Explication code

Table des matières

Lien du gitHub: https://github.com/RemiEC/Play-to-Heal

Introduction

Dans notre boitier, ce sont les myowares qui sont chargés de capter les signaux émis par les muscles. Mais il ne suffit pas de connecter ces myowares à la manette adaptative Xbox afin de déclencher les actions des boutons. En effet, les signaux envoyés par les myowares étant analogiques, il nous faut un intermédiaire qui puisse les convertir en signaux digitaux et plus précisément binaires : « 1 » si le muscle est contracté, « 0 » sinon. Et cet intermédiaire, c'est l'ESP32 avec son processeur qui va nous permettre de traiter les données reçues du myoware.

Dans l'ordre, on a donc les myowares qui enregistrent les signaux musculaires, l'ESP32 qui reçoit ces signaux, les traite et envoie un signal digital binaire à la manette adaptative qui active ou non les boutons pour le jeu. Mais la question qui se pose est : comment différencier un signal venant d'un muscle contracté et celui d'un muscle au repos ? La réponse est qu'il faut mettre en place des seuils, et c'est là toute l'utilité du code Arduino que nous avons implémenté.

Bien évidemment, le code Arduino ne se résume pas à mettre en place un seuil en brut établi de manière empirique, puis de comparer si les signaux reçus sont au-dessus ou en-dessous pour déclencher ou non les boutons. Et cela pour 2 raisons :

- d'une part, tous les muscles du corps n'envoient pas les mêmes signaux et l'intensité du signal d'un état contracté pour l'un peut correspondre à un état au repos pour l'autre ;
- d'autre part, le même muscle de deux utilisateurs différents peut envoyer des signaux complètement différents.

Il nous faut donc établir ces seuils de manière dynamique pour chaque utilisateur et pour chaque muscle de ces derniers.

Pour les connaisseurs de l'IDE Arduino, le code se sépare en 3 parties :

1. Une partie de déclaration de constantes (pins notamment), variables globales et bibliothèques à utiliser.

2. La partie setup qui comme son nom l'indique met en place l'environnement de travail du processeur (compter le nombre de muscles à connecter, établir leur seuil).
3. La partie loop qui itère à l'infini pour détecter si les signaux envoyés correspondent bien à des contractions.

Il est à noter que sur le github, plusieurs versions du code sont à disposition:

- Tout d'abord, il y a les versions monomuscles et multimuscles. Dans la version multimuscles, on va donner la possibilité à l'utilisateur de connecter plusieurs muscles au boitier là où la version monomuscle ne permet d'en connecter qu'un seul. Dans la suite du rapport, c'est la version multimuscles qui est présentée.
- Ensuite, il y a le code SIG et le code RAW. La différence se fait au niveau du traitement du signal reçu dans l'ESP32. Le code SIG récupère les signaux captés par le myoware et réalise une simple moyenne sur ces valeurs pour déterminer le seuil de contraction d'un muscle (processus détaillé ci-dessous). Le code RAW va récupérer le signal du myoware en amplitude et non en fréquence. Ces données sont plus précises mais plus volatiles d'où la nécessité de devoir filtrer les signaux reçus. Et pour ce faire on utilise des librairies bien spécifiques détaillées ci-dessous. Comme les codes SIG et RAW sont quasi-similaires, nous n'allons pas faire deux parties distinctes mais simplement mettre du texte en rouge pour indiquer les différences entre les deux codes.

Librairies, constantes, variables globales

Les librairies sont la première chose à mentionner dans un code. Elles sont appelées avec « #include » :

1. #include <Wire.h> : permet une communication I2C entre le Myoware et l'ESP32. Il permet aussi la transmission d'information sur les pins du OLED https://www.arduino.cc/en/reference/wire
2. #include <U8g2lib.h> : cette bibliothèque permet d'écrire des informations sur l'OLED. Elle n'est pas indispensable au traitement des signaux mais elle reste très importante pour afficher les résultats et permettre une interaction avec l'utilisateur https://www.arduino.cc/reference/en/libraries/u8g2/
3. #include <Filters.h> et #include <Filters/Butterworth.hpp> : ces deux bibliothèques permettent d'appeler des méthodes de filtre de butterworth pour filtrer les données reçues du myoware.
4. #include "logo.h" : n'est pas une bibliothèque et est très accessoire, elle permet seulement d'afficher le logo du pôle léonard de vinci lors du lancement du programme

Les constantes sont des éléments très importants qui correspondent aux pins des capteurs, des boutons, des transistors, et aux valeurs maximum ou minimum (nombre de muscles max) à définir :

1. Pin_Capteur[] : tableau comprenant les pins de l'ESP32 qui reçoivent les signaux des myowares. Si 3 muscles peuvent être connectés à l'ESP32, alors 3 entiers devraient se trouver dans ce tableau. Cela n'est pas grave si ce tableau possède 3 entiers et que un seul muscle est connecté, il y aura juste des valeurs inutilisées.
2. Pin_Transistor[] : tableau contenant les valeurs des pins qui renvoient le signal digital à la manette adaptative. Il doit y avoir autant de valeurs dans ce tableau que dans le tableau précédent.
3. Max_muscles : nombre maximum de muscles pouvant être connectés au boitier (contraint par le PCB de l'ESP32 et le nombre de myowares à disposition)

4. buttonPin, buttonHigh : pins du bouton
5. const int nEchantillons = 20 : nombre de valeurs que l'on met dans le tableau d'échantillon pour calculer une moyenne glissante des valeurs captées par le myoware.
6. auto filter1 = butter<4>(f_n) : premier filtre passe-haut ; f_n correspond à la fréquence de coupure
7. auto filter2 = butter<4>(f_n2) : deuxième filtre passe-haut ; f_n2 correspond à la fréquence de coupure.
   Veuillez noter que plus tard dans le code, on effectue la soustraction du premier filtre par ce deuxième filtre pour faire un filtre passe-bande.

Les variables globales sont des objets accessibles et modifiables partout dans le code que ce soit le setup, le loop ou encore les fonctions

1. Theshold[Max_muscles] : tableau contenant les seuils pour chaque muscle
2. Array_values[128] : tableau accessoire qui sert à l'affichage des courbes sur l'OLED du signal renvoyé par les myowares. 128 correspond au nombre de pixels sur la largeur de l'écran OLED.
3. float echantillon[nEchantillons] : tableau d'échantillon avec lequel on calcule la moyenne glissante.

Setup

On commence par initialiser l'environnement avec la librairie de l'OLED (police), puis le Serial et enfin on déclare les pin modes pour le bouton.

Partout dans le code on retrouve des appels à la fonction DisplayOLED. Cette fonction prend en paramètre un string et un int et affiche sur l'OLED le string à la position indiquée par l'entier (0 : en haut, 1 : milieu, 2 : en bas). Attention à ne pas rentrer de string trop long, auquel cas le texte sortira de l'écran (pas de retour à la ligne automatique). Cette fonction sert surtout à guider l'utilisateur dans les étapes à suivre, comme un tutoriel. Au total, sur les 7 fonctions qui composent ce programme, 4 sont destinées à l'affichage OLED et ne sont donc pas indispensables au bon fonctionnement du programme. Elles restent cependant extrêmement importante pour l'UX.

Avant de rentrer dans les calculs de seuils, il nous faut déterminer combien de muscles l'utilisateur veut connecter. La fonction CountMuscles va donc demander à l'utilisateur d'appuyer sur le bouton afin d'incrémenter le nombre de muscle. Si le nombre de muscles dépasse le nombre max de muscles que l'on peut connecter, alors la fonction retourne à 1. Pour valider son choix, l'utilisateur doit rester appuyer pendant deux secondes sur le bouton.

Une fois le nombre de muscles à connecter sélectionné, on rentre dans la boucle qui va calculer les seuils et qui va se répéter autant de fois qu'il y a de muscles à connecter. Pour calculer les seuils, il nous faut deux choses : la moyenne du signal lorsque le muscle est décontracté et la moyenne du signal lorsque le muscle est contracté. De cette manière, on va pouvoir établir une valeur entre moyenne_max et moyenne_min qui sera représentative d'une contraction. Si on prenait moyenne max comme seuil, il se pourrait que l'on rate certaines contractions. Et si on prenait moyenne min comme seuil, le programme pourrait interpréter des contractions là où il n'y en a pas. Par conséquent, une valeur assez éloignée des deux moyennes nous assure une bonne précision avec peu d'erreur pour la détection de contraction. Pour le code RAW, les moyennes min et max sont calculées en prenant la moyenne des racines carrées des moyennes glissantes calculées avec le tableau d'échantillons (qui s'update constamment) . La valeur du seuil est ensuite mise dans le tableau threshold[].

Loop

Dans la boucle, pour chaque muscle, il nous faut vérifier si le signal capté par le myoware est supérieur ou non au seuil précédemment calculé. Pour se faire, on crée une boucle for qui itère sur chaque muscle et qui appelle la fonction SendSignal.

SendSignal va simplement comparer la valeur lue par le myoware avec le seuil pour ce muscle et mettre la pin transistor de ce muscle à High si il y a contraction (« *digitalWrite(pin_Transistor[m], HIGH);* ») ou Low sinon. Pour le RAW, on prend la moyenne glissante des dernières valeurs lues par le myoware (sur lesquelles on a appliqué le filtre passe-bande), on prend la racine carrée de cette moyenne et on la compare au seuil. On affiche également sur l'OLED s'il y a contraction ou non afin de voir plus facilement si le programme fonctionne correctement.

Nota Bene

Il est à noter que le code RAW, bien qu'il fonctionne, reste incomplet et mérite d'être peaufiné afin de traiter des signaux plus propres et plus précis. Celui-ci renvoie un signal en fréquence qui mérite de ce fait d'être traité contrairement au SIG qui renvoie un signal en amplitude. A l'heure actuelle, le code SIG est très efficace et détecte les contractions de manière fiable et sûre. Le perfectionnement du code RAW fait donc partie des axes d'améliorations du projet.

De plus, pour passer de l'analyse SIG au RAW, il suffit simplement de changer la nappe (fil bleu) sur le PCB myoware et de téléverser le bon code dans l'ESP32 (pris sur git).

Pour ce qui est des améliorations à apporter au code, la fonction display OLED pourrait être grandement simplifiée si les tableaux de string étaient bien pris en compte par Arduino (ce qui n'est pas du tout le cas et cela implique énormément de bugs). Certains morceaux de codes peuvent être remplacés par des fonctions existantes pour réduire le nombre de ligne. De même que les délais peuvent être diminués voire supprimés dans certains cas.