shape['coords'][1][1] - shape['coords'][0][1]))
    elif shape['shape'] == 'circle':
        draw.ellipse(shape['coords'], fill=shape['color'])
        radius = (shape['coords'][1][0] - shape['coords'][0][0]) / 2
```

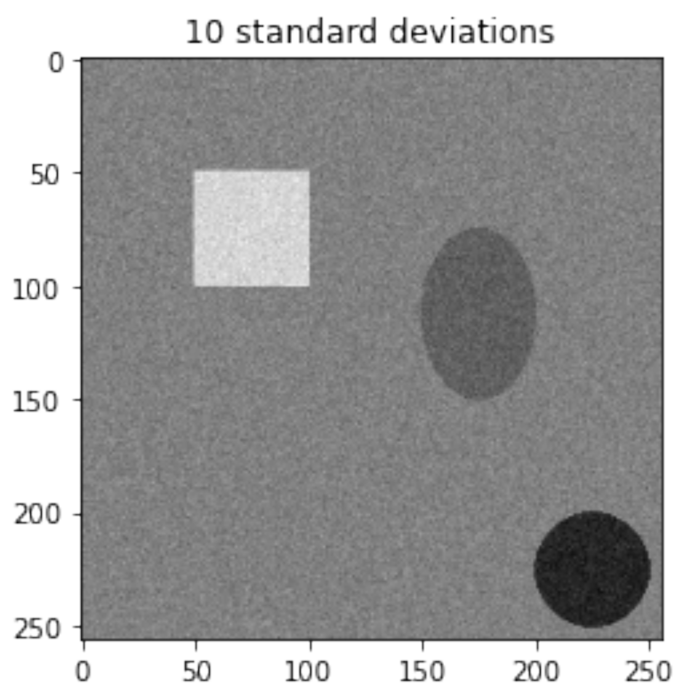
```
        area_dict[f"object_{i+1}"] = np.pi * radius * radius
    elif shape['shape'] == 'ellipse':
        draw.ellipse(shape['coords'], fill=shape['color'])
        radius_x = (shape['coords'][1][0] - shape['coords'][0][0]) / 2
        radius_y = (shape['coords'][1][1] - shape['coords'][0][1]) / 2
        area_dict[f"object_{i+1}"] = np.pi * radius_x * radius_y

# Save the image
SEG1.save('SEG1.jpg')

# Display the image
plt.imshow(SEG1, cmap='gray')
plt.show()

# Print the area of each object
for object_id, area in area_dict.items():
    print(f'Area of {object_id}: {area} pixels')
```

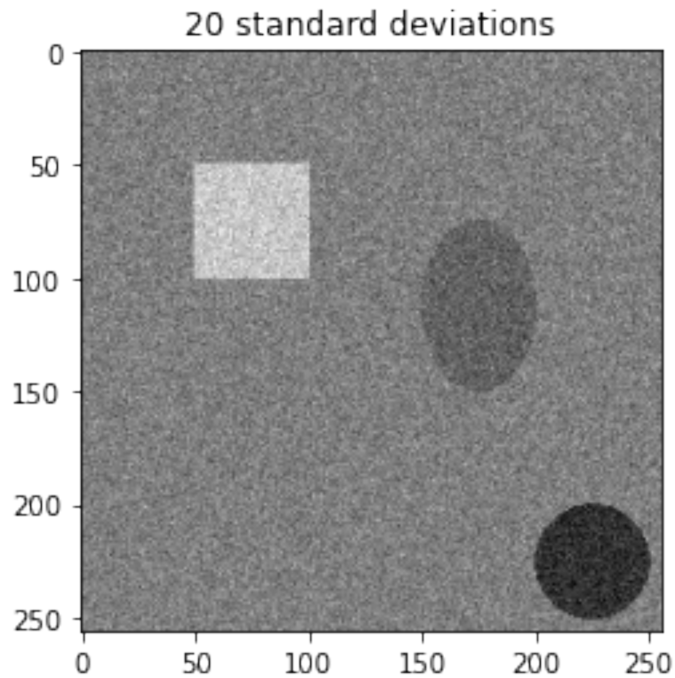


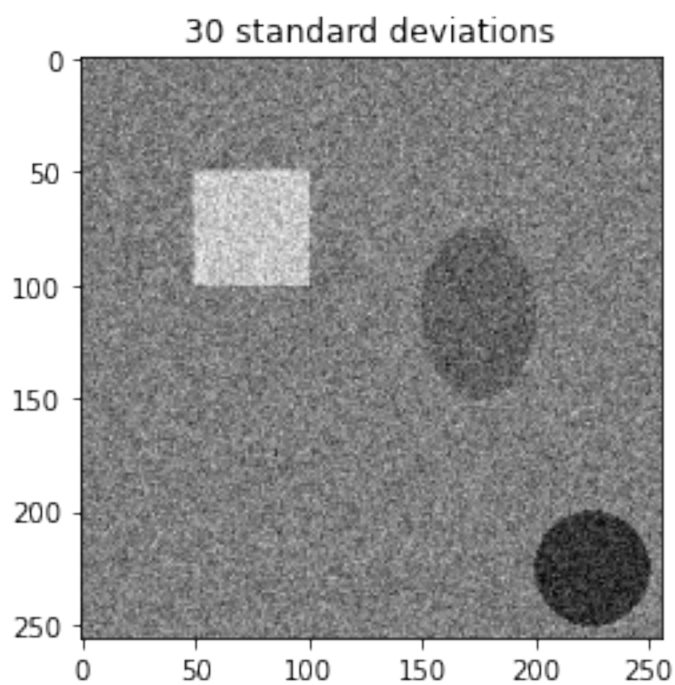

```
Area of object_1: 2500 pixels
Area of object_2: 1963.4954084936207 pixels
Area of object_3: 2945.243112740431 pixels
```

[94]:
```
# Additive Gaussian Noise Only
# Define standard deviations for the Gaussian noise
std_devs = [10, 20, 30]
```

```python
# Convert the image to a numpy array
SEG1_array = np.array(SEG1)

# Create the noisy images
for i, std_dev in enumerate(std_devs):
    # Generate Gaussian noise
    noise = np.random.normal(0, std_dev, SEG1_array.shape)

