21MIA1025 - Abhineswari M - CSE4076

TASK 03 - perform edge segmentation on the image provided and segment out the green apple and orange using preprocessing techniques and edge detection algorithm.

In [59]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Read the image
image = cv2.imread('images.jpeg')
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
```

Step 1: Feature Extraction from Segmented Objects

Segment the Image

Segmentation is the process of dividing an image into parts or regions, usually by separating the objects of interest from the background.

How to Do It: Use techniques like thresholding, edge detection, or more advanced methods like watershed or contour-based segmentation.

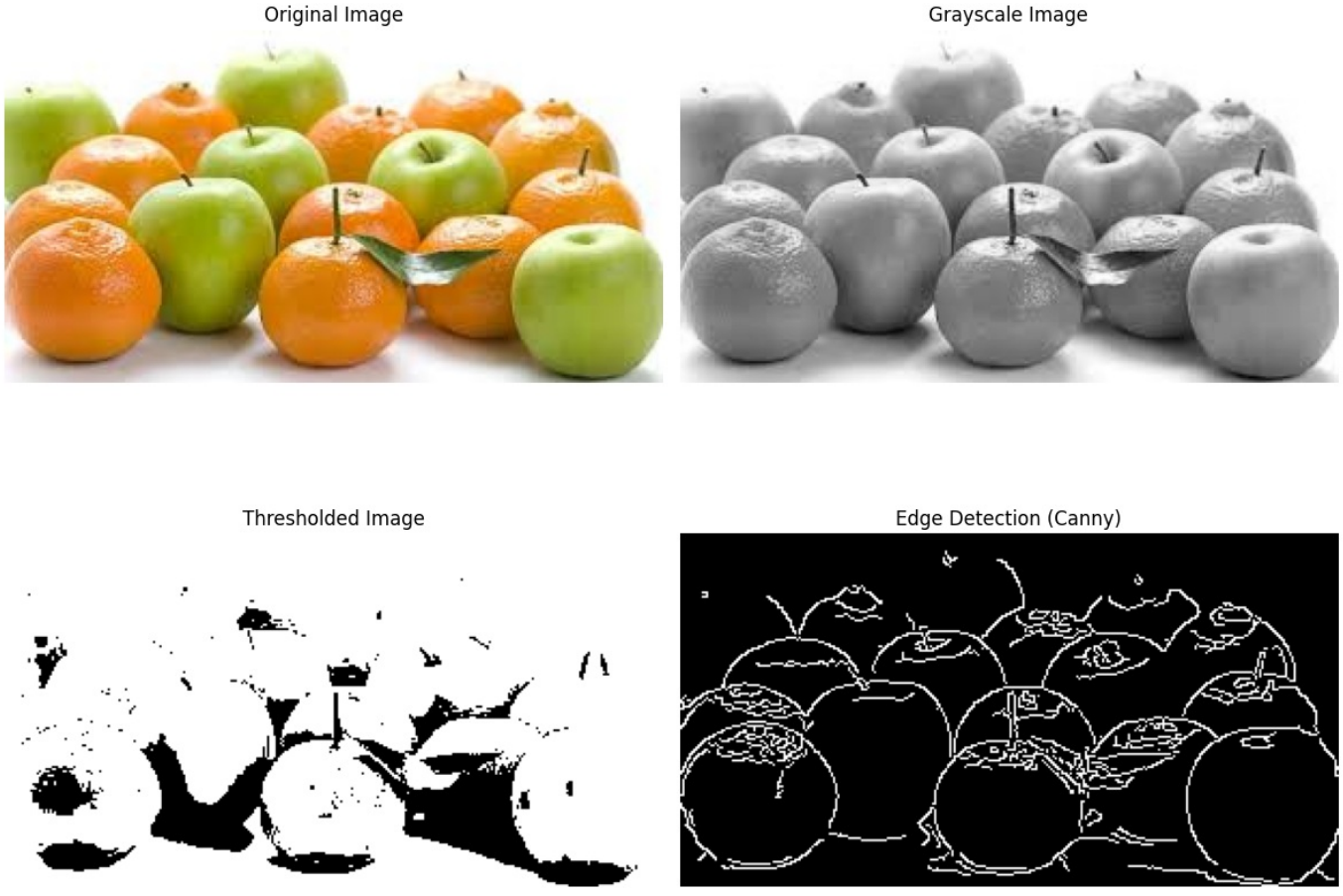
Example: Thresholding: Converts the image to binary (black and white) based on a certain intensity value.

In [60]:
```
# Convert the image to grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Apply thresholding (using Otsu's method to automatically determine the threshold)
_, thresh_image = cv2.threshold(gray_image, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)

