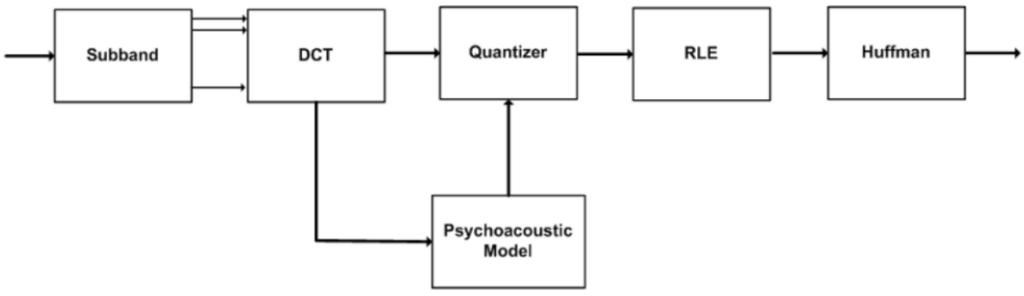
Εργασία Συστημάτων Πολυμέσων

Τζατζίδης Αντώνης

AEM:9938

Σύνοψη

Σκοπός της εργασίας ήταν η δημιουργία ενός απλοποιημένου κωδικοποιητή και αποκωδικοποιητή για το MPEG1 Layer III ακολουθώντας με τη σειρά τα ακόλουθα βήματα και αντιστέφοντας τα κατα την αποκωδικοποίηση.



Subband Filtering

Στη main βρίσκεται η συνάρτηση *plot_frequencies(h,M,L)* που σχεδιάζει τα μέτρα των συναρτήσεων μεταφοράς των φίλτρων h_i ως προς τη συχνότητα σε Hertz και Barks, για την υλοποίηση των ερωτημάτων 3.1.2 και 3.1.3. Για την μετατροπή των Hertz σε Barks χρησιμοποίησα τη σχέση (15). Η συνάρτηση *coder0(wavin, h, M, N)* δέχεται τα δεδομένα ενός wav αρχείου και υπολογίζει το συνολικό αριθμό των iterations. Σε κάθε επανάληψη, γεμίζει το buffer με $M*N+L-M$ στοιχεία τα οποία αναλύονται σε μπάντες απο τη συνάρτηση *frame_sub_analysis()* που μας δόθηκε. Σε κάθε επανάληψη τα frames ενώνονται στο Ytot το οποίο καταλήγει να έχει μέγεθος αριθμός_επαναλήψεων*36x32. Στο τελευταίο iteration γεμίζουμε τα επιπλέον $L-M$ στοιχεία του buffer με 0.

Η *decoder0(wavin, h, M, N, ds)* αντιστρέφει τη διαδικασία γεμίζοντας το buffer με $(N-1+L/M)$xM στοιχεία απο το Ytot τα οποία ενώνονται στη x_hat αφου περάσουν από τα φίλτρα σύνθεσης μέσω της συνάρτησης *frame_sub_synthesis()*. Όπως στην *coder0()* στην τελευταία επανάληψη τα τελευταία L/MxM στοιχεία του buffer γίνονται 0. Πρόσθεσα το έξτρα όρισμα ds που είναι το μέγεθος των δεδομένων για να μπορώ να υπολογίσω από πριν τον αριθμό των επαναλήψεων.

Η *coder0(wavin, h, M, N)* καλεί διαδοχικά τις *coder0()* και *decoder0()*

Τα buffer προκαλούν lag $L-M$ στοιχείων, γι'αυτό στο SNR συγκρίνουμε τα $[L-M:]$ στοιχεία του αρχικού δείγματος με τα $[:-(L-M)]$ του παραγόμενου και παίρνουμε SNR ≈ 83 dB ενώ το αποκωδικοποιημένο σήμα ακουστικά δεν έχει καμία διαφορά με το αρχικό.