    # Add the noise to the image
    SEG2 = SEG1_array + noise

    # Ensure the values are within the valid range [0, 255]
    SEG2 = np.clip(SEG2, 0, 255).astype(np.uint8)

    # Convert the array back to an image
    SEG2_image = Image.fromarray(SEG2)

    # Save the image
    SEG2_image.save(f'SEG2_{i+1}.jpg')

    # Display the image
    plt.imshow(SEG2_image, cmap='gray')
    plt.title(f'{std_dev} standard deviations')
    plt.show()
```

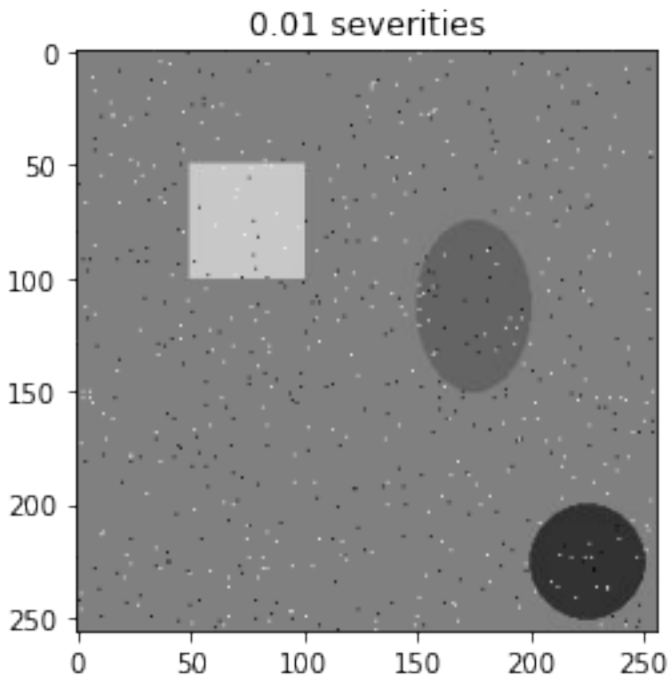


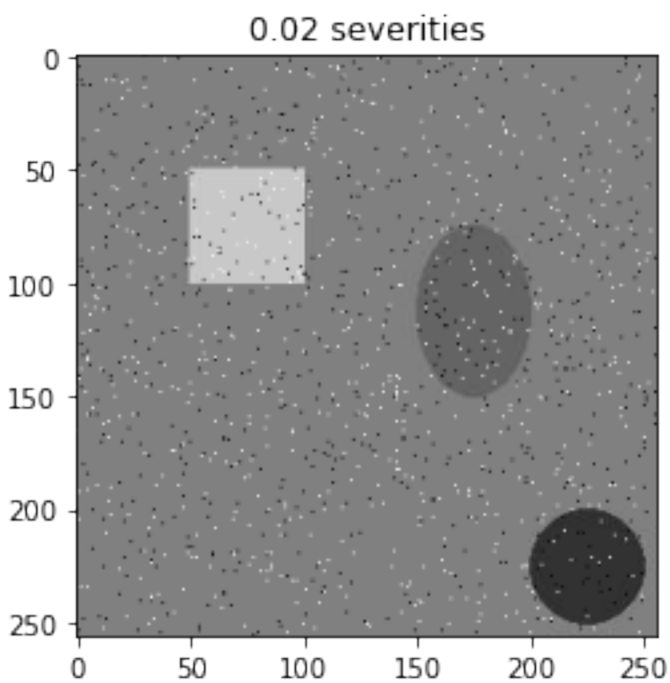

20 standard deviations
0
50
100
150
200
250
0
50
100
150
200
250

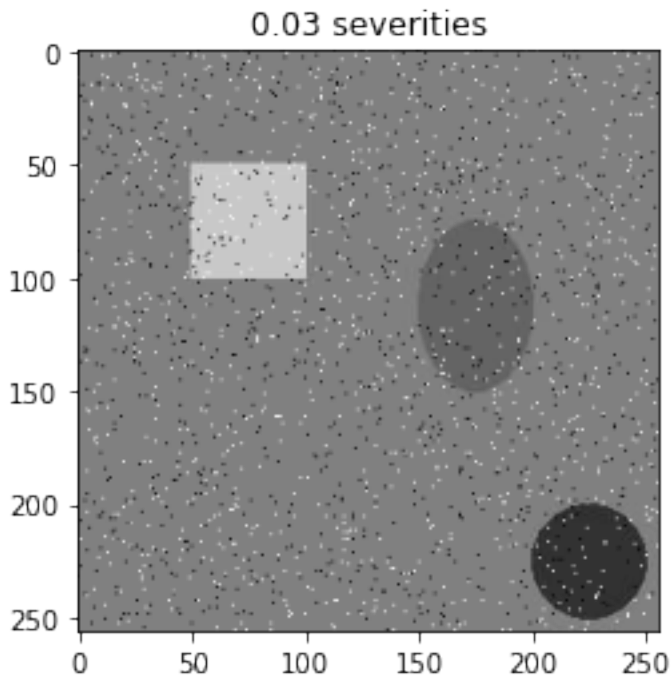

30 standard deviations
0
50
100
150
200
250
0
50
100
150
200
250

[95]:

```python
# Based on SEG1 (Random Impulse Noise Only)
# Define severities for the impulse noise
severities = [0.01, 0.02, 0.03]  # Severity is the fraction of total pixels

# Create the noisy images
for i, severity in enumerate(severities):
    # Open the corresponding SEG2 image
    SEG1_image = Image.open(f'SEG1.jpg')
    SEG1_array = np.array(SEG1_image)

    # Create a copy for manipulation
    SEG3 = SEG1_array.copy()

    # Calculate the number of pixels to alter
    num_pixels = int(severity * SEG3.size)

    # Generate random positions for the impulse noise
    positions = np.random.choice(SEG3.size, num_pixels, replace=False)

    # Add impulse noise (salt-and-pepper noise)
    SEG3.flat[positions[:num_pixels // 2]] = 0  # Minimum value
    SEG3.flat[positions[num_pixels // 2:]] = 255  # Maximum value

    # Convert the array back to an image
    SEG3_image = Image.fromarray(SEG3)

    # Save the image
    SEG3_image.save(f'SEG3_1_{i+1}.jpg')

    # Display the image
    plt.imshow(SEG3_image, cmap='gray')
    plt.title(f'{severity} severities')
    plt.show()
```


0.01 severities
0
50
100
150
200
250
0
50
100
150
200
250

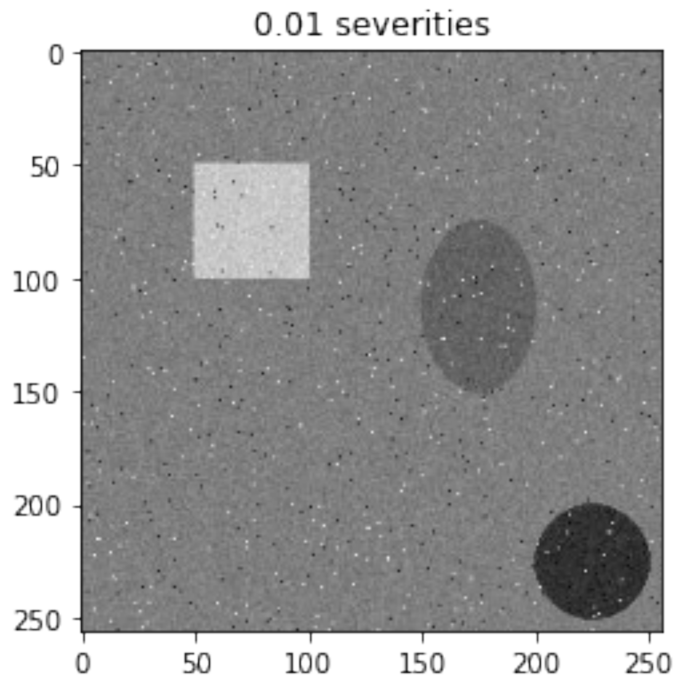

0.02 severities
0
50
100
150
200
250
0
50
100
150
200
250

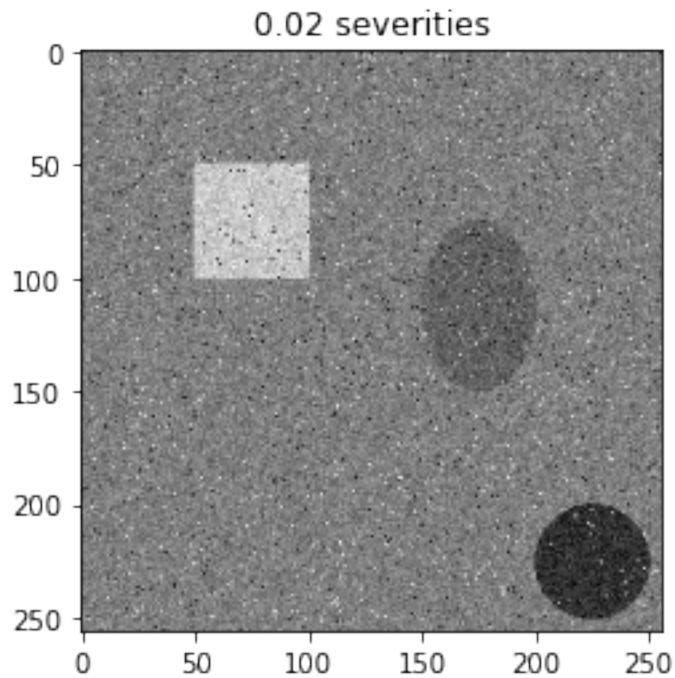


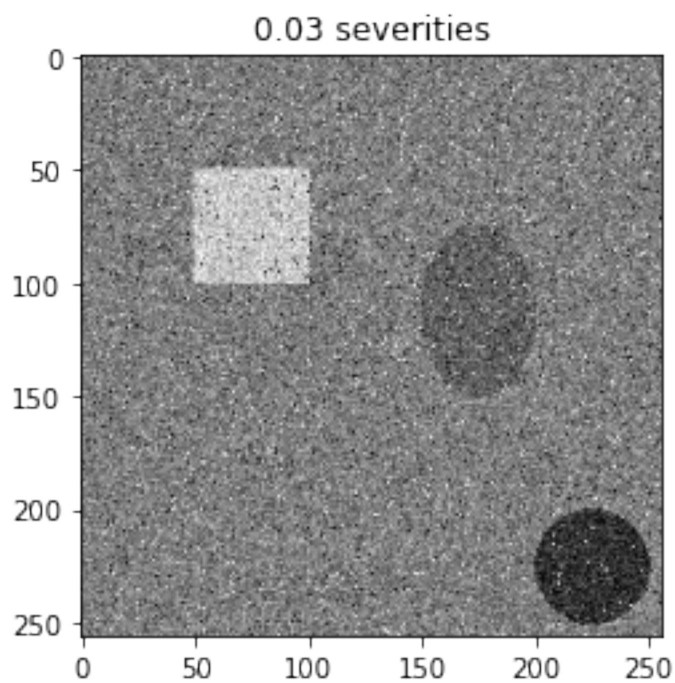
[96]:

```
# Based on SEG2 (Random Impulsive Noise based on Additive Gaussian Noise)
# Define severities for the impulse noise
severities = [0.01, 0.02, 0.03]  # Severity is the fraction of total pixels

# Create the noisy images
for i, severity in enumerate(severities):
    # Open the corresponding SEG2 image
    SEG2_image = Image.open(f'SEG2_{i+1}.jpg')
    SEG2_array = np.array(SEG2_image)

    # Create a copy for manipulation
    SEG3 = SEG2_array.copy()

    # Calculate the number of pixels to alter
    num_pixels = int(severity * SEG3.size)

    # Generate random positions for the impulse noise
    positions = np.random.choice(SEG3.size, num_pixels, replace=False)

    # Add impulse noise (salt-and-pepper noise)
    SEG3.flat[positions[:num_pixels // 2]] = 0  # Minimum value
    SEG3.flat[positions[num_pixels // 2:]] = 255  # Maximum value

    # Convert the array back to an image
```

```
SEG3_image = Image.fromarray(SEG3)

# Save the image
SEG3_image.save(f'SEG3_2_{i+1}.jpg')

# Display the image
plt.imshow(SEG3_image, cmap='gray')
plt.title(f'{severity} severities')
plt.show()
```




0.02 severities
0
50
100
150
200
250
0
50
100
150
200
250


0.03 severities
0
50
100
150
200
250
0
50
100
150
200
250

2 Problem 2

[116]:
```python
def relative_signed_area_error(true_areas, measured_areas):
    sum_true_areas = sum(true_areas)
    sum_measured_areas = sum(measured_areas)

    if sum_true_areas == 0:
        raise ValueError("Sum of true areas cannot be zero")

    # relative signed area error
    relative_error = (sum_true_areas - sum_measured_areas) * 100 /␣
↪sum_true_areas

    return relative_error

def labelling_error(incorrectly_labeled_pixels, true_areas):
    sum_true_areas = sum(true_areas)

    if sum_true_areas == 0:
        raise ValueError("Sum of true areas cannot be zero")

    l_error = incorrectly_labeled_pixels * 100 / sum_true_areas

    return l_error

# test of relative_signed_area_error()
true_areas = [150, 200, 250]     # Areas of objects in the original image
measured_areas = [160, 190, 240]   # Areas of objects after segmentation

relative_error = relative_signed_area_error(true_areas, measured_areas)

print(f'Relative signed area error: {relative_error} %')

# test of labelling_error()
incorrectly_labeled_pixels = 300   # Number of incorrectly labeled pixels
true_areas = [150, 200, 250]     # Areas of objects in the original image

l_error = labelling_error(incorrectly_labeled_pixels, true_areas)

print(f'Labelling error: {l_error} %')
```

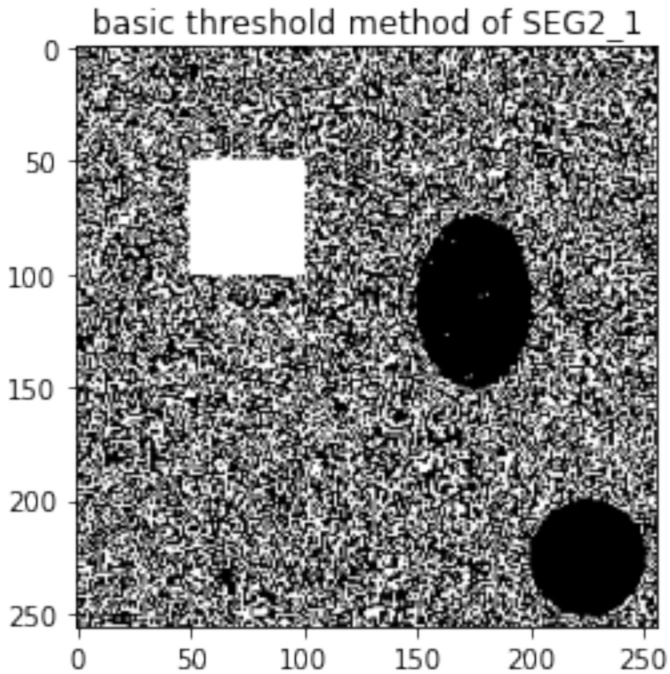
```
Relative signed area error: 1.6666666666666667 %
Labelling error: 50.0 %
```

3 Problem 3

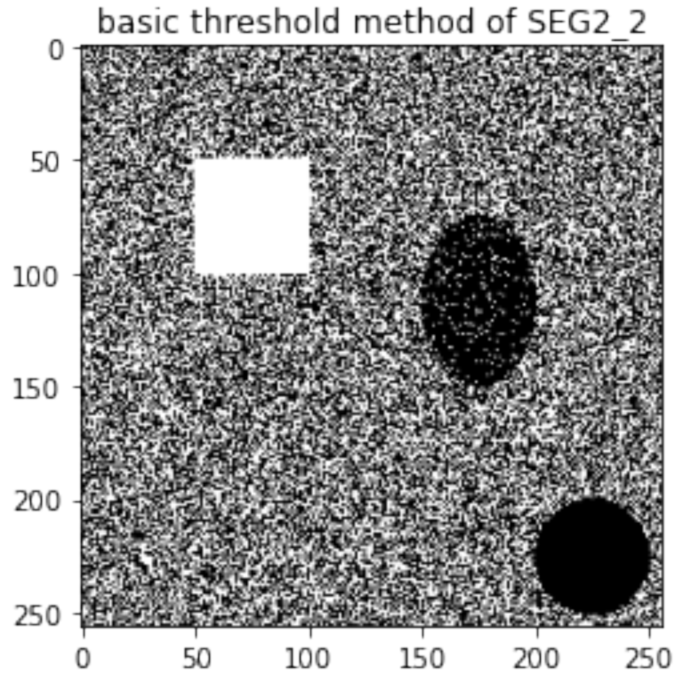
[117]:
```
def compare_segmentation_images(true_image, segmented_image):
    true_array = np.array(true_image)
    segmented_array = np.array(segmented_image)

    true_positive = (true_array > 128) & (segmented_array > 128)
    true_negative = (true_array <= 128) & (segmented_array <= 128)
    false_positive = (true_array <= 128) & (segmented_array > 128)
    false_negative = (true_array > 128) & (segmented_array <= 128)

    correctly_labeled_pixels = np.sum(true_positive) + np.sum(true_negative)
    incorrectly_labeled_pixels = np.sum(false_positive) + np.sum(false_negative)

    true_areas = [np.sum(true_positive)]
    measured_areas = [np.sum(segmented_array > 128)]

    return true_areas, measured_areas, incorrectly_labeled_pixels
```

[118]:
```
def threshold_image(image, threshold=128):
    return image.point(lambda p: p > threshold and 255)

# Apply the threshold and comparason function to the SEG2 images
for i in range(3):
    # Open the corresponding SEG2 image
    SEG2_image = Image.open(f'SEG2_{i+1}.jpg')

    # Apply the threshold
    thresholded_image = threshold_image(SEG2_image)

    # Save the image
    thresholded_image.save(f'SEG2_thresholded_{i+1}.jpg')

    # Display the image
    plt.imshow(thresholded_image, cmap='gray')
    plt.title(f'basic threshold method of SEG2_{i+1}')
    plt.show()

    # Calculate the true_areas, measured_areas, and incorrectly_labeled_pixels
    true_areas, measured_areas, incorrectly_labeled_pixels =␣
↪compare_segmentation_images(SEG1, thresholded_image)

    # Calculate the relative signed area error
    relative_error = relative_signed_area_error(true_areas, measured_areas)
    print(f"Relative signed area error for SEG2_thresholded_{i+1}:␣
↪{relative_error}%")
```

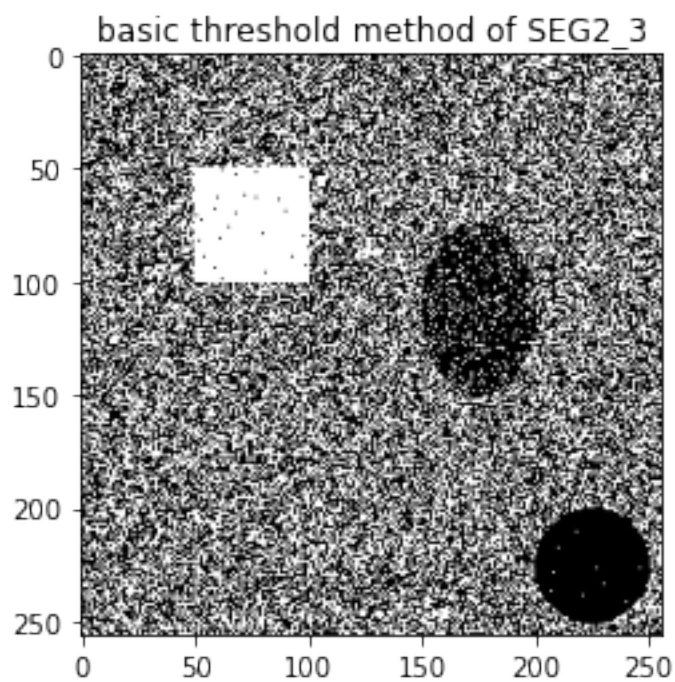
```python
    # Calculate the labelling error
    l_error = labelling_error(incorrectly_labeled_pixels, true_areas)
    print(f"Labelling error for SEG2_thresholded_{i+1}: {l_error}%")
```




