Manuel d'intégration du code

1. Introduction

Ce manuel d'intégration s'adresse aux développeurs souhaitant comprendre, modifier ou étendre le projet Ornithoptère. Il présente l'architecture système, les composants principaux, et les points d'extension pour personnaliser le comportement de l'appareil.

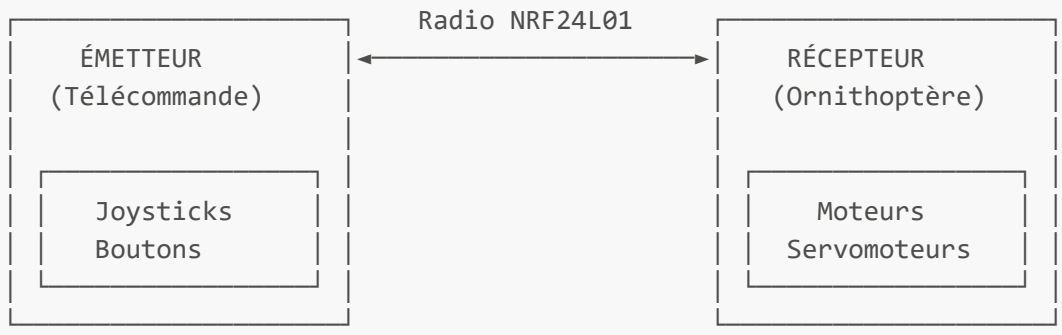
Le projet est conçu pour piloter un ornithoptère via des modules radio NRF24L01 et des microcontrôleurs ESP32/ESP8266 utilisant MicroPython.

2. Architecture système

2.1 Vue d'ensemble

Le système est composé de deux parties distinctes :

- **Radiocommande** : Télécommande avec joysticks et boutons qui capture les commandes utilisateur et les transmet via radio
- **Drone** : Il reçoit les commandes et contrôle les actionneurs



2.2 Structure des fichiers

```
source/
├── Communication
│   ├── antenne.py          # Abstraction NRF24L01 (émission/réception)
│   └── nrf24l01.py         # Driver bas niveau NRF24L01
├── Périphériques d'entrée
│   ├── joystick.py         # Classe Joystick (lecture ADC)
│   └── bouton.py           # Classe Bouton (lecture digitale)
├── Périphériques de sortie
│   ├── moteur.py           # Classe Moteur (PWM)
│   └── servomoteurs.py     # Classe ServoMoteur (PWM)
├── Utilitaires
│   └── logger.py           # Classe Logger (fichiers logs)
└── Programmes principaux
    ├── programme_antenne_emission.py    # Programme émetteur
    └── programme_antenne_reception.py   # Programme récepteur
```

3. Composants principaux

3.1 Communication radio (`antenne.py`)

Classe `Antenne`

- **Rôle** : Abstraction haut niveau de la communication radio
- **Modes** : `'émetteur'` ou `'récepteur'`
- **Méthodes principales** :
  - `send(message)` : Envoi de messages (dict, str, bytes)
  - `receive()` : Réception de messages avec décodage automatique JSON

Configuration par défaut :

```
# Broches SPI
sck_pin=18, mosi_pin=23, miso_pin=19
# Broches contrôle NRF24L01
ce_pin=26, csn_pin=27
# Communication
channel=76, address=b"2Node"
```

3.2 Périphériques d'entrée

Classe `Joystick` (`joystick.py`)

- **Méthodes** : `read()`, propriétés `x`, `y`, `bt`
- **Configuration** : broches ADC + bouton avec pull-up

Classe `Bouton` (`bouton.py`)

- **Méthodes** : `is_pressed()`, propriété `value`
- **Configuration** : broche digitale avec pull-up

3.3 Périphériques de sortie

Classe `Moteur` (`moteur.py`)

- **Contrôle** : PWM pour moteurs DC
- **Méthodes** : `set_speed(value)`, propriétés `speed`, `frequency`

Classe `ServoMoteur` (`servomoteurs.py`)

- **Contrôle** : PWM pour servomoteurs
- **Méthodes** : `angle(deg)`, propriétés `deg`, `frequency`

3.4 Utilitaires

Classe `Logger` (`logger.py`)

- **Méthodes** : `debug()`, `info()`, `warning()`, `error()`
- **Sortie** : Fichier texte avec horodatage

4. Configuration et déploiement

4.1 Scripts de déploiement

`deploy.sh` : Automatise le déploiement sur ESP32

- Sélection du programme (émission/réception)
- Détection automatique des ports USB
- Nettoyage et upload des fichiers
- Renommage en `main.py`

`start.sh` : Environnement de développement

- Session tmux avec 3 panneaux
- Terminaux série pour monitoring
- Navigation à la souris

4.2 Prérequis système

Voir le fichier `setup.md` pour l'installation des dépendances :

- Python 3 + venv
- mpfshell
- tmux, screen

5. Bonnes pratiques

5.1 Gestion d'erreurs

- Toujours vérifier les valeurs reçues avant utilisation
- Implémenter des timeouts pour les communications
- Ajouter des logs pour le débogage

5.2 Performance

- Maintenir une fréquence de communication constante
- Éviter les calculs longs dans les boucles principales
- Utiliser des interruptions pour les événements critiques

5.3 Sécurité

- Implémenter un mode "failsafe" en cas de perte de signal
- Limiter les valeurs de commande dans des plages sûres
- Ajouter des protections contre les commandes incohérentes

6. Extension et maintenance

6.1 Ajout de nouvelles fonctionnalités

1. **Identifier le composant concerné** (communication, capteur, actionneur)
2. **Créer une nouvelle classe** en suivant les patterns existants
3. **Modifier les programmes principaux** pour intégrer la fonctionnalité
4. **Mettre à jour la documentation** et les schémas de câblage
5. **Tester** sur le matériel cible

6.2 Débogage et dépannage

- Utiliser la classe `Logger` pour tracer l'exécution
- Vérifier les connexions matérielles avec le schéma pinout
- Tester les composants individuellement avant l'intégration
- Surveiller les logs des terminaux série avec `start.sh`