LOGO MEVEM

MEVEM

Mesure de la verse du maïs

DOCUMENTATION TECHNIQUE

Version 1.0

Équipe de développement MEVEM

4 septembre 2025

Table des matières

1 Architecture générale **4**
1.1 Vue d'ensemble du système . . . 4
1.2 Composants logiciels principaux . . . 4
1.2.1 Modules Python . . . 4
1.2.2 Dépendances externes . . . 4

2 Module de communication série **4**
2.1 Architecture du module . . . 4
2.2 Protocoles de communication . . . 5
2.2.1 Formats de données supportés . . . 5
2.2.2 Analyse des trames . . . 5
2.3 Système de calibration . . . 6
2.3.1 Structure des données de calibration . . . 6
2.3.2 Algorithme de conversion . . . 6

3 Serveur web et API **7**
3.1 Architecture Flask . . . 7
3.2 API REST . . . 7
3.2.1 Endpoints de gestion des ports . . . 7
3.2.2 Endpoints de calibration . . . 7
3.2.3 Endpoints de mesure . . . 7
3.2.4 Endpoints de configuration . . . 8
3.3 Communication WebSocket . . . 8
3.3.1 Événements WebSocket . . . 8
3.3.2 Structure des données temps réel . . . 8

4 Traitement des données **8**
4.1 Système de moyennage . . . 8
4.2 Export des données . . . 9
4.2.1 Génération Excel . . . 9

5 Interface utilisateur **10**
5.1 Technologies web utilisées . . . 10
5.2 Composants principaux . . . 11
5.2.1 Graphique temps réel . . . 11
5.2.2 Gestion des événements WebSocket . . . 11

6 Déploiement et build **12**
6.1 Système de build multi-plateforme . . . 12
6.2 Configuration PyInstaller . . . 12
6.3 Gestion des dépendances . . . 13
6.3.1 Fichier requirements.txt . . . 13
6.3.2 Optimisations de build . . . 13

7 Sécurité et performance **14**
7.1 Mesures de sécurité . . . 14
7.1.1 Sécurisation des communications . . . 14
7.1.2 Sécurisation des ports série . . . 14
7.2 Optimisations de performance . . . 14
7.2.1 Gestion mémoire . . . 14
7.2.2 Optimisations réseau . . . 15

8 Tests et validation **15**
8.1 Stratégie de test . . . 15
8.1.1 Types de tests implémentés . . . 15
8.1.2 Cas de tests critiques . . . 15
8.2 Tests de performance . . . 15
8.2.1 Benchmarks . . . 15
8.2.2 Tests de charge . . . 15

9 Maintenance et évolution **16**
9.1 Structure de versioning . . . 16
9.2 Logs et debugging . . . 16
9.2.1 Système de logging . . . 16
9.2.2 Niveaux de debugging . . . 17
9.3 Roadmap d'évolution . . . 17
9.3.1 Version 1.1 (Q2 2024) . . . 17
9.3.2 Version 1.2 (Q3 2024) . . . 17
9.3.3 Version 2.0 (Q4 2024) . . . 17

10 Annexes techniques **18**
10.1 Annexe A : Schémas de base de données . . . 18
10.1.1 Format des fichiers de calibration . . . 18
10.2 Annexe B : Protocoles de communication détaillés . . . 18
10.2.1 Spécifications électriques . . . 18
10.2.2 Timing des communications . . . 19
10.3 Annexe C : Code source des utilitaires . . . 19
10.3.1 Script de vérification système . . . 19
10.4 Annexe D : Configuration de développement . . . 21
10.4.1 Configuration IDE recommandée . . . 21
10.4.2 Configuration pre-commit . . . 21

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

[DIAGRAMME : Architecture système]
Schéma montrant les composants et leurs interactions