```
Relative signed area error for SEG2_thresholded_1: -1018.5697808535178%
Labelling error for SEG2_thresholded_1: 1018.5697808535178%
```



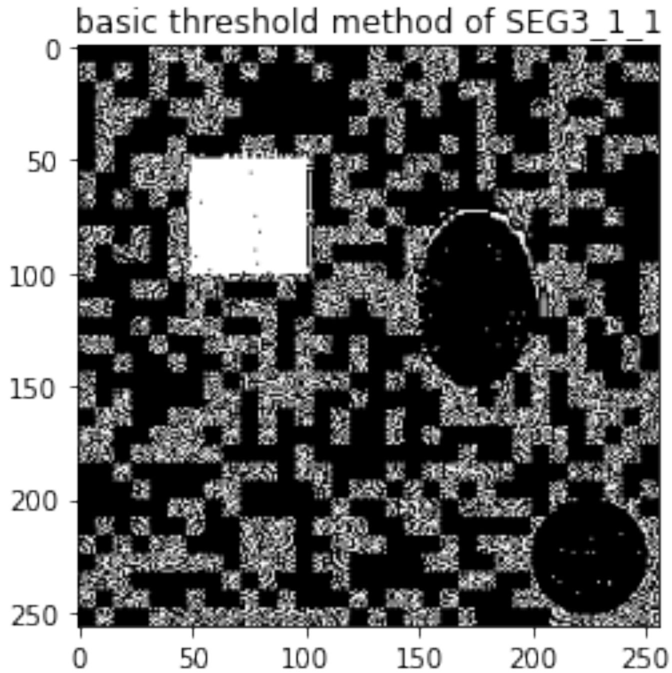

```
Relative signed area error for SEG2_thresholded_2: -1077.8162245290273%
Labelling error for SEG2_thresholded_2: 1077.8162245290273%
```



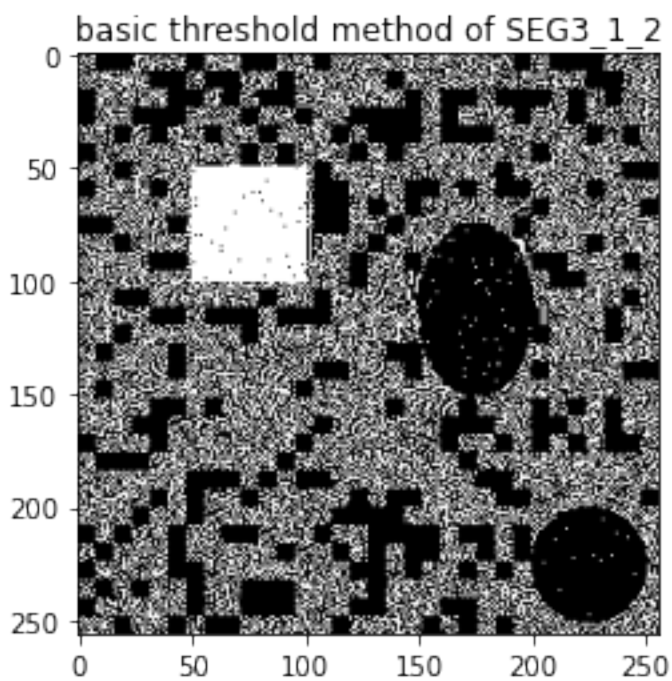
```
Relative signed area error for SEG2_thresholded_3: -1103.416149068323%
Labelling error for SEG2_thresholded_3: 1104.386645962733%
```

[119]:

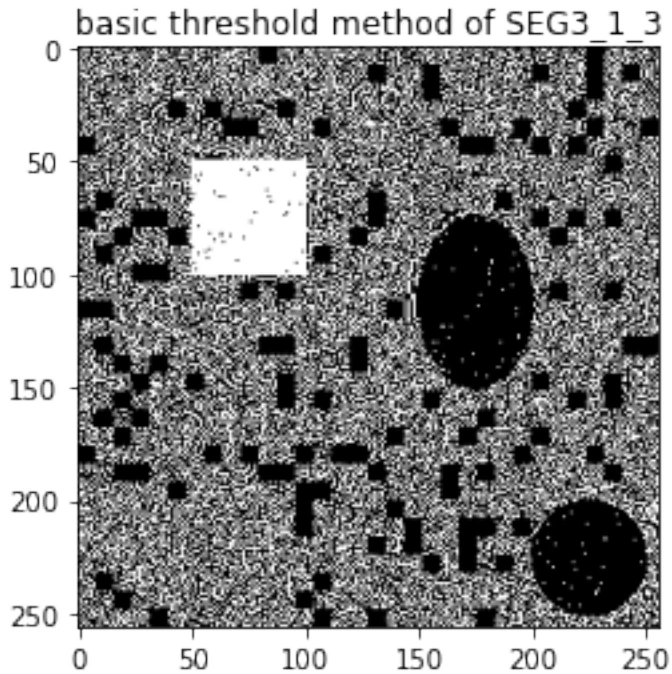
```python
# Apply the threshold to the SEG3_1 images (Random Impulse Noise Only)
for i in range(3):
    # Open the corresponding SEG2 image
    SEG3_image = Image.open(f'SEG3_1_{i+1}.jpg')

    # Apply the threshold
    thresholded_image = threshold_image(SEG3_image)

    # Save the image
    thresholded_image.save(f'SEG3_1_thresholded_{i+1}.jpg')

    # Display the image
    plt.imshow(thresholded_image, cmap='gray')
    plt.title(f'basic threshold method of SEG3_1_{i+1}')
    plt.show()

    # Calculate the true_areas, measured_areas, and incorrectly_labeled_pixels
    true_areas, measured_areas, incorrectly_labeled_pixels =␣
↪compare_segmentation_images(SEG1, thresholded_image)

    # Calculate the relative signed area error
    relative_error = relative_signed_area_error(true_areas, measured_areas)
    print(f"Relative signed area error for SEG3_1_thresholded_{i+1}:␣
↪{relative_error}%")

    # Calculate the labelling error
    l_error = labelling_error(incorrectly_labeled_pixels, true_areas)
    print(f"Labelling error for SEG3_1_thresholded_{i+1}: {l_error}%")
```



Relative signed area error for SEG3_1_thresholded_1: -519.3361636433809%
Labelling error for SEG3_1_thresholded_1: 519.7221150135083%



```
Relative signed area error for SEG3_1_thresholded_2: -779.7118380062306%
Labelling error for SEG3_1_thresholded_2: 780.9968847352025%
```

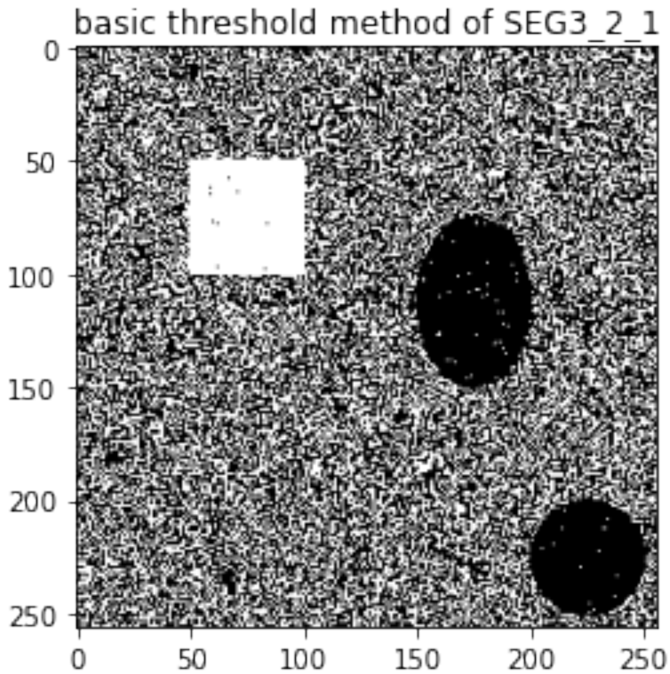