# Display the original and segmented images
plt.figure(figsize=(10, 10))

# Original image
plt.subplot(1, 2, 1)
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.title('Original Image')
plt.axis('off')

# Segmented image
plt.subplot(1, 2, 2)
plt.imshow(thresh_image, cmap='gray')
plt.title('Segmented Image with threshold')
plt.axis('off')

plt.show()
```

Edge Detection using Canny

In [61]:
```
# Convert to grayscale
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Apply thresholding
_, binary = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY)

# Apply Canny edge detection
```

```
edges = cv2.Canny(gray, 100, 200)

# Set up the matplotlib figure and axes
fig, axes = plt.subplots(2, 2, figsize=(12, 10))
# Original Image
axes[0, 0].imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
axes[0, 0].set_title('Original Image')
axes[0, 0].axis('off')
# Grayscale Image
axes[0, 1].imshow(gray, cmap='gray')
3
axes[0, 1].set_title('Grayscale Image')
axes[0, 1].axis('off')
# Thresholded Image
axes[1, 0].imshow(binary, cmap='gray')
axes[1, 0].set_title('Thresholded Image')
axes[1, 0].axis('off')
# Edge Detected Image
axes[1, 1].imshow(edges, cmap='gray')
axes[1, 1].set_title('Edge Detection (Canny)')
axes[1, 1].axis('off')
# Adjust layout and show the plot
plt.tight_layout()
plt.show()
```



2.Identify the Objects of Interest

After segmentation, identify the specific objects in the image. Each object will be a separate region or set of pixels.

How to Do It: You can use contours or connected component analysis to label and isolate each object.

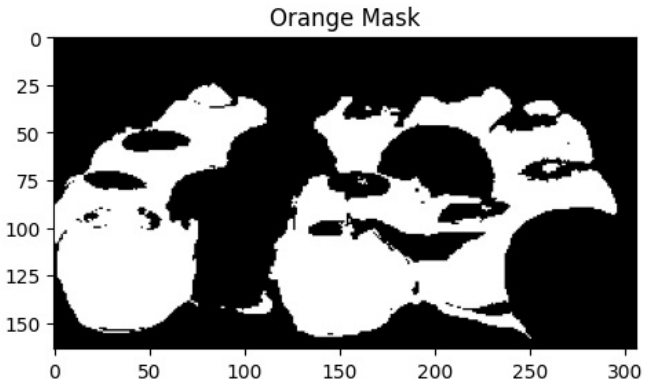
Contours: Find the outlines of objects in the image.

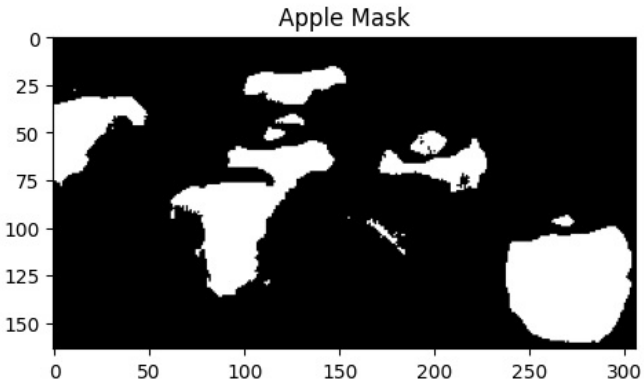
In [62]:
```
# Convert to RGB (since OpenCV uses BGR by default)
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
# Convert the image to HSV color space
hsv_image = cv2.cvtColor(image_rgb, cv2.COLOR_RGB2HSV)

# Define color range for oranges (in HSV)
orange_lower = np.array([10, 100, 100])
orange_upper = np.array([25, 255, 255])

# Define color range for green apples (in HSV)
apple_lower = np.array([30, 100, 100])
apple_upper = np.array([85, 255, 255])

