RAPPORT :

DRONE LIVREUR ROS2

Réalisé par :

- EL IDRISSI Aicha
- BOUZIANE Nouha
- EL-KASMI Hafsa

Sous l'encadrement de :

Mr. BELKEBIR Hicham

Mr. BELKEBIR Taoufik

I. Introduction :

Le projet 'Drone Livreur' implémenté en ROS2 vise à développer un drone autonome capable de transporter un colis d'un point A à une destination B tout en évitant les obstacles et en surveillant l'état de sa batterie.

Nous avons choisi d'utiliser ROS2 pour ses capacités avancées en termes de communication inter-nœuds, de modularité et de prise en charge des systèmes distribués. ROS2 offre également des outils puissants pour la gestion des capteurs et la synchronisation des données en temps réel, ce qui est essentiel pour un projet comme le nôtre. Quant à Python, son intégration fluide avec ROS2, sa syntaxe simple et ses vastes bibliothèques standards nous ont permis de développer des algorithmes complexes tout en réduisant le temps de développement.

II. Objectifs :

L'objectif principal est de permettre au drone de se déplacer efficacement d'un point à un autre tout en évitant les obstacles et en surveillant l'état de sa batterie. Pour ce plusieurs sous-objectifs ont été poses afin de l'achever :

1. **Hiérarchie contenant 3packages principaux :**
   - Package navigation : pour générer un chemin optimal évitant les obstacles
   - Package Delivery : pour Déplacer le drone.
   - Package PowerManagement : pour surveiller l'état de la batterie

2. **Conception et implémentation des nœuds ROS2 dans chaque package :**

3. **Mise en place d'actions et de services ROS2 :**
   - Créer un service ROS2 nommé « PlanPath » permettant à l'utilisateur de définir une destination en fournissant les coordonnées (x, y).
   - Intégrer des actions ROS2 pour la surveillance de la progression du chemin et pour notifier l'utilisateur en cas de problèmes ou de succès.

4. **Optimisation de la communication inter-nœuds :**
   - S'assurer que les nœuds échangent efficacement des données à l'aide des topics et services ROS2, notamment pour les capteurs et la génération de chemin

5. **Validation et simulation :**
   - Tester les nœuds et le service dans un environnement simulé via Gazebo pour valider la navigation autonome.

III. Intérêt :

Le projet "Drone Livreur Intégré" vise à concevoir un drone autonome capable de livrer des colis tout en gérant son parcours, son état de batterie et l'évitement d'obstacles. Ce projet présente un intérêt important pour l'automatisation des livraisons et l'innovation technologique.

Intérêt du projet :

1. **Réduction des coûts et des délais**
   Automatiser les livraisons pour plus d'efficacité, surtout dans les zones difficiles d'accès.

2. **Technologie de pointe**
   Utilisation de ROS2 et de capteurs embarqués pour un apprentissage pratique en robotique.

3. **Applications variées**
   Le projet peut être adapté pour des missions comme la surveillance ou l'assistance en urgence.
4. **Formation pratique**
   Développement de compétences en robotique, gestion des communications et utilisation de simulateurs comme Gazebo.
5. **Impact environnemental**

- En automatisant les livraisons avec des drones, on peut réduire l'empreinte carbone liée au transport, en remplaçant les véhicules traditionnels par des solutions aériennes électriques.

6. **Adaptabilité pour des projets réels**

- Ce projet n'est pas qu'un exercice académique ; il constitue une base pour des applications réelles, notamment dans les start-ups de livraison autonome

IV. Analyse :

Package NAVIGATION :

❖ Objectifs

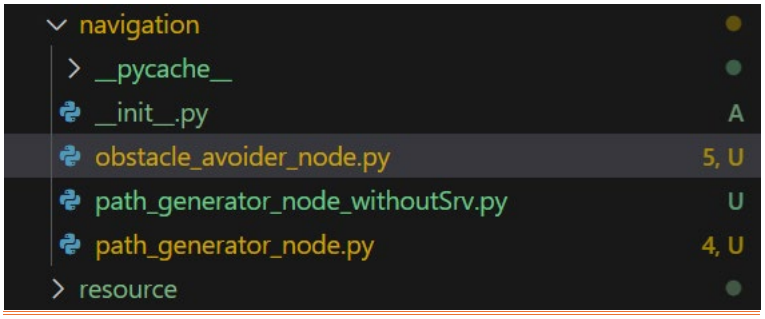
1. **Génération automatique de chemins** : Mettre en place un algorithme pour déterminer un chemin optimal reliant un point de départ à une destination prédéfinie.
2. **Évitement des obstacles** : Adapter dynamiquement le chemin initial en fonction des obstacles détectés par le capteur LiDAR du drone.

❖ Résultats attendus

- Un nœud de génération de chemins « ***path_generator_node*** » capable de publier un chemin valide et optimal en direction de la destination sélectionnée.
- Un nœud d'évitement d'obstacles « ***obstacle_avoider_node*** » capable de modifier le chemin initial en temps réel pour contourner les obstacles détectés.

❖ Analyse

→ **Hierchie du package navigation :**



➢ ***Path Generator Node :***

- Ce nœud est chargé de générer un chemin vers une destination sélectionnée et utilise les fonctions suivantes :
- **Service PlanPath** : Ce service permet à l'utilisateur de définir une destination en fournissant des coordonnées à travers une requête.

```
float64 x
float64 y
---
bool success
string message
```

- **Génération du chemin** : En fonction des coordonnées fournies, le nœud calcule un ensemble de points intermédiaires qui forment un chemin linéaire. Les poses générées sont publiées sur le topic *Generator_path*.

➢ ***Obstacle Avoider Node*** :

- Cette fonction est le constructeur de la classe **PathGeneratorNode**.
- Elle initialise le nœud ROS2 avec le nom path_generator_node.
  - **Souscription** :
    - Elle crée un subscriber pour le topic destination.
    - Ce subscriber attend des messages de type PoseStamped, représentant une destination cible.
    - Le callback associé est destination_callback.
  - **Publication** :
    - Un publisher est créé pour publier des messages de type Path sur le topic planned_path. Ces messages contiennent le chemin planifié.
- Deux variables sont initialisées :
  - **self.destination** pour stocker la destination reçue.
  - **self.timer** pour gérer les mises à jour régulières du chemin.
- Un message de log informe que le noeud est actif.

  - **Callback pour la destination**

  - **Fonction destination_callback**

Cette fonction est appelée automatiquement chaque fois qu'un message est reçu sur le topic destination. Elle extrait les coordonnées (x, y) de la destination depuis le message PoseStamped.

