Date : 07/04/2025

# Compte-rendu séance 14 Téo Baillot d'Estivaux :

## Objectif de la séance :

L'objectif de cette séance est de quasiment finaliser le projet pour pouvoir en faire une démonstration. Comme il reste beaucoup de choses à faire j'ai pas mal avancé chez moi, d'où la longueur de ce rapport. Les tâches restantes étaient d'ajouter le fonctionnement automatique de la pince. L'idée est d'avoir un bouton sur la carte émettrice qui permettrait de passer d'un mode. Le premier mode serait le mode manuel qui permettrait donc de contrôler les mouvements du bras robotique à distance à l'aide des données envoyées par l'accéléromètre et le capteur flex, et le deuxième mode permettrait d'effectuer des mouvements en boucle comme dans les usines. Le bouton dont les données seraient également envoyées par wifi permettra donc de passer d'un mode à l'autre dès qu'on clique dessus.

## Code des mouvements automatiques :

Tout d'abord j'ai commencé par établir un code simple permettant de mettre en place les mouvements du bras robotique sans communiquer entre les deux cartes où utiliser les boutons. J'ai quand même eu des difficultés avec ce code car j'avais pour objectif de faire une série de mouvements précis mais en me suis rendu compte que les servos moteurs ont une précision d'au maximum 1 degré ce qui n'est pas suffisant pour pouvoir faire les mouvements que je voulais faire car je n'arrivais pas à replacer le cube dans la rampe que j'ai faite lors de la précédente séance.

Je suis tout de même arrivé à faire ce code qui s'approche du résultat attendu et que je modifierai une fois la nouvelle version de la rampe mise en place :

```
#include <ESP32Servo.h>

#include <esp_now.h>

#include <WiFi.h>


// Définition des broches pour les servomoteurs

#define PIN_SERVO 16   // Servo SG90

#define PIN_SERVO2 13  // Servo SG902

#define PIN_SERVO3 14  // Servo SG903

#define PIN_SERVO4 15  // Servo SG904

#define SERVO_PIN_5 4  // Servo SG905 (pince)


// Création des objets servo
```

Confidential C

```
Servo sg90;   // Premier servo (base)
Servo sg902;  // Deuxième servo (épaule)
Servo sg903;  // Troisième servo (coude)
Servo sg904;  // Quatrième servo (poignet)
Servo sg905;  // Cinquième servo (pince)

// Délai entre les mouvements pour une animation plus fluide
const int delayBetweenMovements = 15; // en millisecondes
const int pauseBetweenPositions = 100; // pause entre positions en millisecondes

// 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

// Fonction pour déplacer un servo progressivement
void moveServoSmooth(Servo &servo, int startPos, int 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);
  }
 }
}
```

Confidential C

```
// Fonction pour déplacer tous les servomoteurs avec un servo spécifique en premier
void moveToPosition(int pos1, int pos2, int pos3, int pos4, int pos5, int firstServo) {
 // 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);
```

Confidential C

```
  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(int pos1, int pos2, int pos3, int pos4, int pos5) {
  moveToPosition(pos1, pos2, pos3, pos4, pos5, BASE);
}

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

  // Attachement des servos à leurs broches respectives
  sg90.attach(PIN_SERVO);
  sg902.attach(PIN_SERVO2);
  sg903.attach(PIN_SERVO3);
  sg904.attach(PIN_SERVO4);
  sg905.attach(SERVO_PIN_5);

  // Position initiale (position neutre)
  sg90.write(45);
  sg902.write(0);
  sg903.write(0);
  sg904.write(0);
  sg905.write(60); // Pince ouverte

  delay(2000); // Pause initiale pour permettre aux servos de se positionner
```

Confidential C

```
}

void loop() {
  // Position initiale
  moveToPosition(44, 0, 0, 0, 60, premier);

  // Position 1 : Base (servo 1) en premier
  Serial.println("Déplacement vers Position 1");
  moveToPosition(59, 15, 42, 66, 60);

  // Fermeture de la pince
  Serial.println("Fermeture de la pince");
  moveServoSmooth(sg905, 60, 0);
  delay(pauseBetweenPositions);

  // Position 2 : Épaule (servo 2) en premier
  Serial.println("Déplacement vers Position 2");
  moveToPosition(59, 10, 42, 66, 0, premier);

  // Position 3 : Ordre par défaut (Base en premier)
  Serial.println("Déplacement vers Position 3");
  moveToPosition(59, 15, 30, 40, 0);

  // Position 4 : Coude (servo 3) en premier
  Serial.println("Déplacement vers Position 4");
  moveToPosition(59, 35, 15, 25, 0);

  // Ouverture de la pince
  Serial.println("Ouverture de la pince");
  moveServoSmooth(sg905, 0, 60);
  delay(pauseBetweenPositions);
```

Confidential C

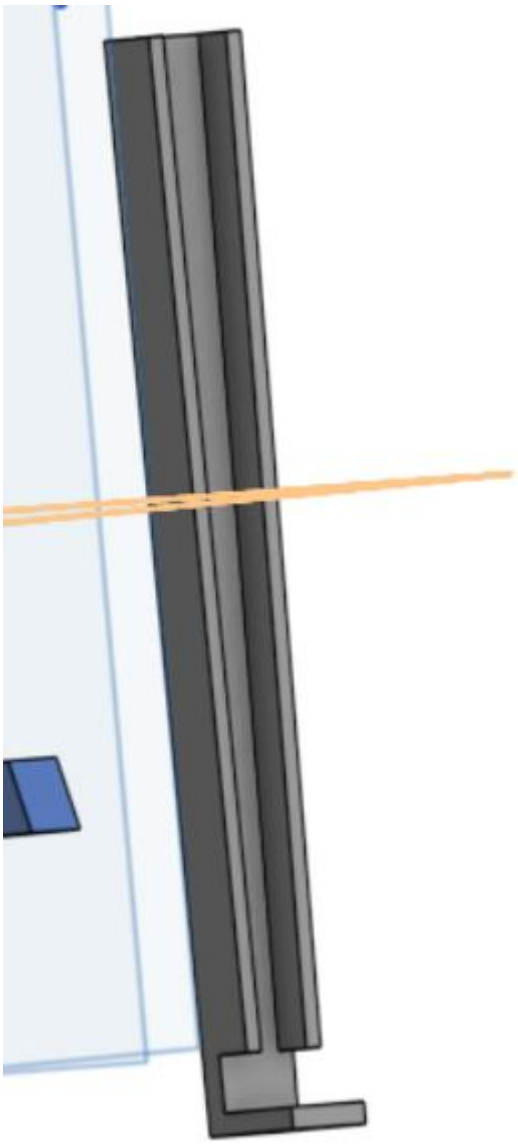
```
  // Pause avant de recommencer la séquence
  delay(1000);

  Serial.println("Redémarrage de la séquence");
}
```

## Amélioration de la rampe :

Pour faire face aux problèmes d'imprécision des servo moteurs, j'ai donc modélisé en 3d une nouvelle rampe qui me permettra de poser plus simplement le cube dans le rampe car elle est plus large est requiert donc moins de précision :

Confidential C



Une fois l'impression faite, j'ajusterais donc les mouvements automatiques du bras robotique.

## Impressions 3d :

Lors de test j'ai encore cassé la pince du bras robotique. J'ai donc pris le temps d'aller faire les impressions ainsi que celles de la nouvelle rampe après mon travail pendant la semaine en entreprise afin de pouvoir avancer sur le projet chez moi.

## Code final :

Une fois que j'ai fini de réaliser les mouvements automatiques du bras robotiques, j'ai modifié le code de la carte émettrice pour pouvoir envoyer les données du bouton poussoir. J'ai également modifié le code de la carte réceptrice pour implémenter le fonctionnement automatique est également changer le code pour avoir une fonction pour le fonctionnement manuel. J'ai aussi

Confidential C

géré les données reçus par le bouton poussoir afin de pouvoir passer d'un mode à l'autre comme désiré. Finalement, voici les codes finaux du projet :

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 25      // 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");
```

Confidential C

```
    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;

  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 !");
  }
}
```

Confidential C

```
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");
```

Confidential C

```
  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>

#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 = 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 = 65; // Définition de l'angle maximum pour les servos
```

Confidential C

```
const int SERVO_MIN_ANGLE_FLEX = 0; // 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


// 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
```

Confidential C

```
bool currentMode = true;      // Mode par défaut: manuel (true = manuel, false = automatique)

// Fonction pour déplacer un servo progressivement
void moveServoSmooth(Servo &servo, int startPos, int 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);
    }
  }
}

// Fonction pour déplacer tous les servomoteurs avec un servo spécifique en premier
void moveToPosition(int 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();
```

Confidential C

```
  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);
}
```

Confidential C

```
// Version simplifiée qui utilise BASE comme servo par défaut à bouger en premier
void moveToPosition(int 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;
```

Confidential C

```
            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() {
  // 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);
  servoAngle = constrain(servoAngle, SERVO_MIN_ANGLE_FLEX, SERVO_MAX_ANGLE);
  if (abs(servoAngle - currentAngleSg905) >= ANGLE_CHANGE_THRESHOLD) {
    sg905.write(servoAngle);
    currentAngleSg905 = servoAngle;
  }

  // Calcul de l'angle pour l'accéléromètre X
```

Confidential C

```
  int mappedAngleSg90 = map(receivedData.accelX, 12000, -12000, 0, 90);
  mappedAngleSg90 = constrain(mappedAngleSg90, SERVO_MIN_ANGLE,
SERVO_MAX_ANGLE);
  if (abs(mappedAngleSg90 - currentAngleSg90) >= ANGLE_CHANGE_THRESHOLD) {
    sg90.write(mappedAngleSg90);
    currentAngleSg90 = mappedAngleSg90;
  }

  // Autres mouvements en fonction de Y
  if (receivedData.accelY < -10000) {
    sg902.write(0);
    sg903.write(0);
    sg904.write(0);
    currentAngleSg902 = 15;
    currentAngleSg903 = 43;
    currentAngleSg904 = 65;
  } else if (receivedData.accelY < -7700) {
    int angle1 = map(receivedData.accelY, -10000, -7700, 0, 50);
    int angle2 = map(receivedData.accelY, -10000, -7700, 0, 35);
    sg902.write(angle1);
    sg903.write(0);
    sg904.write(angle2);
    currentAngleSg902 = angle1;
    currentAngleSg903 = 0;
    currentAngleSg904 = angle2;
  } else if (receivedData.accelY < -5400) {
    int angle1 = map(receivedData.accelY, -7700, -5400, 50, 35);
    int angle2 = map(receivedData.accelY, -7700, -5400, 0, 15);
    int angle3 = map(receivedData.accelY, -7700, -5400, 35, 45);
    sg902.write(angle1);
    sg903.write(angle2);
```

```
    sg904.write(angle3);
    currentAngleSg902 = angle1;
    currentAngleSg903 = angle2;
    currentAngleSg904 = angle3;
  } else if (receivedData.accelY < -3100) {
    int angle1 = map(receivedData.accelY, -5400, -3100, 35, 30);
    int angle2 = map(receivedData.accelY, -5400, -3100, 15, 20);
    int angle3 = map(receivedData.accelY, -5400, -3100, 45, 45);
    sg902.write(angle1);
    sg903.write(angle2);
    sg904.write(angle3);
    currentAngleSg902 = angle1;
    currentAngleSg903 = angle2;
    currentAngleSg904 = angle3;
  } else if (receivedData.accelY < -800) {
    int angle1 = map(receivedData.accelY, -3100, -800, 30, 25);
    int angle2 = map(receivedData.accelY, -3100, -800, 20, 35);
    int angle3 = map(receivedData.accelY, -3100, -800, 45, 45);
    sg902.write(angle1);
    sg903.write(angle2);
    sg904.write(angle3);
    currentAngleSg902 = angle1;
    currentAngleSg903 = angle2;
    currentAngleSg904 = angle3;
  } else if (receivedData.accelY < 1500) {
    int angle1 = map(receivedData.accelY, -800, 1500, 25, 20);
    int angle2 = map(receivedData.accelY, -800, 1500, 35, 40);
    int angle3 = map(receivedData.accelY, -800, 1500, 45, 45);
    sg902.write(angle1);
    sg903.write(angle2);
    sg904.write(angle3);
```

Confidential C