# Create masks for oranges and apples
```

```
mask_orange = cv2.inRange(hsv_image, orange_lower, orange_upper)
mask_apple = cv2.inRange(hsv_image, apple_lower, apple_upper)
```

Extracting Apple and Orange Masks

In [63]:
```
# Display the masks
plt.figure(figsize=(12, 12))
plt.subplot(1, 2, 1)
plt.imshow(mask_orange, cmap='gray')
plt.title("Orange Mask")
plt.subplot(1, 2, 2)
plt.imshow(mask_apple, cmap='gray')
plt.title("Apple Mask")
plt.show()
```



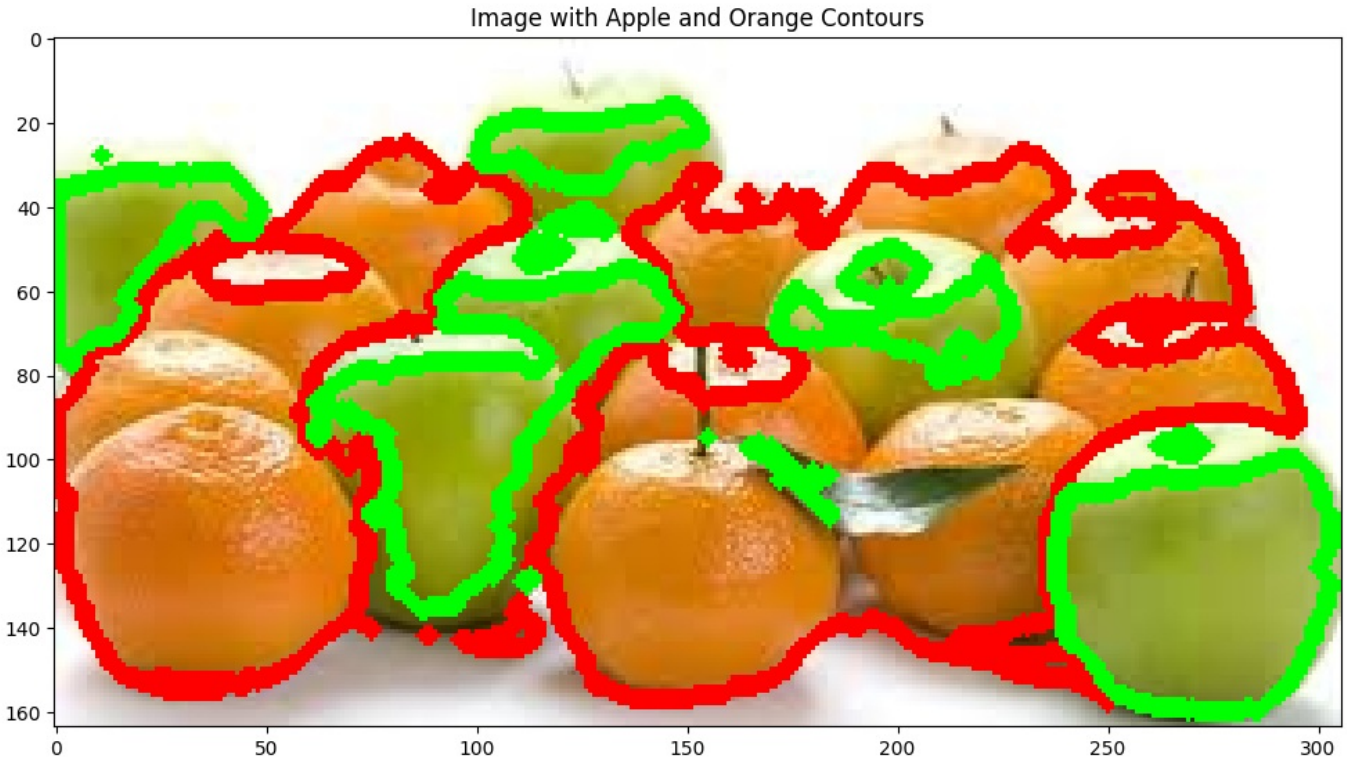


Finding the Contours for apples and oranges

In [64]:
```
contours_orange, _ = cv2.findContours(mask_orange, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

contours_apple, _ = cv2.findContours(mask_apple, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

# Draw contours on the original image
image_with_contours = image_rgb.copy()

# Draw orange contours
cv2.drawContours(image_with_contours, contours_orange, -1, (255, 0, 0), 3) # Blue color for oranges

# Draw apple contours
cv2.drawContours(image_with_contours, contours_apple, -1, (0, 255, 0), 3) # Green color for apples
```

Out[64]:
```
array([[[255, 255, 255],
        [255, 255, 255],
        [255, 255, 255],
        ...,
        [255, 255, 255],
        [255, 255, 255],
        [255, 255, 255]],

       [[255, 255, 255],
        [255, 255, 255],
        [255, 255, 255],
        ...,
        [255, 255, 255],
        [255, 255, 255],
        [255, 255, 255]],

       [[255, 255, 255],
        [255, 255, 255],
        [255, 255, 255],
        ...,
        [255, 255, 255],
        [255, 255, 255],
        [255, 255, 255]],

       ...,

       [[250, 248, 253],
        [249, 247, 252],
        [247, 245, 248],
        ...,
        [229, 234, 230],
        [237, 241, 242],
        [237, 241, 242]],

       [[251, 249, 254],
        [250, 248, 253],
        [248, 246, 249],
        ...,
        [225, 230, 226],
        [233, 237, 238],
        [234, 238, 239]],

