ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΣΥΣΤΗΜΑΤΩΝ ΜΕΤΑΔΟΣΗΣ ΠΛΗΡΟΦΟΡΙΑΣ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ ΥΛΙΚΩΝ

Σύστημα Αναγνώρισης Ήχων Καρδιάς και Πνεύμονα μέσω της Τεχνικής Audio Fingerprinting

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΑΠΟΣΤΟΛΑΚΗ ΜΑΡΙΑ

Επιβλέπων : Δημήτριος - Διονύσιος Κουτσούρης
Καθηγητής Ε.Μ.Π.

Αθήνα, Νοέμβριος 2019

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΣΥΣΤΗΜΑΤΩΝ ΜΕΤΑΔΟΣΗΣ ΠΛΗΡΟΦΟΡΙΑΣ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ ΥΛΙΚΩΝ

Σύστημα Αναγνώρισης Ήχων Καρδιάς και Πνεύμονα μέσω της Τεχνικής Audio Fingerprinting

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΑΠΟΣΤΟΛΑΚΗ ΜΑΡΙΑ

Επιβλέπων : Δημήτριος - Διονύσιος Κουτσούρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 11η Νοεμβρίου 2019.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....
Δ. – Δ. Κουτσούρης Καθηγητής Ε.Μ.Π.	Γ. Ματσόπουλος Καθηγητής Ε.Μ.Π.	Π. Τσανάκας Καθηγητής Ε.Μ.Π.

Αθήνα, Νοέμβριος 2019

(Υπογραφή)

.....

ΑΠΟΣΤΟΛΑΚΗ ΜΑΡΙΑ

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Στην παρούσα διπλωματική εργασία παρουσιάζεται και προσομοιώνεται ένα σύστημα, το οποίο κάνει χρήση των ακουστικών αποτυπωμάτων (audio fingerprints) για την αναγνώριση βιοϊατρικών σημάτων, και συγκεκριμένα ήχων καρδιάς και πνεύμονα.

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 ενός ηχητικού αντικειμένου, προς όφελος της έγκυρης ιατρικής διάγνωσης.

Η υλοποίηση του συστήματος βασίστηκε στο σύστημα ελεύθερου λογισμικού Dejavu. 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. Στο τέλος, παρέχονται προτάσεις για μία καλύτερη προσέγγιση της κεντρικής ιδέας που αντιπροσωπεύεται, με στόχο την αξιοποίηση της συγκεκριμένης τεχνολογίας για την ενίσχυση της διαγνωστικής διαδικασίας σε συνδυασμό με τον επιβλέποντα ιατρό.

Η δομή της διπλωματικής εργασίας συνοψίζεται ως ακολούθως: Σε πρώτο στάδιο πραγματοποιείται μία σύντομη βιβλιογραφική αναφορά στις παθήσεις του καρδιαγγειακού και αναπνευστικού συστήματος και στις βασικές τεχνικές διάγνωσης αυτών. Στη συνέχεια, αφού 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, βιοϊατρικά σήματα, παθήσεις καρδιαγγειακού, παθήσεις αναπνευστικού, καλύτερη διάγνωση.

Abstract

This thesis presents and simulates a system that uses audio fingerprints to identify biomedical signals, in particular heart and lung sounds.

The purpose of this paper is to utilize this technique to uniquely analyze and recognize an audio object for the benefit of valid medical diagnosis.

This project was based on the Dejavu free software system. This system is implemented in Python and is based on Audio Fingerprinting with the use of Hash Functions. This thesis analyzes both the process of the fingerprint extraction and the search method in the database. Finally, suggestions are made for a better approach to the central idea being represented, aiming on the utilization of this technology and the enhancement of the diagnostic process, in conjunction with the supervising doctor.

The structure of this thesis is summarized as follows: First, a brief bibliographical reference is made to the diseases of the cardiovascular and respiratory system and their basic diagnostic techniques. Then, after defining the concept of audio fingerprint and other basic parameters, the process of audio fingerprinting and the restrictions that are set are described in detail. In addition, applications using audio fingerprints and key technologies of this kind are explained. Finally, there is a description of the system being used and the results of the simulations are presented as well as the conclusions.

Key Words: audio fingerprint, audio recognition, database, search algorithms, hash functions, biomedical signals, cardiovascular diseases, respiratory diseases, better diagnosis.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω θερμά τον καθηγητή κ. Δημήτριο Κουτσούρη που μου έδωσε την ευκαιρία να αναπτύξω τη διπλωματική μου εργασία, καθώς και τον υποψήφιο διδάκτορα κ. Παναγιώτη Κατρακάζα και την υποψήφια διδάκτορα και φίλη κα. Ουρανία Μαντά για την υποστήριξή τους και την καθοδήγησή τους κατά τη διάρκεια της εκπόνησης της παρούσας εργασίας. Η συνεργασία μας ήταν άψογη και οι συμβουλές τους πολύτιμες σε όλα τα στάδια και τη διάρκεια της εργασίας μου, υποδεικνύοντάς μου την ορθή και επιστημονική προσέγγιση του θέματος.

Θα ήθελα ακόμα να ευχαριστήσω τους φίλους και συναδέλφους μου, Μαριλένα Καραντινού, Ανδριάνα Δημητρίου, Γιώργο Νικολάου, Βαγγέλη Νταβέλη και Γρηγόρη Καραγεώργο για την συμπαράσταση και τις συμβουλές που μου προσέφεραν σε τεχνικά κυρίως θέματα το τελευταίο έτος των σπουδών μου.

Τέλος, οφείλω να ευχαριστήσω την οικογένεια μου, και ιδιαίτερα τα αδέλφια και συναδέλφους μου, Βαγγέλη και Δημήτρη Αποστολάκη, για την στήριξη και τη βοήθεια που μου παρείχαν καθόλη τη διάρκεια των σπουδών μου.

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΕΥΡΕΤΗΡΙΟ ΣΧΗΜΑΤΩΝ ΚΑΙ ΠΙΝΑΚΩΝ

1 ΕΙΣΑΓΩΓΗ

1.1 ΤΟ ΠΡΟΒΛΗΜΑ

Η συχνότητα εμφάνισης και η εξάπλωση καρδιαγγειακών ασθενειών αυξάνεται με σταθερό ρυθμό ανά τον κόσμο τα τελευταία χρόνια. Στατιστικά δεδομένα που παρέχονται από τον ιστότοπο World Health Organization (WHO)[1] επιβεβαιώνουν τη θέση αυτή : Οι καρδιοαγγειακές ασθένειες (CVDs) είναι ο νούμερο 1 λόγος θανάτου παγκοσμίως. Κάθε χρόνο περισσότεροι άνθρωποι πεθαίνουν από τις ασθένειες αυτές, παρά από οποιονδήποτε άλλο λόγο. Η εκτίμηση για το 2013 ήταν πως 17.9 εκατομμύρια θάνατοι μπορούν να αποδοθούν σε Cardiovascular Diseases (CVDs), αποτελώντας το 31% του συνόλου των θανάτων παγκοσμίως. Από αυτό τον αριθμό 7.4 εκατομμύρια θάνατοι οφειλόταν σε στεφανιαία νόσο, ενώ 6.7 εκατομμύρια θάνατοι σε εμφράγματα.

Εξίσου συχνή είναι και η εμφάνιση αναπνευστικών παθήσεων, με ολοένα και αυξανόμενο ρυθμό εξάπλωσης σε παγκόσμια κλίμακα, με το φαινόμενο να έχει σημαντικότερες διαστάσεις στα μεγάλα αστικά κέντρα. Πρόσφατες στατιστικές του World Health Organization[1] δίνουν απτά στοιχεία : Οι αναπνευστικές ασθένειες προξενούν στην υγεία τεράστια επιβάρυνση σε παγκόσμιο επίπεδο και είναι η δεύτερη, μετά τις καρδιαγγειακές παθήσεις (συμπεριλαμβανομένου του εγκεφαλικού επεισοδίου) συχνότερη αιτία θανάτων . Οι αναπνευστικές νόσοι αποτελούν πέντε από τις 30 πιο συχνές αιτίες θανάτου: Υπολογίζεται ότι 65 εκατομμύρια άνθρωποι έχουν μέτρια έως σοβαρή χρόνια αποφρακτική πνευμονική νόσο (COPD), από την οποία περίπου 3 εκατομμύρια πεθαίνουν κάθε χρόνο, κάνοντας την, την τρίτη κύρια αιτία θανάτου παγκοσμίως. Η λοίμωξη του κατώτερου αναπνευστικού είναι τέταρτη, ο βρογχικός και ο πνευμονικός καρκίνος είναι έκτοι. Η φυματίωση είναι δωδέκατη και το άσθμα είναι εικοστό όγδοο. Συνολικά, πάνω από 1 δισεκατομμύριο άνθρωποι υποφέρουν από οξεία ή χρόνια αναπνευστική ανεπάρκεια. Η σκληρή πραγματικότητα είναι ότι, κάθε χρόνο, 4 εκατομμύρια άνθρωποι πεθαίνουν πρόωρα από χρόνια αναπνευστική νόσο.

Ως επακόλουθο, όλων των παραπάνω, η επινόηση τεχνικών έγκυρης διάγνωσης είναι σημαντική για την άμεση ανίχνευση πιθανών κινδύνων και την αντιμετώπισή τους.

1.2 ΚΑΡΔΙΑΚΕΣ ΠΑΘΗΣΕΙΣ ΚΑΙ ΔΙΑΓΝΩΣΗ

Η καρδιακή νόσος περιγράφει μια σειρά από καταστάσεις που επηρεάζουν την καρδιά. Ασθένειες κάτω από την ομπρέλα των καρδιακών παθήσεων περιλαμβάνουν, μεταξύ άλλων, ασθένειες αιμοφόρων αγγείων, όπως η νόσος στεφανιαίας αρτηρίας, προβλήματα καρδιακού ρυθμού (αρρυθμίες) και τα εκ γενετής καρδιακά ελαττώματα (συγγενή καρδιακά ελαττώματα). Ο όρος "καρδιακή νόσος" χρησιμοποιείται συχνά εναλλακτικά με τον όρο "καρδιαγγειακή νόσος". Οι καρδιαγγειακές παθήσεις (CVDs) είναι διαταραχές της καρδιάς και των αιμοφόρων αγγείων και γενικά αναφέρονται σε καταστάσεις που περιλαμβάνουν στενωμένα ή αποκλεισμένα αιμοφόρα αγγεία που μπορεί να οδηγήσουν σε καρδιακή προσβολή, πόνο στο στήθος (στηθάγχη) ή εγκεφαλικό επεισόδιο. Άτομα που κινδυνεύουν από καρδιαγγειακά νοσήματα μπορεί να επιδείξουν αυξημένη αρτηριακή πίεση, γλυκόζη και λιπίδια ενώ, ευπαθείς κατηγορίες είναι και άτομα υπέρβαρα και παχύσαρκα. Άλλες καρδιακές νόσοι, όπως αυτές που επηρεάζουν τους μυς, τις βαλβίδες ή το ρυθμό της καρδιάς, θεωρούνται επίσης μορφές καρδιακών παθήσεων. Η αιτία μεγάλου αριθμού των καρδιακών νοσημάτων είναι συνήθως η παρουσία ενός συνδυασμού βλαβερών για την υγεία συνηθειών, όπως η χρήση καπνού, η ανθυγιεινή διατροφή και η παχυσαρκία, η σωματική αδράνεια και η χρήση αλκοόλ, η υπέρταση, ο διαβήτης και η υπερλιπιδαιμία[2].

1.2.1 Καρδιαγγειακές Παθήσεις

- Στεφανιαία νόσος - ασθένεια των αιμοφόρων αγγείων που προμηθεύει τον καρδιακό μυ.
- Εγκεφαλοαγγειακή νόσος - ασθένεια των αιμοφόρων αγγείων που προμηθεύουν τον εγκέφαλο.
- Περιφερική αρτηριακή νόσος - ασθένεια των αιμοφόρων αγγείων που τροφοδοτούν τα χέρια και τα πόδια.
- Ρευματική καρδιακή νόσος - βλάβη του καρδιακού μυός και των καρδιακών βαλβίδων από ρευματικό πυρετό, που προκαλείται από στρεπτοκοκκικά βακτήρια.
- Συγγενής καρδιακή νόσο - δυσπλασίες της δομής της καρδιάς που υπάρχουν κατά τη γέννηση.
- Φλεβική θρόμβωση και πνευμονική εμβολή - θρόμβοι αίματος στις φλέβες των ποδιών, οι οποίοι μπορούν να αποκολληθούν και να μετακινηθούν στην καρδιά και τους πνεύμονες.

- Κοιλιακή μαρμαρυγή - κατάσταση στην οποία υπάρχει ασυντόνιστη συστολή των κοιλιών του καρδιακού μυός.
- Περικαρδίτιδα - φλεγμονή των δύο πετάλων του περικαρδίου (σπλαγχνικό και τοιχωματικό).
- Ενδοκαρδίτιδα - νόσος που χαρακτηρίζεται από λοίμωξη και φλεγμονή των καρδιακών βαλβίδων και του ενδοκαρδίου.
- Στηθάγχη - παροξυσμός θωρακικού πόνου που εντοπίζεται στην προκάρδια περιοχή, και άλλες[2].

1.2.2 Συμπτώματα Καρδιαγγειακής Νόσου

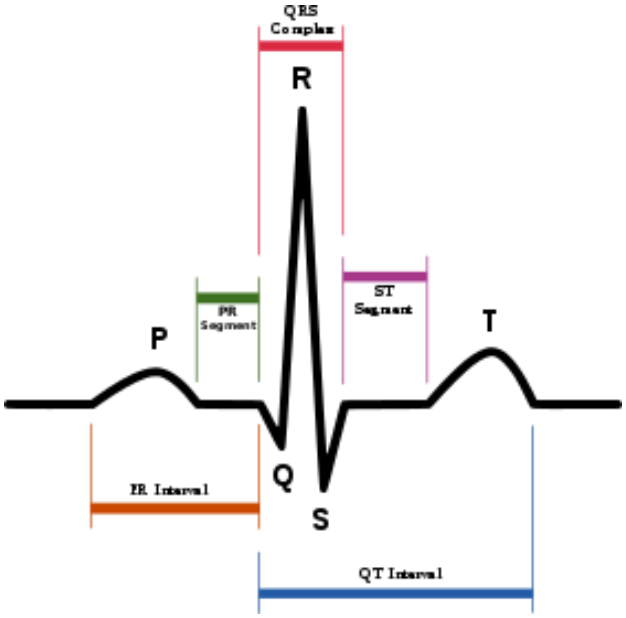
- Πόνος, σφίξιμο, πίεση ή δυσφορία στο στήθος (στηθάγχη).
- Δυσκολία στην αναπνοή.
- Πόνος, μούδιασμα, αδυναμία ή κρύο στα πόδια ή στα χέρια.
- Πόνος στο λαιμό, στο σαγόνι, στην άνω κοιλιά ή στην πλάτη.
- Γρήγορος καρδιακός παλμός (ταχυκαρδία) ή αργός καρδιακός παλμός (βραδυκαρδία).
- Ζάλη.
- Λιποθυμικές τάσεις, και άλλα[2].

Οι άνθρωποι που βιώνουν αυτά τα συμπτώματα πρέπει να αναζητήσουν αμέσως ιατρική περίθαλψη στις εγκαταστάσεις πρωτοβάθμιας περίθαλψης. Ο εντοπισμός των ατόμων που κινδυνεύουν περισσότερο από καρδιαγγειακά νοσήματα και η εξασφάλιση κατάλληλης θεραπείας μπορεί να αποτρέψει τους πρόωρους θανάτους.

1.2.3 Τεχνικές Διάγνωσης των Καρδιαγγειακών Παθήσεων

Ηλεκτροκαρδιογράφημα (ΗΚΓ)

Καταγράφει την ηλεκτρική δραστηριότητα των μυών της παλλόμενης καρδιάς αποδίδοντας μέσω του ηλεκτροκαρδιογράφου σε ειδικό χαρτί ή οθόνη χαρακτηριστικά γραφήματα, τα επονομαζόμενα επάρματα. Η αυτοματοποιημένη ερμηνεία ΗΚΓ είναι η χρήση της τεχνητής νοημοσύνης και λογισμικού αναγνώρισης προτύπων και βάσεις γνώσεων(knowledge base) για να πραγματοποιήσει αυτόματα την ερμηνεία, την υποβολή εκθέσεων δοκιμών, και με τη βοήθεια υπολογιστή την διάγνωση του ηλεκτροκαρδιογραφήματος που λαμβάνεται συνήθως από έναν ασθενή. Με το ηλεκτροκαρδιογράφημα(ΗΚΓ) μπορούμε να διαπιστώσουμε οξείες (πχ. οξύ έμφραγμα του μυοκαρδίου) αλλά και χρόνιες (κολπική μαρμαρυγή, καρδιακές αρρυθμίες) διαταραχές που μπορούν να αφορούν τον καρδιακό ρυθμό αλλά και την αρχιτεκτονική της καρδιάς. Είναι μια απλή, γρήγορη, ανώδυνη και φθηνή εξέταση της δραστηριότητας της καρδιάς, με αρκετά καλή ειδικότητα και ευαισθησία[3].



Εικόνα 1:*Τυπική μορφή του ΗΚΓ για ένα κύκλο της ηλεκτρικής δραστηριότητας της καρδιάς[3].*

Παλμική οξυμετρία

Η παλμική οξυμετρία είναι απλή εξέταση για την εκτίμηση της οξυγόνωσης ή καλύτερα τον κορεσμό σε οξυγόνο στο αίμα (το ποσοστό της αιμοσφαιρίνης που μεταφέρει οξυγόνο). Πρόκειται για μια σύντομη διαδικασία, μη επεμβατική και ανώδυνη. Το αποτέλεσμά της χρησιμοποιείται σαν γενικός δείκτης της μεταφοράς του οξυγόνου στους περιφερικούς ιστούς[4].

Holter Ρυθμού

Το Holter ρυθμού (ή Holter καρδιάς), αναφέρεται στην 24ωρη, 48ωρη ή μέχρι και 7 ημερών καταγραφή του καρδιακού ρυθμού, σε συνθήκες αντιπροσωπευτικές της τυπικής καθημερινότητας του εξεταζόμενου, με χρήση φορητού εξοπλισμού. Συνήθως, χρησιμοποιείται όταν υπάρχουν συμπτώματα διαλείπουσας αρρυθμίας (ή ασυμπτωματικής ισχαιμίας), για παρακολούθηση της ανταπόκρισης στην αγωγή και τροποποίηση της αγωγής σε παθήσεις όπως η κολπική μαρμαρυγή, για διαστρωμάτωση κινδύνου σε παθήσεις που υπάρχει κίνδυνος αιφνίδιου καρδιακού θανάτου, και άλλα[5].

Τεστ Κοπώσεως

Η δοκιμασία κοπώσεως ή τεστ κοπώσεως ή stress test είναι μια απλή, αναίμακτη εξέταση, που συμβάλλει στη διάγνωση και παρακολούθηση της στεφανιαίας νόσου. Τα δύο κύρια χαρακτηριστικά της δοκιμασίας κοπώσεως είναι η εκτέλεση ελεγχόμενης σωματικής άσκησης από τον εξεταζόμενο και η συνεχής ηλεκτροκαρδιογραφική παρακολούθηση αυτού κατά τη διάρκεια της εκτέλεσης της άσκησης καθώς και μετά από αυτήν[6].

Τεχνολογίες Απεικόνισης (Imaging Technologies)

Χρησιμοποιούνται ευρύτατα στην διάγνωση της αθηροσκλήρωσης η οποία χαρακτηρίζεται από αποσπασματικές πλάκες εσωτερικής επιφάνειας (αθηρώματα) που προσβάλλουν τον αυλό μεσαίων και μεγάλων αρτηριών. Επειδή δεν έχουν όλες οι αθηροσκληρωτικές πλάκες τον ίδιο κίνδυνο, μελετώνται διάφορες τεχνολογίες απεικόνισης ως ένας τρόπος για να εντοπιστούν πλάκες ιδιαίτερα ευάλωτες στη ρήξη. Ωστόσο, αυτές οι τεχνικές δεν χρησιμοποιούνται ακόμη κλινικά.

Οι μη επεμβατικές τεχνικές απεικόνισης που μπορούν να αξιολογήσουν τη μορφολογία και τα χαρακτηριστικά της πλάκας περιλαμβάνουν :

- Τρισδιάστατη Αγγειακή Υπερηχογραφία.
- Υπολογιστική Τομογραφία (CT) Αγγειογραφία.

- Μαγνητική Τομογραφία (MR) Αγγειογραφία.

Χρησιμοποιούνται επίσης επεμβατικές δοκιμές με καθετήρα. Αυτές περιλαμβάνουν :

- Ενδοαγγειακή υπερηχογραφία, η οποία χρησιμοποιεί έναν μετατροπέα υπερήχων στην άκρη ενός καθετήρα για την παραγωγή εικόνων του αρτηριακού αυλού και του τοιχώματος.
- Αγγειοσκόπηση, η οποία χρησιμοποιεί ειδικούς οπτικούς καθετήρες που μπορούν να απεικονίσουν άμεσα την αρτηριακή επιφάνεια.
- Θερμογραφία πλάκας, η οποία χρησιμοποιείται για την ανίχνευση της αυξημένης θερμοκρασίας σε πλάκες με ενεργό φλεγμονή.
- Οπτική ομόκεντρη συνοχή, η οποία χρησιμοποιεί υπέρυθρο φως λέιζερ για απεικόνιση
- Η ελαστογραφία, η οποία χρησιμοποιείται για την αναγνώριση μαλακών πλούσιων σε λιπίδια πλακών.
- Το ανοσοσπινθηρογράφημα είναι μία μη επεμβατική εναλλακτική λύση που χρησιμοποιεί ραδιενεργούς ιχνηθέτες που εντοπίζονται σε ευαίσθητες πλάκες.
- Η τομογραφία εκπομπής ποζιτρονίων (PET) απεικόνισης του αγγειακού συστήματος είναι μια άλλη αναδυόμενη προσέγγιση για την αξιολόγηση της ευάλωτης πλάκας[7].

Εικόνα 2: *Ενδοαγγειακό υπερηχογράφημα[7].*

Ακτινογραφία Θώρακος

Τα ευρήματα της ακτινογραφίας θώρακος που υποδηλώνουν καρδιακή ανεπάρκεια περιλαμβάνουν μια διευρυμένη καρδιακή σιλουέτα, υγρό στο κύριο ρήγμα και οριζόντιες γραμμές στην περιφέρεια των κατώτερων οπίσθιων πνευμονικών πεδίων (σειρές Kerley B). Μπορεί επίσης να υπάρχει παλμική φλεβική συμφόρηση του ανώτερου λοβού και ενδιάμεσο ή κυψελιδικό οίδημα. Η προσεκτική εξέταση της καρδιακής σιλουέτας σε μια πλευρική προβολή μπορεί να εντοπίσει συγκεκριμένη μεγέθυνση κοιλιακής και κολπικής κοιλότητας. Η ακτινογραφία μπορεί επίσης να προτείνει εναλλακτικές διαγνώσεις (π.χ. ΧΑΠ, πνευμονία, ιδιοπαθή πνευμονική ίνωση, καρκίνο του πνεύμονα)[7].

Ηχοκαρδιογραφία

Συνεισφέρει στη διάγνωση καρδιακών όγκων, στην περίπτωση των οποίων συχνά η διάγνωση καθυστερεί, επειδή τα συμπτώματα μιμούνται εκείνα των πολύ πιο κοινών διαταραχών. Το Διοισοφάγειο ηχοκαρδιογράφημα είναι καλύτερο για την οπτικοποίηση κολπικού όγκου, ενώ η Διαθωρακική ηχοκαρδιογραφία είναι καλύτερη για κοιλιακούς όγκους[7].

Τεχνικές Τεχνητής Νοημοσύνης

Τεχνικές τεχνητής νοημοσύνης[8] (AI) χρησιμοποιούνται κατά κόρον στην ιατρική διάγνωση. Με την πρόοδο της επιστήμης, ο όγκος των συσσωρευμένων δεδομένων σε διάφορους τομείς έχει αυξηθεί τρομακτικά. Με την ανάλυση των συσσωρευμένων δεδομένων θα μπορούσαν να αποκαλυφθούν οι κρυφές χρήσιμες πληροφορίες τους. Με το συνδυασμό νέων τεχνολογιών όπως Data Mining, Classification, Neural Networks, Machine Learning Algorithms,Databases, μπορούμε και πετυχαίνουμε πολύ πιο ακριβή και γρήγορα αποτελέσματα για την πρόβλεψη καρδιακών παθήσεων. Για παράδειγμα : Ο Ram Bilas Pachori[9] και οι συνεργάτες του έχουν μελετήσει τη διάγνωση της καρδιακής νόσου με τη χρήση Q κυματιδίων που λαμβάνονται από τα σήματα του καρδιακού ρυθμού. Δεδομένου ότι η χειροκίνητη εισαγωγή δεδομένων εμφανίζει σφάλματα και είναι χρονοβόρα συνίσταται η χρήση της μεθόδου Wavelet Tunable-Q (TQWT) στην παρούσα μελέτη. Χρησιμοποιώντας τη μηχανή υποστήριξης ελαχίστων τετραγώνων (LS-SVM), αναφέρουν ακρίβεια 96,8%, ευαισθησία ίση με 100% και εξειδίκευση 93,7%. Υπάρχει πληθώρα τέτοιων τεχνικών και διαρκώς εμφανίζονται καινούριες πιο αποδοτικές και ενημερωμένες.

1.3 ΠΑΘΗΣΕΙΣ ΤΟΥ ΑΝΑΠΝΕΥΣΤΙΚΟΥ ΚΑΙ ΔΙΑΓΝΩΣΗ

Ο πνεύμονας είναι το πιο ευάλωτο εσωτερικό όργανο σε λοιμώξεις και τραυματισμούς από το εξωτερικό περιβάλλον λόγω της συνεχούς έκθεσής του σε σωματίδια, χημικά και μολυσμένους μικροοργανισμούς στον ατμοσφαιρικό αέρα. Σε παγκόσμιο επίπεδο, τουλάχιστον 2 εκατομμύρια άνθρωποι εκτίθενται στον τοξικό καπνό του καυσίμου βιομάζας, που συνήθως καίγεται αναποτελεσματικά σε ανεπαρκώς αεριζόμενες εσωτερικές σόμπες ή τζάκια. Ένα δισεκατομμύριο άνθρωποι εισπνέουν μολυσμένο υπαίθριο αέρα και ένα δισεκατομμύριο εκτίθενται στον καπνό του τσιγάρου. Αν και η αναπνευστική δυσλειτουργία προκαλεί αναπηρίες και θάνατο σε όλες τις περιοχές του κόσμου και σε όλες τις κοινωνικές τάξεις, η φτώχεια, ο συνωστισμός, η περιβαλλοντική έκθεση και γενικά οι κακές συνθήκες διαβίωσης αυξάνουν την ευαισθησία σε αυτή τη μεγάλη ομάδα ασθενειών. Οι ασθένειες που παρουσιάζονται στο αναπνευστικό σύστημα ανήκουν σε διάφορες κατηγορίες. Παράγοντες περιβαλλοντικοί, γενετικοί, κληρονομικοί ή συνδυασμός τους, είναι σε θέση να επηρεάζουν τους πνεύμονες και να προάγουν την προσβολή τους από διάφορες ασθένειες, όπως χρόνιες αποφρακτικές παθήσεις των πνευμόνων, το εμφύσημα, η χρόνια βρογχίτιδα και άλλες[10].

1.3.1 Παθήσεις του Πνεύμονα και του ευρύτερου Αναπνευστικού

- Άσθμα - χρόνια πάθηση στην οποία επηρεάζονται οι βρόγχοι και τα βρογχιόλια.
- Κυστική ίνωση - κληρονομική πάθηση που επηρεάζει τις βλέννες και τον ιδρώτα των ασθενών. Λόγω των προβλημάτων που δημιουργούνται, συσσωρεύεται βλέννα στους πνεύμονες και είναι αιτία συχνών μολύνσεων.
- Χρόνιες αποφρακτικές παθήσεις των πνευμόνων (ΧΑΠ) - περιλαμβάνουν τη χρόνια βρογχίτιδα και το εμφύσημα. Συχνά οι δύο παθήσεις συνυπάρχουν δημιουργώντας μια σύνθετη πάθηση που αποκαλείται χρόνια αποφρακτική νόσος των πνευμόνων.
- Χρόνια βρογχίτιδα - χαρακτηρίζεται από φλεγμονή και βλάβες της βλεννογόνου των βρόγχων.
- Εμφύσημα - προκαλεί προοδευτικές βλάβες στις κυψελίδες των πνευμόνων.
- Καρκίνος του πνεύμονα - χαρακτηρίζεται από τον ανεξέλεγκτο και αναρχικό πολλαπλασιασμό ανώμαλων κυττάρων.
- Πνευμονία - περιλαμβάνει ένα μεγάλο φάσμα μολυσματικών ασθενειών που προκαλούνται λόγω προσβολής των πνευμόνων από διάφορα μικρόβια, βακτήρια, ιούς, παράσιτα και μύκητες.

