

Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

**Audio Fingerprinting**

Ομάδα :

* Αργυρόπουλος Χρήστος 19013
* Αλεξίου Αντριάνα 19007
* Αυλωνίτης Κωνσταντίνος-Οδυσσέας 19014

**ΠΕΡΙΛΗΨΗ**

Στην παρούσα εργασία παρουσιάζει και προσομοιώνει ένα σύστημα, το οποίο κάνει χρήση των ακουστικών αποτυπωμάτων (audio fingerprints) για την αναγνώριση ηχητικών σημάτων και συγκεκριμένα μουσικούς ήχους.

Σκοπός της εργασίας αυτής είναι να συνδυάσει και να εκμεταλλευτεί τις δυνατότητες που παρέχει η τεχνική αυτή, σχετικά με την ανάλυση και την αναγνώριση με μοναδικό τρόπο μιας συσκευής , προς την διευκόλυνση του χρήστη.

Το συγκεκριμένο σύστημα είναι υλοποιημένο σε Python και βασίζεται στην τεχνολογία Audio Fingerprinting με χρήση Hash Functions. Στα πλαίσια της παρούσας αναλύεται τόσο η διαδικασία εξαγωγής του ακουστικού αποτυπώματος , όσο και η μέθοδος αναζήτησης στη βάση δεδομένων.

Η δομή της εργασίας συνοψίζεται ως ακολούθως: Σε πρώτο στάδιο ορίζεται η έννοια του audio fingerprint και άλλες βασικές έννοιες και παράμετροι, περιγράφεται αναλυτικά η διαδικασία του audio fingerprinting και οι περιορισμοί που τίθενται. Επιπλέον, αναλύονται εφαρμογές συστημάτων που χρησιμοποιούν ακουστικά αποτυπώματα και οι βασικές τεχνολογίες αυτού του είδους που έχουν αναπτυχθεί. Τέλος περιγράφεται το σύστημα που χρησιμοποιήθηκε και παρατίθενται τα αποτελέσματα των προσομοιώσεων καθώς και τα συμπεράσματα.

**Λέξεις Κλειδιά:** ακουστικό αποτύπωμα, αναγνώριση ήχου, βάση δεδομένων, αλγόριθμοι αναζήτησης, συνάρτηση κατακερματισμού .

**ΕΙΣΑΓΩΓΗ**

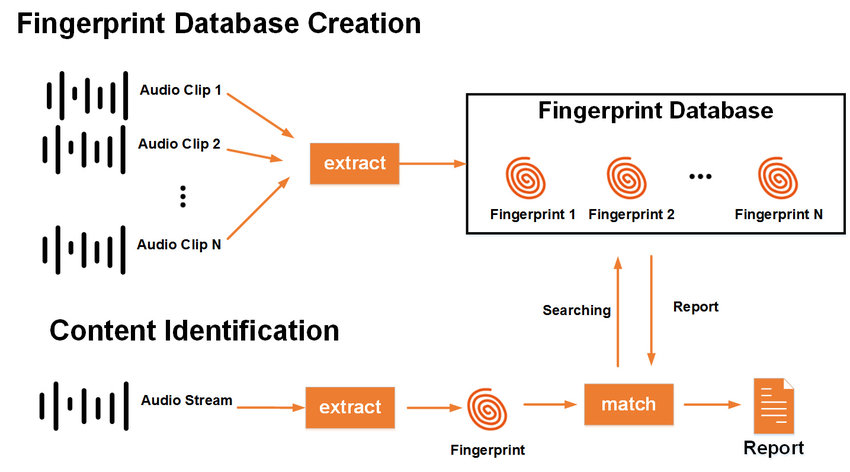
Ο σκοπός αυτού του κεφαλαίου είναι να παρουσιάσει με μεγαλύτερη ακρίβεια το αναγνωριστικό ήχου (audio fingerprint) και τις κύριες έννοιες που σχετίζονται με αυτό. Θα εξηγηθούν οι γενικές αρχές λειτουργίας ενός συστήματος αναγνώρισης ήχου βασιζόμενο στο περιεχόμενό του, καθώς και η δομή ενός τέτοιου συστήματος, ενώ θα αναλυθούν επίσης τα συνιστώσα μέρη και τα χαρακτηριστικά τους.

**ΟΡΙΣΜΟΣ ΤΟΥ AUDIO FINGERPRINT**

Ο όρος "Audio Fingerprint"[1] αναφέρεται μια συνοπτική αναπαράσταση ενός ακουστικού αντικειμένου. Αυτή η αναπαράσταση περιέχει μια μοναδική περιγραφή του ακουστικού αντικειμένου, με βάση το περιεχόμενό του. Ο απώτερος στόχος είναι να ελεγχθεί εάν δύο αντικείμενα είναι "ίδια" ακουστικά, συγκρίνοντας όχι απευθείας τα ίδια αντικείμενα, αλλά τα ακουστικά αποτυπώματά τους, που θα μπορούσαν να αποκαλούνται "audio fingerprints".

**Δομή**

Ένα σύστημα Audio Fingerprinting[19] αποτελείται από δύο βασικά μέρη : τη μέθοδο εξαγωγής audio fingerprint και την αναζήτηση αυτού στη βάση δεδομένων. Για να καταστεί αποτελεσματικό το σύστημα, απαιτείται τόσο η μέθοδος εξαγωγής όσο και η στρατηγική αναζήτησης να λειτουργούν άρτια. Παρακάτω δίνεται σχηματικά η δομή ενός τέτοιου συστήματος :

****

**Εικόνα 1:** Δομή Συστήματος Audio Fingerprinting

**ΓΕΝΙΚΗ ΛΕΙΤΟΥΡΓΙΑ**

* **Δημιουργία ΒΔ**: Από δεδομένα ηχητικά σήματα(μουσικά κομμάτια, βιοσήματα, λοιποί ήχοι) εξάγονται τα reference audio fingerprints. Τα reference audio fingerprints είναι streams από συνεχόμενα audio fingerprints που εξάγονται από την συνολική διάρκεια του ηχητικού αποσπάσματος. Αυτά καταχωρούνται στη βάση δεδομένων και κάθε reference audio fingerprint συνδέεται με τα μεταδεδομένα που αντιστοιχούν στο απόσπασμα από το οποίο προήλθε.
* **Αναγνώριση**: Εξάγεται το audio fingerprint από το άγνωστο απόσπασμα και αναζητάται στη βάση δεδομένων. Αν γίνει ταυτοποίηση, τότε ανακτώνται τα μεταδεδομένα που αντιστοιχούν σε αυτό κι έτσι γίνεται η αναγνώριση.

**Η ΣΥΝΑΡΤΗΣΗ ΤΩΝ AUDIO FINGERPINTS**

Η εξαγωγή των audio fingerprints [2][3] γίνεται μέσω μίας συνάρτησης F, η οποία δέχεται ως είσοδο το ακουστικό αντικείμενο A, το οποίο αποτελείται από έναν μεγάλο αριθμό bits, και επιστρέφει το audio fingerprint Β, το οποίο αποτελείται από έναν πολύ μικρότερο αριθμό bits. Η συνάρτηση F δεν αποσκοπεί στην επίτευξη μαθηματικής ισότητας ανάμεσα στα αντικείμενα, αλλά στην επίτευξη ακουστικής ισοδυναμίας. Αυτό σημαίνει ότι αντικείμενα που είναι ακουστικά ίδια πρέπει να παράγουν όμοια audio fingerprints. Αντίθετα, διαφορετικά αντικείμενα πρέπει να παράγουν διαφορετικά audio fingerprints. Συνεπώς, με μαθηματικούς όρους, μία σωστά σχεδιασμένη συνάρτηση F που παράγει audio fingerprints πρέπει να έχει ένα κατώφλι T, τέτοιο ώστε:

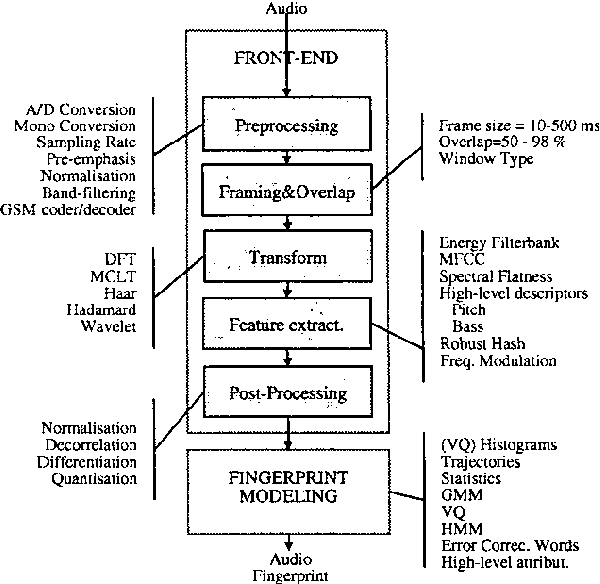
|| 𝐹(𝑋) − 𝐹(𝑌) || ≤ 𝑇

αν τα αντικείμενα X και Y είναι ακουστικά ίσα, και

|| 𝐹(𝑋) − 𝐹(𝑌)|| > 𝑇

Αν τα αντικείμενα X και Y είναι ακουστικά διάφορα.

Η διαδικασία της εξαγωγής του ακουστικού αποτυπώματος αποτελεί το πρώτο βασικό μέρος ενός συστήματος αναγνώρισης ήχου βασισμένου στο περιεχόμενό του. Παρά τις διαφορετικές τεχνικές που χρησιμοποιούνται για την εξαγωγή των ακουστικών αποτυπωμάτων, όλες έχουν κάποια κοινά βασικά στοιχεία και ακολουθούν συγκεκριμένα βήματα. Η ιδέα είναι να αντληθούν από το σήμα ανθεκτικά χαρακτηριστικά και να δημιουργηθεί ένα ακουστικό αποτύπωμα με πολύ μικρότερες διαστάσεις από το αρχικό ακουστικό αντικείμενο.

  
**Εικόνα 2:** Εξαγωγή του Audio Fingerprint

**Προεπεξεργασία (Preprocessing):** Αρχικά, το σήμα υποβάλλεται σε διαδικασία ψηφιοποίησης, αναπαριστώντας το σε μια πρότυπη μορφή[17]. Σε ορισμένες περιπτώσεις, το σήμα υποβάλλεται σε επεξεργασία για να προσομοιώσει το κανάλι μετάδοσης, όπως η εφαρμογή ζωνοπερατού φιλτραρίσματος για αναγνώριση μέσω τηλεφώνου ή η κωδικοποίηση/αποκωδικοποίηση GSM σε ένα σύστημα αναγνώρισης μέσω κινητού τηλεφώνου.

**Διαχωρισμός σε καρέ και επικάλυψη (Framing & Overlap):** Μια βασική υπόθεση είναι ότι το σήμα μπορεί να θεωρηθεί στατικό για ένα διάστημα μερικών milliseconds. Επομένως, το σήμα χωρίζεται σε frames , και σε κάθε ένα από αυτά εφαρμόζεται ένα πολλαπλασιαστικό παράθυρο, όπως το Hanning, για να ελαχιστοποιηθούν οι ασυνέχειες στην αρχή και στο τέλος κάθε καρέ. Επίσης, απαιτείται μεγάλη επικάλυψη μεταξύ των καρέ για να διασφαλιστεί η ευστάθεια σε χρονικές ολίσθησεις (time-shifting).

