Date : 11/04/2025

Compte-rendu séance 15 Téo Baillot d'Estivaux :

Objectif de la séance :

L'objectif de cette séance est d'améliorer la démonstration que l'on fera, notamment en modifiant encore les modélisations 3d utilisés pour la rampe car le problème est que lorsque l'on déplace à une même position un servo moteur, d'une fois à l'autre il y a un léger décalage d'un ou deux millimètres, et ce manque de précision pose des problèmes lors de la démonstration. Je vais donc ajuster la largeur de la rampe pour pallier ce problème. L'objectif sera également de peaufiner le code que ce soit au niveau de l'optimisation ou de la gestion des délais internes ainsi que de faire des tests poussés pour trouver des failles dans le fonctionnement de la pince. Il faudra également que je teste le fonctionnement de la carte de Matis et réaliser le gant qui permettra de contrôler les mouvements du bras robotique.
Enfin, il faudra que je code le soft pour le nouveau composant de la carte réceptrice qui va permettre de détecter la surtension du moteur qui permet de fermer la pince et qui traduit que le servo-moteur force, ce qui peut désaxer la pince et l'empêcher de fonctionner correctement par la suite.

Pour pouvoir réaliser tout ceci, j'ai donc dû travailler de nombreuses heures chez moi, sans quoi nous n'aurions pas pu finir ce projet.

Amélioration de la démonstration :

Pour améliorer la démonstration que l'on fera lors de la présentation, j'ai ajusté les mouvements du mode automatique de façon à ce qu'ils soient le plus fluide, le plus précis, et le plus plaisant à voir possible. J'ai également voulu faire en sorte que la pince se déplacer plus rapidement en réduisant le délai entre les positions des servos-moteurs sauf que je me suis rapidement rendu compte que réduire ce délai augmente considérablement l'imprécision des servo-moteurs. J'ai donc assez rapidement laissé tomber l'idée d'augmenter la vitesse des mouvements.

En testant le mode manuel du bras robotique, je me suis dis qu'il faudrait quand même que je fasse en sorte que les mouvements du bras robotique soient moins brusques et plus contrôlable pour que ce soit plus simple à utiliser. J'ai donc modifié l'association des valeurs de l'accéléromètre et des mouvements et j'ai également fait une nouvelle fonction permettant de bouger les servo-moteurs plus doucement et donc d'avoir des mouvements moins brusque :
void moveServoSmoothManual(

```
  Servo &servo0, float &current0, int target0,  //  sg90
  Servo &servo1, float &current1, int target1,  // sg902
  Servo &servo2, float &current2, int target2,  //  sg903
  Servo &servo3, float &current3, int target3   // sg904
) {
  int maxSteps = max(max(abs(current0 - target0), abs(current1 - target1)),
```

```
            max(abs(current2 - target2), abs(current3 - target3)));
if (maxSteps == 0) return;

float step0 = (target0 - current0) / maxSteps;
float step1 = (target1 - current1) / maxSteps;
float step2 = (target2 - current2) / maxSteps;
float step3 = (target3 - current3) / maxSteps;

float pos0 = current0;
float pos1 = current1;
float pos2 = current2;
float pos3 = current3;

for (int i = 0; i <= maxSteps; i++) {
  servo0.write(round(pos0));
  servo1.write(round(pos1));
  servo2.write(round(pos2));
  servo3.write(round(pos3));

  pos0 += step0;
  pos1 += step1;
  pos2 += step2;
  pos3 += step3;

  delay(10);
}

current0 = target0;
current1 = target1;
current2 = target2;
current3 = target3;
```

```
}
```

La difficulté de cette fonction a été de faire en sorte de pouvoir déplacer les servo-moteurs en même temps alors qu'ils ne font pas les mêmes mouvements. Pour ce faire, l'idée est de calculer le déplacement maximum parmi tous les servo-moteurs. Ensuite on calcule le nombre de pas à faire pour chaque servo-moteur en fonction du nombre de pas maximum. Puis on fait une boucle avec le nombre de déplacements maximum obtenus et on déplace les servo-moteurs avec les différents pas calculés.

