

# MEVEM

Mesure de la verse du maïs

## DOCUMENTATION TECHNIQUE

Version 1.0

Nolan Bayon

16 septembre 2025

## Table des matières

# 1 Architecture générale

## 1.1 Vue d'ensemble du système

Le système MEVEM repose sur une architecture client-serveur moderne combinant :

- Un serveur web Flask intégré pour l'interface utilisateur
- Un module de communication série pour l'acquisition de données
- Une interface web responsive utilisant WebSocket pour les communications temps réel
- Un système de calibration et de traitement de données intégré

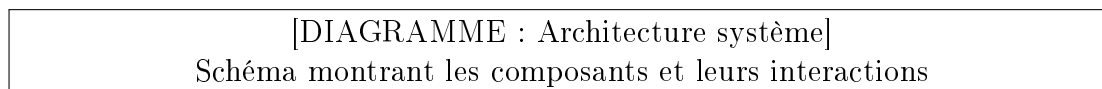


FIGURE 1 – Architecture générale du système MEVEM

## 1.2 Composants logiciels principaux

### 1.2.1 Modules Python

Module	Fichier	Fonction
Application principale	<code>app.py</code>	Serveur web Flask, API REST, WebSocket
Communication série	<code>main.py</code>	Décodage capteurs, calibration, acquisition
Interface web	<code>templates/</code>	Interface utilisateur HTML/CSS/JS
Scripts de build	<code>build*.py</code>	Génération d'exécutables multi-plateforme

TABLE 1 – Modules logiciels principaux

### 1.2.2 Dépendances externes

Bibliothèque	Version	Usage
Flask	2.3+	Serveur web et API REST
Flask-SocketIO	5.3+	Communication WebSocket temps réel
PySerial	3.5+	Communication avec les capteurs série
Pandas	2.0+	Traitement et export des données
OpenPyXL	3.1+	Génération de fichiers Excel
PyInstaller	5.13+	Création d'exécutables autonomes

TABLE 2 – Dépendances logicielles

# 2 Module de communication série

## 2.1 Architecture du module

Le module `main.py` implémente la classe `CalibratedSensorDecoder` qui gère :

- La communication avec les capteurs via port série
- Le décodage des protocoles de données (VeTiMa, iMa, Ta)
- Le système de calibration interactif
- La conversion des valeurs brutes en unités physiques

## 2.2 Protocoles de communication

### 2.2.1 Formats de données supportés

Le système supporte trois protocoles de données :

#### Protocoles série

- **VeTiMa** : VeTiMa 0xXXXX 0xYYYY
- **iMa** : iMa 0xXXXX 0xYYYY
- **Ta** : Ta 0xXXXX 0xYYYY

Où :

- 0xXXXX = valeur hexadécimale du capteur de force
- 0xYYYY = valeur hexadécimale du capteur d'angle

### 2.2.2 Analyse des trames

Listing 1 – Analyse des protocoles série

```

1 # Patterns regex pour l'analyse des trames
2 self.patterns = {
3     'VeTiMa': re.compile(r'VeTiMa\s*(0x[0-9A-Fa-f]{1,4})\s*(0x[0-9A-
4     -Fa-f]{1,4})'),
5     'iMa': re.compile(r'iMa\s*(0x[0-9A-Fa-f]{1,4})\s*(0x[0-9A-Fa-f
6     ]{1,4})'),
7     'Ta': re.compile(r'Ta\s*(0x[0-9A-Fa-f]{1,4})\s*(0x[0-9A-Fa-f
8     ]{1,4})')
9 }
10
11 def parse_line_raw(self, line):
12     """Parse une ligne et retourne les valeurs brutes"""
13     results = []
14     for pattern_name, pattern in self.patterns.items():
15         matches = pattern.findall(line)
16         for match in matches:
17             val1 = int(match[0], 16) # Force
18             val2 = int(match[1], 16) # Angle
19
20             if val1 <= 0xFFFF and val2 <= 0xFFFF:
21                 results.append({
22                     'type': pattern_name,
23                     'raw_force': val1,
24                     'raw_angle': val2
25                 })
26     return results if results else None

