Συστήματα Πολυμέσων

Ρούσος Σταμάτης

ΑΕΜ : 10704

20 Φεβρουαρίου 2026

1 Introduction

Σε αυτη την εργασια θα φτιαξουμε ενα συστημα Encoding/Decoding ηχου κατα το προτυπο AAC (Advanced Audio Coding). Σκοπος ειναι να παρουμε ενα αρχικο σημα ηχου, να το συμπιεσουμε - κωδικοποιησουμε σε ενα σημα μικροτερων "διαστασεων" (λιγοτερα bits) οσο περισσοτερο γινεται, και ταυτοχρονα να το αποκωδικοποιησουμε - ανακατασκευασουμε οσο πιο καλα γινεται, δηλαδη να ειναι οσο δυνατον περισσοτερο ομοιο με το αρχικο (ελαχιστη αντιληπτη διαφορα στην ακοη του σηματος). Με αλλα λογια, αν x το αρχικο σημα ησχου και $\tilde{x}$ η εξοδος του αποκωδικοποιητη, θελουμε:

$$x \approx \tilde{x}$$

Αρα επιδιωκουμε μεγιστη συμπιεση με trade off την ποιοτητα.

2 SSC (Sequence Segmentation Control)

Η βαθμιδα SSC (Sequence Segmentation Control) κατηγοριοποιει τα frames σε τυπους. Στο AAC χρησιμοποιουνται τεσσερις τυποι frame:

- **OLS** (Only Long Sequence): long frame, καταλληλο για στασιμα/ομαλα τμηματα σηματος.

- **ESH** (Eight Short Sequence): 8 short subframes, καταλληλο για μεταβατικα/κρουστικα τμηματα (attacks), ωστε να μειωνεται το pre-echo.
- **LSS** (Long Start Sequence): μεταβατικο frame που ομαλα μεταβαινει απο OLS σε ESH.
- **LPS** (Long Stop Sequence): μεταβατικο frame που ομαλα μεταβαινει απο ESH σε OLS.

Η βασικη ιδεα ειναι η ανιχνευση αποτομων αυξησεων ενεργειας (attacks) στο *επομενο* frame, ωστε το *τρεχον* frame να επιλεχθει ως καταλληλο μεταβατικο (LSS) πριν απο την εισοδο σε short sequence (ESH). Αντιστοιχα, οταν δεν υπαρχει attack στο επομενο frame, το συστημα μπορει να επιστρεψει σε long frames, χρησιμοποιωντας LPS για ομαλη μεταβαση.

Στην υλοποιηση, η ανιχνευση attacks γινεται με high-pass φιλτρο και ελεγχο ενεργειας σε 8 διαδοχικα τμηματα των 128 samples. Υπολογιζεται ενα μετρο *attack value* ως λογος της ενεργειας ενος τμηματος προς τον μεσο ορο των προηγουμενων τμηματων, και αν ξεπεραστουν συγκεκριμενα thresholds, το επομενο frame θεωρειται ESH. Στη συνεχεια, με βαση τον τυπο του προηγουμενου frame και την πληροφορια αν το επομενο θα ειναι ESH, αποφασιζεται ο τυπος του τρεχοντος frame. Τελος, επειδη υπαρχουν δυο καναλια, υπολογιζεται τυπος ανα καναλι και συνδυαζεται σε εναν τελικο τυπο με βαση εναν πινακα συνδυασμου.

Στην υλοποιηση μου, το αρχειο SSC περιεχει τις εξης συναρτησεις/δομες:

- _is_next_esh: Ελεγχει αν το **επομενο** frame (ανα καναλι) πρεπει να χαρακτηριστει ως ESH, με βαση attack detection.
- SSC: Επιλεγει τον τυπο του **τρεχοντος** frame (OLS, LSS, ESH, LPS) με βαση το προηγουμενο frame type και την προβλεψη για το επομενο.

- _COMBINE_TABLE: Πινακας που συνδυαζει τους τυπους των δυο καναλιων σε εναν τελικο τυπο frame.
- _HP_A, _HP_B: Συντελεστες του high-pass IIR φιλτρου που χρησιμοποιειται για attack detection.

2.1 _is_next_esh(next_frame_ch)

Η συναρτηση αυτη παιρνει ως ορισμα το σημα του **επομενου** frame για **ενα** καναλι:

$$next\_frame\_ch \in \mathbb{R}^{2048}$$

δηλαδη ενα μονοδιαστατο διανυσμα 2048 δειγματων.

Λειτουργια: Σκοπος της ειναι να ανιχνευσει αν το επομενο frame περιεχει attack, ωστε να χρειαστει short sequence (ESH). Η διαδικασια ειναι:

1) Για την ανιχνευση αποτομων μεταβολων (attacks) στο επομενο frame χρησιμοποιειται το υψιπερατο φιλτρο:

$$\mathbf{H(z) = \frac{0.7548 - 0.7548z^{-1}}{1 - 0.5095z^{-1}}}$$

Στο πεδιο z ισχυει:

$$Y(z) = H(z)X(z)$$

δηλαδη:

$$Y(z)(1 - 0.5095z^{-1}) = X(z)(0.7548 - 0.7548z^{-1})$$

Περνωντας στο διακριτο πεδιο χρονου, με την αντιστοιχιση $z^{-1}X(z) \leftrightarrow x[n-1]$ και $z^{-1}Y(z) \leftrightarrow y[n-1]$, προκυπτει η εξισωση διαφορας:

$$y[n] - 0.5095y[n-1] = 0.7548x[n] - 0.7548x[n-1]$$

η οποια γραφεται:

$$\boxed{y[n] = 0.5095y[n-1] + 0.7548\big(x[n] - x[n-1]\big)}$$

Παρατηρουμε οτι ο ορος $x[n] - x[n-1]$ αποτελει διακριτη παραγωγο. Επομενως, οσο μεγαλυτερη ειναι η διαφορα μεταξυ διαδοχικων δειγματων, τοσο μεγαλυτερο γινεται το $y[n]$. Το φιλτρο επομενως τονιζει τις αποτομες μεταβολες του σηματος ("attacks").

Ο ορος $0.5095y[n-1]$ εισαγει μνημη στο συστημα και δινει ομαλοτητα στην αποκρισή του. Το φιλτρο ειναι IIR πρωτης ταξης, καθως περιλαμβανει αναδρομικο ορο. Ο πολος του συστηματος βρισκεται στο:

$$z = 0.5095$$

και επειδη $|0.5095| < 1$, ο πολος βρισκεται εντος του μοναδιαιου κυκλου, αρα το φιλτρο ειναι ευσταθες.

2) Χρηση του φιλτρου στο SSC

Το SSC λαμβανει το επομενο frame $i+1$ και υπολογιζει για ολα τα δειγματα του την εξοδο του φιλτρου:

$$y[n] = 0.5095y[n-1] + 0.7548x[n] - 0.7548x[n-1]$$

Χωριζει ολο το frame σε 8 διαδοχικα τμηματα μηκους 128 δειγματων. Για καθε τμημα $l = 0, 1, \ldots, 7$ υπολογιζεται η ενεργεια:

$$s_l^2 = \sum_{n=128l}^{128(l+1)-1} y[n]^2$$

Για καθε $l = 1, \ldots, 7$ υπολογιζεται επιπλεον ο μεσος ορος των ενεργειων των προηγουμενων τμηματων:

$$\overline{s}_{l-1}^2 = \frac{1}{l}\sum_{m=0}^{l-1} s_m^2$$

και οριζεται το μετρο attack:

$$d_{s_l}^2 = \frac{s_l^2}{\frac{1}{l}\sum_{m=0}^{l-1} s_m^2}$$

Οσο μεγαλυτερη ειναι η ενεργεια s_l^2 σε καποιο τμημα (δηλαδη οσο περισσοτερη μεταβολη υπαρχει στο σημα ($y[n]$) την οποια εντοπισε το φιλτραρισμα), και οσο μικροτερος ειναι ο μεσος ορος των προηγουμενων ενεργειων (δηλαδη οσο μεγαλυτερη ησυχια προηγειται), τοσο μεγαλυτερο γινεται το "attack" $d_{s_l}^2$

Εαν για εστω ενα τμημα l ισχυει ταυτοχρονα:

$$s_l^2 > 10^{-3} \quad \text{και} \quad d_{s_l}^2 > 10$$

τοτε το επομενο frame κατηγοριοποιειται ως **EIGHT SHORT SEQUENCE**.

Για την κατηγοριοποιηση του τρεχοντος frame i μας αρκει να γνωριζουμε αν το επομενο frame $(i+1)$ θα ειναι EIGHT SHORT SEQUENCE ή οχι, οπως προκυπτει απο τη λογικη μεταβασεων.

2.2 _COMBINE_TABLE

Επειδη ο ελεγχος attack γινεται ανα καναλι, μπορει να προκυψει διαφορετικος τυπος τρεχοντος frame για το αριστερο και το δεξι καναλι. Η AAC λογικη απαιτει τελικα **εναν** κοινο τυπο frame, ωστε να χρησιμοποιηθει κοινη δομη παραθυρου/MDCT για τα δυο καναλια.

Ο πινακας _COMBINE_TABLE ειναι ενα dictionary που υλοποιει ακριβως αυτον τον συνδυασμο:

$$(\text{type}_{ch0}, \text{type}_{ch1}) \mapsto \text{frame\_type}$$

Για παραδειγμα:

- (OLS, OLS) → OLS
- (OLS, ESH) → ESH

- (OLS, LSS) $\rightarrow$ LSS
- (ESH, OLS) $\rightarrow$ ESH
- (LPS, OLS) $\rightarrow$ LPS
- (LSS, LPS) $\rightarrow$ ESH

Γενικα, ο πινακας τεινει να επιλεγει τον "πιο απαιτητικο" τυπο (π.χ. ESH) οταν τα καναλια διαφωνουν, ωστε να μην χαθει attack σε κανενα καναλι.

2.3 SSC(frame_T, next_frame_T, prev_frame_type)

Η συναρτηση αυτη υλοποιει τον Sequence Segmentation Control και επιλεγει τον τυπο του **τρεχοντος** frame.

Ορισματα:

- frame_T: το τρεχον frame στο πεδιο χρονου, διαστασεων (2048,2) για stereo.
- next_frame_T: το επομενο frame στο πεδιο χρονου, διαστασεων (2048,2).
- prev_frame_type: string που δηλωνει τον τυπο του προηγουμενου frame (OLS, LSS, ESH, LPS).

Λειτουργια: Η συναρτηση ακολουθει τα βηματα:

1. **Ελεγχος αν το επομενο frame θα ειναι ESH (ανα καναλι):**

$$next\_esh\_ch0 = \_is\_next\_esh(next\_frame\_T[:, 0])$$

$$next\_esh\_ch1 = \_is\_next\_esh(next\_frame\_T[:, 1])$$

2. **Αποφαση τρεχοντος frame type ανα καναλι:** Στη συνεχεια υπαρχει μια εσωτερικη βοηθητικη λογικη decide(prev, next_is_esh) που αποφασιζει τον τυπο του **τρεχοντος** frame για καθε καναλι, με βαση:
   - τον τυπο του **προηγουμενου** frame

- το αν το **επομενο** frame προβλεπεται ESH

Η λογικη μεταβασεων ειναι:

- Αν **prev = OLS**:
  - αν **next_is_esh = True** $\Rightarrow$ **current = LSS**
  - αλλιως $\Rightarrow$ **current = OLS**
- Αν **prev = ESH**:
  - αν **next_is_esh = True** $\Rightarrow$ **current = ESH**
  - αλλιως $\Rightarrow$ **current = LPS (Long Stop Sequence)**
- Αν **prev = LSS**:

  current = ESH

  δηλαδη μετα απο LSS ακολουθει παντα ESH.
- Αν **prev = LPS**:

  current = OLS

  δηλαδη μετα απο LPS επιστρεφουμε σε long frames.

Με βαση τα παραπανω, προκυπτουν:

$$ch0\_type = decide(prev\_frame\_type, next\_esh\_ch0)$$

$$ch1\_type = decide(prev\_frame\_type, next\_esh\_ch1)$$

3. **Συνδυασμος καναλιων:** Τελος, οι δυο τυποι συνδυαζονται με τον πινακα _COMBINE_TABLE:

$$frame\_type = \_COMBINE\_TABLE[(ch0\_type, ch1\_type)]$$

Επιστρεφει: Επιστρεφει ενα string:

$$frame\_type \in \{OLS, LSS, ESH, LPS\}$$

το οποιο ειναι ο τελικος τυπος του τρεχοντος frame.