       [[252, 250, 255],
        [251, 249, 254],
        [249, 247, 250],
        ...,
        [220, 225, 221],
        [230, 234, 235],
        [230, 234, 235]]], dtype=uint8)
```

In [65]:
```
# Display the image with contours
plt.figure(figsize=(12, 12))
plt.imshow(image_with_contours)
plt.title("Image with Apple and Orange Contours")
plt.show()
```



3. Extract Features from Each Object

Once the objects are isolated, you can extract features from each segmented object.

Feature Types:

Shape Features: Describe the geometry of the object.

Area: The number of pixels inside the object.

Perimeter: The length of the boundary of the object.

Bounding Box: The smallest rectangle that can enclose the object.

Centroid: The center of the object.

Edge Features: Extract edges within the segmented object.

How to Do It: Apply edge detection (e.g., Canny) only to the pixels within the object's region.

Texture Features: Analyze the surface quality of the object.

How to Do It: Compute texture descriptors like Local Binary Pattern (LBP) or Gray-Level Co-occurrence Matrix (GLCM) for the object's pixels.

Color Features: If working with color images, analyze the color distribution within the object.

How to Do It: Compute color histograms or color moments for the segmented object.

In [66]:
```python
# Function to calculate features for each contour
def calculate_features(contours, object_name):
    features = []
    for contour in contours:
        area = cv2.contourArea(contour)
        perimeter = cv2.arcLength(contour, True)
        x, y, w, h = cv2.boundingRect(contour)
        features.append({
            'Object': object_name,
            'Area': area,
            'Perimeter': perimeter,
            'Bounding Box': (x, y, w, h)
        })
    return features

# Calculate features for oranges
orange_features = calculate_features(contours_orange, "Orange")

# Calculate features for apples
apple_features = calculate_features(contours_apple, "Apple")
```

Printing the extracted features

In [67]:
```
for feature in orange_features + apple_features:
    print(feature)
```

```
{'Object': 'Orange', 'Area': 0.0, 'Perimeter': 2.0, 'Bounding Box': (88, 142, 2, 1)}
{'Object': 'Orange', 'Area': 65.0, 'Perimeter': 54.62741661071777, 'Bounding Box': (96, 134, 19, 12)}
{'Object': 'Orange', 'Area': 1.5, 'Perimeter': 11.071067690849304, 'Bounding Box': (160, 75, 5, 3)}
{'Object': 'Orange', 'Area': 0.0, 'Perimeter': 0.0, 'Bounding Box': (160, 74, 1, 1)}
{'Object': 'Orange', 'Area': 0.0, 'Perimeter': 2.0, 'Bounding Box': (267, 67, 2, 1)}
{'Object': 'Orange', 'Area': 20.5, 'Perimeter': 21.071067690849304, 'Bounding Box': (257, 67, 9, 5)}
{'Object': 'Orange', 'Area': 1.5, 'Perimeter': 5.414213538169861, 'Bounding Box': (164, 37, 3, 2)}
{'Object': 'Orange', 'Area': 95.0, 'Perimeter': 44.627416372299194, 'Bounding Box': (247, 34, 19, 8)}
{'Object': 'Orange', 'Area': 13543.5, 'Perimeter': 977.4772635698318, 'Bounding Box': (113, 27, 183, 132)}
{'Object': 'Orange', 'Area': 7078.0, 'Perimeter': 615.6711342334747, 'Bounding Box': (0, 25, 112, 131)}
{'Object': 'Apple', 'Area': 1.5, 'Perimeter': 6.2426406145095825, 'Bounding Box': (111, 128, 3, 3)}
{'Object': 'Apple', 'Area': 3.0, 'Perimeter': 9.656854033470154, 'Bounding Box': (181, 111, 4, 4)}
{'Object': 'Apple', 'Area': 0.0, 'Perimeter': 2.0, 'Bounding Box': (183, 104, 2, 1)}
{'Object': 'Apple', 'Area': 0.0, 'Perimeter': 2.8284270763397217, 'Bounding Box': (179, 103, 2, 2)}
{'Object': 'Apple', 'Area': 3145.0, 'Perimeter': 224.5096663236618, 'Bounding Box': (238, 100, 66, 61)}
{'Object': 'Apple', 'Area': 30.0, 'Perimeter': 52.76955199241638, 'Bounding Box': (165, 96, 16, 14)}
{'Object': 'Apple', 'Area': 0.0, 'Perimeter': 0.0, 'Bounding Box': (155, 95, 1, 1)}
{'Object': 'Apple', 'Area': 32.0, 'Perimeter': 26.14213526248932, 'Bounding Box': (262, 94, 12, 6)}
{'Object': 'Apple', 'Area': 3.5, 'Perimeter': 9.41421353816986, 'Bounding Box': (62, 91, 2, 5)}
{'Object': 'Apple', 'Area': 2559.5, 'Perimeter': 381.8061292171478, 'Bounding Box': (62, 55, 86, 82)}
{'Object': 'Apple', 'Area': 570.5, 'Perimeter': 193.09545266628265, 'Bounding Box': (171, 54, 57, 28)}
{'Object': 'Apple', 'Area': 137.5, 'Perimeter': 51.69848430156708, 'Bounding Box': (188, 50, 19, 13)}
{'Object': 'Apple', 'Area': 38.0, 'Perimeter': 27.313708305358887, 'Bounding Box': (111, 48, 11, 7)}
{'Object': 'Apple', 'Area': 0.0, 'Perimeter': 0.0, 'Bounding Box': (31, 48, 1, 1)}
{'Object': 'Apple', 'Area': 36.5, 'Perimeter': 35.55634891986847, 'Bounding Box': (117, 41, 15, 6)}
{'Object': 'Apple', 'Area': 1091.5, 'Perimeter': 182.9533178806305, 'Bounding Box': (0, 32, 50, 47)}
{'Object': 'Apple', 'Area': 0.0, 'Perimeter': 0.0, 'Bounding Box': (11, 28, 1, 1)}
{'Object': 'Apple', 'Area': 579.0, 'Perimeter': 135.7401144504547, 'Bounding Box': (100, 16, 54, 20)}
```

4.Analyze and Use the Features

Once the features are extracted, you can analyze them for tasks like object recognition, classification, or further image analysis.

For example, you might compare the shape or texture features of different objects to classify them into categories.

In [68]:
```
from PIL import Image
import numpy as np
from sklearn.cluster import KMeans
from collections import Counter