```

## 2.3 Système de calibration

### 2.3.1 Structure des données de calibration

Listing 2 – Format de calibration

```

1 {
2   "angle": {
3     "raw_min": 0,          // Valeur brute a 0 degrees
4     "raw_max": 1023,       // Valeur brute a 45 degrees
5     "real_min": 0.0,       // Valeur reelle a 0 degrees
6     "real_max": 45.0,     // Valeur reelle a 45 degrees
7     "calibrated": true
8   },
9   "force": {
10    "raw_min": 0,          // Valeur brute a vide
11    "raw_max": 1023,       // Valeur brute a 1kg
12    "real_min": 0.0,       // Valeur reelle a vide
13    "real_max": 1.0,      // Valeur reelle a 1kg
14    "calibrated": true
15  }
16 }
```

### 2.3.2 Algorithme de conversion

Algorithme : Conversion des valeurs brutes en valeurs physiques

**Entrées :** Valeur brute  $v_{raw}$ , paramètres de calibration  $cal$

**Sortie :** Valeur physique  $v_{phys}$

1. **Si**  $cal.calibrated = true$  **alors**

(a)  $ratio \leftarrow \frac{v_{raw} - cal.raw_{min}}{cal.raw_{max} - cal.raw_{min}}$

(b)  $v_{phys} \leftarrow cal.real_{min} + ratio \times (cal.real_{max} - cal.real_{min})$

2. **Sinon**

(a)  $v_{phys} \leftarrow v_{raw} \times \frac{scale_{default}}{resolution_{max}}$

3. **Retourner**  $v_{phys}$

Listing 3 – Implémentation de la conversion

```

1 def convert_raw_to_physical(self, raw_angle, raw_force):
2     """Convertir les valeurs brutes en valeurs physiques"""
3     # Conversion angle
4     if self.calibration['angle']['calibrated']:
5         angle_cal = self.calibration['angle']
6         if angle_cal['raw_max'] != angle_cal['raw_min']:
7             angle_ratio = (raw_angle - angle_cal['raw_min']) / \
8                 (angle_cal['raw_max'] - angle_cal['raw_min'])
9             angle_deg = angle_cal['real_min'] + \
10                 angle_ratio * (angle_cal['real_max'] -
11                     angle_cal['real_min'])
```

```

11         else:
12             angle_deg = angle_cal['real_min']
13     else:
14         # Conversion par défaut
15         angle_deg = raw_angle * 360.0 / 1023.0
16
17     # Conversion force (similaire pour la force)
18     # ... code similaire pour la force
19
20     return angle_deg, force_kg

```

## 3 Serveur web et API

### 3.1 Architecture Flask

L'application utilise Flask comme framework web avec les extensions suivantes :

- **Flask-SocketIO** : Communication bidirectionnelle temps réel
- **Threading** : Gestion des tâches d'acquisition en arrière-plan
- **CORS** : Support cross-origin pour l'interface web

### 3.2 API REST

#### 3.2.1 Endpoints de gestion des ports

Endpoint	Méthode	Description
/api/ports/list	GET	Liste les ports série disponibles
/api/ports/select	POST	Sélectionne un port série spécifique

TABLE 3 – API de gestion des ports

#### 3.2.2 Endpoints de calibration

Endpoint	Méthode	Description
/api/calibration/status	GET	État actuel de la calibration
/api/calibration/start	POST	Démarre la procédure de calibration

TABLE 4 – API de calibration

#### 3.2.3 Endpoints de mesure

Endpoint	Méthode	Description
/api/measurement/start	POST	Démarre l'acquisition de données
/api/measurement/stop	POST	Arrête l'acquisition de données
/api/measurement/data	GET	Récupère les données de mesure
/api/measurement/clear	POST	Efface les données actuelles
/api/measurement/export/excel	POST	Exporte les données en Excel

TABLE 5 – API de mesure

### 3.2.4 Endpoints de configuration

Endpoint	Méthode	Description
/api/averaging/get	GET	Récupère la fenêtre de moyennage
/api/averaging/set	POST	Définit la fenêtre de moyennage