Ajout du code pour vérifier si le servo-moteur force :

Comme expliqué dans l'introduction, on a décidé d'utiliser un INA219AxDCN qui va permettre de voir en lisant la tension du servo moteur qui permet de fermer la pince s'il force ou non, et le cas échéant, de faire en sorte qu'il ne force plus. Pour ce faire, j'ai donc crée une fonction qui me permet de lire le courant qui est tiré par le servo-moteur est décidé si je change la position du servo-moteur en fonction du courant qu'il tire :

```
void adjustServoAngleIfNeeded(Servo &servo, float &currentAngle, int targetAngle) {
 float current_mA = ina219.getCurrent_mA();
  if(currentAngle < targetAngle) {
   servo.write(targetAngle);
   currentAngle = targetAngle;
  }
  else if (currentAngle > targetAngle) {
    // Fermeture lente : angle diminue doucement
    for (int pos = currentAngle; pos >= targetAngle; pos-=0.25) {
      current_mA = ina219.getCurrent_mA();

      if (current_mA < CourantMax) {
        Serial.println("La pince force");
        break;
      }

      servo.write(pos);
      delay(75);  // ralentir la fermeture
      currentAngle = pos;
```

```
    }
  }
}
```

Pour que ce code fonctionne avec notre bras robotique, j'ai dû refaire le modèle de roue dentée car le problème avec l'ancienne est que dès que le servo-moteur forçait la roue continuait a tourner quasiment comme si de rien n'était car ses dents étaient trop petites. Pour remplacer l'ancienne roue dentée par la nouvelle, j'ai donc essayé de faire et d'imprimer une dizaine de roue dentée différentes avant de réussir à en avoir une qui correspond bien car le problème est que la taille de ma roue dentée est égale au nombre de dent de ma roue dentée multiplié par le diamètre de la roue dentée. Sachant que je veux garder un trou de 5mm de diamètre au centre de la roue dentée pour pouvoir la fixer au servo-moteur, il m'est physiquement impossible de faire la roue dentée idéale. J'ai donc fait plusieurs roues dentées qui se rapprochent de celle idéale et les ai testées à chaque fois jusqu'à en avoir une qui me convienne et qui permette de pouvoir voir si le servo-moteur force.

Tests divers :

J'ai effectué énormément de tests différents qui m'ont permis de voir si le bras robotique fonctionne bien, y compris sur une longue période, et je me suis rendu compte qu'à cause de l'imprécision des servo-moteurs, la pince finissais par mettre le cube sur le bord de la rampe et donc ne réussit plus à le récupérer car il se coince au bout d'un moment.

J'ai également vérifié que l'on peut passer du mode automatique au mode manuel plusieurs fois, il n'y a pas de problème de ce côté-là. J'ai cependant remarqué que parfois il y a un rebond du bouton poussoir qui fait que lorsque on appuie dessus une fois le rebond fait que le code comprend qu'on appuie deux fois, j'ai donc ajouté un délai après le premier appuie du bouton pendant lequel on ne relève plus de nouvelle valeur pour éviter ce problème de rebond.

J'ai également retravaillé la mobilité de la pince dans le mode manuel pour peaufiner le contrôle du bras robotique de façon manuel.

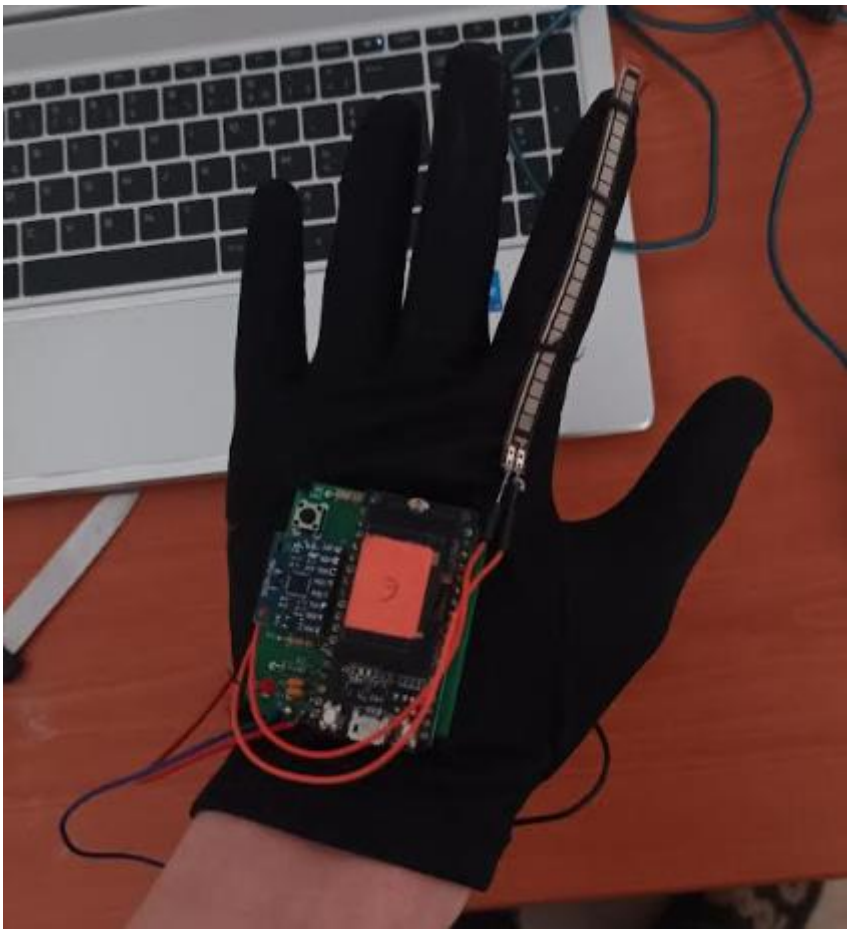
J'ai ensuite essayer de nouveau de travailler sur le problème des pertes de paquets qui reste mais n'ai malheureusement pas réussi à régler ce problème.

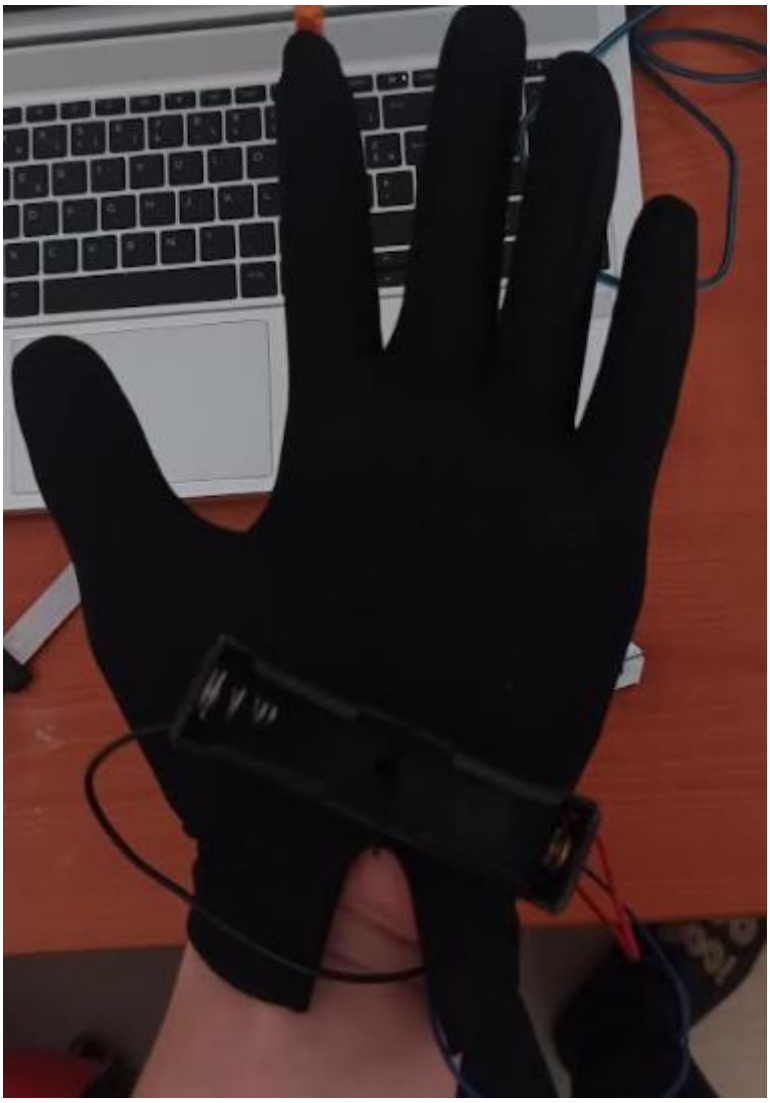
J'ai également un peu aidé Matis à faire les soudures sur la carte puis j'ai retesté le bon fonctionnement avec les nouvelles cartes.

Une fois les cartes finis, j'ai testé leur fonctionnement, la carte réceptrice fonctionne très bien, mais il y a eu des problèmes avec la carte émettrice car les PIN de l'accéléromètre étaient mélangés donc il ne fonctionnait pas. De plus, le GPIO du bouton poussoir a été relié au PIN du rst et j'ai mis 2 heures à comprendre pourquoi le bouton poussoir ne fonctionnait plus car je ne m'attendais pas du tout à ce que ce soit ça l'erreur.

Conception du gant pour contrôler le bras robotique :

J’ai également fait le gant qui permet de contrôler le bras robotique en cousant la carte, la batterie, et le capteur flex au gant, ce qui m’a pris un certain temps pour avoir un bon résultat :





Codes finaux :

Les codes finaux sont :

Code de la carte émettrice :

```
#include <Wire.h>

#include <WiFi.h>

#include <esp_now.h>


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

#define FLEX_SENSOR_PIN 39 // GPIO pour le capteur flex

#define BUTTON_PIN 18      // GPIO pour le bouton poussoir

#define FILTER_SAMPLES 30  // Nombre d'échantillons pour le filtrage
```

```
uint8_t broadcastAddress[] = {0x30, 0xAE, 0xA4, 0x6F, 0x06, 0x84};

// Historique pour filtrer les données des capteurs
float flexHistory[FILTER_SAMPLES];
int filterIndex = 0;

// Variables pour le debounce du bouton
bool buttonPressed = false;
unsigned long lastDebounceTime = 0;
const unsigned long debounceDelay = 50;

// Structure pour envoyer les données des capteurs
struct SensorData {
  float accelX;
  float accelY;
  int flex;
  bool buttonState;
};
SensorData dataToSend;

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);
  Wire.begin(); // Initialisation I²C

  // Configuration du bouton (simple INPUT au lieu de INPUT_PULLUP)
```

```
pinMode(BUTTON_PIN, INPUT);

// Vérifier la connexion avec le MPU-6050
Wire.beginTransmission(MPU6050_ADDR);
if (Wire.endTransmission() != 0) {
  Serial.println("MPU-6050 non détecté !");
  while (1);
}
Serial.println("MPU-6050 détecté !");

// Activer le MPU-6050
Wire.beginTransmission(MPU6050_ADDR);
Wire.write(0x6B);
Wire.write(0x00);
Wire.endTransmission();

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

esp_now_peer_info_t peerInfo;
memcpy(peerInfo.peer_addr, broadcastAddress, 6);
peerInfo.channel = 0;
peerInfo.encrypt = false;

peerInfo.ifidx = WIFI_IF_STA;
if (esp_now_add_peer(&peerInfo) != ESP_OK) {
```

```
    Serial.println("Échec de l'ajout du pair");
    return;
  }

  analogReadResolution(12);
}

void readAccelData(int16_t *ax, int16_t *ay) {
  Wire.beginTransmission(MPU6050_ADDR);
  Wire.write(0x3B);
  Wire.endTransmission(false);
  Wire.requestFrom(MPU6050_ADDR, 4, true);

  if (Wire.available() == 4) {
    *ax = Wire.read() << 8 | Wire.read();
    *ay = Wire.read() << 8 | Wire.read();
  } else {
    Serial.println("Erreur de lecture MPU-6050 !");
  }
}

float getFilteredValue(float *history, int pin) {
  float current = analogRead(pin);
  history[filterIndex] = current;
  float sum = 0;
  for (int i = 0; i < FILTER_SAMPLES; i++) {
    sum += history[i];
  }
  filterIndex = (filterIndex + 1) % FILTER_SAMPLES;
  return sum / FILTER_SAMPLES;
}
```

```
void loop() {
  int16_t ax, ay;
  readAccelData(&ax, &ay);

  // Lecture de l'état du bouton avec gestion du debounce
  int buttonReading = digitalRead(BUTTON_PIN);

  // Si l'état du bouton change, réinitialiser le timer de debounce
  if ((millis() - lastDebounceTime) > debounceDelay) {
    // Mettre à jour l'état du bouton
    dataToSend.buttonState = (buttonReading == HIGH);
    lastDebounceTime = millis();
  }

  dataToSend.accelX = ax;
  dataToSend.accelY = ay;
  dataToSend.flex = getFilteredValue(flexHistory, FLEX_SENSOR_PIN);

  Serial.print("Bouton: ");
  Serial.println(dataToSend.buttonState ? "Appuyé" : "Relâché");
  Serial.println(dataToSend.flex);
  esp_now_send(broadcastAddress, (uint8_t *)&dataToSend, sizeof(dataToSend));

  delay(10);
}
```

Code de la carte réceptrice :

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

```
#include <esp_now.h>
#include <WiFi.h>
#include <Adafruit_INA219.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

Adafruit_INA219 ina219;       //Création d'une instance de l'INA

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 = 3100;     // Définition de la valeur minimale du capteur flex
const int FLEX_MAX = 2800;     // 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 = 60; // Définition de l'angle maximum pour les servos
const int SERVO_MIN_ANGLE_FLEX = 2; // Définition de l'angle maximum pour le servo du capteur flex
const int ANGLE_CHANGE_THRESHOLD = 8; // Seuil à dépasser pour changer l'angle du servo dans certains cas
const int CourantMax=-100; // courant maximum à ne pas dépasser pour ne pas faire forcer les servos moteurs
```

```
// Variables pour le mode automatique
unsigned long lastMoveTime = 0;
int sequenceStep = 0;
const int delayBetweenMovements = 10; // en millisecondes
const int pauseBetweenPositions = 0; // pause entre positions en millisecondes

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

// Variables pour la gestion du bouton
bool buttonPressed = false;    // Pour détecter si le bouton a été pressé
bool prevButtonState = false;  // État précédent du bouton
unsigned long lastDebounceTime = 0;
const unsigned long debounceDelay = 50;  // Délai anti-rebond en ms

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
  bool buttonState;        // État du bouton: true = appuyé, false = relâché
};

SensorData receivedData;      // Déclaration de l'objet `receivedData` pour stocker les données reçues
bool currentMode = true;      // Mode par défaut: manuel (true = manuel, false = automatique)

void adjustServoAngleIfNeeded(Servo &servo, float &currentAngle, int targetAngle) {
 float current_mA = ina219.getCurrent_mA();
```

```
  if(currentAngle < targetAngle) {
   servo.write(targetAngle);
   currentAngle = targetAngle;
  }
  else if (currentAngle > targetAngle) {
    // Fermeture lente : angle diminue doucement
    for (int pos = currentAngle; pos >= targetAngle; pos-=0.25) {
      current_mA = ina219.getCurrent_mA();

      if (current_mA < CourantMax) {
        Serial.println("La pince force");
        break;
      }

      servo.write(pos);
      delay(75);  // ralentir la fermeture
      currentAngle = pos;
    }
  }
}

// Fonction pour déplacer un servo progressivement
void moveServoSmooth(Servo &servo, int startPos, float endPos) {
  if (startPos < endPos) {
    for (int pos = startPos; pos <= endPos; pos++) {
      servo.write(pos);
      delay(delayBetweenMovements);
    }
  } else {
    for (int pos = startPos; pos >= endPos; pos--) {
      servo.write(pos);
```

```
        delay(delayBetweenMovements);
      }
    }
}

void moveServoSmoothManual(
    Servo &servo0, float &current0, int target0,  //  sg90
    Servo &servo1, float &current1, int target1,  // sg902
    Servo &servo2, float &current2, int target2,  //  sg903
    Servo &servo3, float &current3, int target3   // sg904
) {
    int maxSteps = max(max(abs(current0 - target0), abs(current1 - target1)),
                       max(abs(current2 - target2), abs(current3 - target3)));
    if (maxSteps == 0) return;

    float step0 = (target0 - current0) / maxSteps;
    float step1 = (target1 - current1) / maxSteps;
    float step2 = (target2 - current2) / maxSteps;
    float step3 = (target3 - current3) / maxSteps;

    float pos0 = current0;
    float pos1 = current1;
    float pos2 = current2;
    float pos3 = current3;

    for (int i = 0; i <= maxSteps; i++) {
      if(current0-pos0<ANGLE_CHANGE_THRESHOLD){
        servo0.write(round(pos0));
      }
      servo1.write(round(pos1));
      servo2.write(round(pos2));
```

```
    servo3.write(round(pos3));



    pos0 += step0;
    pos1 += step1;
    pos2 += step2;
    pos3 += step3;

    delay(10);
  }

  current0 = target0;
  current1 = target1;
  current2 = target2;
  current3 = target3;
}

// Fonction pour déplacer tous les servomoteurs avec un servo spécifique en premier
void moveToPosition(float pos1, int pos2, int pos3, int pos4, int pos5, int firstServo) {
  // Définition des constantes pour identifier les servos
  const int BASE = 1;    // sg90
  const int PREMIER = 2;  // sg902
  const int DEUXIEME = 3;   // sg903
  const int TROISIEME = 4; // sg904
  const int PINCE = 5;   // sg905

  // Sauvegarde des positions actuelles
  int currentPos1 = sg90.read();
  int currentPos2 = sg902.read();
  int currentPos3 = sg903.read();
  int currentPos4 = sg904.read();
```

```
  int currentPos5 = sg905.read();

  // Déplacer d'abord le servo spécifié
  switch(firstServo) {
    case BASE:
      moveServoSmooth(sg90, currentPos1, pos1);
      break;
    case PREMIER:
      moveServoSmooth(sg902, currentPos2, pos2);
      break;
    case DEUXIEME:
      moveServoSmooth(sg903, currentPos3, pos3);
      break;
    case TROISIEME:
      moveServoSmooth(sg904, currentPos4, pos4);
      break;
    case PINCE:
      moveServoSmooth(sg905, currentPos5, pos5);
      break;
  }

  // Déplacer ensuite les autres servos
  if (firstServo != BASE) moveServoSmooth(sg90, currentPos1, pos1);
  if (firstServo != DEUXIEME) moveServoSmooth(sg903, currentPos3, pos3);
  if (firstServo != TROISIEME) moveServoSmooth(sg904, currentPos4, pos4);
  if (firstServo != PINCE) moveServoSmooth(sg905, currentPos5, pos5);
  if (firstServo != PREMIER) moveServoSmooth(sg902, currentPos2, pos2);

  // Pause pour permettre l'exécution de l'action
  delay(pauseBetweenPositions);
}
```

```
// Version simplifiée qui utilise BASE comme servo par défaut à bouger en premier
void moveToPosition(float pos1, int pos2, int pos3, int pos4, int pos5) {
  const int BASE = 1;
  moveToPosition(pos1, pos2, pos3, pos4, pos5, BASE);
}

// Fonction à appeler lorsqu'on reçoit des données
void OnDataRecv(const esp_now_recv_info_t *recv_info, const uint8_t *incomingData, int len) {
  if (len == sizeof(receivedData)) {
    memcpy(&receivedData, incomingData, sizeof(receivedData)); // Copie les données reçues

    // Gestion du bouton avec anti-rebond
    if (receivedData.buttonState != prevButtonState) {
      lastDebounceTime = millis();
    }

    // Si l'état du bouton est stable pendant assez longtemps
    if ((millis() - lastDebounceTime) > debounceDelay) {
      // Si le bouton est appuyé et n'était pas appuyé auparavant
      if (receivedData.buttonState && !buttonPressed) {
        buttonPressed = true;

        // Changer de mode
        currentMode = !currentMode;
        Serial.print("Mode changé: ");
        Serial.println(currentMode ? "Manuel" : "Automatique");

        // Réinitialiser la séquence automatique si on entre en mode automatique
        if (!currentMode) {
          sequenceStep = 0;
```

```
            lastMoveTime = millis();
          }
        }
        // Si le bouton est relâché
        else if (!receivedData.buttonState) {
          buttonPressed = false;
        }
      }

      prevButtonState = receivedData.buttonState;

      // Si on est en mode manuel, traiter les données des capteurs
      if (currentMode) {
        modeManuel();
      }
      // Le mode automatique est géré dans la boucle principale
    }
}

// Fonction pour le mode manuel (contrôlé par les capteurs)
void modeManuel() {
  int servoAngle = map(receivedData.flex, FLEX_MIN, FLEX_MAX, SERVO_MAX_ANGLE,
SERVO_MIN_ANGLE_FLEX);
  servoAngle = constrain(servoAngle, SERVO_MIN_ANGLE_FLEX, SERVO_MAX_ANGLE);
  adjustServoAngleIfNeeded(sg905, currentAngleSg905, servoAngle);
  currentAngleSg905 = servoAngle;

  int mappedAngleSg90 = map(receivedData.accelX, 12000, -12000, 0, 90);
  mappedAngleSg90 = constrain(mappedAngleSg90, SERVO_MIN_ANGLE,
SERVO_MAX_ANGLE);
  currentAngleSg90 = mappedAngleSg90;
```

```
int angle1, angle2, angle3;


if (receivedData.accelY < -10000) {
   angle1 = 0; angle2 = 0; angle3 = 0;
} else if (receivedData.accelY < -5400) {
   angle1 = map(receivedData.accelY, -10000, -5400, 0, 50);
   angle2 = 0;
   angle3 = map(receivedData.accelY, -10000, -5400, 0, 35);
} else if (receivedData.accelY < -4250) {
   angle1 = map(receivedData.accelY, -5400, -4250, 50, 35);
   angle2 = map(receivedData.accelY, -5400, -4250, 0, 15);
   angle3 = map(receivedData.accelY, -5400, -4250, 35, 45);
} else if (receivedData.accelY < -2000) {
   angle1 = map(receivedData.accelY, -4250, -2000, 35, 30);
   angle2 = map(receivedData.accelY, -4250, -2000, 15, 20);
   angle3 = 45;
} else if (receivedData.accelY < 2000) {