def load_and_process_image(image_path):
    # Open the image
    img = Image.open(image_path)
    # Resize for faster processing
    img = img.resize((100, 100))
    # Convert to numpy array
    return np.array(img)

def extract_dominant_colors(image_array, n_colors=2):
    # Reshape the image to be a list of pixels
    pixels = image_array.reshape(-1, 3)

    # Perform K-means clustering to find dominant colors
    # Set n_init explicitly to suppress the warning
    kmeans = KMeans(n_clusters=n_colors, n_init=10)
    kmeans.fit(pixels)

    # Get the colors
    colors = kmeans.cluster_centers_

    # Convert to integer RGB values
    colors = colors.astype(int)

    return colors

def classify_fruit(color):
    # Define color ranges (in RGB)
    green_apple_range = np.array([100, 180, 0])
    orange_range = np.array([230, 120, 0])

    # Calculate distances
    green_apple_distance = np.linalg.norm(color - green_apple_range)
    orange_distance = np.linalg.norm(color - orange_range)

    if green_apple_distance < orange_distance:
        return "Green Apple"
    else:
        return "Orange"
```

In [69]:
```
image_path = "images.jpeg"
image_array = load_and_process_image(image_path)

# Extract dominant colors
dominant_colors = extract_dominant_colors(image_array)

# Classify each dominant color
classifications = [classify_fruit(color) for color in dominant_colors]

# Count occurrences of each fruit
fruit_counts = Counter(classifications)

print("Fruit Classification Results:")
for fruit, count in fruit_counts.items():
    print(f"{fruit}: {count}")

print("\nDominant Colors (RGB):")
for color in dominant_colors:
    print(color)
```

```
Fruit Classification Results:
Orange: 2

Dominant Colors (RGB):
[243 237 218]
[182 140  44]
```

Step 2 : Object Detection using the extracted features

1. Extract Features from a Reference Object

First, you need a reference object, which is the object you want to detect in other images.

Extract features from this reference object as described previously (shape, edge, texture, color).

2. Extract Features from the Target Image

Segment the target image where you want to detect the object.

Extract the same set of features from all segmented regions or objects in the target image.

3. Compare Features for Object Detection

Compare the features extracted from the reference object with the features extracted from each segmented object in the target image.

Use similarity measures to find the closest match. Here's how you can do it:

Similarity Measures:

Euclidean Distance: Compare feature vectors by calculating the Euclidean distance. The smaller the distance, the more similar the objects are.

Correlation: Measure the correlation between feature vectors. Higher correlation means higher similarity.

Histogram Comparison: If using color histograms, compare them using methods like Chi-square or Intersection.

In [70]:
```
from scipy.spatial import distance

# Function to calculate Euclidean distance between two objects' features (area and perimeter)
def euclidean_distance(obj1, obj2):
    return distance.euclidean([obj1['Area'], obj1['Perimeter']], [obj2['Area'], obj2['Perimeter']])

# Compare features of the first orange and the first apple
dist = euclidean_distance(orange_features[0], apple_features[0])
print(f"Euclidean Distance between the first orange and the first apple: {dist}")
```

