冯楷翔期末课程设计——python应用案例

结合自己的学科，制作一个python应用的案例，完成相应的背景介绍（文字叙述）、功能介绍、实现思想或方式、采用所学的相应函数和库（numpy 或 pandas等，其他自学库可以）完成数据分析，并展示结果（图表绘制）。

要求：描述本案例自己的学科应用，用python完成的功能部分。

案例介绍：

这个项目案例介绍将基于一个雷达设备，对聚类后得到的点云数据进行卡尔曼滤波实现轨迹跟踪、管理和展示。我们从雷达设备获得了一系列的点云数据，这些数据经过聚类后形成了多个目标点。通过使用卡尔曼滤波器，我们能够有效地预测并跟踪这些目标的轨迹，同时展示和管理这些轨迹的变化。

案例背景

在雷达监控系统中，目标的检测和跟踪是非常重要的任务。雷达设备通常提供环境中的点云数据，这些数据通过聚类算法可以识别出不同的物体或目标。为了确保系统能实时、准确地预测和跟踪目标的运动轨迹，卡尔曼滤波作为一种高效的状态估计方法被广泛应用。

案例目标

1. **卡尔曼滤波实现目标的轨迹跟踪**：通过对聚类后的点云数据进行卡尔曼滤波，估算目标的真实位置、速度等状态。
2. **目标管理**：根据卡尔曼滤波的结果，对多个目标进行有效的管理，能够区分新目标、消失目标等。
3. **轨迹展示**：通过可视化展示跟踪到的目标轨迹，帮助系统开发者或用户观察目标的运动过程。

案例实现

1. **点云数据获取与预处理**
   - 雷达设备通过接口提供点云数据。
   - 使用聚类算法DBSCAN对点云进行处理，获取每个目标的位置信息。

```
data = scipy.io.loadmat('横着走然后向后.mat')
xydata = data['xydata']
```

2. **卡尔曼滤波器设计**

- 卡尔曼滤波器模型通常由状态转移矩阵、观测矩阵和噪声矩阵等构成。这里我们假设每个目标的状态包括位置、速度和加速度。
- 使用卡尔曼滤波器的状态方程：

$$x_k = A \cdot x_{k-1} + B \cdot u_k + w_k \tag{1}$$

$$z_k = H \cdot x_k + v_k \tag{2}$$

其中，x 为目标的位置，u为控制输入，w和 v 为过程噪声和观测噪声。

- 通过观测到的点云数据进行卡尔曼滤波更新目标的位置和速度。

```
# 参数设置
L_max = 8  # 最大未关联次数
hithold = 7
gate_threshold = 0.5  # 关联门限

T = 1
processnoise = 0.2
measurenoise = 1

# 状态转移矩阵 F 和观测矩阵 H
F = np.array([[1, T, 0, 0],
              [0, 1, 0, 0],
              [0, 0, 1, T],
              [0, 0, 0, 1]])

H = np.array([[1, 0, 0, 0],
              [0, 0, 1, 0]])

# 过程噪声协方差矩阵 Q 和观测噪声协方差矩阵 R
Q = (processnoise ** 2) * np.array([
    [T ** 3 / 3, T ** 2 / 2, 0, 0],
    [T ** 2 / 2, T, 0, 0],
    [0, 0, T ** 3 / 3, T ** 2 / 2],
    [0, 0, T ** 2 / 2, T]
])
```

3. 目标跟踪与管理

- 采用卡尔曼滤波器实现多目标跟踪算法。
- 当目标消失或未被观测时，及时移除目标实例；当有新目标出现时，创建新的卡尔曼滤波器实例。

```python
#main中每个帧，执行
# 更新已确认轨迹和临时轨迹
all_tracks = cf_tracks + temp_tracks
all_tracks, remaining_detections = update_tracks(all_tracks, sumpoint)

# 分离已确认轨迹和临时轨迹
cf_tracks = [track for track in all_tracks if track.hits >= hithold]
temp_tracks = [track for track in all_tracks if track.hits < hithold and track.livetime <= L_max]

# 创建新轨迹
if remaining_detections.size > 0:
    for det in remaining_detections:
        new_track = Track(det, i + 1)
        temp_tracks.append(new_track)
        print(f'创建新轨迹，ID: {new_track.id}, 帧: {i + 1}, 位置: ({det[0]:.2f}, {det[1]:.2f})')

# 删除未确认的老轨迹已经在 update_tracks 中处理

# 辅助函数
def update_tracks(tracks, detections):
    updated_tracks = []
    remaining_detections = detections.copy()
    for track in tracks:
        track.predict()
        min_distance = np.inf
        best_detection = None
        best_idx = -1
        for idx, detection in enumerate(remaining_detections):
            predicted_pos = H @ track.state
            distance = np.linalg.norm(detection - predicted_pos.flatten()[:2])
            if distance < min_distance:
                min_distance = distance
                best_detection = detection
                best_idx = idx
        if min_distance < gate_threshold:
            track.update(best_detection)
            updated_tracks.append(track)
            if best_idx >= 0:
                remaining_detections = np.delete(remaining_detections, best_idx, axis=0)
        else:
            track.increment_livetime()
            updated_tracks.append(track)
    return updated_tracks, remaining_detections
```

4. **轨迹展示**
   - 采用Python的可视化库（Matplotlib）展示卡尔曼滤波器估计的目标轨迹。
   - 将每个目标的估计位置和轨迹用不同颜色标识，形成动态更新的轨迹展示。

```
def plot_tracking(cf_tracks, sumpoint, frame_num, colors, ax):
    ax.cla()
    ax.set_xlim(-3, 3)
    ax.set_ylim(0, 3)
    ax.set_xlabel('X 坐标')
    ax.set_ylabel('Y 坐标')
    ax.set_title(f'使用卡尔曼滤波器的目标跟踪 - 帧 {frame_num}')
    ax.grid(True)

    # 绘制当前检测点
    if sumpoint.size > 0:
        ax.scatter(sumpoint[:, 0], sumpoint[:, 1], s=100, marker='s', color=[0.5, 0.5, 0.5], label='检测点')

    # 绘制已确认的轨迹
    for track in cf_tracks:
        color = colors[(track.id - 1) % len(colors)]
        history = np.array(track.history)
        ax.plot(history[:, 0], history[:, 1], 'o-', color=color, linewidth=2, label=f'目标 {track.id}')

    ax.legend(loc='upper right')
```

案例结构

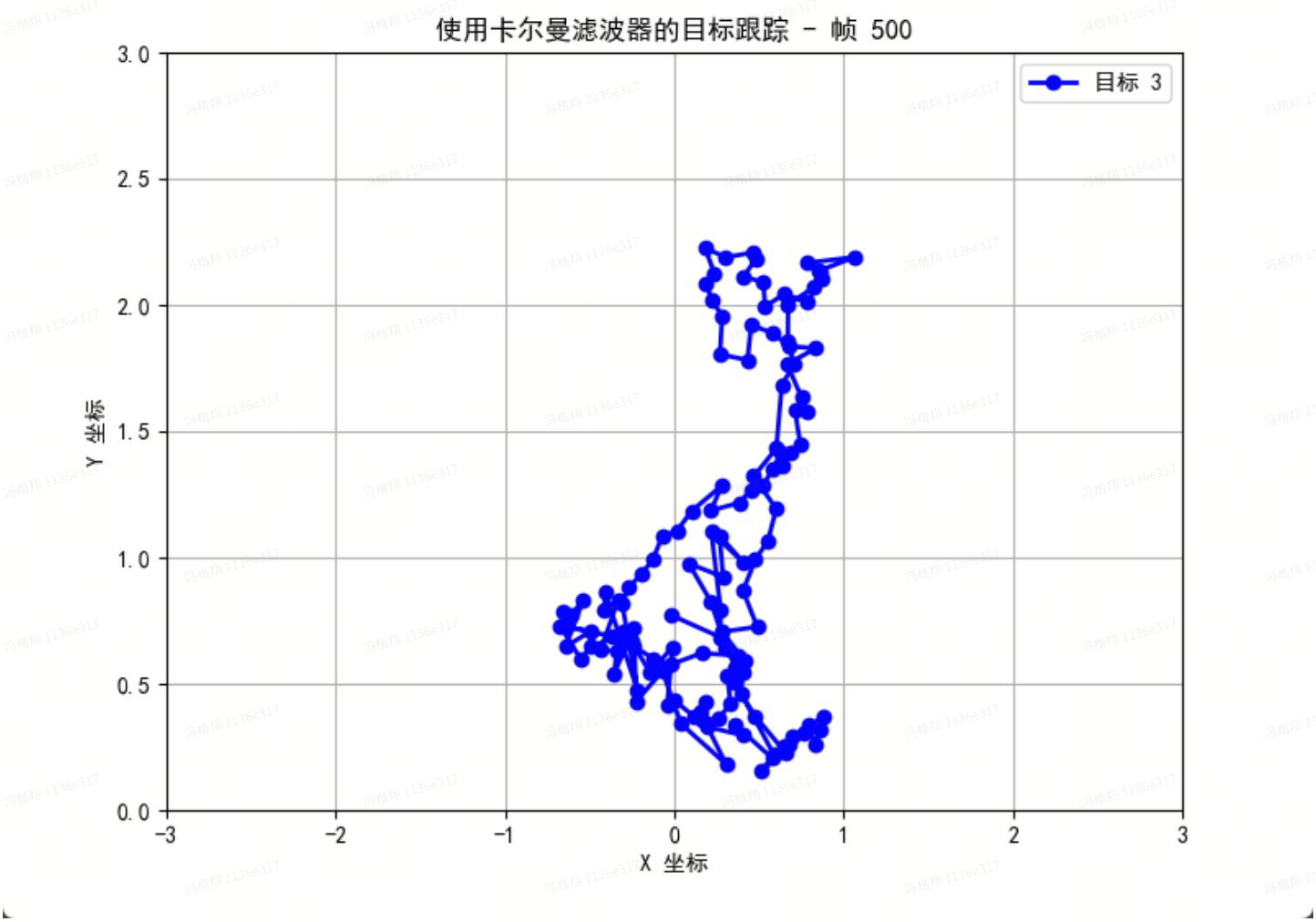
1. **数据输入模块**
   - 获取并加载雷达设备的预处理后的点云数据。
2. **卡尔曼滤波模块**
   - 实现卡尔曼滤波算法，包括状态预测、更新和协方差计算。
   - 实现多目标管理与数据关联。
3. **轨迹管理模块**
   - 管理多个目标的卡尔曼滤波状态。
   - 处理目标消失、新目标出现的情况。
4. **可视化展示模块**

- 展示目标的轨迹。
- 实时更新目标位置，显示卡尔曼滤波预测与实际观测的差异。

案例流程

1. **点云数据输入**：雷达设备定期生成点云数据，预处理后传输至处理系统。
2. **卡尔曼滤波**：使用卡尔曼滤波器预测和更新目标的运动状态，处理多个目标的跟踪。
3. **目标管理**：根据目标的状态更新目标列表，添加新的目标实例或删除已消失的目标。
4. **轨迹展示**：使用可视化工具动态显示目标轨迹，支持3D展示和轨迹重放。

案例展示



在面对目标左右前后移动时，程序可以将其识别为一个目标，并关联其航迹，达到了预期效果。（每帧关联的动画在文档外的tracking_animation.gif中）

案例应用

该项目能够广泛应用于室内目标跟踪等领域。还可能被用于在自动驾驶中，雷达设备通过点云数据提供对周围环境的实时感知。通过卡尔曼滤波的目标跟踪和预测功能，可以精确地跟踪道路上的其

他车辆、行人和障碍物，并预测其运动轨迹，帮助决策系统进行路径规划和障碍物避免。

总结

通过本项目的实施，我们实现了基于雷达点云数据的目标跟踪与管理，并通过卡尔曼滤波器提高了目标轨迹预测的准确性。该方法在多目标动态跟踪和管理方面具有较强的实用价值，可以作为自动化系统中的重要组成部分。