

Αναγνώριση ήχου με τη χρήση ηχητικού αποτυπώματος

Περιεχόμενα

1. Αποκωδικοποίηση mp3.....	3
2. Μετατροπή σε τομέα συχνοτήτων.....	4
3. Παραγωγή αποτυπωμάτων.....	5
4. Αναζήτηση βάσει αποτυπωμάτων.....	7
5. Σύγκριση μεταξύ δύο αρχείων.....	8
6. Χρονική ολίσθηση κατά την παραγωγή αποτυπώματος.....	8
7. Χρονική παραμόρφωση (time stretching).....	9
8. Μίξη πολλαπλών αρχείων.....	11
9. Διασκευές.....	11
10. Οδηγίες χρήσης.....	12
11. Ρύθμιση παραμέτρων.....	14
12. Configuration files.....	15
13. Πειράματα/Παραδείγματα αναγνώρισης ήχου.....	16

Αποκωδικοποίηση

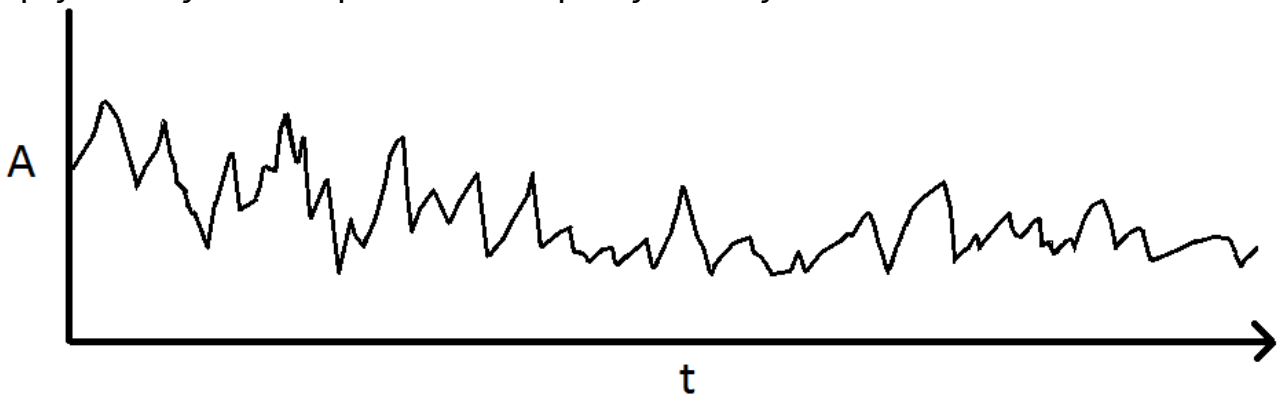
Ένα αρχείο mp3 περιέχει αριθμούς που αντιπροσωπεύουν διαφορετικές εντάσεις του ήχου σε χρονικές στιγμές. Το κάθε αρχείο περιέχει τόσους αριθμούς (“σημεία”) ανά second όσο είναι το sample rate του. Το bit depth του ήχου είναι πόσα bit (ψηφία) ακρίβειας έχει ο κάθε αριθμός-σημείο. Συνήθως, τα αρχεία είναι κωδικοποιημένα σε sample rate των 44100 samples/second με bit depth ίσο με 8bit (single precision), 16bit (double precision, 2 bytes/sample) ή 24bit.

Όταν αποκωδικοποιούμε ένα αρχείο mp3 σε PCM (ένταση ανά χρονική στιγμή) ωστόσο, μπορούμε να επιλέξουμε εμείς σε τι sample rate και bit depth θα το αποκωδικοποιήσουμε καθώς μπορούμε να συμπληρώσουμε ή να ενώσουμε samples για μετατροπή. Το sample rate καθορίζει τη μέγιστη συχνότητα που μπορεί να περιέχουν τα δεδομένα που θα εξάγουμε και το bit depth την ακρίβεια του κάθε sample.

Για τη διαδικασία μας λοιπόν αποκωδικοποιούμε τα αρχεία mp3 σε μορφή 16bit PCM (ένταση ανά χρονική στιγμή) με 44100 samples/second ώστε να έχουμε τα raw audio data. Έτσι έχουμε μία ακολουθία αριθμών τύπου:

58, 92, -192, 5421, 5302, 3581, 8, ...

Αυτοί οι αριθμοί αντιπροσωπεύουν την ένταση του ήχου στη χρονική στιγμή που αντιστοιχεί στον καθένα και έχουμε 44100 “στιγμές” ανά δευτερόλεπτο. Αν δημιουργούσαμε ένα γράφημα από μία μεγάλη τέτοια ακολουθία αριθμών θα είχαμε μία εικόνα της κυματομορφής (μεταβολής της έντασης) του ήχου με οριζόντιο άξονα τον χρόνο που θα έμοιαζε κάπως έτσι:



Μετατροπή

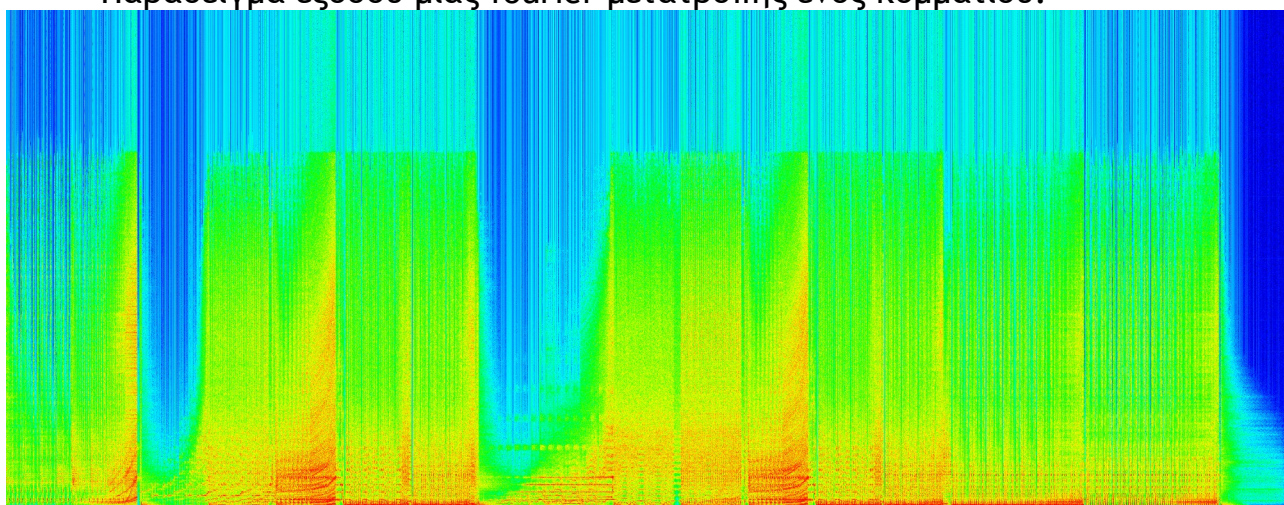
(fourier, από τον τομέα χρόνου στον τομέα συχνοτήτων)

Επιλέγουμε ένα χρονικό διάστημα που επιθυμούμε να καλύπτει το κάθε αποτύπωμα και με βάση αυτό έχουμε το μέγεθος παραθύρου της μετατροπής μας. Αν για παράδειγμα το χρονικό διάστημα είναι 100ms και αποκωδικοποιήσαμε το αρχείο με 44100 samples/sec τότε το παράθυρο μετατροπής θα είναι $44100 \cdot (100/1000) = 4410$. Έτσι παίρνουμε 4410 από τους παραπάνω αριθμούς και πραγματοποιούμε μετατροπή fourier σε αυτούς για να πάρουμε τις εντάσεις των συχνοτήτων που περιέχονται στο χρονικό διάστημα που καλύπτουν αυτοί οι αριθμοί.

Σημαντική σημείωση εδώ είναι πως με τη μετατροπή fourier έχουμε έξοδο στοιχείων πλήθους $M = N/2 + 1$ όπου N είναι το πλήθος των στοιχείων που εισάγαμε. Έτσι, αν εισάγαμε 4410 αριθμούς θα πάρουμε $M = 4410/2 + 1$, δηλαδή $M = 2206$ στοιχεία εξόδου. Τα στοιχεία εξόδου καλύπτουν όλο το εύρος συχνοτήτων που περιέχονται στο αρχείο μας, δηλαδή από 0hz έως $F_s = \text{sample rate}$, δηλαδή $F_s = 44.1\text{kHz}$ (44100hz). Αυτό σημαίνει πως κάθε στοιχείο εξόδου καλύπτει F_s/M συχνότητες. Άρα, για παράδειγμα, το 5ο στοιχείο εξόδου αντιστοιχεί στις συχνότητες $4 \cdot F_s/M$ μέχρι $5 \cdot F_s/M$, δηλαδή $4 \cdot 44.1/2206\text{kHz}$ έως $5 \cdot 44.1/2206\text{kHz}$.

