UID: 120061660, 119646878, 117987683

Names: Aryan Tajne, Hita Thota, Amruth Nare

CMSC426 Project 1: Color Segmentation using GMM

Introduction

Have you ever played with these adorable Nao robots? Click here to watch a cool demo.

Nao robots are star players in RoboCup, an annual autonomous robot soccer competitions. Would you like to help us in Nao’s soccer training? We need to train Nao to detect a soccer ball and estimate the depth of the ball to know how far to kick.

Nao’s training has two phases:

- Color Segmentation using Gaussian Mixture Model (GMM)
- Ball Distance Estimation

What you need to do

To make logistics easier, we have collected camera data from Nao robot on behalf of you and saved the data in the form of color images. Click here to download, or **run the following code block to download the training image folder to the file directory of the notebook**. The image names represent the depth of the ball from Nao robot in centimeters. -We will release the test dataset 48 hours before the deadline. **Test images are available here to download.**

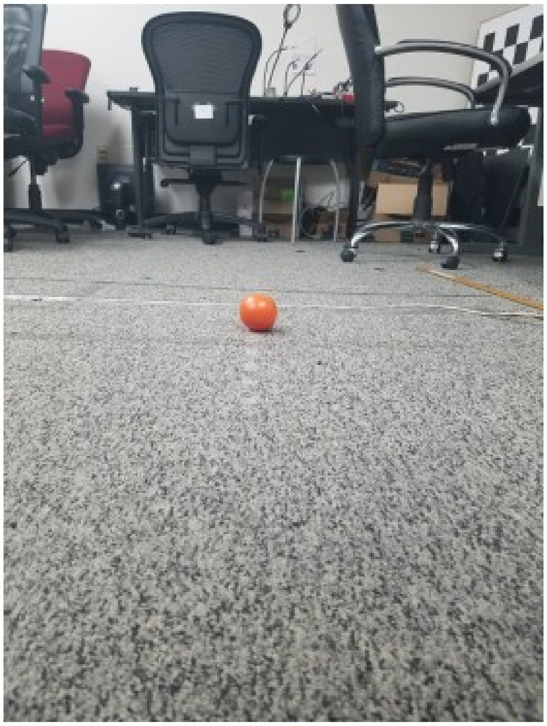
```
# # Download training images from Google Drive
import gdown
gdown.download_folder(id="18Mx2Xc9UNFZYajYu9vfmRFlFCcna5I0J",
quiet=True, use_cookies=False)
```

```
['/content/train_images/68.jpg',
 '/content/train_images/76.jpg',
 '/content/train_images/91.jpg',
 '/content/train_images/99.jpg',
 '/content/train_images/106.jpg',
 '/content/train_images/114.jpg',
 '/content/train_images/121.jpg',
 '/content/train_images/137.jpg',
 '/content/train_images/144.jpg',
 '/content/train_images/152.jpg',
 '/content/train_images/160.jpg',
 '/content/train_images/168.jpg',
 '/content/train_images/176.jpg',
 '/content/train_images/192.jpg',
 '/content/train_images/200.jpg',
 '/content/train_images/208.jpg',
 '/content/train_images/216.jpg',
 '/content/train_images/223.jpg',
 '/content/train_images/231.jpg',
 '/content/train_images/248.jpg',
 '/content/train_images/256.jpg',
 '/content/train_images/264.jpg',
 '/content/train_images/280.jpg']
```

```
# # Download testing images from Google Drive
gdown.download_folder(id="1Yl4_5O_ZEkz_KJVs0_vS5TrZUqMYkwr4",
quiet=True, use_cookies=False)
```

```
['/content/test_images/1.jpg',
 '/content/test_images/2.jpg',
 '/content/test_images/3.jpg',
 '/content/test_images/4.jpg',
 '/content/test_images/5.jpg',
 '/content/test_images/6.jpg',
 '/content/test_images/7.jpg',
 '/content/test_images/8.jpg']
```

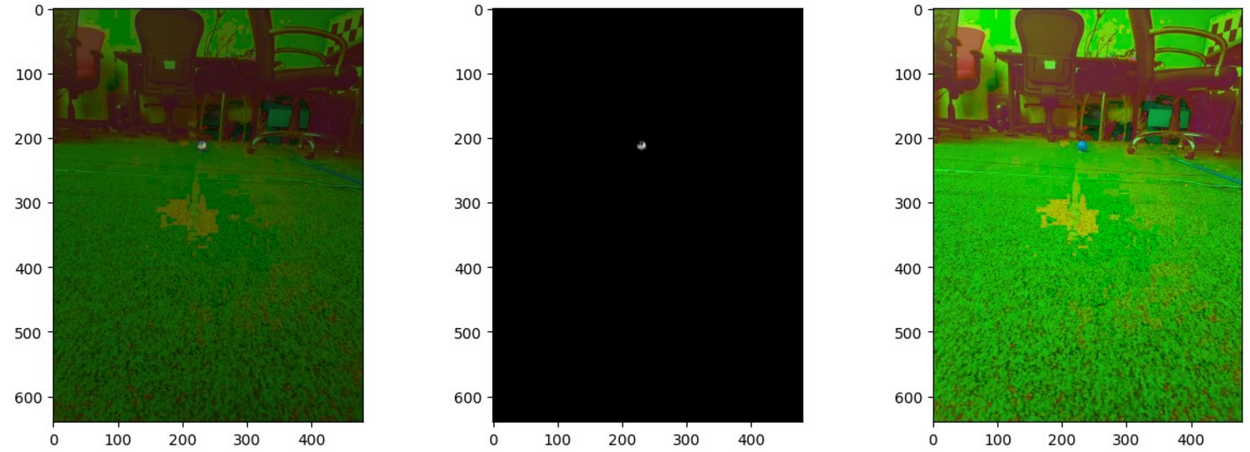
```
# Check whether the training images were successfully imported
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np

train_image = mpimg.imread('./train_images/106.jpg')
plt.imshow(train_image)
plt.axis("off")
plt.show()
```