- Πνευμονική υπέρταση - σχετίζεται με την αποτυχία της αριστερής κοιλίας.
- φυματίωση - μόλυνση που προκαλείται από ένα είδος βακτηριδίου που προσβάλλει κυρίως και πρωταρχικά τους πνεύμονες. Το βακτηρίδιο αυτό προκαλεί φλεγμονή στον πνευμονικό ιστό και στη συνέχεια τον καταστρέφει.
- Ψευδομεμβρανώδης λαρυγγίτιδα (ή λαρυγγοτραχειοβρογχίτιδα) αποτελεί λοίμωξη του αναπνευστικού συστήματος που οφείλεται σε ιογενή λοίμωξη της ανώτερης αναπνευστικής οδού[10].

1.3.2 Συμπτώματα Παθήσεων του Αναπνευστικού και του Πνεύμονα

- Δύσπνοια, βήχας, συριγμός, ρόγχος.
- Επιπόλαιες αναπνοές, ταχύπνοια.
- Εισολκή.
- Αιμόπτυση.
- Δυσκολία στον ύπνο.
- Δυσανεξία στην άσκηση.
- Κυάνωση.
- Βλέννα.
- Πόνος στο στήθος.
- Πυρετός.
- Εφίδρωση, κ.α[10].

1.3.3 Τεχνικές Διάγνωσης των Παθήσεων του Πνεύμονα και του Αναπνευστικού

Σπιρομέτρηση

Η σπιρομέτρηση μετράει πόσο υγιείς είναι οι πνεύμονές και μπορεί να χρησιμοποιηθεί στη διάγνωση και την παρακολούθηση πνευμονικών παθήσεων. Στη διάρκεια της εξέτασης, ο ασθενής εκπνέει όσο το δυνατό περισσότερο αέρα μπορεί, όσο πιο δυνατά μπορεί, μέσα σε μια συσκευή που ονομάζεται σπιρόμετρο. Η εξέταση μετράει πόσο πολύ αέρα μπορεί να

φυσήξει συνολικά ο εξεταζόμενος και πόσο πολύ αέρα μπορεί να εκπνεύσει στο πρώτο δευτερόλεπτο της εξέτασης[11].

Βρογχοσκόπηση

Βρογχοσκόπηση ονομάζεται η διαδικασία κατά την οποία ο πνευμονολόγος εξετάζει τους μεγάλους αεραγωγούς την τραχεία και τους βρόγχους που αποτελούν πολύ σημαντικές οδούς για τη μεταφορά του αέρα στους πνεύμονες. Η βρογχοσκόπηση μπορεί να πραγματοποιηθεί με την χρήση είτε εύκαμπτου, είτε άκαμπτου βρογχοσκοπίου με την κάθε μέθοδο να έχει τα δικά της πλεονεκτήματα και δυσκολίες. Αμφότεροι οι τύποι βρογχοσκοπίου έχουν ένα πλευρικό κανάλι, μέσω του οποίου μπορεί να περάσει και να οδηγηθεί προς το κάτω άκρο του ένας λεπτός καθετήρας, συνοδευόμενος από μία λαβίδα βιοψίας. Ο καθετήρας χρησιμοποιείται για την λήψη μικρού δείγματος για βιοψία από την εσωτερική επένδυση των βρόγχων, ή να αφαιρέσει μικρά αντικείμενα από τους αεραγωγούς. Οι λόγοι για τους οποίους απαιτείται η βρογχοσκόπηση είναι πολλοί, αλλά συνοπτικά, οι πιο κοινοί είναι οι ενδείξεις βήχα, βήχα που συνοδεύεται από αιμόπτυση ή και ευρήματα σκιών σε ακτινογραφίες θώρακος όπου μπορούν να εντοπιστούν όγκοι σε κάποιον βρόγχο, αλλά και υπόνοιες καρκίνου του πνεύμονα[12].

Τεχνολογίες Απεικόνισης (Imaging Technologies)

Χρησιμοποιούνται στη διάγνωση μεγάλου αριθμού παθήσεων του αναπνευστικού και του πνεύμονα, όπως είναι : η βρογχιεκτασία, η πνευμονία, η οξεία βρογχίτιδα, ο καρκίνος του πνεύμονα, το πνευμονικό απόστημα και άλλες. Η διάγνωση βασίζεται στο ιστορικό και τις τεχνολογίες απεικόνισης, που συνήθως περιλαμβάνουν :

- Υπολογιστική τομογραφία υψηλής ανάλυσης (CT).
- Συνήθεις ακτινογραφίες θώρακος (Chest X-rays)[13].

Τεστ Πνευμονικής Λειτουργίας

Τα τεστ πνευμονικής λειτουργίας παρέχουν μετρήσεις : ροής αέρα, όγκους πνεύμονα, ανταλλαγής αερίων, απόκρισης σε βρογχοδιασταλτικά και λειτουργίας των αναπνευστικών μυών.

Οι βασικές δοκιμές πνευμονικής λειτουργίας που είναι διαθέσιμες στην περιπατητική ρύθμιση περιλαμβάνουν: Σπιρομέτρηση και Παλμική οξυμετρία.

Η σπιρομετρία και η παλμική οξυμετρία παρέχουν φυσιολογικά μέτρα πνευμονικής λειτουργίας και μπορούν να χρησιμοποιηθούν για να περιορίσουν γρήγορα μια διαφορική

διάγνωση και να προτείνουν μια μετέπειτα στρατηγική συμπληρωματικών εξετάσεων ή θεραπείας.

Πιο περίπλοκες δοκιμές περιλαμβάνουν :

Μέτρηση των όγκων των πνευμόνων, του θωρακικού τοιχώματος, και του αναπνευστικού συστήματος συμμόρφωσης (η οποία απαιτεί μέτρηση της οισοφαγικής πίεσης).

Πλήρης δοκιμή καρδιοπνευμονικής άσκησης :

Αυτές οι δοκιμές παρέχουν μια πιο λεπτομερή περιγραφή των φυσιολογικών ανωμαλιών και την πιθανή υποκείμενη παθολογία. Η επιλογή και η ακολουθία των δοκιμών καθοδηγούνται από πληροφορίες που λαμβάνονται από την ιστορία και τη φυσική εξέταση[14].

Άλλες τεχνικές διάγνωσης αναπνευστικών και πνευμονικών παθήσεων είναι : το ιστορικό και η φυσική εξέταση του ασθενούς, καλλιέργεια πτυέλων για βακτήρια και μυκοβακτηρίδια για τον προσδιορισμό οργανισμών αποικιοποίησης, ειδικές δοκιμές για ύποπτες αιτίες. Επιπλέον, τεχνικές μηχανικής μάθησης[15], πολύπλοκοι αλγόριθμοι, χρήση νευρωνικών δικτύων και άλλες καινοτόμες τεχνολογίες υπεισέρχονται διαρκώς και βελτιώνουν τη διάγνωση.

1.4 Ο Σκοπος και η Δομη της Εργασιας

1.4.1 Σκοπός

Με βάση όλα τα προηγούμενα γίνεται σαφές πως την ευθύνη της έγκυρης διάγνωσης οποιασδήποτε ασθένειας αναλαμβάνει ο τομέας της βιοϊατρικής τεχνολογίας, όπου με την εφαρμογή εξειδικευμένου εξοπλισμού παρέχει την δυνατότητα της συνεχούς παρακολούθησης, ενώ με έξυπνους αλγόριθμους μπορεί να βοηθήσει τους γιατρούς να ανιχνεύσουν έγκυρα συμπτώματα ασθενειών ή ακόμα να γίνει αυτόματη διάγνωση χωρίς την παρουσία κάποιου ειδικού.

Στην παρούσα διπλωματική εργασία παρουσιάζεται και περιγράφεται ένα σύστημα αναγνώρισης ήχων καρδιοπαθειών[16] και πνευμονοπαθειών[17] με βάση το περιεχόμενό τους. Το σύστημα αυτό στηρίζεται στην έννοια του ακουστικού αποτυπώματος (Audio Fingerprinting) και χρησιμοποιείται ήδη σε μία πληθώρα εφαρμογών, κυρίως σχετιζόμενων με τη μουσική βιομηχανία (αναγνώριση μουσικών κομματιών). 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 ενός ηχητικού αντικειμένου, προς όφελος της έγκυρης ιατρικής διάγνωσης. Θα μπορούσε να χρησιμοποιηθεί για να βελτιώσει την ποιότητα και την ταχύτητα της διάγνωσης, συνδυαστικά με τον επιβλέποντα ιατρό ; Μπορεί να συνεισφέρει στις ήδη προσφερόμενες υπηρεσίες υγείας μειώνοντας ταυτόχρονα το κόστος που σχετίζεται με αυτές ;

1.4.2 Δομή

Στο Κεφάλαιο 2 παρουσιάζονται οι γενικές αρχές των μεθοδολογιών για Audio Fingerprinting και γίνεται ανάλυση των βασικών προσεγγίσεων που διατίθενται τόσο στη διεθνή βιβλιογραφία όσο και σε εμπορικό επίπεδο. Πιο αναλυτικά εξηγούνται ζητήματα και έννοιες σχετικά με τα audio fingerprints τη λειτουργία τους και τα χαρακτηριστικά τους, γίνεται εκτενής ανάλυση της γενικής λειτουργίας και δομής του audio fingerprinting ως τεχνικής, διευκρινίζονται οι περιορισμοί που διέπουν την τεχνική αυτή και παρουσιάζονται διάφορες εφαρμογές της. Τέλος αναλύονται σύντομα, εστιάζοντας στα βασικά τεχνικά χαρακτηριστικά τους, νέες και παλαιότερες τεχνολογίες (εμπορικές και μη) που κάνουν χρήση των ακουστικών αποτυπωμάτων.

Στα κεφάλαια 3 έως 5, αναλύεται η μέθοδος που επιλέχθηκε και χρησιμοποιήθηκε, ο τρόπος υλοποίησης της, τα αποτελέσματα χρήσης της και τέλος γίνεται μία αξιολόγηση των αποτελεσμάτων αυτών, αναφέρονται περιορισμοί που προέκυψαν και παρατίθενται προτάσεις για μελλοντική έρευνα. Πιο αναλυτικά, στο Κεφάλαιο 3 περιγράφεται το σύστημα που χρησιμοποιήθηκε, υλοποιημένο σε Python, όπως επίσης και η δομή και η λειτουργία του. Αμέσως μετά, στο Κεφάλαιο 4 παρουσιάζονται τα αποτελέσματα πειραμάτων που εκτελέστηκαν με βάση την προηγούμενη υλοποίηση, σε σχέση με την ευρωστία, την ταχύτητα και την αποδοτικότητα του συστήματος. Εν κατακλείδι, στο Κεφάλαιο 5 συνοψίζονται τα συμπεράσματα και δίνονται προτάσεις για περαιτέρω έρευνα και ενασχόληση.

2 ΘΕΩΡΗΤΙΚΟ ΜΕΡΟΣ – STATE OF THE ART

Σκοπός του κεφαλαίου αυτού είναι να οριστεί με όσο το δυνατόν μεγαλύτερη ακρίβεια το audio fingerprint καθώς και βασικές έννοιες που σχετίζονται με αυτό. Οι γενικές αρχές λειτουργίας ενός συστήματος αναγνώρισης ήχου με βάση το περιεχόμενο του, καθώς και η δομή ενός τέτοιου συστήματος και να αναλυθούν τα επιμέρους μέρη και τα χαρακτηριστικά τους.

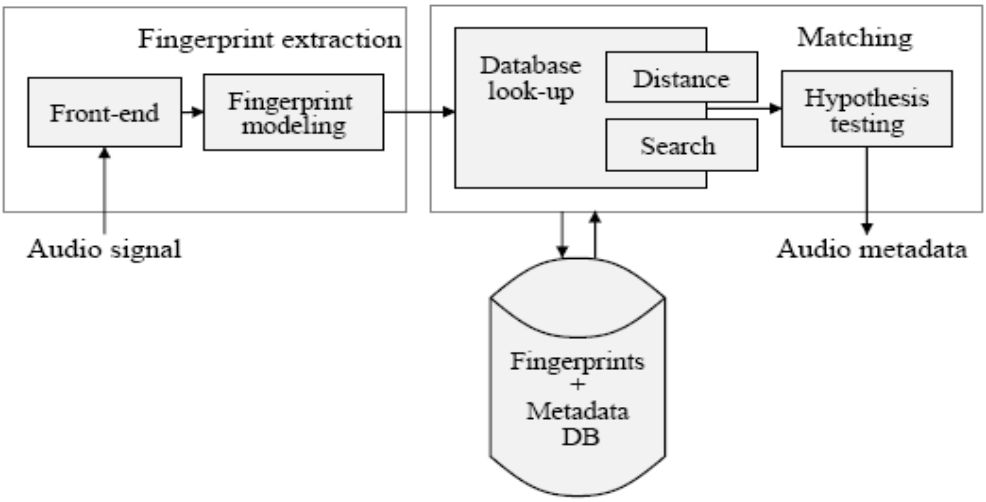
2.1 ΟΡΙΣΜΟΣ ΤΟΥ AUDIO FINGERPRINT

Audio Fingerprint[18] καλείται μία σύντομη αναπαράσταση ενός ακουστικού αντικειμένου. Η αναπαράσταση αυτή περιγράφει μοναδικά το ακουστικό αντικείμενο, βασισμένη στο περιεχόμενό του. Απώτερος στόχος, βέβαια, είναι να ελεγχθεί αν δύο αντικείμενα είναι “ίσα” ακουστικά, συγκρίνοντας όχι τα ίδια άμεσα, αλλά τα ακουστικά αποτυπώματά τους, όπως θα μπορούσαν να αποκαλεστούν τα audio fingerprints.

2.2 ΠΕΡΙΓΡΑΦΗ ΤΟΥ AUDIO FINGERPRINTING

2.2.1 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 :



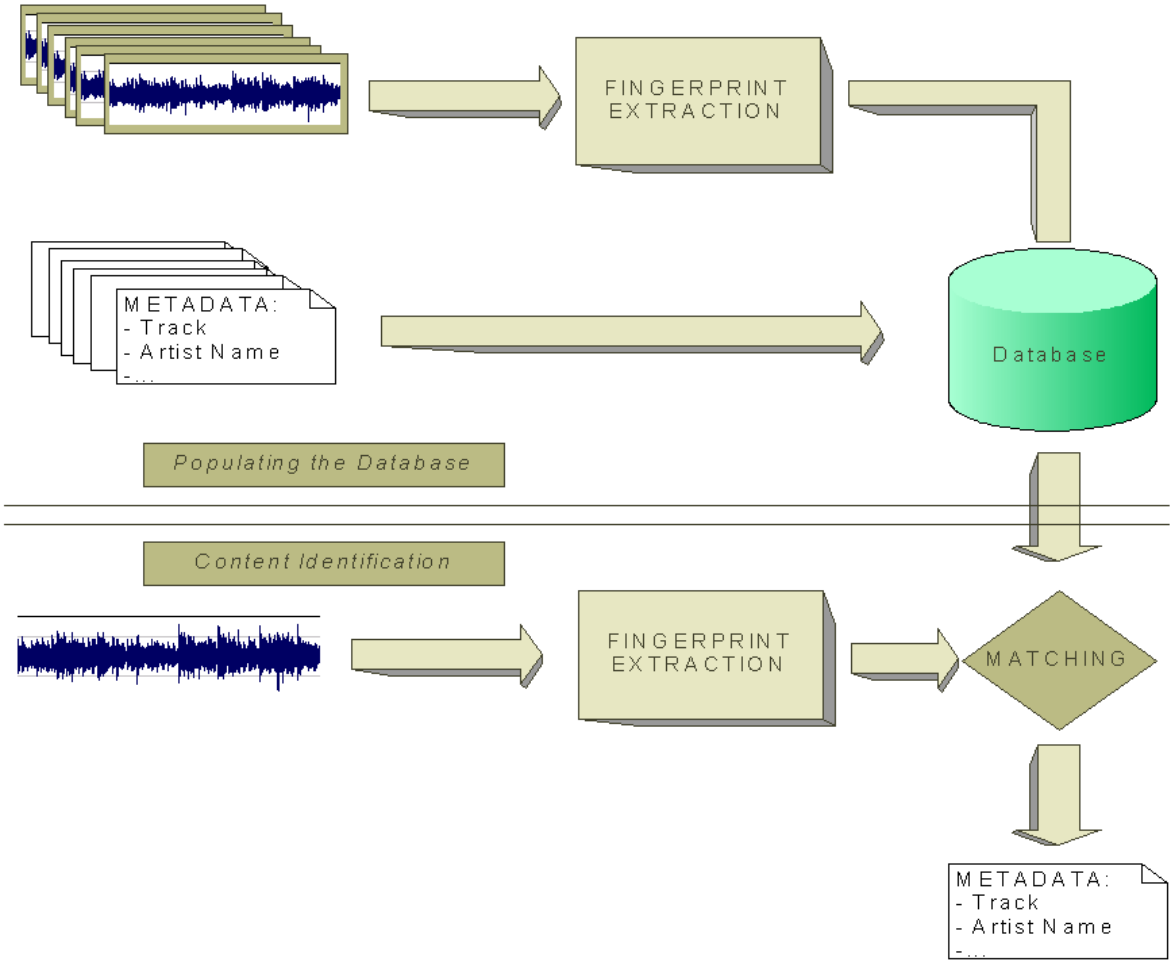
Εικόνα 3: *Δομή Συστήματος Audio Fingerprinting[20].*

2.2.2 Γενική Λειτουργία

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.

Παρακάτω φαίνεται σχηματικά η γενική λειτουργία ενός τέτοιου συστήματος :



***Εικόνα 4**: Γενική λειτουργία Audio Fingerprinting[21].*

Η λειτουργία του audio fingerprinting είναι παρόμοια με αυτή του ανθρώπινου αυτιού. Για παράδειγμα, όταν ακούσει για πρώτη φορά ένα μουσικό κομμάτι και μάθει τις πληροφορίες για αυτό, το καταχωρεί στη ‘’βάση δεδομένων’’ του. Έτσι όταν ξανακούσει κάπου έστω και λίγα δευτερόλεπτα από το ίδιο τραγούδι, άσχετα αν παίζει από CD, MP3, ραδιόφωνο κτλ. ή αν έχει θόρυβο, είναι σε θέση να το αναγνωρίσει.

2.3 ΕΞΑΓΩΓΗ ΤΟΥ AUDIO FINGERPRINT

2.3.1 Η συνάρτηση των Audio Fingerprints

Η εξαγωγή των audio fingerprints γίνεται μέσω μίας συνάρτησης F η οποία παίρνει το ακουστικό αντικείμενο A, που αποτελείται από έναν πολύ μεγάλο αριθμό bits, και επιστρέφει το audio fingerprint B, αποτελούμενο από έναν πολύ μικρότερο αριθμό bits. Δεδομένου ότι δεν επιδιώκεται μαθηματική αλλά ακουστική ισότητα ανάμεσα στα αντικείμενά προκύπτει το συμπέρασμα ότι τα ακουστικά ίσα αντικείμενα πρέπει να έχουν σαν αποτέλεσμα την παραγωγή όμοιων μεταξύ τους audio fingerprints. Και συμπληρωματικά, διαφορετικά αντικείμενα πρέπει να παράγουν διαφορετικά audio fingerprints. Επομένως, με μαθηματικούς όρους, μία σωστά σχεδιασμένη συνάρτηση F που θα παράγει audio fingerprints πρέπει να έχει ένα κατώφλι T, τέτοιο ώστε:

$$\left|\left| F(X) - F(Y) \right|\right| \leq T$$

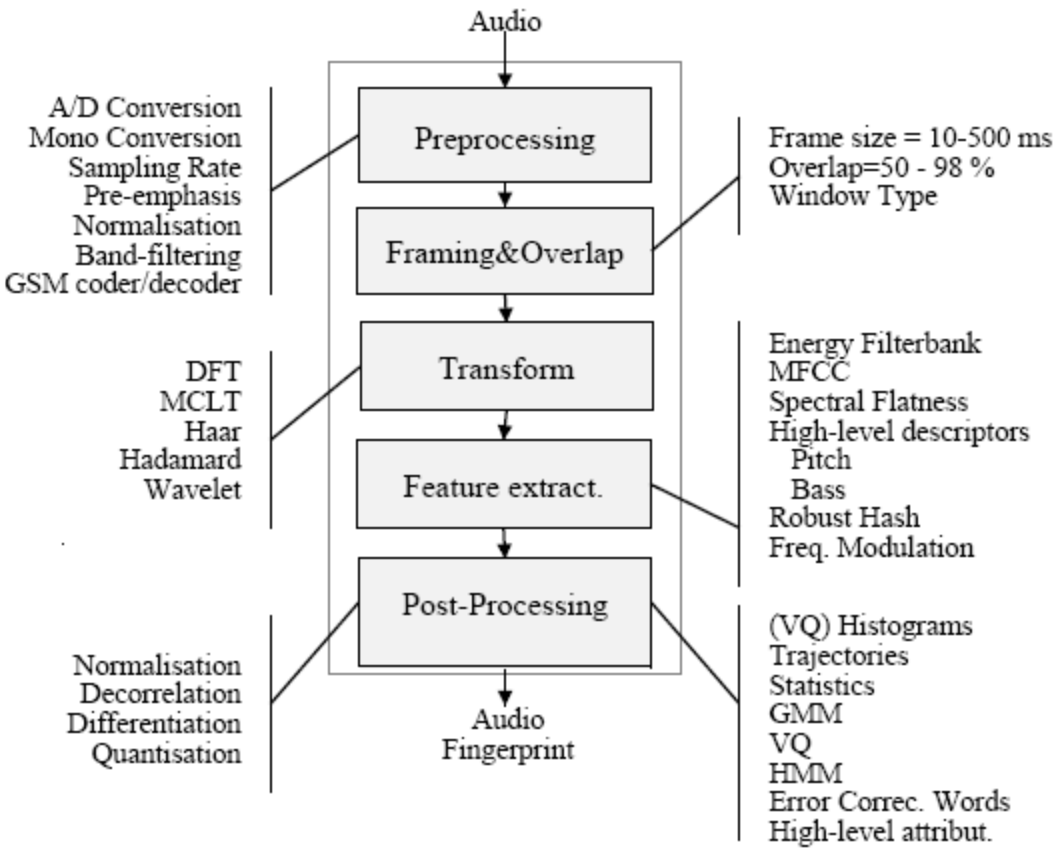
αν τα αντικείμενα X και Y είναι ακουστικά ίσα, και

$$\left|\left| F(X) - F(Y) \right|\right| > T$$

Αν τα αντικείμενα X και Y είναι ακουστικά διάφορα.

Η εξαγωγή του ακουστικού αποτυπώματος είναι το πρώτο βασικό μέρος ενός συστήματος αναγνώρισης ήχου με βάση το περιεχόμενο του και όπως ήδη αναφέρθηκε, δεν αρκεί από μόνο του για την αποτελεσματική λειτουργία του συστήματος. Παρά τις διαφορετικές τεχνικές εξαγωγής των audio fingerprints που έχουν αναπτυχθεί, όλες αποτελούνται από κάποια κοινά βασικά μέρη και ακολουθούν συγκεκριμένα βήματα. Η ιδέα είναι η ίδια : με βάση εύρωστα χαρακτηριστικά του σήματος εξάγεται ένα ακουστικό αποτύπωμα που οι διαστάσεις του είναι κατά πολύ μικρότερες από αυτές του ίδιου του ακουστικού αντικειμένου.

Τα επιμέρους βήματα της διαδικασίας εξαγωγής ενός audio fingerprint φαίνονται στην Εικόνα 5 και περιγράφονται στη συνέχεια.



Εικόνα 5: *Εξαγωγή του Audio Fingerprint[19].*

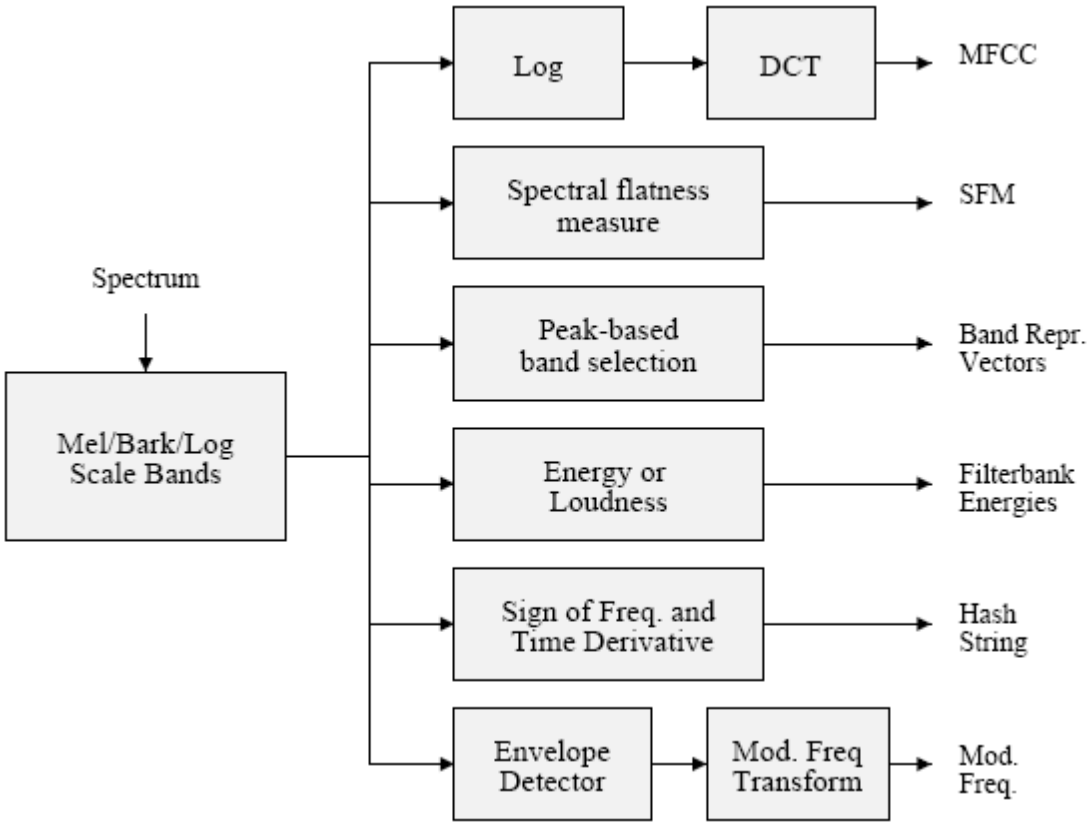
Preprocessing : Αρχικά το σήμα ψηφιοποιείται, εάν είναι απαραίτητο, και στη συνέχεια μετατρέπεται σε μία πρότυπη μορφή, π.χ. μονοφωνικό, 16 bits PCM, συχνότητας 5kHz[21],[22]. Κάποιες φορές το σήμα υφίσταται επεξεργασία ώστε να προσομοιώνει το κανάλι μεταφοράς του, π.χ. υπόκειται ζωνοπερατό φιλτράρισμα όταν πρόκειται για αναγνώριση μέσω τηλεφώνου ή υπόκειται GSM κωδικοποίηση/ αποκωδικοποίηση σε ένα σύστημα αναγνώρισης μέσω κινητού τηλεφώνου.

Framing & Overlap : Μία βασική υπόθεση σε αυτό το σημείο είναι ότι το σήμα μπορεί να θεωρηθεί στάσιμο για ένα διάστημα μερικών milliseconds. Επομένως το σήμα χωρίζεται σε frames σε κάθε ένα από τα οποία εφαρμόζεται κάποιο πολλαπλασιαστικό παράθυρο, π.χ. Hanning, για να ελαχιστοποιηθούν οι ασυνέχειες στην αρχή και το τέλος κάθε frame. Επίσης, πρέπει να υπάρχει μεγάλη επικάλυψη μεταξύ των frames ώστε να εξασφαλιστεί η ευρωστία σε χρονικές ολισθήσεις(time-shifting).

Transformation : Στη συνέχεια κάθε frame μετασχηματίζεται. Οι περισσότερες μέθοδοι χρησιμοποιούν μετασχηματισμούς από το πεδίο του χρόνου στο πεδίο της συχνότητας.

Σκοπός είναι, κυρίως, να επιτευχθεί συμπίεση και αφαίρεση θορύβου. Οι συνηθέστεροι μετασχηματισμοί είναι: Fast Fourier Transform(FFT)[22],[23],[24], Discrete Cosine Transform(DCT)[25],[26],[27],[28], Haar Transform, Walsh-Hadamard Transform, Modulated Complex Lapped Transform και άλλοι.

Feature Extraction : Σε αυτό το βήμα εξάγονται τα χαρακτηριστικά στα οποία θα βασιστεί ο υπολογισμός του ακουστικού αποτυπώματος. Αναλόγως με την τεχνική μπορούμε να δούμε μια πλειάδα χαρακτηριστικών. Τα χαρακτηριστικά που χρησιμοποιούνται είναι κυρίως : οι συντελεστές Fourier[24],[29], οι Mel-Frequency Cepstrum Coefficients (MFCC)[21],[25],[26],[28], το Spectral Flatness Measure (SFM)[30], οι ενέργειες συγκεκριμένων μπαντών του φάσματος[22],[31], οι Linear Predictive Coding (LPC) coefficients και άλλα. Στην Εικόνα 6 φαίνονται κάποια παραδείγματα εξαγωγής χαρακτηριστικών.



Εικόνα 6: *Παραδείγματα εξαγωγής χαρακτηριστικών[19].*

Post-Preprocessing : Τα χαρακτηριστικά που έχουν εξαχθεί από προηγούμενο βήμα σε κάποιες τεχνικές χρησιμοποιούνται κατευθείαν για να δημιουργήσουν το ακουστικό αποτύπωμα. Παρόλα αυτά υπάρχουν διαφορετικές μέθοδοι στις οποίες απαιτείται περαιτέρω επεξεργασία, η οποία υλοποιείται σε αυτό το βήμα. Αυτό σημαίνει ότι τα

χαρακτηριστικά μπορούν να κβαντιστούν[22],[31], να παραγωγιστούν[21],[25],[30], να κανονικοποιηθούν[25], να υποστούν διάφορους μετασχηματισμούς όπως Oriented Principal Component Analysis (OPCA)[24],[29] και τα λοιπά. Επιπλέον σε αυτό το βήμα πραγματοποιείται μοντελοποίηση των audio fingerprints με διάφορες μεθόδους, όπως π.χ. Hidden Markov Models (HMM)[21],[25],[28], διαδικασία που θεωρείται απαραίτητη από ορισμένες τεχνικές.

2.4 ΑΝΑΖΗΤΗΣΗ ΣΤΗ ΒΑΣΗ ΔΕΔΟΜΕΝΩΝ

Το δεύτερο βασικό μέρος ενός συστήματος αναγνώρισης ήχου βασισμένο στα audio fingerprints κι εξίσου σημαντικό με το πρώτο, είναι η αναζήτηση των ακουστικών αποτυπωμάτων στη βάση δεδομένων. Η αναζήτηση ενός audio fingerprint στη βάση δεδομένων είναι μία “ασαφής αναζήτηση” (fuzzy search). Αυτό σημαίνει ότι με βάση το ακουστικό αποτύπωμα που έχει εξαχθεί από το άγνωστο αντικείμενο, ψάχνουμε για το ποιο audio fingerprint αναφοράς της βάσης δεδομένων μοιάζει περισσότερο με το ζητούμενο.

Η μέθοδος που θα επιλεχθεί να χρησιμοποιηθεί για την αναζήτηση του audio fingerprint εξαρτάται από την αναπαράσταση του. Αυτό σημαίνει ότι ανάλογα με την τεχνική εξαγωγής του ακουστικού αποτυπώματος θα έχουμε και διαφορετική μέθοδο αναζήτησης. Οι συνηθέστερα χρησιμοποιούμενες μέθοδοι από τις διάφορες τεχνικές είναι ευριστικές[32],[33],[34],[35],[36]. Η βασική αρχή των περισσοτέρων, ωστόσο, είναι : η δημιουργία μιας δομής δεδομένων με έναν δείκτη, ούτως ώστε το προκύπτον προς αναζήτηση ακουστικό αποτύπωμα να εντοπίζεται με το μικρότερο δυνατό αριθμό συγκρίσεων, μεταξύ του αποτυπώματος αναζήτησης και αυτών που βρίσκονται αποθηκευμένα στη βάση δεδομένων. Σκοπός αυτής της πρακτικής είναι να γίνει η αναζήτηση λιγότερο χρονοβόρα.

Ένας άλλος βασικός παράγοντας στην αναζήτηση των audio fingerprints είναι το είδος της απόστασης που θα χρησιμοποιηθεί κατά τη σύγκριση. Βασιζόμενοι, λοιπόν, στο ότι πραγματοποιούμε αναζήτηση εγγύτητας, χρησιμοποιούμε διαφόρων ειδών αποστάσεις (distance metrics) από διάφορες τεχνικές audio fingerprinting. Ορισμένες από τις αποστάσεις που προτιμώνται στην αναζήτηση των ακουστικών αποτυπωμάτων είναι : η Ευκλείδεια απόσταση όπως και κάποιες διαφοροποιημένες εκδοχές της, η απόσταση Manhattan, η απόσταση Hamming[22],[31] και άλλες.

Τέλος, κατά τη σύγκριση των audio fingerprint αναλόγως με την απόσταση που χρησιμοποιείται έχουμε και κάποια αποτελέσματα. Το σημαντικό, λοιπόν, είναι ο τρόπος λήψης της απόφασης : αν το ακουστικό αποτύπωμα που αναζητείται βρίσκεται στη βάση δεδομένων ή όχι. Για να εντοπιστεί ένα αποτύπωμα στη βάση δεδομένων πρέπει τα

αποτελέσματα που λαμβάνονται να βρίσκονται κάτω από ένα κατώφλι. Το κατώφλι αυτό εξαρτάται από την μέθοδο με την οποία εξήχθη το audio fingerprint, από την ομοιότητα των ακουστικών αποτυπωμάτων της βάσης δεδομένων και από το μέγεθος της βάσης δεδομένων. Όσο μεγαλύτερη είναι η βάση, τόσο αυξάνεται η πιθανότητα να συμβεί ένα false positive, δηλαδή να γίνει μία τυχαία και λανθασμένη ταυτοποίηση. Γίνεται επομένως απόλυτα αντιληπτό το πόσο δύσκολη είναι η επιλογή αυτού του κατωφλίου και πόσοι διαφορετικοί παράγοντες επηρεάζουν αυτήν και μόνο την απόφαση.

2.5 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 ορίζει κατά πόσον ένα audio fingerprint είναι αρκετά “δυνατό” ώστε ένα ηχητικό απόσπασμα να αναγνωριστεί ακόμη και αν έχει υποστεί σοβαρές παραμορφώσεις. Ο ρυθμός false negative χρησιμοποιείται για να εκφράσει την ευρωστία. Όταν τα audio fingerprints ακουστικά ίσων αντικειμένων είναι πολύ διαφορετικά για να οδηγήσουν σε θετική αναγνώριση, τότε συμβαίνει ένα false negative, δηλαδή δεν γίνεται αναγνώριση ενώ θα έπρεπε να είχε γίνει. Προκειμένου να επιτευχθεί ικανοποιητική ευρωστία το audio fingerprint πρέπει να βασίζεται σε χαρακτηριστικά του ήχου που μένουν ανεπηρέαστα στο θόρυβο. Με αυτόν τον τρόπο σημαντικά παραμορφωμένα ακουστικά αντικείμενα θα οδηγούν σε σχεδόν ταυτόσημα audio fingerprints.

Αξιοπιστία : Ο δείκτης αυτός ορίζει πόσο αξιόπιστη είναι η αναγνώριση, δηλαδή πόσο συχνά ένα ηχητικό απόσπασμα μπορεί να αναγνωριστεί λάθος. Ο ρυθμός false positive χρησιμοποιείται για να εκφράσει την αξιοπιστία. Ένα false positive συμβαίνει όταν γίνεται αναγνώριση ενώ δεν θα έπρεπε να έχει γίνει.

Μέγεθος : Ο δείκτης αυτός αφορά το μέγεθος των audio fingerprints, το οποίο συνήθως εκφράζεται σε bits per second. Για τη γρήγορη αναζήτηση τους τα audio fingerprints, στην πλειοψηφία των περιπτώσεων, αποθηκεύονται στη μνήμη RAM. Γίνεται φανερό επομένως, πόσο σημαντικός είναι ο ρόλος του μεγέθους των audio fingerprints αφού καθορίζει τις απαιτήσεις σε μνήμη του server.

Granularity : Ο δείκτης αυτός ορίζει πόσα δευτερόλεπτα ενός ακουστικού αντικειμένου χρειάζονται για να εξαχθεί ένα audio fingerprint και να αναγνωριστεί το ηχητικό απόσπασμα. Ο δείκτης αυτός εξαρτάται από τη συνάρτηση εξαγωγής των audio fingerprints. Προκειμένου η ποιότητα αναγνώρισης να είναι ικανοποιητική, απαιτείται ένα ελάχιστο κάτω όριο της

χρονικής διάρκειας του ηχητικού αποσπάσματος, το οποίο καθορίζει η συνάρτηση εξαγωγής ακουστικών αποτυπωμάτων F.

Ταχύτητα Αναζήτησης : Ο δείκτης αυτός ορίζει το χρόνο που χρειάζεται για να βρεθεί ένα audio fingerprint στη βάση δεδομένων. Ο χρόνος αυτός εξαρτάται από το πλήθος των ακουστικών αντικειμένων που βρίσκονται αποθηκευμένα στη βάση και επηρεάζεται όσο αυτό μεγαλώνει.

Οι πέντε αυτές παράμετροι συγκεντρώνουν τα σημαντικότερα χαρακτηριστικά των audio fingerprints και επηρεάζουν άμεσα η μία την άλλη[18],[19].

2.6 ΠΕΡΙΟΡΙΣΜΟΙ

Κάθε σύστημα αναγνώρισης ήχου με τη χρήση των audio fingerprints θα πρέπει να ικανοποιεί κάποιες προϋποθέσεις και περιορισμούς προκειμένου να λειτουργεί αξιόπιστα και κατά το δυνατόν ικανοποιητικά. Οι περιορισμοί αυτοί παρουσιάζονται παρακάτω :

- Η εξαγωγή των audio fingerprints, είναι πολύ σημαντικό, να είναι γρήγορη και υπολογιστικά απλή διαδικασία. Έτσι καθίσταται δυνατή η χρήση τους σε εφαρμογές με περιορισμένες υπολογιστικές δυνατότητες .

- Τα audio fingerprints πρέπει να είναι όσο το δυνατό μικρότερα ώστε να μειωθούν οι απαιτήσεις σε μνήμη και χωρητικότητα.

- Η αναζήτηση στη βάση δεδομένων πρέπει να είναι γρήγορη και να μην είναι υπολογιστικά πολύπλοκη ώστε η ταυτοποίηση να γίνεται άμεσα ακόμη και σε πολύ μεγάλες βάσεις δεδομένων και με περιορισμένους υπολογιστικούς πόρους.

- Η μέθοδος αναζήτησης πρέπει να μπορεί να πραγματοποιήσει την αναγνώριση ανάμεσα από έναν μεγάλο αριθμό ακουστικών αντικειμένων της βάσης δεδομένων, γρήγορα και αξιόπιστα, δίνοντας σωστά αποτελέσματα. Σκοπός είναι να μειωθούν τα false positives και false negatives, παίρνοντας ακριβή αποτελέσματα.

- Το σύστημα θα πρέπει να μπορεί να κάνει αξιόπιστη αναγνώριση ανεξαρτήτως της συμπίεσης και της παραμόρφωσης του ήχου ή των παρεμβολών από το κανάλι μετάδοσης. Αυτό σημαίνει ότι το audio fingerprint που εξάγεται πρέπει να είναι εύρωστο. Πρέπει ανεξαρτήτως της μορφής παραμόρφωσης του σήματος, όπως ισοστάθμιση, θόρυβο, D-A/A-D μετατροπές, κωδικοποιήσεις φωνής και ήχου όπως GSM και MP3, χρονική ολίσθηση και άλλα, να μπορεί να αναγνωρίσει ακουστικά αντικείμενα με τη χρήση ενός αποσπάσματος λίγων δευτερολέπτων.

Συνοψίζοντας όλα τα προηγούμενα, μία εφαρμογή αναγνώρισης ήχου, που ανταποκρίνεται στις παραπάνω βασικές προϋποθέσεις μπορεί να λειτουργήσει με μεγάλη ταχύτητα, αξιοπιστία, ως αναπόσπαστο κομμάτι απαιτητικών εφαρμογών και χρησιμοποιώντας περιορισμένους υπολογιστικούς πόρους, χωρίς να επιβαρύνει το ευρύτερο σύστημα[18],[19].

2.7 ΕΦΑΡΜΟΓΕΣ ΤΟΥ AUDIO FINGERPRINTING

Η χρήση της τεχνικής Audio Fingerprinting είναι κυρίως διαδεδομένη στην αναγνώριση μουσικών κομματιών, όπου και βρίσκει μία πλειάδα εφαρμογών. Εδώ παρατίθενται οι βασικότερες :

- Η παρακολούθηση ραδιοφωνικών εκπομπών είναι μία από τις πιο γνωστές και αφορά την δημιουργία λιστών των μουσικών κομματιών που παίζονται μέσω του ραδιοφώνου ή του διαδικτύου, κυρίως για στατιστικούς λόγους και λόγους πνευματικών δικαιωμάτων. Σε ένα τέτοιο σύστημα πολλά monitoring sites εξάγουν τα ακουστικά αποτυπώματα από τα κανάλια εκπομπής και ένα κεντρικό site αναλαμβάνει και συλλέγει τα αποτυπώματα σε μία μεγάλη βάση δεδομένων που έχει[21],[22],[25],[30],[31]και κάνει την αναγνώριση.

- Μία άλλη εφαρμογή είναι η αναγνώριση μουσικών κομματιών μέσω του κινητού τηλεφώνου[22],[31]. Βασισμένη στη συγκεκριμένη κατηγορία, έχει αναπτυχθεί μία πληθώρα applications για να εξυπηρετήσει αυτή την ανάγκη. Επιπλέον σε αυτή την κατηγορία ανήκουν κι άλλες παρόμοιες εφαρμογές, όπως π.χ. σε ένα διαδικτυακό ραδιοφωνικό σταθμό που υποστηρίζει τέτοιου είδους λειτουργία, ο ακροατής πατάει ένα κουμπί και βλέπει τις πληροφορίες που αναζητεί στην οθόνη του.

- Το φιλτράρισμα του μουσικού υλικού που διακινείται στο διαδίκτυο μέσω του file sharing είναι μία ακόμη εφαρμογή χρήσης των audio fingerprints. Έτσι με απώτερο ενδεχομένως σκοπό την καταπολέμηση της πειρατείας[37] μπορεί να γίνει παρακολούθηση και καταγραφή των μουσικών κομματιών που διακινούνται.

- Ένας ακόμη τρόπος χρήσης του audio fingerprinting είναι η δυνατότητα οργάνωσης μουσικών συλλογών από χρήστες. Παραδείγματος χάρη, μία συλλογή τεραστίου όγκου, με χιλιάδες MP3, "άτακτα" δομημένων είναι αδύνατο να λειτουργήσει χρηστικά δίνοντας τα σωστά αποτελέσματα. Ωστόσο το κατάλληλο πρόγραμμα που αναγνωρίζει καθένα από τα κομμάτια, εγγράφει τα μεταδεδομένα σε αυτό και ονομάζει τα αντίστοιχα αρχεία, μπορεί να οργανώσει μία τέτοια συλλογή και να παρέχει τις σωστές πληροφορίες.

- Τέλος, υπάρχουν και άλλες πολλές εφαρμογές που χρησιμοποιώντας τα audio fingerprints μπορούν να πραγματοποιήσουν την εξέταση ακεραιότητας μουσικών δεδομένων, την επαλήθευση πως μία καλλιτεχνική δουλειά ανήκει σε συγκεκριμένο καλλιτέχνη και άλλα παρόμοια παραδείγματα.

2.8 ΤΕΧΝΟΛΟΓΙΕΣ AUDIO FINGERPRINTING

Υπάρχουν πολλές διαφορετικές τεχνολογίες που χρησιμοποιούν audio fingerprints για την αναγνώριση ακουστικού υλικού. Εδώ θα παρουσιαστούν κάποιες από αυτές τις τεχνολογίες και θα αναλυθούν κυρίως σε τέσσερα τεχνικά χαρακτηριστικά τους :

- Η πολυπλοκότητα εξαγωγής audio fingerprint.
- Ο αριθμό δειγμάτων που χρειάζονται για την εξαγωγή του audio fingerprint και το μέγεθος του αποτυπώματος.
- Η ταχύτητα αναζήτησης στη βάση δεδομένων.
- Η αξιοπιστία.

Shazam

Το σύστημα audio fingerprinting Shazam, αναπτύχθηκε από την εταιρεία Shazam Entertainment Limited. Το σύστημα χρησιμοποιεί ένα ηχητικό απόσπασμα διάρκειας 15 δευτερολέπτων. Ο αλγόριθμος χρησιμοποιεί μια συνδυαστική ανάλυση συσχέτισης χρόνου-συχνότητας του ήχου και μία συνάρτηση κατακερματισμού (hash function) δημιουργώντας τα audio fingerprints που τελικά αποθηκεύει στη βάση δεδομένων.

Η ταχύτητα αναζήτησης στη βάση δεδομένων είναι υψηλή. Σε μία βάση δεδομένων 20000 μουσικών κομματιών σε έναν προσωπικό υπολογιστή, ο χρόνος αναζήτησης είναι της τάξεως των 5 – 500 milliseconds, αριθμός που μεταβάλλεται με τη ρύθμιση συγκεκριμένων παραμέτρων. Το σύστημα είναι σε θέση να αναγνωρίσει, μόλις σε μερικές εκατοντάδες milliseconds, ακόμη και πολύ κακής και εξασθενημένης ποιότητας ηχητικά αποσπάσματα. Ενώ για ακουστικό υλικό “ραδιοφωνικής ποιότητας”, η αναγνώριση μπορεί να γίνει σε χρόνο μικρότερο από 10 milliseconds.

Σύμφωνα με την Shazam Entertainment Ltd η αξιοπιστία του συστήματος είναι ιδιαίτερα υψηλή, όπως έδειξαν τα αποτελέσματα πειραμάτων σε μία βάση δεδομένων 2 εκατομμυρίων μουσικών κομματιών . Η ίδια η εταιρεία, ωστόσο, δηλώνει ότι περιστασιακά λαμβάνει αναφορές για false positives, παρόλα αυτά επισημαίνει ότι το επιτρεπόμενο ποσοστό των false positives είναι μία σχεδιαστική παράμετρος του συστήματός της, που έχει επιλεγεί κατάλληλα για την εφαρμογή.

Ο αλγόριθμος είναι πολύ γρήγορος και μπορεί να χρησιμοποιείται για την παρακολούθηση πνευματικών δικαιωμάτων σε μια ταχύτητα αναζήτησης πάνω από 1000 φορές ταχύτερη σε σχέση με τον πραγματικό χρόνο, επιτρέποντας έτσι σε ένα μέτριο διακομιστή να παρακολουθεί με επάρκεια πολλές ροές πολυμέσων[38].

AudioID

Το σύστημα αναγνώρισης μουσικών κομματιών AudioID[30] 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 χρειάζονται 352800 δείγματα. Το αποτύπωμα που εξάγεται έχει μέγεθος 2.5 kBytes ανά λεπτό, δηλαδή 334 Bytes για 8 δευτερόλεπτα που απαιτούνται.

Η εξαγωγή του audio fingerprint βασίζεται στο AudioSpectrumFlatness Low Level Descriptor όπως ορίζει το πρότυπο MPEG-7[39] . 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. Η μέτρηση αφορά σχετικά πειράματα του Fraunhofer που έγιναν με μία 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 μία βάση δεδομένων 15000 αποτυπωμάτων και η ορθή αναγνώριση ήταν πάνω από 98%, ενώ σε άλλη σειρά πειραμάτων που πραγματοποιήθηκε με μεγαλύτερη βάση δεδομένων, 90000 αποτυπωμάτων, τα αποτελέσματα ήταν ακόμη καλύτερα με την ορθή αναγνώριση να ανέρχεται στο 99% αυτή τη φορά.

TRM

Τη δική της τεχνολογία, που ονομάζεται TRM[37], δημιούργησε η εταιρεία Relatable. Η τεχνολογία αυτή παράγει ένα audio fingerprint μεγέθους 512 bits και χρειάζεται χρόνο 30 δευτερολέπτων ηχητικού αποσπάσματος.

Σχετικά με την αξιοπιστία του συστήματος τα αποτελέσματα δεν ήταν τόσο υψηλά. Αυτό αποδεικνύεται από τα σχετικά πειράματα της εταιρείας, όπου η ορθή αναγνώριση έφτασε οριακά στο 73%. Το σύστημα βέβαια έχει σχεδιαστεί περισσότερο για να έχει χαμηλή πολυπλοκότητα παρά για να επιτυγχάνει υψηλή αξιοπιστία.

Όσον αφορά την αναζήτηση στη βάση δεδομένων, είναι γρήγορη. Σύμφωνα με τη Relatable, σε μία βάση δεδομένων με πάνω από 5 εκατομμύρια TRM audio fingerprints που κατέχει η εταιρεία All Media Guide (AMG)[40], είναι δυνατόν να εξυπηρετηθούν πάνω από 5000 αναζητήσεις ανά δευτερόλεπτο.

Όσον αφορά τον τρόπο εξαγωγής των audio fingerprints είναι γνωστό ότι το σύστημα είναι σχεδιασμένο να έχει χαμηλή πολυπλοκότητα εξαγωγής των αποτυπωμάτων. Παρόλα αυτά δεν έχει αποκαλυφθεί από την εταιρεία η ακριβής μέθοδος εξαγωγής.

MusicDNA

Το σύστημα MusicDNA ανέπτυξε η εταιρεία Cantametrix. Η τεχνολογία του συστήματος αυτού χρησιμοποιεί ένα ηχητικό απόσπασμα 15 δευτερολέπτων δειγματοληπτούμενο στα 11025 Hz. Χρειάζονται δηλαδή 165376 δείγματα. Το audio fingerprint που εξάγεται είναι μεγέθους μικρότερου από 100 Bytes και είναι ένα διάνυσμα διαστάσεως 30.

Σε πειράματα που πραγματοποιήθηκαν από την Cantametrix για την αξιοπιστία του συστήματος έγινε φανερό ότι όσο μεγαλώνει το μέγεθος της βάσης δεδομένων, τόσο μεγαλώνει και ο αριθμός των false positives, γίνονται δηλαδή τυχαίες ταυτοποιήσεις, ενώ τα false negatives μένουν οριακά αμετάβλητα.

Τα αποτελέσματα αυτά φαίνονται παρακάτω :

Large-scale test results						
DB Size	217,000		527,000		1,042,000	
Format	False Positive	False Negative	False Positive	False Negative	False Positive	False Negative
MP3 128 kb/s (Blade, Lame encoders)	1.53%	0.42%	2.11%	0.4%	2.68%	0.32%
MP3 32 kb/s (Blade, Lame encoders)	1.96%	2.58%	2.69%	2.50%	3.40%	2.33%

Εικόνα 7: *Πειράματα Αξιοπιστίας[34].*

Η αναζήτηση στη βάση δεδομένων είναι γρήγορη. Χρειάζονται λιγότερα από 300 milliseconds σε έναν υπολογιστή με 864 MHz για την αναζήτηση ενός αποτυπώματος 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

Αυτό το σύστημα R.A.R.E (Robust Audio Recognition Engine) σχεδιάστηκε από την Microsoft και χρησιμοποιεί έναν αλγόριθμο που ονομάζεται DDA (Distortion Discriminant Analysis)[24],[29]. Για την εξαγωγή του audio fingerprint χρησιμοποιεί 6 δευτερόλεπτα ηχητικού αποσπάσματος δειγματοληπτημένου στα 11025 Hz. Χρειάζονται δηλαδή 66150 δείγματα. Το αποτύπωμα είναι ένα διάνυσμα πραγματικών αριθμών, διαστάσεως 64. Για την εξαγωγή του αποτυπώματος εφαρμόζεται δύο φορές μετασχηματισμός OPCA (Oriented Principal Component Analysis).

Η αξιοπιστία του συστήματος είναι ιδιαίτερα υψηλή, όπως έδειξαν τα αποτελέσματα πειραμάτων και προσομοιώσεων της Microsoft. Ο ρυθμός false positive είναι 1.5×10^{-8} 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[34]. Πειράματα της Microsoft έδειξαν ότι σε μία βάση δεδομένων 240000 audio fingerprints ο χρόνος αναζήτησης μπορεί να φτάσει τα 0.577 δευτερόλεπτα, για 1000 αναζητήσεις. Η πολυπλοκότητα της τεχνικής αυτή 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 εναπομείναν ποσοστό, δηλαδή το υπόλοιπο 99%, είναι ο χρόνος που χρειάζεται η αναζήτηση στη βάσης δεδομένων[34]. Παρόλα αυτά δεδομένου του διπλού μετασχηματισμού OPCA, που αναφέρθηκε ότι εφαρμόζεται, η πολυπλοκότητα δεν είναι τόσο χαμηλή όσο ισχυρίζεται η εταιρεία.

AudioDNA

Μία ακόμη τεχνολογία audio fingerprinting είναι και το AudioDNA[21]. Το ακουστικό αποτύπωμα που εξάγεται από το σύστημα είναι ένα διάνυσμα διαστάσεως 32, το οποίο περιέχει μια ακολουθία από 32 “γονίδια” όπως αποκαλούνται[21] με βάση τη μέθοδο.

Η διαδικασία εξαγωγής του αποτυπώματος είναι ιδιαίτερα χρονοβόρα. Για μία αναγνώριση απαιτούνται 77 δευτερόλεπτα, λαμβάνοντας υπόψη ότι ο μέσος χρόνος διάρκειας ενός τραγουδιού είναι περίπου 3.5 λεπτά, αυτό σημαίνει ότι είναι μόλις 3.12 φορές πιο γρήγορα από τον πραγματικό χρόνο.

Σχετικά με την ταχύτητα αναζήτησης στη βάση δεδομένων μετά από σχετικά πειράματα προκύπτει ότι είναι 50 φορές ταχύτερη από τον πραγματικό χρόνο. Λαμβάνοντας υπόψη και πάλι ότι ο μέσος χρόνος διάρκειας ενός τραγουδιού είναι περίπου 3.5 λεπτά, σε μία βάση δεδομένων 50000 αποτυπωμάτων, χρειάστηκαν 4.8 δευτερόλεπτα σε έναν υπολογιστή με επεξεργαστή Pentium III 993 MHz.

Όσον αφορά την αξιοπιστία του συστήματος αποδεικνύεται ότι είναι υψηλή, με βάση πειράματα που πραγματοποίησε το ινστιτούτο που ανέπτυξε την τεχνολογία AudioDNA. Τα αποτελέσματα των πειραμάτων φαίνονται στον πίνακα 1.

Πίνακας 1: *Πειράματα Αξιοπιστίας[21].*

Compression format		False negatives	False positives
MP3	128 kbps	0	0
	96 kbps	0	0
	72 kbps	0	0
	48 kbps	1	0
	32 kbps	15	0
	24 kbps	32	0
RealAudio	128 kbps	0	0
	96 kbps	0	0
	72 kbps	0	0
	48 kbps	2	0
	32 kbps	20	0

Philips

Η εταιρεία Philips ανέπτυξε, επίσης, μία δική της τεχνολογία audio fingerprinting[22],[31],[20],[41].[42] Το σύστημα της Philips χρησιμοποιεί ένα ηχητικό απόσπασμα 3.329 δευτερολέπτων δειγματοληπτημένου στα 5515 Hz. Αυτό σημαίνει ότι χρειάζονται 18368 δείγματα. Το audio fingerprint που εξάγεται αποτελείται από 256 sub-fingerprints, 16-bits το καθένα, οπότε έχει μέγεθος 8192 bits, δηλαδή ένα kByte.

Ο αλγόριθμος που κάνει την αναζήτηση στη βάση δεδομένων είναι σχεδιασμένος από την Philips. Μία αναζήτηση σε μία βάση δεδομένων εκατομμυρίων αποτυπωμάτων διαρκεί μόλις μερικά milliseconds. Είναι εμφανές, λοιπόν το πόσο γρήγορη κάνει τη διαδικασία.

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

Σε πειράματα που πραγματοποίησε η Philips, σχετικά με το πόσο εύρωστο είναι το σύστημα της, αποδείχθηκε ότι ο ρυθμός false positive ήταν πολύ χαμηλός, για την ακρίβεια 3.6×10^{-20} . Αυτό κατοχυρώνει ότι η αξιοπιστία του συστήματος είναι πολύ υψηλή.

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

3 ΤΟ ΣΥΣΤΗΜΑ ΠΟΥ ΧΡΗΣΙΜΟΠΟΙΕΙΤΑΙ

Στο κεφάλαιο αυτό περιγράφεται το σύστημα αναγνώρισης ηχητικών αντικειμένων με βάση τα audio fingerprints που χρησιμοποιήθηκε. Το σύστημα είναι ελεύθερο λογισμικό και ονομάζεται “Dejavu”[43]. Το συγκεκριμένο σύστημα είναι υλοποιημένο σε Python και βασίζεται στην τεχνολογία Fingerprinting με χρήση Hash Functions. Αντίστοιχες τεχνολογίες χρησιμοποιούν εφαρμογές, όπως το Shazam, το SoundHound και άλλες. Παρακάτω αναλύεται τόσο η διαδικασία εξαγωγής του ακουστικού αποτυπώματος, όσο και η μέθοδος αναζήτησης στη βάση δεδομένων.

3.1 ΕΞΑΓΩΓΗ ΤΟΥ AUDIO FINGERPRINT

3.1.1 Ο ήχος ως σήμα

Το πρώτο στάδιο στη διαδικασία της εξαγωγής των Audio Fingerprints είναι να οριστεί και να καταγραφεί το σήμα που λαμβάνεται.

Ο ήχος, όπως αποδεικνύεται , είναι ψηφιακά κωδικοποιημένος ως μια μακρά λίστα αριθμών. Σε ένα μη συμπιεσμένο αρχείο .wav, υπάρχουν πολλοί από αυτούς τους αριθμούς, για την ακρίβεια 44100 ανά δευτερόλεπτο, ανά κανάλι. Ένα κανάλι είναι μια ξεχωριστή ακολουθία δειγμάτων που μπορεί να παίξει ένα ηχείο. Για μία διάταξη δύο καναλιών χρησιμοποιούνται δύο ακουστικά. Ένα, μόνο κανάλι, ονομάζεται "μονοφωνικό". Σήμερα, τα σύγχρονα συστήματα ήχου, μπορούν να υποστηρίξουν πολλά περισσότερα κανάλια. Αλλά, εάν ο ήχος δεν καταγραφεί ή αναμειχθεί με ίσο αριθμό καναλιών, τα επιπλέον ηχεία είναι περιττά και κάποια απλά αναπαράγουν το ίδιο με άλλα ηχεία.

Θεωρείται ότι ο μέσος χρόνος διάρκειας των ήχων που χρησιμοποιήθηκαν είναι 5 δευτερόλεπτα. Αυτό σημαίνει ότι ένα αρχείο μήκους 5 δευτερολέπτων έχει σχεδόν 450 χιλιάδες δείγματα.

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

3.1.2 Δειγματοληψία

Το δεύτερο στάδιο στη διαδικασία και αφού ορίστηκε το σήμα ως ένας ακριβής αριθμός δειγμάτων, είναι η σωστή δειγματοληψία.

Η επιλογή των 44100 δειγμάτων ανά δευτερόλεπτο φαίνεται αρκετά αυθαίρετη, αλλά σχετίζεται με το θεώρημα Nyquist-Shannon Sampling. Μέσω του πολύπλοκου αυτού μαθηματικού τρόπου ορίστηκε ένα θεωρητικό όριο στη μέγιστη συχνότητα που δύναται να καταγραφεί με ακρίβεια κατά την εγγραφή. Αυτή η μέγιστη συχνότητα βασίζεται στο πόσο γρήγορα "δειγματοληπτείται" το σήμα.

Προκειμένου αυτό να γίνει πιο εύκολα κατανοητό, δίνεται ένα απλό παράδειγμα :

Έστω ότι κάποιος παρακολουθεί μια λεπίδα ανεμιστήρα που κάνει μια πλήρη περιστροφή με ρυθμό ακριβώς μία φορά το δευτερόλεπτο (1 Hz). Και έστω ότι κρατάει τα μάτια του κλειστά, αλλά τα ανοίγει για μία φορά, ανά δευτερόλεπτο. Εάν ο ανεμιστήρας εξακολουθεί να κάνει ακριβώς μια πλήρη περιστροφή κάθε 1 δευτερόλεπτο, θα φαίνεται σαν να μην έχει μετακινηθεί η λεπίδα του ανεμιστήρα. Κάθε φορά που ο παρατηρητής ανοίγει τα μάτια του, η λεπίδα συμβαίνει να βρίσκεται στο ίδιο σημείο. Υπάρχει όμως ένα πρόβλημα. Στην πραγματικότητα, όπως είναι λογικό, η λεπίδα του ανεμιστήρα θα μπορούσε να κάνει 0, 1, 2, 3, 10, 100 ή ακόμα και 1 εκατομμύριο περιστροφές ανά δευτερόλεπτο και αν δεν γινόταν ποτέ αντιληπτό, θα εξακολουθούσε να φαίνεται στάσιμη. Έτσι, προκειμένου ο παρατηρητής να είναι βέβαιος ότι κάνει σωστές δειγματοληψίες (ή "βλέπει") υψηλότερων συχνοτήτων (ή "περιστροφών"), θα πρέπει να δειγματοληπτεί (ή να "ανοίγει τα μάτια του") πιο συχνά. Για να την ακρίβεια, πρέπει η δειγματοληψία να γίνεται δύο φορές συχνότερα από τη συχνότητα που ενδιαφέρει να καταγραφεί ώστε να υπάρχει η βεβαιότητα ότι ανιχνεύεται σωστά.

Στην περίπτωση της εγγραφής ήχου, "επιτρέπεται" απώλεια σε συχνότητες υψηλότερες από 22050 Hz, δεδομένου ότι το ανθρώπινο αυτί αδυνατεί να συλλάβει ακόμη και συχνότητες άνω των 20.000 Hz. Έτσι, από Nyquist, η δειγματοληψία γίνεται με συχνότητα διπλάσια της μέγιστης, δηλαδή :

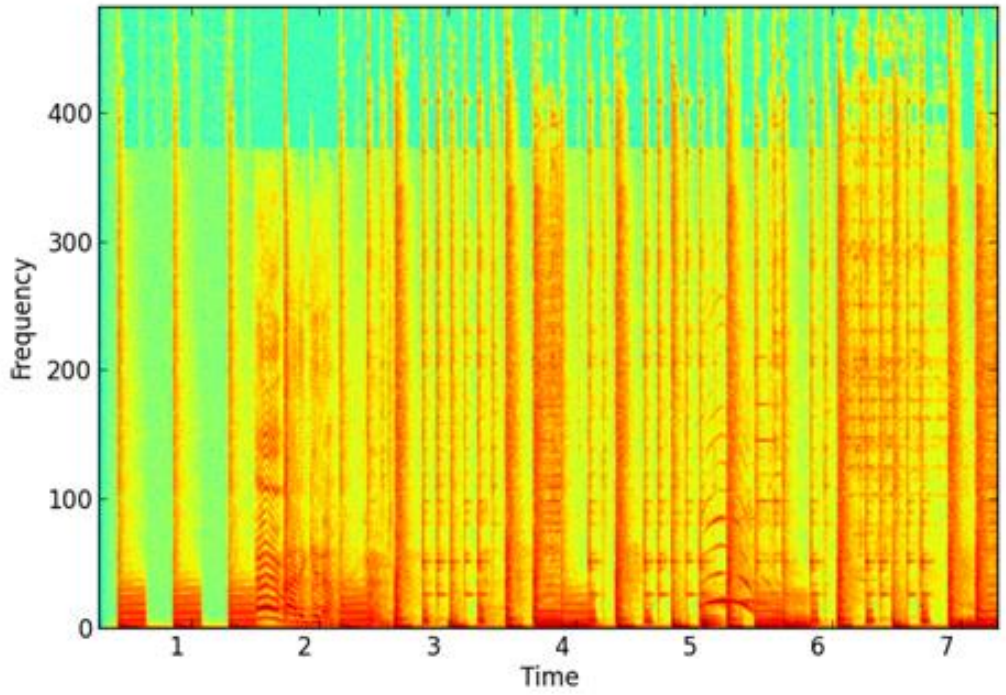
Απαιτούμενα δείγματα ανά δευτερόλεπτο = Υψηλότερη συχνότητα * 2 = 22050 * 2 = 44100

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).

3.1.3 Φασματογράφημα

Το τρίτο στάδιο στη διαδικασία, είναι η παραγωγή ενός φασματογραφήματος βασισμένο στα ορθώς δειγματοληπτημένα δείγματα, του προηγούμενου σταδίου.

Δεδομένου ότι αυτά τα δείγματα δεν είναι σήμα ιδιαίτερα καλής ποιότητας, εφαρμόζεται επαναλαμβανόμενα Fast Fourier Transform (FFT) σε "μικρά παράθυρα" χρόνου, στα δείγματα του ήχου, για να δημιουργηθεί ένα φασματογράφημα του. Ακολουθεί ένα φασματογράφημα των πρώτων δευτερολέπτων ενός τυχαίου ήχου από αυτούς που χρησιμοποιήθηκαν.



Εικόνα 8: *Φασματογράφημα*

Όπως φαίνεται, είναι απλώς ένας πίνακας 2D, όπου το πλάτος είναι συνάρτηση του χρόνου και της συχνότητας. Το FFT δείχνει την ισχύ (πλάτος) του σήματος για τη συγκεκριμένη συχνότητα, δίνοντάς με αυτόν τον τρόπο μια στήλη. Αν επαναληφθεί αυτό αρκετές φορές, μετακινώντας το παράθυρο του FFT και έπειτα τοποθετηθούν οι επιμέρους στήλες μαζί, δημιουργείται ένα φασματογράφημα σε μορφή πίνακα 2D. Εάν η καταγραφή γινόταν για ένα μόνο ηχητικό τόνο, τότε το φασματογράφημα που θα δημιουργούταν θα έπρεπε να έχει

μια ευθεία οριζόντια γραμμή για τη συχνότητα του τόνου. Αυτό συμβαίνει επειδή η συχνότητα δεν διαφέρει από παράθυρο σε παράθυρο.

Είναι σημαντικό να σημειωθεί ότι οι τιμές συχνότητας και χρόνου είναι διακριτοποιημένες, αντιπροσωπεύοντας η καθεμία ένα διάστημα, ενώ το πλάτος λαμβάνει πραγματικές τιμές. Το χρώμα δείχνει την πραγματική τιμή του πλάτους (κόκκινο -> υψηλότερο, πράσινο -> χαμηλότερο) στο εκάστοτε διάνυσμα διακριτοποιημένων συντεταγμένων (χρόνος, συχνότητα).

Το φασματογράφημα, επομένως, χρησιμοποιείται για να αναγνωριστεί ο ήχος και να προσδιοριστεί μοναδικά. Το πρόβλημα είναι ότι πρέπει να απομονωθεί όσο το δυνατόν περισσότερο ο θόρυβος και να βρεθεί ένας αξιόπιστος τρόπος να “συλλαμβάνονται” μοναδικά τα fingerprints από το ηχητικό σήμα.

3.1.4 Ανεύρεση κορυφών

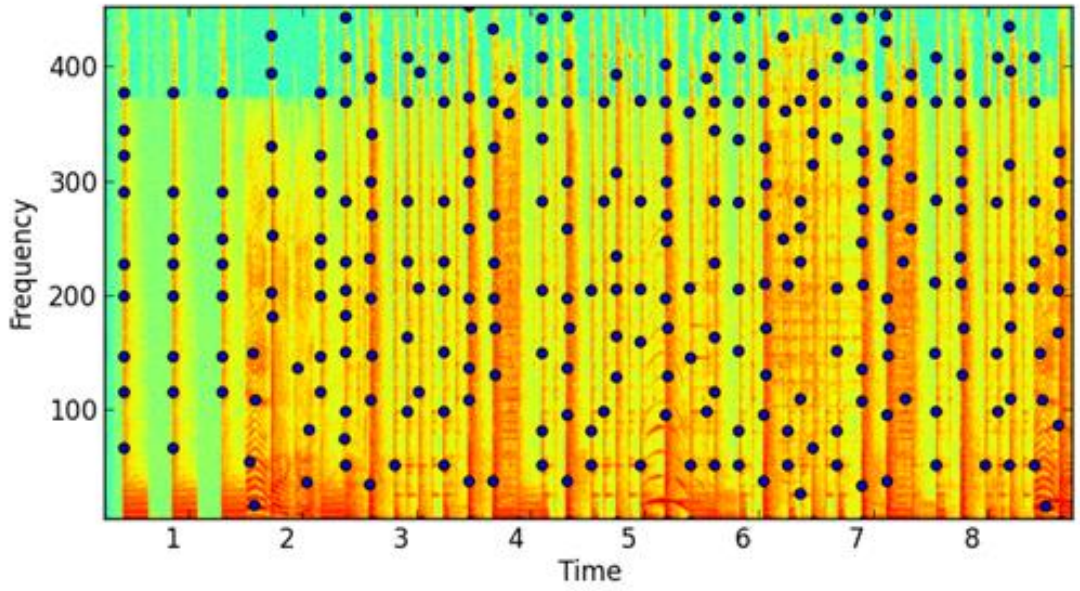
Το τέταρτο στάδιο στη διαδικασία, είναι η ανεύρεση των κορυφών στο φασματογράφημα που παράχθηκε προηγουμένως και δίνει τη λύση στον προηγούμενο προβληματισμό.

Αφού δημιουργήθηκε, λοιπόν, το φασματογράφημα του ηχητικού σήματος, το επόμενο βήμα είναι να βρεθούν 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-"γειτονιά" γύρω από αυτήν. Άλλα ζεύγη (χρόνου, συχνότητας) γύρω από αυτό έχουν χαμηλότερη τιμή πλάτους και επομένως είναι λιγότερο πιθανό να επιβιώσουν από τον θόρυβο.

Η εύρεση των κορυφών είναι μία πολύ δύσκολη διαδικασία. Στο συγκεκριμένο σύστημα το φασματογράφημα αντιμετωπίζεται ως εικόνα και χρησιμοποιούνται τα εργαλεία και οι τεχνικές επεξεργασίας εικόνων από το scipy για να βρεθούν οι κορυφές. Με το συνδυασμό ενός υψηπερατού φίλτρου (που τονίζει τα υψηλά πλάτη) και δομών τοπικών μέγιστων της scipy απομονώνονται οι “δυνατές” κορυφές στο φασματογράφημα.

Οι ανθεκτικές αυτές στο θόρυβο κορυφές, είναι τα σημεία ενδιαφέροντος στο εκάστοτε ηχητικό αντικείμενο, τα οποία και το χαρακτηρίζουν. Το φασματογράφημα δεν χρησιμεύει άλλο αφού εντοπίστηκαν οι κορυφές. Τα πλάτη έχουν εξυπηρετήσει το σκοπό τους.

Στην εικόνα που ακολουθεί φαίνεται η επιλογή των κορυφών για το φασματογράφημα του προηγούμενου παραδείγματος :



Εικόνα 9: *Εντοπισμός κορυφών σε φασματογράφημα*

Είναι ευδιάκριτο ότι υπάρχουν πολλές κορυφές. Δεκάδες χιλιάδες ανά ηχητικό αντικείμενο, για την ακρίβεια. Με τη διαδικασία αυτή απομονώθηκε όλη η μέχρι τώρα σημαντική πληροφορία σε δύο μεταβλητές, το χρόνο και τη συχνότητα, τα οποία έχουν μετατραπεί σε διακριτές, ακέραιες τιμές.

Δημιουργείται, λοιπόν, ένα σύστημα που συγκρατεί κορυφές από ένα σήμα, σε διακριτά ζεύγη (χρόνος, συχνότητα), δίνοντάς κάποια περιθώρια να “επιβιώσουμε” τον θόρυβο. Η πληροφορία του σήματος μετατρέπεται σε διακριτή και ως άμεσο επακόλουθο μειώνεται η πληροφορία των κορυφών από άπειρη σε πεπερασμένη, πράγμα που σημαίνει ότι οι κορυφές που βρέθηκαν σε ένα ηχητικό αντικείμενο θα μπορούσαν να συγκρούονται, εξάγοντας τα ίδια ζευγάρια που εξήγαγαν κορυφές από άλλα αντικείμενα.

3.1.5 Fingerprint hashing

Τέλος, το πέμπτο στάδιο στη διαδικασία, χρησιμοποιεί τις κορυφές που εξήχθησαν και δημιουργεί τα fingerprints μέσω μίας συνάρτησης κατακερματισμού (hash function).

Μέσω της συνάρτησης κατακερματισμού (hash function) πραγματοποιείται ο συνδυασμός των κορυφών σε fingerprints και έτσι παύει να αποτελεί τροχοπέδη το γεγονός ότι μπορεί να εντοπισθούν παρόμοιες κορυφές.

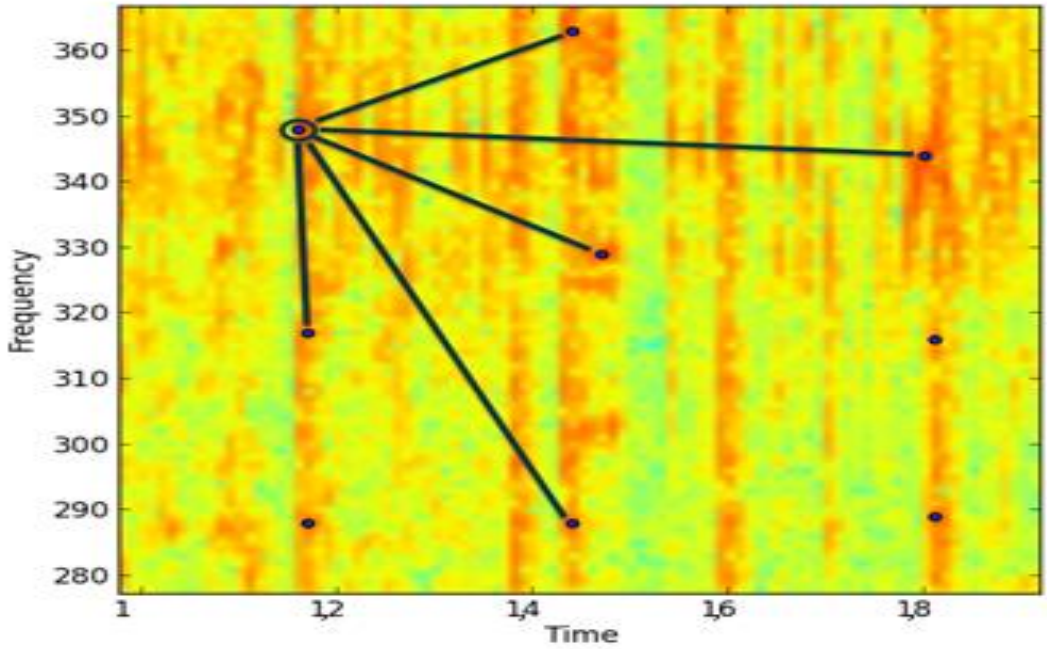
Ένα hash function παίρνει έναν ακέραιο αριθμό ως είσοδο και επιστρέφει έναν άλλο ακέραιο ως έξοδο. Μία καλή συνάρτηση κατακερματισμού, όχι μόνο επιστρέφει τον ίδιο ακέραιο σαν έξοδο κάθε φορά που λαμβάνει την ίδια είσοδο, αλλά έχει και πολύ μικρό αριθμό διαφορετικών εισόδων που δίνουν την ίδια έξοδο.

Με δεδομένο τις κορυφές του φασματογραφήματος και συνδυάζοντας τις συχνότητες των κορυφών αυτών με τη διαφορά χρόνου μεταξύ τους, δημιουργείται ένα hash, που αντιπροσωπεύει ένα μοναδικό αποτύπωμα για το κάθε ηχητικό αντικείμενο.

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

Υπάρχουν πολλοί διαφορετικοί τρόποι να υλοποιηθεί αυτό, το Shazam έχει το δικό του, το SoundHound άλλο, και ούτω καθεξής. Με το να λαμβάνονται υπόψη περισσότερες από μια τιμές κορυφών δημιουργούνται fingerprints που έχουν περισσότερη εντροπία και συνεπώς περιέχουν περισσότερη πληροφορία. Αυτό τα κάνει πολύ καλύτερα στην αναγνώριση ήχου, καθώς θα “συγκρούονται” λιγότερο.

Γίνεται καλύτερα κατανοητό τι συμβαίνει με το παρακάτω ζουμαρισμένο φασματογράφημα:



Εικόνα 10: *Ζουμ σε κορυφές φασματογραφήματος*

Το Shazam στον επίσημο “οδηγό” του (white paper), ονομάζει 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 πολλοί διαφορετικοί τρόποι ομαδοποίησης σημείων και fingerprints. Περισσότερες κορυφές σε ένα fingerprint σημαίνει ένα πιο σπάνιο fingerprint που θα μπορούσε να αναγνωρίσει πολύ καλύτερα ένα τραγούδι. Αλλά περισσότερες κορυφές σημαίνει επίσης και λιγότερο ισχυρό fingerprint εν όψει του θορύβου.

3.2 Η ΒΑΣΗ ΔΕΔΟΜΕΝΩΝ

3.2.1 Εκμάθηση ηχητικού αντικειμένου

Ένα σύστημα αναγνώρισης ήχου έχει δύο βασικούς πυλώνες:

1. Εκμάθηση νέων ηχητικών αντικειμένων δημιουργώντας το fingerprint τους.
2. Αναγνώριση άγνωστων ηχητικών αποσπασμάτων, αναζητώντας τα στη βάση δεδομένων των ήδη υπαρκτών αντικειμένων.

Η βάση δεδομένων υλοποιείται χρησιμοποιώντας MySQL, και θα περιέχει δύο πίνακες:

- fingerprints
- sounds

Πίνακας fingerprints

```
CREATE TABLE fingerprints (
    hash binary(10) not null,
    sound_id mediumint unsigned not null,
    offset int unsigned not null,
    INDEX(hash),
    UNIQUE(sound_id, offset, hash)
);
```

Όπως φαίνεται από το παραπάνω κομμάτι κώδικα, εκτός από ένα hash και ένα ID ήχου, υπάρχει και ένα offset. Αυτό αντιστοιχεί στο χρονικό παράθυρο του φασματογραφήματος από το οποίο προέκυψε το συγκεκριμένο hash. Αυτό θα χρειαστεί σε επόμενο βήμα, όταν θα πρέπει να ξεχωρίσουν τα όμοια hashes. Μόνο τα hashes που "ευθυγραμμίζονται" θα προέρχονται από το πραγματικό σήμα για το ποιο γίνεται η ταυτοποίηση.

Επιπλέον υπάρχει ένα INDEX για το hash. Εξαιτίας του ότι χρειαζόμαστε μια πραγματικά γρήγορη ανάκτηση δεδομένων σε εκείνο το σημείο, ο λόγος ύπαρξης του είναι ιδιαίτερα σημαντικός. Όλα τα queries θα πρέπει να αντιστοιχίζονται με το hash.

Επίσης, ο δείκτης UNIQUE διασφαλίζει ότι δεν θα υπάρξουν διπλότυπα. Με την αποφυγή της δημιουργίας στη βάση δεδομένων αντιγράφων που ήδη υπάρχουν, διαφυλάσσεται ο χώρος και διευκολύνεται η αντιστοίχιση των ήχων.

Ακόμη, χρησιμοποιείται ένα binary (10) πεδίο για το hash. Ο λόγος είναι η ύπαρξη πολλών από αυτά τα hashes και είναι ιδιαίτερα σημαντικό να περιοριστεί η αλόγιστη χρήση του δίσκου για αποθήκευση των τιμών hash μιας και η εξοικονόμηση χώρου είναι επιτακτική ανάγκη.

Για το hash των fingerprint, χρησιμοποιείται ένα hash SHA-1 και στη συνέχεια περιορίζεται στο μισό του μεγέθους του (κρατώντας μόλις τους πρώτους 20 χαρακτήρες). Αυτό μειώνει τη χρήση byte ανά hash στο μισό:

char(40) => char(20) – από τα 40 bytes στα 20 bytes

Στη συνέχεια λαμβάνεται αυτή τη δεκαεξαδική κωδικοποίηση και μετατρέπεται σε δυαδική, μειώνοντας για άλλη μια φορά σημαντικά τον χρησιμοποιούμενο χώρο:

char(20) => binary(10) – από τα 20 bytes στα 10 bytes

Έτσι, από τα 320 bits τελικά καταλήγει στα 80 bits για το πεδίο hash, μια μείωση κατά 75%.

Ενώ λοιπόν, με ένα πεδίο char (40) για κάθε hash, θα απαιτούνταν πάνω από 1 GB χώρου μόνο για τα fingerprints, με πεδίο binary (10), περιορίζεται το μέγεθος του πίνακα σε μόλις 377 MB για 5,2 εκατομμύρια fingerprints.

Όλη η παραπάνω διαδικασία έχει σαν αποτέλεσμα να χαθεί μέρος της πληροφορίας (τα hashes, από στατιστικής άποψης, θα συγκρούονται πολύ πιο συχνά), έχει δηλαδή μειωθεί σημαντικά η "εντροπία" του hash. Ωστόσο, είναι σημαντικό ότι η εντροπία (ή η πληροφορία) περιλαμβάνει επίσης το πεδίο offset, το οποίο είναι 4 bytes. Αυτό φέρνει τη συνολική εντροπία κάθε fingerprint σε:

10 bytes (hash) + 4 bytes (offset) = 14 bytes = 112 bits = 2^112 ~= 5.2+e33 πιθανά fingerprints

Πίνακας sounds

Ο πίνακας “sounds” είναι αρκετά πιο απλός. Ουσιαστικά χρησιμοποιείται μόνο για τη διατήρηση πληροφοριών σχετικά με τους ήχους, με το να αντιστοιχείται ένα sound_id στο όνομα του ήχου.

```
CREATE TABLE sounds (
    sound_id mediumint unsigned not null auto_increment,
    sound_name varchar(250) not null,
    fingerprinted tinyint default 0,
    PRIMARY KEY (sound_id),
    UNIQUE KEY sound_id (sound_id)
);
```

Η σημαία (flag) “fingerprinted” χρησιμοποιείται από το Dejavu εσωτερικά για να αποφασιστεί αν θα δημιουργηθεί το fingerprint για ένα αρχείο ή όχι. Το bit τίθεται αρχικά στο 0 και αλλάζει σε 1 μόνο μετά την ολοκλήρωση της διαδικασίας αποτύπωσης του fingerprint.

3.3 ΑΛΓΟΡΙΘΜΟΣ ΑΝΑΖΗΤΗΣΗΣ:

Συνοψίζοντας όλα τα προηγούμενα : πρώτα δίνεται ως είσοδος ένα κομμάτι ήχου, στη συνέχεια εκτελείται FFT σε επικαλυπτόμενα παράθυρα σε όλο το μήκος του, εξάγονται οι κορυφές και τέλος σχηματίζεται ένα fingerprint.

Αφού έχει πραγματοποιηθεί η διαδικασία του fingerprinting σε γνωστά κομμάτια ήχου, δηλαδή έχει γίνει η εισαγωγή των fingerprints στη βάση δεδομένων και το καθένα φέρει κάποιο sound_ID, τότε μπορεί εύκολα να γίνει η ανάκλησή του.

Ο ψευδοκώδικας έχει την εξής μορφή:

```
channels = capture_audio()

fingerprints_matching = [ ]
for channel_samples in channels
   hashes = process_audio(channel_samples)
   fingerprints_matching += find_database_matches(hashes)

predicted_sound = align_matches(fingerprints_matching)
```

Με κάθε νέα εγγραφή ενός καινούριου δείγματος, το οποίο αποτελεί υποτμήμα του αρχικού ηχητικού κομματιού, εξάγονται κάποια hashes. Τα hashes αυτά έχουν ένα offset, το οποίο όμως είναι σχετικό με την αρχή του δείγματος.

Όταν αρχικά αποτυπώθηκαν τα fingerprints, καταγράφηκε ένα offset στη βάση δεδομένων το οποίο ήταν το απόλυτο offset του hash. Τα σχετικά hashes από το δείγμα και το απόλυτο hash από τη βάση δεδομένων δεν αντιστοιχίζονται ποτέ, εκτός και αν ξεκινήσει η καταγραφή ενός δείγματος ακριβώς από την αρχή του ήχου. Κάτι τέτοιο όμως είναι αρκετά απίθανο.

Παρόλα αυτά, αν και δεν είναι ίδια, παρέχουν μία σημαντική πληροφορία για τις αντιστοιχίσεις του πραγματικού σήματος. Η πληροφορία αυτή, είναι, ότι όλα τα σχετικά offset θα βρίσκονται στην ίδια απόσταση. Αυτό απαιτεί την παραδοχή, ότι το κομμάτι ήχου

παίζεται και δειγματοληπτείται με την ίδια ταχύτητα που καταγράφηκε. Στην πραγματικότητα, θα δημιουργούταν ούτως ή άλλως πρόβλημα, στην περίπτωση που η ταχύτητα αναπαραγωγής ήταν διαφορετική, καθώς αυτό θα επηρέαζε τη συχνότητα της αναπαραγωγής και επομένως και τις κορυφές στο φασματογράφημα.

Σύμφωνα με αυτή την υπόθεση, για κάθε αντιστοίχιση υπολογίζεται η διαφορά μεταξύ των offset:

διαφορά = offset της βάσης δεδομένων από τον αρχικό ήχο - offset δείγματος από την εγγραφή

η οποία αποδίδει πάντα έναν ακέραιο αριθμό, αφού ο ήχος στη βάση δεδομένων θα έχει πάντα μήκος μεγαλύτερο η ίσο του δείγματος. Όλες οι πραγματικές αντιστοιχίσεις έχουν την ίδια διαφορά. Έτσι, οι αντιστοιχίσεις στη βάση δεδομένων αλλάζουν ώστε να έχουν την εξής μορφή:

(sound_id, διαφορά)

Τελικά, ελέγχοντας όλες τις αντιστοιχίσεις μπορεί να προβλεφθεί το ID του ηχητικού αντικειμένου για το οποίο εντοπίζεται ο μεγαλύτερος αριθμός διαφοράς, κάτι που είναι εύκολο να οπτικοποιηθεί αν απεικονιστεί ως ιστόγραμμα.

4 ΑΠΟΤΕΛΕΣΜΑΤΑ

Στο κεφάλαιο αυτό παρουσιάζονται τα αποτελέσματα προσομοιώσεων που έγιναν σχετικά με την ταχύτητα, την απόδοση, και την αξιοπιστία του συστήματος. Οι προσομοιώσεις έγιναν σε προσωπικό υπολογιστή με επεξεργαστή Intel Pentium 2127U, 1.9GHz και μνήμη RAM 4GByte.

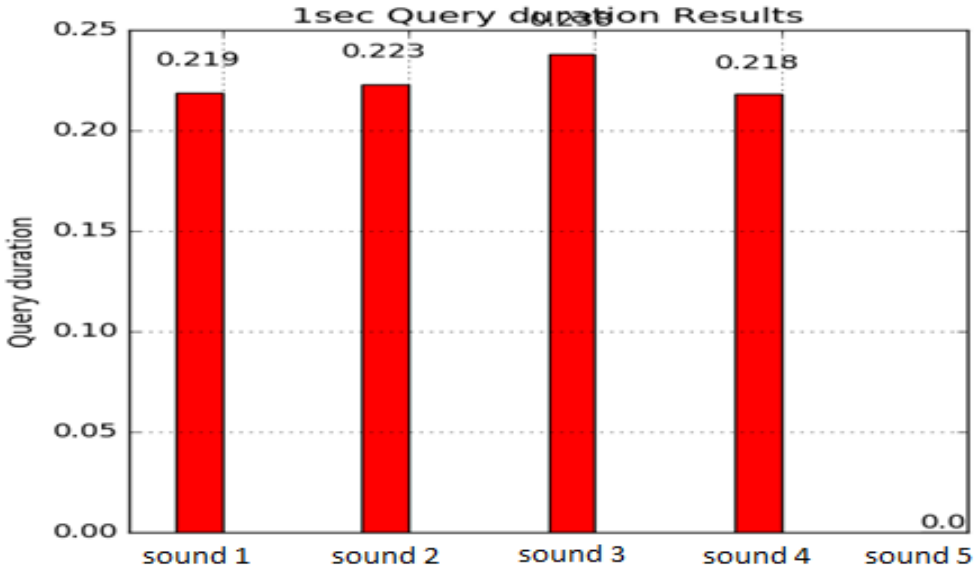
4.1 ΤΑΧΥΤΗΤΑ ΑΝΤΙΣΤΟΙΧΙΣΗΣ

Για να αξιοποιηθεί πραγματικά το πλεονέκτημα ενός συστήματος ηχητικών αποτυπωμάτων, θα πρέπει να μην χρειάζεται πολύς χρόνος για την αποτύπωση των fingerprint. Ακόμη και με ελάχιστα δευτερόλεπτα ακρόασης είναι πολύτιμο να προκύπτουν γρήγορα και ορθά αποτελέσματα.

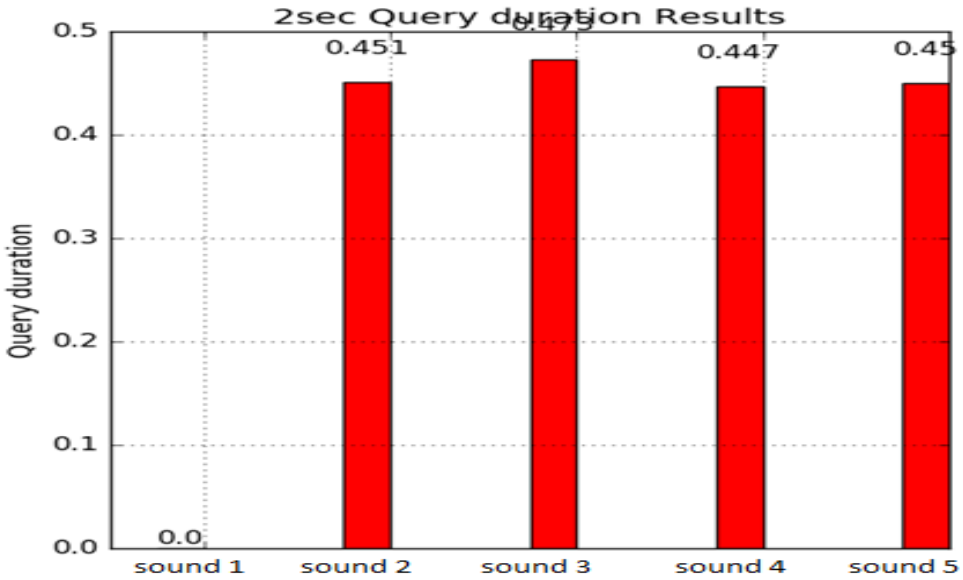
Σε προσομοιώσεις που έγιναν για τον προσδιορισμό της ταχύτητα αντιστοίχισης (Query Duration) επιλέχθηκαν τυχαία 5 ήχοι (διάρκειας μεταξύ 2 και 8 δευτερολέπτων). Η βάση δεδομένων που χρησιμοποιήθηκε περιείχε reference audio fingerprints από 94 συνολικά ήχους (διάρκειας 1 έως 16 δευτερολέπτων), μεταξύ των οποίων ήταν και οι 5 που επιλέχθηκαν.

Η τυχαία επιλογή των ήχων που χρησιμοποιήθηκαν βασίζεται στο γεγονός ότι η ταχύτητα αντιστοίχισης είναι ως επί το πλείστον ανεξάρτητη του ηχητικού αντικειμένου και εξαρτάται περισσότερο από το μήκος του δημιουργούμενου φασματογραφήματος. Το μήκος του φασματογραφήματος εκφράζει ουσιαστικά την διάρκεια του ηχητικού αποσπάσματος. Μεγαλύτερη διάρκεια σημαίνει και μεγαλύτερο μήκος φασματογραφήματος. Αυτό, συνεπάγεται περισσότερες κορυφές και κατ’ επέκταση περισσότερα fingerprints, άρα και μεγαλύτερος χρόνος αντιστοίχισης. Με βάση τα παραπάνω αναμένεται να υπάρχει κάποια σχετική αναλογία μεταξύ του χρόνου εγγραφής και του χρόνου αντιστοίχισης.

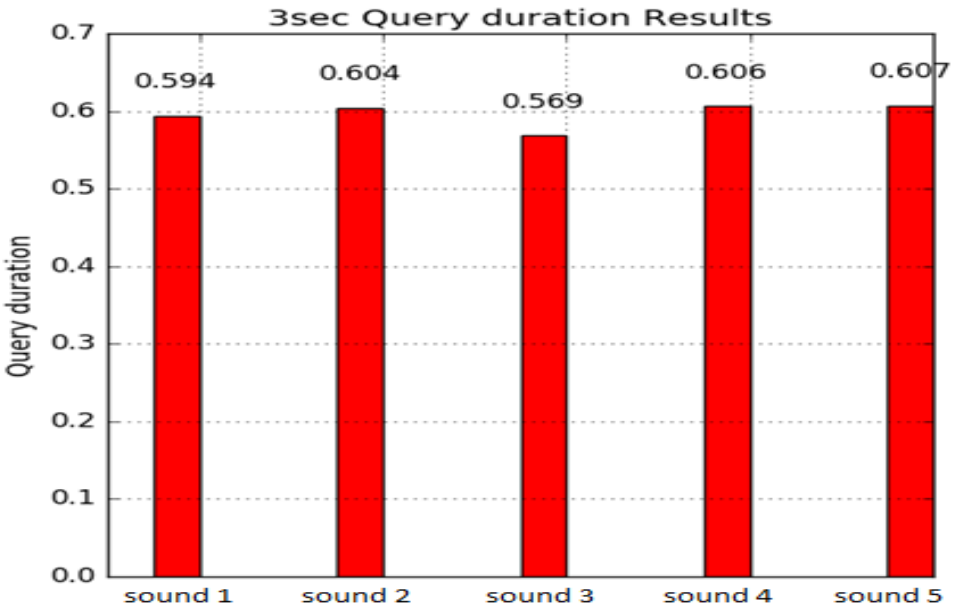
Παρακάτω φαίνονται τα αποτελέσματα της ταχύτητας αντιστοίχισης για χρονική διάρκεια εγγραφής 1, 2, 3, 4 και 5 δευτερολέπτων αντίστοιχα :



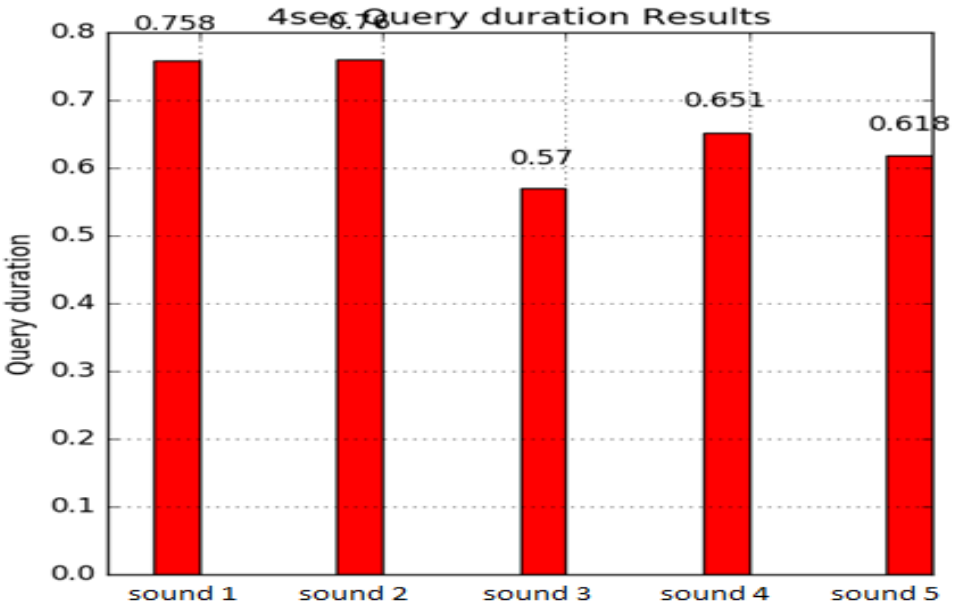
Εικόνα 11*: Αποτελέσματα αντιστοίχισης στο πρώτο δευτερόλεπτο*



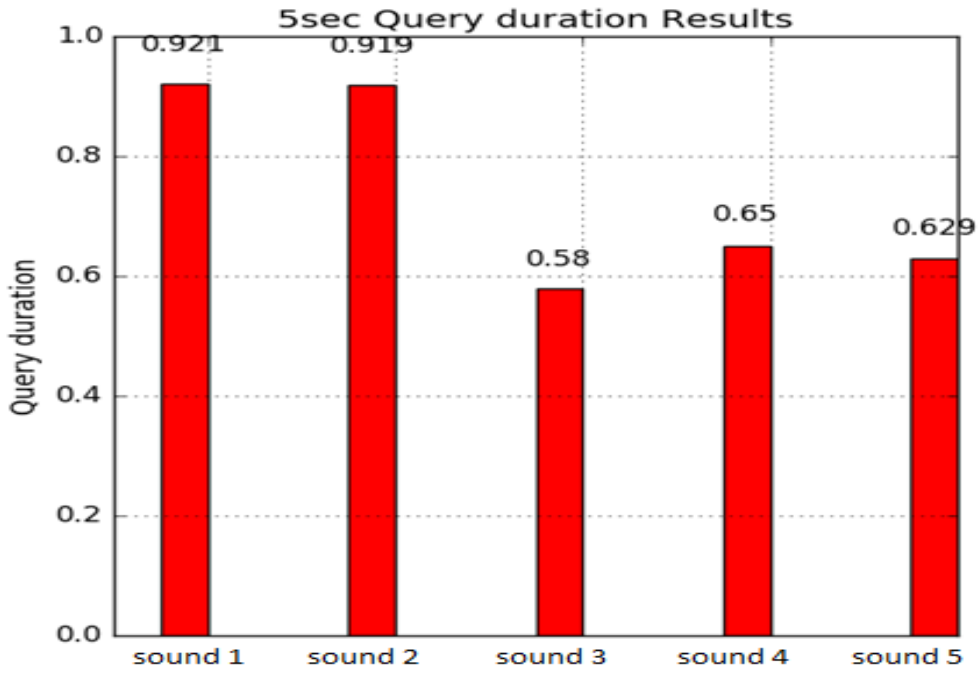
Εικόνα 12: *Αποτελέσματα αντιστοίχισης στα 2 δευτερόλεπτα*



Εικόνα 13: *Αποτελέσματα αντιστοίχισης στα 3 δευτερόλεπτα*



Εικόνα 14: *Αποτελέσματα αντιστοίχισης στα 4 δευτερόλεπτα*



Εικόνα 15: *Αποτελέσματα αντιστοίχισης στα 5 δευτερόλεπτα*

Παρατηρώντας τα διαγράμματα, γίνεται αντιληπτό ότι ο χρόνος που χρειάζεται για να γίνει η αντιστοίχιση προσεγγίζει κατά μέσο όρο το ένα πέμπτο (1/5) του χρόνου εγγραφής.

Στα διαγράμματα που φαίνεται να απουσιάζει ο χρόνος αντιστοίχισης είναι γιατί το σύστημα δεν κατάφερε να πραγματοποιήσει αντιστοίχιση.

Επίσης, φαίνεται κάποια αποτελέσματα να αποκλίνουν από αυτό το μέσο όρο. Στην πραγματικότητα, αυτό οφείλεται στη μικρή διάρκεια (περίπου 3 δευτερόλεπτα) των συγκεκριμένων ήχων (sound 3, sound 4, sound 5), για αυτό και παραμένει σχεδόν ίδιος ο χρόνος εξυπηρέτησης του query από τα 3 δευτερόλεπτα και μετά.

Θεωρώντας ότι ο συνολικός χρόνος αναζήτησης, είναι ο χρόνος από τη στιγμή που θα ξεκινήσει η εγγραφή μέχρι και τη στιγμή που θα λάβουμε την τελική απάντηση από το σύστημα και αφού στο σύστημά η διαδικασία αυτή υλοποιείται σε ένα μόνο νήμα (thread), ο χρόνος αναζήτησης είναι το άθροισμα του χρόνου αντιστοίχισης και του χρόνου εγγραφής.

1 (εγγραφή) + 1/5 (αντιστοίχιση) = 6/5 * χρόνο εγγραφής = 1,2*χρόνο εγγραφής

Ένας σημαντικός περιορισμός είναι φυσικά το round trip time (RTT) για την πραγματοποίηση αντιστοίχισης. Δεδομένου ότι η βάση βρίσκεται τοπικά στα πειράματά μας, δεν υπάρχει επιπλέον καθυστέρηση για τη μεταφορά των δεδομένων. Αυτό θα αύξανε το RTT στο συνολικό υπολογισμό, αλλά δεν θα επηρέαζε τη διαδικασία αντιστοίχισης.

4.2 ΑΠΟΔΟΣΗ ΚΑΙ ΑΞΙΟΠΙΣΤΙΑ

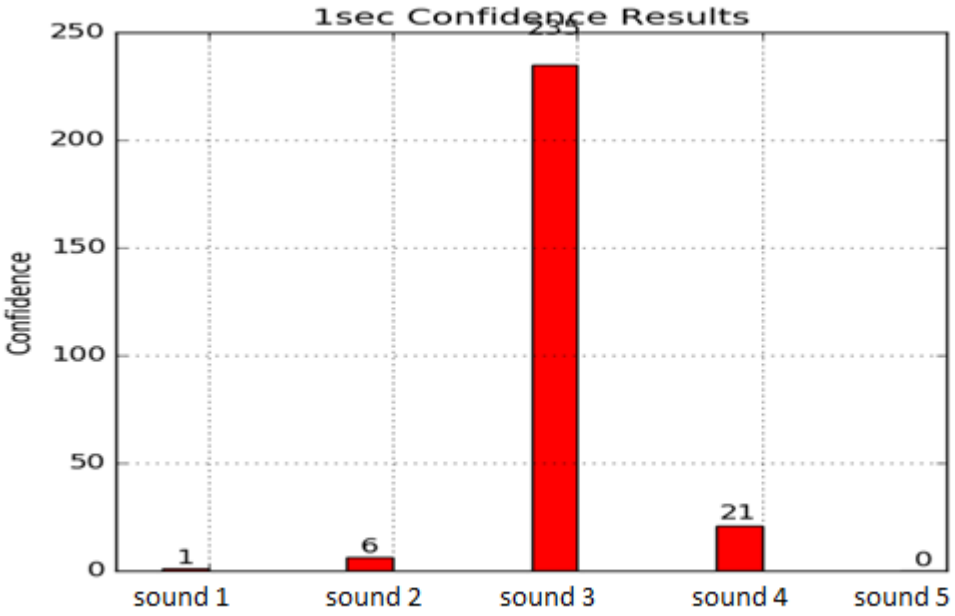
Για την αξιολόγηση της αξιοπιστίας και της απόδοσης του συστήματος (Confidence Results, Matching Percentage) πραγματοποιήθηκε μία σειρά προσομοιώσεων σε δύο φάσεις :

Φάση 1

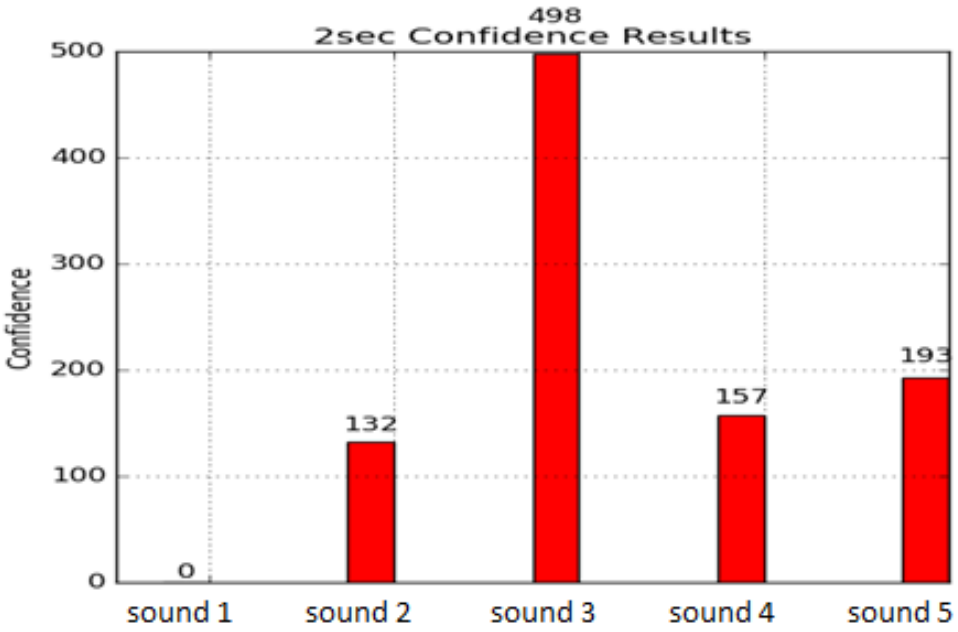
Στην πρώτη φάση, επιλέχθηκαν τυχαία 5 ήχοι (διάρκειας μεταξύ 2 και 8 δευτερολέπτων). Η βάση δεδομένων που χρησιμοποιήθηκε περιείχε reference audio fingerprints από 94 συνολικά ήχους (διάρκειας 1 έως 16 δευτερολέπτων), μεταξύ των οποίων ήταν και οι 5 που επιλέχθηκαν.

Σε αυτή τη φάση μετρήθηκε η αξιοπιστία του συστήματος σε σχέση με το χρόνο εγγραφής. Ουσιαστικά, η μέτρηση αυτή είναι το πλήθος των fingerprints που το σύστημα κατάφερε να αντιστοιχίσει για κάθε έναν από τους 5 αυτούς ήχους για τον εκάστοτε χρόνο εγγραφής.

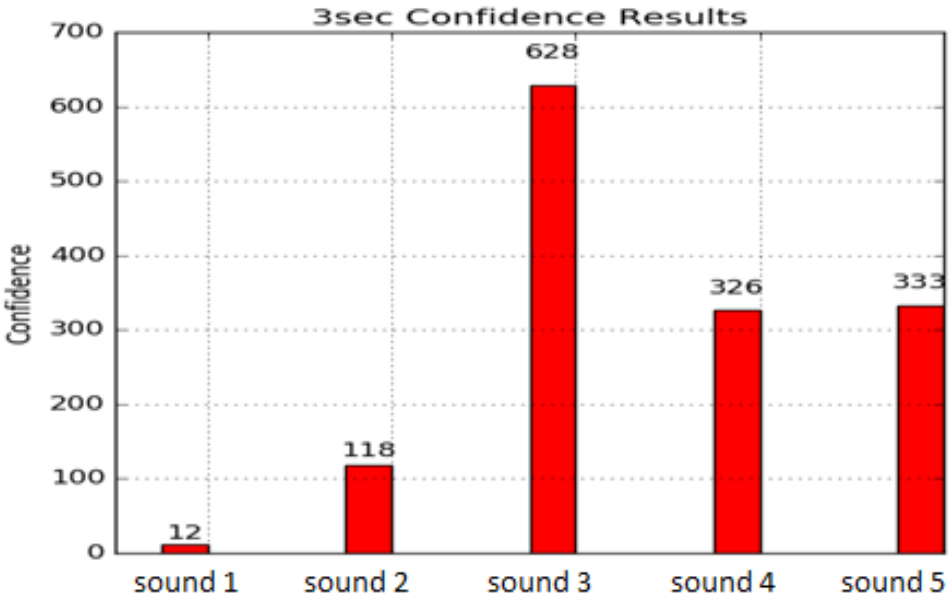
Τα αποτελέσματα αυτά φαίνονται στα επόμενα διαγράμματα :



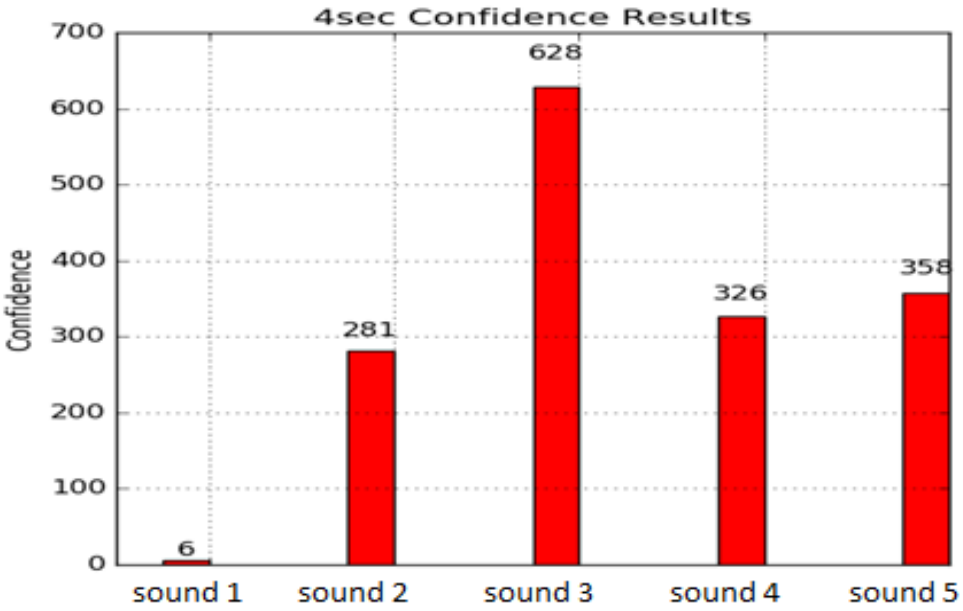
Εικόνα 16: *Αποτελέσματα αξιοπιστίας στο πρώτο δευτερόλεπτο*



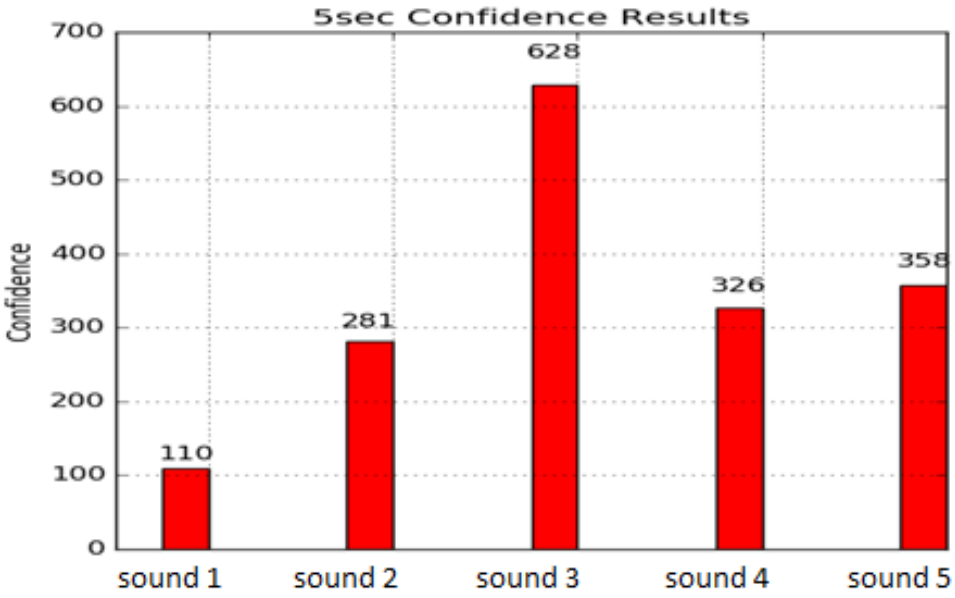
Εικόνα 17: *Αποτελέσματα αξιοπιστίας στα 2 δευτερόλεπτα*



Εικόνα 18: *Αποτελέσματα αξιοπιστίας στα 3 δευτερόλεπτα*



Εικόνα 19: *Αποτελέσματα αξιοπιστίας στα 4 δευτερόλεπτα*



Εικόνα 20: *Αποτελέσματα αξιοπιστίας στα 5 δευτερόλεπτα*

Εύκολα γίνεται αντιληπτό, παρατηρώντας τα διαγράμματα, ότι όσο αυξάνεται ο χρόνος εγγραφής, τόσο περισσότερα είναι και τα αποτυπώματα που επιτυγχάνει να αντιστοιχίσει το σύστημα στον κάθε ήχο. Συνεπώς, τόσο καλύτερες είναι και οι πιθανότητες να αναγνωριστεί σωστά ένα ηχητικό απόσπασμα.

Φάση 2

Στη δεύτερη φάση, μετρήθηκε η συνολική απόδοση του συστήματος και για τους 94 ήχους που βρίσκονται στη βάση δεδομένων. Πραγματοποιήθηκαν δύο ξεχωριστά πειράματα :

1. Ανάγνωση από το δίσκο (mp3 -> wav δεδομένα)
2. Αναπαραγωγή του ήχου από τα ηχεία με το Dejavu να ακούει στο μικρόφωνο του φορητού υπολογιστή.

Τα αποτελέσματα φαίνονται παρακάτω :

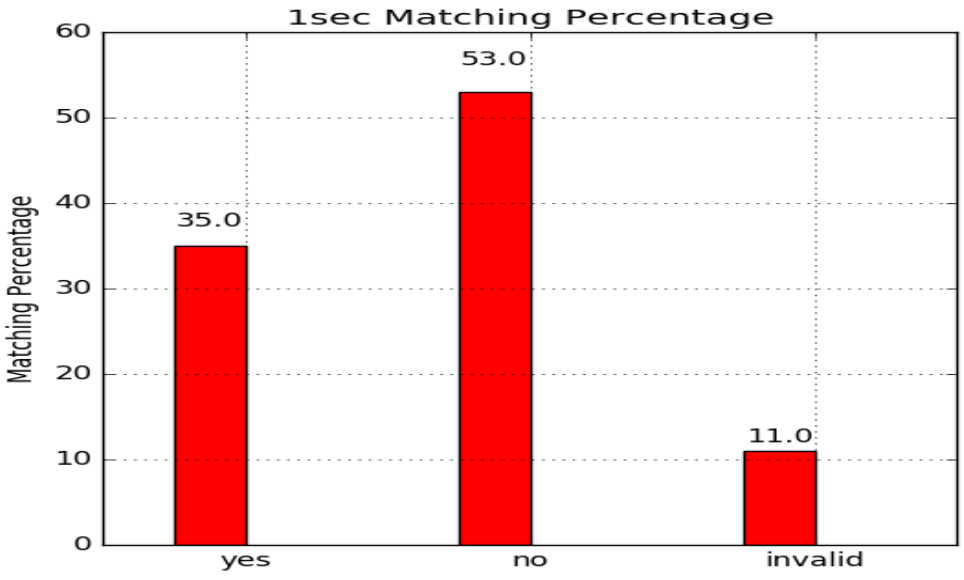
1. Ανάγνωση από το δίσκο

Σε αυτό το πείραμα, δεδομένου ότι το Dejavu πήρε όλα τα δείγματα του ήχου απευθείας από το δίσκο χωρίς θόρυβο, το ποσοστό επιτυχίας ήταν 100%. Δεν έγιναν λάθη σε κανέναν από τους 94 ήχους των οποίων δημιουργήθηκαν τα fingerprints.

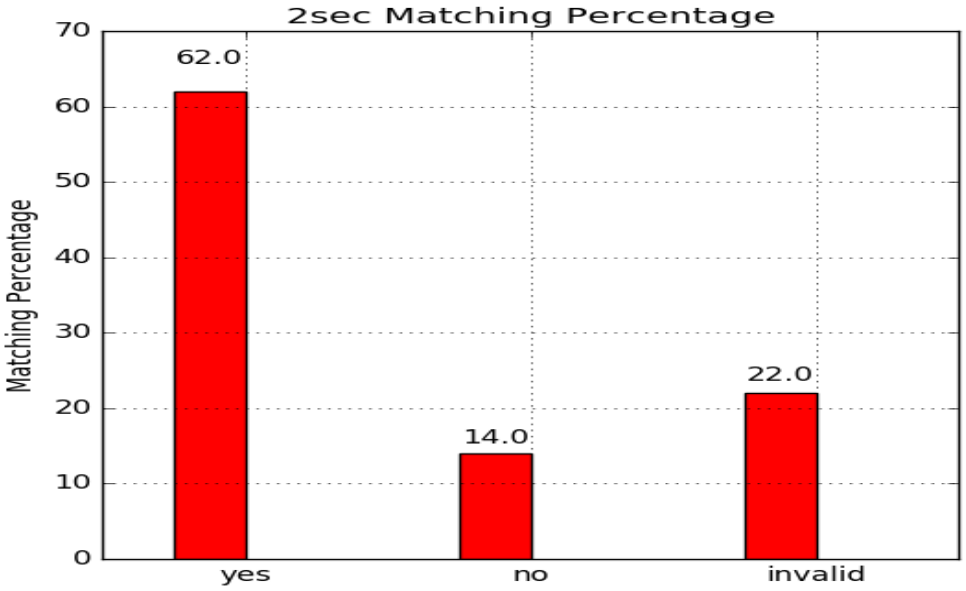
2. Ανάγνωση από το μικρόφωνο

Εδώ χρησιμοποιήθηκε ένα script που επιλέγει τυχαία n δευτερόλεπτα ήχου από το αρχικό αρχείο mp3 για αναπαραγωγή από τα ηχεία, και το Dejavu ακούει από το μικρόφωνο. Επιπλέον, προστέθηκε κάποιος θόρυβος.

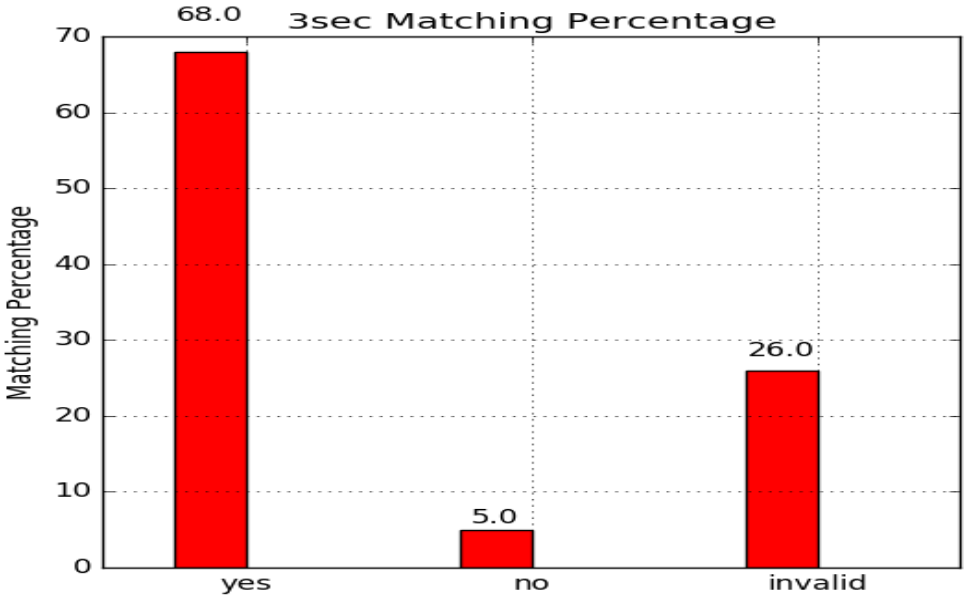
Ακολουθούν τα αποτελέσματα για διαφορετικές τιμές χρόνου ακρόασης (n):



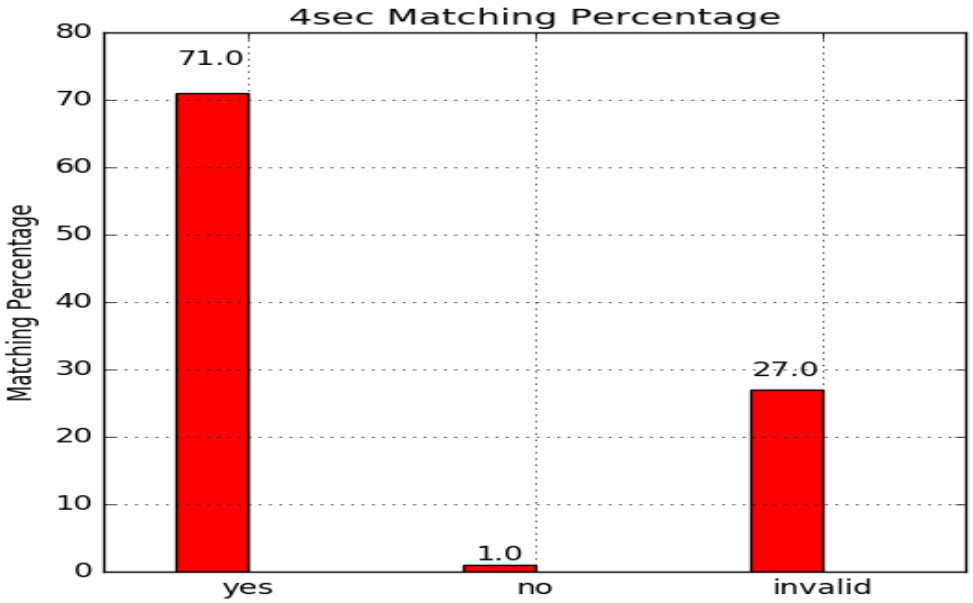
Εικόνα 21: *Συνολική απόδοση του συστήματος στο πρώτο δευτερόλεπτο*



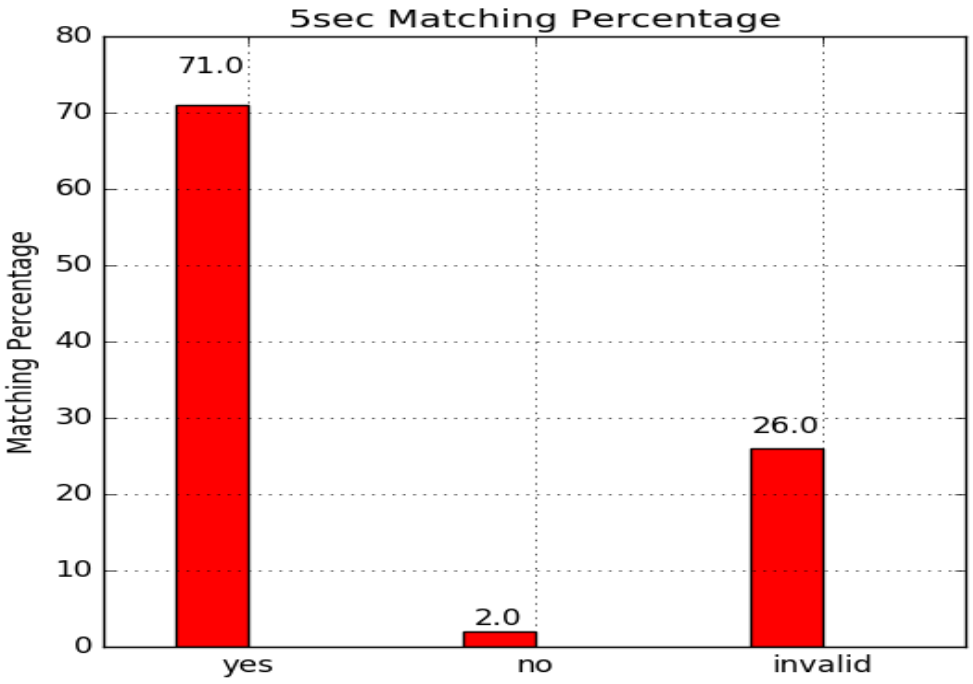
Εικόνα 22: *Συνολική απόδοση του συστήματος στα 2 δευτερόλεπτα*



Εικόνα 23: *Συνολική απόδοση του συστήματος στα 3 δευτερόλεπτα*



Εικόνα 24: *Συνολική απόδοση του συστήματος στα 4 δευτερόλεπτα*



Εικόνα 25: *Συνολική απόδοση του συστήματος στα 5 δευτερόλεπτα*

Και παρακάτω φαίνονται τα αντίστοιχα ποσοστά:

Πίνακας 2: *Ποσοστό Συνολικής Απόδοσης του Συστήματος*

Αριθμός Δευτερολέπτων	**Αριθμός Σωστών**	**Ποσοστό Ακρίβειας**
1	32 / 94	35.0%
2	58 / 94	62.0%
3	63 / 94	68.0%
4	66 / 94	71.0%
5	66 / 94	71.0%

Όπως φαίνεται από τα αποτελέσματα, με ένα μόνο δευτερόλεπτο, τυχαία επιλεγμένο από οπουδήποτε στον ήχο, το Dejavu βρίσκει σωστά το 35%, ενώ στα 2 δευτερόλεπτα, το ποσοστό βελτιώνεται σημαντικά επιτυγχάνοντας ακρίβεια 62%. Παρόλα αυτά, από τα 3 δευτερόλεπτα και μέχρι τα 5 το ποσοστό αυτό δεν έχει βελτιωθεί αρκετά. Συγκεκριμένα έχει φτάσει στο 71%. Ακόμη και μετά τα 5 δευτερόλεπτα εγγραφής το ποσοστό αυτό παραμένει στα ίδια επίπεδα, γεγονός αναμενόμενο αφού ο μέσος όρος διάρκειας των ήχων που βρίσκονται στη βάση δεδομένων είναι τα 5 δευτερόλεπτα.

Τέλος, σημαντική είναι η παρατήρηση ότι εμφανίζεται μεγάλο ποσοστό invalid στην αντιστοίχιση. Περισσότερα για αυτό αναλύονται στο επόμενο κεφάλαιο.

4.3 ΑΠΟΔΟΣΗ ΚΑΙ ΑΠΟΘΗΚΕΥΣΗ

Για τους 94 ήχους που δημιουργήθηκαν fingerprints στη βάση δεδομένων χρησιμοποιήθηκαν 37,7 MB χώρου. Παρακάτω φαίνεται η χρήση του δίσκου συγκριτικά για διαφορετικούς τύπους ηχητικής πληροφορίας :

Πίνακας 3: *Ενδεικτικά αποτελέσματα της χρήσης του δίσκου*

Τύπος ηχητικής πληροφορίας	**Αποθηκευτικός Χώρος σε MB**
mp3	33.9
Wav	188.5
Fingerprints	37,7

Υπάρχει μια αρκετά άμεση ανταλλαγή μεταξύ του απαιτούμενου χρόνου εγγραφής και του απαιτούμενου αποθηκευτικού χώρου.

Από τον πίνακα φαίνεται ότι τα audio fingerprints καταλαμβάνουν πολύ μεγάλο χώρο (λίγο περισσότερο από τα αρχεία MP3). Αυτό, ωστόσο δεν είναι ανησυχητικό αφού υπάρχουν δεκάδες και μερικές φορές εκατοντάδες χιλιάδες hashes ανά ήχο. Ουσιαστικά ανταλλάχθηκε η καθαρή πληροφορία ολόκληρου του ακουστικού σήματος στα αρχεία .wav, για το 20% περίπου του χώρου αποθήκευσης αυτής σε μορφή fingerprint. Με τον τρόπο αυτό επιτυγχάνονται αξιόπιστα αποτελέσματα σε μικρό χρονικό διάστημα. Παρά το γεγονός ότι το ποσοστό επιτυχίας αντιστοίχισης κυμαίνεται περίπου στο 75%, δεν παύει να αποτελεί το μέγιστο για τα πειράματα μας, και επιτυγχάνεται ήδη από το 4 δευτερόλεπτο εγγραφής. Επομένως, το tradeoff χώρος / ταχύτητα φαίνεται να αποδίδει.

5 ΠΡΟΒΛΗΜΑΤΑ ΚΑΙ ΠΕΡΙΟΡΙΣΜΟΙ ΣΧΕΤΙΚΑ ΜΕ ΤΗΝ ΥΛΟΠΟΙΗΣΗ

Στην παρούσα διπλωματική εργασία προσομοιώθηκε ένα σύστημα, το οποίο κάνει χρήση των audio fingerprints για την αναγνώριση ήχων καρδίας και πνεύμονα. Η υλοποίηση βασίστηκε στο ελεύθερου λογισμικού σύστημα : Dejavu. Αφού περιγράφηκε το σύστημα, στη συνέχεια πραγματοποιήθηκαν πειράματα και προσομοιώσεις και παρουσιάστηκαν τα αποτελέσματα τους. Στο κεφάλαιο αυτό παρουσιάζονται και αναλύονται τα προβλήματα και οι περιορισμοί που προέκυψαν βάση των αποτελεσμάτων αυτών. Παράλληλα, καταγράφονται κάποιες προτάσεις και ιδέες για βελτιώσεις και μελλοντική έρευνα. Τέλος, επισημαίνονται κάποιοι πολύ σημαντικοί παράγοντες που επηρεάζουν, δυσχεραίνουν και πιθανώς καθιστούν ακατάλληλη τη συγκεκριμένη προσέγγιση, για την αναγνώριση βιοϊατρικών σημάτων.

5.1 ΧΡΟΝΙΚΗ ΠΟΛΥΠΛΟΚΟΤΗΤΑ

Σε γενικές γραμμές, η χρονική απόδοση του συστήματος είναι αρκετά καλή. Η βάση δεδομένων, στην οποία πραγματοποιείται η αναζήτηση, βρίσκεται τοπικά στον υπολογιστή και έτσι δεν έχουμε επιπλέον επιβάρυνση για τη μεταφορά των δεδομένων.

Επιπλέον, ο χρόνος που χρειάζεται για να γίνει η αντιστοίχιση είναι τις τάξεως των millisecond. Προσεγγίζει, όπως φάνηκε από τα αποτελέσματα, το ένα πέμπτο (1/5) του χρόνου εγγραφής. Παρά το γεγονός ότι ο χρόνος αυτός είναι πολύ καλός, υπάρχει περιθώριο βελτίωσης. Βασική αιτία επιβάρυνσης του χρόνου αντιστοίχισης είναι η εύρεση των κορυφών στο φασματογράφημα. Μία διαδικασία που απαιτεί συγκεκριμένο χρόνο για να ολοκληρωθεί και δεν είναι εύκολο ο χρόνος αυτός να αλλάξει δραματικά.

Ωστόσο, όπως ειπώθηκε σε προηγούμενο κεφάλαιο ο συνολικός χρόνος αναζήτησης υπολογίζεται ως άθροισμα του χρόνου αντιστοίχισης και του χρόνου εγγραφής και είναι μία μονονηματική διαδικασία. Αυτό, για το συνολικό χρόνο αναζήτησης, σημαίνει ότι θα είναι το άθροισμα του χρόνου που χρειάζονται τα επιμέρους στάδια της διαδικασίας για να ολοκληρωθούν.

Κάποιες προτάσεις για την περεταίρω βελτίωση της χρονικής πολυπλοκότητας του αλγορίθμου θα μπορούσε να είναι ο πειραματισμός με multithreading και ο πειραματισμός με αντιστοίχιση (matching) σε πραγματικό χρόνο.

5.2 ΠΟΣΟΣΤΟ "ΑΔΥΝΑΜΙΑΣ" ΑΝΤΙΣΤΟΙΧΙΣΗΣ (INVALID MATCHING)

Στα τελευταία πειράματα παρατηρήθηκε η εμφάνιση μεγάλου ποσοστού invalid matches.

Ο λόγος είναι ότι δεν καλύπτεται ικανοποιητικά το φάσμα των συχνοτήτων, δηλαδή :

Τα invalid matches αφορούν ήχους , που το σήμα που πάρθηκε κατά την εγγραφή, ήταν πολύ ασθενές και το σύστημα δεν κατάφερε να βγάλει ένα αρκετά "γεμάτο" φασματογράφημα. Κατ' επέκταση, δεν μπόρεσαν να εντοπιστούν κορυφές ώστε να εξαχθούν τα fingerprints για να πραγματοποιηθεί η αναζήτηση στη βάση δεδομένων.

Ασθενές σήμα μπορεί να σημαίνει, ότι κατά την εγγραφή του ήχου πετύχαμε απόσπασμα που περιείχε μεγάλο "κομμάτι" σιωπής ή απόσπασμα πολύ χαμηλής έντασης που δεν "γέμιζε" το φασματογράφημα.

5.3 ΤΑΧΥΤΗΤΑ ΕΓΓΡΑΦΗΣ ΚΑΙ ΑΝΤΙΣΤΟΙΧΙΣΗ

Η αντιστοίχιση των fingerprints βασίζεται στα offset, τα σχετικά (offset δείγματος από την εγγραφή) και τα απόλυτα (offset αρχικού ήχου στη βάση δεδομένων). Για την ακρίβεια, όπως εξηγήθηκε στο κεφάλαιο 3, για κάθε αντιστοίχιση υπολογίζεται η διαφορά μεταξύ των offset (απόλυτο offset – σχετικό offset). Η αντιστοίχιση εξαρτάται άμεσα από τη ροή του χρόνου τόσο του δείγματος όσο και του καταγεγραμμένου ήχου.

Οπότε αν ο ήχος που έχει αποθηκευτεί στη βάση έχει άλλη ταχύτητα από αυτή του ήχου που ακούμε είναι αδύνατο να κάνουμε αντιστοίχιση (match).

Στην πραγματικότητα, όμως το προηγούμενο πρόβλημα δεν προλαβαίνει να μας απασχολήσει, διότι όταν ο ήχος αναπαράγεται σε άλλη ταχύτητα τότε έχει και άλλη συχνότητα. Συνεπώς θα προκύψει τελείως διαφορετικό φασματογράφημα από το αρχικό. Σαν αποτέλεσμα θα προκύψουν άλλες κορυφές, που θα έχουν άλλες αποκλείσεις μεταξύ τους και επομένως θα έχουμε τελείως διαφορετικά ζεύγη (χρόνου, συχνότητας) οπότε και τελείως διαφορετικά fingerprints. Αυτό σημαίνει ότι δεν μπορεί να γίνει η αντιστοίχιση. Επομένως, γίνεται αντιληπτό ότι το ίδιο ισχύει σε κάθε περίπτωση που απλά έχουμε διαφορετική συχνότητα ακόμη και αν η ταχύτητα είναι η ίδια.

Όλα τα παραπάνω γεννούν έναν πολύ σοβαρό προβληματισμό : πώς εξασφαλίζεται η συνέπεια στην ταχύτητα και τη συχνότητα αναπαραγωγής όταν οι ήχοι που μας ενδιαφέρουν είναι βιοϊατρικά σήματα ;

Ως βιοϊατρικό σήμα, ορίζονται οι διακυμάνσεις φυσικών μεγεθών στον χρόνο, οι οποίες συμβαίνουν στα όργανα του ανθρώπινου σώματος. Εύκολα αντιλαμβανόμαστε πόσο μπορούν να διαφέρουν οι διακυμάνσεις αυτές σε κάθε άνθρωπο και πόσοι πολλοί και διαφορετικοί παράγοντες τις επηρεάζουν ακόμη και όταν πρόκειται για τον ίδιο κάθε φορά οργανισμό. Παράγοντες όπως : εκ γενετής δυσπλασίες στη δομή των οργάνων (καρδία, πνεύμονες, κ.α.), η φυσιολογία των οργάνων, το ανθρώπινο σώμα στο σύνολό του και οι ιδιαιτερότητες που το χαρακτηρίζουν επηρεάζουν τα βιοσήματα. Παράγοντες όπως η ηλικία, το βάρος, το ύψος, η σωματική δραστηριότητα, η διατροφή, το κάπνισμα, η κατανάλωση αλκοόλ και τόσα άλλα μπορούν επίσης να επηρεάσουν τα βιοσήματα. Η κληρονομικότητα, οι συνθήκες διαβίωσης, το ιστορικό ασθενείας είναι ακόμη μερικοί λόγοι που μπορούν να μεταβάλλουν τις διακυμάνσεις αυτές στο ανθρώπινο σώμα, καθιστώντας θεωρητικά άπειρες τις διαφορετικές τιμές συχνότητας και ταχύτητας που μπορούμε να λάβουμε.

Ταυτόχρονα, η ακριβής και έγκαιρη διάγνωση είναι οι βασικοί πυλώνες για την υψηλή ποιότητα περίθαλψης των ασθενών. Η σωστή διάγνωση προστατεύει τη ζωή του ασθενή και εξασφαλίζει την υγεία του. Η διαδικασία της διάγνωσης είναι μια πολύπλοκη, επικεντρωμένη στον ασθενή, δραστηριότητα και τα διαγνωστικά σφάλματα επιμένουν σε όλες τα στάδια της περίθαλψης συνεχίζοντας να βλάπτουν έναν μεγάλο αριθμό ασθενών. Αντιλαμβανόμαστε, λοιπόν, πόσο σημαντική είναι η ορθή διάγνωση και πόσο δύσκολη και λεπτή διαδικασία η εξαγωγή των αποτελεσμάτων που την επηρεάζουν.

Καταλήγοντας, λοιπόν, στον αρχικό μας προβληματισμό και συνδυάζοντας όλες τις προηγούμενες πληροφορίες, οφείλει να γίνει πολύ προσεχτική μελέτη πριν ένα εργαλείο ή μία νέα τεχνολογία εισαχθεί επίσημα στην ιατρική επιστήμη και παίξει ενεργό ρόλο στη διαγνωστική διαδικασία. Υπάρχουν σημαντικοί παράγοντες που πρέπει να επαναπροσδιοριστούν, ώστε να καταστεί η ιδέα της συγκεκριμένης διπλωματικής εργασίας εφαρμόσιμη. Ωστόσο, η παρούσα υλοποίηση θέτει τα θεμέλια, ανοίγοντας ένα μελλοντικό πεδίο μελέτης και έρευνας. Η τεχνολογία Audio Fingerprinting για την αναγνώριση ηχητικών αντικειμένων, σε συνδυασμό με τη μηχανική μάθηση, τα νευρωνικά δίκτυα και πληθώρα άλλων καινοτόμων τεχνολογιών ίσως μας δώσει σύντομα τη λύση για την έγκυρη και γρήγορη διάγνωση παθήσεων που αποτελούν μάστιγα της εποχής.

6 Βιβλιογραφια

[1] ‘WHO | World Health Organization’. [Online]. Available: https://www.who.int/. [Accessed: 06-Nov-2019].

[2] ‘Heart disease - Symptoms and causes’, *Mayo Clinic*. [Online]. Available: https://www.mayoclinic.org/diseases-conditions/heart-disease/symptoms-causes/syc-20353118. [Accessed: 04-Nov-2019].

[3] ‘Electrocardiography - Wikipedia’. [Online]. Available: https://en.wikipedia.org/wiki/Electrocardiography. [Accessed: 06-Nov-2019].

[4] M. K. Diab, ‘Systems and methods for determining blood oxygen saturation values using complex number encoding’, US9622693B2, 18-Apr-2017.

[5] W. M. Smith, F. Riddell, M. Madon, and M. J. Gleva, ‘Comparison of diagnostic value using a small, single channel, P-wave centric sternal ECG monitoring patch with a standard 3-lead Holter system over 24 hours’, *Am. Heart J.*, vol. 185, pp. 67–73, Mar. 2017.

[6] V. X. Afonso, W. J. Tompkins, T. Q. Nguyen, K. Michler, and Shen Luo, ‘Comparing stress ECG enhancement algorithms’, *IEEE Eng. Med. Biol. Mag.*, vol. 15, no. 3, pp. 37–44, Jun. 1996.

[7] ‘Cardiovascular Disorders’, *MSD Manual Professional Edition*. [Online]. Available: https://www.msdmanuals.com/professional/cardiovascular-disorders. [Accessed: 05-Nov-2019].

[8] Z. Arabasadi, R. Alizadehsani, M. Roshanzamir, H. Moosaei, and A. A. Yarifard, ‘Computer aided decision making for heart disease detection using hybrid neural network-Genetic algorithm’, *Comput. Methods Programs Biomed.*, vol. 141, pp. 19–26, Apr. 2017.

[9] M. Gandhi and S. N. Singh, ‘Predictions in heart disease using techniques of data mining’, in *2015 International Conference on Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE)*, Greater Noida, India, 2015, pp. 520–525.

[10] Forum of International Respiratory Societies and European Respiratory Society, *The global impact of respiratory disease*. 2017.

[11] T. Eaton, S. Withy, J. E. Garrett, R. M. L. Whitlock, H. H. Rea, and J. Mercer, ‘Spirometry in Primary Care Practice’, *Chest*, vol. 116, no. 2, pp. 416–423, Aug. 1999.

[12] A. Ernst and F. J. F. Herth, *Introduction to Bronchoscopy*. Cambridge University Press, 2017.

[13] ‘Pulmonary Disorders’, *MSD Manual Professional Edition*. [Online]. Available: https://www.msdmanuals.com/professional/pulmonary-disorders. [Accessed: 05-Nov-2019].

[14] A. Bobbio *et al.*, ‘Changes in pulmonary function test and cardio-pulmonary exercise capacity in COPD patients after lobar pulmonary resection’, *Eur. J. Cardiothorac. Surg.*, vol. 28, no. 5, pp. 754–758, Nov. 2005.

[15] R. Palaniappan, K. Sundaraj, and N. U. Ahamed, ‘Machine learning in lung sound analysis: A systematic review’, *Biocybern. Biomed. Eng.*, vol. 33, no. 3, pp. 129–135, Jan. 2013.

[16] ‘Heart and Lung Sounds Reference Guide and Library’, *Easy Auscultation*. [Online]. Available: https://www.easyauscultation.com/heart-lung-sounds-reference-guide. [Accessed: 09-Nov-2019].

[17] ‘Lung Sounds | Over 50 Lessons, Reference Guides and Quiz’, *Easy Auscultation*. [Online]. Available: https://www.easyauscultation.com/lung-sounds. [Accessed: 09-Apr-2019].

[18] P. Cano, E. Batlle, T. Kalker, and J. Haitsma, ‘A Review of Audio Fingerprinting’, *J. VLSI Signal Process. Syst. Signal Image Video Technol.*, vol. 41, no. 3, pp. 271–284, Nov. 2005.

[19] P. Cano, E. Batle, T. Kalker, and J. Haitsma, ‘A Review of Algorithms for Audio Fingerprinting’, in *2002 IEEE Workshop on Multimedia Signal Processing.*, St.Thomas, VI, USA, 2002, pp. 169–173.

[20] P. J. O. Doets and R. L. Lagendijk, ‘Extracting quality parameters for compressed audio from fingerprints’, in *in 6th International Conference on Music Information Retrieval (ISMIR*, 2005, pp. 498–503.

[21] ‘(PDF) Robust sound modeling for song detection in broadcast audio | Harald Mayer - Academia.edu’. [Online]. Available: https://www.academia.edu/17450889/Robust_sound_modeling_for_song_detection_in_broadcast_audio. [Accessed: 03-Nov-2019].

[22] N. Memon, B. Sankur, and H. Özer, ‘ROBUST AUDIO HASHING FOR CONTENT IDENTIFICATION’, p. 4.

[23] P. Cano Vila, Universitat Pompeu Fabra, and X. Serra, *Content-based audio search: from fingerprinting to semantic audio retrieval*. Barcelona: Universitat Pompeu Fabra, 2007.

[24] 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.

[25] E. Batlle, J. Masip, E. Guaus, P. Circumvalacio, and B. Catalunya-Spain, ‘Automatic Song Identification in Noisy Broadcast Audio’, p. 7.

[26] M. Audio, E. Wold, T. Blum, D. Keislar, and J. Wheaton, *Content-Based Classification, Search, and Retrieval of Audio*. .

[27] V. Venkatachalam, L. Cazzanti, N. Dhillon, and M. Wells, ‘Automatic identification of sound recordings’, *IEEE Signal Process. Mag.*, vol. 21, no. 2, pp. 92–99, Mar. 2004.

[28] U. P. Fabra, P. Circumvalacio, and B. Catalunya-Spain, ‘Eloi Batlle, Jaume Masip, Pedro Cano’, p. 4, 2003.

[29] 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.

[30] E. Allamanche, ‘Content-based Identification of Audio Material Using MPEG-7 Low Level Description’, in *ISMIR*, 2001.

[31] J. Haitsma and T. Kalker, ‘A Highly Robust Audio Fingerprinting System’, p. 9.

[32] 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.

[33] 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.

[34] J. Goldstein, J. C. Platt, and C. J. C. Burges, ‘Indexing High Dimensional Rectangles for Fast Multimedia Identification’, 2003.

[35] C. Yang, ‘Music Database Retrieval Based on Spectral Similarity’, p. 9.

[36] M. L. Miller, M. A. Rodriguez, and I. J. Cox, ‘Audio Fingerprinting: Nearest Neighbor Search in High Dimensional Binary Spaces’, *J. VLSI Signal Process. Syst. Signal Image Video Technol.*, vol. 41, no. 3, pp. 285–291, Nov. 2005.

[37] *Applying Steganography to Standard Network Traffic Paul Miller*. 2004.

[38] A. L.-C. Wang, ‘An Industrial-Strength Audio Search Algorithm’, p. 7.

[39] J. Hunter, ‘Adding Multimedia to the Semantic Web - Building an MPEG-7 Ontology’, p. 24.

[40] ‘Sales Inquiry AllMediaGuide.com | Uniregistry’, *Uniregistry.com*. [Online]. Available: https://uniregistry.com/market/domain/allmediaguide.com?landerid=www5dc5cf65118bf9.87655516. [Accessed: 09-Nov-2019].

[41] J. S. Seo, J. Haitsma, and T. Kalker, ‘KAIST, Dept. of EECS 373-1 Guseong Dong, Yuseong Gu, Daejeon 305-701, Korea’, p. 4, 2002.

[42] P. J. O. Doets and R. L. Lagendijk, ‘STOCHASTIC MODEL OF A ROBUST AUDIO FINGERPRINTING SYSTEM’, p. 4.

[43] ‘worldveil/dejavu’, *GitHub*. [Online]. Available: https://github.com/worldveil/dejavu. [Accessed: 09-Nov-2019].

7 ΠΑΡΑΡΤΗΜΑ

7.1 ΑΛΓΟΡΙΘΜΟΙ ΣΥΣΤΗΜΑΤΟΣ ΑΝΑΓΝΩΡΙΣΗΣ ΉΧΩΝ

Στη συνέχεια παρατίθενται οι αλγόριθμοι που υλοποιούν τα βασικά μέρη του συστήματος Audio Fingerprinting Dejavu σε κώδικα Python.

"_init_py."

```
from dejavu.database import get_database, Database
import dejavu.decoder as decoder
import fingerprint
import multiprocessing
import os
import traceback
import sys


class Dejavu(object):

    SONG_ID = "song_id"
    SONG_NAME = 'song_name'
    CONFIDENCE = 'confidence'
    MATCH_TIME = 'match_time'
    OFFSET = 'offset'
    OFFSET_SECS = 'offset_seconds'

    def __init__(self, config):
        super(Dejavu, self).__init__()

        self.config = config

        # initialize db
        db_cls = get_database(config.get("database_type", None))

        self.db = db_cls(**config.get("database", {}))
```

```
        self.db.setup()

        # if we should limit seconds fingerprinted,
        # None|-1 means use entire track
        self.limit = self.config.get("fingerprint_limit", None)
        if self.limit == -1:  # for JSON compatibility
            self.limit = None
        self.get_fingerprinted_songs()

    def get_fingerprinted_songs(self):
        # get songs previously indexed
        self.songs = self.db.get_songs()
        self.songhashes_set = set()  # to know which ones we've computed before
        for song in self.songs:
            song_hash = song[Database.FIELD_FILE_SHA1]
            self.songhashes_set.add(song_hash)

    def fingerprint_directory(self, path, extensions, nprocesses=None):
        # Try to use the maximum amount of processes if not given.
        try:
            nprocesses = nprocesses or multiprocessing.cpu_count()
        except NotImplementedError:
            nprocesses = 1
        else:
            nprocesses = 1 if nprocesses <= 0 else nprocesses

        pool = multiprocessing.Pool(nprocesses)

        filenames_to_fingerprint = []
        for filename, _ in decoder.find_files(path, extensions):

            # don't refingerprint already fingerprinted files
            if decoder.unique_hash(filename) in self.songhashes_set:
                print "%s already fingerprinted, continuing..." % filename
                continue

            filenames_to_fingerprint.append(filename)

        # Prepare _fingerprint_worker input
        worker_input = zip(filenames_to_fingerprint,
                           [self.limit] * len(filenames_to_fingerprint))
```

```
    # Send off our tasks
    iterator = pool.imap_unordered(_fingerprint_worker,
                                   worker_input)

    # Loop till we have all of them
    while True:
        try:
            song_name, hashes, file_hash = iterator.next()
        except multiprocessing.TimeoutError:
            continue
        except StopIteration:
            break
        except:
            print("Failed fingerprinting")
            # Print traceback because we can't reraise it here
            traceback.print_exc(file=sys.stdout)
        else:
            sid = self.db.insert_song(song_name, file_hash)

            self.db.insert_hashes(sid, hashes)
            self.db.set_song_fingerprinted(sid)
            self.get_fingerprinted_songs()

    pool.close()
    pool.join()

def fingerprint_file(self, filepath, song_name=None):
    songname = decoder.path_to_songname(filepath)
    song_hash = decoder.unique_hash(filepath)
    song_name = song_name or songname
    # don't refingerprint already fingerprinted files
    if song_hash in self.songhashes_set:
        print "%s already fingerprinted, continuing..." % song_name
    else:
        song_name, hashes, file_hash = _fingerprint_worker(
            filepath,
            self.limit,
            song_name=song_name
        )
        sid = self.db.insert_song(song_name, file_hash)

        self.db.insert_hashes(sid, hashes)
```

```
            self.db.set_song_fingerprinted(sid)
            self.get_fingerprinted_songs()

    def find_matches(self, samples, Fs=fingerprint.DEFAULT_FS):
        hashes = fingerprint.fingerprint(samples, Fs=Fs)
        return self.db.return_matches(hashes)

    def align_matches(self, matches):
        """
            Finds hash matches that align in time with other matches and finds
            consensus about which hashes are "true" signal from the audio.

            Returns a dictionary with match information.
        """
        # align by diffs
        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

        # extract idenfication
        song = self.db.get_song_by_id(song_id)
        if song:
            # TODO: Clarify what `get_song_by_id` should return.
            songname = song.get(Dejavu.SONG_NAME, None)
        else:
            return None

        # return match info
        nseconds = round(float(largest) / fingerprint.DEFAULT_FS *
```

```
                fingerprint.DEFAULT_WINDOW_SIZE *
                fingerprint.DEFAULT_OVERLAP_RATIO, 5)
        song = {
            Dejavu.SONG_ID : song_id,
            Dejavu.SONG_NAME : songname,
            Dejavu.CONFIDENCE : largest_count,
            Dejavu.OFFSET : int(largest),
            Dejavu.OFFSET_SECS : nseconds,
            Database.FIELD_FILE_SHA1 : song.get(Database.FIELD_FILE_SHA1, None),}
        return song

    def recognize(self, recognizer, *options, **kwoptions):
        r = recognizer(self)
        return r.recognize(*options, **kwoptions)


def _fingerprint_worker(filename, limit=None, song_name=None):
    # Pool.imap sends arguments as tuples so we have to unpack
    # them ourself.
    try:
        filename, limit = filename
    except ValueError:
        pass

    songname, extension = os.path.splitext(os.path.basename(filename))
    song_name = song_name or songname
    channels, Fs, file_hash = decoder.read(filename, limit)
    result = set()
    channel_amount = len(channels)

    for channeln, channel in enumerate(channels):
        # TODO: Remove prints or change them into optional logging.
        print("Fingerprinting channel %d/%d for %s" % (channeln + 1,
                                       channel_amount,
                                       filename))
        hashes = fingerprint.fingerprint(channel, Fs=Fs)
        print("Finished channel %d/%d for %s" % (channeln + 1, channel_amount,
                                     filename))
        result |= set(hashes)

    return song_name, result, file_hash
```

```
def chunkify(lst, n):
    """
    Splits a list into roughly n equal parts.
    http://stackoverflow.com/questions/2130016/splitting-a-list-of-arbitrary-size-into-only-roughly-n-equal-parts
    """
    return [lst[i::n] for i in xrange(n)]
```

“database.py”

```
from __future__ import absolute_import
import abc


class Database(object):
    __metaclass__ = abc.ABCMeta

    FIELD_FILE_SHA1 = 'file_sha1'
    FIELD_SONG_ID = 'song_id'
    FIELD_SONGNAME = 'song_name'
    FIELD_OFFSET = 'offset'
    FIELD_HASH = 'hash'

    # Name of your Database subclass, this is used in configuration
    # to refer to your class
    type = None

    def __init__(self):
        super(Database, self).__init__()

    def before_fork(self):
        """
        Called before the database instance is given to the new process
        """
        pass

    def after_fork(self):
        """
        Called after the database instance has been given to the new process
```

```
        This will be called in the new process.
        """
        pass

    def setup(self):
        """
        Called on creation or shortly afterwards.
        """
        pass

    @abc.abstractmethod
    def empty(self):
        """
        Called when the database should be cleared of all data.
        """
        pass

    @abc.abstractmethod
    def delete_unfingerprinted_songs(self):
        """
        Called to remove any song entries that do not have any fingerprints
        associated with them.
        """
        pass

    @abc.abstractmethod
    def get_num_songs(self):
        """
        Returns the amount of songs in the database.
        """
        pass

    @abc.abstractmethod
    def get_num_fingerprints(self):
        """
        Returns the number of fingerprints in the database.
        """
        pass

    @abc.abstractmethod
    def set_song_fingerprinted(self, sid):
```

```
        """
        Sets a specific song as having all fingerprints in the database.

        sid: Song identifier
        """
        pass

    @abc.abstractmethod
    def get_songs(self):
        """
        Returns all fully fingerprinted songs in the database.
        """
        pass

    @abc.abstractmethod
    def get_song_by_id(self, sid):
        """
        Return a song by its identifier

        sid: Song identifier
        """
        pass

    @abc.abstractmethod
    def insert(self, hash, sid, offset):
        """
        Inserts a single fingerprint into the database.

          hash: Part of a sha1 hash, in hexadecimal format
           sid: Song identifier this fingerprint is off
        offset: The offset this hash is from
        """
        pass

    @abc.abstractmethod
    def insert_song(self, song_name):
        """
        Inserts a song name into the database, returns the new
        identifier of the song.

        song_name: The name of the song.
        """
```

```
        pass

    @abc.abstractmethod
    def query(self, hash):
        """
        Returns all matching fingerprint entries associated with
        the given hash as parameter.

        hash: Part of a sha1 hash, in hexadecimal format
        """
        pass

    @abc.abstractmethod
    def get_iterable_kv_pairs(self):
        """
        Returns all fingerprints in the database.
        """
        pass

    @abc.abstractmethod
    def insert_hashes(self, sid, hashes):
        """
        Insert a multitude of fingerprints.

            sid: Song identifier the fingerprints belong to
        hashes: A sequence of tuples in the format (hash, offset)
        -   hash: Part of a sha1 hash, in hexadecimal format
        - offset: Offset this hash was created from/at.
        """
        pass

    @abc.abstractmethod
    def return_matches(self, hashes):
        """
        Searches the database for pairs of (hash, offset) values.

        hashes: A sequence of tuples in the format (hash, offset)
        -   hash: Part of a sha1 hash, in hexadecimal format
        - offset: Offset this hash was created from/at.

        Returns a sequence of (sid, offset_difference) tuples.
```

```
                sid: Song identifier
        offset_difference: (offset - database_offset)
        """
        pass


def get_database(database_type=None):
    # Default to using the mysql database
    database_type = database_type or "mysql"
    # Lower all the input.
    database_type = database_type.lower()

    for db_cls in Database.__subclasses__():
        if db_cls.type == database_type:
            return db_cls

    raise TypeError("Unsupported database type supplied.")


# Import our default database handler
import dejavu.database_sql
```

“database_sql.py”

```
from __future__ import absolute_import
from itertools import izip_longest
import Queue

import MySQLdb as mysql
from MySQLdb.cursors import DictCursor

from dejavu.database import Database


class SQLDatabase(Database):
    """
    Queries:

    1) Find duplicates (shouldn't be any, though):

        select `hash`, `song_id`, `offset`, count(*) cnt
        from fingerprints
```

```
    group by `hash`, `song_id`, `offset`
    having cnt > 1
    order by cnt asc;

2) Get number of hashes by song:

    select song_id, song_name, count(song_id) as num
    from fingerprints
    natural join songs
    group by song_id
    order by count(song_id) desc;

3) get hashes with highest number of collisions

    select
        hash,
        count(distinct song_id) as n
    from fingerprints
    group by `hash`
    order by n DESC;

=> 26 different songs with same fingerprint (392 times):

    select songs.song_name, fingerprints.offset
    from fingerprints natural join songs
    where fingerprints.hash = "08d3c833b71c60a7b620322ac0c0aba7bf5a3e73";
"""

type = "mysql"

# tables
FINGERPRINTS_TABLENAME = "fingerprints"
SONGS_TABLENAME = "songs"

# fields
FIELD_FINGERPRINTED = "fingerprinted"

# creates
CREATE_FINGERPRINTS_TABLE = """
    CREATE TABLE IF NOT EXISTS `%s` (
        `%s` binary(10) not null,
        `%s` mediumint unsigned not null,
```

```
            `%s` int unsigned not null,
        INDEX (%s),
        UNIQUE KEY `unique_constraint` (%s, %s, %s),
        FOREIGN KEY (%s) REFERENCES %s(%s) ON DELETE CASCADE
    ) ENGINE=INNODB;""" % (
        FINGERPRINTS_TABLENAME, Database.FIELD_HASH,
        Database.FIELD_SONG_ID, Database.FIELD_OFFSET, Database.FIELD_HASH,
        Database.FIELD_SONG_ID, Database.FIELD_OFFSET, Database.FIELD_HASH,
        Database.FIELD_SONG_ID, SONGS_TABLENAME, Database.FIELD_SONG_ID
    )

    CREATE_SONGS_TABLE = """
        CREATE TABLE IF NOT EXISTS `%s` (
            `%s` mediumint unsigned not null auto_increment,
            `%s` varchar(250) not null,
            `%s` tinyint default 0,
            `%s` binary(20) not null,
        PRIMARY KEY (`%s`),
        UNIQUE KEY `%s` (`%s`)
    ) ENGINE=INNODB;""" % (
        SONGS_TABLENAME, Database.FIELD_SONG_ID, Database.FIELD_SONGNAME, FIELD_FINGE
RPRINTED,
        Database.FIELD_FILE_SHA1,
        Database.FIELD_SONG_ID, Database.FIELD_SONG_ID, Database.FIELD_SONG_ID,
    )

    # inserts (ignores duplicates)
    INSERT_FINGERPRINT = """
        INSERT IGNORE INTO %s (%s, %s, %s) values
            (UNHEX(%%s), %%s, %%s);
    """ % (FINGERPRINTS_TABLENAME, Database.FIELD_HASH, Database.FIELD_SONG_ID, Databa
se.FIELD_OFFSET)

    INSERT_SONG = "INSERT INTO %s (%s, %s) values (%%s, UNHEX(%%s));" % (
        SONGS_TABLENAME, Database.FIELD_SONGNAME, Database.FIELD_FILE_SHA1)

    # selects
    SELECT = """
        SELECT %s, %s FROM %s WHERE %s = UNHEX(%%s);
    """ % (Database.FIELD_SONG_ID, Database.FIELD_OFFSET, FINGERPRINTS_TABLENAME, Datab
ase.FIELD_HASH)
```

```
    SELECT_MULTIPLE = """
        SELECT HEX(%s), %s, %s FROM %s WHERE %s IN (%%s);
    """ % (Database.FIELD_HASH, Database.FIELD_SONG_ID, Database.FIELD_OFFSET,
           FINGERPRINTS_TABLENAME, Database.FIELD_HASH)

    SELECT_ALL = """
        SELECT %s, %s FROM %s;
    """ % (Database.FIELD_SONG_ID, Database.FIELD_OFFSET, FINGERPRINTS_TABLENAME)

    SELECT_SONG = """
        SELECT %s, HEX(%s) as %s FROM %s WHERE %s = %%s;
    """ % (Database.FIELD_SONGNAME, Database.FIELD_FILE_SHA1, Database.FIELD_FILE_SHA1,
SONGS_TABLENAME, Database.FIELD_SONG_ID)

    SELECT_NUM_FINGERPRINTS = """
        SELECT COUNT(*) as n FROM %s
    """ % (FINGERPRINTS_TABLENAME)

    SELECT_UNIQUE_SONG_IDS = """
        SELECT COUNT(DISTINCT %s) as n FROM %s WHERE %s = 1;
    """ % (Database.FIELD_SONG_ID, SONGS_TABLENAME, FIELD_FINGERPRINTED)

    SELECT_SONGS = """
        SELECT %s, %s, HEX(%s) as %s FROM %s WHERE %s = 1;
    """ % (Database.FIELD_SONG_ID, Database.FIELD_SONGNAME, Database.FIELD_FILE_SHA1, D
atabase.FIELD_FILE_SHA1,
           SONGS_TABLENAME, FIELD_FINGERPRINTED)

    # drops
    DROP_FINGERPRINTS = "DROP TABLE IF EXISTS %s;" % FINGERPRINTS_TABLENAME
    DROP_SONGS = "DROP TABLE IF EXISTS %s;" % SONGS_TABLENAME

    # update
    UPDATE_SONG_FINGERPRINTED = """
        UPDATE %s SET %s = 1 WHERE %s = %%s
    """ % (SONGS_TABLENAME, FIELD_FINGERPRINTED, Database.FIELD_SONG_ID)

    # delete
    DELETE_UNFINGERPRINTED = """
        DELETE FROM %s WHERE %s = 0;
    """ % (SONGS_TABLENAME, FIELD_FINGERPRINTED)
```

```
def __init__(self, **options):
    super(SQLDatabase, self).__init__()
    self.cursor = cursor_factory(**options)
    self._options = options

def after_fork(self):
    # Clear the cursor cache, we don't want any stale connections from
    # the previous process.
    Cursor.clear_cache()

def setup(self):
    """
    Creates any non-existing tables required for dejavu to function.

    This also removes all songs that have been added but have no
    fingerprints associated with them.
    """
    with self.cursor() as cur:
        cur.execute(self.CREATE_SONGS_TABLE)
        cur.execute(self.CREATE_FINGERPRINTS_TABLE)
        cur.execute(self.DELETE_UNFINGERPRINTED)

def empty(self):
    """
    Drops tables created by dejavu and then creates them again
    by calling `SQLDatabase.setup`.

    .. warning:
        This will result in a loss of data
    """
    with self.cursor() as cur:
        cur.execute(self.DROP_FINGERPRINTS)
        cur.execute(self.DROP_SONGS)

    self.setup()

def delete_unfingerprinted_songs(self):
    """
    Removes all songs that have no fingerprints associated with them.
    """
    with self.cursor() as cur:
        cur.execute(self.DELETE_UNFINGERPRINTED)
```

```
def get_num_songs(self):
    """
    Returns number of songs the database has fingerprinted.
    """
    with self.cursor() as cur:
        cur.execute(self.SELECT_UNIQUE_SONG_IDS)

        for count, in cur:
            return count
        return 0

def get_num_fingerprints(self):
    """
    Returns number of fingerprints the database has fingerprinted.
    """
    with self.cursor() as cur:
        cur.execute(self.SELECT_NUM_FINGERPRINTS)

        for count, in cur:
            return count
        return 0

def set_song_fingerprinted(self, sid):
    """
    Set the fingerprinted flag to TRUE (1) once a song has been completely
    fingerprinted in the database.
    """
    with self.cursor() as cur:
        cur.execute(self.UPDATE_SONG_FINGERPRINTED, (sid,))

def get_songs(self):
    """
    Return songs that have the fingerprinted flag set TRUE (1).
    """
    with self.cursor(cursor_type=DictCursor) as cur:
        cur.execute(self.SELECT_SONGS)
        for row in cur:
            yield row

def get_song_by_id(self, sid):
    """
```

```
        Returns song by its ID.
        """
        with self.cursor(cursor_type=DictCursor) as cur:
            cur.execute(self.SELECT_SONG, (sid,))
            return cur.fetchone()

    def insert(self, hash, sid, offset):
        """
        Insert a (sha1, song_id, offset) row into database.
        """
        with self.cursor() as cur:
            cur.execute(self.INSERT_FINGERPRINT, (hash, sid, offset))

    def insert_song(self, songname, file_hash):
        """
        Inserts song in the database and returns the ID of the inserted record.
        """
        with self.cursor() as cur:
            cur.execute(self.INSERT_SONG, (songname, file_hash))
            return cur.lastrowid

    def query(self, hash):
        """
        Return all tuples associated with hash.

        If hash is None, returns all entries in the
        database (be careful with that one!).
        """
        # select all if no key
        query = self.SELECT_ALL if hash is None else self.SELECT

        with self.cursor() as cur:
            cur.execute(query)
            for sid, offset in cur:
                yield (sid, offset)

    def get_iterable_kv_pairs(self):
        """
        Returns all tuples in database.
        """
        return self.query(None)
```

```
def insert_hashes(self, sid, hashes):
    """
    Insert series of hash => song_id, offset
    values into the database.
    """
    values = []
    for hash, offset in hashes:
        values.append((hash, sid, offset))

    with self.cursor() as cur:
        for split_values in grouper(values, 1000):
            cur.executemany(self.INSERT_FINGERPRINT, split_values)

def return_matches(self, hashes):
    """
    Return the (song_id, offset_diff) tuples associated with
    a list of (sha1, sample_offset) values.
    """
    # Create a dictionary of hash => offset pairs for later lookups
    mapper = {}
    for hash, offset in hashes:
        mapper[hash.upper()] = offset

    # Get an iteratable of all the hashes we need
    values = mapper.keys()

    with self.cursor() as cur:
        for split_values in grouper(values, 1000):
            # Create our IN part of the query
            query = self.SELECT_MULTIPLE
            query = query % ', '.join(['UNHEX(%s)'] * len(split_values))

            cur.execute(query, split_values)

            for hash, sid, offset in cur:
                # (sid, db_offset - song_sampled_offset)
                yield (sid, offset - mapper[hash])

def __getstate__(self):
    return (self._options,)

def __setstate__(self, state):
```

```
        self._options, = state
        self.cursor = cursor_factory(**self._options)


def grouper(iterable, n, fillvalue=None):
    args = [iter(iterable)] * n
    return (filter(None, values) for values
            in izip_longest(fillvalue=fillvalue, *args))


def cursor_factory(**factory_options):
    def cursor(**options):
        options.update(factory_options)
        return Cursor(**options)
    return cursor


class Cursor(object):
    """
    Establishes a connection to the database and returns an open cursor.


    ```python
    # Use as context manager
    with Cursor() as cur:
        cur.execute(query)
    ```
    """
    _cache = Queue.Queue(maxsize=5)

    def __init__(self, cursor_type=mysql.cursors.Cursor, **options):
        super(Cursor, self).__init__()

        try:
            conn = self._cache.get_nowait()
        except Queue.Empty:
            conn = mysql.connect(**options)
        else:
            # Ping the connection before using it from the cache.
            conn.ping(True)

        self.conn = conn
```

```
        self.conn.autocommit(False)
        self.cursor_type = cursor_type

    @classmethod
    def clear_cache(cls):
        cls._cache = Queue.Queue(maxsize=5)

    def __enter__(self):
        self.cursor = self.conn.cursor(self.cursor_type)
        return self.cursor

    def __exit__(self, extype, exvalue, traceback):
        # if we had a MySQL related error we try to rollback the cursor.
        if extype is mysql.MySQLError:
            self.cursor.rollback()

        self.cursor.close()
        self.conn.commit()

        # Put it back on the queue
        try:
            self._cache.put_nowait(self.conn)
        except Queue.Full:
            self.conn.close()
```

“decoder.py”

```
import os
import fnmatch
import numpy as np
from pydub import AudioSegment
from pydub.utils import audioop
import wavio
from hashlib import sha1

def unique_hash(filepath, blocksize=2**20):
    """ Small function to generate a hash to uniquely generate
    a file. Inspired by MD5 version here:
    http://stackoverflow.com/a/1131255/712997
```

```
    Works with large files.
    """
    s = sha1()
    with open(filepath , "rb") as f:
        while True:
            buf = f.read(blocksize)
            if not buf:
                break
            s.update(buf)
    return s.hexdigest().upper()


def find_files(path, extensions):
    # Allow both with ".mp3" and without "mp3" to be used for extensions
    extensions = [e.replace(".", "") for e in extensions]

    for dirpath, dirnames, files in os.walk(path):
        for extension in extensions:
            for f in fnmatch.filter(files, "*.%s" % extension):
                p = os.path.join(dirpath, f)
                yield (p, extension)


def read(filename, limit=None):
    """
    Reads any file supported by pydub (ffmpeg) and returns the data contained
    within. If file reading fails due to input being a 24-bit wav file,
    wavio is used as a backup.

    Can be optionally limited to a certain amount of seconds from the start
    of the file by specifying the `limit` parameter. This is the amount of
    seconds from the start of the file.

    returns: (channels, samplerate)
    """
    # pydub does not support 24-bit wav files, use wavio when this occurs
    try:
        audiofile = AudioSegment.from_file(filename)

        if limit:
            audiofile = audiofile[:limit * 1000]
```

```
            data = np.fromstring(audiofile._data, np.int16)

            channels = []
            for chn in xrange(audiofile.channels):
                channels.append(data[chn::audiofile.channels])

            fs = audiofile.frame_rate
        except audioop.error:
            fs, _, audiofile = wavio.readwav(filename)

            if limit:
                audiofile = audiofile[:limit * 1000]

            audiofile = audiofile.T
            audiofile = audiofile.astype(np.int16)

            channels = []
            for chn in audiofile:
                channels.append(chn)

        return channels, audiofile.frame_rate, unique_hash(filename)


def path_to_songname(path):
    """
    Extracts song name from a filepath. Used to identify which songs
    have already been fingerprinted on disk.
    """
    return os.path.splitext(os.path.basename(path))[0]
```

“fingerprint.py”

```
import numpy as np
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt
from scipy.ndimage.filters import maximum_filter
from scipy.ndimage.morphology import (generate_binary_structure,
                                      iterate_structure, binary_erosion)
```

```
import hashlib
from operator import itemgetter

IDX_FREQ_I = 0
IDX_TIME_J = 1

######################################################################
# Sampling rate, related to the Nyquist conditions, which affects
# the range frequencies we can detect.
DEFAULT_FS = 44100

######################################################################
# Size of the FFT window, affects frequency granularity
DEFAULT_WINDOW_SIZE = 4096

######################################################################
# Ratio by which each sequential window overlaps the last and the
# next window. Higher overlap will allow a higher granularity of offset
# matching, but potentially more fingerprints.
DEFAULT_OVERLAP_RATIO = 0.5

######################################################################
# Degree to which a fingerprint can be paired with its neighbors --
# higher will cause more fingerprints, but potentially better accuracy.
DEFAULT_FAN_VALUE = 15

######################################################################
# Minimum amplitude in spectrogram in order to be considered a peak.
# This can be raised to reduce number of fingerprints, but can negatively
# affect accuracy.
DEFAULT_AMP_MIN = 10

######################################################################
# Number of cells around an amplitude peak in the spectrogram in order
# for Dejavu to consider it a spectral peak. Higher values mean less
# fingerprints and faster matching, but can potentially affect accuracy.
PEAK_NEIGHBORHOOD_SIZE = 20

######################################################################
# Thresholds on how close or far fingerprints can be in time in order
# to be paired as a fingerprint. If your max is too low, higher values of
# DEFAULT_FAN_VALUE may not perform as expected.
```

```
MIN_HASH_TIME_DELTA = 0
MAX_HASH_TIME_DELTA = 200

######################################################################
# If True, will sort peaks temporally for fingerprinting;
# not sorting will cut down number of fingerprints, but potentially
# affect performance.
PEAK_SORT = True

######################################################################
# Number of bits to throw away from the front of the SHA1 hash in the
# fingerprint calculation. The more you throw away, the less storage, but
# potentially higher collisions and misclassifications when identifying songs.
FINGERPRINT_REDUCTION = 20

def fingerprint(channel_samples, Fs=DEFAULT_FS,
                wsize=DEFAULT_WINDOW_SIZE,
                wratio=DEFAULT_OVERLAP_RATIO,
                fan_value=DEFAULT_FAN_VALUE,
                amp_min=DEFAULT_AMP_MIN):
    """
    FFT the channel, log transform output, find local maxima, then return
    locally sensitive hashes.
    """
    # FFT the signal and extract frequency components
    arr2D = mlab.specgram(
        channel_samples,
        NFFT=wsize,
        Fs=Fs,
        window=mlab.window_hanning,
        noverlap=int(wsize * wratio))[0]

    # apply log transform since specgram() returns linear array
    arr2D = 10 * np.log10(arr2D)
    arr2D[arr2D == -np.inf] = 0  # replace infs with zeros

    # find local maxima
    local_maxima = get_2D_peaks(arr2D, plot=False, amp_min=amp_min)

    # return hashes
    return generate_hashes(local_maxima, fan_value=fan_value)
```

```
def get_2D_peaks(arr2D, plot=False, amp_min=DEFAULT_AMP_MIN):
    # http://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.morphology.iterate_str
ucture.html#scipy.ndimage.morphology.iterate_structure
    struct = generate_binary_structure(2, 1)
    neighborhood = iterate_structure(struct, PEAK_NEIGHBORHOOD_SIZE)

    # find local maxima using our fliter shape
    local_max = maximum_filter(arr2D, footprint=neighborhood) == arr2D
    background = (arr2D == 0)
    eroded_background = binary_erosion(background, structure=neighborhood,
                                       border_value=1)

    # Boolean mask of arr2D with True at peaks
    detected_peaks = local_max - eroded_background

    # extract peaks
    amps = arr2D[detected_peaks]
    j, i = np.where(detected_peaks)

    # filter peaks
    amps = amps.flatten()
    peaks = zip(i, j, amps)
    peaks_filtered = [x for x in peaks if x[2] > amp_min]  # freq, time, amp

    # get indices for frequency and time
    frequency_idx = [x[1] for x in peaks_filtered]
    time_idx = [x[0] for x in peaks_filtered]

    if plot:
        # scatter of the peaks
        fig, ax = plt.subplots()
        ax.imshow(arr2D)
        ax.scatter(time_idx, frequency_idx)
        ax.set_xlabel('Time')
        ax.set_ylabel('Frequency')
        ax.set_title("Spectrogram")
        plt.gca().invert_yaxis()
        plt.show()

    return zip(frequency_idx, time_idx)
```

```
def generate_hashes(peaks, fan_value=DEFAULT_FAN_VALUE):
    """
    Hash list structure:
      sha1_hash[0:20]    time_offset
    [(e05b341a9b77a51fd26, 32), ... ]
    """
    if PEAK_SORT:
        peaks.sort(key=itemgetter(1))

    for i in range(len(peaks)):
        for j in range(1, fan_value):
            if (i + j) < len(peaks):

                freq1 = peaks[i][IDX_FREQ_I]
                freq2 = peaks[i + j][IDX_FREQ_I]
                t1 = peaks[i][IDX_TIME_J]
                t2 = peaks[i + j][IDX_TIME_J]
                t_delta = t2 - t1

                if t_delta >= MIN_HASH_TIME_DELTA and t_delta <= MAX_HASH_TIME_DELTA:
                    h = hashlib.sha1(
                        "%s|%s|%s" % (str(freq1), str(freq2), str(t_delta)))
                    yield (h.hexdigest()[0:FINGERPRINT_REDUCTION], t1)
```

“recognise.py”

```
import dejavu.fingerprint as fingerprint
import dejavu.decoder as decoder
import numpy as np
import pyaudio
import time


class BaseRecognizer(object):

    def __init__(self, dejavu):
        self.dejavu = dejavu
        self.Fs = fingerprint.DEFAULT_FS

    def _recognize(self, *data):
```

```
        matches = []
        for d in data:
            matches.extend(self.dejavu.find_matches(d, Fs=self.Fs))
        return self.dejavu.align_matches(matches)

    def recognize(self):
        pass  # base class does nothing


class FileRecognizer(BaseRecognizer):
    def __init__(self, dejavu):
        super(FileRecognizer, self).__init__(dejavu)

    def recognize_file(self, filename):
        frames, self.Fs, file_hash = decoder.read(filename, self.dejavu.limit)

        t = time.time()
        match = self._recognize(*frames)
        t = time.time() - t

        if match:
            match['match_time'] = t

        return match

    def recognize(self, filename):
        return self.recognize_file(filename)


class MicrophoneRecognizer(BaseRecognizer):
    default_chunksize   = 8192
    default_format      = pyaudio.paInt16
    default_channels    = 2
    default_samplerate  = 44100

    def __init__(self, dejavu):
        super(MicrophoneRecognizer, self).__init__(dejavu)
        self.audio = pyaudio.PyAudio()
        self.stream = None
        self.data = []
        self.channels = MicrophoneRecognizer.default_channels
        self.chunksize = MicrophoneRecognizer.default_chunksize
```

```
    self.samplerate = MicrophoneRecognizer.default_samplerate
    self.recorded = False

def start_recording(self, channels=default_channels,
                    samplerate=default_samplerate,
                    chunksize=default_chunksize):
    self.chunksize = chunksize
    self.channels = channels
    self.recorded = False
    self.samplerate = samplerate

    if self.stream:
        self.stream.stop_stream()
        self.stream.close()

    self.stream = self.audio.open(
        format=self.default_format,
        channels=channels,
        rate=samplerate,
        input=True,
        frames_per_buffer=chunksize,
    )

    self.data = [[] for i in range(channels)]

def process_recording(self):
    data = self.stream.read(self.chunksize)
    nums = np.fromstring(data, np.int16)
    for c in range(self.channels):
        self.data[c].extend(nums[c::self.channels])

def stop_recording(self):
    self.stream.stop_stream()
    self.stream.close()
    self.stream = None
    self.recorded = True

def recognize_recording(self):
    if not self.recorded:
        raise NoRecordingError("Recording was not complete/begun")
    return self._recognize(*self.data)
```

```
    def get_recorded_time(self):
        return len(self.data[0]) / self.rate

    def recognize(self, seconds=10):
        self.start_recording()
        for i in range(0, int(self.samplerate / self.chunksize
                              * seconds)):
            self.process_recording()
        self.stop_recording()
        return self.recognize_recording()


class NoRecordingError(Exception):
    pass
```

“testing.py”

```
from __future__ import division
from pydub import AudioSegment
from dejavu.decoder import path_to_songname
from dejavu import Dejavu
from dejavu.fingerprint import *
import traceback
import fnmatch
import os, re, ast
import subprocess
import random
import logging

def set_seed(seed=None):
    """
    `seed` as None means that the sampling will be random.

    Setting your own seed means that you can produce the
    same experiment over and over.
    """
    if seed != None:
        random.seed(seed)
```

```
def get_files_recursive(src, fmt):
    """
    `src` is the source directory.
    `fmt` is the extension, ie ".mp3" or "mp3", etc.
    """
    for root, dirnames, filenames in os.walk(src):
        for filename in fnmatch.filter(filenames, '*' + fmt):
            yield os.path.join(root, filename)

def get_length_audio(audiopath, extension):
    """
    Returns length of audio in seconds.
    Returns None if format isn't supported or in case of error.
    """
    try:
        audio = AudioSegment.from_file(audiopath, extension.replace(".", ""))
    except:
        print "Error in get_length_audio(): %s" % traceback.format_exc()
        return None
    return int(len(audio) / 1000.0)

def get_starttime(length, nseconds, padding):
    """
    `length` is total audio length in seconds
    `nseconds` is amount of time to sample in seconds
    `padding` is off-limits seconds at beginning and ending
    """
    maximum = length - padding - nseconds
    if padding > maximum:
        return 0
    return random.randint(padding, maximum)

def generate_test_files(src, dest, nseconds, fmts=[".mp3", ".wav"], padding=10):
    """
    Generates a test file for each file recursively in `src` directory
    of given format using `nseconds` sampled from the audio file.

    Results are written to `dest` directory.

    `padding` is the number of off-limit seconds and the beginning and
    end of a track that won't be sampled in testing. Often you want to
    avoid silence, etc.
```

```
    """
    # create directories if necessary
    for directory in [src, dest]:
        try:
            os.stat(directory)
        except:
            os.mkdir(directory)

    # find files recursively of a given file format
    for fmt in fmts:
        testsources = get_files_recursive(src, fmt)
        for audiosource in testsources:

            print "audiosource:", audiosource

            filename, extension = os.path.splitext(os.path.basename(audiosource))
            length = get_length_audio(audiosource, extension)
            starttime = get_starttime(length, nseconds, padding)

            test_file_name = "%s_%s_%ssec.%s" % (
                os.path.join(dest, filename), starttime,
                nseconds, extension.replace(".", ""))

            subprocess.check_output([
                "ffmpeg", "-y",
                "-ss", "%d" % starttime,
                '-t' , "%d" % nseconds,
                "-i", audiosource,
                test_file_name])

def log_msg(msg, log=True, silent=False):
    if log:
        logging.debug(msg)
    if not silent:
        print msg

def autolabel(rects, ax):
    # attach some text labels
    for rect in rects:
        height = rect.get_height()
        ax.text(rect.get_x() + rect.get_width() / 2., 1.05 * height,
            '%d' % int(height), ha='center', va='bottom')
```

```
def autolabeldoubles(rects, ax):
    # attach some text labels
    for rect in rects:
        height = rect.get_height()
        ax.text(rect.get_x() + rect.get_width() / 2., 1.05 * height,
            '%s' % round(float(height), 3), ha='center', va='bottom')

class DejavuTest(object):
    def __init__(self, folder, seconds):
        super(DejavuTest, self).__init__()

        self.test_folder = folder
        self.test_seconds = seconds
        self.test_songs = []

        print "test_seconds", self.test_seconds

        self.test_files = [
            f for f in os.listdir(self.test_folder)
            if os.path.isfile(os.path.join(self.test_folder, f))
            and re.findall("[0-9]*sec", f)[0] in self.test_seconds]

        print "test_files", self.test_files

        self.n_columns = len(self.test_seconds)
        self.n_lines = int(len(self.test_files) / self.n_columns)

        print "columns:", self.n_columns
        print "length of test files:", len(self.test_files)
        print "lines:", self.n_lines

        # variable match results (yes, no, invalid)
        self.result_match = [[0 for x in xrange(self.n_columns)] for x in xrange(self.n_lines)]

        print "result_match matrix:", self.result_match

        # variable match precision (if matched in the corrected time)
        self.result_matching_times = [[0 for x in xrange(self.n_columns)] for x in xrange(self.n_lines)]


        # variable mahing time (query time)
```

```
        self.result_query_duration = [[0 for x in xrange(self.n_columns)] for x in xrange(self.n_lines)]


        # variable confidence
        self.result_match_confidence = [[0 for x in xrange(self.n_columns)] for x in xrange(self.n_line
s)]

        self.begin()

    def get_column_id (self, secs):
        for i, sec in enumerate(self.test_seconds):
            if secs == sec:
                return i

    def get_line_id (self, song):
        for i, s in enumerate(self.test_songs):
            if song == s:
                return i
        self.test_songs.append(song)
        return len(self.test_songs) - 1

    def create_plots(self, name, results, results_folder):
        for sec in range(0, len(self.test_seconds)):
            ind = np.arange(self.n_lines) #
            width = 0.25       # the width of the bars

            fig = plt.figure()
            ax = fig.add_subplot(111)
            ax.set_xlim([-1 * width, 2 * width])

            means_dvj = [x[0] for x in results[sec]]
            rects1 = ax.bar(ind, means_dvj, width, color='r')

            # add some
            ax.set_ylabel(name)
            ax.set_title("%s %s Results" % (self.test_seconds[sec], name))
            ax.set_xticks(ind + width)

            labels = [0 for x in range(0, self.n_lines)]
            for x in range(0, self.n_lines):
                labels[x] = "song %s" % (x+1)
            ax.set_xticklabels(labels)
```

```
        box = ax.get_position()
        ax.set_position([box.x0, box.y0, box.width * 0.75, box.height])

        #ax.legend( (rects1[0]), ('Dejavu'), loc='center left', bbox_to_anchor=(1, 0.5))

        if name == 'Confidence':
            autolabel(rects1, ax)
        else:
            autolabeldoubles(rects1, ax)

        plt.grid()

        fig_name = os.path.join(results_folder, "%s_%s.png" % (name, self.test_seconds[sec]))
        fig.savefig(fig_name)

    def begin(self):
        for f in self.test_files:
            log_msg('---------------------------------------------------')
            log_msg('file: %s' % f)

            # get column
            col = self.get_column_id(re.findall("[0-9]*sec", f)[0])
            # format: XXXX_offset_length.mp3
            song = path_to_songname(f).split("_")[0]
            line = self.get_line_id(song)
            result = subprocess.check_output([
                "python",
                "dejavu.py",
                '-r',
                'file',
                self.test_folder + "/" + f])

            if result.strip() == "None":
                log_msg('No match')
                self.result_match[line][col] = 'no'
                self.result_matching_times[line][col] = 0
                self.result_query_duration[line][col] = 0
                self.result_match_confidence[line][col] = 0

            else:
                result = result.strip()
```

```
result = result.replace(" \'", ' "')
result = result.replace("{\'", '{"')
result = result.replace("\':", '":')
result = result.replace("\',", '",')

# which song did we predict?
result = ast.literal_eval(result)
song_result = result["song_name"]
log_msg('song: %s' % song)
log_msg('song_result: %s' % song_result)

if song_result != song:
    log_msg('invalid match')
    self.result_match[line][col] = 'invalid'
    self.result_matching_times[line][col] = 0
    self.result_query_duration[line][col] = 0
    self.result_match_confidence[line][col] = 0
else:
    log_msg('correct match')
    print self.result_match
    self.result_match[line][col] = 'yes'
    self.result_query_duration[line][col] = round(result[Dejavu.MATCH_TIME],3)
    self.result_match_confidence[line][col] = result[Dejavu.CONFIDENCE]

    song_start_time = re.findall("\_[^\_]+",f)
    song_start_time = song_start_time[0].lstrip("_ ")

    result_start_time = round((result[Dejavu.OFFSET] * DEFAULT_WINDOW_SIZE *
        DEFAULT_OVERLAP_RATIO) / (DEFAULT_FS), 0)

    self.result_matching_times[line][col] = int(result_start_time) - int(song_start_time)
    if (abs(self.result_matching_times[line][col]) == 1):
        self.result_matching_times[line][col] = 0

    log_msg('query duration: %s' % round(result[Dejavu.MATCH_TIME],3))
    log_msg('confidence: %s' % result[Dejavu.CONFIDENCE])
    log_msg('song start_time: %s' % song_start_time)
    log_msg('result start time: %s' % result_start_time)
    if (self.result_matching_times[line][col] == 0):
        log_msg('accurate match')
    else:
        log_msg('inaccurate match')
```

```
        log_msg('-------------------------------------------------\n')
```

“wavio.py”

```
# wavio.py
# Author: Warren Weckesser
# License: BSD 3-Clause (http://opensource.org/licenses/BSD-3-Clause)
# Synopsis: A Python module for reading and writing 24 bit WAV files.
# Github: github.com/WarrenWeckesser/wavio

import wave as _wave
import numpy as _np


def _wav2array(nchannels, sampwidth, data):
    """data must be the string containing the bytes from the wav file."""
    num_samples, remainder = divmod(len(data), sampwidth * nchannels)
    if remainder > 0:
        raise ValueError('The length of data is not a multiple of '
                         'sampwidth * num_channels.')
    if sampwidth > 4:
        raise ValueError("sampwidth must not be greater than 4.")

    if sampwidth == 3:
        a = _np.empty((num_samples, nchannels, 4), dtype=_np.uint8)
        raw_bytes = _np.fromstring(data, dtype=_np.uint8)
        a[:, :, :sampwidth] = raw_bytes.reshape(-1, nchannels, sampwidth)
        a[:, :, sampwidth:] = (a[:, :, sampwidth - 1:sampwidth] >> 7) * 255
        result = a.view('<i4').reshape(a.shape[:-1])
    else:
        # 8 bit samples are stored as unsigned ints; others as signed ints.
        dt_char = 'u' if sampwidth == 1 else 'i'
        a = _np.fromstring(data, dtype='<%s%d' % (dt_char, sampwidth))
        result = a.reshape(-1, nchannels)
    return result


def readwav(file):
    """
```

```
    Read a WAV file.

    Parameters
    ----------
    file : string or file object
        Either the name of a file or an open file pointer.

    Return Values
    -------------
    rate : float
        The sampling frequency (i.e. frame rate)
    sampwidth : float
        The sample width, in bytes.  E.g. for a 24 bit WAV file,
        sampwidth is 3.
    data : numpy array
        The array containing the data.  The shape of the array is
        (num_samples, num_channels).  num_channels is the number of
        audio channels (1 for mono, 2 for stereo).

    Notes
    -----
    This function uses the `wave` module of the Python standard libary
    to read the WAV file, so it has the same limitations as that library.
    In particular, the function does not read compressed WAV files.

    """
    wav = _wave.open(file)
    rate = wav.getframerate()
    nchannels = wav.getnchannels()
    sampwidth = wav.getsampwidth()
    nframes = wav.getnframes()
    data = wav.readframes(nframes)
    wav.close()
    array = _wav2array(nchannels, sampwidth, data)
    return rate, sampwidth, array


def writewav24(filename, rate, data):
    """
    Create a 24 bit wav file.

    Parameters
```

```
----------
filename : string
    Name of the file to create.
rate : float
    The sampling frequency (i.e. frame rate) of the data.
data : array-like collection of integer or floating point values
    data must be "array-like", either 1- or 2-dimensional.  If it
    is 2-d, the rows are the frames (i.e. samples) and the columns
    are the channels.

Notes
-----
The data is assumed to be signed, and the values are assumed to be
within the range of a 24 bit integer.  Floating point values are
converted to integers.  The data is not rescaled or normalized before
writing it to the file.

Example
-------
Create a 3 second 440 Hz sine wave.

>>> rate = 22050  # samples per second
>>> T = 3         # sample duration (seconds)
>>> f = 440.0     # sound frequency (Hz)
>>> t = np.linspace(0, T, T*rate, endpoint=False)
>>> x = (2**23 - 1) * np.sin(2 * np.pi * f * t)
>>> writewav24("sine24.wav", rate, x)

"""
a32 = _np.asarray(data, dtype=_np.int32)
if a32.ndim == 1:
    # Convert to a 2D array with a single column.
    a32.shape = a32.shape + (1,)
# By shifting first 0 bits, then 8, then 16, the resulting output
# is 24 bit little-endian.
a8 = (a32.reshape(a32.shape + (1,)) >> _np.array([0, 8, 16])) & 255
wavdata = a8.astype(_np.uint8).tostring()

w = _wave.open(filename, 'wb')
w.setnchannels(a32.shape[1])
w.setsampwidth(3)
w.setframerate(rate)
```

```
    w.writeframes(wavdata)
    w.close()
```

“dejavu.py”

```
#!/usr/bin/python

import os
import sys
import json
import warnings
import argparse

from dejavu import Dejavu
from dejavu.recognize import FileRecognizer
from dejavu.recognize import MicrophoneRecognizer
from argparse import RawTextHelpFormatter

warnings.filterwarnings("ignore")

DEFAULT_CONFIG_FILE = "dejavu.cnf.SAMPLE"


def init(configpath):
    """
    Load config from a JSON file
    """
    try:
        with open(configpath) as f:
            config = json.load(f)
    except IOError as err:
        print("Cannot open configuration: %s. Exiting" % (str(err)))
        sys.exit(1)

    # create a Dejavu instance
    return Dejavu(config)


if __name__ == '__main__':
```

```
parser = argparse.ArgumentParser(
    description="Dejavu: Audio Fingerprinting library",
    formatter_class=RawTextHelpFormatter)
parser.add_argument('-c', '--config', nargs='?',
                    help='Path to configuration file\n'
                         'Usages: \n'
                         '--config /path/to/config-file\n')
parser.add_argument('-f', '--fingerprint', nargs='*',
                    help='Fingerprint files in a directory\n'
                         'Usages: \n'
                         '--fingerprint /path/to/directory extension\n'
                         '--fingerprint /path/to/directory')
parser.add_argument('-r', '--recognize', nargs=2,
                    help='Recognize what is '
                         'playing through the microphone\n'
                         'Usage: \n'
                         '--recognize mic number_of_seconds \n'
                         '--recognize file path/to/file \n')
args = parser.parse_args()

if not args.fingerprint and not args.recognize:
    parser.print_help()
    sys.exit(0)

config_file = args.config
if config_file is None:
    config_file = DEFAULT_CONFIG_FILE
    # print "Using default config file: %s" % (config_file)

djv = init(config_file)
if args.fingerprint:
    # Fingerprint all files in a directory
    if len(args.fingerprint) == 2:
        directory = args.fingerprint[0]
        extension = args.fingerprint[1]
        print("Fingerprinting all .%s files in the %s directory"
              % (extension, directory))
        djv.fingerprint_directory(directory, ["." + extension], 4)

    elif len(args.fingerprint) == 1:
        filepath = args.fingerprint[0]
        if os.path.isdir(filepath):
```

```
            print("Please specify an extension if you'd like to fingerprint a directory!")
            sys.exit(1)
        djv.fingerprint_file(filepath)

    elif args.recognize:
        # Recognize audio source
        song = None
        source = args.recognize[0]
        opt_arg = args.recognize[1]

        if source in ('mic', 'microphone'):
            song = djv.recognize(MicrophoneRecognizer, seconds=opt_arg)
        elif source == 'file':
            song = djv.recognize(FileRecognizer, opt_arg)
        print(song)

    sys.exit(0)
```

“run_tests.py”

```
from dejavu.testing import *
from dejavu import Dejavu
from optparse import OptionParser
import matplotlib.pyplot as plt
import time
import shutil

usage = "usage: %prog [options] TESTING_AUDIOFOLDER"
parser = OptionParser(usage=usage, version="%prog 1.1")
parser.add_option("--secs",
                  action="store",
                  dest="secs",
                  default=5,
                  type=int,
                  help='Number of seconds starting from zero to test')
parser.add_option("--results",
                  action="store",
                  dest="results_folder",
                  default="./dejavu_test_results",
```

```
                help='Sets the path where the results are saved')
parser.add_option("--temp",
                action="store",
                dest="temp_folder",
                default="./dejavu_temp_testing_files",
                help='Sets the path where the temp files are saved')
parser.add_option("--log",
                action="store_true",
                dest="log",
                default=True,
                help='Enables logging')
parser.add_option("--silent",
                action="store_false",
                dest="silent",
                default=False,
                help='Disables printing')
parser.add_option("--log-file",
                dest="log_file",
                default="results-compare.log",
                help='Set the path and filename of the log file')
parser.add_option("--padding",
                action="store",
                dest="padding",
                default=10,
                type=int,
                help='Number of seconds to pad choice of place to test from')
parser.add_option("--seed",
                action="store",
                dest="seed",
                default=None,
                type=int,
                help='Random seed')
options, args = parser.parse_args()
test_folder = args[0]

# set random seed if set by user
set_seed(options.seed)

# ensure results folder exists
try:
   os.stat(options.results_folder)
except:
```

```
    os.mkdir(options.results_folder)

# set logging
if options.log:
    logging.basicConfig(filename=options.log_file, level=logging.DEBUG)

# set test seconds
test_seconds = ['%dsec' % i for i in range(1, options.secs + 1, 1)]

# generate testing files
for i in range(1, options.secs + 1, 1):
    generate_test_files(test_folder, options.temp_folder,
                        i, padding=options.padding)

# scan files
log_msg("Running Dejavu fingerprinter on files in %s..." % test_folder,
        log=options.log, silent=options.silent)

tm = time.time()
djv = DejavuTest(options.temp_folder, test_seconds)
log_msg("finished obtaining results from dejavu in %s" % (time.time() - tm),
        log=options.log, silent=options.silent)

tests = 1  # djv
n_secs = len(test_seconds)

# set result variables -> 4d variables
all_match_counter = [[[0 for x in xrange(tests)] for x in xrange(3)] for x in xrange(n_secs)]
all_matching_times_counter = [[[0 for x in xrange(tests)] for x in xrange(2)] for x in xrange(n_secs
)]
all_query_duration = [[[0 for x in xrange(tests)] for x in xrange(djv.n_lines)] for x in xrange(n_secs
)]
all_match_confidence = [[[0 for x in xrange(tests)] for x in xrange(djv.n_lines)] for x in xrange(n_s
ecs)]

# group results by seconds
for line in range(0, djv.n_lines):
    for col in range(0, djv.n_columns):
        # for dejavu
        all_query_duration[col][line][0] = djv.result_query_duration[line][col]
        all_match_confidence[col][line][0] = djv.result_match_confidence[line][col]
```

```
        djv_match_result = djv.result_match[line][col]

        if djv_match_result == 'yes':
            all_match_counter[col][0][0] += 1
        elif djv_match_result == 'no':
            all_match_counter[col][1][0] += 1
        else:
            all_match_counter[col][2][0] += 1

        djv_match_acc = djv.result_matching_times[line][col]

        if djv_match_acc == 0 and djv_match_result == 'yes':
            all_matching_times_counter[col][0][0] += 1
        elif djv_match_acc != 0:
            all_matching_times_counter[col][1][0] += 1

# create plots
djv.create_plots('Confidence', all_match_confidence, options.results_folder)
djv.create_plots('Query duration', all_query_duration, options.results_folder)

for sec in range(0, n_secs):
    ind = np.arange(3) #
    width = 0.25       # the width of the bars

    fig = plt.figure()
    ax = fig.add_subplot(111)
    ax.set_xlim([-1 * width, 2.75])

    means_dvj = [round(x[0] * 100 / djv.n_lines, 1) for x in all_match_counter[sec]]
    rects1 = ax.bar(ind, means_dvj, width, color='r')

    # add some
    ax.set_ylabel('Matching Percentage')
    ax.set_title('%s Matching Percentage' % test_seconds[sec])
    ax.set_xticks(ind + width)

    labels = ['yes','no','invalid']
    ax.set_xticklabels( labels )

    box = ax.get_position()
    ax.set_position([box.x0, box.y0, box.width * 0.75, box.height])
    #ax.legend((rects1[0]), ('Dejavu'), loc='center left', bbox_to_anchor=(1, 0.5))
```

```
    autolabeldoubles(rects1,ax)
    plt.grid()

    fig_name = os.path.join(options.results_folder, "matching_perc_%s.png" % test_seconds[sec])
    fig.savefig(fig_name)

for sec in range(0, n_secs):
    ind = np.arange(2) #
    width = 0.25       # the width of the bars

    fig = plt.figure()
    ax = fig.add_subplot(111)
    ax.set_xlim([-1*width, 1.75])

    div = all_match_counter[sec][0][0]
    if div == 0 :
        div = 1000000

    means_dvj = [round(x[0] * 100 / div, 1) for x in all_matching_times_counter[sec]]
    rects1 = ax.bar(ind, means_dvj, width, color='r')

    # add some
    ax.set_ylabel('Matching Accuracy')
    ax.set_title('%s Matching Times Accuracy' % test_seconds[sec])
    ax.set_xticks(ind + width)

    labels = ['yes','no']
    ax.set_xticklabels( labels )

    box = ax.get_position()
    ax.set_position([box.x0, box.y0, box.width * 0.75, box.height])

    #ax.legend( (rects1[0]), ('Dejavu'), loc='center left', bbox_to_anchor=(1, 0.5))
    autolabeldoubles(rects1,ax)

    plt.grid()

    fig_name = os.path.join(options.results_folder, "matching_acc_%s.png" % test_seconds[sec])
    fig.savefig(fig_name)

# remove temporary folder
shutil.rmtree(options.temp_folder)
```

test_dejavu.sh

```
#####################################
### Dejavu example testing script ###
#####################################

############
# Clear out previous results
rm -rf ./results ./temp_audio

############
# Fingerprint files of extension mp3 in the ./mp3 folder
python dejavu.py -f ./mp3/ mp3

##########
# Run a test suite on the ./mp3 folder by extracting 1, 2, 3, 4, and 5
# second clips sampled randomly
# sampling with random seed = 42, and finally
# store results in ./results and log to dejavu-test.log
python run_tests.py \
   --secs 5 \
   --temp ./temp_audio \
   --log-file ./results/dejavu-test.log \
   --padding 2 \
   --seed 42 \
   --results ./results \
   ./mp3
```