→Si la destination reçue est identique à la précédente, elle ignore la génération du chemin.

→La nouvelle destination est stockée dans self.destination.

→Si un timer existant est actif, il est annulé pour éviter les conflits. Un nouveau timer est créé pour appeler la fonction generate_path toutes les secondes.

- **Génération du chemin**

Fonction generate_path :

- Cette fonction génère un chemin linéaire entre l'origine et la destination actuelle.
- **Publication** : Le chemin complété est publié sur le topic planned_path.
- Un message de log confirme la publication réussie.

4. Fonction principale

Fonction main :

- Cette fonction exécute le nœud ROS2.
- **Arrêt** :
  - En cas d'interruption (CTRL+C), un message de log est affiché.
  - Le nœud est détruit et ROS2 est arrêté grâce à rclpy.shutdown().

❖ Tests :

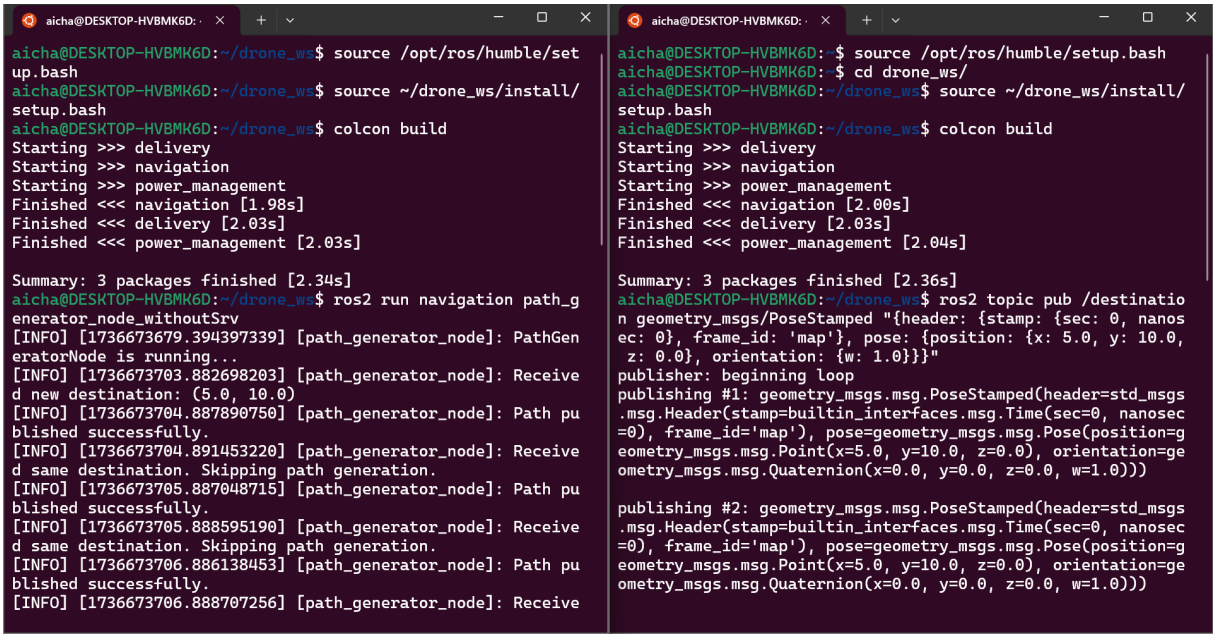
```
aicha@DESKTOP-HVBMK6D:~/drone_ws$ ros2 run navigation path_generator_node
Traceback (most recent call last):
  File "/home/aicha/drone_ws/install/navigation/lib/navigation/path_generator_node", line 33, in <module>
    sys.exit(load_entry_point('navigation==0.0.0', 'console_scripts', 'path_generator_node')())
  File "/home/aicha/drone_ws/install/navigation/lib/navigation/path_generator_node", line 25, in importlib_load_entry_point
    return next(matches).load()
  File "/usr/lib/python3.10/importlib/metadata/__init__.py", line 171, in load
    module = import_module(match.group('module'))
  File "/usr/lib/python3.10/importlib/__init__.py", line 126, in import_module
    return _bootstrap._gcd_import(name[level:], package, level)
  File "<frozen importlib._bootstrap>", line 1050, in _gcd_import
  File "<frozen importlib._bootstrap>", line 1027, in _find_and_load
  File "<frozen importlib._bootstrap>", line 1006, in _find_and_load_unlocked
  File "<frozen importlib._bootstrap>", line 688, in _load_unlocked
  File "<frozen importlib._bootstrap_external>", line 883, in exec_module
  File "<frozen importlib._bootstrap>", line 241, in _call_with_frames_removed
  File "/home/aicha/drone_ws/install/navigation/lib/python3.10/site-packages/navigation/path_generator_node.py", line 5, in
<module>
    from navigation.srv import PlanPath #importer le service PlanPath
ModuleNotFoundError: No module named 'navigation.srv'
[ros2run]: Process exited with failure 1
aicha@DESKTOP-HVBMK6D:~/drone_ws$
```

!!**Problématique rencontrée :** Malgré plusieurs tentatives, le service PlanPath n'a pas pu être utilisé dans le package navigation en raison d'une erreur indiquant l'absence du module navigation.srv.

Décision prise : Nous avons décidé d'éviter l'utilisation du service et d'explorer des alternatives afin de poursuivre le développement du projet.

