Compte rendu séance 4 Téo Baillot d'Estivaux :

Objectifs de la séance :

Pour moi, le but de cette séance sera d'implémenter le code qui permet de contrôler l'écartement de la pince du bras robotique à l'aide du capteur flex dans le reste du code et de régler le problème des valeurs renvoyées par l'accéléromètre qui sont trop approximatives.

Installation de la pince du bras robotique :

J'ai essayé d'installer les nouvelles pièces que j'ai imprimé la séance dernière mais les dents se sont cassées donc il faudra les réimprimer… Si le problème continu de se produire, il faudra trouver une meilleure solution que celle proposée.

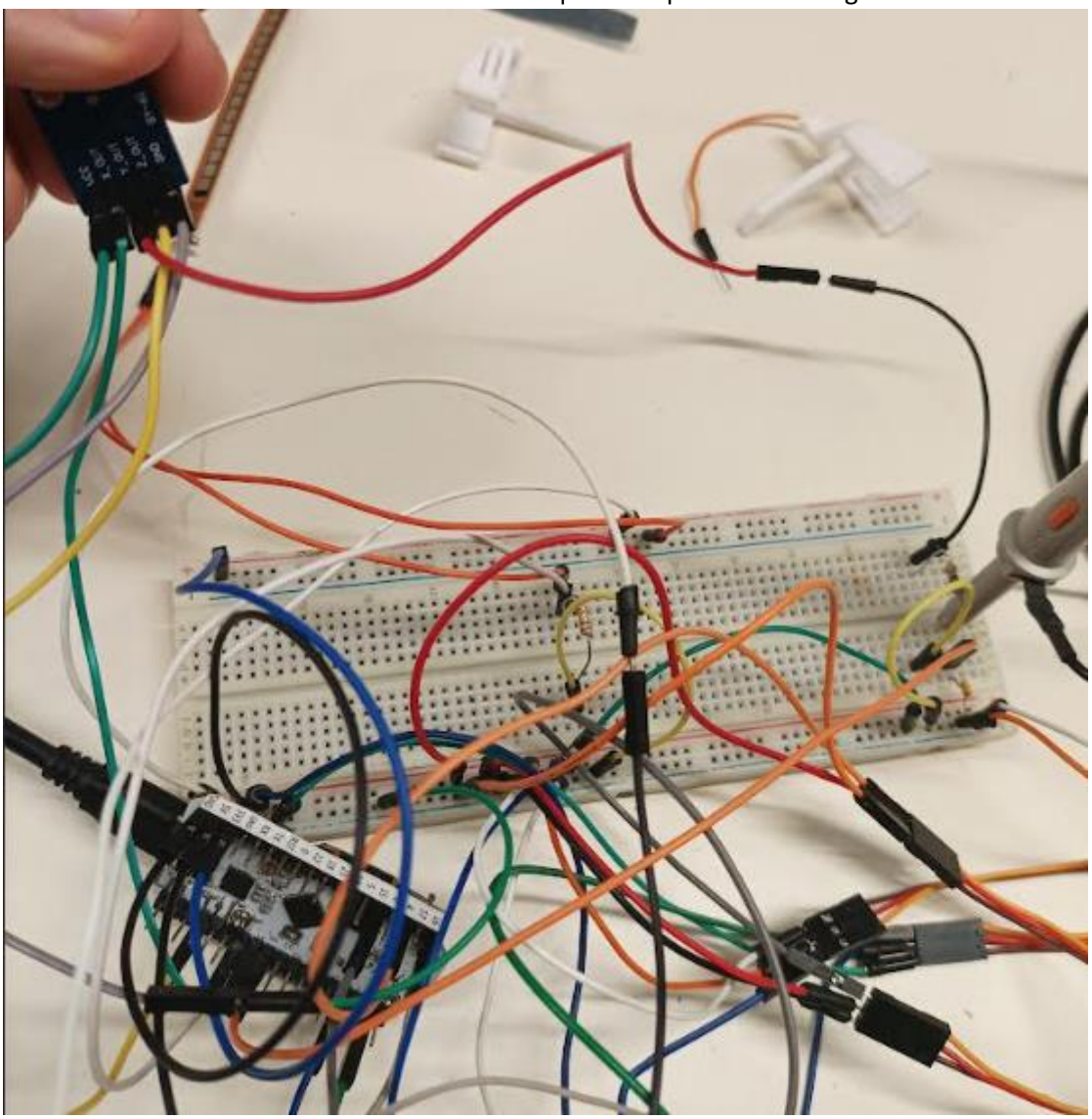
Implémentation du code avec le capteur flex :

Dans la suite de la séance, j'ai implémenté le code du capteur flex qui permet de contrôler l'écartement de la pince au reste du code et ai effectué différents tests pour vérifier le bon fonctionnement du code. Bien que j'ai utilisé les anciennes impressions 3d pour la pince car j'ai cassé les nouvelles, tout semble fonctionner correctement, mais il faudra refaire les tests de façon plus approfondis avec des modélisations 3d de meilleure qualité.

Mise en place du filtre RC pour diminuer le bruit de l'accéléromètre:

Lors de la dernière séance, on avait calculé la valeur du filtre RC que l'on allait installer. Après avoir implémenter le code du capteur flex, j'ai donc travaillé sur la réduction du bruit de l'accéléromètre afin qu'il renvoi des valeurs plus précises.

J’ai donc fait les branchements avec le filtre RC qui correspond au montage suivant :



Finalement, le filtre RC a diminué le bruit par 2 mais la précision des valeurs renvoyées par l’accéléromètre n’est toujours pas satisfaisante car elle ne permet toujours pas d’obtenir des valeurs assez précises pour contrôler correctement la pince.

Modifications du code pour récupérer des valeurs plus précises de l’accéléromètre :

Maintenant que j’ai fait le maximum possible avec les moyens qu’on a pour réduire le bruit de l’accéléromètre du côté hardware, la seule solution possible pour augmenter la précision des valeurs renvoyées par l’accéléromètre est de traiter les valeurs du côté software.

Le code utilise déjà une fonction qui permet de faire une moyenne de 7 échantillons et de ne travailler que sur la valeur renvoyé par ce filtre de moyenne, mais la précision n’étant toujours pas assez bonne, il faut trouver un moyen d’augmenter ce nombre d’échantillons tout en faisant en sorte de ne pas augmenter le délai pour calculer la moyenne des échantillons car sinon les mouvements de la pince se feraient en décalé par rapport aux mouvements de l’accéléromètre.

Finalement, après avoir bien réfléchi, pour encore améliorer la précision de l'accéléromètre, j'ai fait en sorte de changer le code de façon à ce que le filtre qui calcule la moyenne des échantillons utilise plus d'échantillons pour calculer la moyenne (30 au lieu de 7 avant) et que les données qu'on récupère de l'accéléromètre soient récupérés toutes 10ms au lieu de 50ms avant. Ceci permet donc d'avoir une valeur filtrée beaucoup plus précise ce qui a considérablement réduit l'influence du bruit de l'accéléromètre sur le contrôle de notre bras robotique.
Les données étant plus précises, la plage de valeur va maintenant de 750 à 1050 alors que la plage de valeur précédente allait de 1900 à 2200. Cependant, maintenant les données récupérées sont beaucoup plus précises et varient pour une même position de maximum 5 sur la plage de donnée contre 100 à 150 avant ce qui permet à la pince d'avoir beaucoup moins de mouvements aléatoires dus au bruit de l'accéléromètre.
La plage de donnée ayant changée, j'ai du modifier le code pour l'adapter à la nouvelle place de donnée.

Test du nouveau code :

En testant le code, je me suis aperçu qu'un servo moteur ne se déplaçait pas aux positions où il devait être et qu'il n'avait aucune résistance sur la position où il était ce qui veut dire qu'il n'essaye même pas de se repositionner. Après des tests et ne comprenant pas pourquoi il ne se déplace plus alors que les valeurs renvoyées par l'accéléromètre semblent bonnes, j'ai décidé de le remplacer.
J'ai du démonter toute la pince et la remonter et ça ne fonctionnait toujours pas mais je le sentais vibrer donc cette fois-ci j'étais sûr que le servo moteur marché et je me suis tout de suite dit que le problème devait venir de l'alimentation du montage car il commençait à avoir beaucoup de composant et que ça ne m'étonnerait pas que la carte elle-même n'arrive pas à tous les alimenter correctement.
J'ai donc utiliser un générateur de tension continu à 5V au lieu du 5V de la carte ce qui m'a permis d'alimenter correctement tout les servo moteurs et j'ai continué de garder l'alimentation de la carte pour fournir le 3.3V au capteur flex et à l'accéléromètre. Finalement, utiliser ce générateur de tension externe à la carte a confirmé ce que je pensais et a permis de faire fonctionner correctement tous les composants.

Cependant, en testant le fonctionnement de l'écartement de la pince avec le capteur flex, je me suis rendu compte que le servo moteur essayait de réajuster sa position pour de trop petites variations sans arrêt ce qui donnait l'impression que la pince tremblait. Comme la pince s'ouvrait et se fermait trop pour des petits changements de position, j'ai établi un écart type dans le code pour lequel la pince ne bouge pas comme je l'avais fait pour les servos moteurs qui contrôlent les mouvements du bras robotique.

Finalement, après toutes les modifications apportées au code pendant cette séance, voici la version finale du code qui permet d'avoir un contrôle optimal de tout le bras robotique mais sans utiliser de bluetooth :

```
#include <ESP32Servo.h>


#define PIN_ACCEL_X 34
#define PIN_ACCEL_Y 35
#define PIN_ACCEL_Z 36
```

```
#define PIN_SERVO 16
#define PIN_SERVO2 13
#define PIN_SERVO3 14
#define PIN_SERVO4 15
#define FLEX_SENSOR_PIN 39 // Nouvelle broche pour le capteur flex
#define SERVO_PIN_5 4

Servo sg90;
Servo sg902;
Servo sg903;
Servo sg904;
Servo sg905;

const int INITIAL_POSITION_BASE = 45;     // Position initiale du servo sg90
const int INITIAL_POSITION = 0;           // Position initiale des autres
servos
const int MIN_ANGLE = 0;                  // Angle minimum
const int MAX_ANGLE = 90;                 // Angle maximum
const float ANGLE_CHANGE_THRESHOLD = 5.0; // Différence minimale pour un
changement d'angle
const int FILTER_SAMPLES = 30;             // Nombre d'échantillons pour le
filtrage
const int FLEX_MIN = 3500;   // Lecture minimale lorsque le capteur est
détendu
const int FLEX_MAX = 3000;  // Lecture maximale lorsque le capteur est
complètement plié
const int SERVO_MIN_ANGLE = 0;   // Angle minimum du servo pour la pince
const int SERVO_MAX_ANGLE = 90;  // Angle maximum du servo pour la pince

// Déclaration des variables pour le filtrage
float xHistory[FILTER_SAMPLES];
int filterIndex = 0;
float yHistory[FILTER_SAMPLES];

// Positions actuelles des servos
float currentAngleSg90 = INITIAL_POSITION_BASE;
float currentAngleSg902 = INITIAL_POSITION;
float currentAngleSg903 = INITIAL_POSITION;
float currentAngleSg904 = INITIAL_POSITION;
float currentAngleSg905 = SERVO_MIN_ANGLE;


void setup() {
  Serial.begin(115200);
  delay(1000);

  // Déclaration des PIN pour l'accéléromètre et les servos
  pinMode(PIN_ACCEL_X, INPUT);
  pinMode(PIN_ACCEL_Y, INPUT);
```

```
  pinMode(PIN_ACCEL_Z, INPUT);

  // On affecte chaque servo à son PIN
  sg90.attach(PIN_SERVO);
  sg902.attach(PIN_SERVO2);
  sg903.attach(PIN_SERVO3);
  sg904.attach(PIN_SERVO4);
  sg905.attach(SERVO_PIN_5);

  // On lit les résultats de l'accéléromètre sur 12 bits
  analogReadResolution(12);

  // Positionnement initial des servos
  sg90.write(INITIAL_POSITION_BASE);
  sg902.write(INITIAL_POSITION);
  sg903.write(INITIAL_POSITION);
  sg904.write(INITIAL_POSITION);
  sg905.write(SERVO_MIN_ANGLE);

  delay(1000);
}

// Fonction pour faire une moyenne de plusieurs échantillons en X afin d'avoir
une position plus précise
float getFilteredX() {
  float currentX = analogRead(PIN_ACCEL_X);
  xHistory[filterIndex] = currentX;
  filterIndex = (filterIndex + 1) % FILTER_SAMPLES;

  float sum = 0;
  for (int i = 0; i < FILTER_SAMPLES; i++) {
    sum += xHistory[i];
  }
  return sum / FILTER_SAMPLES;
}

// Fonction pour faire une moyenne de plusieurs échantillons en Y afin d'avoir
une position plus précise
float getFilteredY() {
  float currentY = analogRead(PIN_ACCEL_Y);
  yHistory[filterIndex] = currentY;
  filterIndex = (filterIndex + 1) % FILTER_SAMPLES;

  float sum = 0;
  for (int i = 0; i < FILTER_SAMPLES; i++) {
    sum += yHistory[i];
  }

  return sum / FILTER_SAMPLES;
```

```
}

void loop() {
  // Récupération des valeurs filtrées
  float currentX = getFilteredX();
  float currentY = getFilteredY();

  // Contrôle du servo moteur de la pince à l'aide du capteur flex

  // On récupère la valeur renvoyée par le capteur flex
  int flexValue = analogRead(FLEX_SENSOR_PIN);

  // Calcul de l'angle où on doit placer le servo en fonction des valeurs
renvoyées par le capteur flex
  int servoAngle = map(flexValue, FLEX_MIN, FLEX_MAX, SERVO_MAX_ANGLE,
SERVO_MIN_ANGLE);

  // On s'assure que l'angle calculé est compris entre 0 et 90 degré
  servoAngle = constrain(servoAngle, SERVO_MIN_ANGLE, SERVO_MAX_ANGLE);

  if (abs(servoAngle - currentAngleSg905) >= ANGLE_CHANGE_THRESHOLD) {
    // L'angle varie d'au moins ANGLE_CHANGE_THRESHOLD par rapport à l'angle
précédent

    // On change la position du servo et on actualise la valeur de l'angle
courant
      sg905.write(servoAngle);
      currentAngleSg905 = servoAngle;
  }

  // Contrôle du servo moteur en X

  // On map la valeur filtrée de l'accéléromètre en X pour convertir la valeur
reçu en angle
