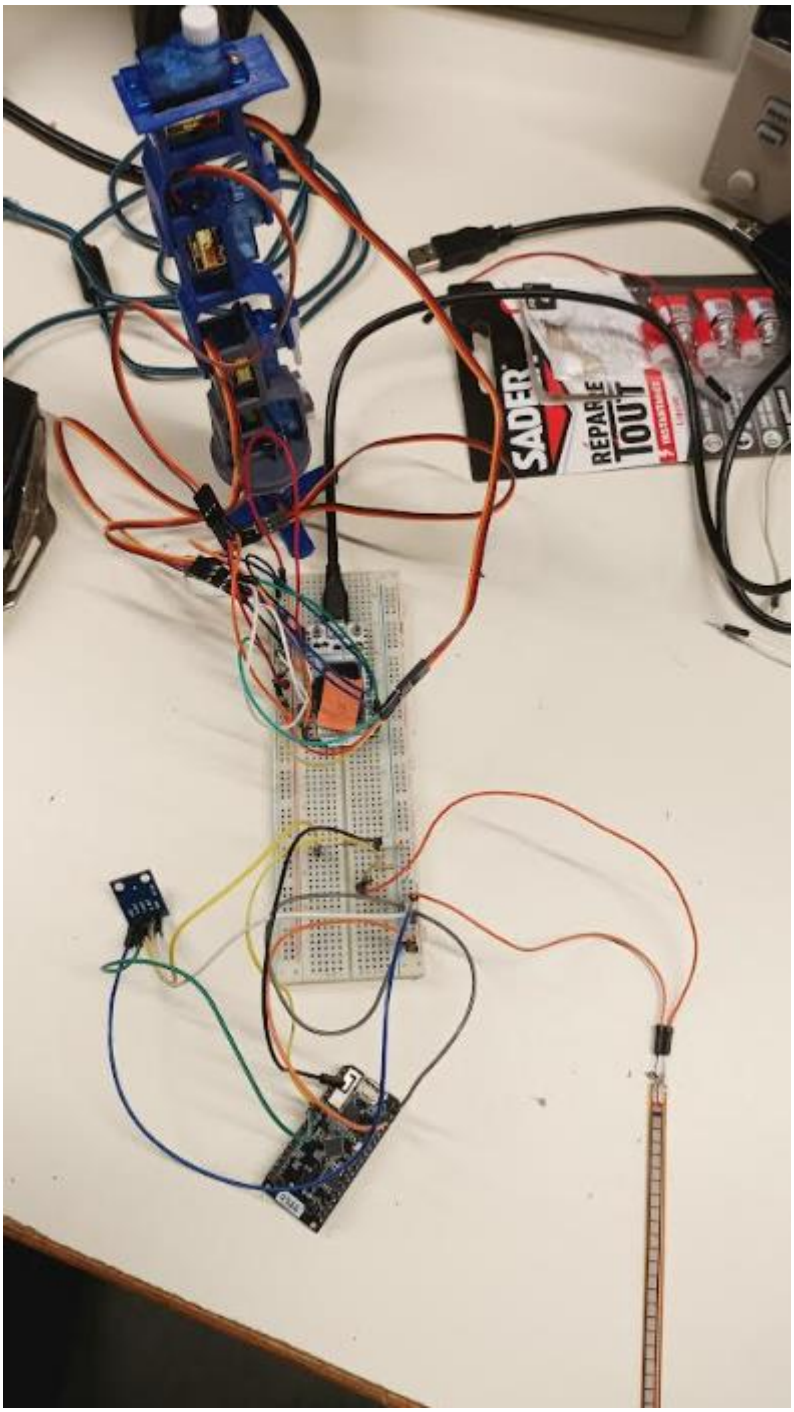
Compte rendu séance 5 Téo Baillot d'Estivaux

Objectifs de la séance :

Puisque j'ai réussi à établir une communication Wifi entre les deux cartes en utilisant ESP-NOW pour envoyer un message d'une carte à l'autre lors de la dernière séance, l'objectif de cette séance sera d'envoyer les données du capteur flex et de l'accéléromètre depuis une carte à l'autre carte qui interprétera ces données pour contrôler les mouvements de la pince. Il faut donc établir le fonctionnement final du bras robotique industriel en utilisant une communication à distance.

Câblage pour le test :

Avant de faire un code et de le tester, il a fallu que je mette en place le montage pour chaque carte. Pour ce faire, j'ai utilisé les mêmes PIN que lorsque je ne faisais pas la communication à distance entre deux cartes. Finalement, j'ai obtenu le montage suivant :



Impression résine :

Avant les vacances de Noël, j'ai lancé les impressions des pièces qui posaient problème et qui cassaient souvent en résine pour essayer de les rendre plus solides. Il a donc fallu mettre ces pièces à laver pendant 10 minutes pour enlever la résine restante des pièces qui étaient collantes puis les passer une heure aux rayons UV. J'essayerais de monter ces pièces sur le bras robotique lors de la prochaine séance.

Code :

Une fois le montage mis en place, j'ai fait les codes pour la carte émettrice et la carte réceptrice. Pour ce faire, je me suis inspiré à la fois des codes que j'ai pu faire lors de la dernière séance pour envoyer un message d'une carte à l'autre et du code de la version finale du fonctionnement du bras robotique sans communication distante.

Finalement, après plusieurs essais, modifications, et tests, j'ai pu faire les codes suivants :

Code de la carte émettrice :

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

#define PIN_ACCEL_X 34        // Définit le GPIO utilisé pour lire les valeurs de l'accéléromètre sur l'axe X
#define PIN_ACCEL_Y 35        // Définit le GPIO utilisé pour lire les valeurs de l'accéléromètre sur l'axe Y
#define FLEX_SENSOR_PIN 39    // Définit le 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 pour la transmission ESP-NOW (adresse MAC de la carte réceptrice)

// Historique pour filtrer les données des capteurs d'accélération
float xHistory[FILTER_SAMPLES];
float yHistory[FILTER_SAMPLES];
int filterIndex = 0; // Index pour parcourir les historiques

// Structure pour envoyer les données des capteurs
struct SensorData {
    float accelX; // Valeur filtrée de l'accélération sur l'axe X
    float accelY; // Valeur filtrée de l'accélération sur l'axe Y
    int flex;     // Valeur du capteur flex
};
SensorData dataToSend; // Instance de la structure utilisée pour l'envoi des données

// Fonction appelée lorsque les données sont envoyées pour savoir si l'envoie est réussi ou non
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");
}

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

```
    // 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 qui sera
appelée à chaque tentative d'envoie de données via ESP-NOW

    esp_now_peer_info_t peerInfo; // Structure contenant les informations sur
la carte réceptrice
    memcpy(peerInfo.peer_addr, broadcastAddress, 6); // On copie l'adresse MAC
de la carte réceptrice
    peerInfo.channel = 0;                           // On définit sur quel
canal Wifi on va communiquer
    peerInfo.encrypt = false;                       // On désactive le
chiffrement des données envoyées

    // Enregistre la carte réceptrice dans la liste des pairs ESP-NOW et met
un message d'erreur si l'ajout échoue
    if (esp_now_add_peer(&peerInfo) != ESP_OK) {
        Serial.println("Échec de l'ajout du pair");
        return;
    }

    analogReadResolution(12); // On configure l'ESP32 pour que son
convertisseur analogique-numérique effectue des lectures avec une résolution
de 12 bits
}

// Fonction pour filtrer les valeurs des capteurs en utilisant un historique
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; // L’opérateur modulo
permet de boucler l’index lorsque filterIndex atteint la fin du tableau.
    return sum / FILTER_SAMPLES;                    // Retourne la moyenne
des valeurs
}

void loop() {
    // Récupère les valeurs filtrées des accéléromètres
```

```
    dataToSend.accelX = getFilteredValue(xHistory, PIN_ACCEL_X);
    dataToSend.accelY = getFilteredValue(yHistory, PIN_ACCEL_Y);

    // Lit la valeur du capteur flex
    dataToSend.flex = analogRead(FLEX_SENSOR_PIN);

    // Envoie les données via ESP-NOW
    esp_now_send(broadcastAddress, (uint8_t *)&dataToSend,
sizeof(dataToSend)); // effectue un cast pour convertir le pointeur vers
dataToSend en un pointeur générique de type uint8_t *, qui est attendu par la
fonction esp_now_send
    delay(10);
}
```

Code de la carte réceptrice :

```
#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 = 3500;      // Définition de la valeur minimale du capteur
flex
const int FLEX_MAX = 3000;      // 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 ANGLE_CHANGE_THRESHOLD = 5; // 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); // Mapping de la valeur flex à un angle de servo
    servoAngle = constrain(servoAngle, SERVO_MIN_ANGLE, 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, 1050, 700, 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 < 850) { // Si la valeur de Y est inférieure à 850
        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 < 900) { // Si la valeur de Y est
inférieure à 900
        int angle2 = map(receivedData.accelY, 850, 900, 0, 30); // Mappe la
valeur de Y à un angle entre 0 et 30°
        sg902.write(angle2); // Envoie l'angle au servo SG902
        sg903.write(0); // Met le servo SG903 à 0°.
        sg904.write(0); // Met le servo SG904 à 0°.
        currentAngleSg902 = angle2; // Met à jour l'angle actuel de SG902
        currentAngleSg903 = currentAngleSg904 = 0; // Met à jour les angles
actuels de SG903 et SG904
    } else if (receivedData.accelY < 950) { // Si la valeur de Y est
inférieure à 950
        int angle3 = map(receivedData.accelY, 900, 950, 0, 30); // Mappe la
valeur de Y à un angle entre 0 et 30°
        sg903.write(angle3); // Envoie l'angle au servo SG903
        sg902.write(30); // Met le servo SG902 à 30°
        sg904.write(0); // Met le servo SG904 à 0°
        currentAngleSg903 = angle3; // Met à jour l'angle actuel de SG903
        currentAngleSg902 = 30; // Met à jour l'angle actuel de SG902
        currentAngleSg904 = 0; // Met à jour l'angle actuel de SG904
    } else if (receivedData.accelY < 1000) { // Si la valeur de Y est
inférieure à 1000
        int angle4 = map(receivedData.accelY, 950, 1000, 0, 30); // Mappe la
valeur de Y à un angle entre 0 et 30°
        sg904.write(angle4); // Envoie l'angle au servo SG904
        sg902.write(30); // Met le servo SG902 à 30°
        sg903.write(30); // Met le servo SG903 à 30°
        currentAngleSg904 = angle4; // Met à jour l'angle actuel de SG904
        currentAngleSg902 = currentAngleSg903 = 30; // Met à jour les angles
actuels de SG902 et SG903
    } else { // Si la valeur de Y est supérieure ou égale à 1000
        sg902.write(30); // Met le servo SG902 à 30°
        sg903.write(30); // Met le servo SG903 à 30°
        sg904.write(30); // Met le servo SG904 à 30°
```

```
        currentAngleSg902 = currentAngleSg903 = currentAngleSg904 = 30; // Met
à jour les angles actuels de SG902, SG903 et SG904
    }
}

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
}
```

Dans ces codes, j'ai essayé d'ajuster au mieux les délais internes au code pour pouvoir transmettre les données à la plus grande fréquence possible. Finalement, il reste un léger délai dû à la communication, mais ce délai reste faible et a été grandement réduit à la suite de nombreux tests et ajustements du code, et il n'empêche pas de pouvoir bien maîtriser le contrôle de la pince. Des tests

complémentaires seront nécessaires mais on s'approche donc de la version finale du bras robotique grâce au travail de cette séance.

Objectif de la prochaine séance :

L'objectif de la prochaine séance sera d'installer les nouvelles pièces en résine sur le bras robotique et de faire des tests approfondis du fonctionnement de la pince pour détecter d'éventuels problèmes que j'aurais pu ne pas voir et s'assurer que tout fonctionne parfaitement, notamment au niveau de la fermeture de la pince car à cause des pièces qui ont eu tendance à se casser au niveau de la pince, j'ai très peu testé cette partie du bras robotique.