动态数据收集场景模拟详解

本文档详细描述了一个基于无人机 (UAV) 在动态传感器网络中收集数据的模拟场景。内容依据以下核心代码文件：

- 主程序: `showcases/Simulating_a_data_collection_scenario/main.py`
- 协议定义: `showcases/Simulating_a_data_collection_scenario/simple_protocol.py`
- 轨迹绘图: `showcases/Simulating_a_data_collection_scenario/plot_trajectories.py`

1. 场景概述

模拟的核心是一个无人机 (UAV) 在包含动态传感器节点的区域内执行数据收集任务。传感器节点可以被激活以产生数据并请求UAV服务，也可以在运行过程中动态地改变其激活或待机状态。UAV的目标是有效地规划路径，访问请求服务的传感器，收集数据，并最终将数据传输回固定的地面站。系统的有效通信范围设置为25米。

1.1. 节点类型与行为

模拟包含三种主要类型的节点：

1.1.1. 传感器节点 (Sensor Nodes)

- **协议类:** `SimpleSensorProtocol`
- **位置:** 节点位置从固定的 `sensor_locations_forest.json` 文件加载。地面站 (ID 0) 位于特定位置，其余传感器分布在预设坐标。
- **状态与行为:** 传感器状态由 `SensorStatus` 枚举定义，并在模拟过程中通过调用 `_update_sensor_status_file` 实时写入 `sensor_statuses.json`。
  - **原始 (RAW):** `SensorStatus.RAW`。所有传感器（地面站除外）的初始状态，表示未被配置。
  - **待机 (Standby):** `SensorStatus.STANDBY`。节点不产生数据也不请求服务。在模拟开始时，未被选为初始激活的传感器会被设为此状态。也可能在动态事件中被UAV设为待机。
  - **激活 (Active):** `SensorStatus.ACTIVE`。
    - 在 `main.py` 中，通过随机抽样选择一部分初始传感器，并调用其 `activate()` 方法 (`showcases/Simulating_a_data_collection_scenario/simple_protocol.py:201`) 来激活。
    - 也可以在动态事件中被UAV激活。
    - 激活后，传感器通过 `_generate_packet` (`showcases/Simulating_a_data_collection_scenario/simple_protocol.py:192`) 以0.5秒的间隔生成数据包，并广播服务请求。
  - **已服务 (Serviced):** `SensorStatus.SERVICED`。当UAV进入其通信范围内时，传感器将数据发送给UAV，状态变为“已服务”，并停止活动。

1.1.2. 无人机 (UAV)

- **协议类:** `SimpleUAVProtocol`
- **初始化:**
  - 在 `main.py` 中创建，并通过 `configure` 方法 (`showcases/Simulating_a_data_collection_scenario/simple_protocol.py:316`) 配置初始要访问的传感器列表 (`initial_sensor_tour_tsp_ids`)、所有传感器的坐标地图、飞行高度 (20米)、地面站信息、重规划阈值 (`replan_threshold`)、动态激活传感器数量 (`num_sensors_to_activate`) 和动态待机传感器数量 (`num_sensors_to_standby`)。

    - 调用 `_build_initial_mission` (showcases/Simulating_a_data_collection_scenario/simple_protocol.py:382)，使用LKH-3路径规划算法计算初始任务路径。
- **行为:**
    - **心跳广播:** 每秒广播其位置并消耗能量。
    - **服务请求处理:** 监听并收集来自传感器的服务请求，存入 `pending_service_requests`。
    - **路径规划与动态事件处理:**
        - **初始路径 (闭环TSP):**
            - **目标节点:** 地面站和所有初始激活的传感器。
            - **构建与求解:** 在 `_build_initial_mission` 中，构建一个标准的闭环TSP问题，并调用 `_solve_tsp_with_lkh` (showcases/Simulating_a_data_collection_scenario/simple_protocol.py:466) 求解最优巡游路径。
        - **动态事件触发与处理:**
            - **触发条件:** 当UAV服务过的传感器数量达到 `replan_threshold` 时触发。
            - **UAV悬停:** 设置 `is_awaiting_replan = True`，并调度1秒后的路径重规划，UAV在此期间悬停。
            - **执行动态事件 (`_trigger_dynamic_event`):** (showcases/Simulating_a_data_collection_scenario/simple_protocol.py:629)
                1. **激活待机传感器:** 从待机列表中随机选择 `num_sensors_to_activate` 个传感器并激活。
                2. **待机当前路径传感器:** 从当前任务队列中未访问的传感器中，随机选择 `num_sensors_to_standby` 个传感器并设为待机。这些被显式停用的传感器信息会被记录在 `dynamically_deactivated_sensors_log` 中。
        - **路径重规划 (`replan_path`):** (showcases/Simulating_a_data_collection_scenario/simple_protocol.py:725)
            - **触发:** 由1秒延迟的 `replan` 定时器触发。
            - **过程:**
                1. **定义起止点:** UAV的当前位置为起点，地面站为终点。
                2. **收集中间点:** 包括当前任务队列中剩余的目标点和所有新收到的服务请求（均需排除被动态设为待机的传感器）。

