PX4 虚拟多机 + 容器化 vuav_agent + 群控脚本里程碑说明

（内部技术文档草稿）

2025-12-05

目录

1 整体目标与当前里程碑

1.1 项目背景与目标简述

本项目面向后续“虚拟无人机集群 + 可信计算/TEE 防护”方向的教学与科研需求，尝试构建一个尽量轻量的基础平台：

- 使用 PX4 SITL（此处采用 SIH 模式）作为虚拟无人机飞控内核；
- 每个虚拟无人机（vUAV）封装在一个 Docker 容器中，容器内同时运行 PX4 与一个自定义的 MAVLink 代理脚本 `vuav_agent.py`；
- 在宿主机上，通过 `swarm_controller.py` 统一连接多个 vUAV 的 agent，实现最基本的“探活、状态查询、起飞命令、位置/高度监控”等群控操作；
- 后续在此基础上逐步叠加：OFFBOARD 控制、编队轨迹规划、可信执行环境中的控制决策等。

截至 2025-12-05，目前平台已经完成的工作可以视为一个阶段性里程碑：

1) **容器镜像构建与复现路径打通**
   - 基于 Ubuntu + PX4 源码 + Python 运行时，构建出自包含的镜像；
   - 镜像内可以直接启动 PX4 SIH 模式，并通过 `pymavlink` 连接到 `udp:127.0.0.1:14540`；
   - 使用 Docker 映射宿主端口，将容器内的 TCP 监听（`vuav_agent`）暴露出来。
2) **多容器虚拟无人机（vUAV）实例可稳定启动**
   - 当前采用固定的三机配置：`vuav1`, `vuav2`, `vuav3`；
   - 每个容器中都有一份 PX4 + `vuav_agent.py`，对外提供统一的 JSON/TCP 控制接口。
3) **容器内 MAVLink 代理脚本 `vuav_agent.py` 初步成型**
   - 负责：
     (a) 与本容器内 PX4 建立 MAVLink 连接（当前为 `udp:127.0.0.1:14540`）；
     (b) 在容器内监听 TCP 端口 `6000`，接收来自宿主机的 JSON 命令；
     (c) 将 JSON 命令翻译为 MAVLink 消息(如 `COMMAND_LONG`、`SET_POSITION_TARGET_LOCAL_NED` 等)，并将结果封装为 JSON 返回；
     (d) 提供基本的“探活、姿态、起飞、局部位置、速度控制占位”等能力。
4) **宿主机群控脚本 `swarm_controller.py` 实现**
   - 在宿主机上维护一个 vUAV 列表（当前为三机）；
   - 提供广播命令、轮询状态、本地位置监控、高度曲线监控等函数；
   - 支持按名称调用单个 vUAV 的 agent，实现细粒度操作；
   - 初步加入了基于 `set_velocity_ned` 的速度控制广播占位函数。
5) **当前可用的基本功能**
   - **容器层面**：
     – 创建/启动/停止 vUAV 容器；
     – 通过 `docker logs` 观测 PX4 日志（包括 Preflight 检查告警、SITL 初始化信息等）。
   - **MAVLink/控制层面**：
     – 通过 `vuav_agent.py` 将以下能力以 JSON RPC 的形式暴露给宿主机：
       * `ping` 探活；
       * `get_status` 读取 ATTITUDE；
       * `takeoff`（`MAV_CMD_NAV_TAKEOFF`，当前能收到 `COMMAND_ACK`，result=0）；
       * `get_local_position` 读取 `LOCAL_POSITION_NED`，用于姿态/高度监控；

* monitor_altitude_curve 对高度 alt $= -z$ 做时间序列采样，辅助判断“是否真正离地”；
* set_velocity_ned/broadcast_velocity_ned 作为基于 SET_POSITION_TARGET_LOCAL_NED 的速度控制占位接口，为后续 OFFBOARD 控制和编队轨迹铺垫。

