Suivi de la déformation de structures avec accéléromètre et gyroscope

June 16, 2023

Antonin Lefevre
antoninlefevre45@icloud.com

Introduction

Ce projet se base sur des boîtiers, chacun constitué d'un ESP32 et d'un capteur MPU6050. Le capteur MPU6050 est capable de mesurer l'accélération et l'inclinaison sur trois axes. Chacun de ces boîtiers est connecté au réseau local et envoie ses données au serveur MQTT, en publiant sur un topic qui lui est propre. Le serveur s'abonne aux topics des capteurs et enregistre les données dans une collection MongoDB.

La partie cruciale du projet est l'algorithme de traitement des données, qui récupère les dernières données des capteurs ainsi que les dernières positions connues des boîtiers. Il calcule ensuite l'inclinaison et la déformation de la structure pour obtenir les nouvelles positions des boîtiers. Ces nouvelles positions sont affichées en temps réel sur Streamlit, fournissant une visualisation dynamique et intuitive de l'état de la structure.

Le code de ce projet est disponible sur Github.

Prérequis

Pour reproduire ce projet de surveillance structurelle, un certain nombre de conditions préalables doivent être remplies. En termes de matériel, vous aurez besoin de plusieurs modules ESP32. Ces puces sont essentielles pour recueillir les données du capteur MPU6050 et les transmettre au serveur via le WiFi intégré. En parlant de capteurs, des capteurs MPU6050 sont également nécessaires. Ces capteurs sont capables de mesurer l'accélération et l'inclinaison sur trois axes. Pour les connecter aux modules ESP32, des câbles de raccordement sont nécessaires. Enfin, les modules ESP32 doivent être alimentés. Des batteries rechargeables ou une source d'alimentation constante peuvent être utilisées selon vos préférences et vos besoins.

Sur le plan logiciel, vous aurez besoin d'un environnement pour programmer vos modules ESP32, comme par exemple Arduino IDE. En ce qui concerne l'algorithme de traitement des données, il est écrit en Python. Vous devrez donc avoir une version de Python installée sur votre ordinateur. Cet algorithme utilise également plusieurs bibliothèques Python, notamment numpy, pandas, paho-mqtt, pymongo, plotly, fastdist et streamlit. Assurez-vous de les installer également.

Pour la gestion de la transmission des données entre les capteurs et le serveur, nous utilisons Mosquitto, un courtier MQTT open source. De plus, les données des capteurs sont stockées dans une base de données MongoDB, vous devrez donc avoir MongoDB installé et configuré correctement.

Pour vous procurer les capteurs ou les ESP32 vous pouvez regarder sur le site AZ-Delivery

Configuration du matériel

Le matériel de notre système se compose essentiellement des boîtiers, chacun contenant les éléments illustrés dans la Figure 1. Ces éléments comprennent un module ESP32, un capteur MPU6050 et un adaptateur pour connecter une batterie. Cette batterie est chargée d'alimenter le module ESP32 et le capteur.

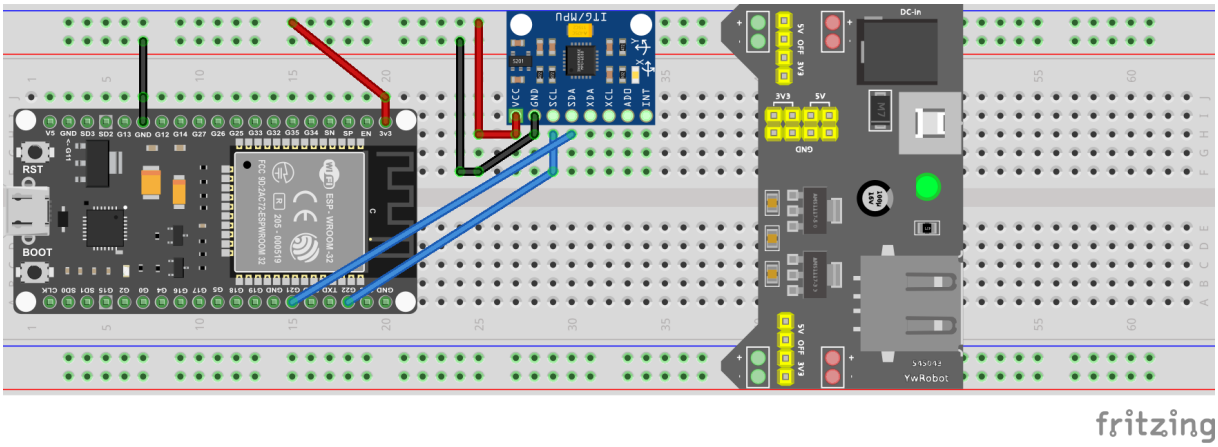

Figure 1: Schéma du cablage.

Il est impératif de vérifier que cet adaptateur est réglé à 3,3V. Une tension supérieure pourrait endommager, voire griller, le capteur et l'ESP32.

La Figure 2 présente une photographie de trois boîtiers une fois entièrement assemblées. Le câblage correspond à celui décrit dans la Figure 1. Cela nous donne un aperçu clair de l'agencement interne de chaque boîtier et de la manière dont les divers éléments sont interconnectés pour former un système de surveillance structurelle complet.

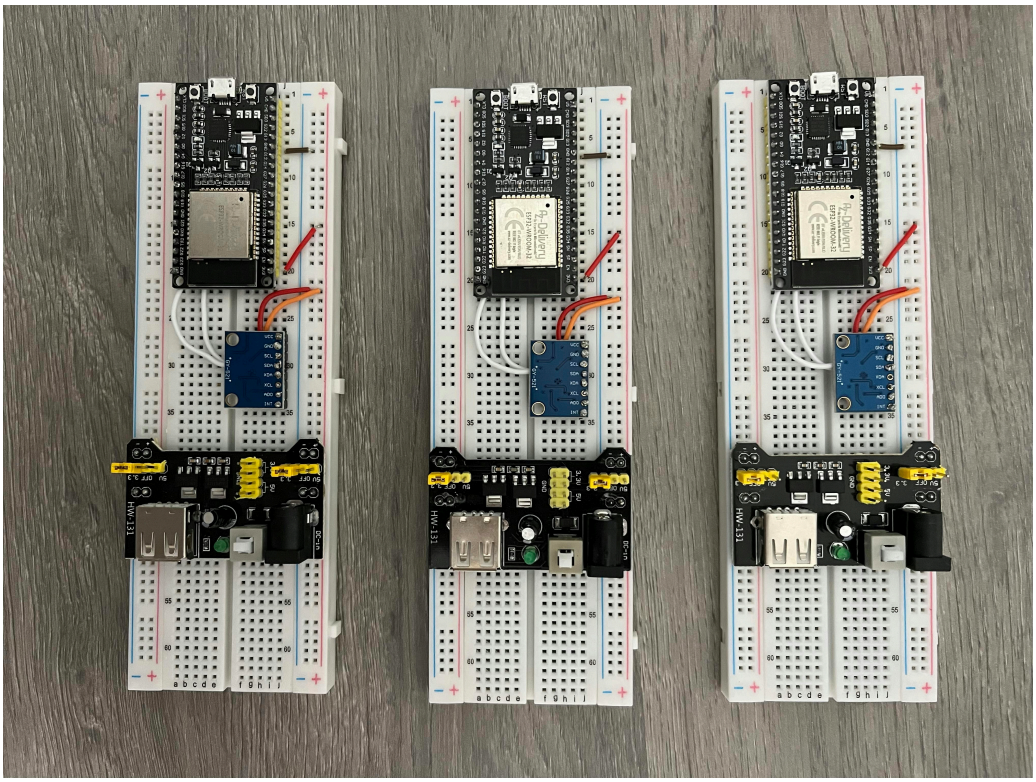

Figure 2: Contenu des boitiers.

Ensuite, c'est à vous de positionner ces boîtiers sur une structure, horizontalement. La seule exigence est que les capteurs soient physiquement connectés. Leurs positions doivent être enregistrées dans la base de données en ordre croissant de leur hauteur.

MQTT

MQTT, pour Message Queuing Telemetry Transport, est un protocole de messagerie léger et efficace qui a été conçu pour les systèmes de communication Machine-à-Machine (M2M) et Internet des Objets (IoT). L'un des atouts de MQTT est son modèle de publication/abonnement qui permet aux messages de parvenir à plusieurs destinataires en même temps, ce qui est idéal pour la surveillance en temps réel des données de capteurs.

Le protocole MQTT fonctionne via un système de « broker » (serveur) et de « clients ». Les clients peuvent être des éditeurs (publishers), qui envoient des messages sur des sujets (topics) spécifiques, et/ou des abonnés (subscribers), qui reçoivent des messages sur les sujets auxquels ils sont abonnés.

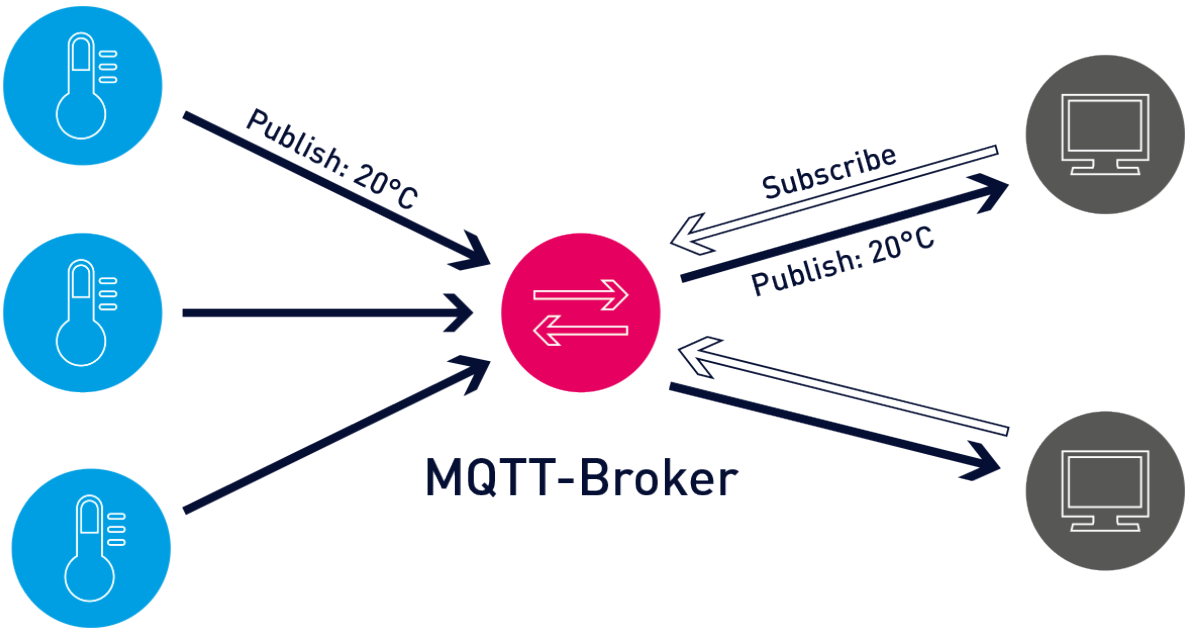

Figure 3: Contenu des boitiers.

Créer un serveur MQTT

Pour ce projet, nous utiliserons Mosquitto, un broker MQTT open-source et facile à utiliser. Une fois Mosquitto installé, vous pouvez démarrer le serveur MQTT.

S'abonner à un topic MQTT

Pour s'abonner à un topic MQTT, nous utiliserons la bibliothèque paho-mqtt de Python. Pour installer paho-mqtt via le gestionnaire de package python PIP:

```
pip install paho-mqtt
```

Voici le script Python qui se connecte à notre broker MQTT et s'abonne au topic « Sensor_1 »:

```
import paho.mqtt.client as mqtt

ssid = config['ssid']
password = config['password']
mqtt_server = config['mqtt_server_ip']

def on_connect(client, userdata, flags, rc):
    print("Connecté avec le code résultat " + str(rc))
    client.subscribe("Sensor_1")

def on_message(client, userdata, msg):
    data = json.loads(msg.payload.decode('utf-8'))
    [...]
    conn.commit()

client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message

client.connect(host=mqtt_server, port=1883, keepalive=60)

client.loop_forever()
```

Publier sur un topic MQTT avec l'ESP32

Sur l'ESP32, nous utiliserons la bibliothèque PubSubClient pour publier sur un topic MQTT. Voici un exemple de code qui envoie un message sur le topic « Sensor_1 » :

```
++
#include <WiFi.h>
#include <PubSubClient.h>

const char* ssid = "your_SSID";
const char* password = "your_PASSWORD";
const char* mqtt_server = "your_MQTT_SERVER_IP_ADDRESS";

WiFiClient espClient;
PubSubClient client(espClient);

void setup() {
  Serial.begin(115200);
  setup_wifi();
  client.setServer(mqtt_server, 1883);
}
```

```
void setup_wifi() {
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) { delay(500); }
  Serial.println("WiFi connected");
}

void loop() {
  if (!client.connected()) {
    reconnect();
  }
  client.loop();
  String payload = "Hello MQTT!";
  client.publish("Sensor_1", payload.c_str());
  delay(10000); // Publish every 10 seconds
}
void reconnect() {
  // Loop until we're reconnected
  while (!client.connected()) {
    if (client.connect("ESP32Client")) {
      client.publish("Sensor_1", "ESP32 connected");
    } else {
      delay(5000);
} }
}
```

Ainsi, à chaque fois que vous exécutez ce script sur l'ESP32, il se connectera à votre réseau WiFi, puis se connectera à votre serveur MQTT et publiera un message sur le topic « Sensor_1 » toutes les 10 secondes.

Récupération et stockage des données

Le processus d'enregistrement des données dans notre projet est essentiel et suit plusieurs étapes. Premièrement, une connexion à la base de données MongoDB est établie grâce à la commande `MongoClient`. Nous nous connectons à la base de données nommée `deformation_structure_db` et plus précisément à la collection `sensor_data`.

Nous commençons par enregistrer les positions initiales des capteurs dans la collection **"history_positions"** à partir d'un fichier JSON si et seulement si la collection est vide. Ces positions sont insérées dans la base de données avec un timestamp correspondant au moment précis de l'insertion.

Ensuite, nous mettons en place un buffer pour stocker temporairement les données des capteurs. Lorsque nous avons recueilli des données de tous les capteurs, nous les enregistrons dans la collection **"sensor_data"**. Les données enregistrées comprennent un timestamp en millisecondes, ainsi que les données d'accélération et de gyroscope pour chaque capteur.

Exemple de document de la collection **“history_positions”** avec 4 capteurs:


```
{
  "timestamp": 1686933655115.183,
  "positions": {
    "capteur0": [
      0,
      0,
      0
    ],
    "capteur1": [
      0,
      0,
      3
    ],
    "capteur2": [
      0,
      0,
      6
    ],
    "capteur3": [
      0,
      0,
      10
    ]
  }
}
```


En ce qui concerne le service MQTT, nous souscrivons aux topics correspondant à chaque capteur. À chaque fois qu’un message est reçu, les données d’accélération et de gyroscope sont extraites et stockées temporairement dans le buffer. Une fois que des données de tous les capteurs sont disponibles, elles sont ensuite stockées dans la base de données MongoDB.

Exemple de document de la collection **“sensor_data”** avec 4 capteurs:


```
{
  "timestamp": 1686933655115.183,
  "acceleration": {
    "capteur0": {
      "x": 3.1,
      "y": 2,
      "z": 8.1
    },
    "capteur1": {
      "x": 3.1,
      "y": 2,
      "z": 8.1
    },
    "capteur2": {
      "x": 3,
      "y": 2.35,
      "z": 8.2
    },
    "capteur3": {
```

```
      "x": 3,
      "y": 2.6,
      "z": 8.1
    }
  },
  "gyroscope": {
    "capteur0": {
      "x": 3,
      "y": 2.1,
      "z": 8
    },
    "capteur1": {
      "x": 3.2,
      "y": 2.3,
      "z": 8.1
    },
    "capteur2": {
      "x": 3.1,
      "y": 2.5,
      "z": 8.3
    },
    "capteur3": {
      "x": 3.09,
      "y": 2.1,
      "z": 8
    }
  }
}
```


Cela garantit une collecte, un traitement et un stockage efficaces et organisés des données essentielles à notre projet.

Algorithme pour le calcul de la déformation

Dans cette étude, nous proposons une méthode pour le suivi de la déformation de structures basée sur l'utilisation de capteurs d'accélération et de gyroscopes. L'algorithme que nous présentons permet d'extraire les informations pertinentes des capteurs, de les traiter et de les visualiser pour une analyse plus approfondie.

Au début de chaque étape de l'algorithme, les données d'accélération et de gyroscope pour chaque capteur sont récupérées à partir des enregistrements les plus récents dans la collection "sensor_data". Ces valeurs sont ensuite comparées aux enregistrements précédents pour isoler les changements dynamiques grâce à la soustraction des vecteurs.

Une fois ces changements calculés, l'algorithme procède à une série d'opérations de calcul pour dériver les informations sur la déformation de la structure. Tout d'abord, la moyenne des vecteurs d'accélération est calculée. Cette moyenne est ensuite soustraite de chaque vecteur individuel pour produire les accélérations résiduelles.

Ce processus est répété avec les vecteurs de gyroscope pour obtenir les rotations résiduelles. Cela fournit une quantification de la manière dont chaque capteur a tourné par rapport à la moyenne.

Les nouvelles positions des capteurs sont ensuite déterminées en ajoutant le vecteur moyen d'accélération à leurs positions initiales. Cela permet de prendre en compte l'inclinaison globale de la structure. Ces positions sont ensuite mises à jour en tenant compte des rotations dérivées des vecteurs gyroscopiques résiduels.

Avec les positions mises à jour, l'étape finale de l'algorithme consiste à estimer la déformation de la structure. Pour ce faire, nous utilisons des courbes de Bézier entre chaque paire de capteurs. Les points de contrôle pour ces courbes sont définis comme les positions mises à jour des capteurs, ainsi que ces même positions additionnées avec les vecteurs d'accélération de chaque capteur.

Voici un exemple de courbe de Bezier en 2D. Nous avons un segment en noir ou les extrémités correspondent à un capteur. Les flèches grises correspondent aux vecteurs accélérations de chaque capteur auquel il est attaché. Ainsi, pour calculer la courbe de Bezier (en bleu) entre les deux capteurs (extrémités du segment) on utilise comme points de contrôles les capteurs aux extrémités ainsi que la pointe des flèches grise, qui est simplement la somme du vecteur représentant les coordonnées du capteur avec son vecteur accélération.

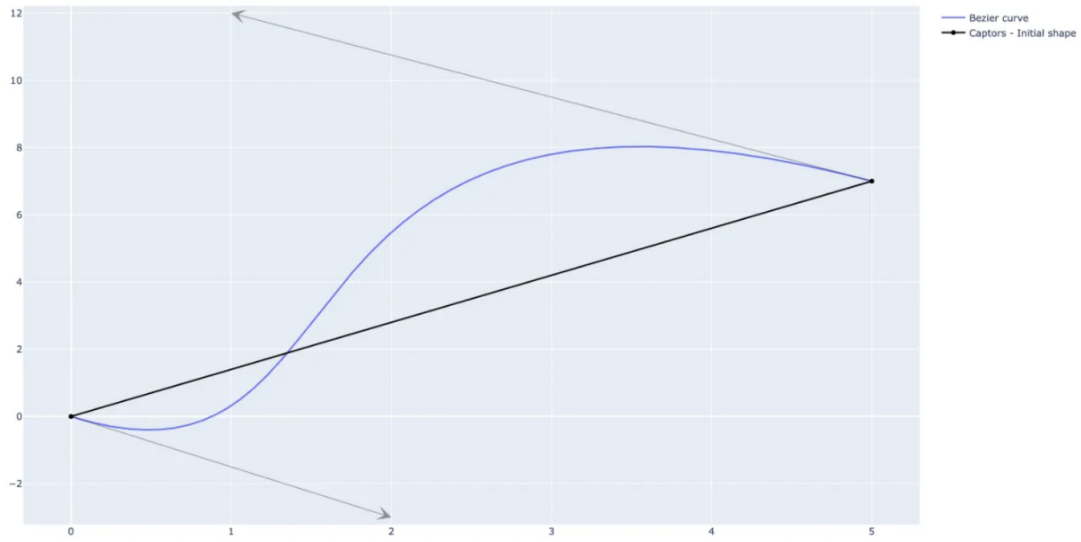

Figure 4: Exemple courbe de Bezier.

Grâce à cette méthode, les déformations peuvent être estimées et visualisées en temps réel, fournissant ainsi un aperçu précieux de l'état dynamique de la structure surveillée. Les informations dérivées de ce processus peuvent alors être utilisées pour des analyses ultérieures, y compris la prédiction de défaillances structurelles ou l'optimisation de la conception structurelle.

Enfin, l'algorithme calcule la distance euclidienne entre les points initiaux et les points déplacés pour chaque capteur ainsi que les déformations maximales de chaque segment. Ces distances, une fois converties dans les unités appropriées, sont renvoyées par l'algorithme et affichées.

Création de l'interface

Dans les exemples ci-dessous, nous utilisons 4 boitiers.

L'interface de l'application est construite en utilisant Streamlit et Plotly. À l'initialisation de l'application, seule la position initiale de chaque capteur est affichée:

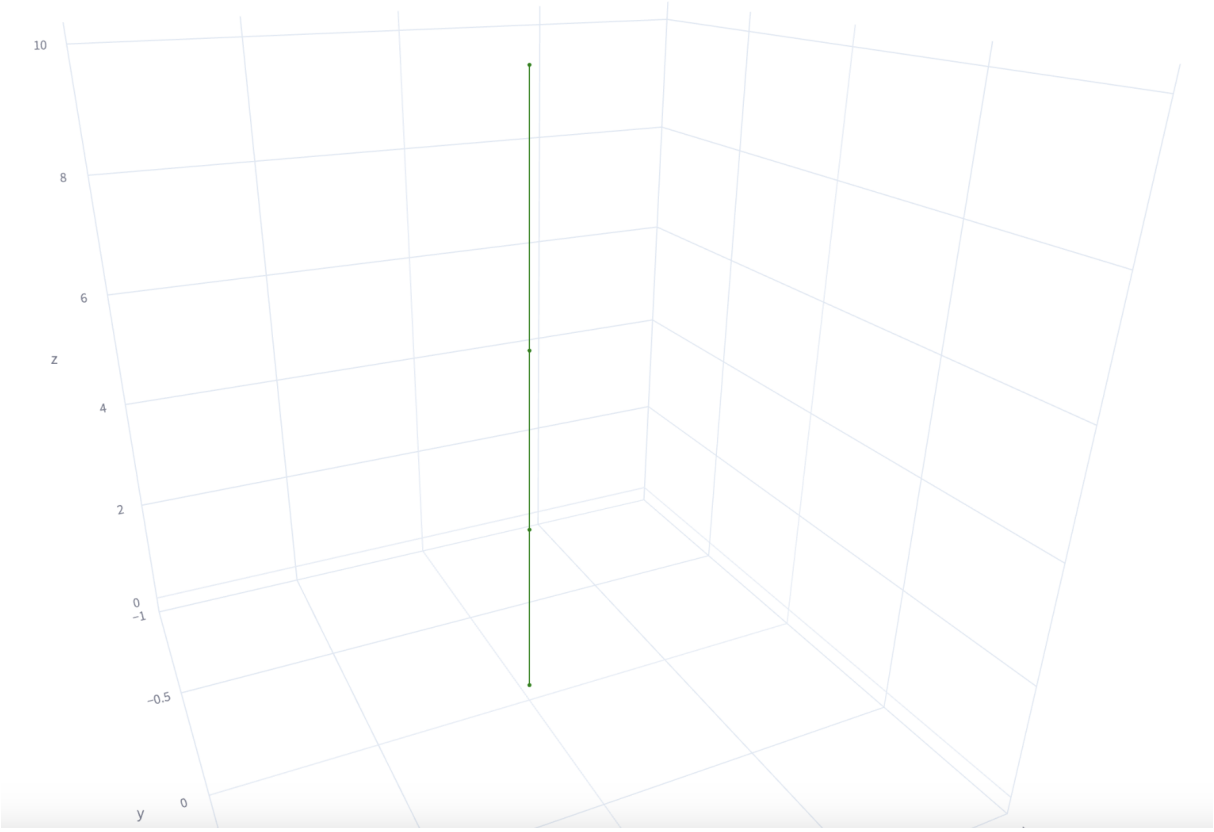

Figure 5: Structure initiale.

Après l'ajout des première données, la structure n'est pas modifiée car ces données servent de calibrage aux futures données. La déformation est calculée aux prochaines données reçues, et la structure est affichée:

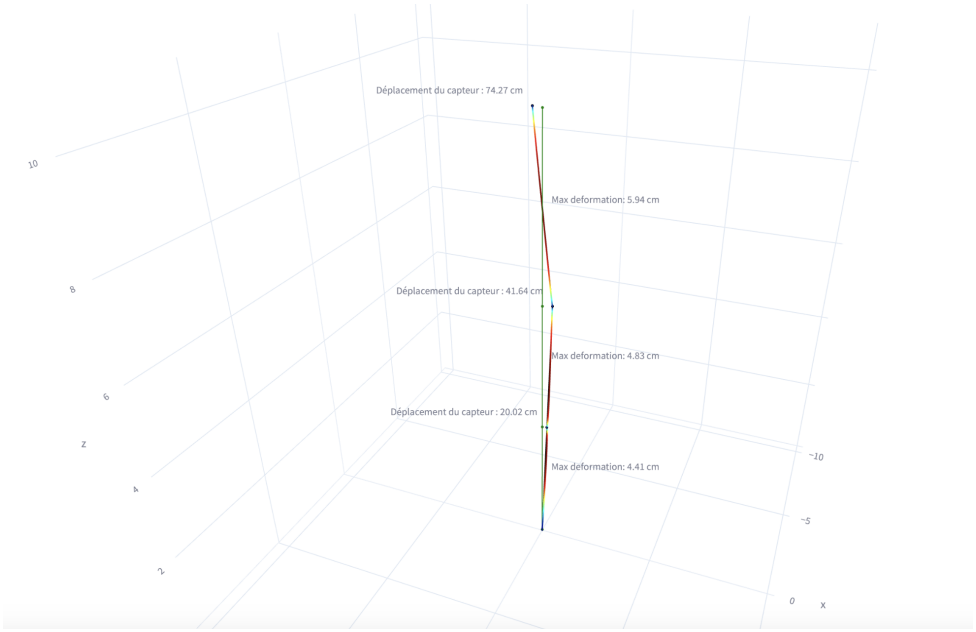

Figure 6: Visualisation de la déformation de la structure (inclinaison, rotation et déformation).

L'interface complète est donnée Figure 7, c'est une simple interface avec une figure pouvant être mise à jour.

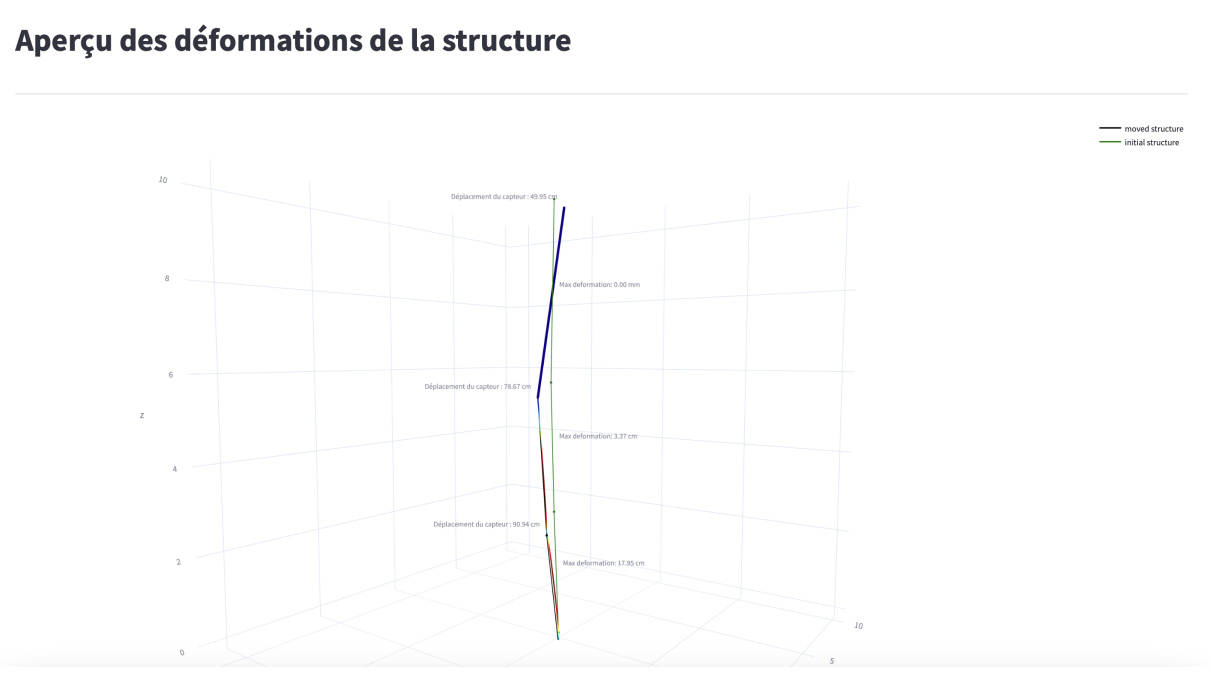

Figure 7: Interface Streamlit.

L'état de la structure est mis à jour en temps réel à l'arrivée des nouvelles données des boitiers sur le serveur MQTT.

Conclusion

En conclusion, ce projet illustre la manière dont une série de capteurs et une structure distribuée peuvent être utilisés pour surveiller en temps réel la déformation d'une structure. À l'aide d'un serveur MQTT pour la communication, MongoDB pour le stockage des données, et Streamlit pour l'interface utilisateur, nous avons créé un système qui non seulement enregistre les données de déformation, mais permet également une visualisation en direct et une analyse détaillée.

En termes d'extensions possibles pour ce projet, les capteurs pourraient être adaptés pour surveiller d'autres aspects physiques tels que la température ou le champ magnétique, selon l'application spécifique. De plus, l'application pourrait être améliorée pour fournir des notifications ou des alertes automatiques basées sur certaines conditions prédéfinies.