In [1]:

```python
#imported libs
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import cv2
from tensorflow.keras.utils import get_file
import os
```

In [4]:

```python
#dataset_dir  = r"E:\APPS\PythonDataSets\caltech\caltech-101\101_ObjectCategories\1
dataset_dir = r"E:\APPS\PythonDataSets\caltech\caltech-101\101_ObjectCategories\fil




# Parameters
batch_size = 32
image_size = (64, 128)

# Load the training and validation datasets
train_dataset = tf.keras.preprocessing.image_dataset_from_directory(
    dataset_dir,
    validation_split=0.0,
    seed=123,
    image_size=image_size,
    batch_size=batch_size
)

val_dataset = tf.keras.preprocessing.image_dataset_from_directory(
    dataset_dir,
    validation_split=0.0,
    seed=123,
    image_size=image_size,
    batch_size=batch_size
)



def dataset_to_numpy(dataset):
    """
    Convert a tf.data.Dataset into NumPy arrays for features and labels.
    Args:
        dataset: A tf.data.Dataset object.
    Returns:
        X: Numpy array of features (images).
        y: Numpy array of labels.
    """
    X = []
    y = []
    for images, labels in dataset:
        X.append(images.numpy())
        y.append(labels.numpy())
    return np.concatenate(X, axis=0), np.concatenate(y, axis=0)

# Convert the train and validation datasets to NumPy arrays
X_train, y_train = dataset_to_numpy(train_dataset)
X_train=[cv2.resize(img.astype(np.uint8), (64, 128)) for img in X_train]
```

```
Found 9145 files belonging to 101 classes.
Found 9145 files belonging to 101 classes.
```

In [7]:

```python
#Accuracy
def accuracy(y_test1, y_pred1):
    y_pred1 = np.array(y_pred1)
    counter = 0
    for i in range(len(y_pred1)):
      if (y_pred1[i] == y_test1[i]):
        counter += 1
    accuracy = counter / len(y_pred1)
    accuracy *= 100
    return accuracy
```

In [8]:

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

# visualizing the results
def visualize_results(y_test,y_predict):
    class_names = [folder for folder in os.listdir(r"E:\APPS\PythonDataSets\caltech
    # Compute confusion matrix
    cm = confusion_matrix(y_test, y_predict)

    # Calculate accuracy for each class
    class_accuracies = (cm.diagonal() / cm.sum(axis=1)) * 100

    # Display class-wise accuracy
    #classes = [f"Class {i}" for i in range(len(class_accuracies))]  # Replace with

    # Plot the accuracies
    plt.figure(figsize=(15, 5))
    plt.bar(class_names, class_accuracies, color='skyblue')
    plt.xlabel("Classes")
    plt.ylabel("Accuracy (%)")
    plt.title("Class-Wise Accuracy")
    plt.xticks(rotation=90)
    plt.ylim(0, 100)  # Accuracy is in percentage
    plt.tight_layout()
    plt.show()
```

In [9]:

```python
#Color Histogran Extraction def
def extract_color_histogram(image, bins=(8, 8, 8)):
    """
    Extract a 3D color histogram from an RGB image.
    Args:
        image (numpy array): Input image in RGB format.
        bins (tuple): Number of bins for each channel (R, G, B).
    Returns:
        numpy array: Flattened color histogram feature vector.
    """
    # Calculate the 3D histogram for the HSV channels
    hist = cv2.calcHist([image], [0, 1, 2], None, bins, [0, 255, 0, 255, 0, 255])
    # Normalize the histogram to ensure invariance to lighting changes
    hist = cv2.normalize(hist, hist).flatten()
```

