INSTITUT UNIVERSITAIRE DE TECHNOLOGIE
UNIVERSITÉ D'ANGERS

université angers

SAE S3

Centrale inertielle

Erwann Dewaele

Hélaric Salomon-Le Moigne

Gr 231

23/09/24-20/12/24

Tuteur : Lionel Leduc – Philippe Lucidarme

Décharge de responsabilités

Ce rapport présente le travail réalisé par un groupe d'étudiants dans le cadre d'un projet pédagogique. Les auteurs et l'Université d'Angers ne garantissent pas que l'information, les documents, la méthodologie et le matériel présentés dans ce document soient complets, conformes à l'état de l'art et exacts ni n'assurent en toutes circonstances la sécurité des biens, des personnes et des utilisateurs. Les auteurs et l'Université d'Angers ne seront pas tenus responsables des dommages éventuels qui pourraient résulter de l'utilisation du contenu du présent rapport.

Licence

Introduction

Aujourd'hui les centrales inertielles sont extrêmement répandues. Cette technologie est présente dans vos smartphones, dans les avions, les drones ou encore les motos. Elles permettent de connaître l'inclinaison par rapport à un point données pour ensuite faire tourner votre écran, redresser un avion ou réduire la puissance fournie au roux arrière et ne pas se retourner.

C'est dans cette optique que nous avons choisi ce projet. C'est un projet qui nous a donné l'opportunité de travailler sur quelque chose d'utilisé par tout le monde.

Sommaire

Etude technologique

1/ Technologie MEMS (Micro électro-Mechanical Systems) : Ce sont des microsystèmes conçus à partir de semi-conducteurs. Ils comprennent des éléments mécaniques permettant de réduire drastiquement la taille des systèmes tels que des capteurs ou actionneurs pour atteindre des dimensions de l'ordre du mm^2. En capteur un MEMS dispose d'une partie mobile et une autre statique. Dans le cas qui nous intéresse, les capteurs, la partie mobile se déplace ce qui entraîne une modification de la résistance, ou capacité, des éléments proches. Cette modification est mesurée et transformée en un signal électrique. Ces signaux passent ensuite par une phase de traitement. Tout d'abord une phase d'amplification, de filtrage, puis une conversion analogique numérique. Après cela, le microcontrôleur peut traiter les données reçues.

Pour créer les MEMS, des techniques issues de la micro-électronique ont été employées tout en les modifiant pour inclure des éléments mécaniques. Il existe plusieurs techniques pour les produire comme la gravure ou le dépôt de matière. Cette technologie est aujourd'hui largement utilisée dans les systèmes embarqués tels que dans les avions, smartphones, drones etc….

2/ IMU (centrale inertielle) : Il existe plusieurs types de centrale inertielle : les gyroscopiques qui possèdent d'excellente performance mais coûte très chère et celle qui nous intéresse les IMU basé sur des MEMS permettant de réduire drastiquement le coût de fabrication mais avec des performances moindres.

Une centrale inertielle à 9 axes est un assemblage d'un accéléromètre permettant de calculer l'accélération linéaire, d'un gyroscope permettant de calculer l'accélération angulaire, et enfin d'un magnétomètre permettant de mesurer les champs magnétiques et donc à savoir l'orientation en se référant au nord magnétique.

Il y a un algorithme permettant de passer de l'accélération à la position appelée AHRS qui consiste à intégrer les valeurs d'accélération renvoyées par les capteurs pour obtenir la vitesse et l'orientation de la centrale. Cependant l'AHRS est un algorithme très compliqué, aussi nous avons pris la décision de prendre une puce avec l'algorithme intégré.

Gestion de projet

Pour suivre notre avancement dans le projet nous avons réalisé un Gant sur Excel.

Centrale Inertielle

Erwann Dewaele-Helaric Salomon-Le Moign

Gr231

Début du projet : lun, 9/23/2024

TÂCHE	AVANCEMENT	DÉBUT	FIN
Première semaine			
études technologique	100%	23/9/24	25/9/24
Choix de l'IMU	100%	23/9/24	25/9/24
Commande de l'IMU	100%	25/9/24	25/9/24
Étude de la librairie de l'IMU	100%	25/9/24	27/9/24
Etude schema ESP 32	100%	25/9/24	27/9/24
Deuxième semaine			
Mise en action de l'IMU	100%	21/10/24	21/10/24
Première version websocket	100%	21/10/24	23/10/24
application pour afficher les résultats dans une fenêtre	100%	23/10/24	10/12/24
Troisième semaine			
Deuxième version de notre protocole websocket	100%	10/12/24	12/12/24
Fonctionnement du capteur WT931	100%	11/12/24	16/12/24
Quatrième semaine			
Intégration du wt931 au reste du projet	100%	17/12/24	17/12/24
Représentation 3D de l'orientation de l'IMU	100%	16/12/24	18/12/24
Rédaction du rapport	100%	16/12/24	20/12/24
Préparation de la soutenance	100%	16/12/24	19/12/24

Cahier des charges

1/ Choix d’un IMU 9 ou 6 axes doté d’un bon rapport qualité-prix.

2/ Récupération de l’orientation en degré en temps réel.

3/ Mise en place d’une communication à distance avec le protocole websocket.

4/ Affichage des informations via une application ou sur une page internet.

5/ Représentation de l’orientation de manière graphique grâce à un cube réagissant au mouvement de l’IMU.

