Compte rendu séance 7 Téo Baillot d'Estivaux :

Objectifs de la séance :

L'objectif de cette séance est de finir le montage de la pince et de tester de façon approfondie son fonctionnement pour s'assurer que tout fonctionne correctement.

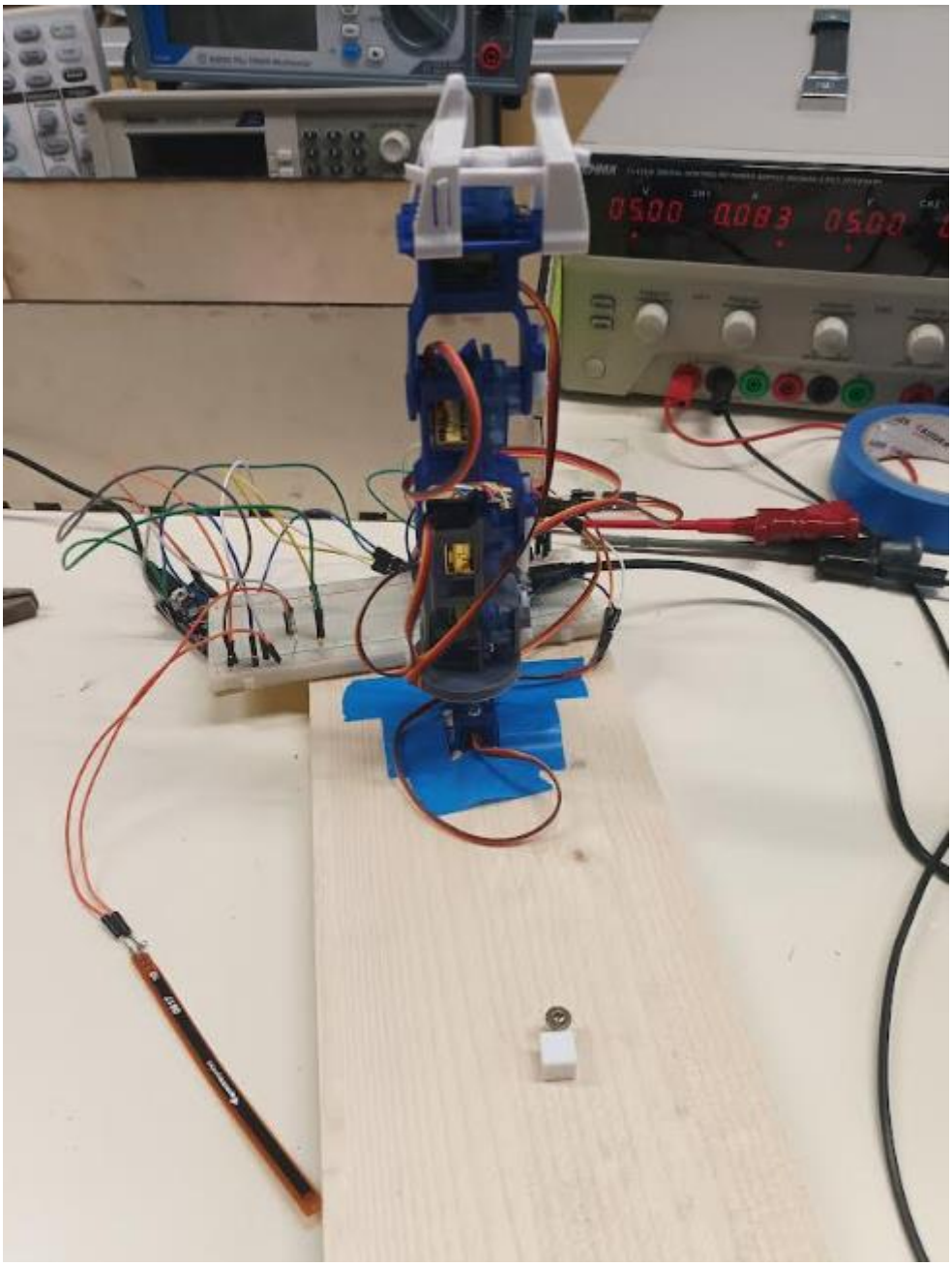
Problèmes de débuts de séance :