```python
    return hist
```

In [10]:
```python
#HOG def
def extract_hog_features(image):
       # HOG parameters
    winSize = (64, 128)
    blockSize = (16, 16)
    blockStride = (8, 8)
    cellSize = (8, 8)
    nbins = 9

    hog = cv2.HOGDescriptor(winSize, blockSize, blockStride, cellSize, nbins)
    hog_features = hog.compute(image)

    return hog_features
```

In [11]:
```python
#LBP def
from skimage.feature import local_binary_pattern

def extract_lbp_features(image, num_points=32, radius=8):
    gray_img = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    grid_size = (8, 8)  # Divide image into a 8x8 grid for histograms
     # Compute LBP
    lbp = local_binary_pattern(gray_img, num_points, radius, method="uniform")
    h, w = lbp.shape

    # Divide the image into grids and compute histograms
    grid_h, grid_w = h // grid_size[0], w // grid_size[1]
    histograms = []

    for i in range(grid_size[0]):
        for j in range(grid_size[1]):
            grid = lbp[i * grid_h:(i + 1) * grid_h, j * grid_w:(j + 1) * grid_w]
            hist, _ = np.histogram(grid, bins=np.arange(0, num_points + 3), density
            histograms.append(hist)

    # Concatenate histograms
    return np.concatenate(histograms)
```

In [18]:
```python
# Step 1: Extract LBP features
lbp_features_train = np.array([extract_lbp_features(image) for image in X_train])
```

In [12]:
```python
# Step 2: Extract HOG features
hog_features_train = np.array([extract_hog_features(image) for image in X_train])
```

In [13]:
```python
# Step 3: Extract Color Histogram features
clhg_features_train = np.array([extract_color_histogram(image) for image in X_train
```

In [31]:
```python
from sklearn.decomposition import PCA
pca = PCA(n_components=2)  # Reduce to 2 dimensions
clhg_reduced= pca.fit_transform(clhg_features_train)
hog_reduced= pca.fit_transform(hog_features_train)
lbp_reduced= pca.fit_transform(lbp_features_train)
```

```
In [14]: import numpy as np
from collections import Counter
from scipy.stats import mode
class KMeans:
    def __init__(self, n_clusters=101, max_iter=300, tol=1e-6):
        self.n_clusters = n_clusters
        self.max_iter = max_iter
        self.tol = tol  # Tolerance for convergence
        self.matching_list= np.zeros(101, dtype=int)

    def fit(self, X):
        # Step 1: Randomly initialize centroids
        np.random.seed(42)
        self.centroids = X[np.random.choice(X.shape[0], self.n_clusters, replace=Fa
        print(X.shape[0])
        print (self.centroids.shape)
        for iteration in range(self.max_iter):
            # Step 2: Assign points to the nearest cluster ,holds the cluster no. o
            self.labels = self._assign_clusters(X)

            # Step 3: Compute new centroids
            new_centroids = np.array([  X[self.labels == i].mean(axis=0) for i in r

            # Check for convergence
            if np.all(np.abs(new_centroids - self.centroids) < self.tol):
                print(f"Converged at iteration {iteration}")
                #after coverging , get the matching class for the clusters
                self._match_cluster(X)# will update the matching list
                break

            self.centroids = new_centroids

    def _assign_clusters(self, X):
        #My implementation , faster implementation than chatgpt implementation
        distances=[]
        for i in range (len(X) ):
            distances.append(     np.linalg.norm(self.centroids - X[i] , axis=1)
        return np.argmin(  np.array(distances) , axis = 1   )


    def _predictt(self,X):#mockup predict function to be used by the _match_cluster
        X = np.array(X).flatten()
        # Compute the Euclidean distances between the test case and all centroids
        distances = np.linalg.norm(self.centroids - X, axis=1)    # (101 x no. of fe
        return np.argmin(distances)