3 Filterbank

Η βαθμιδα αυτη χρησιμοποιει το μετασχηματισμο Modified Discrete Cosine Transform (MDCT) ο οποιος μετασχηματιζει ενα σημα απο το πεδιο του χρονου στο πεδιο της συχνοτητας. Στο πεδιο της συχνοτητας, οι συντελεστες εχουν μικροτερη μεταξυ τους συσχετιση. Συγκεκριμενα, ο μετασχηματισμος εφαρμοζεται σε διαδοχικα frames-subframes και εκτελει:

$$X_{i,k} = 2\sum_{n=0}^{N-1} x_{i,n} cos(\frac{2\pi}{N}(n+n_0)(k+\frac{1}{2})), \quad 0 \leq k < \frac{N}{2}$$

οπου:

- $x_{i,n}$: τα δειγματα του ηχου στο i-οστο frame
- n: δεικτης δειγματος
- k: δεικτης συχνοτητας
- $n_0 = \frac{\frac{N}{2}+1}{2}$

Ο μετασχηματισμος ειναι πληρως αντιστρεπτος, γι'αυτο και σε αυτο το επιπεδο δεν αναμενουμε μεγαλες απωλειες του αρχικου σηματος στο τελικο. Ο τυπος του αντιστροφου μετασχηματισμου ειναι γραμμενος στην εκφωνηση της εργασιας.

Στην υλοποιηση μου, το αρχειο filterbank περιεχει τις εξης συναρτησεις:

- mdct_matrix: Υπολογιζει το κομματι

  $$cos(\frac{2\pi}{N}(n+n_0)(k+\frac{1}{2}))$$

  του MDCT.
- mdct: Υπολογιζει τον τελικο MDCT
- imdct: Κανει τον αντιστροφο MDCT

- kbd_window: Υλοποιει το παραθυρο Kasser Bessel Derived (KBD)
- frame_window: Βρισκει το παραθυρο ενος συγκεκριμενου frame αναλογα με τον τυπο του frame.
- filter_bank: Πολλαπλασιαζει τα δειγματα ενος frame με το παραθυρο, και εφαρμοζει mdct πανω στο αποτελεσμα.
- i_filter_bank: Πολλαπλασιαζει τους συντελεστες MDCT με το παραθυρο και εφαρμοζει αντιστροφο mdct πανω στο αποτελεσμα.

3.1 mdct_matrix(N)

Η συναρτηση αυτη παιρνει ως ορισμα το πληθος δειγματων του frame.

Οριζει τον δεικτη δειγματος n ως πινακα της μορφης (1,N):

$$n = [0\ 1\ 2\ \dots\ N-1]$$

και το δεικτη συχνοτητας k ως πινακα της μορφης (N//2,1), οπου N//2 το πηλικο της ευκλειδιας διαιρεσης του N με το 2,:

$$k = \begin{bmatrix} 0 \\ 1 \\ 2 \\ \vdots \\ (N//2)-1 \end{bmatrix}$$

Οπως ειπαμε υπολογιζει το κομματι:

$$cos(\frac{2\pi}{N}(n+n_0)(k+\frac{1}{2}))$$

του MDCT. Μεσα στο συνημιτονο, υπαρχουν οι πινακες:

$$n+n_0 = [\,0+n_0\ 1+n_0\ 2+n_0\ \dots\ N-1+n_0\,]$$

$$k + 0.5 = \begin{bmatrix} 0.5 \\ 1.5 \\ 2.5 \\ (N//2) - 0.5 \end{bmatrix}$$

Η πραξη που γινεται μεσα στο συνημιτονο μεταξυ αυτων των 2 πινακων ειναι το Outer Product, δηλαδη αν $N = n + n_0$ και $K = k + 0.5$ τοτε:

$$A = N * K = \begin{bmatrix} N_{11}K_{11} & N_{12}K_{11} & \dots & N_{1N}K_{11} \\ N_{11}K_{21} & N_{12}K_{21} & \dots & N_{1N}K_{21} \\ \vdots & \vdots & \ddots & \vdots \\ N_{11}K_{(N//2)1} & N_{12}K_{(N//2)1} & \dots & N_{1N}K_{(N//2)1} \end{bmatrix} =$$

$$\begin{bmatrix} (0+n_0)0.5 & (1+n_0)0.5 & \dots & (N-1+n_0)0.5 \\ (0+n_0)1.5 & (1+n_0)1.5 & \dots & (N-1+n_0)1.5 \\ (0+n_0)2.5 & (1+n_0)2.5 & \dots & (N-1+n_0)2.5 \\ \vdots & \vdots & \ddots & \vdots \\ (0+n_0)\left(\frac{N}{2}-0.5\right) & (1+n_0)\left(\frac{N}{2}-0.5\right) & \dots & (N-1+n_0)\left(\frac{N}{2}-0.5\right) \end{bmatrix}$$

Επειδη ο N*K ειναι πολ/μος διαστασεων (1,N)*(N/2,1), η Python κανει broadcasting εσωτερικα, το οποιο ουσιαστικα ειναι σαν να υπολογιζει (N/2,1)*(1,N) = (N/2,N).

Ο τελικος πινακας C προκυπτει απο το συνημιτονο του πολ/μου του πινακα A με τη σταθερα $\frac{2\pi}{N}$:

$$C = cos(\frac{2\pi}{N} * A)$$

Καθε γραμμη του C, C[k], ειναι μια συνημιτονικη βαση για το δεικτη συχνοτητας k.

3.2 mdct(x,N)

Η συναρτηση αυτη παιρνει ως ορισματα το frame x και το πληθος N των δειγματων του.

Υπολογιζει τον τελικο μετασχηματισμο MDCT ως $MDCT = 2 * C * x$. Ετσι προκυπτει ενα column-vector X με στοιχεια:

$$X[k] = \sum_{n=0}^{N-1} C[k,n] * x[n], \quad 0 \leq k < \frac{N}{2}$$

διαστασεων $(N/2, N) \times (N,) = (N/2,)$, οπου καθε στοιχειο του ειναι ενας συντελεστης MDCT. Καθε συντελεστης, ειναι ενα εσωτερικο γινομενο μιας συνημιτονικης βασης με τα δειγματα του frame. Επομενως, μετρα την ομοιοτητα των δειγματων με αυτη τη βαση. Επειδη

$$f_k = (k + \frac{1}{2}) \cdot \frac{F_s}{N}$$

οπου:

- F_s: Sampling rate
- N: Μηκος παραθυρου MDCT

ενας συντελεστης MDCT θα μπορουσε να ερμηνευτει ως "ποσο απο τη συχνοτητα f_k υπαρχει σε αυτο το frame".

3.3 kbd_window(N,a)

Η συναρτηση αυτη υλοποιει το παραθυρο KBD η συναρτηση του οποιου δινεται στην εκφωνηση. Η υλοποιηση εγινε με χρηση της συναρτησης kaiser της Python.

3.4 frame_window(frame_type)

Η συανρτηση παιρνει ως ορισμα τον τυπο του frame και υπολογιζει το παραθυρο που αντιστοιχει σε αυτον τον τυπο με βαση τη θεωρια στις σελιδες 127-132 του w2203tfa αρχειου. Στην δικη μας εργασια θα χρησιμοποιουμε παντα σταθερο σχημα παραθυρου. Εγω υλοποιησα και χρησιμοποιησα μονο το KBD window. Επομενως, θα ειμαστε μονιμως στην περιπτωση $window\_shape = 1$ (δηλαδη KBD). Δε θα κανουμε εναλλαγες σχηματος οπως γραφει στο αρχειο. Με βαση τα παραπανω:

Για frame type **OLS** (Only Long Sequence) θα εχουμε:

$$W(n) = \begin{cases} W_{\text{LEFT},2048}(n), & 0 \le n < 1024 \\ W_{\text{KBD\_RIGHT},2048}(n), & 1024 \le n < 2048 \end{cases}$$
$$= W_{\text{KBD},2048}(n), \quad 0 \le n < 2048$$

Για **LSS** (Long Start Sequence):

$$W(n) = \begin{cases} W_{\text{KBD},2048}(n), & 0 \le n < 1024 \\ 1.0, & 1024 \le n < 1472 \\ W_{\text{KBD},256}(n - 1472 + 128), & 1472 \le n < 1600 \\ 0.0, & 1600 \le n < 2048 \end{cases}$$

Για **LPS** (Long Stop Sequence):

$$W(n) = \begin{cases} 0.0, & 0 \le n < 448 \\ W_{\text{KBD},256}(n - 448), & 448 \le n < 576 \\ 1.0, & 576 \le n < 1024 \\ W_{\text{KBD},2048}(n), & 1024 \le n < 2048 \end{cases}$$

Για **ESH** (Eight Short Sequence):

$$W_0(n) = W_1(n) = \cdots = W_7(n) = W_{\text{KBD},256}(n), \quad 0 \le n < 256$$

Στην υλοποιηση μου, για καθε περιπτωση frame, με τη συναρτηση concatenate της numpy, ενωσα τους κλαδους της W(n) διαδοχικα, φτιαχνοντας την τελικη συναρτηση παραθυρου W για αυτο το frame ως ενα διανυσμα 2048 στοιχειων.

3.5 filter_bank(frame_T,frame_type)

Η συναρτηση αυτη παιρνει ως ορισματα τα δειγματα του frame και τον τυπο του frame.

Καλει τη συναρτηση frame_window που επιστρεφει το παραθυρο του frame. Αν ο τυπος του frame δεν ειναι ESH, τοτε απλα πολ/ζει τα δειγματα του frame για καθε καναλι με το παραθυρο, και τα περναει στην συναρτηση MDCT οπου επιστρεφει τους 1024 συντελεστες (μιας και τα frames ειναι παντα μεγεθους 2048 δειγματων). Αν το frame ειναι ESH, τοτε επιλεγονται μονο τα κεντρικα 1152 δειγματα (απο 448 μεχρι 1600, τα αλλα αγνοουνται) τα οποια σπαμε σε 8 επικαλυπτομενα κατα 50% τμηματα-subframes των 256 δειγματων ($256 + 7 \cdot 128 = 1152$. Καθε subframe πολ/ζεται με το αντιστοιχο παραθυρο του (στην περιπτωση μας ολα ειναι ιδια), και το αποτελεσμα μπαινει στην mdct η οποια επιστρεφει τους 128 συντελεστες MDCT.

Σημειωνω οτι παρελειψα να εξηγησω τις i-συναρτησεις καθως κανουν απλα την αντιστροφη διαδικασια.

4 AAC_Coder_1

Το αρχειο aac_coder_1 υλοποιει τον κωδικοποιητη και αποκωδικοποιητη του Level 1. Σε αυτο το επιπεδο η συμπιεση ειναι χωρις κβαντισμο και χωρις εντροπικη κωδικοποιηση, οποτε δεν περιμενουμε μεγαλες απωλειες ποιοτητας.

Η βασικη ροη ειναι:

- Χωριζουμε το σημα σε επικαλυπτομενα frames μηκους 2048 δειγματων με επικαλυψη 50% (hop 1024).
- Για καθε frame αποφασιζουμε frame_type με τη βαθμιδα SSC.
- Εφαρμοζουμε filterbank (MDCT με καταλληλα παραθυρα) και αποθηκευουμε τους συντελεστες.

- Στην αποκωδικοποιηση εφαρμοζουμε iFilterbank (IMDCT με τα ιδια παραθυρα) και ανακατασκευαζουμε το σημα με overlap-add.

Στην υλοποιηση μου, το αρχειο περιεχει τις εξης συναρτησεις:

- prepare_signal: Προσθετει padding στο σημα και φροντιζει να διαιρειται ακριβως σε frames
- aac_coder_1: Κωδικοποιητης Level 1 (SSC + filterbank)
- i_aac_coder_1: Αποκωδικοποιητης Level 1 (iFilterbank + overlap-add + αφαιρεση padding)

4.1 prepare_signal(x)

Η συναρτηση prepare_signal παιρνει ως ορισμα το σημα x στο πεδιο του χρονου, σε μορφη πινακα $(N, 2)$ (δυο καναλια), και επιστρεφει το ιδιο σημα με καταλληλο padding ωστε να μπορει να τεμαχιστει σε frames μηκους 2048 με hop 1024.

1) Padding στην αρχη και στο τελος (1024 δειγματα)

Στην αρχη εφαρμοζεται:

$$x \leftarrow pad(x, (HOP, HOP))$$

δηλαδη προστιθενται $HOP = 1024$ μηδενικα δειγματα στην αρχη και 1024 στο τελος.

Ο λογος ειναι οτι στο AAC χρησιμοποιουμε επικαλυψη 50% (hop = 1024). Το πρωτο frame μηκους 2048 χρειαζεται να εχει και "αριστερο" περιεχομενο-δειγματα. Επειδη ομως στην αρχη του σηματος δεν υπαρχουν προηγουμενα δειγματα, προσθετουμε μηδενικα ωστε:

- το πρωτο frame να ξεκιναει σωστα,
- να μπορει να γινει σωστη παραθυρωση και IMDCT χωρις να χανεται πληροφορια στην αρχη.

Για τον ιδιο λογο κανουμε padding στο τελος. Αρα το νεο μηκος μετα το 1ο padding ειναι:

$$n = n_{initial} + 2048$$

2) Padding στο τελος για ακριβη διαιρεση σε frames

Το πρωτο frame ξεκινα στη θεση 0 και εχει μηκος 2048, αρα καλυπτει τα δειγματα:

$$[0,\ 2048)$$

δηλαδη απο 0 εως και 2047.

