Date : 24/03/2025

Compte-rendu séance 11 Téo Baillot d'Estivaux :

Objectifs de la séance :

L'objectif est d'améliorer la transmission des données entre les deux cartes. La séance dernière, je n'ai pas réussi à le faire et cette séance je vais donc continuer à essayer de résoudre ce problème.

Tests :

Au départ, j'ai voulu faire des tests sur la puissance d'émission car je pense que lors de la dernière séance j'avais mis une puissance d'émission trop élevée. Le problème pourrait donc être que comme les deux cartes sont très proches et que comme la puissance de transmission est très élevée le signal sature au bout d'un moment ce qui pourrait expliquer qu'il y a des périodes où la carte émettrice n'arrive plus à envoyer des données. Finalement, après tests, baisser la puissance d'émission n'a pas réellement réglé le problème car il y avait toujours des périodes où la carte n'envoyait pas les données mais cette période variait.

Par la suite j'ai essayé de séparer les tâches sur les deux cœurs de la tâche, l'idée est d'utiliser le premier cœur pour récupérer les données de l'accéléromètre et du capteur flex et d'utiliser le deuxième pour envoyer les données. J'ai donc travaillé au développement d'un code qui ferait ça mais n'ai jamais réussi à faire quelque chose qui fonctionne.

Puis, j'ai commencé à me dire que peut-être le problème était le type de transmission donc je me suis dit qu'au lieu de faire une transmission wifi j'allais transmettre les données en utilisant le module LoRa de la carte mais après avoir réalisé le code (qui était assez similaire à celui du wifi), je me suis rendu compte que la fréquence de transmission était trop faible pour mes besoins.