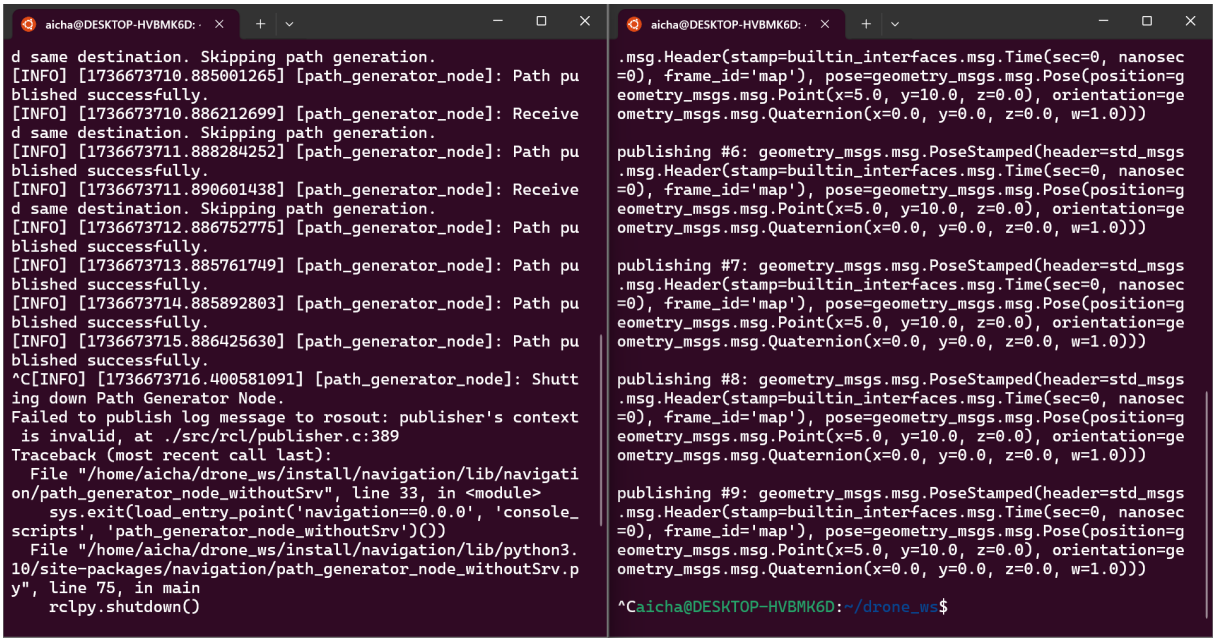
→ Implementer un noeud path_generated_node-withoutSrv

Test du premier nœud



```
aicha@DESKTOP-HVBMK6D:~/drone_ws$ source /opt/ros/humble/setup.bash
aicha@DESKTOP-HVBMK6D:~/drone_ws$ source ~/drone_ws/install/setup.bash
aicha@DESKTOP-HVBMK6D:~/drone_ws$ colcon build
Starting >>> delivery
Starting >>> navigation
Starting >>> power_management
Finished <<< navigation [1.98s]
Finished <<< delivery [2.03s]
Finished <<< power_management [2.03s]

Summary: 3 packages finished [2.34s]
aicha@DESKTOP-HVBMK6D:~/drone_ws$ ros2 run navigation path_generator_node_withoutSrv
[INFO] [1736673679.394397339] [path_generator_node]: PathGeneratorNode is running...
[INFO] [1736673703.882698203] [path_generator_node]: Received new destination: (5.0, 10.0)
[INFO] [1736673704.887890750] [path_generator_node]: Path published successfully.
[INFO] [1736673704.891453220] [path_generator_node]: Received same destination. Skipping path generation.
[INFO] [1736673705.887048715] [path_generator_node]: Path published successfully.
[INFO] [1736673705.888595190] [path_generator_node]: Received same destination. Skipping path generation.
[INFO] [1736673706.886138453] [path_generator_node]: Path published successfully.
[INFO] [1736673706.888707256] [path_generator_node]: Receive
```

