UPMC

Master SESI

PERI

Accéléromètre et Gyroscope

Introduction

Ce TP est une première prise en main d'une centrale 6-axes composée d'un accéléromètre 3D et d'un gyroscope 3D. Ce capteur MEMS coûte environ 2-3 euros. Il est impossible d'avoir un capteur très précis. Il va donc falloir filtrer les signaux pour les débruiter. Comme l'Arduino est un petit micro-contrôleur, il va être nécessaire de faire les calculs en virgule fixe.
Le TP est composé de deux parties. La première consiste en un propotypage sur PC avec gcc, pour bénéficier de la puissance de la machine (éditeur, disque dur rapide, ...) et de tous les outils de debug et de mise au point (comme valgrind). La seconde partie consiste à porter dans l'environnement Arduino les fonctions prototypées sur PC. Les règles de codage et de découpage en fonctions sont celles vues lors des précédents TP, tout comme les fonctions `init` et `loop`.

1 Prototypage

Les fonctions principales se trouvent dans le fichier `test_acceleroNR.c`. Les filtres à coder se trouve dans le fichier `acceleroNR.c`.

1.1 Acquisition du signal

Il y a trois tableaux globaux :
- `X_sig` : le signal en entrée qui est bruité
- `X_fir` : le signal filtrée par un filtre FIR (filtre non récursif)
- `X_iir` : le signal filtrée par un filtre IIR (filtre récursif)

Ces tableaux ont une taille `SIZE+1`.

1.2 Analyse du bruit

La fonction `calc_stats` calcule, sur une fenêtre glissante, la moyenne du signal, sa variance et son écart-type (à ne pas calculer sur Arduino car coûte trop de cycles à cause de la racine carrée). Afin d'avoir un calcul rapide de ces paramètre on utilise l'astuce suivante qui permet d'avoir une complexité de calcul indépendante du nombre d'éléments (`SIZE`). Dans les notations mathématiques suivantes, on pose $k =$ `SIZE`. Comme vu en cours, le calcul de la moyenne et de la variance font intervenir les moments d'ordre 0, 1 et 2 : S, S_x et S_xx. Soit $S_x(n)$ la somme à l'instant n de k points :

$$S_x(n) = x(n) + x(n-1) + \cdots + x(n-k+1) \tag{1}$$

La somme à l'instant suivant est :

$$S_x(n+1) = x(n+1) + x(n) + \cdots + x(n-k+2) \tag{2}$$

En faisant la différence, on obtient $S_x(n+1)$ en fonction de $S_x(n)$:

$$S_x(n+1) = S_x(n) + x(n+1) - x(n-k+1) \tag{3}$$

Il suffit donc d'additionner le nouveau point et de soustraire le plus ancien. Pour cela, il est nécessaire d'avoir en permanence les $k+1$ derniers points. C'est pour cette raison que les tableaux font `SIZE+1` cases et non `SIZE` cases.

Il y a deux façons de mémoriser les $k+1$ derniers points :
- approche HW : utilisation registre à décalage : à chaque top d'horloge, chaque valeur est décalée d'une case vers la gauche (ou vers la droite, en fonction),

— approche SW : utilisation d'un buffer circulaire : à chaque échantillon l'indice d'insertion est incrémenté. Lors qu'il atteint la fin du tableau, il est ramené au début.

Le choix est d'utiliser l'approche SW car elle nécessite moins d'accès mémoire. La variable jouant le rôle d'indice circulaire est `g_i`.

Il y a en plus une seconde variable `g_c` qui joue le rôle de compteur d'itération car lorsqu'on a moins de k valeurs dans les tableaux, on souhaite tout de même pouvoir faire des calculs de moyenne et de variance. Il faut donc pouvoir tester si on est dans ces cas particulier ou pas.

Votre travail consiste à coder la fonction `calc_stats`.

1.3 Filtrage FIR

Le filtre FIR à implémenter est un moyenneur de taille k. Pour avoir une implémentation rapide, on utilisera les mêmes astuces que pour `calc_stats`.

Votre travail consiste à coder la fonction `calc_fir`.

1.4 Filtrage IIR

Le filtre IIR à implémenter est le filtre de taille 1

$$y(n) = (1-\alpha)x(n) + \alpha y(n-1) \tag{4}$$

Mais plutôt que de l'utiliser sous sa forme classique (qui sous entend un codage en flottant), on va l'utiliser sous une forme faisant apparaitre des puissances de 2 :

$$y(n) = \frac{\alpha x(n) + \beta y(n-1)}{\alpha + \beta} \quad \text{avec } \alpha + \beta = 2^p \tag{5}$$

Comme l'accéléromètre est très bruité, on va donner peu d'importance à $x(n)$ et beaucoup à $y(n-1)$. Si on prend $p = 1$ on prendra forcément le couple de valeur $(1, 1)$. Si $p = 2$ on pourra prendre $(1, 3)$. Si $p = 3$, $(1, 7)$, etc.
Votre travail consiste à coder la fonction `calc_iir`.
Un *framework* vous est fourni pour mettre au point le code de vos fonctions. Il suffit de remplir les trous.

2 Portage sur Arduino

La centrale MPU6050 (ie accéléromètre + gyroscope) possède des convertisseurs CAN 16 bits qui échantillonnent le système à une fréquence allant de 4 Hz à 8 kHz. Vu la quantité de calculs et les nombreux affichages à réaliser via le bus USB, on va donc utiliser la centrale à basse fréquence.

2.1 Code faux

Le premier code fourni a été récupéré sur internet et il est faux : il échantillonne bien le système, mais les valeurs mesurées ne correspondent pas aux noms des grandeurs affichées. A vous de trouver l'erreur (elle est aussi présente dans le second code).

2.2 Filtrage de la centrale inertielle MPU6050

Intégrez les fonctions que vous avez prototypé. On se contentera de ne filtrer qu'une des six grandeurs à la fois (il faudra néanmoins les tester). En fonction de la quantité de bruit observée, adaptez votre filtre iir (la paire de valeurs) pour minimiser la variance du bruit. Une fois cela réalisé vous aurez un accéléromètre et un gyroscope relativement stable.

Attention toutefois : les CAN retournant des valeurs 16 bits, dès que vous calculerez S_{xx}, ie des sommes de 32 bits, un overflow va se produire. A vous de le régler.