Workshop #2: **Image Segmentation**

Learning Outcomes:

Upon successful completion of this workshop, you will have demonstrated the abilities to:

- Understand the knowledge of image segmentation and their algorithms.
- Implement segmentation algorithms: Snakes algorithm, K Means algorithm

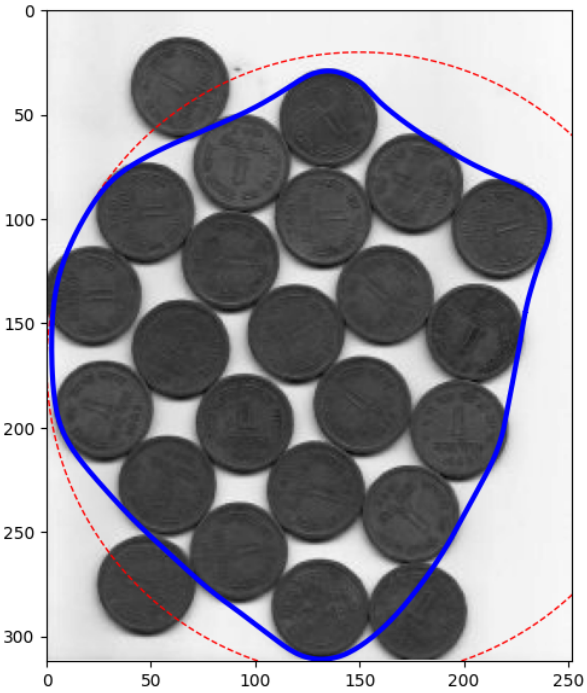
Requirements:

Segmentation is the process of dividing an image into different regions based on the characteristics of pixels to identify objects or boundaries to simplify an image and more efficiently analyze it. The goal of segmentation is to simplify and change the representation of an image into something that is more meaningful and easier to analyze. Image segmentation is the process of assigning a label to every pixel in an image such that pixels with the same label share certain characteristics. In this assignment, students are asked to write a program that implements algorithms for image segmentation. Details of the functions are described below:

Function 1: Snakes algorithm we try to move snake in a direction where energy is minimum. Snake model is designed to vary its shape and position while tending to search through the minimal energy state. Snake propagates through the domain of the image to reduce the energy function, and intends to dynamically move to the local minimum. You are required to implement a Snakes algorithm for active contours.

In [16]:

```
img = cv2.imread('coins.jpg')
img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

s = np.linspace(0, 2*np.pi, 400)
r = 170 + 150*np.sin(s)
c = 150 + 150*np.cos(s)
init = np.array([r, c]).T

# START YOUR CODE

# implement gaussian filter with kernel 3x3 using gaussian() function from skimage.filters
gaussian_img = ?????

# implement active countour by using the active_contour() function from skimage.segmentation, alpha=0.015,
                      #beta=10, gamma=0.001
snake = ???

# END YOUR CODE

fig, ax = plt.subplots(figsize=(7, 7))
ax.imshow(img, cmap=plt.cm.gray)
ax.plot(init[:, 1], init[:, 0], '--r', lw=1)
ax.plot(snake[:, 1], snake[:, 0], '-b', lw=3)
ax.axis([0, img.shape[1], img.shape[0], 0])

plt.show()
```



Function 2: K Means is a clustering algorithm. It is used to identify different classes or clusters in the given data based on how similar the data is. Data points in the same group are more similar to other data points in that same group than those in other groups. The main idea of performing the following process is to find those areas of pixels that share the same color hue parameter value. You are required to implement the K-means for Segmentation.

```
def initialize_clusters(data, k):
    """
    Initialize clusters randomly from data points.
    """
    n_samples, _ = data.shape
    indices = # your code here by using np.random.choice()
    # print(indices)
    return data[indices]
```

```python
def update_clusters(data, assignments, k):
    """
    Update cluster centroids based on the mean of assigned data points.
    """
    new_clusters = np.zeros((k, data.shape[1]))
    for cluster_idx in range(k):
        cluster_data = data[assignments == cluster_idx]
        #Start Your Code

        #End Your Code
    return new_clusters
```

```python
def kmeans(data, k, max_iters=100):
    """
    Perform K-means clustering on the given data.
    """
    # Initialize clusters
    clusters = initialize_clusters(data, k)

    # Iterate until convergence or max iterations
    for _ in range(max_iters):
        # Assign data points to the closest cluster centroid
        assignments = ?? # your code here

        # Update cluster centroids
        new_clusters = ?? # your code here

        # Check for convergence
        if np.allclose(clusters, new_clusters):
            print("Converged.")
            break

        clusters = new_clusters

    return assignments, clusters
```

```python
# Convert image to RGB (OpenCV reads image in BGR format)
image_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# Reshape image data to 2D array
w, h, d = original_shape = tuple(image_rgb.shape)
image_2d = np.reshape(image_rgb, (w * h, d))

n_clusters = 3

# Maximum number of iterations
max_iters = 100

# Perform K-means clustering
assignments, clusters = kmeans(image_2d, n_clusters, max_iters)

# Assign each pixel to its corresponding cluster centroid
clustered_image = np.reshape(assignments, (w, h))
```

```
1 plt.imshow(clustered_image)
```

```
<matplotlib.image.AxesImage at 0x794566f15c00>
```