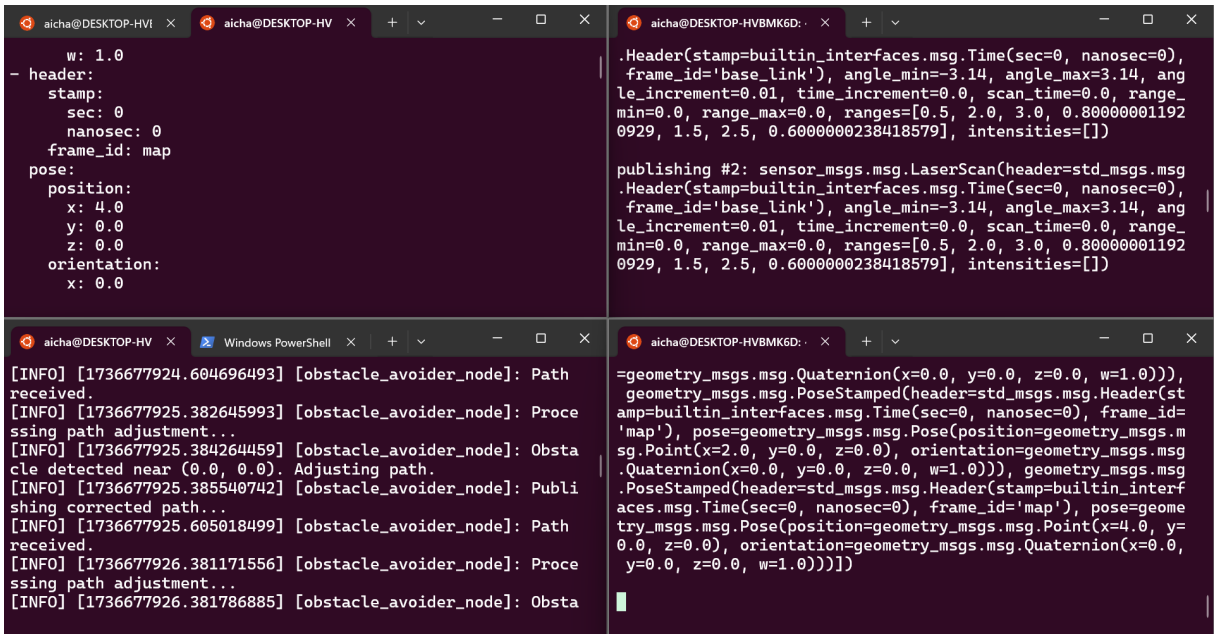
```
aicha@DESKTOP-HVBMK6D:~$ source /opt/ros/humble/setup.bash
aicha@DESKTOP-HVBMK6D:~$ cd drone_ws/
aicha@DESKTOP-HVBMK6D:~/drone_ws$ source ~/drone_ws/install/setup.bash
aicha@DESKTOP-HVBMK6D:~/drone_ws$ colcon build
Starting >>> delivery
Starting >>> navigation
Starting >>> power_management
Finished <<< navigation [2.00s]
Finished <<< delivery [2.03s]
Finished <<< power_management [2.04s]

Summary: 3 packages finished [2.36s]
aicha@DESKTOP-HVBMK6D:~/drone_ws$ ros2 topic pub /destination geometry_msgs/PoseStamped "{header: {stamp: {sec: 0, nanosec: 0}, frame_id: 'map'}, pose: {position: {x: 5.0, y: 10.0, z: 0.0}, orientation: {w: 1.0}}}"
publisher: beginning loop
publishing #1: geometry_msgs.msg.PoseStamped(header=std_msgs.msg.Header(stamp=builtin_interfaces.msg.Time(sec=0, nanosec=0), frame_id='map'), pose=geometry_msgs.msg.Pose(position=geometry_msgs.msg.Point(x=5.0, y=10.0, z=0.0), orientation=geometry_msgs.msg.Quaternion(x=0.0, y=0.0, z=0.0, w=1.0)))

publishing #2: geometry_msgs.msg.PoseStamped(header=std_msgs.msg.Header(stamp=builtin_interfaces.msg.Time(sec=0, nanosec=0), frame_id='map'), pose=geometry_msgs.msg.Pose(position=geometry_msgs.msg.Point(x=5.0, y=10.0, z=0.0), orientation=geometry_msgs.msg.Quaternion(x=0.0, y=0.0, z=0.0, w=1.0)))
```



```
d same destination. Skipping path generation.
[INFO] [1736673710.885001265] [path_generator_node]: Path published successfully.
[INFO] [1736673710.886212699] [path_generator_node]: Received same destination. Skipping path generation.
[INFO] [1736673711.888284252] [path_generator_node]: Path published successfully.
[INFO] [1736673711.890601438] [path_generator_node]: Received same destination. Skipping path generation.
[INFO] [1736673712.886752775] [path_generator_node]: Path published successfully.
[INFO] [1736673713.885761749] [path_generator_node]: Path published successfully.
[INFO] [1736673714.885892803] [path_generator_node]: Path published successfully.
[INFO] [1736673715.886425630] [path_generator_node]: Path published successfully.
^C[INFO] [1736673716.400581091] [path_generator_node]: Shutting down Path Generator Node.
Failed to publish log message to rosout: publisher's context is invalid, at ./src/rcl/publisher.c:389
Traceback (most recent call last):
  File "/home/aicha/drone_ws/install/navigation/lib/navigation/path_generator_node_withoutSrv", line 33, in <module>
    sys.exit(load_entry_point('navigation==0.0.0', 'console_scripts', 'path_generator_node_withoutSrv')())
  File "/home/aicha/drone_ws/install/navigation/lib/python3.10/site-packages/navigation/path_generator_node_withoutSrv.py", line 75, in main
    rclpy.shutdown()
```

```
.msg.Header(stamp=builtin_interfaces.msg.Time(sec=0, nanosec=0), frame_id='map'), pose=geometry_msgs.msg.Pose(position=geometry_msgs.msg.Point(x=5.0, y=10.0, z=0.0), orientation=geometry_msgs.msg.Quaternion(x=0.0, y=0.0, z=0.0, w=1.0)))

publishing #6: geometry_msgs.msg.PoseStamped(header=std_msgs.msg.Header(stamp=builtin_interfaces.msg.Time(sec=0, nanosec=0), frame_id='map'), pose=geometry_msgs.msg.Pose(position=geometry_msgs.msg.Point(x=5.0, y=10.0, z=0.0), orientation=geometry_msgs.msg.Quaternion(x=0.0, y=0.0, z=0.0, w=1.0)))

publishing #7: geometry_msgs.msg.PoseStamped(header=std_msgs.msg.Header(stamp=builtin_interfaces.msg.Time(sec=0, nanosec=0), frame_id='map'), pose=geometry_msgs.msg.Pose(position=geometry_msgs.msg.Point(x=5.0, y=10.0, z=0.0), orientation=geometry_msgs.msg.Quaternion(x=0.0, y=0.0, z=0.0, w=1.0)))

publishing #8: geometry_msgs.msg.PoseStamped(header=std_msgs.msg.Header(stamp=builtin_interfaces.msg.Time(sec=0, nanosec=0), frame_id='map'), pose=geometry_msgs.msg.Pose(position=geometry_msgs.msg.Point(x=5.0, y=10.0, z=0.0), orientation=geometry_msgs.msg.Quaternion(x=0.0, y=0.0, z=0.0, w=1.0)))

publishing #9: geometry_msgs.msg.PoseStamped(header=std_msgs.msg.Header(stamp=builtin_interfaces.msg.Time(sec=0, nanosec=0), frame_id='map'), pose=geometry_msgs.msg.Pose(position=geometry_msgs.msg.Point(x=5.0, y=10.0, z=0.0), orientation=geometry_msgs.msg.Quaternion(x=0.0, y=0.0, z=0.0, w=1.0)))

^Caicha@DESKTOP-HVBMK6D:~/drone_ws$
```

Test du deuxième nœud :



-Le chemin corrigé généré :

```
      x: 0.0
      y: 0.0
      z: 0.0
      w: 1.0
- header:
    stamp:
      sec: 0
      nanosec: 0
    frame_id: map
  pose:
    position:
      x: 2.0
      y: 0.0
      z: 0.0
    orientation:
      x: 0.0
      y: 0.0
      z: 0.0
      w: 1.0
- header:
    stamp:
      sec: 0
      nanosec: 0
    frame_id: map
  pose:
    position:
      x: 4.0
      y: 0.0
      z: 0.0
    orientation:
      x: 0.0
```

Package DELIVERY :

❖ Objectifs

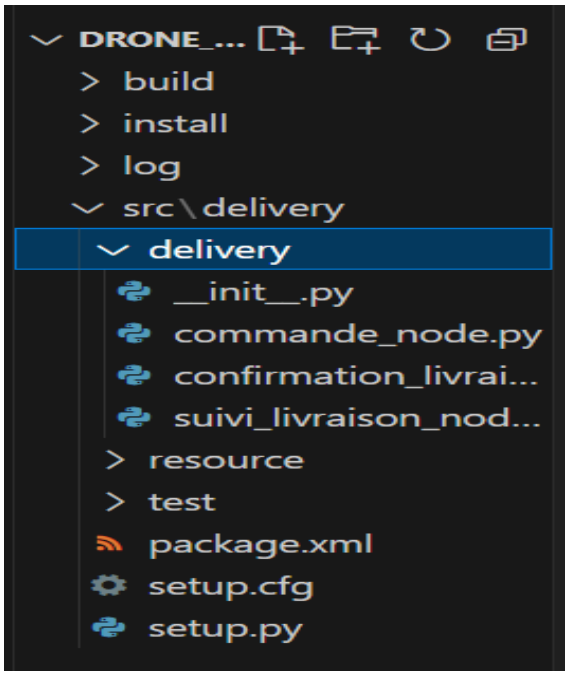
1. Gérer les commandes des utilisateurs pour initier le processus de livraison.
2. Suivre la progression du drone en temps réel pendant la mission de livraison.
3. Gérer la confirmation finale de la livraison et notifier l'utilisateur.