Lorsque je suis arrivé en début de séance, j'ai remarqué que la pince ne fonctionnait plus du tout alors qu'elle fonctionnait en fin de séance dernière. En testant, je me suis aperçu que la plage de valeur renvoyée par l'accéléromètre a changé par rapport à la dernière séance. J'ai donc ajusté la plage de valeur utilisée pour contrôler les servos moteurs dans le code de la carte réceptrice.
Malgré ces modifications, il n'y avait qu'un servo moteur qui fonctionnait sur les 3 alors que le code semblait bon. Je me suis donc dit que le problème venait surement de l'alimentation (même s'il n'y avait pas eu de problèmes lors de la dernière séance) donc j'ai utilisé une alimentation externe pour envoyer du 5V à tous les servos moteurs. Le changement d'alimentation a effectivement réglé le problème et j'ai donc pu reprendre là où j'avais arrêté lors de la dernière séance.
Pour régler le problème de la plage de valeur de l'accéléromètre qui change régulièrement, peut-être qu'à terme je changerais l'accéléromètre analogique pour un accéléromètre numérique qui devrait régler ce problème.

Montage et test de la fermeture de la pince :

Lorsque je suis allé au Fablab pour récupérer les pièces en résine dont j'avais lancé l'impression lors de la dernière séance, je me suis aperçu que celles-ci avaient disparus, j'ai donc relancé une nouvelle fois ces impressions et espère pouvoir les récupérer lors de la prochaine séance.

J'ai donc fait le montage de la pince avec d'anciennes modélisations 3d pas très adaptées pour quand même pouvoir tester.



Après avoir monté la pince, j'ai testé qu'elle fonctionnait correctement et je me suis rendu compte que les valeurs renvoyées par le capteur flex variaient trop pour une même position, j'ai donc ajouté un filtre de moyenne comme pour l'accéléromètre sur le capteur flex afin d'avoir des données plus précises et ai donc du ajuster le fonctionnement du servo moteur depuis la carte réceptrice en fonction de la nouvelle plage de valeur du capteur flex.

Dans la suite des tests que j'ai effectué, je me suis rendu compte que les déplacements de la pince se faisaient par à-coup, je me suis donc penché sur la transmission des données de la carte émettrice à la carte réceptrice et me suis donc rendu compte qu'effectivement certains paquets de données n'arrivaient pas à destination. J'ai donc essayé de régler ce problème. Je me suis d'abord dit que le délai de 10ms à la fin du code était trop court et que c'était peu être à cause de ça que des paquets se perdaient, j'ai donc augmenté ce délai mais ça n'a pas réglé le

problème. Le reste de la séance j'ai cherché une solution pour régler ce problème de paquets qui ne s'envoient pas mais je n'ai malheureusement pas réussi à trouver de solution.

Pendant la séance, j'ai également essayé d'attraper un petit cube avec la pince, ce qui n'est pas très simple à faire car la sensibilité est assez grande et la perte de paquets mentionnée plutôt a pour conséquence que parfois la pince se déplace par à-coup rendant le contrôle de ses mouvements difficile. Il est tout de même possible avec un peu d'entrainement de l'attraper. Un autre problème est que si on sert trop fort l'objet, cela va désaxer la crémaillère et la pince peut se démonter. Pour régler ce problème, il faudra donc ajouter un capteur de pression à la pince ou quelque chose dans le genre.

Version finale du code :

Finalement, la version finale du code pour se semestre est :

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];

float flexHistory[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 = getFilteredValue(flexHistory, FLEX_SENSOR_PIN);
  Serial.println(getFilteredValue(flexHistory, 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 = 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, 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 < 700) { // 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 < 800) { // Si la valeur de Y est inférieure à 800
    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 < 900) { // Si la valeur de Y est inférieure à 900
    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 < 1050) { // Si la valeur de Y est inférieure à 1050
      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 1050
      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
}
```

Objectifs de la prochaine séance :

Cette séance étant la dernière de ce semestre, l'objectif de la prochaine séance sera de présenter le projet et d'en faire une démonstration.