FIGURE 1 – Architecture générale du système MEVEM

1.2 Composants logiciels principaux

1.2.1 Modules Python

Module	Fichier	Fonction
Application principale	`app.py`	Serveur web Flask, API REST, WebSocket
Communication série	`main.py`	Décodage capteurs, calibration, acquisition
Interface web	`templates/`	Interface utilisateur HTML/CSS/JS
Scripts de build	`build*.py`	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

```
# Patterns regex pour l'analyse des trames
self.patterns = {
    'VeTiMa': re.compile(r'VeTiMa\s*(0x[0-9A-Fa-f]{1,4})\s*(0x[0-9A-Fa-f]{1,4})'),
    'iMa': re.compile(r'iMa\s*(0x[0-9A-Fa-f]{1,4})\s*(0x[0-9A-Fa-f]{1,4})'),
    'Ta': re.compile(r'Ta\s*(0x[0-9A-Fa-f]{1,4})\s*(0x[0-9A-Fa-f]{1,4})')
}

def parse_line_raw(self, line):
    """Parse une ligne et retourne les valeurs brutes"""
    results = []
    for pattern_name, pattern in self.patterns.items():
        matches = pattern.findall(line)
        for match in matches:
            val1 = int(match[0], 16)  # Force
            val2 = int(match[1], 16)  # Angle

            if val1 <= 0xFFFF and val2 <= 0xFFFF:
                results.append({
                    'type': pattern_name,
                    'raw_force': val1,
                    'raw_angle': val2
                })
    return results if results else None
```

2.3 Système de calibration

2.3.1 Structure des données de calibration

Listing 2 – Format de calibration

```
{
  "angle": {
    "raw_min": 0,       // Valeur brute a 0 degres
    "raw_max": 1023,    // Valeur brute a 45 degres
    "real_min": 0.0,    // Valeur reelle a 0 degres
    "real_max": 45.0,   // Valeur reelle a 45 degres
    "calibrated": true
  },
  "force": {
    "raw_min": 0,       // Valeur brute a vide
    "raw_max": 1023,    // Valeur brute a 1kg
    "real_min": 0.0,    // Valeur reelle a vide
    "real_max": 1.0,    // Valeur reelle a 1kg
    "calibrated": true
  }
}
```

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

```
def convert_raw_to_physical(self, raw_angle, raw_force):
    """Convertir les valeurs brutes en valeurs physiques"""
    # Conversion angle
    if self.calibration['angle']['calibrated']:
        angle_cal = self.calibration['angle']
        if angle_cal['raw_max'] != angle_cal['raw_min']:
            angle_ratio = (raw_angle - angle_cal['raw_min']) / \
                          (angle_cal['raw_max'] - angle_cal['raw_min
                              '])
            angle_deg = angle_cal['real_min'] + \
                        angle_ratio * (angle_cal['real_max'] -
                            angle_cal['real_min'])
```

```
11          else:
12              angle_deg = angle_cal['real_min']
13      else:
14          # Conversion par d faut
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

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

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

```
# Configuration du moyennage
```

```
averaging_window = 25  # Nombre de valeurs pour la moyenne
angle_accumulator = []  # Accumulateur pour les angles
force_accumulator = []  # Accumulateur pour les forces

def measurement_worker():
    """Worker thread pour la mesure en continu"""
    global current_measurement, measurement_active, decoder
    global averaging_window, angle_accumulator, force_accumulator

    while measurement_active:
        # Lecture des donn es  s rie
        parsed = decoder.parse_line(line)

        if parsed:
            for data in parsed:
                # Accumuler les valeurs
                angle_accumulator.append({
                    'angle': data['angle_deg'],
                    'raw_angle': data['raw_angle']
                })
                force_accumulator.append({
                    'force': data['force_kg'],
                    'raw_force': data['raw_force']
                })

                # Si assez de valeurs, calculer la moyenne
                if len(angle_accumulator) >= averaging_window:
                    avg_angle = sum([item['angle'] for item in
                       angle_accumulator]) / len(angle_accumulator)
                    avg_force = sum([item['force'] for item in
                       force_accumulator]) / len(force_accumulator)

                    # Cr er le point de mesure
                    measurement_point = {
                        'timestamp': time.time() - start_time,
                        'angle': round(avg_angle, 2),
                        'force': round(avg_force, 3),
                        'raw_angle': int(avg_raw_angle),
                        'raw_force': int(avg_raw_force),
                        'samples_count': len(angle_accumulator)
                    }