Για κάθε 4410 αριθμούς που εισάγουμε παίρνουμε ως έξοδο άλλη μία κατακόρυφη στήλη τιμών που αντιστοιχούν σε εντάσεις συχνοτήτων.

Παράδειγμα εξόδου μίας fourier μετατροπής ενός κομματιού:



Όπως μπορούμε να παρατηρήσουμε, οι πολύτιμες πληροφορίες βρίσκονται στις πρώτες 300-400 θέσεις καθώς αυτές διαμορφώνουν το μεγαλύτερο κομμάτι του ήχου που αντιλαμβανόμαστε στη μουσική.

Τέλος, σημαντικό είναι να αναφερθεί πως όταν πραγματοποιούμε μετατροπή fourier καλό είναι να περνάμε το input (τους αριθμούς εντάσεων) από μία συνάρτηση παραθύρου (window function). Αυτό βοηθάει στις περιπτώσεις όπου το frame της μετατροπής διακόπτει την περιοδικότητα του σήματος, καθώς αυτό προκαλεί “artifacts” στα αποτελέσματά μας. Με μία window function αυτό το φαινόμενο εξομαλύνεται. Για να το κάνουμε αυτό απλώς πολλαπλασιάζουμε κάθε αριθμό μας με την window function που επιλέγουμε. Η συνάρτηση που επέλεξα βάσει των αποτελεσμάτων είναι η συνάρτηση Hann:

$$w(n) = \frac{1}{2} \left(1 - \cos\left(\frac{2\pi n}{N-1}\right) \right)$$

Έτσι, πολλαπλασιάζουμε κάθε στοιχείο εισόδου με $W(n)$ όπου n είναι το στοιχείο (η θέση του, όχι η τιμή του) και N είναι το συνολικό πλήθος των στοιχείων εισόδου.

Αποτύπωμα

Όπως είναι κατανοητό, κάθε κατακόρυφη στήλη (slice) στα αποτελέσματα της μετατροπής (βλ. Παραπάνω εικόνα) αντιστοιχεί σε 100ms (ή ότι άλλο χρονικό διάστημα αποφασίσουμε) του αρχικού σήματος. Για την παραγωγή του αποτυπώματος ελέγχουμε όλες τις συχνότητες από την χαμηλότερη επιθυμητή (στις δοκιμές μου από τη 40η θέση, δηλαδή $39 \cdot 44.1/2206$ khz) έως την 300η ($299 \cdot 44.1/2206$ khz) και επιλέγουμε τις δυνατότερες (σε magnitude) από 5 διαφορετικές ζώνες:

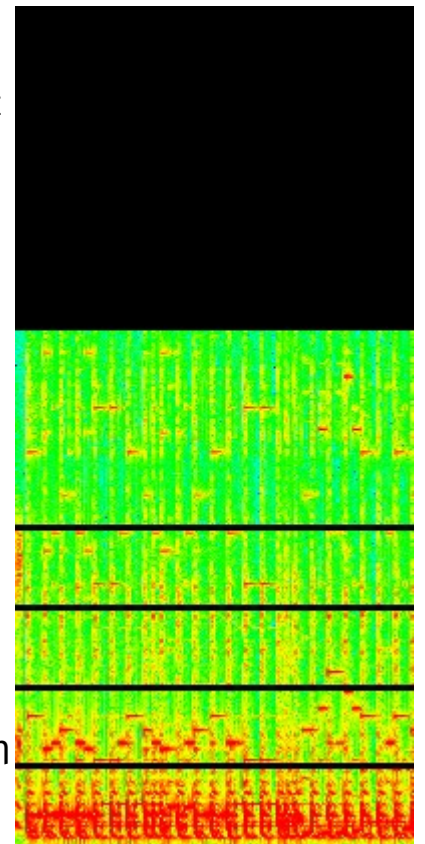
40-80, 80-120, 120-160, 160-200, 200-300

Έτσι, κάθε ζώνη συχνοτήτων έχει εύρος $40 \cdot 44.1/2206$ khz. Για κάθε ζώνη εξετάζουμε τις εντάσεις των συχνοτήτων και σημειώνουμε τη μεγαλύτερη. Στο τέλος της διαδικασίας έχουμε 5 αριθμούς που αντιστοιχούν στις συχνότητες με τη μεγαλύτερη ένταση από κάθε ζώνη.

Αλλάζοντας τις ζώνες ή το εύρος που καλύπτουν οι ζώνες πετυχαίνει διαφορετικά αποτελέσματα αλλά οι παραπάνω αριθμοί παρουσίασαν τα καλύτερα αποτελέσματα στις δοκιμές μου.

Για οπτική κατανόηση παρουσιάζεται δεξιά μία εικόνα από τις εντάσεις συχνοτήτων ενός ηχητικού σήματος. Το μαύρο τετράγωνο αντιστοιχεί στις συχνότητες που αγνοούμε (θέσεις πάνω από την 300η, δηλαδή συχνότητες άνω των $300 \cdot 44.1/2206$ khz). Κάτω από αυτό σημειώνονται οι ζώνες που μελετώνται. Η τελευταία (υψηλότερη ζώνη) είναι μεγαλύτερη από τις άλλες καθώς δεν περιέχει τόσο σημαντικό όγκο πληροφοριών, καθώς και επειδή το ανθρώπινο αυτί αντιλαμβάνεται τον ήχο με λογαριθμικό ρυθμό.

Σε αυτό το σημείο λοιπόν έχουμε 5 αριθμούς για κάθε 100ms ήχου που αποτελούν την “ταυτότητα” του ήχου. Ωστόσο, διαφορετικά αρχεία ή διαφορετικές ηχογραφήσεις του ίδιου τραγουδιού είναι πιθανό να έχουν κάποια απόκλιση στις συχνότητες μέγιστης έντασης μεταξύ τους και κατ’επέκταση για να υπάρχει κάποια ταύτιση οφείλουμε να “στρογγυλέψουμε” τις θέσεις συχνοτήτων που



καταγράφουμε. Έτσι επιλέγουμε έναν παράγοντα (fuzz factor) κάποιων θέσεων και στρογγυλεύουμε τους αριθμούς με βάση αυτόν. Έτσι για κάθε αριθμό πραγματοποιούμε το εξής:

$$\text{Αριθμός} = \text{αριθμός} - \text{αριθμός} \% \text{παράγοντας}$$

Ως παράγοντα στις δοκιμές μου έχω επιλέξει τον αριθμό 3 που παρέχει κάποια ελαστικότητα αλλά διατηρεί την ταυτότητα του κομματιού. Ωστόσο, σε περιπτώσεις όπου ηχογραφούμε ένα τραγούδι από μικρόφωνο για να το αναγνωρίσουμε ο παράγοντας θα έπρεπε να είναι μεγαλύτερος (για παράδειγμα της τάξεως 7-8) καθώς η απόκλιση της ηχογράφησης μας από το αρχείο του τραγουδιού είναι αρκετά μεγάλη.

Τέλος, εφόσον 5 από τους αριθμούς που παράγονται με την παραπάνω διαδικασία αντιστοιχούν σε 100ms ήχου, αυτοί οι 5 αριθμοί είναι και το στοιχείο που θα χρησιμοποιούμε για να συγκρίνουμε δύο διαφορετικά σήματα. Για ευκολία αναζήτησης και αποθήκευσης ενώνουμε αυτούς τους αριθμούς σε έναν σειριακά όπως παρουσιάζεται παρακάτω:

42, 87, 132, 184, 253 -> [4287132184253]

και αυτός ο αριθμός είναι το αποτύπωμα του τραγουδιού για εκείνα τα 100ms. Εφόσον “στρογγυλέψαμε” τις συχνότητες παραπάνω, δεν χρειάζεται να έχουμε τρόπο να διαχωρίσουμε τους αριθμούς αργότερα καθώς δύο όμοια ηχητικά σήματα παράγουν ίδια αποτυπώματα εφόσον οι κορυφαίες (σε ένταση) συχνότητές τους σε εκείνες 5 ζώνες που επιλέξαμε συγκλίνουν μεταξύ τους.

Ο λόγος που κάνουμε αυτή την σειριακή ένωση των αριθμών είναι διότι για να πραγματοποιήσουμε αναζήτηση ενός κομματιού βάσει του ήχου του θα πρέπει να ψάχνουμε αποτελέσματα με βάση τα αποτυπώματα. Επειδή ένα τραγούδι μπορεί να έχει τουλάχιστον 4-5 χιλιάδες αποτυπώματα, θα πρέπει να έχουμε σταθερό χρόνο αναζήτησης ανά αποτύπωμα. Όταν έχουμε μόνο ένα στοιχείο αναζήτησης (το αποτύπωμα ως έναν αριθμό) η διαδικασία αυτή απλοποιείται με ένα hash function (Περισσότερες πληροφορίες παρακάτω).

Αναζήτηση

Επειδή το κάθε τραγούδι καταλήγει να έχει τετραψήφιο αριθμό αποτυπωμάτων (4000+), για να έχουμε αποτελεσματικούς χρόνους αναζήτησης πρέπει να χρησιμοποιήσουμε κάποια “βάση δεδομένων” με κατακερματισμό. Έτσι, τα δεδομένα αποθηκεύονται σε ένα unordered hashmap όπου ο κάθε κόμβος (δηλαδή το κάθε αντικείμενο που περιέχεται στη βάση δεδομένων) έχει ένα αριθμητικό κλειδί αναζήτησης και ως τιμή έχει την λίστα των τραγουδιών που περιέχουν αυτό το αποτύπωμα. Με αυτόν τον τρόπο κάθε αποτύπωμα είναι το hash των τραγουδιών που το περιέχουν.

Έτσι, όταν αναλύουμε ένα τραγούδι σε αποτυπώματα για να το αποθηκεύσουμε στη βάση δεδομένων, για κάθε αποτύπωμα που παράγουμε ελέγχουμε τη βάση για να δούμε αν υπάρχει κόμβος που να αντιστοιχεί σε αυτό το αποτύπωμα. Αν υπάρχει, απλώς προσθέτουμε το όνομα του τραγουδιού που αναλύουμε στη λίστα αποτελεσμάτων. Διαφορετικά δημιουργούμε έναν νέο κόμβο έχοντας ως κλειδί το αποτύπωμα και ως τιμή μία καινούρια λίστα τραγουδιών όπου προσθέτουμε το τραγούδι. Η μορφή της βάσης μας λοιπόν μοιάζει κάπως έτσι:

κόμβος (κλειδί αντικειμένου)	τιμή του κόμβου (λίστα τραγουδιών)
4216212340	τραγούδι1, τραγούδι2, τραγούδι3, τραγούδι4
3335310472	τραγούδι2, τραγούδι3, τραγούδι5, ...
164422088	τραγούδι8, τραγούδι9
253821798	τραγούδι 1, τραγούδι3, τραγούδι5, ...
...	...

Όταν προσπαθούμε να αναγνωρίσουμε ένα τραγούδι από τη βάση δεδομένων ξεκινάμε από την αρχή του τραγουδιού (ή ηχητικού σήματος) και παράγουμε αποτυπώματα. Για κάθε αποτύπωμα που παράγουμε ψάχνουμε αν υπάρχει κόμβος στη βάση δεδομένων με κλειδί το αποτύπωμα που βγάλαμε, δηλαδή αν υπάρχουν τραγούδια που να περιέχουν αυτό το αποτύπωμα. Κάθε τραγούδι που πετυχαίνουμε σε αυτή τη διαδικασία είναι ένα πιθανό αποτέλεσμα οπότε κρατάμε έναν μετρητή για τις φορές που πετύχαμε κάθε τραγούδι. Το τραγούδι (η τα τραγούδια) που πετύχαμε τις περισσότερες φορές είναι πιο πιθανό να περιέχονται στο σήμα που αναλύσαμε. Έτσι, η διαδικασία ταύτισης γίνεται με αθροιστικό τρόπο όπου ο “νικητής” είναι το τραγούδι που έχει (συνολικά) τα περισσότερα κοινά αποτυπώματα με το σήμα που αναλύουμε.

Για αναφορά, ο κώδικας αναζήτησης είναι ο παρακάτω:

```
void Database::matchsong(unsigned long *fingerprints, size_t n, vector<match_data> &results){
    unordered_map<size_t, size_t> r;
    results.clear();
    for(size_t i=0; i<n; i++){
        auto s = base.find(fingerprints[i]);
        if(s!=base.end()){
            for(size_t songindex : s->second)
                r[songindex]++;
        }
    }
    for(auto sd: r)
        results.push_back(match_data(songlist[sd.first], sd.second));
    sort(results.begin(), results.end(), [](const match_data& md1, const match_data& md2)
    {
        return md1.getHits() > md2.getHits();
    });
}
```

Σύγκριση ομοιότητας

Η σύγκριση μεταξύ δύο τραγουδιών για να δούμε ποσοστό ομοιότητας είναι παρόμοια μέθοδος με την αναζήτηση. Αρχικά παράγουμε τα αποτυπώματα του πρώτου τραγουδιού. Μετά παράγουμε του δεύτερου και για κάθε αποτύπωμα του ενός τραγουδιού βλέπουμε αν υπήρξε αυτό το αποτύπωμα στο άλλο. Έστω ότι έχουμε δύο τραγούδια με τίτλους A και B αντιστοίχως.

Το τραγούδι A έχει 4000 αποτυπώματα. Το τραγούδι B έχει 3500 αποτυπώματα. Αν βρούμε 2000 κοινά αποτυπώματα μεταξύ των δύο τότε το A περιέχεται κατά 50% στο B (αθροιστικά) ενώ το B περιέχεται κατά περίπου 57% στο A (αθροιστικά).

Χρονική ολίσθηση

Επειδή όταν επεξεργαζόμαστε ένα τραγούδι είναι πιθανό να υπάρχει κάποια χρονική μετατόπιση του ήχου μέσα στο αρχείο, είτε λόγω μη συγχρονισμένης ηχογράφησης, είτε λόγω διαφορετικών εκδοχών του αρχείου ήχου, κάτι που είναι απαραίτητο για την ακριβή αναγνώριση του ήχου είναι τα αποτυπώματά μας να είναι ελαστικά στη χρονική μετατόπιση. Αυτό επιτυγχάνεται ως εξής:

Κατά τη μετατροπή σε τομέα συχνοτήτων (fast fourier transformation), αντί να μετατρέπουμε κάθε 100ms (ή οποιοδήποτε άλλο χρονικό παράθυρο) και να περνάμε στα επόμενα 100ms, ολισθαίνουμε το παράθυρό μας κατά ένα ποσοστό του χρονικού διαστήματος. Έτσι, αντί να παράγουμε διακριτά αποτυπώματα της μορφής 0-100ms, 100-200ms κλπ, παράγουμε αποτυπώματα της μορφής 0-100ms, 50-150ms, 100-200ms, 150-250 κλπ. Αυτό έχει δύο αποτελέσματα: αρχικά τα αποτυπώματά μας είναι πιο ανεκτικά σε χρονικές μετατοπίσεις και επίσης έχουμε περισσότερες χρονικές “λωρίδες” και κατ’επέκταση πιο καθαρή εικόνα στα spectrograms που μπορεί να παράγουμε αν θελήσουμε.

Στις δοκιμές μου δοκίμασα ολίσθηση κατά 30% (0-100, 30-130, 60-160, ...), 60% (0-100, 60-160, 120-180, ...) και 50% (0-100, 50-150, 100-200, ...) όπου είδα και τα καλύτερα αποτελέσματα (ακρίβεια ανά χρονικό κόστος και κόστος αποθήκευσης).

Σύμφωνα με τα παραπάνω συμπεραίνουμε πως ο χρόνος ολίσθησης δ (του αρχικού pdf) είναι ίσος με:

$$\text{ποσοστό ολίσθησης} * \text{χρονικό διάστημα αποτυπώματος}$$

Έτσι για χρονικό διάστημα αποτυπώματος ίσο με 100ms και ποσοστό ολίσθησης 50% θα έχουμε $\delta=50\text{ms}$

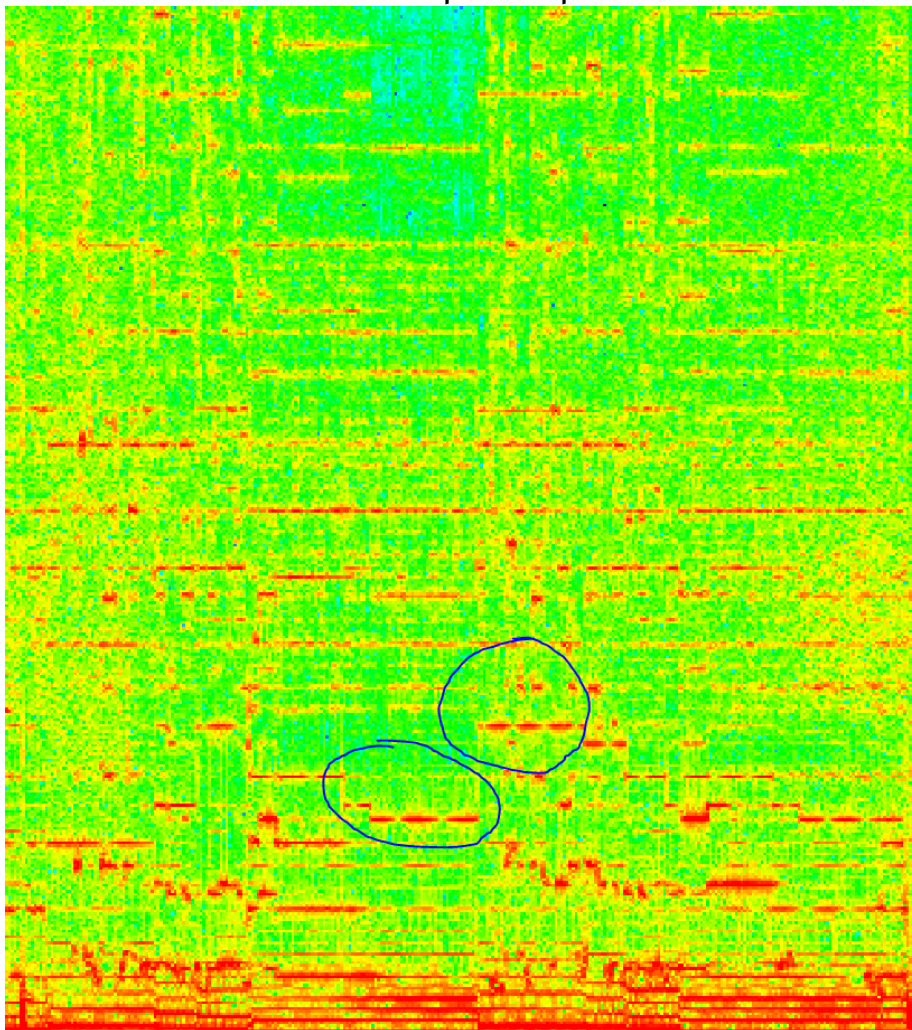
Χρονική παραμόρφωση (time stretching)

Ένα τραγούδι μπορεί να παραμορφωθεί χρονικά με δύο διαφορετικούς τρόπους:

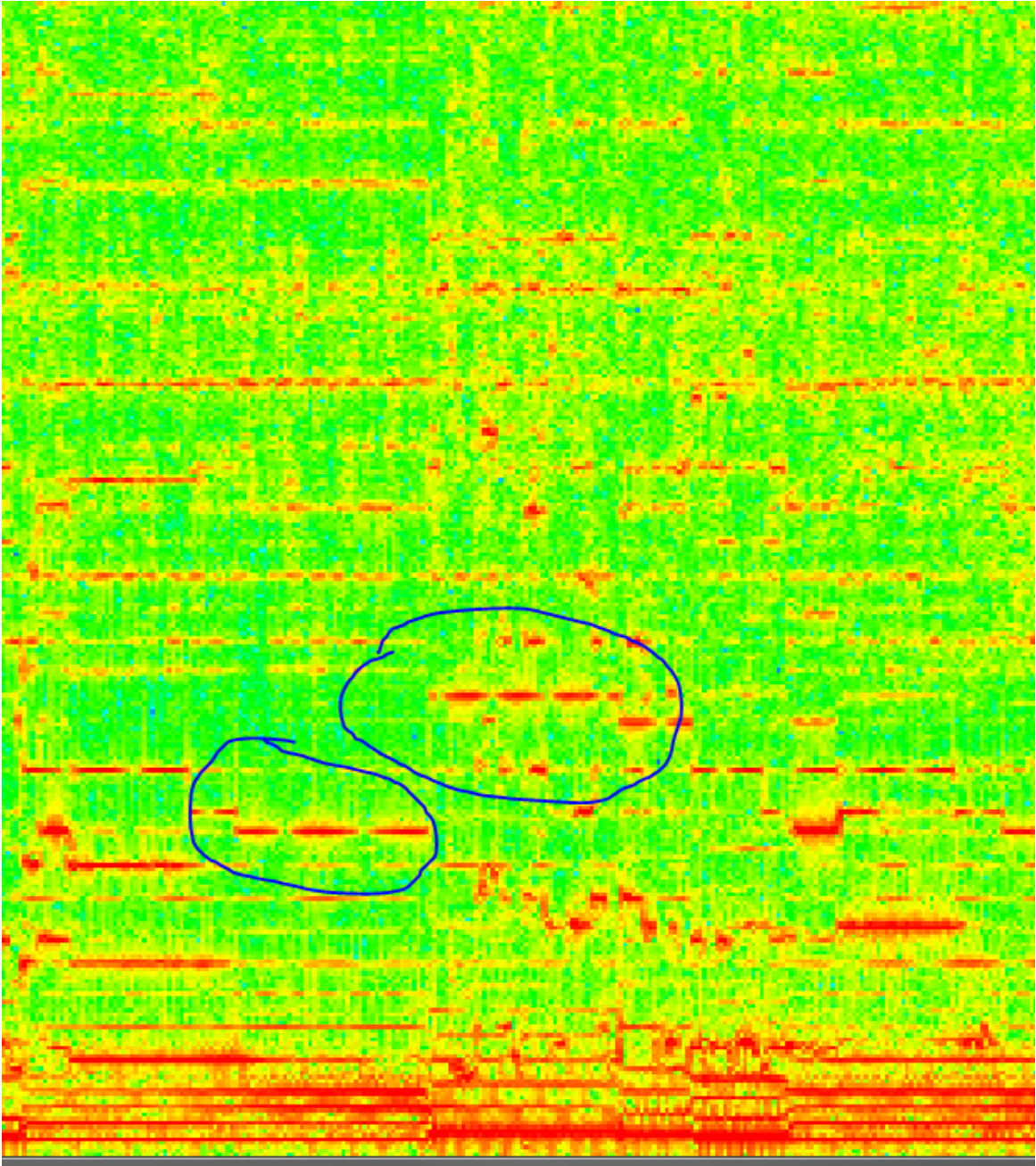
- Με παραμόρφωση τόνου (όπου ο τόνος/pitch, δηλαδή οι συχνότητες παραμορφώνονται αντίστοιχα)
- Με resampling (όπου υπολογίζονται και δημιουργούνται καινούρια samples ώστε το τραγούδι να έχει διαφορετική ταχύτητα / διάρκεια ενώ ο τόνος του παραμένει όσο το δυνατόν πιο πιστός στον αυθεντικό), ο οποίος είναι ο τρόπος που χρησιμοποιούν όλοι οι καλλιτέχνες.

Στην πρώτη περίπτωση τα fingerprints του αυθεντικού τραγουδιού κατά πλειοψηφία δεν θα ταιριάζουν πλέον με τα fingerprints του παραμορφωμένου τραγουδιού. Ωστόσο, στη δεύτερη περίπτωση ένα μεγάλο ποσοστό των samples έχουν διατηρήσει την ταυτότητά τους στον τομέα συχνοτήτων και κατ'επέκταση παράγουν όμοια fingerprints. Παρακάτω παρουσιάζονται spectrograms ενός κομματιού σε κανονική και μισή ταχύτητα με διατήρηση τόνου:

Κανονική ταχύτητα:



Μισή ταχύτητα:



Από τις παραπάνω εικόνες παρατηρούμε πως η “ταυτότητα” στον τομέα συχνοτήτων παραμένει μετά την χρονική παραμόρφωση καθώς παράγονται επιπλέον όμοια αποτυπώματα “ανάμεσα” στα αρχικά. Το αντίστροφο συμβαίνει όταν επιταχύνουμε το σήμα.

Σε πραγματικές δοκιμές ο αλγόριθμος αναγνώρισε επιτυχώς τα κομμάτια που παραμορφώθηκαν (βλ. Παρακάτω).

