Guide de Développement GNSS : Implémentation Complète d'un Système Multi-Antennes avec Python

Table des Matières

---

1. Concepts Fondamentaux GNSS

1.1 Qu'est-ce que le GNSS ?

Le **Global Navigation Satellite System (GNSS)** est un système de géolocalisation par satellites. Pensez-y comme un GPS ultra-précis :

- **GPS** (États-Unis) : 31 satellites
- **GLONASS** (Russie) : 24 satellites
- **Galileo** (Europe) : 30 satellites
- **BeiDou** (Chine) : 35+ satellites

1.2 Principe de Fonctionnement

1. **Chaque satellite** émet en continu :
   - Son **position exacte** dans l'espace (éphémérides)
   - L'**heure précise** de transmission
   - Des **corrections d'horloge**
2. **Le récepteur GNSS** :
   - Reçoit ces signaux de 4+ satellites
   - Mesure le **temps de trajet** de chaque signal
   - Calcule sa **position** par triangulation

1.3 Types de Mesures GNSS

- **Code (pseudorange)** : Distance approximative satellite↔récepteur (~1-3m de précision)
- **Phase porteuse** : Mesure ultra-précise (~1-3mm) mais avec ambiguïté entière inconnue
- **Doppler** : Vitesse relative satellite↔récepteur

1.4 Positionnement Relatif vs Absolu

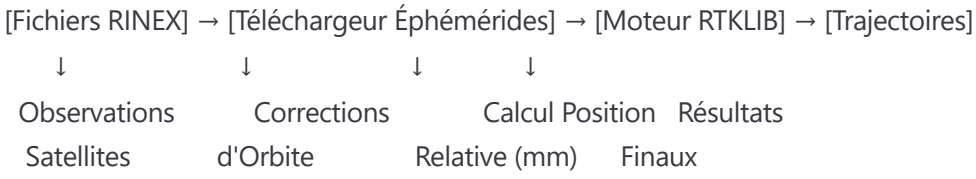
- **Absolu** : Position par rapport au centre de la Terre (précision ~1-5m)
- **Relatif** : Position d'une antenne par rapport à une autre (précision ~1-5mm)

Votre cas : Positionnement relatif entre 3 antennes espacées de 60-100m.

---

2. Architecture du Système

2.1 Vue d'Ensemble



2.2 Composants Principaux

A. **Gestionnaire de Fichiers RINEX**

- Lecture des fichiers d'observation (.obs)
- Validation de la qualité des données
- Extraction des informations temporelles

B. **Gestionnaire d'Éphémérides**

- Téléchargement automatique depuis serveurs IGS
- Gestion du cache local
- Sélection du type optimal (ultra-rapid/rapid/final)

C. **Moteur de Calcul RTKLIB**

- Configuration des paramètres de traitement
- Calcul des lignes de base entre antennes
- Résolution des ambiguïtés de phase

D. **Gestionnaire de Résultats**

- Export des trajectoires
- Calcul des statistiques de précision
- Visualisation des résultats

2.3 Structure de Classes Recommandée

python

```python
class GNSSProcessor:
    """Classe principale orchestrant tout le traitement"""

class RINEXManager:
    """Gestion des fichiers d'observation GNSS"""

class EphemerisDownloader:
    """Téléchargement automatique des éphémérides"""

class RTKLIBEngine:
    """Interface avec la bibliothèque RTKLIB"""

class ResultsAnalyzer:
    """Analyse et export des résultats"""
```

---

3. Installation et Configuration

3.1 Installation des Dépendances

Étape 1 : Installation de pyrtklib

bash

```bash
# Dans l'invite de commande (CMD) ou terminal
pip install pyrtklib
pip install numpy pandas matplotlib
pip install requests  # Pour téléchargement éphémérides
pip install hatanaka  # Pour décompression RINEX
```

Étape 2 : Vérification de l'installation

```python
# Test dans Spyder
import pyrtklib as prl
import numpy as np
import pandas as pd

print("pyrtklib version:", prl.__version__)
print("Installation réussie !")
```