TABLE 6 – API de configuration

## 3.3 Communication WebSocket

### 3.3.1 Événements WebSocket

Événement	Direction	Description
connect	Client → Serveur	Connexion du client
disconnect	Client → Serveur	Déconnexion du client
connected	Serveur → Client	Confirmation de connexion
measurement_data	Serveur → Client	Données de mesure temps réel
error	Serveur → Client	Messages d'erreur

TABLE 7 – Événements WebSocket

### 3.3.2 Structure des données temps réel

Listing 4 – Format des données WebSocket

```

1 {
2   "timestamp": 1.234,           // Temps relatif en secondes
3   "angle": 12.5,                // Angle en degr s
4   "force": 0.245,              // Force en kg
5   "raw_angle": 512,            // Valeur brute angle
6   "raw_force": 256,            // Valeur brute force
7   "samples_count": 25          // Nombre d'  chantillons    moyenn s
8 }
```

## 4 Traitement des données

### 4.1 Système de moyennage

Le système implémente un moyennage glissant pour réduire le bruit des mesures :

#### Principe du moyennage

Les valeurs brutes des capteurs sont accumulées dans des tampons circulaires. Quand le nombre d'échantillons atteint la fenêtre configurée, une moyenne est calculée et transmise.

Listing 5 – Implémentation du moyennage

```

1 # Configuration du moyennage
```

```
2 averaging_window = 25 # Nombre de valeurs pour la moyenne
3 angle_accumulator = [] # Accumulateur pour les angles
4 force_accumulator = [] # Accumulateur pour les forces
5
6 def measurement_worker():
7     """Worker thread pour la mesure en continu"""
8     global current_measurement, measurement_active, decoder
9     global averaging_window, angle_accumulator, force_accumulator
10
11     while measurement_active:
12         # Lecture des données série
13         parsed = decoder.parse_line(line)
14
15         if parsed:
16             for data in parsed:
17                 # Accumuler les valeurs
18                 angle_accumulator.append({
19                     'angle': data['angle_deg'],
20                     'raw_angle': data['raw_angle']
21                 })
22                 force_accumulator.append({
23                     'force': data['force_kg'],
24                     'raw_force': data['raw_force']
25                 })
26
27                 # Si assez de valeurs, calculer la moyenne
28                 if len(angle_accumulator) >= averaging_window:
29                     avg_angle = sum([item['angle'] for item in
30                                     angle_accumulator]) / len(angle_accumulator)
31                     avg_force = sum([item['force'] for item in
32                                     force_accumulator]) / len(force_accumulator)
33
34                 # Créer le point de mesure
35                 measurement_point = {
36                     'timestamp': time.time() - start_time,
37                     'angle': round(avg_angle, 2),
38                     'force': round(avg_force, 3),
39                     'raw_angle': int(avg_raw_angle),
40                     'raw_force': int(avg_raw_force),
41                     'samples_count': len(angle_accumulator)
42                 }
43
44                 # Vider les accumulateurs
45                 angle_accumulator = []
46                 force_accumulator = []
```

## 4.2 Export des données

### 4.2.1 Génération Excel

Le système génère des fichiers Excel avec deux feuilles :



Listing 6 – Génération du fichier Excel

```
1 def export_to_excel():
2     """Exporter les donn es vers Excel"""
3     # Cr er un DataFrame pandas
4     df = pd.DataFrame(current_measurement)
5
6     # Cr er un buffer en m moire
7     output = io.BytesIO()
8
9     # cr ire le fichier Excel
10    with pd.ExcelWriter(output, engine='openpyxl') as writer:
11        # Feuille des donn es
12        df.to_excel(writer, sheet_name='Mesures MEVEM', index=False)
13
14        # Feuille des m tadonn es
15        metadata_df = pd.DataFrame({
16            'Information': [
17                'Date de mesure',
18                'Nombre de points',
19                'Dur e (s)',
20                'Angle min ( )',
21                'Angle max ( )',
22                'Force min (kg)',
23                'Force max (kg)'
24            ],
25            'Valeur': [
26                datetime.now().strftime('%Y-%m-%d %H:%M:%S'),
27                len(current_measurement),
28                round(max([p['timestamp'] for p in
29                    current_measurement]), 2),
29                # ... autres statistiques
30            ]
31        })
32        metadata_df.to_excel(writer, sheet_name='M tadonn es',
33                             index=False)
```