Μίξη πολλαπλών αρχείων

Αν υποθέσουμε πως σε ένα αρχείο ήχου περιέχονται παραπάνω από ένα διαφορετικά τραγούδια και επιχειρήσουμε να το ανιχνεύσουμε με την παραπάνω μέθοδο, αυτό που θα συμβεί είναι το παρακάτω:

Καθώς θα δημιουργούνται αποτυπώματα του ήχου και το κάθε αποτύπωμα “βαθμολογεί” τα τραγούδια που περιέχονται στη βάση, ο αλγόριθμος θα περάσει από τα τραγούδια που περιέχονται στην μίξη. Έτσι, όσο περνάει από το τραγούδι A, τα αποτυπώματα ταιριάζουν, μεταξύ άλλων, με το τραγούδι A. Για κάθε αποτύπωμα που ταιριάζει με το τραγούδι A έχουμε και 100ms του τραγουδιού A που περιέχονται στη μίξη. Αντίστοιχα, όσο ο αλγόριθμος περνάει από το τραγούδι B στη μίξη έχουμε αποτυπώματα που ταιριάζουν με το τραγούδι B από τη βάση δεδομένων. Συνεπώς, τα κορυφαία σε χτυπήματα τραγούδια θα είναι αυτά που περιέχονται στη μίξη.

Επίσης, μπορούμε να θέσουμε ένα χρονικό όριο πάνω από το οποίο ο αλγόριθμος συμπεραίνει ότι ένα τραγούδι περιέχεται στο αρχείο, και συνεπώς να το εμφανίζει. Αν θέσουμε για παράδειγμα το όριο = 40 δευτερόλεπτα, τότε ο αλγόριθμος θα κρίνει ότι ένα τραγούδι υπάρχει στη μίξη εφόσον πετύχει $40 \times 10 = 400$ αποτυπώματα (εφόσον κάθε αποτύπωμα αντιστοιχεί σε 100ms) αυτού του τραγουδιού, και συνεπώς θα το βάλει στη λίστα αποτελεσμάτων.

Σημαντικό είναι να σημειώσουμε πως όσο περισσότερα κομμάτια περιέχει η μίξη και όσο περισσότερα έχουμε στη βάση, τόσο πιο πιθανό είναι να ταιριάζουν και αποτυπώματα από τραγούδια που απλώς τυγχάνει να έχουν όμοια ταυτότητα σε αμελητέα χρονικά διαστήματα.

Διασκευές

Αν υποθέσουμε ότι ένα τραγούδι A περιέχεται στη βάση δεδομένων και σκανάρουμε μία διασκευή του, τότε ο μόνος τρόπος να αναγνωριστεί η διασκευή ως το τραγούδι A είναι αν η διασκευή περιέχει αρκετά τμήματα του A αυτούσια, καθώς διαφορετικές εκτελέσεις (με όργανα, διαφορετική ηχογράφηση, κούρδισμα, φωνητικά, άλλο συγκρότημα, φίλτρα κλπ) οδηγούν σε διαφορετική ταυτότητα στον τομέα συχνοτήτων. Ωστόσο, αν η διασκευή είναι κάποιο remix/edit του αρχικού κομματιού όπου τμήματα ή κανάλια του αρχικού κομματιού περιέχονται, τα αποτυπώματα εκείνων των τμημάτων θα ταυτίζονται με αυτά του κομματιού A ακόμη και αν δεν ταιριάζουν χρονικά.

Αυτό οφείλεται στο γεγονός ότι δεν μας ενδιαφέρει που βρίσκεται το κάθε αποτύπωμα μέσα στο τραγούδι καθώς ο αλγόριθμος χρησιμοποιεί το ποσοστό των αποτυπωμάτων από το τραγούδι υπό αναζήτηση που ταιριάζει με το κάθε τραγούδι στη βάση δεδομένων για να κρίνει στατιστικά την ομοιότητα.

Οδηγίες χρήσης

Δημιουργία βάσης:

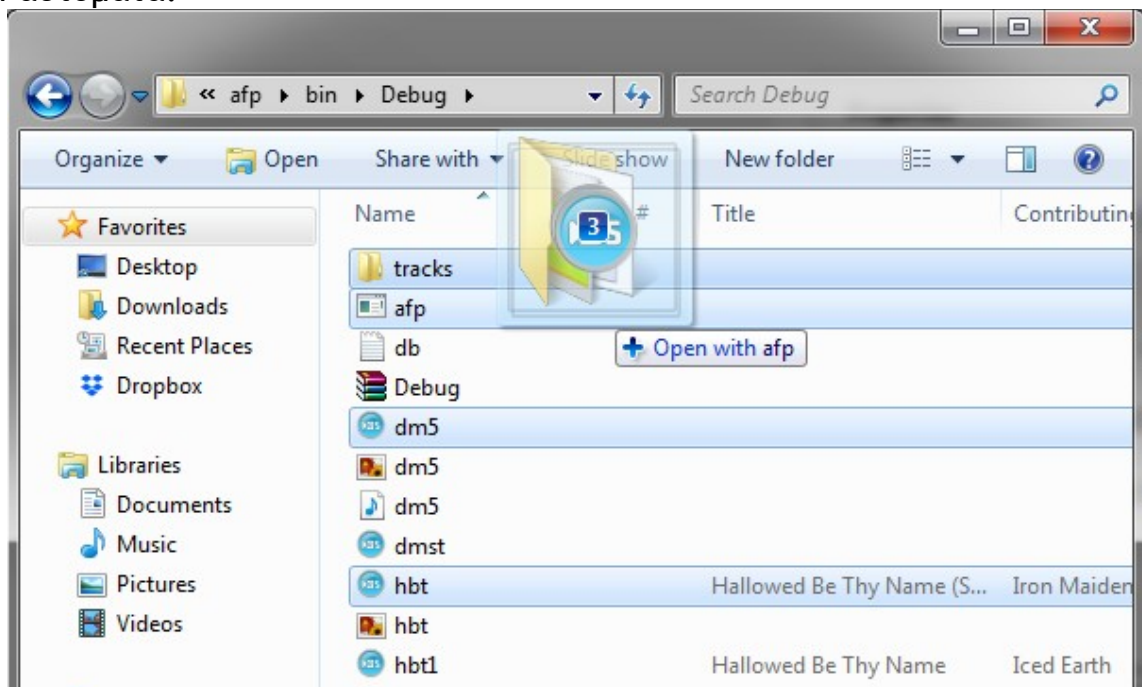
Το πρόγραμμα χρησιμοποιεί ως default αποθηκευτικό προορισμό το αρχείο “db.txt” που βρίσκεται δίπλα στο εκτελέσιμο (“afp.exe”). Συνεπώς όποτε θέλουμε να διαγραφεί η βάση μας απλώς διαγράφουμε το “db.txt”. Αν το αρχείο δεν υπάρχει σημαίνει πως δεν έχει δημιουργηθεί βάση ακόμη και θα δημιουργηθεί κατά το πρώτο σκανάρισμα αρχείου. Όταν το πρόγραμμα ανοίγει επιχειρεί να φορτώσει όλα τα αποτυπώματα από αυτό το αρχείο.

Σκανάρισμα φακέλων/αρχείων:

Υπάρχουν δύο τρόποι να σκανάρουμε αρχεία στη βάση δεδομένων μας. Ο πρώτος είναι να το τρέξουμε και να πληκτρολογήσουμε την εντολή:

“scan <filepath>” όπου filepath μπορεί να είναι είτε κάποιο αρχείο mp3, είτε κάποιος φάκελος με αρχεία mp3. Το πρόγραμμα θα σκανάρει κάθε αρχείο και θα δημιουργήσει αποτυπώματα τα οποία θα εξάγει και στο db.txt για μελλοντική φόρτωση. Μπορούμε να χρησιμοποιήσουμε αυτή την εντολή επανειλημμένα. Παράδειγμα χρήσης: “scan C:\Users\Spark\downloads\music”

Ο δεύτερος τρόπος είναι αντί να ανοίξουμε το πρόγραμμα, να επιλέξουμε όποιους φακέλους και αρχεία mp3 επιθυμούμε και να τα σύρουμε πάνω στο εκτελέσιμο. Έτσι, όταν αυτό ανοίξει θα τα χρησιμοποιήσει ως παραμέτρους για την εντολή scan αυτόματα:



Το πρόγραμμα θα αγνοήσει τα τραγούδια που υπάρχουν ήδη στο db.txt και θα σκανάρει τα καινούρια.

Σημείωση ! Το πρόγραμμα σκανάρει αυτόματα τον φάκελο “tracks” που βρίσκεται στο ίδιο μονοπάτι οπότε μπορείτε να τοποθετήσετε εκεί ό,τι τραγούδια επιθυμείτε.