Choix de l’IMU

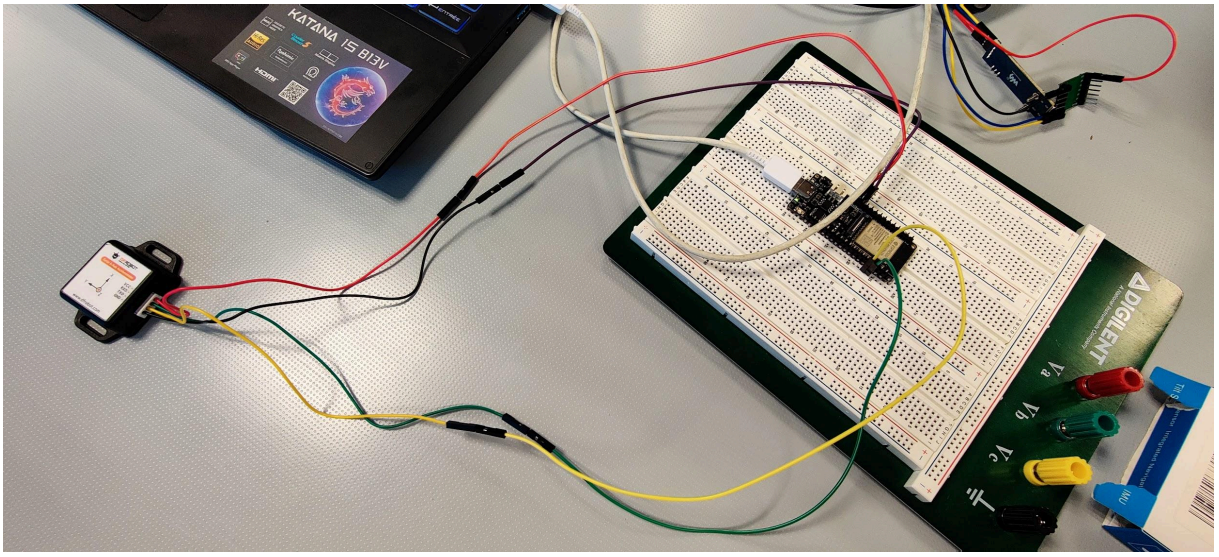
Nous avons commencé par choisir le capteur en comparant le nombre d’axes, si l’algorithme AHRS est disponible ou non, leur tension d’alimentation pour essayer d’avoir une alimentation répandu (3V3 ou 5V), ainsi que leur compatibilité par rapport au différent microcontrôleur.

Premièrement, l'IMU sysrox SRX-INS00-DEV nous intéressait. En effet il est compatible avec arduino ainsi que l’ESP 32 , son alimentation se fait en 3V3, il possède 9 axes ainsi qu’un baromètre permettant de connaître l’altitude. Néanmoins il est plus cher que les autres mais surtout il est compliqué de s'approvisionner, nous n’avons donc pas suivi cette piste.

Notre choix s’est finalement porté sur le DFRobot Sen0386. Il est peu cher et a de bons retours. Cependant c’est un 6 axes et ne possède donc pas de magnétomètre pour repérer sa position par rapport au nord.

Récupération des données

Nous avons connecté notre IMU à l'ESP 32, à l'aide de la librairie du Sen0386 nous avons été capable d'afficher les données du capteur dans l'IDE arduino sous forme de graphique.



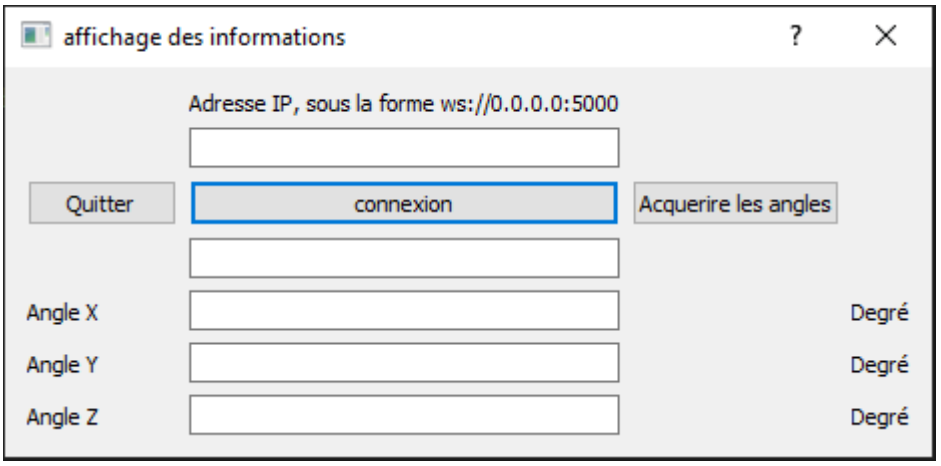
Branchement Sen0386 avec l'ESP 32

Websocket

1er essai

Pour communiquer à distance les infos nous avons utilisé le protocole websocket.

Nous sommes tout d'abord partie pour faire de l'ESP 32 le serveur et sur un pc distinct le client, tous deux connecté au même wifi. Pour l'affichage nous avons réalisé une fenêtre en python.



Tous les boutons sont fonctionnels. Le bouton connexion sert à vérifier la connexion. Ensuite nous appuyons sur acquérir les angles pour afficher les angles en degrés.

Mettre l'image avec les infos

Le problème de cette méthode est que l'affichage n'est pas réactif, cela prend près de 10 secondes pour afficher les informations. De plus, nous n'avons pas réussi à mettre l'actualisation des infos automatique. Nous avons tenté de coder une boucle while pour afficher en permanence, le programme n'affichait plus rien mais les boucles s'effectuaient. Est-ce que le programme réussissait à acquérir les informations, mais ne les affichait pas car il tentait une nouvelle fois d'acquérir les données ou est-ce que le programme n'arrivait juste pas à obtenir les informations et tournait dans le vide.

Après discussion avec m. Lucidarme Nous avons changé notre façon de faire.