– 宿主机使用 swarm_controller.py 可以对三机做统一的 ping、状态轮询、起飞命令分发、位置/高度监控。

2 容器镜像与部署方式

2.1 当前镜像版本与命名约定

当前使用的最新镜像名为：

```
px4-vuav-agent:v5-20251204
```

该镜像主要包含：

- Ubuntu 基础系统（版本见 Dockerfile）；
- 编译好的 PX4 SITL（本项目采用 SIH 模式以简化外部仿真依赖）；
- Python3 运行时与依赖包（如 pymavlink 等）；
- 项目目录 /px4 下的：
  – PX4 可执行程序及其配置；
  – MAVLink 代理脚本 vuav_agent.py；
  – 启动脚本（例如用于在容器启动时并行拉起 PX4 与 agent）。

镜像版本号后缀 v5-20251204 主要用于记录：

- 第 5 次比较大的镜像迭代；
- 镜像内部 PX4 版本、vuav_agent.py 和基础工具链的状态稳定在 2025-12-04 的某个 commit。

2.2 基础运行方式示例

当前建议在宿主机项目目录（例如 $HOME/px4-docker）下，通过以下方式启动三台 vUAV：

```
# 以 vuav1 为例，映射宿主端口 16001 -> 容器内 TCP 6000
docker run --rm -it \
  --name vuav1 \
  -p 16001:6000 \
  px4-vuav-agent:v5-20251204

# 类似地启动 vuav2, vuav3
docker run --rm -it \
  --name vuav2 \
  -p 16002:6000 \
  px4-vuav-agent:v5-20251204

docker run --rm -it \
  --name vuav3 \
  -p 16003:6000 \
```

```
px4-vuav-agent:v5-20251204
```

容器内部启动脚本的基本思路是：

- 在后台启动 PX4 SITL（SIH 模式），监听 MAVLink UDP 端口；
- 稍作等待后，启动 `vuav_agent.py`，并连接到 `udp:127.0.0.1:14540`；
- `vuav_agent.py` 在容器内监听 TCP 6000 端口，等待宿主机连接。

宿主机侧通过端口映射，将 `127.0.0.1:1600X` 与容器内的 `0.0.0.0:6000` 对接，从而形成：

宿主机 swarm_controller.py ⇒ TCP 1600X ⇒ 容器 `vuav_agent.py` ⇒ MAVLink ⇒ PX4

2.3 镜像备份与恢复建议

镜像备份

为了方便日后在其他机器或环境中快速复现当前状态，建议定期导出镜像为 `.tar` 包，例如：

```
# 导出当前镜像（注意版本号保持和上文一致）
docker save -o px4-vuav-agent_v5-20251204.tar px4-vuav-agent:v5-20251204
```

该 `.tar` 文件可放置在：

- Git 仓库的 Release（注意 2 GB 以上不适合直接放 GitHub）；
- 内部 NAS/网盘；
- 或通过硬盘拷贝作为长期备份。

镜像恢复

在新环境或重新安装系统后，只要有上述 `px4-vuav-agent_v5-20251204.tar` 备份文件，即可通过以下命令恢复：

```
# 从 tar 文件加载镜像
docker load -i px4-vuav-agent_v5-20251204.tar

# 加载完成后，可以通过 docker images 查看
docker images | grep px4-vuav-agent
```

只要镜像名和标签仍为 `px4-vuav-agent:v5-20251204`，上文启动命令即可直接复用，无需重新构建。

3 容器内代理程序 vuav_agent.py 概述

3.1 整体功能

`vuav_agent.py` 的职责是：

1) 连接本容器内的 PX4 SITL，使用 MAVLink 建立稳定会话：
   - 当前使用 URL：`udp:127.0.0.1:14540`；
   - 通过 `mavutil.mavlink_connection` 建立连接；
   - 调用 `wait_heartbeat()` 确认 PX4 正常启动。