Για να υπαρξει δευτερο frame, πρεπει να υπαρχει χωρος για αλλο παραθυρο μηκους 2048. Αν το συνολικο μηκος του σηματος ειναι n, τοτε το τελευταιο frame μπορει να ξεκινησει το πολυ μεχρι τη θεση:

$$n - 2048$$

γιατι αν ξεκινησει πιο μετα, το παραθυρο μηκους 2048 δεν θα χωραει ολοκληρο μεσα στο σημα.

Οι θεσεις εκκινησης των frames ειναι:

$$i \cdot 1024, \quad i = 0, 1, 2, \ldots$$

και θελουμε το i-οστο frame να χωραει ολοκληρο, αρα:

$$i \cdot 1024 + 2048 \leq n$$

ισοδυναμα:

$$i \cdot 1024 \leq n - 2048$$

Αρα:

$$i \leq \frac{n - 2048}{1024}$$

Ο μεγιστος ακεραιος i που ικανοποιει την ανισοτητα ειναι:

$$\left\lfloor \frac{n - 2048}{1024} \right\rfloor$$

Επομενως ο αριθμος των frames ειναι:

$$num\_frames = \left\lfloor \frac{n - 2048}{1024} \right\rfloor + 1$$

Το $+1$ υπαρχει γιατι μετραμε και το frame που ξεκινα στη θεση 0.

Οριζουμε:

$$rem = (n - 2048) \bmod 1024$$

Πρεπει:

$$rem = 0 \tag{1}$$

Αν $rem > 0$, τοτε υπαρχει ακεραιος q ωστε:

$$n - 2048 = q \cdot 1024 + rem$$

με:

$$0 < rem < 1024$$

Padding: Προσθετουμε $1024 - rem$ μηδενικα δειγματα στο τελος, αρα το νεο μηκος γινεται:

$$n' = n + (1024 - rem)$$

Θελουμε να υπολογισουμε το νεο υπολοιπο:

$$(n' - 2048) \bmod 1024$$

Αντικαθιστουμε:

$$n' - 2048 = (n - 2048) + (1024 - rem)$$

Και επειδη:

$$n - 2048 = q \cdot 1024 + rem$$

προκυπτει:

$$n' - 2048 = q \cdot 1024 + rem + 1024 - rem$$

Τα rem απλοποιουνται:

$$n' - 2048 = q \cdot 1024 + 1024 = (q + 1) \cdot 1024$$

Επομενως:

$$((q+1)\cdot 1024) \bmod 1024 = 0$$

γιατι ειναι ακριβες πολλαπλασιο του 1024. Για $n = n_{initial} + 2048$, η συνθηκη (1) γινεται:

$$rem = n_{initial} \bmod 1024 = 0$$

Αν $rem > 0$ ακολουθειται η διαδικασια που ειπαμε παραπανω. Ετσι, με το διπλο αυτο padding, το αρχικο σημα τωρα χωριζεται σε ακεραιο αριθμο frames, ισο με:

$$num\_frames = \lfloor \frac{n_{initial}}{1024} \rfloor + 1$$

Στην υλοποιηση εκανα $num\_frames = \lceil \frac{n_{initial}}{1024} \rceil + 1$ ωστε να δημιουργειται παντα ενα επιπλεον frame (με μηδενικα δειγματα οπου δεν υπαρχει σημα). Ετσι, γλιτωνω τον ελεγχο του υπολοιπου rem.

4.2 aac_coder_1(filename_in)

Η συναρτηση aac_coder_1 διαβαζει ενα wav αρχειο και παραγει μια λιστα aac_seq απο dictionaries, ενα για καθε frame.

Ορισματα

- filename_in: το wav αρχειο εισοδου

Επιστρεφει

- aac_seq: λιστα frames, καθε στοιχειο περιεχει:
  - frame_type: OLS, LSS, ESH, LPS
  - chl["frame_F"]: συντελεστες MDCT αριστερου καναλιου
  - chr["frame_F"]: συντελεστες MDCT δεξιου καναλιου

Διαδικασια

1. Διαβαζει το σημα και το κανει 2D.
2. Καλεί prepare_signal για padding.

3. Υπολογιζει το πληθος frames:

$$\text{num\_frames} = \frac{n - 2048}{1024} + 1$$

4. Για καθε frame i παιρνει:

$$frame\_t = x[i \cdot 1024 : i \cdot 1024 + 2048]$$

5. Υπολογιζει και το **next frame** για τη βαθμιδα SSC. Για το τελευταιο frame βαζει μηδενικο frame.

6. Καλει:

$$frame\_type = SSC(frame\_t, next\_frame\_t, prev\_frame\_type)$$

7. Καλει:

$$frame\_f = filter\_bank(frame\_t, frame\_type)$$

8. Αποθηκευει τους συντελεστες:

   - Αν ESH: τους αναδιατασσει σε $(128, 8)$ ανα καναλι
   - Αλλιως: κρατα $(1024, 1)$ ανα καναλι

4.3 i_aac_coder_1(aac_seq, filename_out)

Η συναρτηση i_aac_coder_1 παιρνει τη λιστα frames aac_seq και ανακατασκευαζει το σημα στο πεδιο του χρονου.

Ορισματα

- aac_seq: λιστα frames (οπως παραχθηκε απο τον κωδικοποιητη)
- filename_out: wav εξοδου

Επιστρεφει

- y: το αποκωδικοποιημενο σημα

1) Ανακατασκευη frame_F απο το αποθηκευμενο format
Για καθε frame δημιουργειται πινακας:

$$frame\_f \in \mathbb{R}^{1024\times 2}$$

- Αν frame_type = ESH, οι συντελεστες $(128, 8)$ καθε καναλιου γινονται flatten με column-major (order="F") για να επανελθουν στη σειρα subframes.
- Αλλιως, οι συντελεστες $(1024, 1)$ γινονται reshape σε $(1024,)$.

2) Επιστροφη στο πεδιο χρονου
Καλει:

$$frame\_t = i\_filter\_bank(frame\_f, frame\_type)$$

και παιρνει frame μηκους 2048 δειγματων.

3) Overlap-add reconstruction

Εδω γινεται το πιο σημαντικο κομματι της ανακατασκευης. Επειδη τα frames ειναι επικαλυπτομενα (50%), καθε frame_t πρεπει να προστεθει στην εξοδο στη σωστη θεση:

$$start = i \cdot HOP = i \cdot 1024$$

$$y[start : start + 2048] \leftarrow y[start : start + 2048] + frame\_t$$

Δηλαδη, το δευτερο μισο του προηγουμενου frame και το πρωτο μισο του επομενου frame καταληγουν να αθροιζονται στο ιδιο σημειο του χρονου.

Αυτο το overlap-add δουλευει σωστα επειδη τα παραθυρα του MDCT/IMDCT εχουν σχεδιαστει ωστε να ικανοποιουν συνθηκη τελειας ανακατασκευης (Princen-Bradley). Ετσι, οταν προστεθουν τα επικαλυπτομενα τμηματα, οι παραθυρωμενες συνιστωσες συμπληρωνουν η μια την αλλη και δεν δημιουργουνται ``σπασιματα'' στα οριa των frames.

4) Αφαιρεση padding

Επειδη στην αρχη ειχαμε προσθεσει 1024 δειγματα στην αρχη και 1024 στο τελος, στο τελος αφαιρουμε:

$$y \leftarrow y[1024 : -1024]$$

Ετσι επιστρεφουμε στο αρχικο μηκος του σηματος (χωρις τα τεχνητα μηδενικα που χρησιμοποιηθηκαν για το framing).

5) Αποθηκευση

Τελος, γινεται εγγραφη του wav:

sf.write(filename_out, y, 48000)

5 AAC_Coder_2 - AAC_Coder_3

Στο Level 1 ο κωδικοποιητης περιλαμβανει μονο τις βαθμιδες:

$$SSC \rightarrow FilterBank$$

και ο αποκωδικοποιητης:

$$iFilterBank \rightarrow Overlap\text{-}Add$$

Στα επομενα επιπεδα προστιθενται επιπλεον βαθμιδες και πληροφορια ανα frame.

Τι προστεθηκε στο aac_coder_2 σε σχεση με το Level 1

Στο Level 2 προστεθηκε η βαθμιδα **TNS (Temporal Noise Shaping)** μετα το FilterBank.

Το pipeline γινεται:

$$SSC \rightarrow FilterBank \rightarrow TNS$$

Συγκεκριμενα προστεθηκαν:

- Κληση της συναρτησης tns(frame_F, frame_type) ανα καναλι.
- Αποθηκευση των tns_coeffs για καθε καναλι στο dictionary του frame.
- Στον αποκωδικοποιητη προστεθηκε η βαθμιδα i_tns πριν το i_filter_bank.

Τι προστεθηκε στο aac_coder_3 σε σχεση με το Level 1

Στο Level 3 υλοποιειται ολο το πληρες pipeline του AAC:

$$SSC \rightarrow FilterBank \rightarrow TNS \rightarrow Psycho \rightarrow Quantizer \rightarrow Huffman$$

Σε σχεση με το Level 1, προστεθηκαν:

- **TNS** (οπως στο Level 2).
- **Psychoacoustic Model**:
  - Υπολογισμος SMR ανα μπαντα.
  - Χρηση των δυο προηγουμενων frames για προβλεψη.
- **Quantizer**:
  - Υπολογισμος scalefactor gains.
  - Παραγωγη κβαντισμενων συμβολων S.
  - Παραγωγη G και sfc.
- **Huffman encoding**:
  - Κωδικοποιηση των S.
  - Κωδικοποιηση των sfc με συγκεκριμενο codebook.
  - Αποθηκευση bitstreams και codebook.

Στον αποκωδικοποιητη προστεθηκαν οι αντιστροφες βαθμιδες:

$$Huffman \rightarrow iQuantizer \rightarrow iTNS \rightarrow iFilterBank \rightarrow Overlap\text{-}Add$$

6 Temporal Noise Shaping (TNS)

Στην κωδικοποιηση με MDCT, το σημα χωριζεται σε frames σταθερου μηκους και καθε frame επεξεργαζεται ξεχωριστα. Το ψυχοακουστικο μοντελο θεωρει οτι μεσα σε καθε frame το σημα δεν αλλαζει πολυ (ειναι περιπου στασιμο).

Στην πραξη ομως αυτο δεν ισχυει παντα. Μπορει μεσα στο ιδιο παραθυρο να εμφανιστει μια αποτομη αλλαγη, οπως ενας δυνατος παλμος (π.χ. ενα χτυπημα τυμπανου).

Ο μετασχηματισμος MDCT "απλωνει" την ενεργεια αυτης της αποτομης αλλαγης σε ολο το frame. Οταν μετα γινει κβαντιση, το σφαλμα κβαντισης επισης απλωνεται χρονικα σε ολο το frame.

Αυτο μπορει να δημιουργησει το φαινομενο του **pre-echo**: δηλαδη να ακουστει θορυβος λιγο πριν απο τον δυνατο παλμο. Αυτο ειναι ενοχλητικο, γιατι το ανθρωπινο αυτι καλυπτει (masking) καλυτερα τον θορυβο μετα απο εναν δυνατο ηχο, αλλα πολυ λιγοτερο πριν απο αυτον. Αρα, αν εμφανιστει θορυβος πριν απο τον παλμο, γινεται πιο ευκολα αντιληπτος.

Το **Temporal Noise Shaping (TNS)** χρησιμοποιειται για να μειωσει αυτο το προβλημα. Η ιδεα ειναι οτι πριν την κβαντιση εφαρμοζεται μια μορφη γραμμικης προβλεψης στους συντελεστες MDCT. Ετσι αφαιρειται η μεταξυ τους συσχετιση και το σημα γινεται πιο "ομοιομορφο" (πιο κοντα σε λευκο θορυβο), ωστε να κβαντιστει πιο αποτελεσματικα.

Στον αποκωδικοποιητη εφαρμοζεται το αντιστροφο φιλτρο για να επανελθει η αρχικη φασματικη μορφη. Ομως το σφαλμα κβαντισης εχει πλεον κατανεμηθει διαφορετικα στον χρονο: δεν συγκεντρωνεται πριν απο τον παλμο, αλλα μεταφερεται σε χρονικες περιοχες οπου καλυπτεται καλυτερα απο το ιδιο το σημα.

Με λιγα λογια, το TNS δεν αλλαζει μονιμα το σημα. Αλλαζει τον τροπο που κατανενεται χρονικα ο θορυβος κβαντισης, ωστε να γινεται λιγοτερο αντιληπτος.

Στην υλοποιηση μου, το αρχειο tns περιεχει τις εξης συναρτησεις:

- load_bands: Φορτωνει απο το TableB219.mat τα band tables για long και short frames.
- get_band_edges: Επιστρεφει τα ορια (w_{low}, w_{high}) των bands

αναλογα με τον τυπο frame.