3.2 Structure de Répertoires

```
mon_projet_gnss/
├── src/
│   ├── gnss_processor.py      # Classe principale
│   ├── rinex_manager.py       # Gestion RINEX
│   ├── ephemeris_manager.py   # Gestion éphémérides
│   └── rtklib_engine.py       # Interface RTKLIB
├── data/
│   ├── rinex/                 # Fichiers d'observation
│   ├── ephemeris/             # Corrections d'orbite
│   └── results/               # Résultats de traitement
├── config/
│   └── rtklib_config.conf     # Configuration RTKLIB
└── tests/
    └── test_data/             # Données de test
```

---

4. Formats de Données RINEX

4.1 Qu'est-ce que RINEX ?

RINEX (Receiver Independent Exchange Format) est le format standard pour les données GNSS.

4.2 Types de Fichiers RINEX

A. **Fichiers d'Observation (.obs)**

Contiennent les mesures de votre récepteur :

```
# Exemple de nom : STAT0010.24O
# STAT = nom station, 001 = jour de l'année, 0 = session, 24 = année, O = observation
```

Contenu typique :

- **C1C** : Pseudorange GPS L1

- **L1C** : Phase porteuse GPS L1
- **S1C** : Rapport signal/bruit GPS L1
- **D1C** : Doppler GPS L1

B. **Fichiers de Navigation (.nav)**

Contiennent les éphémérides diffusées :

```
# Exemple : BRDC0010.24N (éphémérides GPS du jour 001, année 2024)
```

4.3 Lecture des Fichiers RINEX avec pyrtklib

```
python
import pyrtklib as prl

# Lecture fichier observation
obs = prl.obs_t()
nav = prl.nav_t()

# Charger les données
ret = prl.readrnxt("STAT0010.24O", 1, "", obs, nav, None)
if ret > 0:
    print(f"Chargé {obs.n} observations")
else:
    print("Erreur lecture RINEX")
```

---

5. Implémentation Étape par Étape

5.1 Étape 1 : Classe de Base

python

```python
import pyrtklib as prl
import numpy as np
import pandas as pd
from datetime import datetime, timedelta
import os
import requests

class GNSSProcessor:
    """
    Processeur GNSS principal pour calcul de lignes de base multi-antennes

    Configuration : 3 antennes, 1 fixe + 2 mobiles
    Distance : 60-100m entre antennes
    Précision visée : millimétrique en relatif
    """

    def __init__(self, base_station_coords=None):
        """
        Initialisation du processeur

        Args:
            base_station_coords: [lat, lon, alt] de la station de référence
                                 Si None, sera calculé automatiquement
        """
        self.base_coords = base_station_coords
        self.antennas = {}  # Dictionnaire des antennes
        self.results = {}   # Résultats des calculs
        self.config = self._default_config()

    def _default_config(self):
        """Configuration par défaut optimisée pour lignes de base courtes"""
        return {
            'mode': 'static',          # Mode statique pour haute précision
            'frequencies': 'l1+l2',    # Utilisation L1+L2 si disponible
            'elevation_mask': 15,      # Masque d'élévation 15° (standard)
            'ionosphere': 'klobuchar', # Modèle ionosphérique basique
            'troposphere': 'saastamoinen', # Modèle troposphérique basique
            'ambiguity_resolution': 'fix-and-hold',  # Résolution d'ambiguïtés
            'phase_error': 0.003,      # Erreur de phase 3mm
            'code_error': 0.3,         # Erreur de code 30cm
        }
```