```
Euclidean Distance between the first orange and the first apple: 4.4999999315429156
```

4. Locate and Identify the Object

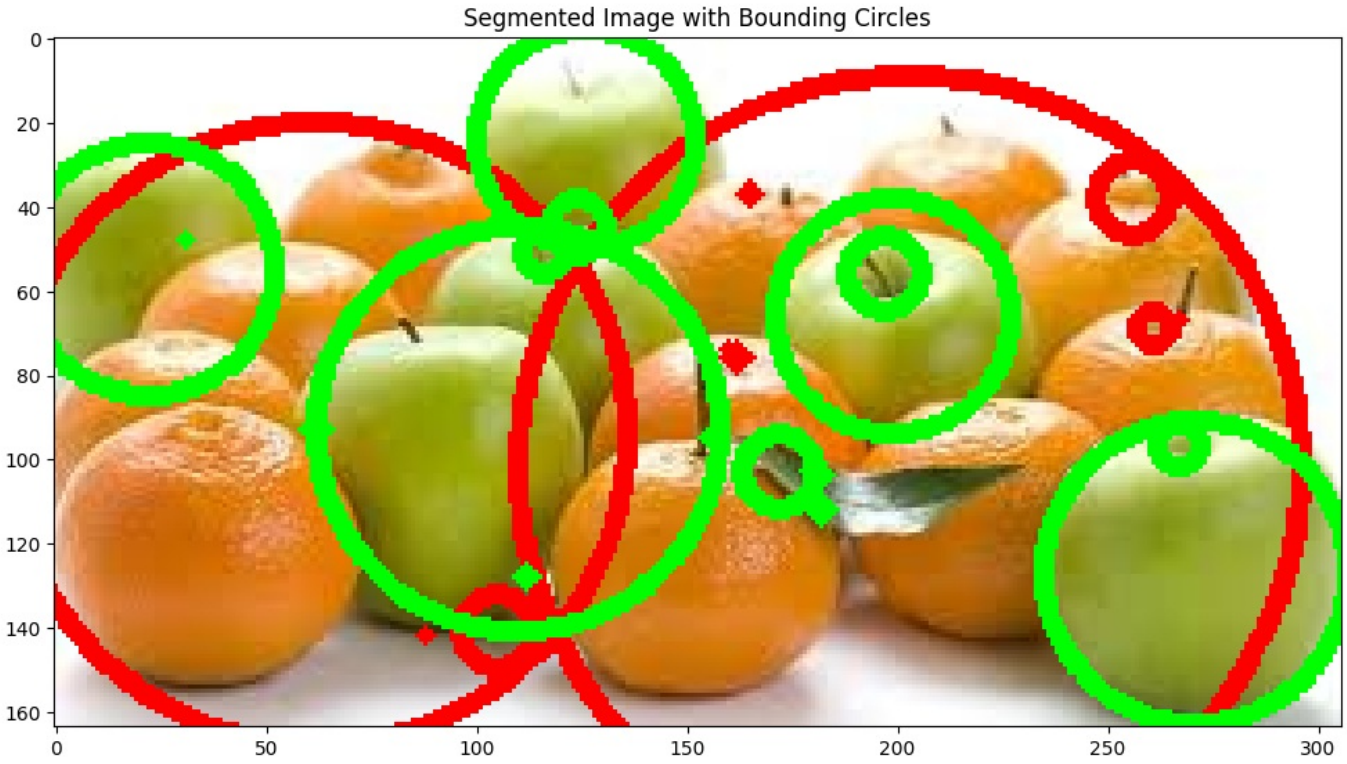
Once you find the best match (i.e., the segmented object with features most similar to the reference object), you can identify and locate the object in the target image.

Mark the detected object using bounding boxes or contours.

In [71]:
```
# Draw bounding circles for oranges
for contour in contours_orange:
```

```
    (x, y), radius = cv2.minEnclosingCircle(contour)
    center = (int(x), int(y))
    radius = int(radius)
    cv2.circle(image_rgb, center, radius, (255, 0, 0), 3)
    # Blue color in RGB