Je suis donc reparti sur le code avec la transmission wifi et j'ai finalement réussi à ajuster les délais internes du code et ajuster la puissance d'émission pour avoir un résultat satisfaisant. Il reste encore des pertes de paquets mais le contrôle de la pince est assez satisfaisant, il n'y a plus de longue période où aucun paquet ne s'envoie. Le code final pour la carte émettrice est :

```
#include <esp_now.h>

#include <esp_wifi.h>

#include <WiFi.h>

#include <Wire.h> // Bibliothèque pour la communication I²C


#define MPU6050_ADDR 0x68 // Adresse I²C du MPU-6050

#define FLEX_SENSOR_PIN 39 // GPIO pour lire les valeurs du capteur flex
```

```
#define FILTER_SAMPLES 30 // Définit la taille de l'historique pour le filtrage des données

uint8_t broadcastAddress[] = {0x30, 0xAE, 0xA4, 0x6F, 0x06, 0x84}; // Adresse MAC du destinataire

// Historique pour filtrer les données du capteur flex
float flexHistory[FILTER_SAMPLES];
int filterIndex = 0; // Index pour parcourir l'historique

// Structure pour envoyer les données des capteurs
struct SensorData {
  float accelX; // Valeur de l'accéléromètre sur l'axe X
  float accelY; // Valeur de l'accéléromètre sur l'axe Y
  int flex; // Valeur du capteur flex
};

SensorData dataToSend; // Instance de la structure utilisée pour l'envoi des données

// Fonction appelée lors de l'envoi des données
void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
  Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Données envoyées avec succès" : "Échec de l'envoi des données");
}

// Fonction pour initialiser le MPU-6050
void initMPU6050() {
  Wire.begin(); // Démarre la communication I²C
  Wire.beginTransmission(MPU6050_ADDR);

  if (Wire.endTransmission() != 0) {
    Serial.println("MPU-6050 non détecté !");
    while (1); // Bloque l'exécution si le capteur est absent
```

```
  }

  Serial.println("MPU-6050 détecté !");
  Wire.beginTransmission(MPU6050_ADDR);
  Wire.write(0x6B); // Registre de gestion de l'alimentation
  Wire.write(0x00); // 0x00 met le capteur en mode actif
  Wire.endTransmission();
}

// Fonction pour lire les données d'accélération du MPU-6050
void readAccelData(int16_t *ax, int16_t *ay) {
  Wire.beginTransmission(MPU6050_ADDR);
  Wire.write(0x3B); // Adresse du premier registre contenant les valeurs d'accélération
  Wire.endTransmission(false);
  Wire.requestFrom(MPU6050_ADDR, 6, true); // Demande 6 octets de données
  if (Wire.available() == 6) {
    *ax = Wire.read() << 8 | Wire.read(); // Axe X
    *ay = Wire.read() << 8 | Wire.read(); // Axe Y
    Wire.read(); Wire.read(); // Ignore l'axe Z
  } else {
    Serial.println("Erreur de lecture du MPU-6050 !");
  }
}

// Fonction pour filtrer les valeurs du capteur flex
float getFilteredValue(float *history, int pin) {
  float current = analogRead(pin); // Lit la valeur brute du capteur
  history[filterIndex] = current; // Ajoute la nouvelle valeur à l'historique
  float sum = 0;

  for (int i = 0; i < FILTER_SAMPLES; i++) { // Calcule la somme des valeurs dans l'historique
```

```
    sum += history[i];
  }

  filterIndex = (filterIndex + 1) % FILTER_SAMPLES; // Boucle l'index lorsque filterIndex atteint la fin du tableau
  return sum / FILTER_SAMPLES; // Retourne la moyenne des valeurs
}

void setup() {
  Serial.begin(115200); // Initialisation de la communication série
  initMPU6050();
  WiFi.mode(WIFI_STA); // Configuration du mode WiFi

  esp_wifi_set_max_tx_power(45);

  // Initialisation d'ESP-NOW
  if (esp_now_init() != ESP_OK) {
    Serial.println("Erreur d'initialisation ESP-NOW");
    return;
  }

  esp_now_register_send_cb(OnDataSent); // Enregistre la fonction de callback

  esp_now_peer_info_t peerInfo; // Structure contenant les informations sur la carte réceptrice
  memcpy(peerInfo.peer_addr, broadcastAddress, 6); // Copie l'adresse MAC de la carte réceptrice
  peerInfo.channel = 0;
  peerInfo.encrypt = false; // Désactivation du chiffrement

  // Ajout du pair ESP-NOW
  if (esp_now_add_peer(&peerInfo) != ESP_OK) {
    Serial.println("Échec de l'ajout du pair");
```

```
    return;
  }

  analogReadResolution(12); // Configure l'ESP32 pour des lectures ADC en 12 bits
  initMPU6050(); // Initialisation du capteur MPU-6050
}

void loop() {
  int16_t ax, ay;
  readAccelData(&ax, &ay); // Lire les valeurs X et Y du MPU-6050

  dataToSend.accelX = ax;
  dataToSend.accelY = ay;
  dataToSend.flex = getFilteredValue(flexHistory, FLEX_SENSOR_PIN); // Lit la valeur du capteur flex

  Serial.println(dataToSend.accelY);

  // Envoie les données via ESP-NOW
  esp_now_send(broadcastAddress, (uint8_t *)&dataToSend, sizeof(dataToSend));
  delay(10);
}
```

Le code de la carte émettrice que j'ai aussi légèrement modifié est :

```
#include <ESP32Servo.h>
#include <esp_now.h>
#include <WiFi.h>

#define PIN_SERVO 16          // Définition de la broche pour le servo SG90
#define PIN_SERVO2 13         // Définition de la broche pour le servo SG902
```

```
#define PIN_SERVO3 14        // Définition de la broche pour le servo SG903

#define PIN_SERVO4 15        // Définition de la broche pour le servo SG904

#define SERVO_PIN_5 4        // Définition de la broche pour le servo SG905


Servo sg90;              // Déclaration de l'objet servo pour SG90 (Servo contrôlé par l'axe X de l'accéléromètre)

Servo sg902;             // Déclaration de l'objet servo pour SG902 (Servo du bas contrôlé par l'axe Y de l'accéléromètre)

Servo sg903;             // Déclaration de l'objet servo pour SG903 (Servo du milieu contrôlé par l'axe Y de l'accéléromètre)

Servo sg904;             // Déclaration de l'objet servo pour SG904 (Servo du haut contrôlé par l'axe Y de l'accéléromètre)

Servo sg905;             // Déclaration de l'objet servo pour SG905 (Servo pour l'ouverture et la fermeture de la pince contrôlé par le capteur flex)


const int FLEX_MIN = 3400;     // Définition de la valeur minimale du capteur flex

const int FLEX_MAX = 2700;     // Définition de la valeur maximale du capteur flex

const int SERVO_MIN_ANGLE = 0; // Définition de l'angle minimum pour les servos

const int SERVO_MAX_ANGLE = 90;// Définition de l'angle maximum pour les servos

const int SERVO_MIN_ANGLE_FLEX = 20;// Définition de l'angle maximum pour le servo du capteur flex

const int ANGLE_CHANGE_THRESHOLD = 1; // Seuil à dépasser pour changer l'angle du servo dans certains cas


float currentAngleSg90 = 0;    // Variable pour suivre l'angle actuel du servo SG90

float currentAngleSg902 = 0;   // Variable pour suivre l'angle actuel du servo SG902

float currentAngleSg903 = 0;   // Variable pour suivre l'angle actuel du servo SG903

float currentAngleSg904 = 0;   // Variable pour suivre l'angle actuel du servo SG904

float currentAngleSg905 = 0;   // Variable pour suivre l'angle actuel du servo SG905


struct SensorData {         // Définition d'une structure pour contenir les données du capteur
  float accelX;          // Accéléromètre sur l'axe X
  float accelY;          // Accéléromètre sur l'axe Y
  int flex;              // Valeur du capteur flex
```

```
};

SensorData receivedData;      // Déclaration de l'objet `receivedData` pour stocker les données reçues


void OnDataRecv(const esp_now_recv_info_t *recv_info, const uint8_t *incomingData, int len) {
  SensorData receivedData;  // Déclaration d'une variable locale pour stocker les données reçues
  memcpy(&receivedData, incomingData, sizeof(receivedData)); // Copie les données reçues dans receivedData


  // Calcul de l'angle du servo pour le capteur flex
  int servoAngle = map(receivedData.flex, FLEX_MIN, FLEX_MAX, SERVO_MAX_ANGLE, SERVO_MIN_ANGLE_FLEX); // Mapping de la valeur flex à un angle de servo
  servoAngle = constrain(servoAngle, SERVO_MIN_ANGLE_FLEX, SERVO_MAX_ANGLE); // On vérifie que le résultat est dans la plage de données valide
  if (abs(servoAngle - currentAngleSg905) >= ANGLE_CHANGE_THRESHOLD) { // Vérification si l'angle a changé d'au moins 5 degré
    sg905.write(servoAngle); // Envoie la nouvelle valeur d'angle au servo SG905
    currentAngleSg905 = servoAngle; // Met à jour l'angle actuel de SG905
  }


  // Calcul de l'angle pour l'accéléromètre X
  int mappedAngleSg90 = map(receivedData.accelX, 12000, -12000, 0, 90); // Mapping de la valeur de l'accéléromètre X à un angle de servo.
  mappedAngleSg90 = constrain(mappedAngleSg90, SERVO_MIN_ANGLE, SERVO_MAX_ANGLE); // On vérifie que le résultat est dans la plage de données valide
  if (abs(mappedAngleSg90 - currentAngleSg90) >= ANGLE_CHANGE_THRESHOLD) { // Vérification si l'angle a changé d'au moins 5 degré
    sg90.write(mappedAngleSg90); // Envoie la nouvelle valeur d'angle au servo SG90
    currentAngleSg90 = mappedAngleSg90; // Met à jour l'angle actuel de SG90
  }


  // Autres mouvements en fonction de Y
  if (receivedData.accelY < -10000) { // Si la valeur de Y est inférieure à 700
```

```
    sg902.write(0); // Met le servo SG902 à 0°.
    sg903.write(0); // Met le servo SG903 à 0°.
    sg904.write(0); // Met le servo SG904 à 0°.
    currentAngleSg902 = currentAngleSg903 = currentAngleSg904 = 0; // Met à jour les angles actuels
  } else if (receivedData.accelY < -7700) { // Si la valeur de Y est inférieure à 800
    int angle1 = map(receivedData.accelY, -10000, -7700, 0, 50);
    int angle2 = map(receivedData.accelY, -10000, -7700, 0, 35); // Mappe la valeur de Y à un angle entre 0 et 30°
    sg902.write(angle1); // Envoie l'angle au servo SG902
    sg903.write(0); // Met le servo SG903 à 0°.
    sg904.write(angle2); // Met le servo SG904 à 0°.
    currentAngleSg902 = angle1; // Met à jour l'angle actuel de SG902
    currentAngleSg903  = 0; // Met à jour les angles actuels de SG903 et SG904
    currentAngleSg904=angle2;
  } else if (receivedData.accelY < -5400) { // Si la valeur de Y est inférieure à 900
    int angle1 = map(receivedData.accelY, -7700, -5400, 50, 35); // Mappe la valeur de Y à un angle entre 0 et 30°
    int angle2 = map(receivedData.accelY, -7700, -5400, 0, 15); // Mappe la valeur de Y à un angle entre 0 et 30°
    int angle3 = map(receivedData.accelY, -7700, -5400, 35, 45); // Mappe la valeur de Y à un angle entre 0 et 30°
    sg902.write(angle1); // Met le servo SG902 à 30°
    sg903.write(angle2); // Envoie l'angle au servo SG903
    sg904.write(angle3); // Met le servo SG904 à 0°
    currentAngleSg902 = angle1; // Met à jour l'angle actuel de SG902
    currentAngleSg903 = angle2; // Met à jour l'angle actuel de SG903
    currentAngleSg904 = angle3; // Met à jour l'angle actuel de SG904
  } else if (receivedData.accelY < -3100) { // Si la valeur de Y est inférieure à 1050
    int angle1 = map(receivedData.accelY, -5400, -3100, 35, 30); // Mappe la valeur de Y à un angle entre 0 et 30°
    int angle2 = map(receivedData.accelY, -5400, -3100, 15, 20); // Mappe la valeur de Y à un angle entre 0 et 30°
```

```
    int angle3 = map(receivedData.accelY, -5400, -3100, 45, 45); // Mappe la valeur de Y à un angle entre 0 et 30°

    sg902.write(angle1); // Met le servo SG902 à 30°

    sg903.write(angle2); // Envoie l'angle au servo SG903

    sg904.write(angle3); // Met le servo SG904 à 0°

    currentAngleSg902 = angle1; // Met à jour l'angle actuel de SG902

    currentAngleSg903 = angle2; // Met à jour l'angle actuel de SG903

    currentAngleSg904 = angle3; // Met à jour l'angle actuel de SG904

  }else if (receivedData.accelY < -800) { // Si la valeur de Y est inférieure à 1050

    int angle1 = map(receivedData.accelY, -3100, -800, 30, 25); // Mappe la valeur de Y à un angle entre 0 et 30°

    int angle2 = map(receivedData.accelY, -3100, -800, 20, 35); // Mappe la valeur de Y à un angle entre 0 et 30°

    int angle3 = map(receivedData.accelY, -3100, -800, 45, 45); // Mappe la valeur de Y à un angle entre 0 et 30°

    sg902.write(angle1); // Met le servo SG902 à 30°

    sg903.write(angle2); // Envoie l'angle au servo SG903

    sg904.write(angle3); // Met le servo SG904 à 0°

    currentAngleSg902 = angle1; // Met à jour l'angle actuel de SG902

    currentAngleSg903 = angle2; // Met à jour l'angle actuel de SG903

    currentAngleSg904 = angle3; // Met à jour l'angle actuel de SG904

  } else if (receivedData.accelY < 1500) { // Si la valeur de Y est inférieure à 1050

    int angle1 = map(receivedData.accelY, -800, 1500, 25, 20); // Mappe la valeur de Y à un angle entre 0 et 30°

    int angle2 = map(receivedData.accelY, -800, 1500, 35, 40); // Mappe la valeur de Y à un angle entre 0 et 30°

    int angle3 = map(receivedData.accelY, -800, 1500, 45, 45); // Mappe la valeur de Y à un angle entre 0 et 30°

    sg902.write(angle1); // Met le servo SG902 à 30°

    sg903.write(angle2); // Envoie l'angle au servo SG903

    sg904.write(angle3); // Met le servo SG904 à 0°

    currentAngleSg902 = angle1; // Met à jour l'angle actuel de SG902

    currentAngleSg903 = angle2; // Met à jour l'angle actuel de SG903

    currentAngleSg904 = angle3; // Met à jour l'angle actuel de SG904
```

```
  } else if (receivedData.accelY < 3800) { // Si la valeur de Y est inférieure à 1050
    int angle1 = map(receivedData.accelY, 1500, 3800, 20, 15); // Mappe la valeur de Y à un angle entre 0 et 30°
    int angle2 = map(receivedData.accelY, 1500, 3800, 40, 50); // Mappe la valeur de Y à un angle entre 0 et 30°
    int angle3 = map(receivedData.accelY, 1500, 3800, 45, 45); // Mappe la valeur de Y à un angle entre 0 et 30°
    sg902.write(angle1); // Met le servo SG902 à 30°
    sg903.write(angle2); // Envoie l'angle au servo SG903
    sg904.write(angle3); // Met le servo SG904 à 0°
    currentAngleSg902 = angle1; // Met à jour l'angle actuel de SG902
    currentAngleSg903 = angle2; // Met à jour l'angle actuel de SG903
    currentAngleSg904 = angle3; // Met à jour l'angle actuel de SG904
  } else if (receivedData.accelY < 6100) { // Si la valeur de Y est inférieure à 1050
    int angle1 = map(receivedData.accelY, 3800, 6100, 15, 10); // Mappe la valeur de Y à un angle entre 0 et 30°
    int angle2 = map(receivedData.accelY, 3800, 6100, 50, 60); // Mappe la valeur de Y à un angle entre 0 et 30°
    int angle3 = map(receivedData.accelY, 3800, 6100, 45, 45); // Mappe la valeur de Y à un angle entre 0 et 30°
    sg902.write(angle1); // Met le servo SG902 à 30°
    sg903.write(angle2); // Envoie l'angle au servo SG903
    sg904.write(angle3); // Met le servo SG904 à 0°
    currentAngleSg902 = angle1; // Met à jour l'angle actuel de SG902
    currentAngleSg903 = angle2; // Met à jour l'angle actuel de SG903
    currentAngleSg904 = angle3; // Met à jour l'angle actuel de SG904
  }
  else if (receivedData.accelY < 8400) { // Si la valeur de Y est inférieure à 1050
    int angle1 = map(receivedData.accelY, 6100, 8400, 10, 5); // Mappe la valeur de Y à un angle entre 0 et 30°
    int angle2 = map(receivedData.accelY, 6100, 8400, 60, 65); // Mappe la valeur de Y à un angle entre 0 et 30°
    int angle3 = map(receivedData.accelY, 6100, 8400, 45, 45); // Mappe la valeur de Y à un angle entre 0 et 30°
```

```
    sg902.write(angle1); // Met le servo SG902 à 30°
    sg903.write(angle2); // Envoie l'angle au servo SG903
    sg904.write(angle3); // Met le servo SG904 à 0°
    currentAngleSg902 = angle1; // Met à jour l'angle actuel de SG902
    currentAngleSg903 = angle2; // Met à jour l'angle actuel de SG903
    currentAngleSg904 = angle3; // Met à jour l'angle actuel de SG904
  }
  else if (receivedData.accelY < 10700) { // Si la valeur de Y est inférieure à 1050
    int angle1 = map(receivedData.accelY, 8400, 10700, 5, 0); // Mappe la valeur de Y à un angle entre 0 et 30°
    int angle2 = map(receivedData.accelY, 8400, 10700, 65, 70); // Mappe la valeur de Y à un angle entre 0 et 30°
    int angle3 = map(receivedData.accelY, 8400, 10700, 45, 45); // Mappe la valeur de Y à un angle entre 0 et 30°
    sg902.write(angle1); // Met le servo SG902 à 30°
    sg903.write(angle2); // Envoie l'angle au servo SG903
    sg904.write(angle3); // Met le servo SG904 à 0°
    currentAngleSg902 = angle1; // Met à jour l'angle actuel de SG902
    currentAngleSg903 = angle2; // Met à jour l'angle actuel de SG903
    currentAngleSg904 = angle3; // Met à jour l'angle actuel de SG904
  } else if (receivedData.accelY < 13000) { // Si la valeur de Y est inférieure à 1050
    int angle1 = map(receivedData.accelY, 10700, 13000, 0, 0); // Mappe la valeur de Y à un angle entre 0 et 30°
    int angle2 = map(receivedData.accelY, 10700, 13000, 70, 80); // Mappe la valeur de Y à un angle entre 0 et 30°
    int angle3 = map(receivedData.accelY, 10700, 13000, 45, 50); // Mappe la valeur de Y à un angle entre 0 et 30°
    sg902.write(angle1); // Met le servo SG902 à 30°
    sg903.write(angle2); // Envoie l'angle au servo SG903
    sg904.write(angle3); // Met le servo SG904 à 0°
    currentAngleSg902 = angle1; // Met à jour l'angle actuel de SG902
    currentAngleSg903 = angle2; // Met à jour l'angle actuel de SG903
    currentAngleSg904 = angle3; // Met à jour l'angle actuel de SG904
```

```
  }
  else { // Si la valeur de Y est supérieure ou égale à 1050
    int angle1 = 0; // Mappe la valeur de Y à un angle entre 0 et 30°
    int angle2 = 75; // Mappe la valeur de Y à un angle entre 0 et 30°
    int angle3 = 65; // Mappe la valeur de Y à un angle entre 0 et 30°
    sg902.write(angle1); // Met le servo SG902 à 30°
    sg903.write(angle2); // Envoie l'angle au servo SG903
    sg904.write(angle3); // Met le servo SG904 à 0°
    currentAngleSg902 = angle1; // Met à jour l'angle actuel de SG902
    currentAngleSg903 = angle2; // Met à jour l'angle actuel de SG903
    currentAngleSg904 = angle3; // Met à jour l'angle actuel de SG904
  }
  Serial.println(receivedData.accelY);
}

void setup() {
  Serial.begin(115200);
  WiFi.mode(WIFI_STA); // Configure l'ESP32 en mode WiFi

  // Initialisation d'ESP-NOW
  if (esp_now_init() != ESP_OK) { // Initialise ESP-NOW pour la communication sans fil
    Serial.println("Erreur d'initialisation ESP-NOW"); // Si l'initialisation échoue, afficher un
message d'erreur
    return;
  }

  esp_now_register_recv_cb(OnDataRecv); // Enregistre la fonction qui sera appelée à chaque
tentative d'envoie de données via ESP-NOW

  sg90.attach(PIN_SERVO);       // Attache le servo SG90 à la broche PIN_SERVO
  sg902.attach(PIN_SERVO2);     // Attache le servo SG902 à la broche PIN_SERVO2
  sg903.attach(PIN_SERVO3);     // Attache le servo SG903 à la broche PIN_SERVO3
```

```
  sg904.attach(PIN_SERVO4);      // Attache le servo SG904 à la broche PIN_SERVO4
  sg905.attach(SERVO_PIN_5);     // Attache le servo SG905 à la broche SERVO_PIN_5

  sg90.write(0);              // Initialise le servo SG90 à 0°
  sg902.write(0);             // Initialise le servo SG902 à 0°
  sg903.write(0);             // Initialise le servo SG903 à 0°
  sg904.write(0);             // Initialise le servo SG904 à 0°
  sg905.write(0);             // Initialise le servo SG905 à 0°
}

void loop() {
  // Boucle principale vide car l'action est déclenchée par la réception de données via ESP-NOW
}
```

Fermeture de la pince :

Une fois que j'ai réussi à régler le problème de transmission, j'ai voulu faire des tests sur la fermeture de la pince car c'est un élément que j'ai très peu testé car ce n'est pas vraiment au point et la modélisation 3d utilisée fait que le fonctionnement de départ n'est pas super. Cependant, en faisant des tests j'ai cassé le haut de la pince et je devrais donc le remplacer lors de la prochaine séance.

Objectif de la prochaine séance :

L'objectif de la prochaine séance pour moi sera donc de travailler sur la fermeture de la pince et d'essayer d'utiliser un capteur de pression que je fixerais sur la pince afin de faire en sorte que quand le servo moteur de la pince force, il arrête de forcer car le faire qu'il force abime la pince et a tendance à la désaxer et donc à la démonter.