5.2 Étape 2 : Gestionnaire RINEX

python

```python
class RINEXManager:
    """Gestionnaire pour les fichiers RINEX"""

    def __init__(self, data_directory="./data/rinex/"):
        self.data_dir = data_directory
        os.makedirs(data_directory, exist_ok=True)

    def load_rinex_file(self, filepath):
        """
        Charge un fichier RINEX et retourne les structures RTKLIB

        Args:
            filepath: Chemin vers le fichier RINEX

        Returns:
            tuple: (obs, nav, success)
        """
        obs = prl.obs_t()
        nav = prl.nav_t()

        try:
            # Lecture du fichier RINEX
            ret = prl.readrnxt(filepath, 1, "", obs, nav, None)

            if ret > 0:
                print(f"✓ Fichier {filepath} chargé avec succès")
                print(f"  - {obs.n} observations")
                print(f"  - Période: {self._get_time_span(obs)}")
                return obs, nav, True
            else:
                print(f"✗ Erreur lecture {filepath}")
                return None, None, False

        except Exception as e:
            print(f"✗ Exception lecture {filepath}: {e}")
            return None, None, False

    def _get_time_span(self, obs):
        """Calcule la période temporelle des observations"""
        if obs.n > 0:
            start_time = prl.gpst2utc(obs.data[0].time)
            end_time = prl.gpst2utc(obs.data[obs.n-1].time)
            return f"{start_time} - {end_time}"
        return "Inconnue"

    def validate_rinex_quality(self, obs):
```

```
    """
    Valide la qualité des données RINEX

    Returns:
        dict: Statistiques de qualité
    """
    stats = {
        'total_epochs': obs.n,
        'satellites_count': {},
        'data_gaps': 0,
        'quality_ok': True
    }

    # Analyse par époque
    for i in range(obs.n):
        epoch = obs.data[i]
        # Compter les satellites par système
        for j in range(epoch.n):
            sat = epoch.data[j].sat
            sys = prl.satsys(sat, None)[0]
            sys_name = {prl.SYS_GPS: 'GPS',
                        prl.SYS_GLO: 'GLONASS',
                        prl.SYS_GAL: 'Galileo'}.get(sys, 'Autre')

            stats['satellites_count'][sys_name] = stats['satellites_count'].get(sys_name, 0) + 1

    # Validation minimale : au moins 4 satellites GPS par époque
    if stats['satellites_count'].get('GPS', 0) / obs.n < 4:
        stats['quality_ok'] = False

    return stats
```

5.3 Étape 3 : Gestionnaire d'Éphémérides

python

```python
class EphemerisManager:
    """Gestionnaire pour téléchargement automatique des éphémérides"""

    def __init__(self, cache_directory="./data/ephemeris/"):
        self.cache_dir = cache_directory
        os.makedirs(cache_directory, exist_ok=True)

        # URLs des serveurs d'éphémérides
        self.servers = {
            'igs': 'https://cddis.nasa.gov/archive/gnss/products/',
            'esa': 'https://navigation-office.esa.int/products/gnss-products/',
        }

    def get_ephemeris_for_date(self, date, ephemeris_type='rapid'):
        """
        Télécharge les éphémérides pour une date donnée

        Args:
            date: datetime object
            ephemeris_type: 'ultra-rapid', 'rapid', ou 'final'

        Returns:
            str: Chemin vers le fichier d'éphémérides
        """

        # Calcul de la semaine GPS et jour
        gps_week, gps_day = self._date_to_gps_week_day(date)