```
    currentAngleSg902 = angle1;
    currentAngleSg903 = angle2;
    currentAngleSg904 = angle3;
  } else if (receivedData.accelY < 3800) {
    int angle1 = map(receivedData.accelY, 1500, 3800, 20, 15);
    int angle2 = map(receivedData.accelY, 1500, 3800, 40, 50);
    int angle3 = map(receivedData.accelY, 1500, 3800, 45, 45);
    sg902.write(angle1);
    sg903.write(angle2);
    sg904.write(angle3);
    currentAngleSg902 = angle1;
    currentAngleSg903 = angle2;
    currentAngleSg904 = angle3;
  } else if (receivedData.accelY < 6100) {
    int angle1 = map(receivedData.accelY, 3800, 6100, 15, 10);
    int angle2 = map(receivedData.accelY, 3800, 6100, 50, 60);
    int angle3 = map(receivedData.accelY, 3800, 6100, 45, 45);
    sg902.write(angle1);
    sg903.write(angle2);
    sg904.write(angle3);
    currentAngleSg902 = angle1;
    currentAngleSg903 = angle2;
    currentAngleSg904 = angle3;
  } else if (receivedData.accelY < 8400) {
    int angle1 = map(receivedData.accelY, 6100, 8400, 10, 5);
    int angle2 = map(receivedData.accelY, 6100, 8400, 60, 65);
    int angle3 = map(receivedData.accelY, 6100, 8400, 45, 45);
    sg902.write(angle1);
    sg903.write(angle2);
    sg904.write(angle3);
    currentAngleSg902 = angle1;
```

Confidential C

```
    currentAngleSg903 = angle2;
    currentAngleSg904 = angle3;
  } else if (receivedData.accelY < 10700) {
    int angle1 = map(receivedData.accelY, 8400, 10700, 5, 0);
    int angle2 = map(receivedData.accelY, 8400, 10700, 65, 70);
    int angle3 = map(receivedData.accelY, 8400, 10700, 45, 45);
    sg902.write(angle1);
    sg903.write(angle2);
    sg904.write(angle3);
    currentAngleSg902 = angle1;
    currentAngleSg903 = angle2;
    currentAngleSg904 = angle3;
  } else if (receivedData.accelY < 13000) {
    int angle1 = map(receivedData.accelY, 10700, 13000, 0, 0);
    int angle2 = map(receivedData.accelY, 10700, 13000, 70, 80);
    int angle3 = map(receivedData.accelY, 10700, 13000, 45, 50);
    sg902.write(angle1);
    sg903.write(angle2);
    sg904.write(angle3);
    currentAngleSg902 = angle1;
    currentAngleSg903 = angle2;
    currentAngleSg904 = angle3;
  } else {
    int angle1 = 0;
    int angle2 = 75;
    int angle3 = 65;
    sg902.write(angle1);
    sg903.write(angle2);
    sg904.write(angle3);
    currentAngleSg902 = angle1;
    currentAngleSg903 = angle2;
```

Confidential C

```
      currentAngleSg904 = angle3;
    }
    Serial.println(receivedData.accelY);
}

// 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(44, 0, 0, 0, 65, PREMIER);
        sequenceStep++;
        break;

      case 1: // Position 1
        Serial.println("Déplacement vers Position 1");
        moveToPosition(59, 15, 44, 64, 65);
        sequenceStep++;
        break;

      case 2: // Fermeture de la pince
        delay(100);
        Serial.println("Fermeture de la pince");
        moveServoSmooth(sg905, 65, 5);
        delay(pauseBetweenPositions);
        sequenceStep++;
        break;

      case 3: // Position 2
        Serial.println("Déplacement vers Position 2");
```

Confidential C

```
      moveToPosition(59, 10, 42, 66, 5, PREMIER);
      sequenceStep++;
      break;

    case 4: // Position 3
      Serial.println("Déplacement vers Position 3");
      moveToPosition(59, 15, 30, 40, 5, PREMIER);
      sequenceStep++;
      break;

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

    case 6: // Ouverture de la pince
      delay(100);
      Serial.println("Ouverture de la pince");
      moveServoSmooth(sg905, 5, 65);
      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;
  }
}
```

Confidential C

```
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) {
    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
```

Confidential C

```
  if (!currentMode) {
    modeAutomatique();
  }
  Serial.println(receivedData.flex);
  // Le mode manuel est géré dans la fonction OnDataRecv
  delay(10); // Petit délai pour éviter de surcharger le processeur
}
```

## Tests :

J'ai également passé beaucoup de temps à faire des tests pour faire des ajustements dans le code et vérifier que tout fonctionne correctement notamment au niveau de la capacité de la pince à attraper des objets et au niveau de la précision des mouvements du bras robotique.

## Prototypage du gant :

J'ai également réalisé un prototype du gant en cousant l'accéléromètre sur un capteur flex, permettant de se rapprocher des manipulations qu'on fera lors de la démo.

Confidential C

## Objectif de la prochaine séance :

Pour moi, l'objectif de la prochaine séance sera 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 problème 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.

Confidential C