```
Relative signed area error for SEG3_1_thresholded_3: -935.5355746677092%
Labelling error for SEG3_1_thresholded_3: 937.21657544957%
```

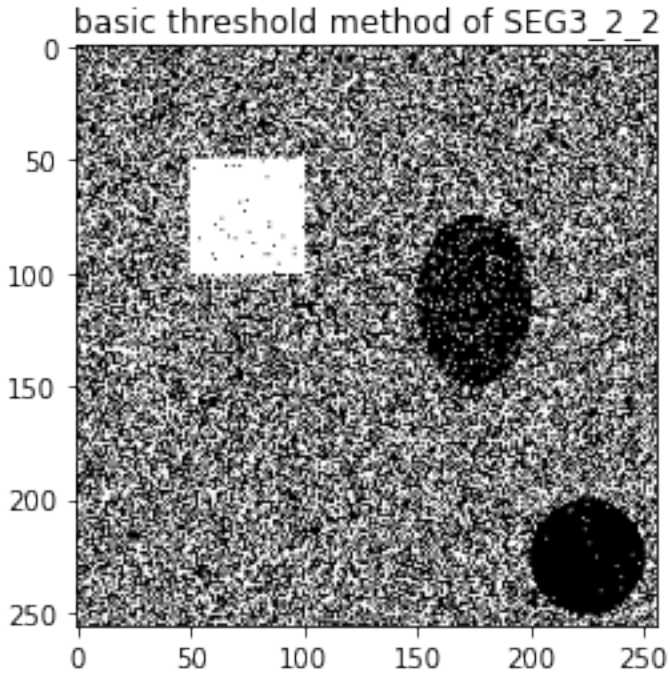
[120]:

```
# Apply the threshold to the SEG3_2 images (Random Impulsive Noise based on␣
↪Additive Gaussian Noise)
for i in range(3):
    # Open the corresponding SEG2 image
    SEG3_image = Image.open(f'SEG3_2_{i+1}.jpg')

    # Apply the threshold
    thresholded_image = threshold_image(SEG3_image)

    # Save the image
    thresholded_image.save(f'SEG3_2_thresholded_{i+1}.jpg')

    # Display the image
    plt.imshow(thresholded_image, cmap='gray')
    plt.title(f'basic threshold method of SEG3_2_{i+1}')
    plt.show()

    # Calculate the true_areas, measured_areas, and incorrectly_labeled_pixels
```

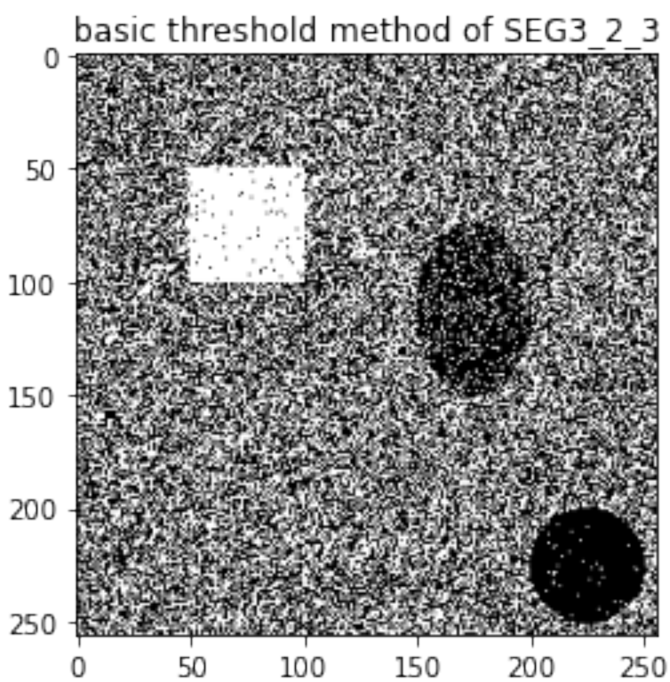
```
    true_areas, measured_areas, incorrectly_labeled_pixels = 
↪compare_segmentation_images(SEG1, thresholded_image)

    # Calculate the relative signed area error
    relative_error = relative_signed_area_error(true_areas, measured_areas)
    print(f"Relative signed area error for SEG3_2_thresholded_{i+1}: 
↪{relative_error}%")

    # Calculate the labelling error
    l_error = labelling_error(incorrectly_labeled_pixels, true_areas)
    print(f"Labelling error for SEG3_2_thresholded_{i+1}: {l_error}%")
```




```
Relative signed area error for SEG3_2_thresholded_1: -1037.0756172839506%
Labelling error for SEG3_2_thresholded_1: 1037.4228395061727%
```

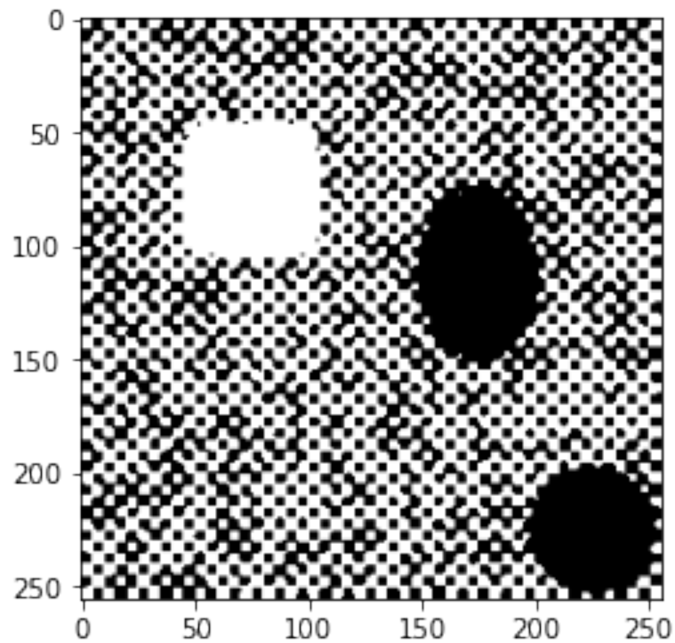


Relative signed area error for SEG3_2_thresholded_2: -1096.6121495327102%
Labelling error for SEG3_2_thresholded_2: 1097.8971962616822%

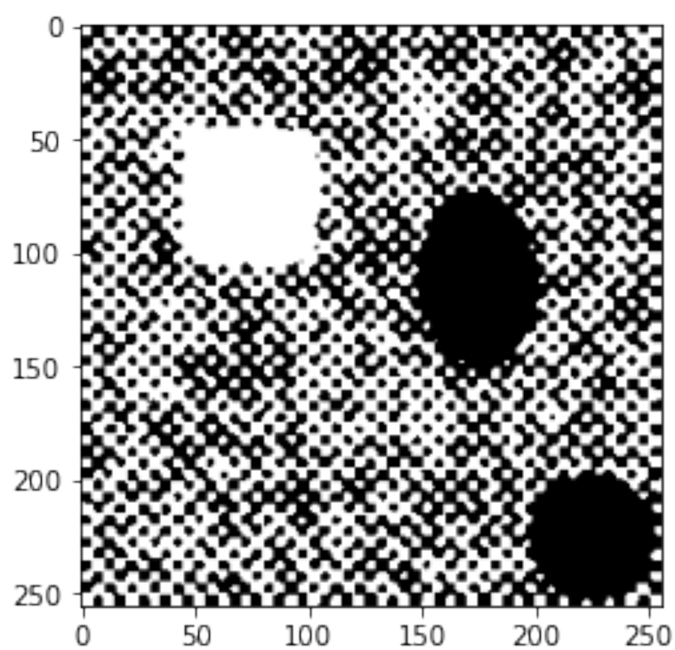


```
Relative signed area error for SEG3_2_thresholded_3: -1133.4256329113923%
Labelling error for SEG3_2_thresholded_3: 1136.3132911392406%
```

[121]:
```python
# Open the original image
SEG1_image = Image.open('SEG1.jpg')
SEG1_array = np.array(SEG1_image)

# Apply the Chan-Vese method and calculate the errors
for i in range(3):
    # Open the corresponding SEG2 image
    SEG2_image = io.imread(f'SEG2_{i+1}.jpg', as_gray=True)

    # Normalize the image to 0-1
    SEG2_image = SEG2_image / 255.0

    # Apply the Chan-Vese segmentation
    cv = segmentation.chan_vese(SEG2_image, mu=0.25, lambda1=1, lambda2=1,␣
↪tol=1e-3, max_iter=200,
                                dt=0.5, init_level_set="checkerboard",␣
↪extended_output=True)

    # The result of the Chan-Vese segmentation is a binary image. Convert it to␣
↪an 8-bit image.
    binary_image = img_as_ubyte(cv[0])

    # Calculate the true and measured areas and the number of incorrectly␣
↪labeled pixels
    true_areas, measured_areas, incorrectly_labeled_pixels =␣
↪compare_segmentation_images(SEG1_image, binary_image)

    # Save the image
    io.imsave(f'SEG2_chanvese_{i+1}.jpg', binary_image)

    # Display the image
    plt.imshow(binary_image, cmap='gray')
    plt.show()

    # Calculate the relative signed area error
    relative_error = relative_signed_area_error(true_areas, measured_areas)
    print(f"Relative signed area error for SEG2_chanvese_{i+1}.jpg:␣
↪{relative_error}%")

    # Calculate the labelling error
    l_error = labelling_error(incorrectly_labeled_pixels, true_areas)
    print(f"Labelling error for SEG2_chanvese_{i+1}.jpg: {l_error}%")
```

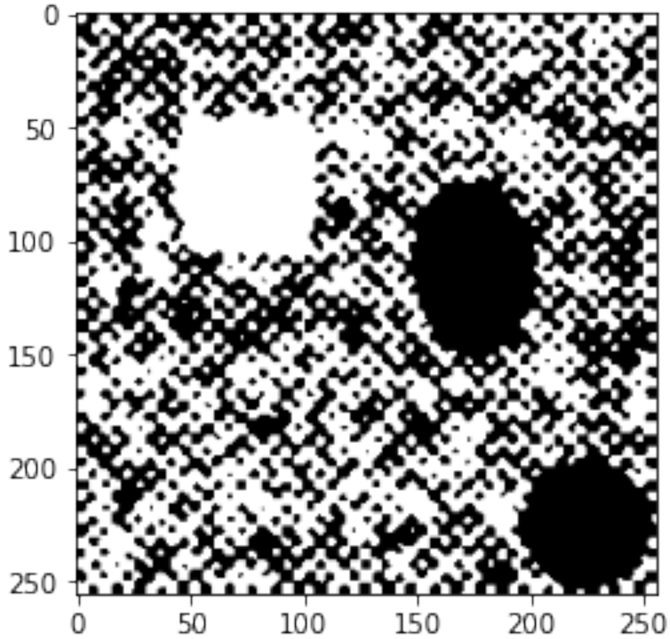


```
Relative signed area error for SEG2_chanvese_1.jpg: -1079.187646861363%
Labelling error for SEG2_chanvese_1.jpg: 1095.7703927492448%
```



```
Relative signed area error for SEG2_chanvese_2.jpg: -1027.693346842283%
```

```
Labelling error for SEG2_chanvese_2.jpg: 1044.984802431611%
```

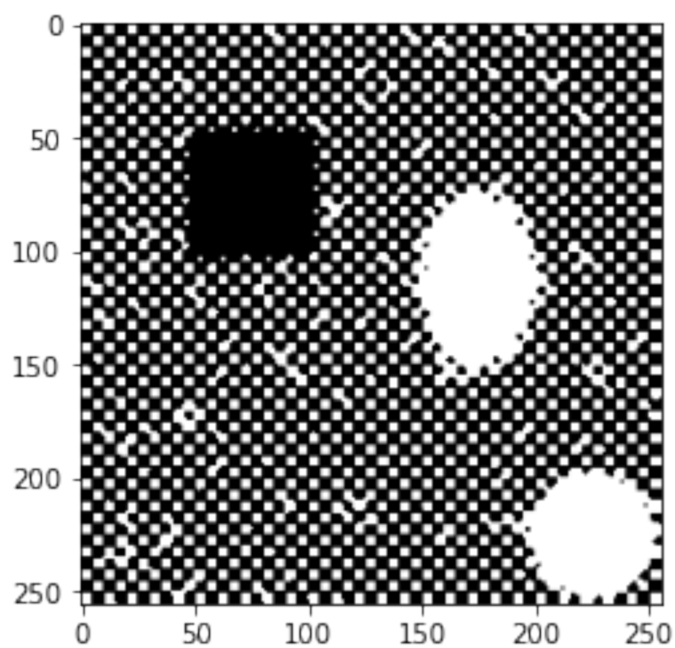


```
Relative signed area error for SEG2_chanvese_3.jpg: -1029.0643662906436%
Labelling error for SEG2_chanvese_3.jpg: 1044.293297942933%
```

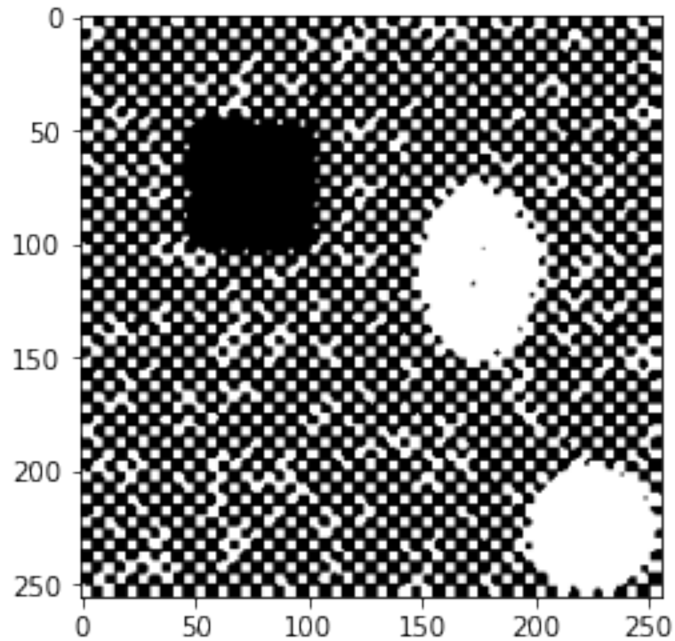
[122]:

```
# Open the original image
SEG1_image = Image.open('SEG1.jpg')
SEG1_array = np.array(SEG1_image)

# Apply the Chan-Vese method and calculate the errors
for i in range(3):
    # Open the corresponding SEG3 image
    SEG3_image = io.imread(f'SEG3_1_{i+1}.jpg', as_gray=True)

    # Normalize the image to 0-1
    SEG3_image = SEG3_image / 255.0

    # Apply the Chan-Vese segmentation
    cv = segmentation.chan_vese(SEG3_image, mu=0.25, lambda1=1, lambda2=1,␣
↪tol=1e-3, max_iter=200,
                                 dt=0.5, init_level_set="checkerboard",␣
↪extended_output=True)

    # The result of the Chan-Vese segmentation is a binary image. Convert it to␣
↪an 8-bit image.
    binary_image = img_as_ubyte(cv[0])
```

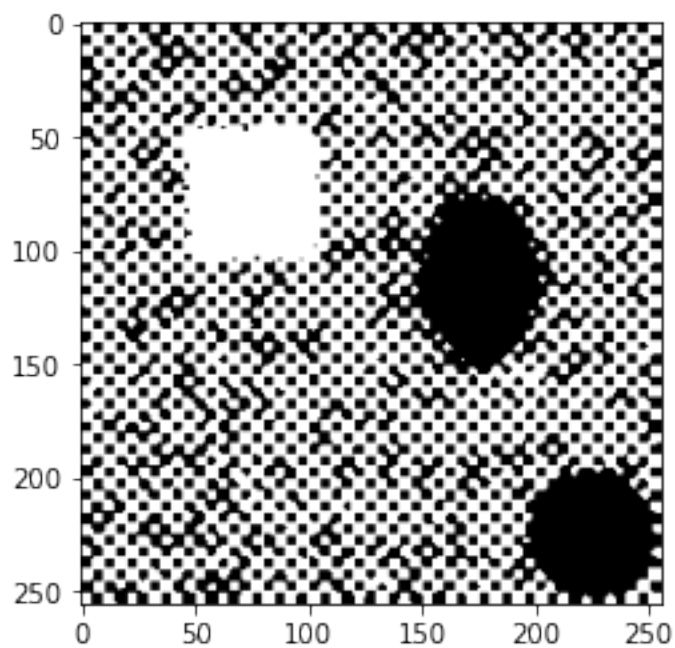
```
    # Calculate the true and measured areas and the number of incorrectly labeled pixels
    true_areas, measured_areas, incorrectly_labeled_pixels = compare_segmentation_images(SEG1_image, binary_image)

    # Save the image
    io.imsave(f'SEG3_1_chanvese_{i+1}.jpg', binary_image)

    # Display the image
    plt.imshow(binary_image, cmap='gray')
    plt.show()

    # Calculate the relative signed area error
    relative_error = relative_signed_area_error(true_areas, measured_areas)
    print(f"Relative signed area error for SEG3_1_chanvese_{i+1}.jpg: {relative_error}%")

    # Calculate the labelling error
    l_error = labelling_error(incorrectly_labeled_pixels, true_areas)
    print(f"Labelling error for SEG3_1_chanvese_{i+1}.jpg: {l_error}%")
```