2) 在 TCP 端口 6000 上监听宿主机的连接请求：
   - 每收到一个新连接，创建一个新的线程处理；
   - 按行解析 JSON（以换行符 "\n" 为分包边界）；
   - 每条 JSON 请求对应一次 process_request 调用，并返回一条 JSON 响应。
3) 将上层 JSON 命令翻译为 MAVLink 消息，并从 MAVLink 收集返回结果，封装为 JSON 返回给宿主机。

3.2 mavlink_connect() 与心跳确认

核心代码结构如下（只保留关键片段）：

```
MAVLINK_URL = "udp:127.0.0.1:14540"

def mavlink_connect():
    print(f"[agent] connecting to PX4 via {MAVLINK_URL} ...")
    m = mavutil.mavlink_connection(MAVLINK_URL)
    m.wait_heartbeat(timeout=10)
    print(f"[agent] HEARTBEAT from PX4: sysid={m.target_system}, compid={m.
        target_component}")
    return m
```

说明：

- wait_heartbeat 有超时（10 秒），若 PX4 未正常启动会直接抛异常/退出；
- 成功收到心跳后，打印系统 ID 和组件 ID，方便区分多机场景；
- 此处假定容器内的 PX4 与 agent 同机，使用本地 UDP 端口通信。

3.3 通用命令发送封装：send_command_long()

```
def send_command_long(mav, command, params, timeout=2.0):
    """
    发送 MAV_CMD_*，并等待 COMMAND_ACK。
    params: 长度为 7 的 list/tuple -> param1..param7
    """
    mav.mav.command_long_send(
        mav.target_system,
        mav.target_component,
        command,
        0,
        *params
    )
    ack = mav.recv_match(type='COMMAND_ACK', blocking=True, timeout=timeout)
    if not ack:
        return {"ok": False, "error": "no COMMAND_ACK"}

    ok = (ack.result == mavutil.mavlink.MAV_RESULT_ACCEPTED)
    return {
```

```
    "ok": ok,
    "result": int(ack.result),
    "command": int(ack.command),
}
```

该函数被用来封装例如：

- 解锁/上锁：MAV_CMD_COMPONENT_ARM_DISARM；
- 起飞：MAV_CMD_NAV_TAKEOFF。

上层通过 JSON 命令触发时，只需关心“命令是否被 PX4 接受（result==0）”，具体原因（例如 Preflight 检查不通过等）可以在 PX4 端日志中进一步排查。

3.4 命令处理逻辑 process_request()

process_request(req, mav) 是 agent 的核心逻辑入口，负责根据 JSON 请求中的 "cmd" 字段分发到不同 MAVLink 操作。整体模式是：

- 解析 cmd = req.get("cmd")；
- 针对不同字符串执行不同分支；
- 若不认识该命令，统一返回 "unknown cmd"。

当前支持的命令集包括（节选）：

- ping：简单探活，直接返回 "pong"；
- get_attitude：阻塞等待一条 ATTITUDE 消息，返回欧拉角（度）；
- get_status：同样基于 ATTITUDE，但封装为 "status" 结构；
- arm/disarm：分别调用 MAV_CMD_COMPONENT_ARM_DISARM 的 1/0 版本；
- takeoff：调用 MAV_CMD_NAV_TAKEOFF，支持通过 "alt" 参数指定目标高度；
- get_local_position：阻塞等待一条 LOCAL_POSITION_NED，返回 (x, y, z, vx, vy, vz)，其中 $z > 0$ 表示向下；
- set_velocity_ned：基于 SET_POSITION_TARGET_LOCAL_NED 发送简单速度控制，只填充 v_x, v_y, v_z 和偏航角速度 yaw_rate，其它字段通过 type_mask 屏蔽（实验性接口，需要 PX4 处于 OFFBOARD 等合适模式）；
- goto_local_ned：航点接口占位，目前仅解析目标 (x, y, z) 并返回“未实现”，为后续路径规划/航点导航逻辑预留扩展点；
- 未实现命令统一返回 "unknown cmd"。