**Μετασχηματισμός (Transformation):** Στη συνέχεια, κάθε frame μετασχηματίζεται . Οι περισσότερες μέθοδοι χρησιμοποιούν μετασχηματισμούς από το πεδίο του χρόνου στο πεδίο της συχνότητας, με σκοπό την συμπίεση και αφαίρεση θορύβου. Οι κοινότεροι μετασχηματισμοί είναι ο Fast Fourier Transform (FFT), ο Discrete Cosine Transform (DCT), ο Haar Transform, ο Walsh-Hadamard Transform, ο Modulated Complex Lapped Transform και άλλοι.

**Εξαγωγή Χαρακτηριστικών (Feature Extraction):** Σε αυτό το βήμα, εξάγονται τα χαρακτηριστικά που θα χρησιμοποιηθούν για τον υπολογισμό του ακουστικού αποτυπώματος. Ανάλογα με την τεχνική, μπορούν να χρησιμοποιηθούν ποικίλα χαρακτηριστικά. Οι συντελεστές Fourier, οι Mel-Frequency Cepstrum Coefficients (MFCC), το Spectral Flatness Measure (SFM), οι ενέργειες συγκεκριμένων μπαντών του φάσματος, οι Linear Predictive Coding (LPC) coefficients και άλλα είναι τα κυριότερα χαρακτηριστικά που χρησιμοποιούνται.

**Μετά την προεπεξεργασία (Post-Preprocessing):** Τα χαρακτηριστικά που έχουν εξαχθεί από προηγούμενο στάδιο σε ορισμένες τεχνικές χρησιμοποιούνται κατευθείαν για τη δημιουργία του ακουστικού αποτυπώματος. Παρόλα αυτά, υπάρχουν διάφορες μεθόδοι όπου απαιτείται περαιτέρω επεξεργασία, η οποία υλοποιείται σε αυτό το στάδιο. Αυτό σημαίνει ότι τα χαρακτηριστικά μπορούν να υποβληθούν σε κβαντισμό[17], παραγωγία, κανονικοποίηση και διάφορους μετασχηματισμούς, όπως ο Oriented Principal Component Analysis (OPCA) και άλλοι. Επιπλέον, σε αυτό το στάδιο πραγματοποιείται μοντελοποίηση των ακουστικών αποτυπωμάτων με διάφορες μεθόδους, όπως για παράδειγμα οι Hidden Markov Models (HMM), μια διαδικασία που θεωρείται απαραίτητη από ορισμένες τεχνικές.

**ΑΝΑΖΗΤΗΣΗ ΣΤΗ ΒΑΣΗ ΔΕΔΟΜΕΝΩΝ**

Το δεύτερο κύριο στάδιο ενός συστήματος αναγνώρισης ήχου βασισμένου στα ακουστικά αποτυπώματα είναι εξίσου σημαντικό με το πρώτο και αφορά την αναζήτηση των ακουστικών αποτυπωμάτων στη βάση δεδομένων. Η αναζήτηση ενός audio fingerprint στη βάση δεδομένων είναι ένας τύπος "ασαφούς αναζήτησης" (fuzzy search). Συγκεκριμένα, με βάση το ακουστικό αποτύπωμα που προκύπτει από το άγνωστο αντικείμενο, αναζητούμε για το ποιο audio fingerprint αναφοράς στη βάση δεδομένων που μοιάζει περισσότερο με αυτό που ζητούμενο.

Η μέθοδος που θα χρησιμοποιηθεί για την αναζήτηση του ακουστικού αποτυπώματος εξαρτάται από την αναπαράστασή του. Συνήθως χρησιμοποιούνται ευριστικές μέθοδοι από τις διάφορες τεχνικές που υπάρχουν [4][5]. Ο βασικός στόχος των περισσοτέρων είναι να δημιουργηθεί μια δομή δεδομένων με έναν δείκτη, ώστε η αναζήτηση του ακουστικού αποτυπώματος να είναι όσο το δυνατόν πιο αποδοτική, με τον ελάχιστο αριθμό συγκρίσεων ανάμεσα στο αποτύπωμα αναζήτησης και αυτά που είναι αποθηκευμένα στη βάση δεδομένων.

Ένας άλλος σημαντικός παράγοντας στην αναζήτηση των audio fingerprints είναι το είδος της απόστασης που θα χρησιμοποιηθεί κατά τη σύγκριση. Εφόσον πραγματοποιούμε αναζήτηση εγγύτητας, χρησιμοποιούμε διάφορους τύπους αποστάσεων (distance metrics) από διάφορες τεχνικές ανίχνευσης ακουστικών αποτυπωμάτων. Ορισμένες από τις προτιμώμενες αποστάσεις για την αναζήτηση των ακουστικών αποτυπωμάτων είναι η Ευκλείδεια απόσταση και ορισμένες διακριτές παραλλαγές της, η απόσταση Manhattan, η απόσταση Hamming και άλλες[6][7].

Τέλος, κατά τη σύγκριση των ακουστικών αποτυπωμάτων, ανάλογα με την απόσταση που χρησιμοποιείται, επιτυγχάνονται αποτελέσματα. Το σημαντικό, λοιπόν είναι ο τρόπος λήψης της απόφασης για το εάν το ακουστικό αποτύπωμα που αναζητείται βρίσκεται στη βάση δεδομένων ή όχι. Για να γίνει η εντοπισμός ενός αποτυπώματος στη βάση δεδομένων, οι λαμβανόμενοι υπολογισμοί πρέπει να είναι κάτω από ένα ορισμένο κατώφλι. Το κατώφλι αυτό εξαρτάται από τη μέθοδο εξαγωγής του ακουστικού αποτυπώματος, την ομοιότητα μεταξύ των ακουστικών αποτυπωμάτων της βάσης δεδομένων και το μέγεθος της βάσης δεδομένων. Όσο μεγαλύτερη είναι η βάση, τόσο αυξάνεται η πιθανότητα εμφάνισης ενός "false positive", δηλαδή μιας τυχαίας και εσφαλμένης ταυτοποίησης. Επομένως, γίνεται κατανοητό πόσο δύσκολη είναι η επιλογή αυτού του ορίου και πόσοι διαφορετικοί παράγοντες επηρεάζουν αυτήν την απόφαση.

**ΒΑΣΙΚΕΣ ΠΑΡΑΜΕΤΡΟΙ ΤΩΝ AUDIO FINGERPRINTS**

Έχοντας δώσει τον ορισμό του audio fingerprint αλλά και της συνάρτησης που το παράγει, είναι σημαντικό να καθοριστούν οι διάφορες παράμετροι που το επηρεάζουν:

**Ευρωστία:** Ο δείκτης αυτός προσδιορίζει την ανθεκτικότητα ώστε το ακουστικό αποτύπωμα να αναγνωριστεί ακόμη και αν έχει υποστεί σοβαρές παραμορφώσεις. Ο ρυθμός false negative χρησιμοποιείται για να μετρήσει την ευρωστία. Ένα false negative συμβαίνει όταν δεν αναγνωρίζεται ένα ηχητικό απόσπασμα παρά την πραγματική του παρουσία. Για μια ικανοποιητική ευρωστία, το ακουστικό αποτύπωμα πρέπει να βασίζεται σε χαρακτηριστικά που δεν επηρεάζονται από το θόρυβο, έτσι ώστε παραμορφωμένα ηχητικά αντικείμενα να οδηγούν σε παρόμοια ακουστικά αποτυπώματα.

**Αξιοπιστία:** Ο δείκτης αυτός ορίζει την αξιοπιστία της αναγνώρισης, δηλαδή τη συχνότητα λανθασμένης αναγνώρισης ενός ηχητικού αποσπάσματος. Ο ρυθμός false positive χρησιμοποιείται για να μετρήσει την αξιοπιστία. Ένα false positive συμβαίνει όταν γίνεται αναγνώριση ενώ δεν θα έπρεπε να γίνει.

**Μέγεθος:** Ο δείκτης αυτός αναφέρεται στο μέγεθος των audio fingerprints, συνήθως μετρώντας τα bits ανά δευτερόλεπτο. Τα ακουστικά αποτυπώματα συνήθως αποθηκεύονται στη μνήμη RAM για γρήγορη αναζήτηση. Έτσι, το μέγεθος των αποτυπωμάτων επηρεάζει την απαίτηση μνήμης του διακομιστή.

**Granularity :** Ο δείκτης αυτός καθορίζει το χρονικό διάστημα που απαιτείται για τη δημιουργία ενός audio fingerprint και την αναγνώριση του ηχητικού αποσπάσματος. Και εξαρτάται από τη συνάρτηση εξαγωγής των audio fingerprints. Για μια ικανοποιητική αναγνώριση, απαιτείται ένα ελάχιστο χρονικό διάστημα του ηχητικού αποσπάσματος, το οποίο καθορίζεται από τη συνάρτηση εξαγωγής ακουστικών αποτυπωμάτων.

**Ταχύτητα Αναζήτησης:** Ο δείκτης αυτός ορίζει τον χρόνο που απαιτείται για την εύρεση ενός ακουστικού αποτυπώματος στη βάση δεδομένων . Ο χρόνος αυτός επηρεάζεται από τον αριθμό των αποθηκευμένων ακουστικών αντικειμένων στη βάση δεδομένων και επηρεάζεται όσο αυξάνεται ο αριθμός αυτός.

Αυτές οι παράμετροι είναι σημαντικές για τον καθορισμό της απόδοσης ενός συστήματος αναγνώρισης ακουστικών αποτυπωμάτων. Η βελτιστοποίηση τους μπορεί να οδηγήσει σε ακριβέστερη και αξιόπιστη αναγνώριση ηχητικών αποσπασμάτων[8][9].

**ΠΕΡΙΟΡΙΣΜΟΙ**

Κάθε σύστημα αναγνώρισης ήχου με τη χρήση audio fingerprints πρέπει να πληροί ορισμένες προϋποθέσεις και περιορισμούς προκειμένου να λειτουργεί αξιόπιστα και με καλή απόδοση. Αυτοί οι περιορισμοί είναι:

**Γρήγορη και απλή εξαγωγή audio fingerprints:** Είναι σημαντικό η διαδικασία εξαγωγής των ακουστικών αποτυπωμάτων να είναι γρήγορη και να απαιτεί χαμηλούς υπολογιστικούς πόρους. Έτσι, το σύστημα μπορεί να χρησιμοποιηθεί ακόμα και σε περιβάλλοντα με περιορισμένους υπολογιστικούς πόρους.

**Μικρό μέγεθος audio fingerprints:** Τα ακουστικά αποτυπώματα πρέπει να είναι όσο το δυνατόν μικρότερα για να μειωθούν οι απαιτήσεις σε μνήμη και χωρητικότητα. Αυτό επιτρέπει την αποθήκευση και αναζήτηση μεγάλου αριθμού ακουστικών αντικειμένων σε μια βάση δεδομένων με περιορισμένους πόρους.

**Γρήγορη αναζήτηση στη βάση δεδομένων:** Η αναζήτηση των ακουστικών αποτυπωμάτων στη βάση δεδομένων πρέπει να είναι γρήγορη και όχι υπολογιστικά πολύπλοκη. Αυτό επιτρέπει την αμεσότητα της ταυτοποίησης ακόμα και σε μεγάλες βάσεις δεδομένων με περιορισμένους υπολογιστικούς πόρους.

**Αξιόπιστη αναγνώριση με ακρίβεια:** Η μέθοδος αναζήτησης πρέπει να είναι ικανή να αναγνωρίζει ακουστικά αντικείμενα από έναν μεγάλο αριθμό ακουστικών αντικειμένων της βάσης δεδομένων με γρήγορο και αξιόπιστο τρόπο, προσφέροντας σωστά αποτελέσματα. Ο στόχος είναι να μειωθούν τα false positives και false negatives παίρνοντας ακριβή αποτελέσματα.