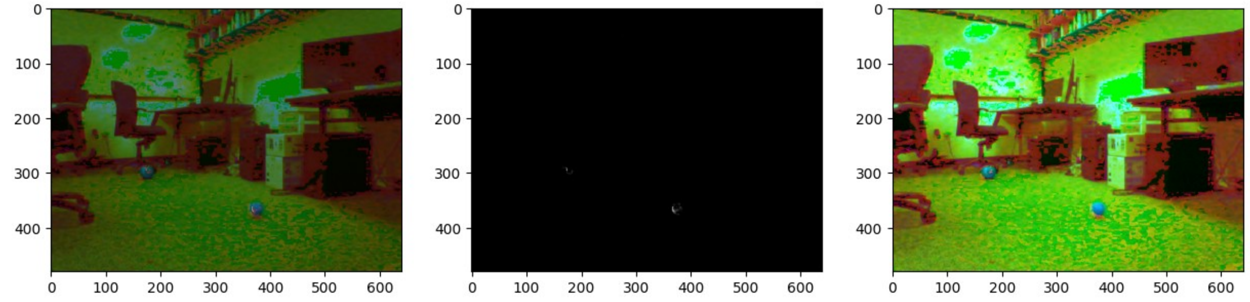


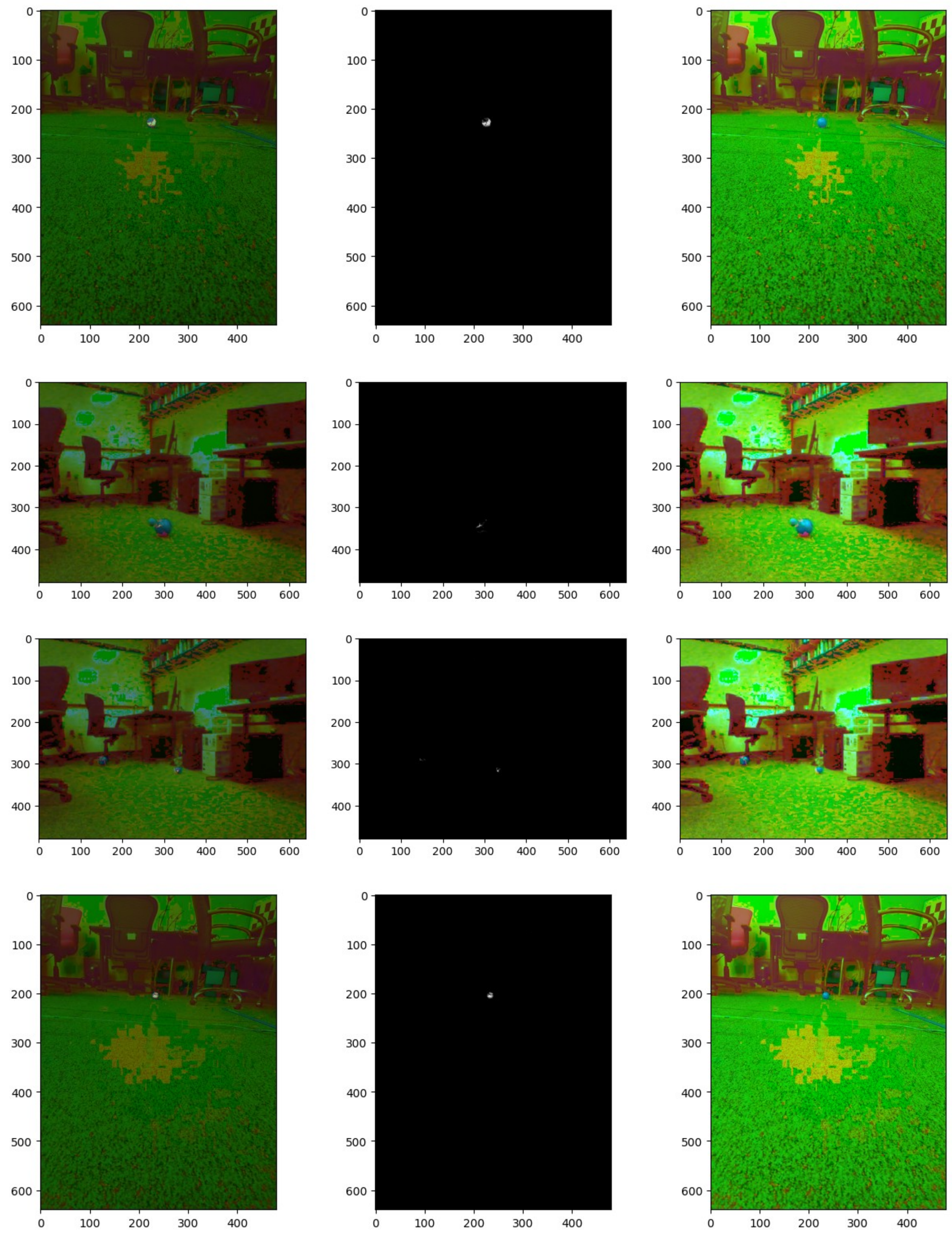
Problem Statement

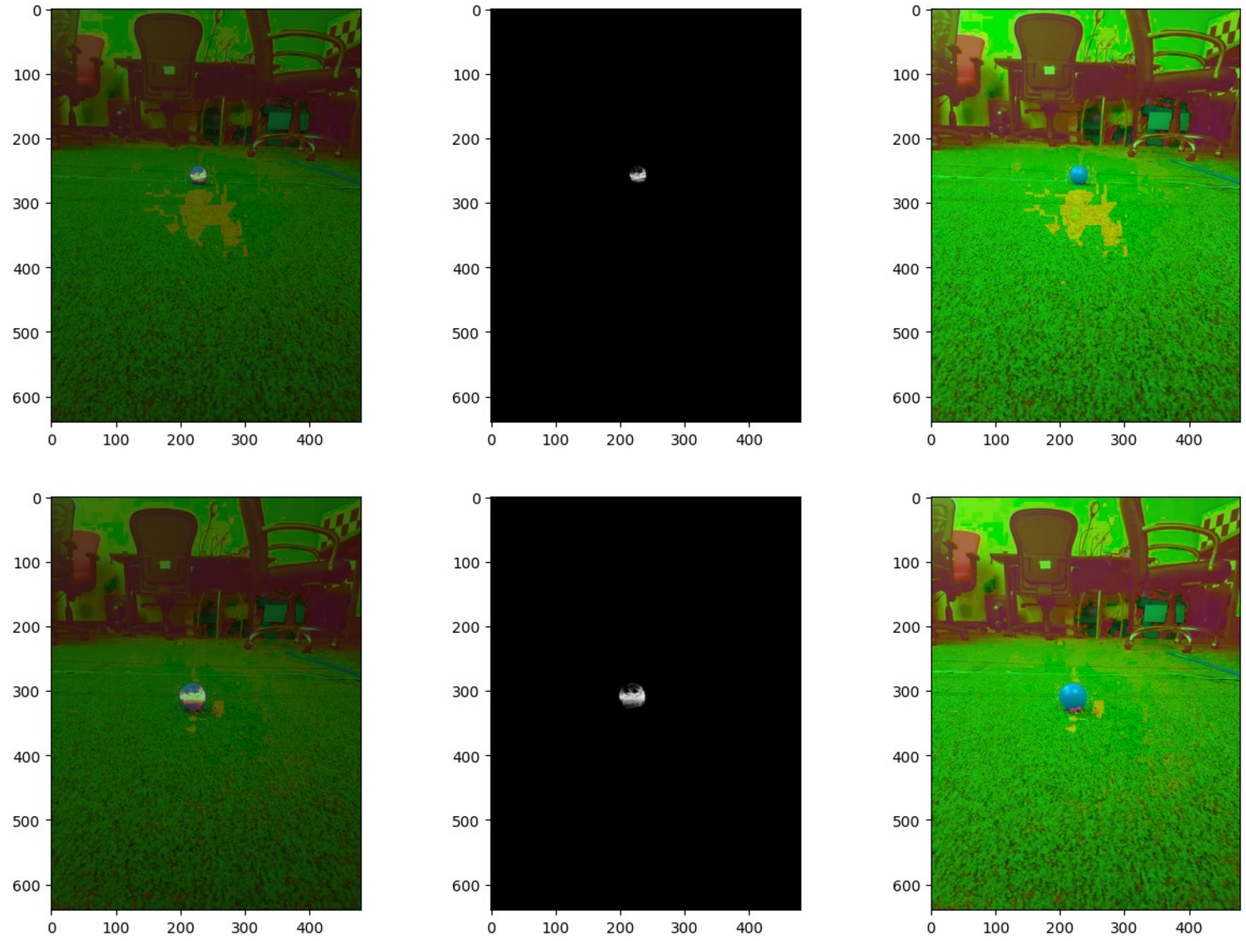
1. Write Python code to cluster the orange ball using Single Gaussian [30 points]

```
import os
import cv2
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from matplotlib.patches import Ellipse
from scipy.optimize import curve_fit
import plotly.graph_objs as go
import numpy as np
from scipy.stats import chi2
from sklearn.cluster import KMeans

train = []

for image in os.listdir('./train_images'):
    img = cv2.imread('./train_images/' + image)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2HLS)
    train.append(img)

train = np.array(train)

orange = []

for img in train:
    lower = np.array([0, 100, 120])
    upper = np.array([30, 240, 250])
    mask = cv2.inRange(img, lower, upper)
    result = cv2.bitwise_and(img, img, mask=mask)
    orange.append(result)

# for i in range(len(orange)):
#     plt.figure(figsize=(15,5))
#     plt.subplot(131)
#     plt.imshow(train[i])
#     plt.subplot(132)
#     plt.imshow(orange[i])
#     plt.show()

orange = np.array(orange)
# print(orange.shape)
flattened = orange.reshape(-1, 3) #23 images of 480x640 pixels =
737280 pixels and then 3 channels HSV
flattened = flattened[(flattened[:, 0] != 0) & (flattened[:, 1] != 0)
& (flattened[:, 2] != 0)] #remove black pixels because they are not
orange
# print(flattened.shape)

mean = np.mean(flattened, axis=0)
cov = np.cov(flattened.T)

def pdf(x, mean, cov):
    x = x - mean
    det = np.linalg.det(cov)
    inv = np.linalg.inv(cov)
    norm = 1.0 / (((2 * np.pi)**(3/2)) * np.sqrt(det))
    exp = np.exp(-0.5 * np.dot(x.T, np.dot(inv, x)))
    pdf = norm * exp
    return pdf

def gaussian_pdf(x, mean, cov): #for 2d
    d = x.shape[1]
    x_m = x - mean
    inv_cov = np.linalg.inv(cov)
    det_cov = np.linalg.det(cov)

    log_norm_const = -0.5 * (d * np.log(2 * np.pi) + np.log(det_cov))
    log_exp_term = -0.5 * np.sum((x_m @ inv_cov) * x_m, axis=1)

    return np.exp(log_norm_const + log_exp_term)

threshold = 1e-5
prior = 0.25

def detector(image, mean, cov, threshold, prior):
    original_shape = image.shape
    image = image.reshape(-1, 3)
    pdf_values = gaussian_pdf(image, mean, cov)
    probability = (pdf_values * prior)
    #probability[probability < threshold] = 0
    return probability.reshape(original_shape[:2])

for image in os.listdir('./test_images'):
    img_arr = []
    img = cv2.imread('./test_images/' + image)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2HLS)
    detected = detector(img, mean, cov, threshold, prior)
    plt.figure(figsize=(15,5))
    plt.subplot(131)
    plt.imshow(img)
    plt.imshow(detected, cmap='hot', alpha=0.5)
    plt.subplot(132)
    plt.imshow(detected, cmap='gray')
    plt.subplot(133)
    plt.imshow(img)
    plt.show()
    img_arr.append(detected)
```



