TP9 - Filtrage

Jiang : 3602103 Li : 3770906

1. Objectif

Le but de ce TP est de d'implémenter différents filtres en flottant et en virgule fixe pour ensuite évaluer la qualité du filtrage.
Les fonctions principales se trouvent dans le fichier test_filterNR.c. Les filtres à coder se trouve dans le fichier filterNR.c. Les différentes fonctions de calcul sont paramétrables pour laisser à chacun la possibilités de tester un grand nombre de cas. Par défaut, les paramètres sont réglés à des valeurs représentatives de la réalité.

2. Signal de base et bruit

2.1 Signal de base

Plusieurs formes de signaux de taille n (paramètre) sont disponibles (fichier sampleNR.c):

generate_sample_constant	constant (constant) signal constant de valeur m = 100
generate_sample_step_up	front montant (step_up) d'amplitude a, passant de m – a à m + a, la transition se faisant au milieu du signal en n/2
generate_sample_step_down	front descendant (step_down) d'amplitude a, passant de m – a à m + a, la transition se faisant au milieu du signal en n/2
generate_sample_step_updown	échelon (step_updown) d'amplitude a, passant de m – a à m + a, la première transition se faisant au premier quart en n/4 et la seconde au troisième quart en 3n/4

tableau(1)

Les signaux seront composés d'échantillons codés sur 8 bits (uint8) stockés dans des tableaux dynamiques (ui8vector).

2.2 Types des valeurs

Dans ce TP , on utilise plusieurs types de valeurs pour calcul. La tableau ci-dessous explique les forms de valeur dans langage C:

Nom dans ce TP	Nom dans C	Taille	Range	printf format string
byte/uint8	unsigned char	1 byte	0 - 255	%c, %hhu
sint8	signed char	1 byte	-128 - 127	%c, %hhd, %hhi
uint16	unsigned short	2 bytes	0 - 65535	%hu
sint16	signed short	2 bytes	-32768 - 32767	%hi, %hd
uint32	unsigned int	4 bytes	0 to 4294967295	%u
sint32	signed int	4 bytes	-2147483648 to 2147483647	%d
float32	float	4 bytes	$5.563x10^{-309}$ to $1.798x10^{+308}$ (15 sf)	%f, %e, %g
float64	double	8 bytes	$7.065x10^{-9865}$ to $1.415x10^{9864}$ (18 sf or 33 sf)	%lf, %e, %g

Tableau(2)

2.3 Bruit et Estimateur d'erreur

Plusieurs bruits sont disponibles (fichier noise.c), mais nous n'utiliserons que le bruit blanc gaussien (bruit additionnel)
— bruit gaussien gaussian_noise_ui8vector, d'écart type sigma
— bruit impusionnel impulse_noise_ui8vector avec une probabilité percent

Ce même fichier contient les fonctions d'analyse qualitative : MSE et PSNR.

MSE (mean square error) mesure la moyenne de square des erreurs ou déviations .
(mean_square_error_ui8vector())

$$mse(x, \tilde{x}) = \frac{1}{n^2} \sum (x - \tilde{x})^2$$

Le PSNR (Peak Signal-to-Noise Ratio) est une mesure de distorsion.(psnr_ui8vector())
Plus grand de PSNR, Moins de distorsion de sortie.

$$psnr(x, \tilde{x}) = 10 \times log_{10} \frac{max(x)^2}{mse(x, \tilde{x})}$$

3 Filtrage

3.1 Filtrage non récursif (FIR)

Le filtre non récursif fourni est un filtre moyenneur codé en flottant (float32) fir_average_f32. Son paramètre est le radius r :

$$y(n) = \frac{1}{2r+1} \sum_{i=-r}^{i=+r} x(n+i)$$

3.1.1 Problème de Division entière :

Codez la version fir_average_i16 où les calculs seront fait dans des variables de types uint16 et où la division sera réalisée via une division entière:

y = s / d; change à : y = (s + (d / 2)) / d;

3.1.2 Problème de Division entière avec :

Codez la version fir_average_q16 : Le paramètre supplémentaire de cette fonction est **q** le nombre de bits de codage. Le PSNR de FIR qui a différente valeur de q est dans la tableau(3).

3.1.3 Comparaison des FIR :

La fonction implémentant le filtre gaussaient étant déjà codée : fir_gauss_f32, comparez les performances de ces 4 filtres : la tableau indique en fonction du niveau de bruit (sigma_noise) et de la largeur du filtre (pour les filtre moyennes) et de q (pour le filtre en virgule fixe), le psnr résultat.

	fir_gauss_f32	fir_average_f32	fir_average_i16	fir_average_q16 (**q= 8**)	fir_average_q16 (**q= 10**)
sigma_noise= 1.4f	43.08 db	-	-	-	-
sigma_noise= 1.0f	38.98 db	-	-	-	-
sigma_noise= 0.5f	29.00 db	-	-	-	-
radius = 2	-	42.37 db	42.37 db	42.28 db	42.28 db
radius = 3	-	49.46 db	49.46 db	43.67 db	47.62 db
radius = 4	-	49.11 db	49.11 db	43.84 db	48.78 db
radius = 5	-	54.04 db	54.04 db	48.56 db	54.29 db
radius = 6	-	55.47 db	55.47 db	34.88 db	50.70 db
radius = 7	-	59.58 db	59.58 db	55.47 db	55.47 db
radius = 8	-	60.47 db	60.47 db	57.81 db	57.81 db
radius = 9	-	60.61 db	60.61 db	34.08 db	43.78 db

tableau(3)

3.2 Filtrage récursif (IIR)

Le filtre récursif fourni est le filtre de FGL codée en flottant iir_f32. Son paramètre est α et ses coefficients sont b_0, a_1 et a_2 :

$$b_0 = (1-\gamma)^2, \quad a_1 = 2\gamma, \quad a_2 = -\gamma^2, \quad \text{avec } \gamma = e^{-\alpha}$$

$$y(n) = b_0 x(n) + a_1 y(n-1) + a_2 y(n-2)$$

3.2.1 Codez la version iir_q16 où les coefficient sont multipliés par 2^q et les calculs se font dans des variables de type sint16.

$$Q=2^q,\ B_0 = b_0 * Q,\ A_1 = a_1 * Q,\ A_2 = a_2 * Q$$

,et si on utilise des variables temporaires pour stocker les différentes valeurs de y(n) : $x_0 = x(n)$, $y_0 = y(n)$, $y_1 = y(n-1)$, $y_2 = y(n-2)$ alors l'équation du filtre devient :

y0 = (B0 * x0 + A1 * y1 + A2 * y2)/Q;

3.2.2 Afin d'augmenter la précision des calculs, on décide d'augmenter le nombre de bits pour coder les échantillons desxety.Pourcelaonpose$X_0 = x_0 \times Q, Y_0 = y_0 \times Q, Y_1 = y_1 \times Q, Y_2 = y_2 \times Q$.Codezlaversioniir_q32 de ce filtre. Pour debugger et mettre au point le code, il est conseillé d'observer le contenu des variables et d'afficher leur valeurs à chaque tour de boucle. Le calcul devient :

3.2.3 Comparer les trois implémentations du filtre récursif à celles des filtres non récursifs. Conclure sur l'utilisation de la virgule fixe.

	iir_f32	iir_q16 q=8	iir_q32 q=8
Alpha= 0.4	47.92 db	25.22 db	25.22 db
Alpha= 0.6	44.67 db	25.22 db	25.22 db
Alpha= 0.8	41.58 db	12.03 db	12.03 db