2e essai

M. Lucidarme nous a donc envoyé un exemple de code pour mettre le serveur sur le pc plutôt que sur l'ESP 32 ce qui permet plus de réactivité.

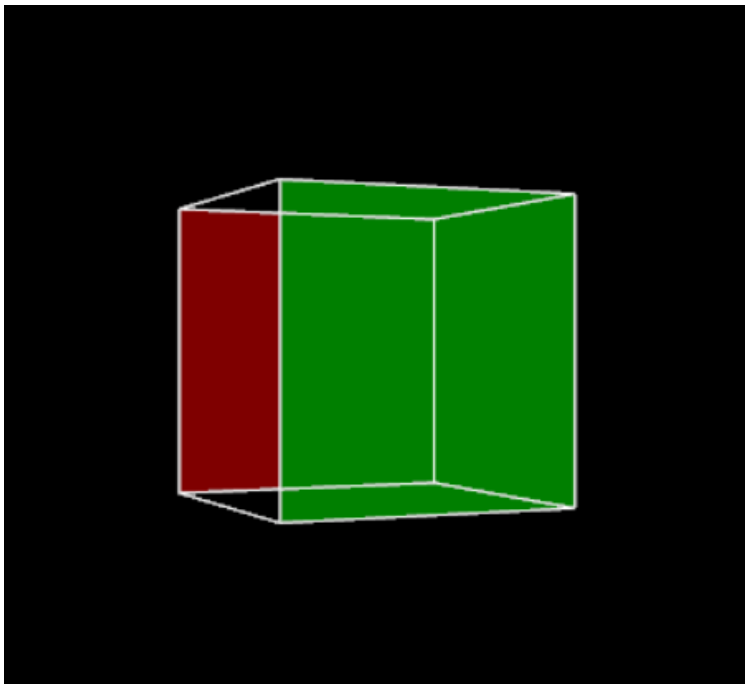
Nous avons à nouveau de rendre fonctionnel l'application présentée précédemment mais le même problème était toujours présent.

L'affichage des informations se fait désormais sur une page internet.

Page web en fonctionnement

De cette manière l'affichage se réalise presque sans délai, et correspond à l'affichage dans le terminal du serveur sur python.

Après cela nous nous sommes attelés à la création d'un cube en 3D permettant un affichage en 3D des informations. Ce cube devait donc réagir en fonction de la rotation de notre IMU, mais également intégrer le protocole websocket permettant d'afficher les informations sur un PC distinct de celui où est le serveur.



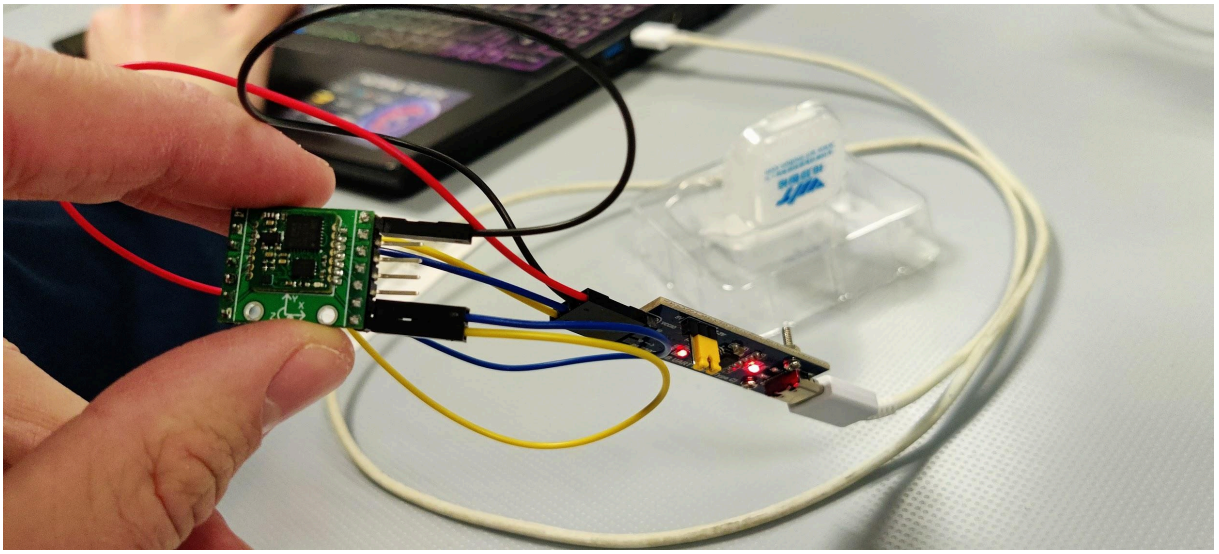
Wit Motion

Fonctionnement du capteur

Devant le fait que l'IMU Sen0386 dérivait au fil du temps, M. Lucidarme nous a informés qu'il souhaitait voir marcher un autre IMU qu'il avait en stock. Le Wit Motion possède 9 axes ce qui devrait en théorie le rendre plus performant que le Sen0386.

Nous avons trouvé une librairie permettant de faire tourner le WT931 mais il y a des erreurs dedans. En effet deux fonctions ont le même nom ce qui empêche le programme de tourner correctement.

Ensuite nous avons appris qu'il existait une application de Wit Motion permettant d'afficher directement les données. Malheureusement l'application n'est pas fonctionnelle.

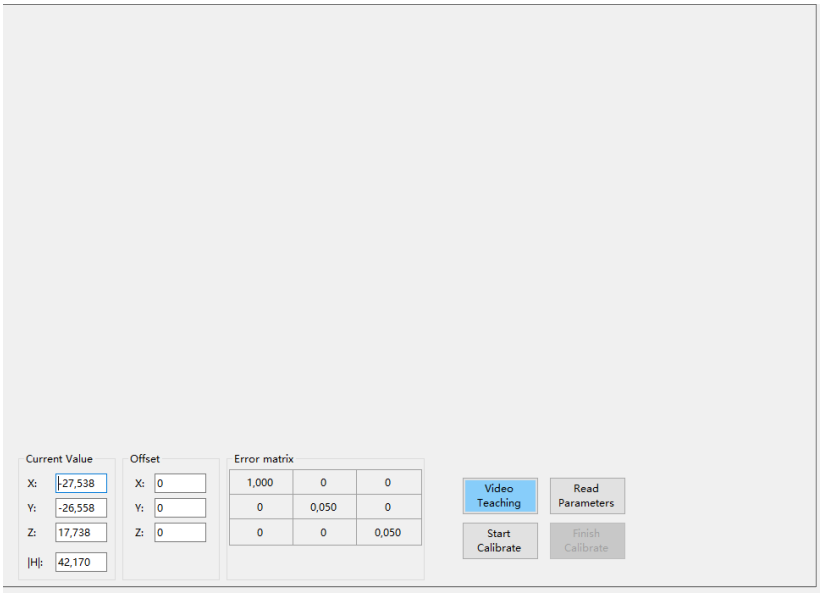


Branchement du WT 931

En effet sur cette vidéo on voit bien qu'il y a plein de chose sur l'écran permettant calibrer l'IMU

5. Witmotion Magnet (Octal Calibration Method) for 9-axis Sensors(fixture proc…

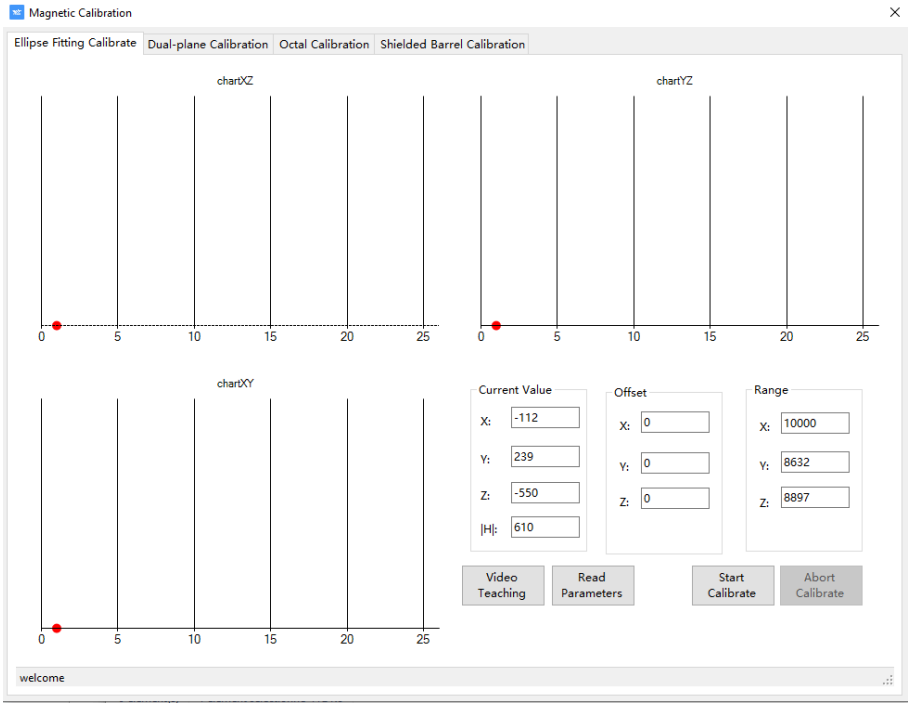
Or sur notre version de l'application il n'y a rien d'affiché.



Ecran de calibration de l'application Wit Motion

Nous n'avons donc aucune idée si notre IMU est bien calibré. De plus, nous avons constaté que l'IMU avait beaucoup de mal à suivre nos mouvements. En effet, il ne revient pas sur sa position d'origine et réagit avec du retard, va souvent trop loin dans ses rotations.

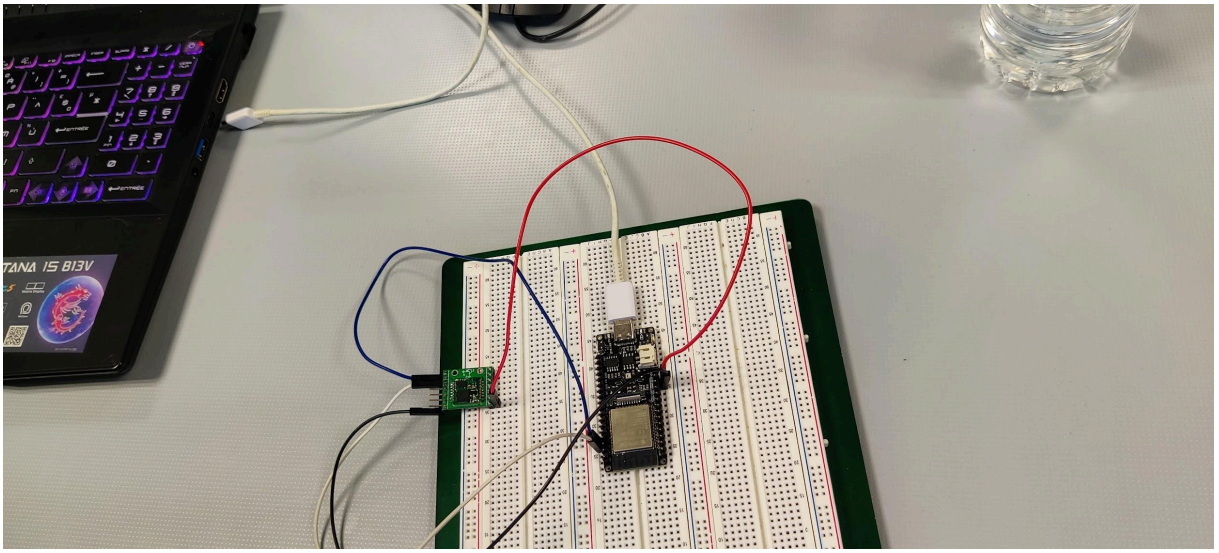
Après avoir contacté Wit Motion, ces derniers nous ont renvoyé une version plus ancienne de leur logiciel. Grâce à cela m. Lucidarme a réussi à calibrer correctement le capteur.



Même page que plus haut sur la version du logiciel renvoyé par Wit Motion

Intégration au projet

Une fois la calibration correctement réalisée nous avons pu étudier le capteur à l'aide de l'application afin de l'intégrer à notre projet sur l'ESP32.



Câblage du wt931 sur l’ESP32

Pour cela nous avons dû étudier la trame envoyée par le capteur dans le but de la décoder et de la convertir en données utilisables dans le reste du projet. Le processus permettant de décoder la trame consiste à sa séparation en plusieurs parties puis à les rassembler selon l'ordre donné dans la documentation du capteur à l'aide d'un décalage bit à bit.

Une fois la trame remplacée dans le bon ordre, il ne reste plus qu'à réaliser un calcul fourni par le fabricant, permettant de passer des données brutes à des angles. Le but n'étant pas de simplement calculer des angles mais également d'envoyer les données sur un serveur python, il est nécessaire de réaliser une vérification de la trame à l'aide d'une valeur nommée le checksum afin de s'assurer qu'aucune valeur erronée ne soit envoyée.

Perspective d'amélioration

Notre projet présente des lacunes. En effet, nos codes présentent des erreurs au bout d'un certain temps, et notre projet ne présente pas une allure convenable.

1. Réalisation d'une carte : Cette étape permettrait une utilisation plus simple que la breadboard mais également une meilleure présentation.

2. Optimisation de la transmission des données : Lors de l'utilisation du wt931 notre projet avait beaucoup de mal à s'actualiser convenablement. Nous pensons que le problème vient de notre code pour récupérer les données. Une amélioration de ce code permettrait sûrement de régler ce problème.

3. Automatisation de choix du réseau wifi: Nous avons également réfléchi à l'automatisation du choix du réseau wifi comme celle de nos téléphones. De sorte à ce que l'ESP32 choisisse si disponible un réseau passé en paramètre. Ce qui rendrait son utilisation plus simple.

Conclusion

En conclusion, ce projet nous aura permis de renforcer nos compétences en programmation. Lors de ce dernier nous avons manipulé plusieurs langages différents en particulier le python et le C. Nous avons également manipulé plusieurs outils totalement nouveaux pour nous, en particulier le protocole websocket qui nous aura permis de faire communiquer l'ESP32 avec un pc sur lequel tourne un serveur python. Nous avons également manipulé openGL pour réaliser notre cube.

Finalement nous aurons réussi à créer une base pour un futur système opérationnel, notamment en intégrant les améliorations citées précédemment.

Annexes

Sommaire

Annexe 1 : Code de la page html client

```html
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Affichage des données du capteur IMU</title>
    <!-- Importation de Google Fonts -->
    <link href="https://fonts.googleapis.com/css2?family=Roboto:wght@400;500;700&display=swap" rel="stylesheet">
    <style>
        body {
            font-family: 'Roboto', sans-serif;
            background-color: #f4f7fc;
            color: #333;
            margin: 0;
            padding: 0;
            display: flex;
            flex-direction: column;
            justify-content: center;
            align-items: center;
            height: 100vh;
            box-sizing: border-box;
        }

        h2 {
            font-size: 2rem;
            color: #4e73df;
            margin-bottom: 20px;
            text-shadow: 1px 1px 3px rgba(0, 0, 0, 0.1);
        }

        .loading {
            font-size: 1.2rem;
            color: #888;
            margin-bottom: 30px;
        }

        table {
            width: 80%;
            border-collapse: collapse;
            margin: 0 auto;
            box-shadow: 0px 4px 6px rgba(0, 0, 0, 0.1);
            border-radius: 10px;
            overflow: hidden;
            background-color: white;
        }
```

```html
        td {
            width: 60px; /* Largeur et hauteur égales pour des carrés */
            height: 60px;
            text-align: center;
            font-size: 16px;
            color: white;
            font-weight: 500;
            border-radius: 8px;
            transition: background-color 0.3s ease;
        }

        td:hover {
            cursor: pointer;
            box-shadow: 0px 2px 8px rgba(0, 0, 0, 0.15);
        }

        /* Style pour les bordures du tableau */
        td:not(:last-child) {
            border-right: 1px solid #e0e0e0;
        }

        td:first-child {
            border-left: 1px solid #e0e0e0;
        }

        .status-container {
            display: flex;
            justify-content: center;
            align-items: center;
            font-size: 1rem;
            margin-top: 20px;
            padding: 10px 20px;
            background-color: #28a745;
            color: white;
            border-radius: 20px;
            box-shadow: 0px 2px 5px rgba(0, 0, 0, 0.1);
            margin-bottom: 20px;
        }
```

```
        .status-container.error {
            background-color: #dc3545;
        }
    </style>
</head>
<body>
    <h2>Angles X|Y|Z de l'IMU</h2>
    <div id="status" class="status-container loading">Connexion au serveur...</div>
    <table id="sensorDataTable">
        <tr>
            <td colspan="8">Chargement des données...</td>
        </tr>
    </table>

    <script>
        // Le reste du code JavaScript reste inchangé
        const ws = new WebSocket('ws://192.168.235.250:8001');

        ws.onopen = () => {
            document.getElementById('status').innerText = 'Connecté au serveur';
            document.getElementById('status').classList.remove('loading');
            document.getElementById('status').classList.add('success');
        };

        ws.onmessage = (event) => {
            const data = event.data;
            console.log("Données reçues :", data);

            const lines = data.trim().split(';');
            const table = document.getElementById('sensorDataTable');

            table.innerHTML = '';

            lines.forEach(line => {
                const cleanedLine = line.trim().replace(/\t+$/, '');
                const values = cleanedLine.split('\t');
```

```
                if (values.every(value => value.trim() === '')) {
                    return;
                }

                const row = document.createElement('tr');

                values.forEach(value => {
                    const numericValue = parseInt(value.trim(), 10);

                    if (!isNaN(numericValue)) {
                        const cell = document.createElement('td');
                        cell.style.backgroundColor = `rgb(0,0,0)`;
                        cell.textContent = numericValue;
                        row.appendChild(cell);
                    }
                });

                if (row.children.length > 0) {
                    table.appendChild(row);
                }
            });
        };
```

```
        ws.onerror = (error) => {
            console.error("Erreur WebSocket: ", error);
            document.getElementById('status').innerText = 'Erreur de connexion au serveur';
            document.getElementById('status').classList.remove('loading');
            document.getElementById('status').classList.add('error');
        };

        ws.onclose = () => {
            document.getElementById('status').innerText = 'Déconnecté du serveur';
            document.getElementById('status').classList.remove('loading');
            document.getElementById('status').classList.add('error');
        };
    </script>
</body>
</html>
```

Annexe 2 : Code du cube avec OpenGL

```
import pygame
from pygame.locals import *

from OpenGL.GL import *
from OpenGL.GLU import *

from websockets.sync.client
import connect

adresse =
"ws://192.168.235.250:8001"

verticies = (
    (1, -1, -1),  # 0
    (1, 1, -1),   # 1
    (-1, 1, -1),  # 2
    (-1, -1, -1), # 3
    (1, -1, 1),   # 4
    (1, 1, 1),    # 5
    (-1, -1, 1),  # 6
    (-1, 1, 1)    # 7
    )

edges = (
    (0,1),
    (0,3),
    (0,4),
    (2,1),
    (2,3),
    (2,7),
    (6,3),
    (6,4),
    (6,7),
    (5,1),
    (5,4),
    (5,7)
    )

surfaces = (
    (0,1 , 2, 3),       #
surface 0
    (4,5 , 7, 6),       #
surface 1
    (1,2 , 7, 5),       #
surface 2
    #(5,7,2,1),
)
normals = [
    ( 0,  0, 1),  # surface 0
    ( 0,  0, -1),  # surface
1
    ( 0,  1, 0),  # surface 2
]

colors = (
    (1, 0, 0, 0.6),
    (0, 1, 0, 0.6),
    (0, 0, 1, 0.6),
)

textureCoordinates = ((0, 0),
(0, 1), (1, 1), (1, 0))


def Cube():

    glEnable(GL_BLEND)
    glEnable(GL_DEPTH_TEST)
    glEnable(GL_MULTISAMPLE)

    glBlendFunc(GL_SRC_ALPHA,
GL_ONE_MINUS_SRC_ALPHA)
    glEnable(GL_LINE_SMOOTH)

#glHint(GL_LINE_SMOOTH_HINT,
GL_DONT_CARE)

glHint(GL_LINE_SMOOTH_HINT,
GL_NICEST);
    glBegin(GL_QUADS)
```

```
    for i_surface, surface in
enumerate(surfaces):



glNormal3fv(normals[i_surface
])

glColor4f(colors[i_surface]
[0], colors[i_surface]
[1],colors[i_surface][2],
colors[i_surface][3])

        for i_vertex, vertex
in enumerate(surface):

glTexCoord2fv(textureCoordina
tes[i_vertex])

glVertex3fv(verticies[vertex]
)
    glEnd()

    glBegin(GL_LINES)
    glColor3f(0.7, 0.7, 0.7)

    for edge in edges:
        for vertex in edge:


glVertex3fv(verticies[vertex]
)
    glEnd()


def main():
    pygame.init()
    display = (800,600)

pygame.display.set_mode(displ
ay, DOUBLEBUF|OPENGL)

    gluPerspective(45,
(display[0]/display[1]), 0.1,
50.0)

    glTranslatef(0.0,0.0,
-10)
    while True:
        for event in
pygame.event.get():
            if event.type ==
pygame.QUIT:
                pygame.quit()
                quit()
        mess =
connect(adresse).recv()
        mess =
mess.split(";")
        Rx = -float(mess[0])
        Rz = float(mess[1])
        Ry = float(mess[2])
        glLoadIdentity()
        gluPerspective(45,
(display[0]/display[1]), 0.1,
50.0)
        glTranslatef(0.0,0.0,
-10)
        glRotatef(Rx, 1, 0,
0)
        glRotatef(Ry, 0, 1,
0)
        glRotatef(Rz, 0, 0,
1)

glClear(GL_COLOR_BUFFER_BIT|G
L_DEPTH_BUFFER_BIT)

        print (Rx, Ry, Rz)
        Cube()
        pygame.display.flip()
        pygame.time.wait(10)


main()
```

Annexe 3 : Code du capteur SEN0386 avec affichage dans le terminal

```
#include <DFRobot_WT61PC.h>

#define FPSerial Serial2
DFRobot_WT61PC sensor(&FPSerial);

float X,Y,Z = 0;

void setup()
{
  //Use Serial as debugging serial port
  Serial.begin(115200);

#if (defined ESP32)
  FPSerial.begin(9600, SERIAL_8N1, /*rx =*/D3,
       /*tx =*/D2);
#else
  FPSerial.begin(9600);
#endif
  sensor.modifyFrequency(FREQUENCY_200HZ);
}

void loop()
{
  if (sensor.available()) {
    X = sensor.Angle.X;
    Y = sensor.Angle.Y;
    Z = sensor.Angle.Z;
  }
  Serial.print("x:");
  Serial.print(X);
  Serial.print("\ty:");
  Serial.print(Y);
  Serial.print("\tz:");
  Serial.println(Z); //angle information of X,
       Y, Z
}
```

Annexe 4 : Code du capteur SEN0386 avec ESP32 en serveur

```
#include <WiFi.h> // Include WIFi Library for ESP32
#include <WebServer.h> // Include WebSwever Library for ESP32
#include <ArduinoJson.h> // Include ArduinoJson Library
#include <WebSocketsServer.h> // Include Websocket Library
#include <DFRobot_WT61PC.h>


//identifiants WiFi
// const char* ssid="esp32"; //Fill in the WIFI name
// const char* password="Rubis974"; //Fill in the WIFI password
const char* ssid="J'ai de la 5g et pas toi"; //Fill in the WIFI name
const char* password="Avecdesfrites"; //Fill in the WIFI password


//Initialisation du capteur
#define FPSerial Serial2
DFRobot_WT61PC sensor(&FPSerial);
unsigned int Z,X,Y = 0;


int interval = 100; // virtual delay
unsigned long previousMillis = 0; // Tracks the time since last event fired


String jsonString; // Temporary storage for the JSON String
String pin_status = ""; // Holds the status of the pin


WebServer server(80); // create instance for web server on port "80"
WebSocketsServer webSocket = WebSocketsServer(81); //create instance for webSocket server on port"81"
```

```
void setup()
{
  pinMode(2, OUTPUT);
  Serial.begin(115200);

  Serial.printf("Connecting to %s",ssid);
  WiFi.begin(ssid,password);      //Connect to WIFI
  while(WiFi.status()!=WL_CONNECTED){      //Wait for the connection to be successful
    delay(500);
    Serial.print(".");
  }
  Serial.println();
  Serial.print("Connected to WiFi network with IP Address: ");
  Serial.println(WiFi.localIP());
  Serial.println("Timer set to 5 seconds (timerDelay variable), it will take 5 seconds before publishing the first reading.");

  server.on("/", [](){});
  server.begin(); // init the server
  webSocket.begin();  // init the Websocketserver
  webSocket.onEvent(webSocketEvent);  // init the webSocketEvent function when a websocket event occurs


#if (defined ESP32)
  FPSerial.begin(9600, SERIAL_8N1, /*rx =*/D3, /*tx =*/D2);
#else
  FPSerial.begin(9600);
#endif

  sensor.modifyFrequency(FREQUENCY_100HZ);
}
```

```
void loop()
{
  server.handleClient();  // webserver methode that handles all Client
  webSocket.loop(); // websocket server methode that handles all Client
  unsigned long currentMillis = millis(); // call millis  and Get snapshot of time
  if ((unsigned long)(currentMillis - previousMillis) >= interval) { // How much time has passed, accounting for rollover with subtraction!
    update_angle(); // update temperature data.
    update_webpage(); // Update Humidity Data
    digitalWrite(2,HIGH);
    previousMillis = currentMillis;   // Use the snapshot to set track time until next event
  }
}

// This function gets a call when a WebSocket event occurs
void webSocketEvent(byte num, WStype_t type, uint8_t * payload, size_t length) {
  switch (type) {
    case WStype_DISCONNECTED:  // enum that read status this is used for debugging.
      Serial.print("WS Type ");
      Serial.print(type);
      Serial.println(": DISCONNECTED");
      break;
    case WStype_CONNECTED:  // Check if a WebSocket client is connected or not
      Serial.print("WS Type ");
      Serial.print(type);
      Serial.println(": CONNECTED");
      if (digitalRead(26) == HIGH) {  //check if pin 22 is high or low
        pin_status = "ON";
        update_webpage(); // update the webpage accordingly
      }
      else {
        pin_status = "OFF"; //check if pin 22 is high or low
        update_webpage();// update the webpage accordingly
      }
      break;
  }
}
```

```
void update_angle(){
  if (sensor.available()) {
    X = sensor.Angle.X;
    Y = sensor.Angle.Y;
    Z = sensor.Angle.Z;
  }
}

void update_webpage()
{
  StaticJsonDocument<100> doc;
  // create an object

  JsonObject object = doc.to<JsonObject>();
  object["Connected"] = "connected";
  object["X"] = Padding(X);
  object["Y"] = Padding(Y);
  object["Z"] = Padding(Z);
  serializeJson(doc, jsonString); // serialize the object and save teh result to teh string variable.
  Serial.println( jsonString ); // print the string for debugging.
  Serial.println(WiFi.localIP());
  Serial.println();
  webSocket.broadcastTXT(jsonString); // send the JSON object through the websocket
  jsonString = ""; // clear the String.
}

String Padding(int Val){
  String x = String(Val);
  String i = "0";
  if(Val<100){
    x = i + x;
  }
  if(Val<10){
    x = i + x;
  }
  return x;
}
```

Annexe 5 : Code du capteur SEN0386 avec serveur python

```
#include <WiFi.h> // Include WIFi Library for ESP32
#include <WiFiMulti.h>
#include <WebSocketsClient.h>  // Include Websocket Library
#include <DFRobot_WT61PC.h>


WiFiMulti WiFiMulti;
WebSocketsClient webSocket;

//identifiants WiFi
// const char* ssid="esp32";  //Fill in the WIFI name
// const char* password="Rubis974";   //Fill in the WIFI password
const char* ssid="J'ai de la 5g et pas toi";  //Fill in the WIFI name
const char* password="Avecdesfrites";   //Fill in the WIFI password


//Initialisation du capteur
#define FPSerial Serial2
DFRobot_WT61PC sensor(&FPSerial);
String Z,X,Y;
String data;
int i = 0;
```

```
void setup()
{
  pinMode(2, OUTPUT);
  pinMode(D6, OUTPUT);
  pinMode(27, INPUT_PULLUP);
  Serial.begin(115200);

  Serial.printf("Connecting to %s",ssid);
  WiFiMulti.addAP(ssid, password);
  while (WiFiMulti.run() != WL_CONNECTED) {
    delay(100);
    Serial.print(".");
  }
  Serial.println();
  
  digitalWrite(2,HIGH);
  digitalWrite(D6,HIGH);
  // webSocket.begin("192.168.246.92", 8000, "/");
  webSocket.begin("192.168.235.250", 8000, "/");
  webSocket.setReconnectInterval(2000);

#if (defined ESP32)
  FPSerial.begin(9600, SERIAL_8N1, /*rx =*/D3, /*tx =*/D2);
#else
  FPSerial.begin(9600);
#endif

  sensor.modifyFrequency(FREQUENCY_100HZ);
}
```

```
void loop()
{
  if(digitalRead(27)==0){
    digitalWrite(D6,LOW);
    delay(30);
    digitalWrite(D6,HIGH);
  }
  webSocket.loop(); // websocket server methode that handles all Client
  update_angle(); // update temperature data.
  data = X + ";" + Y + ";" + Z + "\n";
  Serial.println("Send\n");
  webSocket.sendTXT(data);
  delay(10);
}

void update_angle(){
  if (sensor.available()) {
    X = Padding(sensor.Angle.X);
    Y = Padding(sensor.Angle.Y);
    Z = Padding(sensor.Angle.Z);
  }
}

String Padding(int Val){
  String x = String(Val);
  String i = "0";
  if(Val<100){
    x = i + x;
  }
  if(Val<10){
    x = i + x;
  }
  return x;
}
```

Annexe 6 : Code du serveur python

```
import asyncio
import websockets

i = 1
CLIENTS = set()
# create handler for each
connection
async def handler(websocket):

    global i

    while True:
        data = await
websocket.recv()
        reply = f"{data}"
        print ("Data received
: ", data, ' ' , i)

        #await
websocket.send(reply+str(i))

        for ws in
CLIENTS.copy():
            try:
                await
ws.send(reply)
            except
websockets.ConnectionClosed:
                pass
            #await
websocket.broadcast(CLIENTS,
'test')
        i=i+1

# create handler for each
connection
async def
handlerBrowser(websocket):
    global i
    CLIENTS.add(websocket)
    print (str(len(CLIENTS))
+ ' client web')
    while True:
        data = await
websocket.recv()
        reply = f"{data}"
        print ("Data send :
", data, ' ' , i)

        await
websocket.send(reply)

start_server =
websockets.serve(handler,
"0.0.0.0", 8000)
start_server_chrome =
websockets.serve(handlerBrows
er, "0.0.0.0", 8001)

asyncio.get_event_loop().run_
until_complete(start_server)
asyncio.get_event_loop().run_
until_complete(start_server_c
hrome)
asyncio.get_event_loop().run_
forever()
```

Annexe 7 : Code du capteur WT931 avec serveur python

```
#include <WiFi.h> // Include WIFi Library for ESP32
#include <WiFiMulti.h>
#include <WebSocketsClient.h>  // Include Websocket Library

WiFiMulti WiFiMulti;
WebSocketsClient webSocket;

//identifiants WiFi
// const char* ssid="esp32";  //Fill in the WIFI name
// const char* password="Rubis974";   //Fill in the WIFI password
const char* ssid="J'ai de la 5g et pas toi";  //Fill in the WIFI name
const char* password="Avecdesfrites";   //Fill in the WIFI password

//Initialisation du capteur
#define FPSerial Serial2
float Z,X,Y;
String data;
int tab[9];
long check = 0;

void setup()
{
  pinMode(2, OUTPUT);
  Serial.begin(115200);

  Serial.printf("Connecting to %s",ssid);
  WiFiMulti.addAP(ssid, password);
  while (WiFiMulti.run() != WL_CONNECTED) {
    delay(100);
    Serial.print(".");
  }
  Serial.println();
  
  digitalWrite(2,HIGH);
  // webSocket.begin("192.168.98.250", 8000, "/");
  webSocket.begin("192.168.235.250", 8000, "/");
  webSocket.setReconnectInterval(2000);

#if (defined ESP32)
  FPSerial.begin(115200, SERIAL_8N1, /*rx =*/D3, /*tx =*/D2);
#else
  FPSerial.begin(115200);
#endif
}
```

```
void loop()
{
  webSocket.loop(); // websocket server methode that handles all Client
  update_angle(); // update temperature data.
  data = String(X) + ";" + String(Y) + ";" + String(Z) + ";" + "\n";
  if((check%256)==tab[8]){
    webSocket.sendTXT(data);
  }
  // delay(5);
}

void update_angle(){
  check = 0;
  if (FPSerial.read()==0x55) {
    if (FPSerial.read()==0x53) {
      for(int j=0 ; j<9 ; j++){
        tab[j]=FPSerial.read();
        check += tab[j];
      }
    }
  }
  check = check - tab[8]+ 0x55 + 0x53;
  X = (tab[1]<<8)|tab[0];
  Y = (tab[3]<<8)|tab[2];
  Z = (tab[5]<<8)|tab[4];
  X = (X/32768)*180;
  Y = (Y/32768)*180;
  Z = (Z/32768)*180;
}
```