❖ Résultats attendus

- Un nœud « CommandeNode » pour publier les commandes sur un topic dédié.
- Un nœud « SuiviLivraisonNode » pour suivre la position du drone et surveiller sa progression.
- Un nœud « ConfirmationLivraisonNode » qui implémente un service pour la confirmation de la livraison.

❖ Analyse

→ Hierchie du package navigation :



➢ *CommandeNode :*

Fonctionnalités principales :

- Ce nœud permet de générer et publier des commandes toutes les 2 secondes.
- Le message publié inclut des détails sur la commande, comme le type de colis et l'objectif de livraison.
- **Topic** : commande.

Exemple d'utilisation :

- Une commande typique serait : "Nouvelle commande : Livraison de colis".

Avantages :

- Automatisation de la gestion des commandes pour initier les livraisons sans intervention manuelle continue.

Tests réalisés :

- Validation de la publication correcte sur le topic commande.
- Vérification des journaux pour s'assurer que les messages sont cohérents.

➢ *SuiviLivraisonNode :*

Fonctionnalités principales :

- Ce nœud s'abonne au topic position_drone pour recevoir des mises à jour de position sous forme de messages PoseStamped.
- À chaque mise à jour, il enregistre et affiche la position actuelle du drone dans les logs.

Exemple d'utilisation :

- "Position actuelle du drone : x=10.5, y=20.3".

Avantages :

- Permet un suivi précis en temps réel pour identifier rapidement les problèmes potentiels.

Tests réalisés :

- Simulation de positions aléatoires pour s'assurer que les données sont correctement reçues et affichées.

➢ ***ConfirmationLivraisonNode :***

Fonctionnalités principales :

- Implémente un service ROS2 nommé confirmer_livraison utilisant le type Trigger.
- Le service attend une requête et renvoie une réponse indiquant le succès ou l'échec de la livraison.
- Enregistre un log pour confirmer l'état de la livraison.

Exemple d'utilisation :

Avantages :

- Fournit une interaction claire pour la validation des missions complétées.

Tests réalisés :

Appel manuel et automatisé du service **confirmer_livraison** pour vérifier la réponse correcte et les logs associés.

❖ Tests :

Test du CommandeNode :

- Simulation de commandes répétées pour vérifier leur publication sur le topic **commande.**

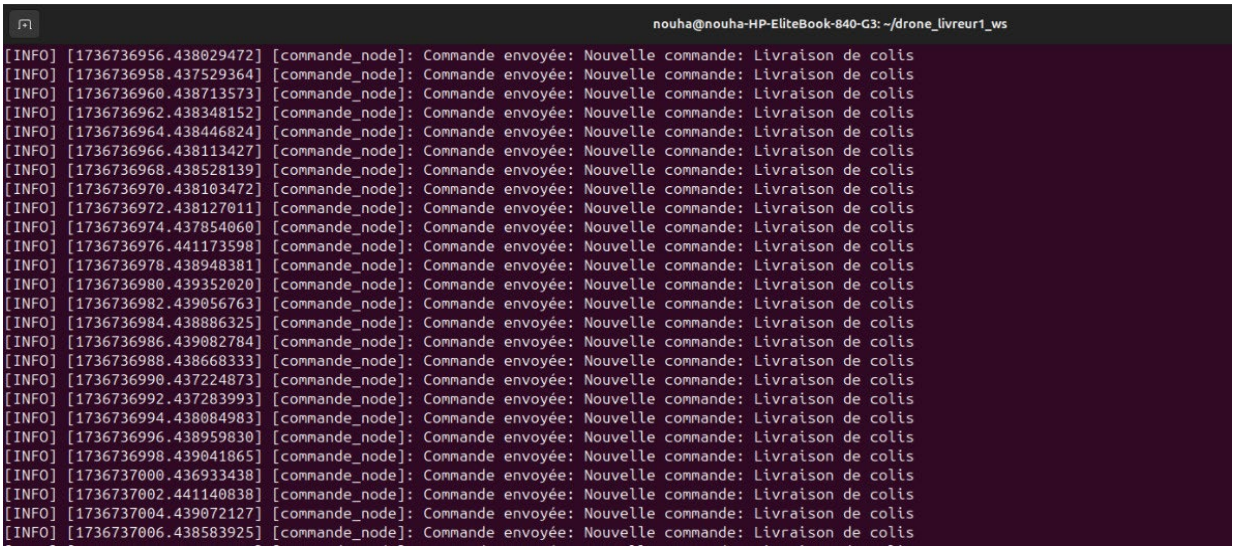
Test du SuiviLivraisonNode :

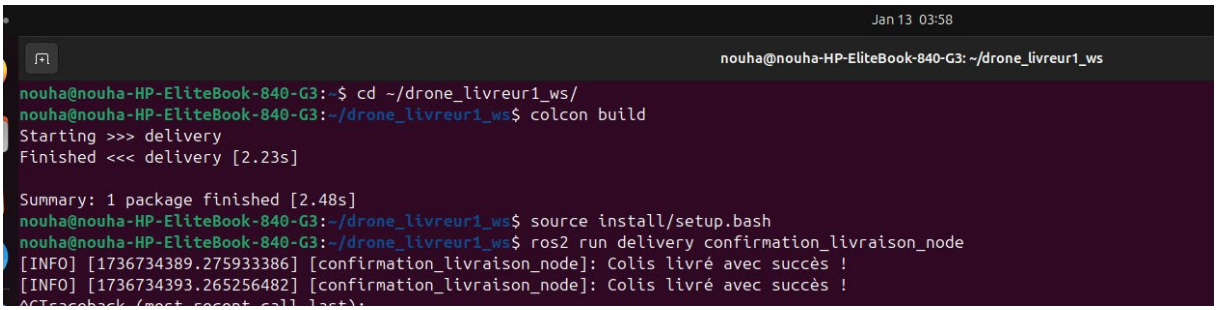
- Simulation de trajectoires en envoyant des positions au topic **position_drone.**

Test du ConfirmationLivraisonNode :

- Appels multiples au service **confirmer_livraison** pour évaluer sa robustesse et la précision des messages retournés.