Εξαγωγή / εισαγωγή αποθηκευτικών αρχείων

Με την εντολή “**save <filepath>**” μπορούμε να εξάγουμε όλη την τρέχουσα βάση δεδομένων (του db.txt) σε ένα αρχείο της επιλογής μας. Παράδειγμα χρήσης: “**save savefile.txt**”

Αντίστοιχα, με την εντολή “**load <filepath>**” φορτώνουμε αποτυπώματα από ένα αρχείο της επιλογής μας στο db.txt. Παράδειγμα χρήσης: “**load C:\Users\Spark\downloads\newdatabase.txt**”

Αναγνώριση αρχείου mp3

Με την εντολή “**identify <filepath>**” το πρόγραμμα παράγει τα αποτυπώματα του αρχείου της επιλογής μας και έπειτα το αναζητά στη βάση δεδομένων του. Τέλος, εμφανίζει τα κορυφαία αποτελέσματα

Αναγνώριση από μικρόφωνο

Ομοίως με την παραπάνω εντολή, αν πληκτρολογήσουμε “**record <νούμερο>**” το πρόγραμμα θα καταγράψει τα επόμενα δευτερόλεπτα (με βάση την επιλογή μας) και θα παράγει αποτυπώματα πριν τα συγκρίνει με τη βάση δεδομένων.

Σημαντική σημείωση: Το πρόγραμμα διαθέτει παραμέτρους ελαστικότητας, χρόνου ανά αποτύπωμα, sample rate, ολίσθησης και ελάχιστου απαιτούμενου χρόνου αντιστοίχισης. Οι παραπάνω παράμετροι παίζουν σημαντικό ρόλο στην παραγωγή των αποτυπωμάτων και αποτελεσμάτων και όταν αλλάζουν πρέπει ο χρήστης να δημιουργήσει τη βάση του από την αρχή (δηλαδή να διαγράψει το αρχείο db.txt).

Επίσης ο ελάχιστος χρόνος αντιστοίχισης κρίνει ποιος είναι ο λιγότερος χρόνος που μπορεί να αντιστοιχεί ένα τραγούδι από τη βάση για να θεωρηθεί πιθανό αποτέλεσμα. Αυτό σημαίνει πως αν θέσουμε για παράδειγμα αυτό τον χρόνο σε 20 δευτερόλεπτα, τότε μόνο τα αποτελέσματα που ταιριάζουν με αποτυπώματα που να αντιστοιχούν σε τουλάχιστον 20 δευτερόλεπτα χρόνου θα εμφανιστούν. Η ηχογράφηση δεν είναι δυνατό να διαρκεί λιγότερο από αυτόν τον χρόνο.

Παραγωγή εικόνας

“**spectrogram <filepath>**” παράγει την εικόνα σε τομέα συχνοτήτων του αρχείου που επιλέγουμε. Παράδειγμα χρήσης: “**spectrogram rave.mp3**”
Το αρχείο εικόνας θα έχει ίδιο όνομα με το αρχείο ήχου αλλά με την κατάληξη .png (π.χ. “rave.png”).

Σύγκριση δύο αρχείων

“**compare <filepath> <filepath>**” συγκρίνει δύο αρχεία ήχου της επιλογής μας και εμφανίζει πόσα κοινά αποτυπώματα έχουν, καθώς και τι ποσοστό του καθενός βρέθηκε στο άλλο αντιστοίχως. Παράδειγμα χρήσης: “**compare rave.mp3 C:\Users\Spark\downloads\music\art-of-dying.mp3**”

Ρύθμιση παραμέτρων

Οι βασικές παράμετροι είναι οι εξής:

παράμετρος	σημασία
sample rate	samples/second στα οποία να αποκωδικοποιούνται τα αρχεία mp3. Ιδανική ποιότητα/κόστος: 44100
Fingerprint length	πόσα ms να διαρκεί το κάθε frame αποτυπώματος (ιδανικός χρόνος 100ms)
slide percent	ποσοστό ολίσθησης κατά την παραγωγή αποτυπωμάτων (ιδανικά αποτελέσματα/κόστος: 50%)
fuzz factor	παράγοντας ανεκτικότητας, μπορεί να είναι από 1 (πρακτικά μη ανεκτικό, δεν δέχεται καμία απόκλιση συχνότητας) έως 10 (μεγάλη ανεκτικότητα, οδηγεί σε ανακριβή αποτελέσματα καθώς πολλά αποτυπώματα ταιριάζουν). Ιδανικός αριθμός: 3. Ωστόσο για αναγνώριση από μικρόφωνο βοηθάει να είναι μεγαλύτερος (π.χ. 5-6).
minimum match time	ελάχιστος χρόνος που επιτρέπεται να ταιριάζει ένα τραγούδι από τη βάση για να θεωρηθεί πιθανό αποτέλεσμα. Η ιδανική τιμή διαφέρει ανάλογα την περίπτωση. Για ηχογράφηση χρειάζεται μικρές τιμές (5-8) καθώς πολλά από τα αποτυπώματα δεν ταιριάζουν από μικρόφωνο. Για αναγνώριση από αρχείο ωστόσο μεγάλες τιμές οδηγούν σε πιο ακριβή αποτελέσματα καθώς από ένα αρχείο 3 λεπτών δεν μπορούμε να δεχτούμε 10 δευτερόλεπτα ως επαρκή για να θεωρήσουμε ότι αντιστοιχεί

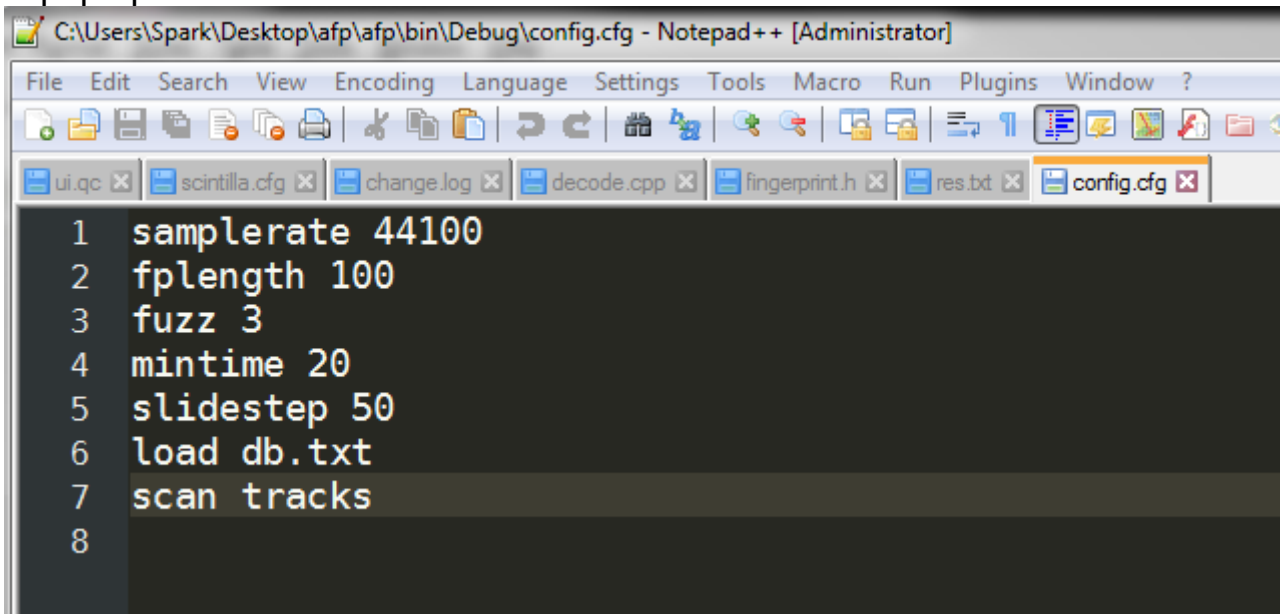
εντολές για την αλλαγή τους:

sample rate:	“<code>samplerate <number></code>” (default: 44100)
fingerprint length(ms):	“<code>fplength <number></code>” (default: 100)
slide percent:	“<code>slidestep <number></code>” (default: 50)
fuzz factor:	“<code>fuzz <number></code>” (default: 3)
min match time:	“<code>mintime <number></code>” (default 20)

Σημείωση !: Η αλλαγή οποιασδήποτε παραμέτρου σημαίνει ότι τα υπάρχοντα αποτυπώματα είναι πλέον μη συμβατά και επομένως πρέπει να διαγραφούν και να ξανα-παραχθούν.

