ESTIMATION DE LA HAUTEUR DES SAUTS À PARTIR DES DONNÉES DE CENTRALES INERTIELLES (IMU)

SOUTENANCE MSR

Présenté par : **Mme Maïva MAGNIFOUET**
Etudiant-Ingénieur (Ucac-Icam)

Dr. Jérôme ROCHETEAU
Enseignant-Chercheur
ICAM Nantes

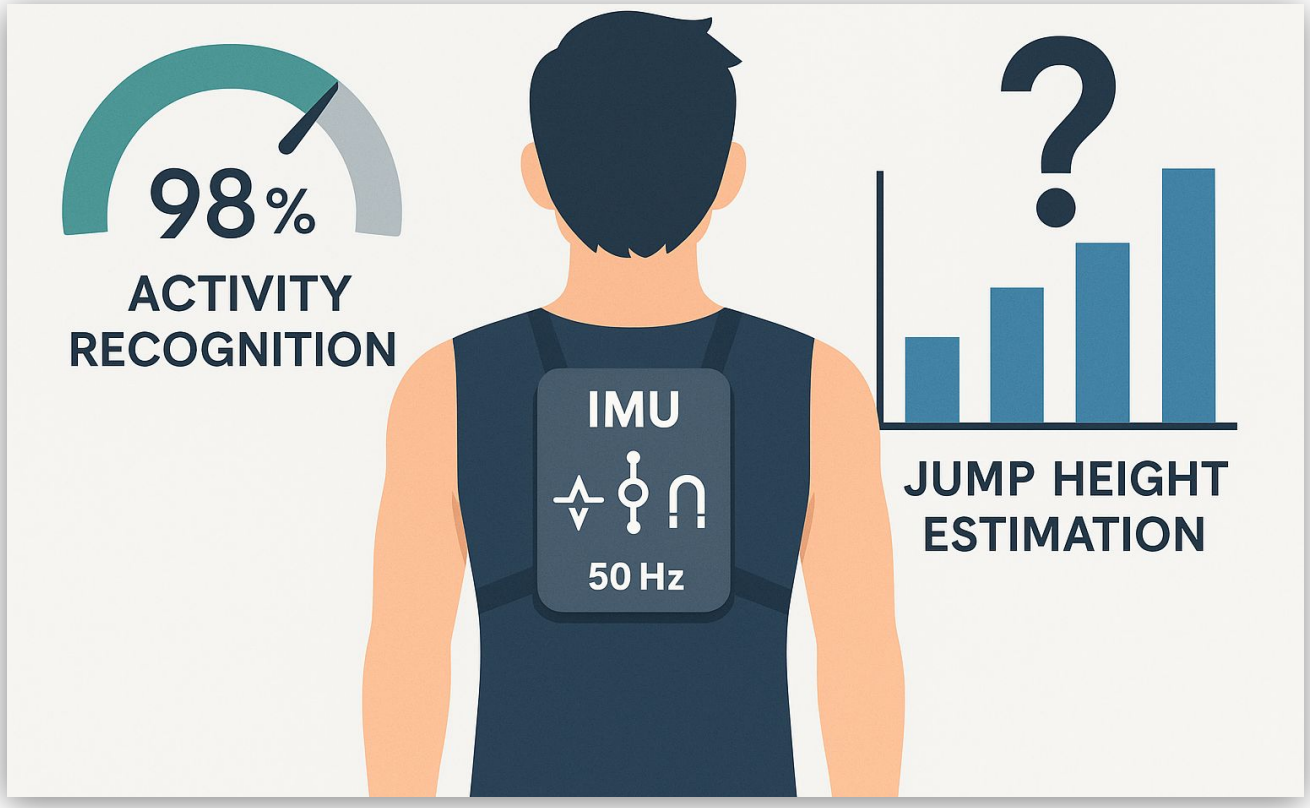
Mr Igor EWOLO
Tuteur IT
UCAC-ICAM

SOMMAIRE

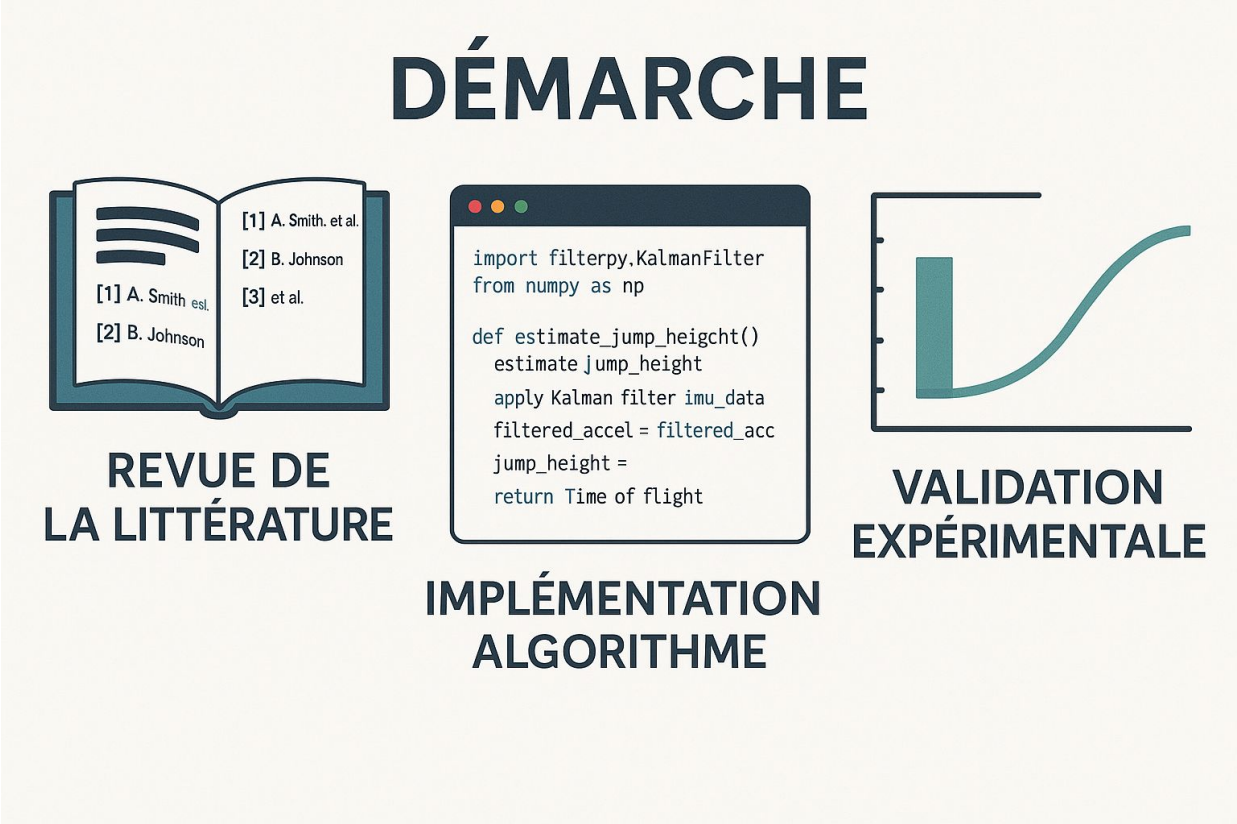
INTRODUCTION

Contexte

Problématique



Démarches


DÉMARCHE
[1] A. Smith et al.
[2] B. Johnson
[3] et al.
[1] A. Smith et al.
[2] B. Johnson
REVUE DE LA LITTÉRATURE
import filterpy.KalmanFilter
from numpy as np
def estimate_jump_heigcht()
estimate jump_height
apply Kalman filter imu_data
filtered_accel = filtered_acc
jump_height =
return Time of flight
IMPLÉMENTATION ALGORITHME
VALIDATION EXPÉRIMENTALE

Objectifs

- Compter le nombre de sauts
- Détecter les activités (HAR)
- Calculer la durée des activités
- Estimer la hauteur de saut

Contributions

Real-Time Jump Activity Recognition and Jump Height Estimation using IMU

Maiva Schela MAGNIFOUET ZEFACK
Icam Nantes, 44470 Carquefou, France
Email: maiva.magnifouet@2025.icam.fr

Jérôme ROCHETEAU
LS2N, UMR 6004, F-44000 Nantes, France
Icam Ouest, 44470 Carquefou, France
Email: jerome.rocheteau@icam.fr

***Abstract*—**
Accurate recognition of jump activities and estimation of jump height are essential tasks in biomechanical analysis and sports performance research. This study introduces a real-time approach based on a low-cost Inertial Measurement Unit (IMU), to estimate jump height. Two state-of-the-art methods – Time of Flight (ToF) and Double Integration (DI) – are implemented and compared to evaluate their accuracy and applicability in real-world scenarios. Complementary signal processing techniques are also applied to reduce noise and improve reliability. The system is developed in collaboration with Les Neptunes de Nantes, a professional women's volleyball club in France, and aims at serving as a practical tool for field-based analysis. The results support the feasibility of using a wearable IMU to provide accurate real-time jumping activity recognition and jump height estimation.

1. Introduction

Volleyball is a sport where vertical jumps are essential for both offensive and defensive plays. Measuring jump height is a key metric for evaluating athlete performance, monitoring progress, and preventing injuries. Traditional solutions such as force plates and motion capture systems are highly accurate, but often expensive, bulky, and restricted to laboratory settings. Recent advances in wearable sensors, particularly inertial measurement units (IMUs), offer a promising alternative for jump analysis in real-world training conditions. However, challenges such as sensor noise, drift, and high cost still limit their widespread use in sports teams. Many commercial IMUs are relatively expensive and inaccessible for daily use by clubs and athletes. This study aims at developing and validating a real-time system for estimating jump height and recognition of jumps using a low-cost, reliable and custom-built IMU. The sensor includes a triaxial accelerometer and a triaxial gyroscope, which are used to implement two state-of-the-art estimation methods: Time of Flight (ToF) and Double Integration (DI). The system also integrates signal filtering techniques to improve accuracy and reduce the impact of noise. Beyond jump height estimation, our system is designed to recognize jumps performed by volleyball players and to count the number of jumps over time making it a complete solution for field-based performance tracking. The study is conducted in collaboration with Les Neptunes de Nantes, a professional women's volleyball team in France, with the aim of providing them with a practical, affordable, and reliable tool tailored to their training needs. Then, the core research question consists in investigating whether affordable IMU-based devices and algorithms can provide accurate and reliable real-time jump height estimations when jump activities have been detected. We hypothesize that low-cost IMU can show a hight level of performance as well as for jumping activity recognition as for jump repetition counting. We investigate whether low-cost IMU can also offer performance comparable to laboratory-grade systems with regard to jump height estimation. The reminder of this paper is organized as follows. The section 2 presents scientific works related to both human activity recognition and jump height estimation. The section 3 presents dataset construction, model training and evaluation that makes us possible to perform jump activity recognition. The section 4 presents implementations of the 2 aforementioned state-of-the-art algorithms for jump height estimation. The section 5 presents the experiments in order to assess the 2 previous algorithms and discuss the analysis results. This source code related to this paper is publicly available[1].

2. Related Work

Recognition of Jump Activity. The automatic recognition of the activity of jumping and of the different kinds of jumps corresponds to the more generic field of Human Activity Recognition (HAR). HAR consists of automatically detecting which activity human beings are doing among a given set of activities. HAR is classically divided into camera-based HAR [1] and sensor-based HAR [2]. Sensor-based HAR corresponds to HAR whose input data are provided by sensors, smartphone-embedded ones or wearable ones, etc. Such sensors are mainly Inertial Measurement Units (IMUs) i.e. 3-axis accelerometers, 3-axis gyroscopes and 3-axis magnetometers. Sensor-based HAR is known to be more precise and fine-grained than camera-based HAR and it requires less computational resources which makes real-time processing possible. HAR mainly consists of time-series classification i.e. predicting the most probable activity (class) from timestamped data or multivariate time-series. In

1. See the repository at https://vit.icam.fr/jerome.rocheteau/ieeestar

IEEE STAR Trento2025

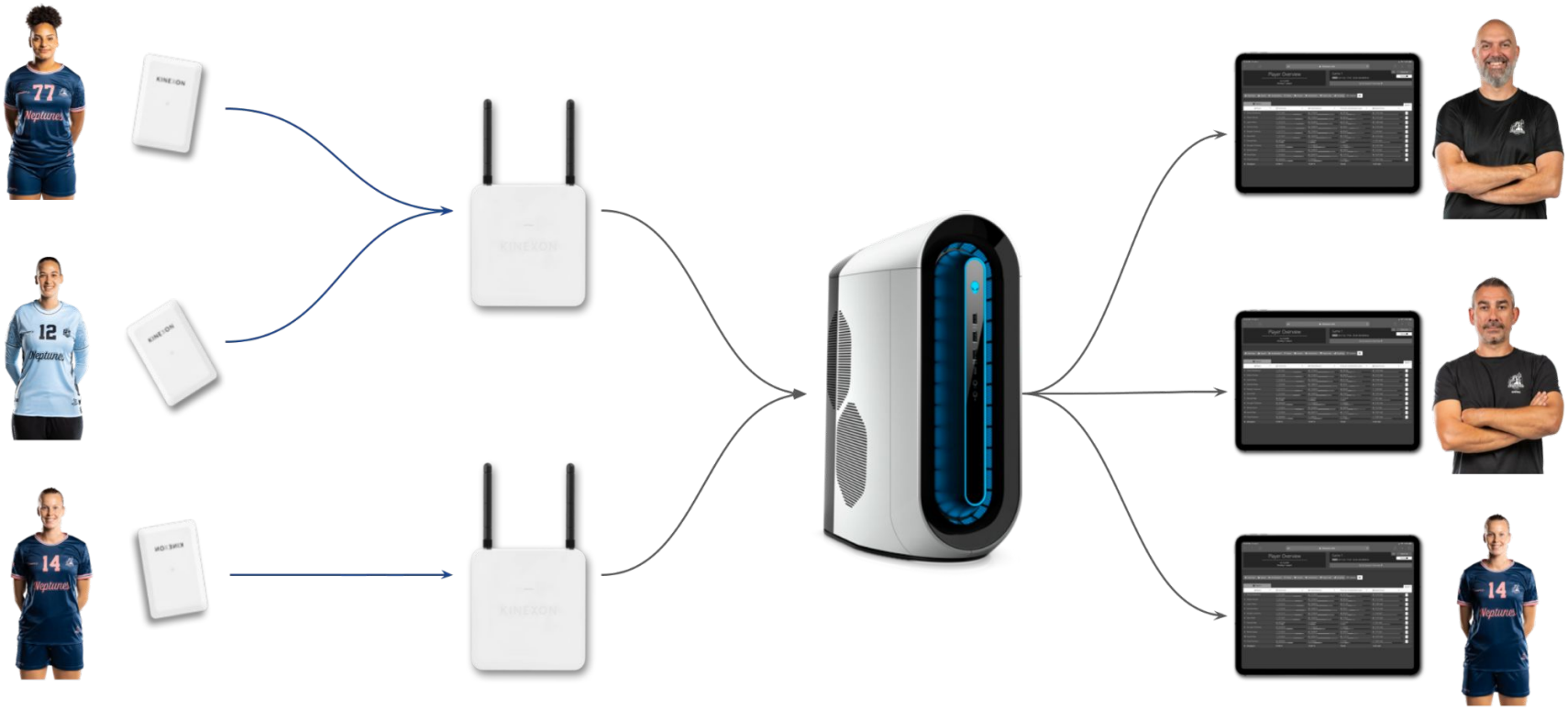
CAS D'ÉTUDE

Besoins

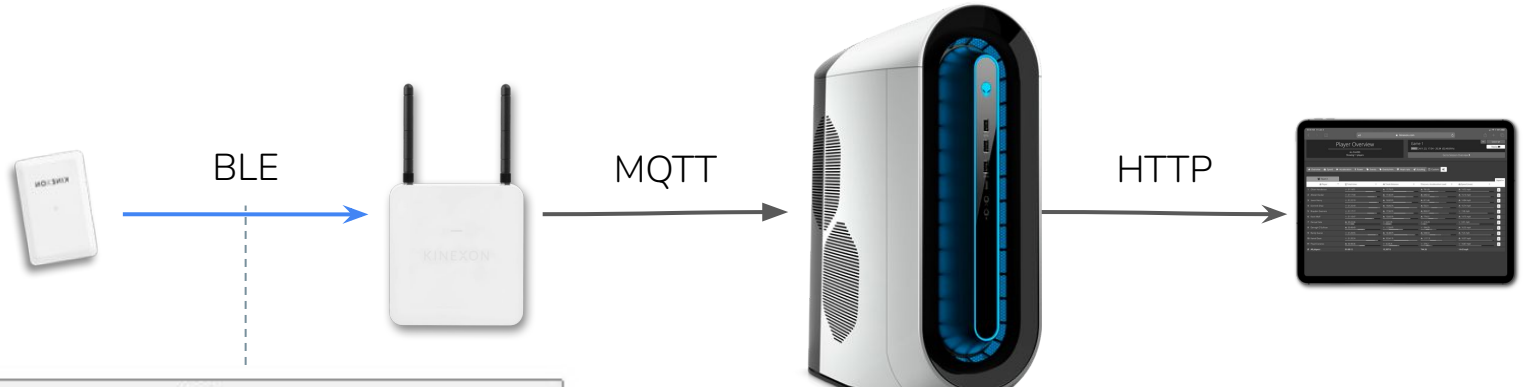
🎯 Suivi de la charge d'entraînement

- Indicateurs sur les **courses**
  - nombre
  - durée
  - distance
  - vitesse
- Indicateurs sur les **sauts**
  - nombre
  - hauteur
- **High Tech** (État de l'art)
- **Low Cost**

