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`

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:** 节点位置在首次运行时随机生成并保存到 `sensor_locations.json`。地面站 (ID 0) 位于区域中心，其余传感器随机分布。
- **状态与行为:**
  - **待机 (Standby):** `SensorStatus.STANDBY`。默认状态或被UAV动态设置为待机，节点不产生数据也不请求服务。
  - **激活 (Active):** `SensorStatus.ACTIVE`。
    - 在 `main.py` 中，通过 `set_active_on_start(True)` (`showcases/Simulating_a_data_collection_scenario/main.py:122`) 激活一部分初始传感器。
    - 也可以在动态事件中被UAV通过调用传感器的 `activate()` 方法 (`showcases/Simulating_a_data_collection_scenario/simple_protocol.py:115`) 激活。
    - 激活后，传感器通过 `_generate_packet` 方法 (`showcases/Simulating_a_data_collection_scenario/simple_protocol.py:106`) 以0.5秒的间隔生成数据包，并累积到批次中。
    - 同时，激活的传感器会通过 `_request_service` 方法 (`showcases/Simulating_a_data_collection_scenario/simple_protocol.py:131`) 每秒广播一次服务请求，直到UAV前来服务或被设为待机。
  - **已服务 (Serviced):** `SensorStatus.SERVICED`。当UAV进入其通信范围内时，传感器通过 `handle_packet` (`showcases/Simulating_a_data_collection_scenario/simple_protocol.py:146`) 将本地缓存的数据包 (`PacketBatch`) 发送给UAV。发送后，传感器状态变为“已服务”，并停止发送数据和请求，直到被再次激活。
- **结束阶段:** 在仿真结束时，每个传感器节点的 `finish` 方法 (`showcases/Simulating_a_data_collection_scenario/simple_protocol.py:185`) 会将其最终状态 (Standby, Active, or Serviced) 写入 `showcases/Simulating_a_data_collection_scenario/sensor_statuses.json` 文件。

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:214`) 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)。
    - 初始能量设置为 50000.0，能量消耗率设置为 1.0 (单位/秒)。
    - 调用 _build_initial_mission 方法 (showcases/Simulating_a_data_collection_scenario/simple_protocol.py:279)，使用LKH-3路径规划算法计算初始任务路径。
    - 初始化 standby_sensor_tsp_ids 列表，包含所有未在初始路径中的传感器 (showcases/Simulating_a_data_collection_scenario/simple_protocol.py:268)。
    - 初始化 is_awaiting_replan = False 标志 (showcases/Simulating_a_data_collection_scenario/simple_protocol.py:262)。
- **行为:**
    - **心跳广播 (Heartbeat):** 每秒通过 _send_heartbeat (showcases/Simulating_a_data_collection_scenario/simple_protocol.py:298) 广播包含当前位置的消息，并消耗能量。
    - **服务请求处理:** 在 handle_packet (showcases/Simulating_a_data_collection_scenario/simple_protocol.py:320) 中监听并收集来自传感器的服务请求，存入 pending_service_requests 字典。
    - **路径规划与动态事件处理:**
        - **初始路径 (闭环TSP):**
            - **目标节点:** 初始路径规划的目标节点集合包括**地面站**和所有在 main.py 中被设置为**初始激活**的传感器（由 num_devices 参数决定）。
            - **构建与求解:** 在 _build_initial_mission 方法中，程序将地面站的位置作为路径的起点和终点（Depot），结合所有初始激活传感器的坐标，构建一个标准的闭环TSP问题。随后调用 _solve_tsp_with_lkh 方法 (showcases/Simulating_a_data_collection_scenario/simple_protocol.py:360)，使用LKH求解器计算出访问这些指定节点并返回地面站的最优巡游路径。
        - **动态事件触发与处理:**