   angle1 = map(receivedData.accelY, -2000, 2000, 30, 25);
   angle2 = map(receivedData.accelY, -2000, 2000, 20, 35);
   angle3 = 45;
} else if (receivedData.accelY < 4000) {
   angle1 = map(receivedData.accelY, -800, 4000, 25, 20);
   angle2 = map(receivedData.accelY, -800, 4000, 35, 40);
   angle3 = 45;
} else if (receivedData.accelY < 6000) {
   angle1 = map(receivedData.accelY, 1500, 6000, 20, 15);
   angle2 = map(receivedData.accelY, 1500, 6000, 40, 50);
   angle3 = 45;
} else if (receivedData.accelY < 8000) {
   angle1 = map(receivedData.accelY, 3800, 8000, 15, 10);
   angle2 = map(receivedData.accelY, 3800, 8000, 50, 60);
```

```
      angle3 = 45;
    } else if (receivedData.accelY < 9000) {
      angle1 = map(receivedData.accelY, 6100, 9000, 10, 5);
      angle2 = map(receivedData.accelY, 6100, 9000, 60, 65);
      angle3 = 45;
    } else if (receivedData.accelY < 11000) {
      angle1 = map(receivedData.accelY, 8400, 11000, 5, 0);
      angle2 = map(receivedData.accelY, 8400, 11000, 65, 70);
      angle3 = 45;
    } else if (receivedData.accelY < 13000) {
      angle1 = 0;
      angle2 = map(receivedData.accelY, 10700, 13000, 70, 80);
      angle3 = map(receivedData.accelY, 10700, 13000, 45, 50);
    } else {
      angle1 = 0;
      angle2 = 75;
      angle3 = 65;
    }

    moveServoSmoothManual(
       sg90,  currentAngleSg90,  mappedAngleSg90,
       sg902,  currentAngleSg902,  angle1,
       sg903,  currentAngleSg903,  angle2,
       sg904,  currentAngleSg904,  angle3
    );

    currentAngleSg902 = angle1;
    currentAngleSg903 = angle2;
    currentAngleSg904 = angle3;
}
```

```
// Fonction pour exécuter la séquence automatique
void modeAutomatique() {
  const int PREMIER = 2;  // Valeur pour premier servo dans moveToPosition

  switch (sequenceStep) {
    case 0: // Position initiale
      moveToPosition(45, 0, 0, 0, 60, PREMIER);
      sequenceStep++;
      break;

    case 1: // Position 1
      Serial.println("Déplacement vers Position 1");
      moveToPosition(61, 14, 43, 68, 60);
      sequenceStep++;
      break;

    case 2: // Fermeture de la pince
      delay(750);
      Serial.println("Fermeture de la pince");
      moveServoSmooth(sg905, 60, 3);
      delay(pauseBetweenPositions);
      sequenceStep++;
      break;

    case 3: // Position 2
      delay(100);
      Serial.println("Déplacement vers Position 2");
      moveToPosition(61, 5, 42, 66, 3, PREMIER);
      sequenceStep++;
      break;
```

```
      case 4: // Position 3
        Serial.println("Déplacement vers Position 3");
        moveToPosition(58, 10, 30, 40, 3, PREMIER);
        sequenceStep++;
        break;

      case 5: // Position 4
        Serial.println("Déplacement vers Position 4");
        moveToPosition(56, 40, 11, 18, 3);
        sequenceStep++;
        break;

      case 6: // Ouverture de la pince
        delay(500);
        Serial.println("Ouverture de la pince");
        moveServoSmooth(sg905, 3, 60);
        delay(pauseBetweenPositions);
        sequenceStep++;
        break;

      case 7: // Pause avant de recommencer
        Serial.println("Redémarrage de la séquence");
        sequenceStep = 0; // Revenir au début de la séquence
        break;
    }
}

void setup() {
  Serial.begin(115200);

  // Initialisation du capteur INA219
```

```
  if (!ina219.begin()) {
    Serial.println("Erreur d'initialisation du capteur INA219");
    while (1); // Arrêt du programme si le capteur n'est pas trouvé
  }

  WiFi.mode(WIFI_STA); // Configure l'ESP32 en mode WiFi

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

  esp_now_register_recv_cb(OnDataRecv); // Enregistre la fonction qui sera appelée à chaque
réception de données

  sg90.attach(PIN_SERVO);
  sg902.attach(PIN_SERVO2);
  sg903.attach(PIN_SERVO3);
  sg904.attach(PIN_SERVO4);
  sg905.attach(SERVO_PIN_5);

  sg90.write(0);
  sg902.write(0);
  sg903.write(0);
  sg904.write(0);
  sg905.write(60); // Pince ouverte à l'initialisation

  // Définir le mode par défaut
  currentMode = true; // Mode manuel par défaut
}
```

```
void loop() {
  // Si en mode automatique, exécuter la séquence
  if (!currentMode) {
    modeAutomatique();
  }
  Serial.println(receivedData.flex);
  delay(10); // Petit délai pour éviter de surcharger le processeur
}
```