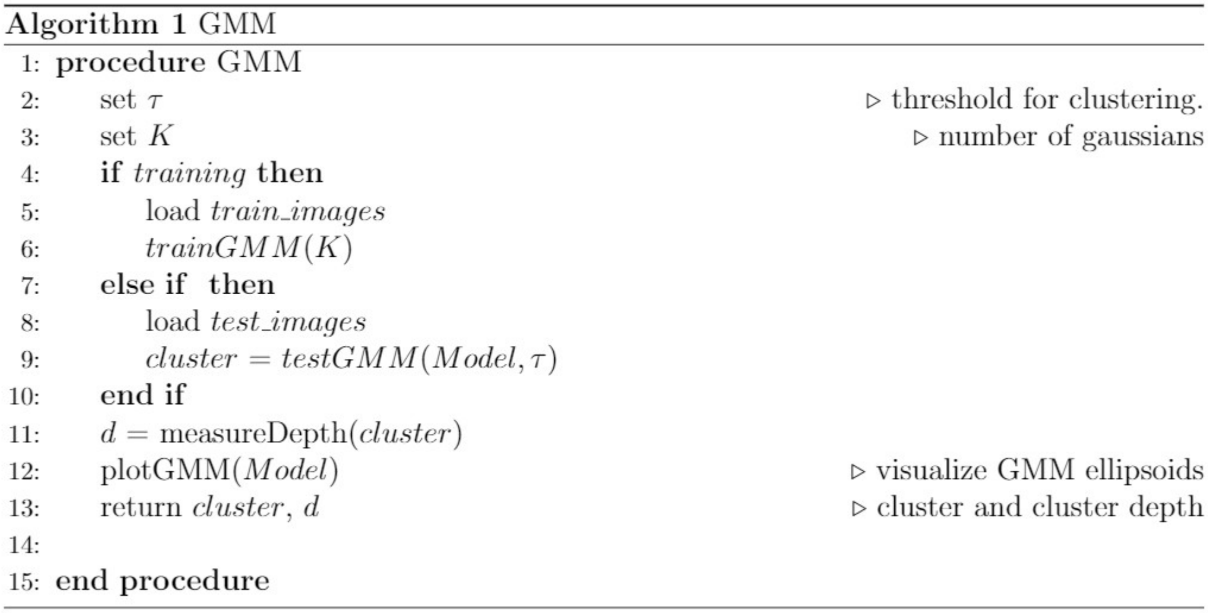



0
100
200
300
400
500
600
0
100
200
300
400
0
100
200
300
400
0
100
200
300
400
500
600



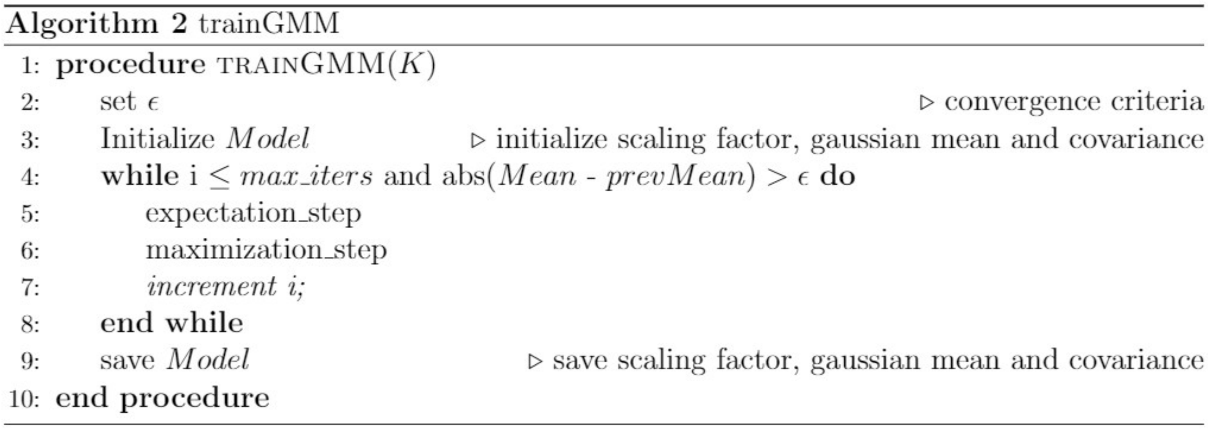
1. Write Python code to cluster the orange ball using Gaussian Mixture Model [40 points] and estimate the distance to the ball [20 points]. Also, plot all the GMM ellipsoids [10 points].

You are NOT allowed to use any built-in Python package(s) like *sklearn.mixture.GaussianMixture* for GMM. To help you with code implementation, we have given the pseudocode :-)



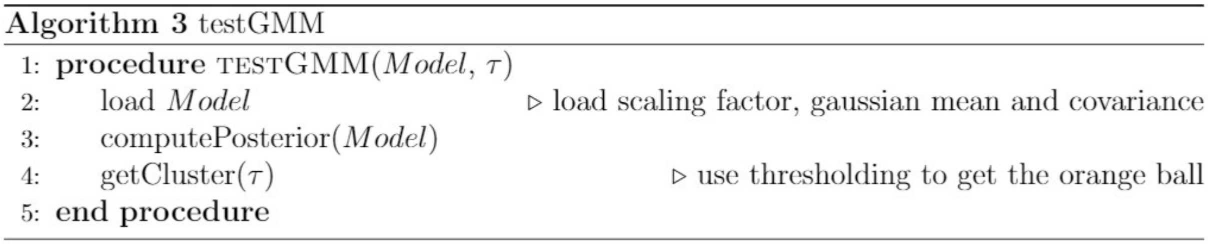
Algorithm 1 GMM

```
procedure GMM
    set $\tau$                                        ▷ threshold for clustering.
    set $K$                                           ▷ number of gaussians
    if $training$ then
        load $train\_images$
        $trainGMM(K)$
    else if  then
        load $test\_images$
        $cluster = testGMM(Model, \tau)$
    end if
    $d$ = measureDepth($cluster$)
    plotGMM($Model$)                                  ▷ visualize GMM ellipsoids
    return $cluster$, $d$                             ▷ cluster and cluster depth

end procedure
```



Algorithm 2 trainGMM

```
procedure TRAINGMM($K$)
    set $\epsilon$                                    ▷ convergence criteria
    Initialize $Model$       ▷ initialize scaling factor, gaussian mean and covariance
    while i $\leq$ $max\_iters$ and abs($Mean$ - $prevMean$) $> \epsilon$ do
        expectation_step
        maximization_step
        $increment\ i;$
    end while
    save $Model$             ▷ save scaling factor, gaussian mean and covariance
end procedure
```