  int mappedAngleSg90 = map(currentX, 1050, 700, 0, 90);

  // On s'assure que l'angle calculé est bien compris entre 0 et 90 degré
  mappedAngleSg90 = constrain(mappedAngleSg90, MIN_ANGLE, MAX_ANGLE);

  // On vérifie si la différence entre la nouvelle position et la position
courante est d'au moins la valeur de ANGLE_CHANGE_THRESHOLD
  if (abs(mappedAngleSg90 - currentAngleSg90) >= ANGLE_CHANGE_THRESHOLD) {
    // La différence entre la nouvelle position et la position courante est
d'au moins la valeur de ANGLE_CHANGE_THRESHOLD

    // On change la position du servo et on actualise la position courante
    currentAngleSg90 = mappedAngleSg90;
    sg90.write(currentAngleSg90);
  }
```

```
  // Contrôle des 3 servos moteurs en Y

  // On vérifie si la valeur de Y renvoyée par l'accéléromètre est inférieur à
1900 (valeur pour laquelle on veut que la pince soit perpendiculaire au sol)
  if (currentY < 850) {
    // La valeur en Y renvoyée par l'accéléromètre est inférieure à 1900

    // On met les 3 servos en position 0 si la position a variée d'au moins
ANGLE_CHANGE_THRESHOLD
    sg902.write(0);
    sg903.write(0);
    sg904.write(0);

    // On actualise la valeur courante de l'angle pour chaque servo
    currentAngleSg902 = 0;
    currentAngleSg903 = 0;
    currentAngleSg904 = 0;
  }
  // On vérifie si la valeur de Y renvoyée par l'accéléromètre est comprise
entre 1900 et 2000 (valeurs pour lesquelles on veut que la pince commence à
descendre)
  else if (currentY >= 850 && currentY < 900) {
    // La valeur de Y renvoyée par l'accéléromètre est comprise entre 1900 et
2000

    // On calcul l'angle qui correspondra à la nouvelle position du servo
moteur du bas
    int angle2 = map(currentY, 850, 900, 0, 30);

    // On vérifie que l'angle varie d'au moins ANGLE_CHANGE_THRESHOLD par
rapport à l'angle précédent
    if (abs(angle2 - currentAngleSg902) >= ANGLE_CHANGE_THRESHOLD) {
      // L'angle varie d'au moins ANGLE_CHANGE_THRESHOLD par rapport à l'angle
précédent

      // On change la position du servo et on actualise la valeur de l'angle
courant
      sg902.write(angle2);
      currentAngleSg902 = angle2;
    }

    // On laisse les deux autres servos en positions 0 qui correspond à leur
valeur courante
    sg903.write(0);
    sg904.write(0);
    currentAngleSg903 = 0;
    currentAngleSg904 = 0;
  }
```

```
  // On vérifie si la valeur de Y renvoyée par l'accéléromètre est comprise entre 2000 et 2100 (valeurs pour lesquelles on veut commencer à déplacer le deuxième servo moteur)
  else if (currentY >= 900 && currentY < 950) {
    // La valeur en Y renvoyée par l'accéléromètre est comprise entre 2000 et 2100

    // On calcul la nouvelle position du deuxième servo moteur
    int angle3 = map(currentY, 900, 950, 0, 30);

    // On vérifie que l'angle varie d'au moins ANGLE_CHANGE_THRESHOLD par rapport à l'angle précédent
    if (abs(angle3 - currentAngleSg903) >= ANGLE_CHANGE_THRESHOLD) {
      // L'angle varie d'au moins ANGLE_CHANGE_THRESHOLD par rapport à l'angle précédent

      // On change la position du servo et on actualise la valeur de l'angle courant
      sg903.write(angle3);
      currentAngleSg903 = angle3;
    }

    // On place les deux autres servo dans la position souhaitée et on ajuste la valeur de leur angle courant
    sg902.write(30);
    currentAngleSg902 = 30;
    sg904.write(0);
    currentAngleSg904 = 0;
  }
  // On vérifie si la valeur de Y renvoyée par l'accéléromètre est comprise entre 2100 et 2200 (valeurs pour lesquelles on veut commencer à déplacer le troisième servo moteur)
  else if (currentY >= 950 && currentY < 1000) {
    // La valeur en Y renvoyée par l'accéléromètre est comprise entre 2100 et 2200

    // On calcul la nouvelle position du troisième servo moteur
    int angle4 = map(currentY, 950, 1000, 0, 30);

    // On vérifie que l'angle varie d'au moins ANGLE_CHANGE_THRESHOLD par rapport à l'angle précédent
    if (abs(angle4 - currentAngleSg904) >= ANGLE_CHANGE_THRESHOLD) {
      // L'angle varie d'au moins ANGLE_CHANGE_THRESHOLD par rapport à l'angle précédent
      sg904.write(angle4);

      // On change la position du servo et on actualise la valeur de l'angle courant
      currentAngleSg904 = angle4;
```

```
    }

    // On place les deux autres servo dans la position souhaitée et on ajuste
la valeur de leur angle courant
    sg902.write(30);
    currentAngleSg902 = 30;
    sg903.write(30);
    currentAngleSg903 = 30;
  }
  // On gère le cas où la valeur en Y renvoyée par l'accéléromètre est
supérieure à 2200
  else {
    // On déplace tous les servos en position 30 et on actualise leur angle
courant
    sg902.write(30);
    sg903.write(30);
    sg904.write(30);
    currentAngleSg902 = 30;
    currentAngleSg903 = 30;
    currentAngleSg904 = 30;
  }
  delay(10);
}
```

Fonctionnement du bluetooth :

Finalement, comme Matis avait commencé à faire le bluetooth mais que finalement on a pensé que c'était plus important qu'il commence le PCB car le cablage commençait à être un peu complexe et que des fils avaient tendance à se détacher de la plaque de test ce qui rendait compliqué les tests puisque à chaque fois il fallait retrouver où brancher le fil qui s'était détaché, donc c'est moi qui ferais la communication bluetooth entre les deux carte.

Pour rappel, le but est de pouvoir contrôler les mouvements de la pince à distance via l'accéléromètre et le capteur flex. Il faut donc utiliser deux cartes, une qui récupère les données de l'accéléromètre et du capteur flex. Celle-ci enverra les données à l'autre carte qui elle interprètera ces données et permettra de contrôler la pince via les servos moteurs.
Il était important d'avoir une version de la pince qui fonctionne parfaitement afin de commencer à travailler avec le bluetooth car le bluetooth pourrait ajouter un délai de transmission ce qui pourrait rendre plus complexe les mouvements de la pince.
J'ai donc commencé à faire des recherches sur le fonctionnement du bluetooth des cartes et comment faire en sorte que les deux cartes communiquent entre-elles.

Objectif de la prochaine séance :

L'objectif de la prochaine séance sera donc de continuer mes recherches sur l'établissement de la connexion Bluetooth entre les deux cartes de façon à ce qu'une carte envoie les données de l'accéléromètre et du capteur flex à l'autre carte et essayer de commencer à coder la transmission des données d'une carte vers l'autre carte.