    #My implementation for matching each cluster number with a real class number
    def _match_cluster(self,X):
        for i in range (self.n_clusters):
            cluster_indices = np.where(y_train==i)[0]
            y_pred = np.array([   self._predictt ( X[x].reshape(1, -1))  for x in c
            self.matching_list[i]= Counter(y_pred).most_common(1)[0][0]


    def predict(self, X): #real predict function that will be used by the user
```

```
        X = np.array(X).flatten()
        # Compute the Euclidean distances between the test case and all centroids
        distances = np.linalg.norm(self.centroids - X, axis=1)   # (101 x no. of fe
        #return self.matching_list[   np.argmin(distances)  ] #smallest distance r
        return np.where(self.matching_list == np.argmin(distances) )[0]

    def clustering_accuracy(self,y_labels):
        cluster_labels = np.zeros(self.n_clusters)  # Array to hold the true label

        for cluster in range(self.n_clusters): # 0 ---- 101
            # Get the indices of all points assigned to this cluster
            cluster_points = y_labels[self.labels == cluster]

            # Assign the most common true label in this cluster
            if len(cluster_points) > 0:
                most_common_label = mode(cluster_points)[0]
                cluster_labels[cluster] = most_common_label

        # Step 3: Predict labels for each test case
        predicted_labels = np.array([cluster_labels[cluster] for cluster in self.la

        # Step 4: Calculate accuracy
        accuracy = np.sum(predicted_labels == y_labels) / len(y_labels) * 100
        return accuracy
```

In [52]:

```
modelhogreduced=KMeans()
modelhogreduced.fit(hog_reduced)

modellbpreduced=KMeans()
modellbpreduced.fit(lbp_reduced)