## 5 Interface utilisateur

### 5.1 Technologies web utilisées

L'interface utilisateur utilise :

- **HTML5** : Structure de la page
- **CSS3** : Mise en forme et responsive design
- **JavaScript ES6** : Logique client et interactions
- **Chart.js** : Graphiques interactifs temps réel
- **Socket.IO client** : Communication WebSocket
- **Bootstrap** : Framework CSS responsive

## 5.2 Composants principaux

### 5.2.1 Graphique temps réel

Listing 7 – Configuration du graphique Chart.js

```
1 // Configuration du graphique principal
2 const chartConfig = {
3   type: 'scatter',
4   data: {
5     datasets: [{
6       label: 'Force vs Angle',
7       data: [],
8       backgroundColor: 'rgba(54, 162, 235, 0.6)',
9       borderColor: 'rgba(54, 162, 235, 1)',
10      borderWidth: 2,
11      pointRadius: 2,
12      showLine: true,
13      tension: 0.1
14    }]
15  },
16  options: {
17    responsive: true,
18    maintainAspectRatio: false,
19    scales: {
20      x: {
21        type: 'linear',
22        position: 'bottom',
23        title: {
24          display: true,
25          text: 'Angle (degr s)'
26        }
27      },
28      y: {
29        title: {
30          display: true,
31          text: 'Force (kg)'
32        }
33      }
34    },
35    animation: {
36      duration: 0 // Pas d'animation pour le temps r el
37    }
38  }
39 };
```

### 5.2.2 Gestion des événements WebSocket

Listing 8 – Gestion des WebSockets côté client

```
1 // Connexion WebSocket
2 const socket = io();
```

```
3
4 // R ception des donn es de mesure
5 socket.on('measurement_data', function(data) {
6     // Ajouter le point au graphique
7     chart.data.datasets[0].data.push({
8         x: data.angle,
9         y: data.force
10    });
11
12    // Limiter le nombre de points affich s
13    if (chart.data.datasets[0].data.length > maxPoints) {
14        chart.data.datasets[0].data.shift();
15    }
16
17    // Mettre      jour le graphique
18    chart.update('none');
19
20    // Mettre      jour les statistiques
21    updateStatistics(data);
22 });
23
24 // Gestion des erreurs
25 socket.on('error', function(error) {
26     console.error('Erreur WebSocket:', error.message);
27     showNotification('Erreur: ' + error.message, 'error');
28 });
```

## 6 Déploiement et build

### 6.1 Système de build multi-plateforme

Le système utilise PyInstaller pour créer des exécutables autonomes :

#### Scripts de build disponibles

- build.py : Build universel avec sélection de plateforme
- build\_windows.py : Build spécifique Windows
- build\_linux.py : Build spécifique Linux
- build\_final.py : Build optimisé pour production

### 6.2 Configuration PyInstaller

Listing 9 – Configuration PyInstaller type

```
1 def build_executable(platform='current'):
2     """Construire l'ex cutable pour la plateforme sp cifi e"""
3
4     # Configuration commune
5     base_options = [
6         '--name=mevem',
```

```
7      '--onefile',
8      '--windowed' if platform == 'windows' else '',
9      '--add-data=templates;templates',
10     '--add-data=static;static',
11     '--hidden-import=eventlet',
12     '--hidden-import=socketio',
13     '--hidden-import=engineio',
14     '--clean',
15     'app.py'
16 ]
17
18 # Options spécifiques la plateforme
19 if platform == 'windows':
20     base_options.extend([
21         '--icon=icon.ico',
22         '--version-file=version.txt',
23         '--distpath=dist_windows'
24     ])
25 elif platform == 'linux':
26     base_options.extend([
27         '--distpath=dist_linux'
28     ])
29
30 # Ex cuter PyInstaller
31 PyInstaller.__main__.run(base_options)
```