```
Relative signed area error for SEG3_1_chanvese_1.jpg: -5641.176470588235%
Labelling error for SEG3_1_chanvese_1.jpg: 6326.923076923077%
```



Relative signed area error for SEG3_1_chanvese_2.jpg: -5915.929203539823%
Labelling error for SEG3_1_chanvese_2.jpg: 6584.29203539823%

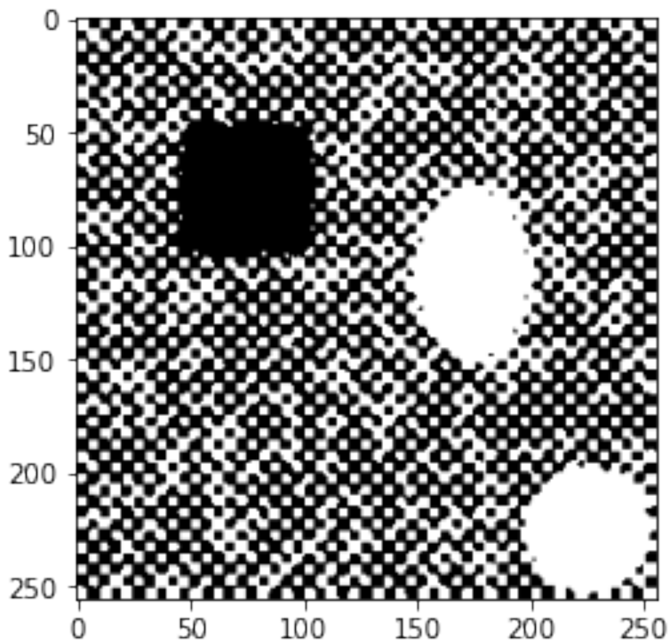


Relative signed area error for SEG3_1_chanvese_3.jpg: -1133.3555038244097%

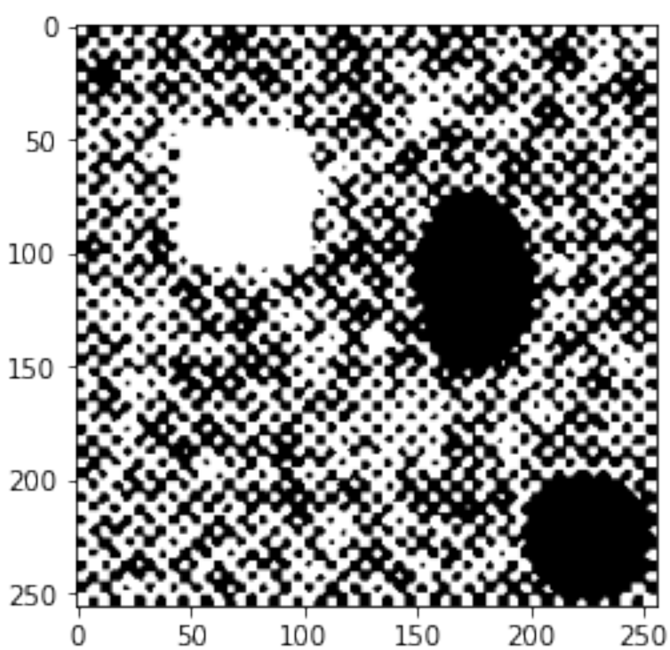
```
Labelling error for SEG3_1_chanvese_3.jpg: 1148.8526770867975%
```

[123]:

```python
# Open the original image
SEG1_image = Image.open('SEG1.jpg')
SEG1_array = np.array(SEG1_image)

# Apply the Chan-Vese method and calculate the errors
for i in range(3):
    # Open the corresponding SEG3 image
    SEG3_image = io.imread(f'SEG3_2_{i+1}.jpg', as_gray=True)

    # Normalize the image to 0-1
    SEG3_image = SEG3_image / 255.0

    # Apply the Chan-Vese segmentation
    cv = segmentation.chan_vese(SEG3_image, mu=0.25, lambda1=1, lambda2=1,␣
↪tol=1e-3, max_iter=200,
                                dt=0.5, init_level_set="checkerboard",␣
↪extended_output=True)

    # The result of the Chan-Vese segmentation is a binary image. Convert it to␣
↪an 8-bit image.
    binary_image = img_as_ubyte(cv[0])

    # Calculate the true and measured areas and the number of incorrectly␣
↪labeled pixels
    true_areas, measured_areas, incorrectly_labeled_pixels =␣
↪compare_segmentation_images(SEG1_image, binary_image)

    # Save the image
    io.imsave(f'SEG3_2_chanvese_{i+1}.jpg', binary_image)

    # Display the image
    plt.imshow(binary_image, cmap='gray')
    plt.show()

    # Calculate the relative signed area error
    relative_error = relative_signed_area_error(true_areas, measured_areas)
    print(f"Relative signed area error for SEG3_2_chanvese_{i+1}.jpg:␣
↪{relative_error}%")

    # Calculate the labelling error
    l_error = labelling_error(incorrectly_labeled_pixels, true_areas)
    print(f"Labelling error for SEG3_2_chanvese_{i+1}.jpg: {l_error}%")
```

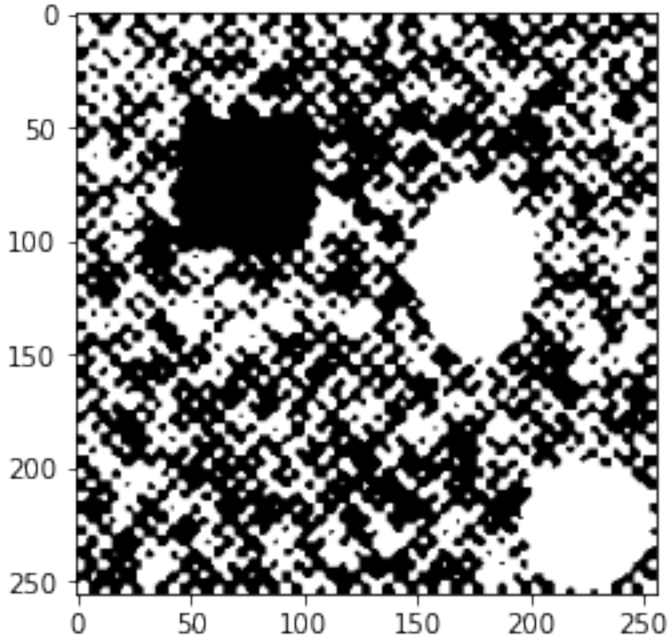


Relative signed area error for SEG3_2_chanvese_1.jpg: -6529.828326180257%
Labelling error for SEG3_2_chanvese_1.jpg: 7175.107296137339%



Relative signed area error for SEG3_2_chanvese_2.jpg: -1029.767127910901%

```
Labelling error for SEG3_2_chanvese_2.jpg: 1046.9794127573405%
```




```
Relative signed area error for SEG3_2_chanvese_3.jpg: -6725.481798715204%
Labelling error for SEG3_2_chanvese_3.jpg: 7369.164882226981%
```

3.0.1 Result

Between Basic thresholding and Chan-vese:

Comparison of SEG2, SEG3_1 (Additive Gaussian Noise Only), and SEG3_2 (Random Impulsive Noise based on Additive Gaussian Noise), sometimes Chan-vese has more accuracy in both Relative signed area error and Labelling error, it's not as stable as Basic threshold, huge error comes several times (color of artificial object blocks sometimes appear to be inverted).

```
[ ]:
```

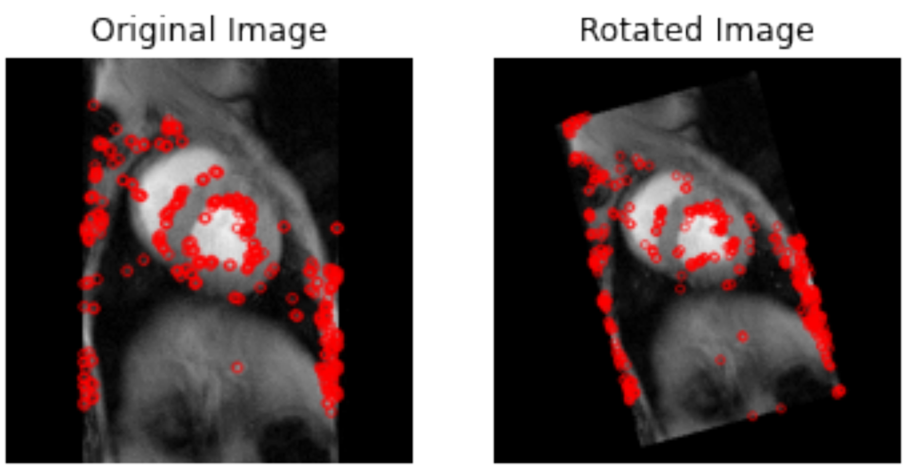
yizjia_577fq4

May 4, 2023

[6]:
```python
import numpy as np
import cv2 as cv
from PIL import Image
from matplotlib import pyplot as plt
```

[7]:
```python
def draw_keypoints(img, kp):
    img_kp = cv.drawKeypoints(img, kp, None, color=(255, 0, 0), flags=0)
    return img_kp

# Load the image
img = Image.open('heart.jpg')

# Rotate the image
img_rotated = img.rotate(15, expand=True)

# Save the rotated image
img_rotated.save('heart.15.jpg')

# Convert PIL images to OpenCV images (numpy arrays)
img = np.array(img)
img_rotated = np.array(img_rotated)

# Convert to grayscale for feature detection
img_gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
img_rotated_gray = cv.cvtColor(img_rotated, cv.COLOR_BGR2GRAY)