Algorithm 3 testGMM

```
procedure TESTGMM($Model$, $\tau$)
    load $Model$             ▷ load scaling factor, gaussian mean and covariance
    computePosterior($Model$)
    getCluster($\tau$)       ▷ use thresholding to get the orange ball
end procedure
```

```python
def gaussian_pdf(x, mean, cov): #for 2d
    d = x.shape[1]
    x_m = x - mean
    inv_cov = np.linalg.inv(cov)
    det_cov = np.linalg.det(cov)

    log_norm_const = -0.5 * (d * np.log(2 * np.pi) + np.log(det_cov))
    log_exp_term = -0.5 * np.sum((x_m @ inv_cov) * x_m, axis=1)

    return np.exp(log_norm_const + log_exp_term)

def pdf(x, mean, cov): #for 1d
    x = x - mean
    det = np.linalg.det(cov)
    inv = np.linalg.inv(cov)
    norm = 1.0 / ((2 * np.pi)**(3/2) * np.sqrt(det))
    exp = np.exp(-0.5 * np.dot(x.T, np.dot(inv, x)))
    pdf = norm * exp
    return pdf

def expectation_step(data, weights, means, covariances):
    N = data.shape[0]
    K = len(weights)

    resp = np.zeros((N, K))

    for i in range(K):
        resp[:, i] = weights[i] * gaussian_pdf(data, means[i],
covariances[i])

    resp_sum = resp.sum(axis=1, keepdims=True)
    resp = np.divide(resp, resp_sum, where=resp_sum!=0)

    return resp

def maximization_step(data, responsibilities):
    N, K = responsibilities.shape
    d = data.shape[1]

    resp = responsibilities.sum(axis=0)
    means = (responsibilities.T @ data) / np.maximum(resp[:,
np.newaxis], 1e-8)
    covariances = np.zeros((K, d, d))

    for k in range(K):
        diff = data - means[k]
        covariances[k] = (responsibilities[:, k, np.newaxis] * diff).T
@ diff / np.maximum(resp[k], 1e-8) #cal covariance matrix
        covariances[k] += np.eye(d) * 1e-6  # numerical stability and
to avoid those warning messages

    weights = resp / N
    return means, covariances, weights

def compute_likelihood_vectorized(pixels, means, covariances):
    K = len(means)
    N = pixels.shape[0]
    likelihoods = np.zeros((N, K))

    for i in range(K):
        likelihoods[:, i] = gaussian_pdf(pixels, mean=means[i],
cov=covariances[i])

    return likelihoods


def compute_posterior_vectorized(pixels, scalars, means, covariances):
    likelihoods = compute_likelihood_vectorized(pixels, means,
covariances)
    weighted_likelihoods = scalars * likelihoods
    sum_weighted_likelihoods = np.sum(weighted_likelihoods, axis=1,
keepdims=True)
    return weighted_likelihoods / sum_weighted_likelihoods

def trainGMM(K, data, max_iters=1000, e=1e-12):

    N, d = data.shape

    np.random.seed(42)
    kmeans = KMeans(n_clusters=K, random_state=42).fit(data)
    means = kmeans.cluster_centers_
    covariances = np.array([np.cov(data.T) + np.eye(d) * 1e-3 for _ in
range(K)]) #numerical stability
    weights = np.ones(K) / K

    prev_means = np.zeros_like(means)
    i = 0

    while i < max_iters and np.linalg.norm(means - prev_means) > e:
        prev_means = means.copy()

        responsibilities = expectation_step(data, weights, means,
covariances)
        means, covariances, weights = maximization_step(data,
responsibilities)

        i += 1

    return weights, means, covariances


def testGMM(Model_parameters, threshold, prior):
    scalars, means, covariances = Model_parameters

    test_image_folder = './test_images'
    result_folder = './result_images'
    if not os.path.exists(result_folder):
        os.makedirs(result_folder)

    images = [f for f in os.listdir(test_image_folder) if
f.endswith('.jpg')]

    cluster_parameters = []

    for image_name in images:
        img = cv2.imread(os.path.join(test_image_folder, image_name))
        img_hls = cv2.cvtColor(img, cv2.COLOR_BGR2HLS)
        height, width, _ = img_hls.shape
        pixels = img_hls.reshape((-1, 3))


        posteriors = compute_posterior_vectorized(pixels, scalars,
means, covariances)
        posteriors = 1 - posteriors
        ball_component = np.argmin(np.sum(posteriors, axis=0))
        segmentation_map = posteriors[:,
ball_component].reshape((height, width))
        binary_map = (segmentation_map > threshold).astype(np.uint8)
        binary_map = (binary_map * 255).astype(np.uint8)
        segmentation_map = 1 - segmentation_map
        contours, _ = cv2.findContours(binary_map, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

        if contours:
            largest_contour = max(contours, key=cv2.contourArea)
            ((x, y), radius) = cv2.minEnclosingCircle(largest_contour)
            cluster_parameters.append((x, y, radius))

        fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(15, 5))

        ax1.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
        ax1.set_title('Original Image')
        ax1.axis('off')

        ax2.imshow(binary_map, cmap='hot')
        ax2.set_title('Binary Map')
        ax2.axis('off')

        ax3.imshow(segmentation_map, cmap='gray')
        ax3.set_title('Segmentation Map')
        ax3.axis('off')
        plt.tight_layout()
        plt.savefig(os.path.join(result_folder,
f'segmented_{image_name}'))
        plt.show()
        plt.close()

    print("Segmentation complete. Results saved in the result
folder.")
    return cluster_parameters, binary_map

def measureDepth(cluster_parameters):
    radii = np.array([params[2] for params in cluster_parameters])
    print("Radii:", radii)

    # Sort radii in descending order (larger radius = closer object)
    sorted_indices = np.argsort(radii)[::-1]
    sorted_radii = radii[sorted_indices]

    # Create relative distances (1 for closest, increasing for
farther)
    relative_distances = np.arange(1, len(radii) + 1)

    # Define the inverse model function
    def inverse_model(x, a, b):
        return a / (x + b)

    # Fit the model to our data
    popt, pcov = curve_fit(inverse_model, relative_distances,
sorted_radii, p0=[np.max(radii), 1], bounds=([0, 0], [np.inf,
np.inf]))

    # Extract the fitted parameters
    a, b = popt
    print("Fitted parameters: a =", a, "b =", b)

    # Function to estimate relative distance based on radius
    def estimate_relative_distance(radius):
        return (a / radius) - b

    # Estimate relative distances for all balls
    estimated_distances = np.array([estimate_relative_distance(radius)
for radius in radii])
    normalized_distances = 1 + (len(radii) - 1) * (estimated_distances
- np.min(estimated_distances)) / (np.max(estimated_distances) -
np.min(estimated_distances))
    final_distances = normalized_distances[np.argsort(sorted_indices)]

    print("Estimated relative distances:", final_distances)
    return final_distances.tolist()
```

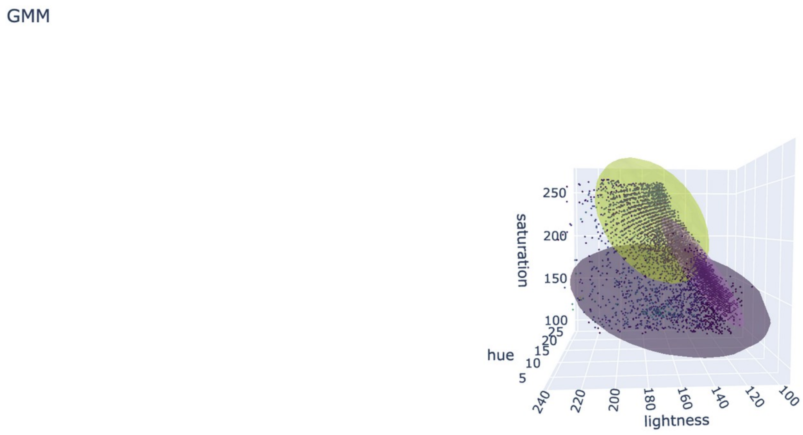
```
def plotGMM(Model_parameters, flattened):

    scalars, means, covariances = Model_parameters
    K = len(scalars)

    fig = go.Figure()
    fig.add_trace(go.Scatter3d(
        x=flattened[:, 0], y=flattened[:, 1], z=flattened[:, 2],
        mode='markers',
        marker=dict(size=1, color=flattened[:, 0],
colorscale='Viridis'),
        showlegend=False
    ))
    def sphere_points(n):
        theta = np.random.uniform(0, 2*np.pi, n)
        phi = np.arccos(np.random.uniform(-1, 1, n))
        x = np.sin(phi) * np.cos(theta)
        y = np.sin(phi) * np.sin(theta)
        z = np.cos(phi)
        return np.column_stack((x, y, z))

    for i in range(K):
        mean = means[i]
        covariance = covariances[i]

        points = sphere_points(500)

        eigenvalues, eigenvectors = np.linalg.eig(covariance)
        scaling_matrix = np.sqrt(eigenvalues) * np.sqrt(chi2.ppf(0.95,
3))
        points = np.dot(points * scaling_matrix, eigenvectors.T) +
mean

        fig.add_trace(go.Mesh3d(
            x=points[:, 0], y=points[:, 1], z=points[:, 2],
            alphahull=0,
            opacity=0.3,
            color=f'rgb({np.random.randint(0, 256)},
{np.random.randint(0, 256)}, {np.random.randint(0, 256)})',
            name=f'Gaussian {i + 1}'
        ))

    fig.update_layout(
        title='GMM',
        scene=dict(xaxis_title='hue', yaxis_title='lightness',
zaxis_title='saturation')
    )

    fig.show()
```

```
threshold = 0.000065
prior = 0.00025
K = 3

train = []

for image in os.listdir('./train_images'):
    img = cv2.imread('./train_images/' + image)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2HLS)
    train.append(img)

train = np.array(train)

orange = []

for img in train:
    lower = np.array([0, 100, 120])
    upper = np.array([30, 240, 250])
    mask = cv2.inRange(img, lower, upper)
    result = cv2.bitwise_and(img, img, mask=mask)
    orange.append(result)

orange = np.array(orange)

flattened = orange.reshape(-1, 3)
print(flattened.shape)
flattened = flattened[(flattened[:, 0] > 5) & (flattened[:, 1] > 5) &
(flattened[:, 2] > 5)]
print(flattened.shape)

Model_parameters = trainGMM(K, flattened)

print("Final GMM parameters:")
print("Weights:", Model_parameters[0])
print("Means:", Model_parameters[1])
print("Covariances:", Model_parameters[2])

plotGMM(Model_parameters, flattened)
```