- **触发条件:** 当UAV服务过的传感器数量 (`len(self.serviced_sensor_tsp_ids)`) 达到或超过 `replan_threshold` 且重规划尚未被调度时 (showcases/Simulating_a_data_collection_scenario/simple_protocol.py:338)。
- **UAV悬停:**
  - 设置 `self.is_awaiting_replan = True`。
  - 清除当前目标 `self.current_destination = None` (showcases/Simulating_a_data_collection_scenario/simple_protocol.py:348)，防止在 `_process_mission_queue` 中选取新目标。
  - 调度 `replan` 定时器，延迟1秒执行 (`self.provider.current_time() + 1.0`) (showcases/Simulating_a_data_collection_scenario/simple_protocol.py:341)。这1秒即为UAV的悬停时间，以等待新激活的传感器产生数据。
- **执行动态事件 (`_trigger_dynamic_event`):** (showcases/Simulating_a_data_collection_scenario/simple_protocol.py:508)
  1. **激活待机传感器:** 从 `self.standby_sensor_tsp_ids` 中随机选择 `self.num_sensors_to_activate` 个传感器，调用其 `activate()` 方法。成功激活的传感器会从待机列表中移除，并如果它们已初始化位置并有数据，其服务请求会被加入 `self.pending_service_requests`。
  2. **待机当前路径传感器:** 从当前任务队列 `self.mission_queue` 中尚未访问且未被动态停用的传感器中，随机选择 `self.num_sensors_to_standby` 个传感器，调用其 `deactivate()` 方法，并将其ID记录到 `self.dynamically_deactivated_tsp_ids`。
- **路径重规划 (`replan_path`):** (showcases/Simulating_a_data_collection_scenario/simple_protocol.py:586)
  - **触发:** 由1秒延迟的 `replan` 定时器触发。
  - **过程:**
    1. **定义起止点:** 将无人机的**当前位置**作为路径的起点，**地面站**作为路径的终点。
    2. **收集中间点 (`points_to_replan_map`):**

        - 包括当前任务队列 `self.mission_queue` 中未完成的目标点（必须未被服务且未被动态设置为待机）。
        - 包括 `self.pending_service_requests` 中的所有新服务请求（这些包含了动态事件中新激活的传感器，且这些传感器也未被服务或动态设置为待机）。
    3. **构建并求解开环TSP:** 调用 `_solve_open_tsp_with_lkh` 方法 (`showcases/Simulating_a_data_collection_scenario/simple_protocol.py:450`)。该方法内部通过“虚拟节点”技巧（创建非对称距离矩阵ATSP，添加虚拟终点等）来生成从当前位置出发，访问所有收集到的中间点，并最终到达地面站的最优开环路径。
    4. **更新任务队列:** 将求解器返回的开环路径（不含起点）作为新的任务序列，并在其末尾追加**真实终点（地面站）**。
    5. 重置 `self.is_awaiting_replan = False` (`showcases/Simulating_a_data_collection_scenario/simple_protocol.py:629`)。
- **任务执行 (`_process_mission_queue`):** (`showcases/Simulating_a_data_collection_scenario/simple_protocol.py:632`)
    - 如果 `is_awaiting_replan` 为 `True`，则UAV悬停，不处理队列。
    - 否则，顺序处理 `mission_queue` 中的航点，使用 `GotoCoordsMobilityCommand` 飞往下一个目标。
    - 到达目的地后，调度下一次移动。
- **数据收集:** 当UAV飞到传感器25米通信范围内时，传感器会自动发送数据。UAV在 `handle_packet` 中接收数据，存入 `buffered_packet_batches`，并记录已服务的传感器ID。
- **数据转储:** 当任务队列为空（即完成所有航点访问并返回地面站）后，在 `handle_telemetry` (`showcases/Simulating_a_data_collection_scenario/simple_protocol.py:652`) 中触发数据转储，将所有缓存的数据包发送给地面站。
- **日志记录:** 在 `finish` 方法 (`showcases/Simulating_a_data_collection_scenario/simple_protocol.py:685`) 中，将飞行轨迹（根据重规划时间点分割为初始和重规划两部分）和能量消耗日志分别保存到 `uav_{id}_initial_trajectory.csv`、

`uav_{id}_replan_trajectory.csv` 和 `uav_{id}_energy.csv` 文件中。

1.1.3. 地面站 (Ground Station)

- **协议类:** `SimpleGroundStationProtocol`
- **位置:** ID为0的节点，位于区域中心。
- **行为:**
  - **数据接收:** `handle_packet` (`showcases/Simulating_a_data_collection_scenario/simple_protocol.py:735`) 接收UAV发送的 `is_data_dump` 消息，记录接收到的数据包批次和接收时间。
  - **确认 (ACK):** 收到数据后，向UAV发送一个确认消息 (`is_ack = True`)，该消息同时包含 `simulation_should_end = True` 标志，用于通知UAV结束仿真。
  - **数据保存:** 在 `finish` 方法 (`showcases/Simulating_a_data_collection_scenario/simple_protocol.py:757`) 中，将所有接收到的数据批次保存到 `gs_{provider_id}_received_data.json` 文件。