❖ Visualisation :





```
Jan 13  03:59
nouha@nouha-HP-EliteBook-840-G3: ~/drone_livreur1_ws
nouha@nouha-HP-EliteBook-840-G3:~$ cd ~/drone_livreur1_ws/
nouha@nouha-HP-EliteBook-840-G3:~/drone_livreur1_ws$ colcon build
Starting >>> delivery
Finished <<< delivery [2.10s]

Summary: 1 package finished [2.30s]
nouha@nouha-HP-EliteBook-840-G3:~/drone_livreur1_ws$ source install/setup.bash
nouha@nouha-HP-EliteBook-840-G3:~/drone_livreur1_ws$ ros2 run delivery suivi_livraison_node
[INFO] [1736734307.999953332] [suivi_livraison_node]: En attente de la position du drone...
```

```
Jan 13  03:59
nouha@nouha-HP-EliteBook-840-G3: ~/drone_livreur1_ws
nouha@nouha-HP-EliteBook-840-G3:~$ cd ~/drone_livreur1_ws/
nouha@nouha-HP-EliteBook-840-G3:~/drone_livreur1_ws$ colcon build
Starting >>> delivery
Finished <<< delivery [2.21s]

Summary: 1 package finished [2.41s]
nouha@nouha-HP-EliteBook-840-G3:~/drone_livreur1_ws$ source install/setup.bash
nouha@nouha-HP-EliteBook-840-G3:~/drone_livreur1_ws$ ros2 topic echo /commande
data: 'Nouvelle commande: Livraison de colis'
---
data: 'Nouvelle commande: Livraison de colis'
---
data: 'Nouvelle commande: Livraison de colis'
---
data: 'Nouvelle commande: Livraison de colis'
---
data: 'Nouvelle commande: Livraison de colis'
---
data: 'Nouvelle commande: Livraison de colis'
---
data: 'Nouvelle commande: Livraison de colis'
---
data: 'Nouvelle commande: Livraison de colis'
---
data: 'Nouvelle commande: Livraison de colis'
---
```

```
ros2: command not found
nouha@nouha-HP-EliteBook-840-G3:~/drone_livreur1_ws$ ros2 service call /confirmer_livraison std_srvs/srv/Trigger
requester: making request: std_srvs.srv.Trigger_Request()

response:
std_srvs.srv.Trigger_Response(success=True, message='Colis livré avec succès !')
```

```
nouha@nouha-HP-EliteBook-840-G3:~/drone_livreur1_ws$ ros2 topic list
/commande
/parameter_events
/position_drone
/rosout
```

```
nouha@nouha-HP-EliteBook-840-G3:~/drone_livreur1_ws$ ros2 service  list
/commande_node/describe_parameters
/commande_node/get_parameter_types
/commande_node/get_parameters
/commande_node/get_type_description
/commande_node/list_parameters
/commande_node/set_parameters
/commande_node/set_parameters_atomically
/confirmation_livraison_node/describe_parameters
/confirmation_livraison_node/get_parameter_types
/confirmation_livraison_node/get_parameters
/confirmation_livraison_node/get_type_description
/confirmation_livraison_node/list_parameters
/confirmation_livraison_node/set_parameters
/confirmation_livraison_node/set_parameters_atomically
/confirmer_livraison
/suivi_livraison_node/describe_parameters
/suivi_livraison_node/get_parameter_types
/suivi_livraison_node/get_parameters
/suivi_livraison_node/get_type_description
/suivi_livraison_node/list_parameters
/suivi_livraison_node/set_parameters
/suivi_livraison_node/set_parameters_atomically
```

Package :PowerManagement:

❖ Objectifs :

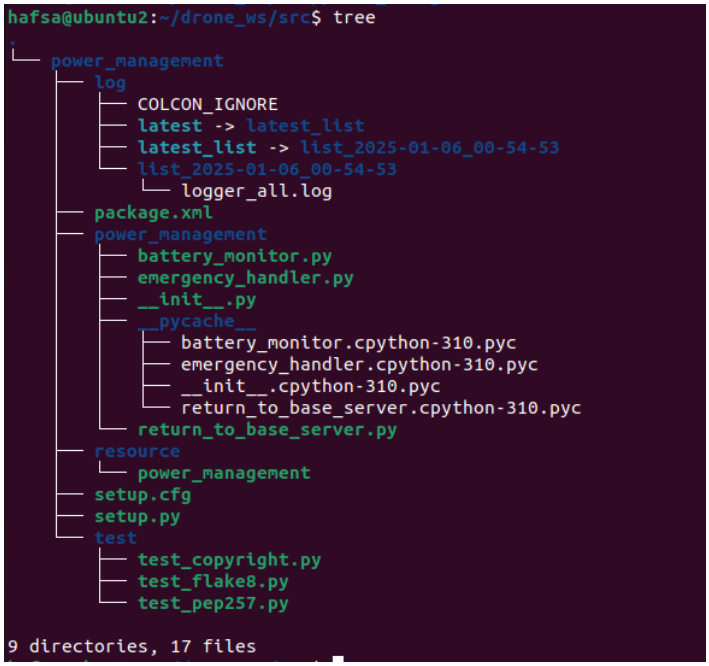
1. **Surveillance de l'état de la batterie**
   Mettre en place un système pour surveiller en temps réel le niveau de la batterie du drone à l'aide de capteurs.
2. **Gestion des urgences liées à la batterie faible**
   Implémenter une fonctionnalité permettant de déclencher un retour d'urgence au point de départ lorsque le niveau de batterie devient critique.

❖ Résultats attendus :

- Un nœud de surveillance de la batterie « **battery_monitor_node** » capable de publier le niveau de batterie en temps réel via un topic.
- Un nœud de gestion d'urgence « **emergency_handler_node** » qui souscrit aux informations de la batterie et déclenche un service pour un retour au point de départ lorsque la batterie atteint un seuil critique.

❖ Analyse :

→Arborescence du Workspace 'drone_ws :



- **Étapes de création de l'arborescence du Workspace 'drone_ws_pour power_management :**

1. Création du workspace

Créez un dossier principal pour workspace: **drone_ws.**

2. Création de la structure des répertoires

Dans le dossier drone_ws, créez un sous-dossier src, puis à l'intérieur de src, créez un autre sous-dossier appelé **power_management.**

3. Création des sous-dossiers dans power_management :

À l'intérieur de **power_management**, créez les sous-dossiers suivants :

log

resource

test

Créez également un fichier package.xml dans le répertoire **power_management.**

4. Création des fichiers Python dans power_management :

Dans le dossier **power_management**, créez un sous-dossier appelé **power_management** et placez-y les fichiers Python suivants :

battery_monitor.py

emergency_handler.py

init_.py

return_to_base_server.py

5. Création des fichiers de configuration :

Dans le dossier **power_management,** créez les fichiers suivants :

setup.cfg

setup.py

- **Explication des nodes :**

  - ✓ **battery_monitor.py**

- **Création d'un nœud :** Ce fichier crée un programme pour un drone qui surveille sa batterie et son état de livraison.
- **Publication des informations** :
  - Il envoie l'état de la batterie (tension, courant et pourcentage) sur un sujet appelé `battery_status`.
  - Il envoie aussi l'état de la livraison (par exemple, "Livraison en cours") sur un autre sujet appelé `delivery_status`.
- **Mise à jour régulière** : Chaque seconde, il met à jour l'état de la batterie et de la livraison grâce à un minuteur. Le programme génère un niveau de batterie aléatoire entre 0 et 100 % pour simuler l'évolution de la batterie.
- **Affichage dans les logs** : Chaque fois qu'il envoie des informations, il les affiche dans le journal pour les suivre.