# Initiate ORB detector
orb = cv.ORB_create()

# Find the keypoints with ORB
key_pt1 = orb.detect(img_gray, None)
key_pt2 = orb.detect(img_rotated_gray, None)

# Compute the descriptors with ORB
key_pt1, des1 = orb.compute(img_gray, key_pt1)
key_pt2, des2 = orb.compute(img_rotated_gray, key_pt2)

# Draw keypoints
```

```python
img_key_pt1 = draw_keypoints(img_gray, key_pt1)
img_key_pt2 = draw_keypoints(img_rotated_gray, key_pt2)

# Display original image and rotated image with keypoints
plt.figure()
plt.subplot(121)
plt.imshow(img_key_pt1, cmap='gray')
plt.title('Original Image')
plt.axis('off')

plt.subplot(122)
plt.imshow(img_key_pt2, cmap='gray')
plt.title('Rotated Image')
plt.axis('off')
plt.show()
```



0.1 Main steps and methods:

Grayscale Conversion: The images are converted to grayscale using cv.cvtColor, as the ORB detector operates on grayscale images.

ORB Feature Detection: An ORB (Oriented FAST and Rotated BRIEF) detector is created with cv.ORB_create(). This detector is used to find significant points (keypoints) in the images.

Keypoint Detection: The ORB detector's detect method is used to find keypoints in the grayscale images, providing a set of points that can be reliably recognized across different images of the same scene.

Descriptor Computation: The ORB detector's compute method calculates "descriptors" for each keypoint. Descriptors are compact vectors that describe the keypoints in a way that is robust to changes in image rotation and illumination, allowing keypoints to be compared between images.

Keypoint Visualization: The keypoints are visualized on the grayscale images using OpenCV's drawKeypoints function. This function draws each keypoint as a red dot.

Reference: ORB (Oriented FAST and Rotated BRIEF), https://docs.opencv.org/3.4/d1/d89/tutorial_py_orb.htm

[ ]:

```

```

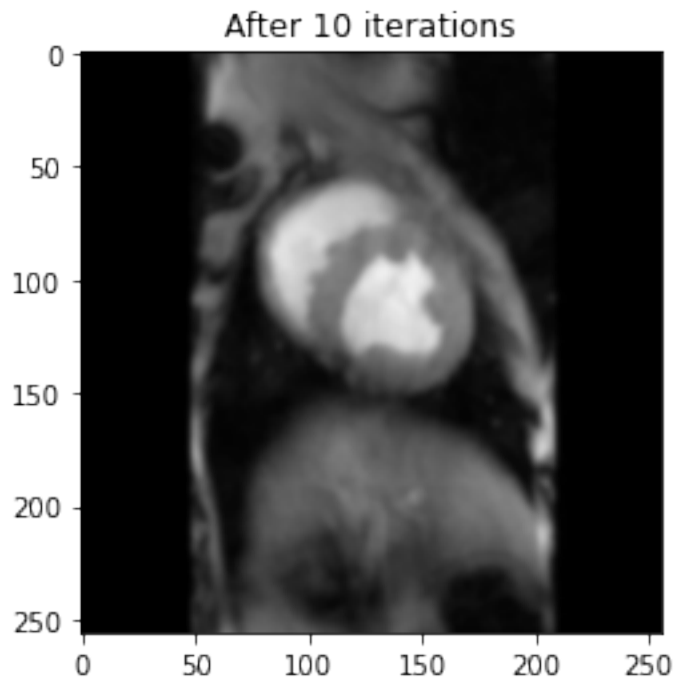
yizjia_577fq5

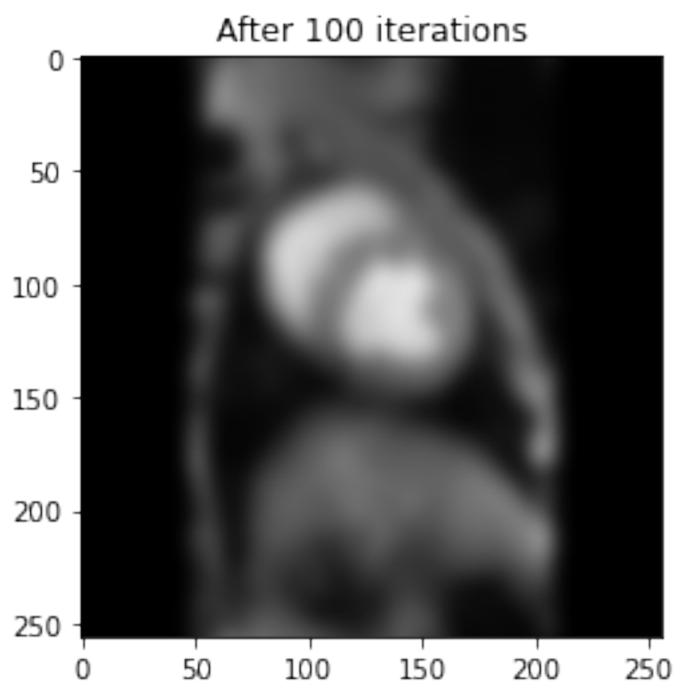
May 4, 2023

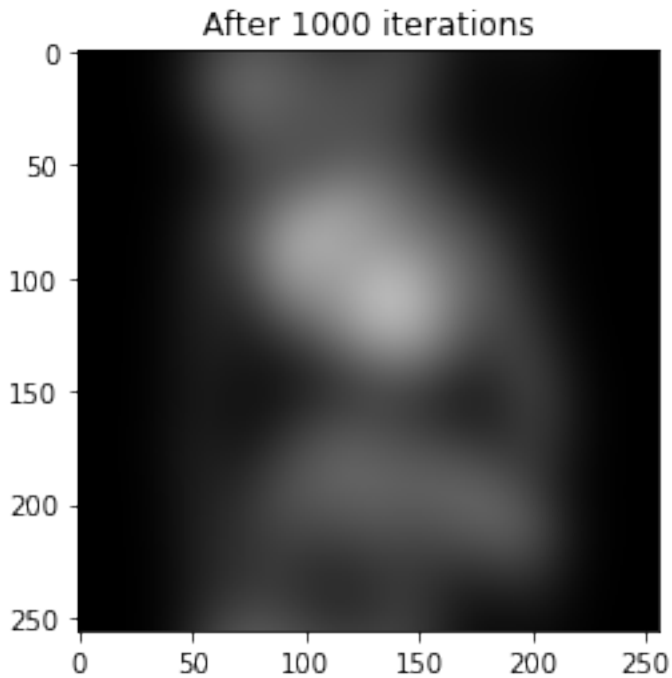
[17]:
```python
import numpy as np
import imageio
import matplotlib.pyplot as plt
```

[18]:
```python
def smooth_image(image, dt, n):
    height, width, _ = image.shape

    # Create a copy of the image to hold the updates
    updated_image = image.copy()

    for _ in range(n):
        # Calculate second derivatives
        d2x = (np.roll(image, -1, axis=1) - 2 * image + np.roll(image, 1,␣
↪axis=1))
        d2y = (np.roll(image, -1, axis=0) - 2 * image + np.roll(image, 1,␣
↪axis=0))

        # Update image using heat equation
        updated_image = image + dt * (d2x + d2y)

        # Replace original image with updated image
        image = updated_image.copy()

    return updated_image

# Load image
image = imageio.imread('heart.jpg').astype(float)

# Set parameters
dt = 0.1

# Smooth image and save outputs
for n in [10, 100, 1000]:
    smoothed = smooth_image(image, dt, n)
    output_file = f'smoothed_{n}.jpg'
    imageio.imsave(output_file, smoothed.astype(np.uint8))
```

```
# Display the image
plt.imshow(smoothed.astype(np.uint8))
plt.title(f'After {n} iterations')
plt.show()
```






After 1000 iterations
0
50
100
150
200
250
0
50
100
150
200
250

[ ]:

Q5 (b) two-dimensional heat equation

$$\frac{\partial I}{\partial t} = \frac{\partial^2 I}{\partial x} + \frac{\partial^2 I}{\partial y}$$

$$I(x, y, 0) = I_0(x, y)$$

If use centered differences for the spatial derivatives, with $\Delta x = \Delta y = 1$, $\Delta t = 0.1$

Altered heat equation:

$$I(x, y, t+\Delta t) = I(x, y, t) + \frac{\Delta t}{\Delta x^2}[I(x+1, y, t) - 2I(x, y, t) + I(x-1, y, t)]$$
$$+ \frac{\Delta t}{\Delta y^2}[I(x, y+1, t) - 2I(x, y, t) + I(x, y-1, t)]$$

$$\Rightarrow \quad I(x, y, t+0.1) = I(x, y, t) + \frac{1}{10}[I(x+1, y, t) - 2I(x, y, t) + I(x-1, y, t]$$
$$+ \frac{1}{10}[I(x, y+1, t) - 2I(x, y, t) + I(x, y-1, t)]$$