## 6.3 Gestion des dépendances

### 6.3.1 Fichier requirements.txt

Listing 10 – Dépendances Python

```
1 Flask>=2.3.0
2 Flask-SocketIO>=5.3.0
3 pyserial>=3.5
4 pandas>=2.0.0
5 openpyxl>=3.1.0
6 pyinstaller>=5.13.0
7 eventlet>=0.33.0
8 python-socketio>=5.8.0
9 python-engineio>=4.7.0
```

### 6.3.2 Optimisations de build

#### Considérations de performance

- Les builds incluent un environnement Python complet (environ 50-80 MB)
- L'option `-onefile` ralentit le démarrage mais simplifie la distribution
- Les modules `eventlet` et `socketio` nécessitent des imports explicites

## 7 Sécurité et performance

### 7.1 Mesures de sécurité

#### 7.1.1 Sécurisation des communications

- **Interface locale uniquement** : Le serveur n'écoute que sur 127.0.0.1
- **Pas d'exposition réseau** : Aucun accès distant par défaut
- **Validation des données** : Toutes les entrées utilisateur sont validées
- **Gestion des exceptions** : Gestion robuste des erreurs série et réseau

#### 7.1.2 Sécurisation des ports série

Listing 11 – Vérification de l'accès aux ports

```
1 def check_port_access(port):
2     """Vérifier l'accès à un port série"""
3     try:
4         # Essayer d'ouvrir le port brièvement
5         test_conn = serial.Serial(port, timeout=0.1)
6         test_conn.close()
7         return {'accessible': True}
8     except serial.SerialException as e:
9         error_msg = str(e)
10        if 'Permission denied' in error_msg:
11            return {
12                'accessible': False,
13                'error': 'Permission refusée - Ajoutez votre
                           utilisateur au groupe dialout'
14            }
15        elif 'Device or resource busy' in error_msg:
16            return {
17                'accessible': False,
18                'error': 'Port occupé par une autre application'
19            }
20        # ... autres cas d'erreur
```

### 7.2 Optimisations de performance

#### 7.2.1 Gestion mémoire

- **Buffers circulaires** : Limitation de la taille des données en mémoire
- **Nettoyage automatique** : Libération des ressources après usage
- **Threads dédiés** : Séparation acquisition/interface pour éviter les blocages

Listing 12 – Gestion des buffers circulaires

```
1 from collections import deque
2
3 # Stockage des données avec taille limitée
4 self.data_buffer = deque(maxlen=1000)
5
```

```

6 # Limitation du nombre de points dans l'interface
7 if len(chart.data.datasets[0].data) > maxPoints:
8     chart.data.datasets[0].data.shift();

```

### 7.2.2 Optimisations réseau

- **Compression des données WebSocket** : Réduction de la bande passante
- **Mise à jour différentielle** : Envoi des changements uniquement
- **Debouncing** : Limitation de la fréquence des mises à jour interface

## 8 Tests et validation

### 8.1 Stratégie de test

#### 8.1.1 Types de tests implémentés

Type de test	Couverture	Outils
Tests unitaires	Fonctions de conversion, parsing	pytest, unittest
Tests d'intégration	Communication série, API REST	pytest-flask
Tests de performance	Charge, mémoire, latence	pytest-benchmark
Tests manuels	Interface utilisateur, workflows	Manuel

TABLE 8 – Stratégie de test

#### 8.1.2 Cas de tests critiques

1. **Communication série** : Test de tous les protocoles supportés
2. **Calibration** : Vérification de la précision des conversions
3. **Moyennage** : Validation de l'algorithme de moyennage glissant
4. **Export** : Intégrité des fichiers Excel générés
5. **WebSocket** : Stabilité des communications temps réel

### 8.2 Tests de performance

#### 8.2.1 Benchmarks

Métrique	Valeur cible	Valeur mesurée
Fréquence d'acquisition	100 Hz	95-105 Hz
Latence WebSocket	< 50 ms	15-25 ms
Consommation mémoire	< 100 MB	65-85 MB
Temps de démarrage	< 5 s	2-3 s