  - ✓ Le programme de fichier **battery_monitor.py** aide à suivre l'état de la batterie d'un drone et son état de livraison, en envoyant ces informations toutes les secondes.

  - ✓ **return_to_base_server.py**

- **Création d'un serveur de service ROS2** : Ce fichier crée un programme qui agit comme un serveur pour le service `return_to_base`, permettant à d'autres parties du système de demander au drone de retourner à la base.
- **Création du service `return_to_base`** : Le programme crée un service appelé `return_to_base` qui attend une demande et renvoie une réponse. Ce service est de type `Trigger`, ce qui signifie qu'il n'a pas de paramètres complexes, juste une demande d'activation.
- **Gestion de la demande `return_to_base`** : Lorsque le service reçoit une demande, le programme envoie un message d'information dans les logs pour signaler que le retour à la base est activé.
- **Réponse du service** : Le programme répond à la demande en envoyant un message de succès avec la phrase "Retour à la base activé."

✓ le programme de fichier **return_to_base.py** permet à d'autres parties du système de demander au drone de retourner à la base et renvoie une confirmation lorsque le retour est activé.

✓ **emergency_handler.py**

✓ **Création d'un nœud ROS2** : Ce fichier crée un programme pour gérer les urgences du drone lorsqu'il y a un problème, comme une batterie faible.
✓ **Abonnement au sujet battery_status** : Le programme écoute en continu les informations sur l'état de la batterie envoyées par le nœud battery_monitor_node.py.
✓ **Vérification de la batterie faible** : Lorsque le niveau de la batterie est inférieur à 20 %, le programme détecte cela et déclenche une action d'urgence.
✓ **Appel du service return_to_base** : Lorsque la batterie est trop faible, il envoie une demande pour que le drone retourne à la base afin d'éviter une panne.
✓ **Gestion de la réponse** : Après avoir envoyé la demande, le programme attend la réponse pour savoir si le retour à la base a bien été activé. Il affiche un message de succès ou d'erreur selon la réponse.
✓ Le programme de fichier **emergency_handler.py** écoute l'état de la batterie du drone et, si celle-ci devient trop faible, il demande au drone de retourner à la base pour éviter une panne.

❖ Tests :

✓ Fonctionnement des nœuds :

Tout d'abord**, battery_monitor.py** doit être exécuté en premier, car il est chargé de surveiller l'état de la batterie du drone. Une fois qu'il est en cours d'exécution, il surveille constamment l'état de la batterie et attend qu'un problème survienne, comme une faible charge.

Si un problème est détecté (par exemple, une batterie faible), **battery_monitor.py** envoie un signal ou une notification à **emergency_handler.py** pour gérer la situation d'urgence. Cela

signifie qu'il faut d'abord que le service de surveillance de la batterie soit lancé avant que le gestionnaire d'urgence ne soit sollicité.

Le service de retour à la base **(return_to_base_server.py)** est ensuite appelé par **emergency_handler.py** lorsqu'une situation d'urgence est détectée. Ce dernier prendra alors le contrôle pour renvoyer le drone à la base en toute sécurité.

En résumé :

1- D'abord, on lance battery_monitor.py pour surveiller la batterie.

2- Ensuite, si la batterie devient faible, battery_monitor.py déclenche l'action dans emergency_handler.py.

3- Enfin, emergency_handler.py appelle return_to_base_server.py pour envoyer le drone à la base.

