In [14]:

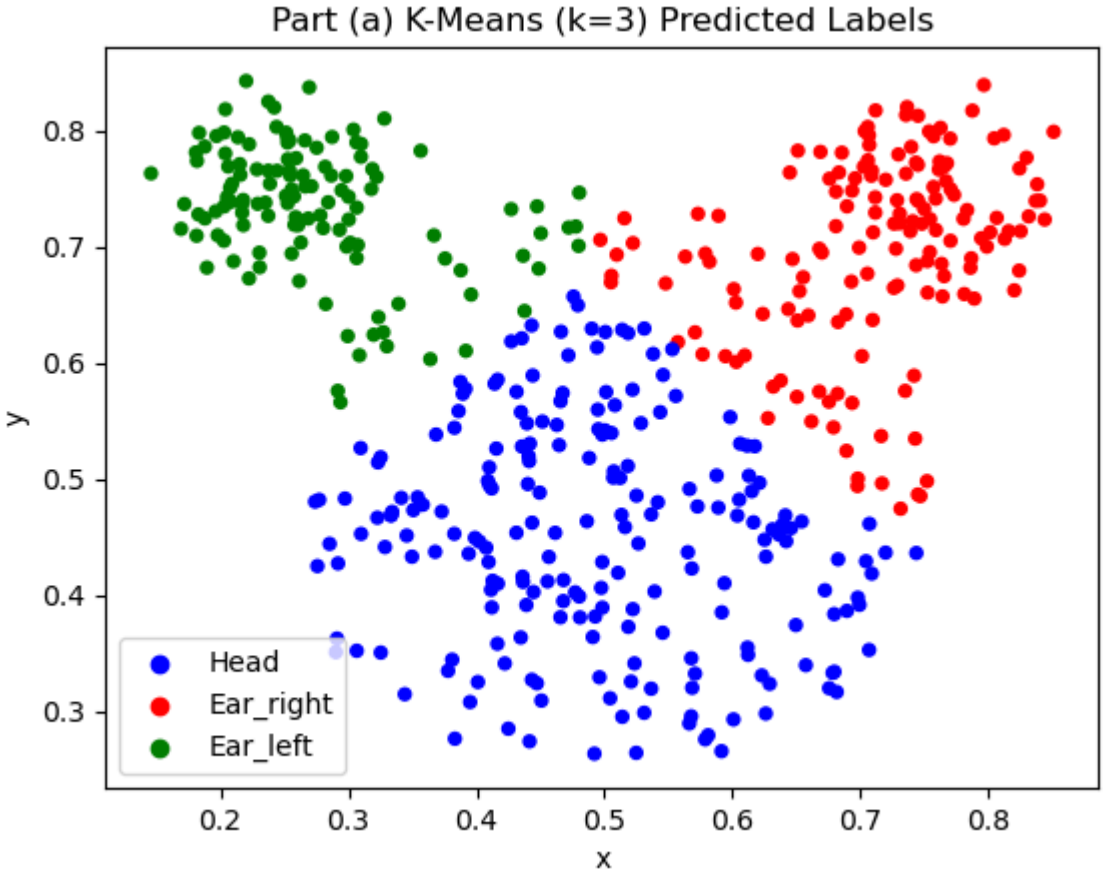
```python
import re
import numpy as np
import matplotlib.pyplot as plt

LINE_RE = re.compile(r'^\s*(-?\d+(?:\.\d+)?)\s+(-?\d+(?:\.\d+)?)\s+(\S+)\s*$')

def load_cluster_txt(path="cluster.txt"):
    xs, ys, labs = [], [], []
    with open(path, "r", encoding="utf-8") as f:
        for line in f:
            m = LINE_RE.match(line)
            if not m:
                continue
            xs.append(float(m.group(1)))
            ys.append(float(m.group(2)))
            labs.append(m.group(3))
    X = np.column_stack([xs, ys]).astype(float)
    y = np.array(labs)
    return X, y

def map_clusters_to_labels(cluster_ids, y_true):
    mapping = {}
    for cid in np.unique(cluster_ids):
        idx = np.where(cluster_ids == cid)[0]
        vals, cnts = np.unique(y_true[idx], return_counts=True)
        mapping[int(cid)] = vals[np.argmax(cnts)]
    y_pred = np.array([mapping[int(c)] for c in cluster_ids])
    return y_pred, mapping

def confusion_matrix(y_true, y_pred, classes):
    idx = {c:i for i,c in enumerate(classes)}
    cm = np.zeros((len(classes), len(classes)), dtype=int)
    for t, p in zip(y_true, y_pred):
        if t in idx and p in idx:
            cm[idx[t], idx[p]] += 1
    return cm

def print_cm_and_acc(cm, classes, title):
    print("\n" + title)
    print("Confusion Matrix (rows=true, cols=pred) order:", classes)
    print(cm)
    per = [cm[i,i] / cm[i].sum() if cm[i].sum() else 0.0 for i in range(len(clas
    overall = np.trace(cm) / cm.sum() if cm.sum() else 0.0
    print("Per-class accuracy:")
    for c, a in zip(classes, per):
        print(f"  {c}: {a:.4f}")
    print(f"Overall accuracy: {overall:.4f}")

def plot_scatter(X, labels, title, color_map):
    colors = [color_map.get(lbl, "black") for lbl in labels]
    plt.figure()
    plt.scatter(X[:,0], X[:,1], c=colors, s=18)
    for name, col in color_map.items():
        plt.scatter([], [], c=col, label=name)
    plt.title(title)
    plt.xlabel("x"); plt.ylabel("y")
    plt.legend()
    plt.show()
```

```
def kmeans(X, k=3, max_iter=300, seed=0):
    rng = np.random.default_rng(seed)
    N = X.shape[0]
    C = X[rng.choice(N, size=k, replace=False)]
    a = np.full(N, -1, dtype=int)

    for _ in range(max_iter):
        d = ((X[:, None, :] - C[None, :, :]) ** 2).sum(axis=2)
        new_a = np.argmin(d, axis=1)
        if np.array_equal(new_a, a):
            break
        a = new_a
        for j in range(k):
            idx = np.where(a == j)[0]
            C[j] = X[idx].mean(axis=0) if len(idx) else X[rng.integers(0, N)]
    return a


def logsumexp(A, axis=1):
    m = np.max(A, axis=axis, keepdims=True)
    return (m + np.log(np.sum(np.exp(A - m), axis=axis, keepdims=True))).squeeze

def log_gaussian_pdf(X, mu, Sigma):
    N, D = X.shape
    Sigma = Sigma + 1e-6 * np.eye(D)
    sign, logdet = np.linalg.slogdet(Sigma)
    if sign <= 0:
        Sigma = Sigma + 1e-3 * np.eye(D)
        sign, logdet = np.linalg.slogdet(Sigma)
    inv = np.linalg.inv(Sigma)
    diff = X - mu
    quad = np.einsum("ni,ij,nj->n", diff, inv, diff)
    return -0.5 * (D * np.log(2*np.pi) + logdet + quad)

def init_from_onehot(X, gamma0):
    N, D = X.shape
    K = gamma0.shape[1]
    Nk = gamma0.sum(axis=0) + 1e-12
    w = Nk / N
    mu = (gamma0.T @ X) / Nk[:, None]
    Sigma = np.zeros((K, D, D))
    for k in range(K):
        diff = X - mu[k]
        Sigma[k] = (gamma0[:, k][:, None] * diff).T @ diff / Nk[k]
        Sigma[k] += 1e-6 * np.eye(D)
    return w, mu, Sigma

def e_step(X, w, mu, Sigma):
    N, D = X.shape
    K = w.shape[0]
    log_prob = np.zeros((N, K))
    for k in range(K):
        log_prob[:, k] = np.log(w[k] + 1e-12) + log_gaussian_pdf(X, mu[k], Sigma
    log_den = logsumexp(log_prob, axis=1)
    gamma = np.exp(log_prob - log_den[:, None])
    nll = -np.sum(log_den)
    return gamma, nll
```

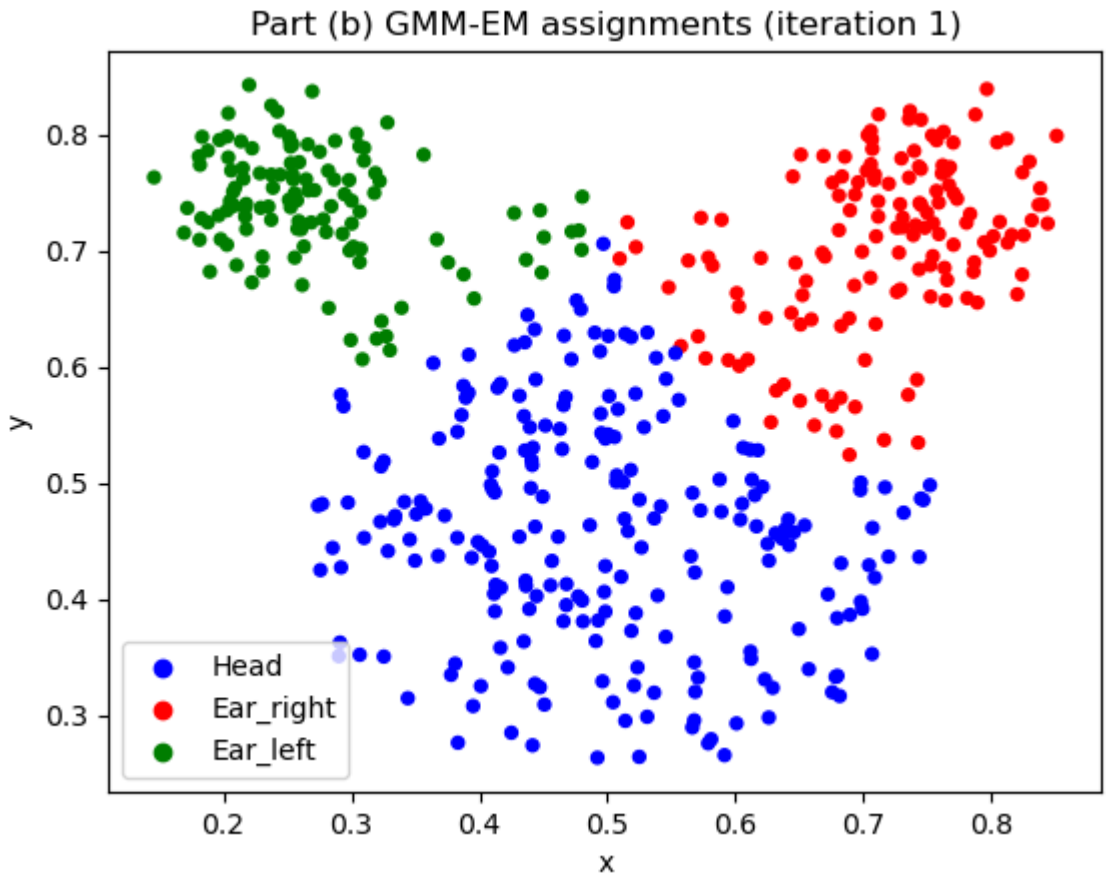
```python
def m_step(X, gamma):
    N, D = X.shape
    Nk = gamma.sum(axis=0) + 1e-12
    K = gamma.shape[1]
    w = Nk / N
    mu = (gamma.T @ X) / Nk[:, None]
    Sigma = np.zeros((K, D, D))
    for k in range(K):
        diff = X - mu[k]
        Sigma[k] = (gamma[:, k][:, None] * diff).T @ diff / Nk[k]
        Sigma[k] += 1e-6 * np.eye(D)
    return w, mu, Sigma

def em_gmm(X, init_labels, k=3, max_iter=200, tol=1e-4):
    N = X.shape[0]
    gamma0 = np.zeros((N, k))
    gamma0[np.arange(N), init_labels] = 1.0

    w, mu, Sigma = init_from_onehot(X, gamma0)

    nll_hist = []
    hard_hist = []
    prev = None

    for _ in range(max_iter):
        gamma, nll = e_step(X, w, mu, Sigma)
        w, mu, Sigma = m_step(X, gamma)

        nll_hist.append(nll)
        hard_hist.append(np.argmax(gamma, axis=1))

        if prev is not None and abs(prev - nll) < tol:
            break
        prev = nll

    return gamma, nll_hist, hard_hist

def main():
    X, y_true = load_cluster_txt("cluster.txt")


    classes = ["Head", "Ear_right", "Ear_left"]

    color_map = {"Head": "blue", "Ear_right": "red", "Ear_left": "green"}


    km_ids = kmeans(X, k=3, seed=0)
    km_pred, km_map = map_clusters_to_labels(km_ids, y_true)
    print("Cluster -> Class mapping:", km_map)
    plot_scatter(X, km_pred, "Part (a) K-Means (k=3) Predicted Labels", color_ma
    cm_a = confusion_matrix(y_true, km_pred, classes)
    print_cm_and_acc(cm_a, classes, "Part (a) results:")



    gamma, nll_hist, hard_hist = em_gmm(X, init_labels=km_ids, k=3, tol=1e-4)

    print("EM iterations:", len(nll_hist))
```

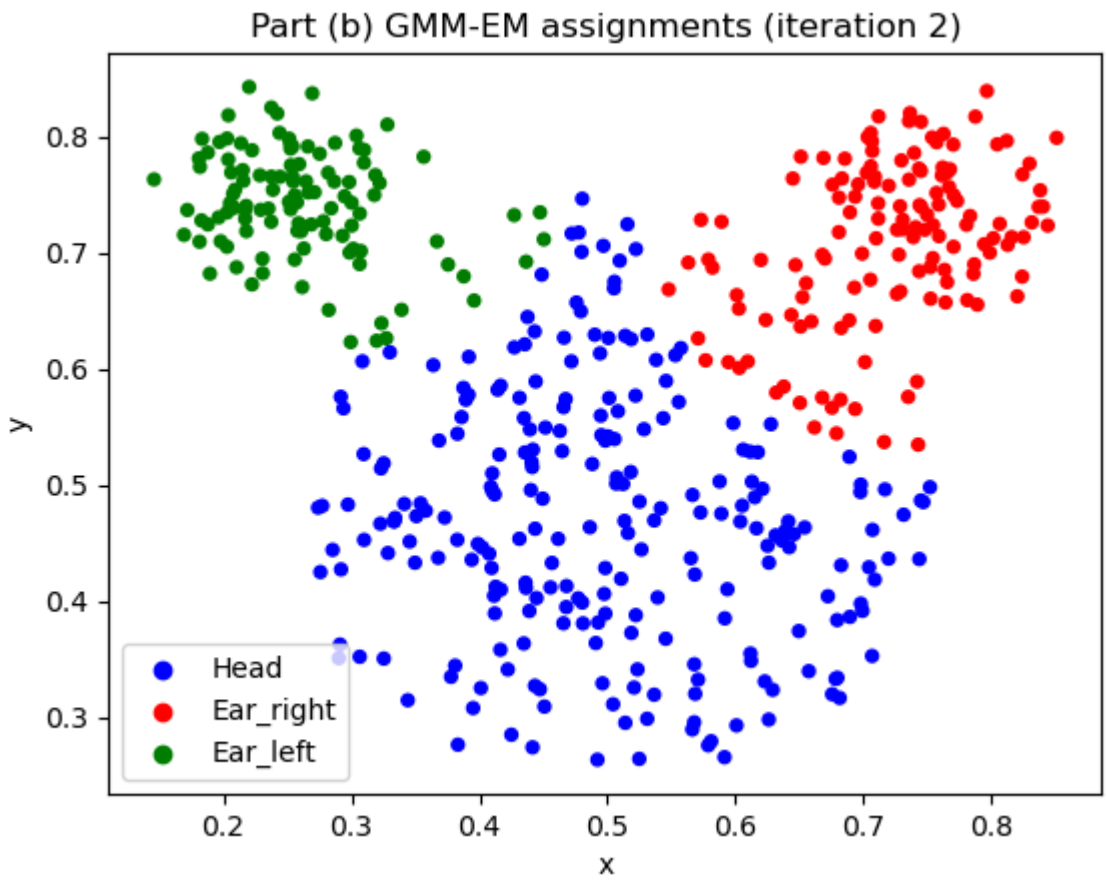
```
    for i in range(min(4, len(hard_hist))):
        pred_i, _ = map_clusters_to_labels(hard_hist[i], y_true)
        plot_scatter(X, pred_i, f"Part (b) GMM-EM assignments (iteration {i+1})"


    final_ids = np.argmax(gamma, axis=1)
    gmm_pred, gmm_map = map_clusters_to_labels(final_ids, y_true)
    print("Cluster -> Class mapping:", gmm_map)
    plot_scatter(X, gmm_pred, "Part (b) GMM-EM Final Predicted Labels (k=3)", co
    cm_b = confusion_matrix(y_true, gmm_pred, classes)
    print_cm_and_acc(cm_b, classes, "Part (b) results:")



if __name__ == "__main__":
    main()
```

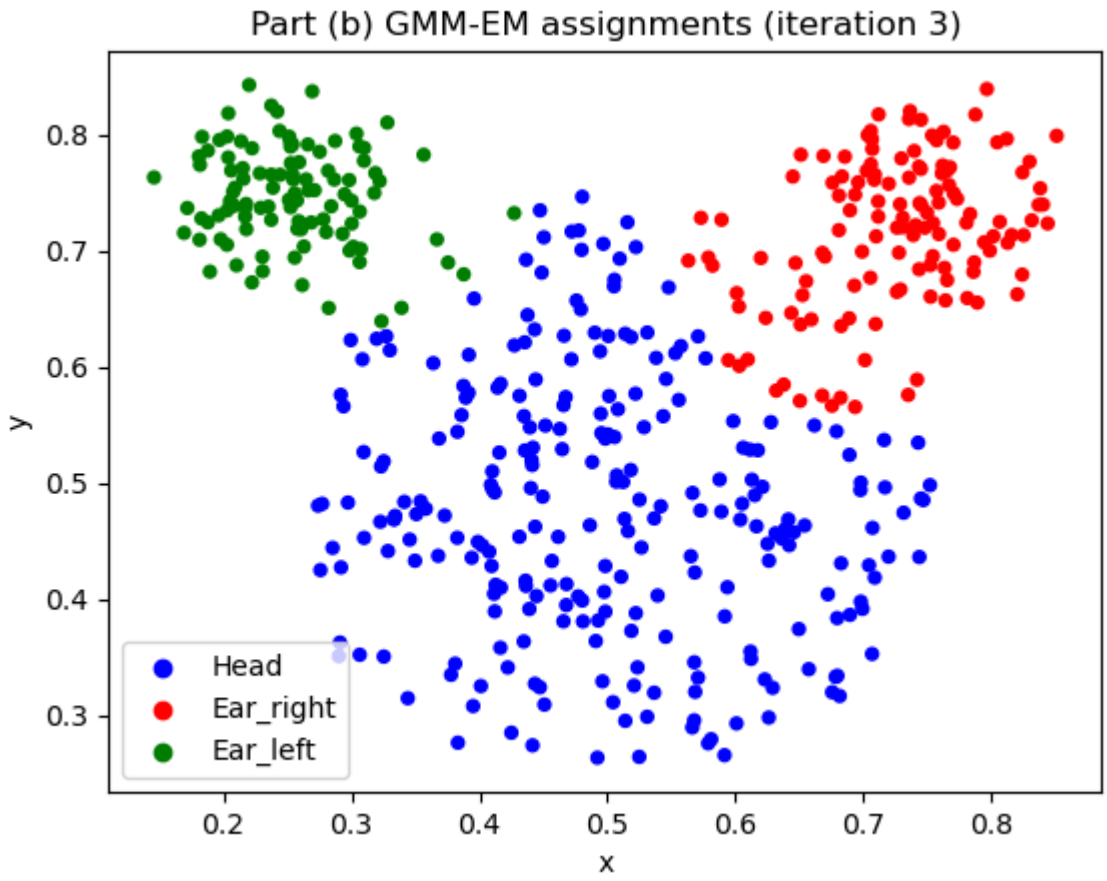
```
Cluster -> Class mapping: {0: np.str_('Ear_left'), 1: np.str_('Head'), 2: np.str_
('Ear_right')}
```

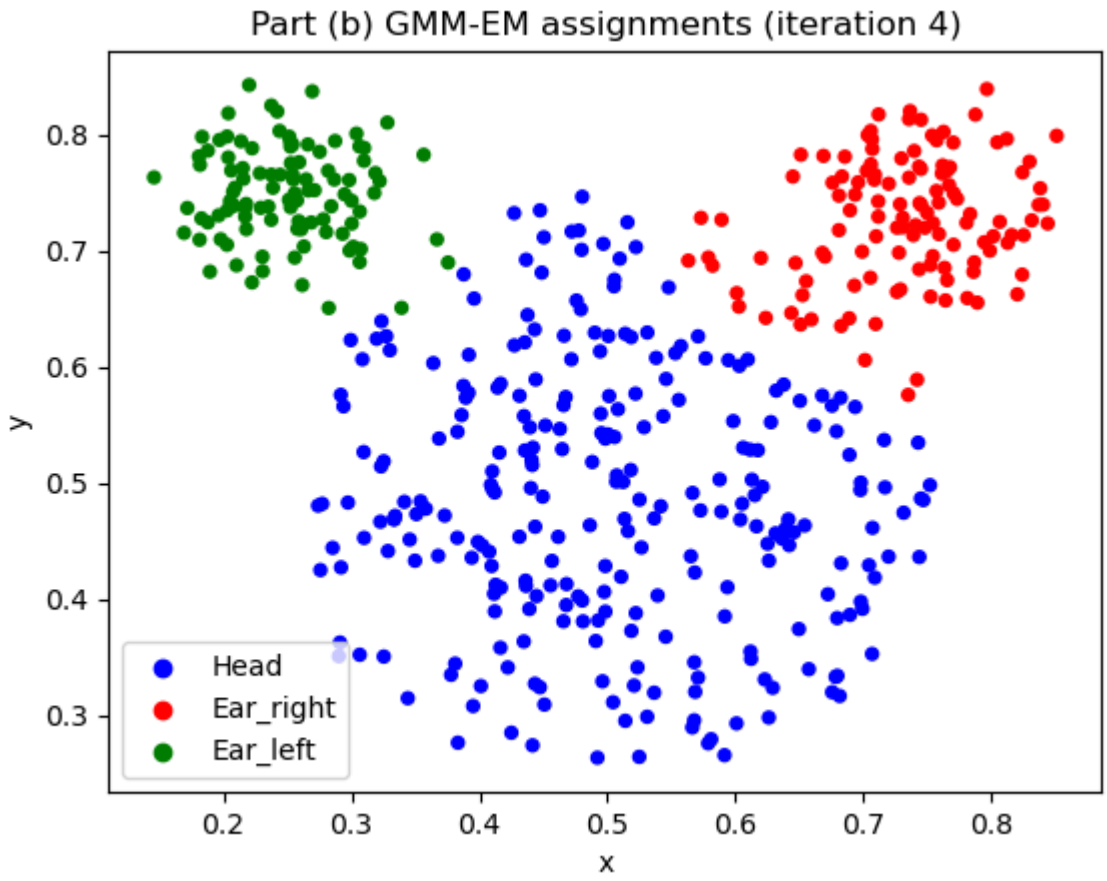



```
Part (a) results:
Confusion Matrix (rows=true, cols=pred) order: ['Head', 'Ear_right', 'Ear_left']
[[211  54  25]
 [  0 100   0]
 [  0   0 100]]
Per-class accuracy:
  Head: 0.7276
  Ear_right: 1.0000
  Ear_left: 1.0000
Overall accuracy: 0.8388
EM iterations: 24
```

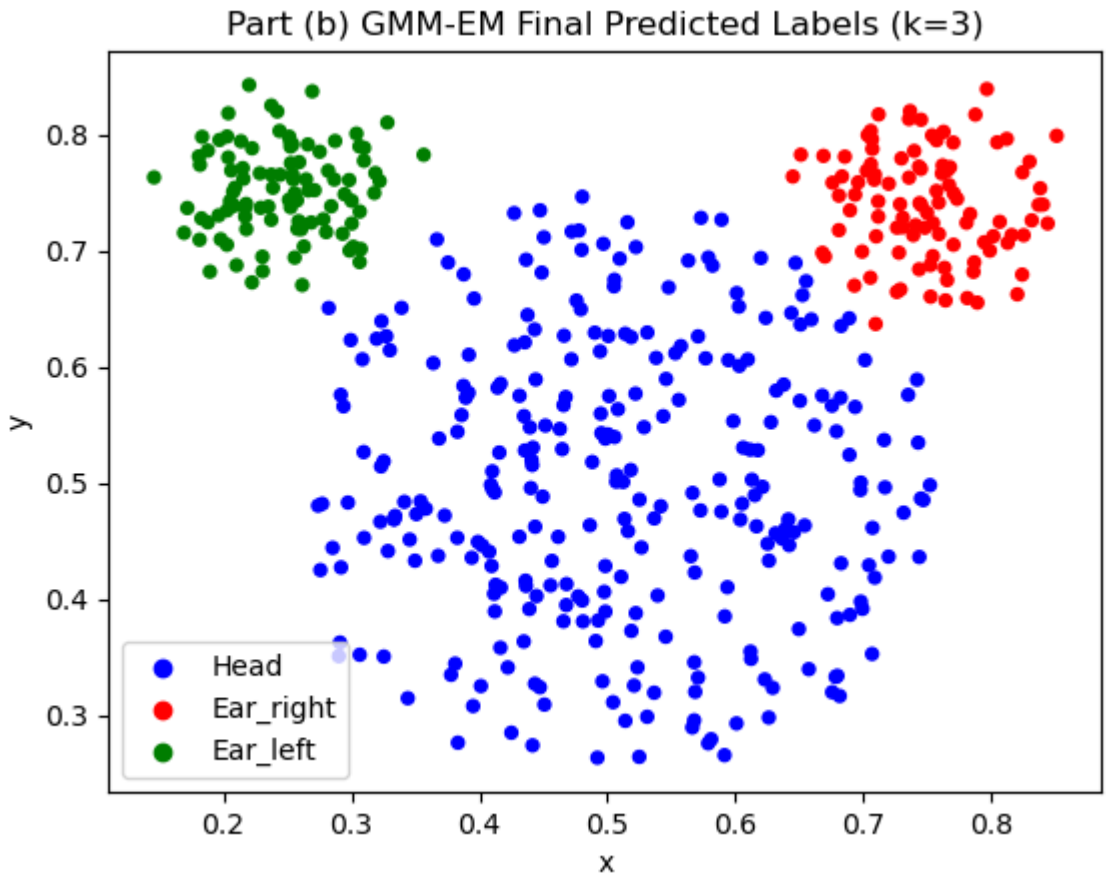

Part (b) GMM-EM assignments (iteration 1)
Head
Ear_right
Ear_left
x
y
0.2
0.3
0.4
0.5
0.6
0.7
0.8


Part (b) GMM-EM assignments (iteration 2)
Head
Ear_right
Ear_left
x
y
0.2
0.3
0.4
0.5
0.6
0.7
0.8






```
Cluster -> Class mapping: {0: np.str_('Ear_left'), 1: np.str_('Head'), 2: np.str_
('Ear_right')}
```




```
Part (b) results:
Confusion Matrix (rows=true, cols=pred) order: ['Head', 'Ear_right', 'Ear_left']
[[289   1   0]
 [  0 100   0]
 [  1   0  99]]
Per-class accuracy:
  Head: 0.9966
  Ear_right: 1.0000
  Ear_left: 0.9900
Overall accuracy: 0.9959
```

Q1.5

The main difference between K-means and GMM-EM is that K-means uses hard distance-based clustering, while GMM-EM models clusters probabilistically with Gaussian distributions. In the K-means result, several Head points are misclassified due to rigid cluster boundaries. GMM-EM produces cleaner separation and higher accuracy (about 0.996 vs 0.839), showing better performance.

In [ ]:

In [ ]: