CAPTEUR VIBRATOIRE AUTO-ALIMENTÉ POUR LA MAINTENANCE PRÉDICTIVE

CHAABNA Hakim, 52110

MP option informatique, TIPE 2026

INTRODUCTION

Figure 1 - Exemple de retards en gare

2 milliards de minutes. C'est le temps perdu chaque année par les voyageurs ferroviaires en France à cause des retards.

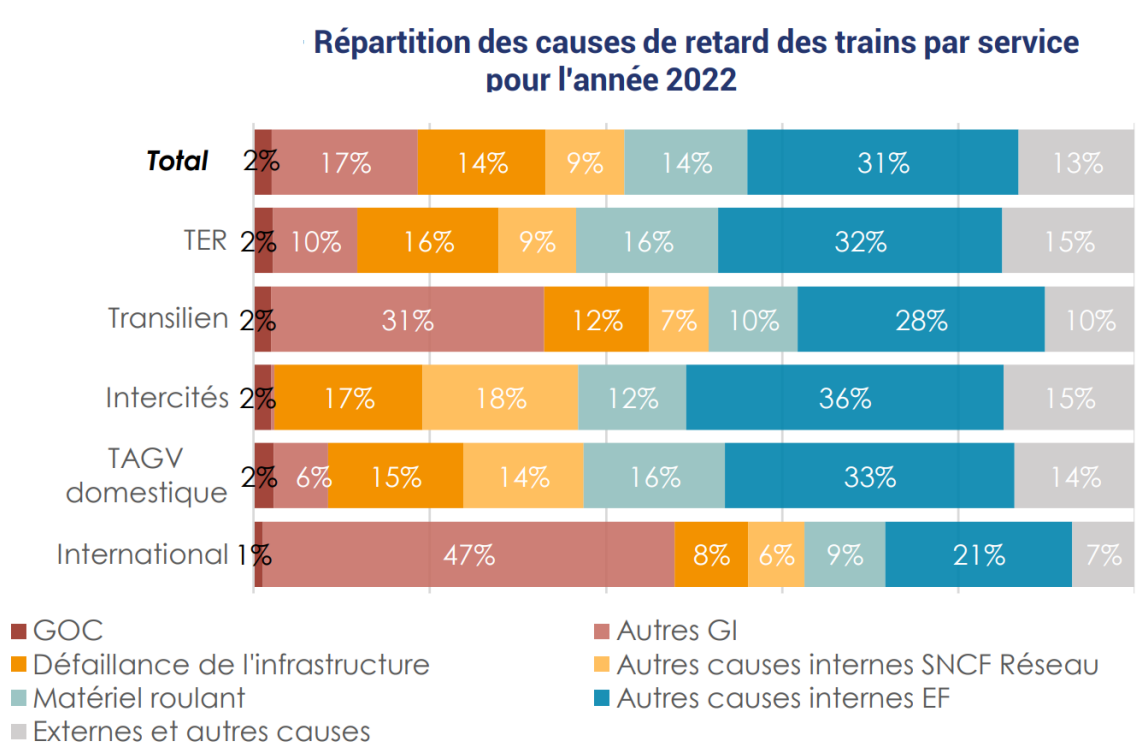


Figure 2 - Répartition des causes de retard des trains

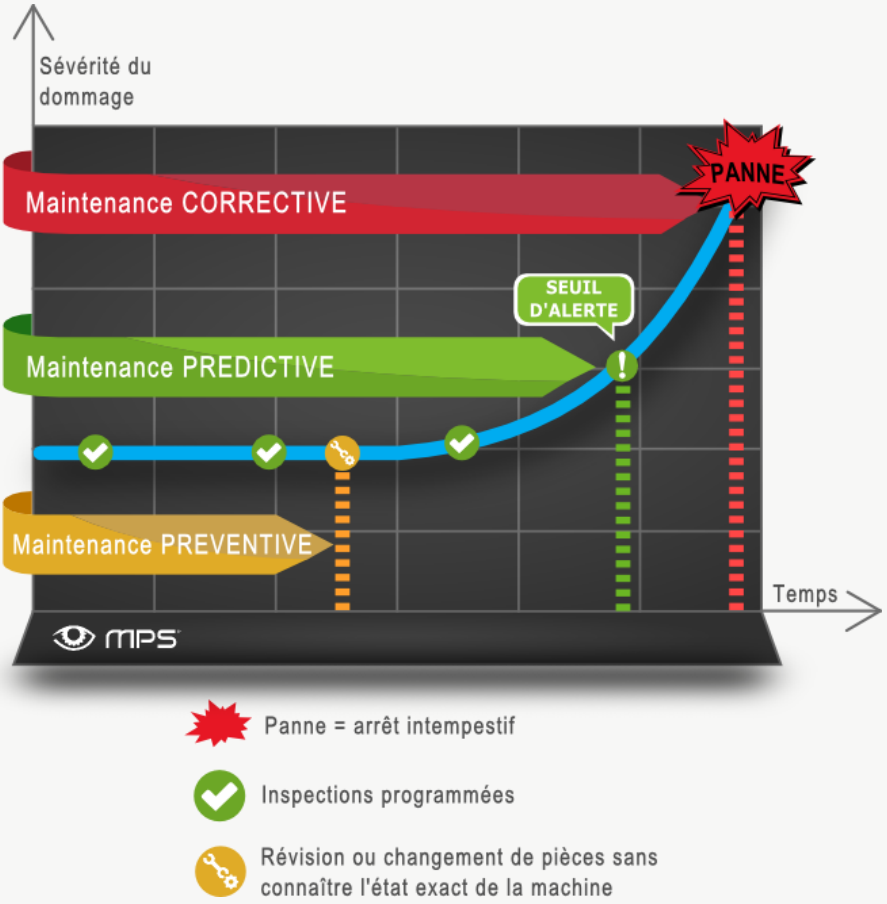


Figure 3 - Positionnement de la maintenance prédictive

PROBLÉMATIQUE

Un capteur peut-il assurer de la maintenance prédictive grâce à un fonctionnement cyclique alimenté par la chaleur de la machine qu'il surveille ?

PLAN

- Architecture du système
- Accumulation d'énergie
- Modélisation du TEG
- Gestion d'énergie
- Bilan énergétique
- Capture des vibrations
- Traitements des données
- Limites

IDÉES

- Générer de l'énergie en exploitant la chaleur dégagée par les machines
- Faire un système peu consommateur en énergie
- Traiter les données récoltées en dehors du système

ARCHITECTURE DU SYSTÈME

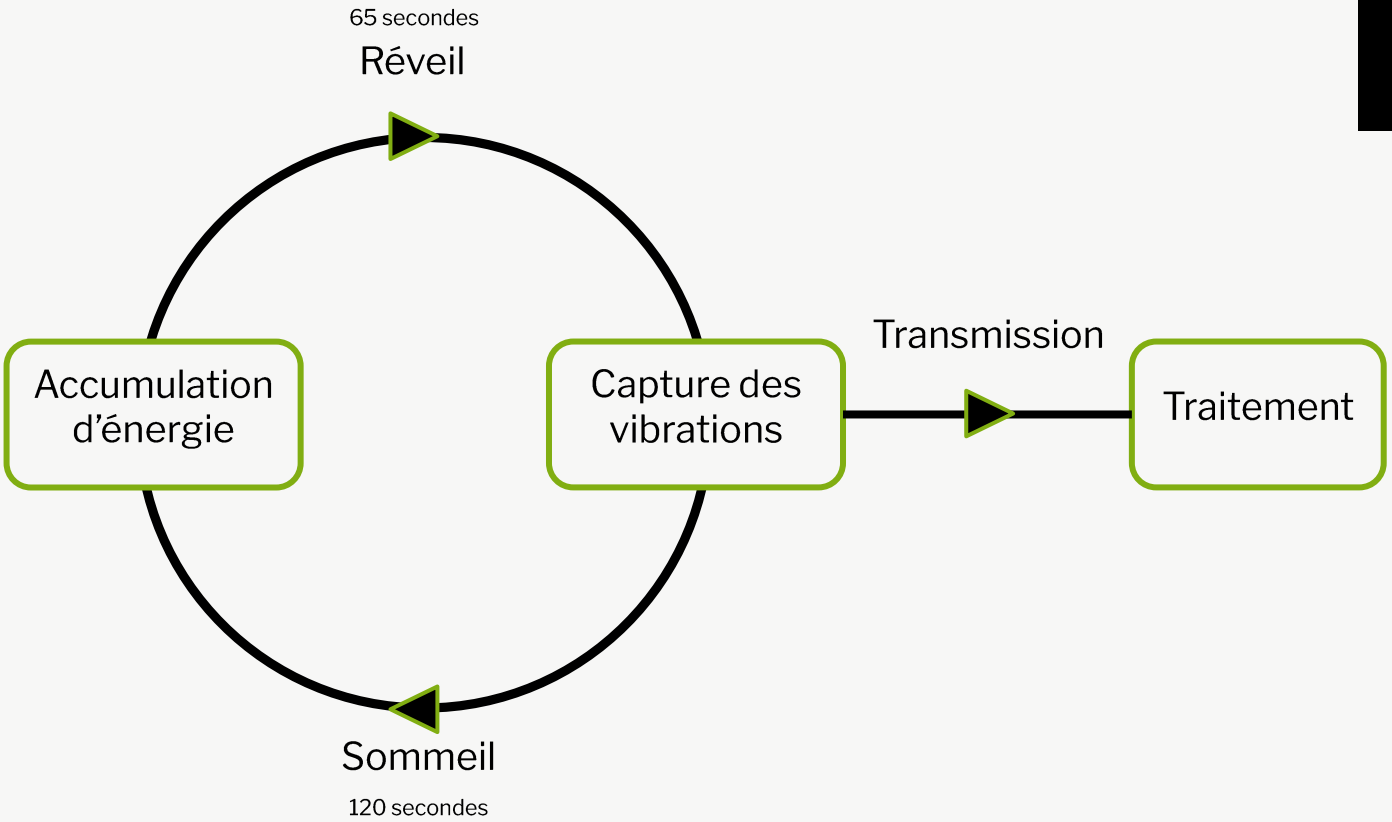


Figure 4 - Cycle de fonctionnement du système

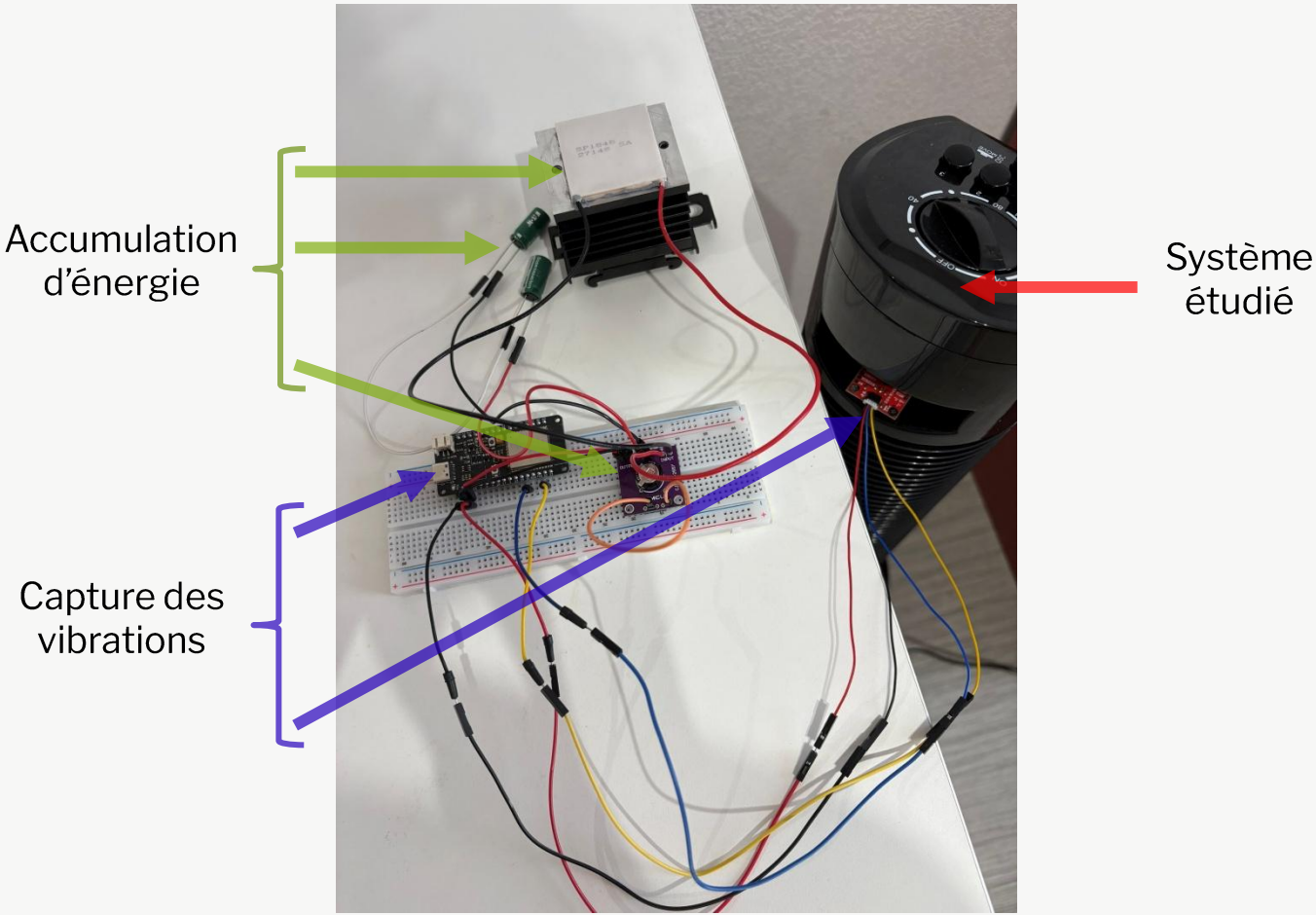


Figure 5 - Vue d'ensemble du montage expérimental

ACCUMULATION D'ÉNERGIE

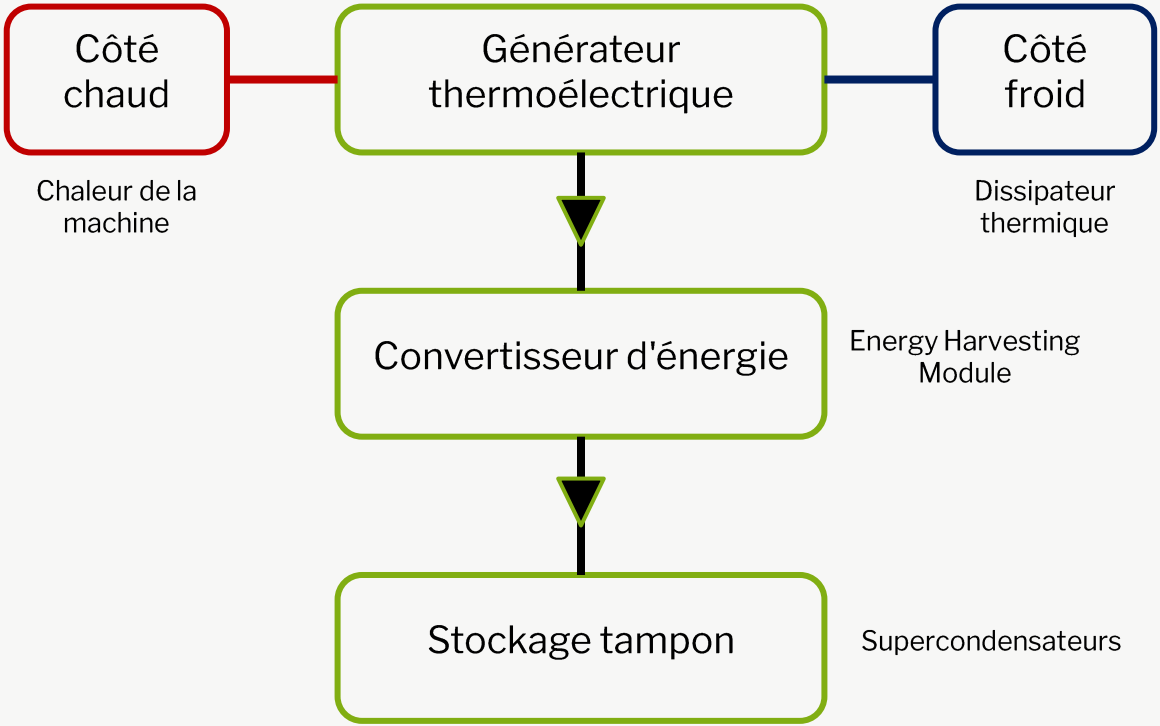


Figure 6 - Chaîne d'accumulation d'énergie

TEG
(générateur
thermoélectrique)

Dissipateur
thermique

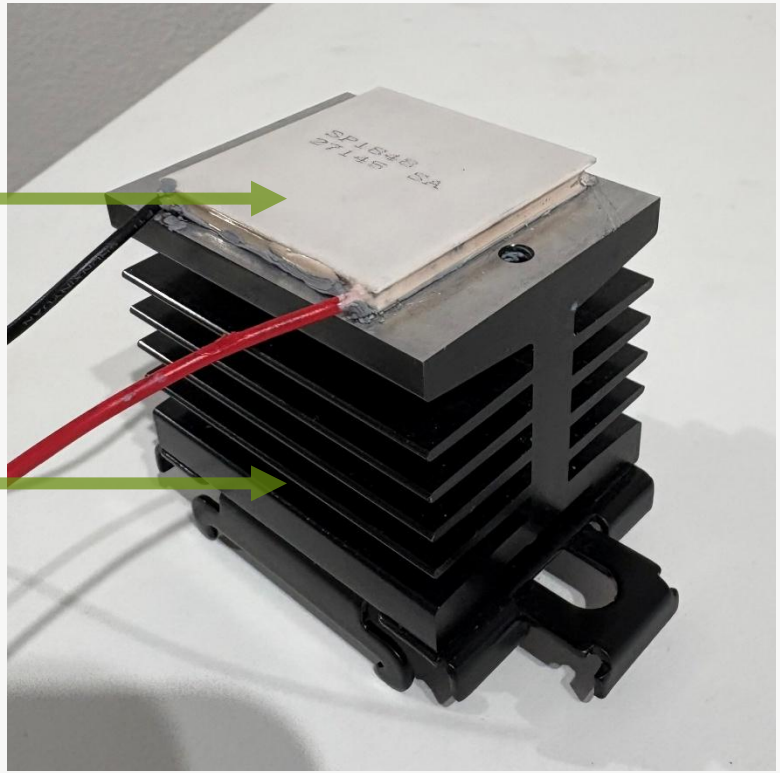


Figure 7 - Générateur thermoélectrique monté sur son dissipateur thermique

MODÉLISATION DU TEG

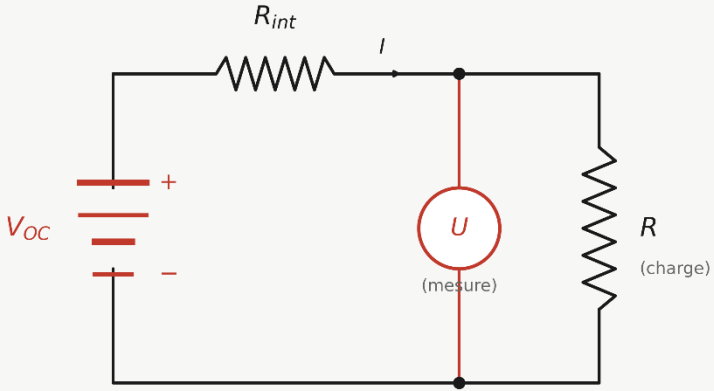


Figure 8 - Circuit de mesure

$$T_{systeme} = 96°$$
$$\Delta T = 21{,}5°$$
$$V_{oc} = 1{,}02\ V$$

R (ohm)	U (Volt)
1	0,21
1,11	0,25
1,25	0,28
1,42	0,32
1,66	0,36
2	0,39
2,5	0,44
3,33	0,5
5,0	0,6
10	0,75

Figure 9 - Relevés expérimentaux

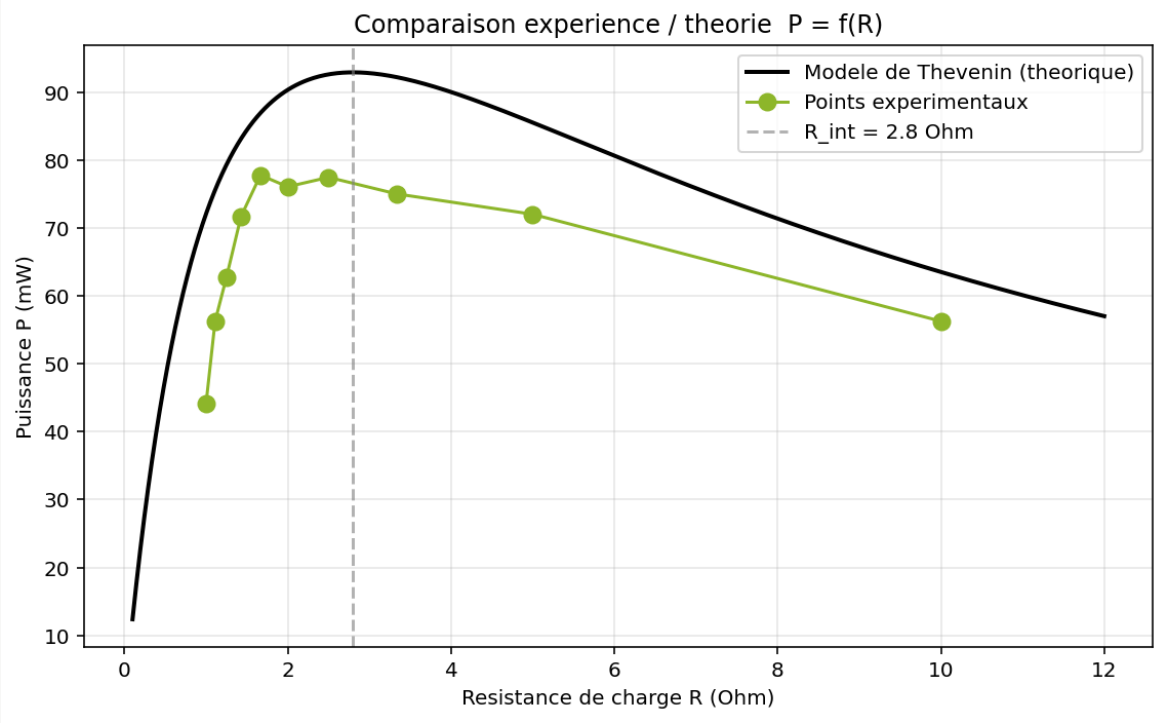


Figure 10 - Comparaison expérience/théorie

$$U = V_{OC} \frac{R}{R_{int} + R}$$

$$P = \frac{U^2}{R} = V_{OC}^2 \frac{R}{(R_{int} + R)^2}$$

$$R = R_{int} \Rightarrow P_{max} = \frac{V_{OC}^2}{4 R_{int}}$$

La puissance max théorique = 92,9 mW
La puissance max expérimentale = 77,7 mW

GESTION D'ENERGIE

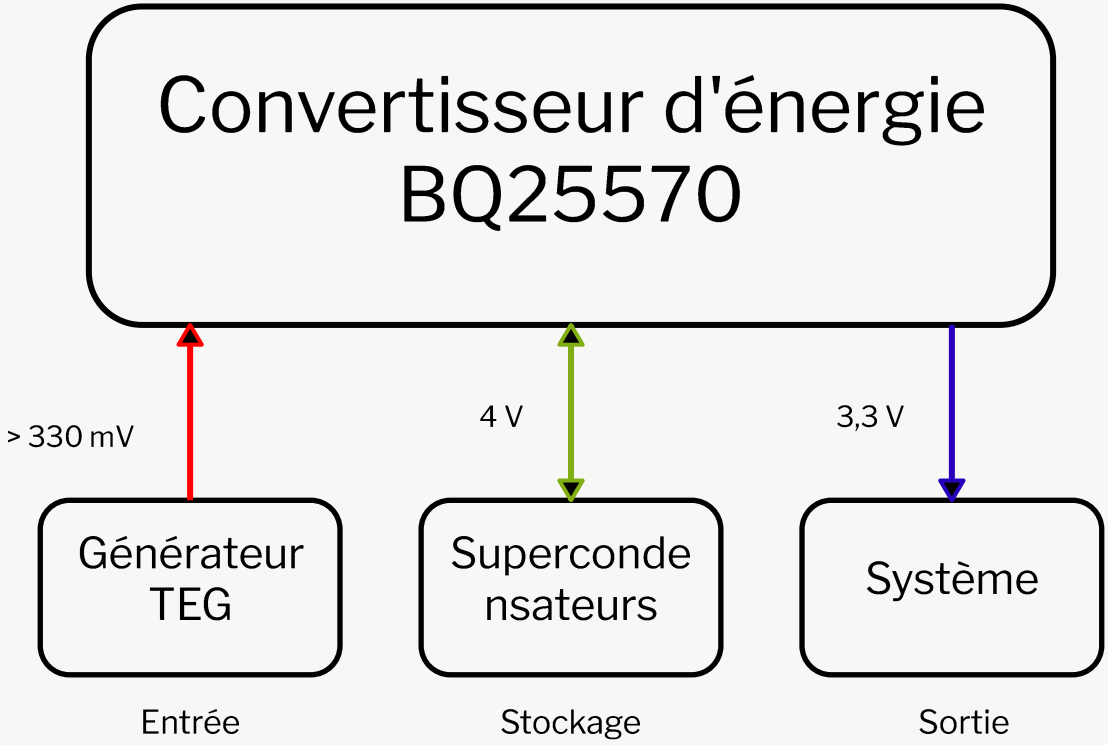


Figure 11 - Architecture de gestion d'énergie

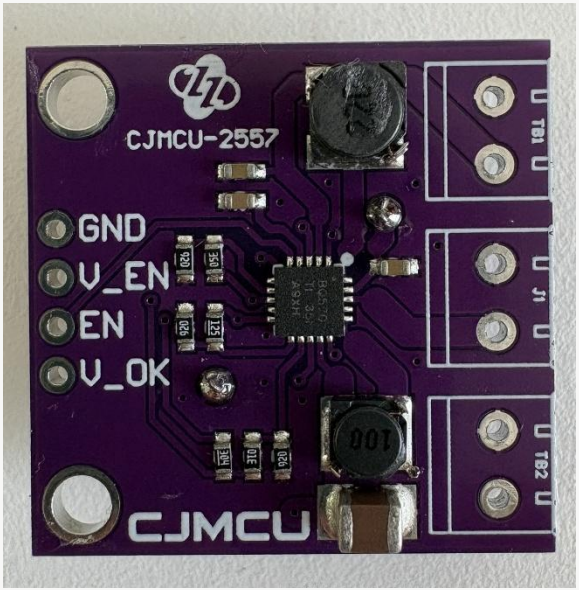


Figure 12 - Module BQ25570

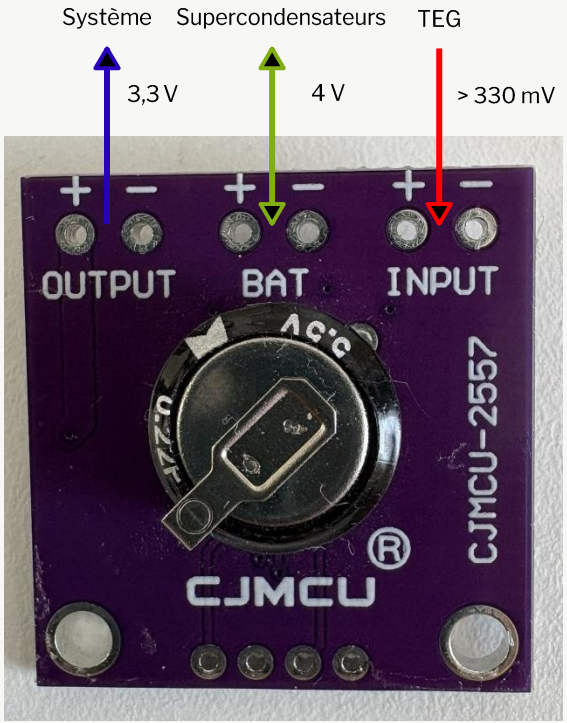


Figure 13 - Repérage des bornes

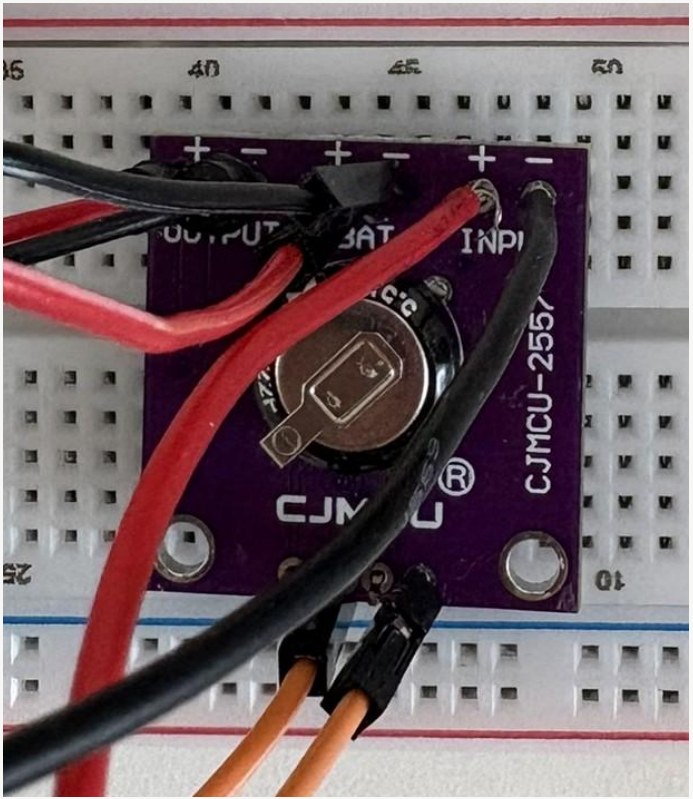


Figure 14 - Câblage du module BQ25570

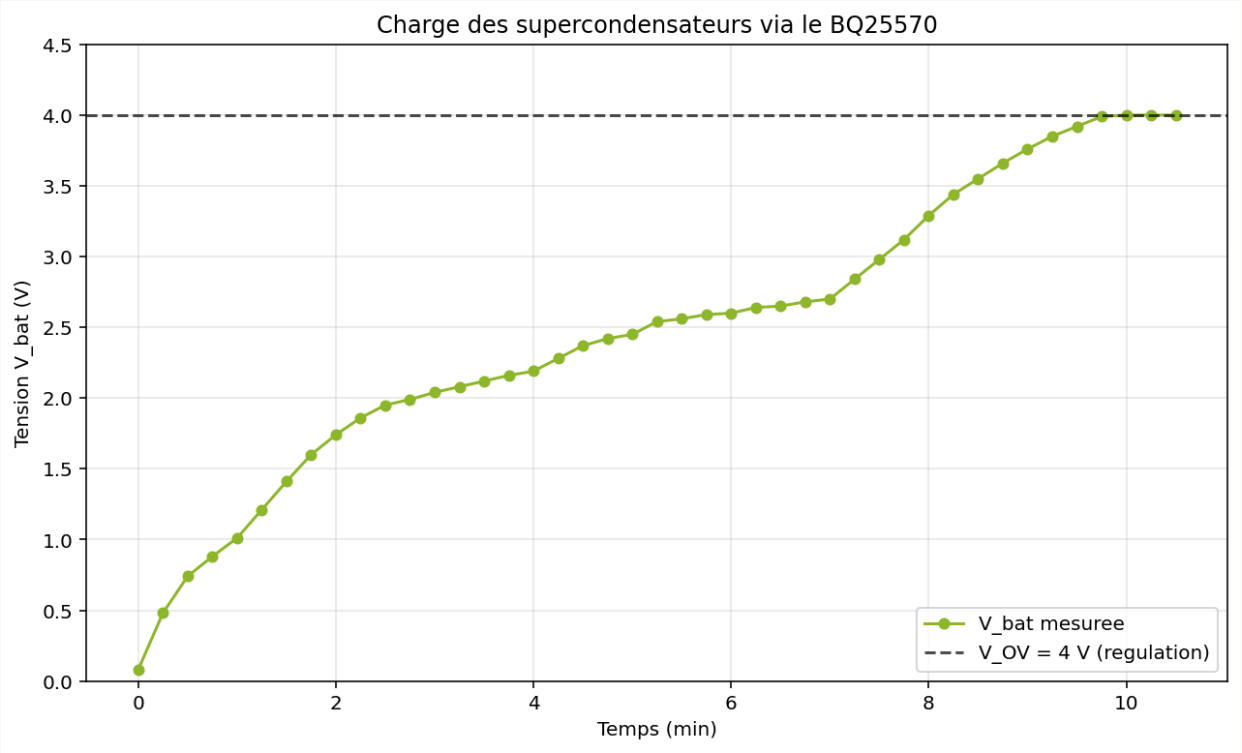
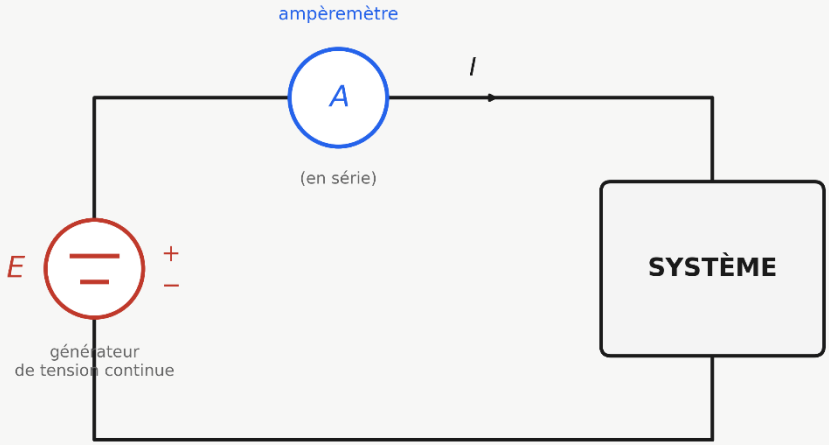


Figure 15 - Courbe de charge des supercondensateurs via le BQ25570

BILAN ÉNERGÉTIQUE

Phase	Durée (s)	Courant (mA)	Puissance (mW)
Réveil	65	0,8	2,64
Sommeil	120	0,015	0,0495
Moyenne	185	0,29	0,96

Figure 16 - Bilan énergétique du cycle



Puissance exploitable du TEG 66-50 mW

Marge d'un facteur 68-52

Figure 17 - Circuit de mesure du courant consommé par le système

CAPTURE DES VIBRATIONS

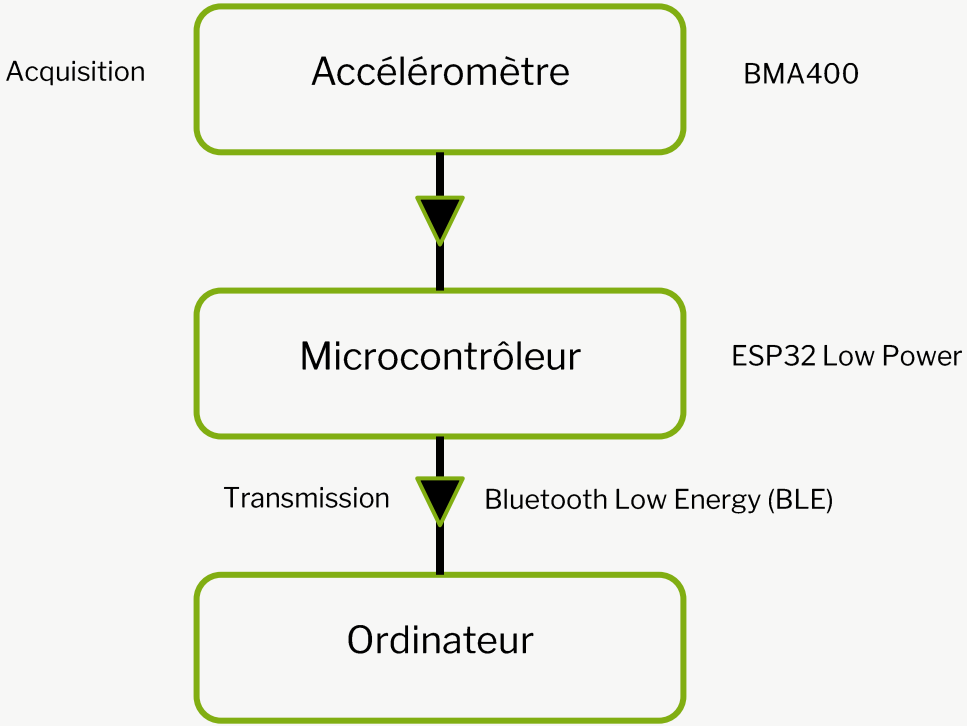


Figure 18 - Chaîne d'acquisition/transmission des vibrations

Accéléromètre

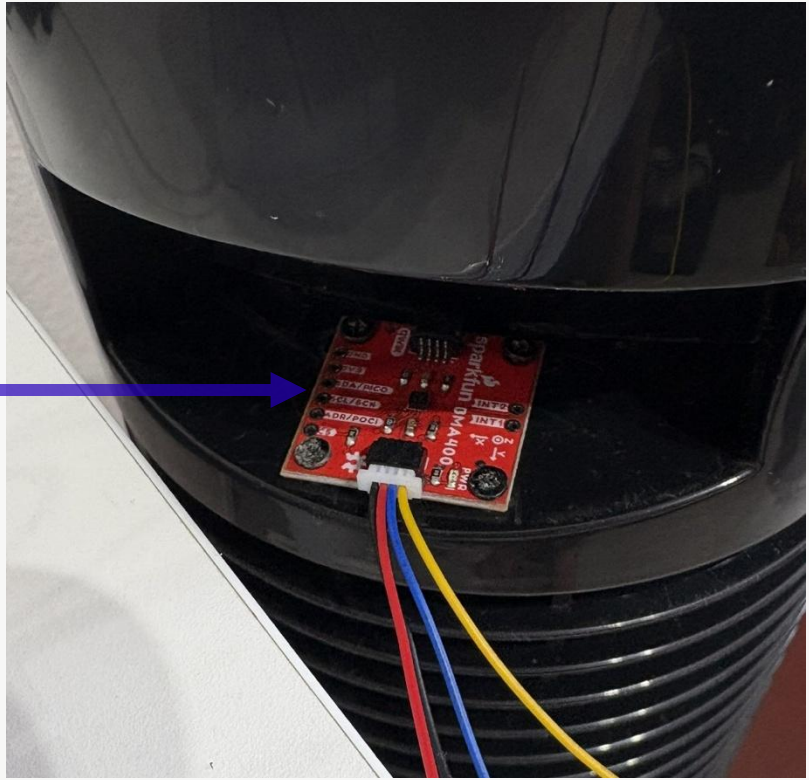


Figure 19 - Accéléromètre BMA400 fixé sur le système surveillé

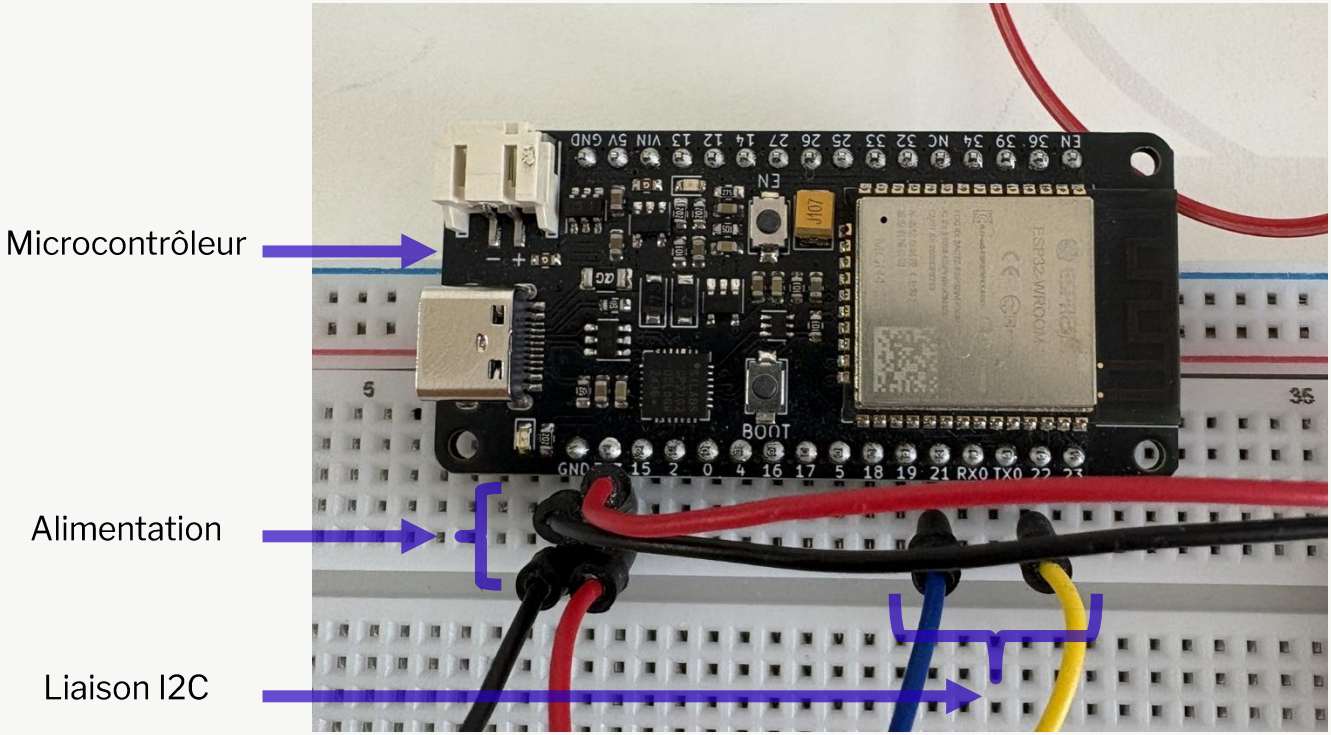


Figure 20 - Microcontrôleur ESP32

TRAITEMENT DES DONNÉES

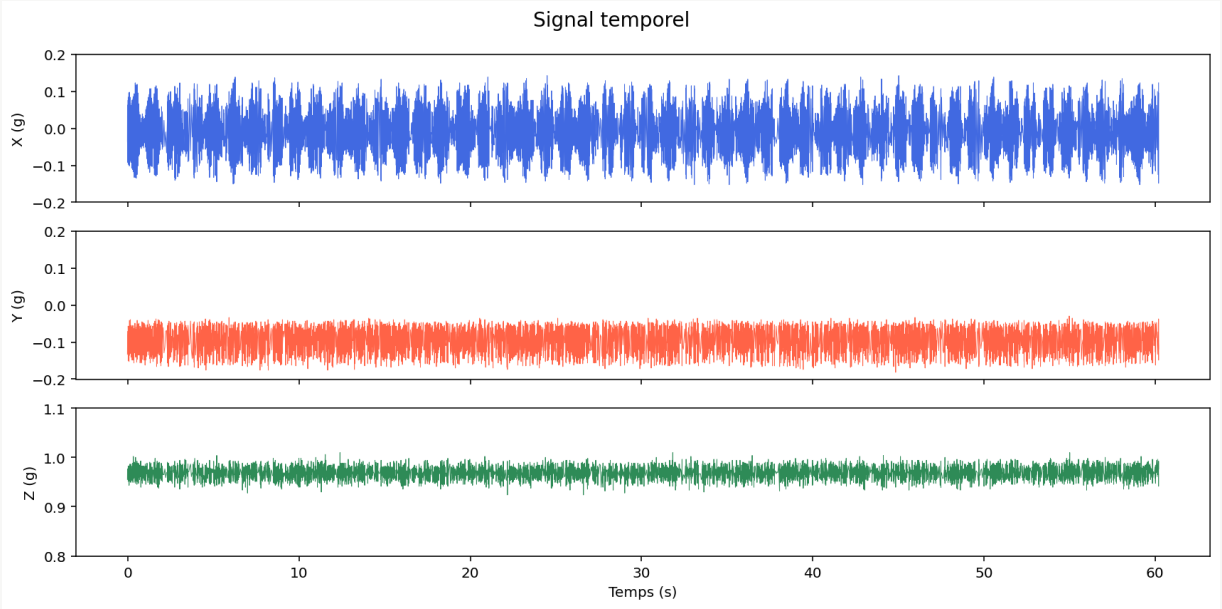


Figure 21 - Signal d'accélération temporel brut

FFT (SPECTRE FRÉQUENTIEL)

Décomposition du signal d'accélération en sinusoïdes à différentes fréquences.

Intérêt : chaque défaut mécanique a une signature fréquentielle propre.

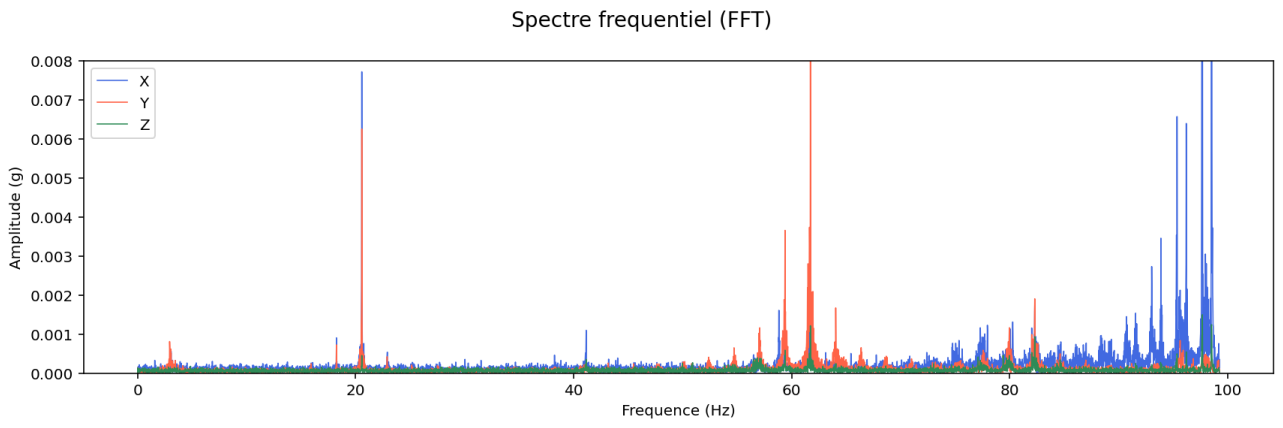


Figure 22 - Spectre fréquentiel à l'état sain

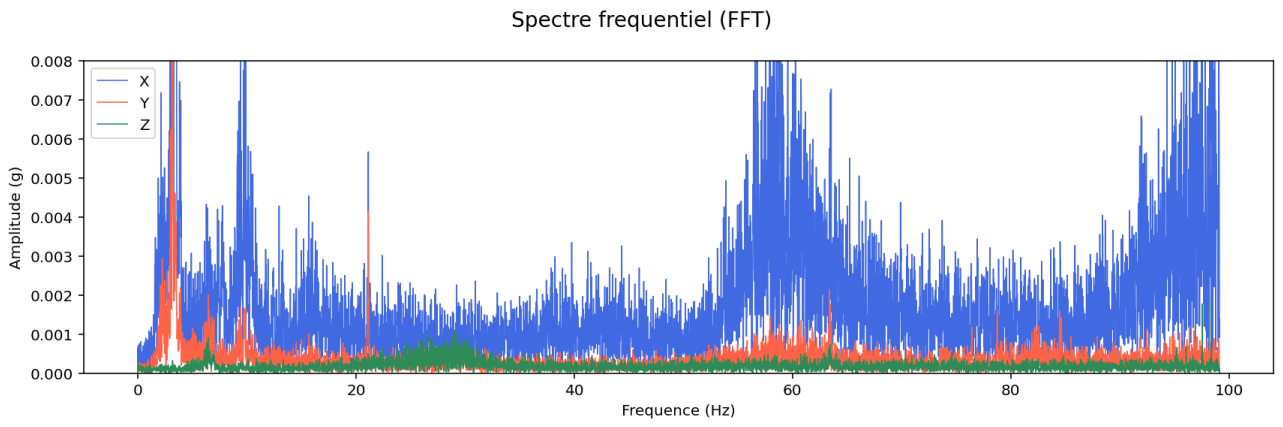


Figure 23 - Spectre fréquentiel en présence de défaut

RMS (LA MOYENNE QUADRATIQUE)

Indicateur qui résume en un nombre l'énergie vibratoire moyenne de la machine.

Intérêt : un RMS qui monte signale une usure progressive et permet de planifier l'intervention.

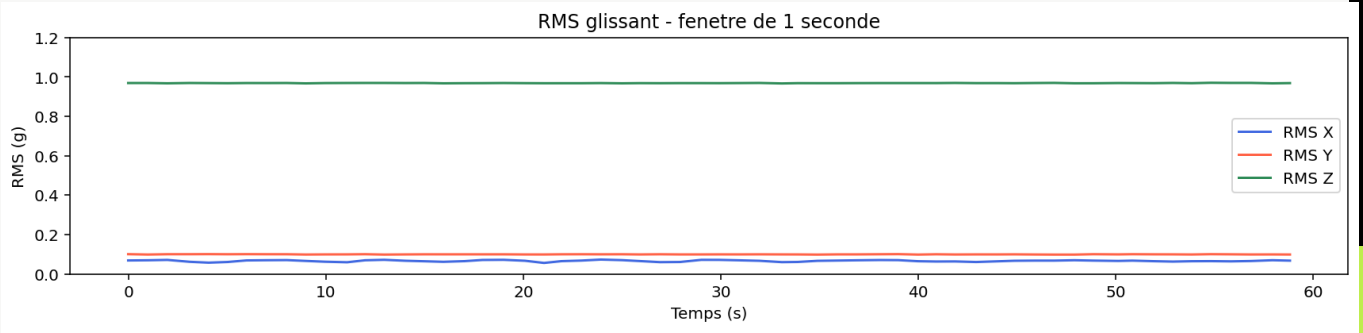


Figure 24 – RMS à l'état sain

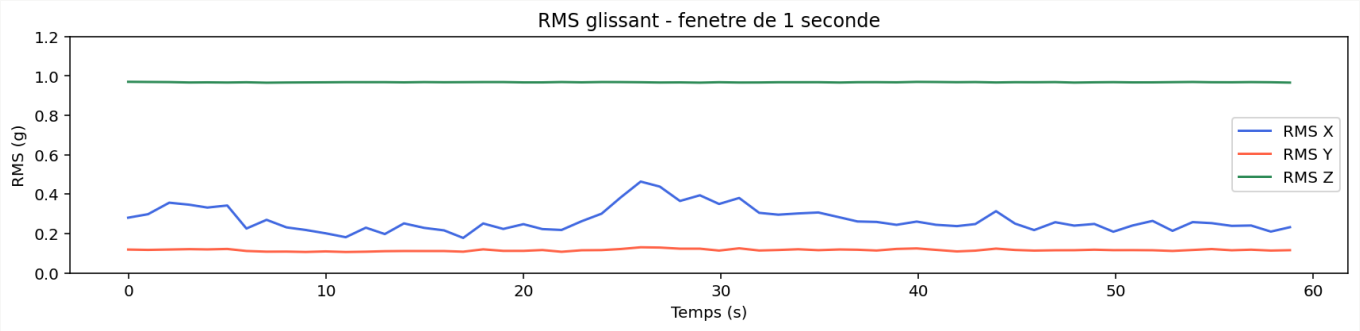


Figure 25 - RMS en présence de défaut

LE KURTOSIS

Indicateur statistique qui mesure la présence de chocs impulsionnels dans le signal.

Intérêt : détecte les défauts naissants bien avant qu'ils ne soient visibles sur le RMS.

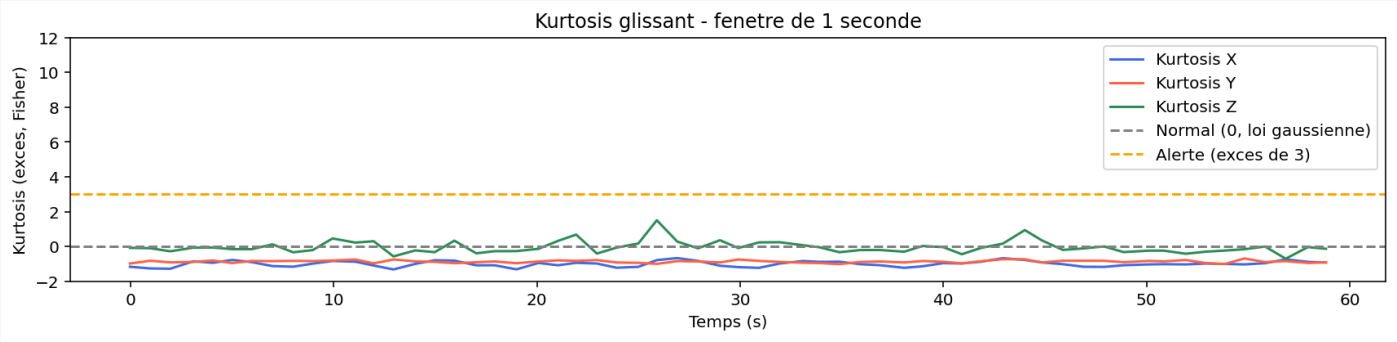


Figure 26 - Kurtosis à l'état sain

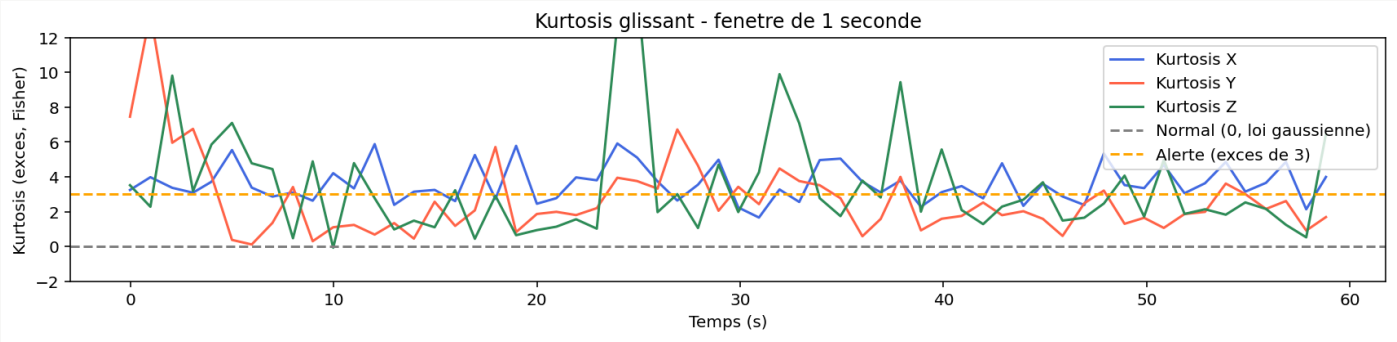


Figure 27 - Kurtosis en présence de défaut

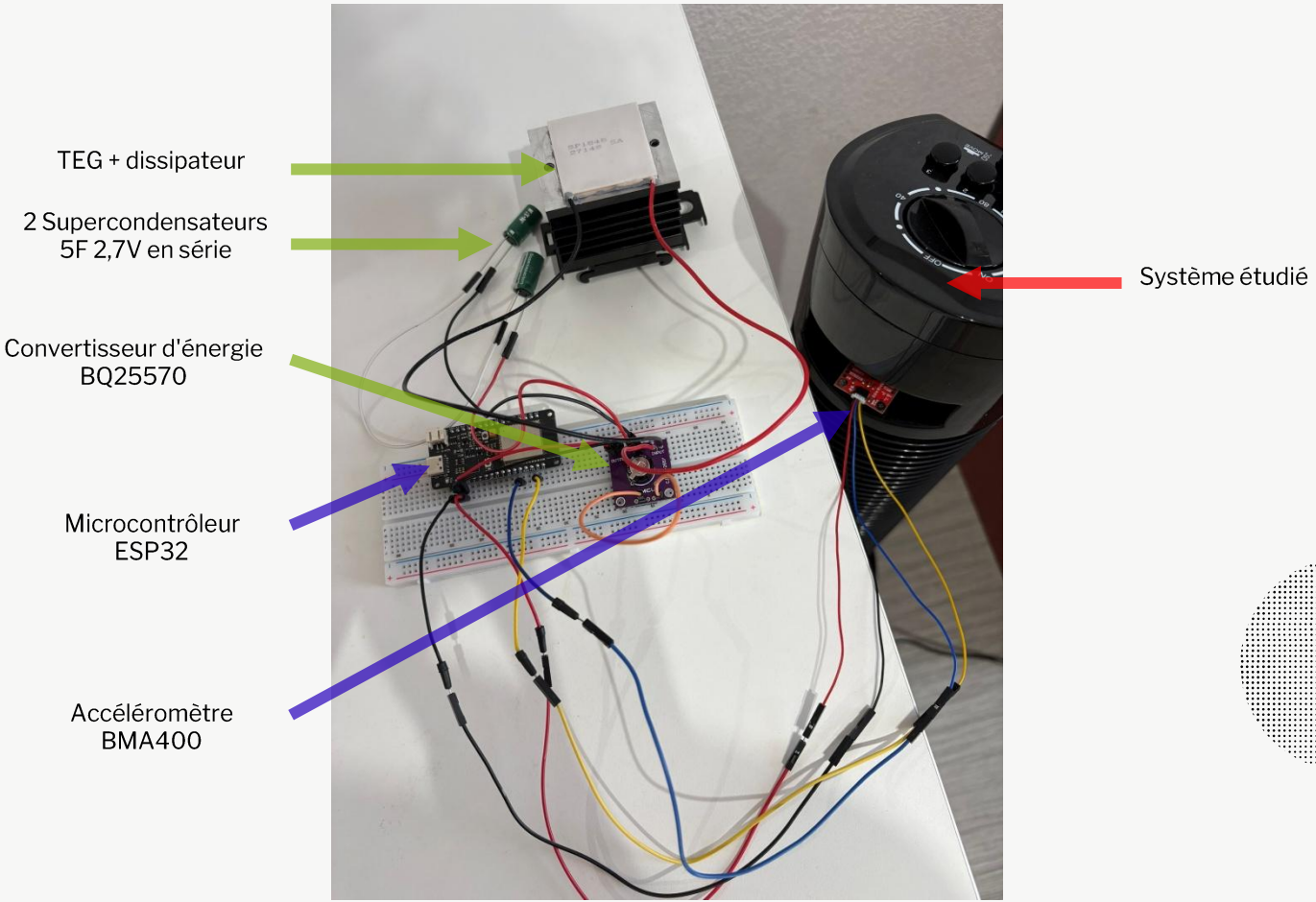


Figure 28 - Montage complet du système annoté

LIMITES

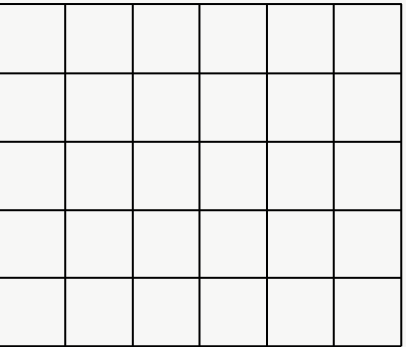
- Dissipateur surdimensionné
- Cold-start lent
- Dégradation du TEG
- Portée de transmission limitée
- Détection non temps réel

CONCLUSION

Un capteur auto-alimenté cycliquement peut assurer une maintenance prédictive avec une marge énergétique confortable, cependant un passage à l'échelle industrielle nécessiterait des optimisations.

MERCI !

Avez-vous des questions sur mon système?



ANNEXES

Figure 1 : https://france3-regions.franceinfo.fr/occitanie/herault/montpellier/sncf-plus-de-7-heures-de-retard-pour-2-tgv-en-provenance-de-lille-et-bruxelles-a-leur-arrivee-a-montpellier-3245113.html
Figure 2 : https://www.autorite-transports.fr/wp-content/uploads/2023/12/essentiel_bilan-marche-ferroviaire2022.pdf
Figure 3 : http://www.mps-maintenance.fr/maintenance-predictive/

Composants :

Accéléromètre : SparkFun Qwiic BMA400 SEN-21208
Microcontrôleur : uPesy ESP32 Wroom Low Power DevKit v1.2
Supercondensateurs : 2 x Eaton Electronics Supercapacitors 5F 2.7V HV1020-2R7505-R
Générateur thermoelectrique : TEG SP1848-27145
Dissipateur : Telituny 8 x 5,5 x 4,5 GGSP1Z1B0Y2W3W3L-X0F3
Convertisseur d'énergie : BQ25570 Energy Harvester Module

Formules :

FTT : $X(k) = \sum_{n=0}^{N-1} a(n)\, e^{-j\,2\pi kn/N}$

RMS : $RMS = \sqrt{\frac{1}{N}\sum_{n=0}^{N-1} a(n)^2}$

KURTOSIS : $K = \frac{\frac{1}{N}\sum (a(n) - \bar{a})^4}{\left(\frac{1}{N}\sum (a(n) - \bar{a})^2\right)^2} - 3$

Code de la figure 10 :

```
import numpy as np
import matplotlib.pyplot as plt

# ============================================================
#  DONNEES EXPERIMENTALES  (a modifier avec tes valeurs)
# ============================================================
# Resistances de charge (en ohms) -- on retire le point R=0 (a vide)
R_exp = np.array([1.0, 1.1111, 1.25, 1.4285, 1.6667, 2.0, 2.5, 3.3333, 5.0, 10.0])
# Tensions mesurees aux bornes de la charge (en volts)
U_exp = np.array([0.21, 0.25, 0.28, 0.32, 0.36, 0.39, 0.44, 0.50, 0.60, 0.75])

# Puissance experimentale : P = U^2 / R  (en milliwatts)
P_exp = (U_exp**2 / R_exp) * 1000  # *1000 pour passer en mW

# ============================================================
#  PARAMETRES DU MODELE DE THEVENIN
# ============================================================
V_OC = 1.02     # tension a vide mesuree (volts)
R_int = 2.8     # resistance interne (ohms)

# ============================================================
#  COURBE THEORIQUE : P(R) = V_OC^2 * R / (R_int + R)^2
# ============================================================
R_theo = np.linspace(0.1, 12, 500)  # plage continue de resistances
P_theo = (V_OC**2 * R_theo / (R_int + R_theo)**2) * 1000  # en mW
```

```
# ============================================================
#  FIGURE
# ============================================================
plt.figure(figsize=(8, 5))
plt.plot(R_theo, P_theo, '-', color='black', linewidth=2,
       label='Modele de Thevenin (theorique)')
plt.plot(R_exp, P_exp, 'o-', color='#8db62a', markersize=8,
       linewidth=1.5, label='Points experimentaux')
plt.axvline(R_int, color='gray', linestyle='--', alpha=0.6,
         label=f'R_int = {R_int} Ohm')
plt.xlabel("Resistance de charge R (Ohm)")
plt.ylabel("Puissance P (mW)")
plt.title("Comparaison experience / theorie  P = f(R)")
plt.grid(True, alpha=0.3)
plt.legend()
plt.tight_layout()
plt.show()
```

Code de la figure 15 :

```
import numpy as np
import matplotlib.pyplot as plt

# ============================================================
#  DONNEES EXPERIMENTALES
# ============================================================
# Tension V_bat relevee toutes les 15 secondes (en volts)
V_bat = [0.08, 0.48, 0.74, 0.88, 1.01, 1.21, 1.41, 1.6, 1.74, 1.86,
      1.95, 1.99, 2.04, 2.08, 2.12, 2.16, 2.19, 2.28, 2.37, 2.42,
      2.45, 2.54, 2.56, 2.59, 2.6, 2.64, 2.65, 2.68, 2.7, 2.84,
      2.98, 3.12, 3.29, 3.44, 3.55, 3.66, 3.76, 3.85, 3.92, 3.99,
      4.0, 4.0, 4.0]

pas = 15

# Axe du temps
temps_s = np.arange(len(V_bat)) * pas
# Conversion en minutes pour un affichage plus lisible
temps_min = temps_s / 60

# ============================================================
#  TRACE DE LA COURBE
# ============================================================
plt.figure(figsize=(9, 5.5))
plt.plot(temps_min, V_bat, 'o-', color='#8db62a', markersize=5,
      linewidth=1.5, label='V_bat mesuree')
```

```
# Ligne horizontale pour reperer V_OV (tension de regulation a 4V)
plt.axhline(4.0, color='black', linestyle='--', alpha=0.7,
        label='V_OV = 4 V (regulation)')

plt.xlabel("Temps (min)")
plt.ylabel("Tension V_bat (V)")
plt.title("Charge des supercondensateurs via le BQ25570")
plt.grid(True, alpha=0.3)
plt.legend()
plt.ylim(0, 4.5)
plt.tight_layout()
plt.show()
```

Code de l'ESP32 :

```
from machine import Pin, I2C, deepsleep
import struct
import time
import bluetooth
from micropython import const

# ======================================================================
# CONFIGURATION DU CYCLE
# ======================================================================
DUREE_ACQUISITION_MS = 65000
DUREE_SLEEP_MS = 120000
TIMEOUT_CONNEXION_MS = 30000

# ======================================================================
# CONFIGURATION BLE (Nordic UART Service)
# ======================================================================
_IRQ_CENTRAL_CONNECT = const(1)
_IRQ_CENTRAL_DISCONNECT = const(2)
_FLAG_NOTIFY = const(0x0010)

_SERVICE_UUID = bluetooth.UUID("6E400001-B5A3-F393-E0A9-E50E24DCCA9E")
_CHAR_UUID    = bluetooth.UUID("6E400003-B5A3-F393-E0A9-E50E24DCCA9E")
```

```
class BLEIMU:
    def __init__(self):
        self._ble = bluetooth.BLE()
        self._ble.active(True)
        self._ble.irq(self._irq)
        self._connected = False
        self._handle = None
        self._register()
        self._advertise()

    def _register(self):
        services = (_SERVICE_UUID, ((_CHAR_UUID, _FLAG_NOTIFY),),)
        ((self._handle,),) = self._ble.gatts_register_services((services,))

    def _irq(self, event, data):
        if event == _IRQ_CENTRAL_CONNECT:
            self._connected = True
            print("PC connecte")
        elif event == _IRQ_CENTRAL_DISCONNECT:
            self._connected = False
            print("PC deconnecte")
            self._advertise()

    def _advertise(self):
        name = b"ESP32-IMU"
        payload = bytes([2, 0x01, 0x06, len(name) + 1, 0x09]) + name
        self._ble.gap_advertise(100000, adv_data=payload)
```

```
 def send(self, data):
      if self._connected:
        try:
          self._ble.gatts_notify(0, self._handle, data)
        except OSError:
          pass

    @property
    def connected(self):
      return self._connected

# ======================================================================
# CONFIGURATION DU CAPTEUR BMA400
# ======================================================================
i2c = I2C(0, scl=Pin(22), sda=Pin(21), freq=400000)
addr = 0x14

chip_id = i2c.readfrom_mem(addr, 0x00, 1)[0]
assert chip_id == 0x90, "Erreur capteur: " + hex(chip_id)

i2c.writeto_mem(addr, 0x19, b'\x02')
time.sleep_ms(5)
def lire_acceleration():
    data = i2c.readfrom_mem(addr, 0x04, 6)
    brut = struct.unpack("<HHH", data)
```

```
  valeurs = []
    for v in brut:
        v = v & 0x0FFF
        if v > 2047:
            v -= 4096
        valeurs.append(v)
    return valeurs          # [x, y, z]
# ============================================================================
# PROGRAMME PRINCIPAL
# A chaque reveil depuis le deep sleep, le programme repart d'ici.
# ============================================================================
print("Reveil - demarrage du cycle")

ble = BLEIMU()
print("En attente de connexion BLE...")

# --- Attente de la connexion du PC (avec timeout) ---
t_debut_attente = time.ticks_ms()
while not ble.connected:
    if time.ticks_diff(time.ticks_ms(), t_debut_attente) > TIMEOUT_CONNEXION_MS:
        print("Pas de connexion, on se rendort")
        deepsleep(DUREE_SLEEP_MS)
    time.sleep_ms(100)

# --- Acquisition pendant 65 secondes ---
print("Connecte - debut acquisition")
SAMPLES_PER_PACKET = 6
buffer = []
interval_us = 5000
```

```
t_debut_acq = time.ticks_ms()
while time.ticks_diff(time.ticks_ms(), t_debut_acq) < DUREE_ACQUISITION_MS:
    t0 = time.ticks_us()

    x, y, z = lire_acceleration()
    buffer.append((x, y, z))

    if len(buffer) >= SAMPLES_PER_PACKET:
        if ble.connected:
            packet = b""
            for sx, sy, sz in buffer:
                packet += struct.pack("<hhh", sx, sy, sz)
            ble.send(packet)
        buffer.clear()

    elapsed = time.ticks_diff(time.ticks_us(), t0)
    remaining = interval_us - elapsed
    if remaining > 0:
        time.sleep_us(remaining)

# --- Fin d'acquisition, on se met en deep sleep ---
print("Fin acquisition - deep sleep " + str(DUREE_SLEEP_MS // 1000) + "s")
ble._ble.active(False)
time.sleep_ms(100)
deepsleep(DUREE_SLEEP_MS)
```

Code de la réception de données :

```
import asyncio
import struct
import csv
import time
from bleak import BleakScanner, BleakClient
import nest_asyncio

nest_asyncio.apply()

CHAR_UUID = "6E400003-B5A3-F393-E0A9-E50E24DCCA9E"

# ================================================================================
# INITIALISATION DU FICHIER CSV
# ================================================================================

filename = "vibrations.csv"
csvfile = open(filename, 'w', newline='')
writer = csv.writer(csvfile)
writer.writerow(["timestamp", "x", "y", "z"])
print("Enregistrement dans : " + filename)

# ================================================================================
# TRAITEMENT DES DONNEES RECUES
# Horodatage REEL : on note l'instant d'arrivee de chaque paquet avec une
# horloge monotone (perf_counter), au lieu de fabriquer un temps a 200 Hz.
# Les 6 echantillons d'un paquet arrivent ensemble : on les repartit
# regulierement dans l'intervalle reel ecoule depuis le paquet precedent.
# ================================================================================
```

```
sample_count = 0
premier_paquet = None     # instant d'arrivee du tout premier paquet
dernier_t = None          # instant (relatif) du paquet precedent


def handle_data(sender, data):
    global sample_count, premier_paquet, dernier_t

    t_arrivee = time.perf_counter()
    if premier_paquet is None:
        premier_paquet = t_arrivee

    n = len(data) // 6              # nb d'echantillons dans CE paquet
    if n == 0:
        return
    t_paquet = t_arrivee - premier_paquet

    # intervalle reel depuis le paquet precedent, reparti sur n echantillons
    if dernier_t is not None:
        dt_intra = (t_paquet - dernier_t) / n
    else:
        dt_intra = 0.005            # 1er paquet : 5 ms suppose
    dernier_t = t_paquet

    for k in range(n):
        x, y, z = struct.unpack("<hhh", data[k * 6:k * 6 + 6])
        ax, ay, az = x / 512, y / 512, z / 512
        t = round(t_paquet - (n - 1 - k) * dt_intra, 5)
        writer.writerow([t, ax, ay, az])
        sample_count += 1
```

```
    if sample_count % 200 == 0:
        print(str(sample_count) + " echantillons (" + format(t_paquet, ".1f") + "s)")


# ================================================================================
# CONNEXION BLE ET ACQUISITION
# ================================================================================

async def main():
    print("Recherche ESP32-IMU...")
    device = await BleakScanner.find_device_by_name("ESP32-IMU")
    if not device:
        print("Capteur non trouve")
        return

    async with BleakClient(device) as client:
        print("Connecte !")
        await client.start_notify(CHAR_UUID, handle_data)
        await asyncio.sleep(60)       # enregistre pendant 60 secondes
        await client.stop_notify(CHAR_UUID)


try:
    loop = asyncio.get_event_loop()
    loop.run_until_complete(main())
finally:
    csvfile.close()
```

Code de l'analyse de données :

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import kurtosis
from scipy.signal import spectrogram


# ================================================================================
# 1. CHARGEMENT DES DONNEES
# ================================================================================

df = pd.read_csv("vibrations.csv")

t = df["timestamp"].values
x = df["x"].values
y = df["y"].values
z = df["z"].values

Fe = (len(t) - 1) / (t[-1] - t[0])
print("Frequence d'echantillonnage effective : {:.2f} Hz".format(Fe))
print("Frequence de Nyquist (max observable)  : {:.2f} Hz".format(Fe / 2))


# ================================================================================
# 2. SIGNAL TEMPOREL
# ================================================================================
```

```
fig, axes = plt.subplots(3, 1, figsize=(12, 6), sharex=True)
fig.suptitle("Signal temporel", fontsize=14)

axes[0].plot(t, x, color="royalblue", linewidth=0.5)
axes[0].set_ylabel("X (g)")
axes[0].set_ylim(-0.2, 0.2)
axes[1].plot(t, y, color="tomato", linewidth=0.5)
axes[1].set_ylabel("Y (g)")
axes[1].set_ylim(-0.2, 0.2)
axes[2].plot(t, z, color="seagreen", linewidth=0.5)
axes[2].set_ylabel("Z (g)")
axes[2].set_ylim(0.8, 1.1)
axes[2].set_xlabel("Temps (s)")

plt.tight_layout()
plt.show()


# ============================================================================
# 3. SPECTRE FREQUENTIEL (FFT)
# ============================================================================

def plot_fft(signal, Fe, label, color):
    N = len(signal)
    signal_centre = signal - np.mean(signal)
    fft = np.abs(np.fft.rfft(signal_centre)) / N
    freqs = np.fft.rfftfreq(N, d=1 / Fe)
    plt.plot(freqs, fft, color=color, linewidth=0.8, label=label)
```

```
fig, ax = plt.subplots(figsize=(12, 4))
fig.suptitle("Spectre frequentiel (FFT)", fontsize=14)
plot_fft(x, Fe, "X", "royalblue")
plot_fft(y, Fe, "Y", "tomato")
plot_fft(z, Fe, "Z", "seagreen")
plt.xlabel("Frequence (Hz)")
plt.ylabel("Amplitude (g)")
plt.legend()
plt.ylim(0, 0.008)
plt.tight_layout()
plt.show()


# ============================================================================
# 4. RMS
# ============================================================================

fenetre = int(round(Fe))   # ~1 seconde d'echantillons
rms_x, rms_y, rms_z, temps_rms = [], [], [], []

for i in range(0, len(x) - fenetre, fenetre):
    rms_x.append(np.sqrt(np.mean(x[i:i + fenetre] ** 2)))
    rms_y.append(np.sqrt(np.mean(y[i:i + fenetre] ** 2)))
    rms_z.append(np.sqrt(np.mean(z[i:i + fenetre] ** 2)))
    temps_rms.append(t[i])

fig, ax = plt.subplots(figsize=(12, 3))
ax.plot(temps_rms, rms_x, label="RMS X", color="royalblue")
ax.plot(temps_rms, rms_y, label="RMS Y", color="tomato")
ax.plot(temps_rms, rms_z, label="RMS Z", color="seagreen")
```

```
ax.set_xlabel("Temps (s)")
ax.set_ylabel("RMS (g)")
ax.set_title("RMS glissant - fenetre de 1 seconde")
ax.legend()
ax.set_ylim(0, 1.2)
plt.tight_layout()
plt.show()


# ============================================================================
# 5. KURTOSIS
# ============================================================================

kurt_x, kurt_y, kurt_z = [], [], []

for i in range(0, len(x) - fenetre, fenetre):
    kurt_x.append(kurtosis(x[i:i + fenetre], fisher=True))
    kurt_y.append(kurtosis(y[i:i + fenetre], fisher=True))
    kurt_z.append(kurtosis(z[i:i + fenetre], fisher=True))

fig, ax = plt.subplots(figsize=(12, 3))
ax.plot(temps_rms, kurt_x, label="Kurtosis X", color="royalblue")
ax.plot(temps_rms, kurt_y, label="Kurtosis Y", color="tomato")
ax.plot(temps_rms, kurt_z, label="Kurtosis Z", color="seagreen")
ax.axhline(y=0, color="gray", linestyle="--", label="Normal (0, loi gaussienne)")
ax.axhline(y=3, color="orange", linestyle="--", label="Alerte (exces de 3)")
ax.set_xlabel("Temps (s)")
ax.set_ylabel("Kurtosis (exces, Fisher)")
ax.set_title("Kurtosis glissant - fenetre de 1 seconde")
```

```
ax.legend()
ax.set_ylim(-2, 12)
plt.tight_layout()
plt.show()


# ================================================================================
# 6. SPECTROGRAMME
# ================================================================================

nperseg = min(256, len(x))

fig, axes = plt.subplots(3, 1, figsize=(12, 8))
fig.suptitle("Spectrogramme", fontsize=14)

for ax, signal, label, cmap in zip(
    axes, [x, y, z], ["X", "Y", "Z"], ["Blues", "Reds", "Greens"]
):
    f, t_spec, Sxx = spectrogram(signal - np.mean(signal), fs=Fe, nperseg=nperseg)
    ax.pcolormesh(t_spec, f, 10 * np.log10(Sxx + 1e-10), cmap=cmap, shading="gouraud")
    ax.set_ylabel("{} - Freq (Hz)".format(label))
    ax.set_ylim(0, Fe / 2)   # toute la bande observable, plus de 50 Hz en dur

axes[-1].set_xlabel("Temps (s)")
plt.tight_layout()
plt.show()
```