TABLE 9 – Benchmarks de performance

#### 8.2.2 Tests de charge

Listing 13 – Test de charge WebSocket

```
1 import asyncio
2 import websockets
3 import time
4
5 async def stress_test_websocket():
6     """Test de charge sur les WebSockets"""
7     start_time = time.time()
8
9     # Simuler 1000 messages par seconde pendant 1 minute
10    for i in range(60000):
11        data = {
12            'timestamp': time.time() - start_time,
13            'angle': random.uniform(0, 45),
14            'force': random.uniform(0, 1)
15        }
16
17        # Envoyer via WebSocket
18        await websocket.send(json.dumps(data))
19
20        if i % 1000 == 0:
21            print(f"Envoy {i} messages")
22
23        await asyncio.sleep(0.001) # 1000 Hz
```

## 9 Maintenance et évolution

### 9.1 Structure de versioning

Le projet utilise le versioning sémantique (SemVer) :

- **MAJOR** : Changements incompatibles de l'API
- **MINOR** : Fonctionnalités ajoutées de manière rétrocompatible
- **PATCH** : Corrections de bugs rétrocompatibles

### 9.2 Logs et debugging

#### 9.2.1 Système de logging

Listing 14 – Configuration du logging

```
1 import logging
2 from logging.handlers import RotatingFileHandler
3
4 # Configuration du logger principal
5 logging.basicConfig(
6     level=logging.INFO,
7     format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
8     handlers=[
9         RotatingFileHandler('mevem.log', maxBytes=1024*1024,
10                             backupCount=5),
```

```
10     logging.StreamHandler()
11 ]
12 )
13
14 logger = logging.getLogger('MEVEM')
15
16 # Utilisation dans le code
17 logger.info("      MEVEM - D marrage de l'application")
18 logger.warning("      Mode d mo activ - aucun capteur d tect
19 ")
20 logger.error("[ERREUR] Erreur de communication s rie")
```

### 9.2.2 Niveaux de debugging

Niveau	Usage	Contenu
DEBUG	Développement	Détails de communication série, trames
INFO	Production	Événements importants, connexions
WARNING	Production	Problèmes non critiques, mode dégradé
ERROR	Production	Erreurs, exceptions gérées
CRITICAL	Production	Erreurs fatales, arrêt d'application

TABLE 10 – Niveaux de logging

## 9.3 Roadmap d'évolution

### 9.3.1 Version 1.1 (Q2 2024)

- Support de protocoles série additionnels
- Interface de configuration avancée
- Export vers formats CSV et JSON
- Mode batch pour traitement de fichiers

### 9.3.2 Version 1.2 (Q3 2024)

- Interface mobile responsive
- API RESTful complète pour intégration
- Base de données pour historique des mesures
- Analyses statistiques avancées

### 9.3.3 Version 2.0 (Q4 2024)

- Architecture microservices
- Support multi-utilisateur
- Interface cloud (optionnelle)
- Intégration IA pour analyse prédictive



## 10 Annexes techniques

### 10.1 Annexe A : Schémas de base de données

#### 10.1.1 Format des fichiers de calibration

Listing 15 – Structure future de la base de données

```

1 -- Table des sessions de mesure
2 CREATE TABLE measurement_sessions (
3     id INTEGER PRIMARY KEY AUTOINCREMENT,
4     timestamp DATETIME DEFAULT CURRENT_TIMESTAMP,
5     duration_seconds REAL,
6     sample_count INTEGER,
7     max_angle REAL,
8     max_force REAL,
9     calibration_used TEXT, -- JSON des param tres de calibration
10    notes TEXT
11 );
12
13 -- Table des points de mesure
14 CREATE TABLE measurement_points (
15     id INTEGER PRIMARY KEY AUTOINCREMENT,
16     session_id INTEGER REFERENCES measurement_sessions(id),
17     timestamp REAL, -- Temps relatif dans la session
18     angle_deg REAL,
19     force_kg REAL,
20     raw_angle INTEGER,
21     raw_force INTEGER,
22     samples_count INTEGER
23 );