# Draw bounding circles for apples
for contour in contours_apple:
    (x, y), radius = cv2.minEnclosingCircle(contour)
    center = (int(x), int(y))
    radius = int(radius)
    cv2.circle(image_rgb, center, radius, (0, 255, 0), 3)
    # Green color in RGB
```

In [72]:
```
# Display the image with bounding circles
plt.figure(figsize=(12, 12))
plt.imshow(image_rgb)
plt.title("Segmented Image with Bounding Circles")
plt.show()
```



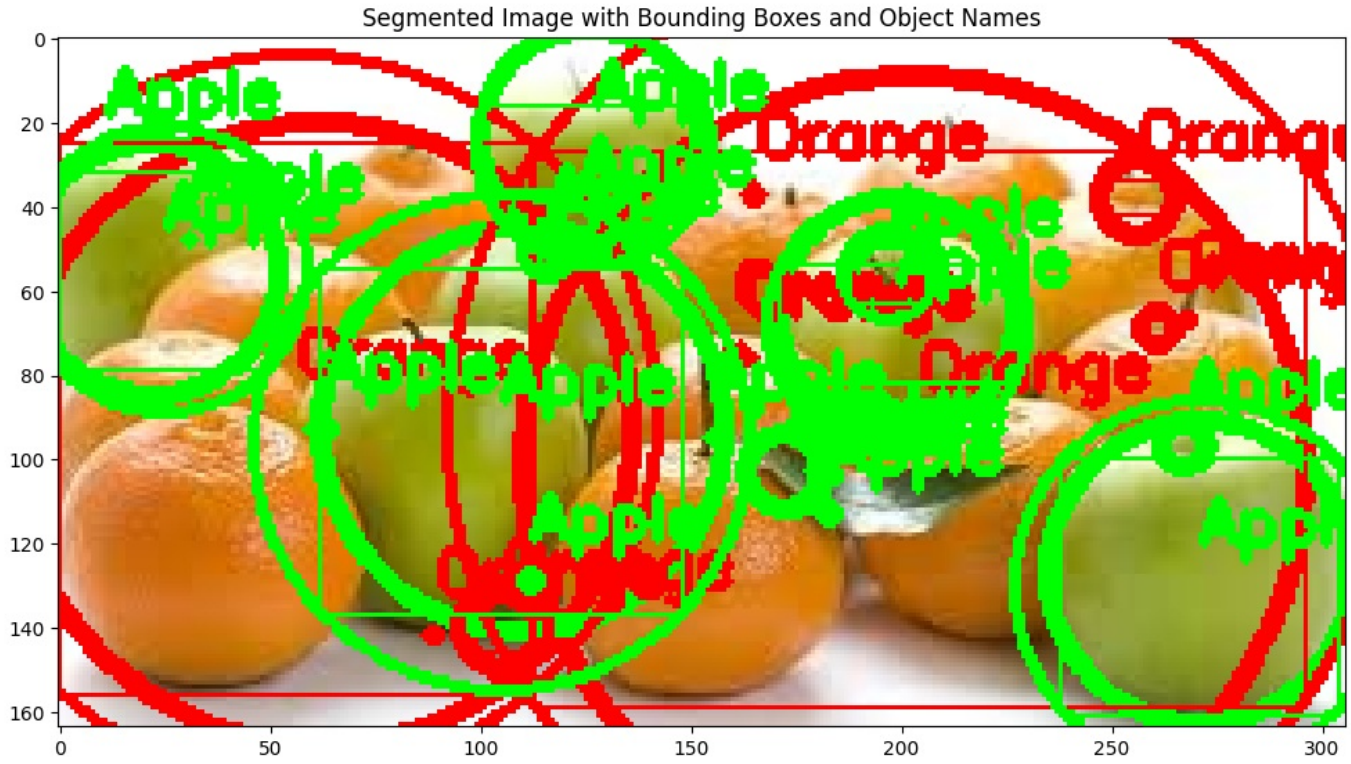
5. Visualize the Results Draw the bounding box or highlight the detected object in the target image to show the result of the object detection.

In [73]:
```
# Draw bounding boxes with object names for oranges
for feature in orange_features:
    x, y, w, h = feature['Bounding Box']
    cv2.rectangle(image_rgb, (x, y), (x + w, y + h), (255, 0, 0), 1) # Blue color in RGB

    # Draw enclosing circle
    contour = np.array([[x, y], [x + w, y], [x + w, y + h], [x, y + h]], dtype=np.int32)
    (cx, cy), radius = cv2.minEnclosingCircle(contour)
    center = (int(cx), int(cy))
    radius = int(radius)
    cv2.circle(image_rgb, center, radius, (255, 0, 0), 2) # Blue color in RGB

    # Put text for orange
    cv2.putText(image_rgb, "Orange", (int(cx), int(cy) - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 0, 0), 2)
```

In [74]:
```
# Draw bounding boxes with object names for apples
for feature in apple_features:
    x, y, w, h = feature['Bounding Box']
    cv2.rectangle(image_rgb, (x, y), (x + w, y + h), (0, 255, 0), 1) # Green color in RGB