3. **构建并求解开环TSP:** 调用 `_solve_open_tsp_with_lkh` (`showcases/Simulating_a_data_collection_scenario/simple_protocol.py:564`)，通过“虚拟节点”技巧求解从起点到终点的最优开环路径。
4. **更新任务队列:** 使用新的开环路径更新任务队列。

- **数据收集与转储:** 飞行中收集数据，任务完成后返回地面站并转储所有数据。
- **日志记录:** 在 `finish` 方法 (`showcases/Simulating_a_data_collection_scenario/simple_protocol.py:832`) 中，将飞行轨迹（分割为初始和重规划）、能量消耗和动态停用传感器日志保存到多个文件中。

1.1.3. 地面站 (Ground Station)

- **协议类:** `SimpleGroundStationProtocol`
- **行为:** 接收UAV转储的数据，发送ACK确认消息以结束仿真，并将接收到的数据保存到 `gs_{id}_received_data.json`。

2. 程序执行逻辑

程序执行由 `main.py` 中的 `main` 和 `run_simulation` 函数控制。

2.1. `run_simulation` 函数

1. **清理与初始化:** 删除旧的 `sensor_statuses.json`，然后重新创建它，将所有传感器的状态初始化为 `RAW`。
2. **加载配置:** 从 `sensor_locations_forest.json` 加载节点坐标。
3. **模拟器构建:** 构建 `SimulationBuilder` 并添加所需处理器。
4. **节点实例化与配置:** 创建所有节点，并为UAV和传感器配置必要的参数和初始状态（激活或待机）。
5. **模拟启动:** 调用 `simulation.start_simulation()`。
6. **后处理与分类:**
   - 仿真结束后，读取最终的 `sensor_statuses.json` 和 `uav_*_dynamically_deactivated_sensors.json` 文件。
   - 根据传感器的初始角色、最终状态和是否被动态停用，为每个传感器分配一个绘图类别（如 `INITIAL_ACTIVE`, `DYNAMIC_DEACTIVATED_EXPLICIT` 等）。

    - 将分类结果保存到 `plot_categories.json`，供绘图脚本使用。
7. **结果收集:** 收集并返回该次运行的性能指标。

2.2. `main` 函数

定义实验场景，循环调用 `run_simulation`，并最终将所有运行的性能指标汇总保存到 `simulation_results.csv`。

3. 结果可视化

通过运行 `plot_trajectories.py` 脚本可以对模拟结果进行可视化。

- **输入:** 该脚本读取 `sensor_locations_forest.json` 来获取所有节点的位置，并读取 `plot_categories.json` 来获取每个传感器的分类。同时，它会查找所有 `uav_*_trajectory.csv` 文件来绘制路径。
- **输出:** 生成一张名为 `trajectories.png` 的3D散点图，其中包含：
    - **地面站:** 黑色金字塔标记。
    - **传感器节点 (根据类别):**
        - `Initial Active Sensors`: 绿色圆圈，表示初始被激活并最终被成功服务的传感器。
        - `Activated -> Standby (Dynamic Event)`: 红色大'X'，表示在动态事件中被UAV明确指令进入待机的传感器。
        - `Activated -> Standby (Unserviced)`: 橙色小'x'，表示初始被激活但因路径重规划等原因最终未被服务的传感器。
        - `Dynamic Newly Activated`: 青色'P'标记，表示在动态事件中从待机状态被激活并成功服务的传感器。
    - **UAV 轨迹:**
        - `Initial UAV Trajectory`: 蓝色实线，表示重规划之前的路径。
        - `Replanned UAV Trajectory`: 红色实线，表示重规划之后的路径。

4. 输出文件

- `showcases/Simulating_a_data_collection_scenario/simulation_results.csv`: 所有模拟运行的详细性能指标。

- showcases/Simulating_a_data_collection_scenario/sensor_statuses.json: 单次运行中，传感器状态的实时记录。
- showcases/Simulating_a_data_collection_scenario/plot_categories.json: 为绘图脚本准备的传感器分类数据。
- showcases/Simulating_a_data_collection_scenario/uav_{id}_initial_trajectory.csv: UAV在重规划之前的飞行轨迹。
- showcases/Simulating_a_data_collection_scenario/uav_{id}_replan_trajectory.csv: UAV在重规划之后的飞行轨迹。
- showcases/Simulating_a_data_collection_scenario/uav_{id}_energy.csv: UAV的能量消耗记录。
- showcases/Simulating_a_data_collection_scenario/uav_{id}_dynamically_deactivated_sensors.json: 在动态事件中被UAV设为待机的传感器日志。
- gs_{id}_received_data.json: 地面站接收到的原始数据批次。
- trajectories.png: 最终生成的可视化轨迹图。