Atelier ROS 2 + MAVROS (avec ZenMav)

Suivi d'une cible mobile ("Ballon") et évaluation automatique

> **But** — Concevoir un nœud ROS 2 qui suit une cible mobile et démontrer la maîtrise des bases ROS 2 dans un contexte drone, en utilisant **MAVROS** (pont ROS 2↔MAVLink) et/ou des abstractions haut-niveau comme **ZenMav**. Le tout est évalué par un nœud moniteur fourni.

---

1) Ce que vous allez apprendre

- Architecture ROS 2 : **nœuds**, **topics**, **messages**, **QoS**, **callbacks**, **timers**, **paramètres**.
- Intégration drone : notions **MAVLink ↔ MAVROS**, frames **ENU/NED**, timestamps et horloge.
- Stratégies de suivi de cible simples (poursuite directe, lissage, limites), sans présumer d'une API unique.
- Lecture d'une **note** de performance émise automatiquement.

---

2) Les trois fichiers fournis (aperçu)

- **Générateur de cible ("Ballon")** : publie une pose cible qui évolue dans le temps et signale le début/fin d'une session.
- **Moniteur** : souscrit aux flux pertinents (cible + drone) et calcule des métriques (erreurs, cumulés, résumé). Sort un rapport (p. ex. CSV).
- **Solution d'exemple** : une implémentation minimale de suivi. **À consulter seulement après votre propre tentative.**

> Les noms de topics exacts, frames et détails d'implémentation sont visibles directement dans les fichiers.

---

3) Rappels ROS 2 essentiels (rclpy)

- **Nœud** : classe qui crée des **publishers**, **subscribers**, **timers** et **services**.
- **Publisher / Subscriber** : `create_publisher(Type, "…", queue)`, `create_subscription(Type, "…", callback, queue)`.
- **Callback** : fonction appelée à la réception d'un message ; évitez les calculs bloquants.
- **Timer** : `create_timer(période_s, callback)` pour une boucle d'asservissement périodique (p. ex. 20–50 Hz).
- **QoS** : choisissez une profondeur raisonnable (KEEP_LAST), préférer **Best Effort** pour flux rapides (IMU) et **Reliable** pour états lents.
- **Paramètres** : exposez des gains/vitesses/altitudes via `declare_parameter/get_parameter` pour itérer sans recompiler.
- **Horodatage** : utilisez les timestamps des messages (et le `dt`) pour la cohérence temporelle.

- **Frames** : ROS 2 côté MAVROS est généralement en **ENU** ; vérifiez vos conversions si votre contrôleur attend **NED**.

---

4) Couches drone : MAVROS, MAVLink et ZenMav

- **MAVROS** : pont ROS 2↔MAVLink. Il expose la télémétrie (pose, IMU, état) et des interfaces de commande (positions/vitesses/attitude) sous forme de topics/services/actions ROS 2.
- **MAVLink** : protocole bas niveau (messages, modes, armement, consignes).
- **ZenMav** : **option** haut-niveau (Python) qui encapsule des séquences courantes (mode/armement/consignes). Vous pouvez **tout** faire avec MAVROS seul, **ou** utiliser ZenMav pour simplifier—au choix du participant.

  L'atelier **n'impose pas** d'API de commande. Choisissez **MAVROS pur** ou **ZenMav** selon vos préférences.

---

5) Architecture de l'atelier (vue logique)

1. Un nœud **Ballon** publie périodiquement une *pose cible* dans une frame fixe (ex. `map`, ENU).
2. Votre nœud **Participant** souscrit cette pose et génère des commandes de guidage (position/vitesse/yaw…) via MAVROS **ou** ZenMav.
3. Un nœud **Moniteur** observe cible et drone, puis calcule une **note** sur une fenêtre temporelle définie.

---

6) Tâches à réaliser (dans l'ordre)

1. **Initialisation** : créer un nœud ROS 2 (rclpy), configurer publishers/subscribers, déclarer des paramètres (altitude cible, gains, limites de vitesse, etc.).
2. **Acquisition** : souscrire à la *pose cible* (Ballon) et à la *pose drone* ainsi qu'aux états utiles (IMU, état de connexion), selon ce qui est exposé par MAVROS.
3. **Référentiels & conversions** : valider frame_id, coordonnée Z (Up/Down), axes X/Y (Est/Nord). Implémenter une conversion ENU↔NED si nécessaire.
4. **Boucle de contrôle** : via un **timer** périodique, calculer la commande en utilisant la dernière cible reçue (poursuite directe, filtrage, limitation des accélérations/vitesses, gestion de dt).
5. **Commande** : publier la consigne appropriée (position locale, vitesse locale, etc.) par l'interface choisie (MAVROS ou ZenMav). Éviter les appels bloquants dans les callbacks ; préférez une file d'état traitée dans le timer.
6. **Déclenchement de session** : interagir avec les signaux fournis (début/fin) pour que votre nœud soit évalué au bon moment.
7. **Instrumentation** : ajouter des logs (INFO/WARN), éventuellement publier une distance estimée pour le débogage (optionnel).

---

7) Critères de réussite (général)

- **Exactitude** : distance moyenne au Ballon faible, dérive limitée.
- **Stabilité** : pas d'oscillations persistantes, commande bornée (vitesses/altitudes limitées), continuité des consignes.
- **Robustesse** : comportement raisonnable si la cible saute/ralentit/accélère ; gestion des messages manquants ponctuels.
- **Qualité ROS 2** : structure claire (timers vs callbacks), QoS adaptés, paramètres configurables, logs utiles.
- **Hygiène** : code lisible, commentaires brefs mais précis, séparation calcul/IO.

Le **Moniteur** exporte un rapport que vous utiliserez pour interpréter votre performance.

---

8) Conseils de conception

- **Découplage** : stockez les dernières mesures dans l'état du nœud et traitez-les dans la boucle timer.
- **Filtrage** : appliquez un lissage simple (p. ex. moyenne exponentielle) sur la cible si nécessaire.
- **Limites** : imposez des plafonds de vitesse/accélération et d'altitude. Prévoyez une zone de sécurité.
- **Temps** : calculez **dt** (à partir des horodatages) pour des lois dépendant de la vitesse.
- **Essais progressifs** : commencez par poursuite naïve, puis ajoutez lissage/anticipation au besoin.

---

9) Tests & validation (général)

- Vérifiez que la fréquence de votre **timer** est stable (journalisez la période effective).
- Inspectez les frames et les champs `header.stamp` des messages.
- Surveillez l'usage CPU et la latence de bout-en-bout (éviter les surcharges dans les callbacks).
- Interprétez les métriques du **Moniteur** pour guider vos ajustements.

---

10) Sécurité & bonnes pratiques

- **Simulation d'abord** : validez en simulateur avant tout essai matériel.
- **Limiteurs** : vitesse/altitude bornées, conditions d'armement/mode gérées proprement.
- **Arrêt sûr** : prévoyez un état "stop/safe" et des garde-fous si la cible devient invalide.

---

11) Pour aller plus loin (optionnel)

- Passage position→vitesse→accélération selon la disponibilité des interfaces MAVROS.
- Petites anticipations ($v \approx$ constante) et saturation douce.
- Paramétrage dynamique (reconfiguration) et traces supplémentaires dans le rapport.

Remarque finale

- Le détail exact des topics, frames et signaux d'orchestration est dans les **fichiers fournis** (Ballon, Moniteur).
- La **solution** est un *exemple* ; privilégiez votre propre conception avant de la consulter.