出于文档篇幅考虑，完整 process_request 源码不在此展开，实际项目中会随本文档一起发布，便于读者直接查看。

3.5 TCP 会话处理 handle_client() 与主循环

handle_client(conn, addr, mav) 的职责是：

- 从 conn.recv 中持续读入字节流，缓存在 buf 中；
- 每当发现换行符 "\n"，就将前面的内容视为一条 JSON；
- 尝试解析 JSON，调用 process_request，并将响应 JSON 加上换行后发回客户端；
- 任意异常都会被捕获并转换为 "handler exception"，避免直接断开连接导致调试困难。

主循环 agent_main() 逻辑：

1) 调用 mavlink_connect() 与 PX4 建立连接；
2) 打开 TCP 监听 socket：

```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
s.bind(("0.0.0.0", 6000))
s.listen(5)
```

3) 每接收到一个新连接，就创建一个守护线程，调用 handle_client 处理；
4) 在容器退出前，主线程不会主动关闭该 socket。

4 宿主机群控脚本 swarm_controller.py 概述

4.1 设计思路

swarm_controller.py 主要是为了在宿主机上统一操作多个 vUAV 的 agent。其设计目标包括：

- 将每个 vUAV 的 IP/端口配置集中在一个字典中，便于日后扩展为 4 机、6 机甚至更多；
- 提供一个通用的 call(name, cmd) 接口，简化对单机的 JSON RPC 调用；
- 提供若干“群体操作”函数，例如广播 ping、广播起飞、轮询状态等；
- 提供简单的观测工具，例如 LOCAL_POSITION_NED 的采样和高度曲线监控。

4.2 vUAV 列表与调用接口

当前版本在脚本开头定义了一个简单的字典：

```
VUS = {
    "vuav1": ("127.0.0.1", 16001),
    "vuav2": ("127.0.0.1", 16002),
    "vuav3": ("127.0.0.1", 16003),
}
```

对应关系为：

- vuav1 → 宿主机 127.0.0.1:16001；
- vuav2 → 宿主机 127.0.0.1:16002；
- vuav3 → 宿主机 127.0.0.1:16003。

通用发送函数 send_cmd：

```
def send_cmd(addr, cmd, timeout=3.0):
    s = socket.create_connection(addr, timeout=timeout)
    s.sendall((json.dumps(cmd) + "\n").encode())
    data = s.recv(4096)
    s.close()
    return json.loads(data.decode("utf-8"))
```

在其基础上，call(name, cmd) 封装了错误处理与打印：

```
def call(name, cmd):
    addr = VUS[name]
```

```
try:
    resp = send_cmd(addr, cmd)
    print(f"[{name}] cmd={cmd} resp={resp}")
    return resp
except Exception as e:
    print(f"[{name}] ERROR sending {cmd}: {e}")
    return {"ok": False, "error": str(e)}
```

4.3 广播与监控函数

整体设计上：

- 广播函数：
  - `broadcast(cmd)`：对所有 vUAV 调用 `call`；
- 状态监控函数：
  - `poll_status_once()`：对所有 vUAV 调用 `get_status`，查看姿态；
  - `monitor_local_position(ticks, interval)`：周期性读取 `LOCAL_POSITION_NED` 并打印位置与速度全状态；
  - `monitor_altitude_curve(ticks, interval)`：只关注高度 alt $= -z$，输出高度时间序列与 min/max 汇总，用于“飞没飞起来”的快速检查；
- 控制函数：
  - `broadcast_takeoff(alt)`：对所有 vUAV 发送 `takeoff` 命令；
  - `monitor_local_position(ticks, interval)`：周期性读取 `LOCAL_POSITION_NED` 并打印位置与速度全状态；
  - `monitor_altitude_curve(ticks, interval)`：只关注高度 alt $= -z$，输出高度时间序列与 min/max 汇总，用于“飞没飞起来”的快速检查；
  - `broadcast_velocity_ned(vx, vy, vz, yaw_rate)`:通过代理的 `set_velocity_ned` 接口向所有 vUAV 下发统一速度指令（实验性接口，依赖 PX4 处于合适的 `OFFBOARD` 等模式）。
- 简单入口：
  - 在 `__main__` 中串联了：广播 ping → 轮询状态 → 广播 arm → 再次轮询 → 广播起飞 → 位置监控与高度曲线监控。

4.4 高度曲线监控 `monitor_altitude_curve`

`monitor_altitude_curve(ticks, interval)` 的逻辑是：

1) 对每个 vUAV 维护一个高度列表；
2) 每次循环对三机分别调用 `get_local_position`；
3) 从返回中取出 z，计算 alt $= -z$（NED 坐标系下 z 向下为正，这里取负号作为“向上高度”）；
4) 打印当前 tick 的高度，同时追加到各自的列表；
5) 所有 tick 完成后，对每个 vUAV 打印最小/最大高度。

这在实践中主要用来回答一个问题：“**虽然 PX4 接受了起飞命令，但它到底有没有真正离地？**”

5 当前一轮运行结果解读

5.1 SITL 初始化与 Preflight 告警

在 px4-vuav-agent:v5-20251204 中，PX4 SIH 启动日志大致为：

- 载入参数，初始化数据管理器 dataman；
- 初始化 SIH 仿真循环（250 Hz）；
- 启动多路 MAVLink 端口（包括与 14540 的本地 Onboard 链接）；
- Logger 启动并创建 .ulg 日志文件；
- 随后可以看到多条 Preflight 告警，例如：
  - `Preflight Fail: Accel 0 uncalibrated`
  - `Preflight Fail: barometer 0 missing`
  - `Preflight Fail: Found 0 compass (required: 1)`

这些告警说明当前配置下，PX4 认为部分传感器尚未完成标定/初始化，因此会阻止实际解锁/起飞（或者在 SITL 中表现为高度与姿态受限），这与后续看到的高度仅在厘米级波动相互印证。

5.2 群控脚本一轮完整输出的关键信息

在当前代码下，依次启动三个容器（vuav1, vuav2, vuav3），然后在宿主机执行：

```
python3 swarm_controller.py
```

观察到的典型输出结构（节选）大致为：

1) **广播 ping**：三机均返回 `"ok": true, "reply": "pong"`；
2) **状态轮询**：大部分机体的 get_status 返回正常 ATTITUDE，偶尔可能有个别出现 `"no ATTITUDE"`（启动初期尚未产生该消息）；
3) **广播 arm**：当前由于 Preflight 检查未通过，返回中 `"ok": false, "result": 1`；
4) **广播 takeoff**：
   - 对每台 vUAV，PX4 返回 COMMAND_ACK 且 result=0；
   - 表明命令层面 “起飞指令被接受”。
5) **LOCAL_POSITION_NED 位置监控**：
   - 对每台 vUAV，在多个 tick 中可以看到 x,y,z,vx,vy,vz 缓慢变化；
   - 例如 z 从约 -0.021 变化到约 -0.023~-0.025 等（单位米）。
6) **高度曲线监控 monitor_altitude_curve**：
   - 高度定义为 alt $= -z$；
   - 本轮运行中，三机的高度大致在 0.02 m to 0.03 m 左右波动；
   - 最终 summary 例如：

```
[vuav1] alt min=0.02 m, max=0.03 m
[vuav2] alt min=0.02 m, max=0.02 m
[vuav3] alt min=0.02 m, max=0.02 m
```

简要总结为：

- 从 **通信链路**视角：
  - agent 可以稳定 ping 通；

  - 状态/位置命令均正常返回;
  - 起飞命令被 PX4 接受（COMMAND_ACK result=0）。
- 从 **物理行为**视角:
  - LOCAL_POSITION_NED 中 x,y,z 随时间缓慢变化,同时高度监控 monitor_altitude_curve 给出的 alt $= -z$ 约在 0.02 m to 0.03 m 之间波动，证明“平台链路完整、仿真器位姿在演化”;
  - 但高度始终停留在厘米量级,结合 PX4 日志中 Preflight Fail: Accel 0 uncalibrated / barometer missing / Found 0 compass 等告警，可判断当前仍处于“未真正离地”的安全保护状态，需要后续补全传感器标定与解锁/模式切换（如 OFFBOARD）流程。