Conclusion :

Dans ce projet, j'ai pu choisir le modèle pour le bras robotique, assembler le bras robotique, coder tout son fonctionnement, et faire le câblage de tous les composants nécessaires au fonctionnement. J'ai donc pu apprendre beaucoup de choses par moi-même comme la communication entre deux cartes. Ce fut un projet très intéressant, et si je devais le refaire, la seule chose que je ferais différemment serait de choisir un autre modèle 3d pour le bras robotique car celui-ci ne permet pas d'être très précis car il n'est pas très stable et le système pour fermer la pince n'est pas très bien et a souvent pour effet de démonter la pince en la sortant de son axe quand on sert trop fort quelque chose.

Finalement, le bras robotique comporte deux cartes qui communiquent entre elles par wifi. J'ai fait deux modes pour contrôler les mouvements du bras robotiques : un mode automatique où le bras robotique va effectuer les mêmes mouvements en boucle comme dans les chaînes d'usinage, et un mode manuel. Dans le mode manuel, les données de l'accéléromètre et du capteur flex qui sont connectés à la carte émettrice sont envoyées par wifi vers la carte réceptrice qui va interpréter ces données afin de calculer les positions des différents servo-moteurs de façon à ce que le bras robotique reproduise les mouvements que l'on fait avec notre main (car les capteurs sont fixés sur un gant). Le passage d'un mode à l'autre sera activé par un bouton situé sur la carte émettrice et dont l'état est transmis à la carte réceptrice permettant de changer de mode à chaque clique.