DCT

Για την υλοποίηση των συναρτήσεων $frameDCT(Y)$ και $iframeDCT(c)$ χρησιμοποίησα τις συναρτήσεις *dct* και *idct* της scipy με την επιλογή norm='ortho' για να μην υπάρχει απώλεια

πληροφορίας. Η *dct* μετασχηματίζει τον πίνακα σε διάνυσμα πριν γίνει ο μετασχηματισμός. Η DCTpower(c) υλοποιεί τη σχέση (10)

Ψυχοακουστικό Μοντέλο

Η $Dksparse(Kmax)$ loopάρει στο [3,Kmax) (δεν έχει νόημα για $i < 3$) και προσθέτει τις συντεταγμένες κάθε μη μηδενικού στοιχείου σύμφωνα με τη σχέση (12) στα διανύσματα k,j. Με βάση αυτά τα διανύσματα (και το data που έχει μέγεθος όσο τα k,j και περιέχει μόνο 1) επιστρέφει τον αραιό πίνακα D.

Η $STinit(c, D)$ ελέγχει αν το διάνυσμα της ισχύς των συντελεστών DCT ικανοποιεί την 1η συνθήκη της σχέσης (11) και στη συνέχεια τη 2η. Τα Δ_k υπολογίζονται από την αντίστοιχη σειρά του πίνακα D. Αν ικανοποιεί και τις 2 σχέσεις προστίθεται στον πίνακα ST, τον οποίο επιστρέφει. Το loop τρέχει στο διάστημα [3,$Kmax - 27$) ώστε όλοι οι έλεγχοι να είναι εντός των ορίων του πίνακα.

Η $calcFreqs(i)$ δέχεται ένα διάνυσμα από indexes και το μετατρέπει σε συχνότητες (Hz) σύμφωνα με τη σχέση (9). Η $Hz2Barks(f)$ μετατρέπει τις συχνότητες από Hz σε Barks σύμφωνα με τον τύπο (15).

Τα tonal components της $STinit(c, D)$ μειώνονται στην $STreduction(ST, c, Tq)$ η οποία κρατάει μόνο αυτά που περνούν τον έλεγχο της σχέσης (14) και διαγράφει αυτό με τη μικρότερη ισχύ όταν 2 από αυτά απέχουν απόσταση μικρότερη του 0.5 bark.

Η $SpreadFunc(ST, PM, Kmax)$ υπολογίζει για κάθε k το διάνυσμα $z(f_i) - z(f_k)$ όπου $i \in [0, Kmax - 1]$ και γεμίζει τον πίνακα SF σύμφωνα με τη σχέση (17). Ο πίνακας αρχικοποιείται με 0 για τα στοιχεία $ST[i][k]$ όπου το $\Delta_z \notin [-3, 8)$.

Οι συναρτήσεις $Masking\_Thresholds(ST, PM, Kmax)$ και $Global\_Masking\_Thresholds(Tm, Tq)$ υλοποιούν τις σχέσεις (16) και (18) αντίστοιχα, ενώ η $Psycho(c, D)$ συνδιάζει όλες τις παραπάνω καλώντας τις με τη σειρά, για να παράξει το συνολικό κατώφλι ακουστότητας για κάθε συντελεστή. Η μετατόπιση της Tq , όσο τη δοκίμασα, δεν είχε κάποια επίδραση στο τελικό αποτέλεσμα.

Κβαντισμός-Αποκβαντισμός

Η $critical\_bands(K)$ αφού μετατρέψει τα index ενός πίνακα μεγέθους K σε συχνότητες, υλοποιεί τον πίνακα 2 και τις μετατρέπει στον αριθμό της μπάντας που ανήκουν.

Η $DCT\_band\_scale(c)$ καλεί την $critical\_bands(K)$ και με βάση αυτή μετατρέπει το διάνυσμα c σε λίστα 25 διανυσμάτων, όπου το κάθε διάνυσμα $i \in [0, 24]$ περιέχει τα στοιχεία που ανήκουν στη μπάντα $i + 1$. Για κάθε ένα από αυτά υπολογίζουμε και αποθηκεύουμε το $sc(i)$ και με βάση αυτό υπολογίζεται το $cs(i)$ για κάθε στοιχείο της κάθε μπάντας.

Ο κβαντισμός γίνεται από την $quantizer(x, b)$ η οποία αφού υπολογίσει πόσα επίπεδα υπάρχουν με βάση τον αριθμό των bits τοποθετεί με τη σειρά το πρώτο επίπεδο, στη συνέχεια όσα βρίσκονται πριν το 0, το επίπεδο που περιέχει το 0, όσα levels βρίσκονται μετά το 0 και το τελευταίο.

Τέλος ελέγχει σε ποιό επίπεδο βρίσκονται τα στοιχεία που πρόκειται να κβαντιστούν και τα κανονικοποιεί από το εύρος [0, $2*levels\_per\_side$] στο [$-levels\_per\_side$,$levels\_per\_side$]. Για τον αποκβαντισμό αντίστοιχα, η $dequantizer(symb\_index, b)$ δημιουργεί αρχικά τις τιμές εξόδου του αποκβαντιστή με βάση των αριθμό των bits και αφού κανονικοποιήσει τις τιμές στο αρχικό εύρος τις αντιστοιχίζει αυτόματα στην κατάλληλη έξοδο του αποκβαντιστή για καθένα απο τα κβαντισμένα στοιχεία.

Η $all\_bands\_quantizer(c, Tg)$ καλέι την $DCT\_band\_scale(c)$ για να δημιουργηθούν τα $cs[i]$ στοιχεία κάθε μπάντας. Στη συνέχεια αφου χωριστεί με ανάλογο τρόπο η Tg για να έχει ίδιες διαστάσεις με το cs, ξεκινάει την εφαρμογή του αλγορίθμου 2.4. Για κάθε μπάντα κβαντίζει και αποκβαντίζει τα στοιχεία της με συγκεκριμένο αριθμό bits και υπολογίζει την ισχύ του θορύβου. Αν είναι μικρότερη της Tg για όλα τα στοιχεία της μπάντας σταματάει σε αυτό τον αριθμό bits αλλιώς τα αυξάνει κατά 1 μέχρι να γίνει μικρότερη. Στο τέλος, όλες οι μπάντες κβαντίζονται με τον μεγαλύτερο αριθμό bits που εντοπίζεται.

Η $all\_bands\_dequantizer(symb\_index, B, SF)$ αποκβαντίζει όλα τα στοιχεία κάθε μπάντας με βάση τον αριθμό των bits που κβαντίστηκαν σύμφωνα με τη σχέση (20).

RLE

Η $RLE(symb\_index, K)$ αφού μετατρέψει τη λίστα των 25 διανυσμάτων με τα κβαντισμένα στοιχεία σε ένα ενιαίο διάνυσμα μετράει τα 0 πριν την εμφάνιση μη μηδενικού στοιχείου και τα αποθηκεύει σε πίνακα με τη μορφή (symbol,leading_zeros) για κάθε εμφάνιση μη μηδενικού. Αν το διάνυσμα των κβαντισμένων συντελεστών τελειώνει με n μηδενικά προσθέτεται μία τελευταία γραμμή του πίνακα με τη μορφή (0,$n-1$).

Η $iRLE(run\_symbols, K)$ παράγει ένα διάνυσμα με τον σωστό αριθμό 0 πριν από κάθε σύμβολο για κάθε γραμμή του πίνακα.

Huffman

Για την υλοποίηση της κωδικοποίησης Huffman η συνάρτηση $huff(run\_symbols)$ αρχικά μετράει πόσες φορές εμφανίζεται κάθε διαφορετική γραμμή του πίνακα run_symbols και τις ταξινομεί με φθίνουσα σειρά ενω μετατρέπει την κάθε σειρά από πίνακα ints σε string για πιο εύκολο χειρισμό. Στη συνέχεια καλεί την $make\_tree(nodes)$ η οποία δημιουργεί το δέντρο απο κάτω προς τα πάνω ενώνοντας συνεχώς τα σύμβολα με τη μικρότερη πιθανότητα και δημιουργώντας νέα με το άθροισμα τους μέχρι να μείνει ένα μόνο root node το οποίο επιστρέφει. Στη συνέχεια η $huffman\_code\_tree(node, code)$ διασχίζει το δέντρο προσθέτοντας 0 και 1 σε κάθε ακμή του γράφου (0 για το αριστερό παιδί, 1 για το δεξί) μέχρι να φτάσει σε κάθε φύλλο, και επιστρέφει ένα dictionary με τον κωδικό που αντιστοιχεί σε κάθε σύμβολο. Τέλος αντιστοιχεί κάθε γραμμή του run_symbols με τον κώδικα του και παράγει το frame_stream.

Η $ihuff(frame\_stream, frame\_symbol\_prob)$ δημιουργεί ένα δέντρο και αντιστοιχίζει κάθε μοναδική διάδα του RLE με ένα δυαδικό κωδικό με τον ίδιο τρόπο και στη συνέχεια αντιστρέφει το dictionary για να μετατρέψει το frame_stream σε διάδες που θα αποτελέσουν τον πίνακα run_symbols του RLE.

MP3 Encode-Decode

Για να κωδικοποιηθεί το MP3 στην $MP3cod(wavin, h, M, N)$ καλείται αρχικά η $coder0()$ για να παραχθεί το Ytot. Στη συνέχεια, loopάρει, κωδικοποιώντας ένα $N \times M$ frame του Ytot σε κάθε επανάληψη. Για να γίνει αυτό, καλεί με τη σειρά τις $frameDCT()$, $Psycho()$, $all\_bands\_quantizer()$, $RLE()$ και $huff()$. Σε κάθε iteration, σώζει το frame_stream και τον πίνακα πιθανοτήτων απο την $huff()$, τον αριθμό των bits και τα scaling factors από τον κβαντισμό.

Οι πληροφορίες αυτές μεταφέρονται στην $MP3decod(Y\_tot, h, M, N, encoder\_info, data\_shape)$ η οποία θα τις χρησιμοποιήσει καλώντας με τη σειρά τις $ihuff()$, $iRLE()$, $all\_bands\_dequantizer()$ και $iframeDCT()$ για να πάρει το αποκωδικοποιημένο σήμα Ytot_final, το οποίο θα περάσει στην $decoder0()$ για να παραχθεί το τελικό $\hat{x}$.

Η $MP3codec0(wavin, h, M, N)$, όπως η $codec0()$, καλεί την $MP3cod()$ και στη συνέχεια την $MP3decod()$ μεταφέροντας της τον πίνακα πληροφοριών encoder_info που είναι απαραίτητος για την αποκωδικοποίηση.

Μετά την κωδικοποίηση και την αποκωδικοποίηση το σήμα ήχου παραμένει ακουστικά κοντά στο αρχικό. Με τα ακουστικά μου παρατήρησα μόνο κάποια μικρά glitches που δεν υπήρχαν στο αρχικό, παρότι πλέον το SNR ≈ 18 dB. Για τον υπολογισμό του βαθμού συμπίεσης συνέκρινα το μέγεθος του αρχικού αρχείου σε bits με το μέγεθος της τελικής κωδικοσειράς huffman (υπέθεσα ότι αφού αποτελείται από 0 και 1 κάθε στοιχείο της κωδικής σειράς θα αντιστοιχεί σε 1 bit). Με βάση αυτό, ο βαθμός συμπίεσης προκύπτει 76%, αν και αυτό θα είναι πρακτικά λίγο μικρότερο αν μετρηθούν και οι υπόλοιπες πληροφορίες που πρέπει να σταλούν για την αποκωδικοποίηση του σήματος.