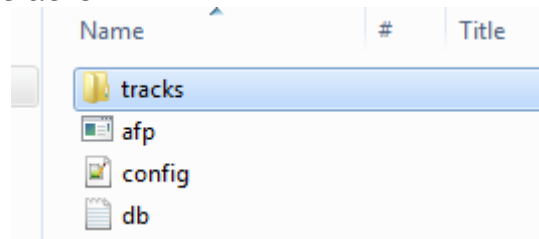
Configuration files

Στον φάκελο του εκτελέσιμου υπάρχει ένα αρχείο config.cfg στο οποίο βρίσκονται οι default παράμετροι, καθώς και οι εντολές για την φόρτωση αρχείων κατά την εκκίνηση του προγράμματος. Μπορείτε να αλλάξετε τις τιμές των παραμέτρων εκεί:



```
1 samplerate 44100
2 fplength 100
3 fuzz 3
4 mintime 20
5 slidestep 50
6 load db.txt
7 scan tracks
8
```

Παρατηρούμε πως στις βασικές εντολές βρίσκεται και η εντολή “scan tracks” που σκανάρει τον φάκελο “tracks”



Τέλος, αν επιθυμούμε μπορούμε να δημιουργήσουμε κι άλλα αρχεία κειμένου (txt, cfg ή οποιαδήποτε άλλη κατάληξη) και να τα εκτελέσουμε ανά πάσα στιγμή είτε μέσα από το πρόγραμμα, είτε από το config.cfg με την εντολή:

“exec <filepath>”

Αν για παράδειγμα γράψουμε σε ένα αρχείο κειμένου:

“compare song1.mp3 song2.mp3” και το αποθηκεύσουμε ως “mycommands.cfg”, μπορούμε μετά να καλέσουμε την εντολή:

“exec mycommands.cfg”

μέσα από το πρόγραμμα για να τρέξουν οι εντολές που βρίσκονται μέσα σε αυτό το αρχείο κειμένου.

Σημείωση! Η εντολή **“help”** εμφανίζει όλες τις παραπάνω επιλογές με οδηγίες.

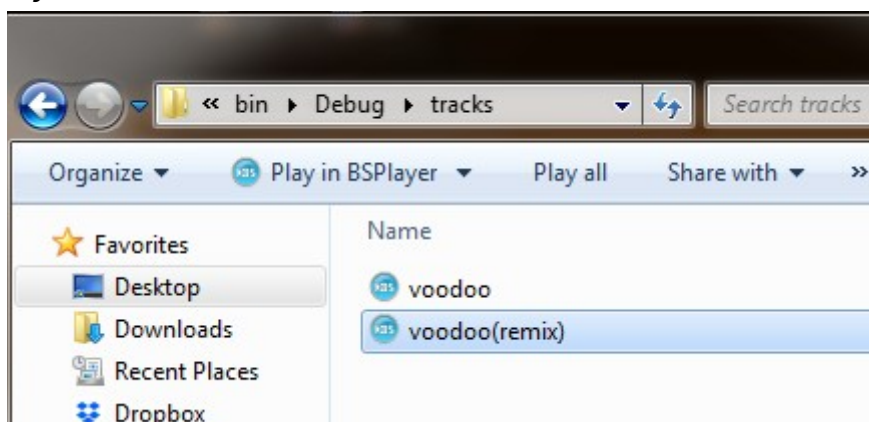
Παραδείγματα αναγνώρισης ήχου Διασκευή

Αρχικά θα πρέπει να βρούμε ένα remix. Ένα καλό παράδειγμα είναι το εξής:

[prodigy - voodoo people](#)
[prodigy - voodoo people \(pendulum remix\)](#)

Ο λόγος είναι πως το remix περιέχει αρκετά τμήματα του αρχικού (για παράδειγμα το κεντρικό synthesizer αλλά αλλάζει τον ρυθμό και την δομή του κομματιού.

Τοποθετούμε λοιπόν τα δύο αρχεία ήχου στον φάκελο tracks και τρέχουμε το πρόγραμμά μας:



Μόλις τρέξει παρατηρούμε πως προσθέτει τα δύο αρχεία στη βάση δεδομένων:

```
C:\Users\Spark\Desktop\afp\afp\bin\Debug\afp.exe
Processing file...
Added: "The Prodigy-Voodoo People (Pendulum Remix)The Prodigy" to the database ("db.txt") with [6485] fi
Processing file...
Added: "The Prodigy-Voodoo People (Official Video)The Prodigy" to the database ("db.txt") with [6184] fi

Audio fingerprinting prototype v0.8, Author George Kiriakidis
Libraries used: mpg123, fftw3, libpng, png++, dirent, zlib
All library rights belong to their respective authors and their work.
Type "help" for list of options
>>
```

Σε αυτή τη φάση πληκτρολογούμε **“identify tracks/voodoo.mp3”**

```
C:\Users\Spark\Desktop\afp\afp\bin\Debug\afp.exe
Processing file...
Minimum time matching required: 20
results:
matches:
1) [Best match]: The Prodigy-Voodoo People (Pendulum Remix)The Prodigy, 6485 hits(324.25 seconds)
2) The Prodigy-Voodoo People (Official Video)The Prodigy, 1704 hits(85.2 seconds)
```

Παρατηρούμε πως το πρόγραμμα εντοπίζει την ομοιότητα στα σημεία όπου τα κρουστά λείπουν και ακούγεται μόνο η μελωδία.

Αν θέλαμε μπορούσαμε να πληκτρολογήσουμε
“[compare tracks/voodoo.mp3 tracks/voodoo\(remix\).mp3](#)”

```
C:\Users\Spark\Desktop\afp\afp\bin\Debug\afp.exe
starting matching...
The two files share 1704 hits, 85.2 seconds (not continuously)
"tracks/voodoo.mp3" matches with "tracks/voodoo(remix).mp3" by 27.555%
"tracks/voodoo(remix).mp3" matches with "tracks/voodoo.mp3" by 26.276%
```

Ο λόγος που τα δύο κομμάτια δεν ταιριάζουν παραπάνω είναι επειδή το remix έχει αρκετά επεξεργασμένο ήχο, δεν πρόκειται απλώς για μία αναδιανομή χρονικών διαστημάτων. Επίσης με ένα μεγαλύτερο fuzz factor θα είχαμε ακόμη καλύτερο matching (καθώς θα είχαμε μεγαλύτερη ελαστικότητα).

Σε αντίθεση, έχουμε τα παρακάτω:

[Dillon Francis - Masta blasta 2.0](#)

[Dillon Francis - Masta blasta 2.0 \(Barely alive edit\)](#)

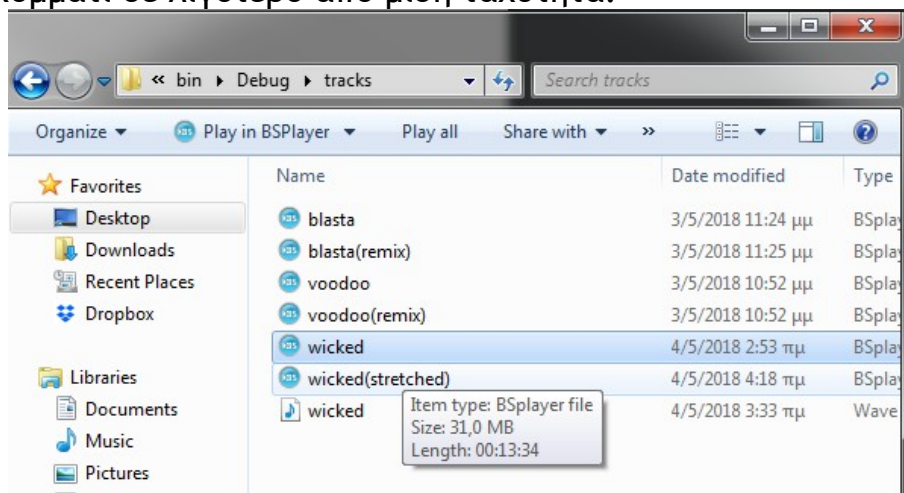
Εδώ έχουμε αλλαγή στην ταχύτητα του κομματιού και μερικά έξτρα εφέ:

```
C:\Users\Spark\Desktop\afp\afp\bin\Debug\afp.exe
starting matching...
The two files share 1856 hits, 92.8 seconds (not continuously)
"tracks/blasta.mp3" matches with "tracks/blasta(remix).mp3" by 31.819%
"tracks/blasta(remix).mp3" matches with "tracks/blasta.mp3" by 49.5859%
```