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

该函数负责执行单次模拟运行。

1. **清理:** 删除上一次运行生成的 `sensor_statuses.json` 文件。
2. **参数配置:** 从 `scenario` 字典中读取 `standby_sensors`, `replan_threshold`, `num_sensors_to_activate`, `num_sensors_to_standby` 等参数。定义模拟区域大小 (`AREA_X`, `AREA_Y`)、UAV飞行高度等常量。
3. **节点位置管理:**
   - 检查 `sensor_locations.json` 是否存在。
   - 若不存在，调用 `generate_sensor_locations` 生成所有节点（地面站、初始激活传感器、待机传感器）的随机坐标并保存。
   - 若存在，则直接通过 `load_sensor_locations` 加载。

4. **模拟器构建:**
    - 创建 `SimulationBuilder` 实例。
    - 添加 `TimerHandler`, `CommunicationHandler` (传输范围25米), 和 `MobilityHandler`。
5. **节点实例化:**
    - 添加地面站、所有传感器和UAV节点到模拟器中。
6. **协议配置:**
    - 获取UAV和传感器协议实例。
    - 调用UAV的 `configure` 方法 (`showcases/Simulating_a_data_collection_scenario/main.py:101`)，传入初始任务所需信息以及从场景配置中读取的动态事件参数。
    - 为每个传感器实例设置其TSP ID。
7. **初始传感器激活:**
    - 根据传入的 `num_devices` 参数，激活相应数量的初始传感器。
8. **模拟启动与结束:**
    - 调用 `simulation.start_simulation()` 启动仿真。
    - 仿真在UAV收到地面站的ACK后自行结束。
9. **结果收集:**
    - 仿真结束后，从地面站、传感器和UAV协议中收集性能指标（如延迟、吞吐量、路径长度、能耗等）。
    - 返回一个包含所有结果的字典。

2.2. `main` 函数

该函数是程序的入口点，负责管理多次模拟运行。

1. **场景定义:** 定义一个包含不同场景的列表 (`scenarios`) (`showcases/Simulating_a_data_collection_scenario/main.py:167`)。每个场景字典包含 `"devices"` (初始激活传感器数), `"replan_threshold"`, `"standby_sensors"`, `"num_sensors_to_activate"`, 和 `"num_sensors_to_standby"`。
2. **循环执行:**
    - 遍历所有场景和指定的重复次数 (`repetitions`)。
    - 在每次循环中，调用 `run_simulation` 函数执行一次模拟。
    - 将每次运行的结果追加到 `results` 列表中。
3. **结果保存:**

- 所有模拟运行结束后，使用 `csv.DictWriter` 将 `results` 列表中的所有数据写入到 `showcases/Simulating_a_data_collection_scenario/simulation_results.csv` 文件中。

3. 关键参数与取值

- **场景配置 (`main.py`):**
  - `scenarios`: 示例场景A: `{"name": "Scenario A", "devices": 50, "replan_threshold": 50, "standby_sensors": 200, "num_sensors_to_activate": 50, "num_sensors_to_standby": 20}`.
  - `repetitions`: 1 (可按需修改)。
- **环境与节点 (`run_simulation`):**
  - `NUM_STANDBY_SENSORS`: 从场景配置中读取。
  - `AREA_X`, `AREA_Y`: 500x500
  - `UAV_FLIGHT_ALTITUDE`: 20.0
- **通信 (`CommunicationHandler`):**
  - `transmission_range`: 25
- **UAV协议 (`SimpleUAVProtocol`):**
  - `initial_energy`: 50000.0
  - `energy_consumption_rate`: 1.0
  - `replan_threshold`: 从场景配置中读取 (例如50个传感器)。
  - `num_sensors_to_activate`: 从场景配置中读取 (例如50个)。
  - `num_sensors_to_standby`: 从场景配置中读取 (例如20个)。
  - **动态事件后悬停时间:** 1.0 秒。
- **传感器协议 (`SimpleSensorProtocol`):**
  - 数据包生成间隔: 0.5 秒。
  - 服务请求广播间隔: 1.0 秒。
- **LKH求解器参数 (`_solve_tsp_with_lkh`, `_solve_open_tsp_with_lkh`):**
  - `MOVE_TYPE = 5`
  - `PATCHING_C = 3`
  - `PATCHING_A = 2`
  - `RUNS = 1`

4. 输出文件

- `showcases/Simulating_a_data_collection_scenario/simulation_results.csv`: 记录所有模拟运行的详细性能指标。
- `sensor_locations.json`: 缓存的节点坐标，用于重复实验。
- `showcases/Simulating_a_data_collection_scenario/sensor_statuses.json`: 记录单次运行结束时所有传感器的最终状态。
- `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的能量消耗随时间变化的记录。
- `gs_{id}_received_data.json`: 地面站接收到的原始数据批次。
- 控制台日志：显示模拟过程中的关键事件和状态。