modelclhgreduced=KMeans()
modelclhgreduced.fit(clhg_reduced)
```

```
9145
(101, 2)
Converged at iteration 83
9145
(101, 2)
Converged at iteration 119
9145
(101, 2)
Converged at iteration 51
```

In [19]:

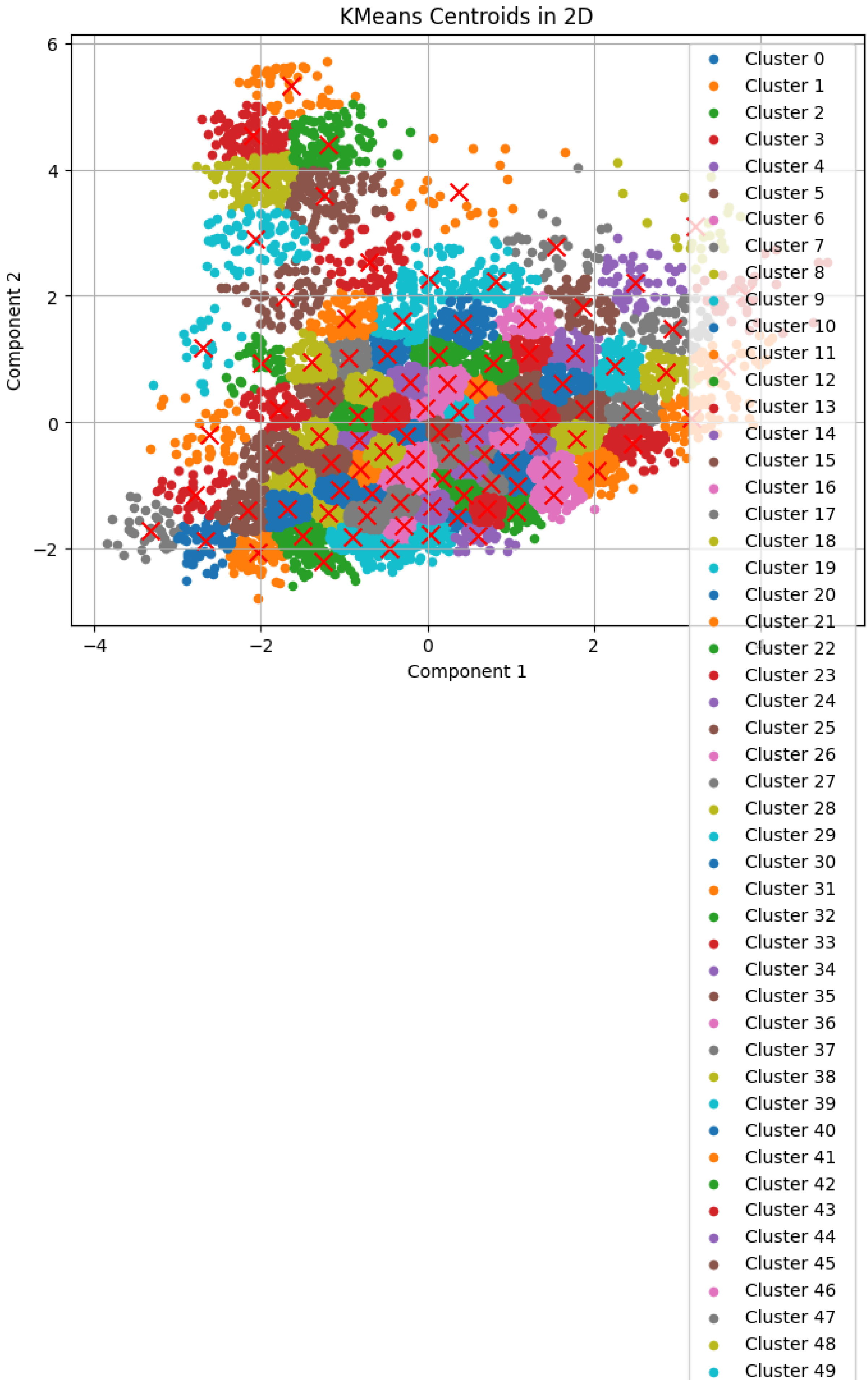
```
model = KMeans()
model.fit(lbp_features_train)
model.clustering_accuracy(y_train)
```

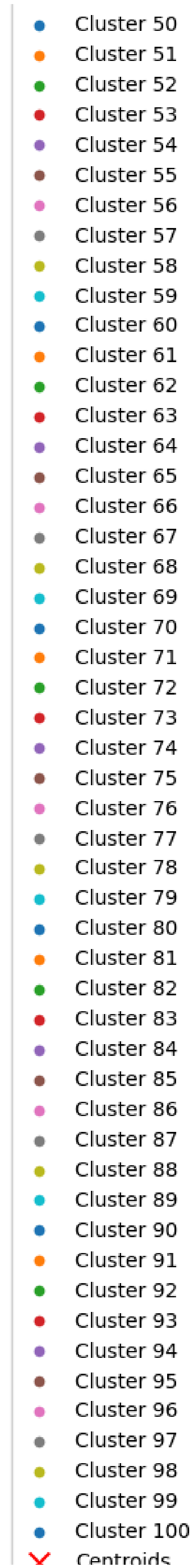
```
9145
(101, 2176)
Converged at iteration 34
```

Out[19]: 32.695462001093496

In [49]:
```
from sklearn.decomposition import PCA


def plot_centroids(model,dataset):
    print(model.labels[0])
    pca = PCA(n_components=2)  # Reduce to 2 dimensions
    centroidss = model.centroids#pca.fit_transform(model.centroids)
    reduced_dataset = dataset#pca.fit_transform(dataset)
    plt.figure(figsize=(8, 6))



    for i in range (101):
        index = np.where(model.labels == i)[0]
        values = reduced_dataset[index]
        plt.scatter(values[:, 1], values[:, 0], label=f'Cluster {i}', s=20)


    plt.scatter(centroidss[:101, 1], centroidss[:101, 0], c='red', marker='x', s=10
    plt.title('KMeans Centroids in 2D')
    plt.xlabel('Component 1')
    plt.ylabel('Component 2')
    plt.legend()
    plt.grid()
    plt.show()