Τέλος, για να ελέγξουμε την αποτελεσματικότητα του αλγορίθμου σε χρονική παραμόρφωση ή μίξη αρκεί απλώς να πάρουμε ένα τραγούδι και να το κάνουμε time stretch. Έχουμε λοιπόν το κομμάτι

[Griz - Wicked](#)

Και το ίδιο κομμάτι σε λιγότερο από μισή ταχύτητα:



Παρατηρούμε πως το μήκος του κομματιού είναι 13:34 σε αντίθεση με τα 6 λεπτά που διαρκεί κανονικά το τραγούδι. Κάνοντάς το identify λοιπόν βλέπουμε τα παρακάτω αποτελέσματα:

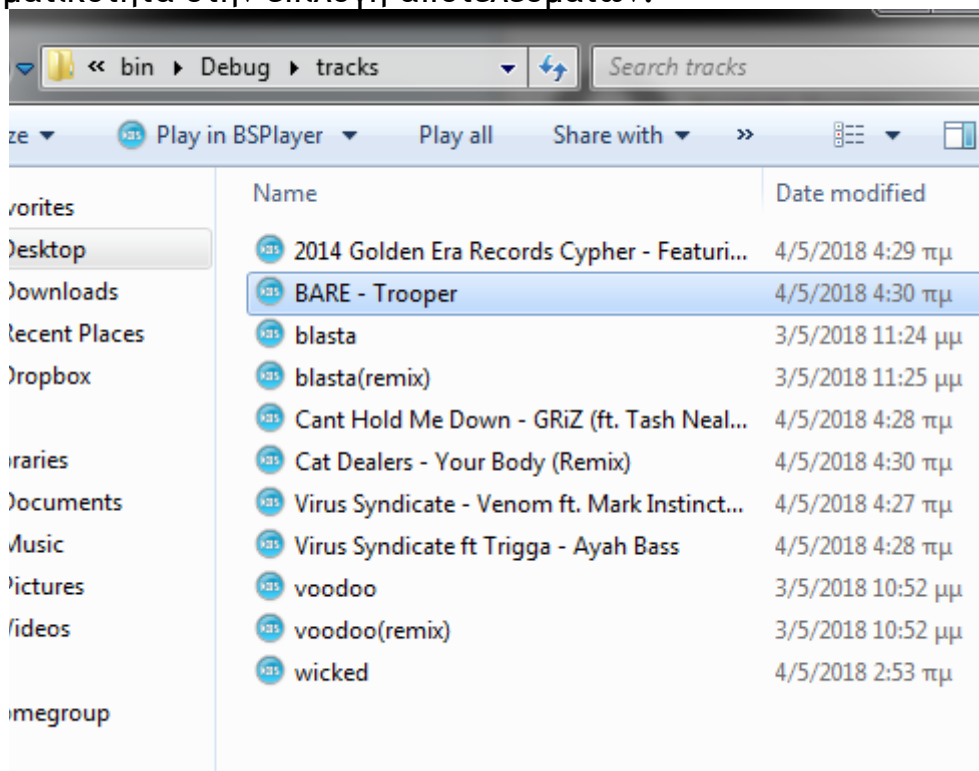
```
C:\Users\Spark\Desktop\afp\afp\bin\Debug\afp.exe
Processing file...
Minimum time matching required: 20
results:
matches:
1) [Best match]: Wicked-GRiZ (ft. Eric Krasno) Good Wwicked, 7858 hits(392.9 seconds)
2) The Prodigy-Voodoo People (Pendulum Remix)The Prodigy, 2071 hits(103.55 seconds)
3) The Prodigy-Voodoo People (Official Video)The Prodigy, 1904 hits(95.2 seconds)
4) Dillon Francis-Masta Blasta 2.0, 793 hits(39.65 seconds)
5) Dillon Francis-Masta Blasta 2.0 (Barely Alive)Dillon Francis, 465 hits(23.25 seconds)

Audio fingerprinting prototype v0.8, Author George Kiriakidis
Libraries used: mpg123, fftw3, libpng, png++, dirent, zlib
All library rights belong to their respective authors and their work.
Type "help" for list of options
>>
```

Σημαντικό είναι να προσθέσουμε πως το stretched version δεν υπάρχει στη βάση δεδομένων, αρκεί το αυθεντικό και το αντιστοιχεί. Αν ακούσετε το τραγούδι σε μισή ταχύτητα (στο youtube) η παραμόρφωση είναι αρκετή για να κάνει το τραγούδι αγνώριστο ακουστικά. Μπορούμε πάντα φυσικά να περιορίσουμε τα αποτελέσματα ορίζοντας έναν μεγαλύτερο ελάχιστο απαιτούμενο χρόνο αντιστοίχισης.

Μίξη

Τέλος, για το θέμα της μίξης δημιουργούμε ένα αρχείο ήχου όπου τοποθετούμε τμήματα (φυσικά τουλάχιστον 15-20 δευτερολέπτων για να υπάρχει περιθώριο αναγνώρισης από τον αλγόριθμο) από 3-4 διαφορετικά κομμάτια. Αρχικά σκάναρα περισσότερα κομμάτια ώστε να έχουμε φανερή αποτελεσματικότητα στην επιλογή αποτελεσμάτων:



Το “mix” μου απλώς περιέχει τμήματα από 4 κομμάτια σε διαφορετικές ταχύτητες (μισή, διπλή, μιάμιση κλπ):



Τα κομμάτια είναι:

- 1) Cat dealers - your body (σε κανονική ταχύτητα)
- 2) Virus syndicate - ayah bass (σε μισή ταχύτητα)
- 3) Griz - can't hold me down (σε διπλάσια ταχύτητα)
- Cat dealers - your body (σε 75% ταχύτητα) (ίδιο με πρώτο)
- 4) Golden era records cypher 2014 (σε 70% ταχύτητα)

Κάνοντας identify mix.mp3 έχουμε τα παρακάτω αποτελέσματα:

```
C:\Users\Spark\Desktop\afp\afp\bin\Debug\afp.exe
Processing file...
Minimum time matching required: 20
results:
matches:
1) [Best match]: Cat Dealers-Your Body (Remix), 1798 hits(89.9 seconds)
2) 2014 Golden Era Records Cypher-Featuring: K21 Briggs Vents Fu2014 Golden Era Records Cypher, 1349 hits(67.45 seconds)
3) Cant Hold Me Down-GRiZ (ft. Tash Neal of The LonCant Hold Me Down, 872 hits(43.6 seconds)
4) Virus Syndicate ft TriggA-Ayah Bass, 792 hits(39.6 seconds)
5) BARE-Trooper, 764 hits(38.2 seconds)
6) Wicked-GRiZ (ft. Eric Krasno) Good Wicked, 681 hits(34.05 seconds)
7) Dillon Francis-Masta Blasta 2.0, 638 hits(31.9 seconds)
8) Virus Syndicate-Venom ft. Mark Instinct (Official Virus Syndicate, 618 hits(30.9 seconds)
9) The Prodigy-Voodoo People (Official Video)The Prodigy, 537 hits(26.85 seconds)
10) The Prodigy-Voodoo People (Pendulum Remix)The Prodigy, 513 hits(25.65 seconds)

Audio fingerprinting prototype v0.8, Author George Kiriakidis
Libraries used: mpg123, fftw3, libpng, png++, dirent, zlib
All library rights belong to their respective authors and their work.
Type "help" for list of options
>>
```

Τα πρώτα 4 αποτελέσματα ταιριάζουν ακριβώς με τα τραγούδια που υπάρχουν στο mix, έχοντας ως σειρά προτεραιότητας τον αριθμό από hits που πέτυχε ο αλγόριθμος.

Όλες οι παραπάνω δοκιμές έγιναν με:

- αποκωδικοποίηση mp3 στα 44100 samples/sec
- χρονικό διάστημα αποτυπώματος 100ms
- ολίσθηση κατά 50% (50ms)
- ανεκτικότητα απόκλισης 5
- ελάχιστο χρόνο αποδοχής αποτελέσματος 20 seconds

Βιβλιοθήκες που χρησιμοποιήθηκαν:

- mpg123 - αποκωδικοποίηση mp3
- libpng - εξαγωγή εικόνας (spectrograms)
- png++ - libpng wrapper για C++
- dirent - για directory iteration
- fftw - μετατροπή fourier (τομέας χρόνου -> συχνότητων)
- zlib - libpng dependency

Το source code είναι σχεδόν ολοκληρωτικά σε C++ για λόγους συμβατότητας με τις βιβλιοθήκες και χαμηλό χρόνο υπολογισμών.

Contact info:
GeorgeKstr@gmail.com