- tns: Κεντρικη συναρτηση TNS (encoder). Χειριζεται long frames και ESH και επιστρεφει τους φιλτραρισμενους συντελεστες MDCT και τους κβαντισμενους συντελεστες TNS.
- tns_subframe: Υλοποιει το TNS για ενα frame/subframe (κανονικοποιηση, LPC, κβαντισμος, ελεγχος ευσταθειας, φιλτραρισμα).
- is_stable: Ελεγχει την ευσταθεια των συντελεστων προβλεψης μεσω των ριζων του αντιστοιχου πολυωνυμου.
- apply_fir: Εφαρμοζει το FIR φιλτρο TNS στους συντελεστες MDCT.
- i_tns: Αντιστροφη TNS (decoder). Εφαρμοζει το αντιστροφο φιλτρο χρησιμοποιωντας τους αποθηκευμενους συντελεστες.
- apply_iir: Εφαρμοζει το IIR φιλτρο (αντιστροφο του FIR) για την ανακατασκευη των αρχικων συντελεστων.

6.1 load_bands()

Η συναρτηση αυτη φορτωνει τους πινακες bands απο το αρχειο TableB219.mat. Συγκεκριμενα:

- B219a: 69 bands για long frames
- B219b: 42 bands για short frames

Οι πινακες αυτοι περιγραφουν υποζωνες συχνοτητας και χρησιμοποιουνται για να υπολογιστει η ενεργεια ανα band κατα την κανονικοποιηση.

6.2 get_band_edges(frame_type)

Η συναρτηση παιρνει ως ορισμα τον τυπο frame και επιστρεφει δυο πινακες (w_{low}, w_{high}) που περιεχουν τα κατω και ανω ορια (σε δεικτες MDCT) για καθε band.

Για **OLS/LSS/LPS** χρησιμοποιει τα long bands, ενω για **ESH** χρησιμοποιει τα short bands.

6.3 tns(frame_F_in, frame_type)

Η συναρτηση αυτη ειναι η κεντρικη συναρτηση του TNS στον encoder. Παιρνει ως εισοδο τους συντελεστες MDCT πριν το TNS και τον τυπο frame.

Για long frames (**OLS/LSS/LPS**), η εισοδος ειναι διαστασεων $(1024, 1)$. Η συναρτηση μετατρεπει το frame σε μονοδιαστατη ακολουθια, καλει τη _tns_subframe και επιστρεφει:

- frame_F_out: τους φιλτραρισμενους συντελεστες MDCT (ιδιες διαστασεις)
- tns_coeffs: τους κβαντισμενους συντελεστες προβλεψης ταξης $p = 4$ (διαστασεων $(4, 1)$)

Για **ESH**, οι συντελεστες MDCT αντιμετωπιζονται ως 8 subframes. Η εισοδος θεωρειται πινακας $(128, 8)$ (128 συντελεστες ανα subframe). Η συναρτηση εφαρμοζει TNS ανεξαρτητα σε καθε subframe καλωντας 8 φορες τη _tns_subframe και επιστρεφει:

- frame_F_out διαστασεων $(128, 8)$
- tns_coeffs διαστασεων $(4, 8)$

6.4 tns_subframe(X, frame_type)

Η συναρτηση αυτη υλοποιει ολη τη διαδικασια TNS για ενα frame (long) η για ενα subframe (short). Τα βηματα που εκτελει ειναι:

- **Βημα 1: Κανονικοποιηση ανα band (normalization).**
  Υπολογιζεται η ενεργεια καθε band:

$$P[b] = \sum_{k=w_{low}[b]}^{w_{high}[b]} X[k]^2$$

και στη συνεχεια υπολογιζεται ο συντελεστης κανονικοποιησης $S_w[k]$:

$$S_w[k] = \sqrt{P[b]}, \quad k \in [w_{low}[b], w_{high}[b]]$$

(αν $P[b] = 0$ τοτε τιθεται $S_w[k] = 1$). Μετα απο αυτο, εφαρμοζεται εξομαλυνση ωστε το $S_w[k]$ να μην αλλαζει αποτομα μεταξυ bands. Τελος, οι κανονικοποιημενοι συντελεστες ειναι:

$$X_w[k] = \frac{X[k]}{S_w[k] + \epsilon}$$

με ϵ μικρη σταθερα για αποφυγη διαιρεσης με το μηδεν.

- **Βημα 2: Υπολογισμος LPC συντελεστων (ταξη $p = 4$).**
  Υπολογιζεται η αυτοσυσχετιση της X_w και κρατιουνται οι τιμες $r[0], r[1], \ldots, r[p]$. Στη συνεχεια σχηματιζεται Toeplitz πινακας R και λυνεται το συστημα:

$$R\,a = r_{1:p}$$

  ωστε να προκυψουν οι συντελεστες $a_1, \ldots, a_p$. Για αριθμητικη σταθεροτητα προστεθηκε μικρη regularization διαγωνια.

- **Βημα 3: Κβαντισμος των συντελεστων προβλεψης.**
  Οι συντελεστες κβαντιζονται με βημα 0.1:

$$a_q = \text{round}\left(\frac{a}{0.1}\right) \cdot 0.1$$

  και περιοριζονται στο διαστημα $[-0.8, 0.7]$. Οι κβαντισμενοι συντελεστες ειναι αυτοι που αποθηκευονται και μεταδιδονται στον decoder.

- **Βημα 4: Ελεγχος ευσταθειας.**
  Γινεται ελεγχος ευσταθειας στους κβαντισμενους συντελεστες. Αν το φιλτρο δεν ειναι ευσταθες, τοτε οι συντελεστες μηδενιζονται.

- **Βημα 5: Εφαρμογη FIR φιλτρου στο αρχικο X.**
  Με τους κβαντισμενους συντελεστες a_q εφαρμοζεται FIR φιλτρο:

  $$H_{\text{TNS}}(z) = 1 - a_1 z^{-1} - a_2 z^{-2} - \cdots - a_p z^{-p}$$

  και το αποτελεσμα Y αποτελει τους συντελεστες MDCT μετα το TNS.

Η συναρτηση επιστρεφει τους φιλτραρισμενους συντελεστες και τους κβαντισμενους συντελεστες προβλεψης.

6.5 is_stable(a)

Η συναρτηση αυτη ελεγχει αν το προβλεπτικο φιλτρο ειναι ευσταθες. Στην υλοποιηση μου σχηματιζεται ενα πολυωνυμο απο τους συντελεστες a και υπολογιζονται οι ριζες του. Αν ολες οι ριζες βρισκονται εντος του μοναδιακου δισκου $|z| < 1$, τοτε το φιλτρο θεωρειται ευσταθες.

Αν για οποιονδηποτε λογο αποτυχει ο υπολογισμος ριζων, η συναρτηση επιστρεφει False.

6.6 apply_fir(X, a)

Η συναρτηση αυτη εφαρμοζει το FIR φιλτρο του TNS στους συντελεστες MDCT.

Το φιλτρο οριζεται στο πεδιο του μετασχηματισμου Z ως:

$$H_{\text{TNS}}(z) = 1 - a_1 z^{-1} - a_2 z^{-2} - \cdots - a_p z^{-p}$$

Στην υλοποιηση μας χρησιμοποιουμε φιλτρο ταξης $p = 4$, αρα:

$$H_{\text{TNS}}(z) = 1 - a_1 z^{-1} - a_2 z^{-2} - a_3 z^{-3} - a_4 z^{-4}$$

Απο τον ορισμο του γραμμικου χρονικα αμεταβλητου φιλτρου ισχυει:

$$H(z) = \frac{Y(z)}{X(z)}$$

Αρα:

$$Y(z) = H(z)X(z)$$

και αντικαθιστωντας το $H_{\text{TNS}}(z)$:

$$Y(z) = X(z)\left(1 - \sum_{\ell=1}^{p} a_\ell z^{-\ell}\right)$$

Αναπτυσσοντας:

$$Y(z) = X(z) - \sum_{\ell=1}^{p} a_\ell z^{-\ell} X(z)$$

Παιρνοντας αντιστροφο μετασχηματισμο Z, και γνωριζοντας οτι

$$z^{-\ell}X(z) \longleftrightarrow X(k-\ell),$$

προκυπτει η σχεση στο διακριτο πεδιο:

$$Y(k) = X(k) - \sum_{\ell=1}^{p} a_\ell X(k-\ell)$$

Για $p = 4$:

$$Y(k) = X(k) - a_1X(k-1) - a_2X(k-2) - a_3X(k-3) - a_4X(k-4)$$

Παρατηρουμε οτι η εξοδος του φιλτρου Y ειναι το σφαλμα $\mathbf{e(k) = X[k]}$−προβλεψη, δηλαδη το TNS χρησιμοποιει prediction error filter. Το φιλτρο εφαρμοζεται σε καθε συντελεστη MDCT $X(k)$ της ακολουθιας, και ετσι προκυπτει μια νεα ακολουθια φιλτραρισμενων συντελεστων $Y(k)$ ιδιου μηκους. Αυτη η ακολουθια ειναι που τελικα μπαινει στον κβαντιστη.

Οπως φαινεται απο τη μορφη του στο πεδιο Z, το φιλτρο της εργασιας μας περιεχει παραγοντες μεχρι z^{-4}. Ο εκθετης στο $z^{-\ell}$ δηλωνει καθυστερηση ℓ δειγματων. Επομενως το φιλτρο εχει μνημη $p = 4$, δηλαδη καθε δειγμα εξοδου $Y(k)$ εξαρταται μονο απο τα $X(k)$, $X(k-1)$, $X(k-2)$, $X(k-3)$ και $X(k-4)$.

Για $k < 4$, η σχεση εφαρμοζεται χρησιμοποιωντας μονο τους ορους που οριζονται (δηλαδη για $k = 0$ δεν υπαρχει καθυστερηση, για $k = 1$ χρησιμοποιειται μονο ο ορος $X(k-1)$ κ.ο.κ.), ωστε το φιλτρο να ειναι αιτιατο.

6.7 i_tns(frame_F_in, frame_type, tns_coeffs)

Η συναρτηση αυτη υλοποιει την αντιστροφη διαδικασια στον decoder. Παιρνει ως εισοδο τους συντελεστες MDCT μετα το TNS και τους αποθηκευμενους κβαντισμενους συντελεστες tns_coeffs.

Για long frames, εφαρμοζεται το αντιστροφο φιλτρο σε ολη την ακολουθια των 1024 συντελεστων και επιστρεφεται πινακας $(1024, 1)$.

Για **ESH**, η εισοδος περιεχει 8 subframes, και η συναρτηση εφαρμοζει το αντιστροφο φιλτρο ανεξαρτητα σε καθε subframe χρησιμοποιωντας τους αντιστοιχους συντελεστες $(4, 8)$.

6.8 apply_iir(Y, a)

Η συναρτηση αυτη εφαρμοζει το αντιστροφο IIR φιλτρο του TNS. Επειδη στον encoder εφαρμοστηκε:

$$Y[k] = X[k] - \sum_{\ell=1}^{p} a[\ell]\, X[k-\ell],$$

στον decoder ανακατασκευαζουμε το X αναδρομικα ως:

$$X[k] = Y[k] + \sum_{\ell=1}^{p} a[\ell]\, X[k-\ell].$$

Η αναδρομηση ξεκινα απο $k = 0$ και για καθε k αθροιζονται μονο οροι με $\ell \leq k$. Το αποτελεσμα ειναι οι συντελεστες MDCT πριν την εφαρμογη TNS.

Συμπεραινοντας, το TNS υπολογιζει τους συντελεστες a_l ετσι ωστε:

$$X[k] \approx \sum_{l=1}^{p} a_l X[k-l]$$

Ψαχνει δηλαδη τη γραμμικη εξαρτηση του καθε συντελεστη απο τους προηγουμενους, αφαιρει αυτο το κομματι (προβλεψη) απο το αρχικο σημα, και δινει στον κβαντιστη το υπολοιπο (error). Ετσι οι συντελεστες γινονται λιγοτερο συσχετισμενοι - πιο "λευκοι", και περιοριζεται το φαινομενο pre-echo που ειπαμε, δηλαδη ο θορυβος να ακουγεται πριν την αποτομη αλλαγη στο σημα, οποτε ο θορυβος γινεται λιγοτερο αντιληπτος.

7 Psychoacoustic Model

Η βαθμιδα αυτη υλοποιει το Ψυχοακουστικο Μοντελο του AAC. Σκοπος της ειναι να υπολογισει για καθε ζωνη συχνοτητας το κατωφλι ακουστοτητας του θορυβου κβαντισμου, δηλαδη ποση παραμορφωση μπορει να προστεθει χωρις να γινει αντιληπτη απο το ανθρωπινο αυτι.

Η βασικη ιδεα ειναι οτι το ανθρωπινο συστημα ακοης δεν εχει ιδια ευαισθησια σε ολες τις συχνοτητες και οτι ισχυρα φασματικα συστατικα μπορουν να καλυψουν ασθενεστερα (masking). Για τον λογο αυτο το μοντελο υπολογιζει ενα κατωφλι ενεργειας ανα μπαντα συχνοτητας και τελικα το Signal to Mask Ratio (SMR), το οποιο καθοδηγει τον κβαντιστη.

Η διαδικασια βασιζεται σε:

- Μετασχηματισμο FFT καθε frame ή subframe
- Υπολογισμο προβλεψιμοτητας απο τα δυο προηγουμενα παραθυρα
- Υπολογισμο ενεργειας ανα μπαντα
- Εφαρμογη spreading function
- Υπολογισμο tonality index
- Υπολογισμο κατωφλιου masking
- Υπολογισμο SMR

Στην υλοποιηση μου, το αρχειο psycho περιεχει τις εξης συναρτησεις:

- load_bands: Φορτωνει τους πινακες μπαντων απο το αρχειο TableB219.mat
- spreading_fun: Υπολογιζει την spreading function μεταξυ δυο μπαντων
- precompute_spreading: Προϋπολογιζει τον πινακα spreading για ταχυτητα
- psycho: Κυρια συναρτηση του ψυχοακουστικου μοντελου
- psycho_subframe: Υλοποιει ολο τον αλγοριθμο για ενα frame ή subframe

7.1 load_bands()

Η συναρτηση αυτη φορτωνει απο το αρχειο TableB219.mat τους πινακες B219a και B219b που αντιστοιχουν στις μπαντες του ψυχοακουστικου μοντελου για:

- Long frames (69 μπαντες)
- Short frames (42 μπαντες)

Καθε γραμμη του πινακα περιεχει πληροφορια για μια μπαντα:

- wlow: κατω οριο FFT index
- whigh: ανω οριο FFT index
- bval: κεντρικη συχνοτητα
- qsthr: κατωφλι σε ησυχια

7.2 spreading_fun(i,j,bval)

Η spreading function υπολογιζει τον συντελεστη εξασθενισης ο οποιος δειχνει ποσο η ενεργεια ενος ηχου σε μια μπαντα συχνοτητων i μασκαρει (καλυπτει) τον ηχο σε αλλες μπαντες συχνοτητων j.

Με αλλα λογια, δειχνει ποσο ανεβαινει το κατωφλι ακουστοτητας της μπαντας j λογω της ενεργειας της μπαντας i, δηλαδη ποσο δυσκολα θα ακουστει κατι στη μπαντα j λογω του ηχου στη μπαντα i.

Ο υπολογισμος γινεται με βαση τις εξισωσεις της εκφωνησης:

$$tmpx = \begin{cases} 3(bval_j - bval_i), & i \geq j \\ 1.5(bval_j - bval_i), & i < j \end{cases}$$

Ακολουθουν οι υπολογισμοι tmpz και tmpy και τελικα:

$$x = 10^{\frac{tmpz+tmpy}{10}}$$

αν tmpy < -100 τοτε x=0.
Το αποτελεσμα εκφραζει τον συντελεστη εξασθενισης.

7.3 precompute_spreading()

Η συναρτηση precompute_spreading υπολογιζει τον πινακα spreading διαστασεων $n\_bands \times n\_bands$ (69×69 για long frame και 42×42 για short frame), οπου το στοιχειο $[i][j]$ ισουται με την τιμη της spreading function για τη μπαντα i ως προς τη μπαντα j, δηλαδη:

$$\text{SPREADING}[i][j] = \text{spreading\_fun}(i, j).$$

Για να υπολογισουμε τη μασκα της μπαντας j, δηλαδη το κατωφλι ακουστοτητας της, υπολογιζουμε τον γραμμικο συνδυασμο:

$$\text{MASK}(j) = \sum_i E(i) \cdot \text{spreading}(i, j),$$

οπου $E(i)$ ειναι η ενεργεια της μπαντας i.

Οσο μεγαλυτερη ειναι η τιμη MASK(j), τοσο μεγαλυτερο ειναι το κατωφλι ακουστοτητας της μπαντας j. Αυτο σημαινει οτι ο κωδικοποιητης μπορει να επιτρεψει μεγαλυτερο θορυβο κβαντισης στη μπαντα j, αφου δε θα ακουστει τοσο ευκολα.

Αρα, μπορει να κανει πιο χοντρη κβαντιση (λιγοτερα bits), και πετυχαινει καλυτερη συμπιεση χωρις αντιληπτη υποβαθμιση του σηματος.

Το τελικο κατωφλι ακουστοτητας της μπαντας j ειναι, σε απλοποιημενη μορφη, ουσιαστικα:

$$Threshold(j) = \max\big(MaskingThreshold(j),\, qsthr(j)\big),$$

οπου $qsthr(j)$ ειναι το φυσικο (absolute) κατωφλι ακουστοτητας του ανθρωπινου αυτιου για τη συγκεκριμενη μπαντα συχνοτητων j.

7.4 psycho(frame_T, frame_type, frame_T_prev_1, frame_T_prev_2)

Η συναρτηση αυτη ειναι η κυρια συναρτηση του ψυχοακουστικου μοντελου.

Παιρνει ως ορισματα:

- frame_T: το τρεχον frame στο πεδιο του χρονου
- frame_type: τον τυπο του frame (OLS, LSS, LPS, ESH)
- frame_T_prev_1: το προηγουμενο frame
- frame_T_prev_2: το προ-προηγουμενο frame

Επιστρεφει:

- SMR: πινακα διαστασεων 69x1 για long frames ή 42x8 για ESH

Για ESH frames:

- Επιλεγονται τα κεντρικα 1152 δειγματα (448-1600)

- Χωριζονται σε 8 επικαλυπτομενα κατα 50% subframes των 256 δειγματων
- Καλειται η _psycho_subframe για καθε subframe

Για long frames:

- Καλειται μια φορα η _psycho_subframe για ολοκληρο το frame

7.5 _psycho_subframe(s, s_prev1, s_prev2, frame_type)

Η συναρτηση αυτη υλοποιει ολοκληρο τον αλγοριθμο για ενα frame ή subframe.

1. Παραθυρωση και FFT

Καθε σημα πολλαπλασιαζεται με παραθυρο Hann:

$$w(n) = 0.5 - 0.5\cos\left(\frac{\pi(n+0.5)}{N}\right)$$

Στη συνεχεια υπολογιζεται ο FFT και κρατουνται τα N/2 θετικα φασματικα δειγματα:

$$r(w) = |FFT|, \quad f(w) = \angle FFT$$

2. Προβλεψη απο προηγουμενα παραθυρα

$$r_{pred} = 2r_{prev1} - r_{prev2}$$
$$f_{pred} = 2f_{prev1} - f_{prev2}$$

3. Υπολογισμος προβλεψιμοτητας

$$c(w) = \frac{\sqrt{(r\cos f - r_{pred}\cos f_{pred})^2 + (r\sin f - r_{pred}\sin f_{pred})^2}}{r + |r_{pred}|}$$

4. Ενεργεια ανα μπαντα

$$e(b) = \sum r(w)^2$$

$$c(b) = \sum c(w) r(w)^2$$

5. Εφαρμογη spreading function

$$ecb(b) = \sum_{bb=0}^{N_B-1} e(bb)\,\text{spreading\_function}(bb, b)$$

$$ct(b) = \sum_{bb=0}^{N_B-1} c(bb)\,\text{spreading\_function}(bb, b)$$

6. Tonality index

$$cb(b) = \frac{ct(b)}{ecb(b)}$$

$$en(b) = \frac{ecb(b)}{\sum_{bb=0}^{N_B-1} \text{spreading\_function}(bb, b)}$$

$$t_b(b) = -0.299 - 0.43 \ln(cb(b))$$

7. Υπολογισμος SNR

$$SNR(b) = t_b(b) \cdot 18 + (1 - t_b(b)) \cdot 6$$

$$bc(b) = 10^{\frac{-SNR(b)}{10}}$$

$$nb(b) = en(b) \cdot bc(b)$$

$$npart = \max(nb(b), \hat{q}_{thr}(b))$$

8. Signal to Mask Ratio

$$SMR(b) = \frac{e(b)}{npart(b)}$$

Το SMR αποτελει την εξοδο της συναρτησης και χρησιμοποιειται στη βαθμιδα κβαντισης.

8 Quantizer

Η βαθμιδα αυτη υλοποιει την κβαντιση και αποκβαντιση των συντελεστων MDCT. Σκοπος της ειναι η συμπιεση του σηματος με ελεγχο του ακουστου σφαλματος, χρησιμοποιωντας την πληροφορια του ψυχοακουστικου μοντελου μεσω του δεικτη SMR (Signal to Mask Ratio).

Η βασικη ιδεα ειναι οτι για καθε κριτικη μπαντα συχνοτητας επιτρεπεται διαφορετικο ποσοστο σφαλματος κβαντισης. Ο κβαντιστης ρυθμιζει τους συντελεστες scale factor $a(b)$ ωστε η ισχυς του σφαλματος κβαντισης $P_e(b)$ να μην υπερβαινει το κατωφλι $T(b) = \frac{P(b)}{SMR(b)}$.

Στην υλοποιηση μου, το αρχειο quantizer περιεχει τις εξης συναρτησεις:

- get_last_quantizer_stats: Επιστρεφει τις τελευταιες debug μετρικες του κβαντιστη.
- load_bands: Φορτωνει τους πινακες μπαντων απο το αρχειο TableB219.mat.
- quantize_mdct: Υλοποιει τον μη ομοιομορφο κβαντιστη MDCT.
- dequantize_mdct: Υλοποιει την αποκβαντιση των συντελεστων MDCT.
- quantize_subframe: Υλοποιει ολο τον αλγοριθμο κβαντισης για ενα frame ή subframe.
- dequantize_subframe: Κανει την αντιστροφη διαδικασια για ενα frame ή subframe.
- aac_quantizer: Υλοποιει ολο το pipeline απο εισοδο wav μεχρι decoded σημα για καθε frame.
- i_aac_quantizer: Αντιστροφη του aac_quantizer.

8.1 get_last_quantizer_stats()

Η συναρτηση αυτη επιστρεφει ενα αντιγραφο του global dictionary `LAST_DEBUG_STATS`. Περιλαμβανει πληροφοριες οπως τις ελαχιστες και μεγιστες τιμες των $a(b)$, τον αριθμο επαναληψεων, τον τροπο τερματισμου (Converged λογω Pe ή Δα ή max_iter), καθως και μετρικες για SMR, $P(b)$, $T(b)$ και $P_e(b)$.

8.2 load_bands()

Η συναρτηση φορτωνει απο το αρχειο TableB219.mat τους πινακες B219a και B219b που αντιστοιχουν στις 69 μπαντες long frame και στις 42 μπαντες short frame αντιστοιχα.

Για καθε μπαντα οριζονται τα ορια δεικτων $w_{low}(b)$ και $w_{high}(b)$, τα οποια καθοριζουν ποιοι συντελεστες MDCT ανηκουν σε καθε κριτικη μπαντα.

8.3 quantize_mdct(X, alpha)

Η συναρτηση αυτη υλοποιει τον μη ομοιομορφο κβαντιστη των συντελεστων MDCT.

Παιρνει ως ορισματα:

- X: συντελεστες MDCT ενος frame ή subframe (στην πραγματικοτητα οι συντελεστες Y που ειναι οι φιλτραρισμενοι MDCT, δηλαδη αυτους που δινει στην εξοδο το TNS)
- α: scale factor της μπαντας

Υπολογιζεται αρχικα ο παραγοντας κλιμακωσης:

$$scale = 2^{-a/4}$$

Στη συνεχεια οι συντελεστες X(k) κβαντιζονται ως:

$$S(k) = sign(X(k)) \cdot \left\lfloor |X(k) \cdot 2^{-a/4}|^{0.75} + 0.4054 \right\rfloor$$

Και η αποκβαντιση ειναι:

$$\hat{X}(k) = sign(S(k)) \; |S(k)|^{4/3} \; 2^{a/4}$$

Το αποτελεσμα ειναι οι ακεραιοι κβαντισμενοι συντελεστες $S(k)$. Ο κβαντιστης λεγεται μη ομοιομορφος λογω της παρουσιας του εκθετη 0.75. Οσο μεγαλωνει το α, η κβαντιση γινεται πιο χοντρη, δηλαδη μεγαλωνει το βημα κβαντισης. Αυτο φαινεται ως εξης:

$$\hat{X}(k) = sign(S(k))\ |S(k)|^{4/3}\ 2^{a/4}$$

$$\Delta x \approx\ |\frac{d\hat{X}}{dS}| = |\frac{d}{dS}(sign(S)\,|S|^{4/3}\,2^{a/4})| = \frac{4}{3}|S|^{1/3}\,2^{a/4}$$

$$S(k) = sign(X(k))\lfloor |X(k)\,2^{-a/4}|^{3/4} + 0.4054 \rfloor \approx sign(X)|X\,2^{-a/4}|^{3/4}$$

$$|S| \approx (|X|\,2^{-a/4})^{3/4} = |X|^{3/4}\,2^{-3a/16}$$

$$|S|^{1/3} \approx (|X|^{3/4}\,2^{-3a/16})^{1/3} = |X|^{1/4}\,2^{-a/16}$$

$$\Delta x \approx \frac{4}{3}|S|^{1/3}2^{a/4} \approx \frac{4}{3}(|X|^{1/4}\,2^{-a/16})2^{a/4} = \frac{4}{3}|X|^{1/4}\,2^{\,a/4-a/16}$$

$$\boxed{\Delta x \approx \frac{4}{3}|X|^{1/4}\,2^{3a/16}}$$

Επειδη $2^{3a/16}$ ειναι γνησιως αυξουσα συναρτηση του a, προκυπτει οτι Δx αυξανεται οταν αυξανεται το a. Οσο λοιπον αυξανεται το α, το $S(k)$ μικραινει ολο και περισσοτερο. Αμα παρουν αρκετα μεγαλες τιμες, τοτε $S(k) \to 0$, δηλαδη τα $X(k)$ σπρωχονται-κβαντιζονται προς τη σταθμη 0 του κβαντιστη. Τοτε εχουμε το μεγιστο σφαλμα κβαντισμου, δηλαδη:

$$a \to \infty \Rightarrow S(k) \to 0 \Rightarrow \hat{X}(k) \to 0 \Rightarrow$$

$$\Rightarrow P_e = \sum (X(k) - \hat{X}(k))^2 \to X(k)^2 = P_{e,max}$$

Η συναρτηση επιστρεφει τα κβαντισμενα $S(k)$.