Architecture

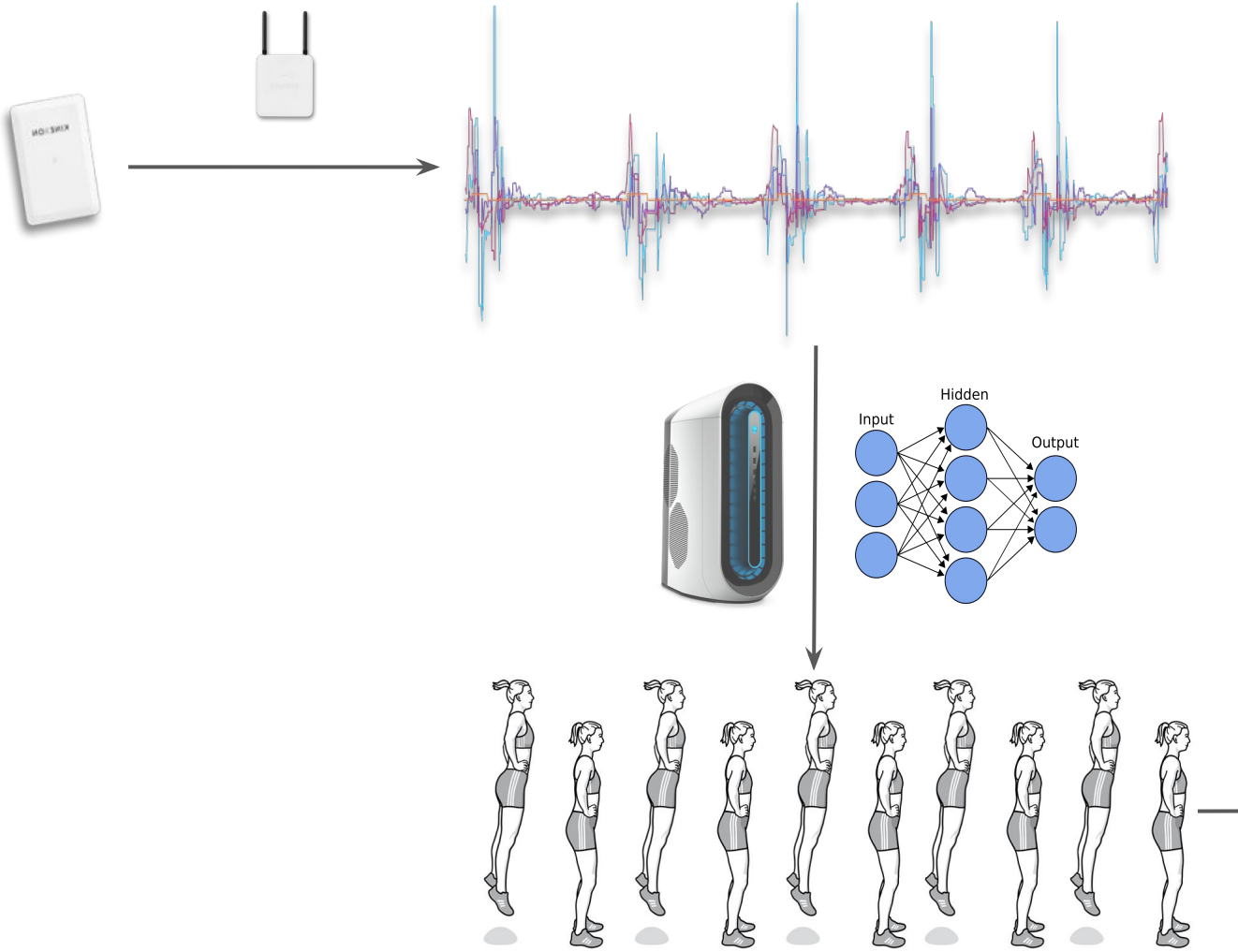


Protocole

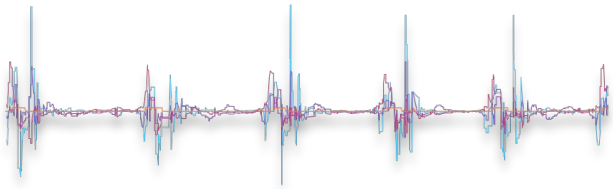


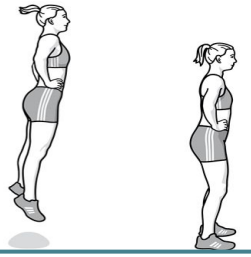
S data
float id : device identifier
float ax : x-axis acceleration float ay : y-axis acceleration float az : z-axis acceleration
float gx : x-axis angular velocity float gy : y-axis angular velocity float gz : z-axis angular velocity
float mx : x-axis magnetic flux density float my : y-axis magnetic flux density float mz : z-axis magnetic flux density
float t : device temperature float v : battery voltage

Hypothèses et Objectifs



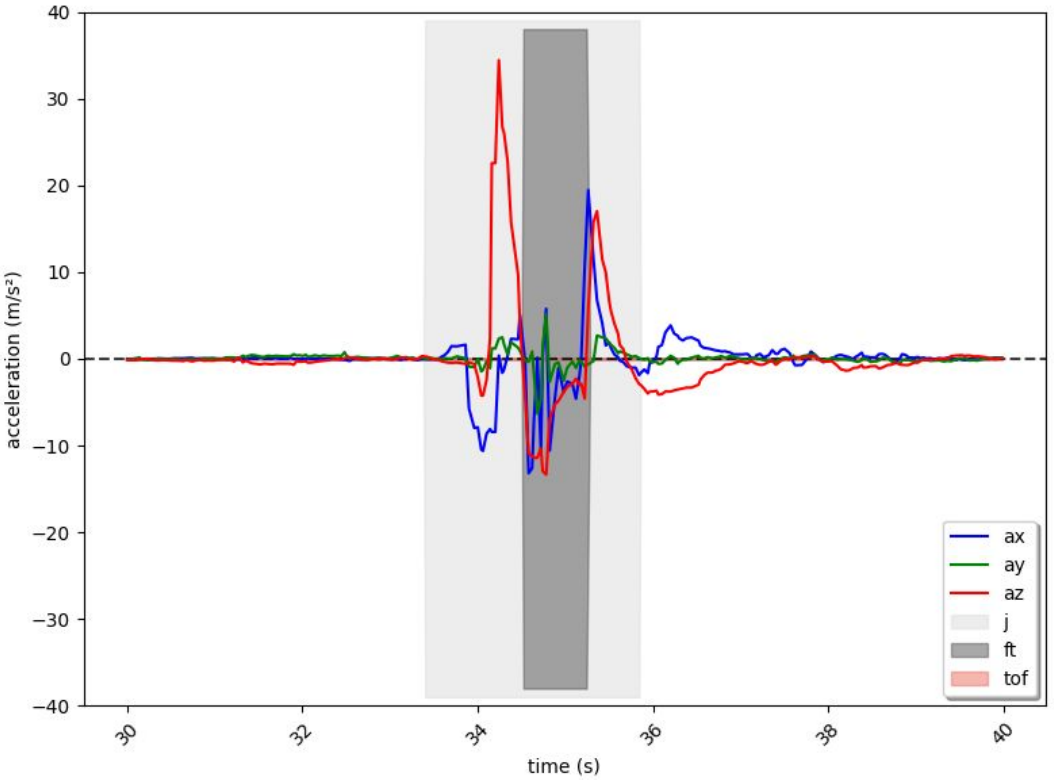
Hypothèses et Objectifs

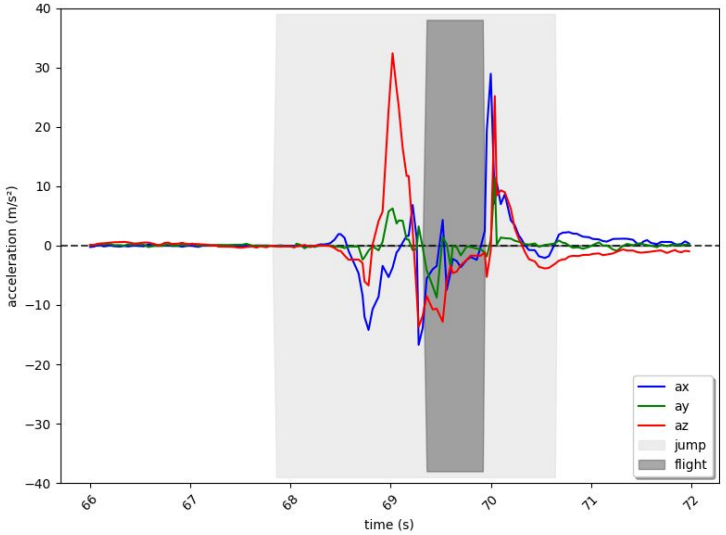




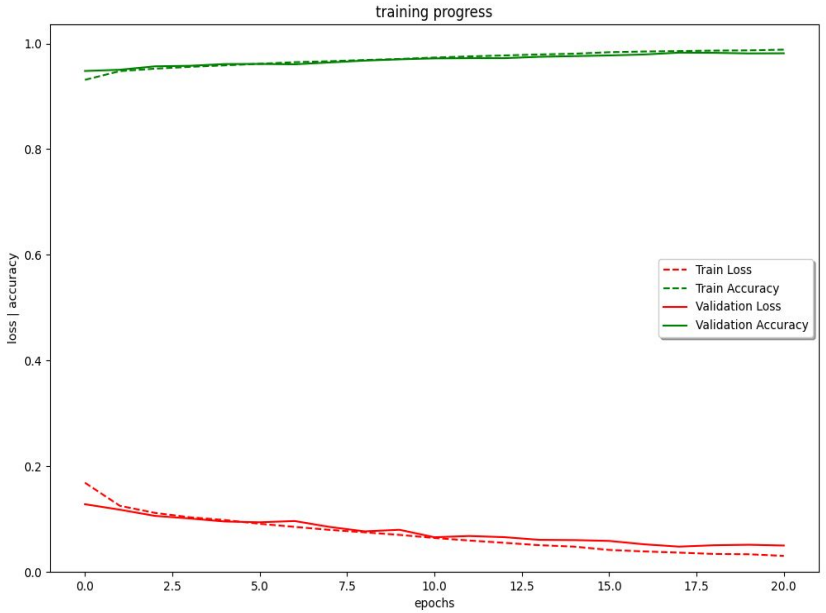
	x-axis	y-axis	z-axis	jumping	standing
0	-0.694638	12.680544	0.503953	1	0
1	5.012288	11.264028	0.953424	1	0
2	4.903325	10.882658	-0.081722	1	0
3	-0.612916	18.496431	3.023717	1	0
4	-1.184970	12.108489	7.205164	1	0
...	...	...	...	...	...
1098204	9.000000	-1.570000	1.690000	0	1
1098205	9.040000	-1.460000	1.730000	0	1
1098206	9.080000	-1.380000	1.690000	0	1
1098207	9.000000	-1.460000	1.730000	0	1
1098208	8.880000	-1.330000	1.610000	0	1

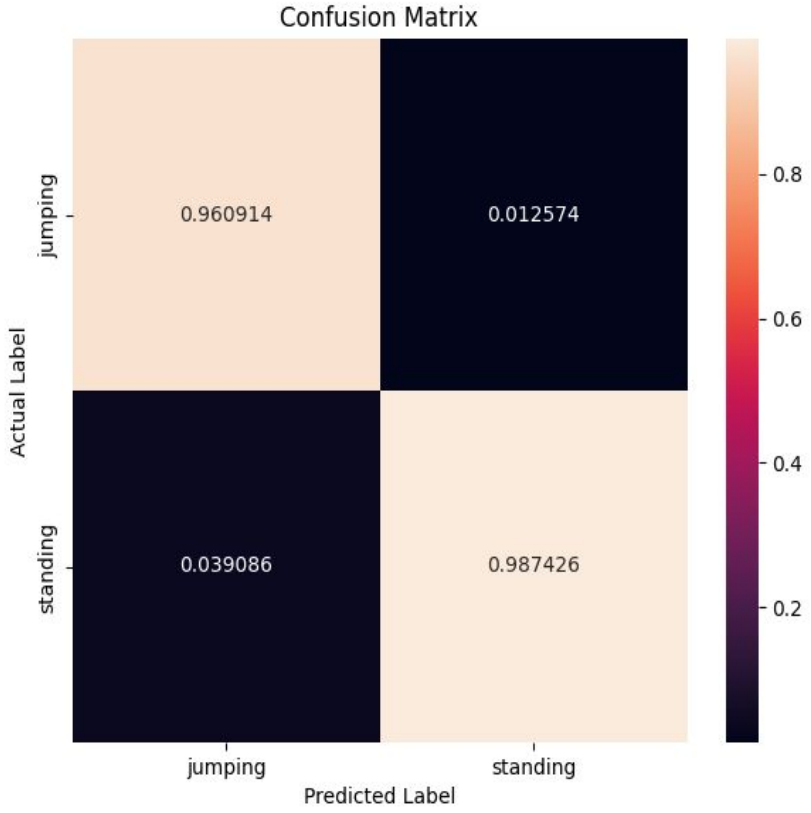
jumping


acceleration (m/s²)
time (s)
ax
ay
az
j
ft
tof


acceleration (m/s²)
time (s)
ax
ay
az
jump
flight

Matrice de Confusion

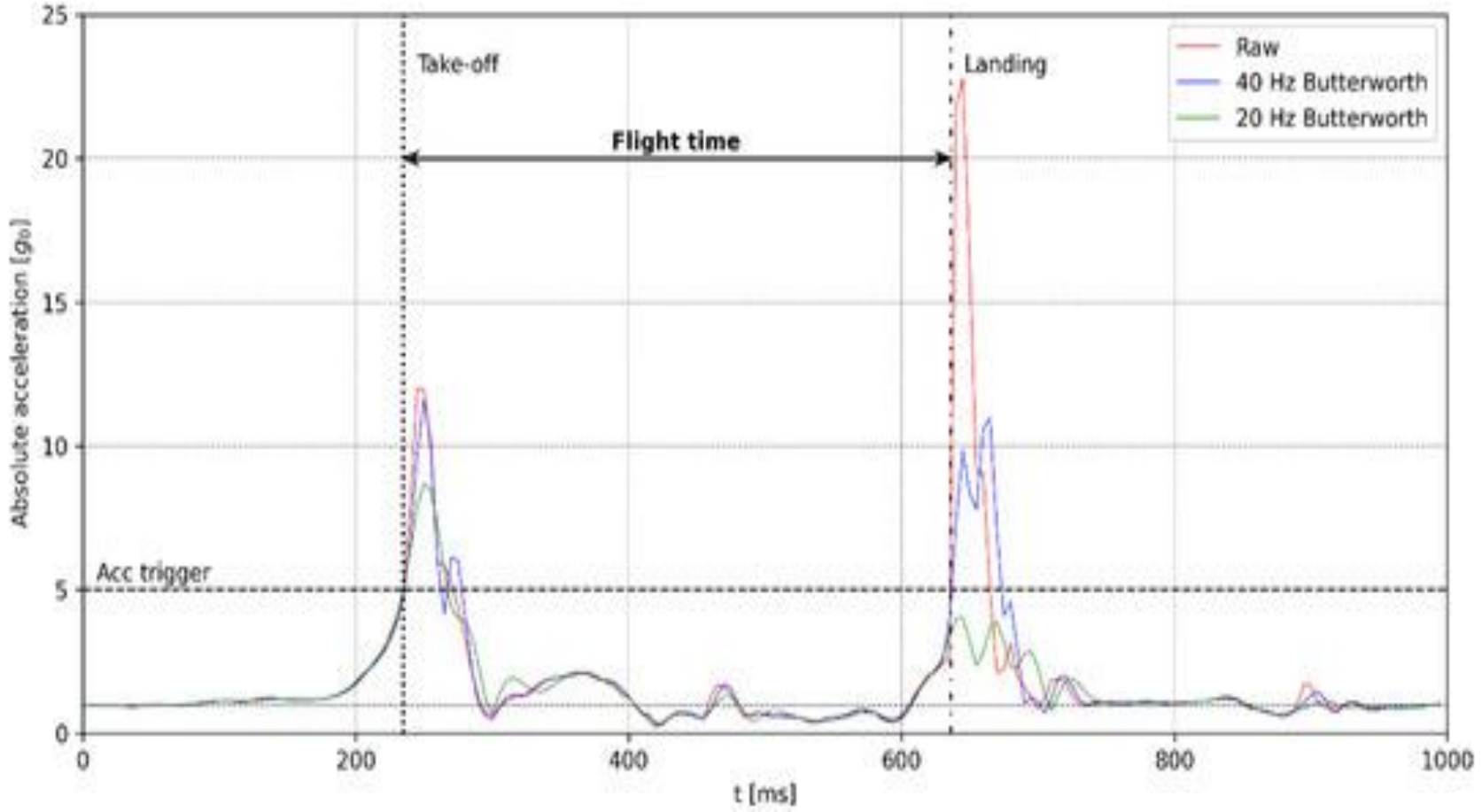




– 50 Hz – 20ms 5,03% 98.1%

ÉTAT DE L’ART

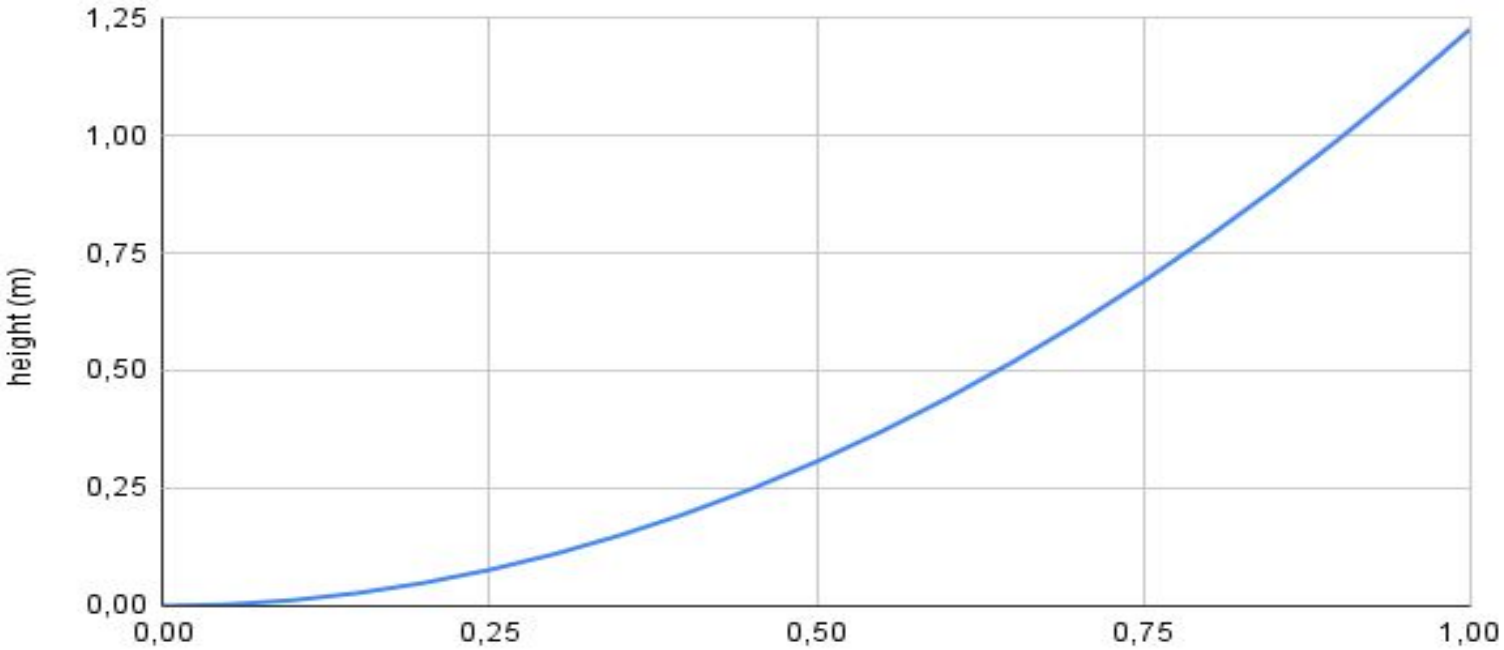
Etat de l'Art



Stefan Marković, Milivoj Dopsaj, Anton Kos, Aleksandar Nedeljković and Anton Umek **Can IMU Provide an Accurate Vertical Jump Height Estimate?,** *Appl. Sci.* **2021**, *11*(24), 12025; **17 December 2021**

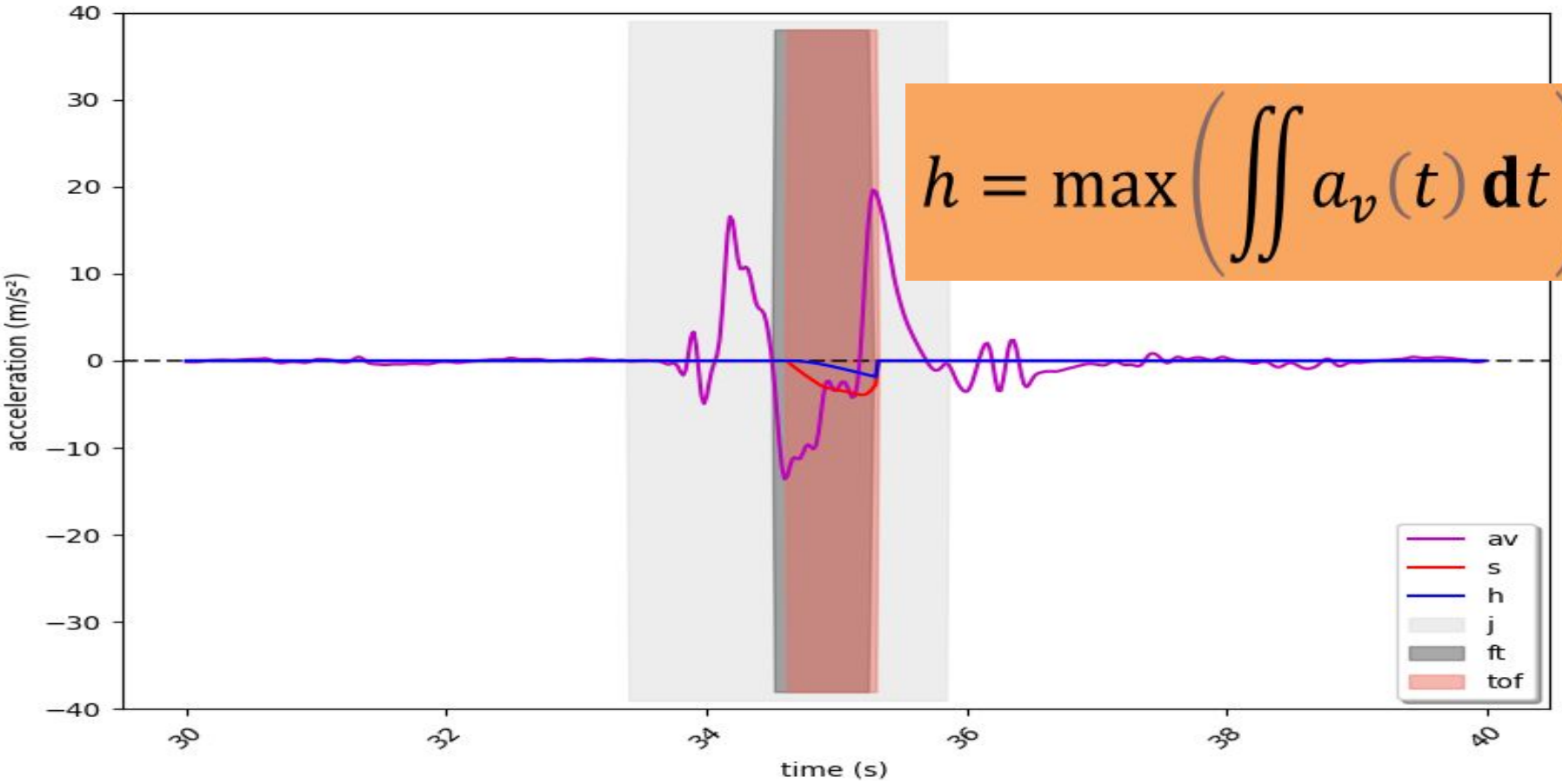
Time of Flight (ToF)

$$h = \frac{gt^2}{8}$$



Max Schmarzo Jump height and time in air relationship, *AStrong by science*;**19 March 2017**

Double Intégration (DI)



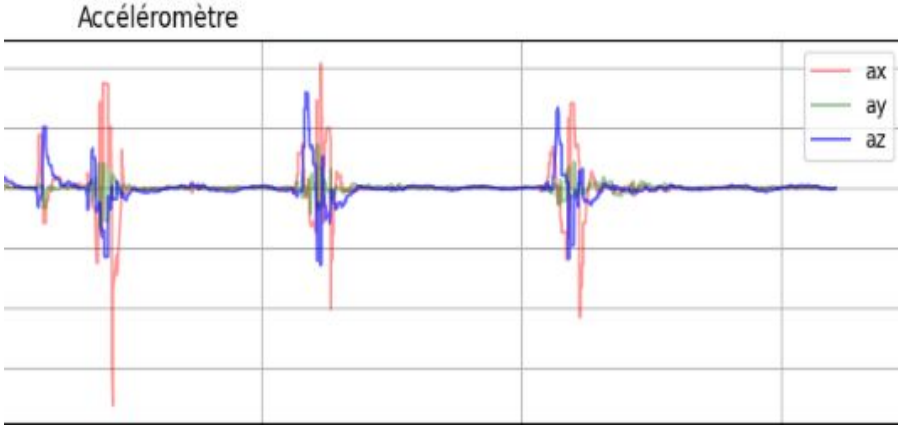
$$h = \max\left(\iint a_v(t)\,\mathbf{d}t\right)$$

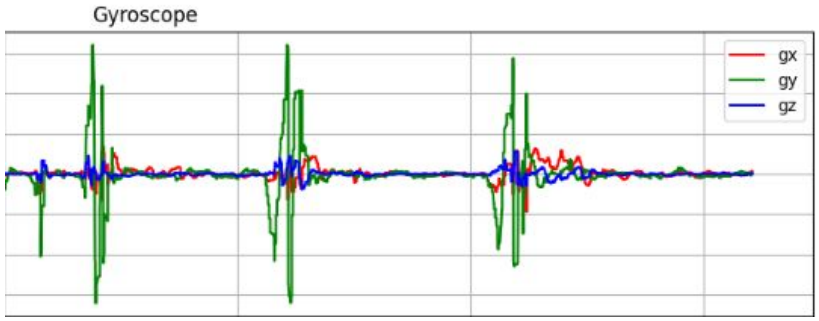
ALGORITHMES

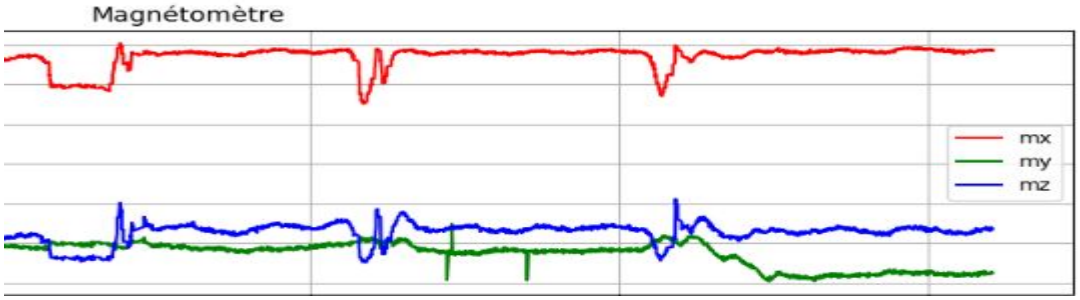
Implementation

1. Acquisition des données
2. Correction de l’orientation
3. Filtrage de l'accélération verticale
4. Détection du décollage et atterrissage
5. Estimation de la hauteur de saut

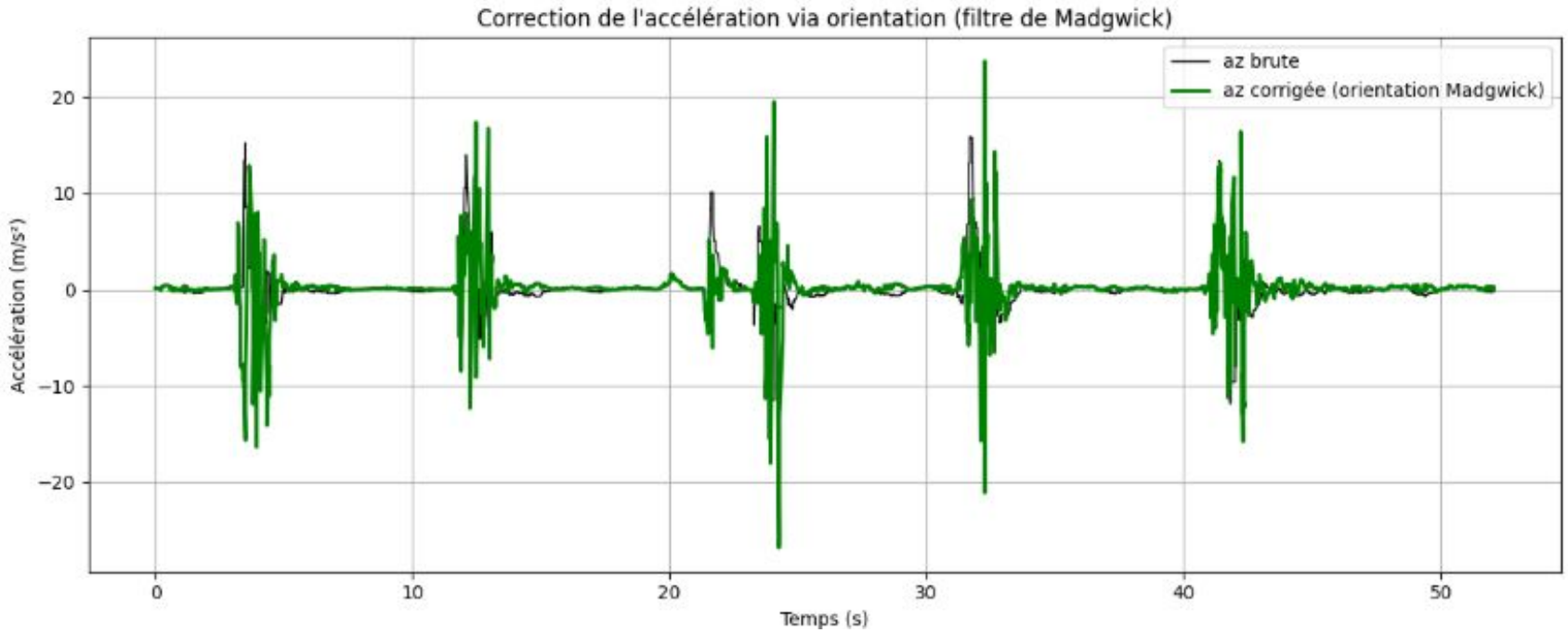
Acquisition des données



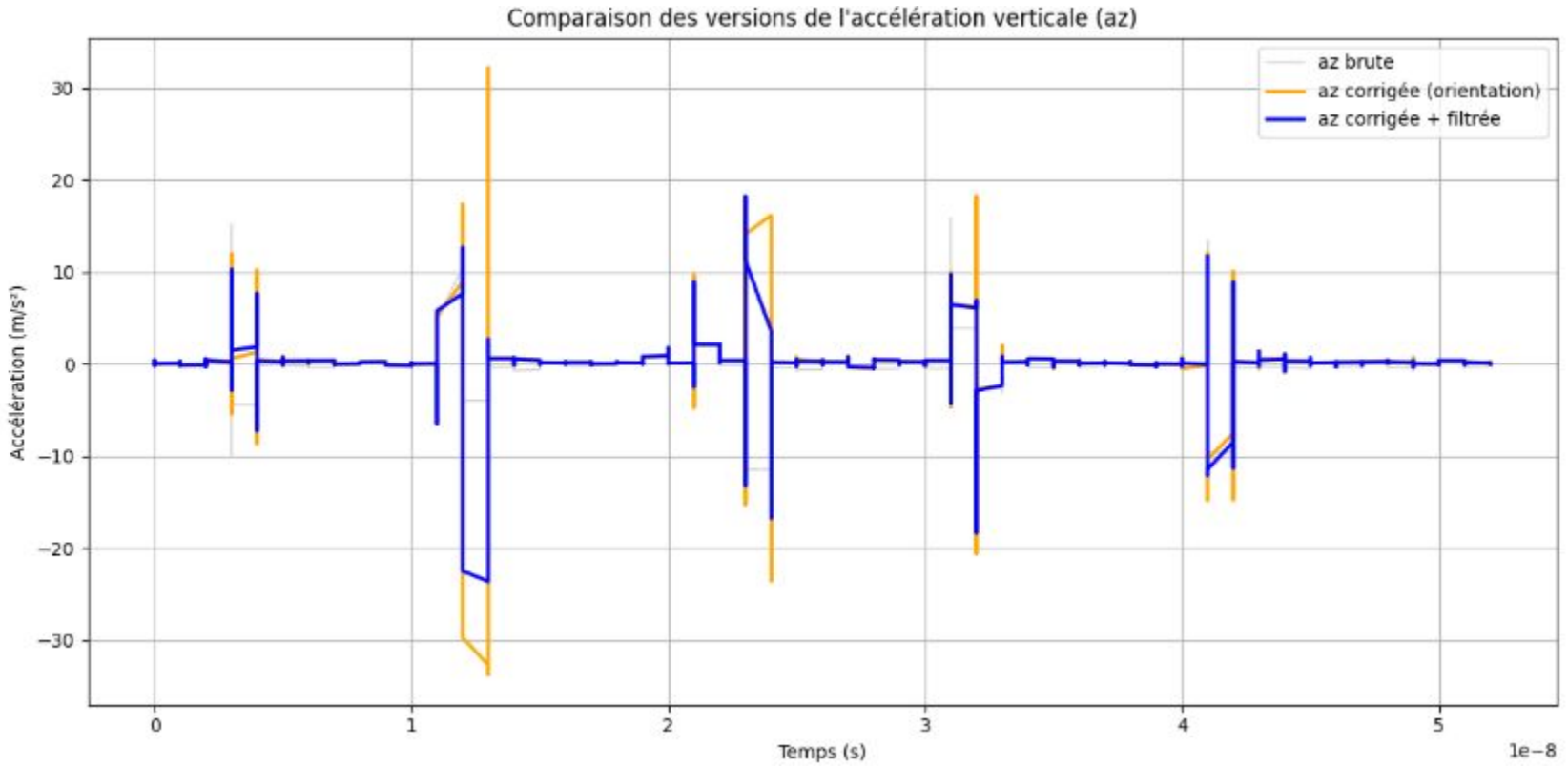




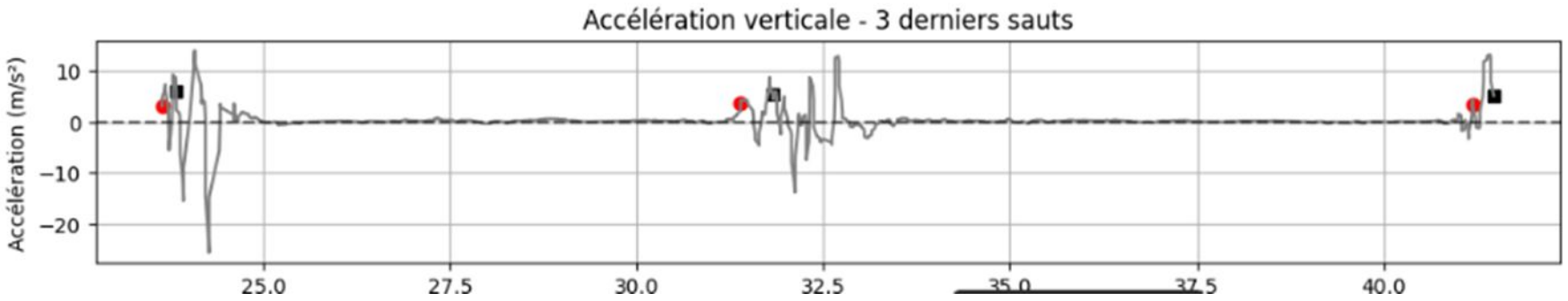
Correction acc. (Filtre de Madgwick)



filtrage(Madgwick + ButterWorth)



Détermination des index



Comparaison des Méthodes

Critère	Time of Flight (ToF)	Double Intégration (DI)
Facilité d'implémentation	✅ Simple	❌ Complexe (intégrations, filtres)
Précision brute	❌ Moyenne	✅ Potentiellement élevée
Sensibilité au bruit	✅ Faible	❌ Très forte
Besoin en filtrage	✅Faible	❌ Élevé (Butterworth, Kalman)

Algorithmes

```
    Exceptions :
    -----------
    FileNotFoundError :
        Si le fichier spécifié n'existe pas.
    ValueError :
        Si le fichier CSV est vide ou mal formaté.
    """
    try:
        # Lecture du fichier CSV sans en-tête
        data = pd.read_csv(path, header=None, names=columns)

        # Suppression des lignes contenant des valeurs manquantes
        data = data.dropna()

        return data
    except FileNotFoundError as e:
        print(f"Erreur : Le fichier spécifié '{path}' est introuvable.")
        raise e
    except pd.errors.EmptyDataError as e:
        print(f"Erreur : Le fichier '{path}' est vide ou mal formaté.")
        raise e

# Appel de la fonction pour lire et nettoyer les données
data = read_data(PATH, COLUMNS)


# Affichage des premières lignes du DataFrame
#print(data.head())
```

```
def butter_lowpass_filter(data, cutoff= cutoff, fs=fs, order=order):
    """
    Applique un filtre passe-bas de Butterworth sur un signal pour réduire le bruit.

    Paramètres :
    ------------
    data : array-like
        Signal d'entrée à filtrer (doit être un tableau NumPy ou une liste).
    cutoff : float
        Fréquence de coupure du filtre en Hz.
    fs : float
        Fréquence d'échantillonnage en Hz.
    order : int, optional
        Ordre du filtre Butterworth (par défaut : 4).

    Retourne :
    ----------
    filtered_data : array
        Signal filtré après application du filtre passe-bas.

    Exceptions :
    ------------
    - ValueError : Si fs <= 0 ou cutoff <= 0 ou cutoff >= fs / 2.
    - TypeError : Si data n'est pas un tableau ou une liste.
    """
```

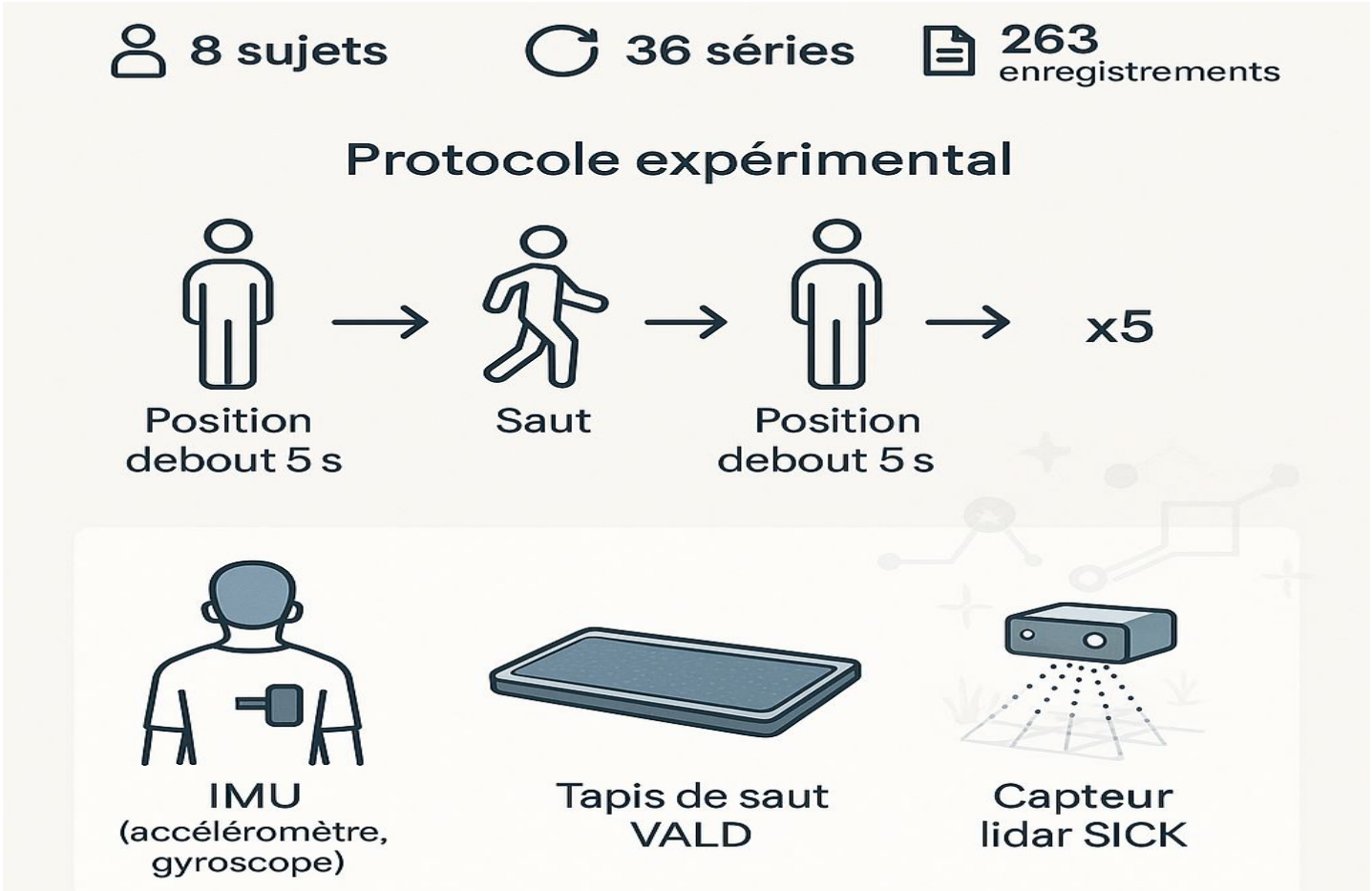
```
def madgwick_filter(data, beta):
    """
    Implémentation d'un filtre de Madgwick simplifié pour la fusion des données IMU.
    Ce filtre estime l'orientation sous forme de quaternions à partir des mesures
    d'un accéléromètre, gyroscope et magnétomètre.

    Paramètres :
    ------------
    data : DataFrame
        Données contenant les mesures des capteurs IMU avec les colonnes :
        ['ax', 'ay', 'az', 'gx', 'gy', 'gz', 'mx', 'my', 'mz', 'timestamp']
    beta : float
        Coefficient de fusion du filtre Madgwick. Contrôle l'équilibre entre la
        prédiction gyroscopique et la correction basée sur les autres capteurs.

    Retourne :
    ----------
    q_history : ndarray
        Tableau des quaternions estimés au cours du temps (N, 4), avec N le nombre d'échantillons.
    data : DataFrame
        Données mises à jour avec la colonne 'dt' représentant l'intervalle de temps.

    Exceptions :
    ------------
    - ValueError : Si des colonnes nécessaires sont absentes.
    - TypeError : Si 'data' n'est pas un DataFrame.
    """

    # Vérification des entrées
    required_columns = ['ax', 'ay', 'az', 'gx', 'gy', 'gz', 'mx', 'my', 'mz', 'timestamp']
    if not isinstance(data, pd.DataFrame):
        raise TypeError("L'entrée 'data' doit être un DataFrame Pandas.")
    if not required_columns.issubset(data.columns):
        raise ValueError(f"Le DataFrame doit contenir les colonnes suivantes : {required_columns}")

    # Conversion des valeurs en numérique (pour éviter les erreurs de type)
    for axis in required_columns - {'timestamp'}:
        data[axis] = pd.to_numeric(data[axis], errors='coerce')

    # Conversion des vitesses angulaires de degrés/s en radians/s
    for axis in ['gx', 'gy', 'gz']:
        data[axis] = np.radians(data[axis])

    # Traitement du temps
    data = data.reset_index(drop=True)  # Réinitialisation des index
    data['timestamp'] = pd.to_datetime(data['timestamp'])  # Conversion en datetime
    data['timestamp'] = (data['timestamp'] - data['timestamp'][0]).dt.total_seconds()  # Temps en secondes
    data['dt'] = data['timestamp'].diff().fillna(0)  # Calcul du pas de temps
```

```
def correct_acceleration_no_filter(data, quaternions):
 # Vérification des entrées
    required_columns = ['ax', 'ay', 'az', 'gx', 'gy', 'gz', 'timestamp']
    if not isinstance(data, pd.DataFrame):
        raise TypeError("L'entrée 'data' doit être un DataFrame Pandas.")
    if not required_columns.issubset(data.columns):
        raise ValueError(f"Le DataFrame doit contenir les colonnes suivantes : {required_columns}")
    if not isinstance(quaternions, np.ndarray) or quaternions.shape[1] != 4:
        raise TypeError("Les quaternions doivent être un ndarray de forme (N, 4).")

    # Réinitialisation de l'index (évite les erreurs lors de l'itération)
    data = data.reset_index(drop=True)

    # Conversion des colonnes en float (gestion des valeurs texte et numériques)
    for col in ['ax', 'ay', 'az', 'gx', 'gy', 'gz']:
        data[col] = pd.to_numeric(data[col], errors='coerce')

    # Gestion des valeurs manquantes
    if data.isna().any().any():
        print("⚠️ Attention : Certaines valeurs sont manquantes (NaN). Elles seront interpolées.")
        data.interpolate(inplace=True)
        data.fillna(method='bfill', inplace=True)
        data.fillna(method='ffill', inplace=True)


    a_z_global = []

    for i in range(len(data)):
        quat = quaternions[i]
        if abs(quat[0]) > abs(quat[1]):
            r = Rotation.from_quat([quat[1], quat[2], quat[3], quat[0]])
        else:
            r = Rotation.from_quat(quat)
        R = r.as_matrix()
        a_corrected = R @ np.array([data['ax'][i], data['ay'][i], data['az'][i]])
        a_z_global.append(a_corrected[2])

    data['az_madgwick'] = np.array(a_z_global)
```

```
np.ndarray)):
'data' doit être un tableau NumPy ou une liste.")


ce d'échantillonnage 'fs' doit être positive.")

' 2:
ce de coupure 'cutoff' doit être comprise entre 0 et Nyquist (fs/2).")

nécessaire


nnel : interpolation ou suppression)

contient des valeurs NaN, qui seront interpolées.")
en(data)), np.arange(len(data))[~np.isnan(data)], data[~np.isnan(data)])
```

Expérimentations & Résultats

Protocole Expérimental



Critères d'Évaluation des Methodes

📌 **Objectifs:**

- **Évaluer la fiabilité des estimations de hauteur de saut par IMU**,
  en comparant **Double Intégration (DI)** et **Time of Flight (ToF)**
  avec un **outil de référence** : le **tapis de saut VALD**
- Évaluer la précision de la **détection des phases du saut** (décollage, atterrissage, temps de vol), en les comparant aux données du **tapis VALD** mais aussi du **Lidar SICK**,
- compter correctement les sauts dans chaque série

1. MAE – Mean Absolute Error (Erreur absolue moyenne)

C’est la moyenne des distances absolues entre les valeurs estimées et les valeurs réelles.

📐 **Formule :**

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n} |\hat{y}_i - y_i|$$

- y^i:valeur estimée (par IMU)
- yî : valeur de référence (ex : tapis de saut)
- nnn : nombre de sauts évalués

🧠 **À retenir :**

- Facile à comprendre
- Donne une idée directe de **l'erreur moyenne en mètres**
- Moins sensible aux erreurs extrêmes

2. **RMSE – Root Mean Squared Error**

C’est la racine carrée du MSE, donc exprimée dans les mêmes unités que la hauteur (mètres).

📐 **Formule :**

$$\mathrm{RMSE} = \sqrt{\mathrm{MSE}} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}$$

🧠 **À retenir :**

- Interprétation facile car **en mètres**
- Sensible aux erreurs importantes (comme MSE), mais comparable directement à MAE
- C’est souvent celle qu’on regarde en premier dans un tableau de résultats

Évaluation des Méthodes

Méthode / Mesure	MAE (m)	RMSE (m)	R^2
ToF jump height	0.362	0.283	–
DI jump height	0.389	0.299	–
Lidar time-of-flight	0.203	0.165	–
Comptage de sauts (IMU)	0.024	0.003	0.993
Take-off / landing index	0.002	0.002	0.999

Evaluation de l'estimation de la Hauteur de saut

Analyse des Résultats

📌 Aspect évalué	🔍 Analyse rapide
🧮 Comptage des sauts	✅ Fiable – 1 erreur sur 66 séries
📡 Lidar (40 Hz)	⚠️ Imprécis – surestime le temps de vol (~55 ms)
🛫 Take-off / Landing	⚠️ Imprécis – par rapport au lidar
📏 Hauteur ToF / DI	⚠️ Erreurs – bruit, dérive, hypothèses simplistes

CONCLUSION

Bilan du Stage

🎯 Objectif	Résultat obtenu
Compter le nombre de sauts	✔️ Fiable, 2 erreurs sur 66 séries
Détecter les activités (HAR)	✔️ Reconnaissance >98 % (IMU à 50 Hz)
Calculer la durée des activités	✔️ Segmentation précise des phases
Estimer la hauteur de saut	✖️ Trop d'erreurs, bruit, dérive, mauvaise sensibilité du capteur

Perspectives

📈 Augmenter la fréquence d’échantillonnage (≥100 Hz)

📈 Mettre en place un système de fusion de capteurs

📈 Intégrer dynamiquement les données du tapis de saut à notre plateforme

📱 Développement d’une application embarquant l’algorithme

MERCI POUR VOTRE ATTENTION

Questions fréquentes et réponses anticipées

❓Pourquoi avoir utilisé un IMU à 50 Hz ?

🎯 Parce que 50 Hz est suffisant pour détecter les phases du saut (HAR).
Cela permet d'avoir un système moins coûteux, plus léger et autonome.

❓Pourquoi ne pas avoir utilisé une méthode machine learning ?

🔍 Parce que l'objectif était d'évaluer des méthodes **physiques interprétables** (ToF et DI).
Une approche ML pourrait être envisagée plus tard, sur une base plus large de données labellisées.

Questions fréquentes et réponses anticipées

❓Pourquoi les erreurs sont-elles élevées pour la hauteur ?

⚠️ À cause :

- du **bruit de mesure** (même après filtrage)
- de la **dérive d'intégration** (DI)
- et des **hypothèses simplificatrices** (ToF suppose un vol symétrique).

Questions fréquentes et réponses anticipées

❓ Pourquoi le Lidar est-il moins fiable que le tapis ?

📡 Parce qu'il fonctionne à **40 Hz**, donc il a du mal à détecter précisément les instants de décollage/atterrissage.

❓ Est-ce que votre système peut être utilisé en match réel ?

🏐 Oui, le système est portable et fonctionne avec une fréquence d'échantillonnage modeste.
Mais il reste encore à optimiser **la précision de la hauteur** pour une exploitation complète sur le terrain.