6 当前进展与下一步工作

6.1 当前进展小结

综合来看，当前平台已经完成:

- **容器层**:可稳定启动 px4-vuav-agent:v5-20251204 镜像的多实例,并通过端口映射暴露 agent 接口;
- **MAVLink 代理层**: vuav_agent.py 已能对接 PX4 心跳、发送 COMMAND_LONG、接收 ATTITUDE 与 LOCAL_POSITION_NED 等消息，并对上层暴露统一的 JSON 命令接口;
- **宿主机群控层**: swarm_controller.py 已可对三机进行统一的 ping、状态轮询、起飞命令分发、局部位置与高度曲线监控，并提供了速度控制广播的占位函数。

这些内容可以视作后续“可信计算保护下的无人机集群控制”的一块基石：只要能在宿主机中将控制逻辑封装为“对 agent 的 JSON 调用”，就能较为平滑地迁移到 TEE/密态环境中执行。

6.2 已知限制与改进方向

目前仍然存在的主要问题和限制包括:

- **PX4 预飞行检查未完全通过**:
  - 日志中存在加速度计未标定、气压计缺失、磁罗盘不存在等告警;
  - 这会限制解锁/起飞的行为，使得当前高度仅在厘米级范围内微调;
  - 需要在 SIH 配置或参数层面进一步排查与调整。
- **缺乏标准化 OFFBOARD 控制流程**:
  - 当前仅实现了基于 SET_POSITION_TARGET_LOCAL_NED 的速度控制占位接口(set_velocity_ned/broa
  - 尚未设计完整的“模式切换 → OFFBOARD 心跳 → 路径/速度命令序列”逻辑。
- **编队与任务层逻辑尚未引入**:
  - 目前更多是“多机单体控制”;
  - 尚未实现编队路径规划、碰撞检测、任务分配等高级功能。
- **控制接口仍然较基础:**已提供简单起飞、局部位置查询,以及基于 SET_POSITION_TARGET_LOCAL_NED 的速度控制占位接口和航点占位接口，但尚未集成闭环控制、路径规划、编队协同等更高级能力。

6.3 下一步工作建议（供后续迭代参考）

(A) **完善 PX4 传感器/参数配置**：

- 在 SIH 模式下，查阅官方文档与示例配置，尽可能消除 Preflight 告警；
- 确认解锁/起飞流程的正确配置，使高度曲线能真实反映“离地”状态。

(B) **系统化 OFFBOARD 流程**：

- 给 `vuav_agent.py` 增加模式切换命令（例如 `MAV_CMD_DO_SET_MODE` 或对应参数操作）；
- 在 `swarm_controller.py` 中封装“进入 OFFBOARD → 周期性发送 setpoint → 退出”的模式；
- 将 `broadcast_velocity_ned` 与 OFFBOARD 流程结合，形成一个可演示的“直线飞行”样例。

(C) **逐步引入编队与任务层逻辑**：

- 从最简单的“三机水平间隔飞行”开始，设计一个离散的参考轨迹；
- 逐步考虑碰撞避免、轨迹跟踪误差等指标；
- 为将来与 TEE 内的控制算法对接预留清晰的接口。

(D) **文档与教学材料完善**：

- 将当前文档整理成对学生/合作者友好的“快速上手指南”版本；
- 适当增加架构图（容器/agent/PX4/宿主机之间的关系）、数据流图等；
- 在后续迭代中逐渐补充更多“典型场景”示例。

本文档旨在对当前平台状态做一次阶段性“对账”，后续每次较大迭代（例如引入 OFFBOARD、编队控制、TEE 集成等）都可以在此基础上增量更新，形成一份持续演化的项目说明。