plot_centroids(modelhogreduced , hog_reduced)
```

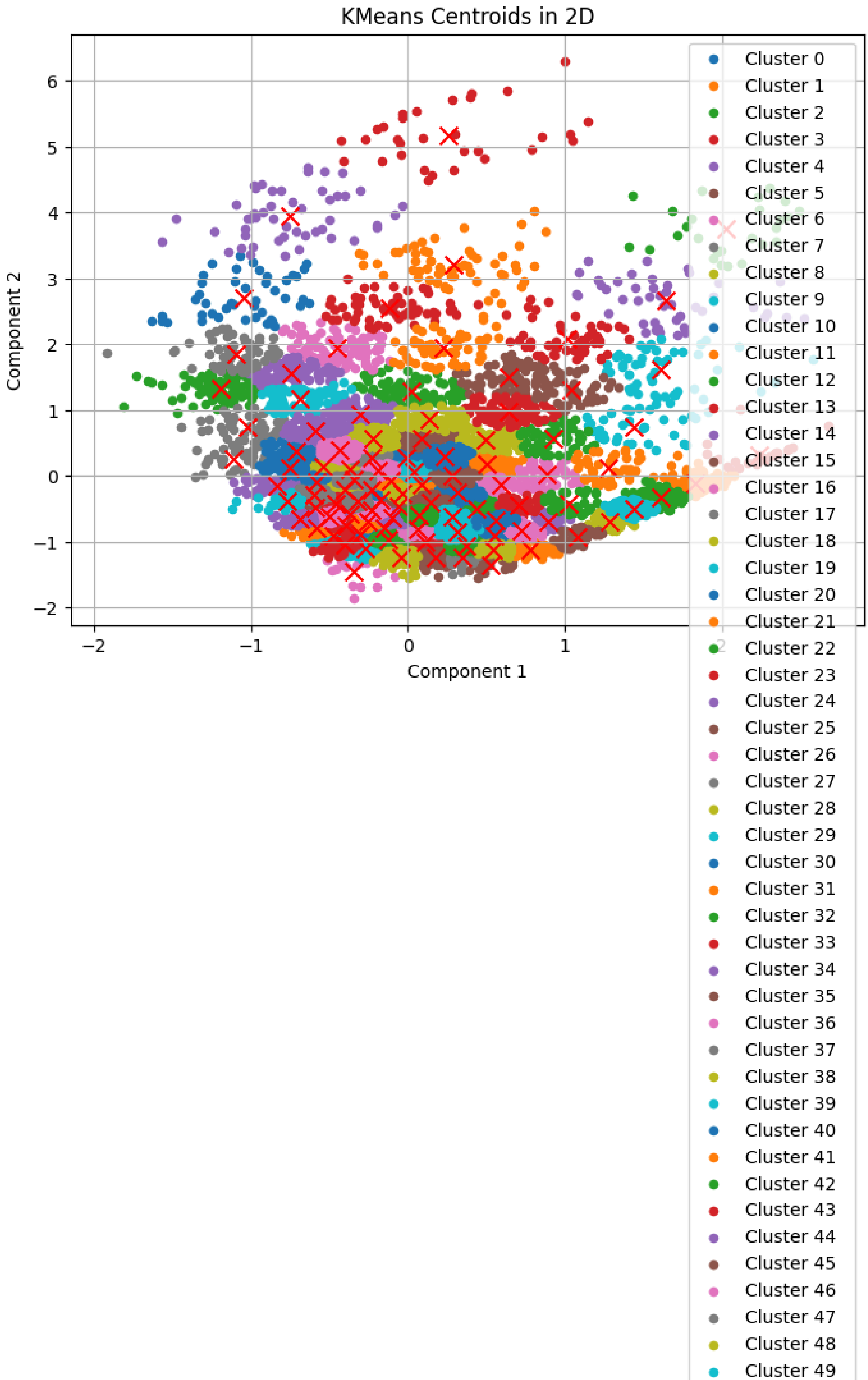
79


KMeans Centroids in 2D
Component 1
Component 2
−4
−2
0
2
6
4
2
0
−2
Cluster 0
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster 5
Cluster 6
Cluster 7
Cluster 8
Cluster 9
Cluster 10
Cluster 11
Cluster 12
Cluster 13
Cluster 14
Cluster 15
Cluster 16
Cluster 17
Cluster 18
Cluster 19
Cluster 20
Cluster 21
Cluster 22
Cluster 23
Cluster 24
Cluster 25
Cluster 26
Cluster 27
Cluster 28
Cluster 29
Cluster 30
Cluster 31
Cluster 32
Cluster 33
Cluster 34
Cluster 35
Cluster 36
Cluster 37
Cluster 38
Cluster 39
Cluster 40
Cluster 41
Cluster 42
Cluster 43
Cluster 44
Cluster 45
Cluster 46
Cluster 47
Cluster 48
Cluster 49

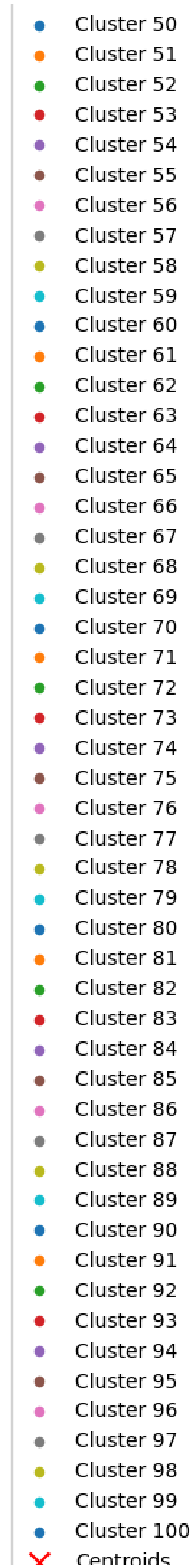

Cluster 50
Cluster 51
Cluster 52
Cluster 53
Cluster 54
Cluster 55
Cluster 56
Cluster 57
Cluster 58
Cluster 59
Cluster 60
Cluster 61
Cluster 62
Cluster 63
Cluster 64
Cluster 65
Cluster 66
Cluster 67
Cluster 68
Cluster 69
Cluster 70
Cluster 71
Cluster 72
Cluster 73
Cluster 74
Cluster 75
Cluster 76
Cluster 77
Cluster 78
Cluster 79
Cluster 80
Cluster 81
Cluster 82
Cluster 83
Cluster 84
Cluster 85
Cluster 86
Cluster 87
Cluster 88
Cluster 89
Cluster 90
Cluster 91
Cluster 92
Cluster 93
Cluster 94
Cluster 95
Cluster 96
Cluster 97
Cluster 98
Cluster 99
Cluster 100
Centroids

In [53]:
```
plot_centroids(modellbpreduced , lbp_reduced)
```

22



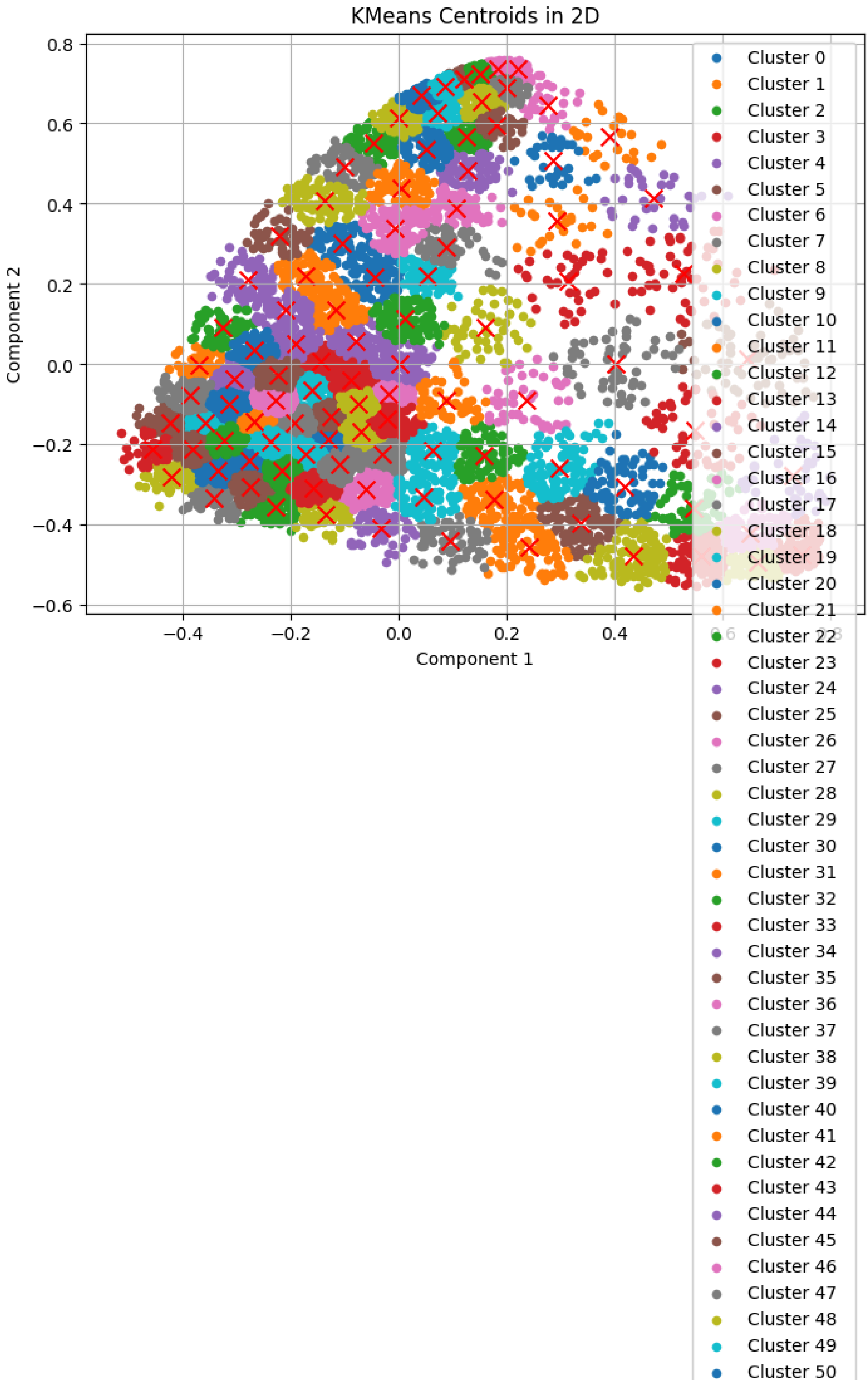
KMeans Centroids in 2D
Component 1
Component 2
Cluster 0
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster 5
Cluster 6
Cluster 7
Cluster 8
Cluster 9
Cluster 10
Cluster 11
Cluster 12
Cluster 13
Cluster 14
Cluster 15
Cluster 16
Cluster 17
Cluster 18
Cluster 19
Cluster 20
Cluster 21
Cluster 22
Cluster 23
Cluster 24
Cluster 25
Cluster 26
Cluster 27
Cluster 28
Cluster 29
Cluster 30
Cluster 31
Cluster 32
Cluster 33
Cluster 34
Cluster 35
Cluster 36
Cluster 37
Cluster 38
Cluster 39
Cluster 40
Cluster 41
Cluster 42
Cluster 43
Cluster 44
Cluster 45
Cluster 46
Cluster 47
Cluster 48
Cluster 49


Cluster 50
Cluster 51
Cluster 52
Cluster 53
Cluster 54
Cluster 55
Cluster 56
Cluster 57
Cluster 58
Cluster 59
Cluster 60
Cluster 61
Cluster 62
Cluster 63
Cluster 64
Cluster 65
Cluster 66
Cluster 67
Cluster 68
Cluster 69
Cluster 70
Cluster 71
Cluster 72
Cluster 73
Cluster 74
Cluster 75
Cluster 76
Cluster 77
Cluster 78
Cluster 79
Cluster 80
Cluster 81
Cluster 82
Cluster 83
Cluster 84
Cluster 85
Cluster 86
Cluster 87
Cluster 88
Cluster 89
Cluster 90
Cluster 91
Cluster 92
Cluster 93
Cluster 94
Cluster 95
Cluster 96
Cluster 97
Cluster 98
Cluster 99
Cluster 100
Centroids

In [54]:

```
plot_centroids(modelclhgreduced , clhg_reduced)
```

67



KMeans Centroids in 2D
Component 1
Component 2
Cluster 0
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster 5
Cluster 6
Cluster 7
Cluster 8
Cluster 9
Cluster 10
Cluster 11
Cluster 12
Cluster 13
Cluster 14
Cluster 15
Cluster 16
Cluster 17
Cluster 18
Cluster 19
Cluster 20
Cluster 21
Cluster 22
Cluster 23
Cluster 24
Cluster 25
Cluster 26
Cluster 27
Cluster 28
Cluster 29
Cluster 30
Cluster 31
Cluster 32
Cluster 33
Cluster 34
Cluster 35
Cluster 36
Cluster 37
Cluster 38
Cluster 39
Cluster 40
Cluster 41
Cluster 42
Cluster 43
Cluster 44
Cluster 45
Cluster 46
Cluster 47
Cluster 48
Cluster 49
Cluster 50


Cluster 51
Cluster 52
Cluster 53
Cluster 54
Cluster 55
Cluster 56
Cluster 57
Cluster 58
Cluster 59
Cluster 60
Cluster 61
Cluster 62
Cluster 63
Cluster 64
Cluster 65
Cluster 66
Cluster 67
Cluster 68
Cluster 69
Cluster 70
Cluster 71
Cluster 72
Cluster 73
Cluster 74
Cluster 75
Cluster 76
Cluster 77
Cluster 78
Cluster 79
Cluster 80
Cluster 81
Cluster 82
Cluster 83
Cluster 84
Cluster 85
Cluster 86
Cluster 87
Cluster 88
Cluster 89
Cluster 90
Cluster 91
Cluster 92
Cluster 93
Cluster 94
Cluster 95
Cluster 96
Cluster 97
Cluster 98
Cluster 99
Cluster 100
Centroids

In [16]:

```
model2 = KMeans()
model2.fit(hog_features_train)
model2.clustering_accuracy(y_train)
```

```
9145
(101, 3780)
Converged at iteration 58
```

Out[16]:

```
41.46528157463095
```

In [17]:

```
model3 = KMeans()
model3.fit(clhg_features_train)
model3.clustering_accuracy(y_train)
```

```
9145
(101, 512)
Converged at iteration 56
```

Out[17]:

```
24.05686167304538
```

In [ ]:

```
"""My implementation for matching each cluster number with a real class number
from collections import Counter
matching_list= np.zeros(101, dtype=int)

cluster_indices = np.where(y_train==4)[0]
y_pred = np.array([   model.predict(lbp_features_train[x].reshape(1, -1)) for x in
matching_list[4]= Counter(y_pred).most_common(1)[0][0]

print(matching_list[4])"""

"""matching_list=[]
for i in range (n_cluster):
    cluster_indices = np.where(y_train==i)[0]
    y_pred = np.array([   model.predict(x_train[x].reshape(1, -1)) for x in cluster
    matching_list[i]= Counter(y_pred).most_common(1)[0][0]
    """
```

```
29
```

Out[ ]:

```
'matching_list=[]\nfor i in range (n_cluster):\n    cluster_indices = np.where(y_t
rain==i)[0]\n    y_pred = np.array([   model.predict(x_train[x].reshape(1, -1)) fo
r x in cluster_indices   ])\n    matching_list[i]= Counter(y_pred).most_common(1)
[0][0]\n    '
```