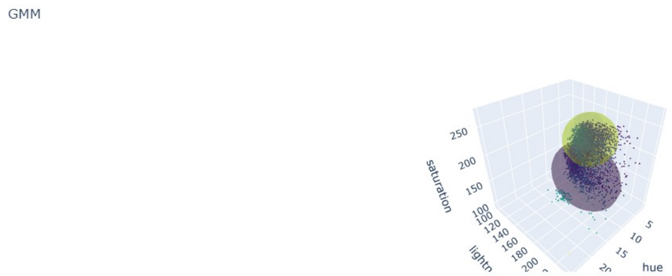
```
(7065600, 3)
(7575, 3)
Final GMM parameters:
Weights: [0.14540747 0.36804394 0.48654859]
Means: [[  9.08631472 165.98758772 144.37392698]
 [  9.46824821 175.32696651 216.80280146]
 [  7.2243035  142.89897002 172.72548639]]
Covariances: [[[ 6.21167596e+00  2.43056411e+01 -3.29690408e+00]
  [ 2.43056411e+01  6.05990279e+02  1.17538375e+02]
  [-3.29690408e+00  1.17538375e+02  3.37543404e+02]]

 [[ 1.92828789e+00 -4.78861375e-01  7.81510995e+00]
```

```
  [-4.78861375e-01  2.04337120e+02  1.03586493e+02]
  [ 7.81510995e+00  1.03586493e+02  4.01713761e+02]]

 [[ 1.06294443e+00  6.50220531e+00  1.17647618e+01]
  [ 6.50220531e+00  9.14423900e+01  1.43271346e+02]
  [ 1.17647618e+01  1.43271346e+02  2.80696920e+02]]]
```

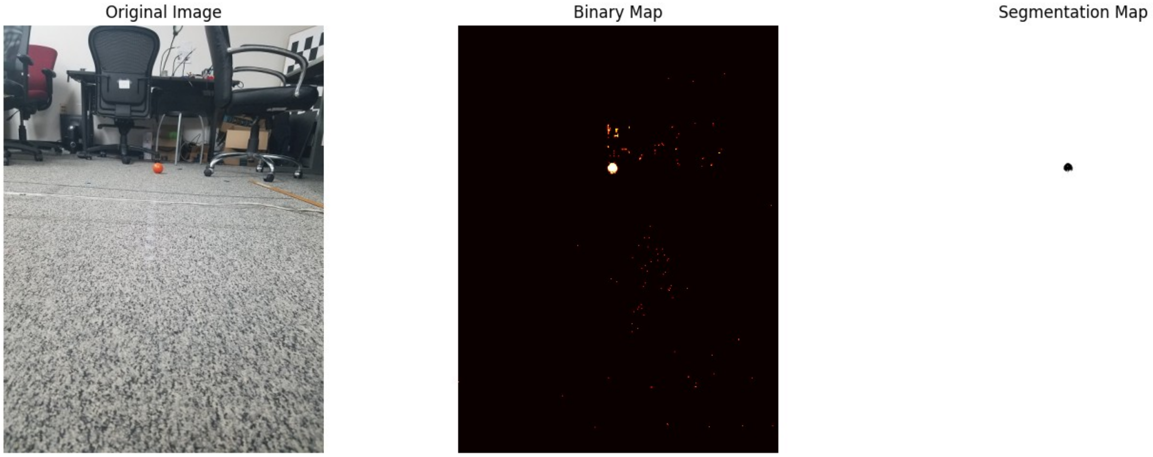
3D Plot:



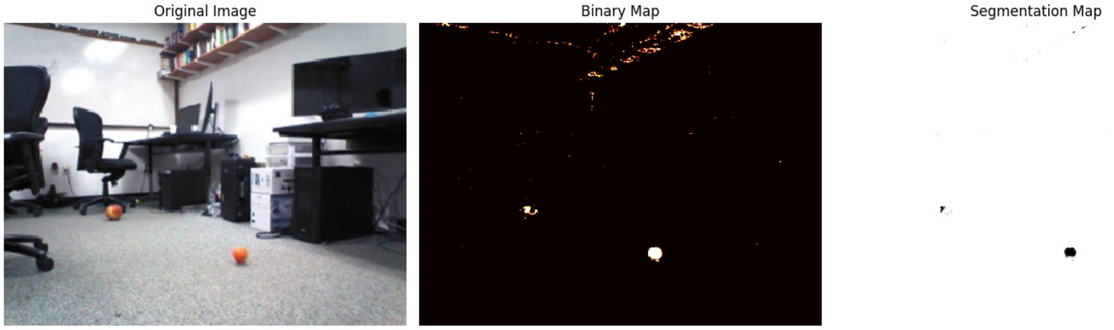
Another view of the plot:



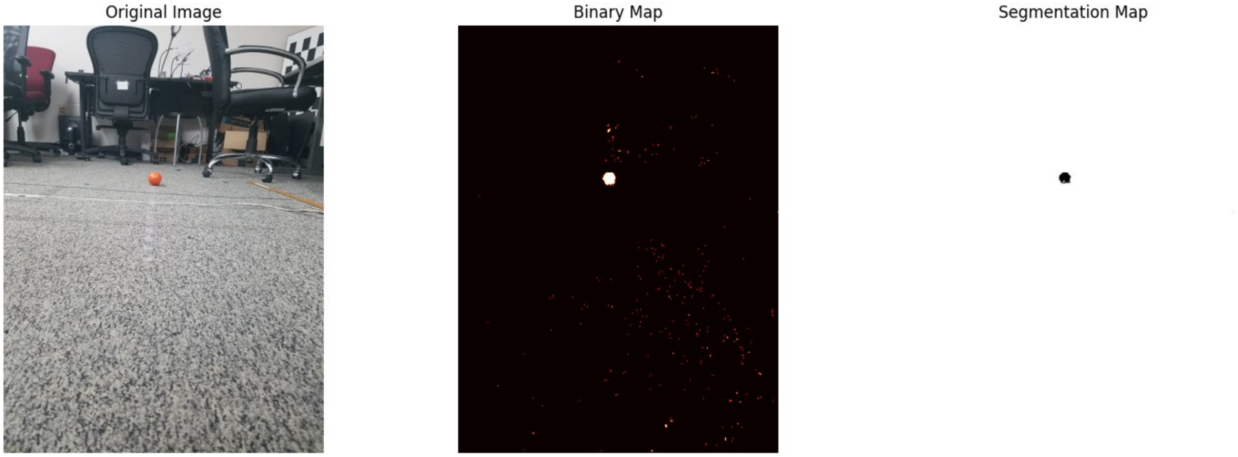
```
cluster_parameters, bin_2 = testGMM(Model_parameters,threshold, prior)
print(cluster_parameters)
distance = measureDepth(cluster_parameters)

<ipython-input-9-8ca9b5013fbe>:67: RuntimeWarning:

invalid value encountered in divide
```



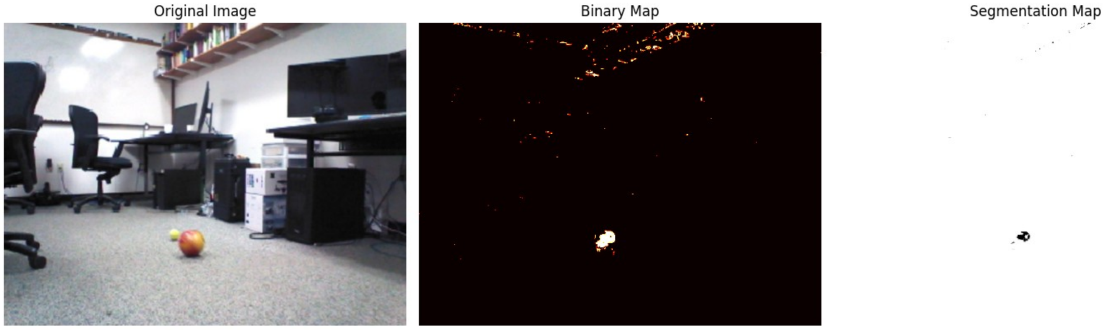
```
<ipython-input-9-8ca9b5013fbe>:67: RuntimeWarning:
invalid value encountered in divide
```



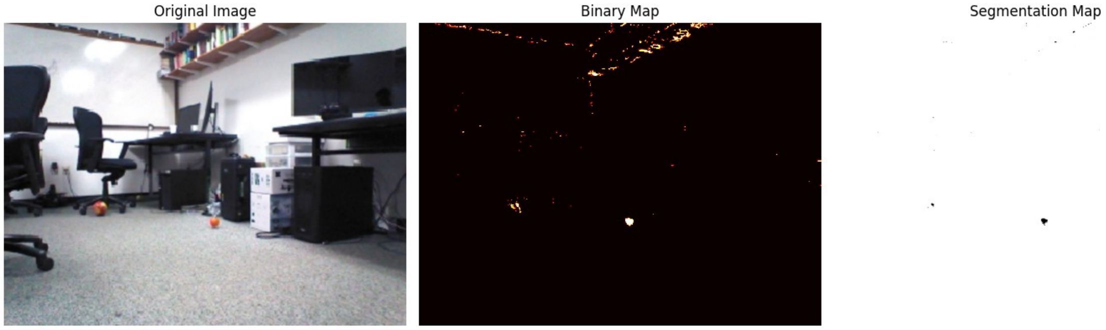
```
<ipython-input-9-8ca9b5013fbe>:67: RuntimeWarning:
invalid value encountered in divide
```