**Ευρωστία σε παραμορφώσεις και περιβαλλοντικές παρεμβολές:** Το σύστημα πρέπει να μπορεί να αναγνωρίζει ακουστικά αντικείμενα ανεξαρτήτως των παραμορφώσεων του ήχου, όπως ισοστάθμιση, θόρυβος, μετατροπές D-A/A-D, κωδικοποίηση φωνής και ήχου (όπως GSM και MP3), χρονική ολίσθηση και άλλες παρεμβολές. Το ακουστικό αποτύπωμα πρέπει να είναι ανθεκτικό, έτσι ώστε να επιτρέπει την αξιόπιστη αναγνώριση ακόμα και με απόσπασμα ήχου διάρκειας λίγων δευτερολέπτων.

Συνολικά, μια εφαρμογή αναγνώρισης ήχου που πληροί αυτές τις βασικές προϋποθέσεις μπορεί να λειτουργεί με υψηλή ταχύτητα και αξιοπιστία, αποτελώντας σημαντικό κομμάτι απαιτητικών εφαρμογών και χρησιμοποιώντας περιορισμένους υπολογιστικούς πόρους, χωρίς να επιβαρύνει το γενικότερο σύστημα[9][10]**.**

**ΕΦΑΡΜΟΓΕΣ ΤΟΥ AUDIO FINGERPRINTING**

Η τεχνική του Audio Fingerprinting ευρέως χρησιμοποιείται για την αναγνώριση μουσικών κομματιών και παρουσιάζει πολλές εφαρμογές. Ορισμένες από τις κύριες εφαρμογές περιλαμβάνουν:

1. **Παρακολούθηση ραδιοφωνικών εκπομπών:** Χρησιμοποιείται για τη δημιουργία λιστών με τα μουσικά κομμάτια που παίζονται σε ραδιοφωνικούς σταθμούς ή στο διαδίκτυο, κυρίως για στατιστικούς και λόγους πνευματικών δικαιωμάτων. Σε αυτό το σύστημα, πολλά monitoring sites εξάγουν τα ακουστικά αποτυπώματα από τα κανάλια εκπομπής, ενώ ένα κεντρικό site αναλαμβάνει τη συλλογή και αποθήκευση των αποτυπωμάτων σε μια μεγάλη βάση δεδομένων για την αναγνώριση[6],[11],[12][13].
2. **Αναγνώριση μουσικών κομματιών μέσω κινητού τηλεφώνου:** Έχουν αναπτυχθεί πολλές εφαρμογές για την αναγνώριση μουσικών κομματιών μέσω κινητού τηλεφώνου[11],[13]. Αυτές οι εφαρμογές επιτρέπουν στους χρήστες να αναγνωρίζουν τα κομμάτια που ακούνε. Παραδείγματα περιλαμβάνουν εφαρμογές για διαδικτυακούς ραδιοφωνικούς σταθμούς, όπου ο ακροατής μπορεί να πατήσει ένα κουμπί και να δει πληροφορίες για το τραγούδι στην οθόνη του κινητού του.
3. **Φιλτράρισμα μουσικού υλικού στο διαδίκτυο:** Τα audio fingerprints χρησιμοποιούνται για το φιλτράρισμα μουσικού υλικού που κυκλοφορεί στο διαδίκτυο μέσω του file sharing. Αυτό μπορεί να συμβάλει στην καταπολέμηση της πειρατείας μουσικής με την παρακολούθηση και καταγραφή των μουσικών κομματιών που διακινούνται.
4. **Οργάνωση μουσικών συλλογών από χρήστες:** Τα audio fingerprints χρησιμοποιούνται για την οργάνωση μουσικών συλλογών από τους χρήστες. Με τη βοήθεια ενός κατάλληλου προγράμματος, τα κομμάτια αναγνωρίζονται, εγγράφονται τα μεταδεδομένα τους και ονομάζονται τα αρχεία ανάλογα, επιτρέποντας τη σωστή οργάνωση της συλλογής και την παροχή των σωστών πληροφοριών.
5. Άλλες εφαρμογές: Τα audio fingerprints μπορούν να χρησιμοποιηθούν για πολλές άλλες εφαρμογές, όπως η εξέταση της ακεραιότητας μουσικών δεδομένων, η επαλήθευση της αντιστοίχισης μιας καλλιτεχνικής δουλειάς σε συγκεκριμένο καλλιτέχνη και άλλα παρόμοια παραδείγματα.

**ΤΕΧΝΟΛΟΓΙΕΣ AUDIO FINGERPRINTING**

**Shazam**

Το Shazam Entertainment Limited ανέπτυξε το σύστημα audio fingerprinting Shazam. Το σύστημα αξιοποιεί ένα ηχητικό απόσπασμα διάρκειας 15 δευτερολέπτων. Αξιοποιώντας μια συνδυαστική ανάλυση του ήχου σε σχέση με τον χρόνο και τη συχνότητα, ο αλγόριθμος δημιουργεί audio fingerprints μέσω μιας συνάρτησης κατακερματισμού (hash function), τα οποία αποθηκεύονται στη βάση δεδομένων.

Η ταχύτητα αναζήτηση στη βάση δεδομένων είναι υψηλή. Σε έναν προσωπικό υπολογιστή με μια βάση δεδομένων που περιέχει 20.000 μουσικά κομμάτια, ο χρόνος αναζήτησης κυμαίνεται από 5 έως 500 χιλιοστά του δευτερολέπτου, ανάλογα με τις παραμέτρους που έχουν ρυθμιστεί. Το σύστημα είναι ικανό να αναγνωρίζει ακόμη και πολύ χαμηλής ποιότητας ηχητικά αποσπάσματα σε λίγες εκατοντάδες χιλιοστά του δευτερολέπτου. Ακόμη, για ακουστικό υλικό “ραδιοφωνικής ποιότητας” , η αναγνώριση μπορεί να γίνει σε λιγότερο από 10 χιλιοστά του δευτερολέπτου.

Σύμφωνα με τη Shazam Entertainment Ltd, το σύστημα είναι αξιόπιστο, καθώς έχουν δείξει τα αποτελέσματα πειραμάτων σε μια βάση δεδομένων με 2 εκατομμύρια μουσικά κομμάτια. Ωστόσο, η εταιρεία αναφέρει ότι περιοδικά λαμβάνει αναφορές για λανθασμένες αναγνώρισεις, αν και το ποσοστό των λανθασμένων αναγνώρισεων έχει επιλεγεί κατάλληλα ως σχεδιαστική παράμετρος του συστήματος.

Ο αλγόριθμος είναι εξαιρετικά γρήγορος και μπορεί να χρησιμοποιηθεί για την παρακολούθηση πνευματικών δικαιωμάτων με μια ταχύτητα αναζήτησης που είναι πάνω από 1000 φορές γρηγορότερη από τον πραγματικό χρόνο, επιτρέποντας σε μια μετρίου επιπέδου διακομιστή να παρακολουθεί αποτελεσματικά πολλαπλές ροές πολυμέσων.

**AudioID**

Το σύστημα αναγνώρισης μουσικών κομματιών AudioID σχεδιάστηκε από το ινστιτούτο Fraunhofer. Το σύστημα χρησιμοποιεί 8 δευτερόλεπτα μουσικού αποσπάσματος και η δειγματοληψία γίνεται στα 44.1 kHz. Αυτές οι πληροφορίες συνεπάγονται ότι απαιτούνται 352800 δείγματα. Το αποτύπωμα που προκύπτει έχει μέγεθος 2.5 kBytes ανά λεπτό, δηλαδή 334 Bytes για τα 8 δευτερόλεπτα που απαιτούνται.

Η εξαγωγή του audio fingerprint βασίζεται στο AudioSpectrumFlatness Low Level Descriptor όπως ορίζει το πρότυπο MPEG-7. Για την εξαγωγή ενός audio fingerprint χρειάζονται μόλις κάποια milliseconds ανά αντικείμενο δεδομένου ότι είναι χαμηλής πολυπλοκότητας.

Η ταχύτητα αναζήτησης στη βάση δεδομένων είναι υψηλή. Σε βαθύτερη ανάλυση, ο χρόνος αναζήτησης ήταν 80 φορές μικρότερος του πραγματικού χρόνου, με τη χρήση ενός υπολογιστή με επεξεργαστή Pentium 500 MHz. Αυτή η μέτρηση αφορά σχετικά πειράματα του Fraunhofer που έγιναν με μια βάση αποτυπωμάτων 15000 audio fingerprints.

Για να ελεγχθεί η αξιοπιστία του συγκεκριμένου συστήματος, διεξήχθησαν κάποια πειράματα από το ινστιτούτο Fraunhofer. Τα πειράματα αυτά έγιναν με μια βάση δεδομένων 15000 αποτυπωμάτων, και η ακρίβεια της αναγνώρισης ήταν πάνω από 98%. Επιπλέον, σε μια άλλη σειρά πειραμάτων που πραγματοποιήθηκε με μια μεγαλύτερη βάση δεδομένων, περίπου 90000 αποτυπώματα, τα αποτελέσματα ήταν ακόμη καλύτερα, με την ακρίβεια της αναγνώρισης να φτάνει το 99%.

**TRM**

Η εταιρεία Relatable ανέπτυξε μια τεχνολογία με την ονομασία TRM. Αυτή η τεχνολογία παράγει ένα αποτύπωμα ήχου (audio fingerprint) με μέγεθος 512 bits και απαιτεί 30 δευτερόλεπτα ήχου για να το δημιουργήσει.

Όσον αφορά την αξιοπιστία του συστήματος, τα αποτελέσματα δεν ήταν τόσο υψηλά. Σύμφωνα με σχετικά πειράματα που διεξήχθησαν από την εταιρεία, ο ρυθμός σωστής αναγνώρισης έφτασε προσεγγιστικά το 73%. Το σύστημα έχει σχεδιαστεί κυρίως για χαμηλή πολυπλοκότητα παρά για υψηλή αξιοπιστία.

Όσον αφορά την αναζήτηση στη βάση δεδομένων, είναι γρήγορη. Σύμφωνα με την Relatable, σε μια βάση δεδομένων με πάνω από 5 εκατομμύρια TRM audio fingerprints που κατέχει η εταιρεία All Media Guide (AMG), είναι δυνατό να εξυπηρετηθούν πάνω από 5000 αναζητήσεις ανά δευτερόλεπτο.

Όσον αφορά τον τρόπο εξαγωγής των audio fingerprints, είναι γνωστό ότι το σύστημα έχει σχεδιαστεί με χαμηλή πολυπλοκότητα για την εξαγωγή αυτών των αποτυπωμάτων. Ωστόσο, η ακριβής μέθοδος εξαγωγής δεν έχει αποκαλυφθεί από την εταιρεία.

**MusicDNA**

Η εταιρεία Cantametrix ανέπτυξε το σύστημα MusicDNA. Η τεχνολογία αυτού του συστήματος χρησιμοποιεί ένα ηχητικό απόσπασμα διάρκειας 15 δευτερολέπτων, δειγματοληπτούμενο στα 11025 Hz, αποτελούμενο από 165376 δείγματα. Το audio fingerprint που εξάγεται έχει μέγεθος μικρότερο από 100 Bytes και αποτελεί ένα διάνυσμα 30 διαστάσεων.