- **Test du premier nœud : battery_monitor.py**

```
hafsa@ubuntu2:~/drone_ws$ colcon build --packages-select power_management
Starting >>> power_management
Finished <<< power_management [9.49s]

Summary: 1 package finished [11.8s]
hafsa@ubuntu2:~/drone_ws$ source ~/drone_ws/install/setup.bash
hafsa@ubuntu2:~/drone_ws$ source /opt/ros/humble/setup.bash
hafsa@ubuntu2:~/drone_ws$ ros2 run power_management battery_monitor
[INFO] [1736719935.426246042] [battery_monitor]: Battery Status: 85.0%
[INFO] [1736719935.428089212] [battery_monitor]: Delivery Status: Delivery in progress
[INFO] [1736719936.357687730] [battery_monitor]: Battery Status: 83.0%
[INFO] [1736719936.359554654] [battery_monitor]: Delivery Status: Delivery in progress
[INFO] [1736719937.358008282] [battery_monitor]: Battery Status: 5.0%
[INFO] [1736719937.359926310] [battery_monitor]: Delivery Status: Delivery in progress
[INFO] [1736719938.358860779] [battery_monitor]: Battery Status: 52.0%
[INFO] [1736719938.363314774] [battery_monitor]: Delivery Status: Delivery in progress
[INFO] [1736719939.359839483] [battery_monitor]: Battery Status: 73.0%
[INFO] [1736719939.366039310] [battery_monitor]: Delivery Status: Delivery in progress
[INFO] [1736719940.359215606] [battery_monitor]: Battery Status: 19.0%
[INFO] [1736719940.376307278] [battery_monitor]: Delivery Status: Delivery in progress
[INFO] [1736719941.359198670] [battery_monitor]: Battery Status: 12.0%
[INFO] [1736719941.366371236] [battery_monitor]: Delivery Status: Delivery in progress
[INFO] [1736719942.357623953] [battery_monitor]: Battery Status: 41.0%
[INFO] [1736719942.359749773] [battery_monitor]: Delivery Status: Delivery in progress
[INFO] [1736719943.356814444] [battery_monitor]: Battery Status: 36.0%
[INFO] [1736719943.357947032] [battery_monitor]: Delivery Status: Delivery in progress
[INFO] [1736719944.358320386] [battery_monitor]: Battery Status: 86.0%
[INFO] [1736719944.361503486] [battery_monitor]: Delivery Status: Delivery in progress
[INFO] [1736719945.357417856] [battery_monitor]: Battery Status: 96.0%
[INFO] [1736719945.358948505] [battery_monitor]: Delivery Status: Delivery in progress
[INFO] [1736719946.357756437] [battery_monitor]: Battery Status: 9.0%
[INFO] [1736719946.361058371] [battery_monitor]: Delivery Status: Delivery in progress
[INFO] [1736719947.364522872] [battery_monitor]: Battery Status: 66.0%
[INFO] [1736719947.367152257] [battery_monitor]: Delivery Status: Delivery in progress
[INFO] [1736719948.358879030] [battery_monitor]: Battery Status: 61.0%
[INFO] [1736719948.362858973] [battery_monitor]: Delivery Status: Delivery in progress
[INFO] [1736719949.359866175] [battery_monitor]: Battery Status: 32.0%
[INFO] [1736719949.364405565] [battery_monitor]: Delivery Status: Delivery in progress
```

```
hafsa@ubuntu2:~/drone_ws$ colcon build --packages-select power_management
Starting >>> power_management
Finished <<< power_management [7.00s]

Summary: 1 package finished [8.09s]
hafsa@ubuntu2:~/drone_ws$ source ~/drone_ws/install/setup.bash
hafsa@ubuntu2:~/drone_ws$ source /opt/ros/humble/setup.bash
hafsa@ubuntu2:~/drone_ws$ ros2 run power_management return_to_base_server
[INFO] [1736720341.679126946] [return_to_base_server]: Service "return_to_base" appelé, retour à la base...
[INFO] [1736720499.372373659] [return_to_base_server]: Service "return_to_base" appelé, retour à la base...
[INFO] [1736720504.367690424] [return_to_base_server]: Service "return_to_base" appelé, retour à la base...
[INFO] [1736720507.368886213] [return_to_base_server]: Service "return_to_base" appelé, retour à la base...
[INFO] [1736720509.368911664] [return_to_base_server]: Service "return_to_base" appelé, retour à la base...
[INFO] [1736720512.376080191] [return_to_base_server]: Service "return_to_base" appelé, retour à la base...
[INFO] [1736720520.369357420] [return_to_base_server]: Service "return_to_base" appelé, retour à la base...
[INFO] [1736720521.367710313] [return_to_base_server]: Service "return_to_base" appelé, retour à la base...
[INFO] [1736720522.383868246] [return_to_base_server]: Service "return_to_base" appelé, retour à la base...
[INFO] [1736720527.367301395] [return_to_base_server]: Service "return_to_base" appelé, retour à la base...
[INFO] [1736720529.383025488] [return_to_base_server]: Service "return_to_base" appelé, retour à la base...
[INFO] [1736720538.391868462] [return_to_base_server]: Service "return_to_base" appelé, retour à la base...
[INFO] [1736720545.374531246] [return_to_base_server]: Service "return_to_base" appelé, retour à la base...
[INFO] [1736720550.366972518] [return_to_base_server]: Service "return_to_base" appelé, retour à la base...
[INFO] [1736720551.363522379] [return_to_base_server]: Service "return_to_base" appelé, retour à la base...
[INFO] [1736720554.379116410] [return_to_base_server]: Service "return_to_base" appelé, retour à la base...
```

- **Test du : return_to_base.py :**

```
hafsa@ubuntu2:~/drone_ws$ ros2 service list
/battery_monitor/describe_parameters
/battery_monitor/get_parameter_types
/battery_monitor/get_parameters
/battery_monitor/list_parameters
/battery_monitor/set_parameters
/battery_monitor/set_parameters_atomically
/return_to_base
/return_to_base_server/describe_parameters
/return_to_base_server/get_parameter_types
/return_to_base_server/get_parameters
/return_to_base_server/list_parameters
/return_to_base_server/set_parameters
/return_to_base_server/set_parameters_atomically
hafsa@ubuntu2:~/drone_ws$ ros2 service call /return_to_base std_srvs/srv/Trigger
waiting for service to become available...
requester: making request: std_srvs.srv.Trigger_Request()

response:
std_srvs.srv.Trigger_Response(success=True, message='Retour à la base activé.')

hafsa@ubuntu2:~/drone_ws$
```

Appel du service **/return_to_base** avec le type de service std_srvs/srv/Trigger. Une fois la commande envoyée, le système attend que le service soit disponible, puis il reçoit une réponse positive indiquant que l'action 'Retour à la base' a été activée avec succès. La réponse contient également un message de confirmation : 'Retour à la base activé'."

- **Test du :emergency_handler.py:**

----------L'activation du retour à la base

```
hafsa@ubuntu2:~/drone_ws$ colcon build --packages-select power_management
Starting >>> power_management
Finished <<< power_management [7.06s]

Summary: 1 package finished [8.24s]
hafsa@ubuntu2:~/drone_ws$ source ~/drone_ws/install/setup.bash
hafsa@ubuntu2:~/drone_ws$ source /opt/ros/humble/setup.bash
hafsa@ubuntu2:~/drone_ws$ ros2 run power_management emergency_handler
[INFO] [1736720477.490945250] [emergency_handler]: Service "return_to_base" non disponible, attente...
[INFO] [1736720478.495716054] [emergency_handler]: Service "return_to_base" non disponible, attente...
[WARN] [1736720499.367075978] [emergency_handler]: Batterie faible, envoi du signal pour retourner à la base...
[INFO] [1736720499.380754378] [emergency_handler]: Retour à la base activé avec succès.
[WARN] [1736720504.361265441] [emergency_handler]: Batterie faible, envoi du signal pour retourner à la base...
[INFO] [1736720504.387635748] [emergency_handler]: Retour à la base activé avec succès.
[WARN] [1736720507.363067288] [emergency_handler]: Batterie faible, envoi du signal pour retourner à la base...
[INFO] [1736720507.391358092] [emergency_handler]: Retour à la base activé avec succès.
[WARN] [1736720509.362848009] [emergency_handler]: Batterie faible, envoi du signal pour retourner à la base...
[INFO] [1736720509.391729848] [emergency_handler]: Retour à la base activé avec succès.
[WARN] [1736720512.364689313] [emergency_handler]: Batterie faible, envoi du signal pour retourner à la base...
[INFO] [1736720512.395885478] [emergency_handler]: Retour à la base activé avec succès.
```

V. Conclusion :

Le projet "Drone Livreur ROS2" a permis d'intégrer plusieurs packages clés pour répondre aux objectifs définis. Chaque package joue un rôle spécifique dans la réussite de la mission, en tirant parti de ROS2 et Python pour développer une solution modulaire, efficace et extensible.