8.4 dequantize_mdct(S, alpha)

Η συναρτηση αυτη υλοποιει την αποκβαντιση των συντελεστων MDCT.

Εφαρμοζεται ο τυπος:

$$\hat{X}(k) = sign(S(k)) \cdot |S(k)|^{4/3} \cdot 2^{a/4}$$

Επιστρεφεται η αποκβαντισμενη τιμη $\hat{X}(k)$.

8.5 quantize_subframe(X, frame_type, SMR)

Η συναρτηση αυτη υλοποιει τον κυριο αλγοριθμο κβαντισης για ενα frame ή subframe.

Παιρνει ως ορισματα:

- X: συντελεστες MDCT (στην πραγματικοτητα οι συντελεστες Y που ειναι οι φιλτραρισμενοι MDCT, δηλαδη αυτους που δινει στην εξοδο το TNS)
- frame_type: τυπος frame (OLS, LSS, LPS, ESH)
- SMR: Signal to Mask Ratio ανα μπαντα, υπολογισμενο απο το psychoacoustic model

Επιστρεφει:

- S: κβαντισμενους συντελεστες
- sfc: scale factors σε μορφη DPCM
- G: global gain

1. Υπολογισμος ενεργειας και κατωφλιου

Για καθε μπαντα υπολογιζεται:

$$P(b) = \sum_{k=w_{low}(b)}^{w_{high}(b)} X(k)^2$$

και

$$T(b) = \frac{P(b)}{SMR(b)}$$

2. Αρχικη εκτιμηση των scale factors

Οριζεται μια αρχικη τιμη:

$$\hat{a} = \frac{16}{3} \log_2 \left(\frac{(\max |X|)^{0.75}}{8191} \right)$$

και τιθεται $a(b) = \hat{a}$ για ολες τις μπαντες.

3. Επαναληπτικη αυξηση των a

Σε καθε επαναληψη:

- γινεται κβαντιση και αποκβαντιση
- υπολογιζεται η ισχυς σφαλματος

$$P_e(b) = \sum_{k=w_{low}(b)}^{w_{high}(b)} (X(k) - \hat{X}(k))^2$$

Αν $P_e(b) < T(b)$ γινεται αυξηση του $a(b)$ κατα 1, αν ικανοποιεται ο περιορισμος:

$$|a(b+1) - a(b)| \leq 60$$

για τις 2 γειτονικες μπαντες $b-1$, $b+1$ της μπαντας b. Η διαδικασια τερματιζει οταν δεν μπορει να αυξηθει κανενα $a(b)$ ή οταν ο αλγοριθμος φτασει στο max_iter.

4. DPCM κωδικοποιηση

Μετα την τελικη τιμη των $a(b)$:

$$G = a(0)$$

$$sfc(b) = a(b) - a(b-1)$$

8.6 dequantize_subframe(S, sfc, G, frame_type)

Η συναρτηση αυτη κανει την αντιστροφη διαδικασια.

Αρχικα ανακατασκευαζει τα $a(b)$ απο το DPCM:

$$a(0) = G, \quad a(b) = a(b-1) + sfc(b)$$

Στη συνεχεια εφαρμοζει την dequantize_mdct ανα μπαντα και επιστρεφει τους αποκβαντισμενους συντελεστες.

8.7 aac_quantizer(frame_F, frame_type, SMR)

Η συναρτηση αυτη διαχειριζεται την κβαντιση για ολοκληρο frame.

Για ESH frames:

- χωριζει σε 8 subframes

- καλει την quantize_subframe για καθε subframe
- ενωνει τα αποτελεσματα

Για long frames:

- καλει απλα μια φορα την quantize_subframe

8.8 i_aac_quantizer(S, sfc, G, frame_type)

Η συναρτηση αυτη υλοποιει την αποκαντιση σε επιπεδο frame.
Για ESH frames αποκβαντιζει καθε subframe ξεχωριστα, ενω για long frames καλει μια φορα την dequantize_subframe και επιστρεφει τους αποκβαντισμενους συντελεστες.

9 AAC_Coder_3

Το αρχειο αυτο υλοποιει τον κωδικοποιητη και τον αποκωδικοποιητη του AAC Level 3. Ουσιαστικα υλοποιει ολες τις βαθμιδες frame-by-frame.

Encoder:

$$x[n]\ \text{(WAV, time)} \xrightarrow[\text{frame blocking}]{\text{SSC}} \{x_i[n]\}_i,\ \text{frame\_type}_i$$

$$\{x_i[n]\}_i \xrightarrow[\text{MDCT + window}]{\text{FilterBank}} X_i(k) \xrightarrow[\text{prediction filter}]{\text{TNS}} Y_i(k)$$

$$Y_i(k) \xrightarrow[\text{masking model}]{\text{Psycho}} SMR_i(b) \xrightarrow{\text{Quantizer}} \big(S_i(k),\ sfc_i(b),\ G_i\big)$$

$$\big(S_i(k),\ sfc_i(b),\ G_i\big) \xrightarrow[\text{entropy coding}]{\text{Huffman}} \text{bitstream}_i \xrightarrow[\text{store}]{\text{.mat}} \text{AAC\_seq}_3$$

Decoder:

$$\text{AAC\_seq}_3 \xrightarrow[\text{load}]{\text{.mat}} \text{bitstream}_i \xrightarrow[\text{entropy decoding}]{\text{iHuffman}} \big(S_i(k),\ sfc_i(b),\ G_i\big)$$

$$\big(S_i(k),\ sfc_i(b),\ G_i\big) \xrightarrow[\text{dequantization}]{\text{iQuantizer}} \hat{Y}_i(k) \xrightarrow[\text{inverse filter}]{\text{iTNS}} \hat{X}_i(k)$$

$$\hat{X}_i(k) \xrightarrow[\text{iMDCT + window}]{\text{iFilterBank}} \hat{x}_i[n] \xrightarrow[\text{50\% overlap-add}]{\text{Overlap-Add}} \hat{x}[n]\ \text{(WAV, time)}$$

Ο κωδικοποιητης δημιουργει ενα αντικειμενο aac_seq_3, δηλαδη μια λιστα απο dictionaries, οπου για καθε frame αποθηκευονται:

- ο τυπος frame (OLS, LSS, LPS, ESH)
- για καθε καναλι (chl, chr): το huffman bitstream των S, το codebook, το huffman bitstream των scalefactors sfc, το global gain G, και οι συντελεστες TNS

Επιπλεον, αποθηκευεται και ενα CSV αρχειο με μετρικες ανα frame (bits, codebook, non-zero S, αριθμος επαναληψεων quantizer).

Στην υλοποιηση μου, το αρχειο περιεχει τις εξης συναρτησεις:

- aac_coder_3: Κωδικοποιει ενα WAV αρχειο σε AAC Level 3 και επιστρεφει τη δομη aac_seq_3.
- i_aac_coder_3: Αποκωδικοποιει τη δομη aac_seq_3 και παραγει το ανακατασκευασμενο WAV αρχειο.

9.1 aac_coder_3(filename_in, filename_aac_coded)

Η συναρτηση αυτη υλοποιει τον κωδικοποιητη AAC Level 3. Διαβαζει το αρχειο ηχου, το σπαει σε επικαλυπτομενα frames και κωδικοποιει καθε frame ξεχωριστα.

1. Φορτωση και παραμετροι

Διαβαζεται το WAV αρχειο (στερεο) και οριζονται:

- $N = 2048$ (frame_size)
- hop_size = 1024 (50% overlap)

Επισης φορτωνεται ο πινακας Huffman codebooks (LUT) με τη συναρτηση load_LUT().

2. Padding και αριθμος frames

Υπολογιζεται ο αριθμος frames που απαιτουνται ωστε να καλυφθει ολο το σημα. Αν τα τελευταια δειγματα δεν επαρκουν για πληρες frame, γινεται padding με μηδενικα δειγματα.

3. Μεταβλητες καταστασης

Ο κωδικοποιητης κραταει μνημη απο:

- τον προηγουμενο τυπο frame (prev_frame_type)

- τα 2 προηγουμενα frames στο πεδιο του χρονου (prev_frame_T), τα οποια χρειαζονται στο psychoacoustic model (predictability)

4. Επεξεργασια ανα frame

Για καθε frame:

- Εξαγεται το τρεχον frame $frame\_T$ μηκους 2048.
- Εξαγεται και το επομενο frame $next\_frame\_T$ (η μηδενικο αν δεν υπαρχει) για να γινει η επιλογη τυπου frame απο το SSC.
- Υπολογιζεται ο τυπος frame (OLS, ESH, LSS, LPS) με τη συναρτηση SSC(frame_T, next_frame_T, prev_frame_type).
- Δημιουργειται ενα dictionary για το frame με πεδια για τα δυο καναλια.

5. Επεξεργασια ανα καναλι

Για καθε καναλι (chl, chr) εκτελουνται διαδοχικα τα βηματα:

5a. Filter Bank (MDCT)

Εφαρμοζεται η βαθμιδα filter bank ωστε να παρουμε τους συντελεστες MDCT. Για ESH frames, γινεται reshape σε 8 subframes (128x8).

5b. TNS

Εφαρμοζεται το Temporal Noise Shaping. Επιστρεφονται οι φιλτραρισμενοι συντελεστες (που θα ειναι εισοδος στον quantizer) και οι συντελεστες a_1, a_2, a_3, a_4 του φιλτρου TNS, οι οποιοι αποθηκευονται ωστε να χρησιμοποιηθουν στον αποκωδικοποιητη.

5c. Psychoacoustic Model

Υπολογιζεται το SMR για το καναλι, με εισοδο:

- το τρεχον frame στο χρονο
- τον τυπο frame
- τα 2 προηγουμενα frames για predictability

Το SMR αποθηκευεται στο dictionary του καναλιου.

5d. Quantizer

Κβαντιζονται οι TNS συντελεστες με τη συναρτηση aac_quantizer. Παραγονται:

- S: οι κβαντισμενοι συντελεστες
- sfc: τα DPCM scalefactors
- G: το global gain

Το G αποθηκευεται, ενω επιπλεον γινεται εκτυπωση debug πληροφοριων ανα τακτα διαστηματα (SMR, P, T, Pe, alpha, bits) για long frames, χρησιμοποιωντας τη get_last_quantizer_stats().

5e. Huffman Encoding

Κωδικοποιουνται ξεχωριστα:

- Τα scalefactors sfc με force_codebook = 11 (οπως ζητειται), παραγοντας ενα bitstream sfc_stream.
- Οι κβαντισμενοι συντελεστες S με αυτοματη επιλογη codebook με βαση το maxAbs, παραγοντας bitstream stream και το codebook.

Τα bitstreams και το codebook αποθηκευονται στο dictionary του καναλιου.

6. Logs μετρικων

Για καθε frame και καναλι, καταγραφονται σε CSV:

- bits_S, bits_sfc
- nonzero (πληθος μη μηδενικων S)
- maxAbs (μεγιστη απολυτη τιμη S)
- codebook που επιλεχθηκε
- αριθμος επαναληψεων του quantizer

7. Ενημερωση καταστασης

Στο τελος καθε frame ενημερωνονται:

- prev_frame_type
- τα 2 προηγουμενα frames prev_frame_T

και προστιθεται το dictionary του frame στη λιστα aac_seq_3.

Τελικα η συναρτηση επιστρεφει τη λιστα aac_seq_3 και αποθηκευει το CSV μετρικων.

9.2 i_aac_coder_3(aac_seq_3, filename_out)

Η συναρτηση αυτη υλοποιει τον αποκωδικοποιητη AAC Level 3. Παιρνει ως εισοδο τη δομη aac_seq_3 που παρηχθη απο τον κωδικοποιητη και ανακατασκευαζει το σημα στο χρονο.

1. Αρχικοποιηση

Φορτωνεται το Huffman LUT και οριζονται $N = 2048$, hop_size=1024. Δημιουργειται πινακας εξοδου audio_out με το συνολικο πληθος δειγματων.

2. Αποκωδικοποιηση ανα frame και καναλι

Για καθε frame:

- Διαβαζεται ο τυπος frame απο το dictionary.
- Για καθε καναλι γινεται Huffman decoding:
  - Τα scalefactors απο το codebook 11.
  - Τα S απο το codebook που εχει αποθηκευτει. Αν το codebook ειναι 0, τοτε σημαινει all-zero section και τα S θετονται μηδενικα.
- Γινεται reshape/padding ωστε τα S να εχουν παντα 1024 συντελεστες για long frames (η να αντιστοιχουν σε ESH οπως απαιτειται).
- Γινεται inverse quantizer (i_aac_quantizer) για να παρουμε τους αποκβαντισμενους TNS συντελεστες.
- Εφαρμοζεται inverse TNS (i_tns) χρησιμοποιωντας τους αποθηκευμενους tns_coeffs.

- Οι τελικοι συντελεστες MDCT τοποθετουνται στον πινακα frame_F_decoded.

3. Inverse Filter Bank

Εφαρμοζεται η i_filter_bank για να επιστρεψει το frame στο πεδιο του χρονου.

4. Overlap-Add

Το ανακατασκευασμενο frame προστιθεται στη σωστη θεση του audio_out με overlap-add, λογω 50% επικαλυψης μεταξυ διαδοχικων frames.

5. Αποθηκευση

Στο τελος, το audio_out γραφεται σε WAV αρχειο και επιστρεφεται.

10 Demo_AAC_3

Το αρχειο αυτο υλοποιει ενα demo για τον AAC Level 3. Σκοπος του ειναι να τρεξει ολοκληρο το pipeline κωδικοποιησης και αποκωδικοποιησης, και στη συνεχεια να υπολογισει βασικες μετρικες αξιολογησης της συμπιεσης, δηλαδη:

- SNR (Signal to Noise Ratio) μεταξυ αρχικου και ανακατασκευασμενου σηματος
- bitrate του κωδικοποιημενου stream (bits/sec)
- compression ratio σε σχεση με το αρχικο PCM WAV

Η συναρτηση του demo καλει τον κωδικοποιητη aac_coder_3 και τον αποκωδικοποιητη i_aac_coder_3. Μετα την αποκωδικοποιηση, γινεται ευθυγραμμιση μηκους του σηματος ωστε να ειναι δυνατη η συγκριση δειγμα προς δειγμα, και τελος υπολογιζονται οι μετρικες.

Στην υλοποιηση μου, το αρχειο περιεχει την εξης συναρτηση:

- demo_aac_3: Εκτελει κωδικοποιηση, αποκωδικοποιηση, και υπολογιζει SNR, bitrate και compression.

10.1 demo_aac_3(filename_in, filename_out, filename_aac_coded)

Η συναρτηση αυτη υλοποιει την επιδειξη του AAC Level 3. Παιρνει ως ορισματα:

- filename_in: το αρχικο WAV αρχειο εισοδου
- filename_out: το WAV αρχειο εξοδου (decoded)
- filename_aac_coded: το αρχειο στο οποιο αποθηκευεται το κωδικοποιημενο αποτελεσμα (π.χ .mat)

Επιστρεφει:

- SNR: το Signal to Noise Ratio σε dB
- bitrate: το bitrate του κωδικοποιημενου stream σε bits/sec
- compression: το compression ratio σε σχεση με το αρχικο PCM WAV

1. Φορτωση αρχικου ηχου και ρυθμιση stereo
Διαβαζεται το WAV αρχειο εισοδου. Υπολογιζονται:

- $n\_samples$: πληθος δειγματων
- $duration = \frac{n\_samples}{f_s}$: διαρκεια σε δευτερολεπτα

2. Κωδικοποιηση και αποκωδικοποιηση
Καλουνται διαδοχικα:

$$aac\_seq\_3 = aac\_coder\_3(filename\_in, filename\_aac\_coded)$$

$$audio\_decoded = i\_aac\_coder\_3(aac\_seq\_3, filename\_out)$$

Η δομη aac_seq_3 περιεχει τα κωδικοποιημενα frames και ο decoder παραγει το ανακατασκευασμενο σημα στο πεδιο του χρονου.

3. Ευθυγραμμιση μηκους για συγκριση
Επειδη η αποκωδικοποιηση μπορει να παραγει σημα ελαφρως μεγαλυτερου ή μικροτερου μηκους (λογω padding), το decoded σημα ρυθμιζεται ωστε να εχει ακριβως $n\_samples$:

- αν ειναι μεγαλυτερο, κοβεται
- αν ειναι μικροτερο, γινεται padding με μηδενικα

4. Υπολογισμος SNR

Υπολογιζεται η ισχυς του αρχικου σηματος:

$$P_s = \sum x^2$$

και η ισχυς του θορυβου (διαφορα input-output):

$$P_n = \sum (x - \hat{x})^2$$

και τελικα:

$$SNR = 10 \log_{10} \left(\frac{P_s}{P_n} \right)$$

Αν το P_n ειναι πολυ μικρο, το SNR θεωρειται απειρο. Εδω φαινεται και αυτο που λεγαμε, πως οταν $a \to \infty \Rightarrow S \to 0 \Rightarrow \hat{X}(k) \to 0 \Rightarrow P_e \to \sum X(k)^2 = P_s \Rightarrow SNR \to 0$ dB

5. Υπολογισμος bitrate

Υπολογιζεται το συνολικο πληθος bits του κωδικοποιημενου stream διατρεχοντας ολα τα frames στη δομη aac_seq_3. Για καθε frame προστιθενται bits που αντιστοιχουν σε metadata και στα κωδικοποιημενα δεδομενα καθε καναλιου:

- bits για frame_type / win_type (σταθερο μικρο overhead)
- bits για τους TNS coeffs
- bits για το global gain G
- bits για scalefactors (sfc), το Huffman stream (stream) και το codebook

Το bitrate υπολογιζεται ως:

$$bitrate = \frac{total\_bits}{duration}$$

6. Υπολογισμος compression ratio

Το αρχικο bitrate του PCM WAV (stereo, 16-bit) ειναι:

$$original\_bitrate = 2 \cdot 16 \cdot f_s$$

και το compression ratio:

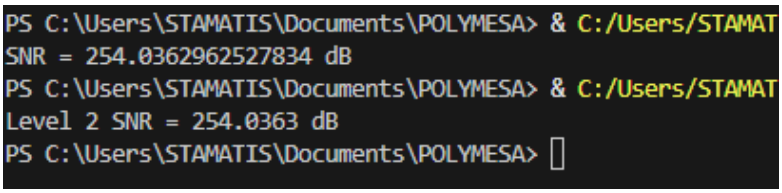
$$compression = \frac{original\_bitrate}{bitrate}$$

Τελικα η συναρτηση επιστρεφει τις μετρικες (SNR, bitrate, compression).

11 Αποτελεσματα Level 1 - Level 2

Τα αποτελεσματα που πηρα για το Level 1 και Level 2 ειναι ιδια. Αυτο συμβαινει διοτι στο Level 1 εφαρμοζουμε απλα input -> SSC -> Filterbank -> iFilterbank -> output, ενω στο Level 2: input -> SSC -> Filterbank -> TNS -> iTNS -> iFilterbank -> output Επειδη οι Filterbank και TNS βαθμιδες ειναι πληρως αντιστρεψιμες, εχουμε ελαχιστες απωλειες που οφειλονται σε στρογγυλοποιησεις και υπολογισμους, γι'αυτο και πετυχαινουμε πολυ μεγαλο και ιδιο SNR.

Level	SNR (dB)
Level 1	254.03629
Level 2	254.0363



Επισης στα παραδοτεα αρχεια, υπαρχει και το output wav των Level 1 και Level 2. Ακουγεται ιδιο με το αρχικο, αφου οπως ειπαμε εχουμε ελαχιστες απωλειες πληροφοριας. Τα αποτελεσματα μπορειτε να τα παρετε αν τρεξετε το run_level_1.py και run_level_2.py αρχεια.

12 Αποτελεσματα Level 3

Ενδεικτικα αποτελεσματα μερικων frames και των μετρικων SNR, bitrate, Compression για το τελικο παραγομενο αρχειο φαινονται παρακατω:

```
        sfc values: [ -19,   22] | Non-zero S:  342/1024 | maxAbs:  3 | codebook:  5

Frame  243 | Type: OLS | Channel: chl
  SMR:        [    0.09,    35.34]
  P(b):       [3.67e-02, 1.38e+04]  (MDCT energy)
  npart(b):   [2.34e-02, 2.87e+02]  (threshold apo psycho)
  T(b):       [1.24e-02, 9.82e+03]  (=P/SMR)
  Pe initial: [1.45e-06, 1.75e-04]
  alpha:      [  -13.83,    18.17] | Iter:  59 | reached max iterations

  [Band 20 Check] SMR=1.86e+01, P=1.26e+04, T=6.75e+02
T = P/SMR = 6.75e+02  (Match: True)
Pe(initial) = 1.88e-05  (Pe < T: True)
  [Bits] sfc:   291 bits | S:  1452 bits | Total:  1743 bits
        sfc values: [ -13,   18] | Non-zero S:  330/1024 | maxAbs:  3 | codebook:  5

Frame  270 | Type: OLS | Channel: chl
  SMR:        [    0.04,    38.07]
  P(b):       [1.15e-03, 3.63e+04]  (MDCT energy)
  npart(b):   [1.42e-01, 3.22e+02]  (threshold apo psycho)
  T(b):       [2.38e-04, 7.68e+03]  (=P/SMR)
  Pe initial: [2.05e-06, 1.98e-04]
  alpha:      [  -29.54,    20.46] | Iter:  59 | reached max iterations

  [Band 20 Check] SMR=2.10e-01, P=3.47e+01, T=1.65e+02
T = P/SMR = 1.65e+02  (Match: True)
Pe(initial) = 3.30e-05  (Pe < T: True)
  [Bits] sfc:   303 bits | S:  1348 bits | Total:  1651 bits
        sfc values: [ -21,   20] | Non-zero S:  303/1024 | maxAbs:  4 | codebook:  5

[INFO] Metrics saved to: coded_level3_metrics.csv
Level 3
 SNR = 10.80 dB
 Bitrate = 177.02 kbps
 Compression = 8.68x
PS C:\Users\STAMATIS\Documents\POLYMESA>
```

Εδω βλεπουμε τα διαστηματα στα οποια ανηκουν οι μετρικες για καθε μπαντα, με τα ακρα του διαστηματος να ειναι η ελαχιστη και μεγιστη τιμη τους. Προφανως, οι τιμες στα ακρα δεν αντιστοιχουν απαραιτητα στην 1η και τελευταια μπαντα συχνοτητων, γι'αυτο απο κατω εκανα και εναν ελεγχο για την μπαντα 20, ωστε να επαληθευσω οτι ισχυει η σχεση $T = \frac{P}{SMR}$. Οι γραμμες στα logs

που αφορουν αυτο τον ελεγχο ειναι απο εκει που λεει [Band 20 Check] μεχρι εκει που λεει (Pe< T: True). Οι μετρικες που βλεπουμε για καθε frame ειναι:

- SMR(b) = e(b)/npart(b) που δινει το ψυχοακουστικο μοντελο
- $P(b) = \sum_{k=w_{low}}^{w_{high}} X(k)^2$ η ενεργεια των MDCT συντελεστων
- npart(b): Ουσιαστικα το κατωφλι ακουστοτητας του θορυβου για τη μπαντα b.
- T(b): το κατωφλι ακουστοτητας του κωδικοποιητη
- $P_e(b)$: Η αρχικη ισχυς του σφαλματος θορυβου κβαντισμου, δηλαδη πριν την αυξηση του a[b]
- alpha: Οι συντελεστες scalefactor gain για καθε μπαντα b
- sfc values: Οι κωδικοποιησεις sfc(b) = Δa(b)
- Non-zero S: Οι μη μηδενικες κωδικοποιησεις S των X(k) (των Y(k) στην πραγματικοτητα, γιατι ο quantizer παιρνει τα φιλτραρισμενα X, δηλαδη την εξοδο Y του TNS).
- maxAbs: Το μεγιστο κατα απολυτιμη τιμη S
- codebook: Το codebook που χρησιμοποιησε ο αλγοριθμος για την huffman κωδικοποιηση των S (το οποιο επιλεγεται με βαση το maxAbs). Για το huffman encoding των sfc χρησιμοποιηθηκε το codebook 11 απο εκφωνηση.
- Bits row: sfc bits, S bits, Total bits, τα αντιστοιχα bits που χρειαστηκαν για την κωδικοποιηση της καθε τιμης.

Σημειωνω εδω οτι, για τα scalefactors , το force_codebook = 11 που ζητησατε, μου πετουσε indexing error. Οι τιμες των scalefactor εβγαιναν μεγαλες, και επειδη ο αλγοριθμος πηγαινε κατευθειαν στην huff_LUT_code_1, ο τροπος με τον οποιο αυτη υπολογιζει το huffman index για την κωδικοποιηση, εδινε index out of bounds,

δηλαδη index που δεν υπηρχε στο huffman codebook. Γι'αυτο τροποποιησα το κομματι της encode _huff απο:

```
if force_codebook is not None:
    return huff_LUT_code_1(huff_LUT_list[force_codebook],
       coeff_sec), force_codebook
```

σε :

```
if force_codebook is not None:
    if force_codebook == 11 and np.max(np.abs(coeff_sec)) > 15:
        return huff_LUT_code_ESC(huff_LUT_list[force_codebook],
           coeff_sec), force_codebook
    return huff_LUT_code_1(huff_LUT_list[force_codebook],
       coeff_sec), force_codebook
```

ετσι ωστε να μπαινει σε ESC huff αν για αυτο το frame βρισκει $|a| > 15$ σε καποιο band.

Για τον quantizer, εχουμε 2 κριτηρια τερματισμου αυξησης του $a[b]$ στην εκφωνηση:

Το $Pe(b) \geq T(b)$ και το $a(b+1) - a(b), a(b) - a(b-1) > 60$

Την επαναληπτικη αυξηση των α την υλοποιησα κατα βαση ως εξης:

```
for iteration in range(60):
    all_done = True

    for b in range(n_bands):
        Pe = np.sum((X[w_low[b]:w_high[b]+1] - X_hat[w_low[b]:
           w_high[b]+1])**2)

        # An Pe < T(b), theloume na auxhsoume to alpha[b]
        if Pe < T[b]:
            can_increase = True

            if b > 0:
                if abs((alpha[b] + 1) - alpha[b-1]) > 60:
                    can_increase = False

            if b < n_bands - 1:
                if abs(alpha[b+1] - (alpha[b] + 1)) > 60:
                    can_increase = False

            if can_increase:
                alpha[b] += 1
                all_done = False
```

```
# Den eixame allagh sta alpa se kanena band, opote exoume th
   veltisth sympiesh
if all_done:
    break
```

Οπως βλεπουμε, ο αλγοριθμος αποτελειται απο ενα εξωτερικο loop επαναληψεων, και ενα εσωτερικο για καθε μπαντα b.

Το εξωτερικο loop, το εβαλα γιατι μπορει τα 2 κριτηρια να θελουν παρα πολλες επαναληψεις να ικανοποιηθουν, και μπορει τα α να αυξηθουν τοσο πολυ που να μου κανουν πρακτικα το σημα αχρηστο (ναι μεν θα εχω πολυ μεγαλη συμπιεση, αλλα ο θορυβος κβαντισμου μπορει να αυξηθει υπερβολικα και το decoded σημα να ειναι απλα θορυβος).

Ξεκινωντας απο το εσωτερικο loop των μπαντων, πρωτα ελεγχουμε αν $P_e(b) < T(b)$. Αν ισχυει, τοτε μονο ελεγχουμε το 2ο κριτηριο ωστε να δουμε αν μπορει να αυξηθει το α(b). Χρησιμοποιησα δυο flags, το all_done, και το can_increase. Το can_increase, ειναι flag που δειχνει αν ικανοποειται το 2ο κριτηριο. Αν δηλαδη $a(b) <= 60$ και για τους 2 γειτονες του a(b), τοτε το α(b) αυξανεται κατα 1. Διαφορετικα μενει σταθερο και προχωραμε στο επομενο band.

Ο τροπος που ελεγξα το 2ο κριτηριο, ειναι με το σκεπτικο "Αν η αυξηση του a[b] κατα 1 οδηγησει σε διαφορα μεγαλυτερη του 60, τοτε μην το αυξησεις". Χωρισα τον ελεγχο σε 2 if, για τις ακραιες περιπτωσεις $b = 0$ και $b = last\_band$. Αν $b = 0$, δεν υπαρχει προηγουμενη μπαντα αρα πρεπει να ελεγχθει μονο η διαφορα του a[b] με την επομενη. Αντιστοιχα αν $b = last\_band$.

Το 2ο flag: all_done δειχνει αν ικανοποιουνται και τα 2 κριτηρια τερματισμου ταυτοχρονα. Αν οχι, τοτε υπαρχει καποιο b που μπορει ακομα να αυξηθει. Αλλιως, σταματαμε την αυξηση ολων των α. Αρχικοποιεται ως True, και αν για καποιο band το can_increase ειναι true, τοτε το α(b) μπορει να αυξηθει και το all_done γινεται False. Αν για ολα τα bands b, το can_increase δεν γινει ποτε true, τοτε σημαινει οτι τελειωσαμε, δηλαδη δεν μπορει καποιο α(b)

να αυξηθει περαιτερω. Οποτε βγαινουμε απο τα 2 loops. Αν το all_done δεν γινει ποτε True, τοτε ο αλγοριθμος σταματαει στο max_iteration που εθεσα, δηλαδη στην παραπανω περιπτωση, στις 60 επαναληψεις.

Με τον αλγοριθμο μου, συναντησα ενα παραδοξο. Οπως ειπαμε, οι max iterations, ειναι οι φορες που θα σαρωσουμε ολα τα α(b), δηλαδη τα α για καθε μπαντα, ωστε να δουμε αν καποιο απο αυτα μπορει να αυξηθει. Αν π.χ $max\_iter = 30$, τοτε 30 φορες θα σαρωσουμε τις μπαντες b, και σε καθε σαρωση ελεγχουμε αν καθε a(b) μπορει να αυξηθει. Επομενως, καθε α(b) θα αυξηθει το μεγιστο κατα 30, αφου αυξανουμε κατα 1.

Οσο πιο μικρο λοιπον το max_iter, τοσο πιο λιγο θα αυξηθουν τα scalefactor gain, αρα τοσο πιο λιγη θα ειναι η συμπιεση, και τοσο πιο λιγος ο θορυβος κβαντισμου δηλαδη μεγαλυτερο το SNR (καλυτερο decoded σημα). Αντιστροφα, οσο πιο μεγαλο το max iterations, τοσο μεγαλωνει η συμπιεση, αρα μικραινει το SNR και το Bitrate (λιγοτερα bits ανα δευτερολεπτο σηματος).

Τα αποτελεσματα που ειχα με $max\_iter = 60$, ειναι αυτα που ειδαμε παραπανω:

- $SNR = 10.8dB$
- $Bitrate = 177.02kbps$
- $Compression = 8.68x$

Το παραδοξο που συναντησα ειναι πως οταν μεγαλωσα αρκετα τα max iterations, το SNR επεφτε οπως αναμεναμε, ομως επεφτε και το Compression. Για να καταλαβω γιατι συνεβαινε αυτο, εβαλα στα logs τα bits που χρειαστηκαν για την κωδικοποιηση των sfc, των S, και το αθροισμα τους: Total bits για καθε frame. Επισης τα sfc values και τα Non-zero S, δηλαδη ποσα S(k) ειναι μη μηδενικα. Προφανως με αυξηση του α, τα S μικραινουν, οποτε περιμενουμε λιγοτερα Non-Zero S.

Παρατηρησα, οτι μετα απο ενα σημειο (60 max iterations) τα non-zero S σωστα μειονωνταν, ομως τα sfc bits αυξανονταν αρκετα. Αυτο οφειλεται στις μεγαλες τιμες που παιρνουν τα α και συγκεκριμενα στις μεγαλες διαφορες α που δημιουργουνται, οποτε αυξανονται τα bits που απαιτουνται για την κωδικοποιηση των sfc. Και μαλλον η encode huff δεν κανει καλη δουλεια για μεγαλα scalefactors-sfc (π.χ δεν επιλεγεται καλο codebook, ή τα codebooks δεν ειναι βελτιστοποιημενα για μεγαλα α). Οποτε τα συνολικα Bits αυξανονται και γι αυτο αυξανεται το Bitrate και πεφτει η συμπιεση.

Δοκιμασα διαφορες λυσεις, οπως να μην κωδικοποιω το sfc(0)=G=a(0), καθως δεν το χρειαζομαι στην κωδικοποιηση, και νομιζω πως επαιρνε μεγαλες τιμες. Το συνειδητοποιησα απο τα sfc values, για τα οποια αναρωτηθηκα "Δε θα πρεπε αφου ειναι διαφορες γειτονικων α, να ειναι ολα απο -60 μεχρι 60 στην χειροτερη περιπτωση? ". Σε μερικα frames το μεγιστο sfc ηταν αρκετα πανω απο το 60. Αυτο μαλλον ηταν το G=a(0). Ωστοσο δεν βοηθησε η αφαιρεση του απο την κωδικοποιηση.

Επισης δοκιμασα να βαλω ενα cap στις τιμες του α, ως ενα εξτρα κριτηριο τερματισμου. Δηλαδη το α(b) να μπορει να αυξηθει μονο αν α(b)<130. Αλλα ουτε αυτο βοηθησε στο προβλημα. Παρακατω παραθετω τα αποτελεσματα που ελαβα οταν ετρεξα το προγραμμα για διαφορετικα max iterations:

Max Iter	SNR (dB)	Bitrate (kbps)	Compression
40	24.71	372.59	4.12x
45	22.20	354.03	4.34x
50	18.82	255.18	6.02x
55	14.73	243.57	6.31x
60	**10.80**	**177.02**	**8.68x**
65	7.68	206.07	7.45x
70	6.44	207.86	7.39x
75	5.97	209.72	7.32x
100	5.97	214.14	7.17x
250	5.97	218.00	7.05x

Παρατηρουμε οτι μετα τις 60 επαναληψεις εμφανιζεται το παραδοξο που ανεφερα. Για καποιο λογο το $maxiter = 60$ αποτελει ενα cap βελτιωσης. Δε μπορω να καταλαβω γιατι συμβαινει αυτο. Προφανως κατι εχω κανει λαθος. Ωστοσο ακολουθησα τις οδηγιες κατα γραμμα. Ισως φταιει η αλλαγη στο huff που εκανα, ομως μου πετουσε error οταν δεν το εβαζα. Ισως το κριτηριο τερματισμου με max iterations δεν ειναι σωστο. Δε ξερω. Παντως μεχρι τις 60 επαναληψεις ο αλγοριθμος δουλευει καλα και πετυχαινω μια καλη συμπιεση.

Παρακατω βλεπουμε ενα αποσπασμα του CSV αρχειου με μετρικες για καθε frame, για $max\_iterations = 60$, και ενα για $max\_iter = 250$. Το αρχειο το εχω συμπεριλαβει στην παραδοση.

Max_iter = 60 :

coded_level3_metrics.csv

```
frame,channel,frame_type,bits_S,bits_sfc,nonzero,maxAbs,codebook,iterations
0,chl,OLS,1482,282,371,3,5,59
0,chr,OLS,1505,263,373,3,5,59
1,chl,OLS,1435,299,349,3,5,59
1,chr,OLS,1555,277,386,3,5,59
2,chl,OLS,1366,280,313,3,5,59
2,chr,OLS,1332,299,305,3,5,59
3,chl,OLS,1382,301,317,3,5,59
3,chr,OLS,1351,282,308,3,5,59
4,chl,OLS,1419,278,333,3,5,59
4,chr,OLS,1392,276,323,3,5,59
5,chl,OLS,1348,286,299,3,5,59
5,chr,OLS,1409,266,318,3,5,59
6,chl,OLS,1363,288,310,3,5,59
6,chr,OLS,1274,300,280,3,5,59
7,chl,OLS,1367,289,304,3,5,59
7,chr,OLS,1327,281,289,3,5,59
8,chl,OLS,1309,311,289,3,5,59
8,chr,OLS,1361,315,304,3,5,59
9,chl,OLS,1289,300,284,3,5,59
```

Max_iter = 250 :

coded_level3_metrics.csv

```
frame,channel,frame_type,bits_S,bits_sfc,nonzero,maxAbs,codebook,iterations
0,chl,OLS,2071,423,348,2,3,189
0,chr,OLS,2097,402,357,2,3,185
1,chl,OLS,2074,374,344,2,3,123
1,chr,OLS,2139,371,374,2,3,173
2,chl,OLS,1974,351,298,2,3,179
2,chr,OLS,1956,381,295,2,3,236
3,chl,OLS,2004,396,305,2,3,126
3,chr,OLS,1992,386,300,2,3,125
4,chl,OLS,1999,382,309,2,3,126
4,chr,OLS,2017,392,307,2,3,126
5,chl,OLS,1297,390,285,3,5,125
5,chr,OLS,1998,389,301,2,3,125
6,chl,OLS,1969,420,294,2,3,121
6,chr,OLS,1939,379,274,2,3,173
7,chl,OLS,2017,406,292,2,3,124
7,chr,OLS,1948,395,276,2,3,127
8,chl,OLS,1957,419,284,2,3,178
8,chr,OLS,1310,410,291,3,5,170
9,chl,OLS,1252,393,276,3,5,120
```

Τελος, μεσα στα αρχεια παραδοσης υπαρχει και το output_level3.wav που ειναι το decoded wav του wav που μας δωσατε.

Ολα τα levels τρεχουν ανοιγωντας και εκτελωντας το αντιστοιχο run_level_x.py αρχειο, ειτε απο terminal, π.χ:

- python run_level_1.py αν ειστε μεσα στο root folder του project
- ή με πληρες path, π.χ: python C:/Users/.../run_level_1.py