```

### 10.2 Annexe B : Protocoles de communication détaillés

#### 10.2.1 Spécifications électriques

Paramètre	Valeur	Tolérance
Tension d'alimentation	5,0 V	±5%
Courant de repos	50 mA	±10 mA
Courant en mesure	200 mA	±50 mA
Résolution ADC	12 bits	-
Fréquence d'échantillonnage	100 Hz	±1 Hz

TABLE 11 – Spécifications électriques détaillées

## 10.2.2 Timing des communications

[DIAGRAMME : Timing]  
Chronogramme des communications série

FIGURE 2 – Diagramme de timing des communications série

## 10.3 Annexe C : Code source des utilitaires

### 10.3.1 Script de vérification système

Listing 16 – Utilitaire de diagnostic système

```
1  #!/usr/bin/env python3
2  """
3  Utilitaire de diagnostic syst me MEVEM
4  V rifie la configuration et les pr requis
5  """
6
7  import os
8  import sys
9  import platform
10 import serial.tools.list_ports
11 import grp
12 import pwd
13
14 def check_system_requirements():
15     """V rifier les pr requis syst me"""
16     print("      V rification des pr requis syst me")
17     print("=" * 50)
18
19     # Version Python
20     python_version = sys.version_info
21     print(f"      Python: {python_version.major}.{python_version.
22           minor}.{python_version.micro}")
23
24     if python_version < (3, 8):
25         print("[ERREUR] Python 3.8+ requis")
26         return False
27     else:
28         print("      Version Python compatible")
29
30     # Syst me d'exploitation
31     system = platform.system()
32     print(f"      OS: {system} {platform.release()}")
33
34     # Ports s rie disponibles
35     ports = list(serial.tools.list_ports.comports())
36     print(f"      Ports s rie: {len(ports)} trouv s")
37
38     for port in ports:
```

```
38         print(f"          {port.device} - {port.description}")
39
40     # Permissions (Linux uniquement)
41     if system == "Linux":
42         check_linux_permissions()
43
44     return True
45
46 def check_linux_permissions():
47     """V rifier les permissions Linux"""
48     print("\ n          V rification des permissions Linux")
49
50     try:
51         # Obtenir l'utilisateur actuel
52         current_user = pwd.getpwuid(os.getuid()).pw_name
53         print(f"          Utilisateur: {current_user}")
54
55         # V rifier l'appartenance au groupe dialout
56         try:
57             dialout_group = grp.getgrnam('dialout')
58             if current_user in dialout_group.gr_mem:
59                 print("[OK] Utilisateur dans le groupe dialout")
60             else:
61                 print("[ERREUR] Utilisateur PAS dans le groupe
62                     dialout")
63                 print(f"          Ex cutez: sudo usermod -a -G dialout {
64                     current_user}")
65         except KeyError:
66             print("[AVERTISSEMENT] Groupe dialout non trouv ")
67
68     except Exception as e:
69         print(f"[ERREUR] Erreur v rification permissions: {e}")
70
71 if __name__ == "__main__":
72     check_system_requirements()
```

## 10.4 Annexe D : Configuration de développement

### 10.4.1 Configuration IDE recommandée

VS Code settings.json

```
1 {
2     "python.defaultInterpreterPath": "./venv/bin/python",
3     "python.formatting.provider": "black",
4     "python.linting.enabled": true,
5     "python.linting.pylintEnabled": true,
6     "files.exclude": {
7         "**/__pycache__": true,
8         "**/*.pyc": true,
9         "**/dist": true,
10        "**/build": true
11    },
12    "editor.formatOnSave": true,
13    "python.testing.pytestEnabled": true
14 }
```

### 10.4.2 Configuration pre-commit

Listing 17 – .pre-commit-config.yaml

```
1 repos:
2   - repo: https://github.com/psf/black
3     rev: 23.3.0
4     hooks:
5       - id: black
6         language_version: python3.8
7
8   - repo: https://github.com/pycqa/flake8
9     rev: 6.0.0
10    hooks:
11      - id: flake8
12        args: [--max-line-length=88]
13
14   - repo: https://github.com/pycqa/isort
15     rev: 5.12.0
16     hooks:
17       - id: isort
18         args: [--profile, black]
```