Σε πειράματα που διεξήχθησαν από την Cantametrix για την αξιοπιστία του συστήματος, παρατηρήθηκε ότι με την αύξηση του μεγέθους της βάσης δεδομένων, αυξάνεται και ο αριθμός των false positives, δηλαδή τυχαίες ταυτοποιήσεις, ενώ τα false negatives παραμένουν σχεδόν αμετάβλητα.

Η αναζήτηση στη βάση δεδομένων είναι γρήγορη και απαιτεί λιγότερο από 300 milliseconds σε έναν υπολογιστή με 864 MHz για να αναζητηθεί ένα αποτύπωμα στη βάση δεδομένων.

Ο μέσος χρόνος εξαγωγής ενός audio fingerprint είναι 2 δευτερόλεπτα ανά τραγούδι. Η εξαγωγή του αποτυπώματος είναι χαμηλής πολυπλοκότητας και απαιτεί περιορισμένους υπολογιστικούς πόρους.

**R.A.R.E**

Η εταιρεία Microsoft σχεδίασε το σύστημα R.A.R.E (Robust Audio Recognition Engine) και χρησιμοποιεί έναν αλγόριθμο που ονομάζεται DDA (Distortion Discriminant Analysis) [14],[15] . Για την εξαγωγή του audio fingerprint χρησιμοποιεί ένα ηχητικό απόσπασμα διάρκειας 6 δευτερολέπτων, δειγματοληπτημένο στα 11025 Hz, αποτελούμενο από 66150 δείγματα. Το αποτύπωμα είναι ένα διάνυσμα πραγματικών αριθμών, διαστάσεως 64. Για την εξαγωγή του αποτυπώματος εφαρμόζεται δύο φορές ο μετασχηματισμός OPCA (Oriented Principal Component Analysis).

Η αξιοπιστία του συστήματος είναι υψηλή, όπως έδειξαν τα αποτελέσματα πειραμάτων και προσομοιώσεων της Microsoft. Ο ρυθμός false positive είναι 1.5 x ανά test clip και ο ρυθμός false negative είναι 0.2% ανά test clip, ανά αποτύπωμα της βάσης δεδομένων.

Σχετικά με την αναζήτηση στη βάση δεδομένων, η Microsoft χρησιμοποιεί μία δική της τεχνική . Πειράματα της Microsoft έδειξαν ότι σε μία βάση δεδομένων με 240000 audio fingerprints, ο χρόνος αναζήτησης μπορεί να φτάσει τα 0.577 δευτερόλεπτα για 1000 αναζητήσεις. Η πολυπλοκότητα της τεχνικής αυτής είναι γραμμική σε σχέση με το μέγεθος της βάσης δεδομένων και βελτιώνει το χρόνο αναζήτησης κατά ένα παράγοντα 56 σε σχέση με τον πραγματικό χρόνο.

Σύμφωνα με τη Microsoft, ο αλγόριθμος DDA που χρησιμοποιείται έχει χαμηλή υπολογιστική πολυπλοκότητα, αφού για την εξαγωγή του αποτυπώματος χρειάζεται μόνο το 1% του συνολικού χρόνου μίας αναγνώρισης. Το υπόλοιπο 99% του χρόνου αφιερώνεται στην αναζήτηση στη βάση δεδομένων[16]. Παρόλα αυτά, λόγω του διπλού μετασχηματισμού OPCA που αναφέρθηκε, η πολυπλοκότητα δεν είναι τόσο χαμηλή όσο ισχυρίζεται η εταιρεία.

**AudioDNA**

Μία εναλλακτική τεχνολογία για το audio fingerprinting είναι το AudioDNA [6]. Το σύστημα αυτό εξάγει ένα ακουστικό αποτύπωμα διαστάσεως 32, το οποίο περιέχει μια ακολουθία από 32 "γονίδια" βάσει της μεθόδου. Ωστόσο, η διαδικασία εξαγωγής αυτού του αποτυπώματος είναι χρονοβόρα. Απαιτούνται 77 δευτερόλεπτα για μία αναγνώριση, ενώ ο μέσος χρόνος διάρκειας ενός τραγουδιού είναι περίπου 3,5 λεπτά. Αυτό σημαίνει ότι η εξαγωγή γίνεται μόλις 3,12 φορές πιο γρήγορα από τον πραγματικό χρόνο.

Όσον αφορά την ταχύτητα αναζήτησης στη βάση δεδομένων, σε σχετικά πειράματα προέκυψε ότι είναι 50 φορές ταχύτερη από τον πραγματικό χρόνο. Μελετώντας ξανά το γεγονός ότι ο μέσος χρόνος διάρκειας ενός τραγουδιού είναι περίπου 3,5 λεπτά, σε μια βάση δεδομένων με 50.000 αποτυπώματα, απαιτήθηκαν 4,8 δευτερόλεπτα σε έναν υπολογιστή με επεξεργαστή Pentium III 993 MHz.

Όσον αφορά την αξιοπιστία του συστήματος, βάσει πειραμάτων που πραγματοποιήθηκαν από το ινστιτούτο που ανέπτυξε την τεχνολογία AudioDNA, αποδεικνύεται ότι είναι υψηλή.

**Philips**

Η εταιρεία Philips ανέπτυξε, επίσης, τη δική της τεχνολογία audio fingerprinting [17],[18],[19]. Το σύστημά της Philips χρησιμοποιεί ένα ηχητικό απόσπασμα διάρκειας 3.329 δευτερολέπτων, δειγματοληπτημένου στα 5515 Hz. Αυτό σημαίνει ότι χρειάζονται 18.368 δείγματα. Το audio fingerprint που εξάγεται αποτελείται από 256 sub-fingerprints, με μέγεθος 16 bits το καθένα, οπότε έχει συνολικό μέγεθος 8192 bits, δηλαδή ένα kilobyte.

Ο αλγόριθμος που χρησιμοποιείται για την αναζήτηση στη βάση δεδομένων είναι σχεδιασμένος από την Philips. Μία αναζήτηση σε μια βάση δεδομένων με εκατομμύρια αποτυπώματα διαρκεί μόνο μερικά milliseconds. Είναι εμφανές, λοιπόν, πόσο γρήγορη είναι η διαδικασία.

Όσον αφορά την πολυπλοκότητα της εξαγωγής των audio fingerprints, είναι αρκετά χαμηλή. Ο αλγόριθμος είναι υλοποιημένος με τη χρήση FFT (Fast Fourier Transform), άρα η μόνη απαιτητική πράξη είναι ο μετασχηματισμός Fourier.

Σε πειράματα που διεξήγαγε η Philips για να ελέγξει την αξιοπιστία του συστήματός της, αποδείχθηκε ότι ο ρυθμός false positive ήταν πολύ χαμηλός, συγκεκριμένα 3.6 x 10-20. Αυτό κατοχυρώνει ότι η αξιοπιστία του συστήματος είναι πολύ υψηλή.

Εκτός από τις παραπάνω τεχνολογίες audio fingerprinting, έχουν αναπτυχθεί και πολλές άλλες. Κάποιες από αυτές έχουν ευρεία χρήση, κάποιες χρησιμοποιούνται σε μικρότερη κλίμακα και κάποιες ακόμη βρίσκονται σε ερευνητικό στάδιο. Μερικές από αυτές είναι: Bing Music, Google Sound Search, Yahoo Music, Sony TrackID, AudioMatch, Tuneprint, SoundHound/Midomi, Echoprint, Moodlogic, Yacast, Auditude, Idioma, eTantrum, Chromaprint.

**ΤΟ ΣΥΣΤΗΜΑ ΠΟΥ ΧΡΗΣΙΜΟΠΟΙΕΙΤΑΙ**

Στο παρόν κεφάλαιο περιγράφεται ένα σύστημα που αναγνωρίζει ηχητικά αντικείμενα με βάση τα ακουστικά αποτυπώματα, χρησιμοποιώντας την τεχνολογία των audio fingerprints [43]. Αυτό το σύστημα έχει υλοποιηθεί σε Python και βασίζεται στη χρήση συναρτήσεων κατακερματισμού (Hash Functions) για τη δημιουργία των αποτυπωμάτων. Αντίστοιχες τεχνολογίες χρησιμοποιούνται σε εφαρμογές όπως το Shazam, το SoundHound και άλλες. Παρακάτω περιγράφονται οι διαδικασίες εξαγωγής των ακουστικών αποτυπωμάτων καθώς και η μέθοδος αναζήτησης στη βάση δεδομένων.

**ΕΞΑΓΩΓΗ ΤΟΥ AUDIO FINGERPRINT**

**Ο ήχος ως σήμα**

Η πρώτη φάση στη διαδικασία εξαγωγής των Audio Fingerprints απαιτεί τον καθορισμό και την καταγραφή του ληφθέντος σήματος. Ο ήχος, όπως έχει αποδειχθεί, κωδικοποιείται ψηφιακά ως μια μακρά λίστα αριθμών. Σε ένα αρχείο μη συμπιεσμένου τύπου .wav, υπάρχουν πολλοί από αυτούς τους αριθμούς, συγκεκριμένα 44100 ανά δευτερόλεπτο ανά κανάλι. Καθένας από αυτούς τους αριθμούς αντιστοιχεί σε ένα δείγμα ήχου. Για την ακρίβεια, ένα κανάλι αντιπροσωπεύει μια ξεχωριστή ακολουθία δειγμάτων που μπορεί να αναπαράγει ένα ηχείο. Σε μια διάταξη με δύο κανάλια, χρησιμοποιούνται δύο ηχεία. Ένα κανάλι μόνο ονομάζεται "μονοφωνικό". Σήμερα, τα σύγχρονα συστήματα ήχου μπορούν να υποστηρίξουν πολλά περισσότερα κανάλια. Ωστόσο, εάν ο ήχος δεν καταγραφεί ή δεν αναμιχθεί με ίσο αριθμό καναλιών, τα επιπλέον ηχεία είναι περιττά και ορισμένα απλά επαναλαμβάνουν το ίδιο ήχο με άλλα ηχεία. Υποθέτετέ ότι ο μέσος χρόνος διάρκειας των ηχογραφήσεων είναι 5 δευτερόλεπτα, πράγμα που σημαίνει ότι ένα αρχείο διάρκειας 5 δευτερολέπτων περιέχει περίπου 450 χιλιάδες δείγματα.

5 δευτερόλεπτα \* 44100 δείγματα ανά δευτερόλεπτο \* 2 κανάλια = 441.000 δείγματα

**Δειγματοληψία**

Το δεύτερο στάδιο στη διαδικασία, μετά τον καθορισμό του σήματος ως έναν ακριβή αριθμό δειγμάτων, είναι η σωστή δειγματοληψία. Η επιλογή των 44100 δειγμάτων ανά δευτερόλεπτο φαίνεται αυθαίρετη, αλλά σχετίζεται με το θεώρημα Nyquist-Shannon Sampling. Βάσει αυτού του θεωρήματος, καθορίζεται ένα θεωρητικό όριο στη μέγιστη συχνότητα που μπορεί να καταγραφεί με ακρίβεια κατά τη δειγματοληψία. Αυτό το όριο βασίζεται στο πόσο γρήγορα "δειγματοληπτείται" το σήμα.

Για να γίνει πιο κατανοητό, μπορούμε να αναφέρουμε ένα απλό παράδειγμα: Ας υποθέσουμε ότι παρακολουθούμε μια λεπίδα ανεμιστήρα που κάνει μια πλήρη περιστροφή ανά δευτερόλεπτο (1 Hz). Κρατώντας τα μάτια κλειστά και ανοίγοντάς τα μία φορά ανά δευτερόλεπτο, θα φαίνεται σαν να μην έχει μετακινηθεί η λεπίδα. Κάθε φορά που ανοίγουμε τα μάτια, η λεπίδα βρίσκεται στην ίδια θέση. Ωστόσο, στην πραγματικότητα η λεπίδα μπορεί να έχει κάνει οποιοδήποτε αριθμό περιστροφών ανά δευτερόλεπτο, χωρίς να είναι αντιληπτό από τον παρατηρητή. Για να είμαστε βέβαιοι ότι γίνονται σωστές δειγματοληψίες υψηλότερων συχνοτήτων, η δειγματοληψία πρέπει να γίνεται πιο συχνά. Συγκεκριμένα, πρέπει η δειγματοληψία να γίνεται διπλάσια της μέγιστης συχνότητας που επιθυμούμε να καταγράψουμε, ώστε να υπάρχει η βεβαιότητα ότι ανιχνεύουμε σωστά τη συχνότητα.

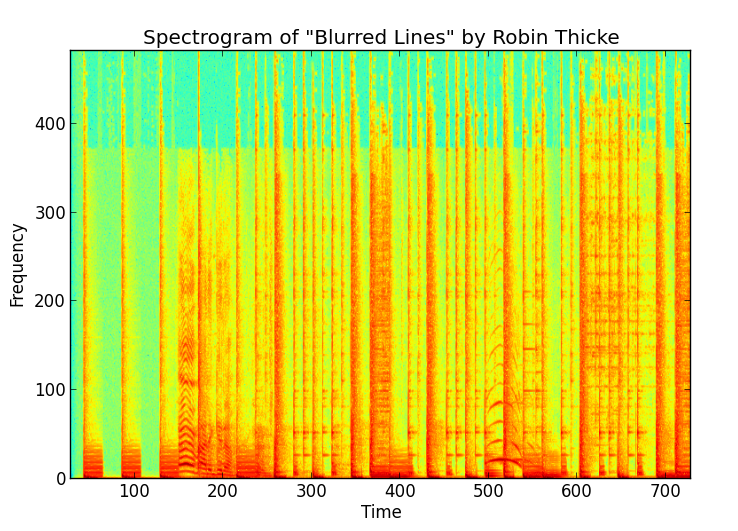
Στην περίπτωση της εγγραφής ήχου, αποδεκτή είναι η απώλεια συχνοτήτων υψηλότερων από 22050 Hz, καθώς το ανθρώπινο αυτί αδυνατεί να αντιληφθεί συχνότητες πάνω από 20.000 Hz. Έτσι, βάσει του θεωρήματος Nyquist, η δειγματοληψία γίνεται με συχνότητα διπλάσια της μέγιστης συχνότητας, δηλαδή :

Απαιτούμενα δείγματα ανά δευτερόλεπτο = Υψηλότερη συχνότητα \* 2 = 22050 \* 2 = 44100

Η μορφή MP3 συμπιέζει αυτό το νούμερο για να εξοικονομήσει χώρο στον σκληρό δίσκο, αλλά ένα καθαρό αρχείο μορφοποίησης .wav στον υπολογιστή είναι απλώς μια λίστα αριθμών 16 bit (με μια μικρή κεφαλίδα).

**Φασματογράφημα**

Το τρίτο στάδιο στη διαδικασία είναι η δημιουργία ενός φασματογραφήματος βασισμένου στα ορθώς δειγματοληπτημένα δείγματα από το προηγούμενο στάδιο. Καθώς αυτά τα δείγματα δεν είναι πολύ ακριβής, εφαρμόζεται επαναλαμβανόμενη μετασχηματισμός Fourier (FFT) σε μικρά παράθυρα χρόνου στα δείγματα ήχου για να δημιουργηθεί ένα φασματογράφημα. Στο παράδειγμα που ακολουθεί, παρουσιάζεται ένα φασματογράφημα για τα πρώτα δευτερόλεπτα ενός τυχαίου ήχου.



**Εικόνα 3: Φασματογράφημα**

Το φασματογράφημα είναι ένας πίνακας 2D, όπου το πλάτος ο πλάτος είναι συνάρτηση του χρόνου και της συχνότητας. Η FFT δείχνει την ισχύ (πλάτος) του σήματος για κάθε συχνότητα, παρέχοντας μια στήλη στο φασματογράφημα. Αν επαναληφθεί αυτή η διαδικασία αρκετές φορές, μετακινώντας το παράθυρο της FFT και συνδυάζοντας τις στήλες, δημιουργείται ένα πλήρες φασματογράφημα σε μορφή 2D πίνακα. Αν η εγγραφή αφορά έναν μόνο ηχητικό τόνο, το φασματογράφημα θα πρέπει να έχει μια οριζόντια γραμμή για τη συχνότητα του τόνου, καθώς η συχνότητα δεν αλλάζει από παράθυρο σε παράθυρο.

Είναι σημαντικό να σημειωθεί ότι οι τιμές της συχνότητας και του χρόνου είναι διακριτοποιημένες, αναπαριστώντας κάθε ένα διάστημα, ενώ το πλάτος παίρνει πραγματικές τιμές. Το χρώμα αναπαριστά την πραγματική τιμή του πλάτους (κόκκινο για υψηλό, πράσινο για χαμηλό) στο συγκεκριμένο διάνυσμα διακριτοποιημένων συντεταγμένων (χρόνος, συχνότητα).

Το φασματογράφημα χρησιμοποιείται για την αναγνώριση και μοναδική προσδιορισμό του ήχου. Ένα πρόβλημα που πρέπει να αντιμετωπιστεί είναι η απομόνωση του θορύβου και η εξαγωγή μοναδικών αποτυπωμάτων από το ηχητικό σήμα.

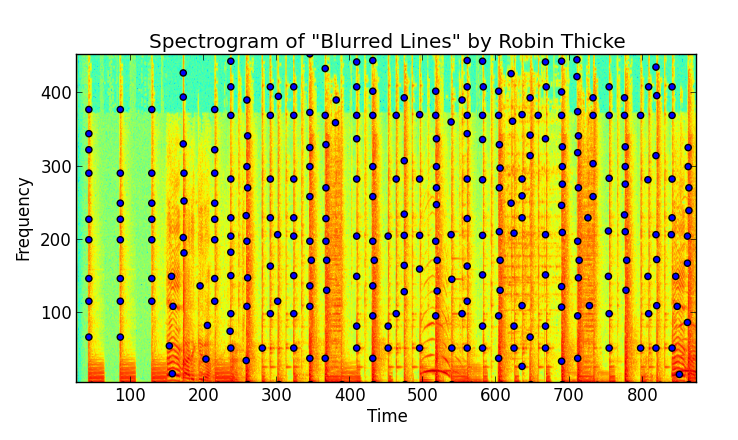
**Ανεύρεση κορυφών**

Το τέταρτο στάδιο στη διαδικασία είναι η εξαγωγή των κορυφών από το προηγούμενο φασματογράφημα, προσφέροντας λύση στο προηγούμενο πρόβλημα. Αφού δημιουργηθεί το φασματογράφημα του ηχητικού σήματος, το επόμενο βήμα είναι να εντοπιστούν οι "κορυφές" του πλάτους. Μια κορυφή ορίζεται ως ένα ζεύγος (χρόνος, συχνότητα) που αντιστοιχεί σε μια τιμή πλάτους που είναι η μεγαλύτερη σε μια περιοχή γύρω από αυτήν. Άλλα ζεύγη (χρόνου, συχνότητας) γύρω από αυτό έχουν χαμηλότερη τιμή πλάτους και επομένως είναι λιγότερο πιθανό να επιβιώσουν από το θόρυβο.

Η εύρεση των κορυφών είναι μία απαιτητική διαδικασία. Στο συγκεκριμένο σύστημα, το φασματογράφημα αντιμετωπίζεται ως εικόνα και χρησιμοποιούνται εργαλεία και τεχνικές επεξεργασίας εικόνας από την βιβλιοθήκη scipy για την εντοπισμό των κορυφών. Χρησιμοποιώντας ένα υψηπερατό φίλτρο που τονίζει τα υψηλά πλάτη και δομές τοπικών μέγιστων της scipy, απομονώνονται οι "δυνατές" κορυφές στο φασματογράφημα.

Αυτές οι ανθεκτικές στο θόρυβο κορυφές είναι τα σημεία ενδιαφέροντος στο εκάστοτε ηχητικό αντικείμενο και το χαρακτηρίζουν. Μετά τον εντοπισμό των κορυφών, το φασματογράφημα δεν χρησιμοποιείται πλέον. Οι κορυφές έχουν εξυπηρετήσει τον σκοπό τους.

Στην παρακάτω εικόνα φαίνεται η επιλογή των κορυφών για το προηγούμενο φασματογράφημα:

 **Εικόνα 4:** Εντοπισμός κορυφών σε φασματογράφημα

Είναι ευδιάκριτο ότι υπάρχουν πολλές κορυφές, δεκάδες χιλιάδες ανά ηχητικό αντικείμενο για την ακρίβεια. Μέσω αυτής της διαδικασίας, έχει απομονωθεί όλη η μέχρι τώρα σημαντική πληροφορία σε δύο μεταβλητές: τον χρόνο και τη συχνότητα, οι οποίες έχουν μετατραπεί σε διακριτές ακέραιες τιμές.

Με αυτόν τον τρόπο δημιουργείται ένα σύστημα που καταγράφει τις κορυφές ενός σήματος σε διακριτά ζεύγη (χρόνος, συχνότητα), δίνοντάς κάποια περιθώρια επιβίωσης στον θόρυβο. Η πληροφορία του σήματος μετατρέπεται σε διακριτή μορφή και, ως αποτέλεσμα, η πληροφορία των κορυφών μειώνεται από άπειρη σε πεπερασμένη, γεγονός που σημαίνει ότι οι κορυφές που εντοπίστηκαν σε ένα ηχητικό αντικείμενο μπορεί να συμπίπτουν, εξάγοντας τα ίδια ζεύγη που απομονώθηκαν από άλλα αντικείμενα.

**Fingerprint Hashing**

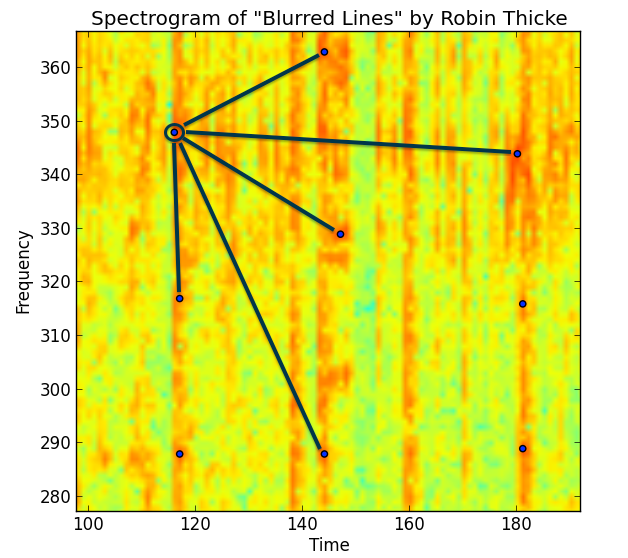
Τέλος, στο πέμπτο στάδιο της διαδικασίας, αξιοποιούνται οι κορυφές που εξήχθησαν για τη δημιουργία fingerprints μέσω μιας συνάρτησης κατακερματισμού (hash function).

Η συνάρτηση κατακερματισμού επιτρέπει τον συνδυασμό των κορυφών σε fingerprints, εξαλείφοντας το ενδεχόμενο εντοπισμού παρόμοιων κορυφών. Αναλαμβάνει να παράγει έναν ακέραιο αριθμό ως αποτέλεσμα, από μια εισαγωγή ακέραιων αριθμών. Μια καλή συνάρτηση κατακερματισμού δεν παράγει μόνο την ίδια έξοδο για την ίδια είσοδο, αλλά έχει επίσης έναν πολύ μικρό αριθμό διαφορετικών εισόδων που δίνουν την ίδια έξοδο.

Με βάση τις κορυφές του φασματογραφήματος και συνδυάζοντας τις συχνότητες και τις χρονικές διαφορές τους, δημιουργείται ένα hash που αντιπροσωπεύει ένα μοναδικό αποτύπωμα για κάθε ηχητικό αντικείμενο.

hash (συχνότητες κορυφών, χρονική διαφορά μεταξύ κορυφών) = fingerprint τιμή hash

Υπάρχουν πολλοί τρόποι υλοποίησης αυτής της διαδικασίας, με το Shazam να χρησιμοποιεί το δικό του σύστημα, ενώ και το SoundHound χρησιμοποιεί ένα διαφορετικό. Λαμβάνοντας υπόψη περισσότερες από μία τιμές κορυφών, δημιουργούνται fingerprints που περιέχουν περισσότερη πληροφορία και έχουν μεγαλύτερη εντροπία. Αυτό τους καθιστά πιο αποτελεσματικούς στην αναγνώριση του ήχου, καθώς η πιθανότητα "σύγκρουσης" είναι μικρότερη. Ωστόσο, η χρήση περισσότερων κορυφών οδηγεί και σε λιγότερο ισχυρά fingerprints λόγω του θορύβου που μπορεί να εισαχθεί.



**Εικόνα 5:** Ζουμ σε κορυφές φασματογραφήματος

Το Shazam, στο επίσημο "οδηγό" του (white paper), αναφέρει αυτές τις ομάδες κορυφών ως ένα είδος "αστερισμού" κορυφών που χρησιμοποιούνται για την αναγνώριση του τραγουδιού. Στην πραγματικότητα, χρησιμοποιούνται ζεύγη κορυφών μαζί με τη χρονική τους διαφορά. Υπάρχουν διάφορες μέθοδοι για την ομαδοποίηση των σημείων και τη δημιουργία των fingerprints. Περισσότερες κορυφές σε ένα fingerprint σημαίνει ένα πιο σπάνιο fingerprint, το οποίο μπορεί να αναγνωρίσει πολύ καλύτερα ένα τραγούδι. Ωστόσο, η χρήση περισσότερων κορυφών οδηγεί επίσης σε ένα λιγότερο ισχυρό fingerprint λόγω του θορύβου που μπορεί να εισαχθεί.

**Προσομοίωση του Audio Fingerprinting**

Η υλοποίηση του συγκεκριμένου project έγινε με VM σε ubuntu3.8 λόγο προβλήματος που εμφανίζεται να έχει το συγκεκριμένο git με τις καινούργιες εκδόσεις του ubuntu δημιουργήσαμε ένα περιβάλλον Conda όπου μας επιτρέπει αν εφαρμόσουμε συγκεκριμένο λογισμικό σε ένα παράθυρο της επιλογής μας .Πιο συγκεκριμένα για να τρέξει το συγκεκριμένο git χρειάστηκε να το τρέξουμε σε ubuntu2.7 και με τις απαραίτητες βιβλιοθήκες και προγράμματα. Υλοποιούνται τα παρακάτω βήματα στο Conda. Για την δημιουργία ενός τέτοιου περιβάλλοντος, αφού το έχουμε κατεβάσει. Θα πρέπει να εκτελεστεί η εντολή με το name της επιλογής μας.

**conda create ‘name’ python==2.7 ‘packages’ matplotlib termcolor scipy pydub pyaudio ffmpeg**

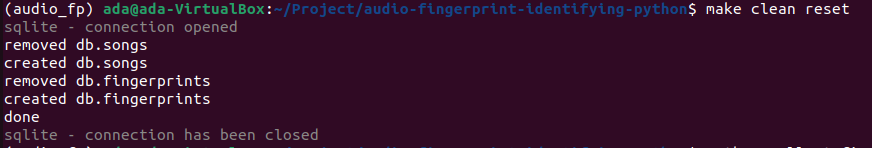
Επιπλέον χρησιμοποιήθηκε ένα git clone

**git clone** [**https://github.com/itspoma/audio-fingerprint-identifying-python.git**](https://github.com/itspoma/audio-fingerprint-identifying-python.git)

για να κατεβάσει τα αρχεία από το Github της επιλογής μας . Για να εκτελεστεί αυτήν η εντολή πρώτα θα πρέπει να έχει εγκατασταθεί η βιβλιοθήκη “git” με την παρακάτω εντολή.

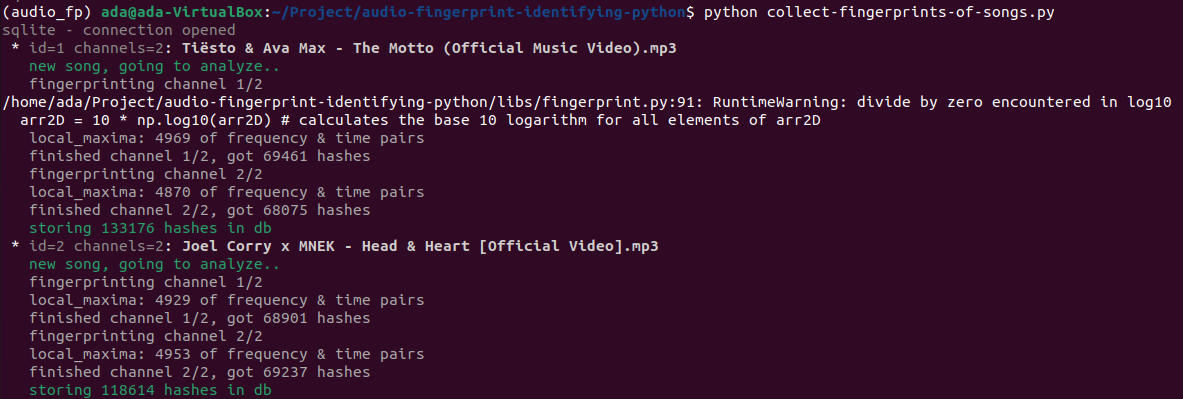
**sudo apt install git**

Προχωρώντας στο φάκελο που έχουμε δημιουργήσει με το όνομα της επιλογής μας θα βλέπουμε δεξιά ότι θα υπάρχει το όνομα αυτό ,που θα συμβολίζει ότι βρισκόμαστε μέσα στο παράθυρο Coda που έχουμε δημιουργήσει με τα συγκεκριμένα χαρακτηριστικά που χρειαζόμαστε , με την εντολή “**make clean reset**” δημιουργεί εκ νέου την βάση δεδομένων διαγράφοντας τα παλιά finger prints που έχουν δημιουργηθεί .



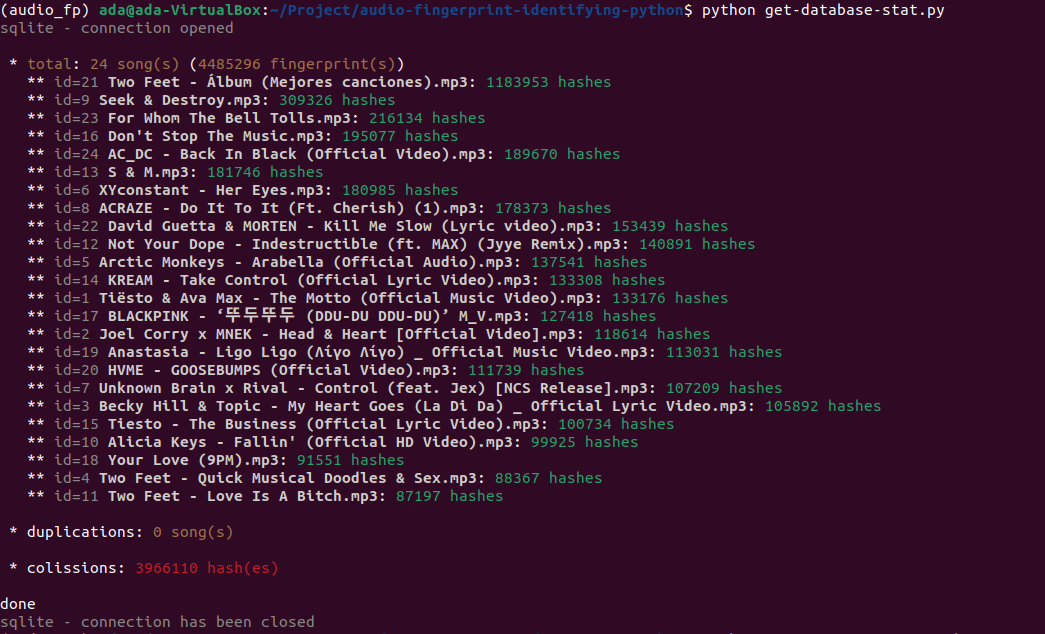
Θα χρειαστεί αν δεν υπάρχουν τραγούδια στη βάση, τότε θα χρειαστεί να προστεθούν. Δημιουργώντας ένα φάκελο και προσθέτοντας μέσα σε αυτόν τραγούδια της επιλογής μας

Έπειτα με την εντολή “ **python collect-fingerprints-of-songs.py**” ξεκινάει η διαδικασία δημιουργίας των ακουστικών αποτυπωμάτων από τα τραγούδια που έχουμε εφαρμόσει και θα τα αποθηκεύσει στην βάση δεδομένων που έχουμε εφαρμόσει .Η διαδικασία αυτή παίρνει λίγο χρόνο ανάλογα με τη ποσότητα των κομματιών που έχει η βάση.

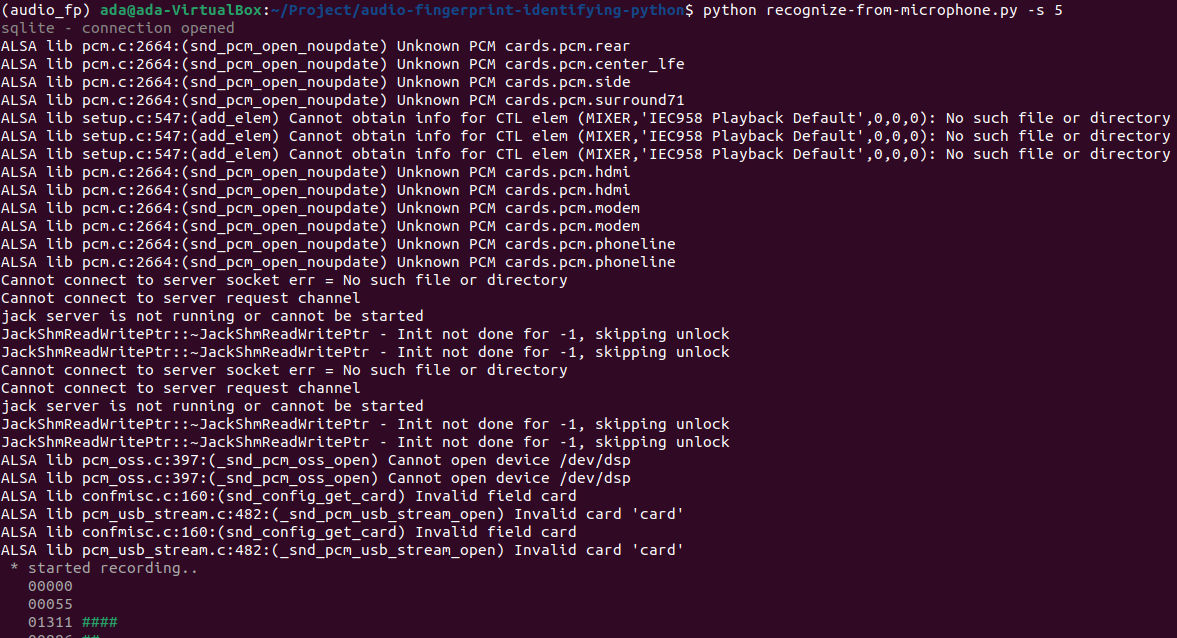


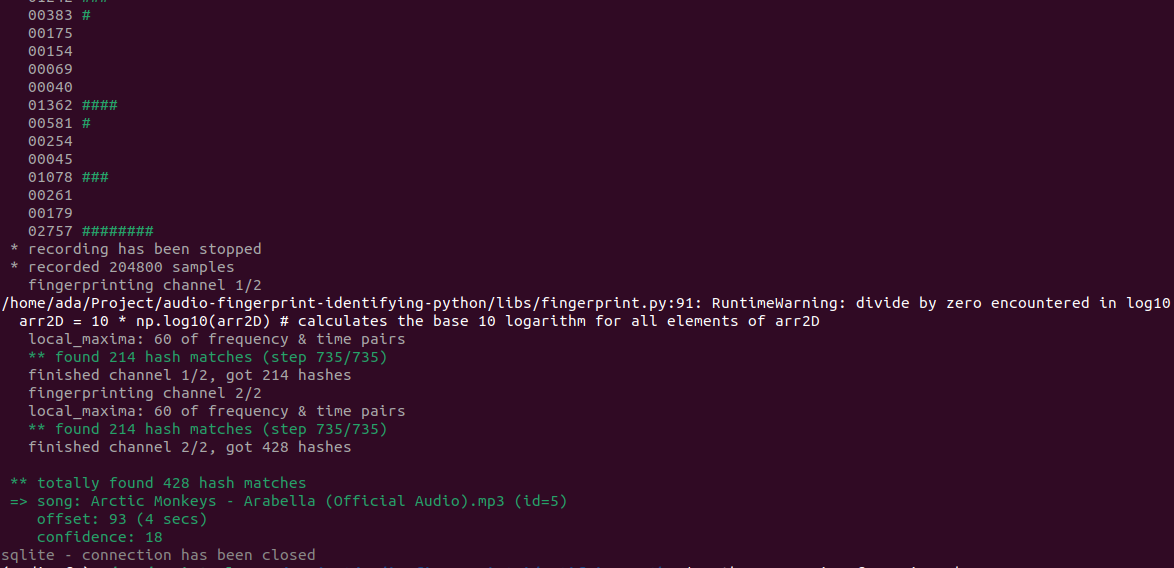


Εκτελώντας την εντολή **“ python get-database-stat.py**” εμφανίζει τα δεδομένα της βάσης.



Φτάνοντας πια στο τελευταίο σκέλος του project με την εντολή “**python recognize-fom-microphone.py -s 5**” εκτελείτε ο κώδικας για την αναγνώριση τραγουδιού για πέντε δευτερόλεπτα ηχογράφησης. Ανοίγοντας το μικρόφωνο μας και καταγράφοντας τον ήχο που του βάζουμε να αναγνωρίσει . Τέλος αν τα ηχητικά αποτυπώματα του ήχου που του βάζουμε να καταγράψει ταιριάζει με κάποια από τα ηχητικά αποτυπώματα που έχουμε στην βάση δεδομένων μας θα μας εμφανίσει το τραγούδι αυτό εκτυπώνοντας το .



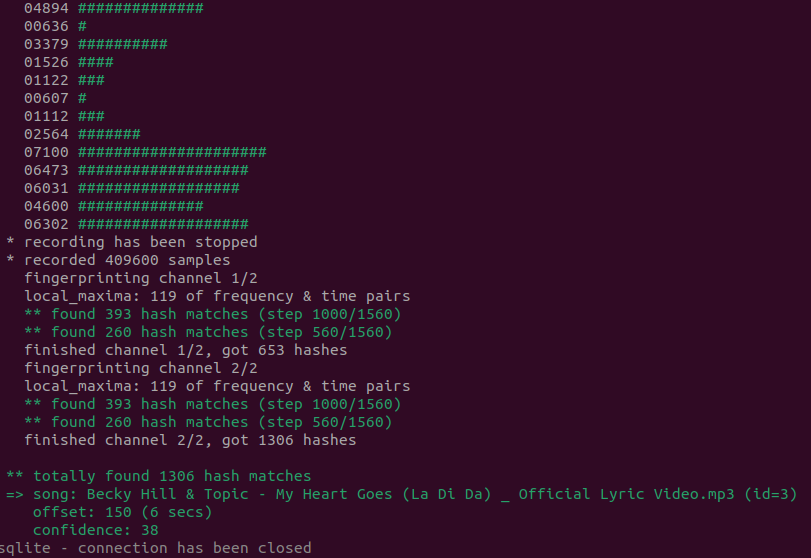


Διεξαχθήκαν κάποια πειράματα με αλλαγή χρόνου καταγραφής των 10, 5 και 3 δευτερολέπτων. Με αποτέλεσμα ήταν να αναγνωρίζει με επιτυχία το τραγούδι.

Για 10 δευτερόλεπτα το αναγνωρίζει 100%.

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης

Περιγραφή που δημιουργήθηκε αυτόματα



Για 5 δευτερόλεπτα το αναγνωρίζει 100%.

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά

Περιγραφή που δημιουργήθηκε αυτόματα

Για 3 δευτερόλεπτα το αναγνωρίζει 100%.

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά

Περιγραφή που δημιουργήθηκε αυτόματα

Επίσης πραγματοποιήθηκε και δοκιμή για τραγούδι που δεν υπάρχει στη βάση δεδομένων. Το αποτέλεσμα είναι να μην αντιστοιχίσει το τραγούδι σε κάποιες περιπτώσεις , αλλά και να το αντιστοιχήσει με κάποιο και αυτά . Αυτό συμβαίνει όπως αναλύσαμε παραπάνω στην θεωρεία ότι ουσιαστικά όσο αυξάνετε το μέγεθος της βάσεις δεδομένων που έχουμε τόσο υπάρχει μεγάλη πιθανότητα να συμβεί ένα falshe positive που είναι η καταχώρηση ενός αγνώστου ηχητικού αποτυπώματος για την βάση μας ,εξάγοντας θετικά αποτελέσματα δηλαδή την αντιστοίχιση με κάποιο κομμάτι στην βάση μας.

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά

Περιγραφή που δημιουργήθηκε αυτόματα

**ΒΙΒΛΙΟΓΡΑΦΙΑ**

[1]:<https://www.researchgate.net/publication/228347102_A_Review_of_Audio_Fingerprinting>

[2]:<https://ismir2002.ismir.net/proceedings/02-FP04-2.pdf>

[3]:<https://www.researchgate.net/publication/236623357_Architectures_and_Performance_for_Network_Coding>

[4] S. Z. Li, ‘Content-based audio classification and retrieval using the nearest feature line method’, IEEE Trans. Speech Audio Process., vol. 8, no. 5, pp. 619–625, Sep. 2000

[5] Guodong Guo and S. Z. Li, ‘Content-based audio classification and retrieval by support vector machines’, IEEE Trans. Neural Netw., vol. 14, no. 1, pp. 209–215, Jan. 2003.

[6] <https://eurasip.org/Proceedings/Eusipco/Eusipco2004/defevent/papers/cr1091.pdf#[0,{%22name%22:%22XYZ%22},null,null,0>]

[7] <https://ismir2002.ismir.net/proceedings/02-FP04-2.pdf>

[8] <file:///C:/Users/user/Downloads/Meccanismiperilriconoscimentoautomatizzatodibranimusicali.pdf>

[9] <https://www.researchgate.net/publication/228347102_A_Review_of_Audio_Fingerprinting>

[10] <https://ieeexplore.ieee.org/document/1203274>

[11] <https://www.academia.edu/17450889/Robust_sound_modeling_for_song_detection_in_broadcast_audio>

[12] <https://www.researchgate.net/publication/2537182_Automatic_Song_Identification_in_Noisy_Broadcast_Audio>

[13] <https://ismir2002.ismir.net/proceedings/02-FP04-2.pdf>

[14] C. J. C. Burges, J. C. Platt, and S. Jana, ‘Distortion discriminant analysis for audio fingerprinting’, IEEE Trans. Speech Audio Process., vol. 11, no. 3, pp.165–174, May 2003

[15] C. J. C. Burges, J. C. Platt, and S. Jana, ‘Extracting noise-robust features from audio data’, in IEEE International Conference on Acoustics Speech and Signal Processing,Orlando, FL, USA, 2002, pp. I-1021-I–1024.

[16] <https://www.researchgate.net/publication/2923068_Indexing_High_Dimensional_Rectangles_for_Fast_Multimedia_Identification>

[17] <https://ieeexplore.ieee.org/document/7079698>

[18] <https://www.researchgate.net/publication/220723736_Extracting_Quality_Parameters_for_Compressed_Audio_from_Fingerprints>

[19] <https://www.semanticscholar.org/paper/A-Highly-Robust-Audio-Fingerprinting-System-Haitsma-Kalker/4f92768276a0823cffeb9435ccda67beaca1f542>

[20] <https://willdrevo.com/fingerprinting-and-audio-recognition-with-python/>

<http://artemis.cslab.ece.ntua.gr:8080/jspui/handle/123456789/17436>

<https://github.com/itspoma/audio-fingerprint-identifying-python.git>

**recognize-from-microphone.py**

#!/usr/bin/python  
import os  
import sys  
import libs  
import libs.fingerprint as fingerprint  
import argparse  
  
from argparse import RawTextHelpFormatter  
from itertools import izip\_longest  
from termcolor import colored  
from libs.config import get\_config  
from libs.reader\_microphone import MicrophoneReader  
from libs.visualiser\_console import VisualiserConsole as visual\_peak  
from libs.visualiser\_plot import VisualiserPlot as visual\_plot  
from libs.db\_sqlite import SqliteDatabase  
# from libs.db\_mongo import MongoDatabase  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 config = get\_config()  
  
 db = SqliteDatabase()  
  
 parser = argparse.ArgumentParser(formatter\_class=RawTextHelpFormatter)  
 parser.add\_argument('-s', '--seconds', nargs='?')  
 args = parser.parse\_args()  
  
 if not args.seconds:  
 parser.print\_help()  
 sys.exit(0)  
  
 seconds = int(args.seconds)  
  
 chunksize = 2\*\*12 # 4096  
 channels = 2#int(config['channels']) # 1=mono, 2=stereo  
  
 record\_forever = False  
 visualise\_console = bool(config['mic.visualise\_console'])  
 visualise\_plot = bool(config['mic.visualise\_plot'])  
  
 reader = MicrophoneReader(None)  
  
 reader.start\_recording(seconds=seconds,  
 chunksize=chunksize,  
 channels=channels)  
  
 msg = ' \* started recording..'  
 print colored(msg, attrs=['dark'])  
  
 while True:  
 bufferSize = int(reader.rate / reader.chunksize \* seconds)  
  
 for i in range(0, bufferSize):  
 nums = reader.process\_recording()  
  
 if visualise\_console:  
 msg = colored(' %05d', attrs=['dark']) + colored(' %s', 'green')  
 print msg % visual\_peak.calc(nums)  
 else:  
 msg = ' processing %d of %d..' % (i, bufferSize)  
 print colored(msg, attrs=['dark'])  
  
 if not record\_forever: break  
  
 if visualise\_plot:  
 data = reader.get\_recorded\_data()[0]  
 visual\_plot.show(data)  
  
 reader.stop\_recording()  
  
 msg = ' \* recording has been stopped'  
 print colored(msg, attrs=['dark'])  
  
  
  
 def grouper(iterable, n, fillvalue=None):  
 args = [iter(iterable)] \* n  
 return (filter(None, values) for values  
 in izip\_longest(fillvalue=fillvalue, \*args))  
  
 data = reader.get\_recorded\_data()  
  
 msg = ' \* recorded %d samples'  
 print colored(msg, attrs=['dark']) % len(data[0])  
  
 # reader.save\_recorded('test.wav')  
  
  
 Fs = fingerprint.DEFAULT\_FS  
 channel\_amount = len(data)  
  
 result = set()  
 matches = []  
  
 def find\_matches(samples, Fs=fingerprint.DEFAULT\_FS):  
 hashes = fingerprint.fingerprint(samples, Fs=Fs)  
 return return\_matches(hashes)  
  
 def return\_matches(hashes):  
 mapper = {}  
 for hash, offset in hashes:  
 mapper[hash.upper()] = offset  
 values = mapper.keys()  
  
 for split\_values in grouper(values, 1000):  
 # @*todo move to db related files* query = """  
 SELECT upper(hash), song\_fk, offset  
 FROM fingerprints  
 WHERE upper(hash) IN (%s)  
 """  
 query = query % ', '.join('?' \* len(split\_values))  
  
 x = db.executeAll(query, split\_values)  
 matches\_found = len(x)  
  
 if matches\_found > 0:  
 msg = ' \*\* found %d hash matches (step %d/%d)'  
 print colored(msg, 'green') % (  
 matches\_found,  
 len(split\_values),  
 len(values)  
 )  
 else:  
 msg = ' \*\* not matches found (step %d/%d)'  
 print colored(msg, 'red') % (  
 len(split\_values),  
 len(values)  
 )  
  
 for hash, sid, offset in x:  
 # (sid, db\_offset - song\_sampled\_offset)  
 yield (sid, offset - mapper[hash])  
  
 for channeln, channel in enumerate(data):  
 # *TODO: Remove prints or change them into optional logging.* msg = ' fingerprinting channel %d/%d'  
 print colored(msg, attrs=['dark']) % (channeln+1, channel\_amount)  
  
 matches.extend(find\_matches(channel))  
  
 msg = ' finished channel %d/%d, got %d hashes'  
 print colored(msg, attrs=['dark']) % (  
 channeln+1, channel\_amount, len(matches)  
 )  
  
 def align\_matches(matches):  
 diff\_counter = {}  
 largest = 0  
 largest\_count = 0  
 song\_id = -1  
  
 for tup in matches:  
 sid, diff = tup  
  
 if diff not in diff\_counter:  
 diff\_counter[diff] = {}  
  
 if sid not in diff\_counter[diff]:  
 diff\_counter[diff][sid] = 0  
  
 diff\_counter[diff][sid] += 1  
  
 if diff\_counter[diff][sid] > largest\_count:  
 largest = diff  
 largest\_count = diff\_counter[diff][sid]  
 song\_id = sid  
  
 songM = db.get\_song\_by\_id(song\_id)  
  
 nseconds = round(float(largest) / fingerprint.DEFAULT\_FS \*  
 fingerprint.DEFAULT\_WINDOW\_SIZE \*  
 fingerprint.DEFAULT\_OVERLAP\_RATIO, 5)  
  
 return {  
 "SONG\_ID" : song\_id,  
 "SONG\_NAME" : songM[1],  
 "CONFIDENCE" : largest\_count,  
 "OFFSET" : int(largest),  
 "OFFSET\_SECS" : nseconds  
 }  
  
 total\_matches\_found = len(matches)  
  
 print ''  
  
 if total\_matches\_found > 0:  
 msg = ' \*\* totally found %d hash matches'  
 print colored(msg, 'green') % total\_matches\_found  
  
 song = align\_matches(matches)  
  
 msg = ' => song: %s (id=%d)\n'  
 msg += ' offset: %d (%d secs)\n'  
 msg += ' confidence: %d'  
  
 print colored(msg, 'green') % (  
 song['SONG\_NAME'], song['SONG\_ID'],  
 song['OFFSET'], song['OFFSET\_SECS'],  
 song['CONFIDENCE']  
 )  
 else:  
 msg = ' \*\* not matches found at all'  
 print colored(msg, 'red')

**collect-fingerprints-of-songs.py**

#!/usr/bin/python  
import os  
import sys  
import libs  
import libs.fingerprint as fingerprint  
  
from termcolor import colored  
from libs.reader\_file import FileReader  
from libs.db\_sqlite import SqliteDatabase  
from libs.config import get\_config  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 config = get\_config()  
  
 db = SqliteDatabase()  
 path = "mp3/"  
  
 # fingerprint all files in a directory  
  
 for filename in os.listdir(path):  
 if filename.endswith(".mp3"):  
 reader = FileReader(path + filename)  
 audio = reader.parse\_audio()  
  
 song = db.get\_song\_by\_filehash(audio['file\_hash'])  
 song\_id = db.add\_song(filename, audio['file\_hash'])  
  
 msg = ' \* %s %s: %s' % (  
 colored('id=%s', 'white', attrs=['dark']), # id  
 colored('channels=%d', 'white', attrs=['dark']), # channels  
 colored('%s', 'white', attrs=['bold']) # filename  
 )  
 print msg % (song\_id, len(audio['channels']), filename)  
  
 if song:  
 hash\_count = db.get\_song\_hashes\_count(song\_id)  
  
 if hash\_count > 0:  
 msg = ' already exists (%d hashes), skip' % hash\_count  
 print colored(msg, 'red')  
  
 continue  
  
 print colored(' new song, going to analyze..', 'green')  
  
 hashes = set()  
 channel\_amount = len(audio['channels'])  
  
 for channeln, channel in enumerate(audio['channels']):  
 msg = ' fingerprinting channel %d/%d'  
 print colored(msg, attrs=['dark']) % (channeln+1, channel\_amount)  
  
 channel\_hashes = fingerprint.fingerprint(channel, Fs=audio['Fs'], plots=config['fingerprint.show\_plots'])  
 channel\_hashes = set(channel\_hashes)  
  
 msg = ' finished channel %d/%d, got %d hashes'  
 print colored(msg, attrs=['dark']) % (  
 channeln+1, channel\_amount, len(channel\_hashes)  
 )  
  
 hashes |= channel\_hashes  
  
 msg = ' finished fingerprinting, got %d unique hashes'  
  
 values = []  
 for hash, offset in hashes:  
 values.append((song\_id, hash, offset))  
  
 msg = ' storing %d hashes in db' % len(values)  
 print colored(msg, 'green')  
  
 db.store\_fingerprints(values)  
  
 print('end')

## **get-database-stat.py**

#!/usr/bin/python  
from libs.db\_sqlite import SqliteDatabase  
from termcolor import colored  
  
# get summary information  
def printSummary():  
 row = db.executeOne("""  
 SELECT  
 (SELECT COUNT(\*) FROM songs) as songs\_count,  
 (SELECT COUNT(\*) FROM fingerprints) as fingerprints\_count  
 """)  
  
 msg = ' \* %s: %s (%s)' % (  
 colored('total', 'yellow'), # total  
 colored('%d song(s)', 'yellow'), # songs  
 colored('%d fingerprint(s)', 'yellow') # fingerprints  
 )  
 print msg % row  
  
 return row[0] # total  
  
# get songs \w details  
def printSongs():  
 rows = db.executeAll("""  
 SELECT  
 s.id,  
 s.name,  
 (SELECT count(f.id) FROM fingerprints AS f WHERE f.song\_fk = s.id) AS fingerprints\_count  
 FROM songs AS s  
 ORDER BY fingerprints\_count DESC  
 """)  
  
 for row in rows:  
 msg = ' \*\* %s %s: %s' % (  
 colored('id=%s','white',attrs=['dark']), # id  
 colored('%s', 'white', attrs=['bold']), # name  
 colored('%d hashes', 'green') # hashes  
 )  
 print msg % row  
  
# find duplicates  
def printDuplicates():  
 rows = db.executeAll("""  
 SELECT a.song\_fk, s.name, SUM(a.cnt)  
 FROM (  
 SELECT song\_fk, COUNT(\*) cnt  
 FROM fingerprints  
 GROUP BY hash, song\_fk, offset  
 HAVING cnt > 1  
 ORDER BY cnt ASC  
 ) a  
 JOIN songs s ON s.id = a.song\_fk  
 GROUP BY a.song\_fk  
 """)  
  
 msg = ' \* duplications: %s' % colored('%d song(s)', 'yellow')  
 print msg % len(rows)  
  
 for row in rows:  
 msg = ' \*\* %s %s: %s' % (  
 colored('id=%s','white',attrs=['dark']),  
 colored('%s', 'white', attrs=['bold']),  
 colored('%d duplicate(s)', 'red')  
 )  
 print msg % row  
  
# find colissions  
def printColissions():  
 rows = db.executeAll("""  
 SELECT sum(a.n) FROM (  
 SELECT  
 hash,  
 count(distinct song\_fk) AS n  
 FROM fingerprints  
 GROUP BY `hash`  
 ORDER BY n DESC  
 ) a  
 """)  
  
 msg = ' \* colissions: %s' % colored('%d hash(es)', 'red')  
 val = 0  
 if rows[0][0] is not None:  
 val = rows[0]  
  
 print msg % val  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 db = SqliteDatabase()  
 print ''  
  
 x = printSummary()  
 printSongs()  
 if x: print ''  
  
 printDuplicates()  
 if x: print ''  
  
 printColissions()  
  
 print '\ndone'

**sql-execute.py**

#!/usr/bin/python  
import argparse  
import sys  
  
from libs.db\_sqlite import SqliteDatabase  
from termcolor import colored  
from argparse import RawTextHelpFormatter  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 parser = argparse.ArgumentParser(formatter\_class=RawTextHelpFormatter)  
 parser.add\_argument('-q', '--query', nargs='?')  
 args = parser.parse\_args()  
  
 if not args.query:  
 parser.print\_help()  
 sys.exit(0)  
  
 db = SqliteDatabase()  
  
 row = db.executeOne(args.query)  
  
 print

**reset-database.py**

#!/usr/bin/python  
from libs.db\_sqlite import SqliteDatabase  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 db = SqliteDatabase()  
  
 #  
 # songs table  
  
 db.query("DROP TABLE IF EXISTS songs;")  
 print('removed db.songs');  
  
 db.query("""  
 CREATE TABLE songs (  
 id INTEGER PRIMARY KEY AUTOINCREMENT,  
 name TEXT,  
 filehash TEXT  
 );  
 """)  
 print('created db.songs');  
  
 #  
 # fingerprints table  
  
 db.query("DROP TABLE IF EXISTS fingerprints;")  
 print('removed db.fingerprints');  
  
 db.query("""  
 CREATE TABLE `fingerprints` (  
 `id` INTEGER PRIMARY KEY AUTOINCREMENT,  
 `song\_fk` INTEGER,  
 `hash` TEXT,  
 `offset` INTEGER  
 );  
 """)  
 print('created db.fingerprints');  
  
 print('done');