Date : 25/02/2025

Compte-rendu séance 8 Téo Baillot d'estivaux :

Objectifs de la séance :

Après la présentation de la dernière séance, l'objectif de cette séance est d'améliorer le fonctionnement de la pince. La première piste d'amélioration que je vais suivre dans cette séance est le remplacement de l'accéléromètre analogique par un accéléromètre numérique afin d'avoir des valeurs plus précises renvoyées par l'accéléromètre et donc permettre d'augmenter la précision des mouvements de notre bras robotique et de rendre le contrôle de ces mouvements plus simples.

Fonctionnement de l'accéléromètre numérique :

L'accéléromètre est équipé de microstructures MEMS (Micro Electro Mechanical Systems) qui se déforment sous l'effet de l'accélération. Ces structures contiennent des condensateurs dont la capacité varie en fonction du mouvement.

Un circuit interne convertit ces variations de capacité en valeurs numériques grâce à un convertisseur analogique-numérique (ADC).

L'accéléromètre applique souvent un filtrage et d'autres traitements (comme la compensation de la température ou la correction des biais) avant d'envoyer les données.

L'ESP32 peut récupérer les données via I^2C (plus simple, moins de fils) ou SPI (plus rapide).

En I^2C, l'ESP32 communique avec l'accéléromètre en utilisant deux fils : SDA (données) et SCL (horloge).

Dans mon cas, j'utilise un accéléromètre MPU-6050 qui ne peut malheureusement pas utiliser le SPI donc on se contentera d'une communication I^2C.

Développement du code de l'accéléromètre :

Pour développer le code de l'accéléromètre numérique, je vais m'appuyer sur la librairie Wire qui va me permettre de récupérer les données de l'accéléromètre numérique.

Lors du développement du code, j'ai rencontré quelques difficultés à remonter les données de l'accéléromètre. Parfois, l'accéléromètre me renvoyait que des 0 ou que des -1 en X, en Y, et en Z, j'ai donc vérifié à plusieurs reprises mon câblage et l'alimentation et au final le problème venait de mon code.

Finalement, j'ai réussi à faire un code qui me permet de récupérer les données de l'accéléromètre numérique :

#include <Wire.h> // Inclusion de la bibliothèque Wire pour la communication I^2C

```
#define MPU6050_ADDR 0x68  // Adresse I²C du MPU-6050 (adresse par défaut si AD0 est à 0)

void setup() {
  Serial.begin(115200);  // Initialisation de la communication série à 115200 bauds
  Wire.begin();  // Démarre la communication I²C avec l'ESP32 en tant que maître

  // Vérifier si le capteur MPU-6050 est présent sur le bus I²C
  Wire.beginTransmission(MPU6050_ADDR); // Commence une transmission vers le capteur
  if (Wire.endTransmission() != 0) {  // Vérifie si la transmission s'est bien terminée
    Serial.println("MPU-6050 non détecté !"); // Message d'erreur si le capteur n'est pas trouvé
    while (1); // Bloque l'exécution du programme si le capteur est absent
  }
  Serial.println("MPU-6050 détecté !"); // Confirmation que le capteur est bien détecté

  // Activation du capteur (il est en mode veille par défaut au démarrage)
  Wire.beginTransmission(MPU6050_ADDR); // Démarre une communication avec le capteur
  Wire.write(0x6B);  // Sélectionne le registre PWR_MGMT_1 (contrôle de l'alimentation)
  Wire.write(0x00);  // Écrit 0x00 dans ce registre pour activer le capteur
  Wire.endTransmission();  // Termine la communication avec le capteur
}

// Fonction pour lire les valeurs d'accélération sur les 3 axes (X, Y et Z)
void readAccelData(int16_t *ax, int16_t *ay, int16_t *az) {
  Wire.beginTransmission(MPU6050_ADDR); // Démarre une transmission I²C avec le capteur
  Wire.write(0x3B); // Adresse du premier registre contenant les données d'accélération (AX)
  Wire.endTransmission(false); // Demande à conserver le bus actif pour récupérer les données
  Wire.requestFrom(MPU6050_ADDR, 6, true); // Demande la lecture de 6 octets (2 octets par axe)

  if (Wire.available() == 6) { // Vérifie si les 6 octets attendus sont bien reçus
```

```
    *ax = Wire.read() << 8 | Wire.read(); // Lit les 2 octets de l'axe X et les assemble
    *ay = Wire.read() << 8 | Wire.read(); // Lit les 2 octets de l'axe Y et les assemble
    *az = Wire.read() << 8 | Wire.read(); // Lit les 2 octets de l'axe Z et les assemble
  } else {
    Serial.println("Erreur de lecture !"); // Message d'erreur si les données ne sont pas disponibles
  }
}

void loop() {
  int16_t ax, ay, az; // Variables pour stocker les valeurs des axes X, Y et Z
  readAccelData(&ax, &ay, &az); // Appel de la fonction pour récupérer les valeurs

  // Affichage des valeurs d'accélération sur le moniteur série
  Serial.print("Acc X: "); Serial.print(ax);
  Serial.print(" Y: "); Serial.print(ay);
  Serial.print(" Z: "); Serial.println(az);

  delay(10);
}
```