    # Draw enclosing circle
    contour = np.array([[x, y], [x + w, y], [x + w, y + h], [x, y + h]], dtype=np.int32)
    (cx, cy), radius = cv2.minEnclosingCircle(contour)
    center = (int(cx), int(cy))
    radius = int(radius)
    cv2.circle(image_rgb, center, radius, (0, 255, 0), 2) # Green color in RGB

    # Put text for apple
```

```
    cv2.putText(image_rgb, "Apple", (int(cx), int(cy) - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)
```

In [75]:
```
# Display the image with bounding boxes and object names
plt.figure(figsize = (12,12))
plt.imshow(image_rgb)
plt.title("Segmented Image with Bounding Boxes and Object Names")
plt.show()
```

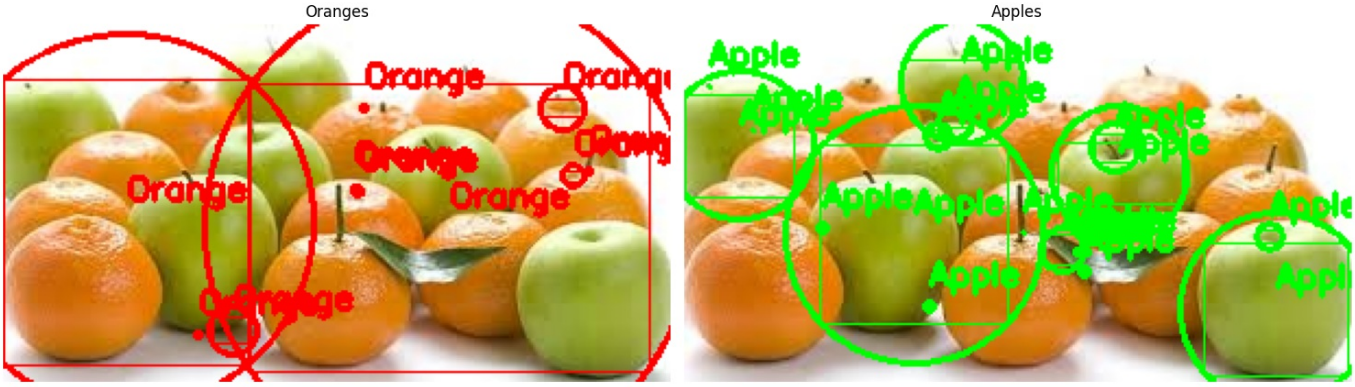


In [76]:
```
image_rgb = cv2.cvtColor(cv2.imread('images.jpeg'), cv2.COLOR_BGR2RGB)

# Create copies of the original image for each display
image_oranges = image_rgb.copy()  # Copy for oranges
image_apples = image_rgb.copy()   # Copy for apples

# Draw bounding boxes and labels for oranges
for feature in orange_features:
    x, y, w, h = feature['Bounding Box']
    cv2.rectangle(image_oranges, (x, y), (x + w, y + h), (255, 0, 0), 1)  # Blue color in RGB

    # Draw enclosing circle
    contour = np.array([[x, y], [x + w, y], [x + w, y + h], [x, y + h]], dtype=np.int32)
    (cx, cy), radius = cv2.minEnclosingCircle(contour)
    center = (int(cx), int(cy))
    radius = int(radius)
    cv2.circle(image_oranges, center, radius, (255, 0, 0), 2)  # Blue color in RGB

    # Put text for orange
    cv2.putText(image_oranges, "Orange", (int(cx), int(cy) - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 0, 0), 2

# Draw bounding boxes and labels for apples
for feature in apple_features:
    x, y, w, h = feature['Bounding Box']
    cv2.rectangle(image_apples, (x, y), (x + w, y + h), (0, 255, 0), 1)  # Green color in RGB

    # Draw enclosing circle
    contour = np.array([[x, y], [x + w, y], [x + w, y + h], [x, y + h]], dtype=np.int32)
    (cx, cy), radius = cv2.minEnclosingCircle(contour)
    center = (int(cx), int(cy))
    radius = int(radius)
    cv2.circle(image_apples, center, radius, (0, 255, 0), 2)  # Green color in RGB

    # Put text for apple
    cv2.putText(image_apples, "Apple", (int(cx), int(cy) - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)

# Combine both images into one for comparison
combined_image = cv2.addWeighted(image_oranges, 0.5, image_apples, 0.5, 0)  # Blend both images
```

Easier Visualisation

In [77]:
```
# Display the individual and combined images
plt.figure(figsize=(15, 5))
```

```
# Show oranges
plt.subplot(1, 2, 1)
plt.imshow(image_oranges)
plt.title("Oranges")
plt.axis('off')

# Show apples
plt.subplot(1, 2, 2)
plt.imshow(image_apples)
plt.title("Apples")
plt.axis('off')

plt.tight_layout()
plt.show()
```



In [ ]: