Compte rendu séance 3 Téo Baillot d'Estivaux :

Objectifs de la séance :

Le but de cette séance est de commencer à contrôler les servos moteurs à l'aide de l'accéléromètre.

Fonctionnement de l'accéléromètre :

La première difficulté a été de faire une fonction qui permet de récupérer les valeurs renvoyées par l'accéléromètre car il n'existe pas de bibliothèque compatible ESP32 pour l'accéléromètre.

Premier problème, lorsque j'ai voulu afficher les valeurs de x, y et z de l'accéléromètre, je me suis rendu compte que les caractères renvoyés étaient bizarre et incompréhensibles :

```
���Z□�-%KjYZA^IT+�*Wk�□�-@HyPu���u@V��r@W5ü��Z□-%K$□A^IUk*Vk□-b
Y,A^IT+*Wk□-RK@Hy@uvU���Y@mW□R@@XQ@
```

Suite à mes recherches, ceci était dû au baud rate qu'il fallait adapter. Malheureusement, en mettant la même valeur dans le Serial.begin et le baud rate du serial monitor (155200), les caractères affichés étaient toujours incompréhensibles (pour des raisons que je ne comprends pas). J'ai donc essayé différents baud rate dans le serial monitor pour voir lequel fonctionnerait et en passant le baud rate à 74880 j'arrive enfin à afficher des informations compréhensibles dans le serial monitor :

```
Valeur X: 1180.60
Valeur X: 1491.80
Valeur X: 1518.00
Valeur X: 1494.40
Valeur X: 1446.80
Valeur X: 1416.00
Valeur X: 1443.80
Valeur X: 1491.60
Valeur X: 1583.20
```

Par la suite, je me suis rendu compte que la valeur renvoyée par Y et Z était toujours égale à 0 alors que je les avais connectés sur les broches 32 et 33 de la carte qui sont censé permettre de faire de la lecture analogique. J'ai donc testé d'autres PIN et finalement en branchant Y sur le PIN 35 et Z sur le PIN 34, j'arrive à récupérer des valeurs de Y et Z.

Interprétation des valeurs de l'accéléromètre :

Par la suite, l'objectif est maintenant d'interpréter les valeurs récupérées par l'accéléromètre afin de contrôler les mouvements de la pince.

Pour commencer, j'ai voulu interpréter que lorsqu'on tourne autour de l'axe Z, la pince devrait tourner de droite à gauche. J'ai donc dû utiliser différentes formules mathématiques avec des arctan

pour interpréter la rotation autour de Z en fonction du mouvement en X et Y en utilisant des coordonnées polaires.
Mais après beaucoup de temps à essayer sans succès, je me suis dit que ce serait plus simple d'utiliser uniquement les valeurs de X pour faire tourner la pince. Ainsi, j'ai fait un code qui permet de faire tourner la pince lorsqu'on penche l'accéléromètre suivant l'axe X. Le code fonctionnait, mais j'ai tout de même remarqué que le servo moteur était trop sensible et avait tendance à trembler à cause du fait qu'il essayait de changer de position trop souvent donc il a fallu filtrer les échantillons récupérés par l'accéléromètre afin d'augmenter la précision des mouvements du servo moteur en faisant la moyenne de plusieurs échantillons de l'accéléromètre. J'ai également dû déterminer un seuil qui permet d'éviter de bouger la pince si la nouvelle position du servo moteur est trop proche de la position actuelle tout en gardant un mouvement fluide.
Une autre erreur que j'ai fait au début du code est de vouloir récupérer des positions de référence pour X et Y qui correspondent à la valeur de X et de Y lorsqu'on téléverse le code. Le problème est qu'après des tests, je me suis rendu compte que si l'accéléromètre était initialement penché, on ne pouvait pas faire tourner correctement le servo du côté où l'accéléromètre est penché car la plage de valeur renvoyée par l'accéléromètre comportait donc plus de valeur pour tourner dans un sens que dans un autre ce qui rendait plus compliqué le contrôle des servo moteurs. Malheureusement, je ne m'en suis aperçu qu'une fois que je l'avais aussi fait pour Y car le problème était plus flagrant en Y qu'en X. J'ai donc décidé de supprimer cette fonctionnalité et ai utilisé à la place la valeur moyenne de la plage de donnée renvoyée par l'accéléromètre en position initiale.
Pour être plus clair, voici les éléments du code que j'ai supprimé :

Avant le setup :

```
// Variables pour le calcul

float referenceX;  // Référence X
float referenceY;  // Référence Y
```

Dans le setup :

```
  // Calibration de l'accéléromètre
  calibrateAccelerometer();
```

Fonction appelée dans le setup :

```
void calibrateAccelerometer() {
  float sumX = 0;
  float sumY = 0;

  // Initialisation de l'historique
  for (int i = 0; i < FILTER_SAMPLES; i++) {
    xHistory[i] = 0;
  }

  // Récupération des mesures pour calibrage
  for (int i = 0; i < 20; i++) {
    sumX += analogRead(PIN_ACCEL_X);
    sumY += analogRead(PIN_ACCEL_Y);
    delay(50);
```

```
  }

  referenceX = sumX / 20.0;
  referenceY = sumY / 20.0;
  isInitialized = true;

  Serial.print("Valeur X de référence: ");
  Serial.println(referenceX);
  Serial.print("Valeur Y de référence: ");
  Serial.println(referenceY);
  Serial.println("Calibration terminée.");
}
```

Calcul de la position du servo moteur dans le void loop() (le même raisonnement était suivi pour Y) :

```
  // Récupération des valeurs filtrées de X et Y
  float currentX = getFilteredX();
  float currentY = getFilteredY();

  Serial.println(currentX);

  // Calcul de la différence en X par rapport à la valeur de référence
  float xDiff = currentX - referenceX;
  if(xDiff<0){
    xDiff=-xDiff;
  }

  // Conversion de la valeur X observée en une position pour le servo
  float maxDiff = 500;
  float angleOffset = (xDiff / maxDiff) * 45; // Conversion de la différence
en angle

  // Calcul de la nouvelle position du servo
  float targetServoAngle = INITIAL_POSITION - angleOffset;
  targetServoAngle = constrain(targetServoAngle, MIN_ANGLE, MAX_ANGLE); //
Limite l'angle entre MIN_ANGLE et MAX_ANGLE
```

Code pour contrôler les mouvements du bras robotique à l'aide de l'accéléromètre :

Finalement, je suis arrivé à faire ce code qui a permis de contrôler correctement les mouvements du bras robotique à l'aide de l'axe X et de l'axe Y de l'accéléromètre :

```
#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

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

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 = 7;             // Nombre d'échantillons pour le
filtrage

// 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;

bool isInitialized = false;  // Flag d'initialisation

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);

  // Attachement des servos
  sg90.attach(PIN_SERVO);
  sg902.attach(PIN_SERVO2);
  sg903.attach(PIN_SERVO3);
  sg904.attach(PIN_SERVO4);

  // Lecture analogique sur 12 bits (valeur par défaut pour ESP32)
  analogReadResolution(12);
```

```
  // Positionnement initial des servos
  sg90.write(INITIAL_POSITION_BASE);
  sg902.write(INITIAL_POSITION);
  sg903.write(INITIAL_POSITION);
  sg904.write(INITIAL_POSITION);
  delay(1000);
}

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;
}

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() {
  if (!isInitialized) return;

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

  // Servo sg90
  int mappedAngleSg90 = map(currentX, 2300, 1600, 0, 90);
  mappedAngleSg90 = constrain(mappedAngleSg90, MIN_ANGLE, MAX_ANGLE);

  if (abs(mappedAngleSg90 - currentAngleSg90) >= ANGLE_CHANGE_THRESHOLD) {
    currentAngleSg90 = mappedAngleSg90;
    sg90.write(currentAngleSg90);
```

```
  Serial.print("Servo sg90 | Angle: ");
  Serial.println(currentAngleSg90);
}

// Contrôle des servos 2, 3 et 4
if (currentY < 1900) {
  if (abs(currentAngleSg902 - 0) >= ANGLE_CHANGE_THRESHOLD) sg902.write(0);
  if (abs(currentAngleSg903 - 0) >= ANGLE_CHANGE_THRESHOLD) sg903.write(0);
  if (abs(currentAngleSg904 - 0) >= ANGLE_CHANGE_THRESHOLD) sg904.write(0);

  currentAngleSg902 = 0;
  currentAngleSg903 = 0;
  currentAngleSg904 = 0;
} else if (currentY >= 1900 && currentY < 2000) {
  int angle2 = map(currentY, 1900, 2000, 0, 30);
  if (abs(angle2 - currentAngleSg902) >= ANGLE_CHANGE_THRESHOLD) {
    sg902.write(angle2);
    currentAngleSg902 = angle2;
  }
  sg903.write(0);
  sg904.write(0);
  currentAngleSg903 = 0;
  currentAngleSg904 = 0;
} else if (currentY >= 2000 && currentY < 2100) {
  int angle3 = map(currentY, 2000, 2100, 0, 30);
  if (abs(30 - currentAngleSg902) >= ANGLE_CHANGE_THRESHOLD) {
    sg902.write(30);
    currentAngleSg902 = 30;
  }
  if (abs(angle3 - currentAngleSg903) >= ANGLE_CHANGE_THRESHOLD) {
    sg903.write(angle3);
    currentAngleSg903 = angle3;
  }
  sg904.write(0);
  currentAngleSg904 = 0;
} else if (currentY >= 2100 && currentY < 2200) {
  int angle4 = map(currentY, 2100, 2200, 0, 30);
  if (abs(30 - currentAngleSg902) >= ANGLE_CHANGE_THRESHOLD) {
    sg902.write(30);
    currentAngleSg902 = 30;
  }
  if (abs(30 - currentAngleSg903) >= ANGLE_CHANGE_THRESHOLD) {
    sg903.write(30);
    currentAngleSg903 = 30;
  }
  if (abs(angle4 - currentAngleSg904) >= ANGLE_CHANGE_THRESHOLD) {
    sg904.write(angle4);
    currentAngleSg904 = angle4;
  }
```

```
  } else {
    if (abs(30 - currentAngleSg902) >= ANGLE_CHANGE_THRESHOLD)
sg902.write(30);
    if (abs(30 - currentAngleSg903) >= ANGLE_CHANGE_THRESHOLD)
sg903.write(30);
    if (abs(30 - currentAngleSg904) >= ANGLE_CHANGE_THRESHOLD)
sg904.write(30);

    currentAngleSg902 = 30;
    currentAngleSg903 = 30;
    currentAngleSg904 = 30;
  }

  delay(50);
}
```

J'ai tout de même remarqué que pour une même position les valeurs renvoyées par l'accéléromètre varient beaucoup ce qui rend le contrôle du bras robotique compliqué.

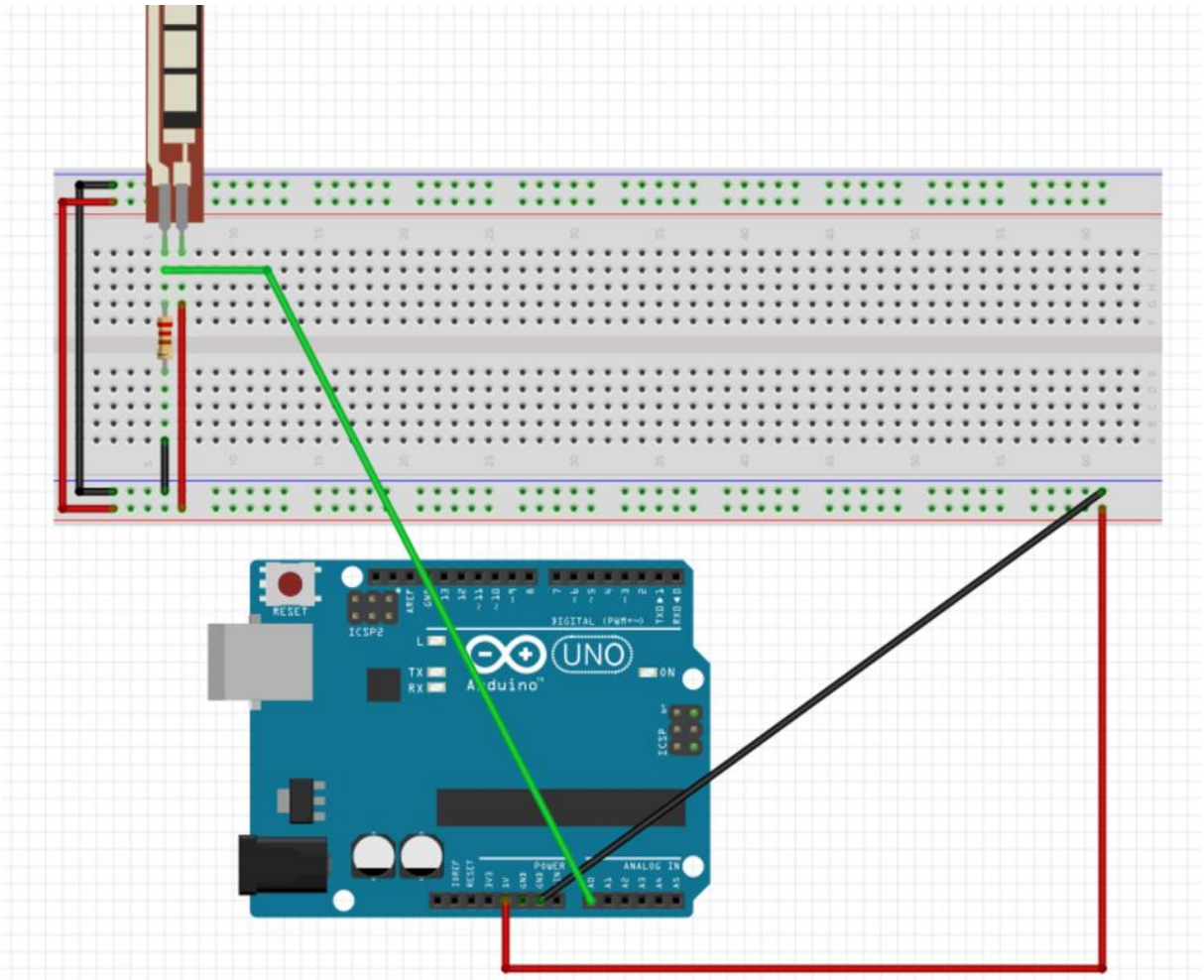
Contrôle du capteur flex :

Par la suite, l'objectif est de contrôler l'écartement de la pince à l'aide d'un capteur flex afin de pouvoir saisir des objets avec la pince.
L'idée est donc de récupérer les valeurs renvoyées par le capteur flex afin de convertir ces valeurs en angle qui définira la position du servo moteur.
En suivant cette idée, j'ai réalisé le code suivant :

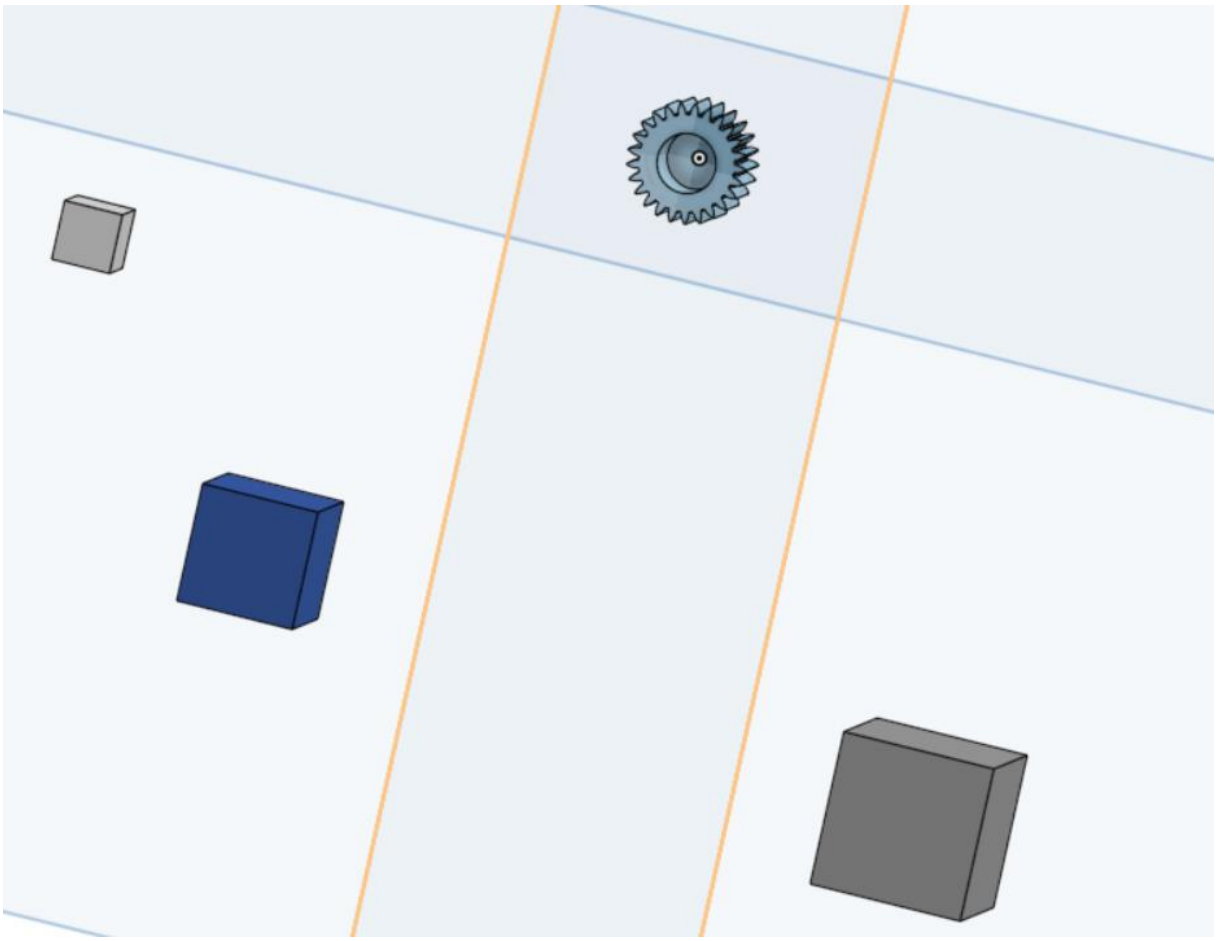
```
#include <ESP32Servo.h>


// Définition des broches
#define FLEX_SENSOR_PIN 39 // Nouvelle broche pour le capteur flex
#define SERVO_PIN_5 4


// Définition des plages de mesure
const int FLEX_MIN = 3500;   // Lecture ADC minimale lorsque le capteur est
détendu
const int FLEX_MAX = 3000;  // Lecture ADC maximale lorsque le capteur est
complètement plié

const int SERVO_MIN_ANGLE = 0;   // Angle minimum du servo
const int SERVO_MAX_ANGLE = 90;  // Angle maximum du servo

Servo sg905; // Objet pour contrôler le servo

void setup() {
  Serial.begin(115200);
  analogReadResolution(12); // Résolution 12 bits pour l'ADC
```

```
  // Configurer la broche du capteur flex
  pinMode(FLEX_SENSOR_PIN, INPUT);

  // Configurer le servo
  sg905.attach(SERVO_PIN_5);
  sg905.write(SERVO_MIN_ANGLE); // Position initiale à 0°

  delay(1000); // Pause pour stabiliser
}

void loop() {
  // Lire la valeur brute du capteur flex
  int flexValue = analogRead(FLEX_SENSOR_PIN);
  // Convertir la valeur du capteur flex en un angle servo
  int servoAngle = map(flexValue, FLEX_MIN, FLEX_MAX, SERVO_MAX_ANGLE,
SERVO_MIN_ANGLE);

  // Limiter l'angle entre les bornes définies
  servoAngle = constrain(servoAngle, SERVO_MIN_ANGLE, SERVO_MAX_ANGLE);

  // Déplacer le servo à l'angle calculé
  sg905.write(servoAngle);

  // Afficher les valeurs pour le débogage
  Serial.print("Capteur Flex (ADC): ");
  Serial.print(flexValue);
  Serial.print(" | Angle Servo: ");
  Serial.println(servoAngle);

  delay(100); // Pause pour lisser les mouvements
}
```

Malheureusement, j'ai commis une erreur dans les branchements du capteur flex ce qui faisait que je n'arrivais pas à récupérer les valeurs du capteur flex car j'avais branché une pate à la masse et une pate directement à un pin de l'esp32.
Finalement, j'ai compris qu'il fallait faire un pont diviseur de tension pour le faire fonctionner et j'ai donc suivi le schéma suivant :



Une fois le bon branchement effectué, le code avait l'air de fonctionner correctement et de faire ce qu'on voulait, mais l'imprécision des pièces 3d empêche la pince de fonctionner parfaitement car les dents de l'engrenage ne sont pas très bien imprimées ce qui fait qu'elle n'entraine pas bien les deux côtés de la pince.
J'ai donc remodélisé l'engrenage et réimprimé les pièces qui représentent la pince du bras robotique en essayant de faire en sorte que les dents s'impriment plus précisément. J'ai également imprimé des petits cubes pour essayer de les imprimer avec la pince. Voici les modélisations :



Test de l'ensemble de la pince :

Malheureusement on s'est rendu compte que le contrôle des mouvements du bras robotique étaient difficiles à contrôler à cause du fait que les valeurs renvoyées par l'accéléromètre étaient trop imprécises (le code fonctionne), je suis donc allé chercher d'autres accéléromètres pour voir si le problème venait de notre accéléromètre mais ça n'a pas réglé le problème.
Il faut donc faire un filtre RC pour essayer de lisser les valeurs renvoyées par l'accéléromètre en réduisant le bruit du capteur. Etant donné que l'on récupère une valeur de notre accéléromètre toutes les 50 millisecondes, il faut que le tho de notre filtre RC valle 5ms de façon à ce qu'on soit sûre qu'au bout de 10 tho la valeur récupérée soit correcte (car au bout de 10 tho on sait que la valeur sera correcte d'après la courbe du filtre RC). On doit donc avoir tho=1/RC=5ms. Pour ce faire, on va utiliser une capacité en céramique de 100pF (une capacité céramique est de meilleure qualité que les traditionnelles et est donc plus adaptée pour du filtrage). On a pas de capacité assez grosse pour nous permettre de faire ce qu'on veut donc on va prendre 470nF et 1Mohms ce qui nous donne une constante de temps de tho=2.1. Je testerais le filtre à la séance prochaine

Objectifs de la prochaine séance :

L'objectif de la prochaine séance sera donc d'implémenter ce filtre pour pouvoir avoir un meilleur contrôle de la pince et implémenter le code pour faire contrôler l'écartement de la pince avec le reste du code.

Conclusion :

Les objectifs de cette séance ont été largement remplis car j'ai beaucoup travaillé chez moi pour faire avancer le projet qui commençait à prendre trop de retard à mon goût.