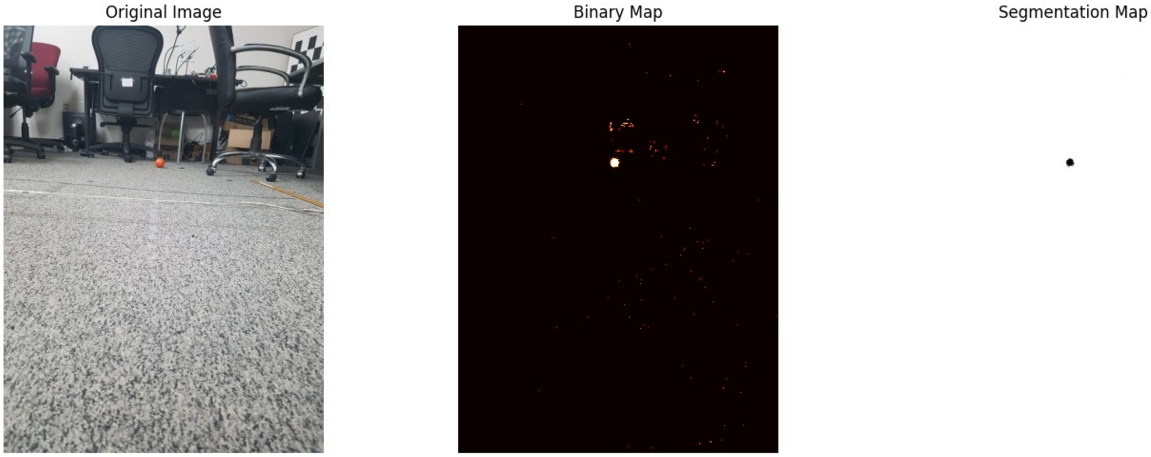
```
<ipython-input-9-8ca9b5013fbe>:67: RuntimeWarning:

invalid value encountered in divide
```



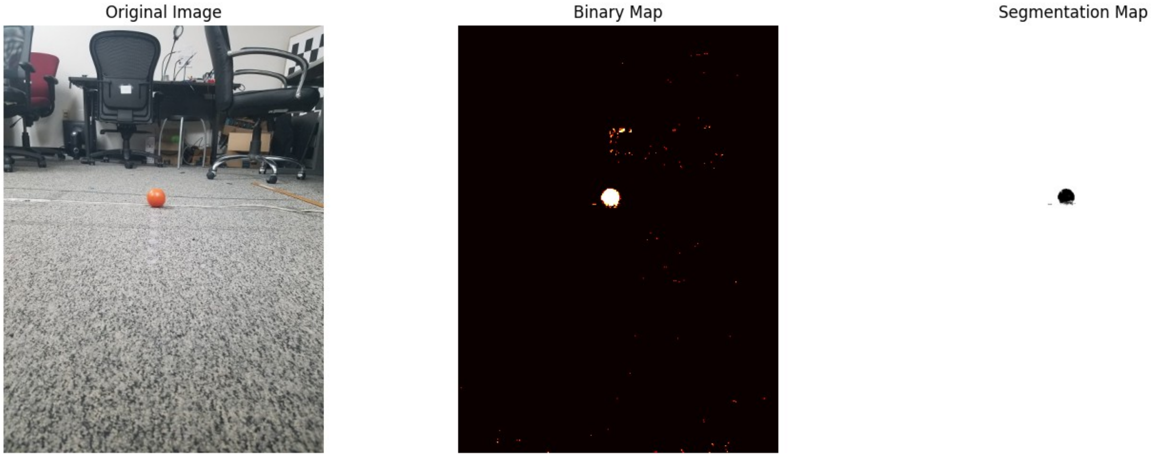
```
<ipython-input-9-8ca9b5013fbe>:67: RuntimeWarning:

invalid value encountered in divide
```



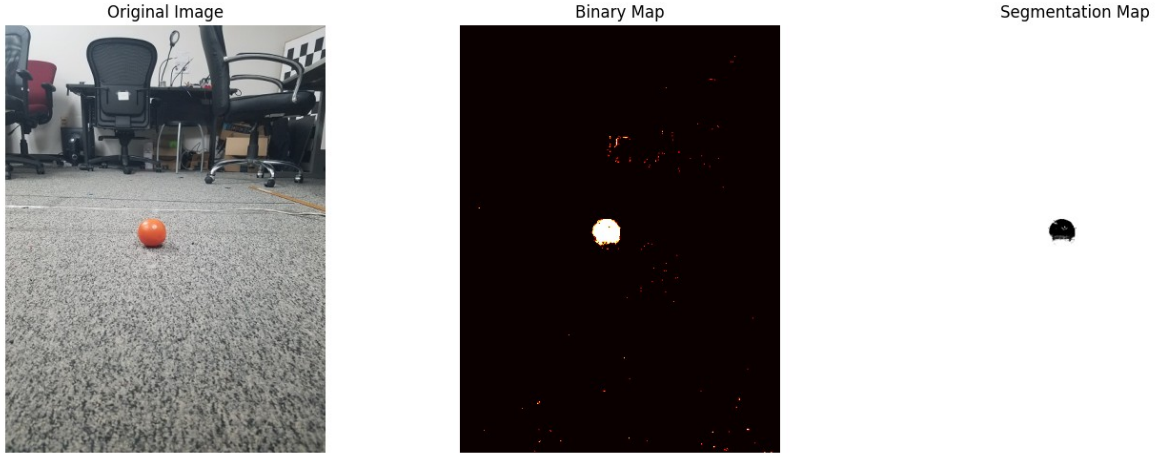
```
<ipython-input-9-8ca9b5013fbe>:67: RuntimeWarning:

invalid value encountered in divide
```



```
<ipython-input-9-8ca9b5013fbe>:67: RuntimeWarning:
invalid value encountered in divide
```



```
<ipython-input-9-8ca9b5013fbe>:67: RuntimeWarning:
invalid value encountered in divide
```



```
Segmentation complete. Results saved in the result folder.
[(231.5, 213.0, 8.381627082824707), (374.0616455078125,
366.8287658691406, 12.651599884033203), (226.09091186523438,
228.81817626953125, 10.09264087677002), (296.5, 343.5,
16.507673263549805), (333.5, 314.5, 7.106435298919678), (234.5, 204.5,
7.382511615753174), (228.9458465576172, 258.0466003417969,
15.253294944763184), (219.5, 310.0, 21.914705276489258)]
Radii: [ 8.38162708 12.65159988 10.09264088 16.50767326  7.1064353
7.38251162
 15.25329494 21.91470528]
Fitted parameters: a = 76.44761338071868 b = 2.4675116886799464
Estimated relative distances: [7.61260469 2.10031897 8.
3.4595558  1.         2.46706056
 4.93490293 6.4239263 ]
```

Video Lecture

Click here for a video lecture to help you better understand the project.

Report

For each section of the project, explain briefly what you did, and describe any interesting problems you encountered and/or solutions you implemented. You must include the following details in your writeup:

- Your choice of color space, initialization method and number of gaussians in the GMM
- Explain why GMM is better than single gaussian
- Present your distance estimate and cluster segmentation results for each test image
- Explain strengths and limitations of your algorithm. Also, explain why the algorithm failed on some test images

As usual, your report must be full English sentences, not commented code. There is a word limit of 1500 words and no minimum length requirement.

Your choice of color space, initialization method and number of gaussians in the GMM?

When initializing our Gaussian Mixture Model parameters for training, we had to choose values for each of the K Gaussians. For the means we used K-Means Clustering where K matched the K Gaussians so we could create K clusters to assign each observation. This would give us a more grounded means to start with which helps our algorithm converge faster and get more accurate results. Alongside this we initialize our covariances to match the covariance of the dataset with a small regularization parameter of 1e-3. The weights were arbitrarily assigned to give an equal initial probability of 1/k to all weights. We chose to have K=3 gaussians as we increased k we noticed our model matched to more white noise in the image. When plotting our gaussians against the observations in our data we noticed that the gaussians were able to map onto and consume more data points while still being generalizable at k=3.

Comparing RGB, HSV, and HSL filters and color spaces for the orange ball, it seemed that HSL gave the cleanest looking masks and had a pretty consistent colorspace. The results were also pretty good with both single gaussian and GMM. When using an RGB mask we were unable to properly isolate the orange of the ball when training and alot of outlier data points skewed our gaussian distributions. In the HSL color space we masked out images to isolate for the orange ball using thresholds for Hue, Saturation, and Lightness. The threshold we used was (0,100,120) to (30,240,250), this best masked out the noise in the image while preserving the orange color of the ball to be detected.