                    # Vider les accumulateurs
                    angle_accumulator = []
                    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

```
def export_to_excel():
    """Exporter les donn es vers Excel"""
    # Cr er un DataFrame pandas
    df = pd.DataFrame(current_measurement)

    # Cr er un buffer en m moire
    output = io.BytesIO()

    #  crire  le fichier Excel
    with pd.ExcelWriter(output, engine='openpyxl') as writer:
        # Feuille des donn es
        df.to_excel(writer, sheet_name='Mesures MEVEM', index=False
            )

        # Feuille des m tadonn es
        metadata_df = pd.DataFrame({
            'Information': [
                'Date de mesure',
                'Nombre de points',
                'Dur e (s)',
                'Angle min (  )',
                'Angle max (  )',
                'Force min (kg)',
                'Force max (kg)'
            ],
            'Valeur': [
                datetime.now().strftime('%Y-%m-%d %H:%M:%S'),
                len(current_measurement),
                round(max([p['timestamp'] for p in
                    current_measurement]), 2),
                # ... autres statistiques
            ]
        })
        metadata_df.to_excel(writer, sheet_name='M tadonn es',
            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

```
// Configuration du graphique principal
const chartConfig = {
    type: 'scatter',
    data: {
        datasets: [{
            label: 'Force vs Angle',
            data: [],
            backgroundColor: 'rgba(54, 162, 235, 0.6)',
            borderColor: 'rgba(54, 162, 235, 1)',
            borderWidth: 2,
            pointRadius: 2,
            showLine: true,
            tension: 0.1
        }]
    },
    options: {
        responsive: true,
        maintainAspectRatio: false,
        scales: {
            x: {
                type: 'linear',
                position: 'bottom',
                title: {
                    display: true,
                    text: 'Angle (degr s)'
                }
            },
            y: {
                title: {
                    display: true,
                    text: 'Force (kg)'
                }
            }
        },
        animation: {
            duration: 0 // Pas d'animation pour le temps r el
        }
    }
};
```

5.2.2 Gestion des événements WebSocket

Listing 8 – Gestion des WebSockets côté client

```
// Connexion WebSocket
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',
```

```
        '--onefile',
        '--windowed' if platform == 'windows' else '',
        '--add-data=templates;templates',
        '--add-data=static;static',
        '--hidden-import=eventlet',
        '--hidden-import=socketio',
        '--hidden-import=engineio',
        '--clean',
        'app.py'
    ]

    # Options sp cifiques    la plateforme
    if platform == 'windows':
        base_options.extend([
            '--icon=icon.ico',
            '--version-file=version.txt',
            '--distpath=dist_windows'
        ])
    elif platform == 'linux':
        base_options.extend([
            '--distpath=dist_linux'
        ])

    # Ex cuter PyInstaller
    PyInstaller.__main__.run(base_options)
```

6.3 Gestion des dépendances

6.3.1 Fichier requirements.txt

Listing 10 – Dépendances Python

```
Flask>=2.3.0
Flask-SocketIO>=5.3.0
pyserial>=3.5
pandas>=2.0.0
openpyxl>=3.1.0
pyinstaller>=5.13.0
eventlet>=0.33.0
python-socketio>=5.8.0
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

```
def check_port_access(port):
    """V rifier l'acc s    un port  s rie"""
    try:
        # Essayer d'ouvrir le port bri vement
        test_conn = serial.Serial(port, timeout=0.1)
        test_conn.close()
        return {'accessible': True}
    except serial.SerialException as e:
        error_msg = str(e)
        if 'Permission denied' in error_msg:
            return {
                'accessible': False,
                'error': 'Permission refus e - Ajoutez votre
                   utilisateur au groupe dialout'
            }
        elif 'Device or resource busy' in error_msg:
            return {
                'accessible': False,
                'error': 'Port occup  par une autre application'
            }
        # ... 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

```
from collections import deque

# Stockage des donn es avec taille limit e
self.data_buffer = deque(maxlen=1000)

```

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

```python
import asyncio
import websockets
import time

async def stress_test_websocket():
    """Test de charge sur les WebSockets"""
    start_time = time.time()

    # Simuler 1000 messages par seconde pendant 1 minute
    for i in range(60000):
        data = {
            'timestamp': time.time() - start_time,
            'angle': random.uniform(0, 45),
            'force': random.uniform(0, 1)
        }

        # Envoyer via WebSocket
        await websocket.send(json.dumps(data))

        if i % 1000 == 0:
            print(f"Envoy  {i} messages")

        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

```python
import logging
from logging.handlers import RotatingFileHandler

# Configuration du logger principal
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
    handlers=[
        RotatingFileHandler('mevem.log', maxBytes=1024*1024,
           backupCount=5),
```

```
        logging.StreamHandler()
    ]
)

logger = logging.getLogger('MEVEM')

# Utilisation dans le code
logger.info("     MEVEM - D marrage de l'application")
logger.warning("       Mode d mo activ  - aucun capteur d tect
    ")
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

```
-- Table des sessions de mesure
CREATE TABLE measurement_sessions (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    timestamp DATETIME DEFAULT CURRENT_TIMESTAMP,
    duration_seconds REAL,
    sample_count INTEGER,
    max_angle REAL,
    max_force REAL,
    calibration_used TEXT, -- JSON des param tres de calibration
    notes TEXT
);

-- Table des points de mesure
CREATE TABLE measurement_points (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    session_id INTEGER REFERENCES measurement_sessions(id),
    timestamp REAL, -- Temps relatif dans la session
    angle_deg REAL,
    force_kg REAL,
    raw_angle INTEGER,
    raw_force INTEGER,
    samples_count INTEGER
);
```

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

```
#!/usr/bin/env python3
"""
Utilitaire de diagnostic syst me MEVEM
V rifie la configuration et les pr requis
"""

import os
import sys
import platform
import serial.tools.list_ports
import grp
import pwd

def check_system_requirements():
    """V rifier les pr requis syst me"""
    print("     V rification des pr requis syst me")
    print("=" * 50)

    # Version Python
    python_version = sys.version_info
    print(f"     Python: {python_version.major}.{python_version.
       minor}.{python_version.micro}")

    if python_version < (3, 8):
        print("[ERREUR] Python 3.8+ requis")
        return False
    else:
        print("    Version Python compatible")

    # Syst me d'exploitation
    system = platform.system()
    print(f"     OS: {system} {platform.release()}")

    # Ports s rie disponibles
    ports = list(serial.tools.list_ports.comports())
    print(f"     Ports s rie: {len(ports)} trouv s")

    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
                        dialout")
62                  print(f"   Ex cutez: sudo usermod -a -G dialout {
                        current_user}")
63          except KeyError:
64              print("[AVERTISSEMENT] Groupe dialout non trouv ")
65
66      except Exception as e:
67          print(f"[ERREUR] Erreur v rification permissions: {e}")
68
69  if __name__ == "__main__":
70      check_system_requirements()
```

10.4 Annexe D : Configuration de développement

10.4.1 Configuration IDE recommandée

VS Code settings.json

```
{
    "python.defaultInterpreterPath": "./venv/bin/python",
    "python.formatting.provider": "black",
    "python.linting.enabled": true,
    "python.linting.pylintEnabled": true,
    "files.exclude": {
        "**/__pycache__": true,
        "**/*.pyc": true,
        "**/dist": true,
        "**/build": true
    },
    "editor.formatOnSave": true,
    "python.testing.pytestEnabled": true
}
```

10.4.2 Configuration pre-commit

Listing 17 – .pre-commit-config.yaml

```
repos:
  - repo: https://github.com/psf/black
    rev: 23.3.0
    hooks:
      - id: black
        language_version: python3.8

  - repo: https://github.com/pycqa/flake8
    rev: 6.0.0
    hooks:
      - id: flake8
        args: [--max-line-length=88]

  - repo: https://github.com/pycqa/isort
    rev: 5.12.0
    hooks:
      - id: isort
        args: [--profile, black]
```