        # Nom du fichier selon le type
        if ephemeris_type == 'ultra-rapid':
            # Ultra-rapid : disponible 3h après l'observation
            filename = f"igu{gps_week}{gps_day}_00.sp3"
            latency_hours = 3
        elif ephemeris_type == 'rapid':
            # Rapid : disponible 17h après l'observation
            filename = f"igr{gps_week}{gps_day}_00.sp3"
            latency_hours = 17
        else:  # final
            # Final : disponible 13 jours après l'observation
            filename = f"igs{gps_week}{gps_day}_00.sp3"
            latency_hours = 13 * 24

        local_path = os.path.join(self.cache_dir, filename)

        # Vérifier si déjà en cache
        if os.path.exists(local_path):
```

```
            print(f"✓ Éphémérides trouvées en cache: {filename}")
            return local_path

        # Vérifier la disponibilité selon la latence
        now = datetime.utcnow()
        availability_time = date + timedelta(hours=latency_hours)

        if now < availability_time:
            print(f"⚠ Éphémérides {ephemeris_type} pas encore disponibles")
            print(f"  Disponibles dans {availability_time - now}")
            # Fallback vers type moins précis mais plus rapide
            if ephemeris_type == 'final':
                return self.get_ephemeris_for_date(date, 'rapid')
            elif ephemeris_type == 'rapid':
                return self.get_ephemeris_for_date(date, 'ultra-rapid')
            else:
                return None

        # Téléchargement
        return self._download_ephemeris(filename, gps_week, local_path)

    def _date_to_gps_week_day(self, date):
        """Convertit une date en semaine GPS et jour"""
        # Époque GPS : 6 janvier 1980
        gps_epoch = datetime(1980, 1, 6)
        delta = date - gps_epoch
        gps_week = int(delta.days // 7)
        gps_day = int(delta.days % 7)
        return gps_week, gps_day

    def _download_ephemeris(self, filename, gps_week, local_path):
        """Télécharge un fichier d'éphémérides"""

        # Construction de l'URL (format IGS)
        url = f"{self.servers['igs']}/{gps_week}/{filename}.Z"

        try:
            print(f"  Téléchargement {filename}...")
            response = requests.get(url, timeout=30)
            response.raise_for_status()

            # Sauvegarde du fichier compressé
            compressed_path = local_path + ".Z"
            with open(compressed_path, 'wb') as f:
                f.write(response.content)

            # Décompression (utiliser la bibliothèque appropriée)
```

```
            # Pour simplifier, on suppose qu'elle existe
            self._decompress_file(compressed_path, local_path)

            print(f"✓ Téléchargement réussi: {filename}")
            return local_path

        except Exception as e:
            print(f"✗ Erreur téléchargement {filename}: {e}")
            return None

    def _decompress_file(self, compressed_path, output_path):
        """Décompresse un fichier .Z (Unix compress)"""
        # Implémentation simplifiée - utiliser une vraie bibliothèque
        # comme python-lzw ou subprocess avec uncompress
        import subprocess
        try:
            subprocess.run(['uncompress', compressed_path], check=True)
            # Le fichier décompressé remplace automatiquement le .Z
        except:
            print("⚠ Décompression manuelle requise")
```

5.4 Étape 4 : Moteur RTKLIB

python

```python
class RTKLIBEngine:
    """Interface avec RTKLIB pour calcul de positionnement"""

    def __init__(self, config):
        self.config = config
        self.processing_options = self._setup_processing_options()

    def _setup_processing_options(self):
        """Configure les options de traitement RTKLIB"""
        opt = prl.prcopt_t()

        # Mode de positionnement
        opt.mode = prl.PMODE_STATIC  # Mode statique pour haute précision

        # Fréquences utilisées
        if self.config['frequencies'] == 'l1+l2':
            opt.nf = 2  # L1 + L2
        else:
            opt.nf = 1  # L1 seulement

        # Type de solution
        opt.soltype = prl.SOLTYPE_COMBINED  # Forward + backward

        # Résolution d'ambiguïtés
        opt.modear = prl.ARMODE_FIXHOLD  # Fix-and-hold
        opt.arthres[0] = 3.0  # Seuil validation ratio

        # Modèles d'erreur
        opt.ionoopt = prl.IONOOPT_BRDC    # Klobuchar
        opt.tropopt = prl.TROPOPT_SAAS    # Saastamoinen

        # Masque d'élévation
        opt.elmin = np.radians(self.config['elevation_mask'])

        # Erreurs de mesure
        opt.err[1] = self.config['phase_error']  # Phase (3mm)
        opt.err[2] = self.config['code_error']   # Code (30cm)

        # Pas de corrections de marées (inutile en relatif court)
        opt.tidecorr = 0

        return opt

    def process_baseline(self, obs_rover, obs_base, nav, baseline_name="baseline"):
        """
        Calcule une ligne de base entre station de référence et rover
```

```
    Args:
        obs_rover: Observations du rover
        obs_base: Observations de la base
        nav: Données de navigation
        baseline_name: Nom de la ligne de base

    Returns:
        dict: Résultats du traitement
    """

    print(f"🔄 Traitement ligne de base: {baseline_name}")

    # Préparation des structures de résultats
    solbuf = prl.solbuf_t()

    # Combinaison des observations
    obs_combined = prl.obs_t()

    # Copie des observations base
    for i in range(obs_base.n):
        prl.addobsdata(obs_combined, obs_base.data[i])

    # Copie des observations rover
    for i in range(obs_rover.n):
        prl.addobsdata(obs_combined, obs_rover.data[i])

    # Traitement RTKLIB
    ret = prl.postpos(0, 0, self.processing_options, solbuf, obs_combined, nav, None)

    if ret > 0:
        print(f"✓ Traitement réussi: {ret} solutions calculées")
        return self._extract_results(solbuf, baseline_name)
    else:
        print(f"✗ Échec du traitement")
        return None

def _extract_results(self, solbuf, baseline_name):
    """Extrait les résultats du buffer de solutions"""

    results = {
        'baseline_name': baseline_name,
        'solutions': [],
        'statistics': {}
    }

    # Extraction des solutions
```

```
    for i in range(solbuf.n):
        sol = solbuf.data[i]

        solution = {
            'time': prl.gpst2utc(sol.time),
            'position': [sol.rr[0], sol.rr[1], sol.rr[2]],  # X, Y, Z
            'quality': sol.stat,  # 1=fix, 2=float, 5=single
            'precision': [sol.qr[0], sol.qr[1], sol.qr[2]], # Écart-types
            'satellites': sol.ns,  # Nombre de satellites
            'age': sol.age,        # Âge des corrections
            'ratio': sol.ratio     # Ratio de validation
        }

        results['solutions'].append(solution)

    # Calcul des statistiques
    results['statistics'] = self._calculate_statistics(results['solutions'])

    return results

def _calculate_statistics(self, solutions):
    """Calcule les statistiques de qualité"""

    if not solutions:
        return {}

    # Filtre solutions de qualité
    fixed_solutions = [s for s in solutions if s['quality'] == 1]

    stats = {
        'total_epochs': len(solutions),
        'fixed_epochs': len(fixed_solutions),
        'fix_rate': len(fixed_solutions) / len(solutions) * 100,
        'mean_satellites': np.mean([s['satellites'] for s in solutions]),
        'mean_ratio': np.mean([s['ratio'] for s in fixed_solutions]) if fixed_solutions else 0,
        'precision_horizontal': 0,
        'precision_vertical': 0
    }

    # Calcul précision si solutions fixes disponibles
    if fixed_solutions:
        positions = np.array([s['position'] for s in fixed_solutions])

        # Conversion en coordonnées locales (approximation)
        # En pratique, utiliser une transformation géodésique complète
        east_std = np.std(positions[:, 0])
        north_std = np.std(positions[:, 1])
```

```
        up_std = np.std(positions[:, 2])

        stats['precision_horizontal'] = np.sqrt(east_std**2 + north_std**2)
        stats['precision_vertical'] = up_std

    return stats
```

---

6. Gestion des Éphémérides

6.1 Types d'Éphémérides et Choix Automatique

python

```python
def auto_select_ephemeris_type(observation_date):
    """
    Sélectionne automatiquement le meilleur type d'éphémérides
    selon la date d'observation et la disponibilité
    """
    now = datetime.utcnow()
    time_diff = now - observation_date

    if time_diff.days >= 13:
        # Final disponible (précision maximale)
        return 'final'
    elif time_diff.total_seconds() >= 17 * 3600:
        # Rapid disponible (bon compromis)
        return 'rapid'
    elif time_diff.total_seconds() >= 3 * 3600:
        # Ultra-rapid disponible (temps réel)
        return 'ultra-rapid'
    else:
        # Utiliser éphémérides diffusées dans RINEX
        return 'broadcast'
```

6.2 Validation des Éphémérides

python

```python
def validate_ephemeris_coverage(ephemeris_file, observation_start, observation_end):
    """
    Vérifie que les éphémérides couvrent la période d'observation

    Returns:
        bool: True si couverture complète
    """
    # Lecture du fichier SP3
    nav = prl.nav_t()
    ret = prl.readsp3(ephemeris_file, nav, 0)

    if ret == 0:
        return False

    # Vérification de la couverture temporelle
    # (Implémentation simplifiée)
    return True
```

---

7. Configuration RTKLIB

7.1 Fichier de Configuration

ini

```
# Configuration RTKLIB pour lignes de base courtes (60-100m)
# Précision millimétrique en mode relatif

# === POSITIONNEMENT ===
pos1-posmode     = static        # Mode statique haute précision
pos1-frequency   = l1+l2         # L1+L2 pour meilleure précision
pos1-soltype     = combined      # Solution combinée (forward+backward)
pos1-elmask      = 15            # Masque élévation 15°
pos1-snrmask_r   = off           # Pas de masque SNR
pos1-snrmask_b   = off
pos1-snrmask_L1  = 0,0,0,0,0,0,0,0
pos1-snrmask_L2  = 0,0,0,0,0,0,0,0
pos1-snrmask_L5  = 0,0,0,0,0,0,0,0

# === MODÈLES D'ERREUR ===
pos1-ionoopt     = brdc          # Klobuchar (suffisant pour courtes distances)
pos1-tropopt     = saas          # Saastamoinen (modèle basique)
pos1-sateph      = brdc          # Éphémérides diffusées (ou precise)
pos1-posopt1     = off           # Pas de station satellite
pos1-posopt2     = off           # Pas de correction marées
pos1-posopt3     = off           # Pas de correction phase center
pos1-posopt4     = off           # Pas de correction relativiste
pos1-posopt5     = off           # Pas de correction phase windup

# === RÉSOLUTION AMBIGUÏTÉS ===
pos2-armode      = fix-and-hold  # Fix-and-hold pour stabilité
pos2-gloarmode   = on            # Ambiguïtés GLONASS activées
pos2-bdsarmode   = on            # Ambiguïtés BeiDou activées
pos2-arfilter    = on            # Filtrage des ambiguïtés
pos2-arthres     = 3             # Seuil validation (ratio > 3)
pos2-arthres1    = 0.1           # Seuil instantané
pos2-arthres2    = 0             # Seuil lock count
pos2-arthres3    = 1e-09         # Seuil max résidus
pos2-arthres4    = 1e-05         # Seuil min satellites
pos2-varholdamb  = 0.1           # Variance hold ambiguïtés
pos2-gainholdamb = 0.01          # Gain hold ambiguïtés

# === DISTANCES DE BASE ===
pos2-baselen     = 0.1           # Distance max ligne de base (100m)
pos2-basesig     = 0.02          # Écart-type distance base

# === ERREURS DE MESURE ===
stats-errphase   = 0.003         # Erreur phase porteuse (3mm)
stats-errphaseel = 0.003         # Erreur phase élévation
stats-errphasebl = 0             # Erreur phase ligne de base
stats-errdoppler = 1             # Erreur Doppler (1 Hz)
```

```
stats-stdbias    = 30          # Biais standard code (30m)
stats-stdiono    = 0.03        # Écart-type ionosphère
stats-stdtrop    = 0.3         # Écart-type troposphère
stats-prnaccelh  = 1           # Accélération horizontale processus
stats-prnaccelv  = 0.1         # Accélération verticale processus
stats-prnbias    = 0.0001      # Biais récepteur processus
stats-prniono    = 0.001       # Ionosphère processus
stats-prntrop    = 0.0001      # Troposphère processus
stats-prnpos     = 0           # Position processus
stats-clkstab    = 5e-12       # Stabilité horloge

# === SOLUTION ===
out-solformat    = llh         # Format lat/lon/height
out-outhead      = on          # En-tête dans fichier sortie
out-outopt       = on          # Options dans fichier sortie
out-outvel       = off         # Pas de vitesse
out-timesys      = gpst        # Système temporel GPS
out-timeform     = tow         # Format temps (time of week)
out-timendec     = 3           # 3 décimales pour temps
out-degform      = deg         # Format degrés
out-fieldsep     =             # Séparateur de champs
out-outsingle    = off         # Pas de solution point unique
out-maxsolstd    = 0           # Écart-type max solution (0
```