Explain why GMM is better than single gaussian

Through GMM we can assess a given data point's likelihood to be in each cluster(gaussian) rather than assigning it to one. With multiple gaussians we can map the observations of data against several models to find more patterns in the data itself. Using multiple gaussians models we can allow for variety in lighting or shading to be accounted for whereas a single gaussian may only be able to map to a certain shade of orange. Different parts of the ball will be colored differently due to lighting thus by using a single gaussian we are effectively averaging across all these color points to create the gaussian. This removes a lot of important information that would allow the model to generalize to new pixels as it does in GMM.

Present your distance estimate and cluster segmentation results for each test image

Segmentation and Distance Estimation are below.

Explain strengths and limitations of your algorithm. Also, explain why the algorithm failed on some test images

Some strengths of the algorithm we wrote include the following: We did not randomize our means. Originally, we did, but we found that it was extremely unreliable. Instead, we used KMeans Clustering, which made the segmentation process more accurate. The mask we used (HSV) was efficient and allowed for ball detection to be more accurate. It was able to filter out irrelevant information more accurately, making it easier for us to focus on the ball. We vectorized our operations to make the process even faster. This allowed us to run the colab notebook and generate images faster. One limitation is that if there are similar shades of orange in the picture, it might cause the algorithm to break down and not accurately match to the ball. The algorithm also failed on some test images because our mask wasn't exactly right. We were initially playing around with different masks and finally settled on HSV, which proved to yield the most accurate results we could generate. Also, we were using randomly generated means for a while, which caused the segmentation to be off half the time, which was not ideal.

For each section of the project, explain briefly what you did, and describe any interesting problems you encountered and/or solutions you implemented.

Single Gaussian: For single gaussian we just got the means and covariances of the data through numpy functions and then we use the pdf equation and bayes theorem for each test image to get the results. The results were then plotted.

trainGMM: For trainGMM we initially initialized the means randomly but we got frustrated with it so we set a seed for reproducibility then finally moved to using KMeans to initialize the means. Then for each iteration we did the expectation step and then the maximization step and continued it until the difference between previous mean and current mean was smaller than e or the max iteration was reached.

testGMM: For testGMM we found the posteriors then got the ball component then got the segmentation map. After that we compute the binary map and then the contours. We then plot it.

Distance estimation: For distance estimation we mapped the largest contour resulting from the binary mask after GMM. This contour was mapped to pixel coordinates for the centroid and the radius of the circle, or orange ball in our case. This radius was then used to calculate distance by calculating the area and fitting it to a / (x + b). Fitting to this model gets us the fitted parameters a, b which we can plug in with the radius of the ball to determine the relative distance. Our algorithm struggled with the distance estimation as the distances themselves were relative and not absolute. So across the 8 images we mapped placeholder distances of 1-8 then we fit the model with the area of the ball to these distances to get a model. Our results were lackluster as the area of the ball may change based on the angle it is placed as the lighting could be different causing different segmentation and thus a different area. Also the difference in distance between the pictures was relatively small in that the area change may not be as apparent. Our distance results are in line with what is expected as in the test images the balls were placed increasingly farther away. This is largely inline with our results as they are gradually increasing with the last 3 balls being the farthest away and thus the largest distance. Our model fitted to a = 69.9 and b=2.12 to map the distance as inversely related to the area.

Cluster segmentation: The cluster segmentation went well and the segmented plots correctly highlighted the orange ball in each image. However as the ball was placed farther away and in different lighting conditions we did notice that portions of the ball were not being detected however the center band where the orange shined was always being segmented. This is likely due to variation in our gaussians which may not fully encapsulate all the orange pixels that are part of the ball in different lighting and shadow conditions.

Interesting problems encountered:

Performance: Initially our performance was very bad due to unoptimized loops but we later used vectorized operations which enabled us to run the entire notebook in under a minute.

Image 1:

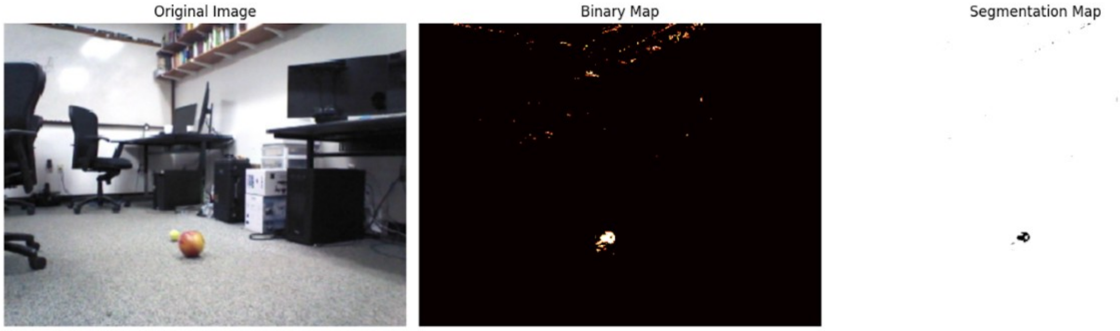


Image 2:

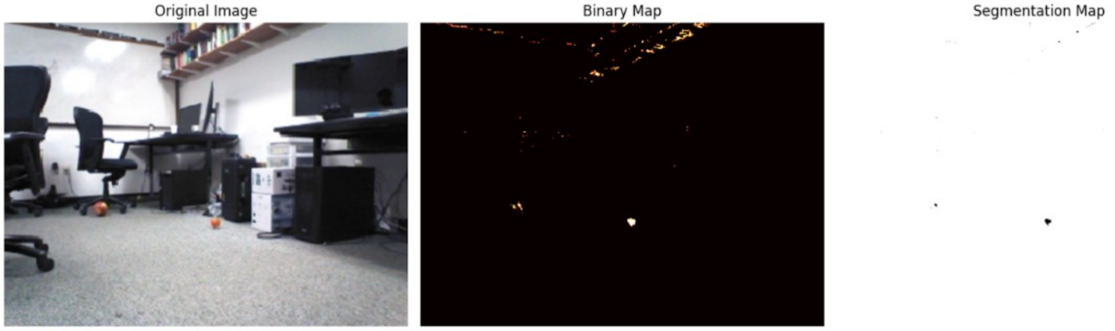


Image 3:

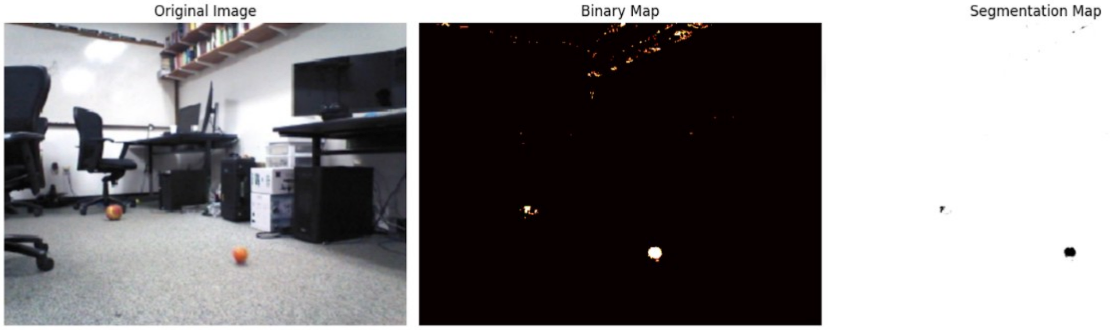


Image 4:

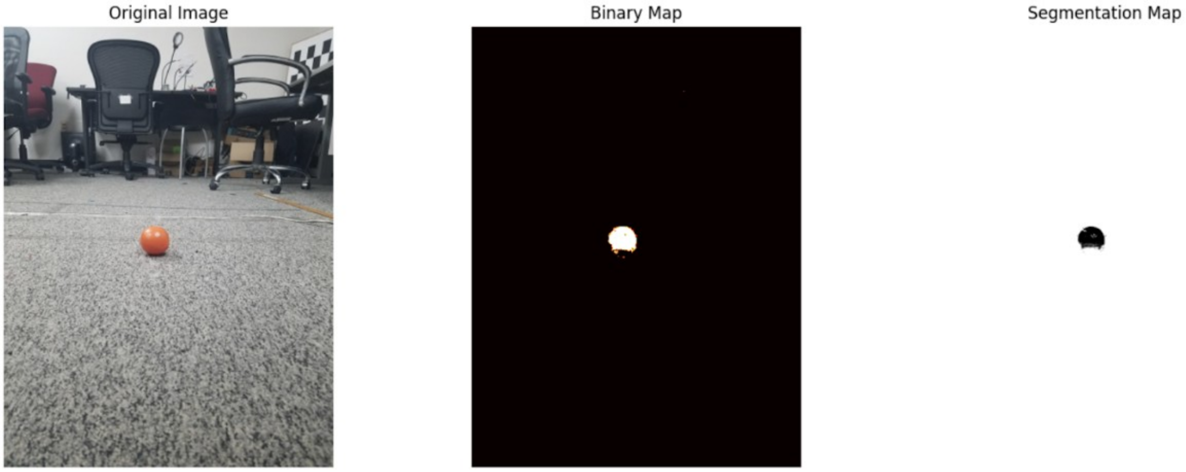


Image 5:

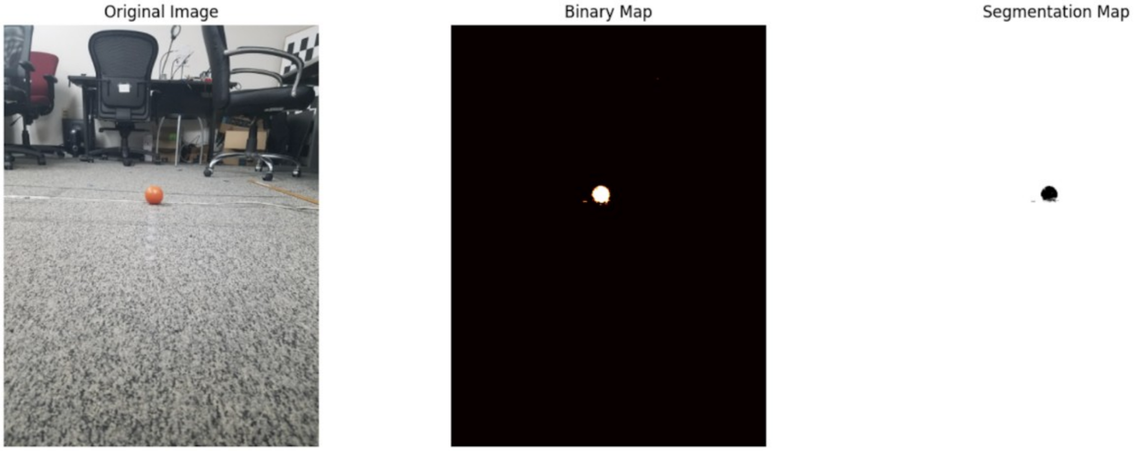


Image 6:

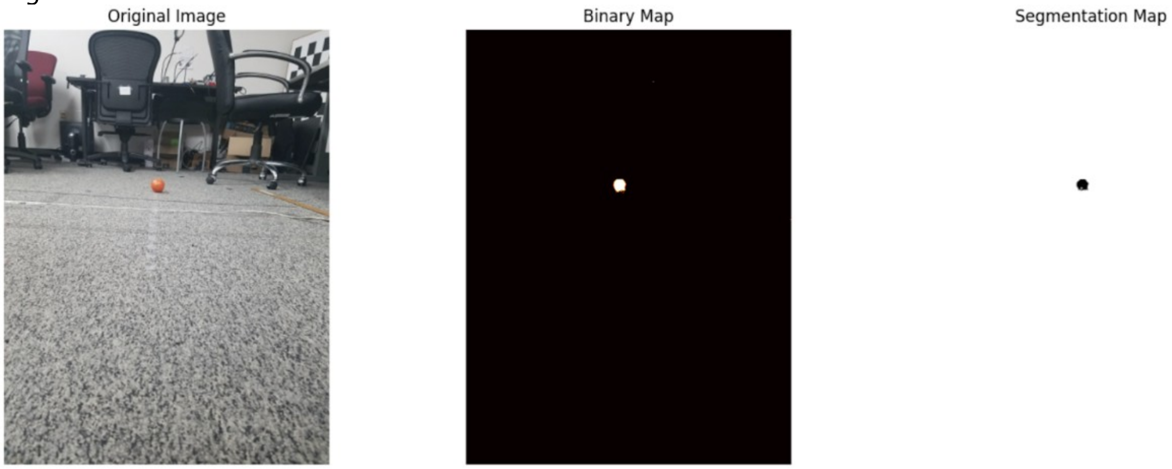


Image 7:

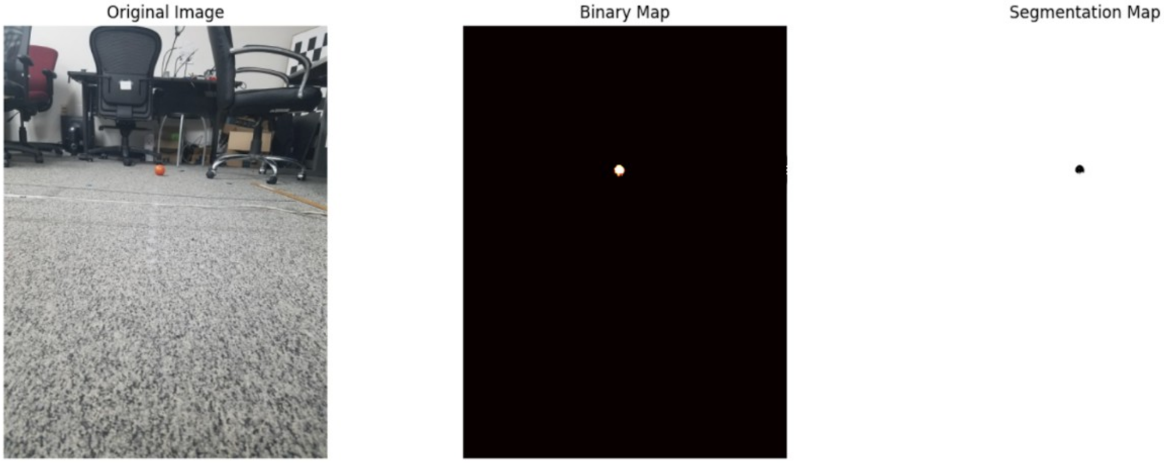
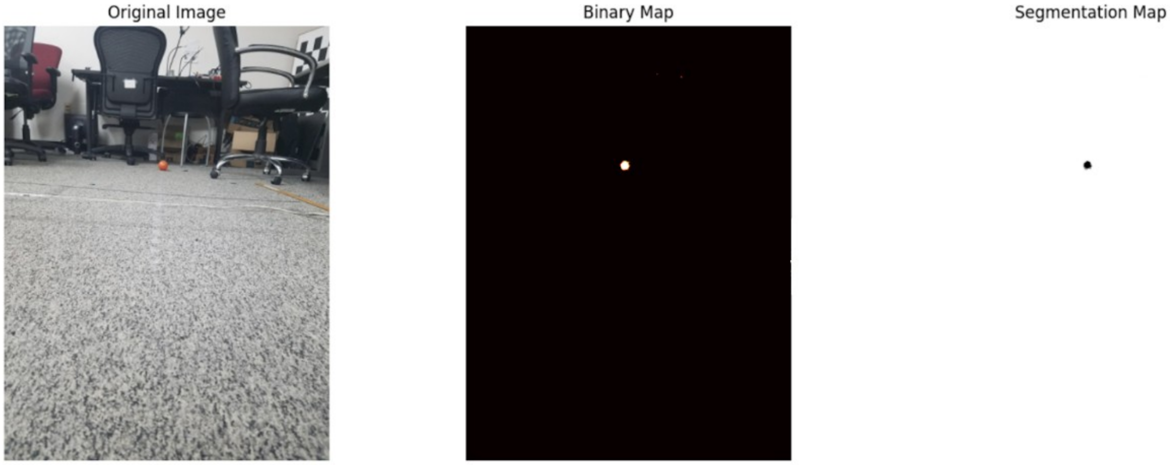
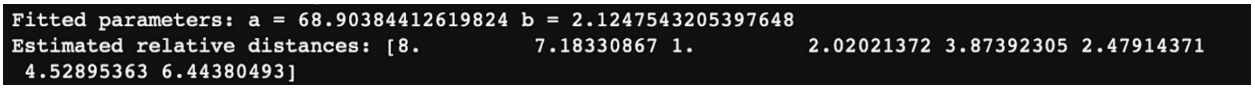


Image 8:



Distance Estimation:



```
Fitted parameters: a = 68.90384412619824 b = 2.1247543205397648
Estimated relative distances: [8.         7.18330867 1.         2.02021372 3.87392305 2.47914371
 4.52895363 6.44380493]
```

Submission Guidelines

If your submission does not comply with the following guidelines, you'll be given ZERO credit.

Your submission on ELMS(Canvas) must be a pdf file, following the naming convention **YourDirectoryID_proj1.pdf**. For example, xyz123_proj1.pdf.

All your results and report should be included in this notebook. After you finished all, please export the notebook as a pdf file and submit it to ELMS(Canvas).

Collaboration Policy

You are encouraged to discuss the ideas with your peers. However, the code should be your own, and should be the result of you exercising your own understanding of it. If you reference anyone else's code in writing your project, you must properly cite it in your code (in comments) and your writeup. For the full honor code refer to the CMSC426 Fall 2023 website.