Modification du code complet :

Une fois la réalisation du code de l'accéléromètre effectué, mon objectif a été de l'implémenter dans l'ancien code de la carte émettrice pour remplacer le code qui servait avant à l'accéléromètre analogique.

Code de la carte émettrice :

```
#include <esp_now.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 pour la transmission ESP-NOW (adresse MAC de la carte réceptrice)

// 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 lorsque les données sont envoyées pour savoir si l'envoi 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");
}

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

  // Vérifier si le capteur répond
  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 !");

  // Réveiller le capteur (il démarre en mode veille)
  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 car non utilisé ici
  } else {
    Serial.println("Erreur de lecture du MPU-6050 !");
  }
}

// Fonction pour filtrer les valeurs du capteur flex 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 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'envoi 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

  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; // Stocke la valeur brute de l'accélération sur l'axe X
  dataToSend.accelY = ay; // Stocke la valeur brute de l'accélération sur l'axe Y

  // Lit la valeur du capteur flex
  dataToSend.flex = getFilteredValue(flexHistory, FLEX_SENSOR_PIN);
  Serial.println(dataToSend.accelX); // Affiche la valeur du capteur flex sur le moniteur série

  // 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 :

Le changement d'accéléromètre a eu pour effet un changement de la plage de valeurs envoyée à la carte réceptrice. Du côté de la carte réceptrice il a donc fallu associer les déplacements des servos moteurs aux nouvelles valeurs renvoyées par l'accéléromètre :

```
#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 < -3000) { // Si la valeur de Y est inférieure à 800
    int angle2 = map(receivedData.accelY, -10000, -3000, 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 < 4000) { // Si la valeur de Y est inférieure à 900
    int angle3 = map(receivedData.accelY, -3000, 4000, 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 < 13000) { // Si la valeur de Y est inférieure à 1050
    int angle4 = map(receivedData.accelY, 4000, 13000, 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
}
```

Test du fonctionnement global avec l'accéléromètre numérique :

J'ai ensuite fait des tests pour comparer les déplacements avec l'accéléromètre par rapport à ceux avec l'accéléromètre analogique et le changement d'accéléromètre a en effet permis de bien améliorer la précision et la maniabilité de la pince.

Cette séance m'a donc permis de bien améliorer le bras robotique.

Objectifs de la prochaine séance :

L'objectif de ma prochaine séance sera d'essayer de régler le problème de transmission de données qui se perdent lors de la transmission entre la carte émettrice et la carte réceptrice afin d'encore augmenter la précision et la maniabilité du bras robotique et d'éviter que le bras robotique se déplace par accoup.